

Manual Técnico

**Grado en Ingeniería Informática
(Doble mención computación + computadores)**

Sistema de mapeo de interiores mediante mediciones láser.

Autor

Manuel Rafael Navarro Fuentes

Director

Dr. Rafael Muñoz Salinas

Octubre, 2020



UNIVERSIDAD DE CÓRDOBA

Índice

Índice de tablas	VI
Índice de figuras	VII
Índice de algoritmos	1
1. Introducción	3
1.1. Organización de la memoria	6
2. Definición del problema	7
2.1. Definición del problema real	7
2.2. Definición del problema técnico	8
2.2.1. Funcionamiento	8
2.2.2. Entorno	8
2.2.3. Vida esperada	9
2.2.4. Ciclo de mantenimiento	9
2.2.5. Competencia	9
2.2.6. Aspecto externo	10
2.2.7. Estandarización	10
2.2.8. Calidad y fiabilidad	10
2.2.9. Programa de tareas	11
2.2.10. Pruebas	12
2.2.11. Seguridad	12
3. Antecedentes	13
3.1. SLAM y métodos de resolución	13
3.2. Sistemas similares al que se pretende construir	15
3.3. Mapas de ocupación	19
4. Objetivos, restricciones y recursos	21
4.1. Objetivos	21
4.2. Restricciones	22
4.2.1. Factores dato	22
4.2.2. Factores estratégicos	22
4.3. Recursos	23
4.3.1. Recursos software	23
4.3.2. Recursos humanos	23
4.3.3. Recursos hardware	24

5. Especificación de requisitos	27
5.1. Requisitos funcionales	27
5.2. Requisitos no funcionales	28
5.3. Descripción de la información	28
5.4. Descripción funcional	29
5.4.1. Casos de uso	30
5.4.2. Relación de requisitos funcionales con los casos de uso	35
5.5. Especificación de la interfaz	36
5.5.1. Ventana principal	37
5.5.2. Ventana ‘Set parameters’	41
6. Diseño del sistema	43
6.1. Diseño procedimental	43
6.1.1. Diagramas ASM	43
6.2. Estructura de ficheros y datos	49
6.2.1. Clase mainWindow	51
6.2.2. Clase parameters	53
6.2.3. Clase Simulation	55
6.2.4. Clases Position y Map	57
6.2.5. Clase Work	60
6.3. Diseño de los algoritmos de SLAM	63
6.3.1. Diseño del algoritmo de fuerza bruta	64
6.3.2. Diseño del algoritmo usando Levenberg Marquardt	65
6.4. Diseño de la interfaz	66
6.4.1. Ventana principal	67
6.4.2. Ventana ‘Set parameters’	68
7. Pruebas	71
7.1. Pruebas del software	71
7.1.1. Pruebas funcionales	71
7.2. Pruebas de los algoritmos	78
7.3. Discusión de los resultados.	82
8. Conclusiones	91
8.1. Conclusiones sobre los objetivos	91
8.2. Conclusiones sobre las pruebas	91
8.3. Futuras mejoras	92
9. Agradecimientos	95
Bibliografía	96

Índice de tablas

2.1. Distribución temporal del proyecto	12
5.1. Tabla genérica para los casos de uso.	29
5.2. Caso de uso C.U.1 - Creación de nuevo mapa	31
5.3. Caso de uso C.U.2 - Guardar mapa	31
5.4. Caso de uso C.U.3 - Guardar simulación	32
5.5. Caso de uso C.U.4 - Resetear aplicación	32
5.6. Caso de uso C.U.5 - Cargar simulación	33
5.7. Caso de uso C.U.6 - Modificar velocidad de simulación	33
5.8. Caso de uso C.U.7 - Comenzar/Parar simulación	34
5.9. Caso de uso C.U.8 - Pulsar un punto en el mapa	34
5.10. Caso de uso C.U.9 - Calcular distancia entre puntos	35
5.11. Caso de uso C.U.10 - Modificación de los parámetros	35
5.12. Relación de requisitos funcionales con los casos de uso	36
6.1. Definición de la clase.	51
6.2. Especificación de la clase.	51
6.3. Definición de la clase mainWindow.	52
6.4. Especificación de la clase mainWindow.	53
6.5. Definición de la clase parameters.	54
6.6. Especificación de la clase parameters.	55
6.7. Definición de la clase simulation.	56
6.8. Especificación de la clase simulation.	57
6.9. Definición de la clase Position.	58
6.10. Especificación de la clase Position.	58
6.11. Definición de la clase Map.	59
6.12. Especificación de la clase Map.	60
6.13. Definición de la clase work.	61
6.14. Especificación de la clase work.	62
7.1. Plantilla para documentar las pruebas	72
7.2. Caso de prueba C.P.1.1 Creación de nuevo mapa (flujo principal).	72
7.3. Caso de prueba C.P.1.2 Creación de nuevo mapa (flujos alternativos).	72
7.4. Caso de prueba C.P.2.1 Guardar mapa (flujo principal).	73
7.5. Caso de prueba C.P.2.2 Guardar mapa (flujos alternativos).	73
7.6. Caso de prueba C.P.3.1. Guardar simulación (flujo principal)	73
7.7. Caso de prueba C.P.3.2 Guardar simulación (flujos alternativos).	74
7.8. Caso de prueba C.P.4.1. Resetear aplicación (flujo principal)	74
7.9. Caso de prueba C.P.4.2 Resetear aplicación (flujos alternativos).	74

7.10. Caso de prueba C.P.5.1. Cargar simulación (flujo principal)	75
7.11. Caso de prueba C.P.5.2 Cargar simulación (flujos alternativos).	75
7.12. Caso de prueba C.P.6 Modificar velocidad de simulación (flujo principal). .	75
7.13. Caso de prueba C.P.7.1 Comenzar/Parar simulación (flujo principal).	76
7.14. Caso de prueba C.P.7.2 Comenzar/Parar simulación (flujos alternativos). .	76
7.15. Caso de prueba C.P.8 Pulsar un punto en el mapa (flujo principal).	77
7.16. Caso de prueba C.P.9.1 Calcular distancia entre puntos (flujo principal). .	77
7.17. Caso de prueba C.P.9.2 Calcular distancia entre puntos (flujos alternativos).	77
7.18. Caso de prueba C.P.10.1 Modificación de los parámetros (flujo principal). .	78
7.19. Caso de prueba C.P.10.2 Modificación de los parámetros (flujos alternati- vos).	78
7.20. Tabla de resultados para brute force.	82
7.21. Tabla de resultados para Levenberg Marquardt.	82

Índice de figuras

1.1. Distintos tipos de LiDARs[5][6]	3
1.2. Ejemplo gráfico del uso del sistema.	4
1.3. Creación del mapa de una zona de interior.	4
3.1. Descripción gráfica del problema SLAM	14
3.2. Autopilot de Tesla [14]	16
3.3. Diseño de un plano que se podría utilizar en ARcore para localizar el dispositivo.[16]	17
3.4. Mapa del hogar creado por un robot conga.[18]	18
3.5. Tecnología vSLAM de roomba.[19]	18
4.1. RPLIDAR A1M8 [6]	24
4.2. Mapa realizado con las medidas del LiDAR [6]	25
5.1. Funcionalidades accesibles para el usuario final y WBS.	30
5.2. Primer boceto de la interfaz de usuario.	38
5.3. Segundo boceto de la interfaz de usuario (colores).	39
5.4. Boceto final de la interfaz de usuario.	40
5.5. Boceto final de la ventana ‘Set parameters’.	41
6.1. Diagrama ASM de creación de nuevo mapa.	45
6.2. Diagrama ASM de simulación.	47
6.3. Diagrama ASM de ajuste de parámetros.	48
6.4. Diagrama ASM del cálculo de distancia entre dos puntos.	49
6.5. Jerarquía global de ficheros	50
6.6. Interfaz final de la aplicación.	67
6.7. Interfaz final de la ventana ‘Set parameters’ (Brute force activo).	68
6.8. Interfaz final de la ventana ‘Set parameters’ (Levenberg Marquardt activo).	69
7.1. Entorno de prueba vista 1.	79
7.2. Entorno de prueba vista 2.	79
7.3. Entorno de prueba vista 3.	80
7.4. Entorno de prueba vista 4.	80
7.5. Mapa generado por brute force con parámetros: 2 / 2 / 3 / 20 / 20 / 60.	84
7.6. Mapa generado por brute force con parámetros: 2 / 2 / 4 / 20 / 20 / 60.	85
7.7. Mapa generado por brute force con parámetros: 2 / 2 / 4 / 20 / 20 / 60.	86
7.8. Mapa generado por brute force con parámetros: 3 / 3 / 4 / 20 / 20 / 60.	87
7.9. Mapa generado por brute force con parámetros: 4 / 4 / 5 / 40 / 40 / 100.	88
7.10. Mapa generado por Levenberg Marquardt con parámetros: 3 / 3 / 5.	89

Índice de algoritmos

1. fit(try_P, current_dist[], theta[], count) 63
2. bruteForce(P, dist[], theta[], count, range_x, range_y, range_angle, x_var, y_var, angle_var) 65
3. error(sol[], err[], map, dist[], theta[], count) 65
4. levmarq(it, min_error, x_var, y_var, angle_var, map, dist[], theta[], count) 66

Capítulo 1

Introducción

Uno de los problemas al que se enfrenta el mundo actual de la robótica es la identificación de posibles obstáculos, identificación de la posición de la máquina, y, en general, el mapeo de una zona con todo lo que ello conlleva[1][2][3].

Para solventar este problema existen distintas posibles soluciones o aproximaciones, como se comentará en los antecedentes (mediante cámaras, mediante sensores láser, mediante grafos, etc.). El caso que se plantea en este TFG sería el mapeo 2D de interiores, bien sea para uso industrial, donde dicho sistema podría identificar obstáculos u otro tipo de necesidad, o bien sea para uso doméstico como por ejemplo los comúnmente conocidos robots de limpieza, entre otros usos.

Para ello utilizaremos una tecnología láser denominada LiDAR (light detection and ranging), que en entornos terrestres y teniendo en cuenta que hay situaciones donde la iluminación es pobre, aporta resultados más precisos y con menor coste computacional que las tecnologías de obtención de mapas de profundidad mediante imágenes, aunque con un mayor coste económico.[4]



Figura 1.1: Distintos tipos de LiDARs[5][6]

Esta tecnología se basa en el cálculo del tiempo que transcurre entre el lanzamiento

de un rayo láser y su vuelta al origen, obteniendo así la distancia entre el sistema y el objeto. En nuestro caso, dispondremos de un LiDAR el cuál es capaz de obtener hasta 2000 muestras por segundo girando constantemente para obtener los distintos puntos del entorno.

El objetivo general de este proyecto es el desarrollo de un sistema para obtener mapas del entorno utilizando tecnología láser de bajo coste. El sistema diseñado permitirá generar un mapa utilizando un sensor LiDAR que sera cogido en la mano por una persona que se ira moviendo en el entorno a medir para así realizar el mapeo. El sistema dispondrá de una interfaz gráfica de usuario para visualizar el mapa y manipularlo. El mapa generado podrá ser exportado para su posterior uso.

En la figura 1.2 se puede apreciar un ejemplo gráfico del uso del sistema, un humano llevará el sistema en la mano mientras se crea el mapa del entorno. En la figura 1.2 se puede ver cómo el sistema crea el mapa de la habitación en la que está el humano.



Figura 1.2: Ejemplo gráfico del uso del sistema.

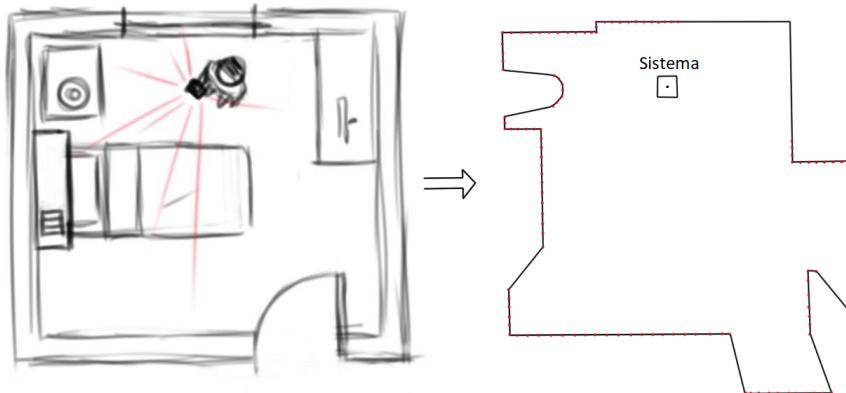


Figura 1.3: Creación del mapa de una zona de interior.

Como se puede apreciar, el sistema ignora al humano, prolongando las líneas de los

obstáculos encontrados a los lados del humano. A priori pensamos que no hay objetos detrás del humano, pero si los hubiese, el sistema lo reconocería cuando el humano se diese la vuelta. En este caso, por ejemplo, se reconocería el filo restante de la ventana y el resto del mueble.

1.1. Organización de la memoria

La presente documentación tratará de desglosar las especificaciones necesarias para llevar a cabo el desarrollo e implementación del sistema.

La documentación ha sido dividida como sigue:

- Definición del proyecto: se describirán las características (capítulo 2) y objetivos del proyecto (capítulo 4 sección 4.1), así como se aportará una solución concordante a dichos objetivos y restricciones (capítulo 5).
- Análisis del sistema: se especificarán las distintas partes que compondrán el sistema y se detallarán estas de forma que el sistema quede completamente definido (capítulo 6).
- Pruebas: se detallarán las pruebas que se realizarán sobre el sistema y los resultados obtenidos cuando dicho sistema sea implementado (capítulo 7).
- Conclusiones: se resumirá el trabajo realizado comentando los puntos más importantes y algunos de los problemas encontrados (capítulo 8 secciones 8.1 y 8.2).
- Mejoras y ampliaciones: se comentarán una serie de posibles mejoras y ampliaciones que han surgido a lo largo del desarrollo del proyecto, pero no están incluidos en los objetivos del proyecto, y por motivos técnicos o temporales no se han implementado (capítulo 8 sección 8.3).
- Bibliografía.

Capítulo 2

Definición del problema

En este capítulo se tratará de identificar y describir con detalle el problema al que nos enfrentamos. Para ello, se enfocará dicho problema desde dos perspectivas:

- Problema real: se hace referencia al problema general, que podría ser descrito por el usuario al que preocupa dicho problema.
- Problema técnico: se define el problema desde el punto de vista de la ingeniería, transformando el problema real en detalles técnicos que serán útiles para dar una solución a dicho problema.

2.1. Definición del problema real

Actualmente, y cada día más, surge la necesidad de identificar un sistema dentro de un entorno. También es importante, a su vez, conocer dicho entorno en ámbitos muy diversos como el industrial o el doméstico, como ya se comentaba en la introducción. Dicho problema se conoce en el mundo de la robótica como SLAM (simultaneous localization and mapping).

Este problema consiste en que un sistema debe ser capaz de localizarse a sí mismo dentro de un mapa que el propio sistema está creando con los datos obtenidos del entorno mediante sensores. El problema viene dado por la falta de sensores que nos digan exactamente dónde se encuentra un sistema dentro del mapa que está creando.

Existen diversas soluciones a dicho problema, como se comentará en los antecedentes, pero ninguna de ellas es válida para todos los casos. En nuestro caso, se utilizará un sistema láser para obtener datos del entorno y mediante software, se pretenderá contrastar los datos en cada medición para estimar la posición del sistema respecto al mapa en cada momento.

Con la realización de este TFG se pretende dar solución a un problema muy común en robótica, utilizando un sistema de bajo coste y consumo, que podrá ser implementado en diferentes situaciones del ámbito doméstico e industrial. Dicho sistema podrá ser utilizado tanto por expertos como por usuarios novicios.

2.2. Definición del problema técnico

Una vez definido el problema real, se tratará de convertir dicho problema en una serie de especificaciones técnicas que nos ayuden a afrontar el problema y obtener una solución desde el punto de vista de la ingeniería.

Para este propósito, se utilizará una técnica muy extendida en la gestión de proyectos que es la técnica PDS (Product Design Specification). Esta técnica consiste en realizar una serie de preguntas para responderlas y así obtener los detalles técnicos del problema.

2.2.1. Funcionamiento

En este apartado se explicará la funcionalidad del sistema que se va a diseñar, expresando de forma técnica las características del problema real.

El sistema que se diseñe deberá encargarse de recibir información del medio mediante un sistema de mediciones láser y procesar dicha información para realizar un mapa. Con la misma información de las medidas y el mapa, el sistema deberá ser capaz de localizarse a sí mismo utilizando algún algoritmo que le permita contrastar la información del mapa anterior con las mediciones actuales.

Actualización: ya que debido a las circunstancias de la época en la que nos encontramos (pandemia provocada por el coronavirus en 2020) no se ha podido utilizar una infraestructura con una raspberry, se ha optado por diseñar una interfaz gráfica de usuario. Por lo tanto, el sistema contará con una interfaz gráfica de usuario que permitirá elegir distintas opciones y parámetros para el mapeado, así como una opción para cargar simulaciones o guardar datos para crear simulaciones.

2.2.2. Entorno

Como entorno se entiende el conjunto de aspectos que rodean y condicionan al problema tanto desde el punto de vista software como hardware. A continuación se detallan una serie de características del entorno:

- Entorno de uso del software: este software está destinado a ser utilizado en general para crear mapas en tiempo real de zonas de interior, como pueden ser entornos industriales o domésticos. Esto permitirá al sistema localizarse a la vez que crea el mapa, lo que supondrá una ventaja para realizar tareas paralelas. El sistema se utilizará en sistemas linux dado que ofrece una mayor compatibilidad con librerías para el desarrollo de este tipo de proyectos.
- Entorno hardware: el sistema debe estar dotado de una batería para conferirle autonomía y poder realizar el mapeo sin necesidad de cables. Idealmente, el sistema debería ser un sistema pequeño que pueda ser portado por un humano para mapear mientras anda.
- Entorno de desarrollo: el sistema se diseñará en Visual Studio Code[7] dada su facilidad de uso y depuración para programas de terminal, con lo cuál se podrá depurar fácilmente cualquier error presente en el software durante el desarrollo.

- La parte de la interfaz de usuario se realizará en Qt Creator[8], una herramienta muy potente para diseñar interfaces de usuario tanto en sistemas linux como windows. En nuestro caso, como ya se ha comentado, se creará en un sistema linux. Esta herramienta permite grandes opciones de depuración y una gran facilidad de uso para crear y modificar elementos gráficos.

2.2.3. Vida esperada

La estimación de la vida esperada para un proyecto software es compleja, ya que, como se sabe, el ámbito informático sufre constantes cambios y avances día a día. Esta aplicación, dada su conexión con el LiDAR concreto que se va a utilizar, podría estimarse una vida esperada similar a la vida de dicho LiDAR. Sin embargo, si se modifica la conexión con el LiDAR, sería una aplicación perfectamente válida para cualquier tecnología láser. Además, esta aplicación proporcionará un algoritmo de SLAM que bien podrá servir de base para el diseño de otras aplicaciones. Una estimación aproximada podría ser de 4 a 6 años de vida.

2.2.4. Ciclo de mantenimiento

En este caso, se ha de tener en cuenta lo mencionado anteriormente. Ya que esta aplicación será concreta para un tipo de LiDAR (en este caso varios tipos del mismo fabricante), el ciclo de mantenimiento puede consistir tanto en el mantenimiento del software como del hardware. Se podría dividir en 3 tipos:

- Perfeccionamiento: consistirá en realizar mejoras al software o hardware para buscar una mayor eficiencia y preparar el sistema para cambios futuros, hacerlo más tolerante a los cambios en lugar de diseñar un sistema cerrado. Esto le aportará un aumento en la vida esperada.
- Adaptación: consiste en realizar una serie de modificaciones referentes a las necesidades que se planteen en los usuarios.
- Corrección: como en la mayoría de sistemas informáticos, existe la posibilidad de que el sistema contenga algún fallo. En ese caso, se tomarán las medidas necesarias para su corrección.

2.2.5. Competencia

Las distintas aplicaciones y sistemas similares a este se explicarán en mayor detalle en los antecedentes, pero en general, las tecnologías láser se utilizan en entornos 3D y rara vez se utilizan para crear mapas 2D. Otro tipo de sistemas que se utilizan para la creación de mapas y en general para SLAM, utilizan cámaras, pero suelen ser más costosos tanto computacional como económico. Por esto, este sistema permitiría a usuarios tanto expertos como usuarios novicios utilizar un sistema de bajo coste para crear un mapa de una superficie en 2D que podrían utilizar para distintos fines.

2.2.6. Aspecto externo

Ya que el sistema estará dotado de autonomía, se podría usar tanto una placa tipo raspberry como un ordenador personal portátil. En el caso de usarse una raspberry, sería conveniente utilizar algún tipo de estructura que facilite el transporte del sistema, para que la persona lo pueda manejar con facilidad y crear mapas a su antojo.

Para la presentación física de este proyecto, se usará un CD-ROM en el cuál irán incluidos todos los documentos necesarios para el uso e implementación del sistema. En caso de buscar una copia digital del proyecto, también se podrá encontrar en un repositorio de github.

La interfaz de usuario será amigable y tendrá unas facilidades de uso como pueden ser los botones naturales y la asociación de botones por funcionalidades. A su vez, se creará dicha interfaz en inglés para optar a una mayor acogida por la comunidad mundial.

2.2.7. Estandarización

Para el desarrollo del código del proyecto se utilizarán una serie de buenas prácticas que se llevan a cabo de forma general en los lenguajes orientados a objetos y en programación:

- Utilizar nombres de variables con nombres explicativos y que quede claro, sin ser muy extensos, cuál es la función de la variable.
- Tabular y en general ordenar el código de forma que sea comprensible y queden claras las diferencias entre partes.
- Relacionado con lo anterior, dividir el código en clases a la hora de un lenguaje orientado a objetos, permitiendo una mayor claridad y entendimiento del código.
- Documentar de forma adecuada el código, así como realizar comentarios cuando se estime oportuno, dejando clara la funcionalidad de todas las variables, funciones, clases, etc.

2.2.8. Calidad y fiabilidad

La calidad y fiabilidad de los proyectos de informática cumplen un papel fundamental debido a la integración de estos en la sociedad. En nuestro caso, esta calidad y fiabilidad se obtendrá gracias a dos medidas:

La primera, para obtener un software de calidad, es necesario recurrir a técnicas estándar como las comentadas en el apartado anterior. Por otro lado, es necesario a su vez realizar un buen diseño del software, que recae en la calidad del proyectista. También es importante, si se diseña un interfaz gráfica, consultar los detalles de diseño con un profesional, o en su defecto, con una persona perteneciente al campo del diseño gráfico.

Para obtener fiabilidad, se llevarán a cabo una serie de pruebas que pretenderán no ser exhaustivas (ya que los casos de prueba serían muy numerosos) pero sí probar gran parte del software con ciertas técnicas de ingeniería de fallos.

2.2.9. Programa de tareas

Este trabajo de fin de grado se organizará en las siguientes fases:

- Investigación: en esta fase se incluyen las tareas referentes a la investigación del problema, antecedentes (sistemas similares, posibles soluciones del estado del arte, etc.), la investigación y/o familiarización con las herramientas que se van a utilizar, así como las que se podrían utilizar (opencv, Qt, c++, ROS, etc.) para realizar una comparación entre estas y decidir qué herramientas serán las más adecuadas al problema.
- Análisis y especificación de requisitos: cuando el problema esté descrito, será el momento de pasar a la fase de análisis y especificación de requisitos, donde se especificará de forma más técnica las necesidades de nuestro problema y se podrá contrastar, consecuentemente, el resultado final del proyecto con los requisitos que se han definido.
- Diseño: cuando el problema haya quedado completamente analizado y especificado, se pasará a diseñar una solución que reúna las características que han sido especificadas para dar con la solución óptima al problema.
- Implementación: tras la definición, se implementará dicha definición en un sistema real.
- Evaluación y pruebas: el sistema real se someterá a unas pruebas previamente definidas que contrastarán el resultado del proyecto con las especificaciones y los requisitos definidos en las fases anteriores.
- Documentación: la documentación será una actividad paralela a todo el proyecto, que se realizará a medida que el desarrollo del mismo avance, y con ello sea necesario documentar el progreso.

Si bien es cierto que estas son las fases en las que se organizará el proyecto, bien es sabido que pocas veces la realidad es exactamente como hemos imaginado el proyecto, y generalmente hay que hacer nuevas investigaciones o especificaciones de requisitos conforme el desarrollo avanza. Es decir, estas fases no son fijas en el tiempo, sino que puede haber variaciones debido a problemas dinámicos que pueden aparecer. Sin embargo, el proyectista se compromete a realizar un estudio en profundidad para generar los menores cambios posibles en el futuro.

En la tabla 2.1 se puede ver la distribución temporal del proyecto organizada por fases.

Meses / Actividad	Investigación	Análisis y especificación de requisitos	Diseño	Implementación	Evaluación y pruebas	Documentación	Suma
Febrero	50	20				20	90
Marzo		10	60			20	90
Abril			20	50		20	90
Mayo				70		20	90
Junio					70	20	90
Horas	50	30	80	120	70	100	450

Tabla 2.1: Distribución temporal del proyecto

2.2.10. Pruebas

Las pruebas que se realizarán serán detalladas en el capítulo de pruebas. De forma general, se puede decir que las pruebas se realizarán tras la implementación del software o durante el desarrollo del mismo y serán de dos tipos:

- Caja blanca: este tipo de pruebas se realizan teniendo en cuenta la estructura interna del software para comprobar si se cumplen todas las especificaciones lógicas. Estas pruebas se realizarán durante el desarrollo del software para ir comprobando el progreso del proyecto.
- Caja negra: este tipo de pruebas se realizarán para comprobar el funcionamiento final del software completo. Estas pruebas se realizarán al finalizar el software para determinar si cumple las especificaciones y requisitos descritos en la memoria.

2.2.11. Seguridad

Este apartado, aunque forma parte de la especificación de la PDS, no es realmente importante en el proyecto que nos atañe ya que no será necesaria ninguna información personal del usuario. Por lo tanto, el único problema de seguridad que podría haber, sería una mala intencionalidad por parte del proyectista (extracción de datos del usuario sin su consentimiento, como podría ser, por ejemplo, su localización), y dado que el código del software se podrá revisar, se podrá ver que no existe tal intencionalidad.

Capítulo 3

Antecedentes

Como ya se comentó en la introducción, la tecnología LiDAR se usa en diversos campos y aplicaciones. Este apartado se dividirá a su vez en tres subapartados: primero se explicará el problema del mapeo y localización simultáneos (SLAM) en robótica y qué métodos de resolución o técnicas se pueden aplicar. Después, se expondrán ejemplos de sistemas similares al que se pretende construir. Por último, se comentarán distintas aproximaciones a los mapas de ocupación.

3.1. SLAM y métodos de resolución

Uno de los principales problemas de la robótica consiste en el mapeo y localización simultáneos, SLAM, de sus siglas en inglés simultaneous localization and mapping. Este problema consiste en realizar un mapa del entorno por el que un robot se mueve a la vez que estima su trayectoria al desplazarse por ese entorno.

Este problema, dicho de otra forma, trata de colocar a un robot en un entorno y que sea capaz de crear un mapa de dicho entorno, situándose a sí mismo dentro de este mapa y utilizar dicho mapa a su vez para el propósito de situarse. Ambos conceptos (mapeo y localización), fueron en un principio entendidos como independientes, pero a lo largo de la historia nos hemos dado cuenta que son dependientes.[9]

Como se puede apreciar en la figura 3.1, extraída de [10], es necesario estimar la posición del robot y de las marcas, ya que estas posiciones nunca son medidas directamente.

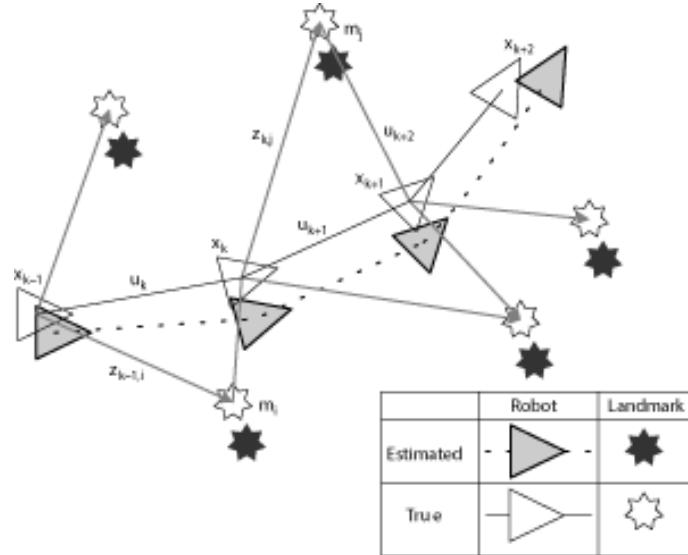


Figura 3.1: Descripción gráfica del problema SLAM

Este problema ha sido resuelto de formas muy distintas, pero no existe una solución óptima aplicable a todos los casos. Algunas de las soluciones que plantea el estado del arte consisten en usar cámaras, GPS, sonar o sensores láser.

Los sensores que debe poseer un sistema para solventar el problema de SLAM podrían clasificarse en dos: propioceptivos y exteroceptivos. Los sensores exteroceptivos son los ya comentados, cámaras, GPS, etc. y proporcionan información acerca del entorno en el que se encuentra el sistema. Los sensores propioceptivos proporcionan información acerca de la velocidad a la que se mueve el sistema, cambios de posición y aceleración, y son sensores como: codificadores, acelerómetros, giroscopios, etc.[9]

Una de las aproximaciones, de forma general, es resolver este problema utilizando cámaras, lo que se conoce como visual SLAM. Dentro de este método, existen distintas formas de obtener la solución. Normalmente, se clasifican en soluciones mono o estéreo. Esto quiere decir que una solución monoSLAM utilizaría una sola cámara para resolver el problema, mientras que una solución estéreo utilizaría varias cámaras con sus campos de visión parcialmente solapados. MonoSLAM se basa en la utilización de imágenes anteriores en el tiempo para solaparlas con las nuevas, pero se basa en la hipótesis de que la velocidad angular y lineal del sistema serán constantes, lo cual no permite giros bruscos de cámara.[9]

Según [9], esta aproximación se utiliza ya que suele ser más barata y ocupar menos espacio que otras opciones como pueden ser utilizar sensores láser o sonar. Otra de las ventajas que brinda la utilización de esta técnica es la obtención de colores y texturas, lo que permitiría al sistema realizar otro tipo de tareas de más alto nivel, a parte de su menor consumo energético. Sin embargo, el uso de cámaras presenta una serie de problemas, y que existen muchas situaciones (cambios de luz, cámaras con poca resolución, superficies con poca textura, imágenes difuminadas, etc.) en las que los resultados obtenidos por este método son poco precisos.

Otra aproximación, independiente del tipo de sensor usado, como se comenta en [11],

consiste en representar el problema en un grafo, donde los nodos representan las poses del robot en diferentes tiempos y puntos, y las conexiones representan limitaciones entre las poses. Una vez que el grafo ha sido construido, el mapa se construye de forma que la configuración espacial de los nodos es más consistente con respecto a las limitaciones de las conexiones.

Se puede observar que hay numerosas formas de enfrentar este problema, pero no se puede determinar que ninguna sea óptima para todos los casos. Como se comenta en [9], la mayoría de soluciones, independientemente del tipo de sensor utilizado, son soluciones probabilísticas. Se basa en la reducción de la incertidumbre sobre la posición del sistema con respecto al mapa.

3.2. Sistemas similares al que se pretende construir

Muchas veces, esta tecnología LiDAR se usa conjuntamente con redes CNN (convolutional neural network [12]) para obtener predicciones de distinta índole, como se puede ver en los siguientes artículos:

Tanto en [1] como en artículos citados en este mismo artículo, la tecnología LiDAR es utilizada para obtener un mapa del manto sobre el terreno de vegetación en zonas pequeñas con gran precisión, que se integra posteriormente con un modelo de CNN para obtener características espectrales y de textura para realizar una regresión de la altura de la vegetación per-pixel.

En [2], se usa la tecnología LiDAR para obtener una nube de puntos y un mapa de densidad de una posible área de aterrizaje para helicópteros. Este mapa de densidad después es procesado por una CNN para calcular la probabilidad de buen aterrizaje en ese área.

Pasando ahora a sistemas más similares a nuestro objetivo, tenemos ciertos sistemas orientados a robótica:

En [13] utilizan la tecnología LiDAR para crear mapas de profundidad 3D y posteriormente aplicar distintas técnicas de análisis de escenas 3D de visión artificial como puede ser la selección de vecindario, extracción y selección de características y clasificación, siendo así posible identificar objetos y asignar etiquetas para temas de fotogrametría, robótica o visión artificial.

Existe también un ejemplo de conducción autónoma que no utiliza la tecnología LiDAR, sino que simplemente obtiene información 3D mediante 8 cámaras externas que captan imágenes 2D y una red CNN que interpreta dichas imágenes para obtener un mapa de profundidad. Este es el caso de los automóviles Tesla [14]. Si bien es cierto que los sensores LiDAR son menos económicos que las cámaras RGB convencionales, al implementar 8 cámaras y un ordenador lo suficientemente potente como para implementar dicha red neuronal, entre otros, puede que el coste de implementación de un sistema con tecnología LiDAR con esas capacidades no difiera en gran medida del coste de implementación de este sistema.



Figura 3.2: Autopilot de Tesla [14]

Otro de los métodos que se podría usar para resolver o aplicar SLAM es el ARcore de google [15]. Se trata de un kit de desarrollo (SDK) que dispone de gran cantidad de funcionalidades de realidad virtual como pueden ser: añadir elementos a un entorno virtual, interaccionar con ellos, detectar distintas propiedades en las superficies reales, seguir el desplazamiento de tu dispositivo en un entorno, etc. En este último, es donde se podría incluir el término SLAM. Sin embargo, mediante la API solo se puede obtener la nube de puntos 3D y la localización del dispositivo, por lo que habría que utilizar dicha API para obtener los datos y crear un mapa. Este mapa además sería 3D y habría que añadir procesamiento para convertirlo en 2D si se desea. Como ya se comentaba al principio, .

En [16] diseñan un plano de un mapa y usan la API para luego localizarse en dicho plano, pero ARcore no estaría funcionando aquí para la creación del mapa, sino que sería creado a priori.

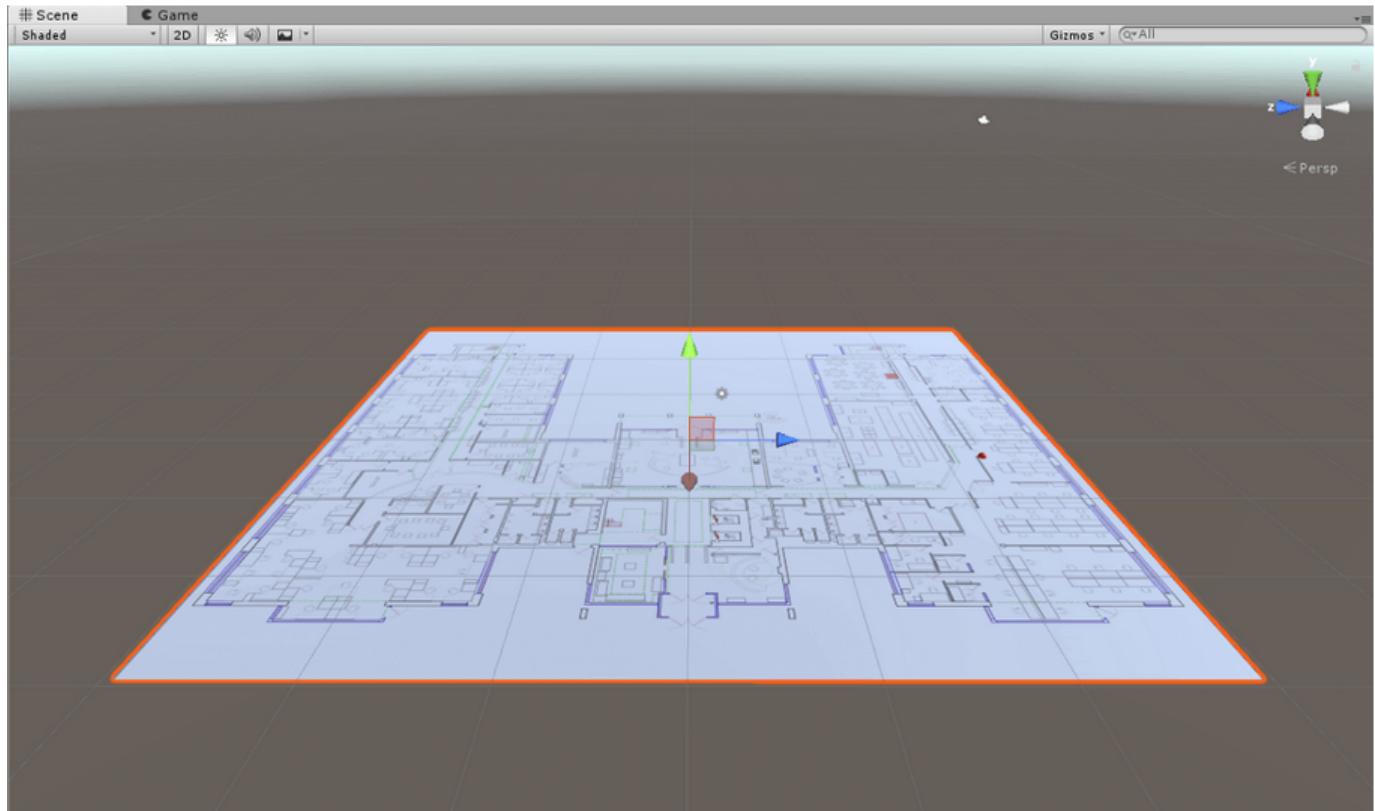


Figura 3.3: Diseño de un plano que se podría utilizar en ARcore para localizar el dispositivo.[16]

ARcore además cuenta con el inconveniente de necesitar ciertas funcionalidades de los sensores del dispositivo para localizarse, por lo que hay dispositivos en los que no está disponible.

Otro escenario donde la tecnología LiDAR y el problema SLAM juegan un papel importante es en el ámbito del hogar. Son ya muy conocidos los robots de limpieza que barren, friegan y en general mantienen el hogar limpio. Muchos de estos robots no cuentan con la tecnología LiDAR y no son capaces de crear un mapa del entorno, aunque algunos cuentan con dicha tecnología y pueden realizar SLAM. Es el caso por ejemplo de los robots conga[17]. Si nos fijamos en la web, la mayoría de ellos cuentan con una serie de funcionalidades básicas en robótica como la esquiva de obstáculos mediante sensores de proximidad o evitar caídas por cambios de profundidad, pero solo los robots de alta gama cuentan con tecnología LiDAR para crear un mapa de la zona y ubicarse dentro de ella.

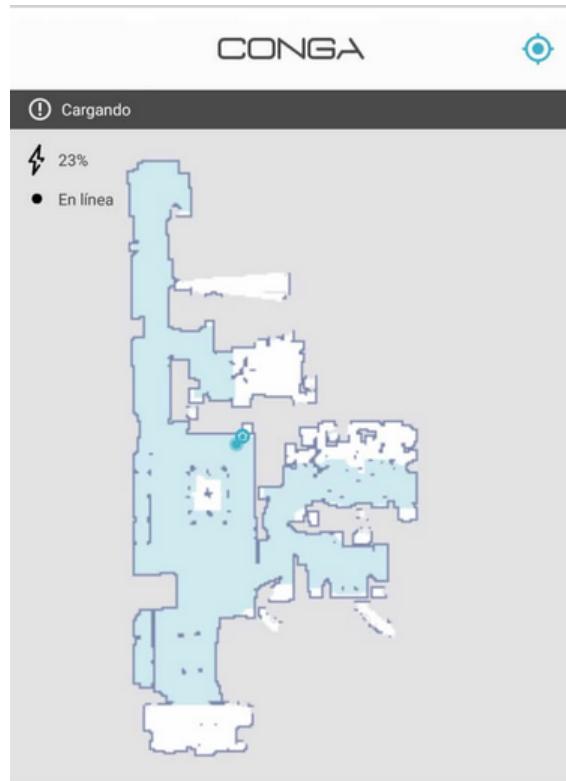


Figura 3.4: Mapa del hogar creado por un robot conga.[18]

Existen otros robots de limpieza como roomba que utilizan una técnica ya comentada, visual SLAM. Estos robots obtienen 230.400 puntos por segundo del entorno y generan un mapa en función de estos puntos y de una tecnología llamada vSLAM diseñada por irobot[19].



Figura 3.5: Tecnología vSLAM de roomba.[19]

Como último ejemplo, existen versiones de SLAM desarrolladas en ROS (Robot Operating System [20]) que utilizan tecnología LiDAR para crear los mapas. ROS es un entorno de desarrollo que está específicamente diseñado para desarrollar y manipular robots. Este entorno está muy aceptado y extendido en la comunidad de robótica dado que dispone de muchas utilidades como pueden ser la integración de distintas tecnologías por parte de los fabricantes (como muchos tipos de LiDAR) o simplemente una serie de funcionalidades para facilitar el desarrollo de proyectos: contempla una alta modularidad y distribución, permite el uso de una interfaz gráfica, dispone de herramientas para editar los distintos ficheros y directorios del proyecto de forma sencilla, etc.

En ROS por ejemplo está diseñado hector SLAM [21], una solución a SLAM utilizando LiDAR cuya tasa de actualización es bastante alta (según la documentación, 40Hz). En nuestro caso, se ha descartado la opción de usar ROS ya que, según las pruebas que se han realizado con este entorno en el portátil que se usará, ROS consume muchos recursos del sistema, lo que no lo hace una aproximación atractiva para los objetivos del proyecto.

Podemos ver, por tanto, que la mayoría de los sistemas que utilizan la tecnología LiDAR, y aún más los que utilizan la técnica de visual SLAM, pretenden identificar objetos 3D por diversos motivos (entre ellos, por ejemplo, la conducción autónoma o la altura de la vegetación, pero también la medición de los aerosoles atmosféricos [22][23]).

El sistema que se pretende crear en este TFG desecha la idea de obtener mapas de profundidad 3D y opta por una solución más sencilla y óptima para la situación que se plantea, que es simplemente obtener un mapa 2D para encontrar obstáculos, independientemente del obstáculo que se encuentre. Esto permite utilizar sensores LiDAR de bajo coste cuyo precio es similar al de una cámara RGB convencional. Así pues, se obtendrían mejores resultados para SLAM con una menor carga computacional, y mapas muy fácilmente comprensibles.

3.3. Mapas de ocupación

En [24] presentan los conocidos como “Stochastic Map” o mapas de probabilidad. En este artículo, estos mapas pretenden representar la probabilidad de que un objeto esté en un cierto punto concreto, pero basándose en los probables fallos de mediciones. En nuestro caso, aunque también existan fallos, las mediciones del LiDAR son bastante precisas y las mayores diferencias de probabilidad de ocupación con respecto a una casilla se darán por errores en la estimación de la pose (posición y orientación) del sistema.

En [25] la autora expone distintos tipos de mapas que podrían usarse para el mapeo y la localización de un robot en dicho mapa (en su caso usa un sensor de ultrasonidos, pero el tipo de sensor no impide el uso de estos mapas, a excepción de uno de ellos, el geométrico). Tales mapas son los siguientes:

- Mapas de rejilla: consisten en subdividir el entorno en una serie de celdas (como bien podrían ser los píxeles de una pantalla) y asignar una probabilidad de ocupación a cada celda (o probabilidad de encontrar un objeto en dicha celda). Estos mapas cuentan con la ventaja de que son simples de implementar y mantener incluso en entornos grandes o complejos. Además, es muy fácil representar la posición del

sistema dentro de estos mapas, ya que se corresponderá con las casillas que este ocupe en el mundo real. El mayor inconveniente es la gran cantidad de memoria necesaria para almacenarlos.

- Mapas topológicos: consisten en un grafo en el que los nodos representan posibles posiciones del sistema y las conexiones representan limitaciones, que bien pueden ser distancias. El principal inconveniente de este tipo de mapas es su dificultad para ser representados, la posible ambigüedad de escenarios y la baja precisión que ofrecen con respecto a la representación del sistema, ya que no es necesario conocer la posición exacta.
- Mapas geométricos: este tipo de mapas son creados a partir de la obtención de las geometrías de los obstáculos (esquinas cóncavas, convexas, muros, etc.) y estas geometrías son obtenidas a través de la reflexión de las ondas de sonido en dichos obstáculos.

En nuestro caso, el mapa de ocupación implementado será un mapa de rejilla dinámico, que permitirá la detección de objetos en tiempo real, pudiendo dichos objetos aparecer y desaparecer en el mapa. Se eligieron este tipo de mapas por su facilidad de representación y manejo, pudiendo manejarse como una simple imagen.

El mapa por tanto será un mapa de probabilidades, teniendo un valor alto los píxeles que más rayos consecutivos concentren. Las probabilidades se calcularán dado un peso para el valor anterior y un peso para el valor nuevo. Ambos valores se sumarán y ese será el nuevo valor de probabilidad. Esto permitirá no caer en errores de medidas si una medida aparece únicamente una vez.

Capítulo 4

Objetivos, restricciones y recursos

4.1. Objetivos

En líneas generales, el objetivo de este proyecto es diseñar un sistema que sea capaz de crear un mapa 2D de una zona de interior utilizando tecnología láser, así como una interfaz gráfica de usuario para facilitar la utilización por parte de un humano del sistema de mapeo.

Para ello, es necesario que se cumplan los siguientes objetivos:

- El sistema debe ser capaz de obtener información del medio mediante la tecnología láser LiDAR.
- La estructura de datos se deberá poder imprimir para obtener una representación visual del mapa.
- El sistema podrá ser portado por un humano que se moverá libremente por el entorno para realizar el mapeado.
- El sistema dispondrá de un parámetro para evitar mapear al humano cuando éste porte dicho sistema. Los obstáculos encontrados a los lados del humano de prolongarán, pensando a priori que no hay obstáculos detrás del humano más que las prolongaciones de los ya existentes.
- El sistema estará compuesto por el LiDAR, un sistema de procesamiento y una batería para conferirle autonomía.
- El sistema mostrará en tiempo real el mapa generado del entorno.
- El sistema deberá ser capaz de localizarse a si mismo dentro del mapa.
- El mapa generado podrá ser exportado para su posterior uso.
- El sistema dispondrá de una interfaz gráfica de usuario
- La interfaz será amigable e intuitiva.
- La interfaz implementará como mínimo las opciones para crear un mapa real y probar simulaciones.

4.2. Restricciones

Las restricciones presentes en el ámbito del diseño se dividirán en dos tipos (factores dato y factores estratégicos) y no se podrán modificar durante el desarrollo

4.2.1. Factores dato

Son factores dato aquellas restricciones que vienen definidas por la naturaleza o el ámbito del problema, o por el cliente que desee el sistema, y no existen posibilidades a la hora de elegir. Dichas restricciones serían las siguientes para nuestro proyecto:

- Dado que, por decisión del director, se utilizará el LiDAR RPLIDAR A1M8, se habrá de utilizar su kit de desarrollo para el desarrollo del proyecto.
- El sistema deberá cumplir los objetivos previamente especificados.
- El sistema será destinado como mínimo a usuarios que utilicen el sistema operativo ubuntu 18.04, siendo posible optar a mayor compatibilidad.
- El desarrollo debe tener una duración aproximada de 450h.
- El sistema, como se especifica en otros apartados, debe ser de bajo coste y consumo, aunque no existan limitaciones formales con respecto a estos.
- Las librerías utilizadas serán de código abierto, para permitir un menor coste y una mayor compatibilidad con versiones futuras, así como mayor adaptación y demás ventajas que supone utilizar código abierto.
- Los recursos utilizados serán de los que dispongan tanto el director del trabajo de fin de grado como el proyectista.
- El sistema contará con un control de errores que informe al usuario de la causa del problema ocurrido.

4.2.2. Factores estratégicos

Los factores estratégicos son aquellas restricciones que se plantean a la hora de diseñar y desarrollar el proyecto, pero que entre ellas existen una serie de posibilidades. Son dichas restricciones para nuestro proyecto las siguientes:

- Como lenguaje de programación solo hay dos opciones (C y C++) dado que el software de desarrollo del LiDAR está escrito en C. Se usará C++ debido a su facilidad de integración con distintas herramientas como ROS, opencv y el propio software de desarrollo del LiDAR.
- Como entorno programación se utilizará Visual Studio Code por razones ya descritas (facilidad de depuración, de obtención de datos, etc.).

- Como sistema operativo se usará linux, concretamente la distribución de ubuntu, dada la familiaridad del proyectista con dicho sistema y la facilidad de instalación de las librerías y frameworks necesarios. Además, es un sistema operativo gratuito.
- Para la elaboración de la documentación se ha optado por utilizar overleaf, un editor de LaTeX gratuito en línea que permite control de versiones y almacenamiento en la nube. Para la presentación, se utilizará la herramienta gratuita de google presentaciones.
- Como entorno de desarrollo para la interfaz de usuario se utilizará Qt creator, un potente framework de desarrollo para aplicaciones gráficas, que cuenta con muchas ayudas de depuración, así como señalizaciones y ayudas a la hora de codificar para facilitar el desempeño y conseguir un menor número de errores sintácticos y por lo tanto un menor tiempo empleado. Qt además podría utilizarse para desarrollar aplicaciones para windows y es de código abierto y gratuito.

4.3. Recursos

Para el desarrollo del proyecto se emplearán los siguientes recursos. Estos recursos pueden estar sujetos a cambios, que se notificarán en la memoria del proyecto:

4.3.1. Recursos software

Los siguientes recursos software se utilizarán para el desarrollo, pruebas y documentación del proyecto:

- **Sistema operativo:** Ubuntu LTS 18.04, ubuntu LTS 20.04.
- **Librerías y frameworks:** opencv, Qt, RPLIDAR sdk, levmarq.
- **Lenguaje de programación:** C++.
- **Entornos de desarrollo:** Visual Studio Code, Qt creator.
- **Editor de texto:** para la documentación se utilizará el editor de texto en línea overleaf y dicha documentación se creará mediante L^AT_EX.
- **Otros:** se utilizó la herramienta online de visual paradigm [26] para crear el diagrama WBS.

4.3.2. Recursos humanos

Para el desarrollo de este proyecto se requerirá la participación del proyectista Manuel Rafael Navarro Fuentes, el cual se encargará del diseño, implementación y, en general, desarrollo del proyecto y del director del proyecto Rafael Muñoz Salinas, que se encargará de la supervisión del avance del proyecto y aportará ideas y correcciones al mismo.

4.3.3. Recursos hardware

Para desarrollar el programa que realice SLAM y la interfaz gráfica de usuario para utilizarlo, se hará uso de un portátil personal con las siguientes características:

- **Procesador:** APU AMD Quad-Core A8-3530MX (1.9GHz, 4MB L2 Cache)
- **Memoria RAM:** 6GB DDR3
- **Sistema de almacenamiento:** 750 GB (5400 rpm S-ATA)

Para el sistema de mapeo se utilizará el LiDAR: RPLIDAR A1M8 de slamtec[6]. Este LiDAR, como se puede ver en la página web, dispone de un rango de detección máximo aproximado de 12, y mínimo de unos 15cm, lo cuál para entornos de interior normalmente es más que suficiente. Su capacidad de procesamiento es de 8000 muestras por segundo y se puede variar según las necesidades del usuario.

El LiDAR en cuestión es el siguiente:



Figura 4.1: RPLIDAR A1M8 [6]

Cada muestra medida con el LiDAR contiene 4 datos:

- Angle_z_q14: ángulo desde el que se tomó la medida en representación z.
- Dist_mm_q2: distancia en milímetros desde el LiDAR hasta un obstáculo.
- Quality: calidad de la medida.
- Flag: syncbit.

Estas medidas permiten representar puntos en un mapa como se puede ver en la siguiente imagen:

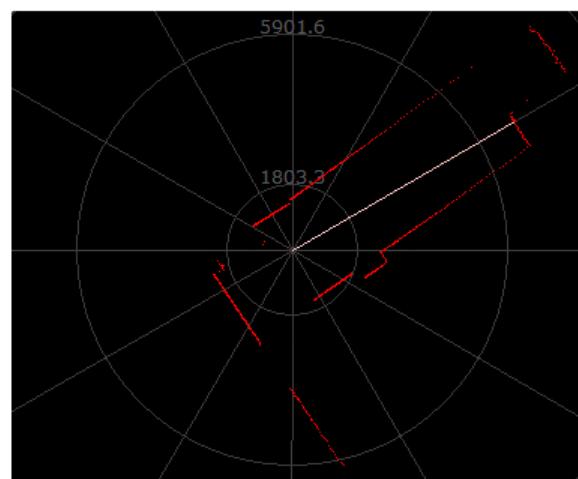


Figura 4.2: Mapa realizado con las medidas del LiDAR [6]

Capítulo 5

Especificación de requisitos

En este capítulo se detallarán los requisitos necesarios para implementar el sistema de acuerdo a las especificaciones vistas en los capítulos anteriores. Se detallarán los requisitos funcionales, los no funcionales, la descripción de la información y la descripción funcional.

5.1. Requisitos funcionales

- RF-1 La aplicación debe permitir la creación de nuevos mapas.
- RF-2 La aplicación debe permitir guardar los mapas creados cuando se desee.
- RF-3 La aplicación debe ser capaz de resetearse cuando el usuario lo desee, siempre que la creación del mapa haya finalizado.
- RF-4 La aplicación debe ser capaz de permitir al usuario modificar los parámetros que se utilizan para crear los mapas.
- RF-5 El guardado del mapa debe generar una imagen PNG con dicho mapa y el nombre que el usuario desee.
- RF-6 La aplicación debe disponer una vista previa del mapa en tiempo real.
- RF-7 La aplicación debe mostrar en la vista previa del mapa el sistema de mapeo y su orientación.
- RF-8 El programa debe disponer de una interfaz gráfica
- RF-9 La aplicación debe permitir al usuario cargar una simulación.
- RF-10 La aplicación debe permitir al usuario guardar una simulación.
- RF-11 La aplicación debe permitir al usuario calcular la distancia entre dos puntos que el usuario decida.
- RF-12 La aplicación debe permitir al usuario modificar la velocidad de simulación.
- RF-13 La aplicación debe permitir al usuario parar y continuar la simulación cuando este lo decida.
- RF-14 La aplicación debe permitir al usuario guardar una imagen del mapa simulado.

- RF-15 La aplicación no permitirá crear un nuevo mapa cuando una simulación está en curso y viceversa.
- RF-16 La aplicación no permitirá cargar una simulación cuando otra está activa.
- RF-17 La aplicación no permitirá modificar los parámetros cuando una simulación esté activa y no pausada o un mapa esté siendo creado.

5.2. Requisitos no funcionales

- RNF-1 El sistema deberá ser seguro, eficaz, robusto y fiable.
- RNF-2 El sistema dispondrá de un mantenimiento sencillo.
- RNF-3 El sistema debe poder ser usado tanto por un usuario experto como por un novicio.
- RNF-4 El sistema debe tener una interfaz sencilla, intuitiva y manejable.
- RNF-5 El sistema debe responder antes los errores que se produzcan en tiempo de ejecución.
- RNF-6 El sistema será ejecutado en un sistema operativo linux.
- RNF-7 El tiempo de respuesta entre cada frame en la creación del mapa deberá ser aceptable para el usuario.
- RNF-8 El coste del desarrollo debe ser mínimo.
- RNF-9 El coste del sistema físico debe ser mínimo.
- RNF-10 Se evitará el uso de software privativo.

5.3. Descripción de la información

La información necesaria para solventar el problema que usará el sistema se podría dividir en dos apartados:

- Lecturas del LiDAR: estas lecturas contendrán toda la información necesaria para representar puntos en el mapa de los obstáculos encontrados. Estos datos los proporcionará el LiDAR haciendo uso de su kit de desarrollo.
- Parámetros necesarios para los algoritmos: estos parámetros tendrán unos valores por defecto para que el usuario novicio no tenga que preocuparse de ajustarlos, mientras que el usuario experto podrá modificar dichos parámetros a su antojo para obtener un mapa a medida. Los parámetros serían los siguientes:
 - Posición inicial: posición inicial del sistema con respecto al mapa.

- Tamaño del mapa.
- Peso del último valor del pixel: este peso sirve para ajustar las probabilidades de ocupación de cada pixel.
- Algoritmo: permitirá elegir qué algoritmo usar a la hora de realizar SLAM.
- Variación de X: variación de X entre dos estimaciones.
- Variación de Y: variación de Y entre dos estimaciones.
- Variación del ángulo: variación del ángulo entre dos estimaciones.
- Rango de X: rango de X en el que se probarán las soluciones para SLAM.
- Rango de Y: rango de Y en el que se probarán las soluciones para SLAM.
- Rango del ángulo: rango del ángulo en el que se probarán las soluciones para SLAM.
- Baudrate: baudrate que usará el driver del LiDAR para conectarse con este.
- Puerto: puerto al que estará conectado el LiDAR.

El fichero que se utilizará para la simulación tendrá la siguiente forma (en bucle) y será un fichero TXT:

- Ángulo: ángulo desde el que se tomó la medida.
- Distancia: desde el LiDAR hasta un obstáculo.
- Calidad de la medida.
- Flag: syncbit.

5.4. Descripción funcional

En esta sección se detallarán los casos de uso para describir la funcionalidad del sistema de la forma en la que lo percibirán los usuarios finales. La tabla 7.1 se utilizará para describir todos los casos de uso. Como se puede observar, no dispone uno de los campos típicos para definir los casos de uso. Esto se debe a que nuestro sistema solo dispondrá de un actor, el usuario final, por lo que las tablas obviarán este detalle.

CÓDIGO - NOMBRE DEL CASO DE USO	
Descripción	Función del caso de uso.
Precondición	Condiciones necesarias para que se ejecute el caso de uso.
Flujo principal	Flujo natural que desarrollará el caso de uso.
Postcondición	Condiciones en las que debe quedar el sistema tras el caso de uso.
Flujo alternativo	Flujos que podrían ejecutarse en este caso de uso.

Tabla 5.1: Tabla genérica para los casos de uso.

Para dar una visión general de los casos de uso del sistema, se plantea el diagrama de la figura 7.1, obtenido con la herramienta online gratuita de visual paradigm[26]. Este diagrama se podría utilizar como WBS (work breakdown structure) ya que nuestro sistema realmente no presenta una profundidad de casos de uso anidados. Este diagrama ha sido modificado para incluir los casos de uso de la interfaz gráfica.

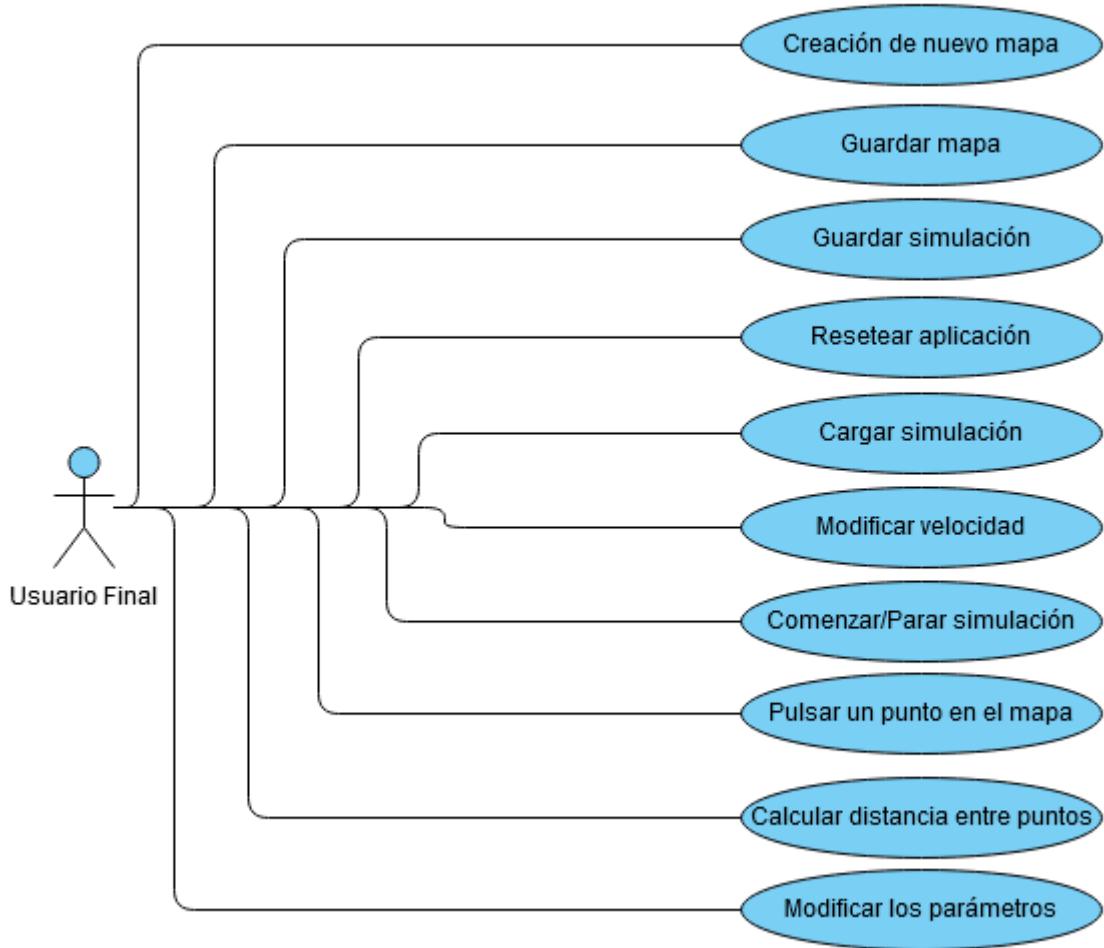


Figura 5.1: Funcionalidades accesibles para el usuario final y WBS.

5.4.1. Casos de uso

Los casos de uso serían los siguientes:

C.U.1 - Creación de nuevo mapa	
Descripción	Se iniciará la creación de un nuevo mapa en tiempo real.
Precondición	<ol style="list-style-type: none"> 1. No se puede haber iniciado otro mapa. 2. No se puede haber empezado una simulación. 3. No se puede haber finalizado un mapa y no haber reseteado. 4. El LiDAR debe estar conectado 5. El puerto y el baudrate especificados deben ser correctos.
Flujo principal	<ol style="list-style-type: none"> 1. El LiDAR se conectará automáticamente a la aplicación mediante el driver correspondiente. 2. Se iniciará la creación de un mapa en tiempo real, mostrándose una previsualización del mapa.
Postcondición	<ol style="list-style-type: none"> 1. El estado de la aplicación pasará a "creando mapa real". 2. El LiDAR estará conectado a la aplicación mediante el driver.
Flujo alternativo	Si no se cumple alguna de las precondiciones, se mostrará un mensaje de error y el estado del sistema continuará como antes del caso de uso.

Tabla 5.2: Caso de uso C.U.1 - Creación de nuevo mapa

C.U. 2 - Guardar mapa	
Descripción	Se guardará un mapa con el formato PNG y el nombre que desee el usuario.
Precondición	<ol style="list-style-type: none"> 1. Debe haberse iniciado la creación de un mapa real. 2. No debe haber una simulación activa. 3. No debe estar finalizado el estado de la creación de mapa real.
Flujo principal	<ol style="list-style-type: none"> 1. El programa mostrará un cuadro de diálogo para seleccionar la ubicación y el nombre del archivo a guardar. 2. El usuario introducirá dicho nombre y ubicación y clicará aceptar.
Postcondición	<ol style="list-style-type: none"> 1. El mapa quedará guardado en la ubicación y con el nombre deseados. 2. El mapa en tiempo real se detendrá y se mostrará la última imagen del mapa en la pantalla.
Flujo alternativo	Si no se cumple alguna de las precondiciones, el sistema mostrará un mensaje de error y mantendrá su estado.

Tabla 5.3: Caso de uso C.U.2 - Guardar mapa

C.U.3 - Guardar simulación	
Descripción	Los datos del LiDAR que se han ido almacenando mientras se creaba el mapa real serán guardados en la ubicación y con el nombre que el usuario desee, en formato TXT.
Precondición	<ol style="list-style-type: none"> 1. El proceso de creación del mapa real debe estar en el estado "finalizado". 2. No se puede estar corriendo una simulación.
Flujo principal	<ol style="list-style-type: none"> 1. El programa mostrará un cuadro de diálogo para seleccionar la ubicación y el nombre del archivo a guardar. 2. El usuario introducirá dicho nombre y ubicación y clicará aceptar.
Postcondición	El fichero de simulación quedará guardado en la ubicación y con el nombre deseados.
Flujo alternativo	Si no se cumple alguna de las precondiciones, se mostrará un mensaje de error y el sistema mantendrá su estado.

Tabla 5.4: Caso de uso C.U.3 - Guardar simulación

C.U.4 - Resetear aplicación	
Descripción	Reseteará la aplicación para poder iniciar cualquier acción como si dicha aplicación se acabase de iniciar.
Precondición	<ol style="list-style-type: none"> 1. Debe haberse ejecutado una simulación o creado un mapa real. 2. No puede haber un mapa real en estado de creación. 3. No puede haber una simulación corriendo (debe estar parada o finalizada).
Flujo principal	La aplicación ejecutará una serie de funciones para limpiar la memoria y restablecer los valores predeterminados.
Postcondición	El sistema debe quedar como al inicio de la aplicación.
Flujo alternativo	Si no se cumple alguna de las precondiciones, se mostrará un mensaje de error y el sistema mantendrá su estado.

Tabla 5.5: Caso de uso C.U.4 - Resetear aplicación

C.U.5 - Cargar simulación	
Descripción	Se guardarán en un buffer de la aplicación los datos de simulación proporcionados por un fichero que elija el usuario.
Precondición	<ol style="list-style-type: none"> 1. No puede haber un mapa real en estado de creación. 2. No puede haber un mapa real en estado de finalizado. 3. No puede haber una simulación corriendo. 4. No puede haber una simulación parada.
Flujo principal	<ol style="list-style-type: none"> 1. Se abrirá un cuadro de diálogo para que el usuario escoja el archivo que desea simular. 2. Los datos de simulación se guardarán en un buffer.
Postcondición	<ol style="list-style-type: none"> 1. Los datos de simulación quedarán listos para iniciar una simulación. 2. Se podrá activar el botón de comenzar una simulación.
Flujo alternativo	Si no se cumple alguna de las precondiciones, se mostrará un mensaje de error y el sistema mantendrá su estado.

Tabla 5.6: Caso de uso C.U.5 - Cargar simulación

C.U.6 - Modificar velocidad de simulación	
Descripción	Se modificará la velocidad a la que se ejecuta la simulación, es decir, la velocidad a la que los frames del mapa serán actualizados.
Precondición	Ninguna
Flujo principal	El programa modificará la variable de velocidad para que se modifique la velocidad de los frames de actualización del mapa en simulación.
Postcondición	La variable velocidad debe quedar modificada y el mapa simulado se deberá actualizar a la velocidad especificada.
Flujo alternativo	Ninguno

Tabla 5.7: Caso de uso C.U.6 - Modificar velocidad de simulación

C.U.7 - Comenzar/Parar simulación	
Descripción	Este caso de uso permitirá comenzar o parar una simulación, así como continuarla.
Precondición	<ul style="list-style-type: none"> 1. Debe haber una simulación cargada. 2. No debe haber una simulación en estado finalizado. 3. No debe haber un mapa real siendo creado. 4. No debe haber un mapa real en estado finalizado. <p>7A. La simulación debe estar en estado sin comenzar. 7B. La simulación debe estar en estado corriendo. 7C. La simulación debe estar en estado parada o detenida.</p>
Flujo principal	<p>7A. La simulación comenzará. 7B. La simulación se detendrá. 7C. La simulación continuará.</p>
Postcondición	<p>7A. Se visualizará el mapa en la pantalla. 7B. Se guardará una imagen del mapa instantáneamente. 7C. La simulación continuará ejecutándose.</p>
Flujo alternativo	Si no se cumple alguna de las precondiciones, se mostrará un mensaje de error y el sistema mantendrá su estado.

Tabla 5.8: Caso de uso C.U.7 - Comenzar/Parar simulación

C.U.8 - Pulsar un punto en el mapa	
Descripción	Cuando se pulse un punto en el mapa, se mostrará la información de este punto.
Precondición	<p>8A. El número de puntos clicados debe ser 0 8B. El número de puntos clicados debe ser 1 8C. El número de puntos clicados debe ser 2</p>
Flujo principal	<p>8A. Se almacenará la información del punto, se mostrará y el aumentará el número de puntos clicados.</p> <p>8B. Se almacenará la información del punto, se mostrará junto con la información del flujo 8A y aumentará el número de puntos clicados.</p> <p>8C. Se eliminará la información del flujo 8B, se modificará la del 8A para mostrar la información de un nuevo punto y devolverá el número de puntos clicados a 1.</p>
Postcondición	La información se mostrará correctamente y el número de puntos también será correctamente modificado.

Tabla 5.9: Caso de uso C.U.8 - Pulsar un punto en el mapa

C.U.9 - Calcular distancia entre puntos	
Descripción	Cuando haya dos puntos clicados, se mostrará en pantalla la distancia entre ambos.
Precondición	El número de puntos clicados debe ser 2.
Flujo principal	Se calculará la distancia entre ambos puntos y se mostrará en la pantalla dicha distancia
Postcondición	Se mostrará en pantalla la distancia.
Flujo alternativo	Si no se cumple alguna de las precondiciones, se mostrará un mensaje de error y el sistema mantendrá su estado.

Tabla 5.10: Caso de uso C.U.9 - Calcular distancia entre puntos

C.U.10 - Modificación de los parámetros	
Descripción	Se modificarán los parámetros necesarios para el algoritmo según especifique el usuario.
Precondición	<ol style="list-style-type: none"> 1. No debe haber un mapa real siendo creado. 2. No debe haber un mapa real finalizado. 3. No debe haber una simulación finalizada. 4. No debe haber una simulación corriendo.
Flujo principal	<p>10A.</p> <ol style="list-style-type: none"> 1. El usuario modificará los parámetros que estime oportuno, siendo todos opcionales. 2. El usuario clicará el botón aceptar. <p>10B. El usuario clicará el botón cancelar, independientemente de su acción anterior.</p>
Postcondición	<p>10A. Los parámetros se quedarán guardados en las variables pertinentes.</p> <p>10B. Los parámetros mantendrán su valor.</p>
Flujo alternativo	Si no se cumple alguna de las precondiciones, se mostrará un mensaje de error y el sistema mantendrá su estado.

Tabla 5.11: Caso de uso C.U.10 - Modificación de los parámetros

5.4.2. Relación de requisitos funcionales con los casos de uso

La tabla 7.11 muestra cómo se relacionan los requisitos funcionales con los casos de uso.

RF\CU	1	2	3	4	5	6	7	8	9	10
1	X									
2		X								
3				X						
4										X
5		X								
6	X									
7	X						X			
8	X	X	X	X	X	X	X	X	X	X
9					X					
10			X							
11										X
12						X				
13							X			
14							X			
15	X									
16					X					
17										X

Tabla 5.12: Relación de requisitos funcionales con los casos de uso

5.5. Especificación de la interfaz

En este capítulo se detallarán los requisitos necesarios para implementar la interfaz del sistema de mapeo. Se detallarán una serie de incrementos en los que se ha ido mejorando la interfaz de forma incremental. El resultado final tratará de ser una interfaz sencilla, fácilmente manejable y con una paleta de colores adecuada para usos prolongados.

Los nombres de los elementos tratarán de ser descriptivos y se añadirá, cuando se estime oportuno, una breve descripción a los elementos al dejar el puntero encima de estos. Para alcanzar una mayor accesibilidad, se ha optado por utilizar como idioma el inglés.

La interfaz constará únicamente de dos ventanas que se destallan a continuación, siendo estas la ventana principal y la ventana "Set parameters".

5.5.1. Ventana principal

En esta ventana se encuentra la mayor parte de la funcionalidad del programa. Desde esta ventana, se podrán crear nuevos mapas y guardar simulaciones de dichos mapas, cargar y ejecutar simulaciones, medir distancias en los mapas creados y obtener acceso a la modificación de los parámetros para la creación del mapa o simulación de uno desde la ventana "Set parameters".

Así mismo, se podrá visualizar en tiempo real la creación del mapa según el sistema se mueve por la superficie a mapear.

Aunque se han planteado una serie de incrementos en los que se ha mejorado la interfaz, durante el proceso de desarrollo será posible modificar dicha interfaz si ello conlleva a una mejora en la facilidad de uso o calidad de la interfaz.

Incremento 1

En este incremento se diseñó un boceto básico de la aplicación, sin colores.

Se incluyen los botones y elementos que cumplen las siguientes funcionalidades:

- Crear un nuevo mapa.
- Guardar dicho mapa.
- Cargar una simulación.
- Guardar las lecturas para crear un archivo de simulación.
- Ajustar los parámetros.
- Obtener la distancia entre dos puntos del mapa.
- Previsualización en tiempo real del mapa.

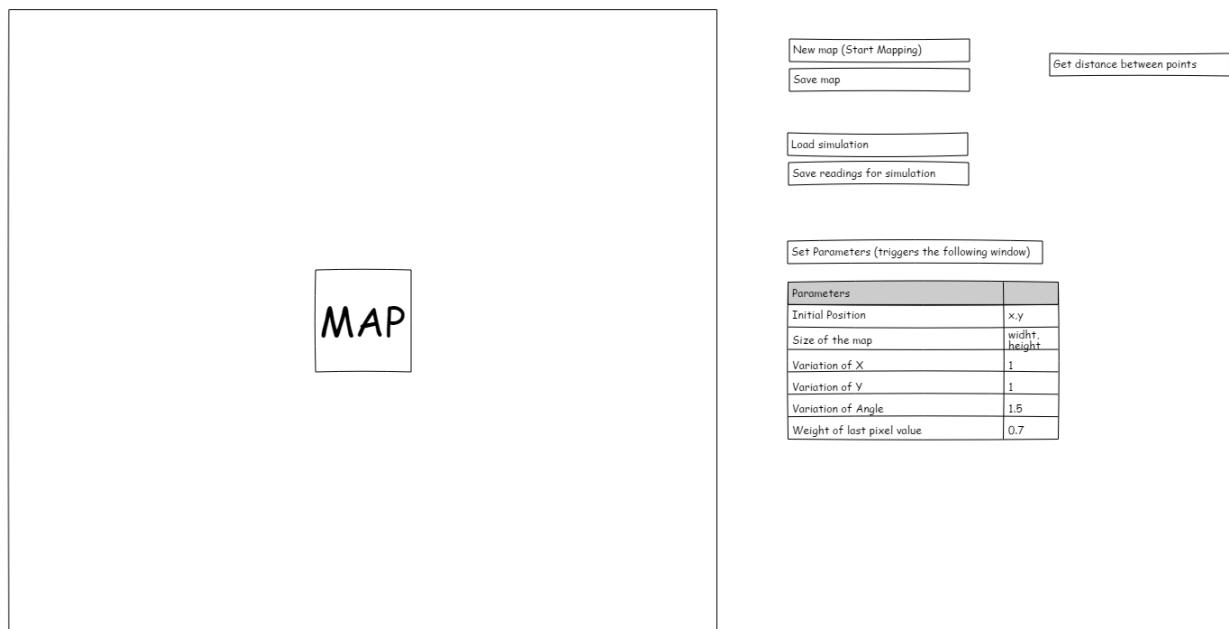


Figura 5.2: Primer boceto de la interfaz de usuario.

Incremento 2

En este incremento se incluyen colores en la interfaz de usuario y se cambia la posición de los botones para un uso más natural del programa. También se desliga la ventana del ajuste de los parámetros, abriendo una ventana nueva.

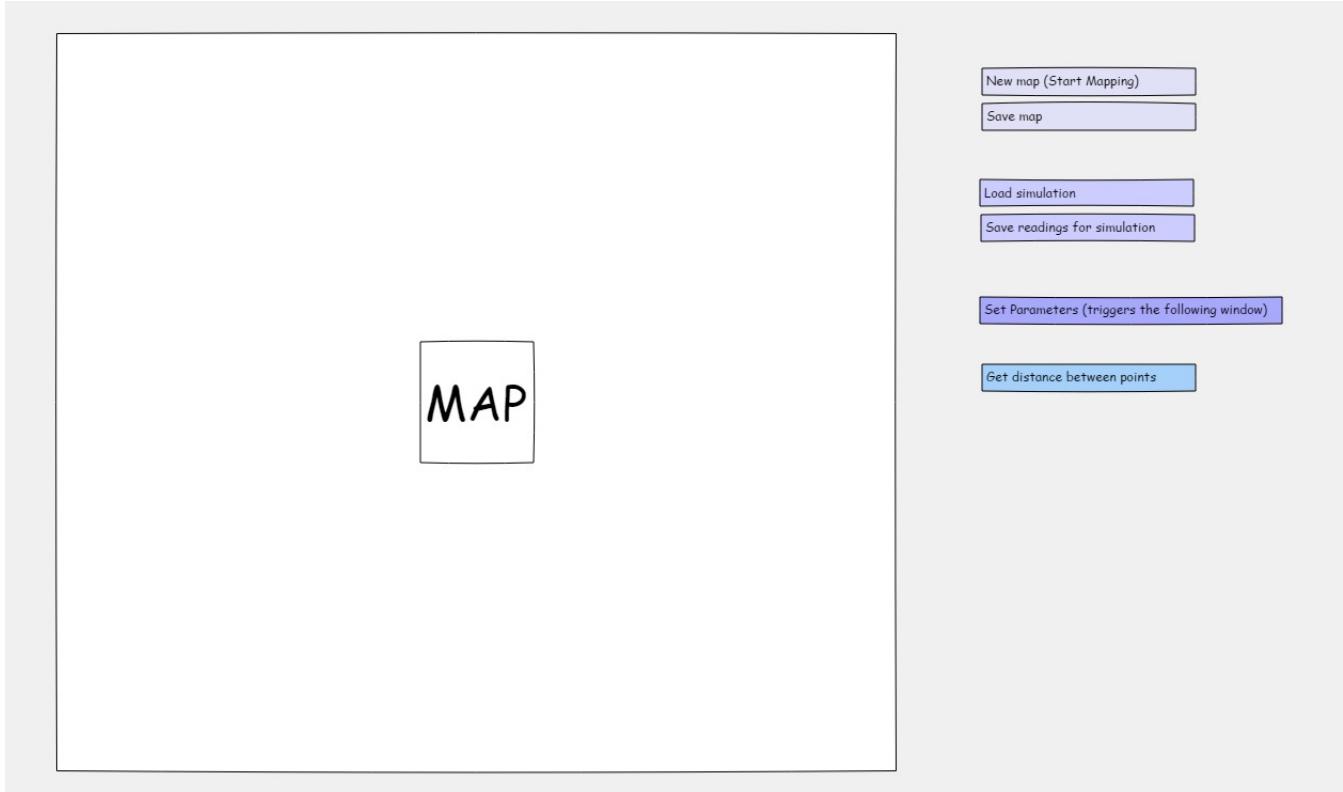


Figura 5.3: Segundo boceto de la interfaz de usuario (colores).

Incremento 3

Este sería el último incremento que se realizó para el diseño de la interfaz. Se añadió una opción para ajustar la velocidad de las simulaciones y se cambiaron los colores por unos más oscuros y que permiten un uso del programa menos degradado en el tiempo.

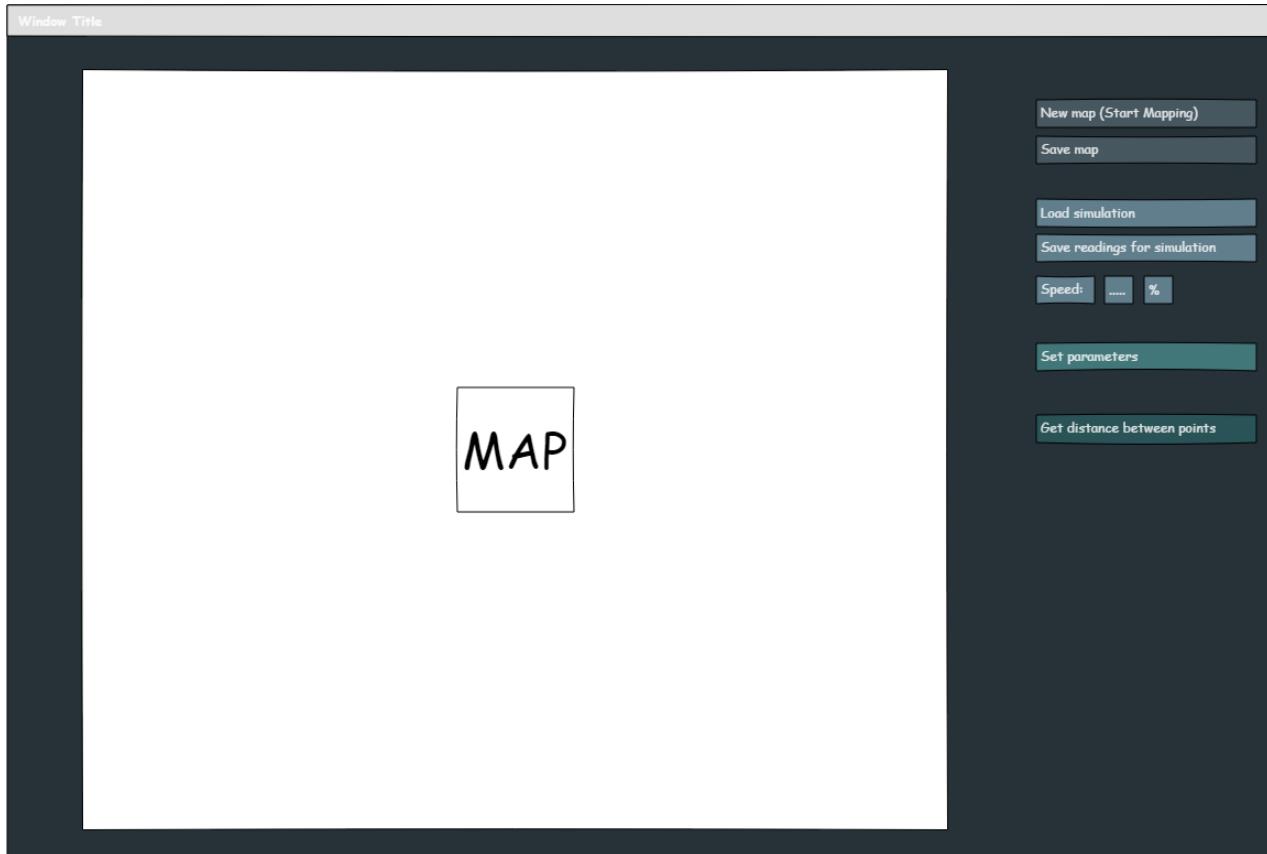


Figura 5.4: Boceto final de la interfaz de usuario.

5.5.2. Ventana ‘Set parameters’

Para esta ventana, al igual que para la ventana principal, se realizaron distintos incrementos para mejorar la apariencia y usabilidad.

El primer incremento consta de distintos parámetros necesarios para el funcionamiento del sistema y carece de color. La ventana parámetros, sin embargo, estaba pensada para formar parte de la ventana principal en el primer incremento, por lo que puede verse en la figura 7.2.

El segundo y último incremento de la ventana parámetros se crea con el cambio de colores de la ventana principal, y únicamente supone un cambio de colores para la ventana. Figura 7.4

Window Title	X	Y	Z
Parameters			
Initial Position	x,y		
Size of the map	width, height		
Variation of X	1		
Variation of Y	1		
Variation of Angle	1.5		
Weight of last pixel value	0.7		

Figura 5.5: Boceto final de la ventana ‘Set parameters’.

Capítulo 6

Diseño del sistema

En este capítulo se detallará el diseño del sistema para que este cumpla con las especificaciones detalladas anteriormente. Para ello, se va a dividir en tres fases:

- Diseño procedimental: se detallarán los diagramas ASM (algorithmic state machine) del sistema para conocer las interacciones.
- Estructura de ficheros y archivos: se detallará la jerarquía de ficheros y las clases incluidas en cada uno.
- Diseño de la interfaz: se detallará la interfaz final del sistema.

6.1. Diseño procedural

En este apartado se procederá a explicar los diagramas ASM y la estructura de ficheros y datos.

En los diagramas ASM se detallarán los distintos estados en los que se puede encontrar el sistema y los escenarios de interacciones mediante dichos diagramas. Se utilizarán diagramas ASM ya que son más descriptivos que los simples diagramas de estado y permiten obtener más fácilmente descripciones programables.

En la estructura de ficheros y datos se explicarán los distintos ficheros del proyecto y su jerarquía, junto con las clases creadas en cada fichero.

6.1.1. Diagramas ASM

Para que los diagramas ASM se entiendan de forma más satisfactoria, se detallan a continuación los estados del sistema:

- Mapa siendo creado.
- Mapa finalizado.
- Simulación cargada.
- Simulación corriendo.

- Simulación parada.
- Simulación finalizada.
- Ajustar parámetros.
- Calcular distancia.
- Punto negro: representa el estado actual del sistema, independientemente de cuál sea este.

Diagrama ASM de creación de nuevo mapa.

El escenario que se visualiza en la figura 9.1 describe la interacción de un usuario con la opción de crear un nuevo mapa. Primero el usuario debe elegir esa opción. Después se comprueban las condiciones necesarias para que se pueda crear un mapa. Si las condiciones son satisfactorias, el sistema pasa al estado mapa siendo creado. Si estando en este estado, el usuario desea guardar el mapa, el sistema pasará al estado finalizado, sino, seguirá en el estado mapa siendo creado. Si el usuario en este caso elige resetear el programa, el programa volverá a su estado inicial, y sino, seguirá en el estado mapa finalizado.

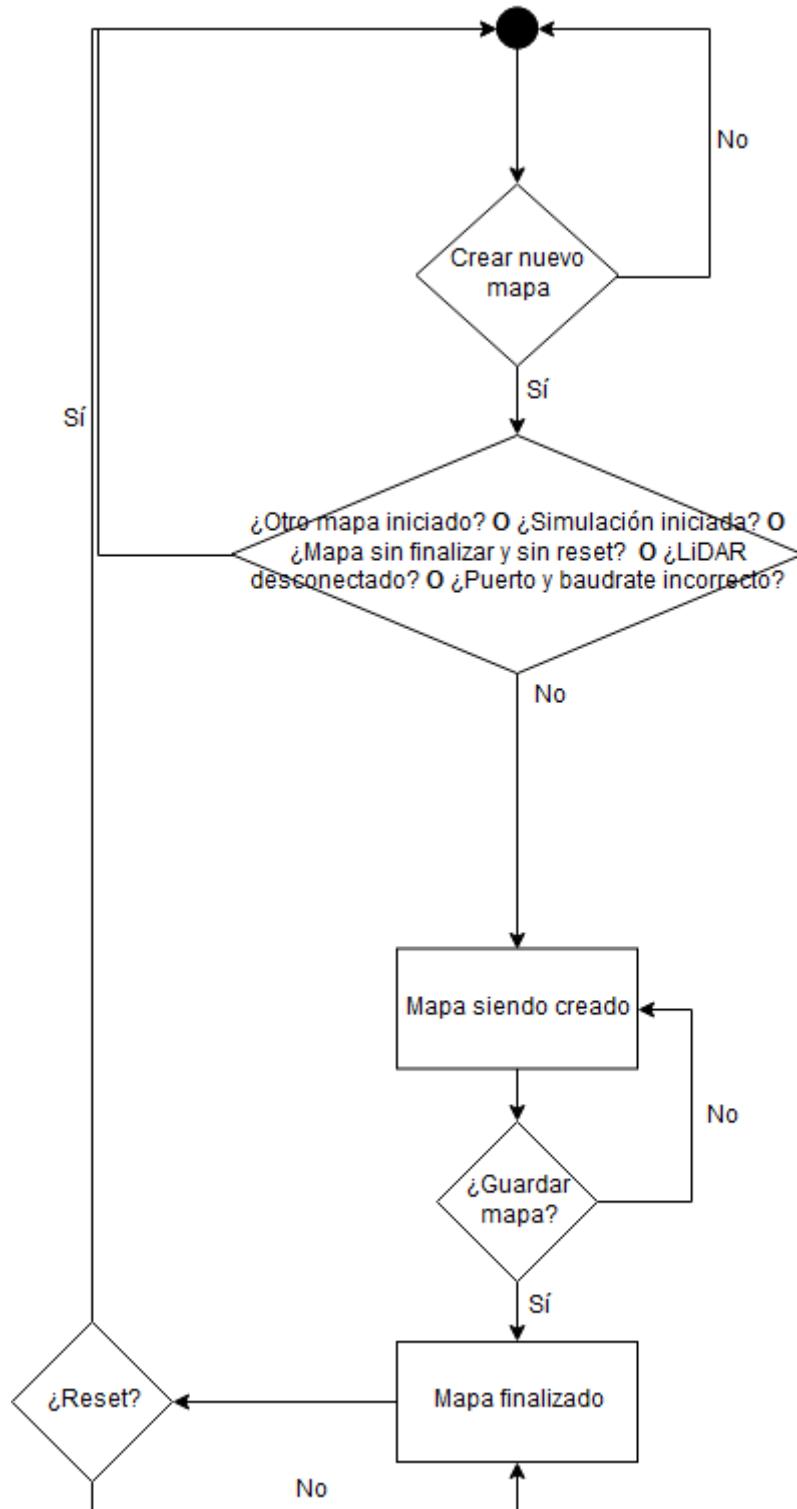


Figura 6.1: Diagrama ASM de creación de nuevo mapa.

Diagrama ASM del escenario de simulación.

El escenario que se visualiza en la figura 9.2 describe la interacción de un usuario con la opción de ejecutar una simulación. Cuando el usuario escoge cargar una simulación, se comprueban las condiciones pertinentes. Si son favorables, se carga la simulación y el

sistema pasa al estado simulación cargada. Cuando el usuario desee comenzar la simulación, esta dará comienzo y el estado del sistema pasará a simulación corriendo. Si decide parar la simulación antes de finalizar, la simulación pasará al estado simulación parada. La simulación podrá ser reanudada por el usuario y el sistema pasará de nuevo al estado simulación corriendo. Si se finaliza la simulación sin que el usuario haya interactuado de nuevo con ella, la simulación pasará al estado simulación finalizada y el usuario podrá escoger entre resetear el sistema o dejarlo en su estado actual.

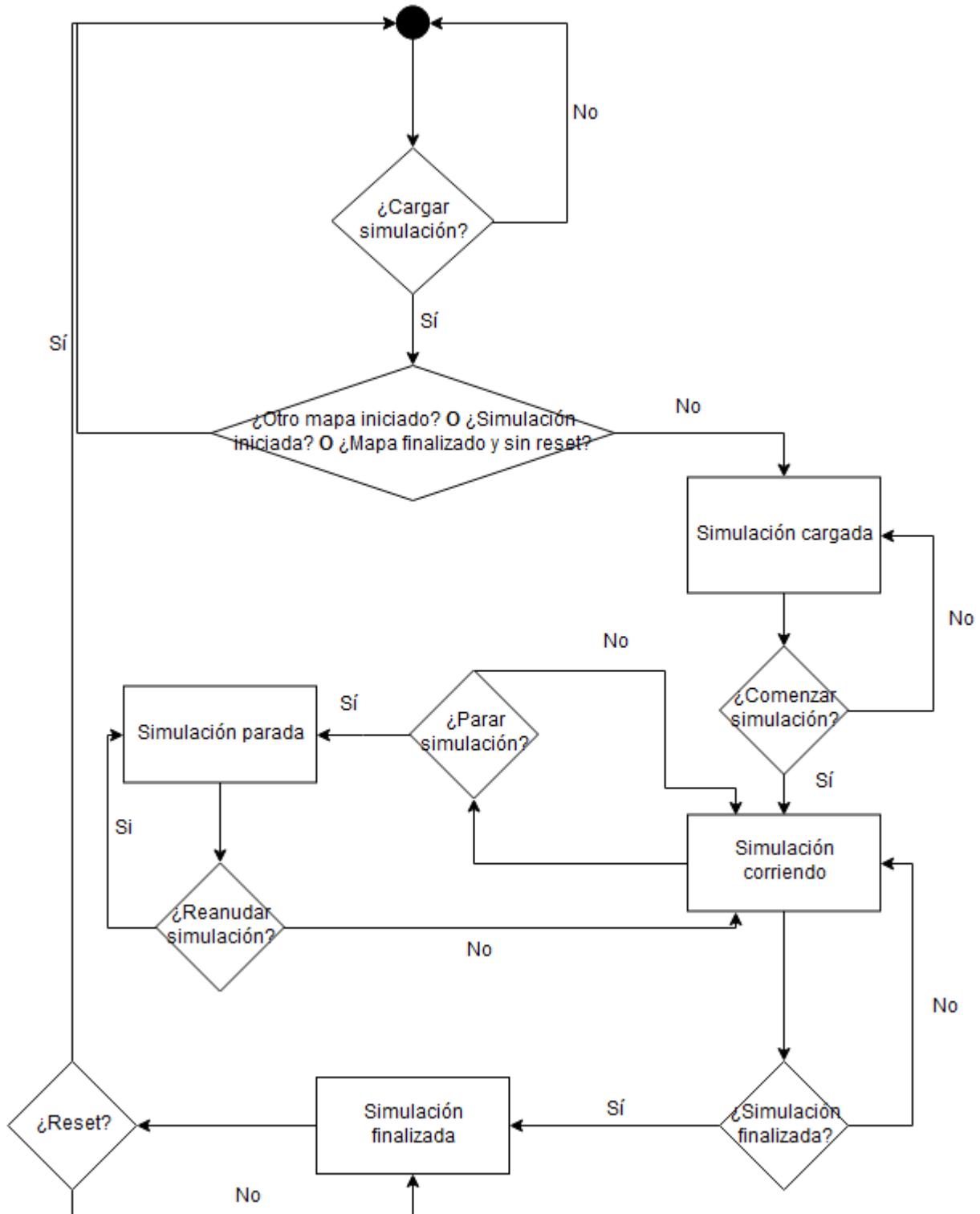


Figura 6.2: Diagrama ASM de simulación.

Diagrama ASMs de ajuste de parámetros.

El escenario que se visualiza en la figura 9.3 describe la interacción de un usuario con la opción de modificar o ajustar los parámetros. Si el usuario decide modificar los parámetros, se deben comprobar una serie de condiciones. Si estas son favorables, el

sistema pasará al estado ajustar parámetros. Cuando estos parámetros sean ajustados, el usuario podrá elegir aceptar, cancelar o seguir modificando parámetros. Si se elige aceptar, los parámetros serán ajustados y el sistema volverá al estado inicial. Si se elige cancelar, los parámetros no se ajustarán y el sistema también volverá a su estado inicial.

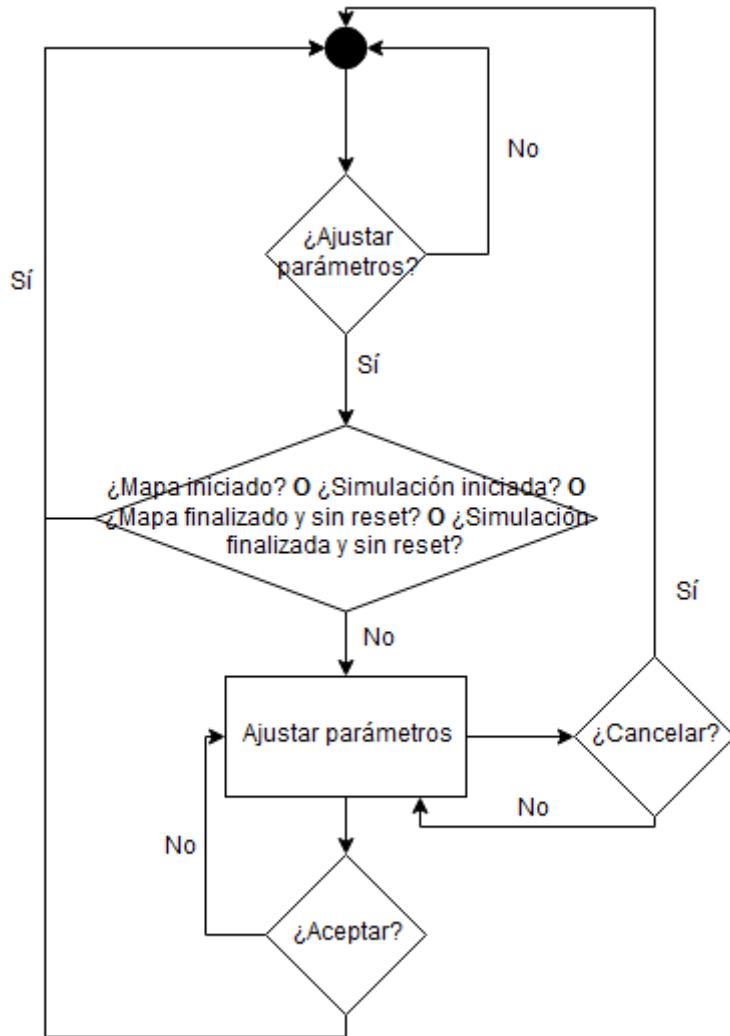


Figura 6.3: Diagrama ASM de ajuste de parámetros.

Diagrama ASM del cálculo de distancia entre dos puntos.

El escenario que se visualiza en la figura 9.4 describe la interacción de un usuario con la opción de calcular distancia entre puntos. Si el usuario decide calcular la distancia entre puntos, el sistema comprobará si se han clicado previamente dos puntos en el mapa. Si estos puntos han sido clicados, se procede a calcular la distancia, mostrar la información al usuario y volver al estado inicial.

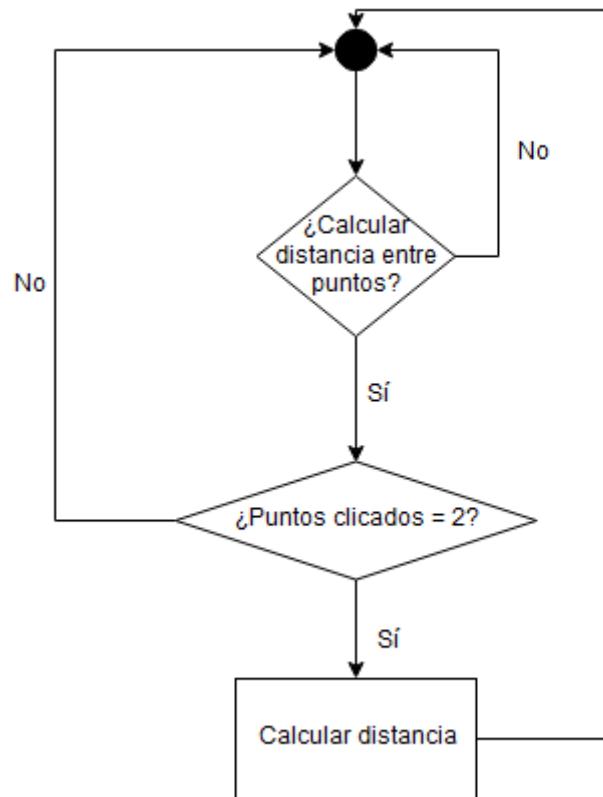


Figura 6.4: Diagrama ASM del cálculo de distancia entre dos puntos.

6.2. Estructura de ficheros y datos

En esta sección se explicará la jerarquía de ficheros y las clases pertenecientes a cada archivo.

La jerarquía global de ficheros se puede observar en la figura 9.5.

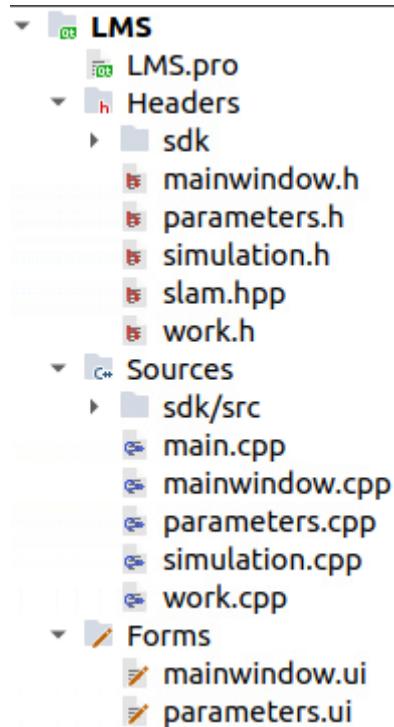


Figura 6.5: Jerarquía global de ficheros

Como se puede observar, hay 4 tipos de ficheros:

- .pro: este fichero es único y define una serie de directrices para que la herramienta disponible en Qt, qmake, cree un makefile para compilar todos los archivos de forma correcta.
- Headers: estos ficheros contendrán las definiciones de las distintas clases que se usarán y algunos de los métodos de cada clase.
- Sources: estos ficheros contendrán el código principal de las clases.
- Forms: estos ficheros serán los encargados de definir la interfaz gráfica de las ventanas necesarias para nuestra aplicación.

A continuación, se explicará detalladamente cada fichero y las funcionalidades que implementa, así como las clases. Se detallarán los ficheros header y sus conexiones con los demás ficheros. Para detallar las clases, se usarán las plantillas de las tablas 9.6 y 9.7.

En las tablas 9.6 y 9.7 se pueden apreciar unos elementos que pueden parecer inusuales, los stlots y las signals. Ambos pertenecen a un tipo de método de Qt. Dichos slots conectan clases mediante el par signal-slot. Cuando una clase ejecuta una señal conectada vía signal-slot, la otra clase ejecuta el slot conectado[27]. Se podría decir que es una forma de manejar interrupciones.

Nombre de la clase
- Atributos
- Métodos
- Slots
- Signals

Tabla 6.1: Definición de la clase.

Nombre	
Descripción	
Atributos	
Métodos	
Slots	
Signals	

Tabla 6.2: Especificación de la clase.

6.2.1. Clase mainWindow

El primer fichero que nos encontramos es mainWindow.h. En este fichero se define la clase mainWindow, la cual se encargará de administrar la mayor parte de la interfaz gráfica y la funcionalidad del programa. Los ficheros mainWindow.cpp y mainWindow.ui forman parte de la especificación de la clase mainWindow. El fichero cpp implementa el código principal de las funciones, mientras que el fichero ui implementa la interfaz gráfica de la ventana principal (mainWindow).

mainWindow
- param: parameters
- pool: QThreadPool
- work: Work
- simulation: Simulation
- Q1,Q2: Qpoint
- points, count: int
- mainWindow()
- on_b_new_map_clicked()
- on_b_save_map_clicked()
- on_b_save_sim_clicked()
- on_b_restart_clicked()
- on_b_load_clicked()
- on_b_start_stop_clicked()
- btnaction()
- on_b_distance_clicked()
- on_map_button_clicked()
- on_text_spdTextChanged(const QString &arg1)
- print_img(const QImage &img)
- sim_finished()
- error_port()
- Sin signals.

Tabla 6.3: Definición de la clase mainWindow.

Nombre	mainWindow
Descripción	Esta ventana se encargará de proporcionar la funcionalidad principal del programa (botones para iniciar y guardar mapas, cargar y guardar simulaciones, abrir la ventana de ajustar parámetros, calcular distancias y cambiar la velocidad de simulación).
Atributos	<ul style="list-style-type: none"> - param: parameters -> clase que proporcionará la ventana para modificar los parámetros. - pool: QThreadPool -> variable que permitirá crear varios hilos de ejecución. - work: Work -> clase que permitirá crear mapas reales. - simulation: Simulation -> clase que permitirá manejar simulaciones. - Q1,Q2: QPoint -> puntos que pulsará el usuario en el mapa para calcular distancias. - points: int -> número de puntos que ha clicado el usuario en el mapa (máximo 2). - count: int -> número de medidas cargados desde el fichero de simulación.
Métodos	<ul style="list-style-type: none"> - mainWindow(): constructor, inicializa las clases, points y count.
Slots	<ul style="list-style-type: none"> - on_b_new_map_clicked(): este slot lanza un hilo para empezar la creación de un nuevo mapa - on_b_save_map_clicked(): este slot guarda la imagen del mapa que se está creando y para la creación. - on_b_save_sim_clicked(): este slot guarda los datos en un fichero de simulación. - on_b_restart_clicked(): este slot resetea la aplicación. - on_b_load_clicked(): este slot carga una simulación. - on_b_start_stop_clicked(): este slot comienza/para una simulación. - btnaction(): este slot abre la ventana de parámetros para modificarlos. - on_b_distance_clicked(): este slot calcula la distancia entre puntos y la muestra. - on_map_button_clicked(): este slot muestra la información de un punto clicado por el usuario en el mapa y la guarda. - on_text_spdTextChanged(QString): este slot cambia la velocidad de simulación. - print_img(QImage): este slot muestra la imagen del mapa en la pantalla. - sim_finished(): este slot muestra al usuario que la simulación ha acabado - error_port(): este slot muestra un mensaje de error al no poder conectar con el puerto especificado.
Signals	No cuenta con ninguna señal.

Tabla 6.4: Especificación de la clase mainWindow.

6.2.2. Clase parameters

El siguiente fichero es parameters.h. En este fichero se define la clase parameters, encargada de administrar los parámetros que se usarán en el programa y modificar dichos parámetros a gusto del usuario. Este fichero estaría relacionado con los ficheros parameters.cpp, que implementa el código principal de las funciones y con parameters.ui, que permite al usuario modificar los parámetros gráficamente.

parameters
<ul style="list-style-type: none"> - <code>ui: Ui::parameters</code> - <code>width_height_, baudrate_, range_x_, range_y_, range_angle_: int</code> - <code>angle_var_, x_var_, y_var_, w_: double</code> - <code>p_: slam::position</code> - <code>port_: char*</code> - <code>levmarq_, ignore_human_: bool</code>
<ul style="list-style-type: none"> - <code>parameters()</code> - <code>clear()</code> - <code>Métodos get y set.</code>
<ul style="list-style-type: none"> - <code>void on_b_accept_clicked();</code> - <code>void on_algorithm_currentIndexChanged(int index);</code> - <code>void on_b_cancel_clicked();</code>
<ul style="list-style-type: none"> - <code>Sin signals.</code>

Tabla 6.5: Definición de la clase parameters.

Nombre	parameters
Descripción	Esta clase permitirá al usuario modificar los parámetros de forma gráfica y a su antojo, pudiendo dejar todos como predeterminados.
Atributos	<ul style="list-style-type: none"> - <code>ui</code>: variable que proporciona la interfaz gráfica. - <code>width_</code>: ancho del mapa. - <code>height_</code>: alto del mapa. - <code>baudrate_</code>: baudrate de la conexión con el LiDAR. - <code>range_x_</code>: rango de X en el que se estimará la posición. - <code>range_y_</code>: rango de Y en el que se estimará la posición. - <code>range_angle_</code>: rango del ángulo en el que se estimará la posición. - <code>angle_var_</code>: variación del ángulo con la que se harán las distintas estimaciones. - <code>x_var_</code>: variación de X con la que se harán las distintas estimaciones. - <code>y_var_</code>: variación de Y con la que se harán las distintas estimaciones. - <code>w_</code>: peso del valor anterior del pixel para obtener el valor nuevo de ocupación de dicho pixel. - <code>p_</code>: punto inicial del sistema en el mapa. - <code>port_</code>: puerto al que está conectado el LiDAR. - <code>levmarq_</code>: variable que definirá si se usa el algoritmo Levenberg Marquardt o fuerza bruta. - <code>ignore_human_</code>: variable que definirá si el ángulo en el que estaría el humano debe ser ignorado o no.
Métodos	<ul style="list-style-type: none"> - <code>parameters()</code>: constructor que inicializa los parámetros a unos valores predeterminados fijados por el programador. - <code>clear()</code>: restablece los parámetros a sus valores predeterminados y limpia memoria. - Métodos get y set: para visualizar y modificar los parámetros respectivamente.
Slots	<ul style="list-style-type: none"> - <code>on_b_accept_clicked()</code>: establece los valores de los parámetros a los valores especificados por el usuario si estos son correctos. - <code>on_algorithm_currentIndexChanged(int index)</code>: establece el algoritmo que se va a utilizar junto con sus parámetros necesarios. - <code>on_b_cancel_clicked()</code>: cancela la edición, no se tienen en cuenta los valores modificados. - <code>on_ignore_human_stateChanged(int arg1)</code>: slot que establece el parámetro de ignorar el humano dependiendo de la elección del usuario.
Signals	Sin signals

Tabla 6.6: Especificación de la clase parameters.

6.2.3. Clase Simulation

El siguiente fichero encontrado es simulation.h. Este fichero define la clase simulation y se encargará de administrar los datos y las funciones necesarias para trabajar con simulaciones y que se cumplan las especificaciones correspondientes. Este fichero está relacionado con simulation.cpp, que implementa el código de las funcionalidades principales de la clase.

Simulation
- running_: bool - started_: bool - finished _: bool - loaded _: bool - levmarq_: bool - ignore human_: bool - start_: int - P_: slam::Position - width_,height_, count_, range_x_, range_y_, range_angle_: int - x_var_, y_var_, angle_var_,w_, speed_: double - mat_: cv::Mat - readings_[]: rplidar_response_measurement_node_hq_t
- init() - run() - clear() - Métodos get y set.
- Sin slots.
- print_img(QImage) - sim_finished()

Tabla 6.7: Definición de la clase simulation.

Nombre	Simulation
Descripción	Esta clase implementa todos los datos y funciones necesarias para simular un mapa con lecturas del LiDAR extraídas de un fichero de simulación.
Atributos	<ul style="list-style-type: none"> - running_: estado de simulación: corriendo. - started_: estado de simulación: empezada. - finished_: estado de simulación: finalizada. - loaded_: estado de simulación: cargada. - speed_: velocidad de simulación. - readings_[]: lecturas del LiDAR almacenadas en memoria, leídas de un fichero de simulación. - mat_: imagen que se enviará a la ventana principal para imprimirla. - start_: variable que indica si se ha procesado el primer frame. - width_: ancho del mapa. - height_: alto del mapa. - range_x_: rango de X en el que se estimará la posición. - range_y_: rango de Y en el que se estimará la posición. - range_angle_: rango del ángulo en el que se estimará la posición. - angle_var_: variación del ángulo con la que se harán las distintas estimaciones. - x_var_: variación de X con la que se harán las distintas estimaciones. - y_var_: variación de Y con la que se harán las distintas estimaciones. - w_: peso del valor anterior del pixel para obtener el valor nuevo de ocupación de dicho pixel. - p_: punto inicial del sistema en el mapa. - levmarq_: variable que definirá si se usa el algoritmo Levenberg Marquardt o fuerza bruta. - ignore human_: variable que definirá si el ángulo en el que estaría el humano debe ser ignorado o no.
Métodos	<ul style="list-style-type: none"> - init(): inicializa los parámetros a unos valores predeterminados y todos sus estados a false. - run(): comienza la simulación y se ejecuta todo el código necesario para realizar SLAM simulado. - clear(): libera memoria y restablece los parámetros predeterminados.
	<ul style="list-style-type: none"> - Métodos get y set: para visualizar y modificar los parámetros respectivamente.
Slots	Sin slots.
Signals	<ul style="list-style-type: none"> - print_img(QImage): envía una imagen del mapa a la ventana principal para que la imprima en pantalla. - sim_finished(): avisa a la ventana principal que la simulación ha terminado para que esta avise al usuario.

Tabla 6.8: Especificación de la clase simulation.

6.2.4. Clases Position y Map

El siguiente fichero, slam.hpp, contiene dos clases, Position y Map, ambos enfocados a la resolución del problema SLAM, en el que se basa nuestro proyecto. Estas clases contienen una serie de datos y funciones para resolver dicho problema.

slam::Position
- x_, y_ : int
- angle_ : double
- Position(int x=0, int y=0, double angle=0)
- Position(slam::Position)
- Operador =.
- Métodos get and set.
- Sin slots
- Sin signals

Tabla 6.9: Definición de la clase Position.

Nombre	slam::Position
Descripción	Esta clase implementa la posición del sistema que se usará para SLAM.
Atributos	<ul style="list-style-type: none"> - x_, y_: posición en los ejes de coordenadas (X,Y) - angle_: ángulo de orientación del sistema.
Métodos	<ul style="list-style-type: none"> - Position(int x=0, int y=0, double angle=0): constructor por defecto, puede recibir las 3 componentes pero son opcionales. - Position(slam::Position): constructor de copia. - Operador =: operador de asignación para una mayor comodidad. - Métodos get and set: para modificar y visualizar las componentes respectivamente.
Slots	Sin slots.
Signals	Sin signals.

Tabla 6.10: Especificación de la clase Position.

slam::Map
<ul style="list-style-type: none"> - map_: cv::Mat - oc_: cv::Mat
<ul style="list-style-type: none"> - Map(int x = 800, int y = 800) - Map(cv::Mat map) - release() - resize(int width, int height) - setValues(int x, int y, int r, int g, int b) - setValues(cv::Point pos, int r, int g, int b) - setValuesOc(int x, int y, int value) - setValuesOc(cv::Point pos, int value) - set_zeros() - lineToObject(int x0, int y0, int x1, int y1, double w) - line(int x0, int y0, int x1, int y1, int r, int g, int b) - fit(slam::Position try_P, double current_dist[], double theta[], int count) - bruteForce(slam::Position P, double dist[], double theta[], int count, int range_x, int range_y, int range_angle, int x_var, int y_var, double angle_var) - drawSystem(slam::Position P, int r, int g, int b) - undrawSystem(slam::Position P) - update(slam::Position P) - Métodos get y set.
<ul style="list-style-type: none"> - Sin slots
<ul style="list-style-type: none"> - Sin signals.

Tabla 6.11: Definición de la clase Map.

Nombre	slam::Map
Descripción	Esta clase implementa los métodos y datos necesarios para realizar SLAM en conjunto con un mapa de ocupación probabilístico y un mapa visual que se mostraría al usuario.
Atributos	<ul style="list-style-type: none"> - map_: mapa visual. - oc_: mapa de ocupación probabilístico.
Métodos	<ul style="list-style-type: none"> - Map(int x = 800, int y = 800): constructor por defecto, puede no recibir ningún parámetro. - Map(cv::Mat map): constructor de copia. - release(): libera la memoria de los dos mapas. - resize(int width, int height): cambia el tamaño de los dos mapas. - setValues(int x, int y, int r, int g, int b): el valor pasado RGB se asigna al pixel con la posición (x,y) proporcionada, en el mapa visual. - setValues(cv::Point pos, int r, int g, int b): el valor pasado RGB se asigna al pixel con la posición pos proporcionada, en el mapa visual. - setValuesOc(int x, int y, int value): el valor pasado en escala de grises se asigna al pixel con la posición (x,y) proporcionada, en el mapa de ocupación. - setValuesOc(cv::Point pos, int value): el valor pasado en escala de grises se asigna al pixel con la posición pos proporcionada, en el mapa de ocupación. - set_zeros(): asigna ceros a todas las posiciones de ambos mapas. - lineToObject(int x0, int y0, int x1, int y1, double w): crea una línea a un objeto, asignando una probabilidad de ocupación a los distintos píxeles entre el sistema y el objeto, dependiendo del peso del valor anterior w. - line(int x0, int y0, int x1, int y1, int r, int g, int b): crea una línea en el mapa entre dos puntos con el valor RGB especificado. - fit(slam::Position try_P, double current_dist[], double theta[], int count): devuelve un valor entre 0 y 1 según la bondad de una posición probada try_P de acuerdo al mapa de ocupación. - bruteForce(slam::Position P, double dist[], double theta[], int count, int range_x, int range_y, int range_angle, int x_var, int y_var, double angle_var): aplica el algoritmo de fuerza bruta para estimar la mejor posición. - drawSystem(slam::Position P, int r, int g, int b): dibuja el sistema en el mapa visual. - undrawSystem(slam::Position P): borra el sistema del mapa visual. - update(slam::Position P): actualiza el mapa visual con la posición del sistema y los valores del mapa de ocupación. - Métodos get y set.
Slots	Sin slots.
Signals	Sin signals.

Tabla 6.12: Especificación de la clase Map.

6.2.5. Clase Work

Por último, tenemos el fichero work.h, el cual implementa la clase Work. Esta clase

contiene los datos y funcionalidades necesarias para crear un mapa real, cumpliendo con las especificaciones del proyecto. Este fichero está relacionado con work.cpp, que implementa el código de las funcionalidades principales de la clase.

Work
<ul style="list-style-type: none"> - <code>running_</code>: bool - <code>started_</code>: bool - <code>finished_</code>: bool - <code>error_</code>: bool - <code>save_</code>: bool - <code>levmarq_</code>: bool - <code>ignore_human_</code>: bool - <code>count_readings_</code>: int - <code>start_</code>: int - <code>P_</code>: slam::Position - <code>width_,height_,baudrate_,range_x_,range_y_,range_angle_</code>: int - <code>angle_var_,x_var_,y_var_,w_</code>: double - <code>port_</code>: char* - <code>levmarq_</code>: bool - <code>mat_</code>: cv::Mat - <code>readings_[]: rplidar_response_measurement_node_hq_t</code>
<ul style="list-style-type: none"> - <code>init()</code> - <code>run()</code> - <code>clear()</code> - Métodos get y set.
<ul style="list-style-type: none"> - Sin slots
<ul style="list-style-type: none"> - <code>print_img(QImage)</code> - <code>error_port()</code>

Tabla 6.13: Definición de la clase work.

Nombre	Work
Descripción	Esta clase implementa los datos y métodos necesarios para crear un mapa en tiempo real mientras se identifica al sistema dentro de dicho mapa (SLAM).
Atributos	<ul style="list-style-type: none"> - running_: estado de simulación: corriendo. - started_: estado de simulación: empezada. - finished_: estado de simulación: finalizada. - loaded_: estado de simulación: cargada. - error_: variable que indica si se ha producido algún fallo en la conexión con el LiDAR. - save_: variable que indica si se debe guardar la imagen del mapa. - count_readings_: número de lecturas almacenadas desde el inicio de la creación del mapa. - readings_[]: lecturas del LiDAR almacenadas en memoria, preparadas para guardar un fichero de simulación. - mat_: imagen que se enviará a la ventana principal para imprimirla. - start_: variable que indica si se ha procesado el primer frame. - width_: ancho del mapa. - height_: alto del mapa. - baudrate_: baudrate de la conexión con el LiDAR. - range_x_: rango de X en el que se estimará la posición. - range_y_: rango de Y en el que se estimará la posición. - range_angle_: rango del ángulo en el que se estimará la posición. - angle_var_: variación del ángulo con la que se harán las distintas estimaciones. - x_var_: variación de X con la que se harán las distintas estimaciones. - y_var_: variación de Y con la que se harán las distintas estimaciones. - w_: peso del valor anterior del pixel para obtener el valor nuevo de ocupación de dicho pixel. - p_: punto inicial del sistema en el mapa. - port_: puerto al que está conectado el LiDAR. - levmarq_: variable que definirá si se usa el algoritmo Levenberg Marquardt o fuerza bruta. - ignore_human_: variable que definirá si el ángulo en el que estaría el humano debe ser ignorado o no.
Métodos	<ul style="list-style-type: none"> - init(): inicializa los parámetros a unos valores predeterminados y todos sus estados a false. - run(): comienza la simulación y se ejecuta todo el código necesario para realizar SLAM en tiempo real. - clear(): libera memoria y restablece los parámetros predeterminados.
	<ul style="list-style-type: none"> - Métodos get y set: para visualizar y modificar los parámetros respectivamente.
Slots	Sin slots.
Signals	<ul style="list-style-type: none"> - print_img(QImage): envía una imagen del mapa a la ventana principal para que la imprima en pantalla. - error_port(): avisa a la ventana principal de que se produjo un error al intentar conectar con el puerto para que la ventana principal avise al usuario.

Tabla 6.14: Especificación de la clase work.

6.3. Diseño de los algoritmos de SLAM

Ya que la funcionalidad principal del sistema es proporcionar una herramienta para crear mapas utilizando SLAM, se deben diseñar unos algoritmos que permitan dicho propósito en un entorno de tiempo real. Las ventajas y desventajas de cada uno de ellos se verán en el capítulo de conclusiones.

Para ello, se utilizarán dos algoritmos:

- **Fuerza bruta:** este método, como cualquier algoritmo de fuerza bruta, se basa en probar todas las posibilidades de posición para encontrar la mejor. Evidentemente, como probar literalmente todas las posiciones sería imposible en tiempo de cálculo, se probarán rangos y variaciones que aporten una buena solución manteniendo un tiempo de cómputo razonable. De igual forma, desde la aplicación se podrán modificar dichos parámetros para que el usuario experimente con ellos si lo desea.
- **Levenberg Marquardt:** este es un algoritmo de optimización local mediante mínimos cuadrados amortiguados. Se utiliza este algoritmo ya que es más inteligente que fuerza bruta y consigue un tiempo de cómputo menor.

Para realizar la optimización, como en cualquier optimización, es necesario tener una función de error o de fitness. Dicha función en nuestro caso es la función fit de la clase map, que ya fue comentada en la especificación de la clase. El diseño de esta función sería similar a como se muestra en el algoritmo 1:

Algorithm 1 fit(try_P, current_dist[], theta[], count)

```

fitness ← 0.0
x ← 0, y ← 0
for i from 0 to count do
    if current_dist[i] > 0.1 then
        x ← (cos((theta[pos]+try_P.angle)*PI/180.0)*current_dist[pos])+try_P.x
        y ← (sin((theta[pos]+try_P.angle)*PI/180.0)*current_dist[pos])+try_P.y
        if x < map.columns AND y < map.rows then
            fitness ← fitness + map → occupationValue(x, y)
        end if
    end if
end for
return (255 * count - fitness)/(255 * count)

```

Para entender este algoritmo, antes se debe entender cómo estará implementado el mapa de ocupación, que ya se comentó a grandes rasgos en los antecedentes. Este mapa contendrá valores entre 0 y 255, los cuales representan una probabilidad de ocupación. Si el valor está por debajo de 30, se toma esa región como desconocida. Si el valor está entre 30 y 161, esa posición del mapa se toma como desocupada. Si el valor está entre 161 y 222, se toma como parcialmente ocupada. Y, finalmente, si está entre 222 y 255, se toma como ocupada. Dichos valores de ocupación aumentarán dependiendo de un parámetro, que marcará el peso que se le da tanto al valor anterior como al nuevo (entre 0 y 1, representando 0 un 0 % y 1 un 100 %). Esta probabilidad será mayor si muchos rayos

inciden sobre la misma casilla a lo largo del tiempo. Estos rangos de valores únicamente serán determinantes a la hora de representar el mapa y no tienen relación alguna con la optimización, puesto que la función fit (y por tanto la optimización del error) toma el valor de probabilidad tal cuál lo extrae del mapa de ocupación (un valor entre 0 y 255). Por último, a las casillas cercanas a las que la distancia del láser marque como ocupadas, también se asignará un cierto valor de ocupación (menor que la casilla exacta donde caiga el rayo), dada la incertidumbre de la medida del sensor.

El algoritmo de la función fit por tanto, básicamente genera un valor de fitness entre 0 y 1 (0 significaría muy buena solución y 1 muy mala) para un punto de prueba, teniendo como datos las medidas de esa iteración y el mapa de ocupación. Para calcular X e Y se utiliza trigonometría dado el ángulo desde el que fueron tomadas las distancias. Si la distancia es válida (mayor a 0.1) y los puntos no se salen del tamaño máximo del mapa, se añade ese valor de ocupación al fitness. Finalmente, la mejor solución sería la que haya alcanzado un valor máximo de fitness, ya que querría decir que con el punto que se ha probado, la mayoría de casillas tenían una probabilidad de ocupación alta, por lo que también es probable que esa sea la posición en la que el sistema ha extraído las medidas.

Sin embargo, para hacer el resultado de la función más simple, se resta a la puntuación máxima que un fitness podría obtener y se divide entre dicha puntuación máxima, obteniendo así un valor entre 0 y 1.

6.3.1. Diseño del algoritmo de fuerza bruta

Este algoritmo también formará parte de la clase map y su propósito es estimar el punto y orientación del sistema tras una medición. Hará uso de la función fit para la optimización.

Algorithm 2 bruteForce(P, dist[], theta[], count, range_x, range_y, range_angle, x_var, y_var, angle_var)

```

 $opt \leftarrow 0.0$ ,  $opt\_try \leftarrow 0.0$ 
 $x \leftarrow 0$ ,  $y \leftarrow 0$ 
 $try\_P \leftarrow (P.x - range\_x/2, P.y - range\_y/2, P.angle - range\_angle/2)$ 
 $prev\_P \leftarrow P$ 
 $opt \leftarrow fit(prev\_P, dist, theta, count)$ 
for i from 0 to range_y with step y_var do
     $try\_P.y \leftarrow prev\_P.y - range\_Y/2 + i$ 
    for j from 0 to range_x with step x_var do
         $try\_P.x \leftarrow prev\_P.x - range\_X/2 + j$ 
        for k from 0 to range_angle with step angle_var do
             $try\_P.angle \leftarrow prev\_P.angle - range\_angle/2 + k$ 
             $opt\_try \leftarrow fit(try\_P, dist, theta, count)$ 
            if  $opt\_try < opt$  then
                 $P \leftarrow try\_P$ 
                 $opt \leftarrow opt\_try$ 
            end if
        end for
    end for
end for

```

Este algoritmo, como se comentaba antes, usará los parámetros que decida el usuario (o los predeterminados) para probar un gran número distinto de posiciones y orientaciones del sistema, viendo cuál se ajusta más al mapa dadas las mediciones de esa iteración.

6.3.2. Diseño del algoritmo usando Levenberg Marquardt

El algoritmo Levenberg Marquardt se basa en la optimización por mínimos cuadrados amortiguados y se utiliza para resolver problemas de mínimos cuadrados no lineales. Al igual que muchos algoritmos de optimización, puede encontrar un mínimo local en lugar de un mínimo global (se comprobará su eficacia en el capítulo de pruebas). El problema SLAM es conocido por muchos por su compleja no linealidad, pero, como comentan en [28], existen algunas aproximaciones lineales que dan resultados bastante buenos. En nuestro caso, se utilizará este algoritmo que aporta un modelo no lineal y se comprobarán los resultados obtenidos.

Algorithm 3 error(sol[], err[], map, dist[], theta[], count)

```

 $err.resize(1)$ 
 $P \leftarrow (sol(0), sol(1), sol(2))$ 
 $err(0) \leftarrow map.fit(P, dist, theta, count)$ 

```

Algorithm 4 levmarq(it, min_error, x_var, y_var, angle_var, map, dist[], theta[], count)

```

Solver.setParams(it, min_error);
sol(0) ← P.x
sol(1) ← P.y
sol(2) ← P.angle
Solver.setDervEpsilon(x_var, y_var, angle_var)
Solver.solve(sol, bind(error, placeholder_1, placeholder_2, map, dist, theta, count))
P.x ← sol(0)
P.y ← sol(1)
P.angle ← sol(2)

```

6.4. Diseño de la interfaz

En esta sección será descrita la interfaz de usuario que finalmente se usará para el proyecto. Se detallarán todas las funcionalidades que incluye la interfaz y los cambios necesarios para implementar las mejoras que se hayan considerado durante el proceso de desarrollo.

6.4.1. Ventana principal

En la ventana principal se pueden apreciar una serie de cambios con respecto al boceto final que se hizo de la misma:

- El botón ‘Save data for simulation’ ha sido movido a la parte superior para aportar una mayor claridad en el conjunto de botones.
- Se ha incluido un botón para reiniciar la aplicación.
- Se han incluido una serie de elementos (etiquetas o labels) para facilitar la visualización de la distancia entre dos puntos.
- Se ha añadido un botón para empezar y parar la simulación, permitiendo el guardado del mapa en cada parada.
- Se ha añadido una etiqueta para que el usuario visualice cuándo se ha acabado una simulación.

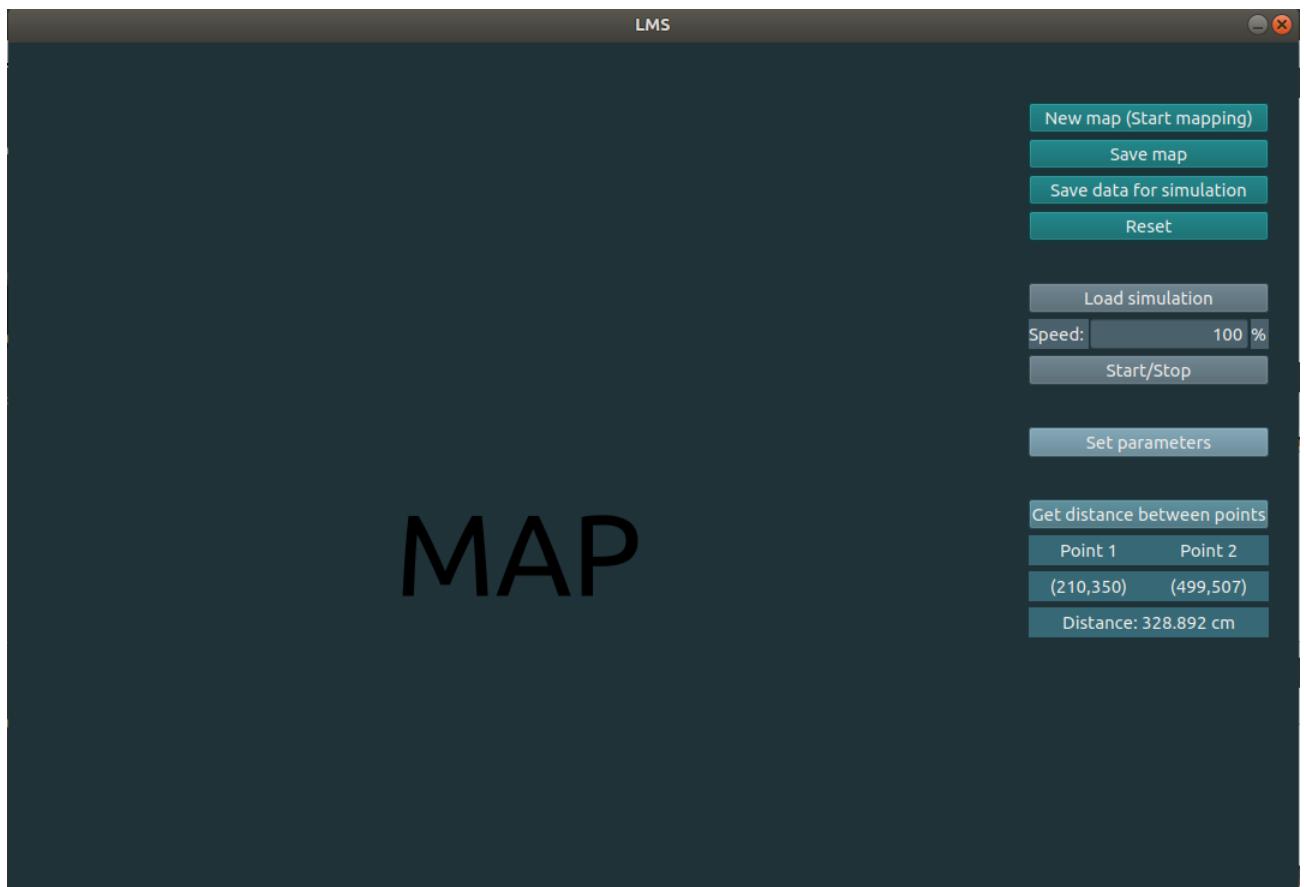


Figura 6.6: Interfaz final de la aplicación.

6.4.2. Ventana ‘Set parameters’

La ventana ‘Set parameters’ ha sufrido una serie de cambios con respecto a la especificación de la interfaz debido a mejoras que se han detectado e implementado a la hora del desarrollo del proyecto. Estas mejoras son:

- Se han separado varios parámetros para identificar distintas partes del sistema.
- Se ha añadido una opción para elegir el algoritmo con el que se creará el mapa y se identificará el sistema dentro de él.
- Se ha añadido una etiqueta indicando que todos los campos son opcionales, ya que disponen de valores por defecto.

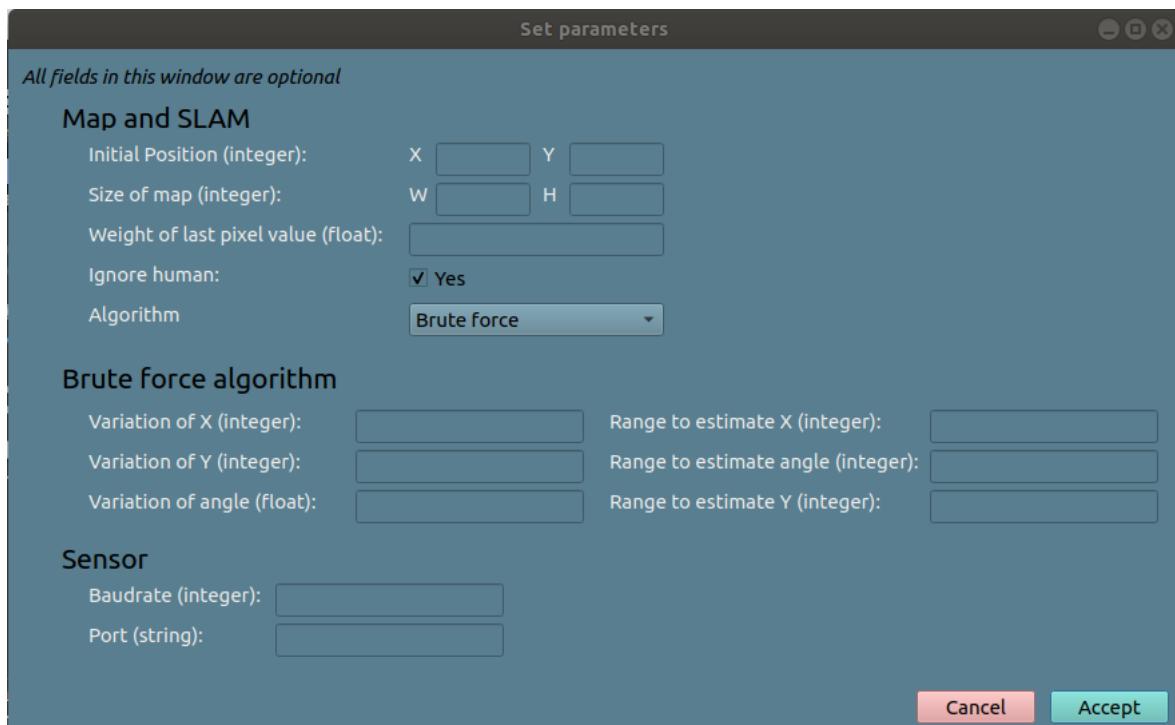


Figura 6.7: Interfaz final de la ventana ‘Set parameters’ (Brute force activo).

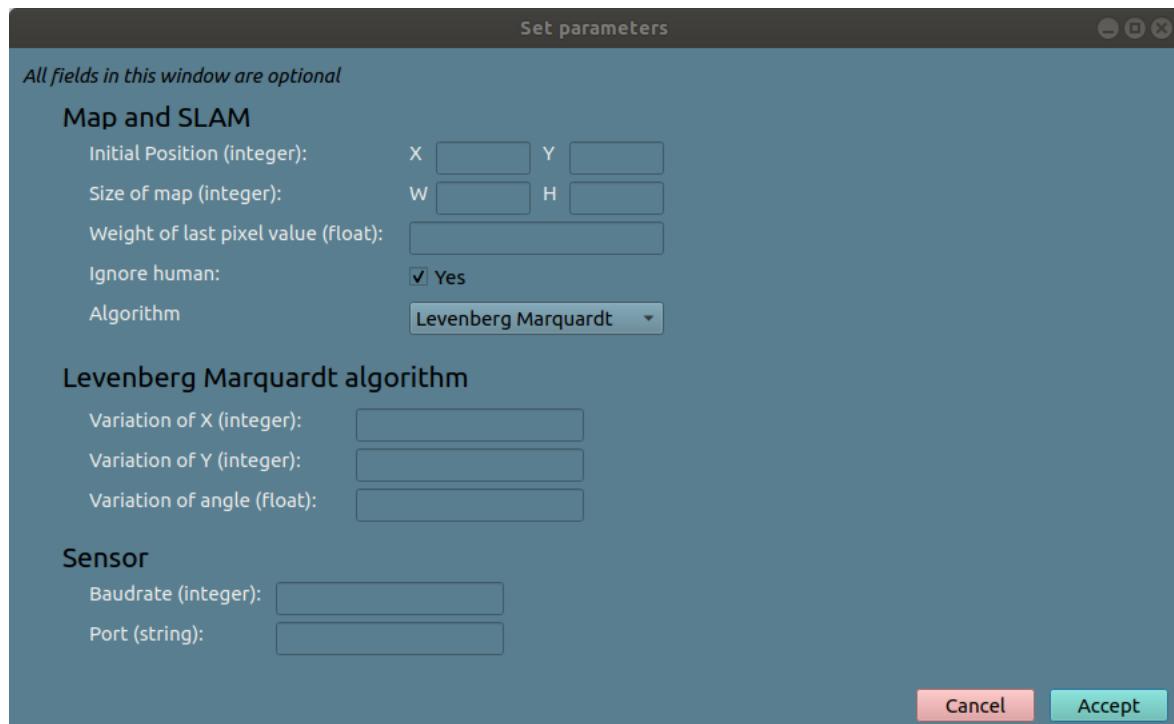


Figura 6.8: Interfaz final de la ventana ‘Set parameters’ (Levenberg Marquardt activo).

Capítulo 7

Pruebas

Este capítulo se dividirá en dos secciones, la sección referente a pruebas del funcionamiento del software, y la sección referente a las pruebas de los algoritmos (datos de precisión, velocidad, etc. para distintos parámetros).

7.1. Pruebas del software

En esta primera parte se explicarán qué tipo de pruebas se ha realizado al software para comprobar su funcionamiento y contrastar el resultado final con los requisitos y especificaciones previamente definidos. Se realizarán pruebas de dos tipos, que ya fueron explicados en el apartado de definición técnica del problema (2.2.10 Pruebas, página 12). Las pruebas son las siguientes:

- **De caja negra o funcionales:** estas pruebas tratarán de comprobar los casos de uso, ya que pretenden determinar la funcionalidad completa del software. Estas pruebas no pretenden ser exhaustivas ya que una prueba exhaustiva conllevaría utilizar muchos recursos en el apartado de pruebas. Sin embargo, se probarán los escenarios alternativos de los casos de uso (que en el caso de nuestro proyecto presentan errores, por lo que el programa debería avisar al usuario y mantener su estado).
- **De caja blanca o estructurales:** estas pruebas serán ejecutadas durante todo el proceso de desarrollo cada vez que se implemente un módulo. Si este módulo presenta fallos, se corregirán. Si dichos fallos son considerados relevantes, serán mencionados en las conclusiones de la memoria, pero no serán considerados dentro de este apartado, dado que la mayoría de las veces se documentarían fallos simples o que poco aportarían al proyecto.

A parte de estas pruebas, se brindó el software a usuarios ajenos al desarrollo y cuyo conocimiento informático es limitado. Estos usuarios supieron desenvolverse usando dicho software y no presentó ningún fallo, por lo que, además de comprobar el buen funcionamiento del software, se pudo comprobar que cumple con las especificaciones de usabilidad.

7.1.1. Pruebas funcionales

Para documentar las pruebas, se utilizará la siguiente plantilla:

CÓDIGO - NOMBRE DEL CASO DE PRUEBA	
Condiciones	Condiciones que deben darse para que se ejecute correctamente el caso de uso a probar.
Salida esperada	Salida que debería producirse.
Salida obtenida	Salida obtenida tras ejecutar el caso de uso.
Resultado	Correcto o incorrecto dependiendo de si la salida esperada coincide con la salida obtenida

Tabla 7.1: Plantilla para documentar las pruebas

C.P.1.1 - Creación de nuevo mapa (flujo principal)	
Condiciones	El usuario ha tenido que pulsar el botón para iniciar un nuevo mapa y se cumplen las precondiciones del caso de uso 1.
Salida esperada	El mapa empezará a crearse y se visualizará en el espacio especificado para ello.
Salida obtenida	El mapa se empezó a crear y se visualizaba correctamente.
Resultado	Correcto ✓

Tabla 7.2: Caso de prueba C.P.1.1 Creación de nuevo mapa (flujo principal).

C.P.1.2 - Creación de nuevo mapa (flujos alternativos)	
Condiciones	El usuario ha tenido que pulsar el botón para iniciar un nuevo mapa en cada uno de los siguientes casos: 1.2A - Se ha iniciado otro mapa. 1.2B - Se ha comenzado una simulación. 1.2C - Se ha finalizado un mapa y no se ha reseñado. 1.2D - El LiDAR no está conectado 1.2E - El puerto y el baudrate especificados son incorrectos.
Salida esperada	El sistema mostrará un mensaje de error dependiendo de la condición que no se cumpla y mantendrá su estado.
Salida obtenida	El sistema ha mostrado un mensaje de error para cada caso.
Resultado	Correcto ✓

Tabla 7.3: Caso de prueba C.P.1.2 Creación de nuevo mapa (flujos alternativos).

C.P.2.1 - Guardar mapa (flujo principal)	
Condiciones	El usuario debe haber pulsado el botón de guardar mapa y se deben cumplir las precondiciones del caso de uso 2.
Salida esperada	Se debe guardar el mapa con el nombre y el destino especificado por el usuario.
Salida obtenida	El mapa se ha guardado correctamente.
Resultado	Correcto ✓

Tabla 7.4: Caso de prueba C.P.2.1 Guardar mapa (flujo principal).

C.P.2.2 - Guardar mapa (flujos alternativos)	
Condiciones	El usuario ha tenido que pulsar el botón para guardar el mapa en cada uno de los siguientes casos: 2.2A - No se ha iniciado un mapa real. 2.2B - Hay una simulación activa 2.2C - Se ha finalizado un mapa y no se ha reseteado.
Salida esperada	El sistema mostrará un mensaje de error dependiendo de la condición que no se cumpla y mantendrá su estado.
Salida obtenida	Para todos los casos, el sistema muestra el mensaje de error correcto.
Resultado	Correcto ✓

Tabla 7.5: Caso de prueba C.P.2.2 Guardar mapa (flujos alternativos).

C.P.3.1 - Guardar simulación (Flujo principal)	
Condiciones	El usuario habrá pulsado el botón de guardar simulación y se darán las precondiciones para el caso de uso 3.
Salida esperada	El fichero de simulación se guardará con los datos adecuados en la ubicación y con el nombre especificados por el usuario.
Salida obtenida	El fichero se ha guardado correctamente.
Resultado	Correcto ✓

Tabla 7.6: Caso de prueba C.P.3.1. Guardar simulación (flujo principal)

C.P.3.2 - Guardar simulación (Flujos alternativos)	
Condiciones	El usuario habrá pulsado el botón de guardar simulación en cada uno de los siguientes casos: 3.2A El proceso de creación de mapa no está en el estado finalizado. 3.2B Hay una simulación corriendo.
Salida esperada	El sistema mostrará un mensaje de error dependiendo de la condición que no se cumpla y mantendrá su estado.
Salida obtenida	Para todos los casos, el sistema muestra el mensaje de error correcto.
Resultado	Correcto ✓

Tabla 7.7: Caso de prueba C.P.3.2 Guardar simulación (flujos alternativos).

C.P.4.1 - Resetear aplicación (Flujo principal)	
Condiciones	El usuario debe haber pulsado el botón para resetear la aplicación y se deben dar las precondiciones para el caso de uso 4.
Salida esperada	La aplicación se reseteará, restablecerá los valores predeterminados y limpiará la pantalla.
Salida obtenida	La aplicación se resetea correctamente.
Resultado	Correcto ✓

Tabla 7.8: Caso de prueba C.P.4.1. Resetear aplicación (flujo principal)

C.P.4.2 - Resetear aplicación (Flujos alternativos)	
Condiciones	El usuario debe haber pulsado el botón para resetear la aplicación en cada uno de los siguientes casos: 4.2A No se ha ejecutado una simulación ni se ha creado un mapa real. 4.2B Hay un mapa real en estado de creación. 4.2C Hay una simulación corriendo (no está ni parada ni finalizada).
Salida esperada	El sistema mostrará un mensaje de error dependiendo de la condición que no se cumpla y mantendrá su estado.
Salida obtenida	Para todos los casos, el sistema muestra el mensaje de error correcto.
Resultado	Correcto ✓

Tabla 7.9: Caso de prueba C.P.4.2 Resetear aplicación (flujos alternativos).

C.P.5.1 - Cargar simulación (Flujo principal)	
Condiciones	El usuario debe haber pulsado el botón de cargar simulación y se deben dar las precondiciones del caso de uso 5.
Salida esperada	La simulación debe quedar cargada y se debe mostrar un mensaje al usuario que muestre dicha información.
Salida obtenida	La simulación se carga correctamente.
Resultado	Correcto ✓

Tabla 7.10: Caso de prueba C.P.5.1. Cargar simulación (flujo principal)

C.P.5.2 - Cargar simulación (Flujos alternativos)	
Condiciones	El usuario debe haber pulsado el botón de cargar simulación en cada uno de los siguientes casos: 5.2A Hay un mapa real en estado de creación. 5.2B Hay un mapa real en estado de finalizado. 5.2C Hay una simulación corriendo. 5.2D Hay una simulación parada.
Salida esperada	El sistema mostrará un mensaje de error dependiendo de la condición que no se cumpla y mantendrá su estado.
Salida obtenida	Para todos los casos, excepto para el 5.2D, el sistema muestra el mensaje de error correcto.
Resultado	5.2D Incorrecto ✗, se procede a corregir, los demás correctos ✓

Tabla 7.11: Caso de prueba C.P.5.2 Cargar simulación (flujos alternativos).

C.P.6 - Modificar velocidad de simulación (Flujo principal)	
Condiciones	El usuario debe haber modificado la velocidad de la simulación.
Salida esperada	La simulación se ejecutará a la velocidad especificada por el usuario.
Salida obtenida	La simulación se modifica correctamente.
Resultado	Correcto ✓

Tabla 7.12: Caso de prueba C.P.6 Modificar velocidad de simulación (flujo principal).

C.P.7.1 - Comenzar/Parar simulación (Flujo principal)	
Condiciones	El usuario debe haber pulsado el botón de parar/continuar la simulación y se deben cumplir las precondiciones del caso de uso 7 (en cada uno de sus flujos principales).
Salida esperada	7.1A La simulación comenzará 7.1B La simulación se detendrá 7.1C La simulación continuará.
Salida obtenida	La simulación comienza se detiene y continúa como debería.
Resultado	Correcto ✓

Tabla 7.13: Caso de prueba C.P.7.1 Comenzar/Parar simulación (flujo principal).

C.P.7.2 - Comenzar/Parar simulación (Flujos alternativos)	
Condiciones	El usuario debe haber pulsado el botón de parar/continuar la simulación en cada uno de estos casos: 7.2A No hay una simulación cargada 7.2B Hay una simulación en estado finalizado. 7.2C No debe haber un mapa real siendo creado. 7.2D No debe haber un mapa real en estado finalizado.
Salida esperada	El sistema mostrará un mensaje de error dependiendo de la condición que no se cumpla y mantendrá su estado.
Salida obtenida	Para todos los casos, el sistema muestra el mensaje de error correcto.
Resultado	7.2D incorrecto ✗, se procede a corregir, los demás correctos ✓

Tabla 7.14: Caso de prueba C.P.7.2 Comenzar/Parar simulación (flujos alternativos).

C.P.8 - Pulsar un punto en el mapa (Flujo principal)	
Condiciones	El usuario debe haber pulsado un punto en el mapa y se deben dar las precondiciones del caso de uso 8.
Salida esperada	8.1A La información del punto se mostrará en pantalla como un único punto. 8.1B La información del punto se mostrará en pantalla como el segundo punto, manteniendo la información del primero. 8.1C Se borrará la información de los puntos 1 y 2 y será reemplazada con la información de un nuevo punto 1.
Salida obtenida	Los puntos se muestran como deberían.
Resultado	Correcto ✓

Tabla 7.15: Caso de prueba C.P.8 Pulsar un punto en el mapa (flujo principal).

C.P.9.1 - Calcular distancia entre puntos (Flujo principal)	
Condiciones	El usuario debe haber pulsado el botón de calcular distancia y hay dos puntos pulsados.
Salida esperada	La información de la distancia entre los dos puntos se mostrará en la pantalla.
Salida obtenida	La distancia se calcula correctamente.
Resultado	Correcto ✓

Tabla 7.16: Caso de prueba C.P.9.1 Calcular distancia entre puntos (flujo principal).

C.P.9.2 - Calcular distancia entre puntos (Flujos alternativos)	
Condiciones	El usuario debe haber pulsado el botón de calcular distancia sin que haya dos puntos pulsados.
Salida esperada	El sistema mostrará un mensaje de error avisando al usuario de que debe pulsar 2 puntos.
Salida obtenida	Para todos los casos, el sistema muestra el mensaje de error correcto.
Resultado	Correcto ✓

Tabla 7.17: Caso de prueba C.P.9.2 Calcular distancia entre puntos (flujos alternativos).

C.P.10.1 - Modificación de los parámetros (Flujo principal)	
Condiciones	El usuario pulsará el botón de modificar parámetros y se darán las precondiciones del caso de uso 10.
Salida esperada	10.1A Los parámetros serán correctamente modificados y almacenados. 10.1B Los parámetros no se modificarán y el sistema mantendrá su estado.
Salida obtenida	Los parámetros se modifican correctamente en el caso A y no se lleva a cabo ninguna acción como es lo esperado en el caso B.
Resultado	Correcto ✓

Tabla 7.18: Caso de prueba C.P.10.1 Modificación de los parámetros (flujo principal).

C.P.10.2 - Modificación de los parámetros (Flujos alternativos)	
Condiciones	El usuario pulsará el botón de modificar parámetros en cada uno de los siguientes casos: 10.2A Hay un mapa real siendo creado. 10.2B Hay un mapa real finalizado. 10.2C Hay una simulación finalizada. 10.2D Hay una simulación corriendo.
Salida esperada	El sistema mostrará un mensaje de error dependiendo de la condición que no se cumpla y mantendrá su estado.
Salida obtenida	Para todos los casos, el sistema muestra el mensaje de error correcto.
Resultado	10.2B y 10.2D incorrectos ✗, se procede a corregirlos, los demás correctos ✓

Tabla 7.19: Caso de prueba C.P.10.2 Modificación de los parámetros (flujos alternativos).

7.2. Pruebas de los algoritmos

Para entender mejor cómo se han hecho las pruebas, primero se procederá a mostrar el entorno en el que se llevaron a cabo (figuras 7.1 a 7.4).



Figura 7.1: Entorno de prueba vista 1.



Figura 7.2: Entorno de prueba vista 2.



Figura 7.3: Entorno de prueba vista 3.



Figura 7.4: Entorno de prueba vista 4.

En este entorno se ha grabado un fichero de simulación (con la estructura que ya se

ha mencionado) para recrear la escena tantas veces como sea necesario, modificando los parámetros de los algoritmos para obtener resultados y compararlos.

La experimentación se llevará a cabo de la siguiente forma: para ambos algoritmos se han tenido en cuenta una serie de parámetros que podría funcionar en un entorno de tiempo real (por ejemplo, no vamos a probar un rango de posibles posiciones de 1m cuando sería imposible llevar esto a la práctica en un ordenador personal). Estos parámetros determinarán la precisión del algoritmo, así como su potencial para detectar grandes cambios en cuanto a la posición y orientación del sistema. Esto deriva en un tiempo de cómputo y una capacidad del sistema para perderse y encontrarse que varían dependiendo de los parámetros usados.

En la tabla 7.20 se pueden apreciar los distintos parámetros que se han usado para el algoritmo de fuerza bruta en comparación con el tiempo de cómputo y tanto las veces que se ha perdido el sistema como las que encontrado. En la tabla 7.21 se puede apreciar la misma relación para el algoritmo Levenberg Marquardt (con distintos parámetros). Los parámetros que se usarán tienen la siguiente forma: “variación de x / variación de y / variación del ángulo / rango para estimar x / rango para estimar y / rango para estimar el ángulo”.

Con los mejores resultados (o los que se puedan usar en un entorno de tiempo real, los cuáles estarán marcados en un color más oscuro), se pasará a estudiar el mapa generado para ver cuál sería la posible mejor combinación de parámetros, que aporte unas mayores prestaciones y cree el mapa más decente. A priori, el tiempo de cómputo debería ser algo mayor en la creación del mapa real que en la simulación, ya que es necesario extraer las mediciones del entorno, por lo tanto, cuando se escogen los mejores parámetros en simulación, se comprobará el tiempo de cómputo en la creación del mapa real. Este tiempo de cómputo ha sido calculado para el PC que aparece en el apartado recursos.

Parameters	Tiempo de cómputo(ms)	Veces perdido	Veces encontrado
1 / 1 / 2 / 40 / 40 / 100	~7100	0	0
1 / 1 / 2 / 30 / 30 / 80	~3400	0	0
1 / 1 / 2 / 20 / 20 / 60	~1100	0	0
2 / 2 / 3 / 40 / 40 / 100	~1250	0	0
2 / 2 / 3 / 30 / 30 / 80	~600	0	0
2 / 2 / 3 / 20 / 20 / 60	~230	0	0
2 / 2 / 4 / 40 / 40 / 100	~900	0	0
2 / 2 / 4 / 30 / 30 / 80	~400	0	0
2 / 2 / 4 / 20 / 20 / 60	~190	0	0
3 / 3 / 4 / 40 / 40 / 100	~460	0	0
3 / 3 / 4 / 30 / 30 / 80	~220	0	0
3 / 3 / 4 / 20 / 20 / 60	~120	0	0
4 / 4 / 5 / 40 / 40 / 100	~260	1	1

Tabla 7.20: Tabla de resultados para brute force.

En el caso de Levenberg Marquardt no tiene sentido aplicar un rango de estimación ya que es un método de búsqueda local, que ejecutará un número de iteraciones hasta minimizar el error, por lo que aparecen menos parámetros en la tabla.

Parameters	Tiempo de cómputo	Veces perdido	Veces encontrado
1 / 1 / 1	~60	4	1
1 / 1 / 1.5	~60	2	0
1 / 1 / 2	~60	3	1
2 / 2 / 1.5	~60	3	1
2 / 2 / 2	~60	2	1
2 / 2 / 3	~60	1	0
2 / 2 / 4	~60	3	0
3 / 3 / 1.5	~60	2	1
3 / 3 / 2	~60	4	0
3 / 3 / 3	~60	1	0
3 / 3 / 4	~60	1	0
3 / 3 / 5	~60	1	0
4 / 4 / 6	~60	3	0
5 / 5 / 6	~60	2	0
6 / 6 / 6	~60	2	0

Tabla 7.21: Tabla de resultados para Levenberg Marquardt.

7.3. Discusión de los resultados.

Como se puede ver en la tabla de los resultados de brute force, hay 4 combinaciones de parámetros que podrían ser interesantes:

- 2 / 2 / 3 / 20 / 20 / 60: esta combinación permite una precisión decente utilizando una variación de la posición en 2cm y la del ángulo en 3º y un tiempo de cómputo (230ms) que permitiría unos 4 frames por segundo (es decir, que el mapa se actualice 5 veces por segundo). Sin embargo, tiene poca tolerancia a cambios bruscos, ya que el rango son 20cm (10cm a cada lado) y 60º (30º a cada lado).

- 2 / 2 / 4 / 20 / 20 / 60: de nuevo existe una baja tolerancia a cambios bruscos, pero disminuye el tiempo de cómputo a costa de disminuir la precisión de la estimación del ángulo. Aún así, como se podrá apreciar en el mapa generado, es un mapa bastante similar al generado por la opción anterior, por lo que depende del caso, podría llegar a ser una opción mejor.
- 3 / 3 / 4 / 30 / 30 / 80: en este caso disminuye la precisión a costa de aumentar la tolerancia a cambios bruscos. El tiempo de cómputo se mantiene similar a la primera opción, lo cuál sugiere que dependiendo de la situación del usuario, podría preferir uno u otro.
- 3 / 3 / 4 / 20 / 20 / 60: esta opción permite el menor tiempo de cómputo (120ms), pero también es la que menor precisión y tolerancia a cambios bruscos tiene.

Si nos paramos a pensar en la tolerancia a cambios bruscos, realmente no solo depende del rango, sino también del tiempo de cómputo. Un menor tiempo de cómputo permitiría actualizar el mapa cada menos tiempo, por lo que los cambios serían menos notables en cada cambio. Dada estos datos, personalmente creo que la segunda opción sería la más acertada, permitiendo algo más de 5 frames por segundo con una precisión de 2cm y 4º. Dado que en 200ms es muy difícil que el sistema se mueva más de 10cm hacia alguna de las direcciones o gire más de 30º, creo que puede ser una opción acertada.

Los mapas generados por las distintas combinaciones serían los siguientes (figuras 7.5-7.9):

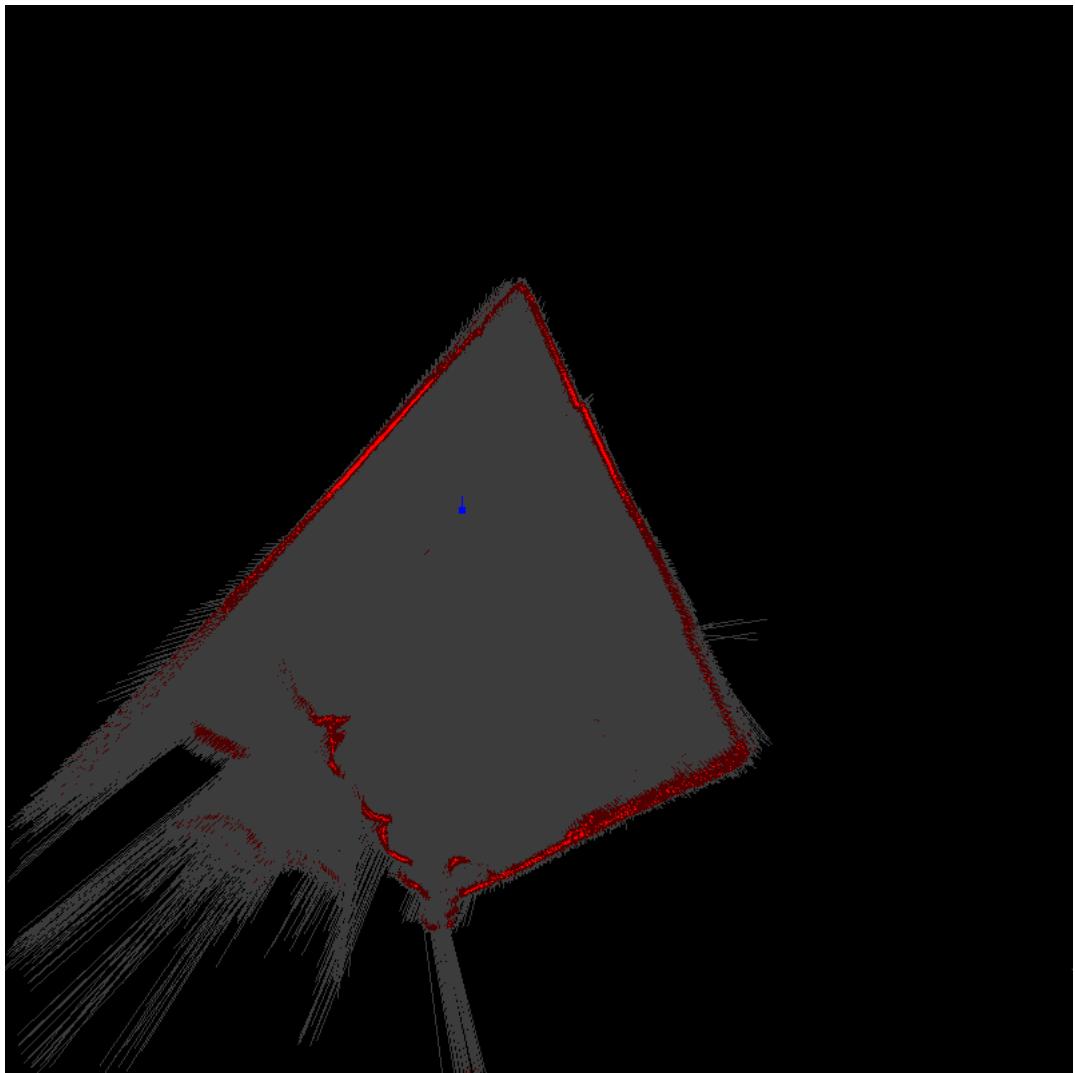


Figura 7.5: Mapa generado por brute force con parámetros: 2 / 2 / 3 / 20 / 20 / 60.

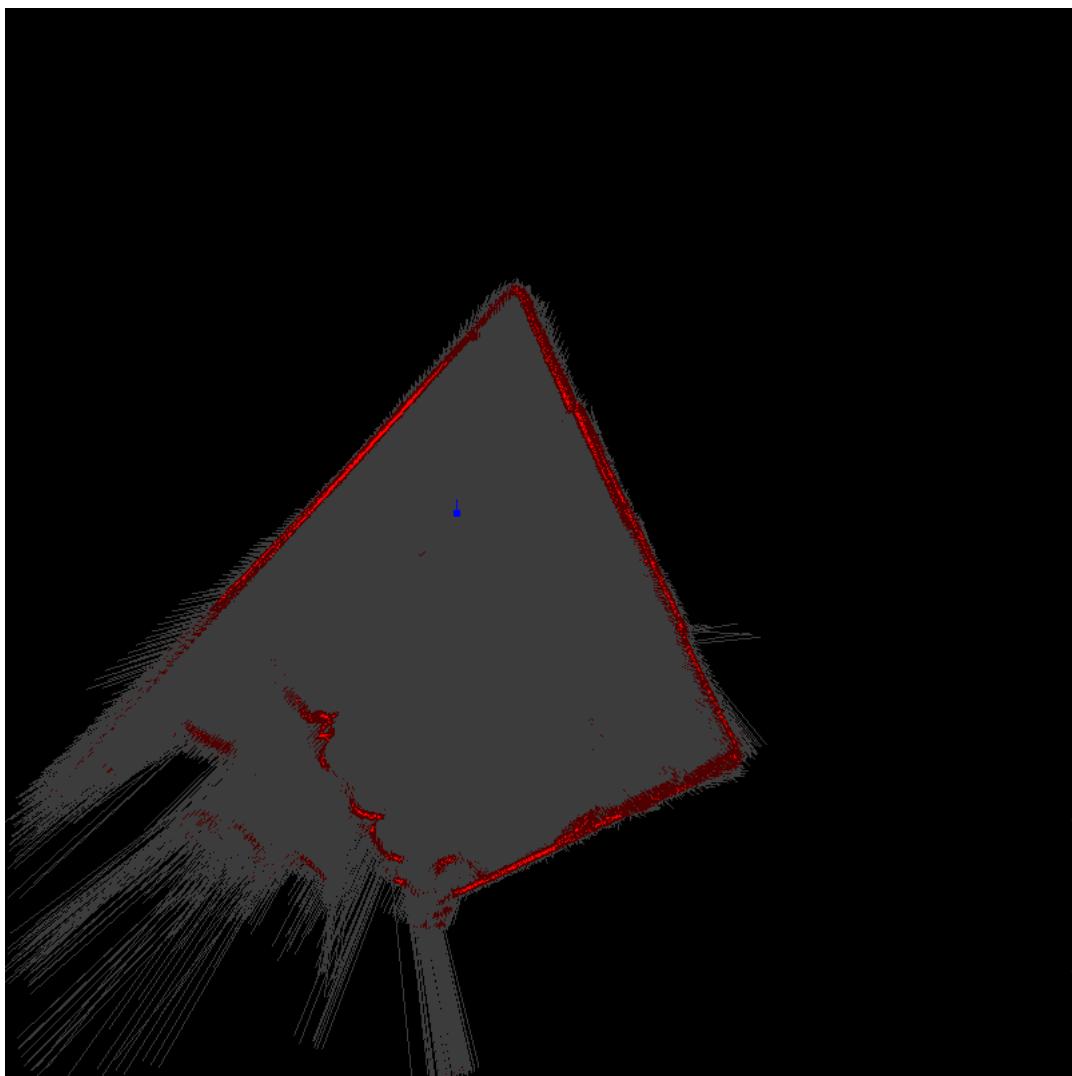


Figura 7.6: Mapa generado por brute force con parámetros: 2 / 2 / 4 / 20 / 20 / 60.

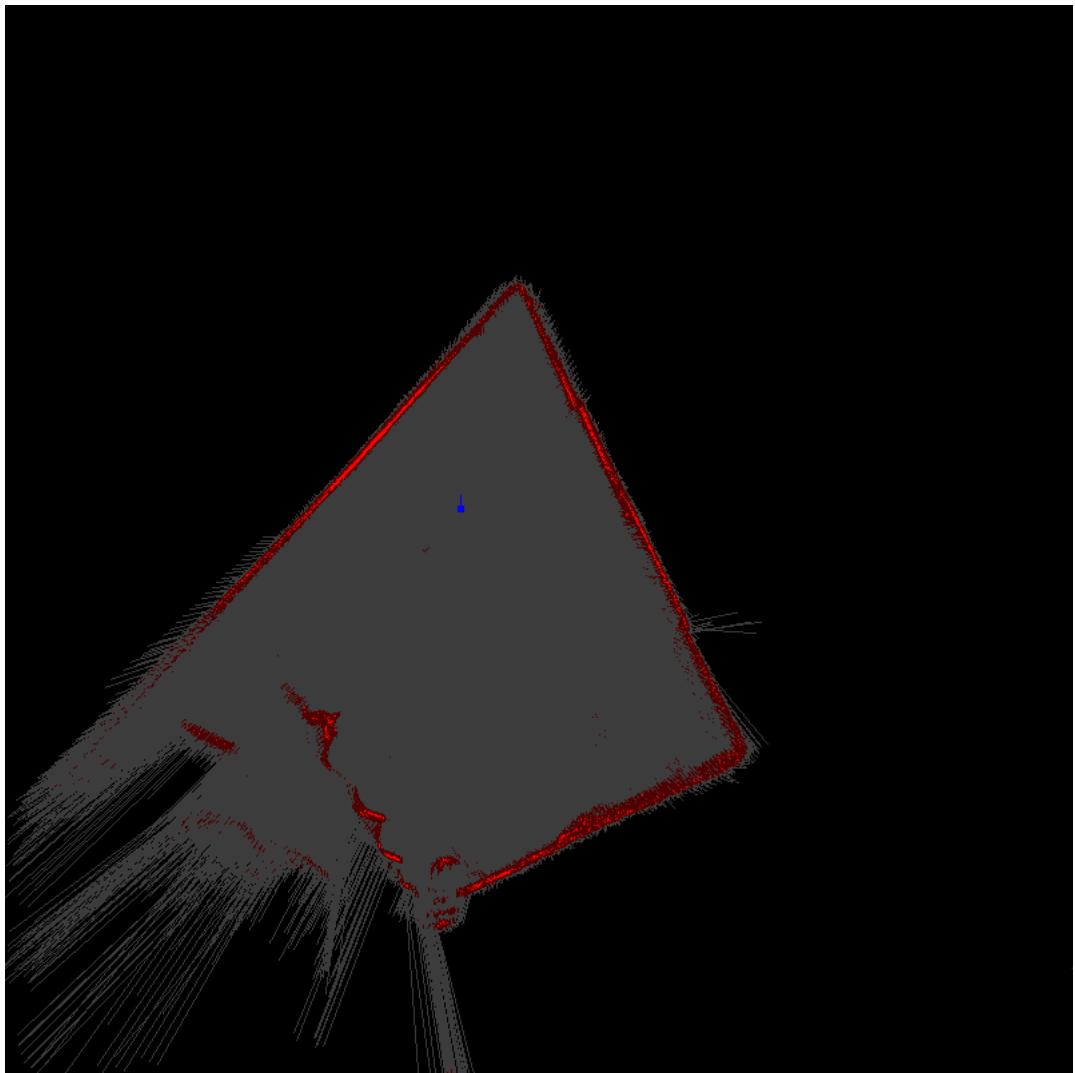


Figura 7.7: Mapa generado por brute force con parámetros: 2 / 2 / 4 / 20 / 20 / 60.

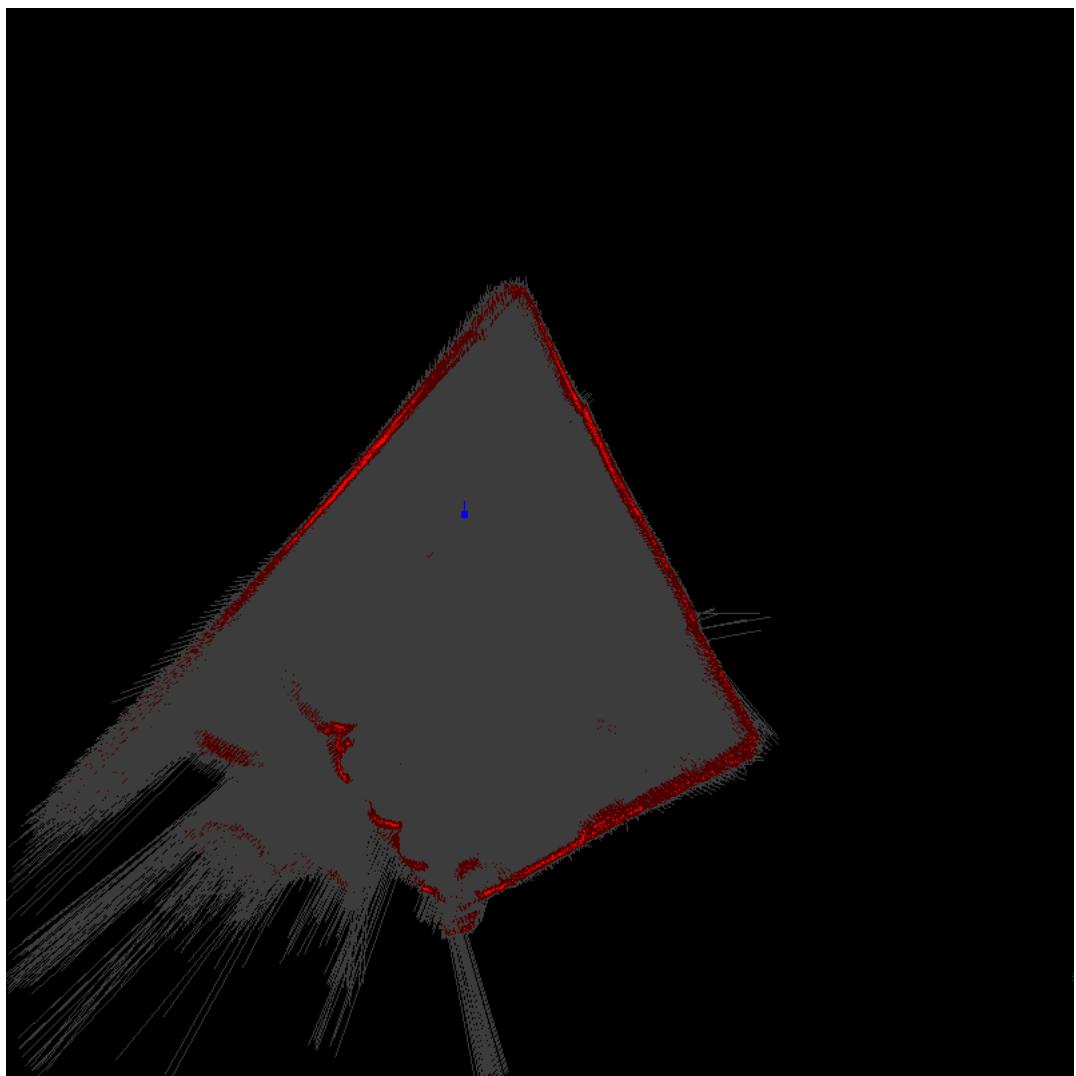


Figura 7.8: Mapa generado por brute force con parámetros: 3 / 3 / 4 / 20 / 20 / 60.

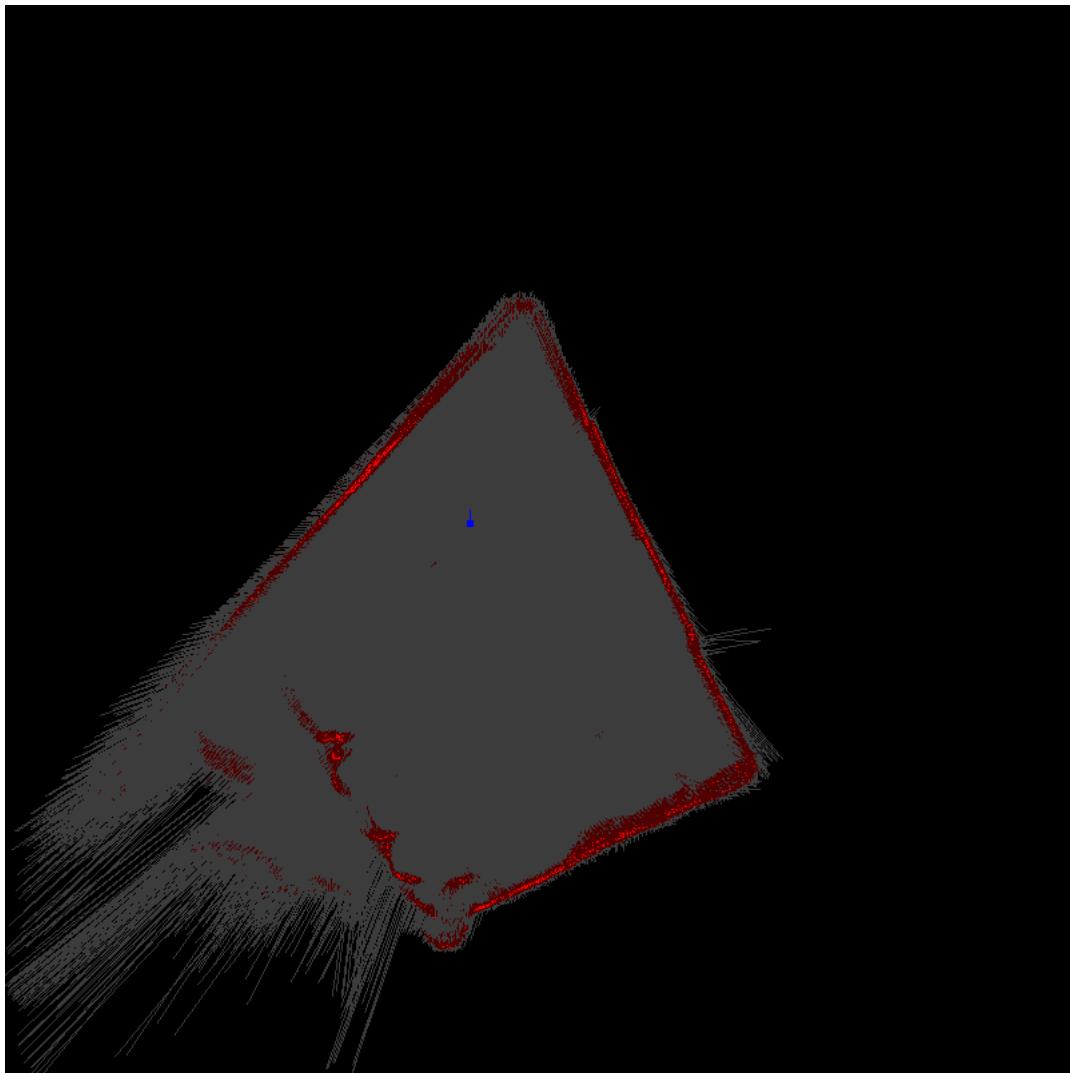


Figura 7.9: Mapa generado por brute force con parámetros: 4 / 4 / 5 / 40 / 40 / 100.

Como se podrá comprobar, la detección láser del LiDAR tiene algunos problemas con materiales en los que la luz puede llegar a traspasarlos, como las cortinas. Si se sigue el recorrido de la simulación, cuando el sistema está cerca se identifican correctamente, pero conforme se aleja, las cortinas son más difíciles de identificar por parte del sistema. Téngase en cuenta que este sistema permite mapear entornos dinámicos, donde los objetos pueden aparecer y desaparecer.

Se ha incluido también el mapa generado para los parámetros 4 / 4 / 5 / 40 / 40 / 100. Como se puede ver, con estos parámetros el sistema sufre una leve pérdida de orientación y además los puntos del mapa se hacen más difusos, dada la pérdida de precisión, por lo que seguir decrementando las prestaciones deja de ser interesante. También se puede apreciar que para los parámetros con más precisión, hay más puntos rojos brillantes, lo que quiere decir que los rayos del LiDAR se concentran más en una zona y por consiguiente el sistema se está identificando con más precisión.

Otro detalle a tener en cuenta, es que en la simulación el tiempo entre dos medidas no importa, ya que se han grabado y están almacenadas en memoria, pero en una situación real, si el tiempo entre medidas es muy alto, el sistema puede haber cambiado de posición

a una totalmente alejada. Esto hace que las veces que se pierde el sistema en simulación sean menores, ya que no depende de la recogida de medidas.

Se puede ver que la precisión de las medidas son bastante acertadas, e incluso se visualiza en los mapas el pequeño desnivel que existe en las puertas del armario que se puede ver en las figuras 7.1-7.3.

Para Levenberg Marquardt los resultados obtenidos son muy buenos con respecto al tiempo, pero sin embargo, el sistema se pierde muchas veces y no es capaz de encontrarse. Hay una serie de combinaciones que hacen que el sistema se pierda menos, pero aún así los mapas generados tienen fallos mucho mayores que los anteriores, como se puede apreciar en la figura 7.10 (el mejor mapa obtenido):

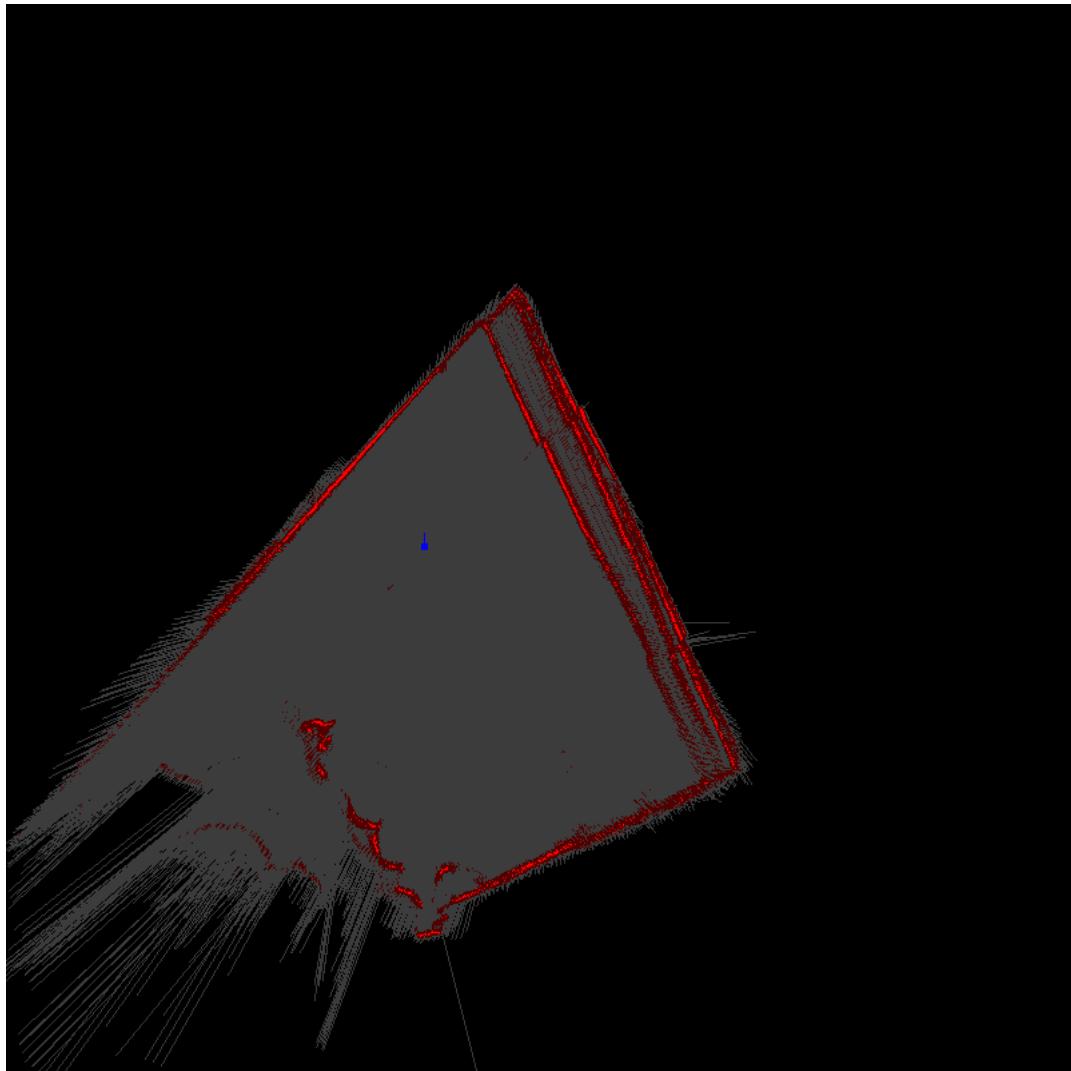


Figura 7.10: Mapa generado por Levenberg Marquardt con parámetros: 3 / 3 / 5.

Esta aproximación utilizando Levenberg Marquardt por lo tanto obtiene unos resultados peores con mapas que no se asemejan demasiado a la realidad. Sin embargo, si se dispusiese de un sistema con muy bajo procesamiento, fuerza bruta no se podría llevar a cabo y sería una buena opción utilizar Levenberg Marquardt.

Como ya comentábamos antes, Levenberg Marquardt aporta un modelo no lineal mediante mínimos cuadrados amortiguados. Por los resultados obtenidos, el algoritmo se estanca en óptimos locales más de lo deseado, por lo que quizás sería interesante utilizar otro modelo no lineal, o tal y como afirman en [28], utilizar un modelo lineal que aporte unos buenos resultados. También se comenta en ese artículo que incluir el mapa en la estimación de la pose del sistema reduce la no linealidad, por lo que un modelo lineal podría ser más eficaz.

Si se implementase en un sistema independiente como podría ser una raspberry pi, una buena opción sería mandar las medidas a un PC con mayor capacidad de procesamiento para que pudiese emplear fuerza bruta.

Capítulo 8

Conclusiones

En este capítulo se detallarán las conclusiones a las que se ha llegado, los resultados obtenidos con el proyecto final y se relacionarán los objetivos que definimos al principio con los que hemos conseguido. También se hablará de los fallos que se comentaron en el capítulo de pruebas y se hablará de las mejoras que se podrían implementar en un futuro, si se continuase con este proyecto.

8.1. Conclusiones sobre los objetivos

Una vez finalizado el proyecto y la fase de pruebas, se puede afirmar que se han cumplido los objetivos que se propusieron al comienzo del proyecto.

Se ha diseñado un sistema con capacidad de extracción de datos del entorno, que es capaz de generar un mapa con dichos datos y visualizarlo en tiempo real (5 frames por segundo con fuerza bruta o 10 frames por segundo con Levenberg Marquardt, usando los parámetros predeterminados), a la vez que se identifica dentro de dicho mapa. Como se ha podido comprobar en el análisis de resultados, la precisión de las medidas y la identificación del sistema son aceptables.

El sistema final está compuesto por el LiDAR y un portátil para utilizar la aplicación y ha sido diseñado para que pueda ser portado por un humano (que será ignorado en las mediciones si así lo desea) o por otro sistema que permita su movimiento.

Se ha diseñado una interfaz gráfica de usuario para facilitar el uso del sistema, y permite guardar los mapas generados como imágenes, calcular distancias entre puntos, guardar las medidas en un fichero de simulación y recrear los mapas reales mediante dichos ficheros de simulación, con opción a parar o continuar dichas simulaciones para guardar una instantánea del mapa actual, o pararse a comprobar la distancia entre puntos.

8.2. Conclusiones sobre las pruebas

Como es normal en un proyecto software, algunas de las pruebas finales fallaron, pero automáticamente se corrigieron dichos errores, dando lugar a un proyecto cuyas funcionalidades principales están a disposición del usuario y cuya estructura interna fue depurada durante la fase de desarrollo gracias a las pruebas estructurales. Todas estas pruebas proporcionan un software de calidad y fiable. Si bien es cierto que el número de pruebas

podría ser mayor, esto, como ya se comentó, requeriría muchos recursos.

Con respecto a SLAM, los resultados obtenidos son buenos tanto para el algoritmo de fuerza bruta como para Levenberg Marquardt, obteniendo en torno a 5 frames por segundo mediante fuerza bruta y unos 10 frames por segundo mediante Levenberg Marquardt, lo cual para un sistema de mapeo de bajo coste y consumo no está mal.

El sistema por tanto, se identificará a sí mismo dentro del mapa 5 veces por segundo, a la vez que toma medidas de su entorno y las dibuja en dicho mapa. Para nuestro caso, ambos algoritmos dan unos resultados aceptables, con una precisión decente, aunque fuerza bruta permita al sistema identificarse mejor. Fuerza bruta podría ser aún más preciso a costa de una mayor carga computacional, lo cuál no es muy rentable para un sistema en tiempo real, como ya se ha comprobado en el capítulo de pruebas.

Algunos de los fallos que se presentaron durante las pruebas estructurales pueden ser dignos de mención dado que podría ser útiles para futuros proyectos:

- El primero podría ser la forma en la que se ha visualizado el mapa en Qt. Para ello, fue necesario crear un hilo de ejecución, ya que las mediciones y demás cálculos necesitan realizarse de manera reiterada. Esto bloquearía a la ventana principal y no podría actualizar los elementos visuales. La solución fue crear un hilo y con ayuda de la herramienta de interrupciones de Qt (signals y slots), avisar a la ventana principal para que actualice el mapa cuando el hilo ha realizado los cálculos y tiene el mapa listo para visualizarlo.
- El segundo fallo destacable es referente a SLAM. En una primera aproximación, se intentó comparar las medidas de una medición completa con las medidas justo anteriores, sin tener en cuenta el mapa completo. Esto provocaba muchos fallos y discordancias, por lo que la solución era crear el mapa y conservar los valores de dicho mapa, para comprar posteriormente con estos valores y no únicamente con los de la iteración anterior.
-

8.3. Futuras mejoras

Como posibles futuras mejoras, se podrían añadir opciones varias a la aplicación, como la opción para realizar zoom en el mapa o añadir una herramienta para crear simulaciones a gusto del usuario.

También se podría mejorar el algoritmo de SLAM utilizando algún tipo de metaheurística como algoritmos basados en poblaciones o trayectoria. Así mismo, también se podría barajar la posibilidad de utilizar una red neuronal convolucional, utilizando los mapas con imágenes, aunque tendría el inconveniente de los datos, que habría que generar una gran cantidad de estos y etiquetarlos.

Otra posible mejora para el algoritmo de Levenberg Marquardt sería utilizar algún tipo de factor de momento para suavizar los posibles errores en la estimación de este modelo.

Capítulo 9

Agradecimientos

He de agradecer la colaboración con este proyecto a Mirian Palacios Caballero, persona que desarrolló las imágenes de ejemplo de uso del sistema y el ícono de la aplicación, así como asesoró el desarrollo de la interfaz de usuario. También he de agradecer la colaboración con el proyecto a las personas que realizaron las pruebas a ciegas del software y por supuesto al director del proyecto Rafael Muñoz Salinas.

Bibliografía

- [1] Nico Lang y Jan Wegner. “Country-wide high-resolution vegetation height mapping with Sentinel-2”. En: *Remote Sensing of Environment* 233 (nov. de 2019), pág. 111347. DOI: 10.1016/j.rse.2019.111347.
- [2] D. Maturana y S. Scherer. “3D Convolutional Neural Networks for landing zone detection from LiDAR”. En: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, págs. 3471-3478. DOI: 10.1109/ICRA.2015.7139679.
- [3] Chen Xiong, Qiangsheng Li y Xinzheng Lu. “Automated regional seismic damage assessment of buildings using an unmanned aerial vehicle and a convolutional neural network”. En: *Automation in Construction* 109 (nov. de 2019). DOI: 10.1016/j.autcon.2019.102994.
- [4] aerial insight. *Lidar vs fotogrametría: ¿qué tecnología es mejor?* <https://www.aerial-insights.co/blog/lidar-vs-fotogrametria/>. Accessed on 2020-01-31.
- [5] <https://www.kickstarter.com/projects/1697979147/lidar-for-everyone-hydro-ilidar?lang=es>. iLiDAR.
- [6] *Web oficial del LiDAR RPLIDAR A1M8.* <https://www.slamtec.com/en/Lidar/A11>.
- [7] *Web oficial de Visual Studio Code.* <https://code.visualstudio.com/>.
- [8] *Web oficial de Qt.* <https://www.qt.io/download>.
- [9] Jorge Fuentes-Pacheco, Jose Ascencio y J. Rendon-Mancha. “Visual Simultaneous Localization and Mapping: A Survey”. En: *Artificial Intelligence Review* 43 (nov. de 2015). DOI: 10.1007/s10462-012-9365-8.
- [10] H. Durrant-Whyte y T. Bailey. “Simultaneous localization and mapping: part I”. En: *IEEE Robotics Automation Magazine* 13.2 (2006), págs. 99-110. ISSN: 1558-223X. DOI: 10.1109/MRA.2006.1638022.
- [11] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss y Wolfram Burgard. “A tutorial on graph-based SLAM”. En: *IEEE Transactions on Intelligent Transportation Systems Magazine* 2 (dic. de 2010), págs. 31-43. DOI: 10.1109/TITS.2010.939925.
- [12] M.V. Valueva, N.N. Nagornov, P.A. Lyakhov, G.V. Valuev y N.I. Chervyakov. “Application of the residue number system to reduce hardware costs of the convolutional neural network implementation”. En: *Mathematics and Computers in Simulation* 177 (2020), págs. 232 -243. ISSN: 0378-4754. DOI: <https://doi.org/10.1016/j.matcom.2020.04.031>. URL: <http://www.sciencedirect.com/science/article/pii/S0378475420301580>.

- [13] Martin Weinmann, Boris Jutzi, Stefan Hinz y Clément Mallet. “Semantic point cloud interpretation based on optimal neighborhoods, relevant features and efficient classifiers”. En: *ISPRS Journal of Photogrammetry and Remote Sensing* 105 (feb. de 2015). DOI: 10.1016/j.isprsjprs.2015.01.016.
- [14] Raúl Álvarez. *El director de inteligencia artificial de Tesla nos explica en vídeo cómo entrena sus redes neuronales para el uso de Autopilot*. <https://www.xataka.com/robotica-e-ia/director-inteligencia-artificial-tesla-nos-explica-video-como-entrenan-sus-redes-neuronales-para-uso-autopilot>. Accessed on 2020-02-02.
- [15] <https://play.google.com/store/apps/details?id=com.google.ar.unity.ddelements&hl=es>. ARcore de google.
- [16] <https://community.arm.com/developer/tools-software/graphics/b/blog/posts/indoor-real-time-navigation-with-slam-on-your-mobile>. Navegación en interiores con ARcore de google.
- [17] https://www.storececotec.com/es/29-robots-aspiradores?gclid=Cj0KCQjwzZj2BRDVARIsq5JuFq6Q5vweGWqt7xMh1Cmt5Y4eSfSjwLx09Lw-LwAaApc6EALw_wcB. Robots conga.
- [18] <https://www.profesionalreview.com/2018/10/21/conga-serie-3090-review/>. Review de conga.
- [19] <https://www.irobot.es/roomba>. Robots conga.
- [20] <https://www.ros.org/>. Web oficial de ROS.
- [21] http://wiki.ros.org/hector_slam. Documentación de Hector SLAM.
- [22] Tiantao Cheng, Chen Xu, Junyan Duan, Wang Yifan, Chunpeng Leng, Jun Tao, Huizheng CHE, Qianshan He, Yunfei Wu, Renjian Zhang, Xiang Li, Jian-Min Chen, Lingdong Kong y Xingna Yu. “Seasonal variation and difference of aerosol optical properties in columnar and surface atmospheres over Shanghai”. En: *Atmospheric Environment* 123 (mayo de 2015). DOI: 10.1016/j.atmosenv.2015.05.029.
- [23] Detlef Müller, A Ansmann, Ina Mattis, Matthias Tesche, U Wandinger, D Althausen y Gianluca Pisani. “Aerosol-type-dependent LIDAR ratios observed with Raman LIDAR”. En: *Journal of Geophysical Research* 112 (ago. de 2007). DOI: 10.1029/2006JD008292.
- [24] Randall Smith, Matthew Self y Peter Cheeseman. “Estimating Uncertain Spatial Relationships in Robotics”. En: vol. 1. Ene. de 1986, págs. 435-461. DOI: 10.1109/ROBOT.1987.1087846.
- [25] Milagros Martínez Díaz. *Algoritmos de construcción de mapas para la navegación de robots, basada en información procedente de Sensores de ultrasonidos*. <http://personales.upv.es/mmartind/6cred/6cred.pdf>.
- [26] *Web oficial de Visual Paradigm*. <https://online.visual-paradigm.com/es/>.
- [27] *Qt, signals and slots*. <https://doc.qt.io/qt-5/signalsandslots.html>.
- [28] Shoudong Huang, Yingwu Lai, Udo Frese y Gagini Dissanayake. “How far is SLAM from a linear least squares problem?” En: oct. de 2010, págs. 3011-3016. DOI: 10.1109/IROS.2010.5652603.