Proyecto Final de Máster

Máster en Ingeniería de Sistemas y de Control

"Navegación de un robot autónomo utilizando balizas y luz estructurada"

Autor: Juan Ignacio Forcén Carvalho

Director: Dictino Chaos García

Curso académico 2014/2015 - Convocatoria de Septiembre

Proyecto Final de Máster

Máster en Ingeniería de Sistemas y de Control

Título:

"Navegación de un robot autónomo utilizando balizas y luz estructurada"

Proyecto Tipo B

Autor: Juan Ignacio Forcén Carvalho

Director: Dictino Chaos García

Autorización

Autorizamos a la Universidad Complutense y a la UNED a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firmado:

Juan Ignacio Forcén Carvalho

Resumen

Se puede definir un robot autónomo como un robot que es capaz de realizar tareas por sí mismo, en un robot móvil esto quiere decir que el robot tiene la capacidad de localizarse y de navegar de un punto a otro en su entorno. La navegación de un robot se compone de: localización, planificación de trayectorias y construcción de mapas e interpretación.

Desde el punto de vista del robot, podemos decir que la construcción de mapas sirve al mismo como una representación del entorno o mundo en el que se mueve; la localización es la capacidad del robot de determinar su posición y orientación respecto a un sistema de coordenadas o referencia; y la planificación de trayectorias implica la capacidad del robot de establecer o calcular cual es la ruta o camino a seguir para ir de una localización a otra de su entorno.

En este proyecto se ha implementado la navegación de un robot autónomo, utilizando la detección de balizas y de luz estructurada, por sendos sistemas de visión artificial para determinar la localización y para sensorizar la geometría del entorno.

La solución realizada es a su vez una solución simple y de bajo coste al problema.

Palabras clave

Navegación basada en marcadores, Luz estructurada, SLAM, Visión por computador, OpenCV.

Índice general

INTRODUCCIÓN 1
1 Introducción 2
1.1 Marco del proyecto 2
1.2 Motivación y objetivos 2
1.3 Alcance del proyecto 2
1.4 Organización del documento 3
2 Estado del arte 4
2.1 Antecedentes de Investigación 4
2.2 Necesidades 7
DESARROLLO 8
3 Robot – Estructura de control 9
4 Localización 13
4.1 Localización por visión 13
4.2 Cálculo de la posición 17
4.3 Aplicación práctica 23
4.4 Filtro de partículas 27
4.5 Odometría 34
5 Mapping 36
5.1 Detección láser 36
5.2 Construcción mapa 44
6 Navegación 48
7 Cálculo de trayectorias 49
8 Diseño software 51
8.1 Introducción 51
8.2 ¿Por qué Linux? 51
8.3 ¿Por qué Debian? 51
8.4 ¿Por qué C++? 52
8.5 ¿Por qué OpenCV 52
8.6 Diagramas UML 53

8.6.1 Diagrama de estado ... 53
8.6.2 Diagramas de actividad ... 54
8.6.3 Casos de uso ... 58
8.6.4 Diagramas de secuencia ... 59
9 Resultados ... 64
9.1 Detección de marcadores ... 64
9.2 Filtro de partículas ... 67
9.3 Odometría ... 68
9.4 Detección láser ... 69
9.5 Mapping ... 72
9.6 Navegación ... 74
9.7 Estructura de control ... 76
9.8 Diseño software ... 76
10 Descripción de la aplicación ... 79
CONCLUSIONES ... 84
11 Conclusiones y líneas futuras ... 85
11.1 Conclusiones ... 85
11.2 Líneas futuras ... 86
Referencias – Bibliografía ... 88
Abreviaturas y acrónimos ... 92
ANEXOS ... 93
A Planificación ... 94
B Diagrama de Gantt ... 95
C Estructura de descomposición del proyecto (EDP) ... 96
D Presupuesto ... 100
Mediciones y precio unitario ... 100
Presupuesto total ... 101
E Tablas de casos de uso ... 102

Índice de figuras

Figura 1: Stanford Cart y Shakey. 4
Figura 2: LS3 de Boston Dinamics. 5
Figura 3: Stanley, primer ganador del "Darpa Grand Challenge". 5
Figura 4: Estructura de control de Stanley. 6
Figura 5: Robot. 9
Figura 6: Placa Arduino Uno R3 y Arduino Motor Shield R3. 9
Figura 7: Robot con portátil cámaras y láser incorporado. 10
Figura 8: Esquema de comunicaciones. 11
Figura 9: Estructura de capas. 12
Figura 10: Marcadores basados en color. 13
Figura 11: Detección marcadores basados en color. 14
Figura 12: Pruebas con marcadores basados en códigos QR. 15
Figura 13: Primeras pruebas con marcadores de realidad aumentada ArUco. 16
Figura 14: Efectos de deformación, ejemplo de calibración. 16
Figura 15: Marcadores en techo 17
Figura 16: Detección primer marcador. 18
Figura 17: Robot se desplaza y sigue detectando el primer marcador. 19
Figura 18: Robot detecta nuevo marcador. 20
Figura 19: Diagrama de flujo detección marcadores. 22
Figura 20: Estados marcadores. 24
Figura 21: Evolución estados marcadores. 25
Figura 22: Secuencia de imágenes en las que se observa la variabilidad en las rotaciones del sistema de coordenadas detectado del marcador en función de la posición con la que se ha obtenido la foto. 26
Figura 23: Imagen que representa 4 posiciones devueltas por 4 marcadores diferentes para una posición de robot. 27
Figura 24: Partículas aleatorias en el espacio de trabajo. 29
Figura 25: Imagen que representa 4 posiciones devueltas por 4 marcadores diferentes para una posición de robot. 30
Figura 26: Representación de partículas aleatorias y medidas de marcadores. 31
Figura 27: Partículas que sobreviven tras unos ciclos. 32
Figura 28: A la izquierda partículas aleatorias creadas en el mapa, a la derecha posición detectada tras unos ciclos. 33
Figura 29: Partículas expandiéndose debido al ruido al no detectar posición. 33
Figura 30: Modelo robot diferencial. 34
Figura 31: A la izquierda láser de línea utilizado, a la derecha ejemplo de uso de esta técnica. 36

Figura 32: Esquema del sistema de medición láser-cámara. 37
Figura 33: Gráfica distancia-altura en píxeles con el láser a 0 grados. 38
Figura 34: Esquema del sistema de medición láser-cámara con el láser inclinado. 39
Figura 35: Gráfica distancia-altura en píxeles con el láser a 14.5 grados. 39
Figura 36: Imagen que representa 4 posiciones devueltas por 4 marcadores diferentes para una posición de robot. 40
Figura 37: Foto sin láser y con láser. 40
Figura 38: Comparación entre distintas condiciones de iluminación entre umbralizar según el método de Otsu y una umbralización fija. 41
Figura 39: Proceso de filtrado en la imagen umbralizada del láser, ejemplo 2. 42
Figura 40: Proceso de filtrado en la imagen umbralizada del láser, ejemplo 1. 42
Figura 41: Detalle línea dibujada en el centro del defecto. 42
Figura 42: Resultado final. 43
Figura 43: Modelo creado a partir de la medición láser. 44
Figura 44: Ejemplo de modelo creado a partir de la medición láser de dos objetos enfrentados a diferente distancia. 45
Figura 45: Ejemplo de modelo creado a partir de la medición láser de un objeto oblicuo. 45
Figura 46: Mapa de probabilidad de ocupación creado por el robot. 46
Figura 47: Mapa de probabilidad de ocupación creado por el robot en el que se aprecian medidas erróneas. 47
Figura 48: Mapa engrosado y ruta en rosa calculada por el planificador de trayectorias. 50
Figura 49: Diagrama de estados. 54
Figura 50: Diagrama de actividad thread "Main". 55
Figura 51: Diagrama de actividad Thread "ImagesProvider". 56
Figura 52: Diagrama de actividad thread "Location". 56
Figura 53: Diagrama de actividad thread "Motion". 57
Figura 54: Diagrama de actividad thread "SerialRead". 58
Figura 55: Diagrama de casos de uso de la aplicación. 59
Figura 56: Diagrama de secuencia de la aplicación. 60
Figura 57: Detalle diagrama de secuencia images provider. 61
Figura 58: Detalle diagrama de secuencia location. 61
Figura 59: Detalle diagrama de secuencia serial read. 62
Figura 60: Detalle diagrama de secuencia Motion. 62
Figura 61: Detalle diagrama de secuencia Main. 63
Figura 62: Robot desplazado 1000mm en "X" desde el punto de origen. 65
Figura 63: Robot desplazado 2000mm en "X" desde el punto de origen. 65
Figura 64: Robot desplazado 3500mm en "X" desde el punto de origen. 66
Figura 65: Robot desplazado 3500mm en "X" y 3000mm en "Y" desde el punto de origen. 66

Figura 66: Detalle detección, medidas marcadores y posición según filtro de partículas. 67
Figura 67: Dispersión alta de partículas y dispersión baja según ruido variable. 68
Figura 68: Partículas desplazadas 1000mm únicamente con la odometría. 69
Figura 69: Detección láser errores. 70
Figura 70: Detección láser errores. 70
Figura 71: Repercusión de medidas erróneas en la creación de mapas. 71
Figura 72: Medidas erróneas provocadas por la reflexión de la luz. 71
Figura 73: Ejemplo Mapa 1. 72
Figura 74: Ejemplo Mapa 1 mejorado por la aplicación. 72
Figura 75: Plano real y plano real tratado. 73
Figura 76: Plano obtenido y plano real superpuesto, detalle objetos. 73
Figura 77: Plano obtenido y plano real superpuesto. 74
Figura 78: Cambio de trayectoria (rosa) a los cuatro vértices de la zona explorada al no moverse el robot al menos un metro durante 2 minutos. 75
Figura 79: Detalle de sensores infrarrojos lateral y frontal. 75
Figura 80: Tiempos de control de los diferentes hilos. 77
Figura 81: Introducir número cámara marcadores. 79
Figura 82: Introducir número cámara láser. 79
Figura 83: Instrucciones de la aplicación mostradas al arrancar. 79
Figura 84: Mapa, en la esquina superior izquierda vemos en azul el modo de navegación actual. 80
Figura 85: Marcadores en azul en el mapa. 80
Figura 86: Tiempos de ejecución de las diferentes tareas en la consola. 81
Figura 87: Izquierda trayectoria planificada por el robot en rosa, derecha partículas en rojo en mapa. 81
Figura 88: Trayectoria seguida por el robot en verde. 81
Figura 89: A la izquierda medición actual del láser en rojo, a la derecha mapa tratado. . 82
Figura 90: Todas las visualizaciones. 82
Figura 91: Imágenes cámara marcadores, cámara frontal con láser, cámara frontal sin láser, detección láser. 83
Figura 92: Diagrama de Gantt 95
Figura 93: Estructura de descomposición del proyecto 96
Figura 94: Investigación inicial. 96
Figura 95: Requisitos. 97
Figura 96: Formación previa. 97
Figura 97: Selección Hardware. 98
Figura 98: Desarrollo Software. 98
Figura 99: Pruebas. 99

Índice de tablas

Tabla 1: Clasificación herramientas localización. 28
Tabla 2: Sensores más comunes. 36
Tabla 3: Mediciones y precio unitario. 100
Tabla 4: Presupuesto total. 101
Tabla 5: Tabla caso de uso "Guardar Mapa". 102
Tabla 6: Tabla caso de uso "Mostrar láser en mapa". 102
Tabla 7: Tabla caso de uso "Mostrar marcadores en mapa". 103
Tabla 8: Tabla caso de uso "Mostrar mapa mejorado". 103
Tabla 9: Tabla caso de uso "Mostrar partículas en mapa". 104
Tabla 10: Tabla caso de uso "Mostrar trayectoria seguida por el robot en mapa". 104
Tabla 11: Tabla caso de uso "Mostrar trayectoria planificada por el robot en mapa". . 105
Tabla 12: Tabla caso de uso "Reset programa". 105
Tabla 13: Tabla caso de uso "Establecer modo teleoperación". 106
Tabla 14: Tabla caso de uso "Establecer modo seguimiento de marcadores". 106
Tabla 15: Tabla caso de uso "Establecer modo de navegación aleatoria". 107
Tabla 16: Tabla caso de uso "Maximizar área explorada". 107
Tabla 17: Tabla caso de uso "Avanzar". 108
Tabla 18: Tabla caso de uso "Retroceder". 108
Tabla 19: Tabla caso de uso "Giro derecha". 109
Tabla 20: Tabla caso de uso "Giro izquierda". 109

INTRODUCCIÓN

1 Introducción

1.1 Marco del proyecto

Este proyecto ha sido desarrollado como proyecto final del **"Máster universitario en Ingeniería de Sistemas y de Control"** de la UNED, en el departamento de Informática y Automática de la Escuela Técnica Superior de Ingeniería Informática.

1.2 Motivación y objetivos

El objetivo principal de este proyecto es **implementar** la **navegación** de un **robot autónomo** utilizando **balizas** y **luz estructurada**.

Para cumplir con el objetivo final se han estimado necesarios los siguientes sub-objetivos:

- **Detectar** en las imágenes diferentes **balizas** conocidas, en función del tamaño de las mismas en las imágenes podremos determinar la distancia y posición a la que se encuentra el robot respecto a ellas.
- Adquirir y **tratar** las **imágenes** con el haz **láser**, para calcular la distancia a los objetos del entorno.
- **Localización** del **robot**, si la posición en el espacio de las balizas detectadas es conocida, conoceremos cual es la posición del robot.
- **Crear** un **mapa** del entorno en base a la posición dada por las balizas y las medidas de las imágenes con láser.
- **Navegación**, el **robot** deberá ser capaz de navegar con la información adquirida de las balizas y del láser.

1.3 Alcance del proyecto

Investigación inicial: Previamente incluso a la planificación de este proyecto, se ha realizado un estudio sobre el estado actual de robots autónomos, más concretamente de localización, mapping y navegación.

Formación: Antes de comenzar el proyecto ya se contaba con profundos conocimientos de programación en C++ y de visión artificial, pero nunca se había trabajado con el entorno Eclipse, ni con las librerías de visión OpenCV.

Selección *Hardware:* En esta fase fue necesario seleccionar las cámaras para adquirir las imágenes de los marcadores, el láser y los detectores de presencia, además fue necesario sustituir el PC y el láser instalados inicialmente.

Instalación *Hardware:* Instalación de las cámaras, láser y detectores de presencia en el robot.

Desarrollo *Software*: Incluye el diseño previo en UML, su posterior corrección, y el desarrollo de la aplicación en C++ junto a las librerías de visión artificial de OpenCV y de marcadores ArUco.

Pruebas*:* Pruebas de funcionamiento del robot en entornos controlados para la posterior corrección, depuración y posibles mejoras del software.

Redacción de la memoria del proyecto. Una vez conseguidos los hitos anteriores y tras haber realizado las pruebas, se analizan los resultados y se obtienen conclusiones a partir de los mismos.

1.4 Organización del documento

El documento está dividido en tres apartados; el primero sirve de **"Introducción"** al proyecto y al estado del arte; en el **"Desarrollo"** se justificarán las soluciones tomadas para cada uno de los problemas planteados, explicándolos y mostrando los resultados obtenidos; por último hay un apartado dedicado a las **"Conclusiones y líneas futuras"** que pretende aglutinar las conclusiones obtenidas por el autor tras la ejecución del proyecto, así como las posibles líneas de continuación del mismo. Además se ha añadido un apartado de **"Anexos"** donde se incluye información complementaria: la planificación con un diagrama de Gantt; la estructura de descomposición del proyecto; el presupuesto y las tablas de casos de uso.

2 Estado del arte

2.1 Antecedentes de Investigación

Un robot móvil es un robot que tiene la capacidad de moverse por su entorno sin estar fijado a una ubicación física, con cierto nivel de autonomía, y con la capacidad de percibir su entorno y reaccionar en consecuencia. Este campo de investigación, el de la robótica móvil, es relativamente reciente se inició hace unos 40 años siendo los primeros robots móviles destinados a investigación Stanford Cart [1] y Shakey [2], [3] en los años 70.

Stanford CART '73: fue desarrollado en la universidad de Stanford y empleado para labores de investigación. Su principal objetivo era estudiar la controlabilidad de un vehículo lunar desde la tierra. Posteriormente sería modificado para estudiar la navegación visual y la navegación autónoma siguiendo una línea.

Shakey: fue el primer robot móvil con la capacidad de percibir y razonar sobre su entorno. Tenía sensores sonar, cámara de tv, sensores de colisión, procesador a bordo y antena para comunicación radio. Surgió con el objetivo de la investigación en el mundo de la robótica y su desarrollo dio lugar a resultados que han tenido repercusiones de gran alcance en los campos de la robótica y la inteligencia artificial, así como la informática en general.

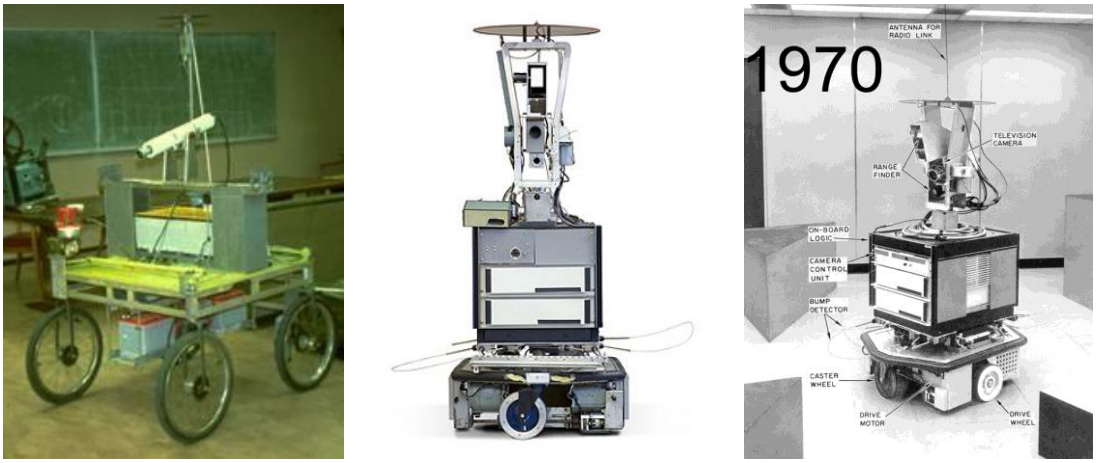

Figura 1: Stanford Cart y Shakey.

Estos primeros robots (Karl Iagnemma del MIT [4]), estaban preparados para operar en interiores, es decir en suelos planos con buena tracción, sin embargo hoy en día la investigación se centra en robots de alta movilidad que sean capaces de navegar en todo tipo de terrenos no estructurados, muchos de ellos con soluciones inspiradas en la

naturaleza. Un ejemplo de esto es el LS3 de Boston Dynamics [5] que podemos ver en la siguiente ilustración.

Figura 2: LS3 de Boston Dinamics.

En los primeros robots, al estar pensados para operar en interiores, les bastaba con diferenciar entre obstáculos y espacio vacío, sin embargo en la actualidad el objetivo es que los robots sean capaces de operar en nuestro medio. Esto obliga a los robots a operar en espacios no estructurados geométricamente que requieren diferenciar y clasificar el espacio que les rodea según sus características.

Un hito importante en la robótica móvil reciente, fue cuando Stanley [6] consiguió ser en 2005 el primer coche autónomo en completar el "Darpa Grand Challenge", hoy en día parte de aquel equipo trabaja en el "Google driverless car"

Figura 3: Stanley, primer ganador del "Darpa Grand Challenge".

Stanley disponía de diferentes sensores (Láseres, Cámara, Radar, GPS position, GPS compass, IMU...) integrados en una estructura de control como la siguiente que le hacía capaz de navegar en su entorno.

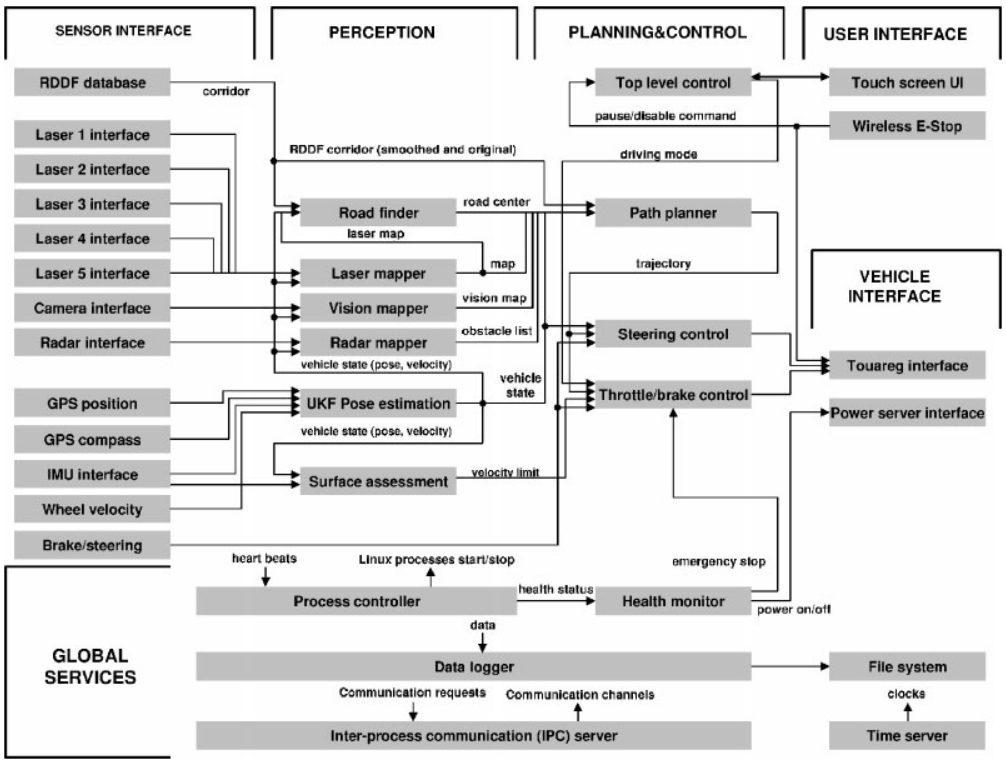

Figura 4: Estructura de control de Stanley.

Uno de los problemas más recurrentes en la robótica móvil desde sus inicios es el problema de localización y mapping conocido por las siglas SLAM, "Simultaneous Location And Mapping", es el problema de construir un mapa de un entorno desconocido a la vez que el robot se localiza en él. Existen diferentes soluciones para este problema siendo dos de las soluciones más extendidas los filtros de partículas o el filtro extendido de Kalman.

Este proyecto se ha enfocado hacia la visión por computador, campo que está en auge en diversas líneas de investigación relacionadas con robots móviles, debido principalmente a la gran cantidad de información que se puede extraer de ella a un coste bajo [7].

Los sistemas de navegación o SLAM centrados en visión artificial, están basados en extraer características de las imágenes obtenidas del entorno y posteriormente encontrarlas en los fotogramas siguientes, de manera que podamos estimar la diferencia de posición, orientación o velocidad [8], [9]. Diferentes líneas de investigación se centran en SLAM basado completamente en visión artificial, teniendo como inconveniente debido a la complejidad de esta metodología, procesar tanta información en tiempo real.

El campo de los robots móviles es un campo multidisciplinar que está todavía en sus comienzos, en el que intervienen diversas materias de la ciencia y computación, donde

los temas tratados en este proyecto localización y mapping han sido y siguen siendo líneas de estudio recurrentes.

2.2 Necesidades

En la actualidad el GI3 "Grupo de Investigación en Informática Industrial" [10] del departamento de Informática y Automática de la UNED, está centrado entre otras líneas de investigación en los robots móviles [11], [12]. En dicho grupo se utiliza entre otros un robot autónomo de construcción propia cuyo objetivo es ampliar la oferta de prácticas disponibles en el máster. Ante la necesidad e inquietud de mejorarlo y dotarlo con nuevas funcionalidades se consideró oportuno equiparlo con un sistema de navegación basado en marcadores y un sistema de medición basado luz estructurada.

DESARROLLO

3 Robot – Estructura de control

Este apartado se centra en mostrar las características y explicar la estructura de control del robot utilizado.

Como se ha mencionado anteriormente el robot perteneciente al departamento de Informática y Automática de la UNED, consta de 4 ruedas de tipo "Skid steer", esto quiere decir que tiene 4 ruedas de tracción sin dirección. En este tipo de robots los cambios de orientación se consiguen por la velocidad diferencial entre las ruedas de ambos lados.

Dos de las cuatro ruedas (una de cada lado) llevan incorporadas un encoder incremental de 6000 pulsos por vuelta, para conocer cuánto gira cada rueda.

Una placa "Arduino" junto a un "Arduino Motor Shield" se encargaran de gobernar los cuatro motores, el encendido/apagado del láser, la lectura de los encoders y los sensores de presencia.

Figura 5: Robot.

Figura 6: Placa Arduino Uno R3 y Arduino Motor Shield R3.

Todo esto es alimentado por una batería de plomo que proporciona una autonomía de alrededor de media hora al robot.

Para la parte desarrollada en este proyecto el robot lleva a bordo un PC con procesador de dos núcleos, a este PC han sido conectadas dos webcams para las tareas de localización y detección láser, ambas webcam disponen de iluminación integrada. En este caso el PC se trata de un portátil de reducido tamaño, que será el encargado de la toma de imágenes, localización, detección láser, mapping y navegación.

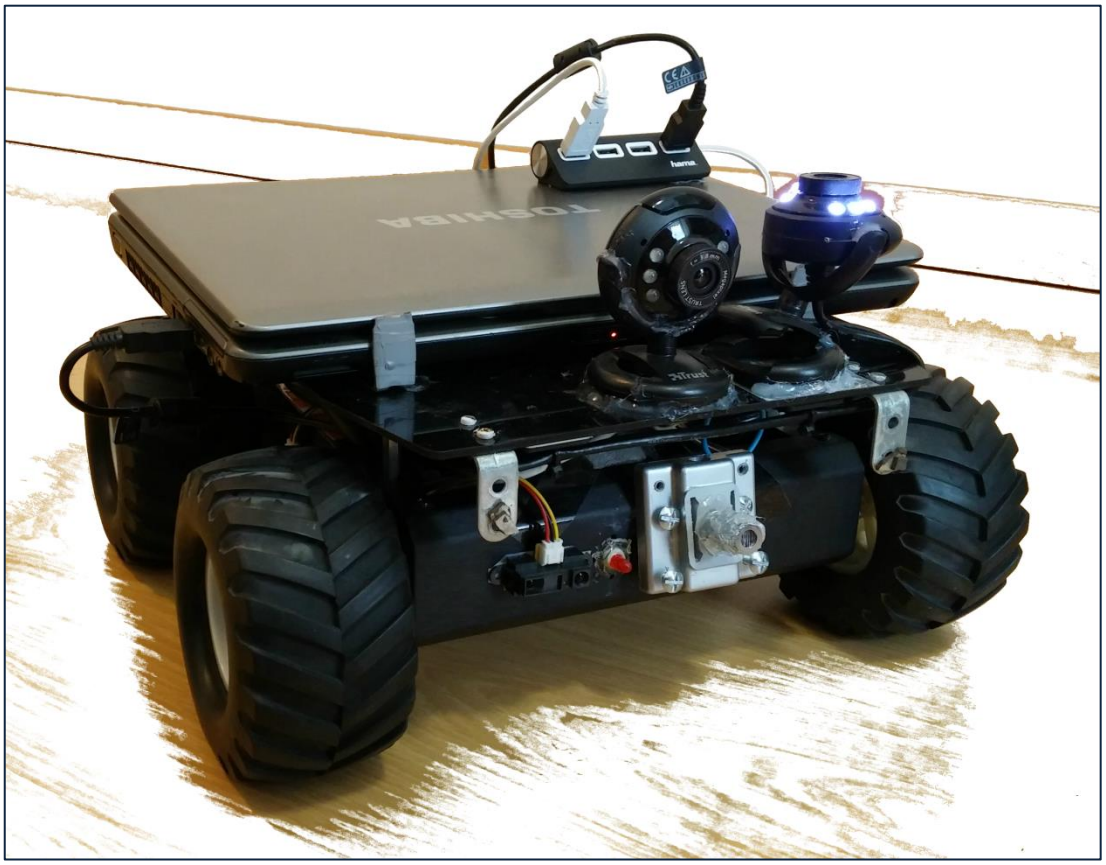

Figura 7: Robot con portátil cámaras y láser incorporado.

Inicialmente el portátil que llevaba a bordo era ligeramente más pequeño, pero para poder trabajar con ambas webcam a la resolución necesaria hubo que sustituir el PC ya que el inicial disponía de varios puertos USB pero todos estaban conectados internamente al mismo bus siendo el ancho de banda insuficiente para la aplicación.

El láser también fue necesario remplazarlo debido a que el haz del inicial no se apreciaba de manera nítida en las imágenes siendo bastante pobre su detección.

En un futuro lo idóneo sería poner un mini pc, de manera que el conjunto quedase más integrado, pero para el desarrollo de este proyecto es indiferente.

El PC se comunicará con el control del robot (Arduino) por comunicación serie, a través del puerto USB al igual que con las webcams, asimismo el robot podrá ser monitorizado o manipulado desde un pc externo con un acceso remoto a través de una red Wifi.

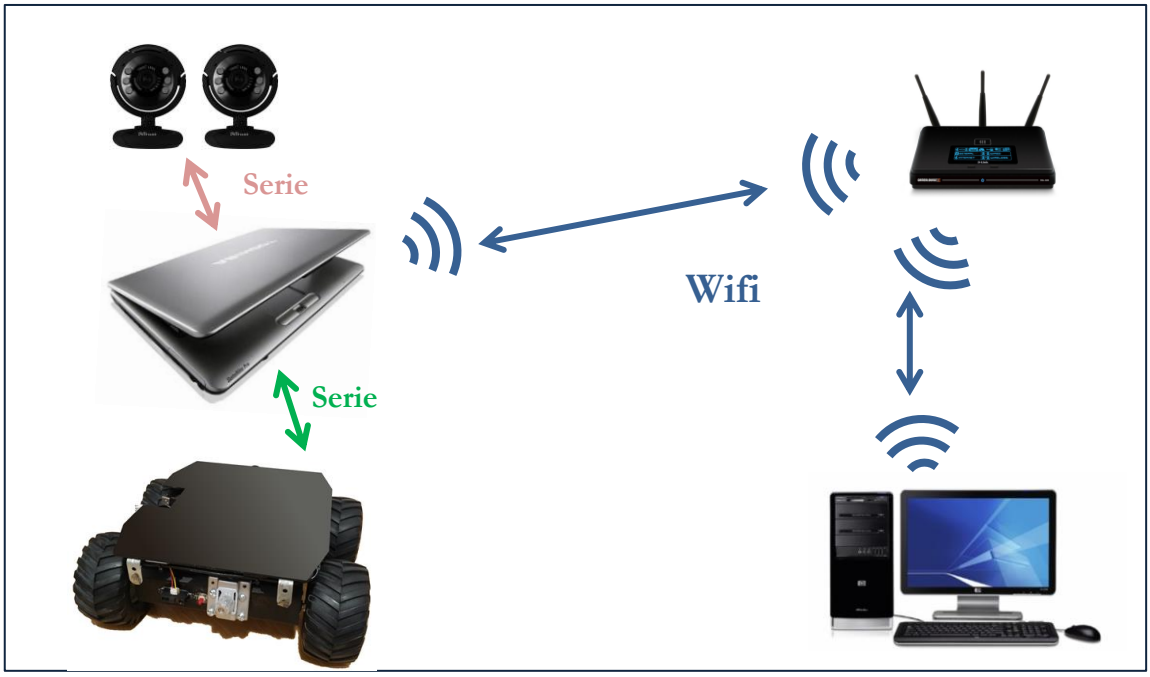

Figura 8: Esquema de comunicaciones.

Como hemos comentado anteriormente en el pc se ejecutarán las tareas de localización, mapping, navegación y comunicación, que contienen las siguientes sub-tareas:

- Localización:
  - Adquisición de imágenes de marcadores en el techo.
  - Detección y gestión de los marcadores.
  - Integración de la odometría y filtro de partículas.
- Mapping:
  - Adquisición de imágenes frontales al robot.
  - Detección láser/cálculo de distancia, cálculo de mapa de probabilidades.
- Navegación:
  - Planificación y ejecución de la trayectoria en función de la localización, el mapa y la odometría.
- Comunicación serie Arduino:
  - Enviar/recibir comandos del robot. Consignas de motores, accionamiento del láser, pulsos de encoder entre ciclos e información de los detectores de distancia.
- Visualización: Interfaz para el usuario.

Representado en una estructura de capas sería:

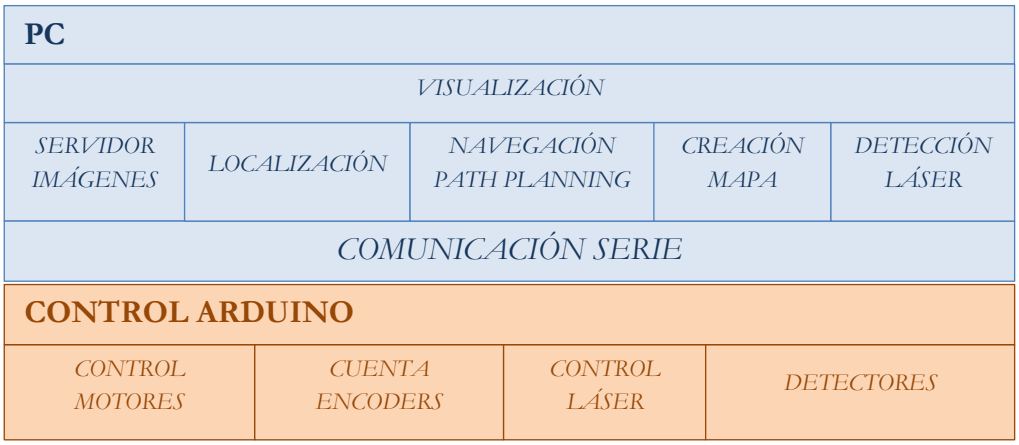

Figura 9: Estructura de capas.

4 Localización

El sistema de localización del robot en el entorno queda descrito en este apartado, habiéndose conseguido a través de la detección de la posición relativa a marcadores visuales y de la odometría del mismo.

4.1 Localización por visión

En diferentes líneas de investigación en las que la localización de robots se basa en sistemas de visión artificial, se buscan puntos de interés o características en las imágenes del entorno que sirvan al robot para localizarse, como en [13] donde se buscan objetos planos cuadrangulares para esta tarea.

Otra línea común de investigación consiste en buscar y detectar marcadores previamente conocidos en las imágenes, si estos no son parte del entorno, es decir han sido generados y colocados para la localización del robot serán marcadores visuales artificiales. Esta línea es la que se ha tomado en este trabajo

En el artículo [14] vemos un caso como el citado, en el que es diseñado el marcador para su posterior detección. En esta aplicación la idea no es diseñar un marcador artificial para luego detectarlo, sino que se han estudiado diferentes posibilidades entre marcadores ya existentes para integrar alguno de ellos en la aplicación.

Inicialmente se barajaron opciones como marcadores basados en color, en códigos QR o incluso códigos de barras, como punto de partida se determinó que estos debían reunir las siguientes condiciones:

- Robusto: Deben ser elementos que el robot pueda localizar sin errores independientemente de las características del entorno.
- Codificable: El marcador debe permitir tantas codificaciones como posiciones queramos registrar.
- Detección sencilla: Una detección compleja o con muchos condicionantes provocará errores en el momento que alguna de las variables cambien.

Inicialmente se hicieron pruebas con marcadores basados en color, como los siguientes:

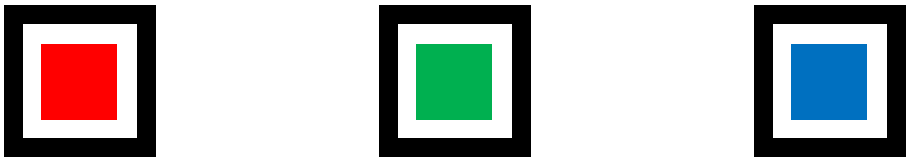

Figura 10: Marcadores basados en color.

En la siguiente imagen los vemos detectados por la aplicación.

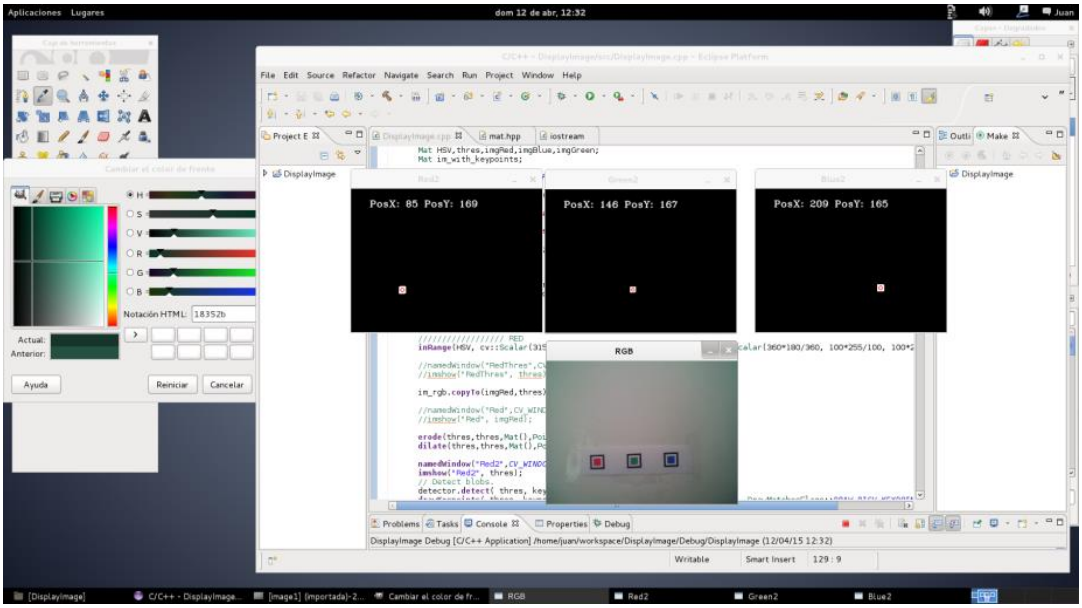

Figura 11: Detección marcadores basados en color.

La gran ventaja de utilizar marcadores basados en colores es la sencillez en su implementación, idealmente la codificación basada en colores permite configuraciones elevadas (en una paleta de colores estándar de 255 colores por canal existen 2^{24} combinaciones) pero en el caso práctico, es muy difícil detectar colores similares en condiciones de iluminación cambiantes, es por esto que no podemos asegurar la robustez en la detección conforme aumentamos las codificaciones. Esta primera opción es por tanto no válida.

Como segunda opción se probaron los marcadores basados en código QR, con respecto a los marcadores basado en color ofrecen ventajas como son:

- Al trabajar blanco y negro la tonalidad de la iluminación no afecta a la detección.
- Número de codificaciones muy elevado. (Hasta 7089 caracteres por marcador si son numéricos).

Figura 12: Pruebas con marcadores basados en códigos QR.

El mayor inconveniente encontrado al utilizar este tipo de marcadores fue la dificultad en la detección correcta, al tener capacidad de contener mucha cantidad de información se trata de un marcador complejo, resultando en que para poder leer el código de forma satisfactoria el marcador debía estar bien orientado y a su vez estar cerca de la cámara o tener un tamaño muy grande.

Por último se estudió la opción de utilizar marcadores de realidad aumentada, estos marcadores surgen de la necesidad de tener información del entorno, y pueden aplicarse a cualquier situación en la que necesitemos obtener información de posición y orientación de un objeto respecto a una cámara. Existen casos en los que estos marcadores han sido aplicados con éxito en el campo de la robótica móvil, por ejemplo para evitar obstáculos o para navegación de robots móviles [15], [16].

Estos marcadores tienen las siguientes ventajas:

- Blanco y negro. Evita errores por color, minimiza errores iluminación.
- Codificación de hasta 1024 marcadores. (En la opción utilizada)
- Código desarrollado para fácil implementación.
- Generación de marcadores: Son creados de manera sencilla e impresos en papel.

Existen diferentes librerías para implementar de manera sencilla la detección de estos marcadores, dos ejemplos de librerías con funcionalidades similares son: ARToolKit SDKs [17] y ArUco [18].

Como comentábamos antes, estos marcadores son llamados marcadores artificiales, y en este caso concretamente son marcadores artificiales pasivos.

En este tipo de marcadores la detección dependerá de las condiciones de luz de la imagen. Para minimizar esto hay líneas de trabajo en los que se ha utilizado marcadores que reflejan la luz infrarroja [19], [20]. Esto mejora la detección en condiciones pobres de luz pero nos obliga a tener que adquirir tanto los marcadores como el equipo de adquisición acondicionado para trabajar con infrarrojos.

Tras el estudio se decidió utilizar la librería de marcadores ArUco, por la sencillez en la implementación y por no ser necesario equipos ni marcadores especiales. Además para favorecer el trabajo con condiciones de baja iluminación, se ha elegido una webcam con iluminación frontal, que favorece o amplía la capacidad de detección en condiciones de baja iluminación.

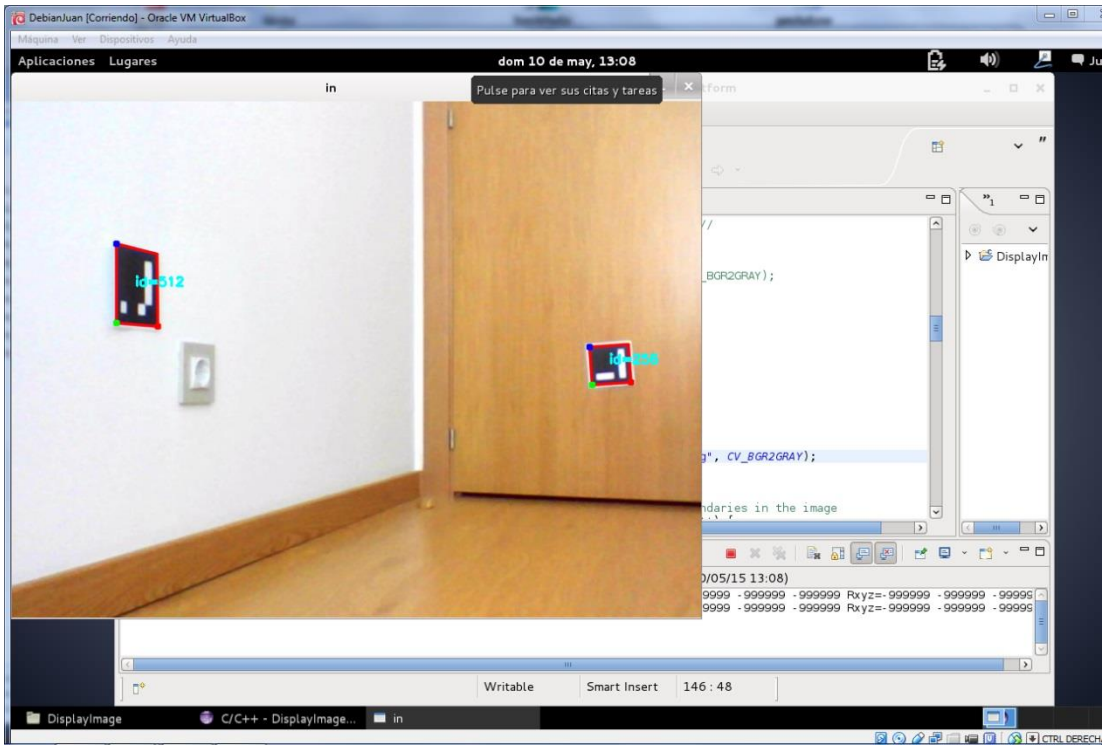

Figura 13: Primeras pruebas con marcadores de realidad aumentada ArUco.

Durante el proceso de implementación de las librerías, para poder obtener la posición y orientación de los marcadores es necesario calibrar la cámara, con el fin de modelar los parámetros intrínsecos de la cámara y corregir las distorsiones.

Figura 14: Efectos de deformación, ejemplo de calibración.

Una vez finalizado este proceso obtendremos de cada marcador su ID, un vector de posición y un vector de traslación con respecto a la cámara.

Con respecto a la posición de los marcadores, tras diferentes pruebas se optó por posicionarlos en el techo, al considerarse que favorecía la detección de los mismos al ser un robot que navega por el suelo.

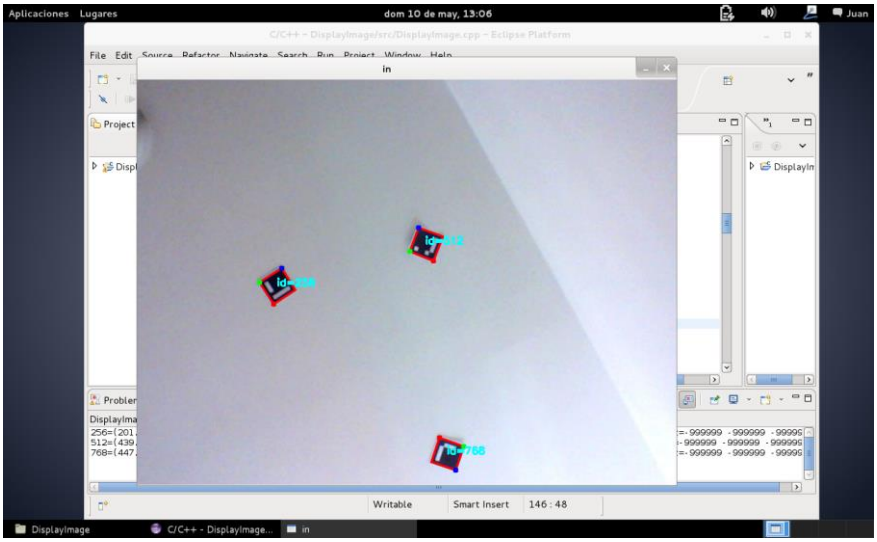

Figura 15: Marcadores en techo

4.2 Cálculo de la posición

Para conocer la posición del robot debemos ser capaces de localizar y calcular la posición de los marcadores con respecto a un origen de coordenadas global.

De cada marcador detectado con la aplicación de visión artificial recibiremos un vector de traslación y otro de rotación, que nos indican la traslación y rotación que debemos aplicar al origen de coordenadas de la cámara para llegar hasta el sistema de coordenadas del marcador.

Estos vectores de traslación y rotación son convertidos a matrices de transformación homogénea, formato en el que trabajaremos.

Como bien es sabido [21] las matrices de transformación homogénea permiten representar conjuntamente la posición y la orientación, además de facilitarnos su manipulación mediante el álgebra lineal.

Las matrices de transformación homogénea tendrán el siguiente formato:

$$T = \begin{bmatrix} R_{3x3} & p_{3x1} \\ f_{1x3} & w_{1x1} \end{bmatrix} = \begin{bmatrix} Rotación & Traslación \\ Perspectiva & Escalado \end{bmatrix}$$

En nuestro caso consideraremos la perspectiva 0 y el escalado 1, por lo que tendremos:

$$T = \begin{bmatrix} R_{3x3} & P_{3x1} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} Rotación & Traslación \\ 0 & 1 \end{bmatrix}$$

La manera simplificada de operar del robot para la localización es la siguiente:

- Cuando el robot vea el primer marcador establecerá su posición y orientación actual como origen del sistema de coordenadas global (Mundo). Almacenará la posición de este primer marcador con respecto al mundo, de manera que la siguiente vez que vea dicho marcador pueda determinar su posición relativa respecto al origen.

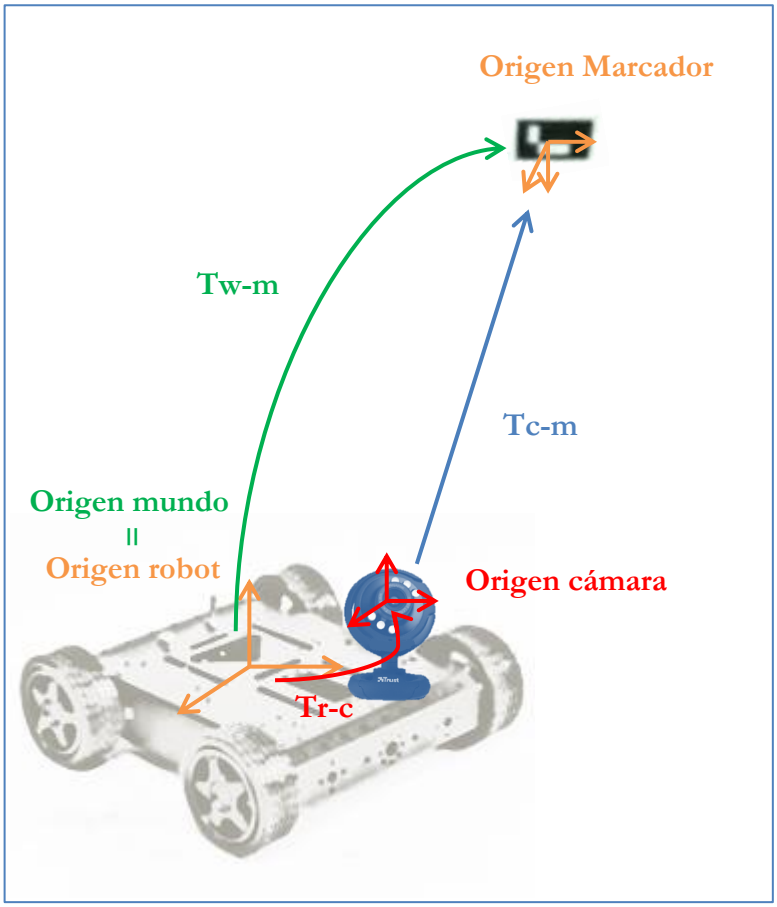

Figura 16: Detección primer marcador.

En la imagen anterior tenemos:

- Origen del robot: es la posición del robot donde se encuentra el origen del sistema de coordenadas del mismo.
- Origen del mundo: es el origen del sistema de coordenadas del mundo, y se hace coincidir con la posición del robot al detectar el primer marcador.
- Origen cámara: es sistema de coordenadas de referencia para la cámara.
- Origen marcador: es sistema de coordenadas de referencia para el marcador.
- T_{r-c}: es la transformación a aplicar al sistema de coordenadas del robot para llegar al sistema de coordenadas de la cámara. (Este valor es constante y depende de la posición de la cámara en el robot).

- T_{c-m}: es la transformación a aplicar al sistema de coordenadas de la cámara para llegar hasta el sistema de coordenadas del marcador. (Este valor viene dado por los valores obtenidos en la detección del marcador, es decir, cómo ve la cámara al marcador en la imagen).
- T_{w-m}: es la transformación a aplicar al origen del mundo para llegar al marcador. (Es decir la posición del marcador en el mundo). Este valor será en este caso (1er marcador detectado):

$$T_{w-m} = T_{r-c} \cdot T_{c-m}$$

- Este valor se almacenará en la lista de marcadores conocidos, esto quiere decir que la próxima vez que sea detectado permitirá calcular la posición del robot.

En la siguiente imagen vemos el caso en el que el robot se ha desplazado y sigue detectando el mismo marcador, este caso nos servirá para entender el proceso de detección de posición.

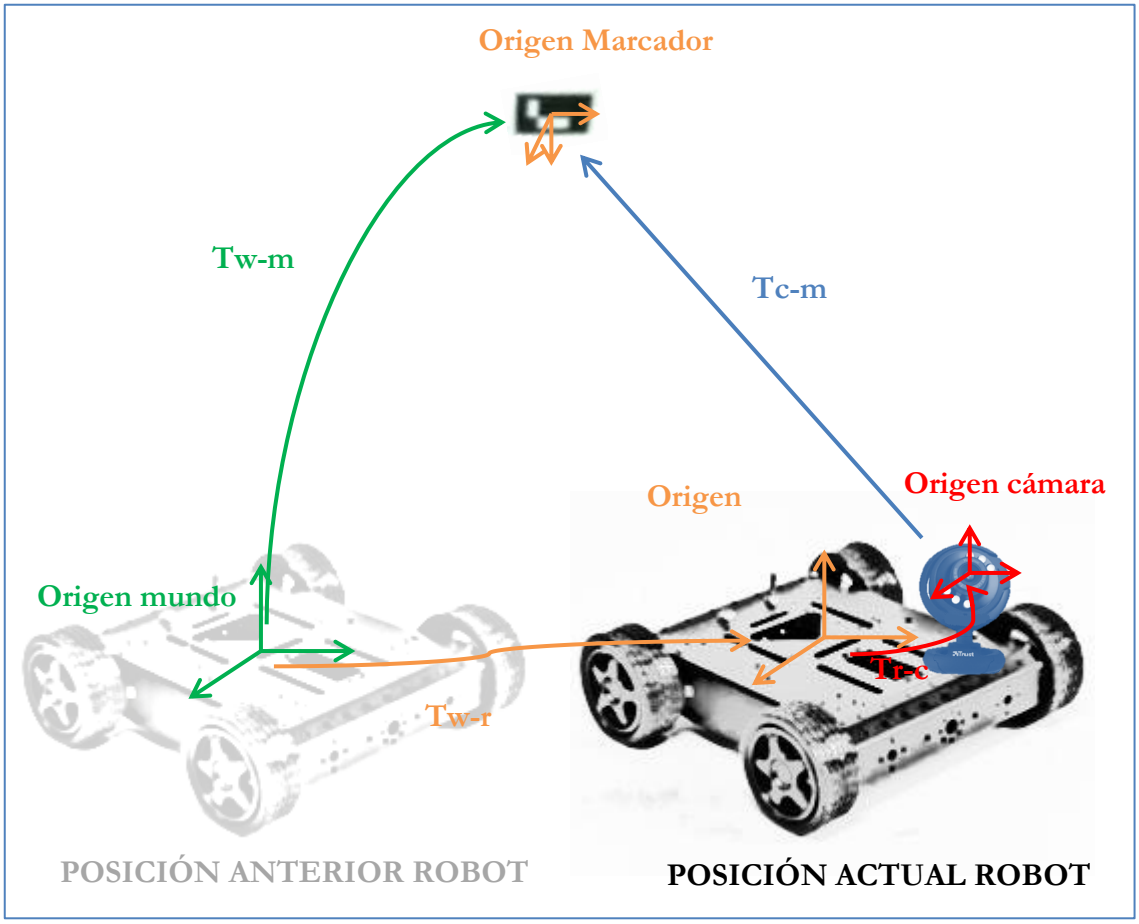

Figura 17: Robot se desplaza y sigue detectando el primer marcador.

De la imagen anterior concluimos que el robot conoce:

- T_{w-m}: Posición del marcador con respecto al mundo. (Lo almacenó en su primera detección).

- T_{r-c}: Es constante.
- T_{c-m}: Esta matriz determina cual es la transformación entre el marcador y la cámara en la posición actual.
- T_{w-r}: Esta matriz determina cual es la transformación entre el sistema de coordenadas del mundo y del robot. Calculando esta transformación sabremos cual es la posición del robot. Conocidos los valores es fácil determinar:

$$T_{w-r} = T_{w-m} \cdot T_{c-m}^{-1} \cdot T_{r-c}^{-1}$$

En la siguiente imagen vemos el caso en que el robot detecta un marcador nuevo junto a uno conocido.

Cada vez que detecte un marcador no conocido en una imagen que incluya a uno conocido se establecerá la transformación entre ambos lo que nos permitirá saber su posición con respecto al mundo e incluir dicho marcador en la lista de marcadores conocidos.

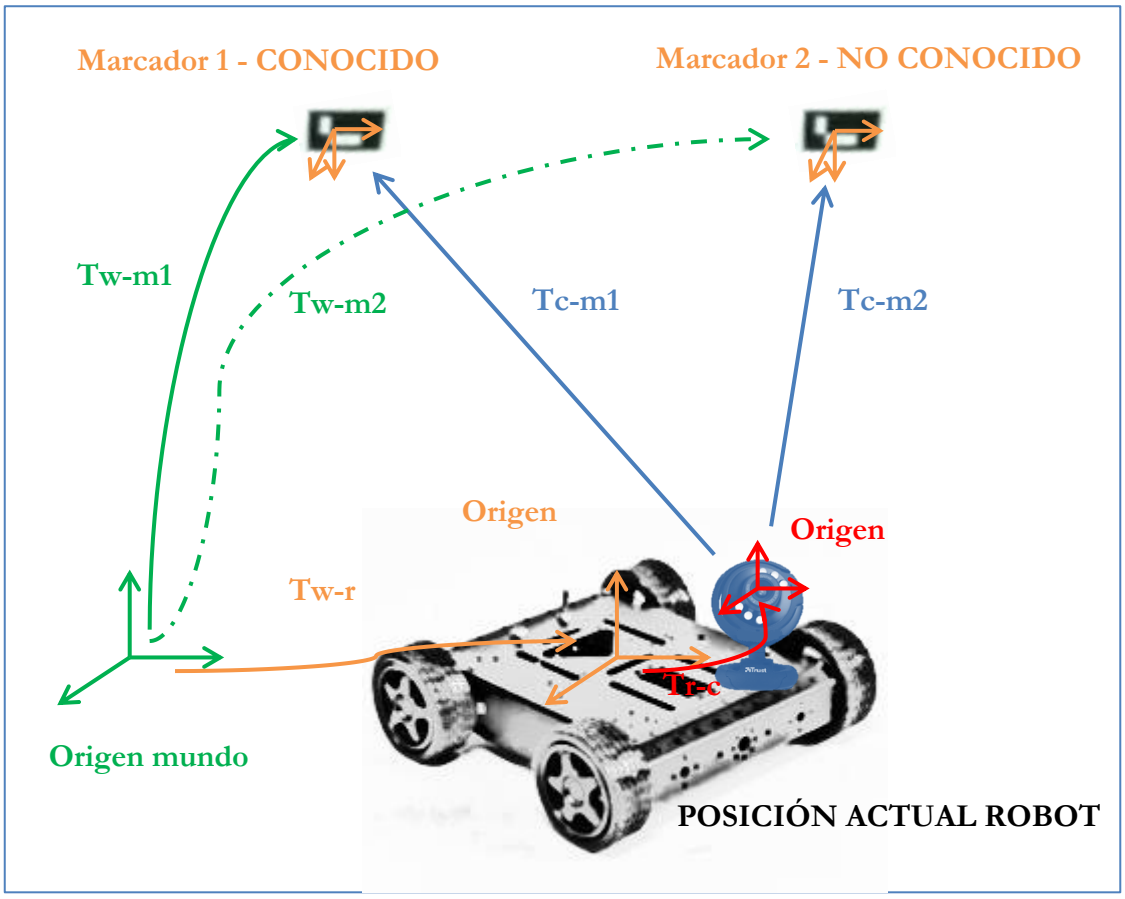

Figura 18: Robot detecta nuevo marcador.

En este caso el robot determinará su posición de la misma manera que lo hizo en el caso anterior.

$$T_{w-r} = T_{w-m_1} \cdot T_{c-m_1}^{-1} \cdot T_{r-c}^{-1}$$

El robot almacenará en la lista de posición de marcadores conocidos la posición del marcador nuevo. Esta posición la obtendrá a partir de la posición obtenida con el marcador anterior aplicando el siguiente cálculo:

$$T_{w-m_2} = T_{w-m_1} \cdot T_{c-m_1}^{-1} \cdot T_{c-m_2}$$

Resumiendo lo anterior:

- El primer marcador establece el origen del mundo.
- Para incorporar un marcador no conocido a la lista debe detectarlo en la imagen junto a uno conocido.
- Todos los marcadores de la lista de marcadores conocidos nos dan la posición del robot en el mundo.

Esta forma de trabajar en la que la posición de los marcadores es obtenida en función de otros marcadores, provocará que los errores en la medida se vayan acumulando, de manera que conforme aumente el número de marcadores en la cadena de reconocimiento, la precisión disminuirá. Esto afectará a la precisión sobre la medida real del robot y por tanto provocara desviaciones en los mapas con respecto a la realidad.

Otra opción hubiese sido conocer la posición absoluta de cada uno de los marcadores de antemano, pero el objetivo de la aplicación no es buscar la máxima precisión en posicionamiento, ni siquiera una precisión de centímetros, lo realmente importante es dotar al robot de un mecanismo de localización sencillo y versátil en su entorno, es decir la capacidad de construir un mapa topológicamente correcto. Con esta solución permitimos al usuario colocar los marcadores de una manera aleatoria, facilitando la sencillez de uso del sistema en su conjunto, manteniendo la línea de este proyecto.

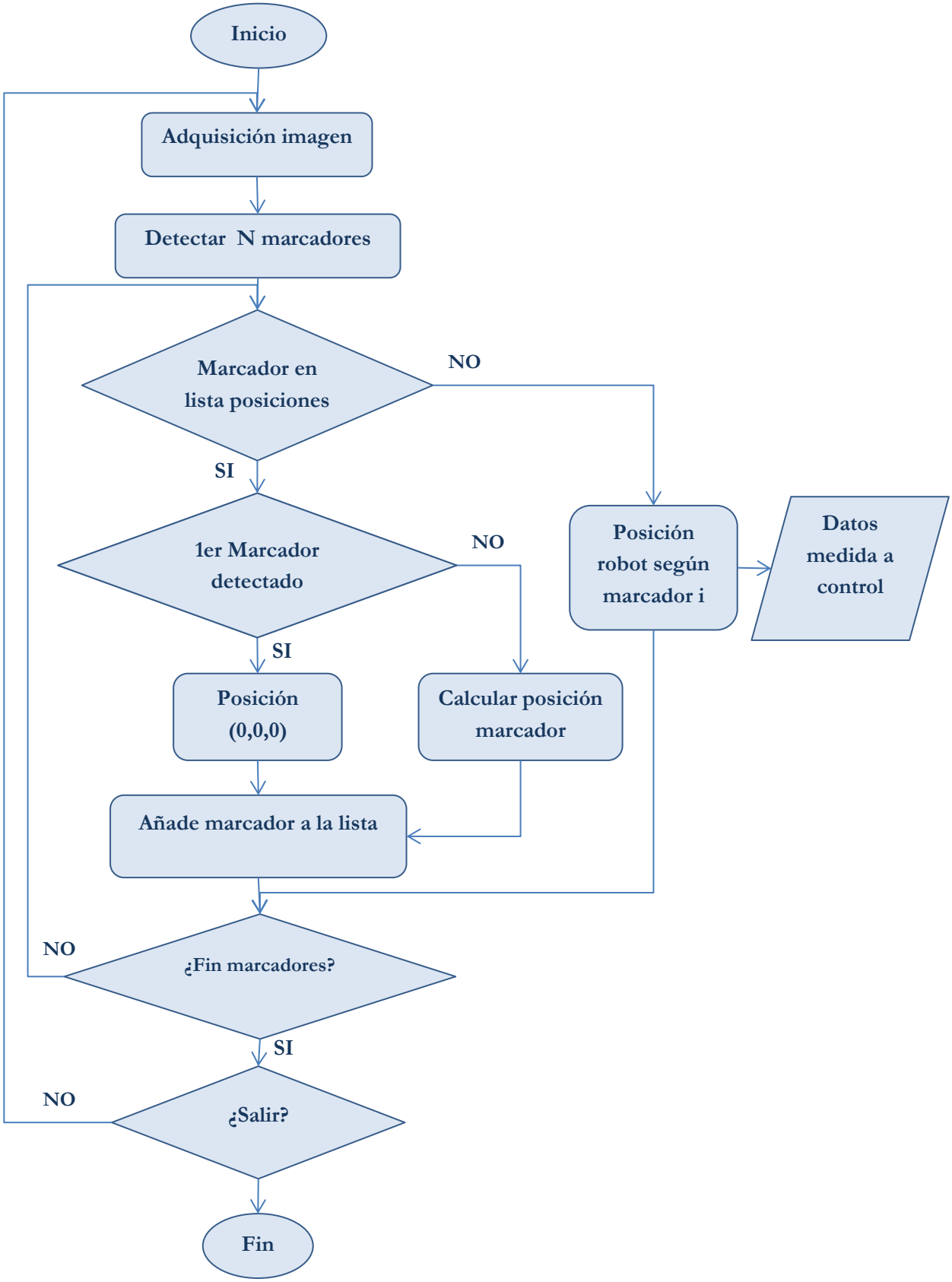

Figura 19: Diagrama de flujo detección marcadores.

4.3 Aplicación práctica

Al implementar lo explicado anteriormente y llevarlo al caso práctico se consideró necesario implementar lo siguiente para conseguir un resultado óptimo.

Con el objetivo de que las medidas sobre la posición del robot sean lo más estables posibles y evitar que errores puntuales en la detección provoquen errores sobre la posición del robot, durante el proceso de detección de marcadores se almacenará la siguiente información de los marcadores que hayan sido detectados al menos una vez.

- Desviación media en las detecciones del marcador.
- Desviación acumulada en los últimos 10 ciclos.
- Posición media en la detección del marcador.
- Estado del marcador: Cada marcador en función de los valores anteriores podrá tener 3 estados:
    - Posición no detectada: Cuando un marcador es detectado por primera vez, antes de incorporarse su posición a la lista de marcadores. Deberá haber sido detectado al menos durante 10 ciclos asegurando una desviación mínima entre sus detecciones. De esta manera evitaremos calcular la posición del marcador en base a una detección errónea.
    - Posición detectada pero no válida: Cuando un marcador es conocido pero la desviación media no está dentro del rango admisible.
    - Posición detectada y válida: Cuando el marcador es conocido y la desviación media de sus últimas medidas está dentro del rango admisible.

Estos estados tendrán las siguientes consecuencias:

- Posición no detectada: Deberá de cumplir una serie de ciclos correctos antes de incorporarse al sistema.
- Posición detectada pero no válida: Éste estado es temporal e indica que el marcador es conocido pero debido a la desviación no nos dará información de la posición del robot.
- Posición detectada y válida: Este marcador al ser válido nos devolverá la posición del robot.

Estos estados se representan en la imagen obtenida del sistema de visión con los siguientes colores:

- Posición no detectada: ROJO.
- Posición detectada pero no válida: AZUL.
- Posición detectada y válida: VERDE.

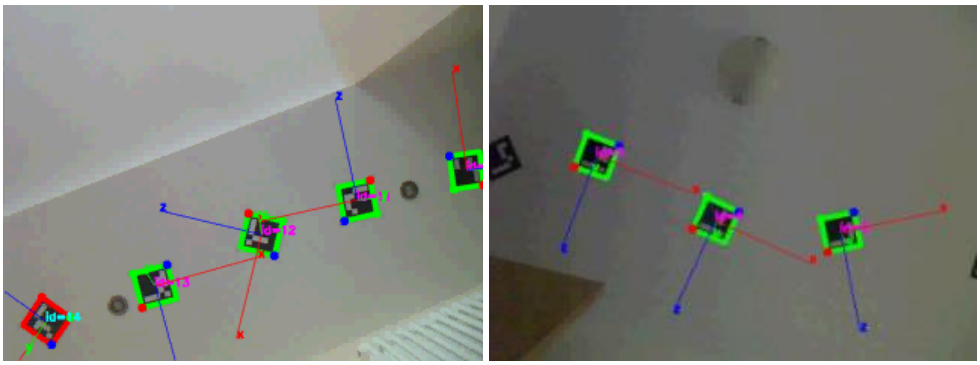

Figura 20: Estados marcadores.

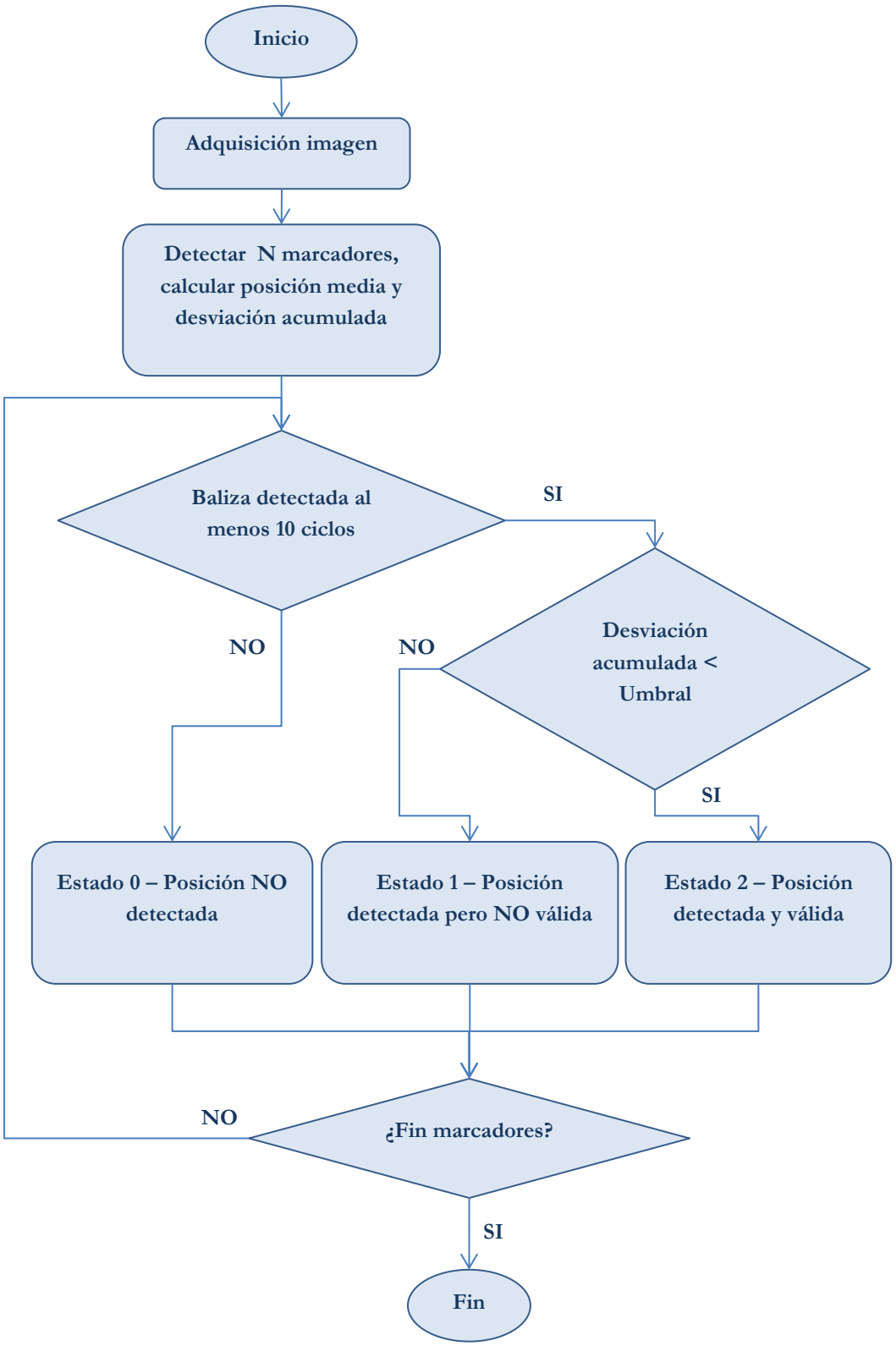

Figura 21: Evolución estados marcadores.

Además, tras diferentes pruebas se observó que la detección de la posición de los marcadores en cuanto a posición es mucho más precisa que las orientaciones, en particular, la orientación del eje perpendicular a la baliza es el que tiene más error. Este efecto se maximiza conforme el origen del marcador es más coincidente con el origen de la cámara.

En las siguientes imágenes podemos ver la detección de un marcador y un testigo completamente perpendicular al centro de la baliza, el cual ha sido utilizado como referencia para el estudio.

Podemos observar también como dependiendo del ángulo desde donde es obtenida la foto, el sistema de coordenadas varía llegando a no ser coincidente con el testigo.

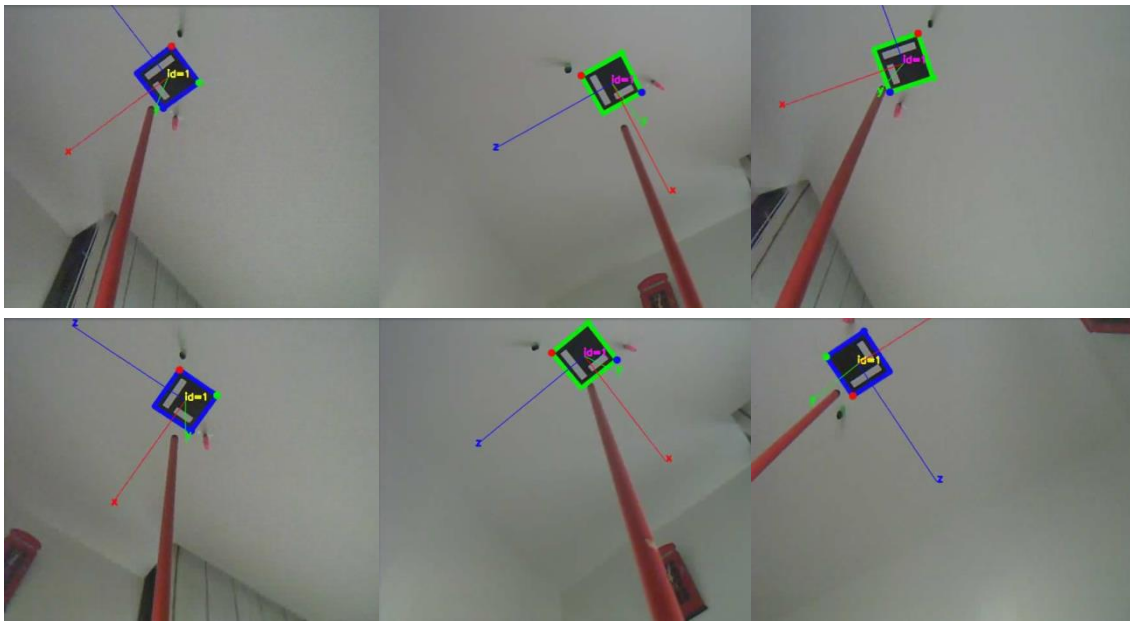

Figura 22: Secuencia de imágenes en las que se observa la variabilidad en las rotaciones del sistema de coordenadas detectado del marcador en función de la posición con la que se ha obtenido la foto.

Estos errores/diferencias en la medida de un marcador dependiendo del punto donde es visto provocan desviaciones en la medida.

Para evitar este problema se procedió a la siguiente simplificación: como los marcadores siempre están en un plano paralelo al de la cámara del robot (Marcadores en techo, robot en el suelo), se decidió anular a efectos de cálculo de la matriz de transformación las rotaciones respecto al eje X e Y, o lo que es lo mismo trabajar con la traslación dada por la matriz de transformación y solo con la rotación entorno al eje Z.

Con esta simplificación conseguimos una medida mucho más robusta de la posición actual del robot.

Es importante por tanto ser conscientes de que esta medida fuerza a que los marcadores estén siempre en el techo, un marcador en la pared devolvería un resultado erróneo al no ser aplicada su transformación de manera completa.

4.4 Filtro de partículas

Cada marcador detectado dentro de la imagen nos dará una posición del robot según el mismo. Como es evidente, estas medidas no serán exactamente iguales debido a errores en la detección, a errores acumulados en la posición de los marcadores o incluso por la fluctuación en la medida.

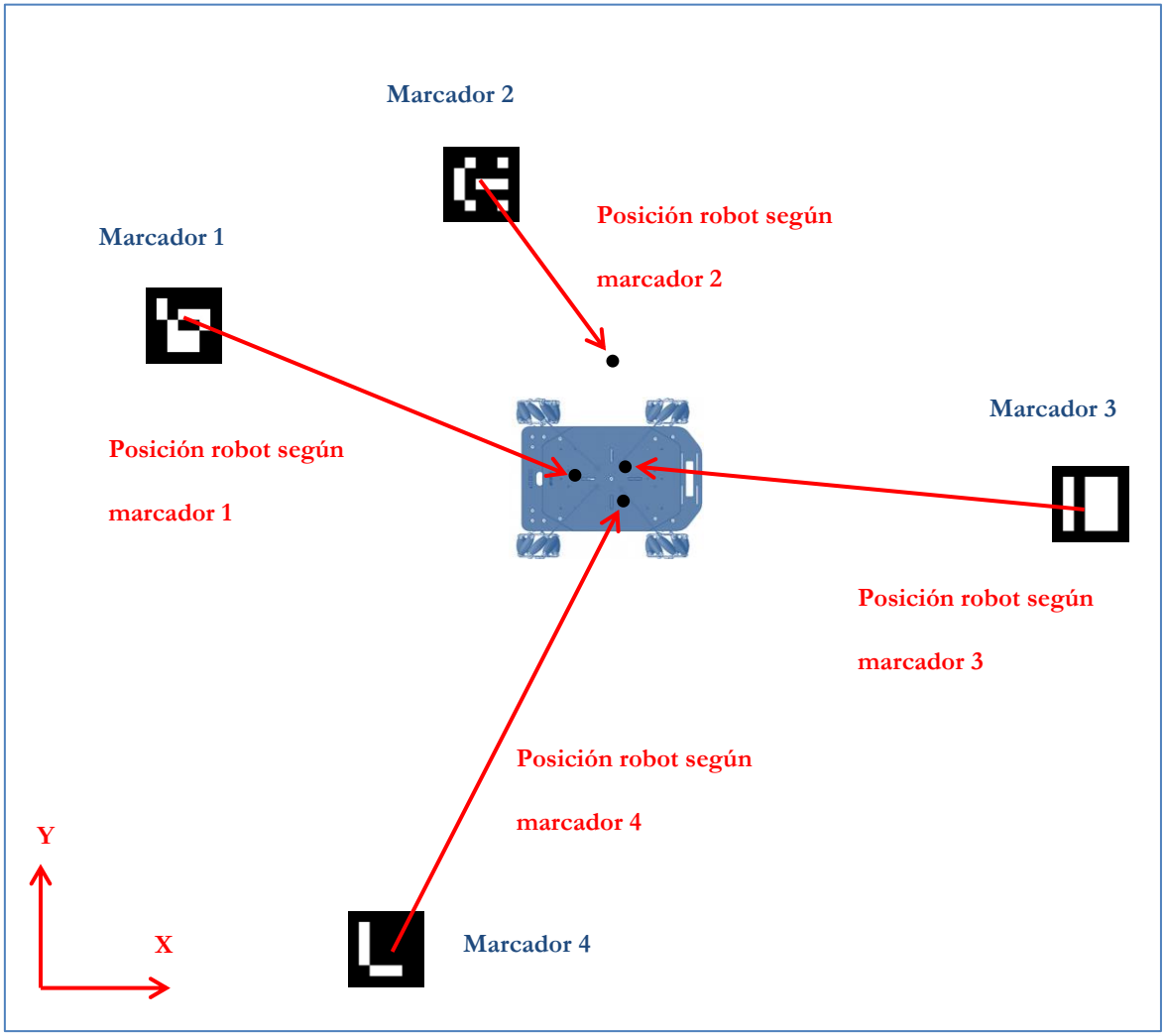

Figura 23: Imagen que representa 4 posiciones devueltas por 4 marcadores diferentes para una posición de robot.

Tendremos tantas posiciones diferentes como marcadores detectados dentro de la imagen, a partir de las cuales determinaremos la posición estimada del robot.

Existen diferentes opciones, tal vez la solución más sencilla fuese determinar la posición en base a un promediado de las medidas o tomar la medida de la baliza que tiene menos

variación. Pero para esto existen herramientas como mapas de probabilidad de ocupación basados en la teoría bayesiana, la utilización del filtro de Kalman o filtros de partículas.

Según Sebastian Thrun [22], podemos clasificar estas herramientas de la siguiente manera:

	State space	Belief	Efficiency	In robotics
Histogram Filters	Discrete	Multimodal	Exponential	Approximante
Kalman Filters	Continuous	Unimodal	Cuadratic	Approximante
Particle Filters	Continuous	Multimodal	Depends	Approximante

Tabla 1: Clasificación herramientas localización.

En esta aplicación se ha optado por el uso de filtros de partículas [23], [24] debido a su eficiencia y su sencilla implementación.

El filtro de partículas partirá de una serie de partículas generadas de manera aleatoria en plano X-Y del robot, y con las medidas de los marcadores determinará la probabilidad de cada una de las partículas.

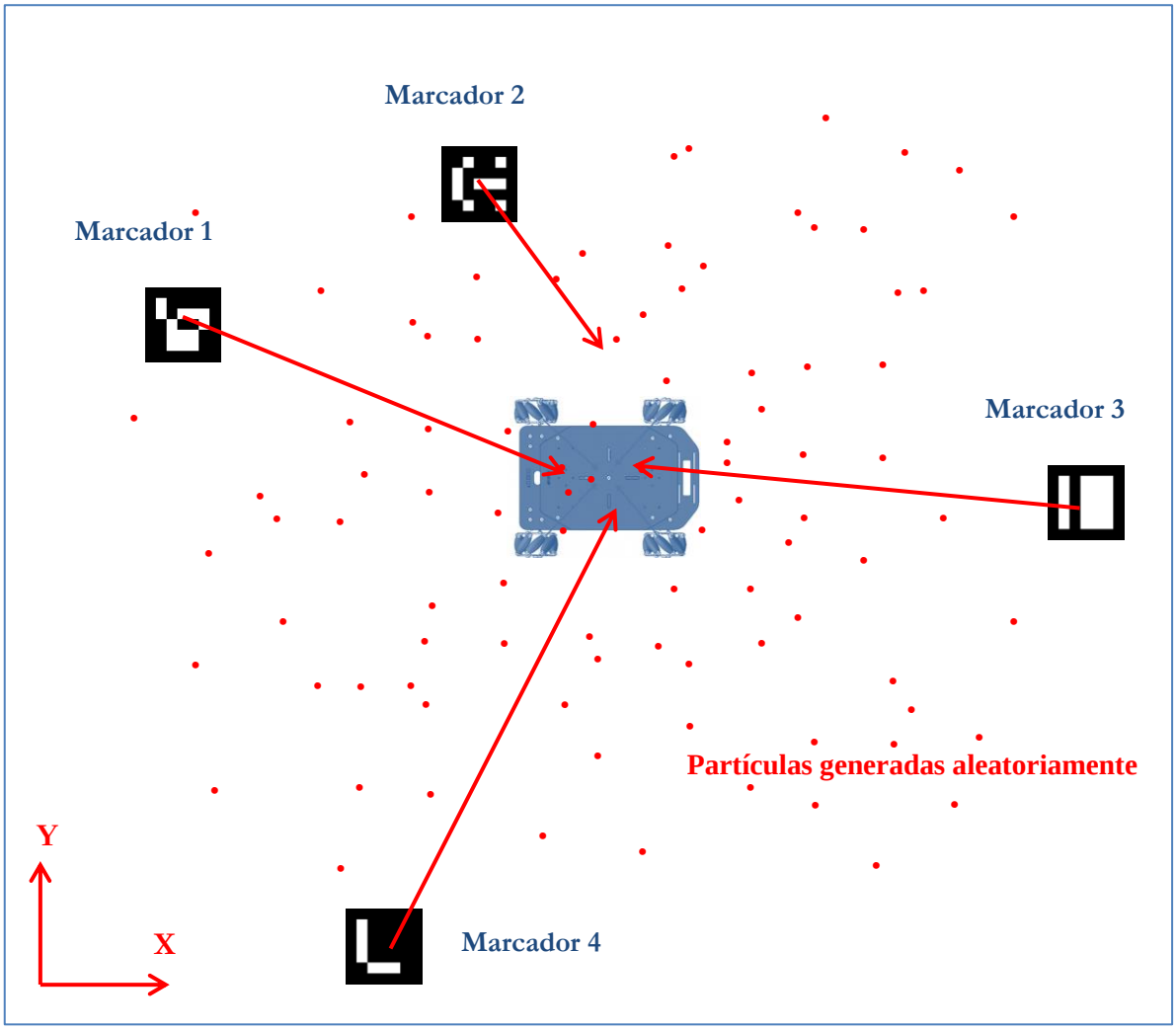

Figura 24: Partículas aleatorias en el espacio de trabajo.

Para determinar la probabilidad de cada partícula según un marcador, se ha utilizado una función gaussiana como la siguiente:

$$f(x) = \frac{1}{\sqrt{2 \cdot \pi \cdot \sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2 \cdot \sigma^2}}$$

Donde:

$$x = |MarkerParticula|; distancia\ entre\ marcador\ y\ partícula$$

$$\mu = 0$$

$$\sigma = f(varianza\ del\ marcador)$$

Con esto conseguimos que cuanto mayor sea la varianza del marcador la curva de probabilidad sea menos acentuada (Azul en imagen), cuanto menor sea la varianza será más acentuada (Rojo en imagen). La "μ" es cero porque suponemos para este cálculo el origen en el marcador, por lo que "x" será la distancia entre partícula y marcador.

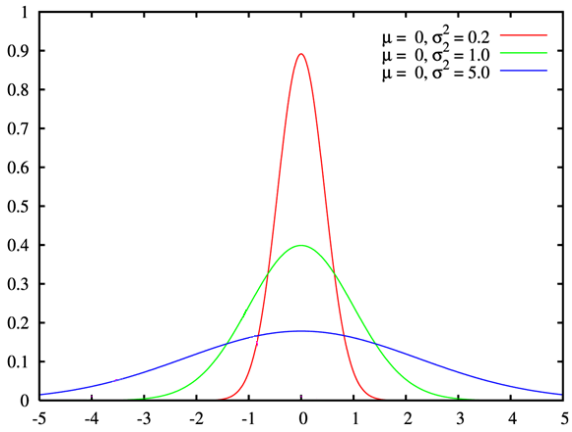

Figura 25: Imagen que representa 4 posiciones devueltas por 4 marcadores diferentes para una posición de robot.

El valor de σ será proporcional a la desviación en las medidas del sensor de manera que cuanta mayor desviación tenga el sensor más ancha será la gaussiana y menor diferencia de probabilidad para diferencias de distancia con respecto al centro. De manera que cuanto mayor sea la diferencia entre medidas del sensor mayor zona de influencia tenga sobre las partículas.

El peso *"W"* total de cada partícula (*"i" partículas*) será el producto de las probabilidades de esa partícula para cada uno de los marcadores (*"m" marcadores*):

$$W(Partícula_i) = \prod_{k=1}^{m} Prob\left(Partícula_{i_{Marcador_k}}\right)$$

Para normalizar las probabilidades:

$$W_{Tot} = \sum_{i=1}^{n} W(Partícula_i)$$

Por lo que la probabilidad de cada partícula normalizada será:

$$P(Partícula_i) = \sum_{i=1}^{n} \frac{W(Partícula_i)}{W_{Tot}}$$

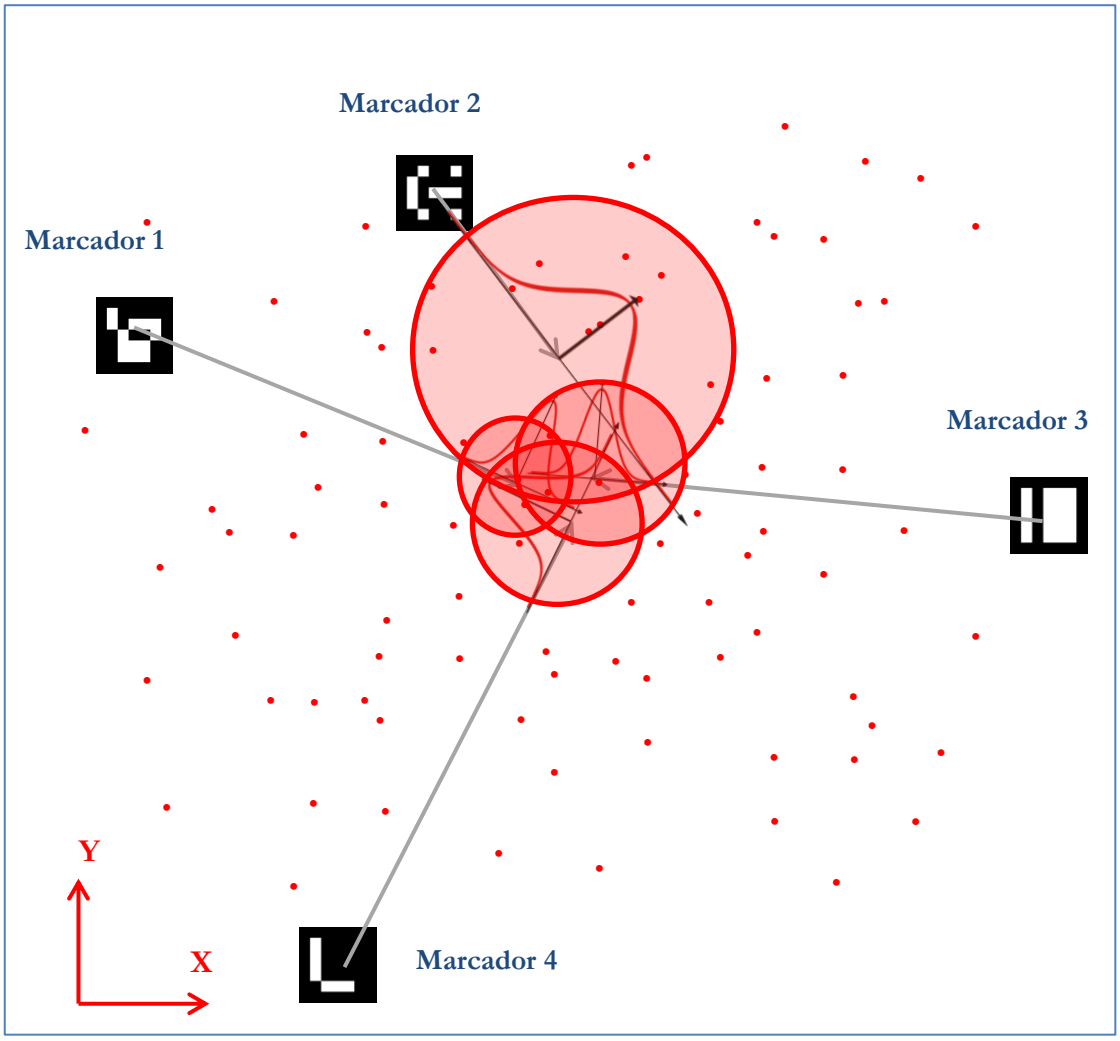

Figura 26: Representación de partículas aleatorias y medidas de marcadores.

Tras determinar y normalizar la probabilidad de cada una de las partículas se realiza un remuestreo (resampling) [25], [26], de manera que las partículas con mayor probabilidad tengan mayor posibilidad de sobrevivir.

De esta manera después de unos pocos ciclos solo quedarán las partículas más próximas a la posición del robot.

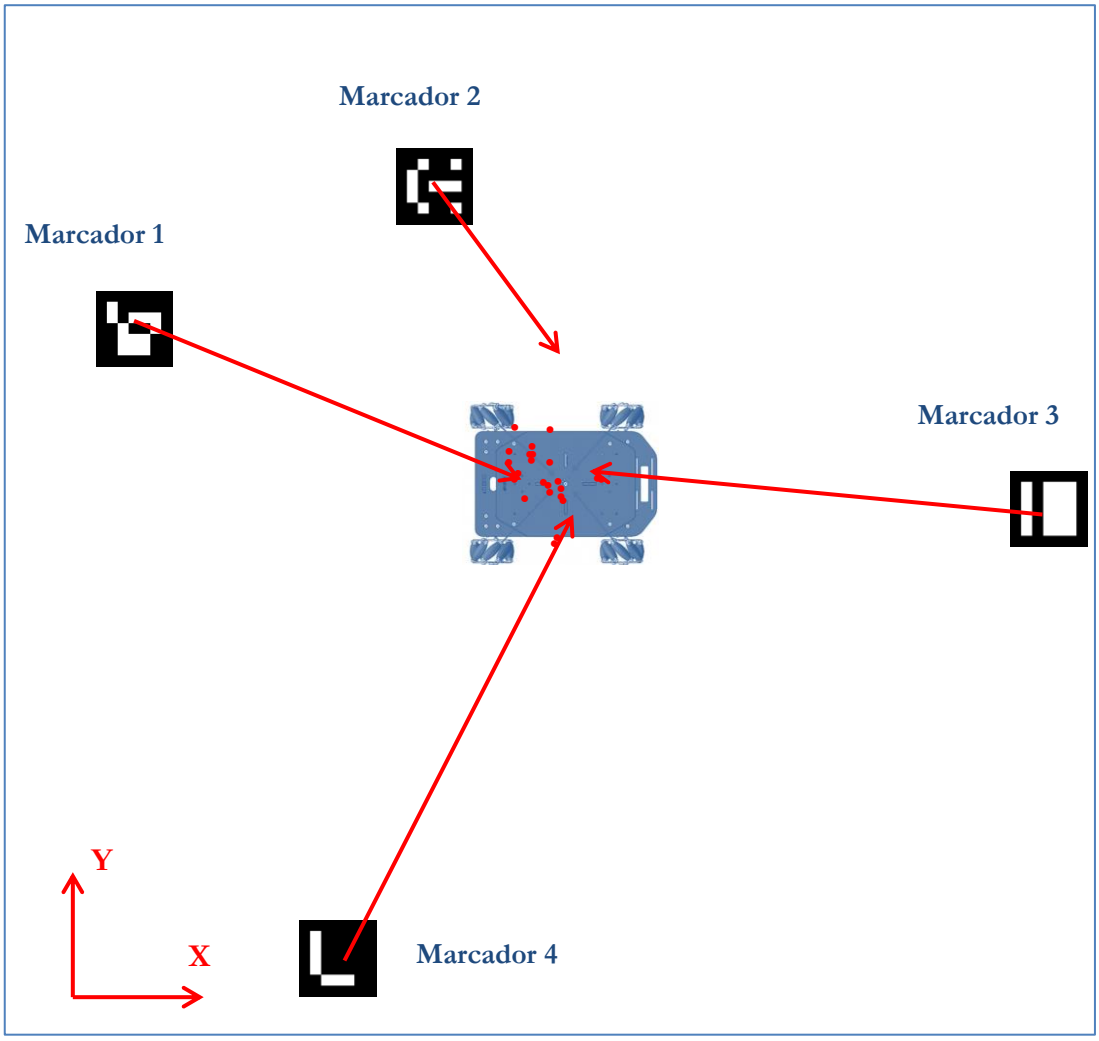

Figura 27: Partículas que sobreviven tras unos ciclos.

Un punto clave en esta implementación es hacer que las partículas se muevan, ya que de otra forma no evolucionaria su posición. Esto lo conseguimos añadiendo ruido aleatorio al conjunto de partículas, de manera que conforme el robot cambia de posición nuevas partículas cogen mayor peso.

Asimismo la cantidad de "ruido" añadido se ha hecho en función de la desviación de posición entre ciclos para facilitar al sistema evolucionar de una posición a otra.

$$ruido = f(Variación\ posición\ robot)$$

$$X_{Partícula_i} = X_{Partícula_i} + ruido$$

$$Y_{Partícula_i} = Y_{Partícula_i} + ruido$$

$$\theta_{Partícula_i} = \theta_{Partícula_i} + ruido$$

Siempre que al menos un marcador sea detectado el sistema tomará por posición del robot la partícula con más peso. Cuando ningún marcador sea detectado la posición del robot será la media de las partículas.

En la siguiente imagen vemos a la izquierda un conjunto de partículas aleatorias creadas en la zona de trabajo y a la derecha las partículas concentradas alrededor de un punto tras unos pocos ciclos de detección.

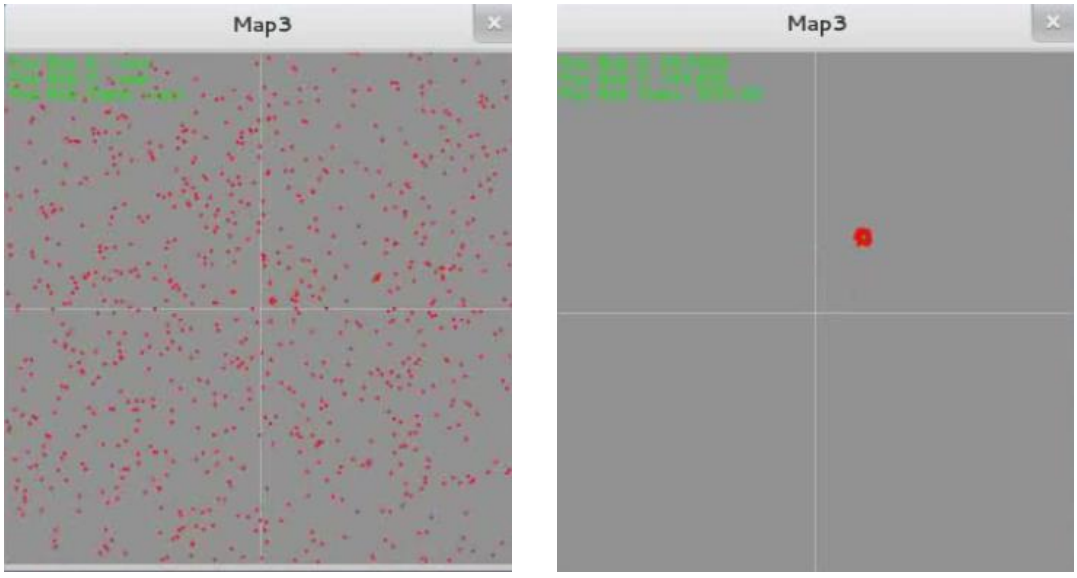

Figura 28: A la izquierda partículas aleatorias creadas en el mapa, a la derecha posición detectada tras unos ciclos.

En la siguiente imagen vemos en la izquierda un conjunto de partículas expandiéndose por el ruido y a la derecha podemos observar en azul la partícula de más peso, esto hará que tenga más probabilidad de sobrevivir, por lo que pocos ciclos después todas las partículas estarán entorno a ese punto.

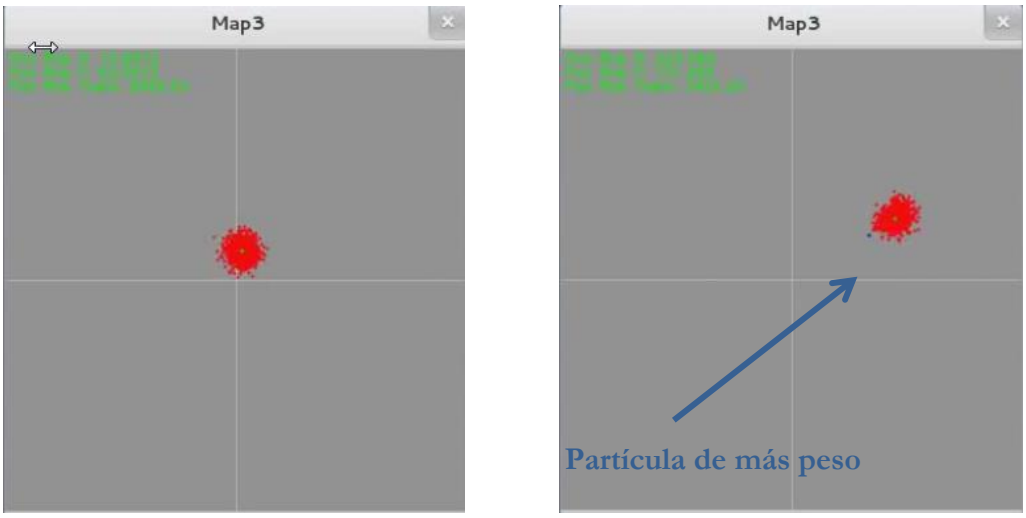

Figura 29: Partículas expandiéndose debido al ruido al no detectar posición.

4.5 Odometría

La información procedente de la odometría del robot ha sido integrada en el filtro de partículas con el objetivo de mejorar la detección de la posición del robot, para ello cada vez que se obtienen datos de la odometría se aplican dichos movimientos a las partículas, consiguiendo:

- Tener una estimación de la posición del robot cuando no existe detección de marcadores.
- Mejorar la rapidez en la respuesta del sistema ya que las partículas se moverán con los datos de la odometría, mientras que si la odometría no existiese el filtro de partículas debería evolucionar con el ruido añadido hasta la nueva posición, cosa que llevaría más tiempo.

Para poder implementar la odometría lo primero ha sido modelar la cinemática del robot, el robot utilizado es del tipo "skid steer" cuyo modelo podemos encontrar en [27], [28] y [29].

Como debido al montaje del sistema de control del robot solo tenemos información de dos ruedas y teniendo en cuenta que la odometría no va a ser el sistema de posicionamiento principal se ha aproximado el modelo al de un robot diferencial [30], [31]:

Partiendo del siguiente esquema:

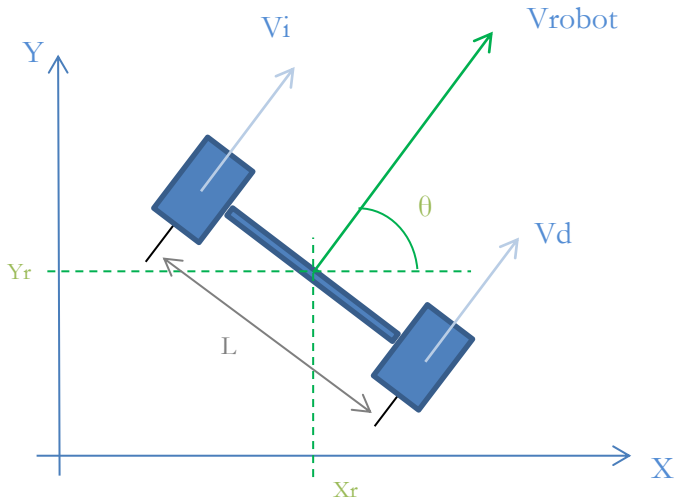

Figura 30: Modelo robot diferencial.

Con el número de incrementos de los encoder podemos obtener la distancia que avanza cada rueda por ciclo, y de esta manera obtener las posiciones y velocidades requeridas con los siguientes cálculos:

$$\Delta d_i(i) = Ni(i) * 2 * \pi * \frac{R}{N_{CuentasEncoder}}$$

$$\Delta d_d(i) = Nd(i) * 2 * \pi * \frac{R}{N_{CuentasEncoder}}$$

Tenemos como dato:

$$R = 57.5$$

$$N_{CuentasEncoder} = 6000$$

$$L = 280$$

Como sabemos con la distancia recorrida por cada una de las ruedas podemos obtener la distancia recorrida por el robot y sus respectivas velocidades:

$$\Delta d_{Robot}(i) = \frac{\Delta d_d(i) + \Delta d_i(i)}{2}$$

De la misma manera podemos saber cuánto se incrementa el ángulo de θ en cada ciclo.

$$\Delta\theta_{Robot}(i) = \frac{\Delta d_d(i) - \Delta d_i(i)}{L}$$

$$\theta_{Robot}(i) = \theta_{Robot}(i-1) + \Delta\theta_{Robot}(i)$$

Por tanto la posición del robot vendrá dada por:

$$\Delta X_{Robot}(i) = \Delta d_{Robot}(i) \cdot \cos\left(\theta_{Robot} + \Delta\theta_{Robot}(i)\right)$$

$$\Delta Y_{Robot}(i) = \Delta d_{Robot}(i) \cdot \sin\left(\theta_{Robot} + \Delta\theta_{Robot}(i)\right)$$

$$X_{Robot}(i) = X_{Robot}(i-1) + \Delta X_{Robot}(i)$$

$$Y_{Robot}(i) = Y_{Robot}(i-1) + \Delta Y_{Robot}(i)$$

5 Mapping

5.1 Detección láser

Para proporcionar al robot información sobre su entorno y que sea capaz de navegar sin colisionar con los obstáculos es necesario dotarle de algún tipo de sensorización.

Para esto algunos de los sensores más comúnmente utilizados son [4]:

Tipo	Detección	Precio
LIDAR (Scanner láser)	Medición precisa de la geometría	Alto
Ultrasonidos	Detección gruesa de obstáculos grandes	Bajo
Infrarrojos	Detección gruesa de obstáculos grandes	Bajo
Láser de distancia puntual	Medición precisa puntual	Medio

Tabla 2: Sensores más comunes.

En este proyecto se ha tratado de buscar una solución económica sin perder la capacidad de tener información de distancia de la geometría de los objetos que se encuentran enfrentados al robot.

La solución planteada para esta tarea consiste en proyectar el haz de un láser, tomar fotos con una webcam y procesarlo mediante un programa de visión artificial. De esta manera la altura del láser en la imagen será función a la distancia que hay entre el robot y el objeto. Esta técnica es utilizada en ocasiones para modelar objeto en 3 dimensiones [32].

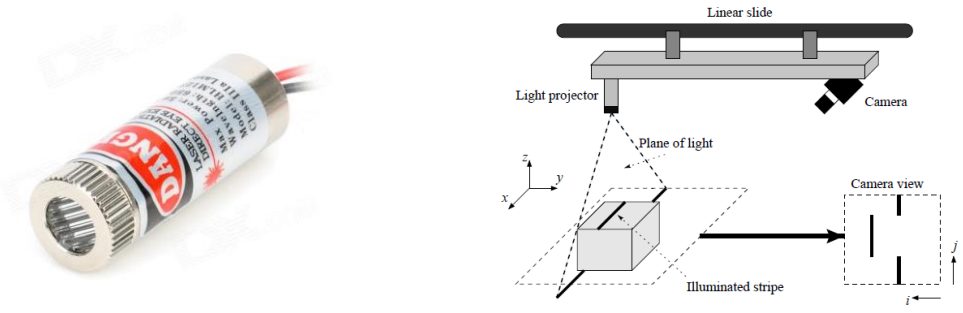

Figura 31: A la izquierda láser de línea utilizado, a la derecha ejemplo de uso de esta técnica.

En la siguiente imagen vemos un esquema que representa la cámara y el láser (haz láser en rojo), se han creado diferentes líneas de proyección que simbolizan los píxeles de la cámara en sentido vertical.

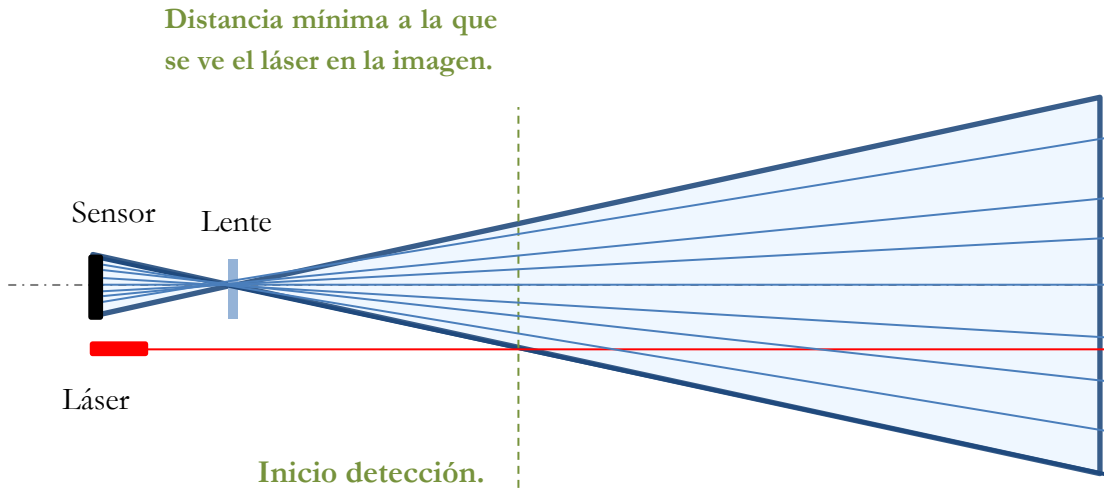

Figura 32: Esquema del sistema de medición láser-cámara.

Una de las primeras observaciones que hacemos es que la distancia entre el centro de la cámara y el láser condicionará el punto de inicio de detección, es decir la distancia mínima a la que el láser se ve en la imagen. El fin de detección vendrá condicionado por el último corte que se produce entre el láser y el haz correspondiente a la proyección de cada pixel vertical.

Para estudiar a que altura en la imagen veríamos los objetos situados frente al sistema calculamos:

Si buscamos la intersección de ambas rectas para los 1024 casos:

Suponemos 1024 píxeles verticales en la imagen:

$$i = 1{:}\,1024$$

Ángulo apertura cámara:

$$\theta_{Ap} = 53º$$

Distancia cámara-láser:

$$d_{c-l} = 110mm$$

Ecuación láser:

$$y_l = \tan\alpha \cdot x_l$$

Ecuación haz del Pixel Vertical "i":

$$y_h = \tan\theta_{C_i} \cdot x_h + d_{c-l}$$

$$Siendo: \theta_{C_i} = -\frac{\theta_{Ap}}{2} + \frac{\theta_{Ap}}{1024} \cdot i$$

Si buscamos la intersección de ambas rectas para los 1024 casos:

$$x_c = -\frac{d_{c-l}}{(\tan\alpha - \tan\theta_{H_i})} \qquad y_c = \tan\alpha \cdot x_c$$

Para una orientación del láser de $\alpha = 0º$ tendremos:

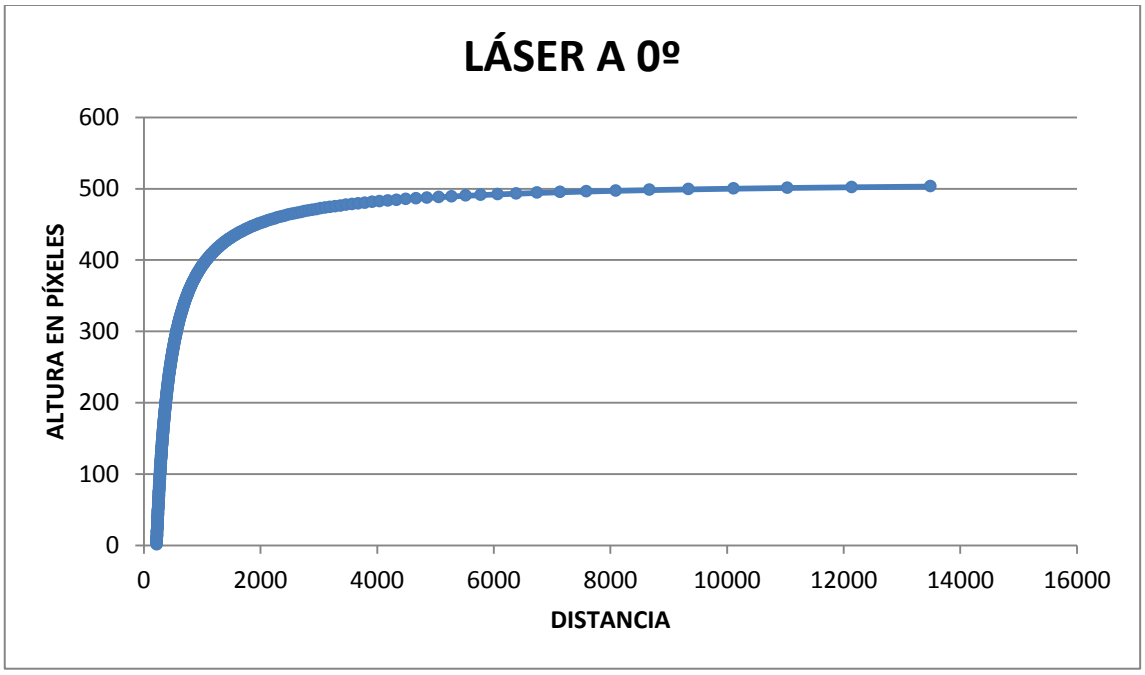

Figura 33: Gráfica distancia-altura en píxeles con el láser a 0 grados.

Como podemos ver en la figura el láser solo se ve en la mitad de los píxeles por estar a cero grados, la detección comienza a 220 mm y tenemos una densidad alta de puntos (450 aprox.) por debajo de los 2000mm por encima de 4000mm la resolución será muy baja.

Si probamos a dar al láser inclinación positiva como representamos en la imagen de la siguiente página, conseguiremos reducir la distancia de inicio y aumentar los cortes en la imagen.

Conforme aumentemos el giro del láser reduciremos el valor mínimo de detección, pero acortaremos el rango de medición, por lo que se trata de buscar un compromiso.

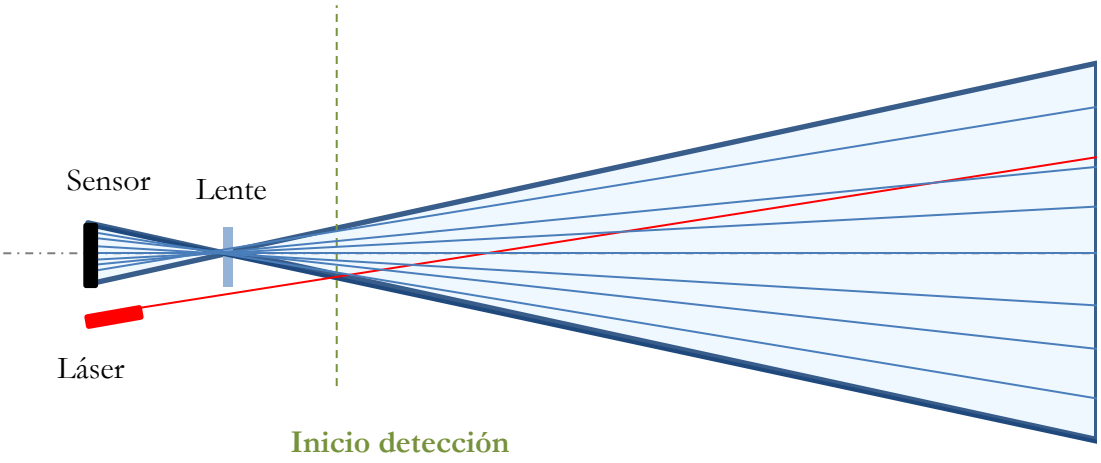

Figura 34: Esquema del sistema de medición láser-cámara con el láser inclinado.

Tras diferentes pruebas finalmente se ha llegado a una orientación de 14,5º. La medición comienza a 145mm y como podemos ver tenemos mayor densidad de puntos que en el caso anterior con bastante resolución hasta los 2500mm.

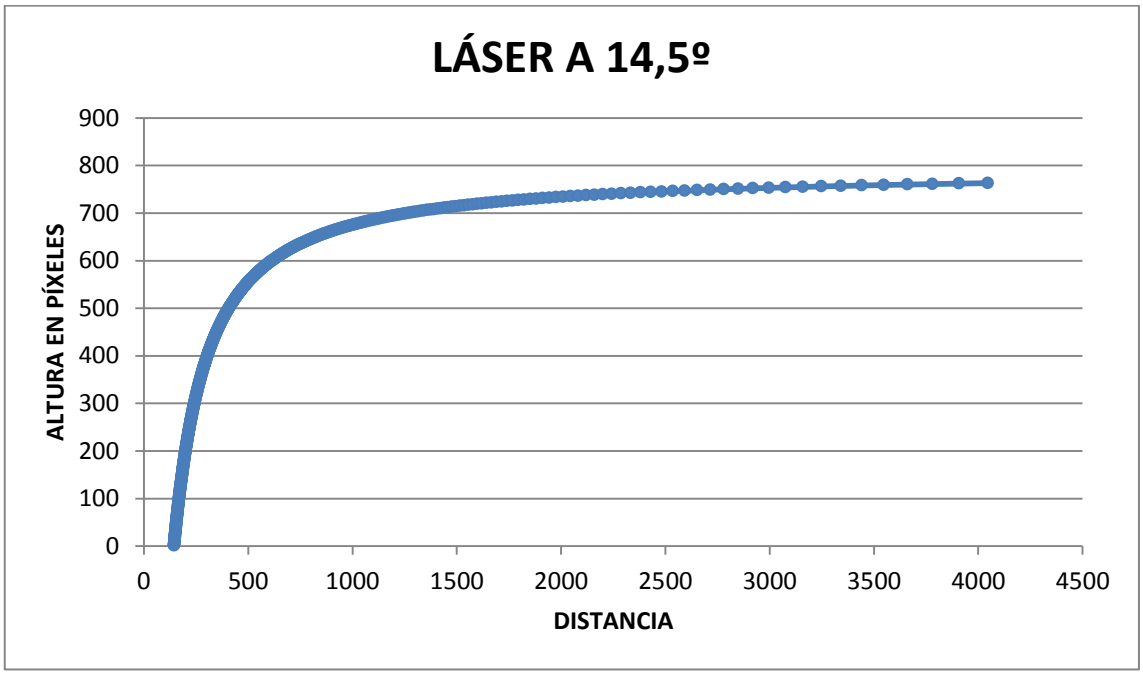

Figura 35: Gráfica distancia-altura en píxeles con el láser a 14.5 grados.

Tras el montaje se ha procedido a la comparación de los datos obtenidos teóricamente y los obtenidos de manera experimental, ambas son bastante similares. (Siguiente gráfica).

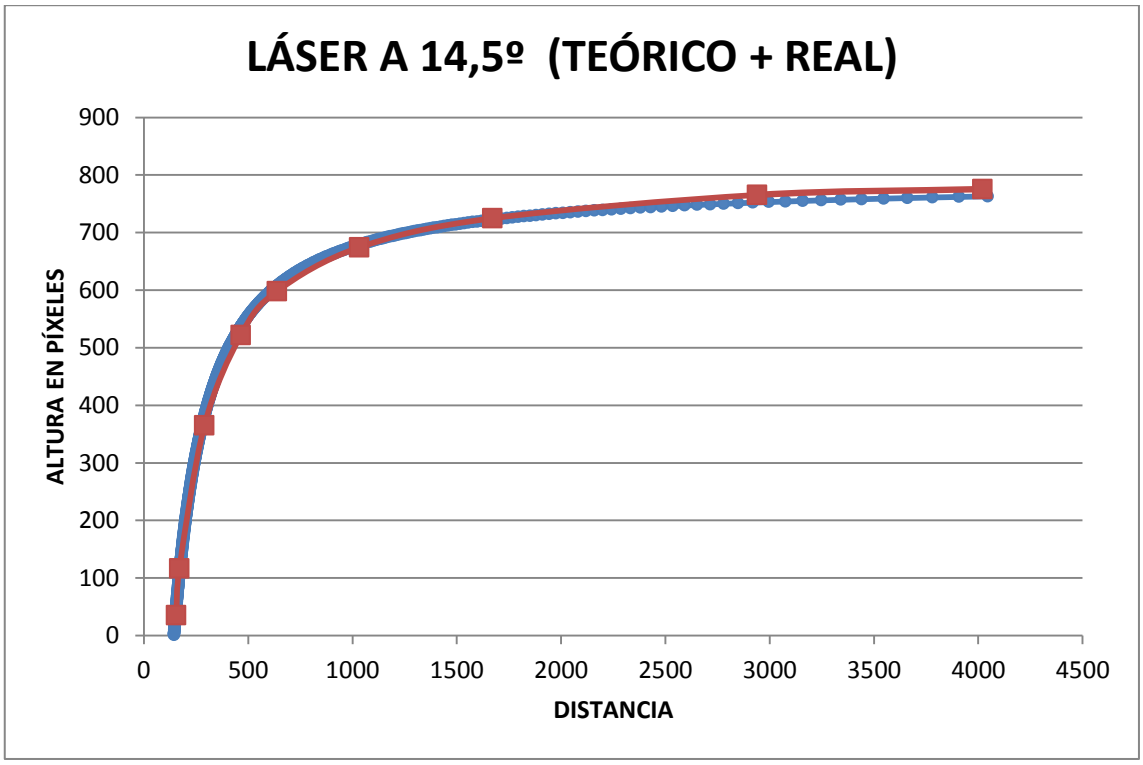

Figura 36: Imagen que representa 4 posiciones devueltas por 4 marcadores diferentes para una posición de robot.

Para detectar el haz láser en la imagen de manera robusta a través del sistema de visión ha sido necesario hacer dos fotos en estático para una misma posición, una con láser y otra sin él. De manera que comparando ambas imágenes podamos detectar la diferencia que será el láser.

Esto se ha hecho así porque con una sola foto la detección del láser en la imagen era mucho más crítica ya que dependía de condiciones de iluminación o del color de los objetos en el entorno.

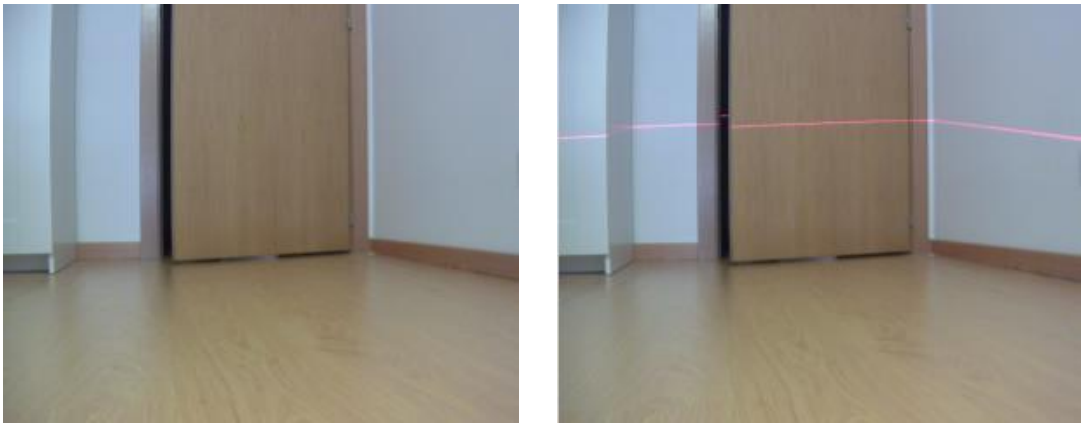

Figura 37: Foto sin láser y con láser.

Una vez calculada la diferencia entre ambas imágenes es necesario umbralizar el resultado. Para esto se ha utilizado el método automático de OTSU [33], de esta manera se consigue un resultado mucho más robusto ante cambios de iluminación. En la siguiente imagen vemos en dos condiciones de iluminación distinta la comparación entre una umbralización fija o mediante el método de OTSU.

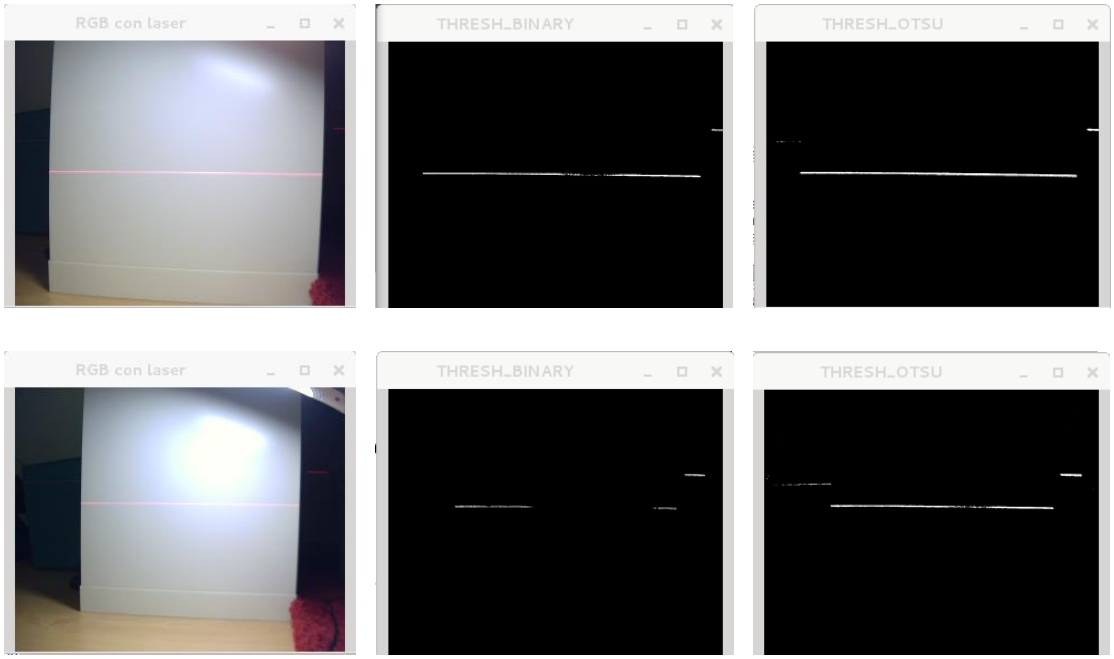

Figura 38: Comparación entre distintas condiciones de iluminación entre umbralizar según el método de Otsu y una umbralización fija.

Como podemos apreciar el método de Otsu se muestra mucho más robusto ante cambios de iluminación. Esto es importante en esta aplicación al tratarse de un robot que navega por diferentes interiores con diferentes condiciones de iluminación.

Para eliminar objetos no deseados ha sido necesario introducir un paso intermedio de filtrado. Este filtro eliminará los objetos pequeños (menos de 10 píxeles), normalmente producidos por ruido; también eliminará los elementos que no tengan las proporciones alargadas habituales de la proyección producida por el haz láser.

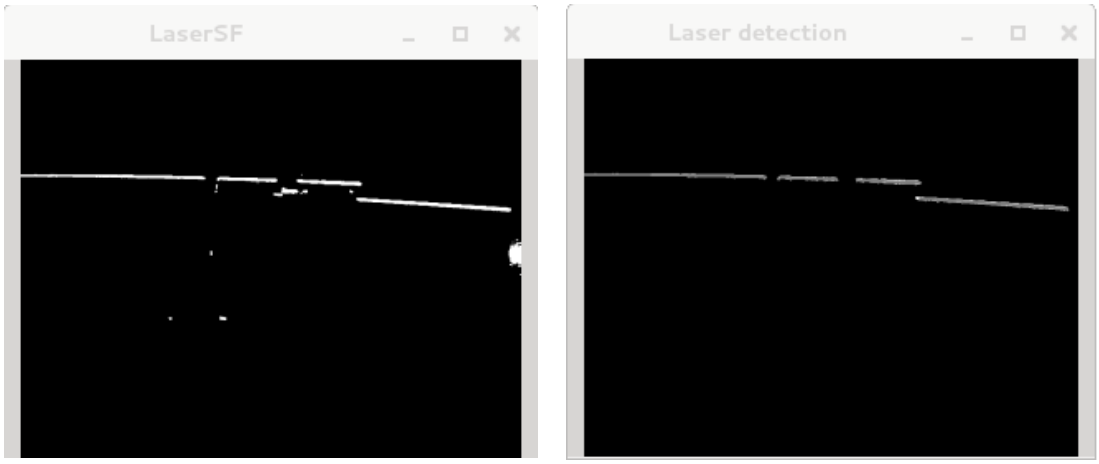

Figura 39: Proceso de filtrado en la imagen umbralizada del láser, ejemplo 1.

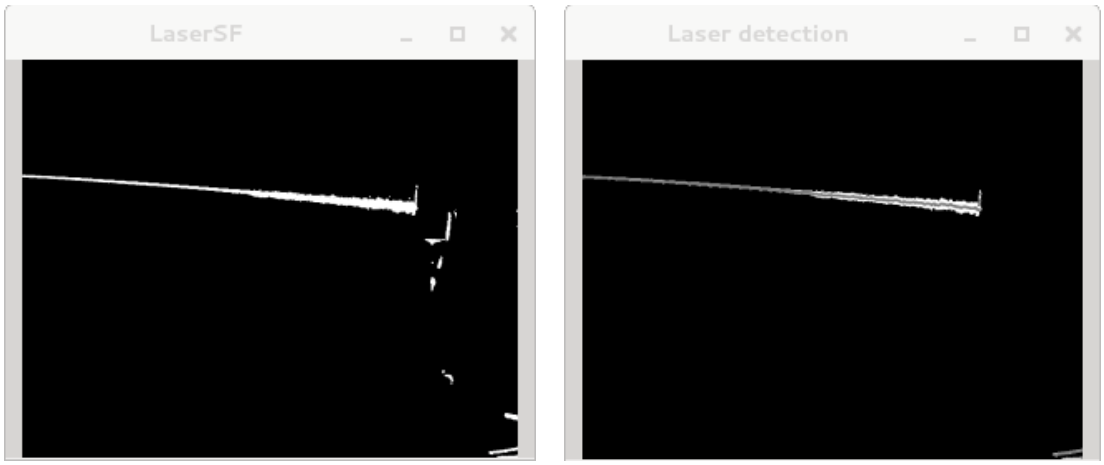

Figura 40: Proceso de filtrado en la imagen umbralizada del láser, ejemplo 2.

Finalmente para determinar la altura en píxeles se ha implementado un algoritmo que busca en sentido vertical el cambio de negro a blanco y luego otra vez a negro, quedándose con el punto intermedio. Esto es porque según la detección el grosor del defecto en varía. En la siguiente imagen vemos un ejemplo en el que se ve como la línea gris es dibujada en el centro del defecto aunque su grosor aumente.

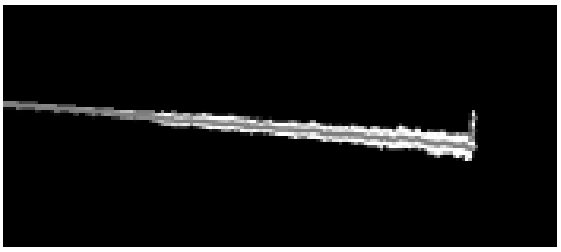

Figura 41: Detalle línea dibujada en el centro del defecto.

Una vez determinada la altura del láser en la imagen en píxeles, el cálculo de para determinar la distancia al objeto es inmediato con las ecuaciones planteadas al comienzo de este apartado.

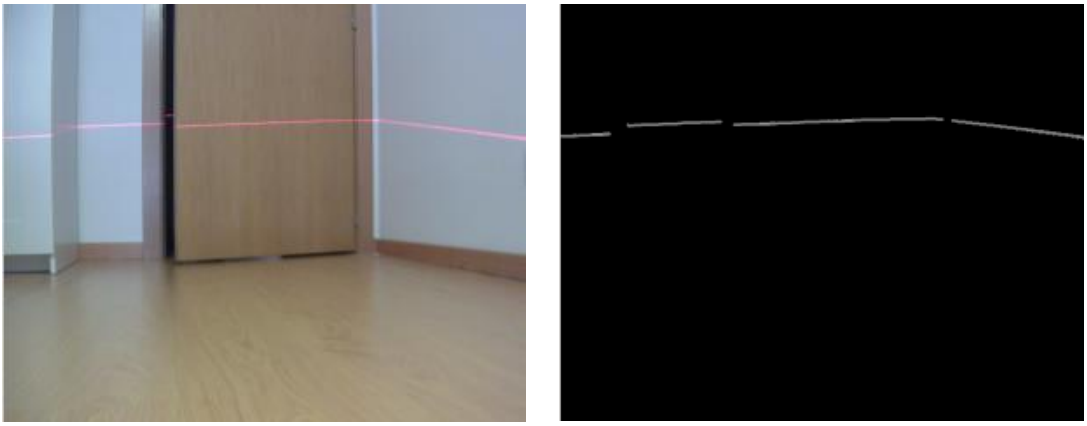

Figura 42: Resultado final.

La imagen tiene una resolución horizontal de 1280 puntos, los cuales se han dividido en 160 medidas. Esto se ha hecho así por eficiencia del código, ya que una sobrecarga de mediciones provocaba un peor rendimiento. Al ser el ángulo de apertura de la cámara 53º si obtenemos 160 medidas, tendremos una medida cada 0.33º que se ha considerado suficiente.

5.2 Construcción mapa

Uno de los objetivos del proyecto es que el robot tuviese la capacidad de navegar por sus inmediaciones, para esto se consideró oportuno que el robot crease un mapa del entorno.

Determinar un mapa del entorno es de especial interés en aplicaciones de robots autónomos ya que permite al mismo localizarse con respecto a una referencia global determinando objetos de interés, obstáculos… Para que el robot pueda crear un mapa de su entorno deberá saber dónde está, en este caso la posición vendrá dada por los marcadores pero suele ser habitual que el robot tenga que utilizar el mismo mapa que construye para localizarse por lo que el problema es recurrente.

Hay diversas técnicas para la creación de mapas [34], la empleada en esta aplicación se trata de una solución muy utilizada en robótica, es la técnica llamada de celdas de ocupación o "occupancy grids" introducida inicialmente en el año 1978 por [35], [36], siendo una de sus grandes ventajas la facilidad en su construcción, esta técnica sigue siendo muy usada en la actualidad en el mundo de la investigación [31] .

Para construir este mapa se utiliza la posición del robot dada por los marcadores y las medidas obtenidas del sistema láser-visión artificial.

Para cada medida de distancia del sistema láser-visión se creará un modelo [38] como el siguiente:

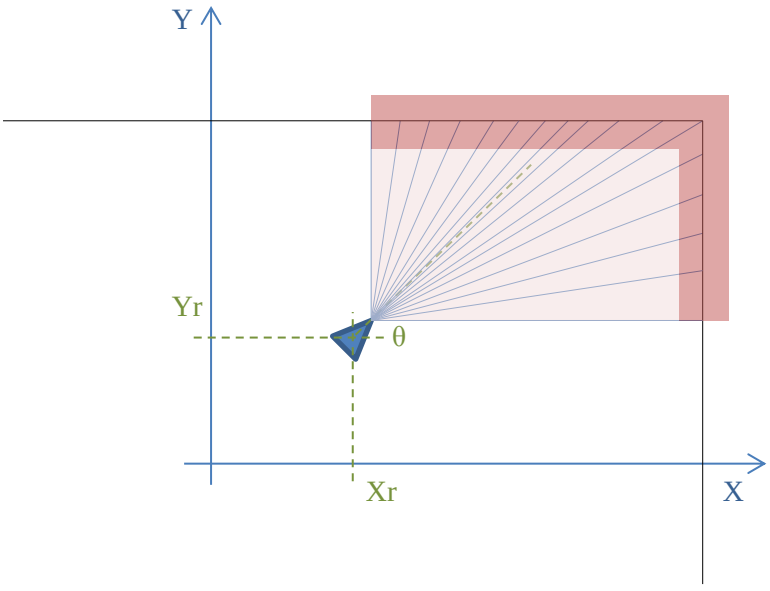

Figura 43: Modelo creado a partir de la medición láser.

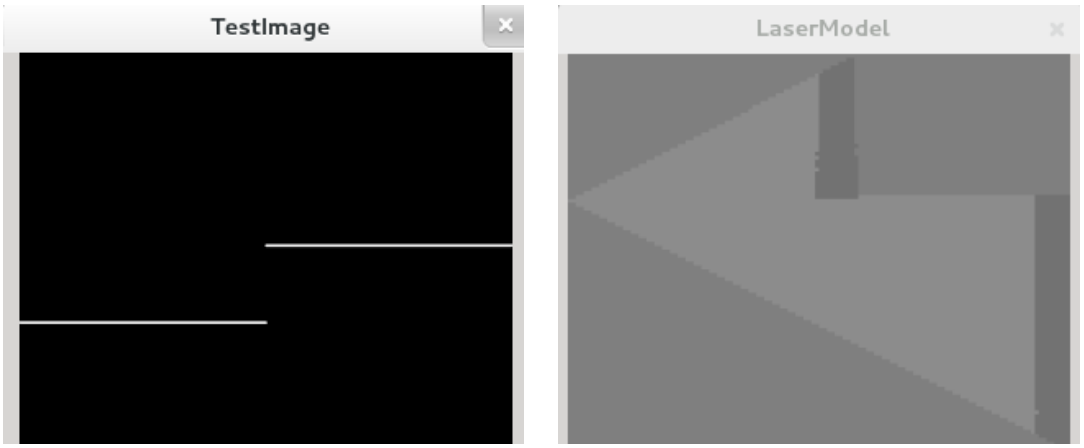

Figura 44: Ejemplo de modelo creado a partir de la medición láser de dos objetos enfrentados a diferente distancia.

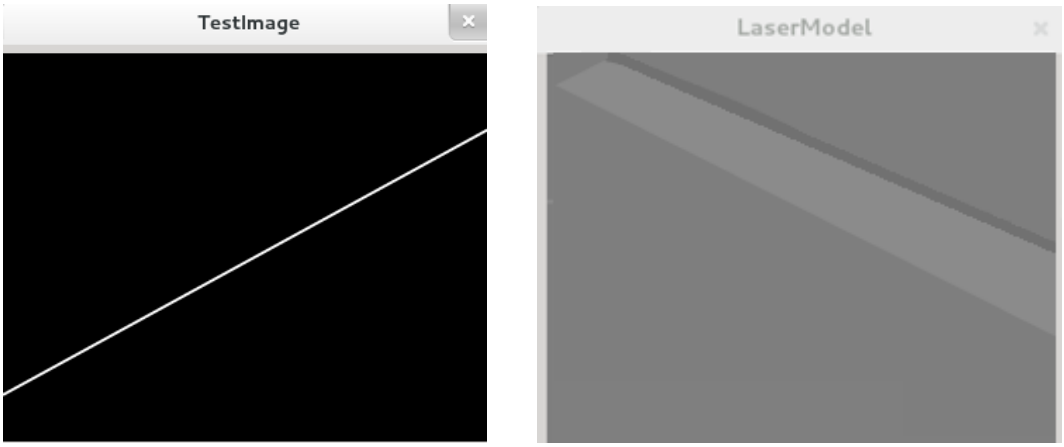

Figura 45: Ejemplo de modelo creado a partir de la medición láser de un objeto oblicuo.

Los cálculos del modelo serían:

$$P\left(\frac{H}{ocupada}\right)=$$

$$\begin{cases} 0.5+\dfrac{0.5}{k_d} & si\ r > d_{med}[i]\ y\ \theta \in [\theta_{med}[i]-dif_{\theta med}/2, \theta_{med}[i]+dif_{\theta med}/2] \\ 0 & si\ r < d_{med}[i]\ y\ \theta \in [\theta_{med}[i]-dif_{\theta med}/2, \theta_{med}[i]+dif_{\theta med}/2] \\ 0.5 & Otro\ caso \end{cases}$$

Siendo $dif_{\theta med}$ la diferencia angular entre una medida y otra, en este caso al haber 160 medidas para 53° de apertura de cámara, este valor será:

$$dif_{\theta med} = 0.33125°$$

Para crear el mapa aplicaremos la teoría bayesiana, en el que el valor de cada celda indica la probabilidad de que dicha celda este ocupada, cumpliendo:

$$P(ocupada)^t = P(ocupada/H)^{t-1}$$

Para luego actualizar el mapa según:

$$P(ocupada)^t = P(ocupada/H)^{t-1}$$

$$P(ocupada/H)^t = \frac{P(H/ocupada) \cdot P(ocupada/H)^{t-1}}{\sum_{n=1}^{k} P(H/E_n) \cdot P(E_n/H)^{t-1}}$$

Donde:

$$\sum_{n=1}^{k} P\left(\frac{H}{E_n}\right) \cdot P\left(\frac{E_n}{H}\right)^{t-1} =$$

$$= P(H/ocupada) \cdot P(ocupada/H)^{t-1} + P(H/\overline{ocupada}) \cdot P\left(\overline{ocupada}/H\right)^{t-1} =$$

$$= P(H/ocupada) \cdot P(ocupada/H)^{t-1} + (1 - P(H/ocupada)) \cdot (1 - P(ocupada/H)^{t-1})$$

Siendo: $P(H/ocupada)^0 = P(ocupada/H)^0 = 0.5$

Tras cada medida nueva del láser el mapa se irá actualizando dando resultados como el siguiente:

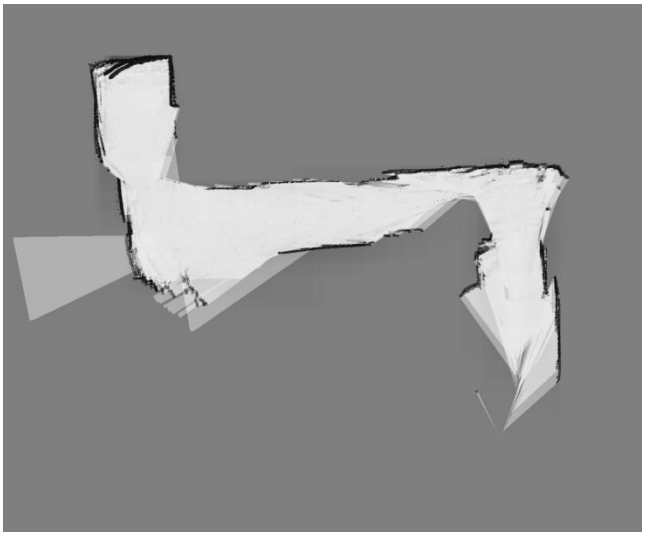

Figura 46: Mapa de probabilidad de ocupación creado por el robot.

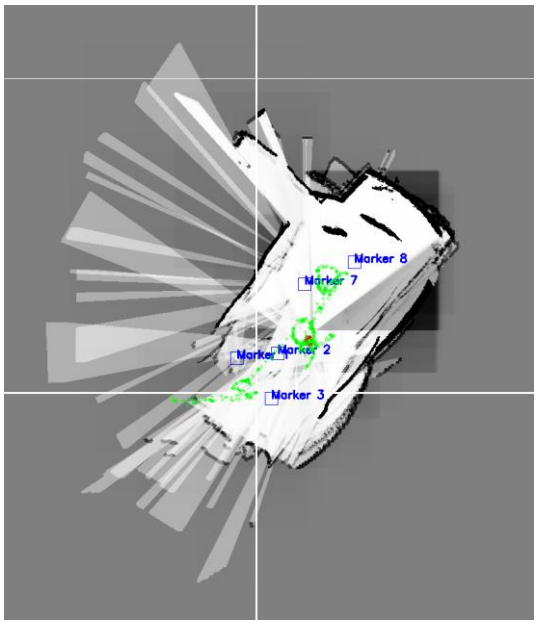

Figura 47: Mapa de probabilidad de ocupación creado por el robot en el que se aprecian medidas erróneas.

6 Navegación

Se han implementado 4 tipos de navegación en el robot:

- Teleoperación: el usuario teleoperará el robot, el cual se limitará a cumplir con los comandos dados por el usuario. Permitiéndose 4 movimientos, avanzar, retroceder, giro a derecha y giro a izquierda.
- Navegación aleatoria: el robot navegará por el entorno tratando de evitar los obstáculos, el robot chequeará en cada ciclo el giro de las ruedas, para comprobar que se está moviendo. En caso de estar bloqueado el robot hará movimientos para tratar de salir de este estado.
- Seguimiento de marcadores: el robot partiendo del marcador con ID 1 seguirá los marcadores de manera incremental, cuando el robot llegue a la posición de un marcador partirá al siguiente si lo detecta dentro de la imagen desde dicha posición.
- Maximizar área explorada: Cuando el robot trabaja en este modo, tratará de ir hacia zonas sin explorar para maximizar el mapa, para ello partiendo del mapa escaneado calculará la trayectoria a seguir en función de los obstáculos.
    - Si el robot se bloquea 60 segundos en este modo, hará un movimiento de desbloqueo.
    - Si el robot se bloquea 3 minutos cambiará la ruta de exploración.

7 Cálculo de trayectorias

En el modo de navegación "Maximizar área explorada" el robot debe ser capaz a partir del mapa de navegar maximizando el mapa. Para esto se le hace ir hacia una zona fuera de su zona conocida, siguiendo los siguientes pasos:

1. Se calcula un punto objetivo fuera de la zona explorada. Sabiendo el tamaño máximo del mapa explorado, si le mandamos uno de los 4 vértices externos tratará de salir por una zona no conocida. Esta simplificación tiene sentido en entornos cerrados en el cual una vez explorado todo el mapa no encontrará ruta de salida.
2. Engrosamiento de obstáculos, para evitar que el robot colisione con obstáculos, el engrosamiento mínimo será la mitad de la dimensión más grande del robot más una distancia de seguridad, en este caso el doble.
3. Cálculo de la trayectoria para llegar desde el punto origen (Actual) hasta el punto destino (Objetivo). Esto se ha hecho con el algoritmo de búsqueda A* explicado a continuación.

Hay diversas técnicas desarrolladas sobre planificación de trayectorias [39], en este caso se ha utilizado el algoritmo llamado "A*" desarrollado por Nils Nilsson en 1968 [40].

Este algoritmo es muy usado en búsqueda de trayectorias, y ofrece un mejor rendimiento en tiempo frente a otros algoritmos. Se trata de un algoritmo que busca la ruta mínima entre dos nodos o puntos teniendo en cuenta el coste mínimo para llegar a ella. Para determinar previamente que nodo elegir se hará uso de la siguiente fórmula:

$$F(x) = G(x) + H(x)$$

Donde:

$G(x)$ Es el número de pasos hasta la posición actual.

$H(x)$ Es una función que es usada como aproximación a la solución. Cuanto más precisa sea, más rápido alcanzará el objetivo el algoritmo. En este caso H(x) se ha igualado a la distancia hasta el punto final desde el actual.

En cada ciclo se recalculará la trayectoria a seguir por el robot para llegar al objetivo desde el punto actual.

Figura 48: Mapa engrosado y ruta en rosa calculada por el planificador de trayectorias.

8 Diseño software

8.1 Introducción

En este apartado se trata de justificar la elección de los lenguajes de programación utilizados para el desarrollo de la aplicación. También se estudiará la estructura y funcionamiento de la misma a través de diagramas UML.

8.2 ¿Por qué Linux?

Hay 3 ventajas fundamentales de Linux que juntas le dan una gran consideración:

- Linux es libre: Esto implica no sólo la gratuidad del software, sino también que es modificable y que tiene una gran cantidad de aplicaciones libres en Internet.
- Linux es uno de los sistemas operativos más robustos, estables y rápidos.
- Necesita pocos recursos: funciona en un 386.
- El manejo de la memoria de Linux evita que los errores de las aplicaciones detengan el núcleo de Linux.

Linux es multitarea y multiusuario: Esta característica imprescindible está en Unix desde su inicio.

8.3 ¿Por qué Debian?

Ventajas frente a Ubuntu

- Estabilidad, Debian es una de las distribuciones GNU/Linux más maduras y estables.
- Debian tiene más bagaje que Ubuntu.
- Debian tiene más paquetes en repositorios oficiales.
- Debian soporta muchas más arquitecturas que Ubuntu, Plataformas: Debian está disponible oficialmente para 11 arquitecturas. Intel x86 / IA-32 (“386"), Motorola 68k (m68k), Sun SPARC (“sparc”), Alpha (“alpha”), Motorola / IBM PowerPC (“powerpc”), ARM (“arm”), MIPS (“mips” y “mipsel”), HP PA-RISC (“hppa”), IA-64 (“ia64"), S/390 (“s390"), y AMD64 (“amd64"). Por su parte, Ubuntu está disponible únicamente para tres plataformas: Intel x86, PowerPC (no soportada oficialmente por Canonical) y AMD64.

Tenemos varios grupos de paquetes en la instalación dependiendo del uso que le vayamos a dar al SO.

Ventajas Ubuntu

- Fácil de usar para usuarios inexpertos.
- Compatibilidad Hardware

Debian y Ubuntu están hechos para distintos usuarios. Ubuntu se dirige a los usuarios inexpertos nuevos en Linux, mientras que Debian es un SO minimalista, sin adornos creado para desarrolladores, y entusiastas del código abierto.

8.4 ¿Por qué C++?

Principalmente los motivos de elegir C++ fueron el conocimiento previo del mismo y que OpenCV ha sido desarrollado en C y C++ aunque tenga interfaces para otros lenguajes.

- Reutilización de código.
- Crear y usar nuevos tipos de datos es más fácil que en otros lenguajes.
- El manejo de memoria en C++ es fácil y transparente.
- Orientación a objetos.
- Sobrecarga de operadores y funciones.
- Rapidez.
- Genera programas compactos.

Es muy potente en lo que se refiere a creación de sistemas complejos, es un lenguaje muy robusto.

8.5 ¿Por qué OpenCV?

OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por Intel y que se ha utilizado en infinidad de aplicaciones. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación.

OpenCV es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X y Windows.

El proyecto proporciona un entorno de desarrollo fácil de utilizar y altamente eficiente, esto se ha logrado, realizando su programación en código C y C++ optimizados, aprovechando además las capacidades que proveen los procesadores multi-núcleo. OpenCV puede además utilizar el sistema de primitivas de rendimiento integradas de Intel, un conjunto de rutinas de bajo nivel específicas para procesadores Intel.

En Windows existen más librerías de visión artificial, como AForge.NET o las librerías de visión de Matlab. La programación es más sencilla en ambas plataformas que en OpenCV donde la programación es un poco ardua, pero OpenCV ofrece mejor rendimiento.

En esta aplicación la velocidad en la adquisición, tratamiento y detección de imágenes con marcadores repercutirá directamente en el error sobre la precisión de la posición, así como en el tiempo de ejecución del programa.

Además OpenCV se ha utilizado en las siguientes aplicaciones conocidas:

- Ha sido usada en el sistema de visión del vehículo no tripulado Stanley de la Universidad de Stanford [6].
- Se utiliza en sistemas de vigilancia de vídeo.
- Es la clave en el programa Swistrack [41], una herramienta de seguimiento distribuida.
- Es el paquete principal de visión en ROS [42].

8.6 Diagramas UML

En este apartado se va a proceder a explicar el funcionamiento del código a través de diagramas UML.

8.6.1 Diagrama de estado

El siguiente diagrama de estado trata de representar los diferentes estados del programa desde el punto de vista del tipo de navegación.

Se han representado seis estados diferentes en la ejecución del programa. Al arrancar el inicializará las variables y procesos necesarios en el estado "Inicializar procesos", una vez en ejecución según haya seleccionado el operario podrá estar en uno de los cuatro estados de navegación existentes "Navegación Aleatoria", "Teleoperación", "Seguimiento de marcadores" o "Maximizar área". Desde cualquier estado de navegación se podrá pasar a otro a través del estado de inicialización del proceso. Por último cuando el usuario lo desee podrá finalizar el proceso existiendo un estado llamado "Finalizar procesos" encargado de terminar todas las tareas.

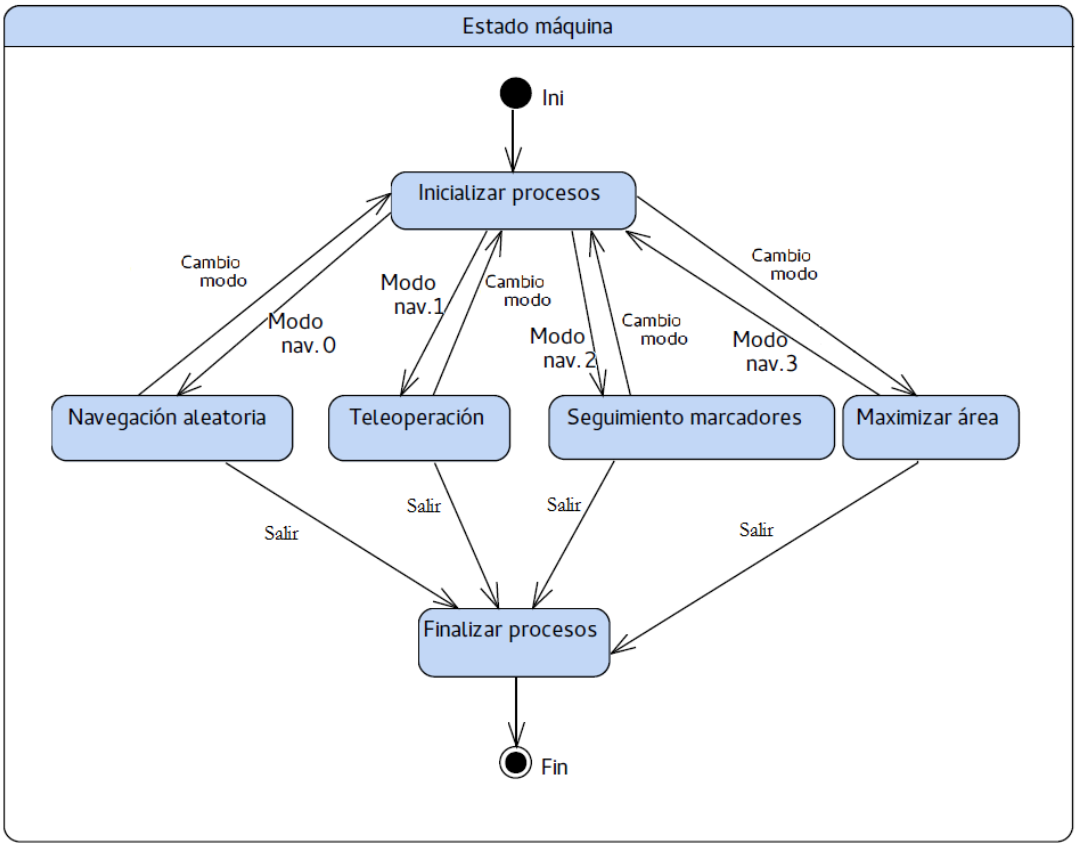

Figura 49: Diagrama de estados.

8.6.2 Diagramas de actividad

Para conseguir el máximo rendimiento en la ejecución del programa se han dividido las tareas en diferentes hilos (threads) [43]. Con esto se consigue que diferentes tareas con diferente latencia se ejecuten por separado sin perjudicar unas a otras. Y exprimir al máximo la multitarea al disponer de un PC de doble núcleo.

Habrá cinco hilos contando con el principal:

- **Main**: Se encargará de inicializar la aplicación, arrancar los demás hilos, de la visualización, de los cambios de modo y de finalizar el proceso si el usuario lo requiere.
- **Thread "ImagesProvider"**: Este thread se encargará de adquirir imágenes de ambas cámaras de manera continua de manera que cuando otro hilo demande un imagen esté preparada la última.
- **Thread "Location"**: Este thread ejecuta la detección de marcadores en las imágenes, el filtro de partículas y aplica las lecturas de odometría a las partículas.

- **Thread "motion"**: Este thread se encargará de los movimientos del robot, de la detección del láser en las imágenes y de la creación y actualización del mapa.
- **Thread "SerialRead"**: Se encarga de la lectura de datos de la comunicación serie con el control del robot.

8.6.2.1 Thread "Main"

El proceso principal cuando se ejecute por primera vez inicializara las variables y componentes necesarios, luego arrancará los demás hilos del programa y entrará en un bucle durante la ejecución en el que actualizará la visualización cíclicamente y atenderá las peticiones de usuario sobre cambio de modo o visualización. Cuando el usuario estime oportuno solicitará finalizar el programa, lo que provocará la salida del bucle principal y la finalización del resto de procesos.

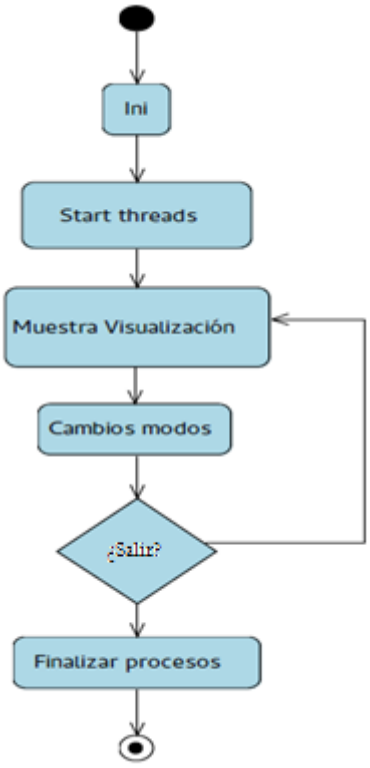

Figura 50: Diagrama de actividad thread "Main".

8.6.2.2 Thread "ImagesProvider"

Este hilo se encargará de obtener cíclicamente imágenes de ambas cámaras, estas cámaras serán almacenadas en un recurso compartido protegido por un mutex [44], cuando el usuario solicite finalizar el proceso principal este hilo saldrá del bucle y finalizará.

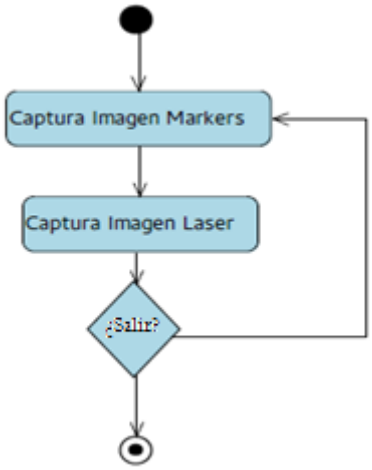

Figura 51: Diagrama de actividad Thread "ImagesProvider".

8.6.2.3 Thread "Location"

Este hilo se encargará de la localización, esto consiste en: Aplicar los cambios detectados en la odometría a las partículas, detectar los marcadores de las imágenes para luego ejecutar el filtro de partículas, por último en base al conjunto de partículas estimará la posición del robot. Cuando el usuario solicite finalizar el proceso principal este hilo saldrá del bucle y finalizará.

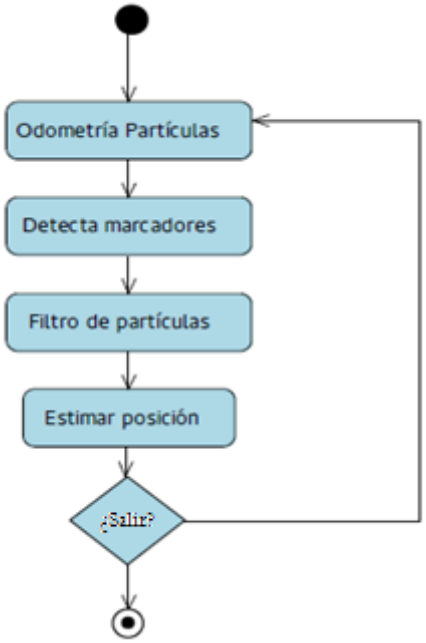

Figura 52: Diagrama de actividad thread "Location".

8.6.2.4 Thread "Motion"

Este hilo se encargará de los movimientos, la detección láser y del Mapping. Primero se realizarán los movimientos que dependerán del modo en el que estemos y posteriormente se procederá a la detección láser con la que se creará el mapa. Cuando el usuario solicite finalizar el proceso principal este hilo saldrá del bucle y finalizará.

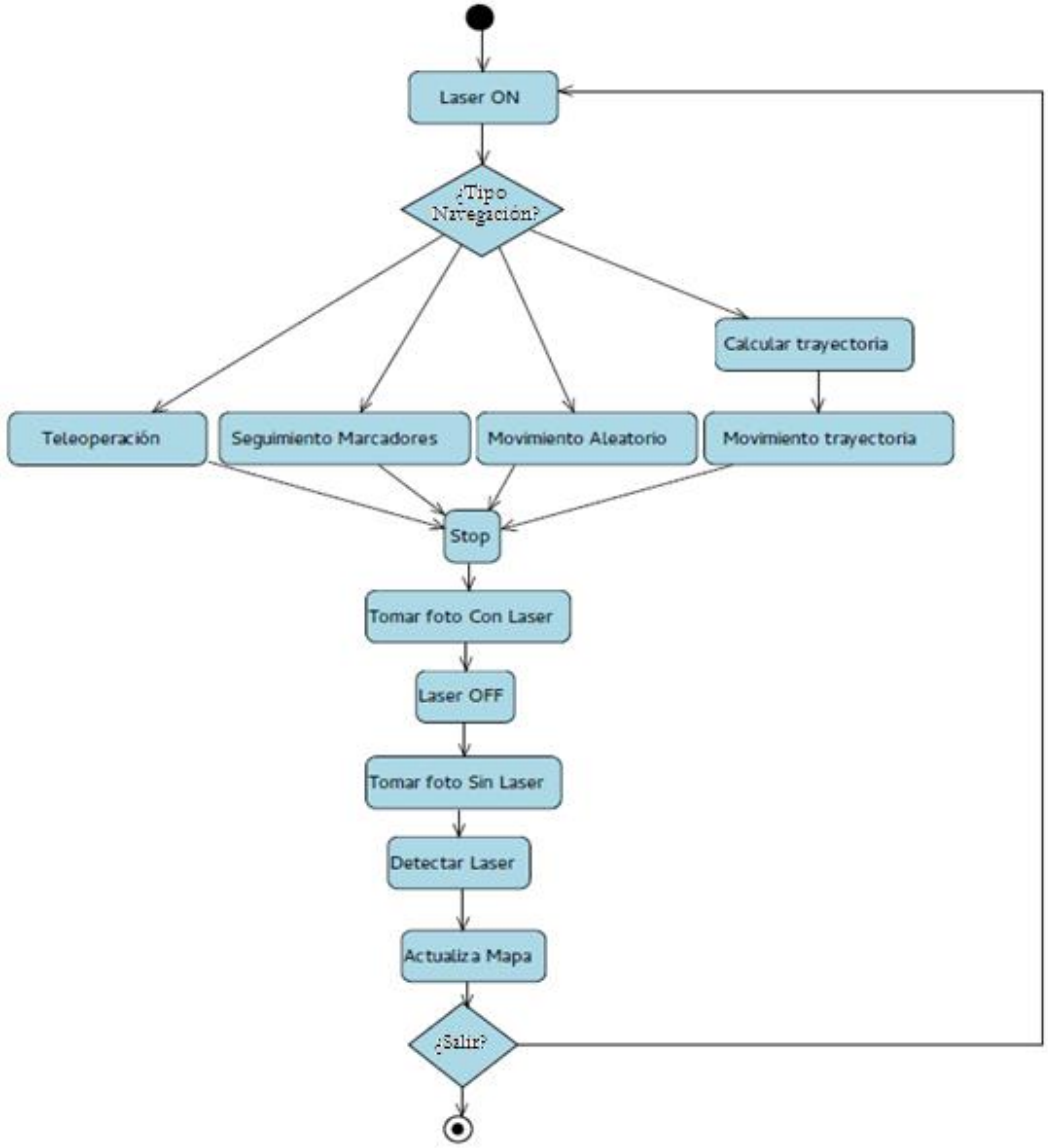

Figura 53: Diagrama de actividad thread "Motion".

8.6.2.5 Thread "SerialRead"

Por último este hilo se encargara de leer cíclicamente los valores del buffer recibidos por la comunicación serie con el control del robot. Cuando el usuario solicite finalizar el proceso principal este hilo saldrá del bucle y finalizará.

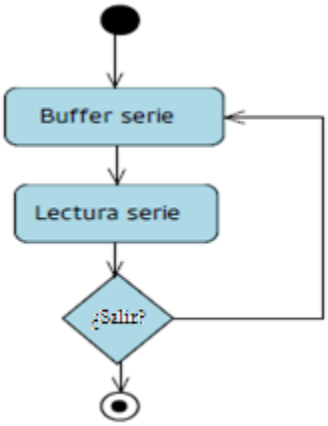

Figura 54: Diagrama de actividad thread "SerialRead".

8.6.3 Casos de uso

En el siguiente diagrama de casos de uso se ha representado las tareas que puede realizar el usuario en el programa. Además de arrancar, parar o resetear programa, el usuario puede cambiar el tipo de modo de navegación, teleoperar el robot o cambiar los modos de visualización como se muestra en el siguiente diagrama. En los anexos se adjunta las tablas correspondientes a los casos de uso.

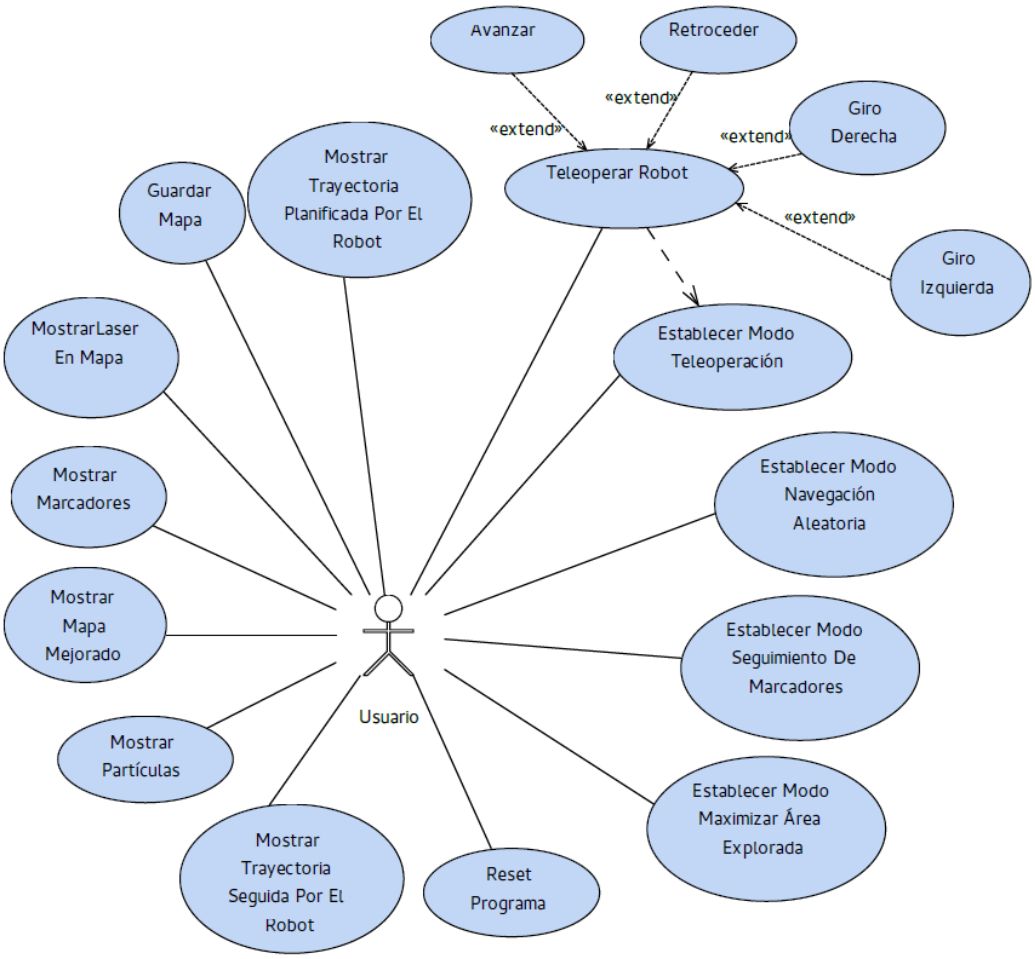

Figura 55: Diagrama de casos de uso de la aplicación

8.6.4 Diagramas de secuencia

A continuación vemos el diagrama de secuencia de la aplicación en el que queda descrito cómo interactúan los diferentes objetos de la aplicación.

Se han representado los 5 hilos, "Main", "ImagesProvider", "Location", "SerialRead", "Motion" y dos clases "auxFunc" y "comSerial" donde se encuentran las funciones auxiliares y las funciones de la comunicación serie.

Como podemos ver el hilo principal o "Main" interactúa con las clases de funciones, inicializa las cámaras y el puerto serie para después arrancar secuencialmente los hilos. Todos los hilos quedarán atrapados en un bucle ejecutando cíclicamente sus tareas hasta que se finalice la aplicación. De la misma manera el hilo principal tras haber arrancado los demás hilos será el encargado de actualizar la visualización y atender los comandos de usuario cíclicamente.

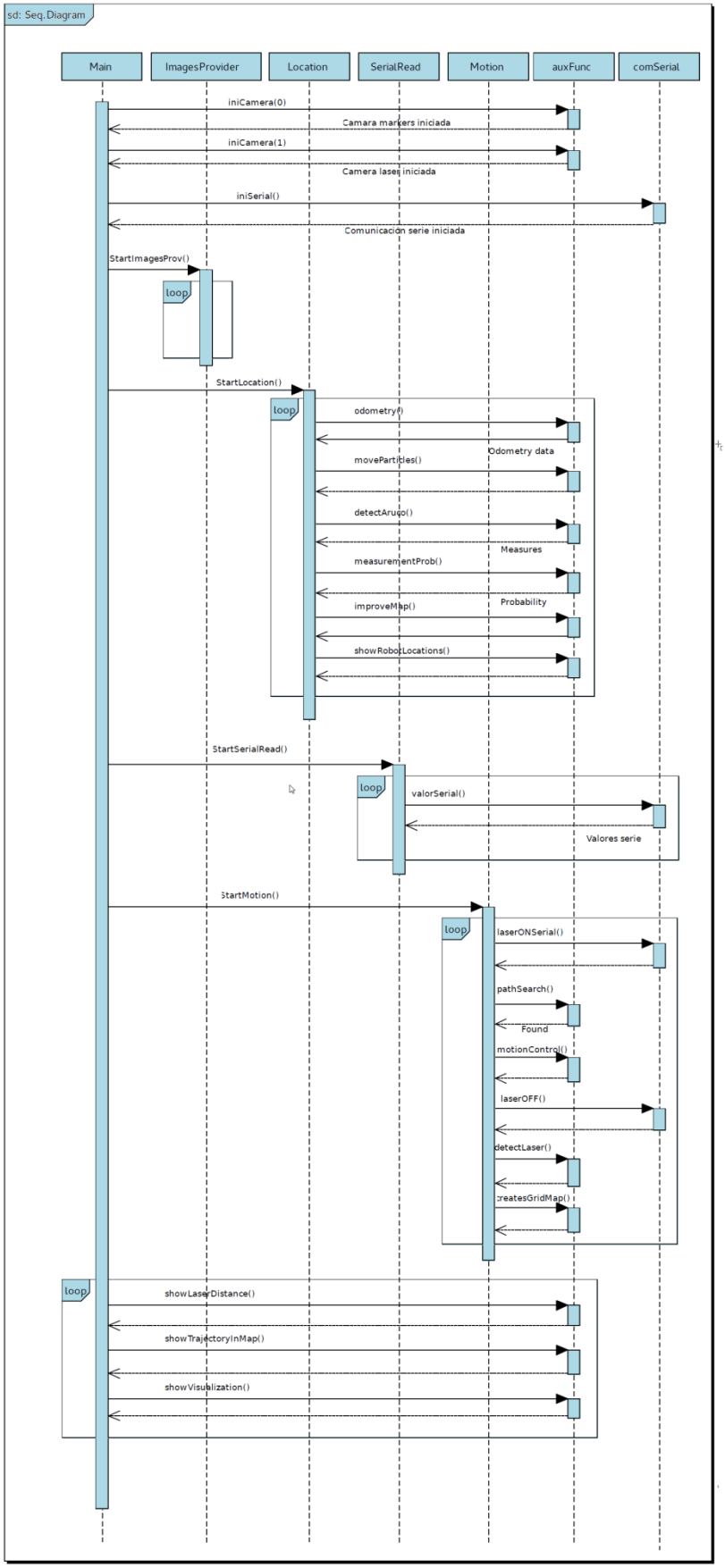

Figura 56: Diagrama de secuencia de la aplicación.

Debido al reducido tamaño para poder meter todo el esquema en la página anterior, a continuación se han ampliado cada uno de los hilos para su estudio.

8.6.4.1 Detalle diagrama de secuencia Images provider

El hilo encargado de servir imágenes como vemos no interactuará con el resto de objetos, simplemente se encargará de adquirir imágenes cíclicamente y guardarla en recursos de memoria compartida, el acceso a zonas de manera compartida será protegido con exclusiones mutuas (mutex) [44].

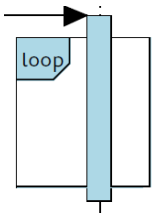

Figura 57: Detalle diagrama de secuencia images provider.

8.6.4.2 Detalle diagrama de secuencia Location

Este hilo se encargará del posicionamiento del robot, esto incluye: odometría, detección de marcadores, y filtro de partículas.

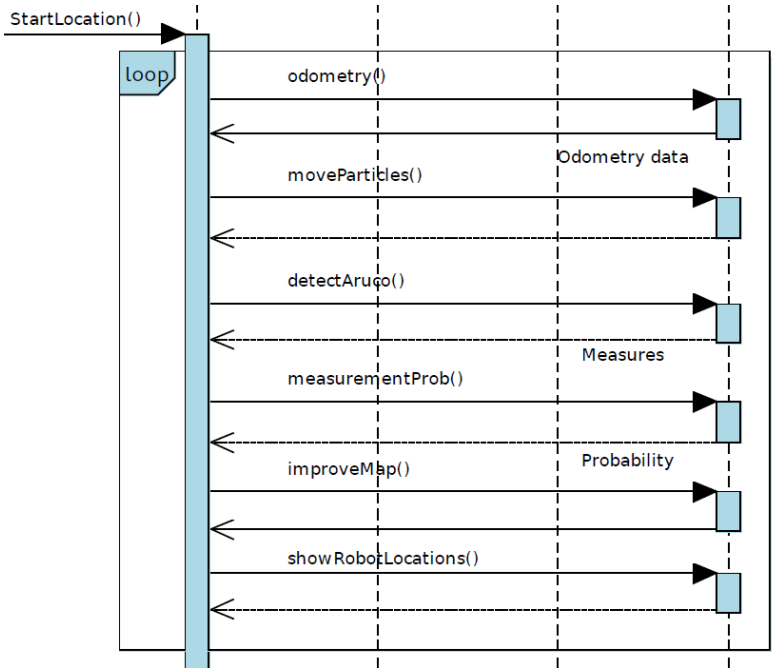

Figura 58: Detalle diagrama de secuencia location.

8.6.4.3 Detalle diagrama de secuencia serial read

Este hilo se encargará de recibir cíclicamente la información proveniente de la comunicación serie.

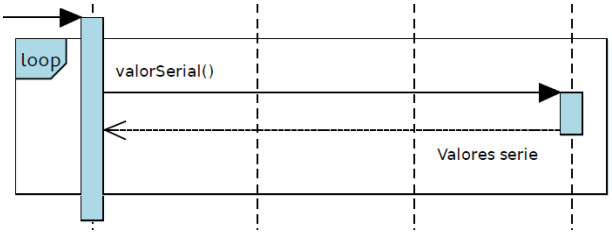

Figura 59: Detalle diagrama de secuencia serial read.

8.6.4.4 Detalle diagrama de secuencia Motion

Esta hilo se encargará de la gestión del láser y su detección, movimientos, planificación de trayectorias y creación del mapa.

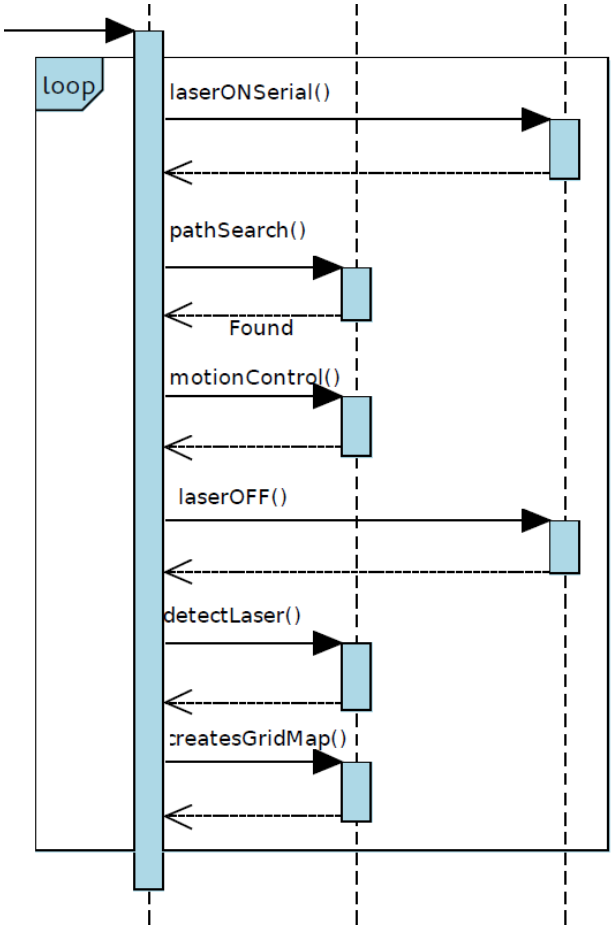

Figura 60: Detalle diagrama de secuencia Motion.

8.6.4.5 Detalle diagrama de secuencia Main

El hilo principal será el encargado de actualizar la visualización

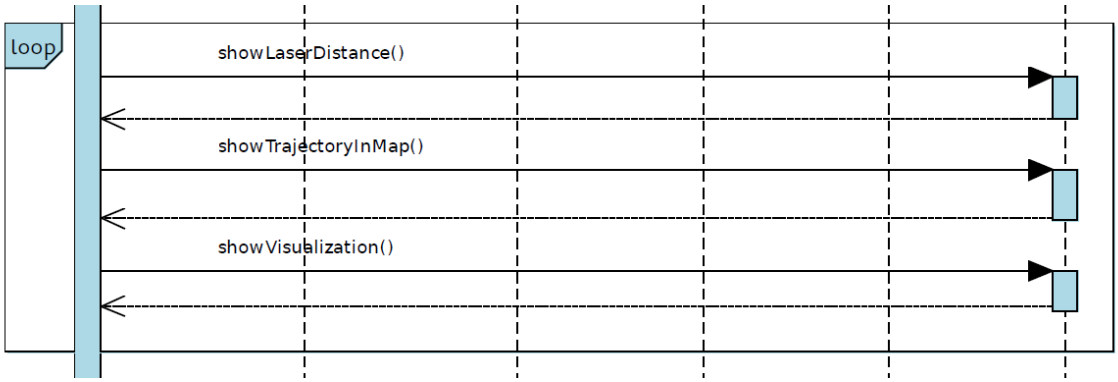

Figura 61: Detalle diagrama de secuencia Main.

9 Resultados

Para evaluar los resultados se van a analizar cada una delas partes del trabajo:

9.1 Detección de marcadores

La detección de marcadores es muy robusta en posicionamiento, mientras que en orientación la medida tiende a fluctuar más, pero con la solución aportada y descrita anteriormente, que consiste en utilizar solo la posición y solo el giro en "Z" el resultado conseguido es muy bueno.

Como pudimos ver durante el proceso de puesta a punto, al realizar esta simplificación cobra especial importancia que la cámara este paralela al techo, ya que si el objetivo no apunta de manera perpendicular al techo, al girar la cámara sobre su eje el sistema incorrectamente interpretará que el robot se ha desplazado. Al ser este proceso de montaje y ajuste crítico de cara a la precisión del posicionamiento se ha creado una aplicación para facilitarlo.

El sistema de clasificación de marcadores para determinar su validez en función del error acumulado evita detecciones incorrectas que provocarían errores en la medida.

Este paso de filtrado y estabilización de la medida de los marcadores es especialmente necesario en este sistema, ya que al ir relacionando posiciones de marcadores consecutivamente cualquier error de la cadena permanecerá en los siguientes marcadores incrementado a su vez el error global.

Esta manera de trabajar provocará que el error sea mayor conforme nos alejemos de las inmediaciones del origen. Esto se puede decir es el contrapunto a la comodidad y versatilidad de poder poner los marcadores de manera aleatoria.

Tal vez una solución en líneas futuras fuese estudiar cual es el límite operativo estimado en cuanto a distancia desde el origen, y una posible solución fuese que algunos marcadores intermedios diesen la posición absoluta reduciendo el error antes de llegar al límite.

Además con la iluminación frontal de la cámara se consigue ampliar el rango de trabajo, ya que en condiciones de baja iluminación el sistema falla en la detección. Para trabajar con oscuridad total bastaría con implementar un sistema de iluminación más potente. Un problema con peor solución ocurre cuando el robot se encuentra en su camino con un foco que lo ilumina directamente, al ser las cámaras colocadas auto exposición la imagen se satura dificultando la detección de los objetos cercanos.

Se han hecho las siguientes pruebas para cuantificar los errores:

Robot desplazado 1000mm en "X" desde el punto de origen:

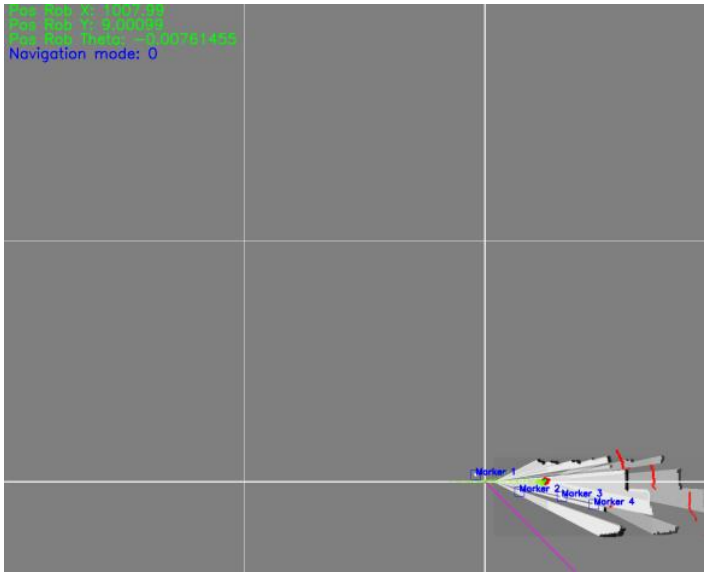

Figura 62: Robot desplazado 1000mm en "X" desde el punto de origen.

Robot desplazado 2000mm en "X" desde el punto de origen:

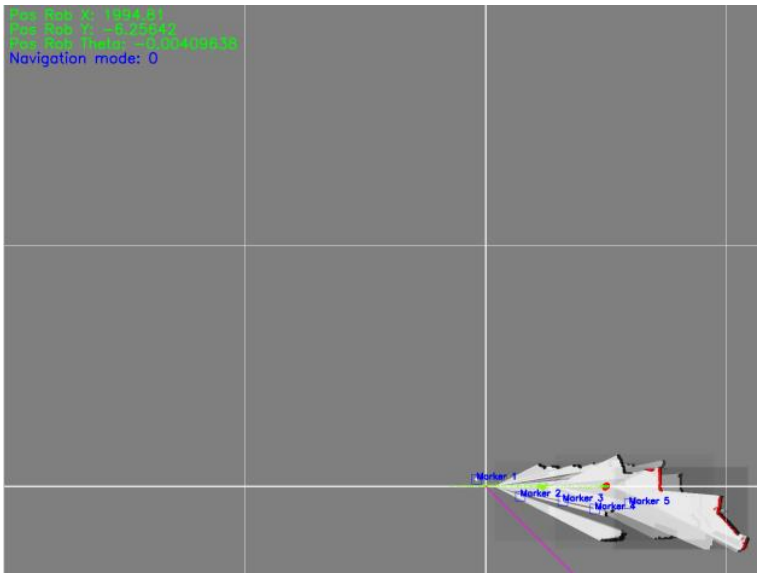

Figura 63: Robot desplazado 2000mm en "X" desde el punto de origen.

Robot desplazado 3500mm en "X" desde el punto de origen:

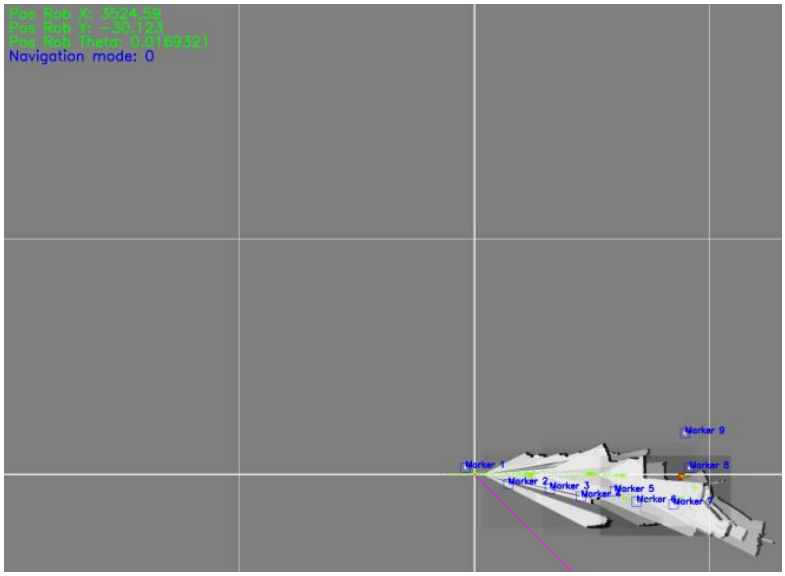

Figura 64: Robot desplazado 3500mm en "X" desde el punto de origen.

Robot desplazado 3500mm en "X" y 3000mm en "Y" desde el punto de origen:

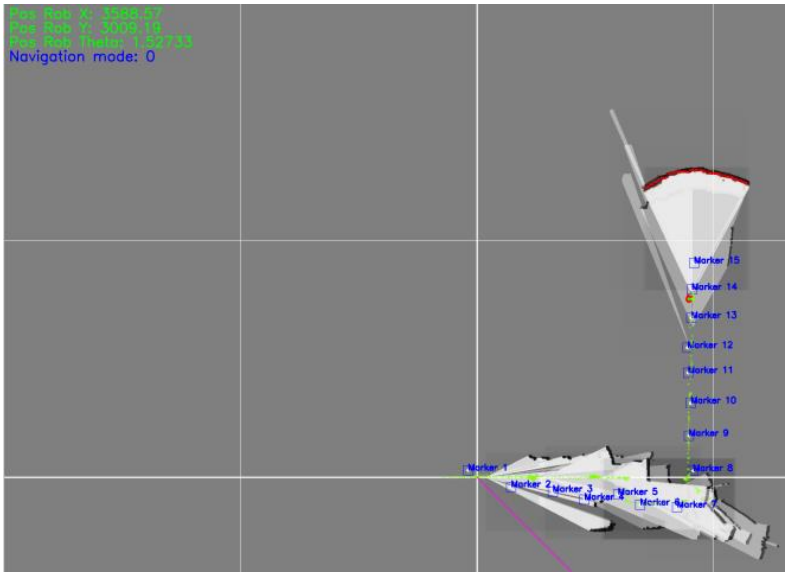

Figura 65: Robot desplazado 3500mm en "X" y 3000mm en "Y" desde el punto de origen.

Como vemos en las imágenes el error máximo en el área desplazada es de unos 10cm. Este error en la localización es admisible para esta aplicación, en la que se busca que el robot cree un mapa topológicamente correcto, no un mapa preciso de medida.

9.2 Filtro de partículas

El filtro de partículas cumple perfectamente con la tarea de localización del robot. Tanto la parte de asignar pesos a cada partícula en función de la distancia a las medidas como el muestreo o resampling consiguen seleccionar las partículas más cercanas a la posición del robot, estando todas las partículas después de unos ciclos cerca de la partículas con más peso.

En la siguiente imagen se han hecho ampliaciones para que podamos ver: en el recuadro blanco las medidas dadas por los marcadores respecto a la posición del robot , en verde la posición dada por el filtro de partículas y correspondiente a la partícula con más peso, y la zona ampliada del mapa de los marcadores que está detectando el robot.

Además el filtro sirve y cumple con la tarea de fusionar la información de diferentes sensores como son las medidas obtenidas de la odometría y las obtenidas a partir de la posición de los marcadores para determinar la posición del robot.

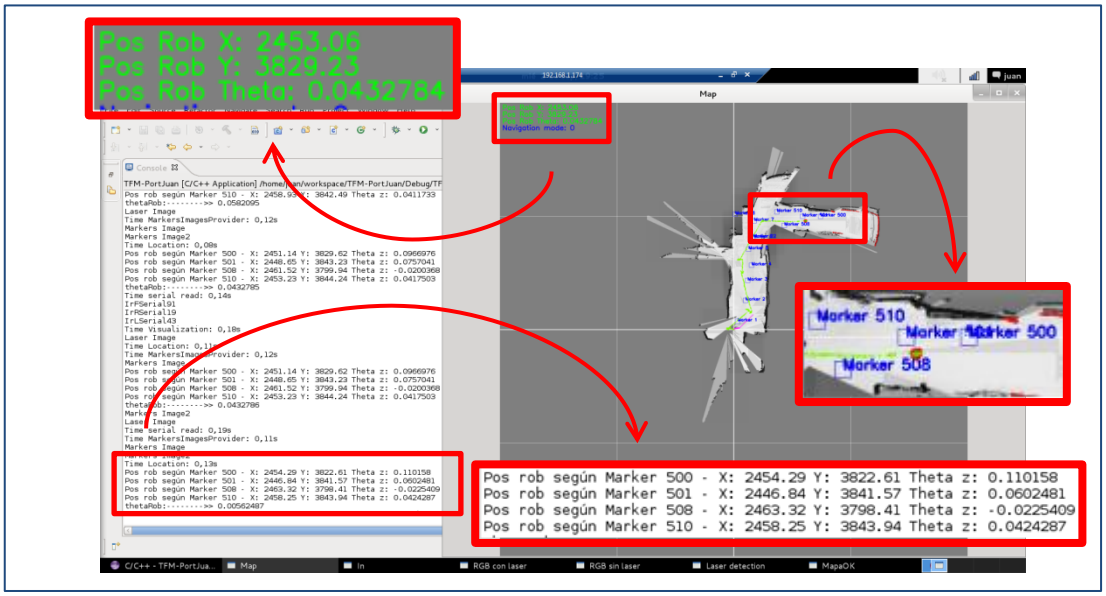

Figura 66: Detalle detección, medidas marcadores y posición según filtro de partículas.

Además al modelar un ruido variable del filtro de partículas en función de la diferencia de posición entre ciclos conseguimos alcanzar la posición de manera ágil sin provocar una dispersión de partículas alta cuando la medida es estable.

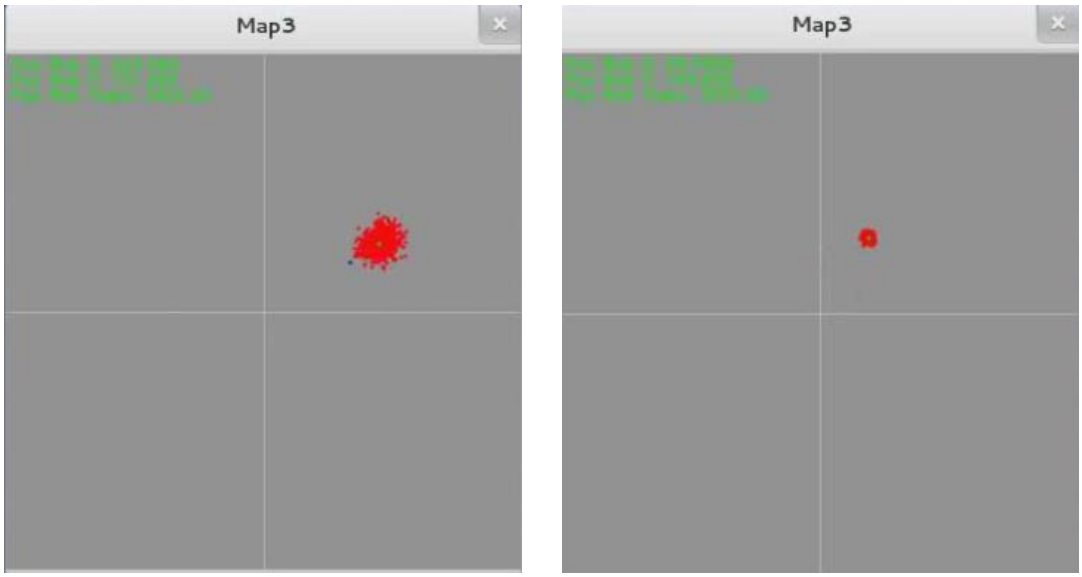

Figura 67: Dispersión alta de partículas y dispersión baja según ruido variable.

9.3 Odometría

La odometría del robot es válida en esta aplicación como ayuda a la localización global al desplazar las partículas en función de los movimientos del robot. El principal inconveniente para un posicionamiento correcto basado solo en la odometría es que al ser "Skid steer" el tipo de locomoción del robot es decir las 4 ruedas son de tracción, el robot para girar debe deslizar o derrapar provocando diferencias en la medida con respecto a la realidad. Esto sumado a posibles diferencias al radio entre ruedas respecto al modelado hace que como sistema único de medición no sea suficiente, si bien en conjunto a otros sistemas como el de marcadores en este caso mejora la detección en cuanto a respuesta o rapidez.

En la siguiente imagen podemos ver la posición estimada del robot en base a la odometría tras haberlo movido un metro. También vemos como al no haber detección de marcadores los números de la posición aparecen en rojo y la dispersión de las partículas aumenta.

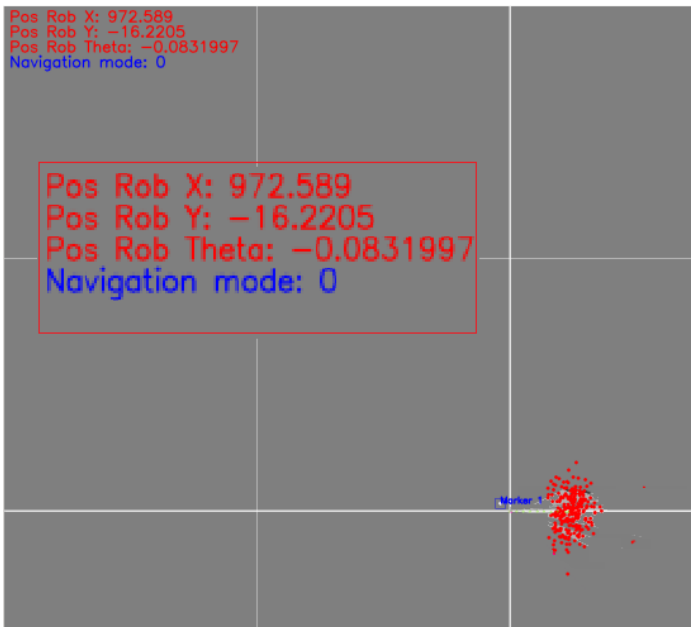

Figura 68: Partículas desplazadas 1000mm únicamente con la odometría.

9.4 Detección láser

El sistema de medición láser basado en un puntero láser de línea y una cámara de visión artificial ha dado muy buenos resultados, consiguiendo hacer mediciones del entorno que permiten crear un mapa coherente con el que el robot puede navegar.

Por otro lado han sido detectados casos en los que el sistema produce errores o mediciones incorrectas como son:

- Diferencias en la reflexión del material sobre el que proyectamos el haz provocan en ocasiones medidas erróneas en este sistema.
- Material traslucido como una puerta de cristal.

Estos dos casos comentados anteriormente los podemos ver en las siguientes imágenes, podemos ver como el láser no es detectado en la puerta porque prácticamente es inapreciable en la imagen, y como en la zona del radiador debido al excesivo reflejo del haz láser la detección es defectuosa:

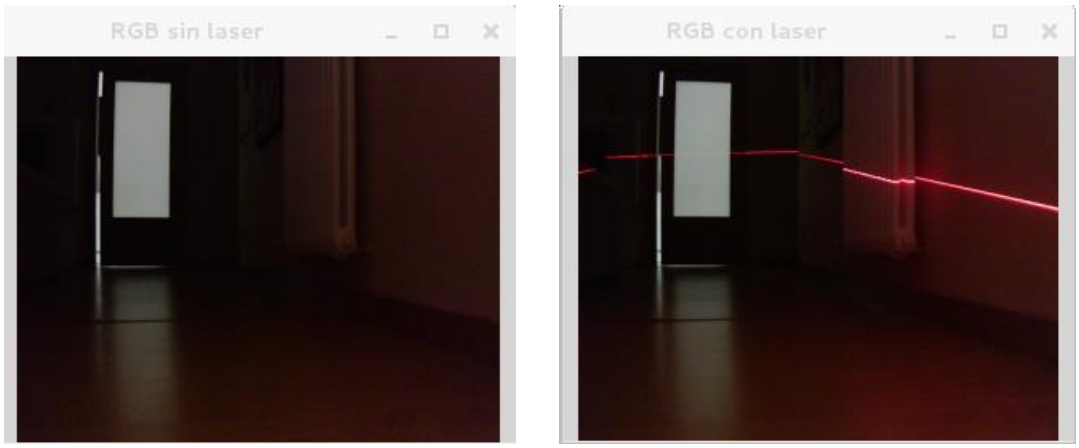

Figura 69: Detección láser errores.

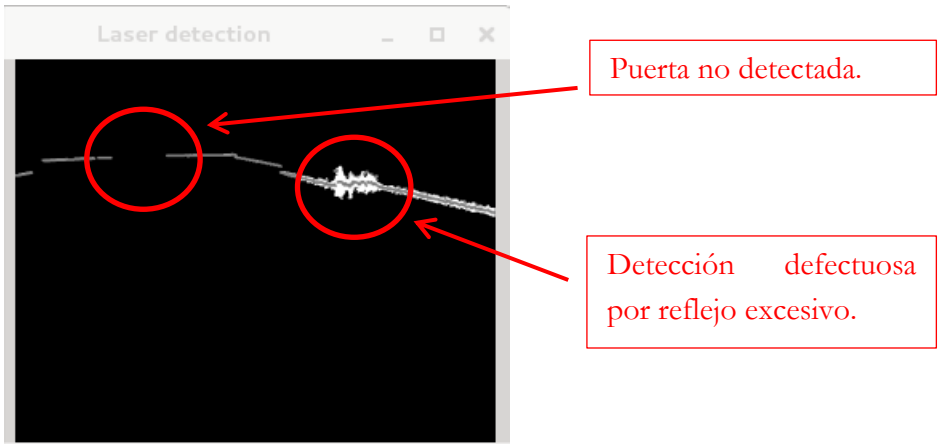

Figura 70: Detección láser errores.

El efecto del láser seguramente podría ser minimizado procesando la imagen de otra manera para extraer el mismo o haciendo un tratamiento local por zonas de la imagen en vez de un tratamiento global, pero el problema de la puerta tiene peor solución ya que en la imagen es indetectable.

Podemos ver en la imagen siguiente como afectan estos dos errores en proceso de creación de mapas.

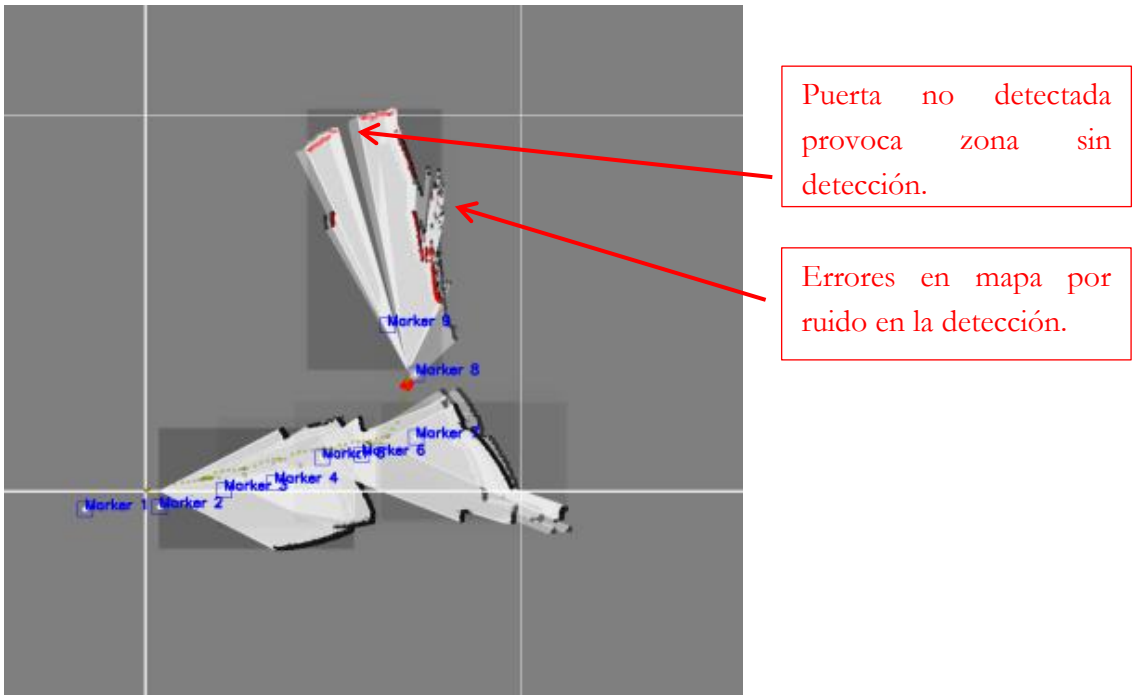

Figura 71: Repercusión de medidas erróneas en la creación de mapas.

En las siguientes imágenes vemos otros dos casos de medidas erróneas provocadas por la reflexión de luz excesiva

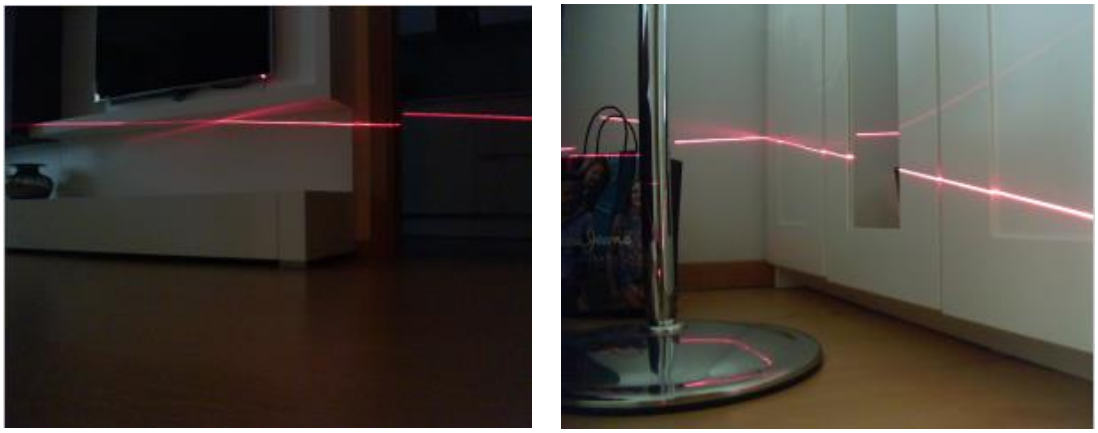

Figura 72: Medidas erróneas provocadas por la reflexión de la luz.

Pese a estos errores se puede decir que el resultado global de la medición basada en luz estructurada es satisfactorio, especialmente si tenemos en cuenta que con un sistema de láser más cámara valorado en unos 30 euros conseguimos en condiciones normales obtener la geometría de los elementos del entorno de manera aceptable. Es importante por tanto ser consciente que un sensor que nos permitiese obtener de manera robusta y precisa la geometría de los objetos del entorno sería un medidor láser de distancia o LIDAR cuyo coste se encuentra entorno a los 1500 euros.

9.5 Mapping

La creación del mapa mediante la técnica de ocupación de celdas basada en la teoría bayesiana a partir de la posición dada por los marcadores y las medidas obtenidas por el sistema láser-visión ha sido positiva, como podemos ver en las siguientes imágenes

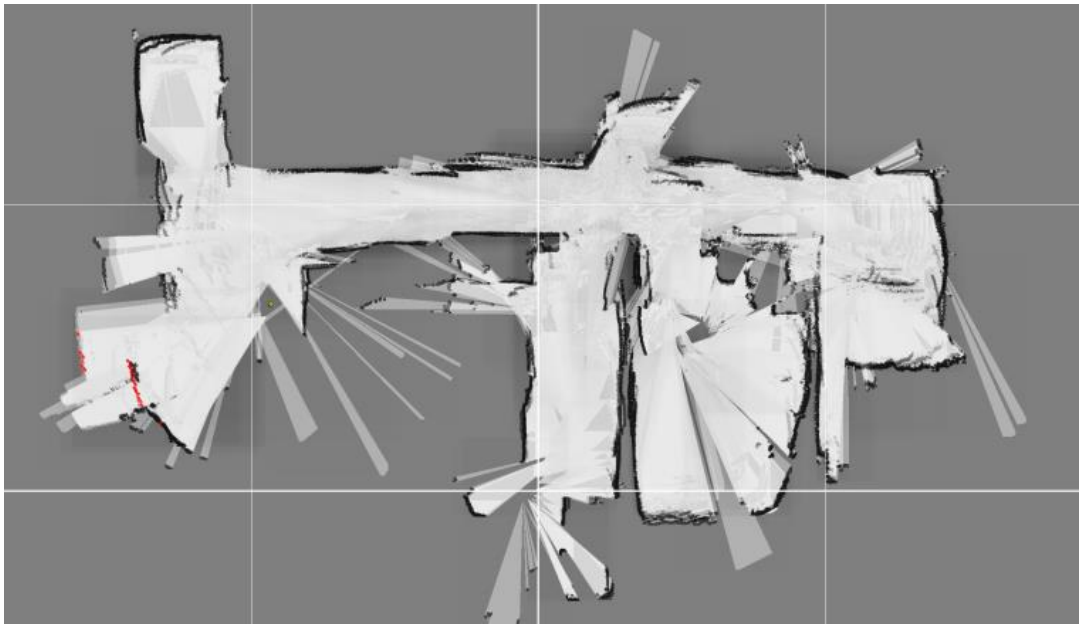

Figura 73: Ejemplo Mapa 1.

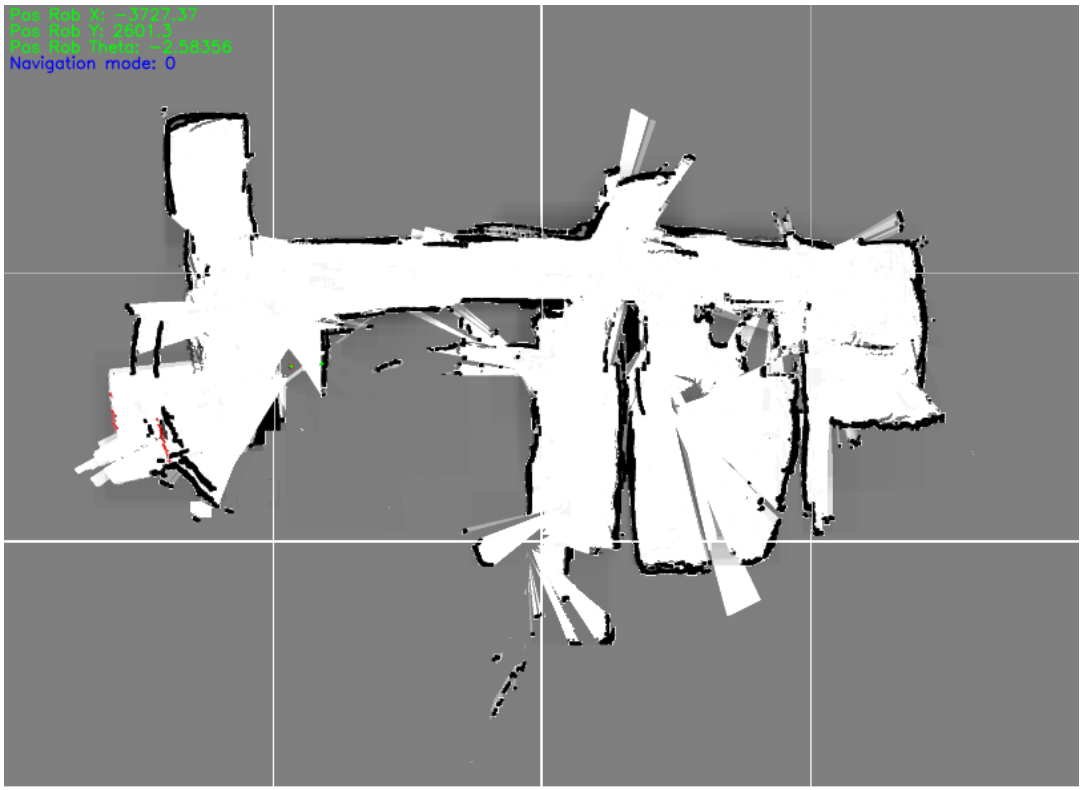

Figura 74: Ejemplo Mapa 1 mejorado por la aplicación.

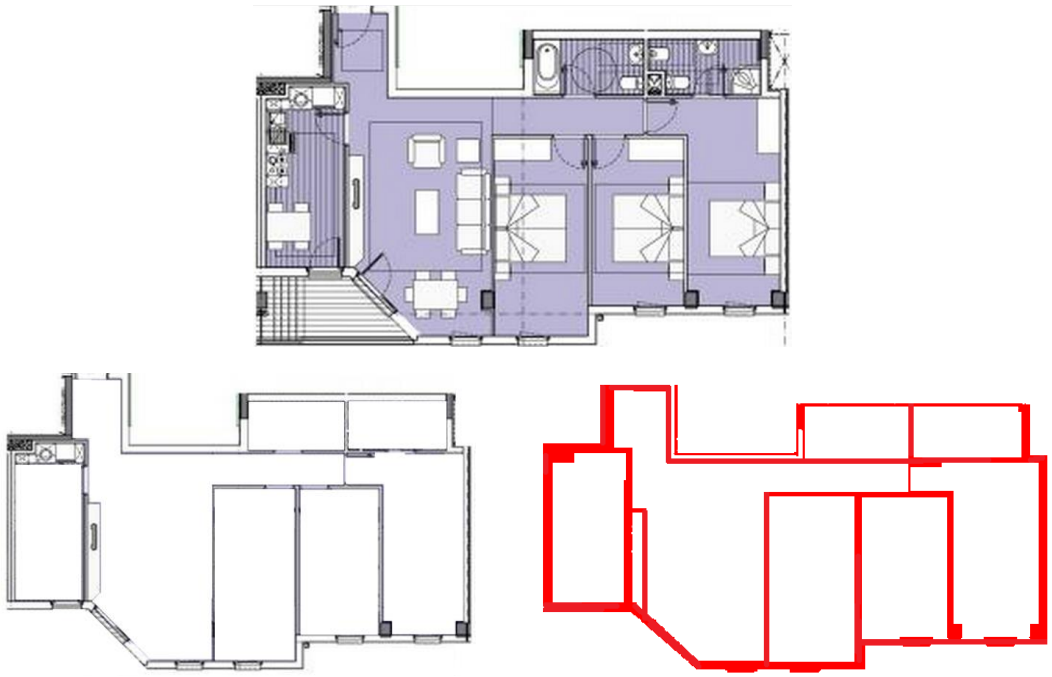

Figura 75: Plano real y plano real tratado.

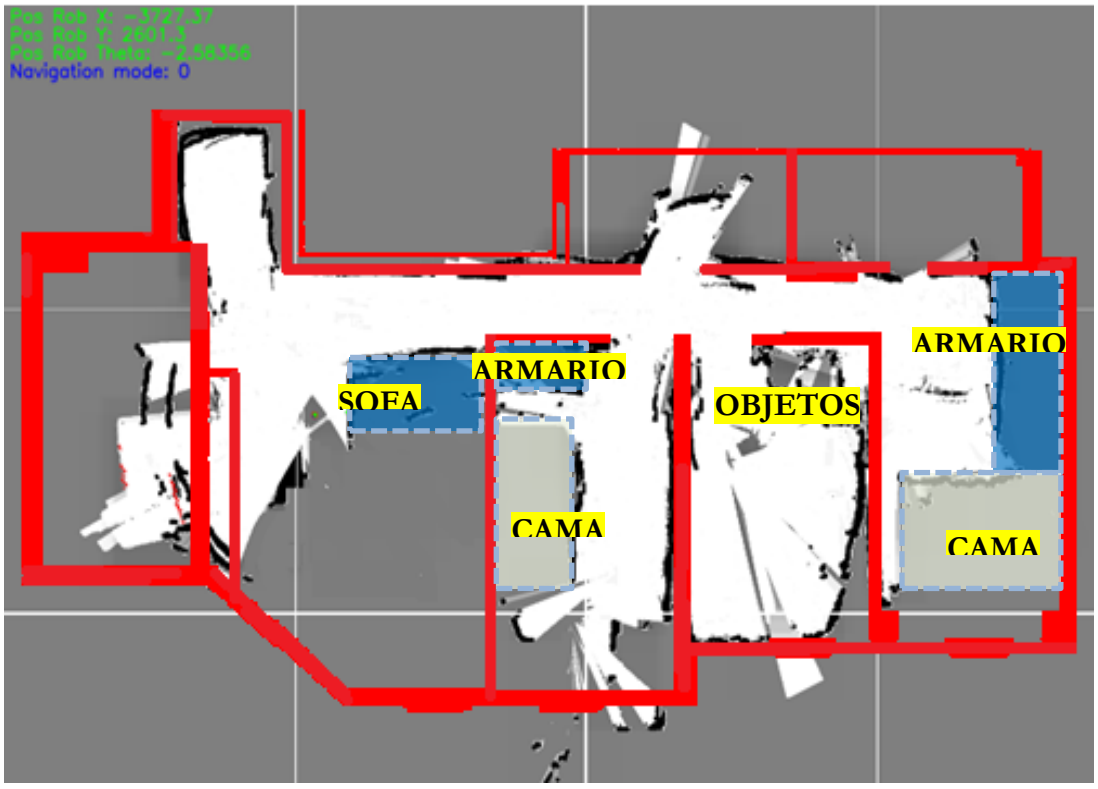

Figura 76: Plano obtenido y plano real superpuesto, detalle objetos.

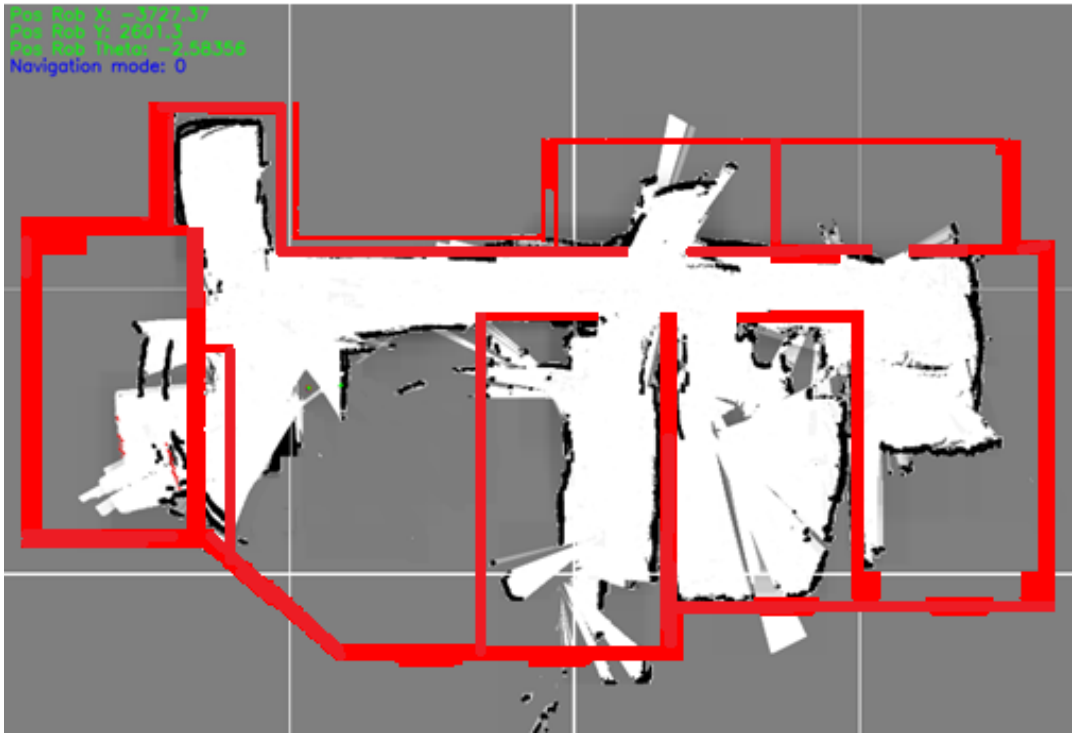

Figura 77: Plano obtenido y plano real superpuesto.

Aunque no sea significativo en los mapas anteriores, el incremento de error en el posicionamiento por balizas podría llegar a provocar la generación de mapas que no se correspondan con la realidad (Dimensiones o formas erróneas), pero al estar pensado el mapa para que el robot navegue sin colisionar con el entorno no será un problema ya que esa será su forma de ver el mundo, como decíamos en el apartado de mapping lo importante es que el mapa sea topológicamente correcto.

9.6 Navegación

Los diferentes modos de navegación implementados en el robot permiten al robot ser teleoperado, recorrer las balizas incrementalmente, navegar de manera aleatoria o navegar hacia una zona desconocida.

Cabe destacar que para evitar colisiones o atascos se han implementado las siguientes mejoras:

- El sistema de medición-láser tiene una zona muerta delante del robot de unos 200 mm, además el robot no sabe si tiene obstáculos laterales, es por esto por lo que se han añadido unos sensores de distancia infrarrojos que permiten en los laterales y en el frontal. Cuando el robot detecte presencia en uno de estos detectores hará un movimiento para evitar el obstáculo.

- En los modos de navegación aleatoria o navegación hacia una zona desconocida el robot ante un bloqueo (Sin pulsos de encoder en ambas ruedas) hará un movimiento de evasión para poder reanudar su operación normal.
- En el caso de navegación hacia una zona desconocida si el robot no se ha movido 1 metro en los últimos 2 minutos cambia el punto de destino.

Estos modos tratan de garantizar en la medida de lo posible la operación normal del robot.

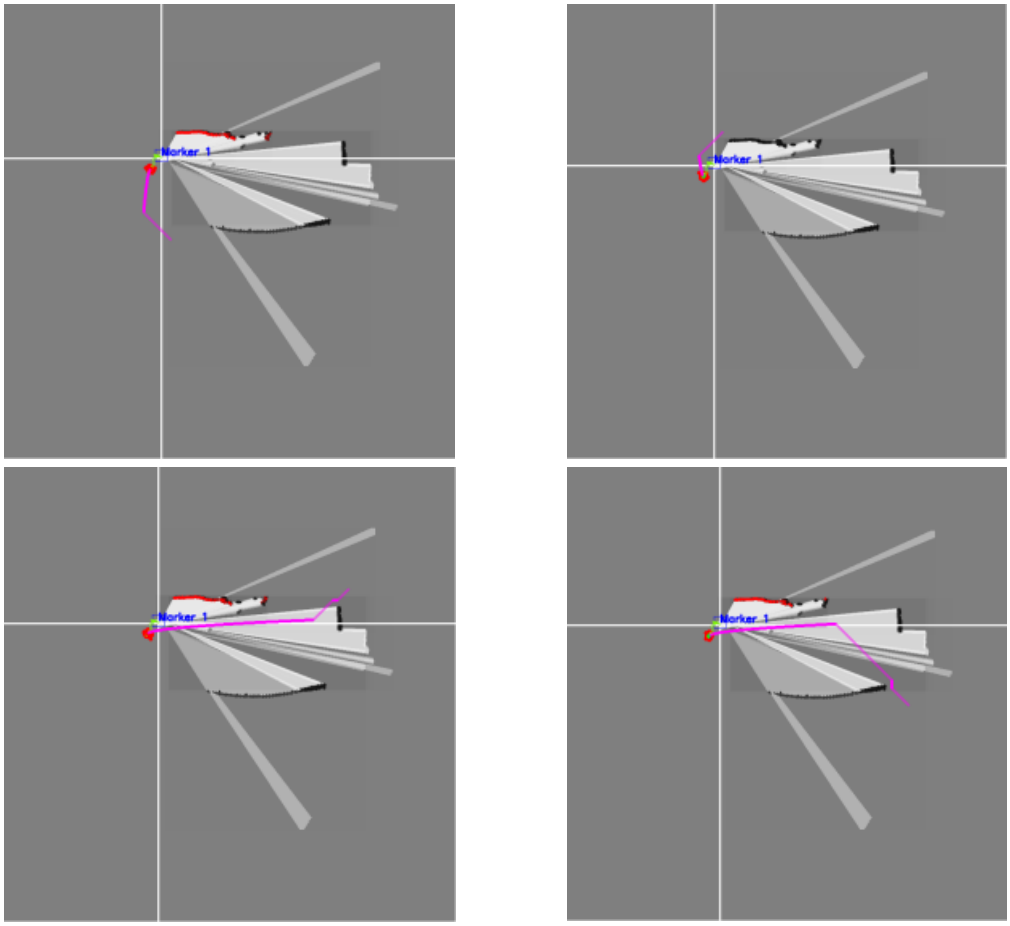

Figura 78: Cambio de trayectoria (rosa) a los cuatro vértices de la zona explorada al no moverse el robot al menos un metro durante 2 minutos.

Figura 79: Detalle de sensores infrarrojos lateral y frontal.

Por otro lado la planificación de trayectorias con el algoritmo A* cumple la tarea de búsqueda de manera eficiente. Inicialmente se había implementado un algoritmo de búsqueda basado en los pasos dados hasta llegar al objetivo, pero el implementar la función A* provocó una mejora de rendimiento notable.

9.7 Estructura de control

La estructura de control del robot ha sido suficiente para la aplicación, donde la navegación se basa en movimientos incrementales hacia adelante, atrás, derecha o izquierda.

En el desarrollo del proyecto se intentó implementar un control continuo cosa que no fue posible debido a que el sistema de control en conjunto no era capaz de funcionar a la rápida velocidad de los motores de las ruedas.

Para conseguir el objetivo de un control continuo se debería trabajar en los siguientes puntos:

- El robot debería ser capaz de moverse más lento. En el montaje actual a la tensión que empiezan a moverse los motores el robot recorre mucha distancia, esto va en contra de la controlabilidad del sistema.
- Tratar de que la comunicación PC-Arduino fuese más fluida. El Pc debería de ser capaz de detectar los marcadores, aplicar el filtro de partículas, aplicar odometría a las partículas, recalcular trayectoria y enviar información al Arduino en un tiempo que asegurase la capacidad de control del robot. Todo esto teniendo en cuenta que al mismo tiempo se hacen otras tareas como la creación del mapa o detección láser. Los requerimientos podrían llevarnos a pensar en algún tipo de estructura descentralizada.
- Como punto crítico destacar que con una velocidad lineal rápida los marcadores no son detectados por lo que el sistema no puede sobrepasar cierta velocidad si quiere seguir detectando su posición en base a los marcadores.

Cabe destacar que esto no fue un inconveniente para la ejecución de este trabajo ya que para detectar el láser basándonos en la diferencia de una imagen con láser y otra sin láser el robot tiene que estar estático cada vez que deseemos hacer un nuevo muestreo.

9.8 Diseño software

En el diseño software se ha tratado dar una estructura coherente con la tarea que se estaba desarrollando, es por esto por lo que se decidió separar la ejecución de las diferentes partes de código en diferentes hilos o threads, para aprovechar al máximo el

doble núcleo y la multitarea del PC con el objetivo de obtener tiempos de control lo más reducidos posible.

Al separar en hilos además establecemos que hilos se ejecuten con más frecuencia que otros, por ejemplo la visualización es el hilo más lento.

Además de separar en diferentes hilos o threads se tomaron diferentes medidas para reducir los tiempos de las diferentes tareas.

- El hilo de visualización y refresco de imágenes en pantalla es el que se ejecuta con menos frecuencia. Ofrece una visualización agradable al usuario sin penalizar rendimiento.
- El hilo de localización es el que más se ejecuta por ser la tarea más crítica. Un posicionamiento erróneo producirá un control y un mapping erróneo.
- Se redujo el número de partículas de 1000 a 500.
- El mapping era una tarea especialmente pesada y hubo que:
    - Reducir el número de medidas láser a 160 en 53º, la sobrecarga de medidas era innecesaria a costa de un elevado coste computacional.
    - Procesar solo la zona del mapa afectada por la nueva medición
    - El mapa es de máximo 2000x2000 puntos que ahora representan 16 metros x 16 metros, cuanto menor es la distancia que representan, más píxeles serán necesarios procesar después de una medida láser. Esto es porque una misma área representa más píxeles o menos según la escala utilizada.
    - Limitar la máxima detección del láser en el mapa, limitando el alcance máximo de representación en el mapa en 3 metros evitamos incrementar el coste computacional descontroladamente en zonas muy amplias. Se han elegido 3 metros por que es la distancia a partir de la cual el sistema de medida láser-visión empieza a ser menos preciso.

Los tiempos de control medios obtenidos para los diferentes hilos son:

```
Time serial read: 0,18s
Time Location: 0,07s
Time MarkersImagesProvider: 0,09s
Time Visualization: 0,24s
Time Motion: 2,89s
Time laser detection and mapping: 0,93s
```

Figura 80: Tiempos de control de los diferentes hilos.

Vemos que el tiempo más largo es el de la tarea "Motion", esto es porque contiene la detección láser y mapping (0,93s); la planificación de trayectorias; los movimientos del robot; y la activación desactivación láser.

El tiempo de ciclo de Arduino son 500 ms, por lo que activar/desactivar el láser más atender la orden de movimiento será aproximadamente 1,5 segundos.

10Descripción de la aplicación

En las siguientes páginas, quedan descritos los diversos modos y utilidades de la aplicación.

Al arrancar la aplicación, el sistema pedirá al usuario que introduzca el número de dispositivo para cada una de las cámaras (Siguientes imágenes). En Linux a cada dispositivo de video conectado al bus le corresponde un número, este puede variar dependiendo del orden de conexión, razón por la que la aplicación pide introducirlo en cada arranque. Si no sabemos cuál es el número podemos utilizar la aplicación "cheese" de Linux para confirmarlo.

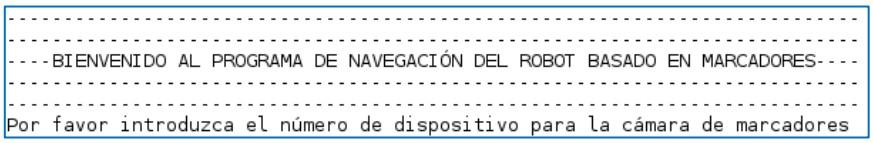

Figura 81: Introducir número cámara marcadores.

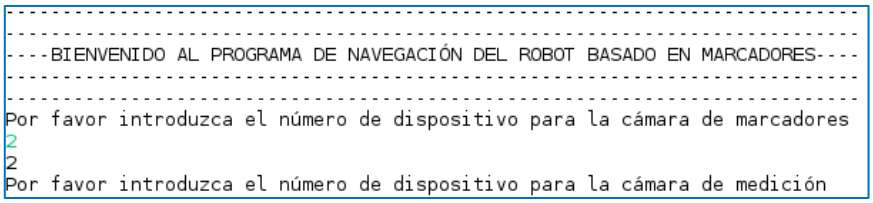

Figura 82: Introducir número cámara láser.

Una vez introducidos estos datos la aplicación nos mostrará las instrucciones y esperará hasta que pulsemos "ENTER" para comenzar.

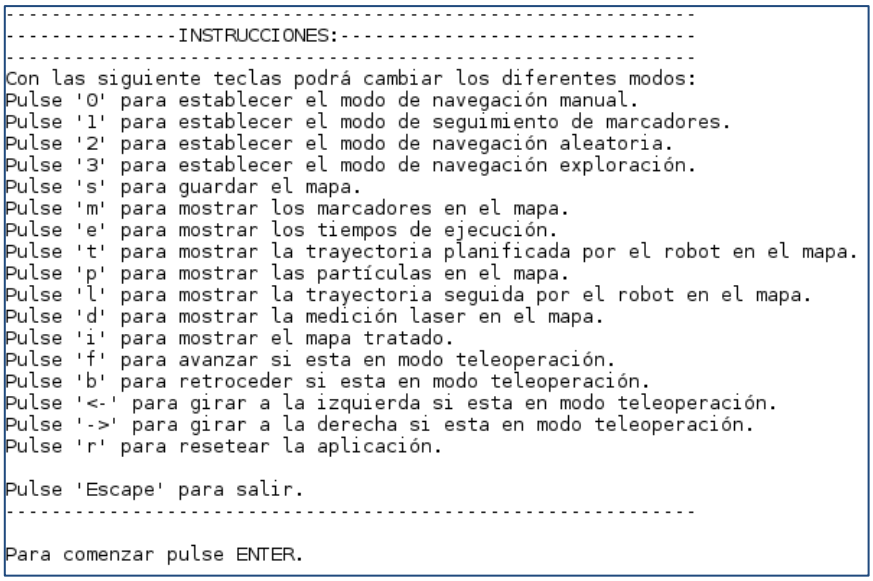

Figura 83: Instrucciones de la aplicación mostradas al arrancar.

El programa al arrancar mostrará por defecto todas las opciones de visualización. En la esquina superior izquierda nos mostrará la posición actual del robot, cuando este en verde significará que detecta al menos un marcador y en rojo que no detecta marcadores.

Por defecto el programa empezará en el modo de navegación manual y el usuario podrá cambiar entre los 4 modos pulsando del 0 al 3, siendo "0 – Navegación manual", "1 – modo seguimiento de marcadores", "2 – modo de navegación aleatoria" y "3 – modo de exploración" como vemos en la imagen anterior. El estado actual lo podemos confirmar en la esquina superior izquierda en azul.

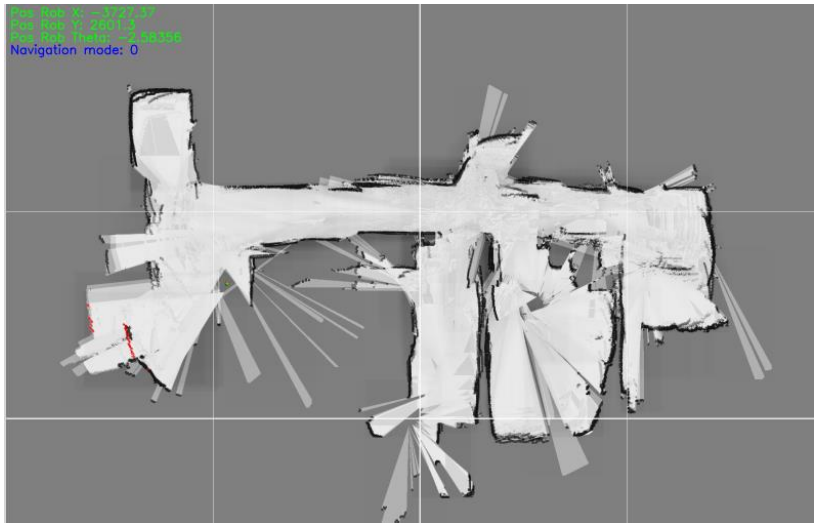

Figura 84: Mapa, en la esquina superior izquierda vemos en azul el modo de navegación actual.

Las demás funcionalidades sirven para:

"s": Guardará el mapa actual en la ruta de la aplicación.

"m": Mostrará los marcadores en el mapa.

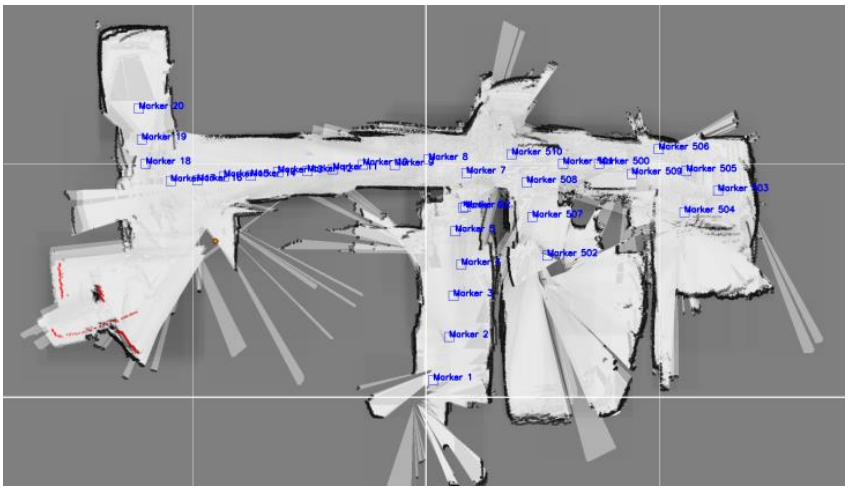

Figura 85: Marcadores en azul en el mapa.

“e”: Mostrará los tiempos de ejecución de las diferentes tareas en la consola.

```
Time Location: 0,09s
Markers Image
Markers Image2
Laser Image
Time MarkersImagesProvider: 0,08s
Time Visualization: 0,21s
Time Location: 0,11s
Markers Image
Markers Image2
Laser Image
Time serial read: 0,21s
Time MarkersImagesProvider: 0,13s
Time Location: 0,09s
Markers Image
Markers Image2
Laser Image
Time MarkersImagesProvider: 0,06s
Time Visualization: 0,19s
Time Location: 0,08s
```

```
Time serial read: 0,09s
IrFSerial90
IrRSerial22
IrLSerial45
Markers Image
Markers Image2
Laser Image
Time MarkersImagesProvider: 0,14s
Time Location: 0,12s
400-f
Time serial read: 0,17s
IrFSerial82
IrRSerial21
IrLSerial44
Markers Image
Markers Image2
Laser Image
Time Location: 0,09s
```

Figura 86: Tiempos de ejecución de las diferentes tareas en la consola.

“t”: Mostrará la trayectoria planificada por el robot en el mapa.

“p”: Mostrará las partículas en el mapa.

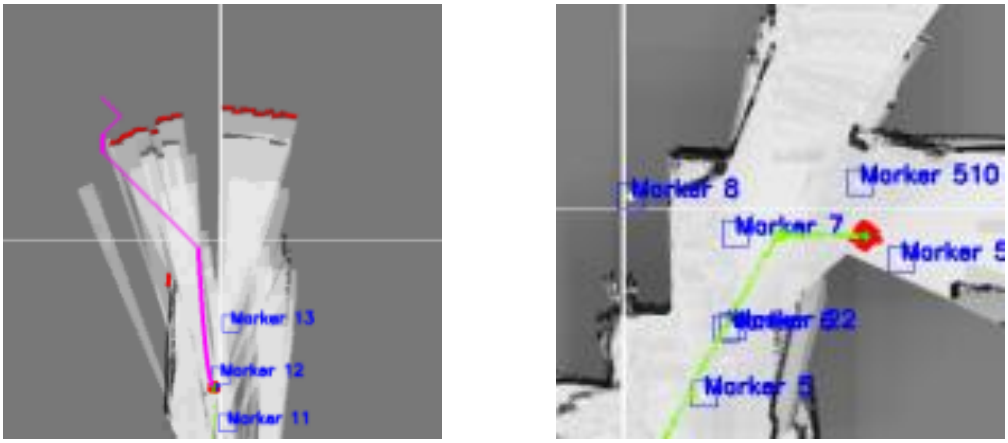

Figura 87: Izquierda trayectoria planificada por el robot en rosa, derecha partículas en rojo en mapa.

“k”: Mostrará la trayectoria seguida por el robot en el mapa.

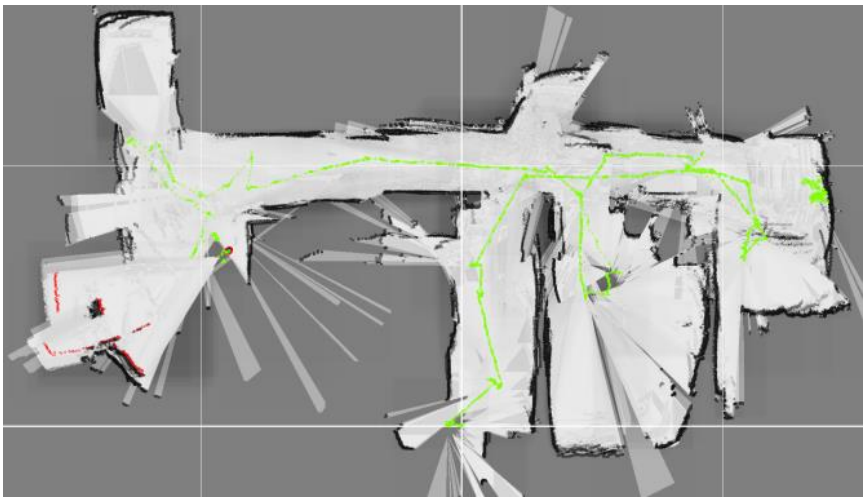

Figura 88: Trayectoria seguida por el robot en verde.

“k”: Mostrará la medición láser del robot en el mapa.

“i”: Mostrará el mapa tratado.

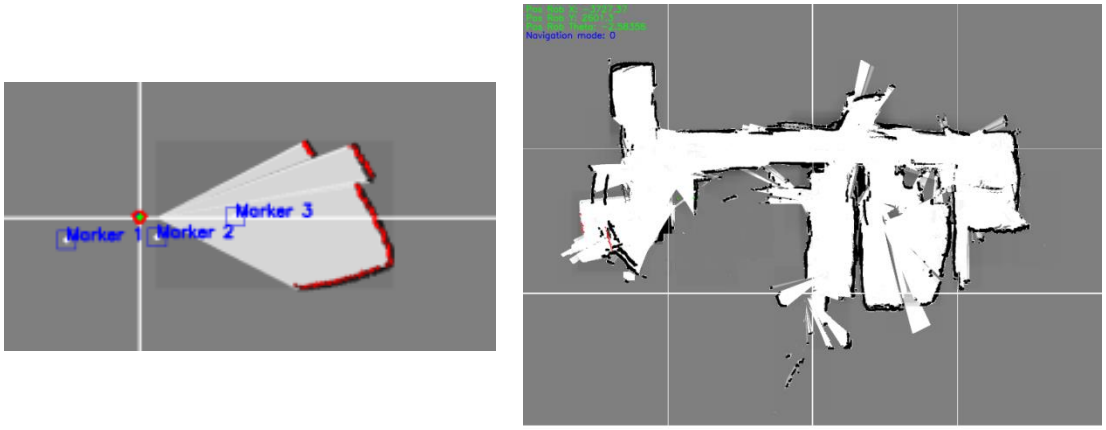

Figura 89: A la izquierda medición actual del láser en rojo, a la derecha mapa tratado.

“r”: Para resetear la aplicación.

“Esc”: Para salir.

A continuación un ejemplo del mapa completo con todas las visualizaciones.

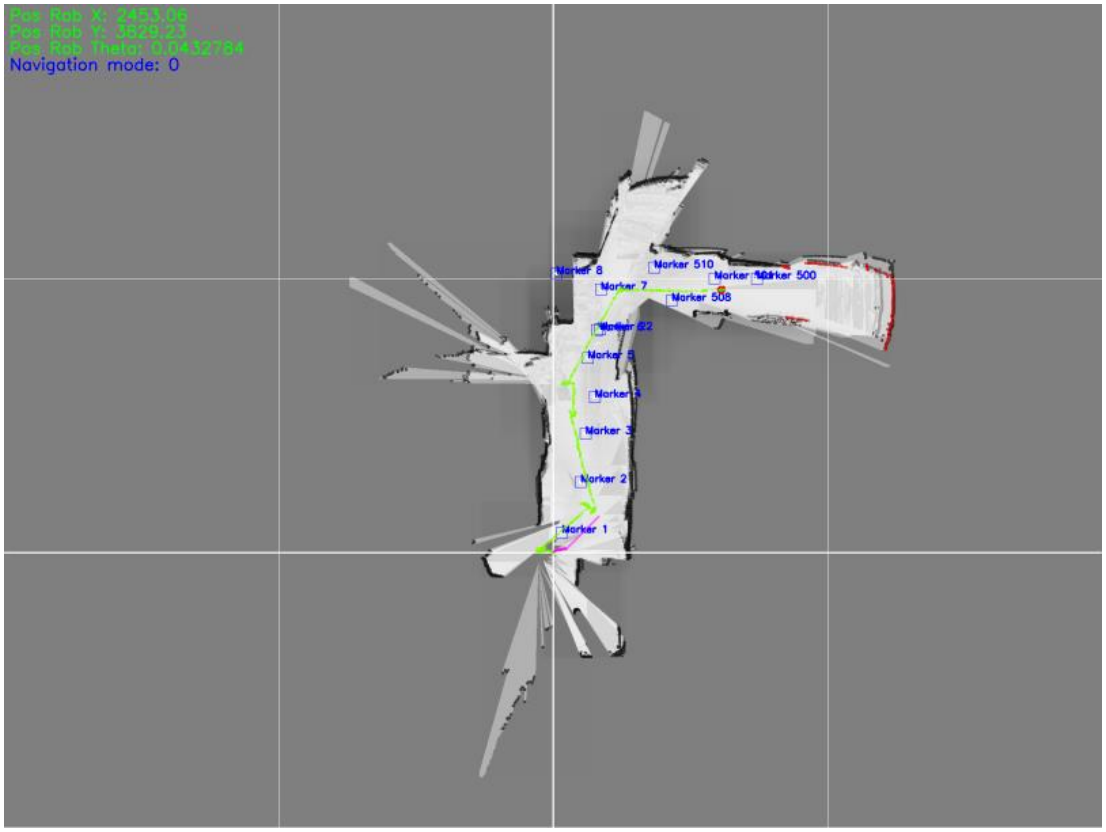

Figura 90: Todas las visualizaciones.

Además la aplicación nos mostrará la imagen de la cámara de los marcadores, y las imágenes de la detección láser (Con láser, sin láser y detección).

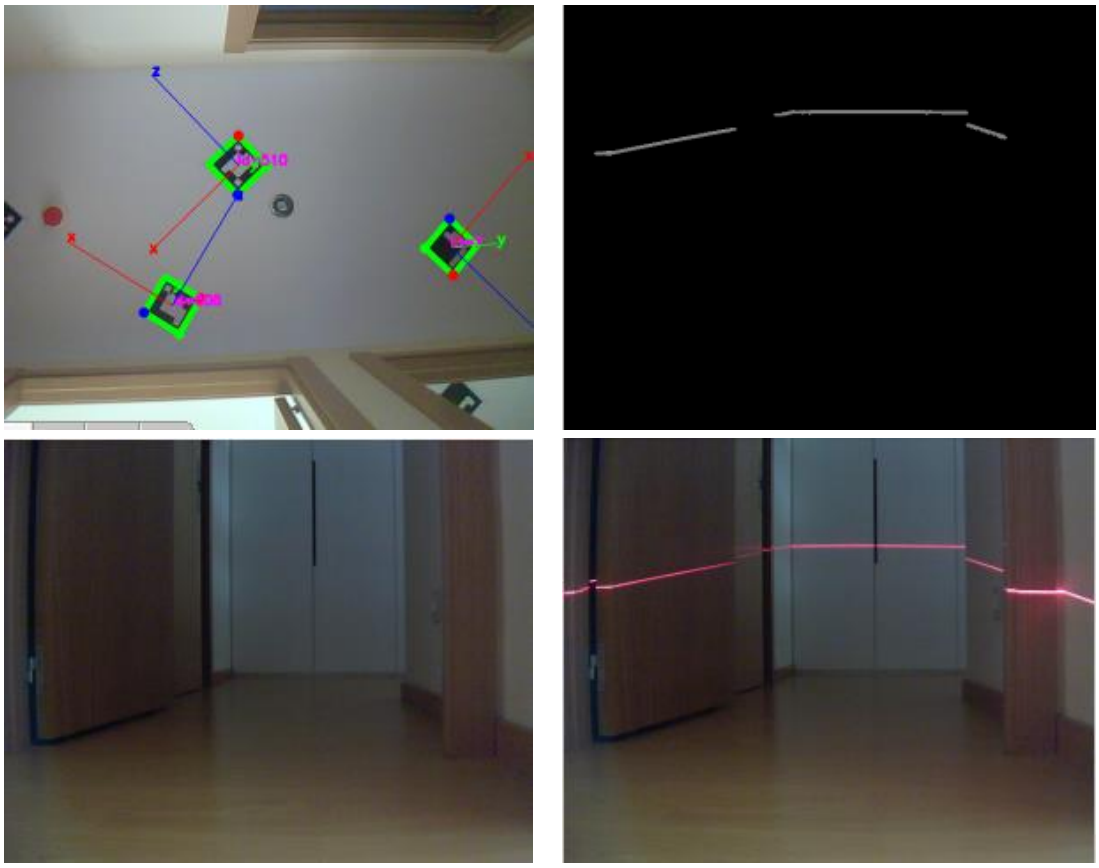

Figura 91: Imágenes cámara marcadores, cámara frontal con láser, cámara frontal sin láser, detección láser.

CONCLUSIONES

11 Conclusiones y líneas futuras

11.1 Conclusiones

En este trabajo se ha tratado de dar una solución global al problema de localización, navegación y mapping de un robot autónomo pensado para trabajar en interiores.

Para llegar a esta solución global ha sido especialmente necesario conocer el estado del arte. Es por esto que se han podido dar soluciones conocidas como el filtro de partículas o el mapa de ocupación basado en la teoría bayesiana.

Así mismo para la localización y la medición del entorno, se han ofrecido soluciones basadas en visión artificial; en el caso de la localización basada en marcadores se ha conseguido una solución robusta a la vez que sencilla para el usuario; por otro lado, con el sistema de medición basado en el conjunto láser-cámara conseguimos tener medidas de la geometría del entorno del robot a un coste muy bajo.

Los diferentes modos de navegación del robot ofrecen desde la forma más sencilla de mover un robot como es la tele-operación a la función de exploración de zonas desconocidas. Es de especial interés el modo de seguimiento de marcadores, modo en el que el usuario marcará la ruta del robot con el posicionamiento de los mismos.

El desarrollo del código se ha hecho en una estructura basada en la multitarea, con el fin de aprovechar al máximo los beneficios de disponer de un procesador de varios núcleos. Esto implica el uso de herramientas software para la sincronización y control de tareas, pero a su vez ofrece un rendimiento mejorado frente a un proceso monotarea. Podemos destacar que el rendimiento es un elemento clave en tareas de localización, navegación o mapping.

En lo relativo a la visualización se ha tratado de mostrar en el mapa toda la información que pueda ser útil para el usuario; posición de los marcadores, ruta seguida, ruta planificada, partículas, posición actual, modo de operación y detección de marcadores. Por otro lado la imagen de la pantalla frontal utilizada para la detección del láser sirve al usuario para ver en todo momento lo que ve el robot.

Por último destacar todo lo aprendido en el desarrollo de este proyecto, al estar centrado en un campo interdisciplinar a la vez que moderno como es la de la robótica móvil.

11.2 Líneas futuras

Para continuar trabajando o evolucionando el sistema en un futuro se plantean las siguientes líneas.

En cuanto al sistema de control:

- Mejorar el sistema de control para conseguir realizar el control del movimiento en modo continuo, para esto habrá que evaluar en primera instancia los puntos críticos como la latencia en la comunicación o la velocidad del robot.
- Pensar en una estructura de control descentralizado, para liberar al PC de toda la carga de procesamiento. Una opción sería disponer de un PC dedicado a la detección de marcadores y localización de posición; que el Arduino se encargase del control del movimiento y evitar colisiones a bajo nivel; y otro PC dedicado a las tareas de visualización, mapping y planificación de movimientos a más alto nivel.

En lo relativo al posicionamiento basado en marcadores:

- Estudiar cuál es el límite operativo por el error acumulado de marcadores enlazados. Esto implicaría estimar la distancia máxima desde el origen. Una opción para corregirlo sería implementar que algunos de los marcadores den la posición absoluta reduciendo el error antes de llegar al límite.
- Relacionar la posición de los marcadores entre sí de manera que busquemos una red que trate de ser congruente en cuanto a la relación de distancias entre ellos y las mediciones, en vez de coger únicamente la primera medida para establecer la cadena de posiciones de marcadores.
- Usar marcadores para identificar objetos del entorno que serían detectados por la cámara frontal del robot.
- Buscar en las imágenes puntos característicos del entorno [45] para mejorar la detección o incluso para navegar en ausencia de marcadores.

La detección láser.

- Se podría estudiar la opción de hacer lecturas en 3D con el sistema láser-cámara. Un motor que moviese el conjunto láser-cámara permitiría crear un sistema de medición del entorno en 3D a un bajo coste.
- Trabajar en la detección del láser, tipos de láser, programa de visión o sistema detección para que el láser pudiese ser detectado sin detener el movimiento. Una opción sería disponer de algún tipo de filtro para el láser de manera que pudiésemos hacer dos fotos con dos cámaras, en una aparecería el láser y en la otra no.

Interfaz usuario:

- Crear una interfaz de usuario que mejorase la experiencia del usuario.

Navegación:

- Establecer un modo de teleoperación segura, de modo que el usuario comandase al robot pero este se detuviese si detecta algún obstáculo.

Referencias – Bibliografía

[1] Http://web.stanford.edu/~learnest/cart.htm, "Stanford Cart (Accesed on 15 August 2015)." .

[2] Https://en.wikipedia.org/wiki/Shakey_the_robot, "Shakey the robot (Accesed on 15 August 2015)." .

[3] N. J. Nilsson, "Shakey the Robot," *SRI AI Cent. Tech. Note*, no. April, 1984.

[4] K. lagnemma Massachusetts Institute of Technology, "State of the Art and Technical Frontiers."

[5] Boston-Dynamics, "Boston Dynamics: Dedicated to the Science and Art of How Things Move." 2012.

[6] M. H. Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, "Stanley: The Robot That Won the DARPA Grand Challenge." p. S. 180, 2012.

[7] M. Chili, "ETH, SLAM--Simultaneous Localization and Mapping. http://margaritachli.com (Accesed on 25 july 2015)," 2011.

[8] S. Frintrop, "Visual Robot Localization and Mapping based on Attentional Landmarks," pp. 3–6.

[9] S. Se, D. Lowe, and J. Little, "Vision-based mobile robot localization and mapping using scale-invariant features," *Proc. 2001 ICRA. IEEE Int. Conf. Robot. Autom. (Cat. No.01CH37164)*, vol. 2, 2001.

[10] "GI3- Publicaciones http://gi3.dia.uned.es/publicaciones.htm (Accesed on 4 September 2015)." .

[11] J. A. L.-O. and S. D. Dictino Chaos, Jesús Chacón, "A Mobile Robots Experimental Environment with Event-Based Wireless Communication - http://www.mdpi.com/1424-8220/13/2/2595/htm (Accesed on 4th of september 2015)." .

[12] J. S. and S. D. María Guinaldo, Ernesto Fábregas , Gonzalo Farias, Sebastián Dormido-Canto , Dictino Chaos, "Virtual and Remote Robotic Laboratory Using EJS, MATLAB and LabVIEW - http://www.mdpi.com/1424-8220/13/2/2595/htm (Accesed on 4th of september)." .

[13] J. B. Hayet, F. Lerasle, and M. Devy, "A visual landmark framework for mobile robot navigation," *Image Vis. Comput.*, vol. 25, no. 8, pp. 1341–1351, 2007.

[14] N. A. Andersen and J. C. Andersen, "Visual Navigation for Mobile Robots," no. 1987, pp. 143–169, 1997.

[15] G. Carmelo, L. Delfa, and V. Catania, "Accurate indoor navigation using Smartphone , Bluetooth Low Energy and Visual Tags," pp. 5–8.

[16] J. L. Sanchez-lopez, J. Pestana, P. D. La Puente, A. Carrio, and P. Campoy, "Visual Quadrotor Swarm for the IMAV 2013 Indoor Competition," *Robot. First Iber. Robot. Conf.*, vol. 253, no. September, pp. 55–63, 2014.

[17] "Open Source Augmented Reality SDK - ARToolKit, http://artoolkit.org/, (Accesed on 15 May 2015)." .

[18] Http://www.uco.es/investiga/grupos/ava/node/ (Accesed on 23 March 2015), "ArUco: a minimal library for Augmented Reality applications based on OpenCv." .

[19] J. López, C. Watkins, D. Pérez, and M. Díaz-Cacho, "Evaluating different landmark positioning systems within the RIDE architecture," *J. Phys. Agents*, vol. 7, no. 1, pp. 3–11, 2013.

[20] J. H. Oh, D. Kim, and B. H. Lee, "An Indoor Localization System for Mobile Robots Using an Active Infrared Positioning Sensor," *J. Ind. Intell. Inf.*, vol. 2, no. 1, pp. 35–38, 2014.

[21] A. Barrientos, "FUNDAMENTOS DE ROBOTICA, ISBN 9788448156367."

[22] S. Thrun, "https://www.udacity.com/course/viewer#!/c-ep245/l-48632907/m-48695666 (Accesed on 12 May 2015)." .

[23] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots," *Am. Assoc. Artif. Intell.*, no. Handschin 1970, pp. 1–6, 1999.

[24] S. Thrun, "Particle Filters in Robotics," *Proc. Uncertain. AI*, vol. 1, pp. 511–518, 2002.

[25] L. Marchetti, G. Grisetti, and L. Iocchi, "A Comparative Analysis of Particle Filter based Localization," *Rob. 2006 Robot Soccer World Cup X*, pp. 442–449, 2007.

[26] J. D. Hol, "Resampling in particle filters," *LiTH-ISY-EX-ET-0283-2004 Linkoping 2004*, p. 45, 2004.

[27] Wei Yu, Emmanuel Collins and Oscar Chuy, W. Yu, E. Collins, and O. Chuy, "Dynamic Modeling and Power Modeling of Robotic Skid-Steered Wheeled Vehicles," *Intechopen.Com*, no. 2005, 2010.

[28] K. Kozlowski and D. Pazderski, "Modeling and control of a 4-wheel skid-steering mobile robot," *Int. J. Appl. Math. Comput. Sci.*, vol. 14, no. 4, pp. 477–496, 2004.

[29] A. Mandow, J. L. Martínez, J. Morales, J. L. Blanco, A. García-Cerezo, and J. González, "Experimental kinematics for wheeled skid-steer mobile robots," *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 1222–1227, 2007.

[30] V. Valencia, a Jonny, and O. Montoya, "Modelo cinemático de un robot móvil tipo diferencial y navegación a partir de la estimación odométrica.," *Scientia*, no. 41, 2009.

[31] L. I. G. Valencia, "Modelado Cinemático y Control de Robots Móviles con Ruedas," p. 306, 2006.

[32] J. Park, G. N. DeSouza, and a. C. Kak, "Dual-beam structured-light scanning for 3-D object modeling," *Proc. Third Int. Conf. 3-D Digit. Imaging Model.*, pp. 65–72, 2001.

[33] X. Chen and S. Li, "A Survey on Otsu Image Segmentation Methods ⋆ One-dimensional Otsu Method Two-dimensional Otsu Method," vol. 10, no. 13, pp. 4287–4298, 2014.

[34] S. Thrun, "Robotic mapping: A survey," *Explor. Artif. Intell. new Millenn.*, no. February, 2002.

[35] Elfes, "Ussing ocupancy grids for mobile robot perception and navigation." .

[36] A. Elfes, "Occupancy Grids: A stochastic spatial representation for active robot perception."

[37] D. Fox, W. Burgard, and S. Thrun, "Probabilistic methods for mobile robot mapping," *Proc. IJCAI-99 Work.*, 1999.

[38] B. Kuipers, "Occupancy Grids," *Cell*, pp. 1–5.

[39] M. Otte, "A Survey of Machine Learning Approaches to Robotic Path-Planning," *Cs.Colorado.Edu*, 2015.

[40] B. H. P.E., N. N.J., and Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths." p. IEEE Transaction of systems science and cybernetic.

[41] N. L. T., D. I. S. & A. L. E. P. F. de L. Lausanne, R. P., C. C., and Correll, "SwisTrack - a flexible open source tracking software for multi-agent systems."

[42] ROS, "www.ros.org (Accesed on 7 of july 2015)." .

[43] E. a Lee, "The problem with Threads," *Publ. by IEEE Comput. Soc. , May 2006*, vol. 0018–9162/, no. May, pp. 33–42.

[44] "Mutual exclusion - Wikipedia, the free encyclopedia, https://en.wikipedia.org/wiki/Mutual_exclusion (Accesed on 6 of August 2015)." .

[45] W. Y. Jeong and K. M. Lee, "CV-SLAM: A new ceiling vision-based SLAM technique," *2005 IEEE/RSJ Int. Conf. Intell. Robot. Syst. IROS*, no. 1, pp. 3070–3075, 2005.

Abreviaturas y acrónimos

UML: Unified Modeling Language.

SLAM: Simultaneous Location And Mapping.

OpenCV: Open Source Computer Vision.

ArUco: Augmented Reality Universidad de Córdoba.

WiFi: Wireless Fidelity.

ANEXOS

A Planificación

La ejecución del proyecto se ha llevado a cabo a lo largo de 8 meses en el departamento de Informática y Automática de la UNED. A continuación se describen brevemente las fases en las que se ha dividido el desarrollo del proyecto:

Investigación inicial: Previamente incluso a la planificación de este proyecto, se ha realizado un estudio sobre el estado actual de robots autónomos más concretamente de localización, mapping y navegación.

Formación: Antes de comenzar el proyecto ya se contaba con profundos conocimientos de programación en C++ y de visión artificial, pero nunca se había trabajado con el entorno Eclipse, ni con las librerías de visión OpenCV.

Selección *Hardware:* En esta fase fue necesario seleccionar las cámaras para adquirir las imágenes de los marcadores y del láser, además fue necesario sustituir el PC y el láser instalados inicialmente.

Instalación *Hardware:* Instalación de las cámaras, láser y detectores de presencia en el robot.

Desarrollo *Software*: Incluye el diseño previo en UML, su posterior corrección, y el desarrollo de la aplicación en C++ junto a las librerías de visión artificial de OpenCV y de marcadores ArUco.

Pruebas*:* Pruebas de funcionamiento del robot en entornos controlados para la posterior corrección, depuración y posibles mejoras del software.

Redacción de la memoria del proyecto. Una vez conseguidos los hitos anteriores y tras haber realizado las pruebas, se analizan los resultados y se obtienen conclusiones a partir de los mismos.

A continuación se refleja mediante un diagrama de Gantt las principales actividades que se desarrollan en las etapas anteriormente mencionadas. Con esto se pueden observar las relaciones entre actividades.

B Diagrama de Gantt

Nombre de la tarea	Fecha de Inicio	Fecha final	Duración
Aprendizaje	**10/12/14**	**06/01/15**	**20d**
Estado del arte	10/12/14	06/01/15	20d
Formación	**22/12/14**	**28/04/15**	**92d**
Linux	22/12/14	09/02/15	35,25d
Eclipse	12/01/15	18/02/15	27,5d
OpenCV	28/01/15	04/03/15	25,5d
ArUco	02/04/15	28/04/15	19d
Selección Hardware	**19/04/15**	**10/08/15**	**82d**
Selección Webcam	19/04/15	20/04/15	2d
Selección Laser	15/05/15	18/05/15	2d
Selección PC	26/07/15	27/07/15	2d
Selección medidores	10/08/15	10/08/15	1d
Desarrollo software	**04/03/15**	**21/08/15**	**122,5d**
Integración marcadores	**04/03/15**	28/05/15	61d
Filtros de partículas	**07/05/15**	20/05/15	10d
Visualización	**18/03/15**	05/08/15	100,25d
Detección laser	**04/04/15**	24/04/15	16d
Creación de mapas	**30/07/15**	13/08/15	11d
Navegación robot	**27/07/15**	21/08/15	19,5d
A*	**06/08/15**	21/08/15	12d
Instalación Hardware	**06/05/15**	**20/05/15**	**11d**
Instalación cámaras	**01/08/15**	03/08/15	2d
Instalación láser	**01/08/15**	03/08/15	2d
Instalación medidores	**11/08/15**	12/08/15	2d
Pruebas	**06/05/15**	**20/05/15**	**11d**
Integración marcadores	**27/03/15**	03/06/15	49d
Detección laser	**20/04/15**	15/06/15	41d
Visualización	**06/05/15**	05/08/15	65,25d
Filtros de partículas	**18/05/15**	10/06/15	17,5d
Creación de mapas	**08/08/15**	27/08/15	14,5d
Navegación robot	**12/08/15**	26/08/15	10,25d
A*	**10/08/15**	21/08/15	10d
Redacción de la memoria del proyecto	**10/08/15**	**03/09/15**	**19d**

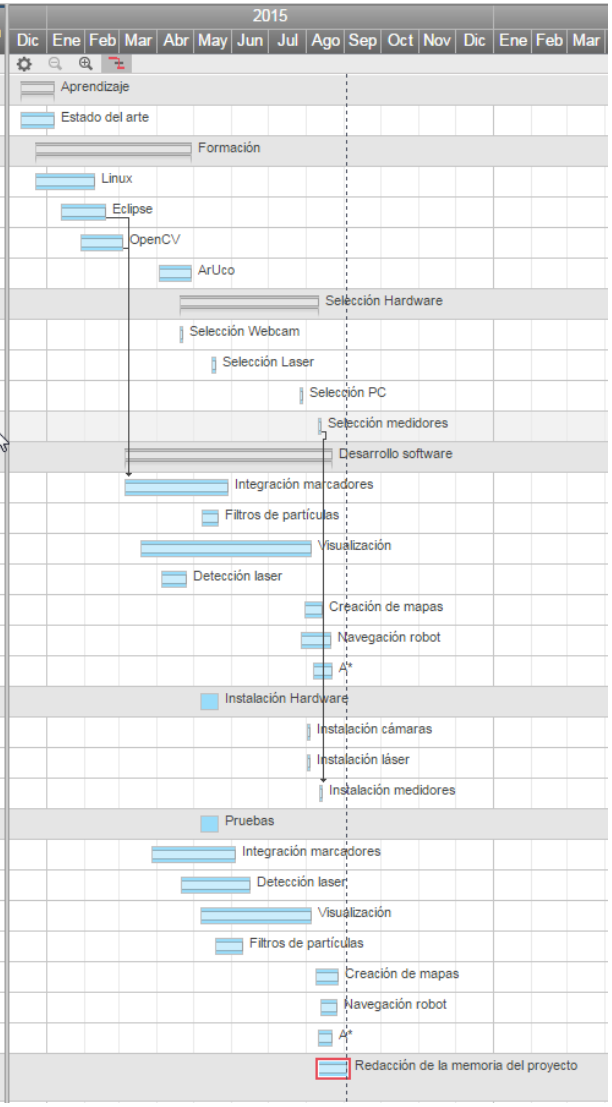

Figura 92: Diagrama de Gantt

C Estructura de descomposición del proyecto (EDP)

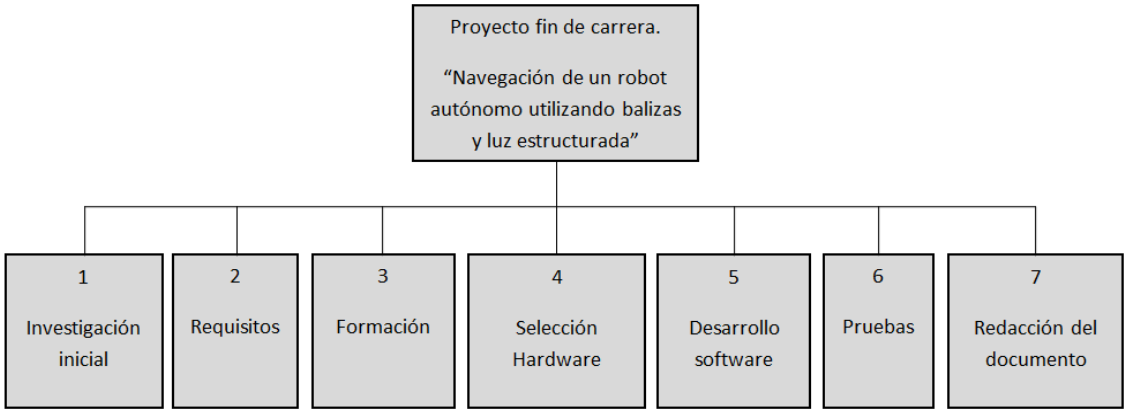

Figura 93: Estructura de descomposición del proyecto

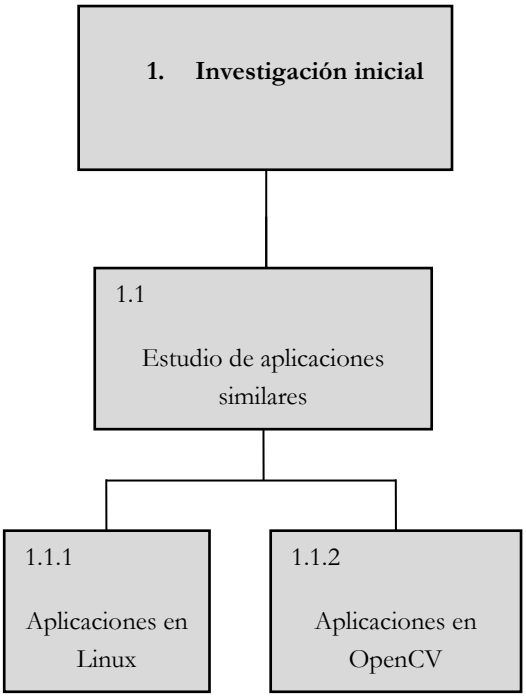

Figura 94: Investigación inicial.

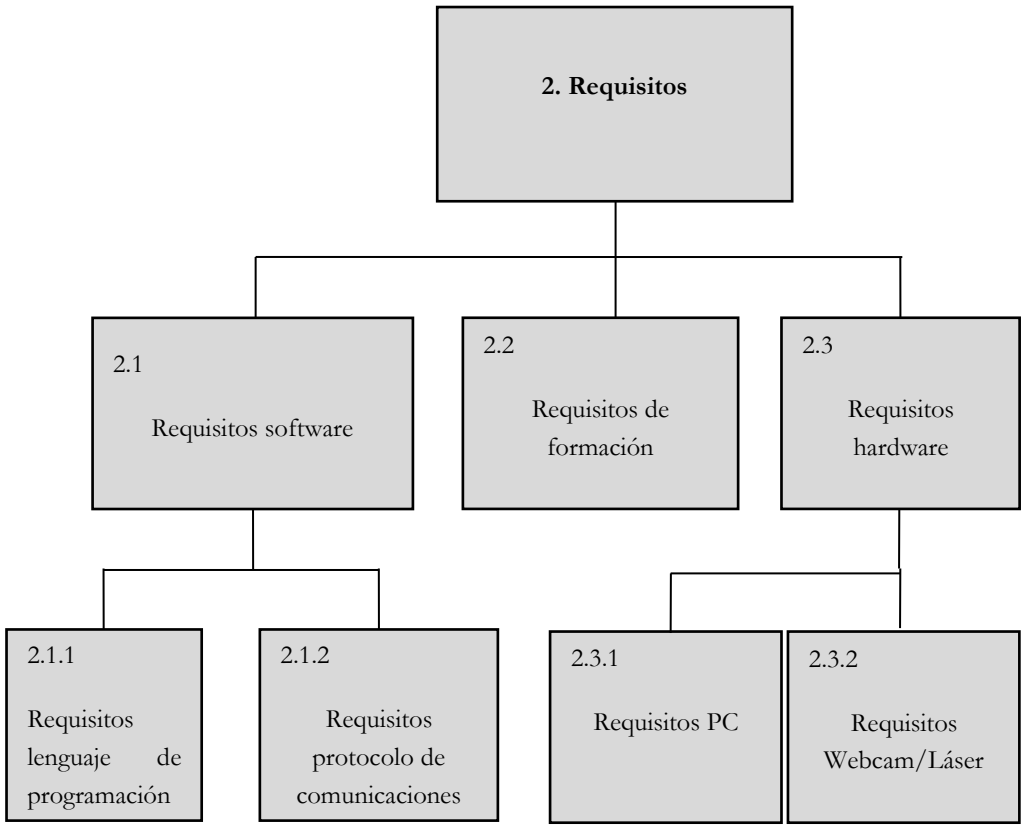

Figura 95: Requisitos.

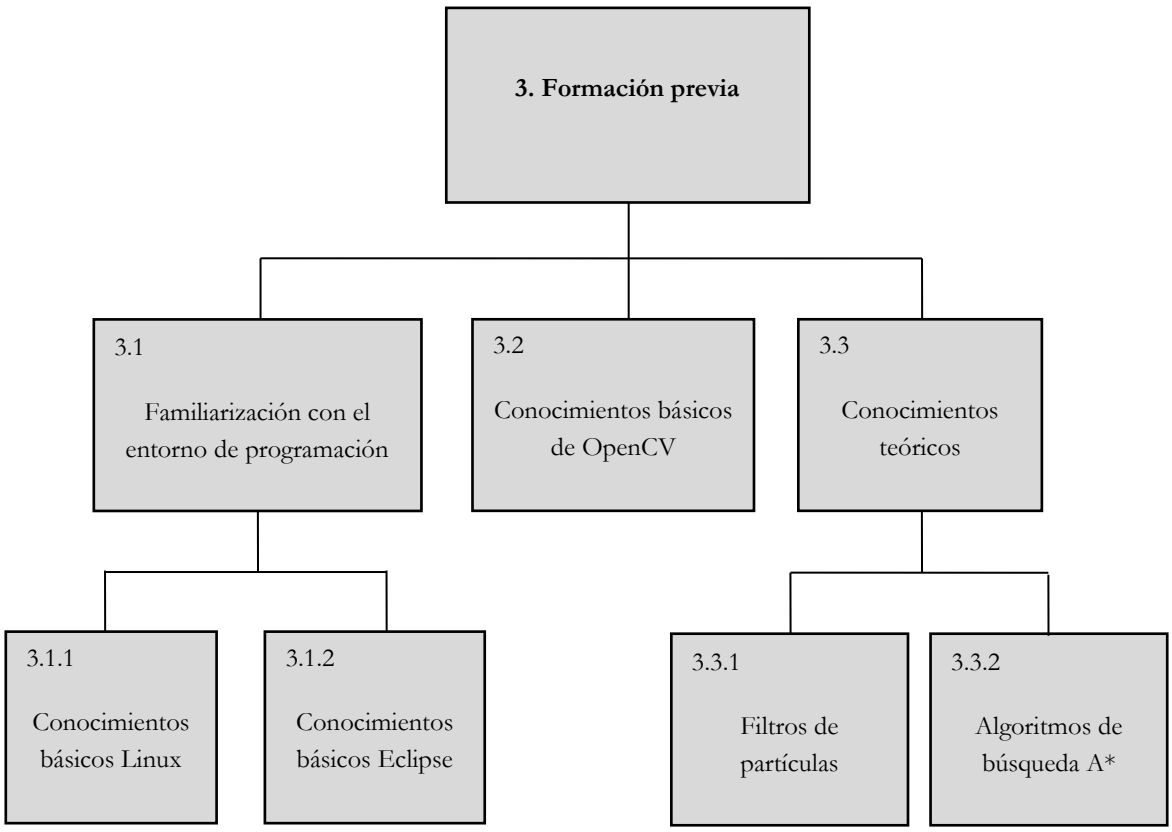

Figura 96: Formación previa.

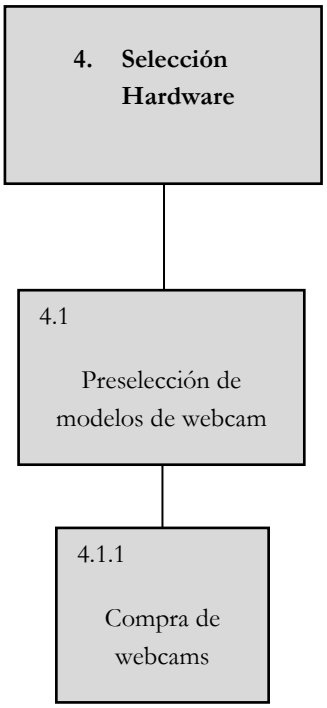

Figura 97: Selección Hardware.

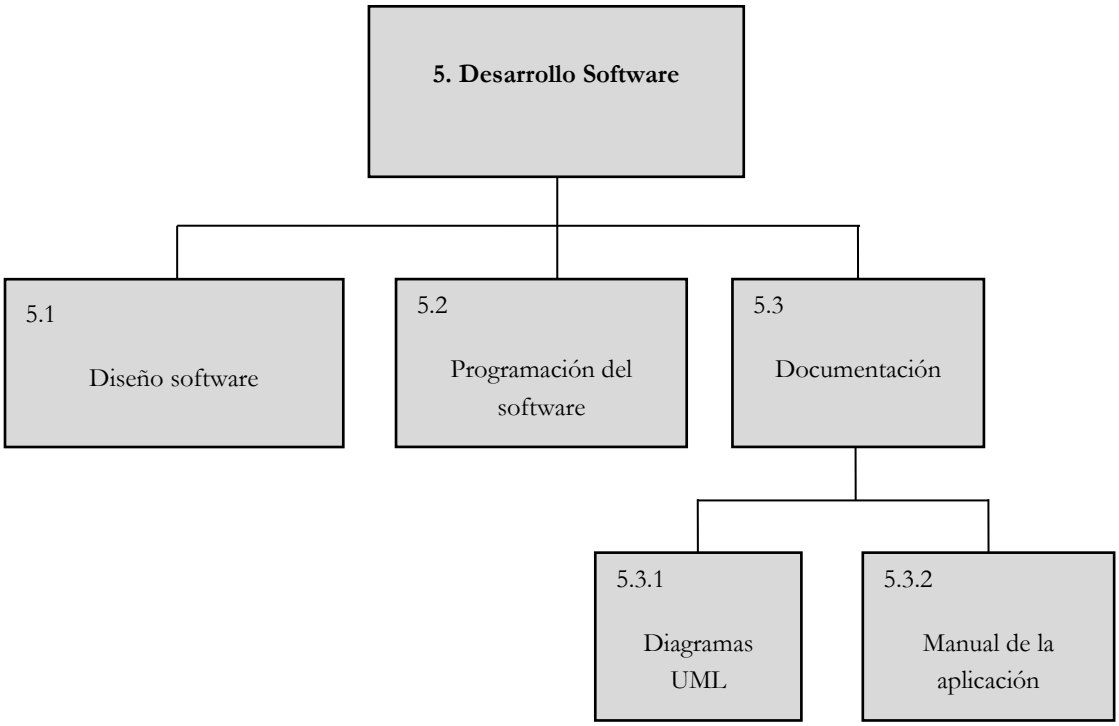

Figura 98: Desarrollo Software.

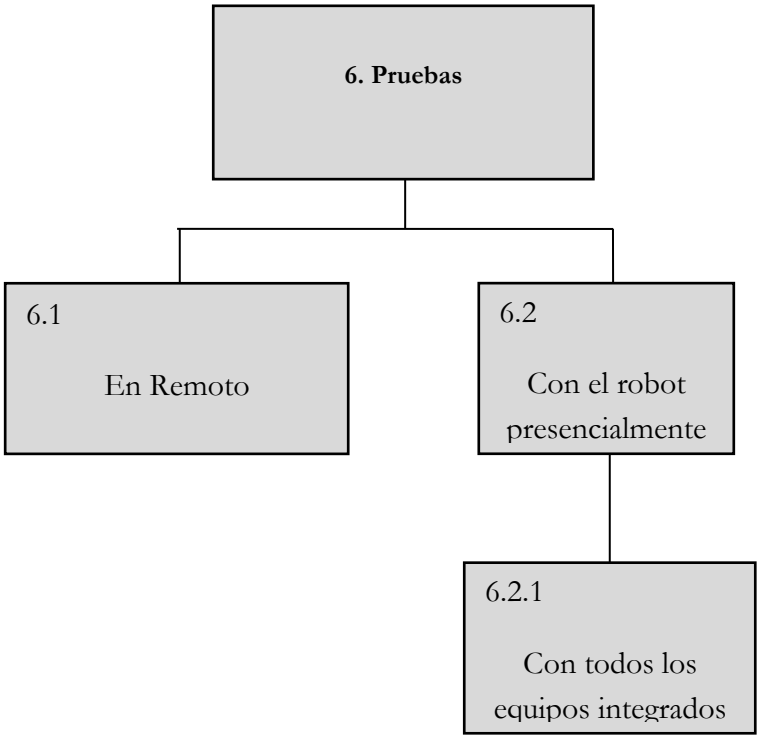

Figura 99: Pruebas.

D Presupuesto

El presupuesto se ha dividido en varios apartados que se comentan a continuación

Mediciones y precio unitario

En el siguiente apartado quedan reflejadas las mediciones de los diversos elementos que componen nuestro sistema para luego realizar el presupuesto.

Descripción	Unidades	Precio (€)
Partida Software		
Linux Debian	1	0 €
Eclipse	1	0 €
OpenCV	1	0 €
Partida Hardware - Robot		
Kit robot	1	400 €
Cámaras	2	24 €
Láser	1	5 €
Portátil	1	800 €
Sensores distancia	3	13 €
Partida de mano de obra		
Proyectante	600 h	25 €/h
Director	100 h	60 €/h

Tabla 3: Mediciones y precio unitario.

Presupuesto total

Descripción	Precio (€)
Partida Software	0 €
Partida Hardware	1292 €
Partida de mano de obra	21000 €
Suma total	22292,0 €
+ 6% Costes indirectos	1337,52 €
+ 13% Beneficio Industrial	2897,96 €
+ 18% IVA	4012,56 €
TOTAL	30540,04 €

Tabla 4: Presupuesto total.

El **presupuesto total** del presente proyecto asciende al total de **30.540,04 €.**

Treinta mil quinientos cuarenta euros y cuatro céntimos.

E Tablas de casos de uso

UC-1	**Guardar Mapa**	
Versión	1.0 Fecha: 1 de Septiembre de 2015.	
Descripción	El mapa actual se almacena en el directorio del programa.	
Precondición	Programa en ejecución.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa la tecla "s".
	2	El mapa se almacena.
Postcondición	El programa continua su ejecución	
Excepciones	**Paso**	**Acción**
	1	Si la ruta es incorrecta el mapa no se almacena.
Rendimiento	**Paso**	**Cota de tiempo**
	-	-
Frecuencia	Una vez por pulsación.	
Comentarios	Ninguno.	

Tabla 5: Tabla caso de uso "Guardar Mapa".

UC-2	**Mostrar láser en mapa**	
Versión	1.0 Fecha: 1 de Septiembre de 2015.	
Descripción	Cambio de estado mostrar medición láser actual en el mapa.	
Precondición	Programa en ejecución.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa la tecla "d".
	2	Se invierte el estado de mostrar mapa.
Postcondición	El programa continua su ejecución.	
Excepciones	**Paso**	**Acción**
	1	No hay medición láser actual no se muestra en el mapa.
Rendimiento	**Paso**	**Cota de tiempo**
	-	-
Frecuencia	Una vez por pulsación.	
Comentarios	Ninguno.	

Tabla 6: Tabla caso de uso "Mostrar láser en mapa".

UC-3	**Mostrar marcadores en mapa**	
Versión	1.0 Fecha: 1 de Septiembre de 2015.	
Descripción	Cambio de estado mostrar marcadores detectados en el mapa.	
Precondición	Programa en ejecución.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa la tecla "m".
	2	Se invierte el estado de mostrar marcadores.
Postcondición	El programa continua su ejecución.	
Excepciones	**Paso**	**Acción**
	1	No hay marcadores no se muestran en el mapa.
Rendimiento	**Paso**	**Cota de tiempo**
	-	-
Frecuencia	Una vez por pulsación.	
Comentarios	Ninguno.	

Tabla 7: Tabla caso de uso "Mostrar marcadores en mapa".

UC-4	**Mostrar mapa mejorado**	
Versión	1.0 Fecha: 1 de Septiembre de 2015.	
Descripción	Alterna visualización entre mapa de probabilidad o mapa mejorado.	
Precondición	Programa en ejecución.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa la tecla "I".
	2	Se alterna visualización del mapa.
Postcondición	El programa continua su ejecución.	
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	**Paso**	**Cota de tiempo**
	-	-
Frecuencia	Una vez por pulsación.	
Comentarios	Ninguno.	

Tabla 8: Tabla caso de uso "Mostrar mapa mejorado".

UC-5	**Mostrar partículas mapa**	
Versión	1.0 Fecha: 1 de Septiembre de 2015.	
Descripción	Cambio de estado mostrar partículas del filtro en el mapa.	
Precondición	Programa en ejecución.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa la tecla "p".
	2	Se invierte el estado de mostrar partículas.
Postcondición	El programa continua su ejecución.	
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	**Paso**	**Cota de tiempo**
	-	-
Frecuencia	Una vez por pulsación.	
Comentarios	Ninguno.	

Tabla 9: Tabla caso de uso "Mostrar partículas en mapa".

UC-6	**Mostrar trayectoria seguida por el robot**	
Versión	1.0 Fecha: 1 de Septiembre de 2015.	
Descripción	Cambio de estado mostrar trayectoria seguida por el robot en el mapa.	
Precondición	Programa en ejecución.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa la tecla "l".
	2	Se invierte el estado de mostrar trayectoria en mapa.
Postcondición	El programa continua su ejecución.	
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	**Paso**	**Cota de tiempo**
	-	-
Frecuencia	Una vez por pulsación.	
Comentarios	Ninguno.	

Tabla 10: Tabla caso de uso "Mostrar trayectoria seguida por el robot en mapa".

UC-7	**Mostrar trayectoria planificada por el robot**	
Versión	1.0 Fecha: 1 de Septiembre de 2015.	
Descripción	Cambio de estado mostrar trayectoria planificada por el robot en el mapa.	
Precondición	Programa en ejecución.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa la tecla "t".
	2	Se invierte el estado de mostrar trayectoria planificada en mapa.
Postcondición	El programa continua su ejecución.	
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	**Paso**	**Cota de tiempo**
	-	-
Frecuencia	Una vez por pulsación.	
Comentarios	Ninguno.	

Tabla 11: Tabla caso de uso "Mostrar trayectoria planificada por el robot en mapa".

UC-8	**Reset programa**	
Versión	1.0 Fecha: 1 de Septiembre de 2015.	
Descripción	Reseteo de variables de programa al estado inicial después del arranque.	
Precondición	Programa en ejecución.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa la tecla "r".
	2	Se invierte el estado de mostrar marcadores.
Postcondición	El programa continua su ejecución.	
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	**Paso**	**Cota de tiempo**
	-	-
Frecuencia	Una vez por pulsación.	
Comentarios	Ninguno.	

Tabla 12: Tabla caso de uso "Reset programa".

UC-9	**Establecer modo teleoperación**	
Versión	1.0 Fecha: 1 de Septiembre de 2015.	
Descripción	Se establece el modo de navegación "Teleoperación"	
Precondición	Programa en ejecución.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa la tecla "0".
	2	El modo de navegación pasa a teleoperación
Postcondición	El programa continua su ejecución.	
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	**Paso**	**Cota de tiempo**
	-	-
Frecuencia	Una vez por pulsación.	
Comentarios	Ninguno.	

Tabla 13: Tabla caso de uso "Establecer modo teleoperación".

UC-10	**Establecer modo seguimiento de marcadores**	
Versión	1.0 Fecha: 1 de Septiembre de 2015.	
Descripción	Se establece el modo de navegación "Seguimiento de marcadores"	
Precondición	Programa en ejecución.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa la tecla "1".
	2	El modo de navegación pasa a "Seguimiento de marcadores"
Postcondición	El programa continua su ejecución.	
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	**Paso**	**Cota de tiempo**
	-	-
Frecuencia	Una vez por pulsación.	
Comentarios	Ninguno.	

Tabla 14: Tabla caso de uso "Establecer modo seguimiento de marcadores".

UC-11	**Establecer modo navegación aleatoria**	
Versión	1.0 Fecha: 1 de Septiembre de 2015.	
Descripción	Se establece el modo de navegación "Navegación aleatoria"	
Precondición	Programa en ejecución.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa la tecla "2".
	2	El modo de navegación pasa a "Navegación aleatoria"
Postcondición	El programa continua su ejecución.	
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	**Paso**	**Cota de tiempo**
	-	-
Frecuencia	Una vez por pulsación.	
Comentarios	Ninguno.	

Tabla 15: Tabla caso de uso "Establecer modo de navegación aleatoria".

UC-12	**Establecer modo maximizar área explorada**	
Versión	1.0 Fecha: 1 de Septiembre de 2015.	
Descripción	Se establece el modo de navegación "Maximizar área explorada"	
Precondición	Programa en ejecución.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa la tecla "3".
	2	El modo de navegación pasa a "Maximizar área explorada"
Postcondición	El programa continua su ejecución.	
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	**Paso**	**Cota de tiempo**
	-	-
Frecuencia	Una vez por pulsación.	
Comentarios	Ninguno.	

Tabla 16: Tabla caso de uso "Maximizar área explorada".

UC-13	**Avanzar**	
Versión	1.0 Fecha: 1 de Septiembre de 2015.	
Descripción	El robot se mueve hacia adelante.	
Precondición	Programa en ejecución.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa la tecla "f".
	2	El robot se mueve hacia adelante.
Postcondición	El programa continua su ejecución.	
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	**Paso**	**Cota de tiempo**
	-	-
Frecuencia	Una vez por pulsación.	
Comentarios	Ninguno.	

Tabla 17: Tabla caso de uso "Avanzar".

UC-14	**Retroceder**	
Versión	1.0 Fecha: 1 de Septiembre de 2015.	
Descripción	El robot se mueve hacia atrás.	
Precondición	Programa en ejecución.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa la tecla "b".
	2	El robot se mueve hacia atrás
Postcondición	El programa continua su ejecución.	
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	**Paso**	**Cota de tiempo**
	-	-
Frecuencia	Una vez por pulsación.	
Comentarios	Ninguno.	

Tabla 18: Tabla caso de uso "Retroceder".

UC-15	**Giro derecha**	
Versión	1.0 Fecha: 1 de Septiembre de 2015.	
Descripción	El robot se mueve hacia la derecha.	
Precondición	Programa en ejecución.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa la flecha "->".
	2	El robot se mueve hacia la derecha.
Postcondición	El programa continua su ejecución.	
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	**Paso**	**Cota de tiempo**
	-	-
Frecuencia	Una vez por pulsación.	
Comentarios	Ninguno.	

Tabla 19: Tabla caso de uso "Giro derecha".

UC-16	**Giro izquierda**	
Versión	1.0 Fecha: 1 de Septiembre de 2015.	
Descripción	El robot se mueve hacia la izquierda.	
Precondición	Programa en ejecución.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa la flecha "<-".
	2	El robot se mueve hacia la izquierda
Postcondición	El programa continua su ejecución.	
Excepciones	**Paso**	**Acción**
	-	-
Rendimiento	**Paso**	**Cota de tiempo**
	-	-
Frecuencia	Una vez por pulsación.	
Comentarios	Ninguno.	

Tabla 20: Tabla caso de uso "Giro izquierda".