Proyecto Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Localización y mapeo simultáneo de objetos usando ROS

Autor: Carlos Rodríguez Alonso
Tutor: Begoña C. Arrue Ullés

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Proyecto Fin de Carrera
Grado en Ingeniería de las Tecnologías Industriales

Localización y mapeo simultáneo de objetos usando ROS

Autor:
Carlos Rodríguez Alonso

Tutor:
Begoña C. Arrue Ullés
Profesor titular

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Trabajo fin de Grado: Localización y mapeo simultáneo de objetos usando ROS

Autor: Carlos Rodríguez Alonso

Tutor: Begoña C. Arrue Ullés

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

Agradecimientos

"If I have seen further it is by standing on the shoulders of giants"

Isaac Newton

Antes de cerrar esta etapa de mi vida me gustaría dar las gracias a todas aquellas personas que han estado apoyándome desde el principio. Siempre he sido una persona muy despistada y muy risueña a la que le ha hecho falta que le negarán el conformismo. Durante mi etapa en secundaria, más de una vez necesite que profesores me recordaran que podía dar mucho más de mí y que debía esforzarme más. En esos momentos no quería escucharlos, me conformaba con aprobar y más de una vez me tuvieron que dar un palo. A día de hoy, me alegro de que hicieran aquello, ya que cambiaron mi forma de plantearme mis estudios y son de las cosas más importantes que me han hecho llegar a donde estoy hoy.

Quiero dar también las gracias a todos los compañeros que he ido conociendo a lo largo de mis estudios. Especialmente a aquellos conocidos en la Universidad. Siempre han sido una fuente de escape que hacían que tantas horas encerrado en la escuela fueran mucho menos duras, y por tanta ayuda como nos hemos prestado para seguir avanzando y que confío en que siga siendo así a lo largo de nuestra vida.

Especialmente quiero dar las gracias a mi familia. A mis padres por exigir siempre más de mí, por esos "- pues si hoy no quieres estudiar más no pasa nada, mañana será otro día", esas alegrías cada vez que les decía una nueva nota aprobada y esos consuelos y "- no te preocupes a la próxima será" con los suspensos. A mi madre por haberme criado desde chico y haber luchado siempre para que saliéramos adelante como solo ella sabía hacer, como una verdadera luchadora. A mi padre, una persona de apariencia serie y gruñona, pero a la vez una de las personas con más amor dentro deseando de repartir, capaz de entrar en una familia rota y poner las vendas necesaria para que esta siguiera adelante, haciéndose cargo de tantas responsabilidades como fueran necesarias con solo el amor como recompensa. A mi hermano menor, por tantos abrazos y tiempo compartido, y esa mirada en sus ojos demostrando que, a pesar de todo lo que diga la gente, son de las personas que más cariño pueden darte. A mi hermano mayor, referente desde pequeño, una persona inteligente que siempre ha prestado su ayuda, que a diferencia de lo que la gente pueda decir "- ¿y no le da coraje que estudies exactamente lo mismo que él?" siempre me ha apoyado en todo lo que he decidido hacer y se ha sentido orgulloso de como he progresado.

Como no podía ser de otra forma, también tengo que mencionar a mis amigos, por tantas risas y momentos compartidos, tantas experiencias vividas y por vivir, tantos "- mañana a las nueve te recojo para ir a la biblioteca" cuando tus fuerzas de seguir dan de sí y por tantos desahogos, que siempre me han apoyado a seguir adelante. Porque, a pesar de todo lo que hemos pasado unos con otros, son la familia que yo he elegido.

A ella, que está ahora sentada junto a mí, preparándose para unos exámenes que no ha podido curas diciendo "- esto suena a chino". Con quien he tenido la suerte de superar desde su inicio esta etapa de mi vida, contenta siempre de cómo me ha ido, apoyándome en los peores momentos, la única persona con la que de verdad puedo desahogarme del mundo, olvidarme del tiempo, que ha aceptado mi forma de ser y la ha querido, mi persona favorita.

Y por último quiero dar las gracias también a Begoña Arrue, por ayudarme a desarrollar este proyecto y a aprender tanto sobre partes de la robótica que no conocía.

Resumen

SLAM es un acrónimo de *Simultaneous Localization And Mapping*. Como su nombre indica, su objetivo es dotar a la plataforma que lo emplee de la autonomía necesaria para realizar el proceso de localización y mapeo simultáneamente en un entorno desconocido. En este trabajo explicaremos las bases de este método, así como diferentes herramientas y software empleado para realizar este. Posteriormente, se presentarán 3 algoritmos del estado del arte de este sistema, así como algunas pruebas realizadas sobre estos mismos.

Abstract

Simultaneous Localization and mapping have been a present problem in robotics for the last decade. It aims to provide the robots with the necessary autonomy to achieve their task, given and unknown environment where it need to localize itself and develop a map. In this work, we present an approach to the technique of SLAM itself and to diverse ways to accomplish it, using different kind of sensors and algorithm; as well as to present different kind of tools to implement it in real robot's system. Hence, we start talking about the history of SLAM and the algorithm itself, including some features detectors; the Extended Kalman Filter, which was the first method to be employ in this kind of technique; a brief explanation about Robot Operating System and its tools; the software where it should be implement, in our case it will be Linux; 3 different state-of-the-art algorithm to carry out this method and some test if possible; and finally a conclusion about what we finally get about this work.

Índice

Índice de Figuras

Notación

A^T	Matriz transpuesta
A^{-1}	Matriz inversa
$\frac{\partial f}{\partial x}$	Derivada parcial de la función f con respecto a x
dx	Diferencial de x
$\dot{x}$	Derivada de x respecto al tiempo
e	número e
Exp(x)	Función Exponencial de x
arg	argumento
sen	Función seno
tg	Función tangente
arctg	Función arco tangente
sen	Función seno
$\approx$	Aproximado a
min	Mínimo
$\sum_{i=1}^{n} f$	Sumatorio desde i igual a 1 hasta n de la función f
x^2	X elevado al cuadrado
Δx	Variación de x
$\|x\|$	Módulo del vector x
$Pr(A)$	Probabilidad del suceso A
MSE	Minimum square error
$\sqrt{x}$	Raíz cuadrada de x
$x \leftarrow f$	Actualización del valor de x a partir de la función f

Acrónimos

SLAM	Simultaneous Localization and Mapping
EKF	Extended Kalman Filter
IA	Inteligencia Artificial
IEEE	Institute of Electricals and Electronics Engineers
TOF	Time of fligth
RANSAC	Random Sampling Consensus
SIFT	Scale-Invariant Feature Transform
SURF	Speeded-Up Robust Features
ORB	Oriented FAST and Rotated BRIEF
FAST	Features from Accelerated Segment Test)
BRIEF	Binary Robust Independent Elementary Features)
ROS	Robot Operating System
IMU	Inertial Measurement unit
ICP	Iteratie Closest Point
EMM	Environment Measurement Model
RMSE	Root-Main-Squares Error

1 INTRODUCCIÓN

Hoy en día, la robótica es uno de los campos de la tecnología con más desarrollo e investigación, tanto con fines económicos como sociales. La robótica industrial vio su auge con el desarrollo de las cadenas de montaje, ya que suponían un medio muy eficiente para reducir los costes de personal. A su vez, la robótica domestica también ha recibido un gran desarrollo en los últimos años con su empleo destinado a la ayuda en tareas domésticas, como el robot de iRobot Roomba, o con fines sociales a la hora de desarrollar entretenimiento.

Uno de los primeros conceptos que desarrollan la entidad de un robot pasa por el término de autonomía, entendido como la capacidad de realizar tareas y tomar decisiones por si mismo, o la cantidad de tiempo que un robot puede realizar tareas sin necesidad de volver a una estación base o detener el proceso realizado. En este trabajo, utilizaremos el termino autonomía con referencias a la primera definición. Uno de los requisitos principales para que el robot pueda realizar tareas de manera autónoma es el conocimiento del entorno que le rodea, tanto en tareas con fines de manipulación, como en aquellas destinadas a la exploración.

Para ello, a lo largo de los años se han realizado tareas de mapeo, así como herramientas que dotarán de una capacidad de localización para los robots, como sistemas basados en GPS o en balizas. Es en este momento cuando surge la investigación sobre realizar estas dos tareas de manera simultánea.

1.1 Objetivos del trabajo

SLAM es uno de los problemas más investigados por la comunidad robótica, ya que es fundamental para dotar de completa autonomía a robot móviles. Debido a esto, muchos investigadores han desarrollado gran número de métodos diferentes para realizar esta tarea, empleando distintos tipos de algoritmos y de sistemas de medida.

Los objetivos planteados para este proyecto se basan en la investigación sobre el algoritmo de SLAM y diversos métodos que emplean este mediante el middleware ROS, entre los que se incluyen *Hector SLAM*, *RGBD-SLAM* y *ElasticFusion*. Estos métodos difieren entre si en el tipo de sensores empleados, el algoritmo empleado para la creación del mapa, así como la metodología que sigue le proceso de posicionamiento.

De esta forma, se pretende implantar unas bases de conocimiento sobre el método expuesto para realizar el mapeo y la localización de manera simultaneas, así como los medios para implantar esta metodología en sistemas robóticos y presentar alguno de los algoritmos del estado del arte actual del sistema, incluyendo un método muy novedoso que emplea un grafo de deformación para sustituir el grafo de posiciones en que se suelen basar los algoritmos actuales; y una visión más centrada en la realización del mapeo

1.2 Estructura de la memoria

El trabajo está dividido en 7 capítulos. En el primero, se expone la idea sobre lo que es SLAM, la formulación y estructura del mismo, así como su origen, los tipos de herramientas que pueden ser empleadas para la ejecución de este mismo y diversos métodos para la extracción de landmark, o puntos característicos del entorno, y la asociación de sus resultados con los valores reales del entorno. En el segundo capítulo se expone la realización de este con el método más sencillo y el inicial que se desarrolló, un filtro extendido de Kalman. El tercer capítulo

hace hincapié sobre las distintas plataformas y herramientas actuales que existen para facilitar el empleo y representación de este modelo. En los siguientes capítulos se explica cómo se desarrollan una serie de métodos, explicando cada parte de su funcionamiento, y se realizan una serie de pruebas en el caso de que sea posible. Por último, en el capítulo final se desarrolla una breve explicación de conjunto comparando los métodos desarrollados, sus pros y sus contras, y se desarrolla la explicación sobre cuál podría ser el desarrollo posterior a este trabajo.

2 ALGORITMO SLAM

SLAM (Simultaneous Localization and Mapping) es una técnica cuyo objetivo consiste en la construcción de mapas de un entorno desconocido y, simultáneamente, estimar la posición dentro de dicho mapa, mediante el uso de la odometría y el entorno, percibido mediante una serie de sensores. Es gran importancia para el desarrollo de la robótica móvil, debido a que en ello radica que el robot pueda llegar a ser completamente autónomo.

2.1 Historia de SLAM

La idea de SLAM surge en 1986 en una conferencia de la IEEE, gracias a la reciente introducción de métodos probabilísticos en la robótica y la IA. El resultado de la conferencia, en la que se tomaron datos de investigadores que ya habían tratado de introducir métodos de estimación a algoritmos de localización y mapeo, y a su vez de otros investigadores presentes en la conferencia que aportaron ideas durante esta; fue que la aplicación de estos métodos probabilísticos era de vital importancia para la robótica y que eran un problema que debía ser resuelto.

En 1990 llegarían las bases para la descripción de las relaciones entre la localización y los landmark (puntos de referencia) de la mano de Smith y Cheeseman [1] así como otros de Durrant-Whyte [2]. Mientras tanto, otros investigadores trabajaban sobre métodos visuales y sensoriales (utilizando un sonar) de navegación en robot móviles aplicando algoritmos basados en el filtro de Kalman. Gracias a estas dos investigaciones aparecería el elemento clave en la discusión, una publicación de Smith R, Self M, Cheeseman P cuyo artículo sería "*Estimating uncertain spatial relationships in robotics. In: Autonomous Robot Vehicles. Springer, pp 167–193*" [3] en el que se muestra el trabajo sobre un robot móvil desarrollando sus tareas en un entorno desconocido tomando datos de los landmark, cuya estimación debe estar correlada con los otros landmark debido a los errores en la estimación de la posición del vehículo.

Este trabajo implicaba que sería necesario un estado inicial en el que se encontraran incluidos la posición del robot y la de cada landmark. Para ello, el estimador debería emplear un vector de estados enorme, que crecería de manera cuadrática con el número de landmark.

Debido a la asumida no convergencia de los errores en el mapeo y a la complejidad computacional del problema, muchos investigadores se centraron en buscar aproximaciones del problema, suponiendo o incluso forzando a minimizar o eliminar las correlaciones entre los diferentes landmark para conseguir una serie de problemas landmark-robot desacoplados. Por otro lado y debido a las mismas razones, otros investigadores continuaron separando los problemas, centrándose en la localización por un lado y el mapeo por otro.

Sin embargo, poco más tarde se descubrió que formulando el problema de la localización y el mapeo como uno único este se volvía convergente. Asimismo, se descubrió que las correlaciones entre los landmark eran un punto fundamental para conseguir la convergencia.

Una vez llegado a esto, el trabajo se centraría en conseguir aumentar la eficiencia computacional y solucionar los problemas asociados a la toma de datos a al cierre del bucle (loop closure) [4].

A partir de entonces, SLAM se convirtió en un tema fundamental para la robótica móvil y se han llevado a cabo muchas investigaciones en las que se han desarrollado nuevos algoritmos, más eficientes y que se siguen desarrollando hoy día.

2.2 Formulación y estructura de SLAM

SLAM es un proceso por el cual un robot móvil es capaz de generar un mapa del entorno y localizarse en el al mismo tiempo sin necesidad de ningún dato previo del entorno mediante el uso de una serie de puntos de referencia denominados landmark.

El problema se puede subdividir en una serie de partes más pequeñas: asignación de landmark, correspondencia de datos, estimación de estado, actualización de estado y actualización de landmark. Una vez dividido, hay muchas formas de solucionar cada parte [5].

El algoritmo funcionará de manera que una vez asignados los landmark iniciales se comienza el movimiento y el robot se guiará basándose en la odometría. Si encuentra nuevos landmark los incluye en el vector de estados y continua su camino. Una vez se visualiza por segunda vez un mismo landmark, es decir, uno incluido en el vector de estado se puede proceder a cerrar el bucle de forma que, conociendo la posición de dicho landmark y del robot en el mapa, la distancia al landmark (orientación y distancia) y las incertidumbres debidas a los errores de medición y posicionamiento, se actualiza el vector de estado, corrigiendo el mapa creado y por tanto la localización de nuestro robot en él.

Este proceso se puede realizar mediante un EKF. Este se encarga de ir actualizando la posición virtual del robot, entendido por esto la posición en la que el robot cree que se sitúa dentro del mapa. Asimismo, también se encarga de estimar las posiciones relativas de los landmark dentro del mapa y con respecto al robot. El proceso se puede resumir de la siguiente forma:

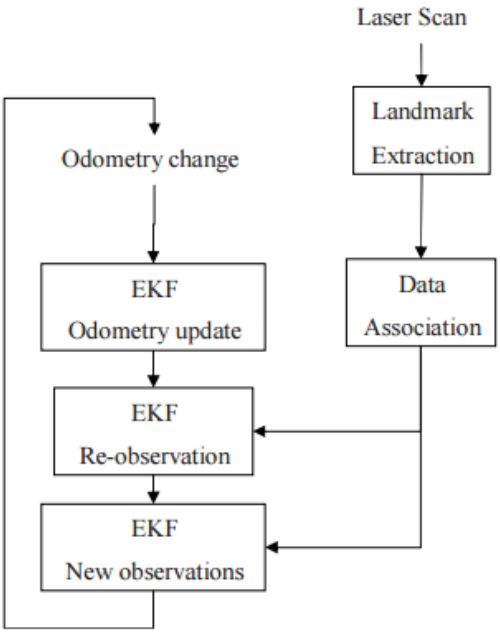

Figura 1Esquema del proceso SLAM

2.3 Sensores y herramientas

La principal característica de un robot móvil es la presencia de ruedas o algún mecanismo que permita el movimiento del mismo; o hélices en el caso de robot aéreos. Conociendo las ordenes que da el controlador a

estos mecanismos de movimiento se puede estimar la posición en la que se encuentra nuestro robot. Esta forma de valorar la posición se conoce como odometría.

Sin embargo, la odometría no suele ser muy precisa, debido a que los errores pasan a ser acumulativos y muy difíciles de solventar ya que suelen ser de carácter arbitrario, como el deslizamiento de una rueda, una ráfaga de viento que desplace el robot o la intervención humana. Estos son conocidos como errores no sistemáticos. Existen también otra serie de errores conocidos como sistemáticos debidos a imperfecciones en la estructura del robot, pero en ellos la acción sobre el robot puede ser compensada.

El uso de giróscopos y acelerómetros ayuda a disminuir estos errores, pero no sirven para solventarlos completamente. Para ello, deberíamos usar sensores esteroceptivos, que son aquellos que se basan en objetos del entorno para realizar las mediciones, normalmente orientados a medir una distancia, como sensores laser, sensores de ultrasonidos o cámaras de visión. También pertenecen a este grupo de sensores aquellos orientados a medir campos magnéticos terrestres (magnetómetros), radares y sistemas de posicionamiento global [6].

2.3.1 Sensor laser

El sensor laser es uno de los más usados en la robótica actual debido a su fiabilidad, así como a su rapidez. Para el uso del sensor laser existen dos posibilidades:

- TOF del inglés *time of flight*, en el cual se emite un haz de luz y se mide el tiempo hasta su recepción debido a la reflexión de este en un objeto. De esta forma, la distancia hasta el objeto será $D = \frac{1}{2}cT$ donde D es la distancia, T el tiempo desde la emisión hasta la recepción y c la velocidad de la luz.
- Medida por desfase (*phase-shift*). En este método, se emite una onda continua hacia un objetivo. Cuando la onda alcanza el objetivo, parte de ella continua por refracción y otra es reflejada, por lo que se produce un cambio en la fase de la onda reflejada. Comparándola con una onde de referencia generada por el mismo sensor podemos saber la distancia que se ha recorrido. Este método no es práctico para medir distancias grandes

Algunas de las principales ventajas de estos sensores son la rapidez en la medición, considerada de reacción instantánea, lo que permite que no sea necesario compensar el movimiento de la plataforma; la precisión en las medidas, ya que en los sensores actuales se llega a una precisión de 10mm aproximadamente; una resolución angular de entre 0,5º o 0,25º; y los datos del sensor pueden ser interpretados directamente como la distancia al objeto, no necesita de un procesamiento posterior como en el caso de las cámaras.

Sin embargo, entre sus desventajas cabe destacar que su valor de mercado es muy alto, solo da información de un plano, por lo que objetos situados por debajo o por encima no serán detectados; y algunos materiales transparentes como el cristal no son detectados.

Aun así, está considerado como el mejor sensor para extraer datos sobre superficies planas.

2.3.2 Sensor de ultrasonidos (Sonar)

Al igual que el sensor laser, es muy usado en robótica, pero en este caso debido a su precio. Suele ser muy utilizado en pequeños proyectos universitarios, aunque también tiene un amplio uso fuera de los entornos académicos. Las bases del funcionamiento son las mismas que las del sensor laser (TOF). El sonar envía una onda sónica a través del entorno y espera a su recepción.

Las principales ventajas de estos sensores son su bajo precio, una gran precisión en la medida, así como las condiciones del entorno de trabajo, ya que puede trabajar en entornos con cualquier tipo de iluminación.

Sin embargo, tienen gran cantidad de desventajas entre las que cabe destacar:

- Reflexiones múltiples: la onda sónica puede ser reflejada por más de un objeto antes de llegar al receptor, causando datos erróneos
- *Crosstalk:* Cuando trabajamos con múltiples sensores de ultrasonido puede ocurrir que los sensores reciban las ondas procedentes de otros sensores.
- Resolución angular deficiente
- Rango limitado de medida: los límites de medida de encuentran entre 0.2 y 10m por lo que no serán útiles para medir grandes distancias.
- Velocidad de propagación de la onda: La velocidad del sonido en el aire es de aproximadamente 340 m/s la cual es notablemente inferior a la de la luz por lo que no obtendremos los resultados tan rápidamente. Además, nos limita la cantidad de datos obtenidos, ya que no puede realizar mediciones al mismo ritmo que el sensor laser

Debido a esto, el empleo de estos sensores para tareas de clasificación es muy complejo. Sin embargo, existen métodos empleando más de un sensor, que permiten solventar algunas de estas desventajas. Estas técnicas son conocidas como técnicas de triangulación y se basan en reglas trigonométricas para obtener sus resultados.

2.3.3 Visión

La visión, a diferencia de los laser y el sonar, permite detector objetos en el entorno e identificarlos. Las imágenes que proporcionan, ya sean en escala de grises o a color, contienen gran cantidad de información que, una vez procesada, puede servir para identificar objetos, formas e incluso distancias si se cuenta con más de una imagen o varias cámaras.

Las principales ventajas son una gran cantidad de información proporcionada, la capacidad de obtener información en 3D y su funcionamiento como sensor pasivo, que no precisa de ordenes de emisión. Sin embargo, son dispositivos que necesitan gran potencia de computo, bastante caros y además muy influenciados por la luminosidad del entorno.

2.3.3.1 Cámara RGB-D

A parte de las cámaras convencionales existen otro tipo de cámaras que combinan un método para medir la profundidad en la imagen. Para ello, el sensor proyecta luz estructurada en una franja del infrarrojo, la cual es captada por una segunda cámara infrarroja, obteniendo así los patrones de la luz en cada pixel. Así, la imagen contendría la información de la profundidad de cada pixel de la imagen. Suelen estar especialmente destinadas a tareas de mapeo 3D, planificación de trayectorias, navegación autónoma, reconocimiento de objetos y seguimiento.

2.3.3.2 Microsoft® Kinect

El sensor Kinect es un controlador diseñado principalmente para el uso en videoconsolas, que permite a los jugadores la interacción con el videojuego sin necesidad de tener contacto físico con un controlador de videojuegos tradicional (mando).

Sin embargo, la utilidad de Kinect va mucho más allá del simple uso en videojuegos, ya que gracias a sus

propiedades es un sensor de gran utilidad para la robótica entre otros.

El sensor Kinect se compone de una cámara RGB, un sensor de profundidad y un micrófono bidireccional que, conjuntamente, capturan el movimiento de objetos 3D, detectan objetos estáticos, reconocimiento facial y acepta comandos de voz.

El rango de visión del sensor es de 1,5 a 3,5 metros de profundidad, 57,5º horizontalmente y 43,5º verticalmente, más un movimiento vertical gracias a un pivote motorizado de ± 27º.

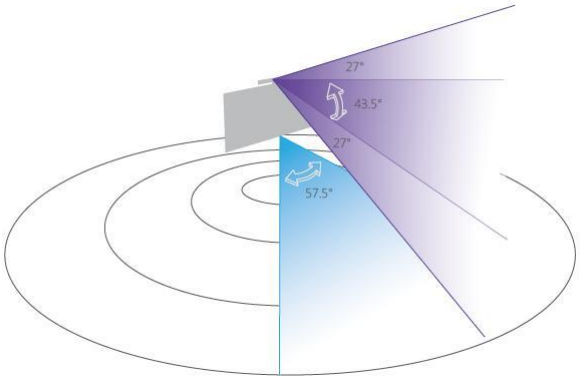

Figura 2 Rango del sensor Microsoft® Kinect

El funcionamiento se basa en un sensor infrarrojo combinado con un sensor CMOS que envía un haz de luz y puede detectar el TOF de manera que puede calcular la distancia a la que están todos los objetos situados en su campo de visión, llegando incluso a distinguir movimientos en tiempo real. La precisión en la profundidad se encuentra alrededor de 1 cm y en altura y anchura con precisión de 3 mm. Un firmware y un procesador incluidos en el sensor se encargan de analizar las imágenes y devolver todos los datos necesarios.

En noviembre del 2002, industrias Adafruit ofreció una recompensa para un controlador de código libre, la cual fue solucionada posteriormente por el español Héctor Martín, que desarrollo un controlador para GNU7Linux.

2.4 Landmarks

Los landmark o puntos de referencia son objetos o puntos del entorno que el robot utiliza para poder estimar su localización. Estos deben ser fácil de distinguir del resto de objetos o puntos del entorno. Por ejemplo, en el caso de que se utilizara una cámara, una silla no sería un buen landmark, ya que probablemente habrá otras sillas del mismo modelo alrededor que podrían causar confusión.

Es muy importante que los landmark sean reobservables y accesibles para el sensor desde diferentes posiciones ya que esto es necesario para utilizarlos para estimar su posición. La primera vez que se observa un landmark solo servirá para situarlo dentro del entorno, será la segunda vez que se vea cuando sirva como ayuda para estimar la posición actual del robot.

Es muy importante que los objetos que sean identificados como landmark sean estacionarios, ya que si se mueven de posición no servirá para estimar la posición, debido a que su posición actual después del movimiento y la que el robot tiene almacenada serán distintas [5].

Para que un objeto sea un buen candidato a ser utilizado como landmark debe cumplir los siguientes requisitos:

- Debe ser fácilmente reobservable
- Deben de ser fácilmente distinguibles de los demás
- Debe ser estacionario

Una vez se ha decidido que puntos del entorno son candidatos a ser landmark se ha de buscar algún método para recoger los datos de las salidas de los sensores. Para ello, existen muchos métodos, entre los que veremos SIFT, SURF, ORB y RANSAC.

2.4.1 SIFT (Scale-Invariant Feature Transform)

Se trata de un algoritmo utilizado en visión artificial para extraer puntos del entorno que puedan ser utilizados posteriormente como puntos de referencia para localización de objetos, detección de movimientos, estereopsis, etc.

Este algoritmo se encarga de localizar esquinas en una imagen, las cuales son puntos de gran utilidad para ser considerados como landmark; y mantenerlos invariantes a cambios de escalado.

El algoritmo consta de 4 pasos:

El primero se denomina detección de extremos en la escala-espacio, el cual se encarga de aplicar una diferencia Gaussiana buscando máximos locales a lo largo del espacio y de la escala. Es decir, para una escala dada **σ** se buscan los máximos locales comparando cada pixel con los 8 vecinos más próximos, los 9 pixeles de la escala superior y los 9 de la escala inferior. Por lo tanto, lo que se busca son máximos que sigan siéndolo ante un cambio de escala.

El segundo paso es la localización de puntos de interés. Una vez se han encontrado los puntos de interés estos deben ser refinados. Para ello se utiliza una serie de Taylor en la escala y el espacio, comparando el resultado con cierto umbral. Si el resultado es menor que el umbral el punto queda descartado. Además, como la diferencia Gaussiana también da puntos de los bordes que deben ser eliminados, se emplea una matriz Hessiana con la que se consigue, gracias a una comparación de sus autovalores, descartar los bordes.

Una vez realizados estos dos pasos se asigna una orientación a cada *keypoint*, consiguiendo así la invariancia sobre rotaciones en la imagen. Esto se consigue calculando el gradiente con cada uno de sus vecinos, y haciendo un histograma con los resultados. El mayor pico del histograma se toma como la orientación del punto de interés. Si hay algún otro pico mayor que el 80% del mayor, se toma para formar otro punto de interés en el mismo punto, pero con diferente orientación.

Posteriormente, se crean los descriptores de cada punto de interés, formado por la orientación de cada uno de sus vecinos en una vecindad de 16x16 dividida en 16 sub-bloques de 4x4.

Teniendo ya los puntos de interés, lo que queda es reconocerlos como el mismo punto en diferentes imágenes. Esto se obtiene a través de la búsqueda del vecino más cercano con los descriptores. En algunos casos, el segundo vecino más cercano puede estar muy próximo al primero por culpa del ruido, por lo que se calcula la razón al más cercano y al segundo más cercano y si no supera cierto umbral el punto es descartado [7].

2.4.2 SURF (Speeded-Up Robust Features)

Se trata de una versión creada para mejorar la velocidad de procesamiento que tenía SIFT. En SIFT se empleaban diferencias Gaussianas para sustituir al Laplaciano del Gaussiano. En SURF, se aproxima el Laplaciano del Gaussiano mediante un *Box Filter* que consiste en aproximar el valor de capa pixel de una vecindad por el valor medio de esta. La gran ventaja consiste en que la convolución de los *box filters* se calcula muy fácilmente con la ayuda de imágenes integradas.

Para localizar los puntos se interés, SURF utiliza un detector basado en el valor del determinante de la matriz Hessiana, tanto en el escalado como en la localización. Para el cálculo de la orientación se emplea una *wavelet de Haar* en la dirección x e y sobre una circunferencia de radio 6s donde s es el valor de escalado del punto de interés. Una vez obtenida la respuesta a la *wavelet* y valoradas con un gaussiano centrado en el punto de interés, estas son representadas como vectores en un espacio definido por la respuesta vertical en la ordenada y la

respuesta horizontal en la abscisa. La orientación dominante se calcula como la suma de todas las respuestas en un cono de ángulo $\frac{\pi}{3}$. Este proceso puede ser simplificado utilizando de nuevo imágenes integrales. Además, muchas aplicaciones no necesitan de invariancia sobre la rotación por lo que existe también Uprigth-SURF o U-SURF que agiliza el proceso eludiendo este paso.

Para conseguir el descriptor de los puntos de interés, se construye una región cuadrada al rededor del punto de interés de 20sx20s subdividida en regiones de 4x4. A cada subregión se le aplica una *wavelet de Haar* en x e y, cuya respuesta se introduce en un vector que forman el descriptor del punto [8].

2.4.3 RANSAC (Random sample consensus)

Es un método matemático empleado para conseguir líneas a partir de un sensor laser. Estas líneas pueden ser utilizadas como landmark. Es uno de los métodos más desarrollados en el uso de extracción de landmark con sensores laser. Es muy útil en entornos interiores, donde los muros suelen formar líneas rectas de gran interés para el mapeo.

RANSAC encuentras estas líneas tomando datos aleatorios de los resultados del láser y empleando aproximación por mínimos cuadrados para encontrar la línea que mejor se ajusta a los resultados obtenidos. Una vez se ha realizado esto, RANSAC comprueba cuantos resultados corresponden a la misma línea. Si el número es superior a cierto umbral (llamado consensus) se puede asumir que hemos encontrado una línea recta [5].

Este algoritmo es muy parecido al empleado en los sistemas de percepción llamado *transformada de Hough*, el cual, mediante el uso de coordenadas polares, examina todas las rectas que pasan por cada punto del contorno de un objeto dentro de una imagen. Una vez hecho esto, se mide cual es la recta más votada, es decir, la recta que más puntos tienen en común y se asume que es una recta perteneciente al contorno.

2.4.4 ORB (Oriented FAST and Rotated BRIEF)

ORB es la fusión de los mecanismos FAST para detector los puntos de referencia y BRIEF para declarar los descriptores de dichos puntos. Primero, se emplea FAST, que se encarga de buscar los puntos de referencia empleando el siguiente algoritmo [9]:

- Se selecciona un punto que puede ser identificado como un punto candidato o no, cuya intensidad será I_p.
- Se elige un valor umbral t.
- Se toma un circulo de 16 pixeles alrededor del pixel seleccionado.

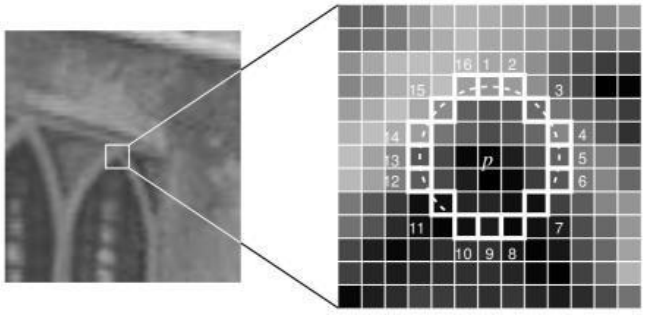

Figura 3 Ejemplo de selección de pixel y circunferencia circundante

- Una vez se ha tomado el pixel se comprueba la intensidad luminosa de los pixeles pertenecientes a la circunferencia. Se toman n pixeles contiguos pertenecientes a la circunferencia y si se comprueba si su intensidad luminosa es mayor que $I_p + t$ o menor que $I_p – t$. Si esto se cumple, nos encontramos con una esquina, candidato a ser punto de referencia.
- Para agilizar el proceso de excluir puntos que no sean esquinas se puede realizar un test rápido. En él se comprueba la intensidad luminosa de los pixeles 1,9,5 y 13 indicados en la Figura 3. Entonces, para que el pixel seleccionado pudiera ser una esquina al menos 3 de los pixeles anteriormente mencionados deben cumplir el umbral establecido.

Una vez empleado FAST para encontrar los puntos de referencia, se aplica *Harris Corner* para buscar los N mejores puntos de referencia entre los seleccionados. También se usa *piramid* para producir características multiescala. Para asegurar la invariancia a la rotación de esto puntos de referencia se aplica el siguiente método. Se localiza el centroide de la intensidad luminosidad de una zona cuyo centro será la esquina que tomamos como punto de referencia. La dirección del vector que une el punto de referencia con el centroide será la orientación. Para mejorar la invariancia, se buscan los momentos en x e y en una región circular del tamaño de la zona.

Para conseguir el descriptor, este algoritmo se basa en BRIEF pero con algunas modificaciones para mejorar la rotación. BRIEF proporciona un atajo frente a otros descriptores, ya que proporciona las cadenas binarias directamente sin necesidad de buscar primero el descriptor. El método para ello consiste en tomar pares de puntos en una única dirección, y comparando sus intensidades luminosas se declara un 1 o un 0 en el descriptor. Esto se aplica para los n_d pares consiguiendo así un descriptor de dimensión n_d. Para mejorar los problemas que tiene BRIEF con la rotación, ORB dirige BRIEF en la orientación de los puntos de referencia. Para los n_d pares de localizaciones se define una matriz S de dimensiones $2 \times n_d$ que contiene las coordenadas de cada punto del par. Entonces, usando la orientación del punto de referencia se rota la matriz S para obtener su versión rotada S_θ [10].

2.4.5 Asociación de datos

Una vez se han obtenido los landmark del sistema se debe buscar algún método para poder garantizar que cuando volvamos a ver el landmark este sea el mismo que vimos en primera instancia. Para ello, se deben establecer una serie de reglas que deberán de ser cumplidas antes de considerar un punto como landmark, ya que en caso de que no se de una correcta correspondencia de los datos se provocará un error fatal en el sistema, debido a que el robot pensará que se encuentra en una posición distinta de la real.

La primera regla será que no consideraremos un punto como landmark (es decir, no lo incluiremos en el vector de estados) hasta que este sea reobservado N veces. Así garantizamos que se trata de un punto que podemos volver a ver. Si tomamos como landmark un punto que no conseguimos volver a ver lo único que aportará al sistema será requerirle más potencia de computo, al estar haciendo más grande el vector de estado con un dato redundante. Uno de los métodos para ello se denomina *nearest-neighbor approach*, acercamiento al vecino más cercano en español, que se trata de comparar el landmark que se ha encontrado con el vecino más cercano, calculando la distancia a él, ya sea la de Mahalanobis o la Euclídea, y compararla con un umbral. Si se supera el umbral se da por sentado que es el mismo landmark que el más cercano y se incrementa el número de veces observado en la base de datos; si no lo supera, se toma como un nuevo landmark y se indica que es la primera vez observado.

3 EKF-SLAM

El filtro extendido de Kalman fue el primer algoritmo implementado en el problema de SLAM ya que muchos investigadores empleaban este método probabilístico para modelos de navegación en robot móviles, lo que llevo a un acercamiento en el campo del mapeo y la localización.

El EKF está basado, como su nombre indica, en el filtro de Kalman, algoritmo desarrollado por el investigador Rudolf E. Kálmán. En 1960 durante una visita de Kálmán a la NASA, Stanley F. Schmidt descubrió la utilidad que este podría tener para problemas de estimación de trayectorias no lineales. Sin embargo, el filtro de Kalman estaba destinado a estimaciones de sistemas lineales, de donde apareció el filtro extendido de Kalman, el cual se encarga de linealizar mediante series de Taylor multivariables los modelos no lineales. Gracias a las aportaciones de Schmidt, el filtro fue incorporado en el sistema de navegación del proyecto Apollo.

El EKF se encarga de estimar la posición del robot a través de la odometría y los landmark. Sin embargo, en SLAM el EKF debe realizar esto teniendo a su vez una estimación del mapa en el que se mueve, teniendo así una incertidumbre en todos los datos. Aun así, una vez que se modifican los valores de las matrices, el resultado y el desarrollo es el mismo que en el caso en que solo se ha de estimar el estado.

EL EKF se puede dividir en 3 pasos:

- Actualización del estado del robot utilizando la odometría
- Actualización del estado estimado usando landmark reobservados
- Inicialización de nuevos landmark.

Además, también hay que tener en cuenta un paso previo en el que se han inicializado las variables e introducido los primeros landmark mediante algún método de obtención de landmark.

3.1 Matrices

Como hemos dicho previamente, una vez modificadas las matrices de manera correcta el filtro de aplica de la misma forma que un EKF normal [5] [11].

3.1.1 Vector de estados

Es uno de los vectores más importantes del sistema junto con la covarianza ya que contiene la información de la posición del robot, así como la de todos los landmark. Es además el vector del que obtenemos el mapa del entorno.

$$x = \begin{bmatrix} X \\ M \end{bmatrix} = \begin{bmatrix} X \\ L_1 \\ \vdots \\ L_n \end{bmatrix}$$

Donde X es el estado del robot, que incluye las posiciones, así como las orientaciones; y $M = (L_1, \dots, L_n)$ son las posiciones de los landmark (incluyendo todas las componentes necesarias).

Este mapa estará modelado mediante una variable Gaussiana usando la estimación del vector de estado y de la matriz de covarianza.

3.1.2 Matriz de covarianza

La covarianza se encarga de medir la correlación que presentan dos variables. En nuestro caso, la matriz de covarianza, P, es fundamental en el sistema ya que contendrá toda la información entre la dependencia lineal entre las variables de estado del sistema, estas con los landmark y de los landmark entre si.

$$P = \begin{bmatrix} P_{XX} & P_{XM} \\ P_{MX} & P_{MM} \end{bmatrix} = \begin{bmatrix} P_{XX} & P_{XL_1} & \cdots & P_{XL_n} \\ P_{L_1X} & P_{L_1L_1} & \cdots & P_{L_1L_n} \\ \vdots & \vdots & \ddots & \vdots \\ P_{L_nX} & P_{L_nL_1} & \cdots & P_{L_nL_n} \end{bmatrix}$$

Inicialmente, la matriz solo contendrá P_{XX} [11] ya que no se habrá visto ningún landmark. Esta se inicializará como una matriz diagonal dando información sobre los posibles errores en la posición inicial. Es importante incluir algo de error en el estado inicial pese a que se piense que es completamente exacto. Normalmente, se inicializará la matriz P_{XX} utilizando los errores en la medida de la posición al cuadrado [5].

3.1.3 La ganancia de Kalman K

La ganancia de Kalman se utiliza como un filtro de los resultados obtenidos de la odometría y de los landmark. De esta forma, nos dará un resultado de la confianza que le aplicamos a los datos del modelo de nuestro robot o de los sensores esteroceptivos.

Dicho con otras palabras, se encarga de dar más importancias dentro del algoritmo a los datos obtenidos o bien de la odometría o bien de los landmark. Si la ganancia de Kalman tiene un valor bajo significará que los resultados obtenidos de la odometría son mucho más fiables que los obtenidos de los sensores de medida. Por el contrario, una ganancia muy alta indicaría que los sensores de medida nos dan resultados más correctos que la odometría del sistema. En este último caso, el algoritmo daría más peso a los resultados obtenidos de los landmark de forma que minimizaría la aportación a la hora de estimar la posición de los datos de la odometría y daría mayor importancia a los obtenidos por los landmark [5].

3.1.4 El Jacobiano del modelo de medida H

Lo primero es definir cómo se comporta el modelo de los sensores de medida, que se encargara de dar un resultado útil al algoritmo de los datos obtenidos por el sensor de medida. Para un sistema 2D el modelo sería el siguiente:

$$\begin{bmatrix} alcance \\ orientación \end{bmatrix} = \begin{bmatrix} \sqrt{(\lambda_x - x)^2 + (\lambda_y - y)^2} + v_r \\ \tan^{-1}\left(\frac{\lambda_y - y}{\lambda_x - x}\right) - \theta + v_\theta \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}$$

Donde x e y representan las posiciones del robot, θ la orientación del robot y λ_x y λ_y las posiciones de los landmark. Esto nos predice los resultados que deberían dar los sensores.

El jacobiano H será la matriz compuesta por las derivadas parciales del modelo de medida de los sensores más las derivadas con respecto a las posiciones del landmark utilizado. Esta matriz nos da como resultado cuanto cambian el alcance y la orientación conforme cambia la posición del robot. De esta forma, nuestra matriz H tomará la siguiente forma:

$$H = \begin{pmatrix} \frac{\partial h_1}{\partial x} & \frac{\partial h_1}{\partial y} & \frac{\partial h_1}{\partial \theta} & \frac{\partial h_1}{\partial x_{L_1}} & \frac{\partial h_1}{\partial y_{L_1}} & \cdots & \frac{\partial h_1}{\partial x_{L_n}} & \frac{\partial h_1}{\partial y_{L_n}} \\ \frac{\partial h_1}{\partial x} & \frac{\partial h_1}{\partial y} & \frac{\partial h_1}{\partial \theta} & \frac{\partial h_2}{\partial x_{L_1}} & \frac{\partial h_2}{\partial y_{L_1}} & \cdots & \frac{\partial h_2}{\partial x_{L_n}} & \frac{\partial h_2}{\partial y_{L_n}} \end{pmatrix}$$

El valor de $\frac{\partial h_1}{\partial x_{L_k}}$ así como para h_2 solo será distinto de 0 en el caso de que se trate del landmark que estamos observando con el sensor y que estamos utilizando en el modelo de media. En ese caso $\frac{\partial h_1}{\partial x_{L_k}} = -\frac{\partial h_1}{\partial x}$. Es igual para la medida en la posición y, y para h_2.

3.1.5 El jacobiano del modelo de predicción A

Al igual que en el caso del jacobiano H, primero debemos ver cómo se comporta el modelo de predicción (modelo del robot móvil).

Para un modelo 2D, siendo x e y las posiciones del robot y θ la orientación del robot en el instante t, en el instante t+dt tendremos que el robot se habrá movido una cantidad dx, dx y habrá rotado dθ de forma que nuestro modelo quedará como:

$$f = \begin{bmatrix} x + dx + dx * q \\ y + dy + dy * q \\ \theta + d\theta + d\theta * q \\ x_{L_1} \\ y_{L_1} \\ \vdots \\ x_{L_n} \\ y_{L_n} \end{bmatrix}$$

Siendo q el ruido del proceso que veremos más adelante. De igual manera que el caso anterior, la matriz A será el resultado de las derivadas parciales del modelo y de los landmark del sistema:

$$A=\begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial \theta} & \frac{\partial f_1}{\partial x_{L_1}} & \frac{\partial f_1}{\partial y_{L_1}} & \cdots & \frac{\partial f_1}{\partial x_{L_n}} & \frac{\partial f_1}{\partial y_{L_n}} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial \theta} & \frac{\partial f_2}{\partial x_{L_1}} & \frac{\partial f_2}{\partial y_{L_1}} & \cdots & \frac{\partial f_2}{\partial x_{L_n}} & \frac{\partial f_2}{\partial y_{L_n}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\partial f_{2n+3}}{\partial x} & \frac{\partial f_{2n+3}}{\partial y} & \frac{\partial f_{2n+3}}{\partial \theta} & \frac{\partial f_{2n+3}}{\partial x_{L_1}} & \frac{\partial f_{2n+3}}{\partial y_{L_1}} & \cdots & \frac{\partial f_{2n+3}}{\partial x_{L_n}} & \frac{\partial f_{2n+3}}{\partial y_{L_n}} \end{bmatrix}$$

Aunque parezca una matriz muy extensa, cuando se realizan las derivadas parciales se simplifica en gran medida, ya que todos los landmark que no estén derivados por ellos mismos darán un 0 en el resultado. También puede evitarse introducir todos los landmark, ya que solo se utilizará para la estimación del estado del robot y podríamos prescindir del resto de datos utilizando posteriormente solo la parte superior que corresponde al estado del robot de la matriz de covarianza P.

3.1.6 Los Jacobianos específicos de SLAM J_{xr} y J_z

Para realizar SLAM con un EKF hay algunas cosas que cambiar con respecto a un estimador de estado normal, y una de estas cosas es el incluir nuevos landmark a la matriz de covarianza. Tendremos entonces por un lado el jacobiano del modelo de predicción de los landmark, tanto en términos de posición (Jxr) como de los resultados que nos indican los sensores de medida (J_z).

Estos son como los jacobianos del modelo, pero desde el punto de vista de cada uno de los landmark con respecto al robot, tanto en coordenadas como en los datos del sensor. Como los landmark no tienen orientación, no será necesario la derivada parcial de esta variable.

$$J_{xr}=\begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial \theta} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial \theta} \end{bmatrix}$$

$$J_z=\begin{bmatrix} \frac{\partial f_1}{\partial h_1} & \frac{\partial f_1}{\partial h_2} \\ \frac{\partial f_2}{\partial h_1} & \frac{\partial f_2}{\partial h_2} \end{bmatrix}$$

Hay que tener en cuenta en este paso la transformación de las coordenadas que nos da el sensor a coordenadas

cartesianas ya que, de esta forma, tenemos las siguientes relaciones:

$$x = h_1 \cos(\theta)$$
$$y = -h_1 \sin(\theta)$$

Que nos permitirán simplificar las derivadas.

3.1.7 El ruido del proceso y de la medida

El sistema tendrá unos ruidos asociados a los errores estimados que puedan aparecer a causa de la odometría y del sensor de medida. Estos se integran en el algoritmo, introduciendo un error estimado a corregir por el sistema que nos dará una muy buena aproximación al error real cometido, por lo que la corrección será aceptable.

En el algoritmo, se introducirán las covarianzas Q y R, que estarán formadas por el valor cuadrático de los ruidos (formulados como variables gaussianas) multiplicados por una constante que indicará como de precisas son las medidas y la odometría.

3.2 Algoritmo

El algoritmo constará de 3 pasos. En el primero de ellos se hará una predicción de la posición del robot mediante el empleo de la odometría, lo que creará una incertidumbre en la verdadera posición del robot que se puede formular como una elipse alrededor de la posición del robot. Una vez realizado, se pasa a la fase de corrección, en la que se emplea un filtro mediante la constante de Kalman y la innovación con los landmark que el robot haya visto de nuevo. Este paso se realizará por cada landmark visto. Finalmente, se deben incluir los landmark nuevos que el robot haya visto y que sean válidos al vector de estados y a la matriz P

3.2.1 Etapa de Predicción

Lo primero que se debe hacer es actualizar el vector de estados mediante los datos de la odometría. Para ellos se formula:

$$x \leftarrow f(x, u, q)$$

Siendo u la señal de control del robot (que proporcionan dx, dy y dθ) y q la perturbación debida al ruido. Una vez actualizado el vector de estados, se debe actualizar la matriz de covarianza.

$$P = APA^T + Q$$

La matriz Q solo contendrá información útil en la primera submatriz 3x3 de dentro de ella por lo que solo afectará a la covarianza del robot, no a las covarianzas cruzadas de los landmark.

3.2.2 Etapa de Corrección

Una vez actualizado el estado mediante la odometría se debe corregir utilizando los landmark del sistema, ya que la predicción no será correcta del todo debido a los errores de la odometría. Esto se realizará de la siguiente

forma: Extrayendo los datos obtenidos del sensor de medida y conociendo la posición de los landmark dentro del mapa que se ha ido formando se puede conocer la posición en la que el robot debería estar dentro del mapa, la cual puede ser distinta de la posición en la que el robot piensa que esta debido a la etapa de predicción.

Este paso debe ser repetido tantas veces como landmark visibles tenga el robot.

Utilizando las posiciones guardadas de los landmark (λ_x y λ_y), y las posiciones estimadas del robot se puede estimar mediante la fórmula presentada anteriormente el alcance y la orientación que debería darnos el sensor de medida. Este valor, comparado con el real obtenido por el sensor conformarán la variable z que da una indicación sobre como de correcto ha sido el posicionamiento, según los datos del sensor.

$$z = h(sensor) - h(x) + v$$

siendo v el ruido en la medida.

Empleando la matriz H como denotamos anteriormente para el landmark que estamos se usa actualmente (donde el resto de valores serán 0), se obtiene la ganancia de Kalman como:

$$K = PH^T (HPH^T + R)^{-1}$$

Finalmente, se actualiza el valor del vector de estados y de la matriz de covarianza corregido mediante la ganancia de Kalman:

$$x \leftarrow x + K * z$$
$$P \leftarrow P - K(HPH^T + R)^{-1} K^T$$

3.2.3 Integración de nuevos landmark en el vector de estados

En el último paso se incluirán los nuevos landmark que sean adecuados para el sistema. Para ello se debe actualizar la matriz de covarianza P y el vector de estados. En el caso del vector de estados, lo único que se debe hacer es incluir los nuevos valores de las posiciones de los landmark al final del vector. Para actualizar la covarianza se emplearán los jacobianos específicos de SLAM que vimos en un apartado anterior. Lo primero que se debe hacer es actualizar la covarianza del ultimo landmark consigo mismo. Este valor será el landmark N+1:

$$P_{LL} = J_{XR} P_{XX} J_{XR}{}^T + J_Z R J_Z{}^T$$

Ahora se deben añadir las covarianzas cruzadas con el resto del mapa:

$$P_{LX} = J_{XR} P_{XK} = J_{XR} \begin{bmatrix} P_{XX} & P_{XM} \end{bmatrix}$$

De esta forma la nueva matriz de covarianza de actualiza como:

$$P = \begin{bmatrix} P & P_{Lx}{}^T \\ P_{Lx} & P_{LL} \end{bmatrix}$$

4 PLATAFORMAS EMPLEADAS

Son muchas las plataformas que se pueden emplear para realizar SLAM, pero una de las más desarrolladas para la implementación de sistemas en robóticos es ROS. Esto se debe a que se trata de un middleware para sistemas robóticos, de formulación modular y muy fácil de implementar en entornos Linux. Además, se trata de un entorno de código abierto totalmente gratuito, con gran cantidad de herramientas para el desarrollo de aplicaciones en sistemas reales como para el desarrollo de simulaciones.

4.1 ROS (Robot Operating System)

ROS o sistema operativo robótico es un sistema de código abierto desarrollado para aprovechar el uso de códigos existentes de un repositorio con el fin de realizar tareas de investigación. Se trata de un framework que incluye funciones de sistema operativo, como paso de mensajes, control de sistemas de bajo nivel, separación de hardware, y administración de paquetes. Además, también incluye herramientas para escribir y desarrollar códigos en un entorno múlti-plataforma.

ROS se origina en 2007 tomando diferentes investigaciones de framework para robot de software libre por el Laboratorio de Inteligencia Artificial de Stanford para el proyecto STAIR (Stanford AI Robot). A partir del 2008, el desarrollo se centra en el laboratorio de investigación de robótica Willow Garage, cuando Eric Berger y Keenan Wyrobek dejaron Stanford para comenzar el programa de robótica personal en Willow Garage. Durante este tiempo, investigadores de más de 20 instituciones colaboraron en desarrollos de modelos federales. En febrero de 2013, ROS comienza el movimiento hacia un software de código abierto (Open Source Robotic Fundation, OSRF) y, será en agosto de ese año cuando se anuncie que Willow Garage será absorbido por una empresa creada por sus fundadores, Suitable Technologies.

ROS está basado en una red de conexiones multidireccional distribuida entre distintos dispositivos que se comunican mediante la infraestructura de comunicaciones de ROS. Está estructura cuenta con diferentes estilos de comunicación.

El principal objetivo de ROS es el de integrar código reusado en nuevos proyectos de investigación y desarrollo. Para ello, se compone de procesos agrupados en paquetes que pueden ser fácilmente compartidos y distribuidos. De esta forma, pequeños fragmentos de código que han sido utilizados para un proyecto se pueden utilizar en uno completamente distinto o ser utilizados como base.

Otros de los objetivos de ROS son un diseño de código corto, de forma que un paquete este compuesto por muchos fragmentos de código corto los cuales pueden ser utilizados individualmente para otras cosas; independencia de lenguajes, ya que se puede implementar actualmente en Python , C++ y Lisp, además de algunas pruebas experimentales en Java; testeado sencillo, gracias a una herramienta llamada rostest; y escalado, entendido como la capacidad de completar un proyecto de largas dimensiones y de largo tiempo de ejecución mediante la unión de paquetes integrados en un sistema.

Actualmente, ROS solo puede ser ejecutado en sistemas operativos con entornos UNIX, aunque principalmente es utilizado en sistemas Ubuntu y Mac OS X.

4.1.1 Sistema de Archivos de ROS

El Sistema de archivos de ROS estará distribuido en las siguientes capas:

- Paquetes: Es la unidad principal para organizar software en ROS. Pueden contener rutinas de ejecución de procesos (nodos), librerías dependientes de otros procesos, conjuntos de datos, ficheros de configuración, etc. Son la parte más atómica de ROS.
- Metapaquetes: Son paquetes especializados para una función. Están compuestos de por un conjunto de paquetes relacionados.
- *Package Manifest:* contienen información básica sobre el paquete, como su nombre, las licencias, dependencias, etc.
- Repositorios: Son un conjunto de paquetes que comparten un VCS system en común.
- Tipos de Mensajes (*message types*): define la estructura de los mensajes en ROS
- Tipos de Servicios (*Service types)*: define las estructuras de recogida y respuestas para los servicios en ROS,

4.1.2 Sistema de Cómputo ROS

El sistema de cómputo de ROS está basado en un grafo formando una red bidireccional de procesos que procesan datos juntos. Los principales conceptos de este grafo son:

- Nodos: Los nodos son los procesos que se están ejecutando. Estos nodos conforman una estructura modular del sistema, de forma que cada parte del robot está dividida en nodos que se ejecutan de manera individual pero que están comunicados por los Topics (que veremos más adelante).
- Master: Se encarga de controlar el proceso completo, de forma que gracias a él los nodos pueden comunicarse entre ellos, intercambiar mensajes e invocar servicios.
- Servidos de parámetros: se encarga de almacenar la información en una localización central. Es una parte del Master.
- Mensajes: Es la forma de comunicación de los nodos. Es simplemente una estructura de datos. Pueden incluir tipos de datos primitivos y cadenas de este tipo de datos.
- Topics: Los mensajes se transportan bajo un sistema de publicador/subscriptor. Los nodos envían mensajes publicándolos en un topic. El topic es un nombre para identificar el mensaje. Los nodos subscritos a dicho topic tendrán acceso a la información publicado por otros nodos en ese topic. Se trata por tanto del medio por el que los nodos pueden enviar mensajes de unos a otros. Para entenderlo de forma más clara podemos ver el caso de un cartero. El cartero se encarga de transportar el mensaje de un punto hasta otro, independientemente de la persona que lo envíe, la que lo reciba o el mensaje en si mismo. Cada nodo puede publicar y estar subscrito a varios topic sin ningún problema, ya que lo que se pretende es separar la producción de información con su consumo. Puede entenderse un topic como un bus de mensajes fuertemente definido.
- Servicios: Los servicios se encargan de las comunicaciones a la hora de hacer peticiones y entregar respuestas. En los topics, los nodos se encargan de publicar información, que puede ser leída por un subscriptor o no. En el caso de los servicios lo que se encarga de es hacer peticiones que necesitan de una respuesta, por lo que existe una dependencia real entre los dos nodos a través de los servicios. De esta forma, un nodo ofrece un servicio bajo un nombre y será otro nodo el que haga una petición a ese servicio mediante un mensaje determinado y se quede esperando a la respuesta.
- Bags: Se tratan de una forma de guardar los mensajes publicados en un topic y repetirlos si fuera necesario. Son un medio muy importante para guardar datos, como datos de sensores, que puede ser difícil de recoger pero que es necesario para realizar pruebas.

La estructura del sistema de cómputo de ROS es la siguiente: El Master se encarga de dar un nombre al sistema. Este guarda la información de registro de los nodos a los topics y a los servicios. Asimismo, los nodos se comunican con el master para indicar su registro, de quien obtienen la información necesaria de los otros nodos

y lo nuevos que se incorporan a él, para establecer las conexiones de manera adecuada. Por otro lado, los nodos se conectan directamente entre ellos. El master solo proporciona la información sobre los topics y servicios a los que está conectado cada nodo. El protocolo de conexión empleado entre nodos más usual en ROS se llama TCPROS, basado en socket estándar TCP/IP.

El hecho de que los nodos estén utilizando un medio intermedio de comunicación como son los topics permiten que el sistema esté completamente desacoplado, ya que un nodo se subscribe a un topic sin saber si alguien está publicando o publica sin saber si nadie está subscrito. Esto permite que los nodos puedan ser iniciados, reiniciados o destruidos sin necesidad de introducir ningún error. En esta arquitectura, los nombres son un de cada parte del sistema son fundamentales, ya que permiten la reconfiguración instantánea del sistema sin inducir a errores.

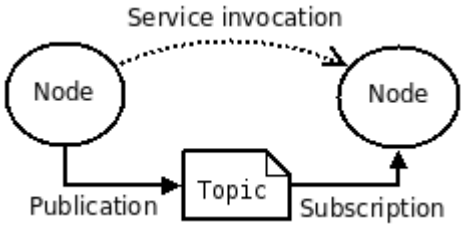

Figura 4 Computation Graph ROS

4.1.3 Herramientas de ROS

ROS cuenta con muchas herramientas para el desarrollo de proyectos de robótica, entre las que veremos dos fundamentales para el desarrollo de pruebas, ya que nos permiten hacer uso de simulaciones, o para el desarrollo de aplicaciones gráficas, como creación de mapas y localización. Las dos fundamentales que son las que se desarrollarán a continuación son rqt y rviz.

4.1.3.1 Rviz (visualizador 3D)

Se trata de un paquete de ROS que contiene un visualizador que permite mostrar en tiempo real la posición actual de tu robot, nubes de puntos o incluso los sistemas de visión del mismo. Además de las prestaciones que otorga el visualizador, también nos permite conocer los errores que se han producido en los sistemas de visión de manera dinámica, una visualización 3D navegable, un creador interactivo del entorno de forma de nuevo dinámica y la posibilidad de mostrar diferentes procesos en una misma ventana de Rviz [12].

4.1.3.2 Rqt

Se trata de una herramienta de ROS que permite el desarrollo de GUI en forma de *plugins*. Los que permite rqt es un fácil manejo de las diferentes opciones que te da ROS mediante una interfaz gráfica mucho más cómoda. Además, alguno de los *plugins* también permite visualizar de forma gráfica las conexiones establecidas entre nodos y topics, así como los servicios requeridos. Rqt también ofrece la posibilidad de que el usuario cree su propio *plugin* mediante los lenguajes de programación C++ o Python.

Rqt se conforma de 3 metapaquetes: rqt, que contiene el núcleo de la herramienta; rqt_common_plugins, que contiene los *plugins* más usuales de ROS; y rqt_robot_plugins, que contiene algunos *plugins* útiles para el desarrollo de aplicaciones destinadas al uso en robot.

4.1.3.3 Visualizador Gazebo (independiente de ROS)

El visualizador gazebo es un sistema que no pertenece a ROS, pero sobre el que se ha realizado una gran adaptación a este sistema, ya que incluso se descarga conjuntamente cuando instalas ROS. Se trata de un visualizador más avanzado, destinado a las simulaciones de robot en entornos 3D con gran cantidad de detalles y facilidades para la creación de los entornos de simulación y los modelos simulados.

El desarrollo de Gazebo comienza en 2002 en la universidad de California del Sur, de la mano de Dr. Andrew Howard y su estudiante Nate Koenig. Los que se buscaba era un simulador de gran calidad para entornos exteriores bajo diferentes condiciones. El nombre fue escogido en referencia a los patios exteriores en jardines, llamados gazebos, ya que se trataba de un simulador de ambientes exteriores. Pese a que la mayoría del desarrollo actual se encuentra en ambientes interiores, el nombre a permanecido siendo el mismo.

Durante los siguientes años, Nate continuo con el desarrollo del proyecto y será en 2009 cuando John Hsu, un ingeniero de Willow Garage, integró ROS y PR2 en Gazebo, haciendo que Gazebo se convirtiera en la herramienta más usada de ROS. En 2012, se forma Open Source Robotics Foundation procedente de Willow Garage y se encarga del proyecto de Gazebo.

Figura 5 Símbolo de Gazebo

Gazebo permite la posibilidad de realizar pruebas de algoritmos, diseño de robot, y entrenamiento de sistemas de inteligencia artificial de manera rápida y sencilla usando entornos realísticos. Además, ofrece la posibilidad de simular poblaciones de robot trabajando de forma simultánea en complejos entornos interiores y exteriores. Asimismo, incluye un motor físico muy realista, alta calidad gráfica e interfaces gráficos y de programación [13].

4.2 Ubuntu/Linux

Como se ha dicho anteriormente, ROS esta optimizado para su uso en entornos Linux y Mac OS X. En este proyecto solo vamos a hablar de Ubuntu/ Linux. Se trata de un sistema operativo de tiempo real basado en las normas UNIX y en Debian, orientado principalmente a ordenadores personales, aunque recientemente se han realizado desarrollos para smartphones y tablets. La palabra Ubuntu procede de una antigua palabra africana cuyo significado era *"humanity to others"* y a su vez *"Soy quien soy por quien somos nosotros"*. Es este espíritu el que quiere dar Ubuntu al mundo de los ordenadores, desarrollando así un entorno libre para ordenadores personales, con un desarrollo cuidado y las máximas cualidades que pueda aportar el proyecto.

En 2004, Linux estaba establecido ya como una empresa, pero el software libre no era una de sus prioridades. Fue entonces cuando Mark Shuttleworth reunió un pequeño equipo de desarrolladores y se encargaron de desarrollar un entorno fácil de usar basado en Debian: Ubuntu.

5 HECTOR-SLAM

Hector SLAM es un paquete perteneciente a ROS que presenta un mapeo basado en rejilla o celdillas con bajos requerimientos computacionales. Combina un exhaustivo escaneo usando un sensor laser LIDAR con una estimación de la posición del sistema tridimensional basado en sensores de inercia.

Este algoritmo está planteado para lugares cerrados y de pequeñas dimensiones donde no es necesario cerrar bucles demasiado grandes y donde la gran cantidad de datos extraídos del LIDAR pueden resultar beneficiosos.

Para realizar la localización y mapeo simultáneos se emplea un SLAM bidimensional basado en los resultados del láser en un mapa plano y sistema de navegación integrado tridimensional basado en una unidad de medida inercial (IMU) que incorpora la información del SLAM bidimensional como fuente para proveer información.

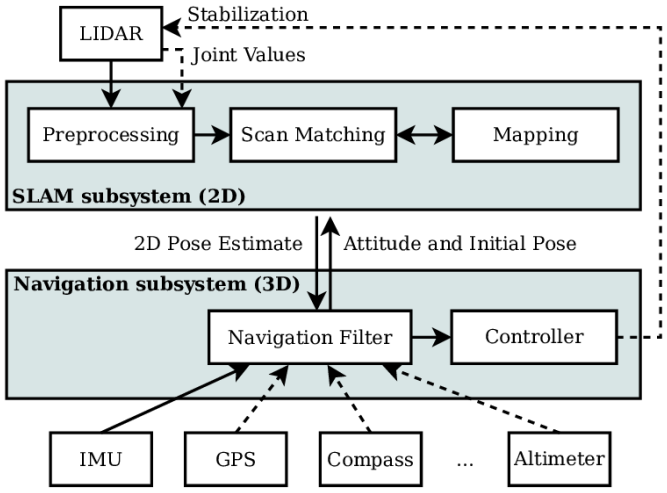

Figura 6 diagrama de bloques del sistema de navegación y mapeo

Normalmente, en los algoritmos de SLAM se hace una distinción entre la parte trasera y la delantera del sistema (frontend and backend system). La parte delantera suele encargarse de estimar el movimiento del robot en tiempo real mientras la parte trasera se encarga de optimizar el grafo de posición. Sin embargo, este sistema no presenta grafo de posición por lo que no cuenta con dicha parte trasera ya que en muchas situaciones se puede prescindir de este bajo las condiciones del mundo real debido a que el sistema es lo suficientemente adecuado para el propósito al que se destina.

La localización usando el escaneo del entorno mediante el láser comienza con el uso de la búsqueda iterativa del punto más cercano (ICP).

El modelo del sistema consiste en una plataforma que presenta 6 grados de libertad. Para conseguir este, el sistema se basa en dos componentes principales. Un filtro de navegación que fusiona la información de la IMU y otros sensores para formar una solución 3D consistente, mientras el 2D SLAM se emplea para suministrar la posición y la información principal. El estado 3D se caracteriza mediante el vector $x = (\Omega^T, p^T, v^T)$ donde $\Omega = (\phi, \theta, \varphi)$ son el roll, el pitch y el yaw, y p y v son las 3 componentes de la posición y la velocidad respectivamente. La mediada de la inercia se compone del vector de entrada $u = (\omega^T, a^T)$, que se corresponden con la velocidad angular y las aceleraciones. Con todo esto, el movimiento de un sólido rígido tridimensional se define con el siguiente conjunto de ecuaciones diferenciales no lineales:

$$\dot{\Omega} = E_{\Omega} \cdot \omega$$
$$\dot{p} = v$$
$$\dot{v} = R_{\Omega} \cdot a + g$$

Donde R_Ω y E_Ω son matrices encargadas de cambiar las coordenadas de cada sistema a las correspondientes y g es la constante de gravedad. Debido al ruido en los sensores, la integración de la velocidad y la posición presentan un fallo acumulativo. Para solucionar esto es necesario emplear información adicional. En este sistema, la información empleada es mediante el sensor láser [14].

Una vez explicado al completo el sistema, se puede presentar su funcionamiento conjunto con la ayuda de la figura 6: Los datos del LIDAR son recogidos por el sistema y procesados para tomar puntos de referencia y generar el mapa. Gracias a esto, se obtiene una estimación de la posición bidimensional que es empleada como punto de partida para el filtro de navegación. Este filtro, incorporando la información sobre otros sensores, genera la posición con sus 6 grados de libertada, que es entregada al controlador para que determine su nuevo movimiento. Asimismo, el filtro de navegación dotará al *frontend* SLAM de la posición inicial, así como de la altura para la generación del mapa y la extracción de landmark. El funcionamiento de las diversas partes del sistema es explicado en los siguientes puntos.

5.1 2D SLAM

Para representar el entorno en el que se sitúa el robot se emplea un mapa de rejilla. Ya que se ha escogido un modelo con 6 grados de libertad, los resultados del escaneo del sensor láser deben ser transformados a un sistema de coordenadas local usando la orientación estimada del sensor laser [14].

5.1.1 Acceso al mapa

Debido a la naturaleza discreta del mapa empleado, ya que estará compuesto por celdillas; que limita la precisión que se podría conseguir y además no permite la interpolación ni derivación de valores, se emplea una interpolación basada en subceldillas para la estimación de la probabilidad de tener una celdilla ocupada y para la derivación. Tomando una coordenada continua del mapa P_m, el valor de esta en nuestro mapa $M(P_m)$ así como el gradiente $\nabla M(P_m)$ pueden ser estimados usando los 4 enteros más cercanos.

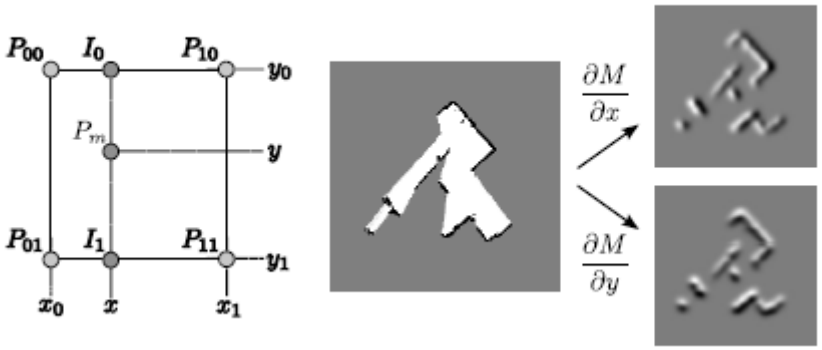

Figura 7 Aproximación empleando los 4 enteros más cercanos

Como vemos en la Fig. 7, tomando los valores de las posiciones de los cuatro enteros más cercanos podemos hacer una aproximación del valor correspondiente en el mapa al punto que buscamos, P_m y sus derivadas.

Las ecuaciones necesarias para ello serían:

$$M(P_m) \approx \frac{y-y_0}{y_1-y_0}\left(\frac{x-x_0}{x_1-x_0}M(P_{11})+\frac{x_1-x}{x_1-x_0}M(P_{01})\right)+\frac{y_1-y}{y_1-y_0}\left(\frac{x-x_0}{x_1-x_0}M(P_{10})+\frac{x_1-x}{x_1-x_0}M(P_{00})\right)$$

$$\frac{\partial M}{\partial x}(P_m) \approx \frac{y-y_0}{y_1-y_0}\left(M(P_{11})-M(P_{01})\right)+\frac{y_1-y}{y_1-y_0}\left(M(P_{10})+M(P_{00})\right)$$

$$\frac{\partial M}{\partial y}(P_m) \approx \frac{x-x_0}{x_1-x_0}(M(P_{11})-M(P_{01}))+\frac{x_1-x}{x_1-x_0}(M(P_{10})-M(P_{00}))$$

Estas aproximaciones se emplearán para calcular los resultados de los escaneos del láser y para formar el mapa.

5.1.2 Coincidencias en el escaneo

Se trata del proceso de combinar los resultados de un escaneo del entorno, con resultados anteriores o con un mapa existente con el fin de localizar el punto desde el que se ha realizado el escaneo. En muchos robots, los resultados de los laser son mucho más precisos que los datos que aporta la odometría, motivo por el cual este proceso es de gran importancia.

Este sistema emplea un método basado en encuadrar los datos recibidos del sensor láser con el mapa aprendido por el momento. Haciendo esto, no es necesario hacer una búsqueda exhaustiva de correspondencia de datos de cada punto, la cual conlleva una gran carga computacional.

Se busca encontrar el valor $\xi=(p_x,p_y,\varphi)$ que minimice:

$$\xi^* = \arg\min \sum_{i=1}^{n}\left[1-M(S_i(\xi))\right]^2$$

Correspondiente a la transformación que devuelve el mejor encuadre con el mapa. S_i son las coordenadas de resultados del láser en coordenadas reales en función de la posición del robot, ξ. La función $M(S_i(\xi))$ devuelve el valor del mapa en dicha posición. También se busca $\Delta\xi$ que optimizará el error de medida.

Este método consiste en aproximaciones lineales escaladas del gradiente del mapa por lo que no se puede asegurar una convergencia cuadrática hacia el mínimo.

5.1.3 Mapa multi-resolución

Como el método presentado se basa en las variaciones en el gradiente es muy probable que pueda quedarse atascado en un mínimo local. Para evitar esto, se emplea un mapa multi-resolución. En este sistema se emplean varios mapas de rejilla teniendo cada uno la mitad de la resolución del anterior. Estos mapas no proceden de uno modificado, sino que todos están almacenados en la memoria y se van actualizando utilizando la posición estimada mediante el proceso de alineamiento del láser. La estimación de la posición se realiza inicialmente en el mapa más basto, y, una vez obtenida, se emplea en el siguiente nivel y así reiterativamente para todos los niveles en el mapa. Gracias a esto, evitamos que el algoritmo se pueda quedar atascado ya que los mínimos locales presentados en un mapa pueden variar respecto al resto de los mapas.

5.2 Estimación del estado 3D

En esta parte veremos la estimación de estado que se realiza mediante el modelo del sistema y la integración de

los datos obtenidos por el SLAM bidimensional. El filtro de navegación trabaja en tiempo real y es actualizado de manera asíncrona mediante los datos del sensor láser y los otros sensores que se integren, los cuales pueden ser sensores GPS, barómetro, etc [14].

5.2.1 Filtro de navegación

Para estimar las posiciones de la plataforma, que exhibe 6 grados de libertad, se emplea un filtro extendido de Kalman con las ecuaciones del modelo presentadas en la introducción del sistema. Además, el vector de estado se aumenta con información proveniente de los datos de los giróscopos y acelerómetros que varían en el tiempo.

La posición y la velocidad se obtiene integrando las ecuaciones del modelo, por lo que se le debe añadir información adicional para realizar correcciones, ya que estas conllevan gran error acumulado en el tiempo. Por lo tanto, para este sistema la posición 2D en el plano se actualiza mediante los resultaos obtenidos con el láser, mientras que para el vector de estados tridimensional será necesario incorporar información adicional con otro tipo de sensores.

5.2.2 Integración de SLAM

Para lograr el mejor rendimiento del sistema posible es necesario que exista un intercambio de información bidireccional entre el SLAM bidimensional y el EKF tridimensional. Para ello, en el proceso de escaneo del entorno se emplea la posición estimada en el plano por el EKF como estimación inicial para el proceso de optimización.

En cuanto al EKF, las covarianzas cruzadas se emplean para fusionar los datos de posición de SLAM con el estado real completo. Usar únicamente las actualizaciones de las medidas mediante el filtro de Kalman daría lugar a fallo, ya que este asume únicamente errores estáticos independientes de las medidas. Tomando la estimación por Kalman como $\hat{x}$, con matriz de covarianza P y la posición de SLAM (ξ*, R) la fusión de los datos resulta en:

$$\left(P^{+}\right)^{-1} = (1-\omega)P^{-1} + \omega C^{T}R^{-1}C$$
$$\hat{x}^{+} = P^{+}((1-\omega)P^{-1} + \omega C^{T}R^{-1}\xi^{*})^{-1}$$

Siendo ω un valor perteneciente al intervalo (0,1) para modificar la actuación que tendrá el SLAM sobre el sistema de estimación y C una matriz que proyecte el estado completo del sistema 3D en el subespacio de SLAM.

Debido a que la inversión completa de la covarianza requiere muchos recursos en potencia de computo, las ecuaciones también pueden ser presentadas de la con las siguientes expresiones:

$$P^{+} = P - (1-\omega)^{-1}KCP$$
$$\hat{x} = \hat{x} + K(\xi^{*} - C\hat{x})$$

Cuya ganancia de Kalman seguiría la expresión:

$$K = PC^{T}\left(\frac{1-\omega}{\omega}R + C^{T}PC\right)^{-1}$$

5.3 Pruebas realizadas

Este método se ha comprobado empleando unas *bag files* en ROS. Estos son archivos que contienen la información pertinente a los datos recibidos por un sensor para realizar una evaluación empleando estos datos. En este trabajo caso, se empleará un archivo como este ya que no contamos con la disposición de un láser LIDAR para realizar las pruebas con un sistema real. En cuanto al paquete de Hector SLAM, el archivo package.xml presenta información referente a los datos de los creadores del método, pagina web donde se encuentra disponible, mantenimiento, dependencias, etc [15].

El *package* Hector Slam esta generado a su vez por una serie de paquetes, dentro de los cuales los más importantes son *hector_mapping*, *hector_geotiff*, y *hector_trayectory_server*.

- ***Hector_mapping***: Este paquete contiene la información necesaria para la generación del mapa y la realización de SLAM bidimensional mediante los datos obtenidos por el LIDAR, sin la necesidad de información adicional acerca de la odometría.
- ***Hector_geotiff***: Aporta las herramientas necesarias para guardar los mapas generados por los otros nodos, así como las trayectorias con información sobre las georreferencias. Este paquete utiliza servicios con otros de los nodos para obtener la información necesaria.
- ***Hector_trayectory_server***: Este paquete se encarga de realizar un seguimiento sobre las transformaciones de la trayectoria obtenidas a partir de las transformaciones de los datos de los sensores y de hacer esta información accesible a través de llamadas a servicios y Topics.

Para realizar las pruebas, debemos lanzar el archivo *tutorial.launch*. En él encontramos una serie de parámetros a cambiar, así como la información acerca de otros dos archivos .launch que se ejecutarán con el primero. Este archivo lanza la aplicación rviz, y ejecuta *mapping_default.laucnh* y *geotiff_mapper.launch*. Estos archivos se pueden consultar en los apéndices I, II y III respectivamente.

En los otros dos archivos de ejecución lo que encontramos es: en el *geotiff_mapper* la ejecución de los nodos *hector_geotiff* y *hector_trayectory_server* explicados previamente, con sus parámetros necesarios; y en *mapping_default.launch* la ejecución del nodo *hector_mapping* con todos los parámetros necesarios para generar SLAM y para la creación del mapa.

Una vez que ejecutamos el archivo *tutorial.launch* se nos abrirá una interfaz gráfica de la herramienta de rviz, y se inicializan los nodos a la espera de tener la información de los sensores. En esta interfaz podemos añadir información adicional que queramos que sea representada, como las medidas de los laser, modificaciones en el mapa, etc. Para aportar los datos de los sensores emplearemos una serie de *bag files* como hemos indicado previamente.

Los resultados obtenidos empleando las bagfiles son los siguientes:

- RoboCup 2011: Rescue Arena

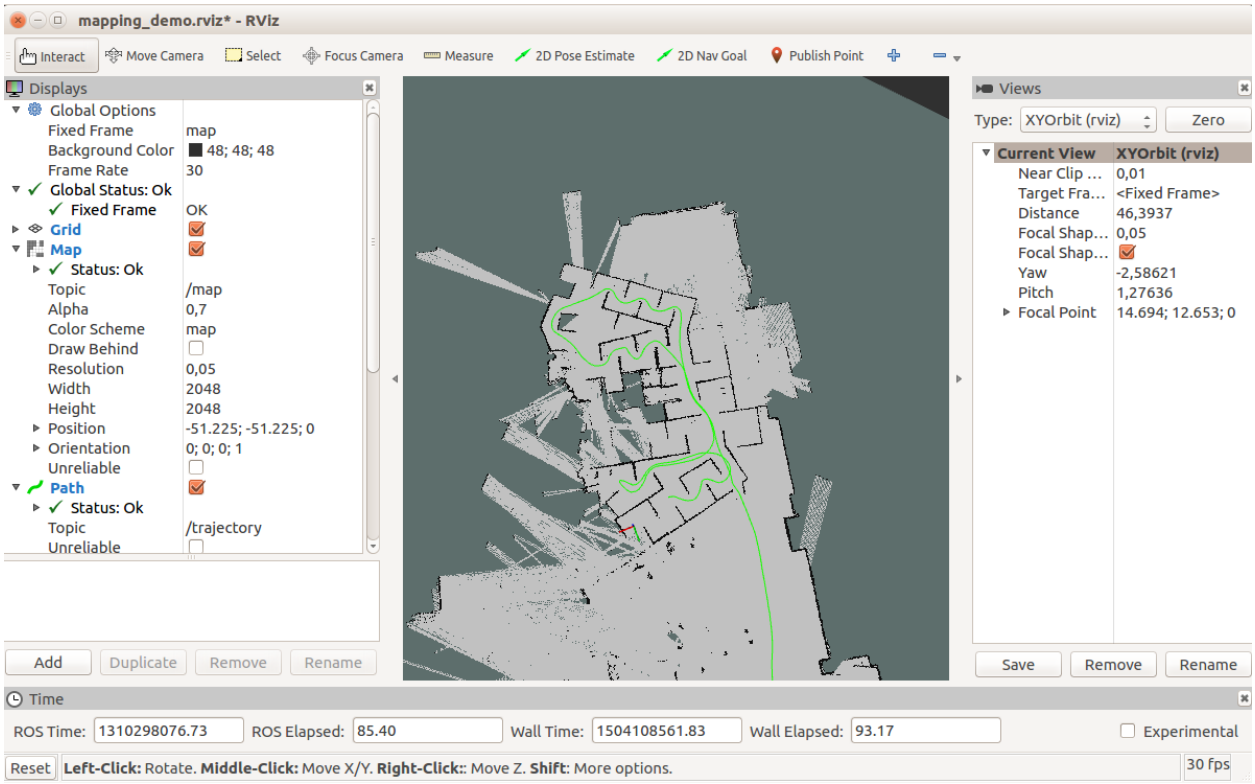

Figura 8 ventana rviz durante el proceso

Figura 9 mapa final obtenido

- Edificio L101:

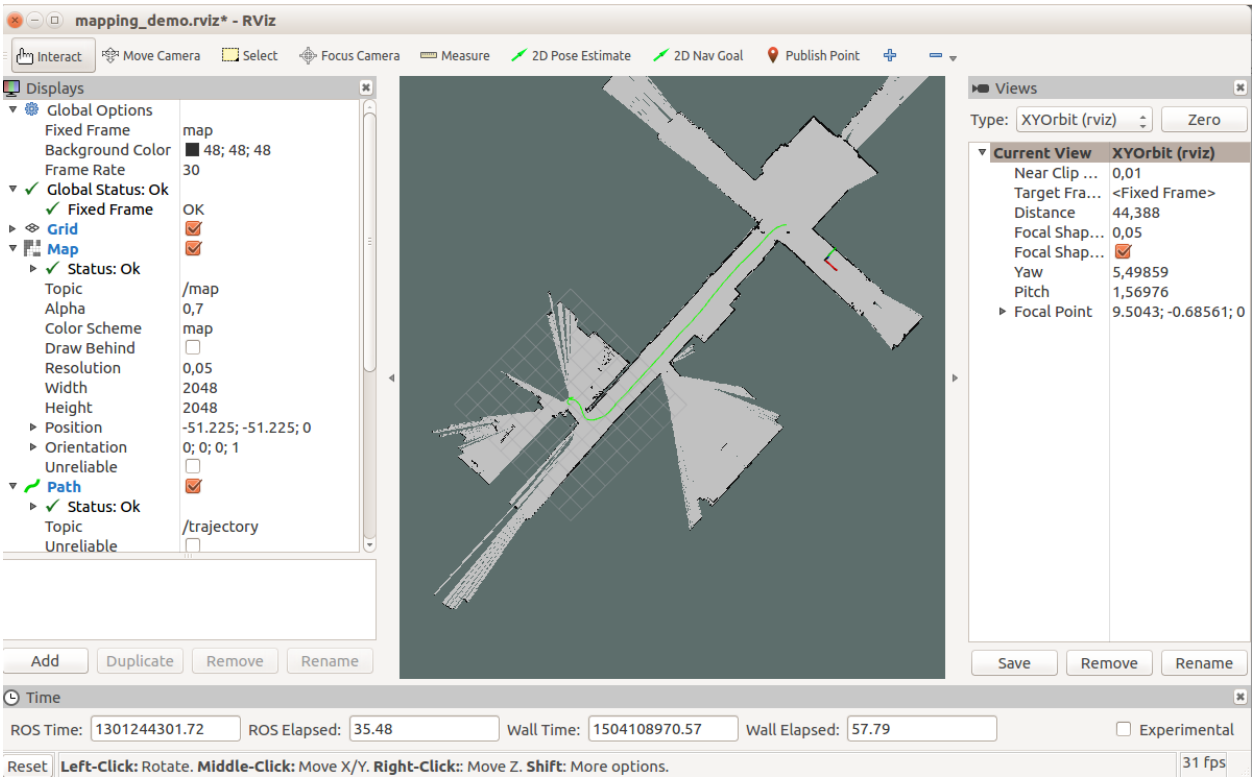

Figura 10 ventana rviz durante el proceso

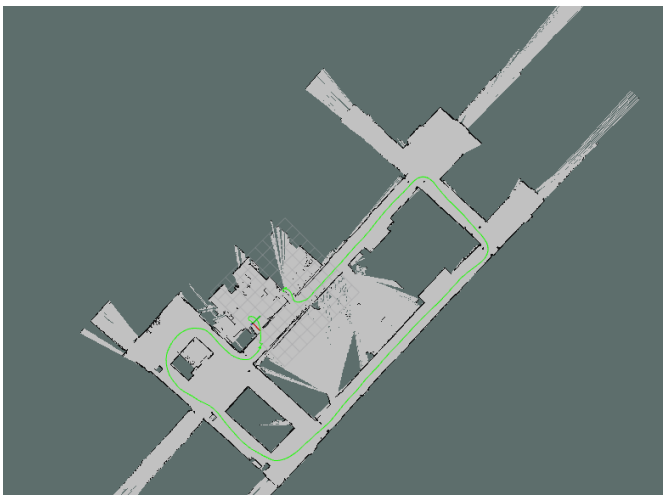

Figura 11 mapa final L101

Como podemos observar en las figuras anteriores, este sistema genera mapas bastante precisos. Hay que destacar que es un sistema que no emplea la odometría y que, gracias a la alta tasa de actualización de los datos del LIDAR el algoritmo se ejecuta muy rápidamente y con una precisión muy buena a la hora de establecer el posicionamiento. Sin embargo, como podemos observar en las imágenes, los mapas generados son bidimensionales, por lo que en entornos de robótica en los que se requiera manipulación no es el método más adecuado.

A pesar de estas carencias, el algoritmo se ha implementado satisfactoriamente tanto en plataformas terrestres móviles como en UAVs.

6 RGB-D SLAM

RGB-D SLAM es uno de los primeros sistemas que aprovechan las gran cantidad de datos en forma de imágenes y de distancias que genera una cámara RGB-D para realizar acciones de mapeo y localización. Este sistema propone el uso de un modelo de medida del entorno (*environment measurement model*, EMM) para mejorar el rendimiento y la robustez del sistema, validando las transformaciones estimadas y las correspondencias entre puntos característicos mediante el empleo del punto iterativo más cercano (*iterative closest point*, ICP). Este sistema permite posicionar el robot durante largas trayectorias y bajo complejas circunstancias, como movimientos rápidos de la cámara como entornos con rasgos pobres.

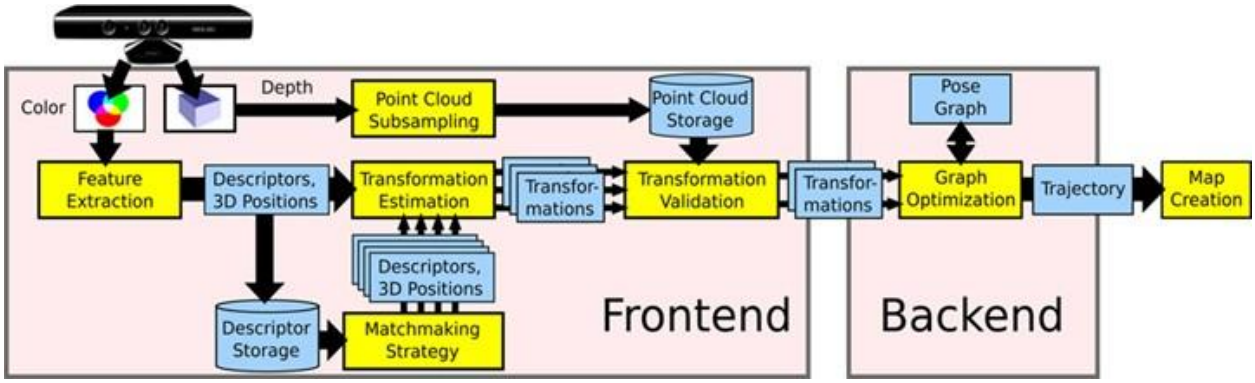

Figura 12 visión esquemática del sistema

Se trata de un sistema basado en grafos, en los que se formula el problema localizando las posiciones en los nodos y estableciendo las acciones de control en las aristas o arcos entre los nodos. Estas acciones de control que representan las aristas son obtenidas mediante observaciones del entorno o por acciones de movimiento realizadas por el robot. Una vez formulado el grafo, el mapa se puede representar buscando las configuraciones espaciales de los nodos con las medidas tomadas de las aristas [16]. Estos sistemas se pueden dividir en 3 partes: *frontend*, *backend* y generación del mapa. La parte "delantera" o *frontend* se encarga de procesar los datos de los sensores para establecer relaciones geométricas en el entorno. Esta parte es especifica al tipo de sensor utilizado y se utiliza también para estimar los movimientos del robot mediante una secuencia de escenas. Para el caso de la cámara RGB-D, las entradas a esta parte del sistema son una imagen a color I_{RGB} y una imagen de profundidades I_D. Mediante estas imágenes se determinan los landmark, creando un vector descriptor de grandes dimensiones procedente de la imagen I_{RGB} y almacenando este con su localización relativa a la posición del robot. Para lidiar con la incertidumbre establecida, la parte "trasera" o *backend* de SLAM se encarga de construir el grafo que representa las relaciones geométricas y sus incertidumbres. Una vez construido este grafo, se encarga de buscar una optimización de este para encontrar una solución más adecuada a la trayectoria del robot. En cuanto a la generación del mapa, usar directamente la información de los sensores sería muy ineficiente por lo que se emplea un mapa probabilístico ocupacional tridimensional proveniente de los datos RGB-D [17].

6.1 Egomotion

Se entiende por egomotion a la estimación del movimiento de una cámara en un entorno rígido. En este sistema, la parte delantera de SLAM usa las entradas de la cámara en forma de landmark para establecer relaciones geométricas que permitan estimar el movimiento del robot entre dos estados. Para establecer los landmark se pueden emplear diversos métodos, entre los que es sistema cuenta con ORB, SURF, SIFT y un método que se basa en el uso de ShiTomasi y SURF. Estos métodos se han visto en una parte previa del trabajo y se encargan de calcular los puntos de referencia, así como de especificar sus descriptores para que estos puedan ser reconocidos posteriormente sin posibilidad de error. Para emplearlos, el sistema utiliza la librería OpenCV excepto para el uso de SIFT, para el que se emplea un entorno basado en GPU. En cuanto a cuál de estos métodos es más adecuado, ORB y ShiTomasi y SURF son buenas opciones para robot con recursos limitados ya que dan resultados más que aceptables o para sistemas que necesitan actuación en tiempo real. Sin embargo, si disponemos de un dispositivo que permite el uso de la GPU, SIFT es la mejor opción, ya que es el que aporta los resultados más precisos.

Debido a la gran cantidad de características presentes en un mapa, que formaran parte de nuestros puntos de referencia, no es posible emplear un único umbral de rechazo. Por ello, este sistema emplea una comparativa entre el vecino más cercano y el segundo vecino más cercano. Asumiendo que un punto solo corresponderá a otro único punto en otra imagen, el segundo vecino más cercano debe estar mucho más lejos que el primero. Así pues, se establece un umbral entre el más cercano y el segundo más cercano para controlar la relación entre falsos positivos y falsos negativos. Además, para tener un sistema sólido contra falsos positivos se emplea RANSAC (también presentado en una parte previa del trabajo) cuando se estima la transformación entre dos imágenes. Para ello, se inicializa la estimación de la transformada a partir de 3 características. Esta es verificada calculando los *inliers*, que presentaremos más tarde; usando un umbral basado en la distancia de Mahalanobis. Para aumentar la solidez en caso de valores ausentes de profundidades, se incluyen características sin valor de profundidad en el paso de verificación.

Para entornos no naturales donde se aprecian estructuras repetitivas (varias sillas del mismo modelo) la efectividad del sistema es limitada resultado en estimaciones de falsas transformaciones. Un umbral para el mínimo número de coincidencias a la hora de estimar la transformación ayuda en entonos con similitudes y repetición de estructuras, pero elevar el valor de este penaliza el resultado en entornos sin dichas ambigüedades.

En cada iteración de RANSAC, se emplean mínimos cuadrados para calcular la estimación del movimiento. La estimación de la transformación puede ser mejorada empleando la minimización de la distancia de Mahalanobis en lugar de la distancia Euclídea. Además, el sistema emplea también la optimización de un pequeño grafo constituido por dos posiciones de la cámara y los *inliers* previamente calculados [17].

6.2 Modelo de Medida del Entorno (EMM)

Obteniendo un gran porcentaje de *inliers* los métodos para la estimación de la egomotion mostrados en el punto anterior funcionan de forma correcta. Sin embargo, tanto RANSAC como ICP son algoritmos que no detectan errores. Por ello, el sistema emplea otro método de verificación independiente del mostrado anteriormente. Este método aprovecha la gran cantidad de datos de profundidades, en concreto aquellas medidas que contienen espacio libre. Se empela un método basado en haces (beam-based), para penalizar transformaciones en las que puntos de las imágenes de profundidad debían haber sido tapados por otros puntos de otra imagen de profundidad.

En concreto, se emplea un procedimiento análogo a los test estadísticos. En este caso, la hipótesis nula sería asumir que, tras la estimación de la transformación, las medidas de la profundidad espacial difieren de la localización de la superficie a medir.

Mediante una serie de procedimientos matemáticos y métodos probabilísticos se llega a que:

$$p(\Delta Y) = N(\Delta Y \mid 0, \Sigma) \in \mathbb{R}^{3N}$$

Siendo ΔY un vector columna formado por cada componente $\Delta y_{ij} = y_i - y_j$, Σ conteniendo las matrices de covarianza pertenecientes a cada punto en una matriz diagonal, y N representando una distribución normal.

Esta ecuación no cuenta con un término para medidas en cortas distancias, ya que se cuenta con que el entono sea estático durante el mapeo y son estas medidas las que se pretende penalizar. En contraste, las medidas de profundidad que se ven proyectadas por detrás de su valor correspondiente son muy comunes. Los puntos que se ven proyectados muy por detrás se sus medidas correspondientes son ignorados.

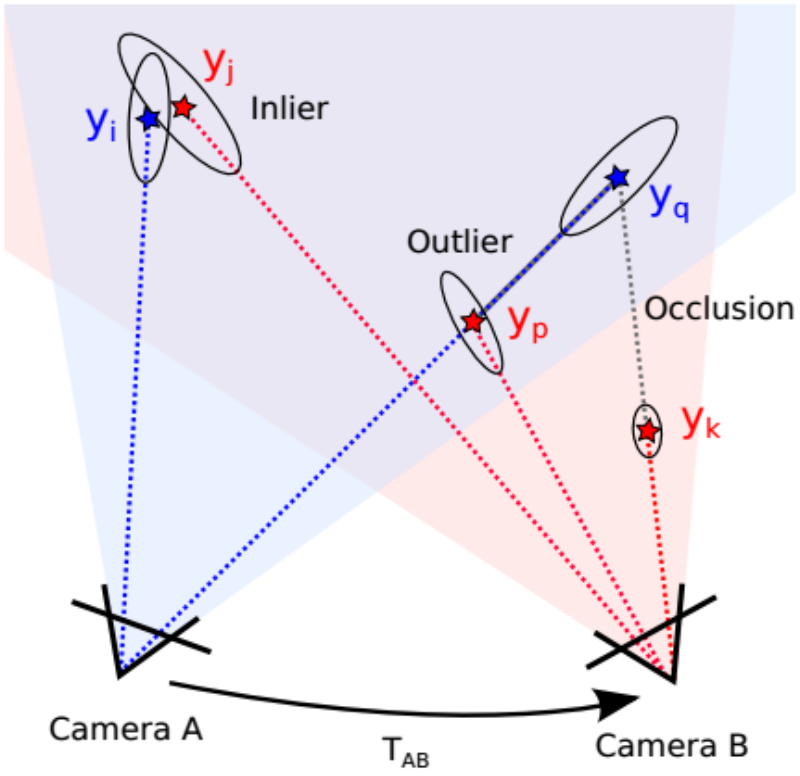

Figura 13 transformación estimada y observaciones en dos imágenes

En la figura 13 se pueden observar los siguientes casos del EMM: Una vez estimada la transformación T_{AB}, proyectando los puntos de la cámara A en B se obtiene que la asociación y_j e y_i se cuenta como *inliers*, ya que no hay nada que caiga en el espacio libre entre las observaciones. El punto y_k tapa la visión del punto y_q a la cámara B por lo que cuenta como un punto ocluido. Además, y_k está fuera del rango de visión de la cámara A por lo que es ignorado. El punto y_p se sitúa en el punto libre entre la cámara A y la observación de y_q por lo que se cuenta como un *outlier.* Por lo tanto, este EMM consta de 2 *inliers* 1 *outlier* y 1 *occlusion.*

Con este modelo se puede emplear un test de la hipótesis estándar para rechazar observaciones por debajo de un nivel de confianza establecido utilizando la distancia de Mahalanobis en una distribución de chi-cuadrada. Sin embargo, este test es muy sensible a errores y, con ello, muy difícil de emplear de manera útil. Con el fin de solventarlo, este sistema emplea una medida que varíe más suavemente con los errores en la transformación. El sistema realiza por lo tanto la hipótesis con las medidas tomadas y se establece a cantidad de *outliers* como medida al rechazo de la transformación.

La fracción de *inliers* es independiente con a la de *outliers* y degenera en un suave incremento en los errores de la transformación por lo que se le aplica un umbral que ayuda a reducir de manera eficaz la cantidad de errores en la estimación de la transformación [17].

6.3 Odometría visual y búsqueda de cierre del bucle

El procedimiento seguido para calcular la egomotion genera información de odometría visual. Sin embargo, estas estimaciones, evaluadas de manera individual, son muy ruidosas y dan lugar a gran cantidad de errores de estimación. Combinando varias estimaciones del movimiento y comparando cada una de las imágenes con otras diferentes de la inmediatamente anterior ayuda a mejorar el rendimiento del sistema y reduce el error acumulado. Asimismo, realizar estimaciones de la transformación con fotogramas mucho más antiguos, *loop closure*, reduce el error acumulado de manera drástica. A pesar de esto, comparar todos los fotogramas con cada uno de los anteriores sería muy costoso para el sistema y lo ralentizaría en gran manera por lo que no se puede asumir este método de trabajo.

Por ello, en este sistema se centran en buscar una estrategia más eficiente, en la que se emplean 3 tipos de candidatos. Primero se busca la transformada mediante el método explicado previamente en el punto de egomotion a los *n* predecesores inmediatos. Seguidamente, se busca un cierre del bucle (*loop closure*) en el vecindario geodésico (geodestic neighborhood[1]) de los fotogramas anteriores. Se genera así un árbol de expansión a partir del grafo de posiciones con el predecesor como raíz inicial. Entonces se eliminan los *n* predecesores inmediatos del árbol para evitar duplicación de los datos y aleatoriamente se eligen *k* fotogramas anteriores para incluir en este. De esta forma, este método guía la búsqueda hacia estimaciones exitosas partiendo de aquella que ya tuvieron éxito previamente. Concretamente, el algoritmo escogerá aquellos fotogramas cercanos a un cierre de bucle para hacer las comparaciones entre fotogramas. Para buscar grandes cierres de bucle, se comparan aleatoriamente *l* fotogramas con una serie de *keyframes* o fotogramas clave. Una imagen es incluida en el grupo de fotogramas clave si no puede ser correspondida con ninguno de los fotogramas claves anteriores. De esta forma, el número de imágenes para realizar las pruebas de reduce de gran manera [17].

6.4 Optimización del grafo

Las transformaciones estimadas entres las posiciones del sensor, realizadas por la parte frontal de SLAM, forman las aristas del grafo de posiciones. Debido a los errores en la estimación, estas aristas no forman una trayectoria. En esta investigación se busca la optimización del grafo utilizando el "*g²o framework*", el cual desarrolla una minimización de la función error de carácter no lineal que puede ser representada como un grafo. De manera más precisa, se minimiza la siguiente función:

$$F(X) = \sum_{\langle i,j \rangle \in C} e\left(x_i, x_j, z_{ij}\right)^T \Omega_{ij} e\left(x_i, x_j, z_{ij}\right)$$

Para buscar la trayectoria optima correspondiente a $X^* = \arg\min_X F(X)$, donde X es el vector que contiene las posiciones del sensor, y z_{ij} y Ω_{ij} representan la media y la matriz de información sobre las transformaciones calculadas por la parte frontal de SLAM. Finalmente, $e\left(x_i, x_j, z_{ij}\right)$ corresponde a la función del error que mide como de bien x_j y x_i se relacionan en función de la restricción z_{ij}. Será 0 si los puntos cumplen exactamente restricción, es decir, si por ejemplo los puntos se corresponden exactamente con lo estimado por la transformación.

[1] Un vecindario geodésico para un pixel p está constituido por los k puntos más cercanos de carácter geodésico, es decir, aquellos puntos pertenecientes a la misma estructura. Para ver más detalles del algoritmo, podemos localizarlo en [18]

La optimización global del grafo es muy beneficiosa en el caso de grandes cierres del bucle, disminuyendo el error acumulado. Desafortunadamente, grandes errores en la estimación del movimiento pueden interferir en el rendimiento de grandes partes del grafo. Esto es el principal problema en zonas en las que hay características muy ambiguas, como entornos con estructuras repetitivas. El método de validación presentado anteriormente

con el EMM mejora el índice de transformaciones erróneas. Sin embargo, la validación no se puede garantizar en todos los casos. El error residual existente en el grafo una vez realizada la optimización permite determinar inconsistencias en las aristas. Para solucionar esto, se introduce un umbral para recortar aquellas aristas que contengan grandes errores después de la convergencia inicial, y después de esto se continua la optimización del grafo.

6.5 Representación del mapa

Usando la trayectoria descrita anteriormente del sistema se podría proyectar en cada punto las medidas tomadas por el sensor y, de esta forma, representar una nube de puntos que constituiría el mapa. Sin embargo, las nubes de puntos contienen gran cantidad de limitaciones por lo que el sistema presenta un método distinto de crear el mapa. Para ello, se emplea mapas de ocupación de rejilla tridimensionales para representar el entorno. En concreto, el sistema emplea el *framework* OctoMap. Las celdillas se gestionan en una estructura de árbol que permite una compactación de la memoria utilizada para la representación. Una ventaja crucial de este sistema es la representación explicita del espacio libre en el mapa, lo que resulta de gran utilidad para sistemas robóticos que necesites realizar tareas en el entorno evitando colisiones o que se encarguen de tareas de exploración.

A pesar de esto, este *framework* también contiene algunos inconvenientes. La creación de un OctoMap necesita de un sistema con mayor cantidad de recursos, ya que cada medida de la profundidad es posicionada dentro del mapa. Este proceso ralentiza el algoritmo, ya que por cada 100000 puntos que se quieran introducir en una celdilla de 5cm se tarda un tiempo de 1s. Por lo tanto, si se busca tener un sistema que genere los mapas de forma continua y en tiempo real, se debe reducir la resolución y solo introducir datos de la nube de puntos de un subconjunto.

Además, hay que remarcar que los mapas de rejillas no se pueden actualizar de forma eficiente en caso de gran cantidad de corrección de errores, como los obtenidos en grandes cierres de bucle. En estos casos, el sistema optará por generar un nuevo mapa partiendo de los datos del anterior y de las correcciones.

6.6 Pruebas realizadas

El algoritmo se ha testeado utilizando una cámara Micrososft® Kinect mediante el middleware ROS. Este se encuentra disponible como un *package* de ROS [19]. Para obtener este paquete de archivos, lo descargamos desde GitHub y seguimos los pasos indicados en el archivo README. Adicionalmente, debemos instalar g2o que se emplea para la optimización del grafo de posiciones y el *package* openni de ROS que permite integrar el uso de la cámara Microsoft® Kinect en Ubuntu y en ROS.
Tal y como están configurados los paquetes en ROS, podemos abrir el archivo *package.xml* de la carpeta principal de este para obtener información útil. En la parte inicial del archivo tendremos una breve descripción, así como el nombre de los autores, correos o la página web en la que se encuentra alojada la información pertinente al paquete. A continuación, encontramos la información sobre las dependencias del paquete. Estas están divididas en 4 tipos de dependencias: build_depend, que representan los paquetes necesarios a la hora de realizar la compilación; buildtool_depend, los cuales son paquetes que contienen herramientas empleadas a la hora de realizar la compilación; run_depend, para aquellas dependencias necesarias a la hora de la ejecución; y por último test_depend, que son aquellos paquetes que únicamente son necesarios a la hora de realizar testeos. Las dependencias de este *package* son las que se observan en el apéndice IV.

En el caso de no tener todos esos paquetes incluidos en las dependencias instalados, el sistema no funcionaría. ROS contiene una herramienta llamada rosdep que instala las dependencias que falten para que un paquete funcione correctamente. En este caso, solo hay que hacer una llamada a la función indicando el paquete del que queremos instalar las dependencias:

```
sudo rosdep install rgbdslam
```

Una vez instaladas todas las dependencias podemos instalar y ejecutar el paquete correctamente. A la hora de ejecutar paquetes, ROS consta con una serie de archivos de extensión .launch. Estos son archivos que se encargan de inicializar los nodos, establecer las relaciones, suscripciones a topics, etc; facilitando así el trabajo, ya que permite no tener que realizar esta tarea cada vez que queremos ejecutar el algoritmo, sino que se realiza una primera vez estableciendo todos los datos y ya solo será necesario ejecutar este archivo. En este paquete cuenta con varias launch files, entre las cuales se encuentran *openni+rgbdslam.launch*, que ejecutará simultáneamente el inicio de la cámara con el paquete openni, y el módulo de rgbdslam, así como la GUI. También tenemos *headless.launch*, la que se encarga de realizar el algoritmo sin el uso de una GUI, sino que se ejecuta mediante llamadas desde ROS. Esta launch file se emplea para el uso del algoritmo en robot, emitiendo las llamadas desde un dispositivo remoto. Además de estas dos, el paquete cuenta con otras tres launch files que son *fast_visual_odometry.launch*, que , como su nombre indica, muestra visualmente los resultados de la odometría; *rgbdslam.launch*, la cual inicializa el algoritmo de forma separada a la inicialización de la cámara; y *qvga-kinect+rgbdslam.launch*, la cual emplea el algoritmo de búsqueda de keypoint SIFT con el uso de la GPU. Para poder ver la interfaz gráfica del sistema, se inicializará por separados rgbdslam y openni. Inicializando rgbdsalm se ve la interfaz gráfica de la fig. 16. Esta consta de 5 imágenes diferentes. En la mitad superior se encuentra la vista tridimensional. En ella se puede ver el mapa tridimensional, así como el grafo de posición. Mediante la pestaña *3D view* alojada en la parte superior de la GUI se puede modificar las vistas de la ventana tridimensional, así como establecer algunos parámetros. La pestaña *2D view* se emplea para modificar lo mostrado en la mitad inferior de la GUI. En las cuatro ventanas se muestra, por orden de izquierda a derecha: La imagen RGB de la cámara, la imagen de profundidades en escala de grises, una tercera imagen mostrando los Keypoint detectados, y por último una imagen denominada *visual flow image* que muestra los movimientos que siguen los keypoint mientras se realiza el algoritmo, empleados para la optimización del grafo y la localización.
Además de permitirnos visualizar el algoritmo y su realización, el otro conjunto de pestañas nos permiten cargar archivos .pcd (extensión de los mapas) y bag files de ROS; guardar los mapas, las nubes de puntos, etc; y modificar algunos de los parámetros del algoritmo.

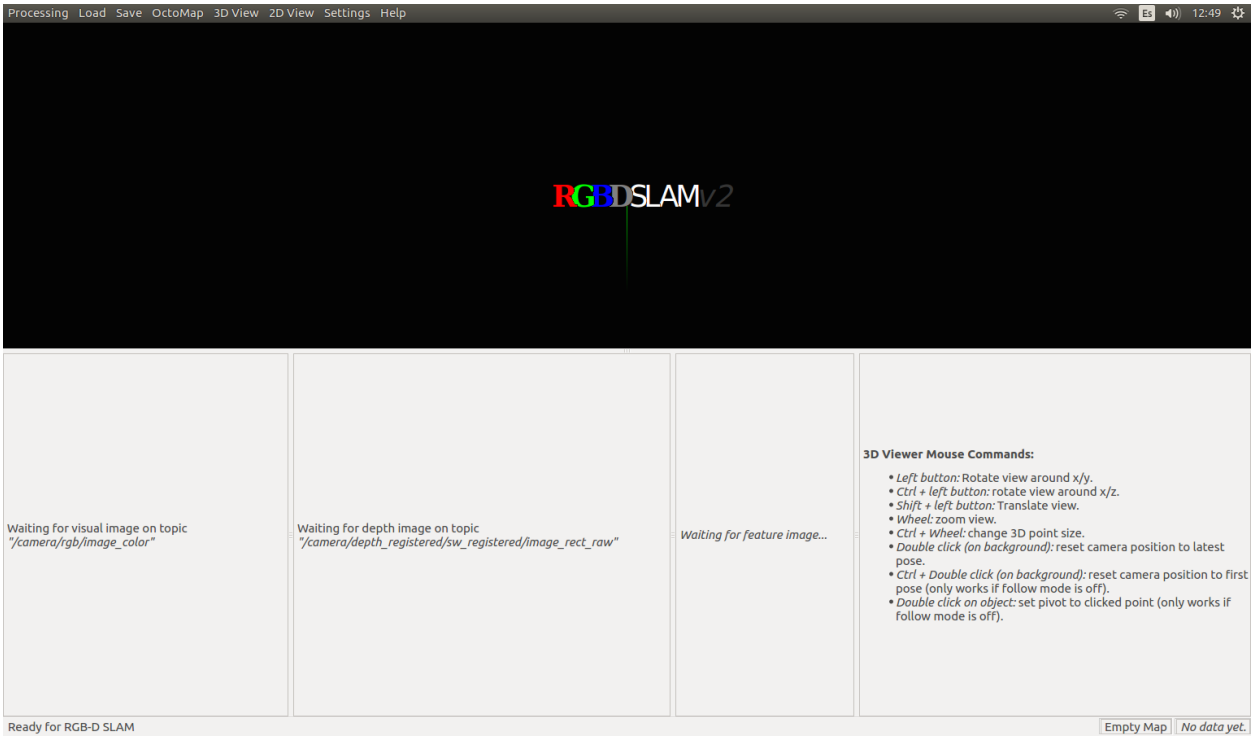

Figura 14 interfaz gráfica rgbdslam

Sin embargo, los creadores del método recomiendan modificar los parámetros pertinentemente en los archivos .launch ya que los cambios realizados en tiempo real en la interfaz gráfica pueden no surtir efecto.

El algoritmo de búsqueda de keypoints predeterminado en la launch file es ORB debido a que era el que proporcionaba mejores resultados en esta segunda versión del paquete. Sin embargo, este se puede cambiar en cambiando el parámetro correspondiente, con el fin de emplear SIFT, SURF o SURF extendido. Para las pruebas, se ha decidido emplear ORB ya que los resultados eran mejores.

Una vez explicado el funcionamiento del paquete, podemos proceder a realizar una serie de pruebas. Las pruebas se han realizado con éxito en dos entornos distintos. Ambos son habitaciones no muy grandes, con distinto nivel de iluminación. Asimismo, también se han realizado distintos tipos de pruebas. Además, se han realizado también una prueba en un pasillo y posteriormente accediendo a una habitación.

En la primera prueba se ha seguido un movimiento similar al realizado por un UAV, en el que hay movimientos en los 3 ejes y en las 3 rotaciones. Sin embargo, presenta muchas menos vibraciones y movimientos menos bruscos que los que realizaría este tipo de plataforma. Los resultados obtenidos han sido los que se pueden apreciar en las figuras 15 y 16.

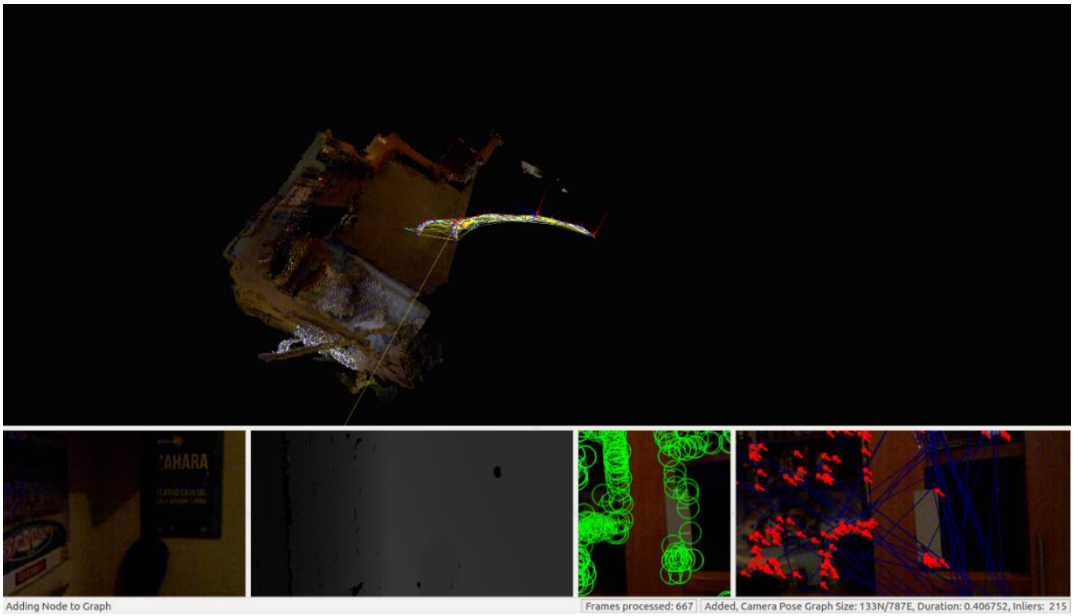

Figura 15 vista del sistema completo de rgbdslam handheld

Figura 16 vista tridimensional de la primera prueba

Por otro lado, la segunda prueba la hemos realizado con una estructura que simula el movimiento de una plataforma móvil en el plano en la misma habitación. Se puede observar que los resultados del mapa tridimensional son más claros. Así mismo, fijándonos en las ventanas de la parte inferior (la visión bidimensional) podemos observar cómo actúa el detector de *keypoint*; y, en la imagen 17, podemos observar que debido a el procedimiento que realiza el sensor de profundidad, la interacción con objetos de vidrio, en este caso la pecera, produce errores en la detección de la profundidad.

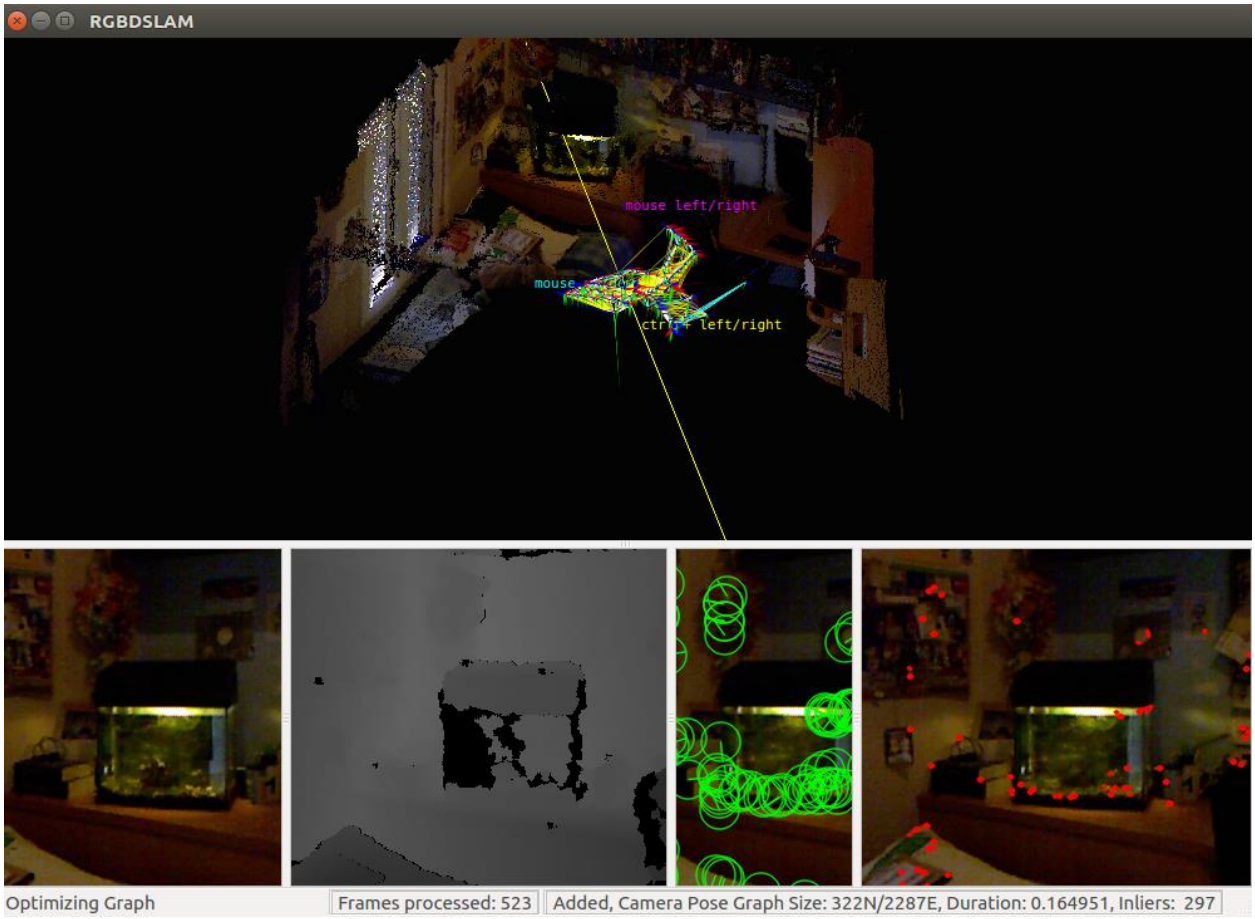

Figura 17 visión completa de la segunda prueba

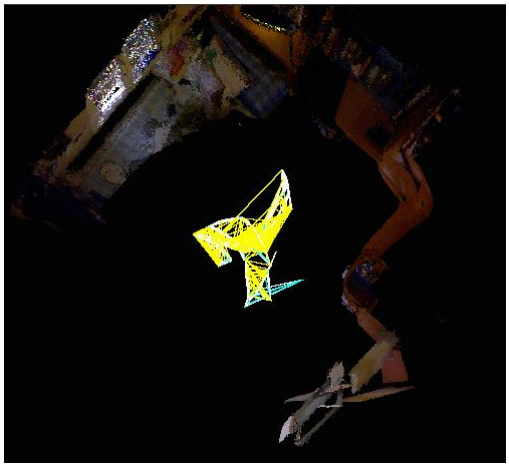

Figura 18 mapa en planta de rgbdslam con un movimineto en el plano

En la figura 19 se muestra el grafo de posiciones. Como podemos observar, las variaciones en la estimación de la posición sobre el eje z son despreciables.

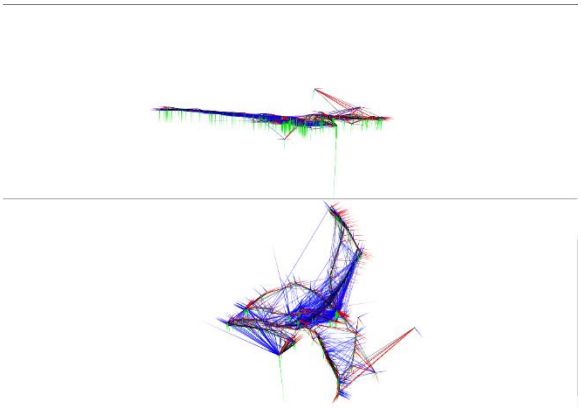

Figura 19 Grafo de posiciones. En la parte superior se muestra el grafo

con el eje z aproximadamente vertical.

En la parte inferior se puede ver el grafo en una posición aproximada al plano xy

La siguiente prueba se ha realizado en otra habitación, de tamaño similar, pero con peor iluminación y con el mismo tipo de movimiento que en la prueba anterior. Debido a la iluminación del entorno, el mapa tridimensional se ve peor, pero tiene la misma calidad y definición, ya que es suficiente para que el sensor de profundidad capte los objetos. Podemos ver los resultados en la figura 26.

Figura 20 rgbdslam en una habitación con mala iluminación
con movimiento en el plano

Por último, se muestran los resultados de las pruebas realizadas en el pasillo.

Figura 22 proceso de rgbdslam en un pasillo

Figura 21 rgbdslam en pasillo y habitación

En general, podemos decir que los resultados obtenidos empleando este método han sido bastante buenos. Tanto el mapa creado como el grafo de posiciones sobre este son claros, el algoritmo se ejecuta de manera rápida y los cambios bruscos afectan al mapa de manera levemente apreciable y muy poco al grafo de posición. En lo referente a la prueba realizada en el pasillo, figuras 21 y 22, los resultados no son tan buenos, ya que se presentan distorsiones en el mapa, debido a que este método no está orientado para su uso en este tipo de entornos con gran cantidad de estructuras similares (puertas y paredes) y con poca iluminación.

7 ELASTICFUSION

ElasticFusion es un sistema para realizar SLAM denso en tiempo real y sin la necesidad de emplear un grafo de posición con el empleo de una cámara RGB-D. Este sistema es capaz de generar mapas basados en el uso de superficies como landmark, usando un modelo (*dense frame-to-model*) de seguimiento de la cámara y fusión de los datos para construir superficies acompañado de un constante refinamiento del modelo mediante el uso de deformaciones no rígidas de estas. Además, se encarga de buscar cierres de los bucles de optimización locales (*local loop closures*) tan pronto como sea posible, dejando el cierre del bucle global (*global loop closure*) para corregir errores acumulados arbitrarios y mantener la consistencia global del sistema.

En métodos de SLAM denso tridimensional, el espacio es mapeado fusionando los datos obtenidos por el sensor en movimiento con la representación continua de las superficies que contiene el entorno. Sin embargo, estos métodos presentan dificultades cuando el sensor presenta movimientos de duraciones extendidas sin revisitar áreas conocidas, y cuando siguen trayectorias entrecruzadas sobre si mismos. Normalmente, para solucionar esto, los algoritmos de SLAM destinan su metodología a la implementación únicamente en una de las dos situaciones. Debido a la gran cantidad de datos obtenidos en *dense SLAM* unir los datos del filtrado con la optimización local no es factible. En vez de eso, estos sistemas se basan en filtrar los elementos por superficies.

El sistema presentado en este capítulo se centra en una visión dirigida principalmente a la generación correcta del mapa, dejando de lado el grafo de posición; que encaja mejor en el modelo que se debe aplicar a un sistema denso. Asimismo, este sistema intenta aplicar *local loop closures* tan rápido y frecuente como sea posible, para suplir las carencias que podría presentar a la hora de estimar las trayectorias el hecho de no presentar un grafo de posiciones. Esto permite emplear una deformación espacial del mapa mediante un grafo de deformación fijado en el interior de las propias superficies que serán definidas. Gracias a esto, consigue una estimación de la trayectoria semejante o incluso mejor que los sistemas densos de SLAM que emplean grafos de posición [20].

7.1 Visión general de las características del sistema

Este sistema sigue una arquitectura típica de los métodos de SLAM denso, en la que se alterna entre el mapeo y la localización. Como muchos otros, hace uso de la programación en GPU. En concreto, emplea CUDA para implementar el seguimiento y la librería de lenguaje OpenGL para las representación y gestión del mapa.

El esquema general del procedimiento seguido por el sistema es:

1. Se realiza una estimación de un modelo del entorno basado en las superficies a través de los datos obtenidos por los sensores.
2. Mientras se realiza el seguimiento y la fusión de datos en la zona del modelo observada más recientemente, denominada como zona activa, segmentos del mapa que no han sido re observado en un período de tiempo δ_t se introducen en la zona inactiva del mapa. Esta zona, como veremos posteriormente, no se emplea para la fusión de los datos ni para la localización.
3. En cada fotograma tomado por el sensor, se busca una correspondencia entre la porción de mapa activo en la posición estima actual de la cámara y la porción de mapa inactivo subyacente en la misma posición. Si esta correspondencia es encontrada, se cierra un bucle local y el modelo es deformado de manera no rígida para reflejar este acontecimiento. La parte inactiva del mapa que ha causado este cierre del bucle se reactiva para permitir la localización y fusión de datos.

4. Para realizar cierres globales del bucle, se añaden las visiones del entorno de manera aleatoria a una base de datos codificada. En cada captación de una imagen, se busca encontrar una correspondencia con esta base de datos. Si esta es encontrada, se comprueba si es consistente con el modelo geométrico del sistema y si lo es, se realiza una deformación del modelo completo, llevando a un alineamiento global de las superficies.

Un esquema de este modelo del sistema se puede encontrar en la figura 23. En ella se muestra el proceso seguido por el sistema: (i) Inicialmente todo el modelo está marcado como zona activa mientras la cámara se mueve a la izquierda; (ii) Con el paso del tiempo, la zona del mapa que fue observada por primera vez va cambiando a zona inactiva; (iii) La cámara re observa la zona inactiva del mapa. En concreto, la zona remarcada en la imagen. En este momento, se cierra un bucle local y se realiza una deformación que corrige los errores en el mapeo y la localización; (iv) La exploración continua hacia la derecha y se cierran más bucles; (v) continua la exploración hacia zonas nuevas del entorno; (vi) La cámara re observa una zona inactiva pero el error acumulado es demasiado grande para realizar un cierre local del bucle; (vii) La disparidad de una zona del mapa con la otra es clara; (viii) Se realiza el cierre del bucle global, lo que alinea la zona inactiva con la activa; (ix) se continua la exploración hacia la derecha cerrando nuevamente bucles locales; (x) Mapa final del entorno con las posiciones de la cámara incluidas y corregidas una vez realizado el cierre global del bucle. La información y la imagen han sido extraídas de [20]

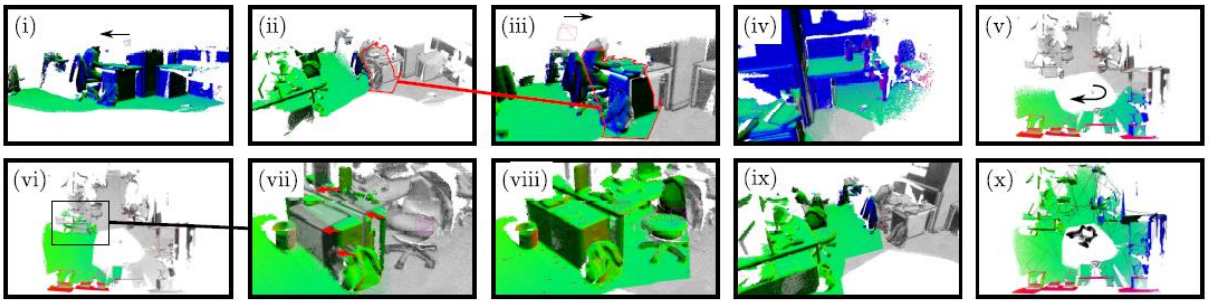

Figura 23 visión general de los pasos del sistema [20]

7.2 Fusión de los datos y seguimiento

La representación del entorno se realiza mediante un conjunto desordenado de superficies incluidas en el conjunto $\mathcal{M}$, donde cada superficie $\mathcal{M}^s$ tiene los siguientes atributos: una posición p, una normal n, un color c, un peso *w*, un radio r, una marca de tiempo inicial t_0 y una marca de tiempo de la última actualización t. El radio de cada superficie se encarga de establecer el espacio total de la superficie, tratando de minimizar los agujeros.

Las reglas del sistema para realizar la inicialización de las superficies y de la fusión de datos de profundidades que sigue el sistema emplean una predicción del mapa de profundidades mediante la representación aleatoria de puntos de los datos en el modelo, al igual que se realiza para los datos del color; y la definición de un umbral de tiempo δ_t que divide el conjunto $\mathcal{M}$ entre superficies activas e inactivas. Hay que tener en cuenta que solo las superficies clasificadas como activas se emplean para la estimación de la posición de la cámara y para la fusión de los datos de las profundidades.

Por otro lado, se define el espacio del dominio de las imágenes como Ω, compuesto por un mapa de profundidades *D*, y una imagen de color *C*. Para cada entrada de datos de la cámara en el tiempo t, se realiza una estimación global de la posición de la cámara P_t comprobando con el mapa actual de profundidades y las imágenes de color con los del modelo activo de la última posición estimada. Todas las posiciones de la cámara se definen mediante un matriz constituida por una matriz de rotación y una de translación [20].

El sistema busca minimizar el error geométrico y el fotométrico. Estos se definen de la siguiente forma:

- **Estimación de la posición a partir de la geometría.**

  Tomando los datos del mapa de profundidades actual D_t^l y el modelo del mapa activo del último fotograma $\hat{D}_{t-1}^a$ se buscan los parámetros de movimiento ξ que minimicen el coste del error sobre los puntos proyectados. La función coste se definirá como:

$$E_{icp} = \sum_k ((v^k - \exp(\hat{\xi})Tv_t^k)\cdot n^k)^2$$

  Siendo v_t^k la proyección del k-ésimo vértice en D_t^l, v^k y n^k los vértices y normal representados en la imagen anterior, T es la estimación de la transformación de la última posición a la actual. Los vértices se asocian utilizando el método ICP presentado en partes anteriores de este trabajo.

- **Estimación de la posición a partir de la fotometría**

  Al igual que en el caso geométrico, se buscan los parámetros de movimiento ξ que minimicen el coste, pero esta vez sobre el error fotométrico entre los pixeles, y partiendo de la imagen de color actual C_t^l y la de la última imagen de color $\hat{C}_{t-1}^a$. Se define la función como:

$$E_{rgb} = \sum_{u\in\Omega} (I(u, C_t^l) - I(\pi(K\exp(\hat{\xi})Tp(u, D_t^l)), C_{t-1}^a))^2$$

  Definiendo p(u,D) como la proyección tridimensional de un punto, la función π(Kp) como la proyección perspectiva de un punto en la imagen, e I(u,C) como la función que calcula la media de la suma de cada canal de color para un pixel.

- **Optimización**

  Ahora, se trata de minimizar el coste total de esta estimación del error. Para ello, definimos una función coste del seguimiento como:

$$E_{track} = E_{icp} + w_{rgb}E_{rgb}$$

  Donde w_{rgb} representa la importancia que tiene la estimación fotométrica dentro del modelo. En este sistema se toma un valor de 0,1 debido a investigaciones de trabajos previos. Una vez calculado el coste total, se emplea el método de los mínimos cuadrados no lineales de Gauss-Newton.

  El resultado de este proceso será la estimación de la posición de la cámara $P_t = TP_{t-1}$ que alinea la posición de la cámara con los datos de las imágenes de profundidad y color.

7.3 Grafo de deformación

Para garantizar la consistencia global y local en el mapa se reflejan los cierres de bucle dentro del conjunto de superficies $\mathcal{M}$. Esto se realiza aplicando una deformación no rígida a todas las superficies, siguiendo las restricciones que estableces los cierres de bucle locales y globales. Para llevar a cabo estas deformaciones, se emplea un grafo de deformación. Este está compuesto por una serie de nodos y aristas distribuidas por el modelo

a ser deformado. Cada nodo G^nse define junto a su posición G_g^n, tiempo de primer muestreo $G_{t_0}^n$, y una serie de nodos vecinos $N(G^n)$. Las uniones entre los vecinos forman las aristas del grafo. En este sistema se ha trabajado con un grafo en el que cada nodo está asociado a k vecinos siendo k=4. Cada nodo además contiene una transformación afín en forma de una matriz 3x3 G_R^ny un vector 3x1 G_t^n. Cuando se realiza la deformación, estos parámetros de la transformación afín se optimizan de acuerdo con las restricciones establecida por los cierres de bucles.

Con el fin de aplicar la deformación, cada superficie $\mathcal{M}^S$ identifica una serie de nodos influentes dentro del grafo de transformación $I(\mathcal{M}^S,G)$. Se define así la posición deformada de una superficie como:

$$\hat{\mathcal{M}}_p^S = \phi(\mathcal{M}^S) = \sum_{n \in I(\mathcal{M}^S,G)} w^n(\mathcal{M}^S)[G_R^n(\mathcal{M}_p^S - G_g^n) + G_g^n + G_t^n]$$

Y la normal deformada de una superficie como:

$$\hat{\mathcal{M}}_n^S = \sum_{n \in I(\mathcal{M}^S,G)} w^n(\mathcal{M}^S) G_R^{n-1^T} \mathcal{M}_n^S$$

Siendo w^n un escalar representando la influencia que tiene el nodo G^n en la superficie $\mathcal{M}^S$.

- **Construcción del grafo**

  Con cada fotograma tomado, un nuevo grafo de deformación se construye para las superficies $\mathcal{M}$. Esto se realiza ya que es menos costoso computacionalmente volver a crear el grafo que modificarlo. Se inicializa el grafo de deformación con los nodos de posiciones y de inicialización en el tiempo iguales a los de las superficies tomados de $\mathcal{M}$. El conjunto de nodos y sus vecinos se ordenan siguiendo un orden temporal en función de su tiempo de inicialización. Este método previene que superficies no correladas en el tiempo influyan a otras, es decir, múltiples muestreos sobre una misma superficie están desconectados de los anteriores y libres para ser alineados.

- **Aplicación**

  Con el fin de aplicar el grafo de deformación para actualizar el mapa los nodos influyentes deben ser determinados. Cuando cada superficie es deformada, el conjunto completo de nodos busca el nodo más cercano en el orden temporal establecido. Esta búsqueda se establece como una búsqueda binaria, debido a que los nodos están ordenados temporalmente. Una vez localizado este nodo, los k nodos más cercanos son seleccionados. Finalmente, se calculan los pesos de cada nodo (w^n) y se aplican las transformaciones definidas previamente para la posición y la normal. Una vez realizado esto, todos los atributos de $\hat{\mathcal{M}}_n^S$ son copiados en $\mathcal{M}^S$. El algoritmo seguido para la aplicación de este grafo de deformación se puede encontrar en la figura 24.

- **Optimización**

  Dado un conjunto de correspondencias entre superficies Q construidas en el momento en que se encuentra un alineamiento en los cierres de bucle, los parámetros del grafo de deformación pueden ser optimizados. Siendo Q^p el elemento que contiene un par de puntos que determinan la posición de destino Q_d^p , la posición que debe alcanzar la posición de destino después de la deformación Q_s^p, así como los tiempos de inicialización de cada uno de ellos, se emplean 4 funciones de costes con el fin de optimizar el resultado. La primera se encarga de maximizar la rigidez en la deformación, E_{rot}; La segunda es un término de regularización que asegura una deformación suave en el grafo, E_{reg}; La tercera es un término restrictivo que minimiza el error del conjunto de posiciones Q. La cuarta comprueba la zona inactiva para asegurar que estamos deformando la zona activa en las coordenadas de la zona inactiva, E_{pin}. La función de coste completa se define como:

$$E_{def} = w_{rot}E_{rot} + w_{reg}E_{reg} + w_{con}E_{con} + w_{con}E_{pin}$$

Una vez calculado el coste total, aplicando los costes apropiados a cada coste parcial, se emplea el algoritmo iterativo de Gauss-Newton para minimizar este con respecto a G_R^n y G_t^n.

Algorithm 1: Deformation Graph Application

Input: $\mathcal{M}^s$ surfel to be deformed
$\mathcal{G}$ set of deformation nodes
α number of nodes to explore
Output: $\hat{\mathcal{M}}^s$ deformed surfel
do
// Find closest node in time
$c \leftarrow \arg\min_i \left\| \mathcal{M}^s_{t_0} - \mathcal{G}^i_{t_0} \right\|_1$
// Gather set of temporally nearby nodes
$\mathcal{I} \leftarrow \emptyset$
for $i \leftarrow -\alpha/2$ **to** $\alpha/2$ **do**
$\mathcal{I}^{i+\alpha/2} \leftarrow c + i$
sort_by_euclidean_distance$(\mathcal{I}, \mathcal{G}, \mathcal{M}^s_{\mathbf{p}})$
// Take closest k as influencing nodes
$\mathcal{I}(\mathcal{M}^s, \mathcal{G}) \leftarrow \mathcal{I}^{0 \to k-1}$
// Compute weights
$h \leftarrow 0$
$d_{max} \leftarrow \left\| \mathcal{M}^s_{\mathbf{p}} - \mathcal{G}^{\mathcal{I}^k}_{\mathbf{g}} \right\|_2$
for $n \in \mathcal{I}(\mathcal{M}^s, \mathcal{G})$ **do**
$w^n(\mathcal{M}^s) \leftarrow \left(1 - \left\| \mathcal{M}^s_{\mathbf{p}} - \mathcal{G}^n_{\mathbf{g}} \right\|_2 / d_{max}\right)^2$
$h \leftarrow h + w^n(\mathcal{M}^s)$
// Apply transformations
$\hat{\mathcal{M}}^s_{\mathbf{p}} = \sum_{n \in \mathcal{I}(\mathcal{M}^s, \mathcal{G})} \frac{w^n(\mathcal{M}^s)}{h} \left[\mathcal{G}^n_{\mathbf{R}} (\mathcal{M}^s_{\mathbf{p}} - \mathcal{G}^n_{\mathbf{g}}) + \mathcal{G}^n_{\mathbf{g}} + \mathcal{G}^n_{\mathbf{t}} \right]$
$\hat{\mathcal{M}}^s_{\mathbf{n}} = \sum_{n \in \mathcal{I}(\mathcal{M}^s, \mathcal{G})} \frac{w^n(\mathcal{M}^s)}{h} {\mathcal{G}^n_{\mathbf{R}}}^{-1^\top} \mathcal{M}^s_{\mathbf{n}}$

Figura 24 algoritmo de aplicación del grafo de deformación

7.4 Cierres locales de bucle

Para garantizar la consistencia local de las superficies este sistema se encarga de cerrar tantos bucles como pueda con el mapa existente conforme se van revisitando las zonas. Como se ha explicado previamente, el sistema se encarga de realizar la fusión de datos y la localización en las zonas activas a la vez que se encarga de clasificar aquellas zonas que no han sido vistas en un intervalo de tiempo δ_t como zonas inactivas. Esta zona inactiva no se emplea ni para la localización ni para la fusión de datos hasta que es revisitada, y por tanto clasificada como activa; y un bucle se cierra entre el modelo activo y el inactivo. El sistema divide el conjunto de superficies $\mathcal{M}$ en dos subconjuntos, el subconjunto activo Θ y el inactivo Ψ.

En cada imagen, si no ha sido localizado un cierre de bucle global, el sistema trata de encontrar una correlación entre los datos de Θ y Ψ. Esto se realiza comprobando las zonas de Ψ y Θ de la última posición estimada. El resultado de esta operación será una matriz de transformación H de la zona activa en la zona inactiva que llevará a las representaciones de dichas zonas a estar alineadas.

Para comprobar la calidad de esta transformación, se comprueba la condición final de la optimización de Gauss-Newton empleada para adaptar las dos superficies. Para ello, el valor de E_{track} explicado en la sección del seguimiento debe ser lo suficientemente pequeño mientras que el número de medidas acertadas debe ser mayor que un umbral establecido. Si estas condiciones se cumplen, se crea el conjunto de limitaciones en las superficies Q definido previamente, que se introducen en el grafo de deformación.

Una vez la deformación se ha realizado, se establece una nueva posición $\hat{P}_t = HP_t$, y se reactiva la zona inactiva que se ha empleado para cerrar el bucle que se empleará para la fusión de datos y la localización.

El método descrito en esta sección se encarga de adaptar las zonas inactivas y las activas para alcanzar cierres locales del bucle. En el caso de que el error acumulado sea demasiado grande, será necesario emplear un cierre global de bucle, en el que adaptará el modelo activo con el modelo inactivo completo.

7.5 Cierre global del bucle

Para realizar los cierres globales de bucle, se emplea un método basado en codificar las imágenes en códigos compactos. Estos códigos puedes ser empleados posteriormente para encontrar diferencias entre imágenes con el fin de, o bien determinar que es suficientemente diferente de las demás como para almacenarla como una *keyframe*, o bien para encontrar la *keyframe* más parecida. Este método se llama *randomised fern encoding*. Estos "ferns" codifican las imágenes como unas cadenas de códigos generadas a partir de test binarios en cada canal RGB-D en un conjunto de pixeles.

En este sistema en vez de comparar con el resto de imágenes RGB-D una vez codificadas, se emplean las representaciones del mapa de superficies una vez este está adaptado y fusionado con los datos de las imágenes de la cámara RGB-D.

En cada imagen se mantiene una base de datos de imágenes codificadas como se ha expuesto previamente y se comprueba esta base de datos justo después de la fase de localización y fusión de datos para comprobar si es necesario realizar un cierre de bucle global. Si se encuentra una paridad entre la visión y la base de datos que debe ser aplicada como un bucle global y no local, se realizan una serie de pasos para asegurar la adaptación global del mapa de superficies.

Primero, se intenta adaptar el fotograma encontrado con el modelo actual del mapa. Esto se lleva a cabo mediante el proceso explicado en la sección de fusión de datos y seguimiento para alinear las imágenes de profundidad y color con los datos de la base de datos. Si se lleva a cabo con éxito, se crea una matriz H que adapta el modelo actual con la imagen encontrada. Una vez realizado esto, se calculan los costes de optimización definidos en el grafo de deformación y se evalúa si la deformación propuesta es acorde al modelo geométrico del mapa. Si E_{con} es muy pequeña la deformación es rechazada. Si no lo es, el grafo de deformación se optimiza empleando Gauss-Newton y se analiza el estado final de este para comprobar si debe ser aplicado. Si después de la optimización E_{con} es suficientemente pequeña y, sobre todo, si E_{def} también lo es el cierre del bucle es aceptado y el grafo de deformación G es aplicado al conjunto completo de superficies $\mathcal{M}$. En este punto, la posición actual también es actualizada como $\hat{P}_t = H\varepsilon_p^i$. A diferencia de la situación posterior al cierre del bucle en los bucles locales, en este caso las superficies activas e inactivas no son revisadas debido a que los cierres de bucles globales dejan las superficies activas e inactivas lo suficientemente adaptadas como para que se realice un cierre del bucle en el siguiente fotograma, y además esto permite al sistema corregir el mapa en caso de que el cierre global del bucle haya sido erróneo.

7.6 Evaluación del método

Debido a que no contamos con el material necesario para realizar unas pruebas reales del método, en este punto comentaremos la evaluación realizada por los creadores del mismo que se encuentra en [20].

En cuanto a la evaluación de la trayectoria, se ha comprado el sistema empleando un archivo de testeo que proporciona medidas reales de las posiciones con respecto a la tierra; y comparándolo con otros algoritmos de

SLAM actuales que emplean también una cámara RGB-D. Para realizar estas comparaciones, se emplea la desviación del error cuadrático medio (RMSE) que se encarga de medir esta desviación sobre las distancias euclídeas entre las posiciones de la cámara y la posición real asociadas en el tiempo. Según lo indicado, los resultados son similares o mejores que los empleados en los otros métodos como se puede apreciar en la figura 26.

En lo referente a la reconstrucción de las superficies, se emplea un dataset que contiene posiciones reales de la cámara en un entorno sintético, así como un modelo 3D real que se emplea para evaluar el rendimiento de la reconstrucción de superficies. La evaluación se realiza calculando la precisión en la reconstrucción de superficies y comparándola con los mismos métodos listados previamente. Los resultados son mejores que en todos los otros sistemas excepto en un caso en el que son iguales. Estos se pueden ver en la figura 25. También se presentan los resultados obtenidos con este dataset para la estimación de trayectorias, pero no son relevantes, ya que ya se han presentado otros resultados.

System	fr1/desk	fr2/xyz	fr3/office	fr3/nst
DVO SLAM	0.021m	0.018m	0.035m	0.018m
RGB-D SLAM	0.023m	**0.008m**	0.032m	0.017m
MRSMap	0.043m	0.020m	0.042m	2.018m
Kintinuous	0.037m	0.029m	0.030m	0.031m
Frame-to-model	0.022m	0.014m	0.025m	0.027m
ElasticFusion	**0.020m**	0.011m	**0.017m**	**0.016m**

Figura 26 evaluación de las trayectorias de sistintos state-of-the-art slam empleando RMSE

System	kt0	kt1	kt2	kt3
DVO SLAM	0.032m	0.061m	0.119m	0.053m
RGB-D SLAM	0.044m	0.032m	0.031m	0.167m
MRSMap	0.061m	0.140m	0.098m	0.248m
Kintinuous	0.011m	0.008m	0.009m	0.150m
Frame-to-model	0.098m	**0.007m**	0.011m	0.107m
ElasticFusion	**0.007m**	**0.007m**	**0.008m**	**0.028m**

Figura 25 evaluación de la reconstrucción de superficies

8 CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se han desarrollado diferentes técnicas de mapeo y localización simultáneos como hemos ido viendo a lo largo de este. Para ello, se ha partido de realizar una explicación sobre SLAM, incluyendo cuando surge este y ante que necesidad, desarrollando la información desde lo básico hasta sistemas actuales de localización y mapeos. Asimismo, se han explicado los diferentes *features detectors* que podían emplear los distintos métodos, así como herramientas útiles para implementar los algoritmos, acometer pruebas y realizar representaciones gráficas en tiempo real de los mismos.

Según la metodología empleada que se explicó previamente en la introducción, se ha partido de la explicación de SLAM y, en concreto, del EKF-SLAM el cual es el algoritmo en el que nació esta metodología, y posteriormente se ha realizado un estudio teórico más a fondo de los sistemas Hector Slam, RGBD-SLAM y ElasticFusion, así como unas pruebas posteriores o, en el caso de que no fuera posible realizar dichas pruebas, una evaluación.

De los resultados de las pruebas citadas y las evaluaciones, los resultados de los distintos algoritmos son buenos y, por lo tanto, muy útiles para lograr el fin que pretenden, dotar al robot de la autonomía necesaria para realizar las tareas que se le encomienden en un entorno desconocido.

Los tres métodos estudiados no son comparables entre si ya que no se enfocan al mismo objetivo. Mientras RGBD-SLAM y ElasticFusion se encargan de dar una visión tridimensional detallada del entorno y realizar la localización, Hector SLAM genera mapas bidimensionales a pesar de que calcula la posición con 6 grados de libertad empleando una potencia de computo mucho menor que los otros dos sistemas. En este sentido, para operaciones en las que el robot solo necesite moverse por el entorno evitando colisiones sin la necesidad de ejercer una tarea de manipulación, podemos decir que Hector SLAM es un método mucho más eficiente que los otros dos. A pesar de esto, también hay que destacar que este sistema emplea un láser LIDAR por lo que, a pesar de que se necesitará un hardware mucho menos potente en el robot para implementar el algoritmo, el precio del LIDAR es mucho más elevado que el de la cámara RGB-D en la que se basan los otros dos sistemas.

En cuanto a los otros dos sistemas, según la evaluación realizada en el capítulo 7.6, ElasticFusion devuelve unos resultados más precisos que RGBD-SLAM a la hora de realizar la localización y la reconstrucción de mapas a pesar de basarse en un grafo de deformación en vez de una optimización de grafo de posiciones. Además, el sistema se ha realizado con un enfoque más centrado en la construcción del mapa y dirigido a cualquier tipo de entornos interiores, tanto pasillos en los que apenas se revisitan lugares previamente mapeados, como en habitaciones con movimientos entrecruzados, a diferencia de RGBD-SLAM que solo está orientados a estos últimos. Sin embargo, también cabe mencionar que ElasticFusion hace uso de la GPU mediante las librerías CUDA por lo que se necesita un hardware más potente y específico que para la implementación de RGBD-SLAM, que además permite realizar el algoritmo offline. Además, también hay que tener en cuenta que mientras que el algoritmo de RGBD-SLAM está completamente definido y creado para su uso en ROS, ElasticFusion presenta su propio código de compilación, aunque también sea de código abierto e introducido en GitHub, por lo que puede resultar más complejo a la hora de su ejecución e implementación en sistemas robóticos.

8.1 Trabajo futuro

La idea inicial consistía en el estudio y la implementación en un sistema real del método ElasticFusion, poco desarrollado hasta ahora por lo que se ha dejado esa puerta abierta como trabajo futuro. Asimismo, además de hacer el desarrollo práctico de este, también se podría realizar el desarrollo de los otros métodos mencionados

en plataformas robóticas reales, no en simulaciones o con los sensores sujetos con la mano; y contando con habitaciones que dotaran de un sistema de posicionamiento preciso para emplearlo como *ground truth* con el fin de realizar las pruebas y comparaciones basadas en el cálculo de unos errores precisos.

También se podría implementar un trabajo multisistemas en los que interactuasen varios robots en un mismo entorno, comprobando los resultados a la hora de realizar los mapeos e implementando algoritmos para evitar colisiones, así como para planificar trayectorias óptimas una vez se ha realizado el mapeo. Además, si la información de estos fuese comunicada se reducirían los tiempos de mapeo y la potencia de computo necesaria para realizar SLAM en el entorno correspondiente.

Otra de las ideas que también surgieron a la hora de la planificación del trabajo fue el control de la odometría. En los sistemas que hacen uso de ella, hemos asumido que esta es una entrada exterior al sistema, pero esta debe ser recogida por una serie de sistemas y procesada para poder ser posteriormente empleada por estos. En concreto, surgió la idea del control de la odometría en z para el empleo de plataformas móviles aéreas con el fin de mantener estas en *hoovering*, implementar algoritmos de control precisos para el sistema o sistemas de posicionamiento y aterrizaje en plataformas móviles; todo esto mediante el uso de un sensor Px4 sonar o algún otro sensor similar o que sea capaz de aportar la misma clase de resultados al sistema.

Figura 27 Px4 sonar

REFERENCIAS

[1] R. Smith y P. Cheeseman, «Representation of spatial uncertainty,» *Int. J. Robotics Research,* nº 5(4):, pp. 56-68, 1987.

[2] H. Durrant-Whyte y L. , «Mobile robot localization by tracking geometric beacons».

[3] R. Smith, P. Cheeseman y M. Self, «Estimating uncertain spatial realtionships in robotics,» *Autonomous Robot Vehicles Spriner ,* pp. 167-193.

[4] H. Durrant-Whyte y T. Bailey, «Simultaneous localisation and mapping (slam): Part i the essential algorithms.,» *IEEE ROBOTICS AND AUTOMATION MAGAZINE,* 2006.

[5] S. Riisgaard y M. R. Blas, SLAM for Dummies, 2004.

[6] G. Zunino, «Sensors and features,» de *Simultaneous Localization and Mapping for Navigation in Realistic Environments*, Estocolmo, Royal Institute of Technology Numerical Analysis and Computer Science, 2002, pp. 5-10.

[7] D. G. Lowe, Distinctive Image Features from Scale-invariant Keypoint, Vancouver, 2004.

[8] H. Bay, T. Tuytelaars y L. Van Gool, SURF: Speeded Up Robust Features, Zurich, 2006.

[9] E. Rosten y T. Drummond, Machine learning for high-speed corner detection, Department of Engineering, Cambridge University.

[10] E. Rublee, V. Rabaus, K. Konolige y G. Bradski, ORB: an efficient alternative to SIFT or SURF, California: Willow Garage.

[11] J. Solà, Simultaneous localization and mapping with extended Kalman filter, 2014.

[12] L. Y. Ku, «RVIZ: a good reason to implement a vision system in ROS,» *The Srious Computer Vision Blog,* 2012.

[13] O. S. R. Foundation, Gazebo [Online], Available: http://gazebosim.org.

[14] O. von Stryk, S. Kholbrecher , J. Meyer y U. Klingauf, A Flexible and Scalable SLAM System with Full 3D Motion Estimation, Darmstadt: Technische Universität Darmstadt.

[15] W. Garage, ROS Hector SLAM [Online], Available: http://wiki.ros.org/hector_slam.

[16] G. Grisetti , R. Kümmerle, C. Stachniss y W. Burgard, A Tutorial on Graph-Based SLAM, Germany: University of Freiburg.

[17] F. Endres, J. Hess, J. Sturm, D. Cremers y W. Burgard, «3D mapping with an RGB-D Camera,» *IEEE TRANSACTIONS ON ROBOTICS,* vol. 30, nº 1, 2014.

[18] G. Facciolo y V. Caselles, Geodesic neigthbourhoods for piecewise afinne interpolation of sparse data, Barcelona: Universitat Pompeu Fabra.

[19] W. Garage, ROS rgbdslam [Online], Available: http://wiki.ros.org/rgbdslam.

[20] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker y A. J. Davison, ElasticFusion: Dense SLAM Without A Pose Graph, London : Deparment of Computing, Imperia College London.

APÉNDICE

Apéndice I: tutorial.launch Hector SLAM

```
<?xml version="1.0"?>

<launch>

  <arg name="geotiff_map_file_path" default="$(find hector_geotiff)/maps"/>

  <param name="/use_sim_time" value="true"/>

  <node pkg="rviz" type="rviz" name="rviz"
    args="-d $(find hector_slam_launch)/rviz_cfg/mapping_demo.rviz"/>

  <include file="$(find hector_mapping)/launch/mapping_default.launch"/>

  <include file="$(find hector_geotiff)/launch/geotiff_mapper.launch">
    <arg name="trajectory_source_frame_name" value="scanmatcher_frame"/>
    <arg name="map_file_path" value="$(arg geotiff_map_file_path)"/>
  </include>

</launch>
```

Apéndice II: geotiff_mapper.launch Hector SLAM

```
<?xml version="1.0"?>

<launch>
  <arg name="trajectory_source_frame_name" default="/base_link"/>
  <arg name="trajectory_update_rate" default="4"/>
  <arg name="trajectory_publish_rate" default="0.25"/>
  <arg name="map_file_path" default="$(find hector_geotiff)/maps"/>
  <arg name="map_file_base_name" default="hector_slam_map"/>

  <node pkg="hector_trajectory_server" type="hector_trajectory_server" name="hector_trajectory_server" output="screen">
    <param name="target_frame_name" type="string" value="/map" />
    <param name="source_frame_name" type="string" value="$(arg trajectory_source_frame_name)" />
    <param name="trajectory_update_rate" type="double" value="$(arg trajectory_update_rate)" />
    <param name="trajectory_publish_rate" type="double" value="$(arg trajectory_publish_rate)" />
  </node>

  <node pkg="hector_geotiff" type="geotiff_node" name="hector_geotiff_node" output="screen" launch-prefix="nice -n 15">
    <remap from="map" to="/dynamic_map" />
    <param name="map_file_path" type="string" value="$(arg map_file_path)" />
    <param name="map_file_base_name" type="string" value="$(arg map_file_base_name)" />
    <param name="geotiff_save_period" type="double" value="0" />
    <param name="draw_background_checkerboard" type="bool" value="true" />
    <param name="draw_free_space_grid" type="bool" value="true" />
    <param name="plugins" type="string" value="hector_geotiff_plugins/TrajectoryMapWriter" />
  </node>

</launch>
```

Apéndice III: Mapping_default.launch Hector SLAM

```
<?xml version="1.0"?>

<launch>
  <arg name="tf_map_scanmatch_transform_frame_name" default="scanmatcher_frame"/>
  <arg name="base_frame" default="base_footprint"/>
  <arg name="odom_frame" default="nav"/>
  <arg name="pub_map_odom_transform" default="true"/>
  <arg name="scan_subscriber_queue_size" default="5"/>
  <arg name="scan_topic" default="scan"/>
  <arg name="map_size" default="2048"/>

  <node pkg="hector_mapping" type="hector_mapping" name="hector_mapping" output="screen">

    <!-- Frame names -->
    <param name="map_frame" value="map" />
    <param name="base_frame" value="$(arg base_frame)" />
    <param name="odom_frame" value="$(arg odom_frame)" />

    <!-- Tf use -->
    <param name="use_tf_scan_transformation" value="true"/>
    <param name="use_tf_pose_start_estimate" value="false"/>
    <param name="pub_map_odom_transform" value="$(arg pub_map_odom_transform)"/>

    <!-- Map size / start point -->
    <param name="map_resolution" value="0.050"/>
    <param name="map_size" value="$(arg map_size)"/>
    <param name="map_start_x" value="0.5"/>
    <param name="map_start_y" value="0.5" />
    <param name="map_multi_res_levels" value="2" />

    <!-- Map update parameters -->
    <param name="update_factor_free" value="0.4"/>
    <param name="update_factor_occupied" value="0.9" />
    <param name="map_update_distance_thresh" value="0.4"/>
    <param name="map_update_angle_thresh" value="0.06" />
    <param name="laser_z_min_value" value = "-1.0" />
    <param name="laser_z_max_value" value = "1.0" />

    <!-- Advertising config -->
    <param name="advertise_map_service" value="true"/>

    <param name="scan_subscriber_queue_size" value="$(arg scan_subscriber_queue_size)"/>
    <param name="scan_topic" value="$(arg scan_topic)"/>

    <!-- Debug parameters -->
    <!--
      <param name="output_timing" value="false"/>
      <param name="pub_drawings" value="true"/>
      <param name="pub_debug_output" value="true"/>
    -->
    <param name="tf_map_scanmatch_transform_frame_name" value="$(arg tf_map_scanmatch_transform_frame_name)" />
  </node>

  <!--<node pkg="tf" type="static_transform_publisher" name="map_nav_broadcaster" args="0 0 0 0 0 0 map nav 100"/>-->
</launch>
```

Apéndice IV: package.xml rgbdslam

```
<?xml version="1.0"?>
<package>
  <name>rgbdslam</name>
  <version>0.2.0</version>
  <description>
    rgbdslam (v2) is a SLAM solution for RGB-D cameras. It provides the current pose of the camera and allows to
    create a registered point cloud or an octomap.

    It features a GUI interface for easy usage, but can also be controlled by ROS service calls, e.g., when running on a robot.

    For installation and usage instructions see the README file of <a href="https://github.com/felixendres/rgbdslam_v2">the github repository</a>.
  </description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="endres@informatik.uni-freiburg.de">Felix Endres</maintainer>


  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>GPLv3</license>
  <url type="website">http://ros.org/wiki/rgbdslam</url>


  <!-- Author tags are optional, mutiple are allowed, one per tag -->
  <!-- Authors do not have to be maintianers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->
  <author>Felix Endres</author>
  <author>Juergen Hess</author>
  <author>Nikolas Engelhard</author>


  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>pcl_conversions</build_depend>
  <build_depend>pcl_ros</build_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>sensor_msgs</build_depend>
  <build_depend>geometry_msgs</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>octomap</build_depend>
  <build_depend>visualization_msgs</build_depend>
  <build_depend>cmake_modules</build_depend>
  <run_depend>pcl_conversions</run_depend>
  <run_depend>pcl_ros</run_depend>
  <run_depend>roscpp</run_depend>
  <run_depend>std_msgs</run_depend>
  <run_depend>sensor_msgs</run_depend>
  <run_depend>octomap</run_depend>
  <run_depend>visualization_msgs</run_depend>
  <run_depend>cmake_modules</run_depend>
  <!-- own stuff -->

  <build_depend>libpcl-all-dev</build_depend>
  <build_depend>tf</build_depend>
  <build_depend>message_generation</build_depend>
  <build_depend>image_transport</build_depend>
  <build_depend>rosbag</build_depend>
  <build_depend>glut</build_depend>
  <build_depend>libglew-dev</build_depend>
  <build_depend>libdevil-dev</build_depend>

  <run_depend>rosbag</run_depend>

  <run_depend>cv_bridge</run_depend>
  <run_depend>libpcl-all</run_depend>
  <run_depend>tf</run_depend>
  <run_depend>message_runtime</run_depend>

  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- You can specify that this package is a metapackage here: -->
    <!-- <metapackage/> -->

    <!-- Other tools can request additional information be placed here -->

  </export>
</package>
  <!--
```

Apéndice V: rgbdslam.launch

```
<!-- This file shows the most important parameters in their default settings,
     to make them easily available for beginners.

     The openni driver has to be started seperately, e.g. with "roslaunch openni_launch openni.launch" -->
<launch>
  <node pkg="rgbdslam" type="rgbdslam" name="rgbdslam" cwd="node" required="true" output="screen">
    <!-- Input data settings-->
    <param name="config/topic_image_mono"             value="/camera/rgb/image_color"/>
    <param name="config/topic_image_depth"            value="/camera/depth_registered/sw_registered/image_rect_raw"/>
    <param name="config/topic_points"                 value=""/> <!--if empty, poincloud will be reconstructed from image and depth -->

    <!-- These are the default values of some important parameters -->
    <param name="config/feature_extractor_type"       value="ORB"/><!-- also available: SIFT, SIFTGPU, SURF, SURF128 (extended SURF), ORB. -->
    <param name="config/feature_detector_type"        value="ORB"/><!-- also available: SIFT, SURF, GFTT (good features to track), ORB. -->
    <param name="config/detector_grid_resolution"     value="3"/><!-- detect on a 3x3 grid (to spread ORB keypoints and parallelize SIFT and SURF) -->
    <param name="config/max_keypoints"                value="600"/><!-- Extract no more than this many keypoints -->
    <param name="config/max_matches"                  value="300"/><!-- Keep the best n matches (important for ORB to set lower than max_keypoints) -->

    <param name="config/min_sampled_candidates"       value="4"/><!-- Frame-to-frame comparisons to random frames (big loop closures) -->
    <param name="config/predecessor_candidates"       value="4"/><!-- Frame-to-frame comparisons to sequential frames-->
    <param name="config/neighbor_candidates"          value="4"/><!-- Frame-to-frame comparisons to graph neighbor frames-->
    <param name="config/ransac_iterations"            value="100"/>
    <param name="config/cloud_creation_skip_step"     value="2"/><!-- subsample the images' pixels (in both, width and height), when creating the cloud (and therefore
reduce memory consumption) -->


    <param name="config/cloud_display_type"           value="POINTS"/><!-- Show pointclouds as points (as opposed to TRIANGLE_STRIP) -->
    <param name="config/pose_relative_to"             value="largest_loop"/><!-- optimize only a subset of the graph: "largest_loop" = Everything from the earliest
matched frame to the current one. Use "first" to optimize the full graph, "inaffected" to optimize only the frames that were matched (not those inbetween for loops) -->
    <param name="config/backend_solver"               value="pcg"/><!-- pcg is faster and good for continuous online optimization, cholmod and csparse are better for
offline optimization (without good initial guess)-->
    <param name="config/optimizer_skip_step"          value="1"/><!-- optimize only every n-th frame -->
  </node>
</launch>
```