UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA

INGENIERÍA MECÁNICA – INGENIERÍA MECATRÓNICA

LOCALIZACIÓN Y MAPEO SIMULTÁNEOS PARA ROBOT DE BÚSQUEDA EN ENTORNOS DE DESASTRE

TESIS

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN INGENIERÍA MECÁNICA

PRESENTA:

GABRIEL ELEAZAR ARROYO SÁNCHEZ

TUTOR

YUKIHIRO MINAMI KOYAMA, FACULTAD DE INGENIERÍA

MÉXICO, D. F. NOVIEMBRE 2013

JURADO ASIGNADO:

Presidente:	Víctor Javier González Villela
Secretario:	Jesús Manuel Dorador González
Vocal:	Yukihiro Minami Koyama
Primer Suplente:	Jesús Savage Carmona
Segundo Suplente:	Billy Arturo Flores Medero Navarro

Lugar o lugares donde se realizó la tesis:

DIVISIÓN DE ESTUDIOS DE POSGRADO, FACULTAD DE INGENIERÍA, UNAM.

TALLE ORG, DIVISIÓN DE CIENCIAS BÁSICAS, FACULTAD DE INGENIERÍA, UNAM.

TUTOR DE TESIS:

YUKIHIRO MINAMI KOYAMA

FIRMA

AGRADECIMIENTOS

Al Programa de Posgrado e Ingeniería de la UNAM que me permitió continuar mi formación.

Al CONACyT por el apoyo económico brindado para la realización de este proyecto y de mis estudios de maestría.

Al programa UNAM-DGAPA-PAPIIT-IT103712 "FinDER: robot de búsqueda en entornos de desastre" por el apoyo económico.

A la División de Ciencias Básicas de la Facultad de Ingeniería por brindar el espacio de trabajo.

A mis sinodales, Víctor Javier González Villela, Jesús Manuel Dorador González, Yukihiro Minami Koyama, Jesús Savage Carmona y Billy Arturo Flores Medero Navarro por sus consejos y el tiempo invertido en las revisiones de esta investigación.

Al Taller ORG de la Facultad de Ingeniería por hacer posible este proyecto y a sus miembros por su invaluable amistad, compañerismo, apoyo y guía.

A mi tutor, Yukihiro Minami Koyama, por su apoyo total hacia esta investigación y hacia mi persona. Su sabio consejo me ubicó en mi camino, su constante motivación me impulsó a seguir adelante, su comprensión me dio confianza y su entereza y dedicación me dió un ejemplo a seguir.

A mis padres, que no solo me dieron la vida, si no que la potenciaron y han sido mi más grande apoyo hasta el día de hoy.

A Jhovanna, por su amistad incondicional.

A Raquel, por creer en mí y no dejarme caer. Sin tí este logro no hubiera sido posible.

Y a Dios, quien es la fuente de amor y fortaleza que me han traido hasta aquí.

CONTENIDO

RESUMEN..... 9

ABSTRACT..... 10

1 INTRODUCCIÓN..... 11

1.1 ANTECEDENTES..... 11

1.1.1 ENTORNOS DE DESASTRE..... 11

1.1.2 ROBÓTICA..... 13

1.1.3 PROYECTO FINDER..... 14

1.2 ESTADO DEL ARTE..... 18

1.3 OBJETIVOS..... 19

1.4 HIPÓTESIS..... 19

SÍNTESIS DE CAPÍTULOS..... 21

2 ANÁLISIS DEL PROBLEMA..... 23

2.1 DEFINICIÓN DEL SISTEMA..... 23

2.2 CRITERIOS DE SELECCIÓN..... 26

2.3 RESTRICCIONES..... 27

2.4 ESPECIFICACIONES..... 29

2.5 METODOLOGÍA..... 29

3 DISEÑO..... 31

3.1 ANÁLISIS DE COMPONENTES PARA RESOLVER EL PROBLEMA..... 31

3.1.1 SENSORES EXISTENTES QUE PUEDEN SER DE AYUDA..... 31

3.1.2 ROS..... 35
3.1.3 AGV..... 36

3.2 ANÁLISIS DE HERRAMIENTAS COMPUTACIONALES PARA PROCESAR LOS DATOS..... 38

3.2.1 SLAM..... 38

4 TOMA DE DECISIONES..... 57

4.1 METODOLOGÍA..... 57

4.2 SELECCIÓN..... 58

5 CONSTRUCCIÓN, PRUEBAS Y RESULTADOS..... 61

5.1 CONSTRUCCIÓN DEL SISTEMA..... 61

5.2 PUESTA EN MARCHA..... 67

5.3 IMPLEMENTACIÓN..... 71

5.4 PRUEBAS Y RESULTADOS..... 75

6. DISCUSIÓN..... 77

7. CONCLUSIONES Y TRABAJO A FUTURO..... 79

8. REFERENCIAS..... 81

9. APÉNDICES..... 83

APÉNDICE A (kinectalaser.launch) 83

APÉNDICE B (slam.launch) 85

APÉNDICE C (finder_slam.cpp) 86

RESUMEN

Los robots móviles son cada vez más usados para tareas repetitivas, peligrosas o difíciles de hacer para humanos, además su autonomía también ha ido mejorando. Sin embargo, en muchas ocasiones los equipos de rescate no cuentan con un mapa del lugar o edificios afectados, esto hace más difíciles las tareas de búsqueda y rescate. En la navegación autónoma o teleoperada es esencial para el robot crear el mapa y localizarse en él al mismo tiempo. Este procedimiento se conoce como Localización y Mapeo Simultáneos o SLAM (por sus siglas en inglés) y es considerado un problema complejo, ya que para la localización el robot requiere un mapa consistente y para construir el mapa es necesaria una buena estimación de la localización. Esta mutua dependencia entre la estimación de la posición y la construcción del mapa es el núcleo del problema del SLAM y requiere trabajo de investigación para encontrar su solución para ambientes amplios y desconocidos.

El propósito de esta tesis es obtener un método de SLAM que pueda ser usado en el sistema de navegación a bordo de un robot de búsqueda y rescate. Para lograr esto se exploran varias técnicas, desde el uso de tecnologías de última generación como el sensor Kinect hasta nuevos y poderosos algoritmos computacionales.

ABSTRACT

Mobile robots are increasingly used for repetitive, dangerous or difficult tasks for humans and their autonomy capabilities are also growing. However, in many cases rescue teams doesn't have a map of the place or buildings affected, that makes the search and rescue tasks even harder. For bout its automous or teleoperated navigation it is essential for the robot to create a map and locate its self at the same time. This procedure is known as Simultaneous Localization and Mapping or SLAM and is considered a complex problem, as for localization the robot requires a consistent map and for constructing the map a good estimation of its localization is needed. This mutual dependency between position estimation and map construction is the core of the SLAM problem and it requires research work to find its solution for an unknown and large environments.

The main purpose of this thesis is to obtain a SLAM method that can be used for the onboard navigation system of a search and rescue robot. To achieve this various techniques are explored, from the highest generation technologies as is the Kinect sensor to many new and powerful computational algorithms.

1

INTRODUCCIÓN

1.1 ANTECEDENTES

1.1.1 ENTORNOS DE DESASTRE

Los desastres son eventos calamitosos, repentinos o imprevisibles que trastornan seriamente el funcionamiento de una comunidad o sociedad y causan enormes pérdidas humanas, materiales, económicas o ambientales.. Se habla de desastres naturales en el momento en que eventos o fenómenos naturales como terremotos, inundaciones o deslizamientos de tierra superan su límite de normalidad ocasionando grandes afectaciones a la comunidad.

Algunos desastres pueden ser causados también por las actividades humanas, que alteran la normalidad del medio ambiente. Se tienen como ejemplo: la contaminación del medio ambiente, la explotación errónea e irracional de los recursos naturales, fallas en reactores nucleares, la construcción de viviendas y edificaciones en zonas de alto riesgo e inclusive conflictos bélicos. Dichos efectos pueden amplificarse debido a la falta de medidas de seguridad, planes de emergencia y sistemas de alerta.

Dependiendo de la capacidad de cada comunidad para reducir el riesgo colectivo de desastres, éstos pueden desencadenar otros eventos que reducirán la posibilidad de sobrevivir a dicho riesgo debido a carencias en la planificación y en las medidas de seguridad. Un ejemplo clásico son los terremotos, que derrumban edificios y casas, dejando atrapadas a personas entre los escombros y rompiendo tuberías de gas que pueden causar incendios y quemar a los heridos bajo las ruinas.

Las áreas con alta probabilidad de desastres naturales se conocen como de alto riesgo. Zonas de alto riesgo sin instrumentación ni medidas apropiadas para responder al desastre natural o reducir sus efectos negativos se conocen como de zonas de alta vulnerabilidad.

Los terremotos en Haití y Chile, el tsunami en Indonesia y los ataques terroristas en Irak han cobrado más de un millón de víctimas, pese a los grandes esfuerzos realizados por los organismos de rescate para auxiliar a los heridos que quedan en riesgo bajo los escombros.

La mayoría de las veces, esto ocurre porque a los rescatistas les surgen una serie de inconvenientes: no pueden entrar a la edificación colapsada ya que ponen en riesgo su propia vida, los caminos para ingresar al lugar están llenos de escombros o tienen dificultad para ubicar el sitio preciso donde están las personas atrapadas.

Es por eso que en la actualidad se está buscando cada vez más el apoyo de nuevas herramientas que permitan actuar con mayor eficiencia y seguridad a los equipos de rescate. Una de las áreas en auge que se encarga de satisfacer esta necesidad es la robótica.

1.1.2 ROBÓTICA

Los robots son máquinas en las que se integran componentes mecánicos, eléctricos, electrónicos y de comunicaciones, y dotadas de un sistema informático para su control en tiempo real, percepción del entorno y programación. A pesar de que el término se estableció desde los años veinte del siglo pasado, la robótica industrial nace en los cincuenta y es hasta la década de los setenta que se empieza a impartir como materia en las universidades.

En la robótica industrial se dota de flexibilidad a los procesos productivos, manteniendo al mismo tiempo la productividad que se consigue con una máquina automática especializada.

Los robots se han vuelto cada vez más presentes en la vida diaria. Se pueden ver en las casas aspirando el piso, cortando el pasto o incluso enfriando los alimentos a la temperatura correcta. Los llamados robots de servicio, robots domésticos, robots de ayuda a discapacitados y robots de ayuda en general forman parte de un área que crece a ritmo acelerado y que en unos años será posible verlos cocinando diversos platillos o llevando a sus dueños al trabajo.

Los grandes avances en el diseño y construcción de los robots los han hecho herramientas cada vez más útiles en todas las áreas del quehacer humano. Sobre todo, han demostrado su valía desarrollando tareas muy repetitivas y tediosas, como rellenar botellas de agua en una envasadora y tareas muy pesadas para los humanos como ensamblar las partes de un automóvil en una línea de producción; también han aumentado el alcance y precisión para muchas tareas como en aplicaciones médicas o incluso realizando tareas muy peligrosas como manipulación de explosivos o elementos radioactivos.

Los robots móviles se diferencian de los robots manipuladores convencionales en que no están anclados, sino que pueden desplazarse por el terreno, por el agua o incluso volar libremente.

En el caso de las tareas de búsqueda y rescate en las situaciones de desastre, ya sea por causas naturales como terremotos o huracanes, o por consecuencia del comportamiento humano como incendios en zonas urbanas o desastres nucleares, la robótica se ha convertido en una gran aliada ayudando a mejorar el desempeño de los equipos de rescate humanos.

Dadas las condiciones de terreno después de un desastre, los robots utilizados deben contar con una gran movilidad. Algunas de las estrategias con potencial para ser usadas son los grupos de robots pequeños que cooperan hacia metas comunes, robots de morfología flexible que se les facilite la movilidad en ambientes irregulares o el uso de vehículos aéreos no tripulados cuyos seis grados de libertad ayudarían a recorrer el lugar entrando desde cualquier ángulo.

1.1.3 PROYECTO FINDER

A la vista de estas necesidades y problemas ya descritos, el grupo de trabajo del Taller .ORG de la Facultad de Ingeniería decidió emprender un proyecto que abriera una línea de investigación encaminada a encontrar soluciones pertinentes y aplicables desarrollado por los alumnos de la Facultad.

Así nació el proyecto FinDER (Finder in DisasterEnvironment Robot), o robot de búsqueda en entornos de desastre, el cual tiene como finalidad el diseño y construcción de un robot capaz de navegar de forma remota (teleoperada) o en modo autónomo en entornos post-desastre de eventos como sismos, derrumbes y explosiones; su tarea principal es la búsqueda de víctimas humanas, pero también se pretende que pueda asistirlas en la medida de sus posibilidades y determinar su estado físico, con el objeto de establecer un plan de rescate.

Este es un proyecto multidisciplinario que abarca muchos temas concernientes a la robótica como diseño mecánico, control, visión artificial, mapeo, localización, procesamiento de señales, entre otras áreas de estudio que se han abordado, para poder dotar al robot con las capacidades necesarias para ayudar a los equipos humanos en una situación de búsqueda y rescate en ambientes de desastre, permitiéndoles trabajar con mayor seguridad y efectividad, por ejemplo, inspeccionando la zona de desastre en búsqueda de factores de peligro como gases nocivos o estructuras poco firmes antes de que ingresen los rescatistas.

Para cumplir con sus objetivos, se ha equipado al robot con diversos sensores y actuadores, tanto comerciales como de diseño propio.

El diseño mecánico del FinDER se ha enfocado en una solución simple pero que permita una gran movilidad sobre terreno agreste, por lo que se decidió usar ruedas de bicicleta de rin 20 en la parte delantera y en el lugar del punto fijo de apoyo, se consideraron dos ruedas omnidireccionales que facilitan el giro sobre su propio eje y la navegación sobre obstáculos irregulares y a cada lado del robot, izquierdo y derecho, se usó un motor independiente, lo que es similar a una configuración diferencial; se decidió emplear una suspensión independiente con brazos articulados al centro del robot, que la velocidad fuera de al menos 0.5 m/s, que fuera capaz de girar sobre su propio eje y que las ruedas tuvieran un alto coeficiente de fricción.

SENSORES Y ACTUADORES

El robot cuenta con conjunto de sensores y actuadores con los que puede percibir su entorno, facilitándole la navegación y permitiéndole lograr la detección de personas y de riesgos, además de poder con ellos interactuar con el ambiente.

UNIDAD DE MEDICIÓN INERCIAL (IMU, POR SUS SIGLAS EN INGLÉS)

Es un conjunto de sensores conformado por acelerómetros, giroscopios, magnetómetro, termómetro y barómetro. Su principal función es brindar información que ayude a determinar la posición de un agente.

CÁMARA PLAYSTATION EYE

El PlayStation Eye es una cámara fabricada por SONY para el PlayStation 3. Este dispositivo puede tomar imágenes con una resolución de 640x480 pixeles en 60 Hz o imágenes con 320x240 pixeles en 120Hz, convirtiéndolo en un excelente aparato para aplicaciones de robótica. El objetivo de esta cámara es utilizarla en la exploración de lugares en donde el acceso sea limitado y será operada con la ayuda de un manipulador.

CÁMARA WEB

Una cámara web con una resolución de 320x240 a 30 Hz, se encarga de obtener imágenes que posteriormente son procesadas en el módulo de detección de víctimas. Esto permite por medio de algoritmos de visión artificial saber cuándo se encuentra una forma humana en el campo de visón.

MICRÓFONO

Se utiliza para escuchar a las víctimas y obtener retroalimentación que dé información que agilice el proceso de búsqueda. Se tiene un arreglo de cuatro micrófonos con el que se puede calcular la ubicación de las señales con base en un procesamiento de los datos guardados por cada uno. La señal obtenida se acondiciona con un filtro que minimiza ruidos del ambiente y amplifica las frecuencias de la voz humana.

SENSOR DE CO_2

Se tiene un arreglo de elementos electrónicos que le permiten a un sensor de gas CO_2 detectar cierta cantidad de concentración de gas circundante en el medio, menos de 500 ppm. La concentración de CO_2 que emite un ser humano es un factor importante para la determinación de signos vitales, ya que es un subproducto característico de la respiración.

SENSOR DE TEMPERATURA

El robot FinDER cuenta con un dispositivo TPA81, que es un sensor de temperatura conformado por una matriz de 8 detectores individuales colocados linealmente. Este dispositivo detecta calor mediante un sistema de señales infrarrojas, el cual identifica ondas en un rango de 2 a 22 μm que es la longitud de onda del calor radiante.

PANTALLA TÁCTIL

Con el fin de dotar al robot de una interfaz humano-máquina adecuada y de fácil operación, se dotó al robot de un una pantalla táctil de 7 pulgadas. Mediante ésta, el operador puede fácil y rápidamente comenzar una misión de búsqueda. La pantalla táctil también podría ofrecer un menú de opciones a la víctima, permitiendo el ingreso de datos o retroalimentación al software de control y navegación.

BRAZO MANIPULADOR

Debido a las diferentes circunstancias en que se pueden encontrar las víctimas, el robot requiere poder interactuar con ellas con el fin de brindarle consumibles de primera necesidad como son agua, medicamentos, cubre bocas, entre otros. Además, resulta conveniente contar con una cámara que se pueda mover más allá de los límites geométricos del robot, por ejemplo para obtener una vista superior del lugar. Por lo tanto, es conveniente que cuente con un brazo robótico que le permita estas capacidades. Para cubrir dicha necesidad se diseñó un brazo robótico a base de aluminio para que fuera ligero. Sus tres motores de corriente directa y sus tres servomotores le dan cinco grados de libertad y el movimiento de las pinzas, que son el actuador terminal. El alcance máximo es de 1 m y puede soportar una carga máxima de 1 kg cuando se encuentra completamente extendido.

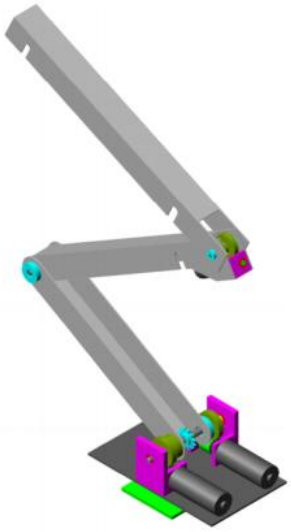


Figura 1.1 Brazo manipulador.

BOCINAS

En un ambiente de desastre, una tarea primordial de un equipo de rescate es establecer la comunicación con las víctimas. Las bocinas permitirán a FinDER entablar una conversación entre el rescatista teleoperador y la víctima, para que ésta reciba instrucciones auditivas y mensajes importantes directamente. Esta capacidad también puede explotarse en modo de operación autónomo. Las características a considerar en su elección son el tamaño, peso y la potencia de amplificación que deberá ser al menos 4 ó 5 W (rms). Se usarán bocinas activas multimedia con alimentación USB y la terminal convencional para señal de audio.

LÁMPARA DE LEDS

Se incluirá una lámpara de leds que facilite la exploración del entorno al robot en condiciones que imposibiliten o desfavorezcan el sensado a través de las cámaras de visión. La lámpara podría apuntarse a una región específica de interés para poder percibir a través de la cámara PlayStation Eye el entorno en busca de víctimas humanas, es por eso que será colocada como complemento terminal en el manipulador del robot.

Figura 1.2FinDER (Finder in Disaster Environment Robot).

1.2 ESTADO DEL ARTE

Los avances actuales en robótica han permitido una mayor autonomía y movilidad a los agentes móviles, los nuevos algoritmos de navegación les dan la capacidad de moverse dentro de ambientes controlados, sorteando obstáculos generalmente fijos y llegar a su objetivo. Un ejemplo de esto son los vehículos de guiado automático o AGV, por sus siglas en inglés, los cuales se usan en fábricas y pueden transitar por sí solos transportando materiales o herramientas.

Sin embargo, los entornos de desastre son ambientes muy complicados para la navegación teleoperada y aún más para la navegación autónoma, se trata de ambientes con geometrías totalmente impredecibles y con posibles obstáculos en los tres ejes de desplazamiento. Es por eso que se han multiplicado los esfuerzos por desarrollar algoritmos confiables que permitan la navegación autónoma de robots móviles en entornos desconocidos.

Las competencias de la DARPA Urban Challenge (DARPA son las siglas en inglés de Defense Advanced Research Projects Agency) tienen como meta construir vehículos autónomos capaces de navegar en ambientes de tránsito urbano. Por su parte, el Instituto Fraunhofer de Manufactura, Ingeniería y Automatización (Fraunhofer IPA, por las siglas en alemán de Fraunhofer-Instituts für Produktionstechnikund Automotisierung) está desarrollando guías móviles autónomos. Son tres robots que guían a los visitantes en un museo y proporcionan información audiovisual adicional. El éxito de las aplicaciones depende altamente de su capacidad de navegar por todo el lugar sin perderse, es decir, en la precisión y robustez de la implementación de sus sistemas de navegación, mapeo y localización.

Figura 1.3 Vehículo ganador de la competencia DARPA 2007.

1.3 OBJETIVOS

Los robots móviles son usados cada vez más en tareas peligrosas o difíciles para humanos y además va aumentando su capacidad de autonomía. Sin embargo, en muchas ocasiones no se tiene previamente un mapa del lugar donde el robot debe operar. Para su navegación autónoma es importante que el robot pueda crear un mapa y localizarse en él para poder navegar en su entorno. Dicho problema de navegación, mapeo y localización constituye uno de los problemas centrales de la robótica móvil. Además, son considerados problemas complejos, ya que para la localización el robot requiere un mapa consistente y para obtener un mapa el robot requiere una buena estimación de su localización. Esta dependencia mutua entre la posición y la estimación del mapa lo convierte en un problema difícil, que requiere investigación para su solución en un espacio desconocido de grandes dimensiones, debido a que es esencial para todas estas tareas.

Para el caso particular de los entornos de desastre, las técnicas de mapeo y localización utilizadas convencionalmente presentan deficiencias, ya que es necesaria una odometría precisa lo cual no siempre es posible, debido a la irregularidad del terreno y el continuo derrape de las ruedas. Por lo tanto, se buscará una solución de mapeo y localización visual combinado 2D y 3D, que obtenga la odometría a partir del análisis de datos tomados por un sensor de profundidad. El desarrollo del mapa será primordialmente en 2D, teniendo la posibilidad de procesar datos 3D en caso de que la geometría del terreno sea especialmente difícil como en espacios irregulares o reducidos.

En esta tesis se toman en cuenta todas estas características para la programación de un sistema de mapeo y localización que pueda montarse en el robot de búsqueda en entornos de desastre llamado FinDER (acrónimo de las palabras en inglés Finder in Disaster Environment Robot) creado en el taller ORG de la Facultad de Ingeniería de la UNAM, y que le haga posible navegar en ambientes irregulares, crear un mapa confiable de su entorno y poder localizarse en él.

14 HIPÓTESIS

Los ambientes de desastre requieren de técnicas especializadas de sensado y procesamiento para transformar la información del ambiente en mapas útiles y en datos que ayuden a una navegación eficiente. Las técnicas usadas actualmente no alcanzan dichos objetivos; sin embargo, debido a la ardua investigación que se está realizando en este ramo, se han desarrollado una gran variedad de técnicas que podrían resolver parcialmente el problema y que, combinadas con nuevas herramientas de sensado o métodos matemáticos más adecuados, darían como resultado una solución al problema que sea útil en situaciones reales y no sólo simulados.

SÍNTESIS DE CAPÍTULOS

En el primer capítulo se presento el contexto y la justificación del presente trabajo. También se abrió el panorama sobre los trabajos de investigación más recientes en el tema y se planteó una hipótesis propia que dará un ángulo para abordar el problema.

El segundo capítulo presenta el análisis detallado del problema y sus implicaciones y se plantea la metodología a seguir en el desarrollo de la tesis.

En los capítulos tres y cuatro se plantean las posibles soluciones al problema y se elige una de ellas siguiendo la metodología ya expuesta.

El capítulo cinco muestra el proceso de construcción y plasma los resultados obtenidos.

En los capítulos seis y siete se hace un análisis de dichos resultados y se concluye dando una postura final sobre la hipótesis planteada y el producto final de este trabajo.

Finalmente en los capítulos nueve y ocho se dan las referencias correspondientes a la información plasmada en este trabajo y se amplía la información sobre algunos procedimientos usados en el desarrollo.

2

ANÁLISIS DEL PROBLEMA

2.1 DEFINICIÓN DEL SISTEMA

Como se expuso en el capítulo anterior, los robots pueden ser de mucha ayuda para los equipos de búsqueda y rescate en los entornos de desastre, sobre todo en la exploración de terrenos peligrosos y de difícil acceso. Sin embargo, debido a las características propias de una situación de desastre, generalmente no se tiene un mapa del lugar que se pretende investigar, además las construcciones en las que se trabaja pueden haber quedado completamente desfiguradas.

En dichas situaciones en que no se cuenta con un mapa ni una idea clara del lugar a donde se desea ir, resulta imprescindible que el robot posea la capacidad de reaccionar adecuadamente a estímulos externos. Este comportamiento reactivo debe permitirle usar las condiciones de su ambiente para tomar las decisiones correctas sobre su actuar inmediato.

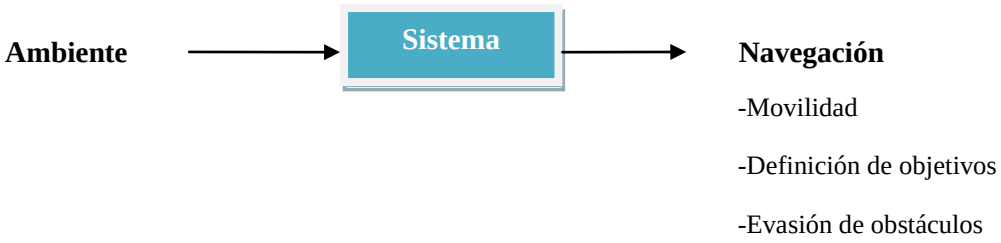


Figura 2.1 Sistema básico de navegación.

Sin embargo, este sistema permite una planeación muy pobre, ayudando poco a los equipos de rescate que requieren de un actuar más efectivo.

Por estas razones es necesario que el robot pueda crear, en tiempo real y a medida de que avanza, un mapa del lugar donde debe operar. Además, para poder navegar de forma correcta, es de vital importancia que pueda localizarse en dicho mapa. La resolución de estos dos problemas es de suma importancia para que el robot pueda moverse adecuadamente y de forma esperada, dentro de los edificios y entre los escombros en la búsqueda de sobrevivientes.

Así, se puede considerar que como resultado de la aplicación del sistema propuesto, se pretende obtener un mapa del lugar, el cual pueda ser utilizado por el robot y por los equipos de rescate, y la localización exacta del robot en el mapa creado.

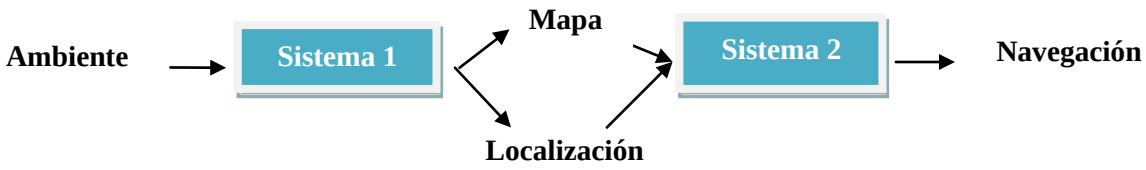


Figura 2.2 Sistema de navegación en dos pasos.

Este problema podría simplemente dividirse en dos etapas de procesamiento. En una se tomarían los datos para dibujar el mapa del lugar y en la otra se calcularía la posición del robot respectiva al mapa.

Sin embargo, para obtener una buena localización del robot, se requiere tener un mapa consistente, que brinde información confiable del ambiente en donde se encuentra, y para poder obtener un mapa suficientemente veraz, es necesaria una buena estimación de la localización del robot que lo realiza. Por lo tato el problema se encamina a la búsqueda de un método que combine estas dos operaciones de forma que una realimente a la otra.

Para obtener las salidas deseadas del sistema, se requiere hacer uso de entradas que proporcionen información suficiente, la cual en este caso una vez procesada, dé como resultado el dibujo de un mapa y la posición del robot en él.

LOCALIZACIÓN

Se debe conocer al menos el punto de partida del robot, el cual se puede tomar como el origen del sistema de referencia y a partir de ahí sensar y medir cada movimiento que realice, ya sea lineal o rotacional. La suma de los componentes de todos los movimientos en cada eje coordenado dará como resultado la posición actual del robot. Para obtener estos datos del ambiente, se requiere un sensor que permita medir la distancia recorrida de un sistema móvil. Además dicho sensor debe estar colocado dentro del sistema y moverse con él.

Estas dos salidas tienen que ver con medición de distancias. En particular para la obtención del mapa, se necesitará hacer uso de la telemetría, para conocer la distancia a la que se encuentran los objetos lejanos. Para el caso de la obtención de la posición, se pide la información del robot referente a los ejes coordenados que definen el entorno del cual se está construyendo el mapa. Será necesario encontrar elementos que brinden dicha información.

MAPEO

Al igual que para el cálculo de la posición, el mapa puede obtenerse de la telemetría, es decir, de la medición de la distancia de los objetos que se encuentran alrededor del robot. Dichos objetos son detectados por medio de los sensores, su presencia y distancia se convierten en los datos que irán formando el mapa global, pero al mismo tiempo determinan la porción del mapa situada justo frente al robot, es decir, es el mismo mapa global pero con el origen coincidente con el origen del sistema coordenado determinado por los sensores. Al tener las coordenadas de esa porción de un único mapa en dos sistemas de referencia diferentes, se puede calcular la posición relativa del sistema coordenado de los sensores.

Para obtener estos datos, existen diversos tipos de sensores que pueden brindar la información deseada, sin embargo, las condiciones especiales del problema requerirá de un sensor que pueda dar una gran cantidad de información en un intervalo de tiempo considerablemente pequeño.

2.2 CRITERIOS DE SELECCIÓN

Para resolver el problema en cuestión existen muchas soluciones posibles. Sin embargo, no todas convienen dadas las circunstancias en las que se pretende aplicar el sistema. Por lo tanto, es necesario establecer ciertos parámetros que ayuden a escoger una de las soluciones generadas para implementarla en el sistema propuesto.

Los parámetros a tomar en cuenta en la selección de la solución serán:

RAPIDEZ

Debido a las condiciones presentes en una situación de desastre, el tiempo se vuelve un factor de vital importancia: cada segundo puede influir en la supervivencia de las víctimas y en la preservación de las condiciones de estabilidad necesarias que hacen posible la entrada de los equipos de rescate.

La velocidad con que se pueda mover el agente móvil depende directamente de la rapidez con que se ejecuten los algoritmos de navegación. Un algoritmo que haga cálculos y dé resultados confiables con gran rapidez permitirá movimientos fluidos. En el caso que la velocidad no sea la adecuada, el robot no podrá moverse de forma rápida y constante o correrá el riesgo de perder información necesaria para los cálculos posteriores, lo que conducirá a interpretaciones erróneas del ambiente, provocando que pierda el conocimiento de su posición respecto del ambiente y que reaccione de forma incorrecta.

PRECISIÓN

La precisión es también un factor importante, sin ella no se podría confiar realmente en los cálculos realizados. Sin embargo, no será necesario ser tan estrictos, no se requieren mediciones exactas, puesto que la precisión puede mejorarse con cada medición, realizando cálculos probabilísticos.

COSTO COMPUTACIONAL

El robot en el que se montará el sistema propuesto, contará con muchos sistemas a parte del mencionado. La mayoría de ellos tendrán como objetivo principal hacer uso de los datos que da el ambiente para determinar la presencia y el estado físico de víctimas, entre ellos el sistema de visión artificial. Todos estos procesos son muy demandantes en cuestión de cálculos; es por eso que se deberá mantener al mínimo posible el costo computacional de este sistema.

COSTO MONETARIO

Al ser un robot que se encuentra en constante peligro y que está expuesto a ser dañado durante su funcionamiento, no resulta conveniente que sea un dispositivo extremadamente costoso. Además, conviene que sea suficientemente accesible como para que un equipo de rescatistas pueda costearlo.

FLEXIBILIDAD

La utilización de robots en situaciones de desastre es aún muy novedosa, dichos dispositivos se encuentran en constante evolución, por lo que resulta importante contar con una solución flexible que permita su implementación en plataformas móviles diferentes.

2.3 RESTRICCIONES

Si bien existen muchas herramientas que permiten desarrollar un sistema que cumpla con los requerimientos ya expresados, es necesario también tener en cuenta las restricciones naturales que se presentan debido a las condiciones de operación. El ambiente y circunstancias en las que se debe desempeñar el robot, conducen a establecer algunas limitaciones a tomar en cuenta en el proceso de diseño, las cuales se enlistan a continuación.

AMBIENTE CAÓTICO

El ambiente principal para el que está diseñado es un ambiente post-desastre, el cual presenta algunas características peculiares como las formas imprevisibles que pueden tener el piso y las paredes, por lo tanto, al momento de procesar las señales de los sensores se debe tener en cuenta que no representan geometrías definidas o comunes. Esta característica hace muy compleja la predicción del sensado.

TELECOMUNICACIÓN COMPROMETIDA

Es un problema recurrente que dentro de edificaciones con paredes gruesas la señal de las redes de telecomunicación pueden empezar a fallar. Además, las líneas eléctricas dañadas y las señales de radio utilizadas por los equipos de rescate generan mucha interferencia, por lo que es de esperarse que en ambientes de desastre no se cuente con la ayuda de sistemas como el GPS ni la internet.

DESLIZAMIENTO DE LAS RUEDAS

Una de las complicaciones que trae el no tener un suelo liso es la imposibilidad de tener una odometría exacta, el derrape constante de las ruedas lleva a buscar nuevas formas de obtener la medición del avance del robot.

BAJAS RESERVAS DE ENERGÍA

Para que el sistema pueda dar los resultados esperados y tenerla oportunidad de recorrer la mayor área posible, es necesario reducir el consumo energético al mínimo. El sistema a diseñar tiene implicaciones directas en este rubro, ya que mientras más cálculos se hagan en la computadora a bordo, mayor será el gasto energético.

PORTABILIDAD

Para lograr realizar los cálculos necesarios con mayor rapidez, con la utilización de todos los sensores posibles que den los datos necesarios para poder determinar el mapa y la localización con exactitud, podría ser necesario utilizar un sistema demasiado pesado, el cual sería difícil de transportar, sobre todo en ambientes como los que frecuentará el robot en cuestión. Por esto es pertinente reducir al máximo el tamaño de los sistemas del robot, para que éste pueda ser más ágil y tener mayor autonomía energética.

2.4 ESPECIFICACIÓNES

Todos los criterios mencionados en los puntos anteriores servirán de guía para la toma de decisiones en el proceso de diseño. Es por eso que deberá tenerse una idea clara, no solo cualitativa sino también cuantitativa de la forma en que estos delimitan el sistema. A continuación se presenta una tabla de especificaciones para cada parámetro:

Tabla 1 Especificaciones.

Parámetro	Especificación
Rapidez	El robot debe ser capaz de moverse junto con los equipos de recate por lo que es deseable que alcance una velocidad promedio de entre 3 y 5 k/h (0.83 - 1.38 m/s)
Precisión	Los errores de medición pueden disminuirse con mediciones sucesivas, sin embargo el cálculo de las distancias que formen el mapa deberá guardar un error menor a los 15 cm.
Costo computacional	El proceso de mapeo debe llevar un tiempo total de máximo 600 ms para ir a la par del movimiento del robot. Además el algoritmo debe poder correr de forma correcta en un procesador Intel Core i5 con 6 GB de memoria RAM.
Costo monetario	Se pretende usar sensores de gran calidad pero sin que el costo final del robot sobrepase los $200,000 M.N.
Flexibilidad	Actualmente se cuenta con tres versiones del robot de búsqueda, el sistema debe poder implementarse en las tres a pesar de sus diferencias físicas y la utilización de sensores diferentes.
Ambiente caótico	El sistema debe permitir al robot sortear obstáculos de hasta15 cm de alto y lograr inclinaciones de hasta 45°.
Telecomunicación comprometida	El sistema debe tener la capacidad hacer cálculos a bordo y en tiempo real para tomar decisiones en cuanto se pierda comunicación con el equipo de teleoperación.
Deslizamiento de las ruedas	El cálculo del movimiento del robot debe ser independiente del cálculo del movimiento de las ruedas.
Bajas reservas de energía	El robot debe tener una autonomía de al menos 30 minutos dando energía al movimiento, comunicación y procesamiento de datos.
Portabilidad	El peso total del robot no debe exceder los 50 kg para poder ser cargado por dos operadores.

2.5 METODOLOGÍA

La metodología a seguir en la presente tesis consistirá en el análisis de todas las posibles soluciones encontradas entre los métodos existentes para el mapeo y la localización de un robot en diferentes entornos. Cada solución será descompuesta para observar sus elementos básicos de forma individual y determinar su pertinencia en el problema a resolver. Posteriormente, se hará una síntesis o reconstrucción de todo lo subdividido en el análisis, formando así un nuevo método que adapte los mejores algoritmos a una técnica nueva que permita resolver el problema de la forma esperada.

3

DISEÑO

3.1 ANÁLISIS DE COMPONENTES PARA RESOLVER EL PROBLEMA

3.1.1 SENSORES EXISTENTES QUE PUEDEN SER DE AYUDA

Existen en el área de robótica muchas herramientas que pueden ser de ayuda para obtener y procesar los datos necesarios para resolver los problemas de mapeo y localización, según lo expuesto en el análisis del problema. Primeramente se analizarán algunos sensores que se consideran útiles en dicha tarea.

A) ENCODERS

Estos sensores miden la cantidad de giro de las ruedas; a partir de esta información y del radio de las ruedas motorizadas puede calcularse la distancia recorrida por el sistema, siempre y cuando no haya deslizamientos y el giro de las ruedas corresponda exactamente al avance lineal del robot.

Los encoders más comúnmente usados en las aplicaciones de robótica utilizan emisores infrarrojos y discos ranurados que se ajustan al eje. Estos pueden ser de dos tipos.

ENCODER INCREMENTAL

Este tipo de encoder se caracteriza porque determina su posición contando el número de impulsos que se generan cuando un rayo de luz infrarroja atraviesa las ranuras de un disco unido al eje.

En el estator hay como mínimo un foto-receptor infrarrojo que obtiene, a partir del giro del rotor, una señal cuadrada con la cual se puede determinar la posición y velocidad, haciendo el cálculo de movimiento relativo al punto de inicio definido al momento de arrancar.

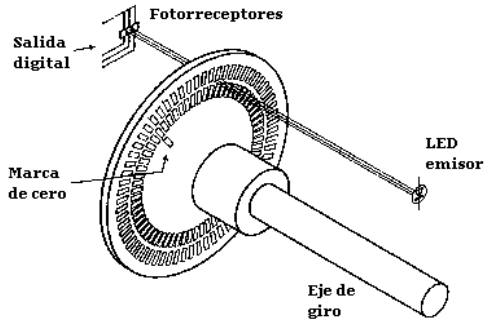


Figura 3.1 Encoder incremental.

ENCODER ABSOLUTO

El principio de funcionamiento de un encoder absoluto es muy similar al de un encoder incremental, sin embargo, presenta importantes diferencias desde el punto de vista funcional. Mientras en los encoders incrementales la posición está determinada por el cómputo del número de impulsos con respecto a la marca de cero, en los encoders absolutos la posición queda determinada mediante la lectura del código de salida, el cual es único para cada una de las posiciones dentro de la vuelta. Por consiguiente, los encoders absolutos no pierden la posición real cuando se corta la alimentación, hasta un nuevo encendido la posición está actualizada y disponible sin tener que efectuar la búsqueda del punto cero.

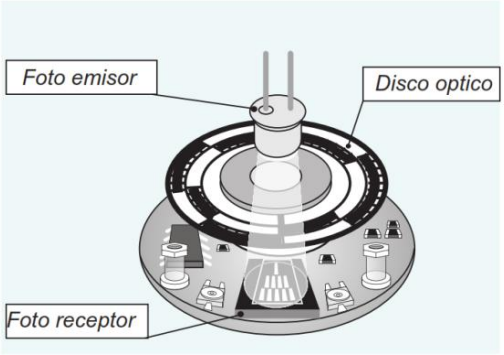


Figura 3.2 Encoder absoluto.

ENCODER DE EFECTO HALL

También existen encoders absolutos del tipo magnético, que usan la posición de los polos magnéticos para establecer la posición del encoder con un sensor de efecto Hall. El sensor es de tipo analógico y produce un código único que puede ser traducido al ángulo del eje

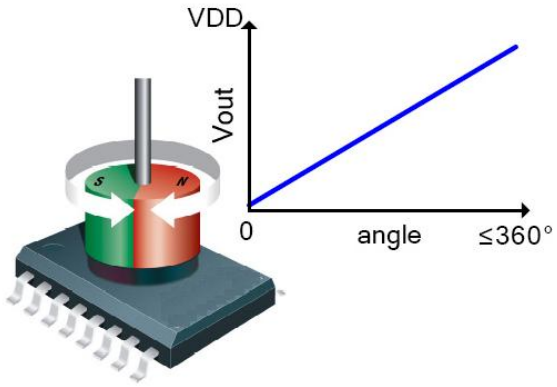


Figura 3.3 Encoder de efecto Hall.

B) IMU

La Unidad de Mediciones Inerciales o IMU, por sus siglas en inglés, es un sensor que permite medir las aceleraciones en los tres ejes de movimiento del robot, mediante las cuales es posible obtener también las velocidades y las distancias recorridas a cada instante. También brinda información de la orientación del dispositivo en cada momento, lo cual permite calcular de forma correcta la dirección del movimiento detectado.

C) CÁMARA

El uso de una cámara permite la medición de distancias a partir de la comparación de pixeles entre toma y toma; la detección de una diferencia entre la posición de ciertos puntos de interés correspondientes a objetos fijos, dan la capacidad de medir el movimiento del robot móvil.

D) SENSORES INFRARROJOS

Son dispositivos capaces de medir la radiación electromagnética infrarroja de los cuerpos en su campo de visión. Generalmente no todos los cuerpos reflejan la radiación necesaria para obtener una medición útil. Es por eso que suele usarse en combinación con un emisor infrarrojo, convirtiéndolo en un sensor activo que emite luz infrarroja, la cual rebota en los objetos que se encuentran en frente. La intensidad con que esta señal llega de regreso al sensor, es utilizada como información que permite conocer la distancia a dichos objetos. Sin embargo, estos sensores generalmente pueden medir distancias hasta de 1.5 m, en condiciones de poca luz solar.

E) SONAR

Es un sensor que utiliza la propagación del sonido para detectar objetos a distancia. Utiliza un generador de ultrasonido, el cual emite sonidos a frecuencias muy altas, incapaces de ser detectadas por el oído humano pero que pueden ser captadas por este sensor después de rebotar en los objetos, permitiendo conocer a qué distancia se encuentran.

F) CÁMARAS ESTÉREO

Esta técnica se basa en la utilización de dos cámaras ubicadas sobre un mismo plano de visión de tal manera que permitan triangular las imágenes obtenidas, lo que permite una medición aproximada de la distancia a la que se encuentran los objetos.

G) TELÉMETRO LÁSER

Al igual que los sistemas anteriores, permite medir distancias de los objetos que se encuentran alrededor, sólo que este sistema, al usar láser, resulta ser mucho más preciso. El rayo láser es un haz de luz tan concentrado que permite detectar variaciones mínimas en las distancias.

Este sensor puede obtener mediciones con una frecuencia de hasta 40 Hz, con un alcance máximo de entre5.6 m y 30 m, un ángulo de visión entre 120º y 270º, una resolución de 0.25º y una precisión de entre ±30 y ±50mm.

H) KINECT

Kinect es un dispositivo que, conectado a la consola de videojuegos de Microsoft, reconoce los movimientos del cuerpo y la voz para realizar comandos sin necesidad de tener controles. Sin embargo, es también un sensor de profundidad basado en un método estructurado de luz infrarroja. También cuenta con una cámara (lo que lo hace un dispositivo RGB-D, o sea, que puede obtener una imagen en RGB con profundidad), un micrófono y un pivote motorizado. Su uso no está limitado al juego con la consola Xbox360, sino que se puede conectar a una computadora y usarla como cualquier otro sensor. Hay muchos controladores y marcos de referencia de código abierto que se pueden usar.

Desde su lanzamiento en noviembre del 2010, ha ganado mucha popularidad, especialmente entre la comunidad científica. Muchos investigadores se han hecho de uno debido a que ha probado ser una solución poderosa en proyectos de sensado de profundidad; además su bajo costo (alrededor de $120 USD) lo hace un sensor mucho más accesible que un telémetro láser, el cual puede costar hasta 30 veces más. Investigaciones recientes se enfocan en el mapeo de entornos en tiempo real, reconocimiento de objetos y rastreo y localización. Se han obtenido resultados impresionantes hasta el momento. El proyecto KinectFusion es una prueba de ello. Este proyecto investiga técnicas para rastrear la posición de cámaras como el Kinect en un entorno con 6 grados de libertad, además de que hace una reconstrucción en tercera dimensión de las superficies a su alrededor para interactuar con ellas.

Este sensor tiene un ángulo de visión vertical de 43º y horizontal de 57º, sus motores internos le permiten una movilidad en el ángulo del pitch de ±27º, puede obtener mediciones de hasta 30 Hz y los micrófonos internos pueden obtener sonidos y digitalizarlos con un convertidor analógico a digital con una resolución de 24bit.

También se han encontrado algunas herramientas computacionales que pueden hacerle frente a la resolución del problema en cuestión.

3.1.2 ROS

Existen diferentes plataformas de software para robots, entre las cuales podemos mencionar: ROS, URBI, OROCOS, Microsoft Robotic Studio, Evolution Robotics y algunos otros, cada una con características diferentes: sistema operativo base, software libre o comercial, módulos de reconocimiento, navegación, simulación incluidos. Uno del los sistemas de mayor auge en la actualidad es un meta–sistema operativo llamado ROS, por las siglas en inglés de Robot Operating System, el cual cuenta con una comunidad creciente y herramientas cada vez más especializadas en el manejo de robots.

ROS es una plataforma que permite desarrollar software para robots y que opera de manera parecida a un sistema operativo.

Este sistema provee servicios para intercomunicar procesos de manera transparente y servicios que permiten la comunicación y el control de dispositivos comúnmente utilizados en robots (tales como cámara de vídeo, lasers, microcontroladores). Además, contiene software que ha sido programado para realizar las tareas más comunes (SLAM, sintetizador de voz, comunicación con microcontroladores, comunicación con otros robots).

ROS se ejecuta sobre varias plataformas, pero su distribución oficial es la distribución de Linux Ubuntu, misma que se adoptó para las dos computadoras a bordo del robot.

La ventaja de ROS sobre otras plataformas de software robótico es que es código abierto y tiene una comunidad creciente que lo apoya y da mantenimiento, así como gran actividad respecto de módulos disponibles para manejadores de dispositivos y algoritmos de robótica móvil.

La arquitectura de mensajes basada en agentes en la que se inspira la plataforma ROS, permite que el software del robot opere como un solo sistema compuesto por diferentes módulos. La modularidad en el desarrollo del software permite, entre otras cosas, descomponer un problema complejo en la

programación de módulos simples, facilitando el desarrollo al permitir distribuir el trabajo entre los miembros del equipo. El diseño del software mediante la realización de módulos es más ágil y mejor organizado.

ROS permite ejecutar diferentes procesos, cada uno encargado de una tarea específica. Los procesos pueden comunicarse mediante una configuración generador–consumidor, o mediante una configuración cliente–servidor. Un ejemplo de un proceso que genere datos es un módulo que recibe datos de un sensor, por otra parte un proceso consumidor podría ser un módulo que recibe esos datos para realizar tareas de control. Un ejemplo de proceso servidor podría ser un módulo que se encargue de realizar una tarea computacional enorme, como el reconocimiento de objetos, y un proceso cliente sería aquél que haría uso de ese servicio para saber si el objeto visualizado es o no un objeto deseado.

Los procesos se comunican mediante mensajes y cada mensaje es de un tipo específico. ROS se encarga completamente del envío de los mensajes. Entonces, suponiendo que se tienen dos computadoras conectadas, si un proceso generador se encuentra en una computadora y un proceso consumidor en otra, ROS autonómicamente se encarga de que los mensajes se dirijan de un proceso a otro.

3.1.3 AGV

AGV se corresponden con las siglas en inglés de Automatic Guided Vehicle, o lo que es lo mismo, vehículo de guiado automático. Los sistemas de AGVs se originaron en 1953, cuando se pensó en hacer realidad el "sueño" de un camión remolque sin conductor. Este primer vehículo precisaba de cable, que enterrado en el suelo de la fábrica creaba un campo magnético que servía de guía al vehículo. Poco a poco se han ido incorporando al mundo industrial a la vez que han crecido en aplicaciones y sofisticación.

Los sistemas de AGVs, de manera simplificada, representan un vehículo que se mueve de manera automática, sin conductor. Los sistemas de AGVs están concebidos para la realización del transporte de materiales, especialmente en tareas repetitivas y con alta cadencia. Este sistema garantiza el transporte de materiales en una ruta predeterminada, de manera ininterrumpida y sin la intervención directa del hombre.

A continuación se describen algunos de los componentes principales de estos sistemas.

HOST

Se encarga de la generación de órdenes, con base en la comunicación establecida con los datos capturados en planta y con el sistema de gestión del cliente, ERP o WMS.

GESTOR DE ÓRDENES

Recibe las órdenes generadas, las trata y reordena buscando la optimización del sistema y respetando las prioridades del cliente.

CONTROL DE TRÁFICO

Asigna las órdenes a cada AGV del sistema y vigila su correcto cumplimiento.

PLATAFORMA MÓVIL

Son los encargados de ejecutar las órdenes y de realizar el movimiento físico de la mercancía.

SISTEMAS DE GUIADO

Hay múltiples sistemas de guiado, convirtiendo a cada uno de ellos en idóneo para cada entorno concreto:

Sistema de guiado láser por reflectores.

Sistema de guiado láser por contorno

Sistema de guiado por puntos magnéticos

Sistema combinado (láser + puntos magnéticos)

Sistema combinado (láser + láser de contorno)

Sistema de guiado por banda magnética

Sistema combinado (láser + banda magnética)

Sistema combinado (láser + filoguiado)

Sistema de guiado óptico

Sistema de guiado por visión artificial

Sistema de guiado filoguiado

FILOGUIADO

Su utilización es amplísima en almacenes, secciones de montaje, talleres de fabricación flexible, integración con robot de ensamblaje, transporte a distancia sin conductor, etc.

El carro filoguiado se desplaza guiándose por un hilo conductor instalado bajo el suelo. Se compone de un robusto bastidor sobre el cual están situadas, en la parte inferior la rueda directriz y de contraste, y en su interior están alojados todos los elementos de mando y de control, así como la batería de alimentación con sus correspondientes equipos de potencia.

El vehículo posee un sensor que detecta el campo magnético creado por el cable, por lo que puede así seguir su recorrido. Los trazados o rutas pueden tener diversas ramas que permiten el trazado de múltiples rutas interestacionales. Al llegar un vehículo a una bifurcación se puede tomar una decisión mediante

frecuencias diferentes en las ramas, donde el vehículo filtraría sólo la del camino correcto, o bien mediante un interruptor que inhibe todas las ramas excepto la que debe tomar.

En la parte frontal se dispondrán las unidades de control, fácilmente accesibles mediante la abertura del panel abisagrado que soporta los elementos de mando del vehículo mismo.

3.2 ANÁLISIS DE HERRAMIENTAS COMPUTACIONALES PARA PROCESAR LOS DATOS

3.2.1 SLAM

El creciente desarrollo de métodos de navegación para robots móviles ha permitido abordar el tema no sólo desde diferentes enfoques independientemente, sino con técnicas que pueden combinar estos enfoques para lograr un mejor desempeño. Así, mediante información redundante, el sistema puede converger a soluciones más confiables y útiles. El método llamado SLAM es una técnica en auge que combina las tareas de mapeo y localización en un solo proceso.

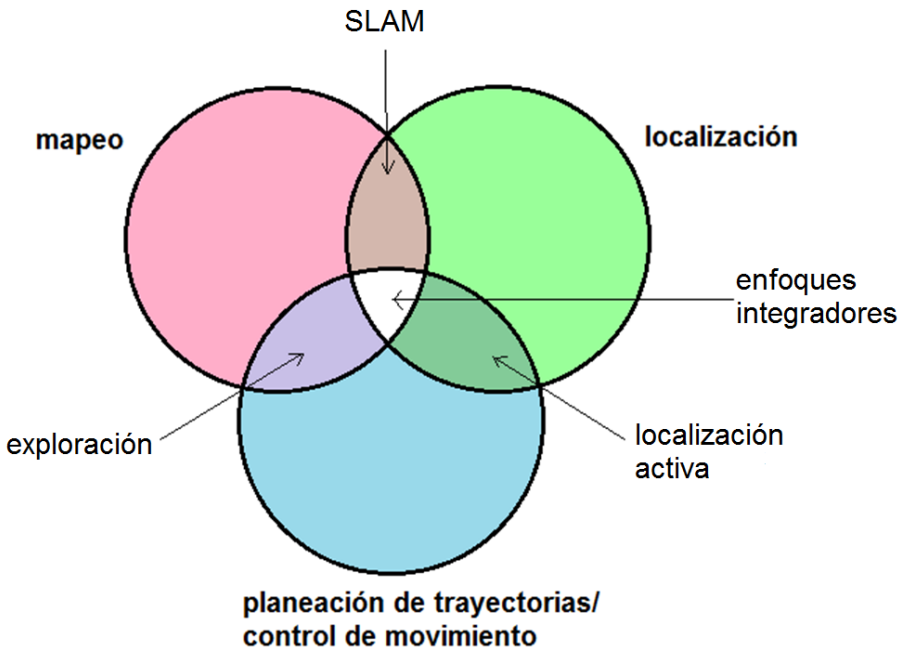


Figura 3.4 Diagrama de mapeo, localización y planeación de trayectorias.

¿QUÉ ES EL SLAM?

Una técnica que últimamente ha estado en auge y se utiliza para resolver este problema tan recurrente y complejo de caracterización y ubicación inmediata dentro de un espacio es el SLAM (Simultaneous Localización And Mapping), que significa localización y mapeo simultáneos.

El SLAM se modela como un problema de estimación de estados secuencial. Mientras el robot se mueve por el ambiente, se toman observaciones por medio del sensor y se integran a la estimación del estado usando una aproximación probabilística. El enfoque usado consiste en un filtro de partículas combinado con un pareo de escaneos (scan matching) y un filtro de Kalman extendido para el cálculo y estimación con supresión de ruido de la posición y la odometría.

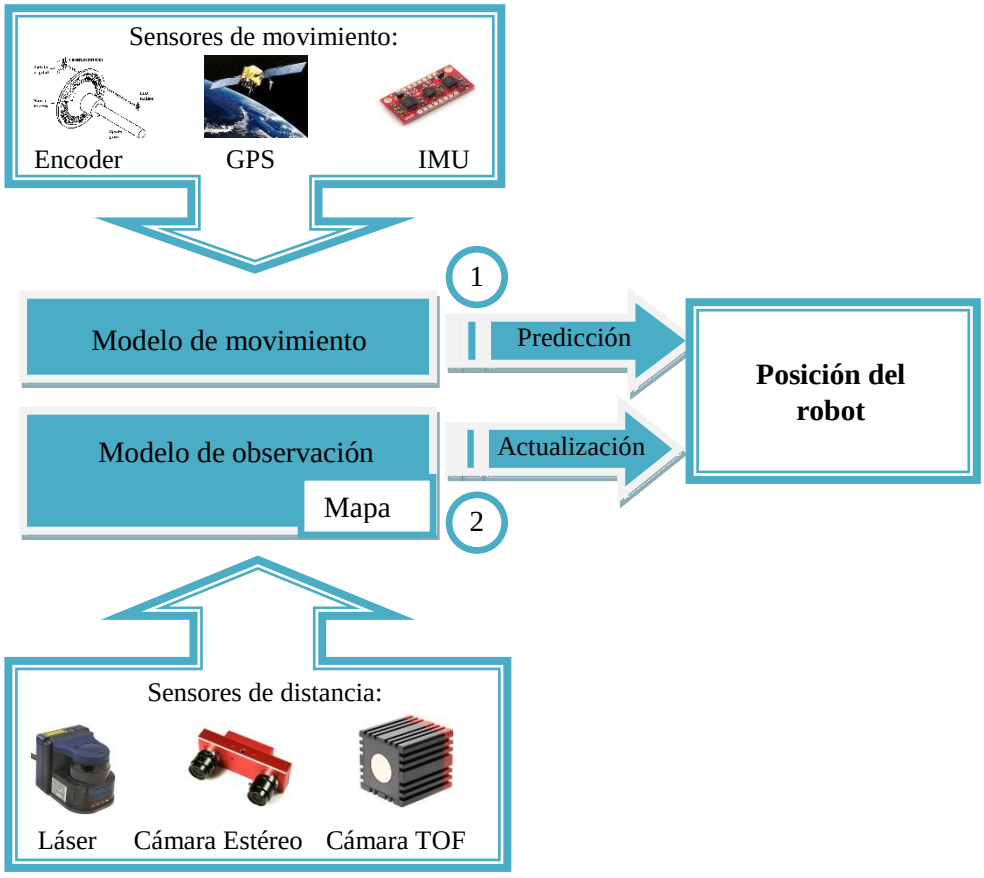


Figura 3.4 Algoritmo de SLAM. (1) El modelo de movimiento determina la nueva posición del robot basado en los sensores de movimiento. (2) El modelo de observación corrige la posición predicha del robot basado en la observación de características o celdas de la malla.

Los algoritmos de SLAM pueden ser clasificados de manera somera por medio de la técnica de estimación de posición y el tipo de representación del mapa. Ambos aspectos se consideran a detalle en la siguiente sección. Como se mencionó anteriormente, un sistema de SLAM típico consiste de las etapas de predicción y actualización.

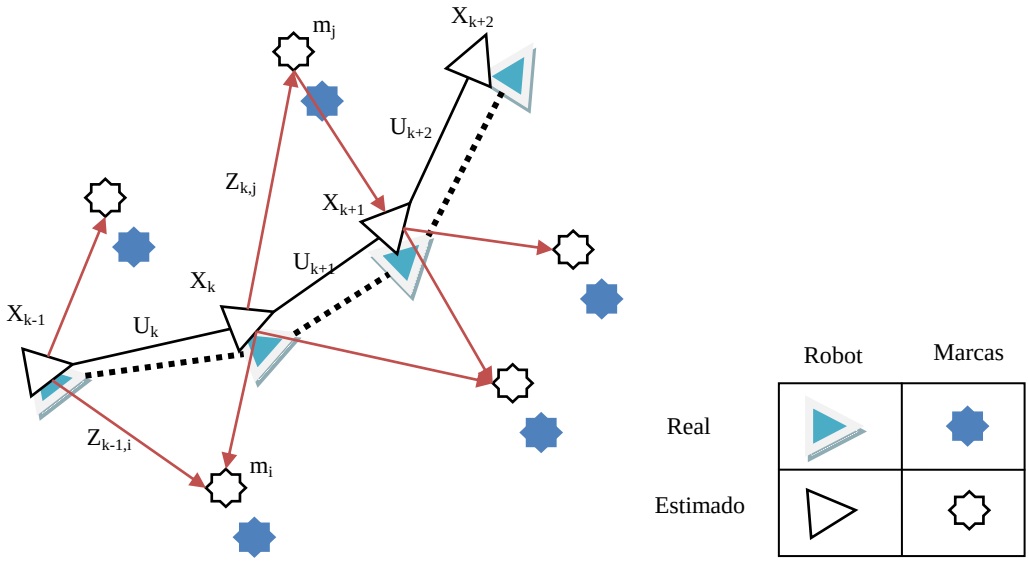


Figura 3.5 El problema esencial del SLAM. La trayectoria del robot y las marcas del ambiente son estimadas de manera simultánea. La ruta verdadera, x_k, (línea sólida) y las marcas verdaderas del ambiente, m_i, (estrellas obscuras) son desconocidas. Los comandos de control, u_k, son usados para una predicción inicial y las observaciones, z_{kj}, son necesarias para el proceso de estimación.

Un algoritmo de SLAM estima la trayectoria verdadera (línea sólida) con observaciones (estrellas obscuras) del ambiente. El paso de predicción permite una estimación inicial de la posición del vehículo, denotado por la línea discontinua gruesa con puntos, y más adelante es corregida en el paso de actualización. Debe notarse que el mapa y la trayectoria son estimados simultáneamente. Éste es el reto más grade y el que marca a esta técnica como un problema difícil.

Las partes del mapa pueden ser nuevas observaciones o puntos re-observados. Las últimas son usadas para el proceso de corrección de la posición del vehículo. Nuevas partes son añadidas al mapa e inicializadas con una incertidumbre por defecto.

A continuación se presenta una categorización basada en la técnica de estimación de la posición y la representación del mapa. Se discute la diferencia entre el filtro de Kalman y el filtro de partículas.

Otra forma de categorizar los algoritmos de SLAM está basada en la representación del mapa.

En esta sección se resaltan las fortalezas y debilidades de los mapas por ocupación de cuadrícula y por características.

CLASIFICACIÓN DE LAS PROPUESTAS DE SLAM

ONLINE Y OFFLINE

Los enfoques online (en línea) procesan la información recibida durante la misión del robot, un estimado del estado actual está disponible en tiempo real. En enfoques offline (fuera de línea) primero se obtiene la información completa y luego se procesa, estimando la trayectoria completa del vehículo así como el mapa.

CARACTERÍSTICAS Y CUADRÍCULAS

El enfoque basado en características extrae características de los datos del sensor, como líneas de un escáner láser, esas se usan después para construir un mapa de características. El enfoque en cuadrículas se usa directamente, así se pueden trazar mapas de entornos con geometrías arbitrarias.

REPRESENTACIONES PARAMÉTRICAS Y NO PARAMÉTRICAS

En la representación paramétrica las distribuciones de probabilidad usan una representación funcional que tiene ventajas computacionales pero no puede modelar densidades arbitrarias. Con representaciones no paramétricas sí se pueden representar densidades arbitrarias, pero con un mayor costo computacional.

SLAM 3D

En situaciones de desastre se pueden encontrar escenarios desconocidos donde se complica la navegación, no sólo en dos dimensiones sino que los pisos rotos, muebles caídos y demás escombros representan un reto en tres dimensiones. Por lo tanto, se vuelve necesario un mapeo tridimensional del entorno que dé la información necesaria para la navegación del robot en sus 6 grados de libertad, detección de peligros y de víctimas.

MÉTODOS DE SLAM

La base estadística del problema del SLAM fue descrita primeramente por Smith et ál. y por Durrant–Whyte. Estos artículos clave proveen técnicas de estimación probabilística para correlacionar características en el ambiente y las posiciones del robot. Además, han sido los primeros en proponer métodos para refinar las incertidumbres de las observaciones continuas.

Una clave encontrada en su trabajo es que debe haber una alta correlación entre características de mapas diferentes y que estas correlaciones crecen con las observaciones sucesivas.

Crowley y Leonard et ál. proponen métodos de SLAM usando segmentos de líneas extraídos de datos ultrasónicos. Otros investigadores como Vadorpe et ál. y González et ál. usan datos de sensores láser en su lugar. Al mismo tiempo Ayache y Faungeras desarrollaron el primer trabajo sobre navegación y mapeo visual.

Una solución consistente al problema del SLAM es un vector de estados que relacione la posición del robot con las características del terreno. Este enorme vector de estados necesita ser actualizado después de cada observación, lo que conlleva un inmenso esfuerzo computacional.

Leonard et ál. intentaron reducir el costo computacional dividiendo el vector de estados en sub partes locales. Este concepto fue desechado más tarde, porque en el año de 1995 un parte–aguas ocurrió en el momento en que se dieron cuenta de que el problema del SLAM es convergente y el enorme vector de estados es esencial. Mientras más crezcan las correlaciones entre las características , la solución resulta mejor.

Hasta aquí, la posición del robot fue descrita por una coordenada y cierta certeza respecto a los puntos de referencia. Murphy introduce los filtros de partículas, que son en general representaciones discretas de funciones de densidad de probabilidades. Con los filtros de partículas la posición del robot es representada por un conjunto discreto de estados. En contraste con los mapas basados en características usados previamente, Murphy propuso un mapa por ocupación de cuadrícula, que es una cuantización del mundo en bloques.

Después, Montemerlo et ál. dio una extensión a los mapas basados en características (FastSLAM). El mapa es estimado con un filtro de Kalman, mientras la posición del robot es representada por un filtro de partículas. Montemerlo demostró que unas cuantas características son suficientes para navegar en un mapa con tamaño de varios kilómetros cuadrados.

SLAM 2D Y POSICIONAMIENTO 3D

Esta técnica permite una percepción del ambiente y una autolocalización suficientemente confiables manteniendo la exigencia de recursos computacionales muy baja. Puede ser usada para obtener un SLAM en escenarios pequeños, donde no es necesario cerrar los bucles y las rápidas lecturas del sensor láser son necesarias. Es posible también una estimación de los movimientos del sistema en sus 6 grados de libertad, logrando así una navegación más ágil.

Esto es posible al combinar un sistema de SLAM 2D basado en la integración de los datos de un láser en un mapa plano con un sistema de navegación en 3D basado en sistemas de medición inercial (IMU), que incorpora la información en 2D del subsistema de SLAM para conseguir una mejor aproximación de su posición

El sistema debe calcular los 6 grados de libertad que presenta en su ambiente, dados por la traslación y rotación de la plataforma. Para conseguir esto, el sistema se forma de dos componentes principales. Un filtro de navegación fusiona la información de las mediciones inerciales y otros sensores disponibles para obtener una solución consistente en 3D, mientras que un sistema de SLAM en 2D es usado para proveer la posición y otra información importante en el plano del suelo. Ambas estimaciones se actualizan individualmente.

Se define el sistema coordenado de navegación teniendo el origen en el punto de partida con el eje z apuntando hacia arriba, y el eje x apuntando en dirección del yaw del escenario en el momento de inicio.

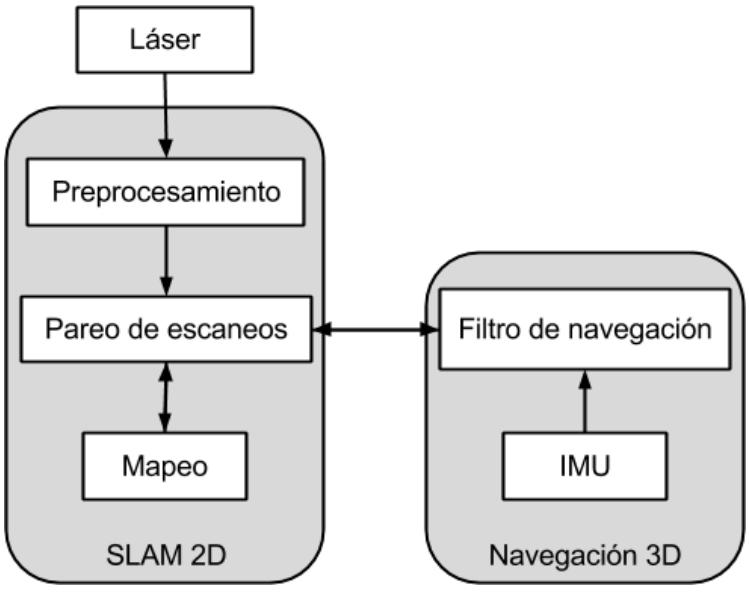


Figura 3.6 Diagrama de sistema de SLAM 2D con navegación 3D.

El sistema de estados en 3D está representado por: $x = (\Omega^{\mathrm{T}} p^T v^T)^T$, donde $\Omega = (\varphi, \theta, \psi)^T$ son los ángulos de Euler correspondientes al roll, pitch y yaw, y $p = (p_x,\ p_y,\ p_z)^T$ y $v = (v_x,\ v_y,\ v_z)^T$ son las posiciones y velocidades de la plataforma expresados en el marco de referencia de la navegación.

Las mediciones de inercia son usadas como un vector de entrada $u = (\omega^T a^T)^T$ con las velocidades angulares en la base del móvil $\omega = (\omega_x,\ \omega_y,\ \omega_z)^T$ y las aceleraciones angulares $\dot{\omega} = (\dot{\omega}_x,\ \dot{\omega}_y,\ \dot{\omega}_z)^T$. El movimiento de un cuerpo rígido arbitrario está descrito por el sistema no lineal de ecuaciones diferenciales

$$\Omega = (\dot{\mathrm{E}}_\Omega \cdot \omega) \tag{1}$$

$$\dot{p} = v \tag{2}$$

$$\dot{v} = R_\Omega \cdot a + g \tag{3}$$

Donde R_Ω es la matriz coseno de direcciones que mapea un vector de la base del cuerpo a la base de navegación, E_Ω mapea las velocidades angulares de la base del cuerpo a las derivadas de los ángulos de Euler y g es el vector constante de la aceleración de lagravedad. Los efectos de esta aceleración pueden ser omitidos cuando se usan sensores de bajo costo.

MÉTODOS DE ODOMETRÍA

La posición del robot respecto a un sistema de referencia inicial es uno de los parámetros más importantes de los que debe disponer un robot móvil. Se conoce como odometría a las técnicas de posicionamiento que emplean información de sensores propioceptivos (aquéllos que adquieren datos del propio sistema), para obtener una aproximación de la posición real a la que se encuentra un sistema móvil, en un determinado instante, respecto a un sistema de referencia inicial.

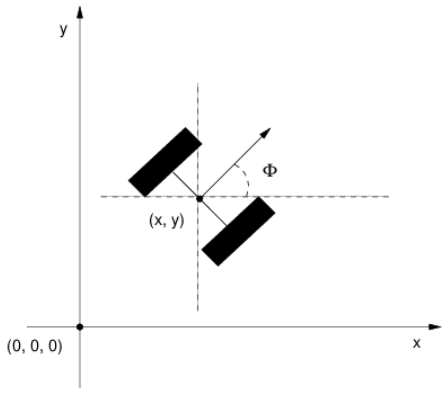


Figura 3.7 Sistemas de referencia para medición de odometría.

En general, como se puede apreciar en la figura 3.7, son suficientes tres parámetros (x, y, θ)para conocer la posición de un sistema móvil: posición respecto al eje x, respecto al eje y y ángulo respecto al eje x.

ODOMETRÍA CON ENCODERS

Un sistema que realice odometría con encoders es capaz de conocer su posición actual a partir de la distancia recorrida por cada una de sus ruedas. Para ello, accede regularmente a las lecturas de los encoders para determinar cuál ha sido el desplazamiento lineal de cada una de las ruedas. Posteriormente, mediante el empleo de una aproximación trigonométrica y conociendo la posición anterior, puede calcular una aproximación a la nueva posición actual real.

Para que estas mediciones sean precisas, es necesario que el sistema móvil se desplace sobre un terreno liso y uniforme para que la medición del giro de las ruedas concuerde con el avance total del sistema. Si la superficie cuenta con protuberancias, el giro de las ruedas moverá el sistema en dirección perpendicular a la superficie. Además, puede haber ocasiones en que las ruedas pierdan contacto con el suelo, esto causará que las ruedas giren sin causar movimiento real en el sistema y por lo tanto se obtendrá un dato erróneo de la posición del robot.

ODOMETRÍA VISUAL

Al igual que la odometría por encoders, la odometría visual se encarga de obtener los datos de la posición de un sistema móvil. La diferencia radica en los sensores que utiliza para obtener la información del ambiente. En este caso en vez de medir el giro de las ruedas, se utiliza una cámara para obtener imágenes a medida que el robot avanza. Cada imagen es procesada a cada paso para obtener características que la destaquen, al comparase con la imagen anterior podrán notarse muchas características similares pero en posiciones diferentes. De esta manera, y teniendo en cuenta que el ambiente es estático, es posible medir indirectamente el movimiento del robot.

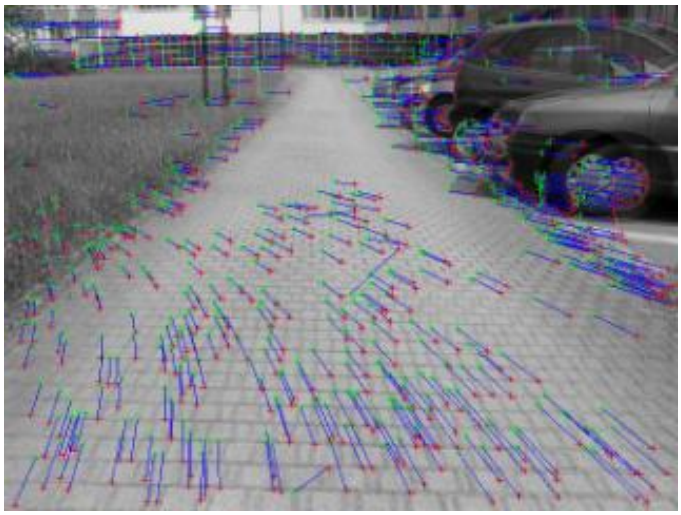


Figura 3.8 Pareo de puntos o point matching.

ESTIMACIÓN DE POSICIÓN

PROBABILIDAD POSTERIOR

En estadística Bayesiana, la probabilidad posterior de un evento aleatorio o una proposición incierta, es la probabilidad condicional que se asigna después de que la evidencia relevante es tomada en cuenta.

Definiéndolo más formalmente, la probabilidad posterior es la probabilidad de los parámetros θ dada una evidencia x : $P(\theta|x)$ .En contraste con la función de la probabilidad de la evidencia, dados los parámetros$P(x|\theta)$.

TÉCNICAS DE ESTIMACIÓN DE LA POSICIÓN

Desde los 90's, métodos probabilísticos como los filtros de Kalman, filtros de partículas y métodos de maximización de expectativas se han vuelto enfoques populares para la resolución del problema de SLAM.

Las principales diferencias entre los filtros de Kalman y los filtros de partículas se exponen a continuación.

Un filtro de Kalman es un filtro lineal recursivo. No se puede suponer linealidad en los movimientos típicos de los robots móviles, de tal forma que se ha desarrollado extensiones a esta técnica.

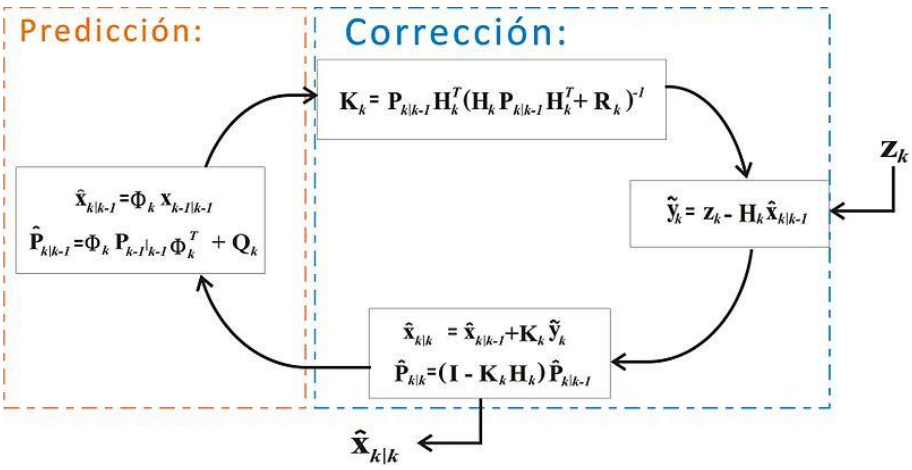


Figura 3.9 Proceso del filtro de Kalman.

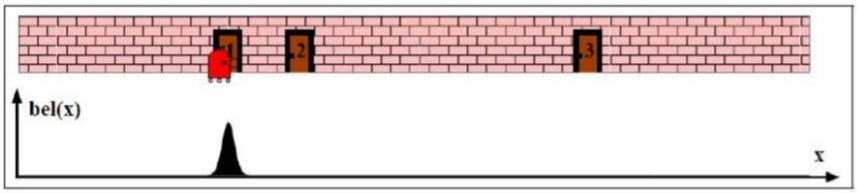
El filtro de Kalman se usa comúnmente para tareas lineales de pequeñas dimensiones y es frecuentemente la primera elección debido a su eficiencia computacional. Mapas más grandes con muchas características conllevan matrices de estados enormes y el filtro se vuelve impráctico. Transiciones de estado no lineales son posibles con el filtro extendido de Kalman,

Un filtro extendido de Kalman (EKF) usa una expansión por series de Taylor para manejar las no-linealidades. El filtro de Kalman comprimido (CKF) es similar al extendido pero reduce la complejidad computacional actualizando solamente un subconjunto local de las características del mapa. Existen muchas extensiones más que el lector puede consultar en el trabajo de Thrun et al., 2005. [3]

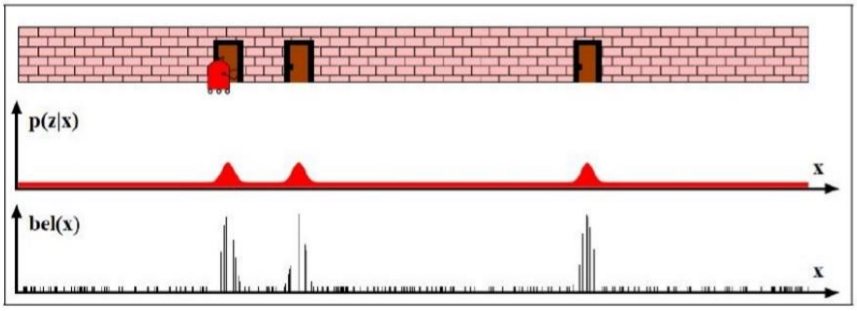
Algunas aplicaciones necesitan seguir la trayectoria de varios objetos simultáneamente o seguir múltiples hipótesis. Esto puede ser logrado por el filtro de partículas, el cual es más robusto ante asociaciones equivocadas debido a su procedimiento de remuestreo probabilístico.

Un filtro de partículas aproxima un filtro de Bayes con un conjunto de estados discretos. La probabilidad de una hipótesis discreta está dada por un peso que se asigna a cada estado. Nuevas observaciones provocan cambios en los pesos y un continuo remuestreo previene la degeneración de la estimación de la posición del robot.

Para ilustrar las diferencias entre un filtro de Kalman y un filtro de partículas se explica un ejemplo de localización. El mapa es conocido y la tarea es estimar la posición del robot. Primero, un robot está equipado con un detector específicamente de puertas y viaja por un corredor como se observa en la figura 3.10.



(a)



(b)

Figura 3.10 Representación del filtro de Kalman y el filtro de partículas en tareas de localización. (a) Un filtro de Kalman representa la posición de un robot con una función gaussiana de densidad unimodal. (b) Un filtro de partículas puede representar funciones de densidad arbitraria con estados discretos.

Las puertas están numeradas y el estimador sabe exactamente su localización en cada puerta detectada.

En el caso del filtro de Kalman, la distribución de la posición del robot tiene un pico alrededor de la localización de la puerta. Cuando no se hacen observaciones el pico cambia según la información de la odometría y la distribución de probabilidad de posición baja debido a la incertidumbre tan alta, de esta forma el robot recorre todo el pasillo detectando las tres puertas. Pero puede pasar que el robot no pueda distinguir entre cada puerta, en este caso el filtro de Kalman falla porque está promediando las tres observaciones. En contraste, un filtro de partículas puede representar funciones de densidades arbitrarias con un conjunto de pesos discretos.

En el primer caso, las puertas están numeradas y se conoce la posición exacta. Las partículas con los pesos más altos están localizadas en la posición de la puerta y la representación de la posición es una versión discreta de la representación paramétrica usada por el filtro de Kalman.

En el segundo caso, cuando los números de las puertas se desconocen, los pesos de las partículas son altas en cada una de las tres observaciones como se ve en la figura. Un filtro de partículas es capaz de calcular hipótesis múltiples simultáneamente y recuperar su localización si alguna vez la pierde.

En un caso con más dimensiones, un filtro de Kalman representa la posición de un robot por un vector de estados y una incertidumbre correspondiente generalmente marcada con elipses. El vector de estados discreto de un filtro de partículas se muestra en la figura 3.11.

Finalmente, el filtro de partículas de Rao–Blackwell usa un simple filtro de Kalman extendido para estimar las características del ambiente y un robusto filtro de partículas para modelar la posición del robot. Esto produce una eficiente implementación del SLAM que escala logarítmicamente con la cantidad de datos sensados.

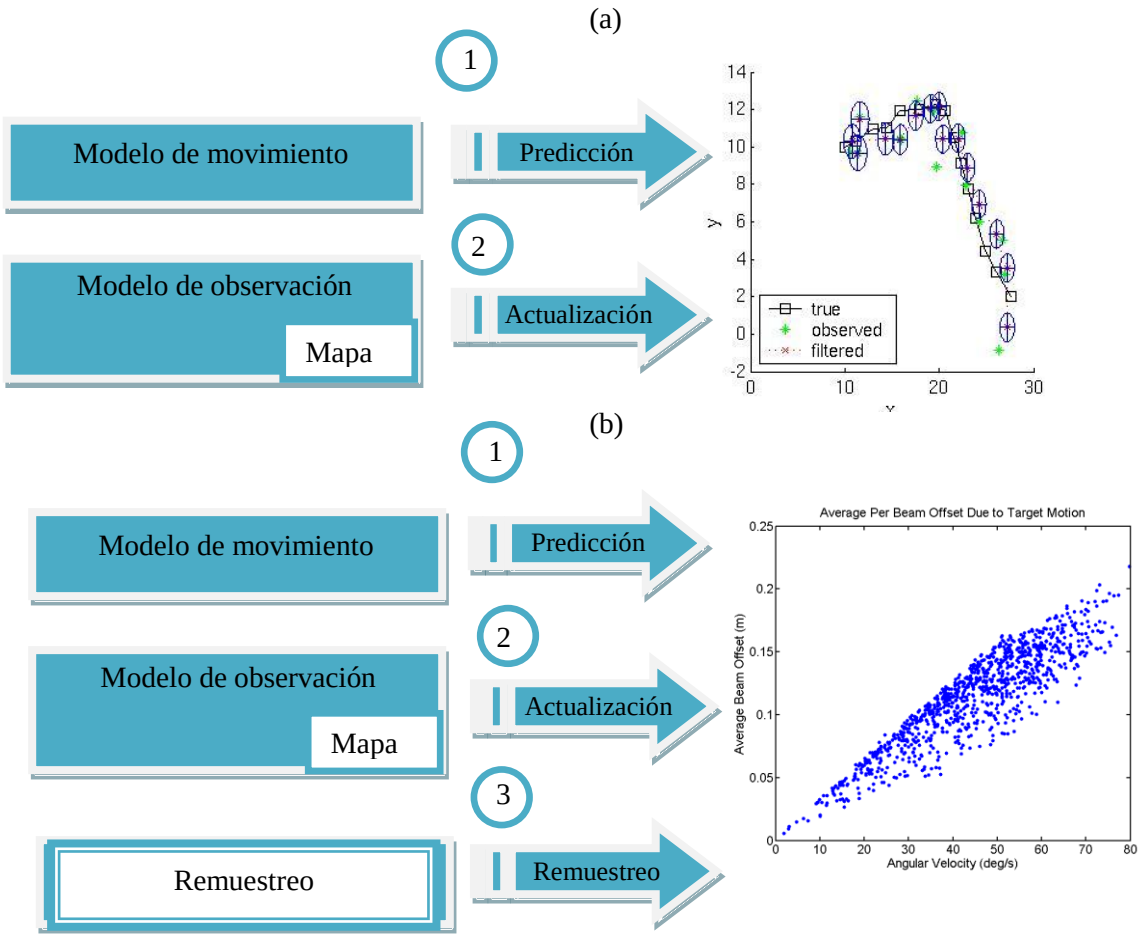


Figura 3.11 Procedimientos del filtro de Kalman y el filtro de partículas. (a) Un filtro de Kalman representa la posición del robot por un vector de estados y una incertidumbre correspondiente, generalmente denotada por elipses. (b) Un filtro de partículas representa la posición con una función de densidad arbitraria con estados discretos.

TÉCNICAS DE MAPEO

OCUPACIÓN DE CUADRÍCULA

Es una técnica basada en una serie de algoritmos computacionales probabilísticos propuesta para representar ambientes con formas arbitrarias y utilizando sensores con mucho ruido o con un alto grado de incertidumbre. Esta técnica asume que la posición del robot es conocida. La idea básica es representar el mapa del ambiente como un campo de valores aleatorios uniformemente espaciados, cada uno representando la presencia de un obstáculo en ese punto del ambiente. Los algoritmos de ocupación de cuadricula calculan aproximaciones de la probabilidad posterior para dichos valores aleatorios.

El objetivo de un algoritmo de mapeo por ocupación de cuadricula es estimar la probabilidad posterior de cada celda en el mapa, teniendo la información de los sensores: $P(m \mid z_{1:t}, x_{1:t})$, donde m es el mapa, $z_{1:t}$es el grupo de mediciones del ambiente del tiempo 1 al tiempo t y $x_{1;t}$es el grupo de posiciones del robot del tiempo 1 al tiempo t.

El mapa está representado como una malla fina sobre el espacio continuo de ubicaciones en el ambiente. Los tipos más comunes de mapas por ocupación de cuadricula son en 2D y describen una rebanada del mundo en 3D.

Siendo m_i la representación de la celda en el mapa con índice i, la notación $P(m_i)$ representa la probabilidad de que la celda m_i esté ocupada. El problema computacional de representar $P(m \mid z_{1:t}, x_{1:t})$es la complejidadde los cálculos. Si un mapa de dimensiones relativamente pequeñas contiene 10,000 celdas, el número de mapas que pueden ser representados es de 2^{100000}. Así, calcular la probabilidad posterior para todos los mapas se vuelve inviable.

El enfoque más común es partir el problema en problemas más pequeños estimando$P(m \mid z_{1:t}, x_{1:t})$para todas las celdas m_i. Cada uno de estos problemas de estimación se vuelve entonces un problema binario. Este acercamiento implica la imposibilidad de modelar dependencias entre celdas vecinas. En lugar de eso, puede obtener un mapa posterior aproximado por medio de factorización. Debido a esta factorización se puede utilizar un filtro de Bayes binario para estimar la probabilidad de ocupación para cada celda.

SLAM 2D

Para representar ambientes arbitrarios, se usa un mapa por ocupación de cuadrícula. Como los datos obtenidos del sensor láser pueden denotar movimientos en seis grados de libertad, el escaneo debe ser transformado a una base local estable usando la altitud aproximada del sensor láser. Tomando estas consideraciones, los datos obtenidos del sensor láser son convertidos en una nube de puntos determinados por la distancia de los objetos al sensor.

La naturaleza discreta de los cálculos en un mapa por ocupación de cuadrícula limita la precisión que se puede alcanzar y no permite un cálculo directo de valores interpolados o sus derivadas. Por lo tanto, se ocupa un algoritmo de interpolación que permite una precisión en rangos más pequeños que la escala de la cuadrícula, logrando la estimación de la probabilidad de ocupación de una celda y sus derivadas. De esta forma, las celdas de la cuadrícula del mapa pueden verse como muestras de una distribución de probabilidades continua subyacente.

Dada una coordenada de un mapa continuo P_m, el valor de ocupación $M(P_m)$ así como el gradiente $\nabla M(P_m) = (\frac{\delta M}{\delta x}(P_m), \frac{\delta M}{\delta y}(P_m))$ pueden calcularse aproximadamente usando las cuatro coordenadas más cercanas $P_{00..11}$. Entonces de la interpolación lineal sobre los ejes "x" y "y" resulta:

$$M(P_m) \approx \frac{y - y_0}{y_1 - y_0}\left(\frac{x - x_0}{x_1 - x_0}M(P_{11}) + \frac{x_1 - x}{x_1 - x_0}M(P_{01})\right) + \frac{y_1 - y}{y_1 - y_0}\left(\frac{x - x_0}{x_1 - x_0}M(P_{10}) + \frac{x_1 - x}{x_1 - x_0}M(P_{00})\right) \tag{4}$$

De las derivadas resulta:

$$\frac{\delta M}{\delta x}(P_m) \approx \frac{y - y_0}{y_1 - y_0}\big(M(P_{11}) + M(P_{01})\big) + \frac{y_1 - y}{y_1 - y_0}\big(M(P_{10}) + M(P_{00})\big) \tag{5}$$

$$\frac{\delta M}{\delta y}(P_m) \approx \frac{x - x_0}{x_1 - x_0}\big(M(P_{11}) + M(P_{01})\big) + \frac{x_1 - x}{x_1 - x_0}\big(M(P_{10}) + M(P_{00})\big) \tag{6}$$

Debe hacerse notar que todas las celdas del mapa están situadas en una cuadrícula regular con distancia de una unidad (en las coordenadas del mapa) de una a otra, lo que simplifica las ecuaciones para la aproximación del gradiente.

SLAM 3D

También es cierto que en los entornos de desastre los obstáculos se pueden encontrar en cualquier plano, no sólo a nivel de suelo. Esos obstáculos representan un gran reto para la movilidad del robot, ya que el sortearlos requiere de un sensado y un procesamiento más extensos.

Si bien por lo general esta técnica no es usada para la navegación de los robots exploradores por su enorme y a veces inalcanzable costo computacional, es verdad que en entornos de desastre se vuelve necesario y hasta indispensable, en casos que el ambiente se vuelve demasiado irregular. Es por eso que se han empezado a desarrollar muchas líneas de investigación en el tema que pretenden encontrar soluciones asequibles para los equipos de cómputo actuales.

Existen ya varios métodos para acercarse a la resolución de este problema. Uno de ellos es el propuesto por Keiji Nagatani et ál. donde el robot obtiene información del ambiente local y estima su posición en un mapa global parcialmente dibujado usando la técnica de correlación.

En la representación del ambiente se usa mucho el método poligonal en tercera dimensión o la representación por medio de mallas cúbicas, sin embargo, en dichos métodos la cantidad de información es tan grande que regularmente se necesita algún método de compresión de los datos para poder navegar en tiempo real. Por lo tanto, el método que se usa es S–DEM (Sphere Digital Elevation Map), el cual puede representar ambientes irregulares sobreponiendo varias cuadrículas con borde esférico, las cuales contienen la información de profundidad del ambiente.

El método se basa en usar características del ambiente para que el robot se pueda localizar después de cada movimiento. El algoritmo realiza los siguientes pasos:

1) Moverse una distancia considerable del punto de sensado anterior
2) Sensar datos del ambiente en tres dimensiones
3) Construir un mapa local
4) Comparar entre el mapa local y el mapa global reconstruido
5) Localizar la posición del robot en el mapa global
6) Añadir la información del mapa local al mapa global, de acuerdo a las estimación de la posición del robot
7) Regresar al paso 1.

Al usar el método S–DEM para mapear el ambiente, un sensor láser toma mediciones girando sobre su propio eje y guardando los datos en una reconstrucción esférica del lugar, donde cada punto sensado se representa por $r(\theta, \phi)$ y el centro de las esfera es el sensor.

Para representar el mapa completo se combinan varios mapas esféricos locales encontrando las posiciones relativas entre ellos. También la localización del robot se realiza detectando una posición relativa entre una de las esferas en el mapa global y la esfera en la posición actual del robot.

Para calcular la posición relativa entre dos esferas se usa una técnica de correlación con el siguiente método: primero, se transforma cada posición límite $r_{gl}(\theta, \phi)$ del mapa global en coordenadas cuadradas

(x, y, z); el segundo paso es mover el origen de la esfera global $(x_g,\ y_g, z_g)$ a una posición arbitraria $(x_v,\ y_v, z_v)$; luego se genera cada posición límite $r'_v(\theta'_v, \phi'_v)$ usando (x, y, z) para generar una esfera virtual con sucentro en $(x_v,\ y_v, z_v)$; finalmente, la correlación entre la esfera virtual y la esfera local se calcula con la siguiente ecuación:

$$d = \sum_{\theta} \sum_{\phi} (r'_v(\theta'_v, \phi'_v) - r_{lc}(\theta, \phi))^2 \tag{7}$$

Donde$r_{lc}(\theta, \phi)$ es la posición de los límites sensados en el mapa local.

Después, se obtiene la posición $(x_v,\ y_v, z_v)$ que minimiza el valor de d y que es relativo a la posición entre la esfera global y la esfera local.

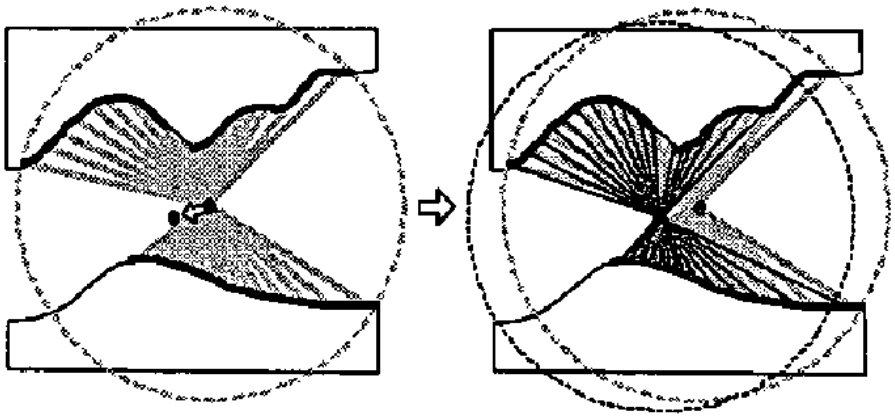


Figura 3.12 Ejemplo de S–DEMvirtual.

Una vez que el robot calcula los valores de correlación d de cada esfera virtual candidata con origen $(x_v,\ y_v, z_v)$, el robot determina su posición eligiendo el valor mínimo de d.

PAREO DE ESCANEOS

Es el proceso de alinear los puntos obtenidos del láser unos con otros o con un mapa existente. Los sensores láser modernos presentan bajo ruido al medir distancias a velocidades muy altas, es por eso que un método que registre dichos escaneos puede dar resultados muy precisos. En muchos sistemas robóticos, la exactitud y la precisión de los sensores láser es mucho mayor que los datos de la odometría, si es que estos existen.

Este enfoque está basado en la optimización de la alineación de los puntos con el mapa que se ha obtenido hasta el momento. El método consiste en usar una aproximación de Gauss–Newton inspirado en trabajos de visión por computadora. De esta forma no es necesaria una búsqueda de asociación de los datos tan exhaustiva. A medida de que los escaneos se alinean con el mapa existente, implícitamente se están alineando con los escaneos previos.

Se busca encontrar la transformación$\xi = (p_x, p_y, \psi)^T$ que minimice la ecuación:

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

(8)

es decir, se busca encontrar la transformación que dé la mejor alineación de los datos del escáner láser con el mapa, donde $S_i(\xi)$ son las coordenadas de los puntos escaneados $s_i = (s_{i,x}, s_{i,y})^T$en la base del mundo. Son una función de ξ, que corresponde a la posición del robot en la base del mundo:

$$S_i(\xi) = \begin{pmatrix} \cos\psi & -\sin\psi \\ \sin\psi & \cos\psi \end{pmatrix} \begin{pmatrix} S_{i,x} \\ S_{i,y} \end{pmatrix} + \begin{pmatrix} p_x \\ p_y \end{pmatrix}$$

(9)

De la función$M(S_i(\xi))$ se obtiene el valor del mapa en las coordenadas dadas por$S_i(\xi)$. Dada una estimación inicial de ξ, se requiere estimar la $\Delta\xi$ que optimice el error de medición según:

$$\sum_{i=1}^{n} [1 - M(S_i(\xi + \Delta\xi))]^2 \to 0$$

(10)

Expandiendo$M\big(S_i(\xi + \Delta\xi)\big)$ por serie de Taylor se obtiene:

$$\sum_{i=1}^{n} [1 - M(S_i(\xi) - \nabla M\big(S_i(\xi)\big) \frac{\partial S_i(\xi)}{\partial \xi} \Delta\xi]^2 \to 0$$

(11)

Esta ecuación se minimiza igualando la derivada parcial respecto a Δξcon cero, y resolviendo para$\Delta\xi$, produce la ecuación de Gauss–Newton para el problema de minimización:

$$\Delta\xi = H^{-1} \sum_{i=1}^{n} [\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi} \Delta\xi]^T [1 - M(S_i(\xi))]$$

(12)

donde:

$$H = [\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi}]^T \; [\nabla M(S_i(\xi)) \frac{\partial S_i(\xi)}{\partial \xi}]$$

(13)

Usando las coordenadas del escáner en la base del mundo se obtiene:

$$\frac{\partial S_i(\xi)}{\partial \xi} = \begin{pmatrix} 1 & 0 & -\sin(\psi) s_{i,x} & -\cos(\psi) s_{i,y} \\ 0 & 1 & \cos(\psi)\, s_{i,x} & -\sin(\psi) s_{i,y} \end{pmatrix}$$

(14)

Usando $\nabla M(S_i(\xi))$ y $\frac{\partial S_i(\xi)}{\partial \xi}$, la ecuación de Gauss–Newton se puede evaluar, permitiendo una diferencia de $\Delta\xi$ del mínimo buscado.

4

TOMA DE DECISIONES

La toma de decisiones es un proceso de selección entre cursos alternativos de acción, basado en un conjunto de criterios, para alcanzar uno o más objetivos.

Herbert Simon

"The new science of Management Decision"

Harper and Row, New York, 1960.

4.1 METODOLOGÍA

Inventada por Stuart Pugh, el método de la matriz de decisión, también llamado método Pugh, es una técnica cuantitativa usada para ordenar por relevancia las opciones multidimensionales de un sistema de soluciones. Es frecuentemente usado en ingeniería para tomar decisiones, pero también puede ser usado para ordenar opciones de inversión, de venta o compra, diseño del producto o para ordenar cualquier conjunto de entidades multidimensionales.

Una matriz de decisión básica consiste en establecer un conjunto de opciones calificadas y sumadas, para obtener una calificación total que puede ser ordenada en relación con las demás opciones.

Una matriz de decisión con pesos opera de la misma forma que la matriz básica, pero introduce el concepto de darle un peso a cada criterio en orden de su importancia. Las calificaciones resultantes reflejan mejor la importancia de los criterios involucrados. Mientras más importantes sean los criterios, mayor es el peso que se le debe dar. Cada opción potencial se califica y multiplica por el peso dado a cada criterio para obtener el resultado final.

La ventaja de utilizar la matriz de decisión es que las opiniones subjetivas sobre una alternativa contra otra pueden hacerse más objetivas. Otra ventaja de este método es que se pueden realizar estudios de sensibilidad. Un ejemplo sería ver qué tanto debe cambiar una opinión para que una alternativa evaluada baja pueda vencer a una alternativa con mejor calificación.

4.2 SELECCIÓN

A continuación se muestra la matriz de decisión realizada para conformar el sistema final. Cada renglón representa una posible herramienta o técnica a ser usada y cada columna representa un criterio de selección o restricción que debe cumplir. De esta forma, se calificará cada propuesta en relación a qué tanto cumple con el criterio de selección o qué tan viable es, dadas las restricciones. Las calificaciones serán marcadas con un número del cero al cinco, donde 0 quiere de decir no cumple o no es viable y 5 quiere decir cumple totalmente o es totalmente viable.

Además, a cada criterio de selección y restricción se le ha dado un peso o valor de importancia del cero al cinco, así, la calificación de cada propuesta dependerá de qué tanto cumpla con cada criterio, teniendo en cuenta qué tan importante es ese criterio para la correcta resolución del problema. La calificación final será la suma total de las casillas en cada renglón.

Finalmente, se elegirán las propuestas que tengan las puntuaciones más altas y que sean complementarias.

MATRIZ DE DECISIÓN

Tabla 2 Matriz de decisión.

Propuesta / criterio	Rapidez	Precisión	Costo computacional	Costo monetario	Flexibilidad	Ambiente caótico	Telecomunicación comprometida	Deslizamientos	Bajas reservas de energía	Portabilidad	TOTAL
Peso	3	4	2	1	5	5	4	4	3	2	
Online	1	1	1	1	5	5	5	2	1	4	**99**
Offline	2	3	3	1	1	3	2	2	3	2	74
Mapeo por característica	5	2	5	1	0	1	2	3	4	3	77
Mapeo por cuadriculas	4	1	2	1	5	5	2	4	2	3	**107**
Mapeo paramétrico	4	2	4	1	1	0	2	3	4	3	72
Mapeo no paramétrico	3	2	2	1	4	5	2	3	2	3	**99**
Filtro de Kalman descent	2	2	3	3	3	4	3	4	3	3	101
Filtro de Kalman extendi	3	3	2	3	4	3	3	3	3	3	**102**
Filtro de partículas	4	3	2	1	4	5	3	4	2	2	**112**
Filtro de Bayes	3	2	3	1	4	3	3	2	3	3	94
Mapa 2D	3	3	4	2	4	3	3	3	4	4	**110**
Mapa 3D	1	1	1	1	5	5	3	2	1	3	89
Odometría por encoders	5	1	4	5	0	0	3	0	4	4	64
Odometría visual	2	2	2	2	5	4	2	5	1	3	**102**
Cámara	2	2	2	5	5	3	3	4	4	4	34
Infrarrojos	3	1	2	5	3	5	2	2	2	1	26
Sonar	2	2	2	5	4	3	2	3	2	3	28
Kinect	3	2	2	5	4	5	2	5	2	3	**111**
Láser	5	4	3	0	3	3	2	5	2	3	107
IMU	4	4	3	1	4	4	4	5	1	3	33
AGV	3	4	3	2	0	0	4	4	3	2	78
SLAM	2	3	1	2	5	4	5	3	2	4	**113**
ROS	5	5	4	5	5	5	4	4	4	5	**152**
Windows	3	5	2	2	2	4	3	4	2	3	105

Tomando en cuenta los resultados obtenidos en la matriz de decisión, se procederá a construir el sistema con los siguientes métodos y herramientas:

Se obtendrán el mapa y la posición del robot por medio de la técnica SLAM o Localización y Mapeo Simultáneos, el cual será una versión online en 2D con un mapa por ocupación de cuadrícula no paramétrico. Se aplicará un filtro de Kalman extendido para reducir el error en las mediciones de distancias combinado con un filtro de partículas. La odometría será obtenida por medio de una técnica visual a través de las mediciones de distancias. El sensor de profundidad que se usará será el Kinect y todo esto estará programado en la plataforma de ROS corriendo sobre Linux.

5

CONSTRUCCIÓN, PRUEBAS Y RESULTADOS

5.1 CONSTRUCCIÓN DEL SISTEMA

El sistema final que dará como resultado el mapa la localización del robot consta de los siguientes procesos:

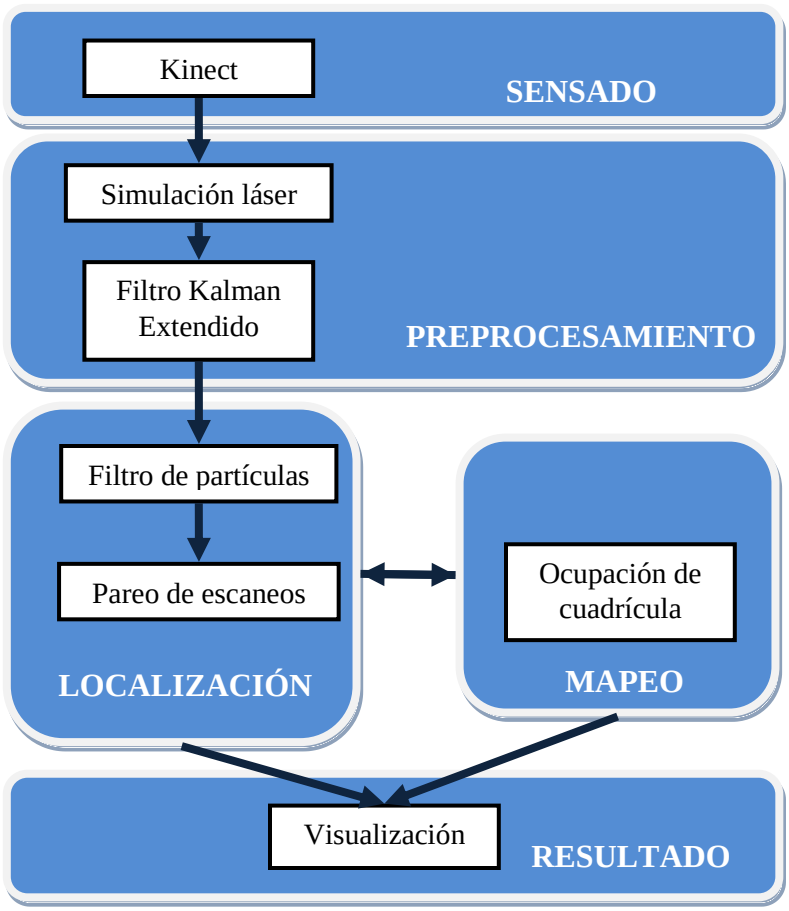


Figura 5.1 Diagrama del sistema final.

Como se comentó en el capítulo 3, una de las facilidades que brinda el sistema ROS es la capacidad de dividir el código en pequeños módulos o nodos que hagan procesos específicos. Con esto en mente el total del código encargado de realizar el SLAM será dividido de la siguiente manera:

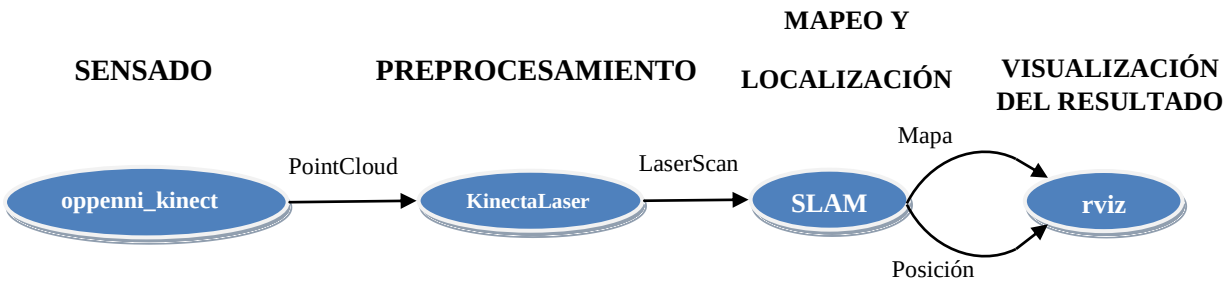


Figura 5.2 Diagrama final de nodos y mensajes. Los óvalos representan nodos y las flechas menasjes.

Otra de las ventajas al utilizar ROS es la gran candidad de código abierto que se encuentra en el sitio oficial, el cual fue desarrollado por investigadores de todo el mundo y puesto a disposición de todos los usuarios de ROS. Esto permite ahorrar tiempo en programación si se encuentra un nodo o paquete se ajuste a las necesidades del sistema en cuestión. Así, para obtener los datos del Kinect se usará el nodo principal de un paquete de código abierto llamado oppenni_kinect. Esta misma situación se presena al momento de visualizar el resultado, pero en esta ocasión se usará el nodo rviz, el cual es una interfaz gráfica de visualización de diferentes tipod de datos.

En el caso de el preprocesamiento y el algoritmo de SLAM se programaron nodos separados, los cuales fueron nombrados KinectaLaser y SLAM respectivaente.

El código de cada nodo se encuentra detallado en los apéndices.

Es importante hacer notar que debido a las importantes variaciones de orientación en pitch y roll que presentará el robot se ha progrmado el sistema de forma que tenga la posibilidad de considerar estos datos en el cálculo final. Por lo tanto el filtro de partículas puede tomar en cuenta datos de un escaneo horizontal pero también de un escaneo vertical, lo cual influirá sobre todo en el resultado de la odometría. De esta forma el robot puede saber si el cambio en los datos de distancia obtenidos del Kinect se deben a un desplazamiento o a una inclinación, dándole así mayor certeza a los resultados.

Dicho proceso se explica de forma gráfica en el siguiente diagrama:

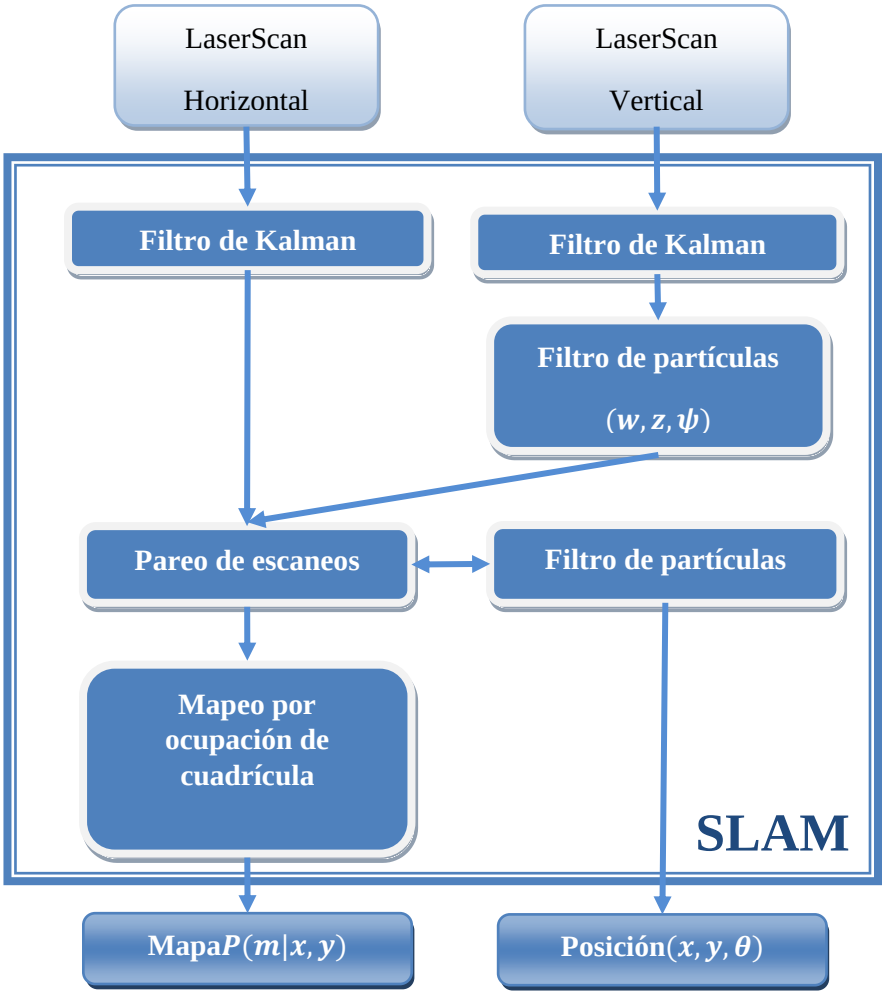


Figura 5.3 Variación de sistema de SLAM para incluir datos de orientación en pitch y roll.

RELACIONANDO LAS BASES

Para poner en marcha el sistema, es necesario correr los nodos en el orden, con los nombres y propiedades pertinentes. Aquí se muestran las diferentes opciones a configurar. La imagen de abajo muestra todas las posibles bases de interés en una vista simplificada en 2D de la navegación de un robot sobre un terreno tosco, las cuales producen un movimiento de la plataforma en direcciones de pitch y de roll.

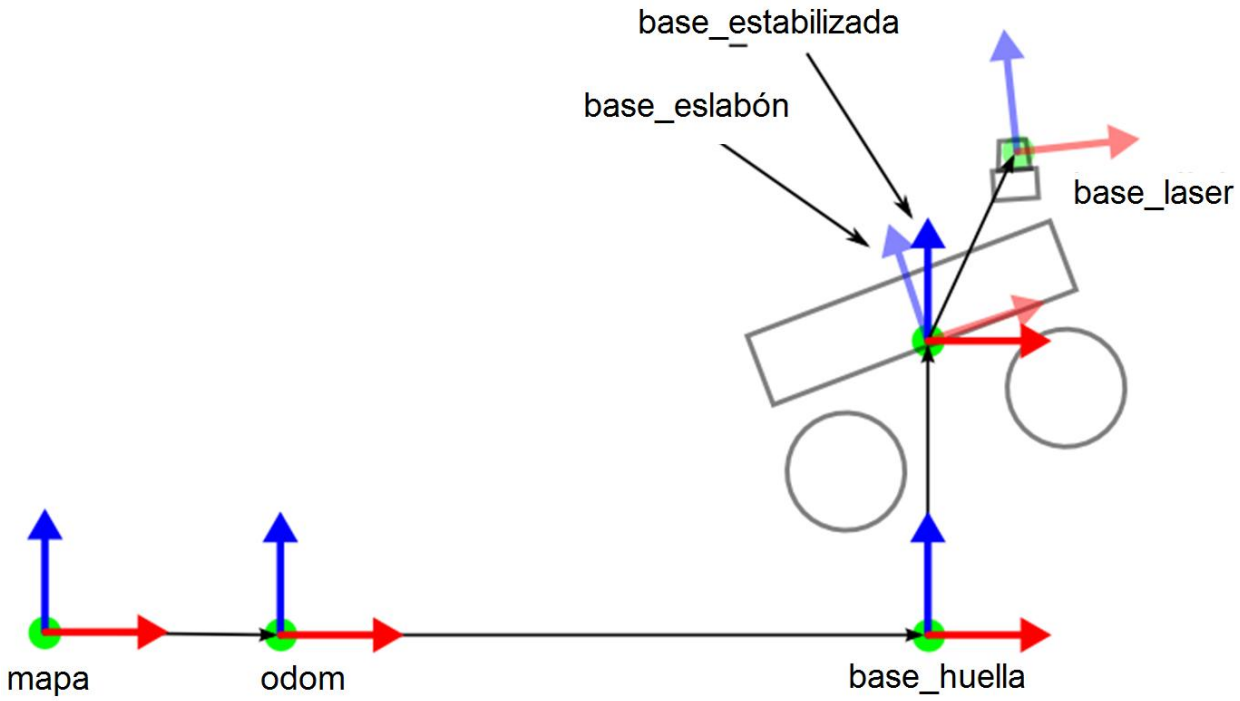


Figura 5.4 Relación de bases del robot y el mundo.

La relación general entre las bases del mapa, la odometría (odom) y la base eslabón usan las convenciones de ROS para nombrar y darle un significado semántico a las bases coordenadas de plataformas móviles. Estas bases se describen a continuación.

Mapa

La base coordenada llamada mapa, es una base fija al mundo con su eje ¨z¨ apuntando hacia arriba. La posición de cualquier plataforma móvil relativa a la base del mapa no debe variar significativamente en el tiempo. Sin embargo, dicha base no es continua, por lo que la posición de la plataforma móvil puede cambiar en saltos discretos en cualquier momento.

En este caso, un componente proveniente de los cálculos de localización hará que la posición del robot en el mapa sea constantemente recalculado debido a que en las observaciones de los sensores se eliminan las variaciones de la posición, pero causa saltos discretos cada vez que se recibe nueva información de los sensores.

La base del mapa es útil como una referencia general, pero los saltos discretos hacen que sea una mala base de referencia local para el sensado y el movimiento de los actuadores.

Odom

La base coordenada llamada odom, o la base de odometría, es también una base fija al mundo. La posición de una plataforma móvil puede variar en el tiempo sin restricciones, esto hace de la base de odometría poco útil como una referencia global. De cualquier forma, la posición de un robot en la base de odometría se garantiza que sea continua, eso quiere decir que la posición de una plataforma móvil en la base de odometría evoluciona siempre de forma suave, sin saltos discretos.

La base de odometría es calculada con base en las fuentes de odometría, que pueden ser encoders, sistemas de odometría visual o unidades de medición de inercia (IMU).

Esta base es útil como una referencia local precisa, pero las variaciones y deslizamientos la hacen una mala referencia global.

Base_eslabón

La base coordenada llamada base_eslabón, es una base rígida fija a la base del robot. La base eslabón puede ser unida a la del robot en una posición u orientación arbitraria, ya que para cada plataforma o hardware se usa un lugar diferente como punto de referencia.

Relación entre las bases

En ROS, las bases coordenadas están relacionadas de forma que cada base puede ser unida a una base padre o a varias bases hijas. Para las bases expuestas aquí la relación es la siguiente:

Mapa→odom→base_eslabón

La base del mapa es padre de la base de odometría y ésta es padre de la base eslabón.

La transformación de la base de odometría a la base eslabón es calculada y publicada por las fuentes de odometría. La transformación de la base del mapa a la base eslabón es calculada por el componente de localización, pero esta componente no publica la transformación de la base del mapa a la base eslabón.

Primero recibe la transformación de la base de odometría a la base eslabón y usa esa información para publicar la transformación del mapa a la odometría.

En este caso particular se usan dos bases intermedias entre la base de odometría y la base eslabón: la base huella, que no provee información de la altura y representa la posición y orientación del robot en 2D; y la base estabilizada, que añade información sobre la altura del robot relativa a la capa del mapa y la odometría. La base eslabón está fija de forma rígida al robot y añade los ángulos del roll y pitch, a diferencia de la base estabilizada. Para esta transformación se puede usar un sistema para estimación de la altitud como AHRS (Attitude and Heading Reference System) o INS (sistema de navegación inercial). Si la plataforma no presenta variaciones en roll y pitch, la base estabilizada y la base eslabón son las mismas y la transformación sería publicada por un publicador estático.

5.2 PUESTA EN MARCHA

Este sistema es capaz de tomar datos de los sistemas de odometría de un sistema o proceso externo o generarlos de forma visual. También es posible generar los datos de posicionamiento 3D visualmente o usar el sistema con robots que generan datos significativos de pitch y roll, es decir, que navegan en entornos toscos y usan un sensor IMU. Por lo tanto, es necesario especificar cómo va a trabajar este sistema y escribirlo en los parámetros que usará el nodo para correr los cálculos. Estos parámetros tendrán efecto en el nodo de SLAM:

publicar odometría en el mapa

<param name="pub_map_odom_transform" value="false"/>

se nombran las bases

<param name="map_frame" value="mapa" />

<param name="base_frame" value="base_frame" />

en caso de publicarse la odometría se hará en el cuadro base

<param name="odom_frame" value="base_frame" />

Los nodos de ROS reciben información de los tópicos a los que están suscritos, la procesan y la envían a los tópicos que ellos publican. A continuación se especifica el flujo de información en los nodos de este sistema:

Tópicos a los que está suscrito

Los datos del láser usados por el sistema de SLAM

scan (sensor_msgs/LaserScan)

Comando del sistema. Si la cadena de caracteres iguala “reset”, el mapa y las coordenadas del robot se inicializan

syscommand (std_msgs/String)

Tópicos que publica

En estos tópicos se actualiza periódicamente la información del mapa.

map_metadata (nav_msgs/MapMetaData)

map (nav_msgs/OccupancyGrid)

la posición estimada del robot sin covarianza

slam_out_pose (geometry_msgs/PoseStamped)

La posición estimada del robot con una estimación gaussiana de la incertidumbre.

poseupdate (geometry_msgs/PoseWithCovarianceStamped)

Servicios

Los servicios son una forma de procesar información solamente en el momento que se necesite, es decir el nodo solo realiza este cálculo en el momento que recibe la información necesaria e inmediatamente después envía el resultado al nodo solicitante.

Se llama a este servicio para obtener los datos del mapa.

dynamic_map (nav_msgs/GetMap)

Parámetros

Los parámetros se usan como constantes en los nodos que lo soliciten y sirven para renombrar variables, ajustar su resolución, sus valores iniciales, entre otras opciones. En el nodo de SLAM se usan los siguientes:

El nombre del la base del robot. Esta base se usa para la localización y las transformaciones de los datos del escáner.

~base_frame (string, default: base_link)

El nombre de la base del mapa.

~map_frame (string, default: map_link)

El nombre de la base de la odometría.

~odom_frame (string, default: odom)

La resolución del mapa en metros. El largo de un lado de una celda.

~map_resolution (double, default: 0.025)

El tamaño, nombre de celdas por eje, del mapa. El mapa es cuadrado de tamaño map_size*map_size.

~map_size (int, default: 1024)

Localización del origen del eje x de la base del mapa relativo a la cuadrícula. 0.5 quiere decir en el medio.

~map_start_x (double, default: 0.5)

Localización del origen del eje y de la base del mapa relativo a la cuadrícula. 0.5 quiere decir en el medio.

~map_start_y (double, default: 0.5)

Distancia en metros que debe recorrer el robot para provocar una actualización en el mapa.

~map_update_distance_thresh (double, default: 0.4)

Movimiento angular en radianes que debe realizar el robot para provocar una actualización en el mapa.

~map_update_angle_thresh (double, default: 0.9)

Periodo en segundos en el que se publica el mapa

~map_pub_period (double, default: 2.0)

El número de cuadrículas multiresolución que puede tener el mapa

~map_multi_res_levels (int, default: 3)

Factor de actualización para los puntos del mapa vacíos. 0.5 quiere decir sin cambio.

~update_factor_free (double, default: 0.4)

Factor de actualización para los puntos del mapa ocupados. 0.5 quiere decir sin cambio.

~update_factor_occupied (double, default: 0.9)

La distancia mínima en metros de los datos obtenidos por el láser que serán usados en el sistema. Puntos más cercanos a este valor serán ignorados.

~laser_min_dist (double, default: 0.4)

La distancia máxima en metros de los datos obtenidos por el láser que serán usados en el sistema. Puntos más lejanos a este valor serán ignorados.

~laser_max_dist (double, default: 30.0)

La altura mínima en metros relativa a la línea del escáner láser que será tomada en cuanta al usar los datos de distancia. Puntos escaneados debajo de esta altura serán ignorados.

~laser_z_min_value (double, default: -1.0)

La altura máxima en metros relativa a la línea del escáner láser que será tomada en cuenta al usar los datos de distancia. Puntos escaneados por encima de esta altura serán ignorados.

~laser_z_max_value (double, default: 1.0)

Determina si la transformación de la base del mapa a la base de la odometría debe ser publicada por el sistema.

~pub_map_odom_transform (bool, default: true)

El tamaño de cola del subscriptor de escaneos. Debe ser ajustado a niveles altos (por ejemplo 50) si los datos deben ser leídos por el nodo finder_slam más rápido que la velocidad en tiempo real.

~scan_subscriber_queue_size (int, default: 5)

Determina si el pareo de escaneos debe ser publicada.

~pub_map_scanmatch_transform (bool, default: true)

Nombre de la base donde se publica el pareo de escaneos.

~tf_map_scanmatch_transform_frame_name (string, default: scanmatcher_frame)

5.3 IMPLEMENTACIÓN

Como se comentó con anterioridad, el sensor imprescindible para el cálculo del SLAM es el sensor de profundidad, ya que a través de éste se pueden obtener los datos del ambiente que se usarán para crear el mapa y los datos del posicionamiento del robot que se usarán para determinar la localización. Típicamente, en todos los sistemas estudiados se usan sensores láser para obtener los datos de profundidad. Dichos sensores son muy precisos y rápidos pero también son bastante caros, por lo tanto no son una elección tan atractiva. Por consiguiente, y como se determinó en el capítulo de toma de decisiones, en lugar de un sensor láser se usará el sensor Kinect de Microsoft, el cual provee información RGBD, es decir, vídeo a color más información de distancia de cada pixel. Esto no sólo permite usar el Kinect como sensor de profundidad que hace factible un SLAM en 2D, sino que es capaz también de recibir información tanto de distancia como de color que permita una reconstrucción en 3D del ambiente.

CONEXIÓN DEL KINECT

Para hacer funcionar el sensor Kinect en ROS, es necesario instalar los controladores del mismo. Existe una variedad de controladores desarrollados por diversas compañías y entidades no lucrativas. De todos ellos, el que ofrece un mayor control sobre las capacidades del Kinect es el controlador llamado “openni”, por lo tanto, será éste el que se instale en el sistema escribiendo en una terminal:

```
sudo apt-get install ros-groovie-openni-kinect
```

Una vez instalado y conectado el Kinect, se verifica que aparezca la información del sensor de profundidad escribiendo en la terminal lo siguiente:

```
roslaunch openni_camera openni_node.launch
```

Luego, en una terminal diferente se abre rviz:

```
rosrun rviz rviz
```

‘’rviz’’ es una herramienta de visualización en 3D para ROS, en ella se puede observar todo tipo de tópicos visuales publicados en el master de ROS.

Ya en rviz se ingresa a Global Options y se modifica la opción Fixed Frame a /openni_camera que es el tópico que publica los datos de profundidad. Después, se añade una instancia PointCloud2 y se selecciona el tópico /camera/rgb/points. Ahora debe poder verse una imagen con datos de profundidad con un código de colores en 3D similar a la que se ve en la Figura 5.5.

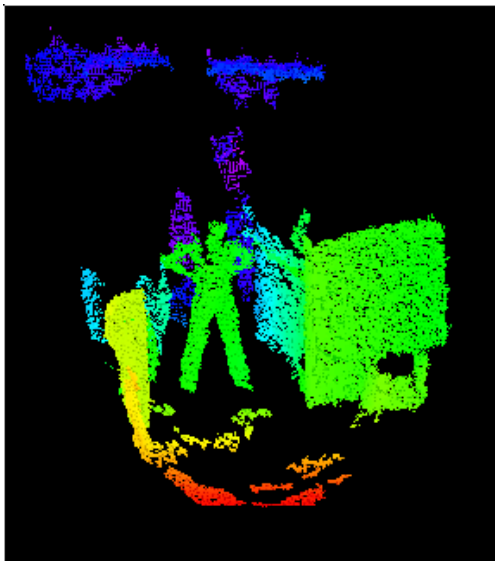


Figura 5.5 Imagen RGB-D obtenida del Kinect.

SIMULACIÓN DE UN ESCANER LÁSER

Para usar la imagen de profundidad en el sistema de SLAM, se deben convertir primero los datos de la nube de puntos para simular un escáner láser. Para esto, es necesario correr un nodo que se encargue de dichos cálculos. Se programó un nodo llamado KinectaLaser, el cual corta una rebanada horizontal de la imagen y usa la distancia más cercana en cada columna como si fuera el dato del sensor láser falso.

Ahora, para ver el escaneo láser simulado en rviz, en una nueva terminal se ejecuta lo siguiente:

```
roslaunch kinectalaser.launch
```

En rviz se ajusta la opción Fixed Frame a /openni_depth_frame y se añade una instancia de Laser Scan seleccionando el tópico /scan, que es el que publica el nodo KinectaLaser.

Ahora, se deben ver los datos del láser simulado obtenidos de la nube de puntos del Kinect. En la imagen se muestran los datos del láser con una cuadrícula de 0.1m y una representación de base coordenada mostrando la base del Kinect obtenida mediante /openni_camera.

El código del programa kinectalaser.launch se puede consultar en el Apéndice A.

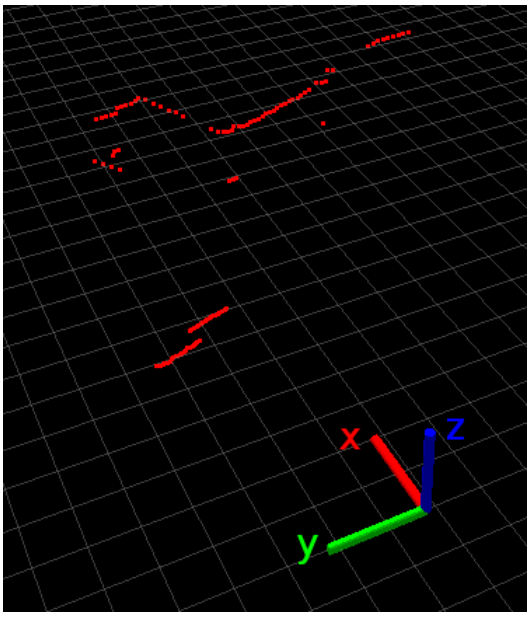


Figura 5.6 Resultado obtenido al simular un escáner láser con el Kinect.

INTEGRACIÓN DE TODOS LOS ELEMENTOS

Finalmente, se hará funcionar el SLAM. Como el algoritmo del SLAM requiere de información sobre la posición del sensor de profundidad, es necesario publicar una transformación que relacione la base del robot (base_link) con la base donde se encuentra el Kinect (openni_camera). Esta transformación hace posible transformar los datos del escáner a la base del robot. Los datos de la posición del sensor se colocan en un archivo ejecutable con los procesos ya creados. Además, se deben poner en marcha al mismo tiempo los nodos que se encargan de dibujar el mapa y de obtener la localización del robot. Todos estos procesos se programaron en un archivo denominado slam.launch, cuyo código se puede consultar el en Apéndice B. Dicho archivo corre el nodo principal del proceso, el encargado de realizar el SLAM, el cual se puede ver a detalle en el apéndice C.

Finalmente, se corre el archivo launch que hace los cálculos del SLAM con los parámetros ya especificados.

```
roslaunch finder slam.launch
```

Ahora, al mover el robot por el ambiente, se puede ver en rviz cómo el sistema va creando el mapa en tiempo real. A medida que el robot avanza, el sistema de pareo de escaneos y el filtro de partículas van depurando las lecturas para mostrar un mapa que encaje de la mejor forma con los datos obtenidos del sensor.

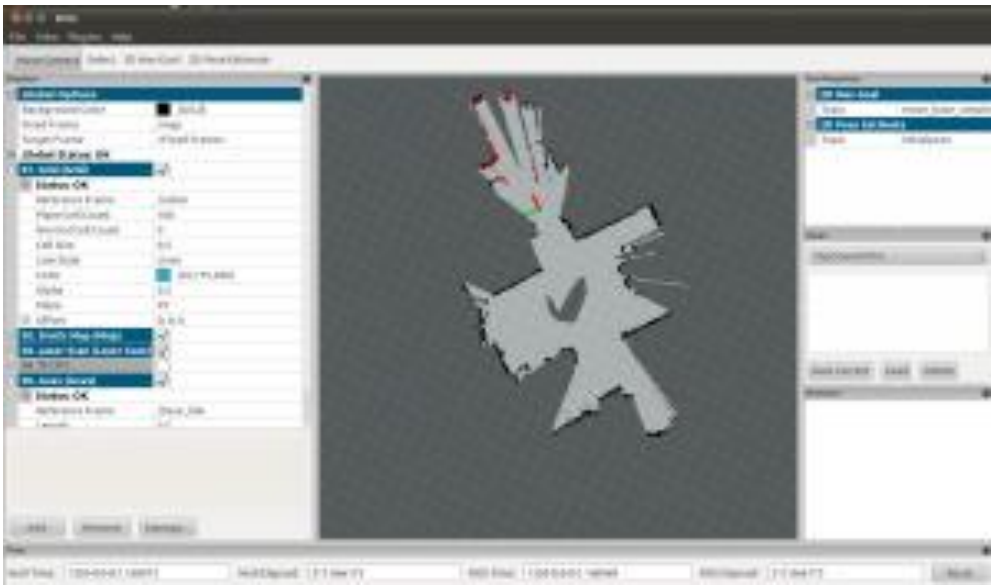


Figura 5.7 Mapa obtenido con el sistema de SLAM propuesto.

5.4 PRUEBAS Y RESULTADOS

El sistema fue probado en el robot de búsqueda en entornos de desastre nombrado FinDER, el cual cuenta con una computadora a bordo con un procesador Intel i5, y 6 GB de memoria RAM.

Al poner el robot en movimiento, lo primero que se pudo notar fue que las lecturas eran muy rápidas para poder ser correctamente procesadas, por lo que los puntos del mapa se fueron encimando dando como resultado mapas irreconocibles. Para corregir este problema, se disminuyó la velocidad de lectura del sensor varias veces hasta lograr un comportamiento estable; finalmente, se le indicó al sensor que enviara un mensaje con los datos obtenidos a una frecuencia de 2 Hertz. Además, fue también necesario limitar la velocidad de avance del robot para que el sistema pudiera encontrar de forma más confiable los escaneos consecutivos coincidentes y así obtener una odometría que no induzca errores en el mapeo. Con esta modificación se obtuvieron mapas más fieles al entorno.

Fue también muy evidente que el sistema trabajó mejor en condiciones de poca luz, quedando completamente inutilizable al encontrarse bajo los rayos directos del Sol. Esto fue debido a que el Kinect obtiene las lecturas de profundidad a través de emisores y receptores infrarrojos y la luz solar posee en su espectro una gran cantidad de luz infrarroja. Esto produce una interferencia enorme en el sensor, el cual se vuelve incapaz de detectar los objetos a su alrededor.

En general, los mapas creados representaron fielmente la geometría del lugar en que se encontraba el robot; sin embargo, se observó que en ambientes un poco más complicados, como derrumbes por ejemplo, la lectura de las distancias al nivel del sensor no siempre son suficientes para tomar decisiones acertadas en cuanto a la navegación, ya que existen muchos objetos que representan un obstáculo en el plano pero el robot sería capaz de sortearlos moviéndose sobre ellos, en el plano vertical.

Por esta razón, el SLAM 2D se combinará con una técnica de SLAM 3D desarrollada en un nodo de ROS llamado RGBDSLAM 6DOF– SLAM. Este nodo permite adquirir modelos 3D coloreados de objetos y escenarios en interiores con la ayuda de un Kinect y sin la necesidad del cálculo paralelo de odometría. Usa un pareo de características SIFT o Transformación de Características de Escala Invariante (Scale Invariant Feature Transform) y SURF o Caracterización Robusta Acelerada (Speeded Up Robust Feature) para comparar las imágenes captadas por la cámara en movimiento, y usa RANSAC o Consenso de Muestras Aleatorias (RANdom SAmple Consensus) para estimar la transformación 3D entre ellas, generando así la función análoga del pareo de escaneos en el SLAM 2D. Para que el procesamiento pueda ser en línea, la imagen actual se compara únicamente con un conjunto reducido de imágenes previas. Así, construye una gráfica cuyos nodos corresponden a las capturas de la cámara y los bordes corresponden a la transformación 3D estimada. Por último, el resultado se optimiza mediante un proceso llamado HOG-Man, otro nodo de ROS que disminuye los errores de posición acumulados.

6

DISCUSIÓN

La calidad del algoritmo de SLAM depende en gran parte del número de características re-observadas. Mientras más regularmente se actualice un punto, un borde o una esquina, mejor será la calidad del sensado y la estimación de la posición del robot. Claramente el algoritmo falla si no se concreta ninguna observación, ya que para el filtro de partículas es importante tener actualizaciones continuas para prevenir la degeneración de la estimación de las coordenadas del robot.

La selección de los componentes y algoritmos que forman el sistema de mapeo y localización, en general, responden a las expectativas que se tenían. Sin embargo, se detectó que el eslabón más débil es el uso del Kinect como sensor de profundidad. La velocidad de sensado en un 25% menor de lo que un sensor láser puede otorgar, además la cantidad tan grande de puntos obtenidos por el Kinect provoca una disminución en la velocidad de los cálculos. En concreto, la utilización del Kinect restringe el movimiento del robot a una velocidad de 0.02 m/s para dar tiempo a que se realicen los cálculos sin que el sistema se pierda.

Teniendo en cuenta estos resultados será necesario hacer modificaciones que permitan al sistema realizar el procesamiento de forma más rápida.

7

CONCLUSIONES Y TRABAJO A FUTURO

El sistema presentado en este trabajo permitió que un robot de rescate pudiera navegar en una gran variedad de ambientes, incluyendo entornos post–desastre, haciendo un mapa del ambiente y al mismo tiempo localizarse en él. Fue probado en el robot FinDER diseñado para búsqueda de personas en entornos de desastre. El uso del sistema de odometría visual permitió al robot moverse por entornos ásperos sin importar el deslizamiento de las ruedas, además la estimación del movimiento en 3D logró dar al sistema buena confiabilidad, aún cuando el robot se moviera en el eje vertical.

El gasto de recursos computacionales del SLAM diseñado fue suficientemente bajo como para poder instalarse en un robot móvil que haga el proceso a bordo de forma autónoma y en tiempo real.

Como se mencionó en el apartado anterior, un aspecto a mejorar es la velocidad de operación del sistema. Como trabajo a futuro será conveniente realizar pruebas del sistema con un sensor láser, el cual ofrece lecturas más rápidas y un ángulo de visión horizontal del triple que el Kinect.

También se considera como un paso importante para lograr un avance considerable en el desarrollo del sistema, el estudio de la fusión de los datos de los diversos sensores. El sistema se volverá más robusto al contar con información redundante y con sensores especializados para reducir el error. Uno de los cálculos que presentan mayor incertidumbre es el del movimiento vertical del robot, lo cual puede mejorarse con el uso de unidades de medición inercial y sensores de presión barométrica.

8

REFERENCIAS

[1] Openslam. Plataform for SLAM reserchers. Disponible en http://openslam.org/rgbdslam.html. Consultado el 30 de marzo de 2013.

[2] Stefan Kohlbrecher, Oskar von Stryk, Johannes Meyer y Uwe Klingauf. *A flexible and scalable SLAM system with full 3D motion estimation.* IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), Darmstadt, Alemania. 2011

[3] Thrun, S., Burgard, W., y Fox, D. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press. Cambridge, Masachusetts. 2005.

[4] Thrun, S., Burgard, W., y Fox, D.*A Probabilistic Approach to Concurrent Mapping and Localization for Mobile Robots*. The MIT Press. Cambridge, Masachusetts. 1997.

[5] Keiji Nagatani, Hiroshi Ishida, Satoshi Yamanaja y Yukata Tamaka. *Three-dimensional Localization and Mapping for Mobile Robot in Disaster Environments*. International Conference on Intelligent Robots and Systems, Las Vegas, Nevada, Octubre 2003.

[6] Peter Zhang, Evangelous Millos y Jason Gu. *General Concept of 3D SLAM*. Dalhousie University, Canada. 2008.

[7] Leonard, John, and P. Newman. *Consistent, convergent, and constant-time SLAM.* International Joint Conference on Artificial Intelligence. Vol. 18. Lawrence Erlbaum Associates Ltd. 2003.

[8] Kim, Sung Joon. *Efficient simultaneous localization and mapping algorithms using submap networks.* The MIT Press. Cambridge, Masachusetts. 2004.

[9] Hiebert-Treuer, Bradley. *An Introduction to Robot SLAM (Simultaneous Localization And Mapping).* Computer Science Master Theses. Middlebury College. Middlebury, Vermount, USA. 2007.

[10] Kuo, Bor-Woei, et ál. *A Light-and-Fast SLAM Algorithm for Robots in Indoor Environments using Line Segment Map.* Journal of Robotics, Vol. 2011, pp. 1-12. Hsinchu City, Taiwan. 2011.

[11] Garulli, Andrea, et ál. *Mobile robot SLAM for line-based environment representation. Decision and Control*.2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on. IEEE, Sevilla, España. 2005.

[12] Bosse, Michael, et ál. *Simultaneous localization and map building in large-scale cyclic environments using the Atlas framework*. The International Journal of Robotics Research. pp. 1113-1139. Londres, Inglaterra. 2004

[13] Fernández, L., Payá, L., Ballesta, M., Amorós, F., & Reinoso, O. *Odometría visual y construcción de un mapa topológico a partir de la apariencia global de imágenes omnidireccionales*. XXXII Jornadas de Automática. Sevilla, España. 2011.

[14] Robot Operating System. Disponible en http://ros.org/wiki/. Consultado el 14 de junio de 2013

9

APÉNDICES

APÉNDICE A

Archivo kinectalaser.launch

```
<launch>
<!-- controladores del kinect -->
<include file="$(find openni_camera)/launch/openni_node.launch"/>

<!-- openni_manager -->
<node pkg="nodelet" type="nodelet" name="openni_manager" output="screen" respawn="true" args="manager"/>

<!--regulando la velocidad de lectura -->
<node pkg="nodelet" type="nodelet" name="pointcloud_throttle" args="load
pointcloud_to_laserscan/CloudThrottle openni_manager">
<param name="max_rate" value="2"/>
<remap from="cloud_in" to="/camera/depth/points"/>
<remap from="cloud_out" to="cloud_throttled"/>
</node>

<!-- simulando el láser -->
<node pkg="nodelet" type="nodelet" name="kinect_laser" args="load pointcloud_to_laserscan/CloudToScan
openni_manager">
<param name="output_frame_id" value="/openni_depth_frame"/>
<remap from="cloud" to="cloud_throttled"/>
</node>
</launch>
```

El primer include abre el archivo launch del nodo openni que hace funcionar la cámara del Kinect, en caso de que no se haya puesto en marcha (openni camera).

El nodo de tipo nodelet llamado openni_manager actúa como el contenedor para los otros nodelets. Los nodelets son como los nodos pero en vez de correr como procesos independientes, se cargan a un nodo padre del tipo nodelet. Estos nodelets que se ejecutan en el mismo proceso pueden intercambiar información fácilmente. Esto es relevante cuando se procesan grandes cantidades de datos. El archivo launch carga dos nodelets para a) modular la velocidad de lectura del sensor rgb-d a 2 Hertz y b) cortar una rebanada de cada lectura, calcular las distancias columna por columna y ensamblarlas como si fueran datos de la lectura de un láser publicados en el tópico /scan.

para asegurar que todos los parámetros necesarios sean publicados correctamente se puede correr el siguiente launch:

```
<launch>

<param name="pub_map_odom_transform" value="true"/><param name="map_frame" value="map"/><param
name="base_frame" value="base_frame"/><param name="odom_frame" value="base_frame"/>

<node pkg="tf" type="static_transform_publisher" name="map_2_base_link" args="0 0 0 0 0 0 /map
/base_stabilized 100"/><node pkg="tf" type="static_transform_publisher" name="base_2_base_stablized_link"
args="0 0 0 0 0 0 /base_stabilized /base_frame 100"/><node pkg="tf" type="static_transform_publisher"
name="base_2_laser_link" args="0 0 0 0 0 0 /base_frame /openni_camera 100"/><node pkg="tf"
type="static_transform_publisher" name="base_2_nav_link" args="0 0 0 0 0 0 /base_frame /nav 100"/>

</node>
</launch>
```

APÉNDICE B

Archivo slam.launch

```
<launch>
  <node pkg="tf" type="static_transform_publisher" name="base_to_kinect_broadcaster" args="-0.115 0 0.226 0 0
0 base_link openni_camera 100" />
 </node>

 <node pkg="finder_slam" type="finder_slam" name="finder_slam" output="screen">
  <param name="scan_topic" value="laser1/scan" />
  <param name="base_frame" value="base_stabilized" />
  <param name="output_timing" value="false"/>
  <param name="use_tf_scan_transformation" value="true"/>
  <param name="use_tf_pose_start_estimate" value="false"/>

  <param name="map_pub_period" value="1.0"/>

  <param name="laser_z_min_value" value = "-0.3"/>

  <param name="update_factor_free" value="0.3"/>

  <param name="map_resolution" value="0.05"/>
  <param name="map_size" value="1024"/>
  <param name="map_start_x" value="0.5"/>
  <param name="map_start_y" value="0.5"/>
  <param name="map_multi_res_levels" value="1"/>

  <remap from="map" to="scanmatcher_map" />
 </node>

 <group unless="$(optenv REALROBOT false)">
  <param name="finder_slam/use_tf_pose_start_estimate" value="false"/>
 </group>

 <param name="finder_slam/odom_frame" value="nav"/>
 <param name="finder_slam/pub_map_odom_transform" value="false"/>
 <node pkg="tf" type="static_transform_publisher" name="map_nav_broadcaster" args="0 0 0 0 0 0 map nav 100"
/>
</launch>
```

APÉNDICE C

Archivo finder_slam.cpp

El siguiente código fue escrito a partir de el código original del nodo hector_mapping disponible en el sitio www.ros.org.

```
//=================================================================================
// Copyright (c) 2011, Stefan Kohlbrecher, TU Darmstadt
//=================================================================================
// Código modificado por Gabriel Arroyo, FI, UNAM. 2013
//=================================================================================

#include "HectorMappingRos.h"

#include "map/GridMap.h"

#include <geometry_msgs/PoseWithCovarianceStamped.h>
#include <nav_msgs/Odometry.h>

#include "sensor_msgs/PointCloud2.h"

#include "HectorDrawings.h"
#include "HectorDebugInfoProvider.h"
#include "HectorMapMutex.h"

#ifndef TF_SCALAR_H
  typedef btScalar tfScalar;
#endif

HectorMappingRos::HectorMappingRos()
  : debugInfoProvider(0)
  , hectorDrawings(0)
  , lastGetMapUpdateIndex(-100)
  , tfB_(0)
  , map__publish_thread_(0)
  , initial_pose_set_(false)
{
  ros::NodeHandle private_nh_("~");

  std::string mapTopic_ = "map";

  private_nh_.param("pub_drawings", p_pub_drawings, false);
  private_nh_.param("pub_debug_output", p_pub_debug_output_, false);
  private_nh_.param("pub_map_odom_transform", p_pub_map_odom_transform_,true);
  private_nh_.param("pub_odometry", p_pub_odometry_,false);
  private_nh_.param("advertise_map_service", p_advertise_map_service_,true);
```

```
  private_nh_.param("scan_subscriber_queue_size", p_scan_subscriber_queue_size_, 5);

  private_nh_.param("map_resolution", p_map_resolution_, 0.025);
  private_nh_.param("map_size", p_map_size_, 1024);
  private_nh_.param("map_start_x", p_map_start_x_, 0.5);
  private_nh_.param("map_start_y", p_map_start_y_, 0.5);
  private_nh_.param("map_multi_res_levels", p_map_multi_res_levels_, 3);

  private_nh_.param("update_factor_free", p_update_factor_free_, 0.4);
  private_nh_.param("update_factor_occupied", p_update_factor_occupied_, 0.9);

  private_nh_.param("map_update_distance_thresh", p_map_update_distance_threshold_, 0.4);
  private_nh_.param("map_update_angle_thresh", p_map_update_angle_threshold_, 0.9);

  private_nh_.param("scan_topic", p_scan_topic_, std::string("scan"));
  private_nh_.param("sys_msg_topic", p_sys_msg_topic_, std::string("syscommand"));
  private_nh_.param("pose_update_topic", p_pose_update_topic_, std::string("poseupdate"));

  private_nh_.param("use_tf_scan_transformation", p_use_tf_scan_transformation_,true);
  private_nh_.param("use_tf_pose_start_estimate", p_use_tf_pose_start_estimate_,false);
  private_nh_.param("map_with_known_poses", p_map_with_known_poses_, false);

  private_nh_.param("base_frame", p_base_frame_, std::string("base_link"));
  private_nh_.param("map_frame", p_map_frame_, std::string("map"));
  private_nh_.param("odom_frame", p_odom_frame_, std::string("odom"));

  private_nh_.param("pub_map_scanmatch_transform", p_pub_map_scanmatch_transform_,true);
  private_nh_.param("tf_map_scanmatch_transform_frame_name", p_tf_map_scanmatch_transform_frame_name_,
std::string("scanmatcher_frame"));

  private_nh_.param("output_timing", p_timing_output_,false);

  private_nh_.param("map_pub_period", p_map_pub_period_, 2.0);

  double tmp = 0.0;
  private_nh_.param("laser_min_dist", tmp, 0.4);
  p_sqr_laser_min_dist_ = static_cast<float>(tmp*tmp);

  private_nh_.param("laser_max_dist", tmp, 30.0);
  p_sqr_laser_max_dist_ = static_cast<float>(tmp*tmp);

  private_nh_.param("laser_z_min_value", tmp, -1.0);
  p_laser_z_min_value_ = static_cast<float>(tmp);

  private_nh_.param("laser_z_max_value", tmp, 1.0);
  p_laser_z_max_value_ = static_cast<float>(tmp);

  if (p_pub_drawings)
```

```
  {
    ROS_INFO("HectorSM publishing debug drawings");
    hectorDrawings = new HectorDrawings();
  }

  if(p_pub_debug_output_)
  {
    ROS_INFO("HectorSM publishing debug info");
    debugInfoProvider = new HectorDebugInfoProvider();
  }

  if(p_pub_odometry_)
  {
    odometryPublisher_ = node_.advertise<nav_msgs::Odometry>("scanmatch_odom", 50);
  }

  slamProcessor = new hectorslam::HectorSlamProcessor(static_cast<float>(p_map_resolution_), p_map_size_,
p_map_size_, Eigen::Vector2f(p_map_start_x_, p_map_start_y_), p_map_multi_res_levels_, hectorDrawings,
debugInfoProvider);
  slamProcessor->setUpdateFactorFree(p_update_factor_free_);
  slamProcessor->setUpdateFactorOccupied(p_update_factor_occupied_);
  slamProcessor->setMapUpdateMinDistDiff(p_map_update_distance_threshold_);
  slamProcessor->setMapUpdateMinAngleDiff(p_map_update_angle_threshold_);

  int mapLevels = slamProcessor->getMapLevels();
  mapLevels = 1;

  for (int i = 0; i < mapLevels; ++i)
  {
    mapPubContainer.push_back(MapPublisherContainer());
    slamProcessor->addMapMutex(i, new HectorMapMutex());

    std::string mapTopicStr(mapTopic_);

    if (i != 0)
    {
      mapTopicStr.append("_" + boost::lexical_cast<std::string>(i));
    }

    std::string mapMetaTopicStr(mapTopicStr);
    mapMetaTopicStr.append("_metadata");

    MapPublisherContainer& tmp = mapPubContainer[i];
    tmp.mapPublisher_ = node_.advertise<nav_msgs::OccupancyGrid>(mapTopicStr, 1, true);
    tmp.mapMetadataPublisher_ = node_.advertise<nav_msgs::MapMetaData>(mapMetaTopicStr, 1, true);

    if ( (i == 0) && p_advertise_map_service_)
    {
```

```
        tmp.dynamicMapServiceServer_ = node_.advertiseService("dynamic_map",
&HectorMappingRos::mapCallback, this);
      }

      setServiceGetMapData(tmp.map_, slamProcessor->getGridMap(i));

      if ( i== 0){
        mapPubContainer[i].mapMetadataPublisher_.publish(mapPubContainer[i].map_.map.info);
      }
    }

    ROS_INFO("HectorSM p_base_frame_: %s", p_base_frame_.c_str());
    ROS_INFO("HectorSM p_map_frame_: %s", p_map_frame_.c_str());
    ROS_INFO("HectorSM p_odom_frame_: %s", p_odom_frame_.c_str());
    ROS_INFO("HectorSM p_scan_topic_: %s", p_scan_topic_.c_str());
    ROS_INFO("HectorSM p_use_tf_scan_transformation_: %s", p_use_tf_scan_transformation_ ? ("true") :
("false"));
    ROS_INFO("HectorSM p_pub_map_odom_transform_: %s", p_pub_map_odom_transform_ ? ("true") :
("false"));
    ROS_INFO("HectorSM p_scan_subscriber_queue_size_: %d", p_scan_subscriber_queue_size_);
    ROS_INFO("HectorSM p_map_pub_period_: %f", p_map_pub_period_);
    ROS_INFO("HectorSM p_update_factor_free_: %f", p_update_factor_free_);
    ROS_INFO("HectorSM p_update_factor_occupied_: %f", p_update_factor_occupied_);
    ROS_INFO("HectorSM p_map_update_distance_threshold_: %f ", p_map_update_distance_threshold_);
    ROS_INFO("HectorSM p_map_update_angle_threshold_: %f", p_map_update_angle_threshold_);
    ROS_INFO("HectorSM p_laser_z_min_value_: %f", p_laser_z_min_value_);
    ROS_INFO("HectorSM p_laser_z_max_value_: %f", p_laser_z_max_value_);

    scanSubscriber_ = node_.subscribe(p_scan_topic_, p_scan_subscriber_queue_size_,
&HectorMappingRos::scanCallback, this);
    sysMsgSubscriber_ = node_.subscribe(p_sys_msg_topic_, 2, &HectorMappingRos::sysMsgCallback, this);

    poseUpdatePublisher_ = node_.advertise<geometry_msgs::PoseWithCovarianceStamped>(p_pose_update_topic_,
1, false);
    posePublisher_ = node_.advertise<geometry_msgs::PoseStamped>("slam_out_pose", 1, false);

    scan_point_cloud_publisher_ = node_.advertise<sensor_msgs::PointCloud>("_cloud",1,false);

    tfB_ = new tf::TransformBroadcaster();
    ROS_ASSERT(tfB_);

    /*
    bool p_use_static_map_ = false;

    if (p_use_static_map_){
      mapSubscriber_ = node_.subscribe(mapTopic_, 1, &HectorMappingRos::staticMapCallback, this);
    }
    */
```

```
  initial_pose_sub_ = new message_filters::Subscriber<geometry_msgs::PoseWithCovarianceStamped>(node_,
"initialpose", 2);
  initial_pose_filter_ = new tf::MessageFilter<geometry_msgs::PoseWithCovarianceStamped>(*initial_pose_sub_,
tf_, p_map_frame_, 2);
  initial_pose_filter_->registerCallback(boost::bind(&HectorMappingRos::initialPoseCallback, this, _1));


  map__publish_thread_ = new boost::thread(boost::bind(&HectorMappingRos::publishMapLoop, this,
p_map_pub_period_));

  map_to_odom_.setIdentity();

  lastMapPublishTime = ros::Time(0,0);
}

HectorMappingRos::~HectorMappingRos()
{
  delete slamProcessor;

  if (hectorDrawings)
    delete hectorDrawings;

  if (debugInfoProvider)
    delete debugInfoProvider;

  if (tfB_)
    delete tfB_;

  if(map__publish_thread_)
    delete map__publish_thread_;
}

void HectorMappingRos::scanCallback(const sensor_msgs::LaserScan& scan)
{
  if (hectorDrawings)
  {
    hectorDrawings->setTime(scan.header.stamp);
  }

  ros::WallTime startTime = ros::WallTime::now();

  if (!p_use_tf_scan_transformation_)
  {
    if (rosLaserScanToDataContainer(scan, laserScanContainer,slamProcessor->getScaleToMap()))
    {
      slamProcessor->update(laserScanContainer,slamProcessor->getLastScanMatchPose());
    }
```

```
  }
  else
  {
    ros::Duration dur (0.5);

    if (tf_.waitForTransform(p_base_frame_,scan.header.frame_id, scan.header.stamp,dur))
    {
      tf::StampedTransform laserTransform;
      tf_.lookupTransform(p_base_frame_,scan.header.frame_id, scan.header.stamp, laserTransform);

      //projector_.transformLaserScanToPointCloud(p_base_frame_ ,scan, pointCloud,tf_);
      projector_.projectLaser(scan, laser_point_cloud_,30.0);

      if (scan_point_cloud_publisher_.getNumSubscribers() > 0){
        scan_point_cloud_publisher_.publish(laser_point_cloud_);
      }

      Eigen::Vector3f startEstimate(Eigen::Vector3f::Zero());

      if(rosPointCloudToDataContainer(laser_point_cloud_, laserTransform, laserScanContainer, slamProcessor-
>getScaleToMap()))
      {
        if (initial_pose_set_){
          initial_pose_set_ = false;
          startEstimate = initial_pose_;
        }else if (p_use_tf_pose_start_estimate_){

          try
          {
            tf::StampedTransform stamped_pose;

            tf_.waitForTransform(p_map_frame_,p_base_frame_, scan.header.stamp, ros::Duration(0.5));
            tf_.lookupTransform(p_map_frame_, p_base_frame_,    scan.header.stamp, stamped_pose);

            tfScalar yaw, pitch, roll;
            stamped_pose.getBasis().getEulerYPR(yaw, pitch, roll);

            startEstimate = Eigen::Vector3f(stamped_pose.getOrigin().getX(),stamped_pose.getOrigin().getY(),
yaw);
          }
          catch(tf::TransformException e)
          {
            ROS_ERROR("Transform from %s to %s failed\n", p_map_frame_.c_str(), p_base_frame_.c_str());
            startEstimate = slamProcessor->getLastScanMatchPose();
          }

        }else{
          startEstimate = slamProcessor->getLastScanMatchPose();
```

```
          }


          if (p_map_with_known_poses_){
            slamProcessor->update(laserScanContainer, startEstimate, true);
          }else{
            slamProcessor->update(laserScanContainer, startEstimate);
          }
        }

      }else{
        ROS_INFO("lookupTransform %s to %s timed out. Could not transform laser scan into base_frame.",
p_base_frame_.c_str(), scan.header.frame_id.c_str());
        return;
      }
    }

    if (p_timing_output_)
    {
      ros::WallDuration duration = ros::WallTime::now() - startTime;
      ROS_INFO("HectorSLAM Iter took: %f milliseconds", duration.toSec()*1000.0f );
    }

    //If we're just building a map with known poses, we're finished now. Code below this point publishes the
localization results.
    if (p_map_with_known_poses_)
    {
      return;
    }

    poseInfoContainer_.update(slamProcessor->getLastScanMatchPose(), slamProcessor-
>getLastScanMatchCovariance(), scan.header.stamp, p_map_frame_);

    poseUpdatePublisher_.publish(poseInfoContainer_.getPoseWithCovarianceStamped());
    posePublisher_.publish(poseInfoContainer_.getPoseStamped());

    if(p_pub_odometry_)
    {
      nav_msgs::Odometry tmp;
      tmp.pose = poseInfoContainer_.getPoseWithCovarianceStamped().pose;

      tmp.header = poseInfoContainer_.getPoseWithCovarianceStamped().header;
      odometryPublisher_.publish(tmp);
    }

    if (p_pub_map_odom_transform_)
    {
      tf::StampedTransform odom_to_base;
```

```
    try
    {
      tf_.waitForTransform(p_odom_frame_, p_base_frame_, scan.header.stamp, ros::Duration(0.5));
      tf_.lookupTransform(p_odom_frame_, p_base_frame_, scan.header.stamp, odom_to_base);
    }
    catch(tf::TransformException e)
    {
      ROS_ERROR("Transform failed during publishing of map_odom transform: %s",e.what());
      odom_to_base.setIdentity();
    }
    map_to_odom_ = tf::Transform(poseInfoContainer_.getTfTransform() * odom_to_base.inverse());
    tfB_->sendTransform( tf::StampedTransform (map_to_odom_, scan.header.stamp, p_map_frame_,
p_odom_frame_));
  }

  if (p_pub_map_scanmatch_transform_){
    tfB_->sendTransform( tf::StampedTransform(poseInfoContainer_.getTfTransform(), scan.header.stamp,
p_map_frame_, p_tf_map_scanmatch_transform_frame_name_));
  }
}

void HectorMappingRos::sysMsgCallback(const std_msgs::String& string)
{
  ROS_INFO("HectorSM sysMsgCallback, msg contents: %s", string.data.c_str());

  if (string.data == "reset")
  {
    ROS_INFO("HectorSM reset");
    slamProcessor->reset();
  }
}

bool HectorMappingRos::mapCallback(nav_msgs::GetMap::Request  &req,
                                   nav_msgs::GetMap::Response &res)
{
  ROS_INFO("HectorSM Map service called");
  res = mapPubContainer[0].map_;
  return true;
}

void HectorMappingRos::publishMap(MapPublisherContainer& mapPublisher, const hectorslam::GridMap&
gridMap, ros::Time timestamp, MapLockerInterface* mapMutex)
{
  nav_msgs::GetMap::Response& map_ (mapPublisher.map_);

  //only update map if it changed
  if (lastGetMapUpdateIndex != gridMap.getUpdateIndex())
```

```
  {

    int sizeX = gridMap.getSizeX();
    int sizeY = gridMap.getSizeY();

    int size = sizeX * sizeY;

    std::vector<int8_t>& data = map_.map.data;

    //std::vector contents are guaranteed to be contiguous, use memset to set all to unknown to save time in loop
    memset(&data[0], -1, sizeof(int8_t) * size);

    if (mapMutex)
    {
      mapMutex->lockMap();
    }

    for(int i=0; i < size; ++i)
    {
      if(gridMap.isFree(i))
      {
        data[i] = 0;
      }
      else if (gridMap.isOccupied(i))
      {
        data[i] = 100;
      }
    }

    lastGetMapUpdateIndex = gridMap.getUpdateIndex();

    if (mapMutex)
    {
      mapMutex->unlockMap();
    }
  }

  map_.map.header.stamp = timestamp;

  mapPublisher.mapPublisher_.publish(map_.map);
}

bool HectorMappingRos::rosLaserScanToDataContainer(const sensor_msgs::LaserScan& scan,
hectorslam::DataContainer& dataContainer, float scaleToMap)
{
  size_t size = scan.ranges.size();

  float angle = scan.angle_min;
```

```
  dataContainer.clear();

  dataContainer.setOrigo(Eigen::Vector2f::Zero());

  float maxRangeForContainer = scan.range_max - 0.1f;

  for (size_t i = 0; i < size; ++i)
  {
    float dist = scan.ranges[i];

    if ( (dist > scan.range_min) && (dist < maxRangeForContainer))
    {
      dist *= scaleToMap;
      dataContainer.add(Eigen::Vector2f(cos(angle) * dist, sin(angle) * dist));
    }

    angle += scan.angle_increment;
  }

  return true;
}

bool HectorMappingRos::rosPointCloudToDataContainer(const sensor_msgs::PointCloud& pointCloud, const
tf::StampedTransform& laserTransform, hectorslam::DataContainer& dataContainer, float scaleToMap)
{
  size_t size = pointCloud.points.size();
  //ROS_INFO("size: %d", size);

  dataContainer.clear();

  tf::Vector3 laserPos (laserTransform.getOrigin());
  dataContainer.setOrigo(Eigen::Vector2f(laserPos.x(), laserPos.y())*scaleToMap);

  for (size_t i = 0; i < size; ++i)
  {

    const geometry_msgs::Point32& currPoint(pointCloud.points[i]);

    float dist_sqr = currPoint.x*currPoint.x + currPoint.y* currPoint.y;

    if ( (dist_sqr > p_sqr_laser_min_dist_) && (dist_sqr < p_sqr_laser_max_dist_) ){

      if ( (currPoint.x < 0.0f) && (dist_sqr < 0.50f)){
        continue;
      }

      tf::Vector3 pointPosBaseFrame(laserTransform * tf::Vector3(currPoint.x, currPoint.y, currPoint.z));
```

```
        float pointPosLaserFrameZ = pointPosBaseFrame.z() - laserPos.z();

        if (pointPosLaserFrameZ > p_laser_z_min_value_ && pointPosLaserFrameZ < p_laser_z_max_value_)
        {
          dataContainer.add(Eigen::Vector2f(pointPosBaseFrame.x(),pointPosBaseFrame.y())*scaleToMap);
        }
      }
    }

    return true;
}

void HectorMappingRos::setServiceGetMapData(nav_msgs::GetMap::Response& map_, const
hectorslam::GridMap& gridMap)
{
    Eigen::Vector2f mapOrigin (gridMap.getWorldCoords(Eigen::Vector2f::Zero()));
    mapOrigin.array() -= gridMap.getCellLength()*0.5f;

    map_.map.info.origin.position.x = mapOrigin.x();
    map_.map.info.origin.position.y = mapOrigin.y();
    map_.map.info.origin.orientation.w = 1.0;

    map_.map.info.resolution = gridMap.getCellLength();

    map_.map.info.width = gridMap.getSizeX();
    map_.map.info.height = gridMap.getSizeY();

    map_.map.header.frame_id = p_map_frame_;
    map_.map.data.resize(map_.map.info.width * map_.map.info.height);
}

/*
void HectorMappingRos::setStaticMapData(const nav_msgs::OccupancyGrid& map)
{
    float cell_length = map.info.resolution;
    Eigen::Vector2f mapOrigin (map.info.origin.position.x + cell_length*0.5f,
                               map.info.origin.position.y + cell_length*0.5f);

    int map_size_x = map.info.width;
    int map_size_y = map.info.height;

    slamProcessor = new hectorslam::HectorSlamProcessor(cell_length, map_size_x, map_size_y,
Eigen::Vector2f(0.0f, 0.0f), 1, hectorDrawings, debugInfoProvider);
}
*/
```

```
void HectorMappingRos::publishMapLoop(double map_pub_period)
{
  ros::Rate r(1.0 / map_pub_period);
  while(ros::ok())
  {
    //ros::WallTime t1 = ros::WallTime::now();
    ros::Time mapTime (ros::Time::now());
    //publishMap(mapPubContainer[2],slamProcessor->getGridMap(2), mapTime);
    //publishMap(mapPubContainer[1],slamProcessor->getGridMap(1), mapTime);
    publishMap(mapPubContainer[0],slamProcessor->getGridMap(0), mapTime, slamProcessor->getMapMutex(0));

    //ros::WallDuration t2 = ros::WallTime::now() - t1;

    //std::cout << "time s: " << t2.toSec();
    //ROS_INFO("HectorSM ms: %4.2f", t2.toSec()*1000.0f);

    r.sleep();
  }
}

void HectorMappingRos::staticMapCallback(const nav_msgs::OccupancyGrid& map)
{

}

void HectorMappingRos::initialPoseCallback(const geometry_msgs::PoseWithCovarianceStampedConstPtr& msg)
{
  initial_pose_set_ = true;

  tf::Pose pose;
  tf::poseMsgToTF(msg->pose.pose, pose);
  initial_pose_ = Eigen::Vector3f(msg->pose.pose.position.x, msg->pose.pose.position.y, tf::getYaw(pose.getRotation()));
  ROS_INFO("Setting initial pose with world coords x: %f y: %f yaw: %f", initial_pose_[0], initial_pose_[1], initial_pose_[2]);
}
```