APROXIMACIÓN AL DESPLAZAMIENTO DE UNA PLATAFORMA MÓVIL, TIPO ROBOT AGRÍCOLA

DANIEL SANTIAGO RONDÓN CÁRDENAS

UNIVERSIDAD MILITAR NUEVA GRANADA

FACULTAD DE INGENIERÍA

PROGRAMA EN INGENIERÍA EN MECATRÓNICA

BOGOTÁ D.C.

2018

APROXIMACIÓN AL DESPLAZAMIENTO DE UNA PLATAFORMA MÓVIL, TIPO ROBOT AGRÍCOLA

DANIEL SANTIAGO RONDÓN CÁRDENAS

TRABAJO DE GRADO PARA OPTAR AL TÍTULO DE INGENIERO MECATRÓNICO

DIRECTOR

ING. LEONARDO ENRIQUE SOLAQUE GUZMAN, PhD.

UNIVERSIDAD MILITAR NUEVA GRANADA

FACULTAD DE INGENIERÍA

PROGRAMA EN INGENIERÍA EN MECATRÓNICA

BOGOTÁ D.C.

2018

ACEPTACIÓN

Nota De Aceptación

_____

_____

_____

_____

_____

_____

_____

Firma del Presidente de los Jurados

_____

Firma Jurado

_____

Firma Jurado

_____

DEDICATORIA

A mis padres y hermanos;

Por su amor y apoyo incondicional,

Por su carácter y templanza,

Por su ejemplo y sabias enseñanzas día a día.

AGRADECIMIENTOS

Primeramente, a Dios y a la Vida, por la infinidad de oportunidades que me han brindado, por cada reto, por cada logro y por la fuerza para levantarme cuando las cosas no salen bien.

Agradezco a mis padres Elena y Daniel, su formación, ella ha hecho de mi lo que hoy en día soy, sus consejos, pues me ayudan a mejorar cada día, y su paciencia, cuando en muchas ocasiones el tiempo me era insuficiente. Sin su apoyo este logro no sería posible. A mi hermano David, porque con su ejemplo he aprendido más de la vida que con mil libros. Por su madurez, fortaleza y sabias palabras en los momentos duros, y por compartir mi felicidad como suya. A mi hermano Ángel, porque su sinceridad e ingenuidad me motivan a ser una mejor persona y el mejor ejemplo posible para él.

A mi tutor de tesis, Ing. Leonardo Solaque, por su guía, colaboración y por poner a disposición todos los recursos necesarios para la culminación de este proyecto. Al Ing. Guillermo Sánchez por su disponibilidad y por sus acertados comentarios que tantas veces nos sacaron de apuros. A Alejandro Salgado, por sus invaluables aportes a este proyecto. A mi tutor del semillero de investigación, Ing. Andrés Cifuentes, por su tiempo, su interés y por despertar la vocación de la investigación en mí. A los profesores y a los diferentes miembros de la Universidad Militar Nueva Granada, su exigencia y pauta serán las bases de mi vida profesional.

Al que fue por muchos años mi grupo de estudio, Sofia, Laudy y Paula, por cada una de esas desveladas, por cada corto, por cada jumper roto, por cada transistor quemado, por cada programa que no quería compilar, por su apoyo y sus consejos, pero sobre todo por su amistad. Gracias a ustedes y sus familias.

Índice general

Índice de Figuras

1 Introducción

En este capítulo se encontrará una muestra del estado del arte referente a robótica móvil e innovaciones en la agricultura. Así mismo se verá también el planteamiento del problema, los objetivos del proyecto y la justificación del mismo.

1.1 Estado del arte

La palabra ‘robot’ normalmente es asociada con humanoides o máquinas muy robustas solo usadas en la industria, sin embargo, hoy en día se encuentra un sinfín de mecanismos y sistemas que facilitan muchas de las tareas que se realizan a diario. ¿Será posible también llamar a estos ‘robots’? Según la RAE, un robot es “Máquina o ingenio electrónico programable, capaz de manipular objetos y realizar operaciones antes reservadas solo a las personas” [1]. Teniendo en cuenta esto, se podría decir que efectivamente, muchas de las máquinas que hoy en dia existen son robots, y que conforme ha ido evolucionando la tecnología, la robótica se ha hecho cada vez más parte de la vida diaria, pues hasta las labores más sencillas como limpiar los pisos tienen por lo menos un mínimo grado de automatización.

Para su estudio, es posible dividir a la robótica en 4 grandes áreas: manipulación mecánica, locomoción, visión computacional e inteligencia artificial [2], pero en el momento de la práctica la gran mayoria de los robots necesita una cuidadosa integración de todos estos campos, por lo que en este caso se podría dividir a los robots según su funcionalidad. De esta manera se llega hasta un tema bastante tratado en los últimos veinte años, la robótica móvil.

La robótica móvil nació con la necesidad de brindar mayor flexibilidad a los procesos de manufactura industrial. Con el paso de los años se encontró una nueva utilidad para estos, la investigación planetaria, esto llevo a grandes investigaciones y a la implementación de algoritmos de inteligencia artificial, con lo que se revolucionó esta industria [3].

Actualmente los robots móviles pueden clasificarse en guiados o no guiados [4]. El vehículo guiado esta limitado a trayectorias previamente definidas en un área de trabajo fija, estas son demasiadas limitantes para las necesidades existentes, por lo que la mayoria de investigaciones se han enfocado hacia robots más autonomos.

Existen muchas aplicaciones en las que el robot genera una trayectoria entre dos puntos determinados, pero usualmente no es posible proveer un mapa detallado del entorno completo de trabajo de un robot móvil y en adicion a ello los ambientes de trabajo suelen ser dinamicos, por lo que en estos casos el robot necesita crear un modelo geometrico completo de lo que sucede a su alrededor [5]. Para conseguir mapear de forma adecuada el robot debe ser capaz de orientarse dentro del mapa

que esta creando, de manera que requiere una combinacion estrategica de sensores. Los más comunmente usados son: camaras estereoscopicas [6], monoscopicas y de profundidad, lasers [7], centrales inerciales y sonares.

Es de resaltar que cuando se habla de navegación en robótica móvil, se debe tener en cuenta que, en condiciones iniciales, la localización del robot en cuanto a su entorno de trabajo es desconocida, y que, por lo general, la información del entorno de trabajo es incompleta, aproximada o inexacta [8]. Por lo que técnicas habituales para control de trayectorias suelen no ser aplicables.

En una investigación realizada en la Universidad Católica de Chile [9] se hizo una comparación entre varios métodos de control y guiado para robot diferenciales, la cual concluyo que un control difuso ofrece relevantes ventajas sobre un controlador clásico como un PID, entre las cuales resaltan los tiempos de respuesta, el consumo de energía y la simplicidad en su implementación.

En investigaciones similares a la de la Universidad Católica [10] [11] [12] se comparan diferentes tipos de controladores inteligentes, como lo son redes neuronales, controles difusos y neuro difusos, en aplicaciones móviles y de evasión de obstáculos. Concluyen que con un control difuso se consigue resultados bastante buenos en tareas simples, aunque no se tenga una base de reglas muy extensa. Por otra parte, la ventaja más significativa que presenta una red neuronal es su capacidad de aprendizaje, sin embargo, es imposible extraer conocimiento de este método.

Una muestra de la sencillez al momento de implementar un control difuso es el desarrollo llevado a cabo en la universidad de Pereira [8], en el cual se ubicaban 8 sensores de ultrasonido sobre un robot móvil dispuestos en 3 grupos, uno adelante y uno a cada lado. Los datos adquiridos por cada grupo son promediados, y enviados a una base de reglas que establecen el comportamiento adecuado dependiendo si el robot tiene un objeto próximo a alguno de los grupos. De esta forma se logra que el robot evada obstáculos con una exactitud y velocidad aceptables, sin necesidad de conocer su dinámica, ni implementar complejas ecuaciones para su control. Adicionalmente este estudio destaca las facilidades de trabajar con el Toolbox de lógica difusa de Matlab, sin tener en cuenta las dificultades en la velocidad de procesamiento y rendimiento de máquina.

Existen diversos métodos de entrada para los sistemas inteligentes, uno de los que más se ha tratado en los últimos años son los landmakrs. Estos consisten en puntos distintivos dentro de un mapa que sirven al robot para establecer una referencia y ubicarse sobre los trazos que está generando, permitiendo así que el problema de la navegación sea traducible a lenguaje computacional.

Al hablar de Landmarks es necesario validar la correcta ubicación de estos. Para determinar ello primero se debe establecer la cantidad correcta de puntos de

referencia y luego donde debe estar cada uno de ellos para que el robot sea capaz de desarrollar la tarea [13].

Una vez determinados los puntos de referencia es posible establecer una trayectoria en medio de estos. Si esta tarea no puede ser realizada es evidencia de que hace falta un nuevo punto de referencia.

El principal uso de los Landmarks es reducir la incerteza que se puede generar a lo largo de una región.

Los Landmarks se deben diseñar de forma tal que cumplan 3 requisitos:

- Detección: El robot debe ser capaz de determinar rápidamente si colocar o no un landmark en su área de visión.
- Localización: El robot debe ser capaz de establecer su posición con respecto a cualquier landmark
- Reconocimiento: El robot debe ser capaz de diferenciar un landmark de otro [13].

Por otro lado, si se habla del campo de la agricultura, se encontrara que los desarrollos tecnologicos se han dado muy lentamente en comparación con otras industrias. Una posible explicación para ello es que en la industria agrícola muchas de las maquinarias son usadas durante una temporada específica del año, podría decirse que incluso existen algunas que solo se utilizan unas cuantas horas al año. Por ejemplo, un utensilio para plantar solo es usado en el momento de plantar, se debe esperar a una nueva cosecha para volver a utilizarlo, y si se tiene en cuenta que muchas de las cosechas no se dan durante todo el año, el tiempo que está en uso un utensilio de este tipo es mínimo, por lo que un inversionista cuestionaría si es realmente necesario y sobre todo útil invertir dinero en la tecnificación de ese tipo de procesos [14].

Hasta el siglo XVIII la mayoría de innovaciones eran mínimas, eran generadas por los mismos granjeros y de uso manual, es decir sin ningún tipo de automatización. Para mediados del siglo XIX, las labores de fuerza empezaron a ser desarrolladas por animales, y luego para finales de este mismo siglo, con la revolución industrial, las máquinas a vapor se hicieron populares y reemplazaron en gran parte a la tracción animal.

El siguiente adelanto se da hacia 1930, cuando las máquinas empezaron a hacerse más grandes para dar paso a la agricultura a gran escala [15]. Sin embargo, hasta ese momento los agricultores seguían siendo la única fuente de inteligencia para controlar estas máquinas.

Con la revolución tecnológica del nuevo milenio fueron muchos los avances logrados, algunas de las potencias en tecnología a nivel mundial han dedicado

esfuerzos a la robótica agrícola. Muchos modelos experimentales han sido desarrollados y unos cuantos comercializados. Por ejemplo, la máquina para clasificación de cítricos desarrollada por Sunkist Corporation [16], que tiene la capacidad de clasificar la fruta dependiendo de su tamaño, color, manchas y magulladuras, y puede clasificar 480 frutas por minuto.

Sistemas de visión también han sido extensamente trabajados en el procesamiento de frutas y comestibles en general. Entre los desarrollos más destacados se encuentran un detector de yemas rotas en un quiebra-huevos comercial, un sistema de remoción de defectos en papas fritas, un inspector de cortes en chocolates y una máquina para detectar defectos en la corteza de la pizza [17].

Otra de las áreas en las que se ha buscado innovar es en la velocidad en la que se desarrollan las tareas del agro. El departamento para ingeniería de la agricultura de Louisiana desarrollo un modelo experimental para realizar trasplantes [18]. La máquina consiste en un brazo robótico de 5 grados de libertad y un gripper, que está montado sobre un trasplantador comercial modificado y es controlado por un micro computador. El sistema no usa ningún tipo de sensor, esta pre programado con las coordenadas de cada planta. Es capaz de trasplantar 6 plantas por minutos, lo que es una quinta parte del tiempo que le tomaría a personal experimentado desempeñar esta tarea.

En Japón se han hecho importantes desarrollos, entre ellos resaltan una máquina capaz de recoger los frutos de un árbol utilizando una cámara de video y un brazo robot [19], un par de rociadores de productos químicos peligrosos, sin conductor; uno guiado utilizando foto sensores para detectar la presencia de troncos y otro que se guía utilizando tubos de PVC previamente colocados en el suelo a lo largo del camino deseado [20].

1.2 Planteamiento del problema

Con la creciente cantidad de nuevos productos, y la alta demanda de los ya existentes, es imposible concebir el mundo actual sin automatización. Sin embargo, esta no se ha desarrollado de la misma manera en todas las áreas, existen sectores de la industria, como lo es la agricultura, en donde la tecnificación parece haber aminorado su ritmo de desarrollo, pues los métodos y mecanismos continúan dependiendo casi por completo de la labor humana, siendo muy similares a los utilizados hace muchos años.

La falta de innovación hace que los costos de producción sean cada vez más altos y las ganancias más bajas, lo que representa un gran problema para países como Colombia que basan su economía en productos agrícolas.

Con el fin de dar solución a esta problemática es necesario crear nuevas líneas de investigación en las cuales se involucren temas como la robótica móvil y la inteligencia artificial con la industria agrícola, de forma que a corto y mediano plazo se creen soluciones más autónomas y capaces de ser implementadas a gran escala en la industria.

Uno de los principales retos al momento de dar mayor autonomía a una máquina es el hecho de que la misma se tenga que desplazar en un entorno desconocido, lo que lleva a preguntarse ¿Cuáles son los requisitos y las estrategias usadas para llevar a la autonomía una plataforma dedicada a labores de agricultura? ¿Cuáles son los algoritmos que permiten la integración de sensores instalados para el desplazamiento de la plataforma en ambiente poco conocidos y estructurados? ¿Es posible guiar a un robot en un entorno completamente desconocido?

Para dar respuesta a estas preguntas se plantea un proyecto con los objetivos que se presentan a continuación.

1.3 Objetivos

Objetivo General

Integrar un algoritmo de localización y guiado a una plataforma móvil terrestre, orientada a labores de agricultura dentro de un entorno dinámico

Objetivos Específicos

- Estudiar técnicas de guiado en plataformas móviles y seleccionar una.
- Simular en un entorno virtual el guiado, haciendo uso del algoritmo seleccionado
- Integrar el algoritmo a una plataforma móvil terrestre, utilizando ROS para programar en el prototipo experimental.
- Validar el funcionamiento y concluir sobre el desempeño de este.

1.4 Justificación

Actualmente la sociedad tiene una fuerte tendencia hacia la automatización de todos los procesos, y se ve en la evidente necesidad de renovarse día a día en todas las áreas del conocimiento.

Una de las áreas en las que quizá menos se ha innovado en las últimas décadas es la agricultura. Para la época de la revolución industrial en 1820 se generaron grandes cambios en la forma en la que los productos eran recogidos, pero desde entonces no ha habido grandes modificaciones que se implementen en la mayor parte de la industria, por los países que tienen una economía basada en la agricultura, como Colombia, se han visto en desventaja. Por lo anterior, se hace necesario generar nuevo conocimiento y nuevos métodos con mayor automatización que puedan suplir las necesidades de innovación en esta área y maximizar la utilidad de la mano de obra humana.

Adicional a ello, productos insignia de Colombia como el café presentan mayores dificultades para la automatización, pues no todos los granos están a la misma altura ni se recogen al mismo tiempo, eso sin tener en cuenta, además, la variabilidad espacial de las propiedades del suelo, las cuales dificultan los movimientos del robot y la selección de espacios para la siembra.

Una de las formas de atacar este problema podría darse a través de la robótica móvil, brindando a los mecanismos de recolección con mayor autonomía y la posibilidad de programación en siembra y vigilancia de cultivos de forma tal que se minimicen las pérdidas a causa de plagas.

Actualmente existen diversas formas de aumentar la autonomía de un sistema, pero en el caso de los robots móviles una de las más usadas es el SLAM, pues brinda al sistema la posibilidad de ubicarse y desplazarse en un entorno desconocido. Si bien los sistemas de SLAM tienen un alto costo computacional debido a la gran de información que tienen que procesar, ya existen soluciones de optimización para que estos modelos puedan ser implementables en tiempo real.

Son muchos y muy distintos los sensores que se pueden llegar a utilizar en la metodología SLAM, los más utilizados son sensores laser, cámaras, IMUs y GPS, ya que se complementan muy bien unos con otros; lo láseres brindan información de presencia de objetos, bien sea cercanos o lejanos, las cámaras, si se utilizan como estéreo visión, dan mayor detalle del ambiente en general, mientras que las IMU y el GPS proporcionan información detallada de la ubicación y desplazamientos que realiza el sistema.

Si a este método se le suma además un sensor que permita obtener información acerca de la variabilidad del suelo se obtendría un sistema muy completo capaz de desempeñar labores semi autónomas y autónomas en el sector agrícola (como siembra, supervisión, recolección, fumigación, entre otros), permitiendo así una potencialización de los sistemas de producción en masa, disminuir el esfuerzo humano, eliminación de riegos tanto para quien trabaja, como para los mismos cultivos, lo que mejorará la calidad de los productos y con ello las ganancias netas.

Para que todo lo anterior sea posible es necesario que el sistema a generar sea muy robusto y adaptable a diferentes ambientes. Tomando el desplazamiento

coordinado dentro de un ambiente dinámico como la parte más importante de los de los requerimientos del sistema, se propone la implementación del algoritmo en un entorno simulado. Este ha de ser modular, de forma tal que segmentos adicionales (como mecanismos para siembra o recolección) puedan ser fácilmente implementados en el futuro.

2 Marco Teórico

En este capítulo se encontrará las referencias teóricas de todos los temas utilizados a lo largo del proyecto. Métodos, conceptos y referencias serán explicados en detalle.

2.1 Robótica Móvil

Con la necesidad de extender la robótica a diferentes campos y aumentar su autonomía, limitando la intervención humana, nació una rama hoy conocida como robótica móvil [21].

Para conseguir dar autonomía a un robot se deben involucrar actividades de planificación, percepción y control. La planificación puede descomponerse en dos etapas, una global, en donde se traza una trayectoria general desde un punto de partida hasta un punto de llegada, y una local, que busca evadir obstáculos no esperados e incluso dinámicos. Para la determinación de dichos obstáculos es necesario relacionar distintas percepciones, expresadas a través de la lectura de diversos sensores. Y finalmente para completar la tarea del desplazamiento es necesario implementar técnicas de control tanto de velocidad como de posición [21].

La manera más fácil de clasificar a los robots móviles es dependiendo del tipo de locomoción dentro de un medio. En lo referente al medio existen 3 clasificaciones: Aéreo, Acuático y Terrestre, este trabajo se centrará en esta última. En general, existen 3 medios de movimiento dentro del medio terrestre: Por patas (Figura 1 a), por ruedas (Figura 1 b) y por orugas (Figura 1 c). De estas 3, los mayores desarrollos se han dado sobre los robots móviles con ruedas. Este hecho se atribuye a que, en comparación con las otras dos categorías, requieren un menor número de partes lo que los hace menos complejos, requieren menor gasto energético en superficies lisas y firmes, y no generan desgaste en las superficies en que se desplazan [22].

Figura 1 Robots Terrestres: (a) Con patas, (b) con ruedas, (c) con orugas. Imagen tomada de [23]

Se puede definir un robot móvil con ruedas como:

“Un sistema electromecánico controlado, que utiliza como locomoción ruedas de algún tipo, y que es capaz de trasladarse de forma autónoma a una meta preestablecida en un determinado espacio de trabajo.” [22].

Es posible clasificar a los robots móviles con ruedas según su configuración cinemática, como se ve en la Figura 2.

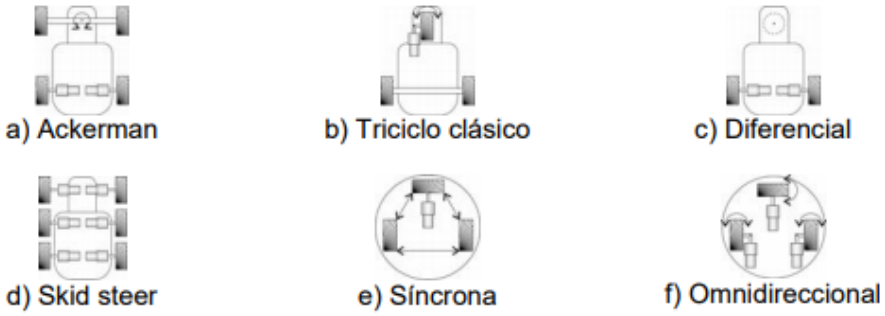


Figura 2 Configuraciones de los robots móviles con ruedas. Imagen tomada de [22]

Para este caso particular el estudio se enfocará en la configuración diferencial, debido principalmente a su facilidad en la implementación, a la simplicidad en su diseño y dinámica y su bajo costo.

En esta configuración la dirección del vehículo está dada por la diferencia de velocidades entre las ruedas laterales, de forma tal que si ambas van a la misma velocidad el robot se desplazará en línea recta, y si una de las dos se encuentra girando a mayor velocidad el vehículo se desplazará hacia el lado contrario. Adicionalmente existen una o más ruedas para dar soporte [21].

Para obtener una primera aproximación a lo que será la simulación previa a la puesta en marcha, se presenta el modelo matemático generalizado de un robot diferencial, generado en Matlab® y simulado utilizando Simulink® (Figura 3).

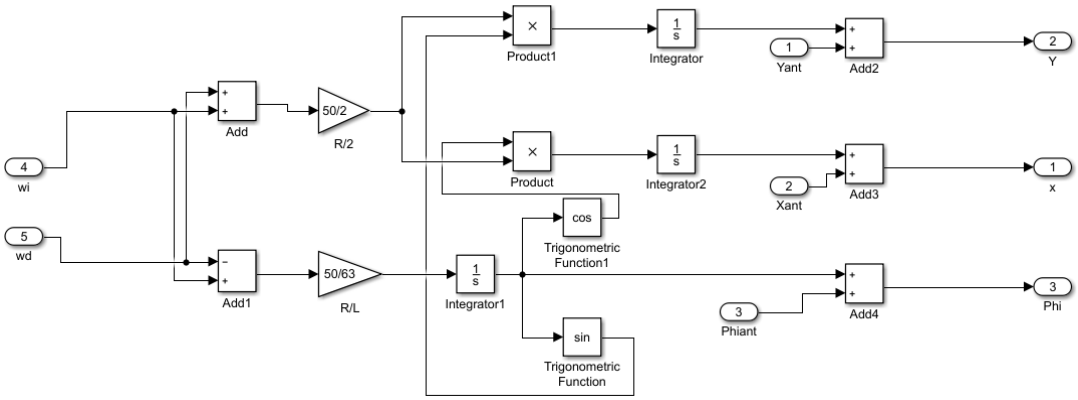


Figura 3 Modelo cinemático Robot Diferencial. Elaboración del Autor

De donde wi y wd son las velocidades angulares de cada rueda, las cuales pueden ser medidas fácilmente por encoders. R es el radio de las ruedas y L es la distancia entre las ruedas. De esto finalmente se obtiene un X y un Y globales, y un ángulo phi de dirección.

2.2 Fusión Sensórica

Dado que el proyecto pretende ser base para un robot agrícola de mayor magnitud y que se desempeñe en un entorno abierto, se llegó a la conclusión de que el rendimiento mejoraría considerablemente al añadir información satelital e inercial sobre la posición del robot.

Sin embargo, este tipo de datos tiende a acumular rápidamente errores por deslizamiento (drift), por lo que normalmente, en sistemas de localización se suelen utilizar sensores adicionales que entreguen medidas absolutas. Pero para que estas medidas adicionales tengan correlación es necesario llevar a cabo el proceso conocido como fusión sensórica [24].

Para realizar la fusión sensórica se suelen utilizar técnicas bayesianas, que estimen estadísticamente el estado del sistema (localización) a partir de las observaciones con ruido que entregan los sensores.

A grandes rasgos, existen 2 técnicas que permiten implementar inferencias bayesianas para este tipo de aplicaciones: Filtro de Kalman y Filtro de Partículas [24], siendo Kalman el más utilizado para resolver la problemática de localización en entornos locales [25], esto debido a que ofrece un menor costo computacional, para resultados similares [24].

El objetivo de la estimación bayesiana es construir una función de densidad de probabilidad de un vector de estado requerido, para este caso de estudio la posición de un robot. Dicha función puede ser traducida como la incerteza que se tiene sobre el vector de estado requerido [26].

2.2.1 Filtro de Kalman (KF)

Es un algoritmo que busca la estimación de estados, en tiempo real, a partir de medidas sensóricas ruidosas y la incertidumbre de estas. Para conseguir una correcta estimación no se utiliza solo el modelo del sistema, sino que también se usa el modelo de la variable medida. Esto permite realizar mejores suposiciones y acercarse más al vector de estado real en el siguiente instante de tiempo.

El algoritmo de Kalman puede descomponerse en dos etapas: una de predicción y una de actualización. En la etapa de predicción se busca obtener el estado siguiente a partir del estado anterior y de las ecuaciones dinámicas. Y en la etapa de actualización se busca corregir la predicción realizada. Esto se puede traducir en las siguientes ecuaciones.

Predicción:

$$\hat{x}_{\bar{k}} = \Phi \hat{x}_{\overline{k-1}} + \Gamma u_{\overline{k-1}}$$

Ecuación 1 Predicción del siguiente estado de Kalman

$$p_{\bar{k}} = \Phi p_{\overline{k-1}} \Phi^T + Q_k$$

Ecuación 2 Covarianza de Kalman

De donde $\hat{x}_{\bar{k}}$ es el estado del sistema en el instante k, Φ es la matriz de transición de estados, Γu es el modelo del ruido presente en el sistema y $p_{\bar{k}}$ es la covarianza en el instante k.

Actualización:

$$K_k = p_{\bar{k}} \mathrm{H}^T [\mathrm{H} p_{\bar{k}} \mathrm{H}^T + R_k]^{-1}$$

Ecuación 3 Ganancia de Kalman

$$\hat{x}_k^+ = \hat{x}_{\bar{k}} + K_k [\mathrm{z}_k - \mathrm{H} \hat{x}_{\bar{k}}]$$

Ecuación 4 Estimación de estados de Kalman

$$p_k^+ = [\mathrm{I} - K_k \mathrm{H}] p_{\bar{k}}$$

Ecuación 5 Actualización de la matriz de covarianzas de Kalman

De donde z_k es una medición del instante k, K_k es la ganancia de Kalman, $\hat{x}_k^+$ es el estado estimado, p_k^+ es la matriz de covarianzas actualizada y H es la matriz de observación.

Estas ecuaciones fueron inicialmente propuestas por Kalman en 1960 [27] para estimar el siguiente estado de un proceso dinámico cualquiera. Luego en 1970 Sorenson [28] utilizo el método de Kalman para integrar información a lo largo del tiempo.

Al aplicar el filtro de Kalman a la localización de vehículos lo que se pretende es estimar su posición dentro de un plano (X, Y) y su orientación θ. Para ello se utiliza una distribución de probabilidad normal. La incertidumbre de la posición estimada estará dada por la covarianza de la función de probabilidad antes mencionada. Cuando el robot efectúa cualquier movimiento la posición estimada se desplaza de acuerdo con la odometría, mientras que con los demás sensores se actualizará la

distribución de probabilidad de la localización, de forma tal que las nuevas lecturas serán cada vez más cercanas a la realidad [25].

Actualmente existen varias versiones del filtro de Kalman, entre las que resaltan EKF (Extended Kalman Filter) y Unscented Kalman Filter (UKF). Para este caso particular se utilizará el filtro extendido de Kalman.

La principal diferencia entre el KF y el EKF consiste en que el filtro de Kalman original fue diseñado para trabajar con sistemas lineales, pero en términos generales los sistemas en el mundo real no son lineales, por lo que la versión extendida busca una linealización sobre la media actual y la covarianza [29]. Sin embargo, la linealización de esta versión del filtro se hace mediante jacobianos, por lo que es posible que se generen singularidades debidas meramente al proceso de cálculo, por lo que los resultados finales pueden verse distorsionados. Por esto se puede afirmar que el filtro no es óptimo, incluso, en medio del proceso iterativo de linealizaciones puede llegar a divergir [30]. Y es de esta dificultad de donde nace el UKF. Este aplica una transformación conocida como UT (Unscented Transformation), que calcula deterministamente una serie de muestras transformadas con las que se estimará la esperanza y la covarianza de la variable aleatoria, evitando así el proceso de linealización con Jacobianos.

2.2.2 Filtros de Partículas

Cuando el problema a tratar resulta ser altamente no lineal y no Gaussiano, métodos como Kalman tienden a no proporcionar una estimación razonable. Los filtros de partículas (PF) son un método alternativo para este tipo de problemas [31].

Un filtro de partículas es una aproximación a un filtro bayesiano recursivo. En vez de describir una función de densidad de probabilidad, como lo haría un filtro bayesiano común, los filtros de partículas se aproximan a la función de densidad de probabilidad tomando muestras aleatorias. De acuerdo con Salmond y Gordon [26], para un número infinito de muestras aleatorias (partículas) el método será exactamente igual al método bayesiano tradicional.

Este método tiene 4 etapas: inicialización, predicción, actualización y re-muestreo.

Inicialización: Se genera un numero inicial de partículas

$$\{x_0{}^{(i)}\}_{i=1}^{N}$$

Ecuación 6 Inicialización de las partículas de PF

Predicción: Se propagan las partículas en el modelo dinámico

$$x_k^{(i)} = \mathrm{fk} - 1(x_{k-1}^{(i)})$$

Ecuación 7 Función de predicción PF

Actualización: Se actualizan los pesos de cada partícula de acuerdo con el error entre la predicción y la medición en el instante k

$$w_k^{(i)} = w_{k-1}^{(i)} \, \mathrm{p}(y_k - \mathrm{h}(x_k^{(i)}))$$

Ecuación 8 Función de actualización de PF

Re-muestreo: En esta etapa se eliminan las partículas de menor peso y se duplican las de mayor peso, creando un nuevo conjunto de partículas que podrán volver a la etapa de predicción [32].

A pesar de que su estructura es bastante simple, este tipo de filtros tienen un alto costo computacional. Además, según Doucett et al. [33], el muestreo en espacios de muchas dimensiones puede llegar a ser ineficiente.

2.2.2.1 Filtros de Partículas Rao-Blackwellized

Para solucionar el problema antes mencionado en cuanto al muestreo en espacios multidimensionales, es posible analizar el problema en dos partes: una de manera analítica y la otra por muestreo, y no todo puramente por muestreo como en los filtros de partículas clásicos. Este método es conocido como Rao-Blackwellized [34]. Esto reduce de manera drástica el espacio que debe ser muestreado, mejorando considerablemente la eficiencia de los filtros [33].

2.3 SLAM

Para comenzar a hablar de SLAM es necesario entenderlo más como un concepto que como un algoritmo único [35]. El problema de mapeo y localización simultanea está ampliamente definido en la literatura, y parte desde el principio de que se tiene un robot autónomo que inicia en una posición desconocida dentro de un medio desconocido, e incrementalmente va generando un mapa de su entorno, mientras que simultáneamente usa este mapa para computar la localización absoluta del vehículo [36].

La dificultad de este procedimiento radica en que para poder localizar se necesita de un mapa, y para poder generar un mapa se requiere un punto de partida, y es por esto que en la mayoría de soluciones planteadas el costo computacional es alto [37].

Uno de los principales retos en materia de SLAM es conocido como Loop-Closure, consiste en que el sistema sea capaz de reconocer que ya ha pasado por un punto y lo asocie dentro del mapa. Esto es más complicado de lo que parece, pues mediciones de lugares similares pueden ser tomadas como iguales y deformar por

completo el mapa, así mismo es posible que al recorrer un mismo lugar, pero desde una perspectiva diferente el sistema lo reconozca como dos lugares diferentes y se generen curvas no deseadas dentro del mapa [38].

2.3.1 Tipos de Slam

A grandes rasgos, y según Stramigioli et al. [39], es posible clasificar al SLAM en SLAM basado en filtros, SLAM por optimización global y SLAM por inteligencia artificial.

2.3.1.1 Basada en *Filtros*

Es la aproximación clásica, que hace predicciones y se actualiza recursivamente. Este método mantiene información del ambiente y de los estados del robot como una función de densidad de población. Con ello realiza labores de predicción y corrección que hacen de la estimación de estados una tarea alcanzable.

En esta clasificación se encuentra incluida toda la familia de filtros de Kalman (EKF, UKF, SEIF, etc.) y los filtros de partículas también

2.3.1.2 Optimización Global

Se basa en guardar algunos marcos clave en el espacio y realizar ajustes para estimar el movimiento. Como todo método matemático de optimización este intenta conservar los valores de las variables intermedias que hacen que el resultado final sea el mejor de los posibles. En este caso la optimización se aplica bien para mejorar los detalles finos del mapa, o bien para encontrar la mejor localización en donde insertar las lecturas de los sensores.

Es un método muy utilizado para aproximaciones de SLAM basadas en visión, como ORB-SLAM o Google Cartographer

2.3.1.3 Inteligencia Artificial

Las aplicaciones de inteligencia artificial en SLAM son variadas, entre las técnicas más resaltables están las bioinspiradas, como las redes neuronales artificiales.

Estos son métodos menos utilizados, pues requieren costo computacional más alto, sin embargo, entregan mejores resultados.

Las redes neuronales artificiales se fundamentan en un gran conjunto de unidades básicas llamadas neuronas, organizadas en capas e interconectadas entre sí, capaces de aprender y llevar a cabo actividades específicas mediante entrenamiento.

En los últimos años se han hecho diferentes investigaciones sobre el modelo del hipocampo, intentando recrearlo artificialmente para aplicaciones de navegación. Uno de los resultados más evidentes está en el método conocido como RatSLAM [40], el cual utiliza el modelo de el hipocampo de roedores para el procesamiento de landmarks.

2.4 Guiado

Para que un robot móvil sea capaz de desenvolverse en el mundo real, además de generar su propio mapa, es necesario que pueda navegar a través de él. Para que pueda hacerlo de manera autónoma, se requieren estrategias de control inteligente que sean capaces de manejar la incertidumbre del medio, y que lo hagan con un bajo costo computacional, de forma tal que los algoritmos puedan correr en línea a velocidades de desplazamiento aceptables [8].

Dentro de entornos normales es difícil que un robot pueda establecer línea de visión directa desde su punto de partida, hasta su punto de finalización (Figura 4), por lo que técnicas de seguimiento de trayectoria deben ser implementadas. Pero en adición, en esos mismos entornos normales, es común que se encuentren también otros robots, personas o en general obstáculos dinámicos, por lo que se hace necesario establecer dos partes del control de trayectorias: una global que establecerá la ruta más corta para alcanzar el objetivo, y una local que evitará los obstáculos, tanto dinámicos, como estáticos.

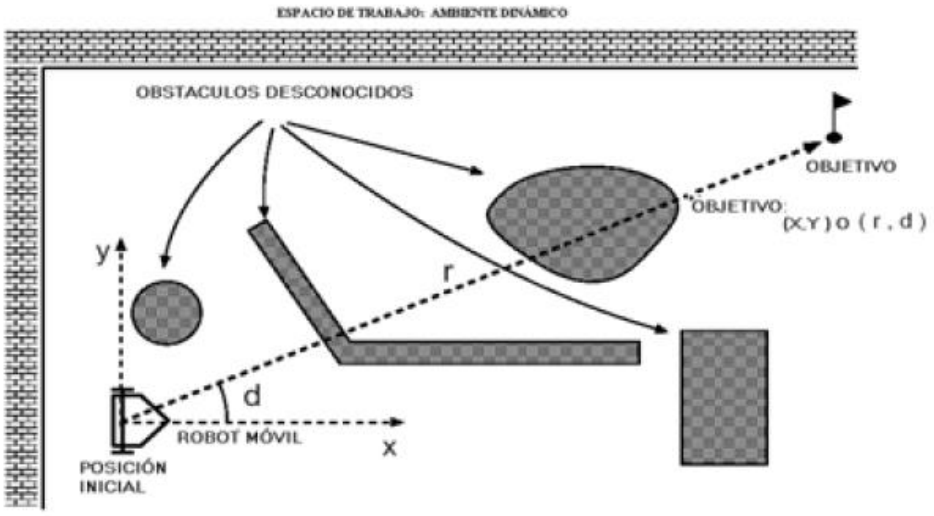


Figura 4 Escenario Típico Para un Robot Móvil. Imagen tomada de [8]

Existen diversas formas de guiar a un robot en medio de espacios desconocidos. En general la regla base consiste en una serie de instrucciones de la forma Percepción – Acción, esto permite que existan múltiples soluciones simples, con costos computacionales excepcionalmente bajos, que usan lógica difusa para generar patrones de comportamiento y reacciones ante diferentes percepciones [8] [41] [42]. Sin embargo, los comportamientos predeterminados presentan serias dificultades ante obstáculos dinámicos.

Por otra parte, se tienen las aproximaciones basadas en LandMarks. Estas consideran regiones en las que el robot puede identificar alguna característica distintiva del espacio de trabajo, que luego puede utilizar para ubicarse en zonas más pequeñas [13]. Así el robot intentará desplazarse dentro de las zonas en las que la incerteza es menor. Como se ve en la Figura 5 el robot intenta estimar su posición actual basado en la percepción de distintos LandMarks, lo que lleva a una doble estimación de posición, primero del Landmark y luego del robot. Y si bien la localización o la medición exacta siempre serán desconocidas, las estimaciones son hechas sobre la posición real tanto del robot como del Landmark, por lo que el error no será acumulativo.

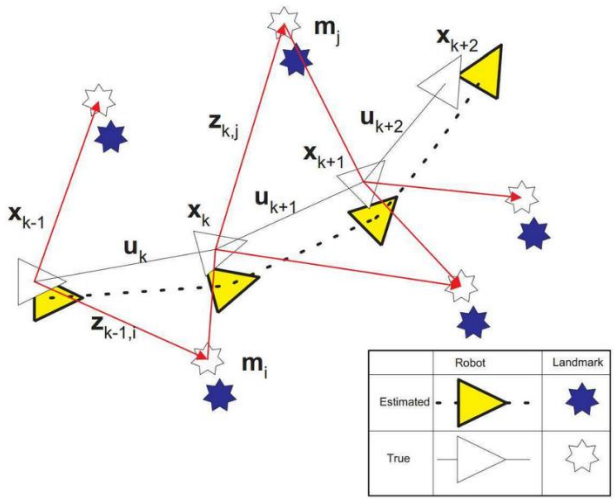


Figura 5 LandMarks siendo observados desde distintos puntos de una trayectoria. Imagen tomada de [43]

El gran problema con los métodos basados en LandMarks es el alto costo computacional que implica cualquier modificación en el espacio de trabajo [44].

Un método bastante más eficiente, en cuanto a costo computacional, es la aplicación de campos potenciales artificiales al seguimiento de trayectorias. Este consiste en generar hacia el robot una fuerza repulsiva desde los obstáculos y una fuerza de atracción desde la meta. Con este método se corre el riego de que el robot se quede atascado en un mínimo sin alcanzar la meta [45].

Otro método existente es conocido como aproximación de ventana, este además de utilizar la cinemática del robot, incluye también restricciones dinámicas, de forma tal que se reduce aún más el espacio de búsqueda, sin embargo, el problema de los mínimos persiste. En la aproximación propuesta por Thrun et al. [46] se sumaron las ventajas de la aproximación de ventana con un planeador de trayectorias, y adicionalmente se usa la información sensórica para un movimiento sin colisiones, la única desventaja es que este método requiere un conocimiento a priori del espacio de trabajo.

Finalmente, uno de los métodos más acertados, de los encontrados en la literatura, es el propuesto por Fox et al. [47], este es conocido como aproximación por ventana dinámica, este consiste en encontrar un conjunto de velocidad angular y velocidad lineal, alcanzables por el robot, de forma tal que no colisione.

Según el método, una velocidad será considerada admisible si según las características dinámicas del robot, este es capaz de detenerse antes de llegar a un obstáculo que se encuentre en la trayectoria descrita según el conjunto de velocidades lineales y angulares.

Para encontrar dichas velocidades se establece un periodo de tiempo corto en el que se simularan cada una de las posibles velocidades alcanzables por el robot y se establecerá si colisiona o no.

Un rasgo resaltable del método es que todas las trayectorias descritas serán circulares, esto debido a que todo desplazamiento es considerado como una combinación de movimientos traslacionales y rotacionales.

Adicionalmente la ventana dinámica restringe las velocidades a las alcanzables según la dinámica del robot, de forma tal que no se desperdicie procesamiento en opciones imposibles.

Así, se tienen 3 características concretas para definir el espacio de búsqueda: trayectorias circulares, velocidades admisibles y la ventana dinámica (Figura 6).

Por otra parte, el método utiliza una maximización de una función objetivo (Ecuación 9) para la optimización de las trayectorias. Está función tiene en cuenta 3 aspectos fundamentalmente: Medición de progreso hacia la meta, la distancia hasta los obstáculos, y la velocidad de avance del robot. Estos 3 son suavizados y su suma ponderada da como resultado la mayor separación lateral posible de los obstáculos.

$$G(v,w) = \sigma(\alpha \cdot progreso(v,w) + \beta \cdot distancia(v,w) + \gamma \cdot velocidad(v,w)$$

Ecuación 9 Maximización de la función objetivo [47]

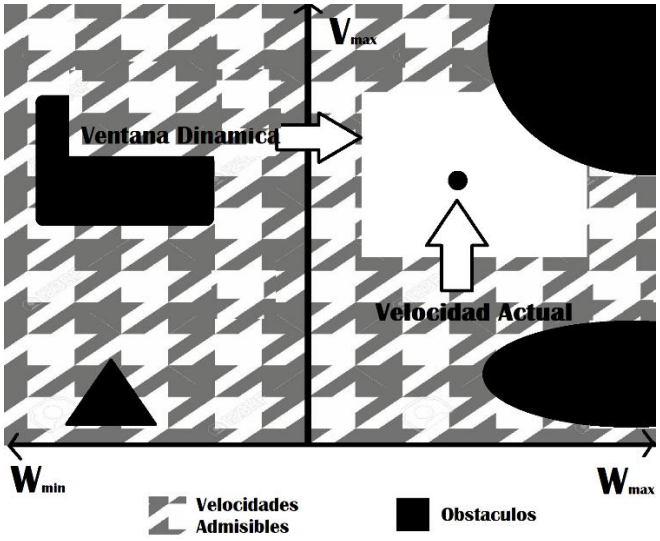


Figura 6 Espacio de Búsqueda para la aproximación por Ventana dinámica. Recreado de [45].

Este método permite evasión de obstáculos a altas velocidades, y existe una versión tanto para robots holonómicos como no holonómicos [47]. Sin embargo, ambas aproximaciones son susceptibles a caer en un mínimo local, y esto se debe a que la única influencia que tiene el movimiento del robot con respecto a la meta es el parámetro *progreso*. Esta limitación puede ser removida con la incorporación de información acerca de la conectividad entre el espacio libre en la selección de un comando de movimiento [45].

La solución ofrecida por Brock y Khatib [45] propone la implementación de los métodos holonómico y no holonómico de Fox [47], adicionando un simple y eficiente planeador de trayectorias. El método plantea el cálculo de una función NF1 que arroja una trayectoria aproximada dentro de la ventana dinámica, que acerque al robot hasta la meta, y que será recalculada conforme se desplace la ventana y cada vez que un nuevo comando de movimiento sea seleccionado.

A manera de resumen se puede decir que existen múltiples técnicas que pretenden dar solución al problema de navegación, sin embargo, la mayoría tienden a caer en mínimos sin alcanzar la meta o tienen costos computacionales demasiado elevados, por lo que sea hace factible implementar técnicas sencillas que complementen los métodos y lo lleven a un final satisfactorio. De esta manera la técnica escogida para el presente proyecto es la propuesta por Brock y Khatib [45].

3 Hardware y Software Utilizados

En este capítulo se presentarán los elementos físicos y computacionales manejados. Se describirán y referenciarán los sensores, la plataforma móvil y el computador utilizados. Además, se definirá brevemente que es ROS y todos los nodos y paquetes que fueron utilizados.

Se describirá primero todo el Software, incluyendo la selección del paquete de ROS a utilizar para SLAM, y la explicación del funcionamiento de los paquetes que se utilizarán para EKF, localización y navegación. Y finalmente se hablará del Hardware.

3.1 Software

3.1.1 ROS

Teniendo que ROS (Robot Operation System) es el lenguaje utilizado en las pruebas de este proyecto y también será el medio por el cual se portará los sistemas de mapeo y navegación a la plataforma para agricultura, a continuación, se define que es y cómo funciona, a groso modo, este sistema operativo.

Según su página oficial [48] el Robot Operation System (ROS) es un conjunto de bibliotecas de software y herramientas que permiten y ayudan a la creación de aplicaciones de robots. Pueden implementarse desde controladores básicos, hasta complejos algoritmos con potentes herramientas de desarrollo.

Sus creadores tienen la convicción de que crear un software realmente robusto y de uso general para robots, es una tarea prácticamente imposible para cualquier individuo, laboratorio o institución, si pretende hacerlo solo. Y es por eso que ROS fue concebido como un medio para la robótica colaborativa y en pro de la educación y el desarrollo productivo.

ROS utiliza códigos abiertos y fomenta el conocimiento colectivo a través de wikis accesibles para cualquiera. Esto ha logrado crear una sólida comunidad que, a través de foros, se apoya y consigue solucionar de manera rápida y efectiva problemas comunes.

ROS fue diseñado para ser lo más modular y distribuido posible. Esto permite que sus usuarios usen tanto o tan poco ROS como lo deseen. Así mismo, apoyándose en el concepto de comunidad, es fácil para los desarrolladores crear paquetes que funcionen de manera independiente. De esta forma, otros usuarios integran

diferentes paquetes, implementan sus propios desarrollos y agregan mayor valor al sistema central de ROS.

Otra de las grandes bondades de ROS es que se encuentra licenciado bajo el estándar *three-clause BSD license*. Esta es una permisiva licencia abierta que permite reutilizar el contenido de la comunidad en aplicaciones en el comercio o en productos de código cerrado.

En ROS los códigos, conocidos normalmente como programas, son llamados nodos, y es posible ejecutar varios de estos simultáneamente. Los conjuntos de nodos forman paquetes, y los paquetes pueden ser subidos y descargados desde una plataforma llamada GitHub.

La comunicación entre nodos se lleva a cabo a través de mensajes que se publican en tópicos. Es decir, si un nodo A y un nodo B desean establecer una comunicación, ambos se suscriben al tópico 1, y en él ambos nodos podrán publicar mensajes y leer todo lo que haya sido previamente publicado.

A continuación, se hace un recuento de los principales paquetes utilizados en este proyecto.

RVIZ

Es el visualizador 3D por defecto de ROS. En él es posible representar diferentes tipos de mensajes (odometría, mapas, nueves de puntos, laser, cámaras, entre muchos otros, como se ve en la Figura 7) o ver los gráficos generados por un conjunto de parámetros en URDF (Unified Robot Description Format), e incluso es posible generar sensores simples y crear simulaciones.

Es una herramienta muy útil para dar claridad sobre los diferentes datos publicados en un tópico de ROS.

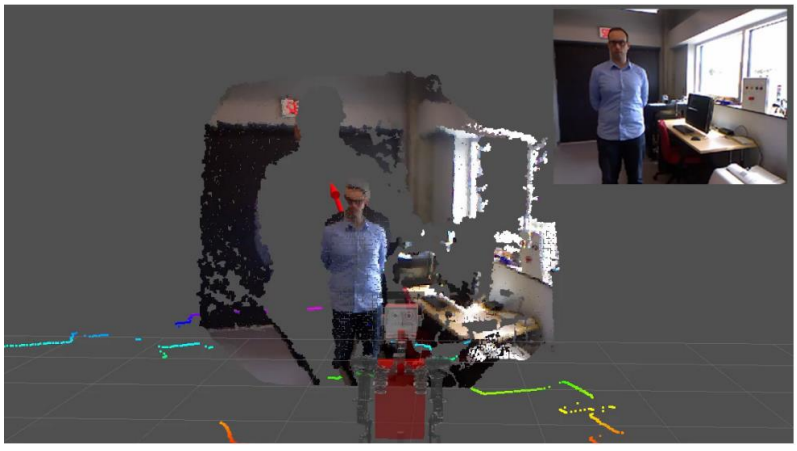


Figura 7 Representación nube de puntos y escaneo laser. Imagen tomada de [49]

Gazebo

Es el simulador por defecto de ROS. Es una poderosa herramienta que permite crear escenarios 3D, con Robots, obstáculos, y diferentes tipos de objetos y funciones como iluminación, gravedad, inercia, colisiones (Figura 8).

Es extremadamente útil para testar cualquier tipo de aplicación robótica antes de llevarla directamente al robot en físico.

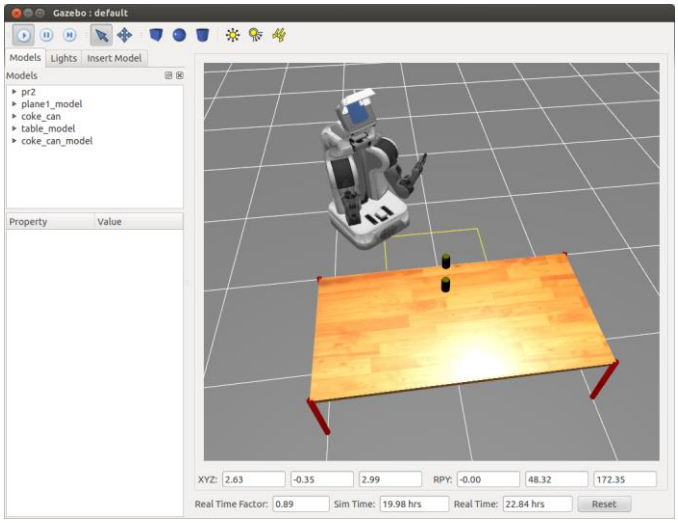


Figura 8 Muestra de las posibles simulaciones realizables en Gazebo. Imagen tomada de [50]

3.1.2 EKF

Robot Localization

Es una colección de nodos, creada por Moore [51], para estimación de estado, que contiene principalmente dos nodos: ekf_localization_node y ukf_localization_node. A demás, el paquete proporciona navsat_transform_node, un nodo que facilita la integración de datos GPS con información inercial y de odometría, generando un nuevo mensaje de odometría [52].

A pesar de que existen otros paquetes que implementan la estimación de estados, estos tienen limitaciones en cuanto a la cantidad de sensores de entrada, únicamente trabajan en planos 2D y el control sobre los datos de los sensores es limitado. Robot Localization pretende superar estas limitaciones y ser tan general como sea posible [53].

3.1.3 SLAM

Paquetes existentes

Google Cartographer

Es un sistema que provee una simulación en tiempo real de localización y mapeo en entornos 2D y 3D, para múltiples configuraciones y distintos tipos de sensores [54].

El algoritmo toma las lecturas realizadas por el láser y las inserta en la posición estimada por la odometría (esta se supone lo suficientemente precisa para cortos periodos de tiempo), y se constituye así un submapa [55].

Cuando un submapa es completado no se insertan nuevos escaneos, estos escaneos se utilizan para un procedimiento de cierre de bucles. Si los escaneos son lo suficientemente parecidos un analizador de escaneos intenta encontrar el escaneo actual en el submapa.

Si la coincidencia es lo suficientemente buena en una ventana cercana a la posición estimada, se agrega al cierre de bucle como una restricción al problema de optimización.

El sistema utiliza una combinación de aproximaciones locales y globales. En la estimación global, cada escaneo consecutivo es contrastado con un pequeño pedazo del mundo, llamado submapa. El proceso de llevar el escaneo hasta el submapa es conocido como scan matching. Este acumula error, que es removido luego con la estimación global.

La construcción de submapas se hace con unos pocos escaneos. Estos submapas adoptan la forma de cuadriculas de probabilidad (Figura 9), de una resolución definida. Estos valores pueden ser considerados como la probabilidad de que un punto de la cuadricula este obstruido.

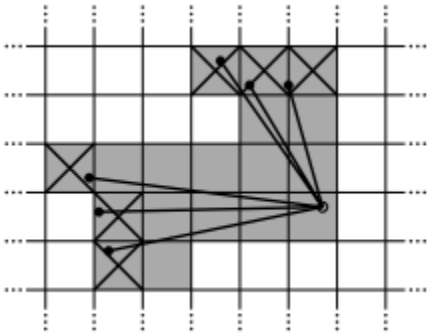


Figura 9 Sub Mapa Google Cartographer. Imagen tomada de [55]

Previo a la inserción del escaneo en el submapa se corre una optimización utilizando Ceres-solver, una librería creada por Google, para modelar y resolver problemas de optimización, que involucren mínimos cuadrados no lineales con restricciones de límites y problemas generales de optimización no restringida.

Héctor Slam

Este es uno de los algoritmos más utilizados en ROS para el mapeo y localización, por ello está ampliamente documentado [56], [57] y los casos de estudio son numerosos [37], [58] .

Requiere como mínimo un sensor laser de alto rendimiento y dependerá en gran medida del desempeño de este. No requiere datos de odometría, sin embargo, es posible incluir información inercial.

Hector_Mapping es el nombre del nodo encargado de recibir la información entregada por el láser y convertirla en un mapa sobre el cual se desplaza un eje coordenado. Este no cuenta con sistema de Loop-Closure (concepto explicado en la sección 2.3) , pero es eficiente para aplicaciones que no lo requieran [57].

Gmapping

Este método utiliza un filtro de partículas, en el cual cada partícula se encarga de un mapa individual del entorno. Para reducir el número de mapas se utiliza la aproximación de Rao- Blackwellized (RBPF) [59].

El paquete utiliza una de las representaciones más comunes para mapas, esta es conocida como Occupacy grid (cuadricula de ocupación). Esta técnica consiste en representar el entorno en el que se encuentra el robot como un campo uniforme de variables binarias, las cuales muestran la existencia o ausencia de obstáculos en una posición dentro del entorno.

El problema de mezclar occupacy grid con RBPF consiste básicamente en que para generar una partícula se necesita la totalidad del mapa, haciendo que este proceso tenga que ser secuencial y sea más costoso computacionalmente. Para solucionar esto Giorgio Grisetti et al. [60] proponen que cada partícula parta con una base que será mejorada por los datos de los sensores, sobre la marcha, para así obtener una nueva distribución. De esta forma se utiliza la mayor parte de la información sensórica directamente para crear partículas.

Adicionalmente, el método utiliza el número de partículas efectivas para decidir si es necesario o no un remuestreo, con lo que se consigue limitar aún más el número de partículas [60].

Karto Slam 2.0

Este paquete utiliza un método propio de los autores llamado Sparse Pose Ajustment (SPA) [61]. Este se basa en la optimización de gráficos de localización, entendiéndose por este último al conjunto de localizaciones de un robot, conectadas por restricciones no lineales, obtenidas de las observaciones hechas por los sensores. Para la optimización se asume el problema como uno de mínimos cuadrados no lineales y se utiliza el método de Levenberg-Marquardt para resolverlo.

Esta optimización consigue que el cierre de bucles con este método sea muy eficiente, y en adición, por traerse de un método gráfico, los resultados en mapas de gran magnitud son bastante buenos. Sin embargo, como en la gran mayoría de métodos gráficos de SLAM, el costo computacional es elevado [62].

Core Slam

Este es quizá el algoritmo más simple de SLAM que se puede conseguir en la bibliografía. El articulo relativo a su creación [63] resalta que algoritmo entero no toma más de 200 líneas de código en C.

Los autores sustentan que, dando confianza a la odometría, y enfocando los esfuerzos a pequeños espacios cerrados con bucles, el algoritmo puede obviar la complejidad de otros métodos.

El algoritmo se divide en dos grandes partes: cálculo de distancias y actualización del mapa. La primera parte es llevada a cabo a través de un filtro de partículas simple, en el que cada posible posición del robot es una partícula creada con base en la información obtenida de un sensor laser. La parte de la actualización consiste básicamente en dibujar en el mapa las líneas correspondientes a los escaneos [62].

Lago Slam

Es un algoritmo de SLAM basado en gráficos, que tiene como fundamento la minimización de una función de costo no lineal y no convexa, esta se encarga de que en cada iteración se genere una solución parcial, que pretende aproximarse a la solución del problema inicial. A diferencia de otros métodos basados en gráficos,

este algoritmo no requiere de una suposición inicial [64] y puede utilizar cualquier optimizador estándar.

PL-Slam

Este método [65] se basa en 3 grandes pilares: odometría estéreo visual, mapeo local y Loop-Closure. Su principal ventaja respecto a otras aproximaciones a SLAM estéreo visual es que toma en cuenta la dificultad para encontrar, en entornos de baja textura, un número suficiente de puntos confiables, por lo que propone trabajar además de puntos, también con segmentos de línea. Como resultado se obtiene un método más robusto que entrega mapas más ricos y diversos.

El proceso de mapeo se fundamente en LandMarks (tanto puntos como segmentos son tomados en cuenta), luego de que estos son obtenidos se comparan la información de las diferentes lecturas de los sensores, de forma tal que se generen puentes entre las lecturas que tengan determinada cantidad de LandMarks en común. Y de este mismo modo es sencillo identificar cuando ya el vehículo ya ha pasado por un sector; solo es necesario obtener un número más alto de coincidencias entre los LandMarks para generar un Loop-Closure.

OpenSeq-Slam

Algoritmo de navegación visual basado en rutas, con importantes logros en reconocimiento de espacios a pesar del cambio de condiciones. Los autores [66] resaltan que el algoritmo es completamente funcional en días soleados de verano o en noches tormentosas de invierno.

La innovación más destacable de este algoritmo es la implementación de secuencias de imágenes en lugar de una sola captura, de esta forma tiene mucha más información, lo cual le permite ser más preciso y robusto ante los cambios en el ambiente.

Selección del Algoritmo de SLAM a Utilizar

El proceso de selección comenzó por ver que en términos generales cualquiera de los algoritmos antes mencionados era útil para el fin de este trabajo, por lo que la elección se basa en cuales el mejor según los recursos existentes.

Lo primero que se hizo fue contrastar la odometría visual generada por las cámaras ZED con la odometría generada con los encoders del robot. Con una visualización simple en RVIZ se hizo evidente que los parámetros de la odometría visual son fuertemente afectados por los factores propios del ambiente, en específico la luz. En entornos donde existen diferentes focos de luz la odometría visual entregada por

la ZED dista mucho de la entregada por los encoders, y debido a que el deslizamiento en las ruedas del robot es aparentemente despreciable se decidió descartar los métodos que utilizan localización netamente visual.

Después de esto se consideró utilizar Core Slam, debido a su simpleza. Pero al leer con más detalle se encontró que este está diseñado para espacios cerrados y que no cuenta con un algoritmo de cierre de bucles, lo que representaría un problema al momento de exportarlo a una plataforma agrícola, como es el propósito de este trabajo.

Esto dejó solo 3 algoritmos posibles: Cartographer, Hector y Gampping. Estos paquetes son de los más usados según la literatura, por tanto, están ampliamente documentados, lo cual sustenta que la correcta elección de estos. Se encontró incluso una tesis de maestría [67] que compara los pros y contras de los 3 métodos.

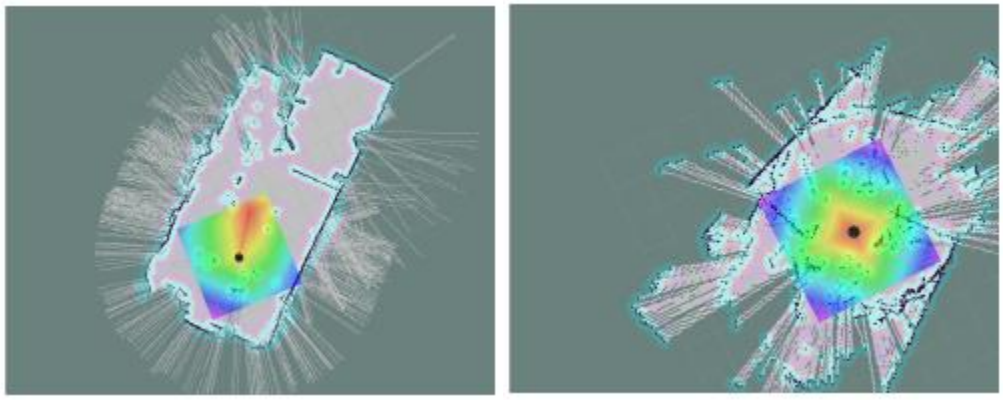
Este documento [67] relata sobre la simulación y puesta en marcha de cada uno de los 3 algoritmos y entrega resultados e impresiones sobre su desempeño y general y con respecto al cierre de bucles.

Coroiu et al. destacan de Gmapping que requiere como mínimo un sensor Lidar e información de odometría. Concluyen que la reconstrucción de mapas en términos generales es buena, que existen ciertos problemas en la simulación con respecto a las colisiones, pero que no es algo de mucha trascendencia pues en entornos reales el robot no debe colisionar con nada. Los autores afirman también que el paquete genera una serie de puntos que no concuerdan con el mapa real, y que estos son debidos a los puntos en los que el Lidar entrega información incoherente (un dato en infinito).Este problema afecta solamente de manera estética al mapa, pues las líneas que denotan los obstáculos quedan bien definidas y para trabajos de navegación el robot no tendrá problemas en detectar los límites del espacio (Figura 10 a).

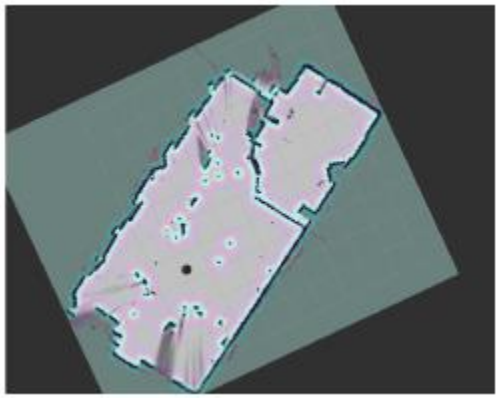
Con respecto a Hector Slam, Coroiu et al. sostienen que a pesar de que el algoritmo puede correr únicamente con un sensor Lidar cuenta también con soporte para recibir datos de una central inercial. Enfatizan en que el algoritmo está diseñado para consumir pocos recursos de máquina y aprovechar al máximo las características del sensor laser. En la simulación este algoritmo tiene poca o nula deriva, es decir que en la mayoría de los casos el algoritmo es capaz de predecir exactamente dónde está el robot, sin embargo, al llevarlo a la práctica los resultados en las rotaciones son fatales. Los autores asumen que este problema es debido a la baja frecuencia de muestreo del láser utilizado y concluyen que el algoritmo requiere un mejor láser (frecuencia de operación > 5Hz) que el que se está usando (Figura 10 b).

Y en cuanto a Cartographer, Coroiu et al. resaltan que los mapas que este genera contienen información probabilística de la ocupación de cada celda, que es mostrada en una escala de grises. También cuenta con un método exclusivo para

cierre de bucles. El resultado es notoriamente mejor que el de los otros dos algoritmos, estéticamente hablando (Figura 10 c).



a) b)



c)

Figura 10 Mapeo de un mismo entorno con a) Gmapping b) Hector y c) Cartographer. Imagen tomada de [67]

Finalmente, como conclusión del trabajo, los autores de [67], seleccionan Cartographer por su evidente superioridad en el cierre de bucles. Sin embargo, resaltan que Gmapping proporciona mapas menos distorsionados y puede utilizar odometría de una forma en que Cartographer no. A pesar de ello también se destaca que la odometría puede no ser un factor importante cuando se utiliza un LIDAR de alto rendimiento. Basados en estas conclusiones, se seleccionó Cartographer como método a utilizar.

Como observaciones a la instalación y uso de Cartographer, se puede decir que:

1) Existen dificultades en el proceso de instalación, principalmente por el conflicto de versiones entre las dependencias del paquete, iniciando porque este sugiere utilizar un compilador diferente (Ninja) al que usan los demás paquetes (Catkin_make), lo que genera conflictos con paquetes previamente compilados, pues ambos generan archivos y carpetas de configuración durante la compilación, que luego son requeridos para poderse ejecutar, y dichos archivos y carpetas son

mutuamente excluyentes. Como solución alternativa se usó un comando de Catkin_make que permite compilar paquetes aisladamente.

Por tratarse de un paquete dirigido a código abierto este es susceptible a todas las modificaciones que hagan los diferentes usuarios a las dependencias. Por este motivo fue necesario revisar una a una las versiones de cada referencia con la cual es posible compilar sin errores el paquete.

2) Luego de instalado fue posible ejecutar los demos sugeridos en los tutoriales oficiales, obteniendo los resultados que se pueden apreciar en la Figura 11.

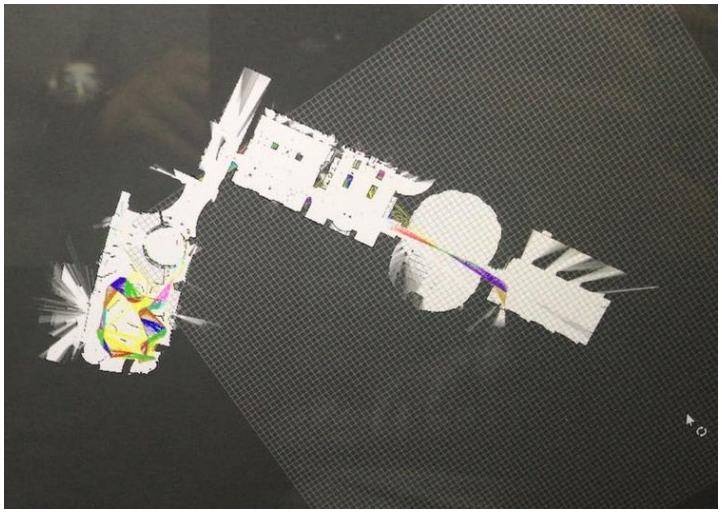


Figura 11 Mapa generado con los datos del tutorial de Google Cartographer. Imagen tomada por el autor

Cuando se intentó generar un mapa con los valores de los sensores utilizados en este proyecto, el algoritmo no tomaba estos como validos a pesar de tener el tipo de dato requerido por el paquete.

Finalmente, en medio de la búsqueda de una solución, se encontró que Cartographer no incluye información de la dinámica del robot, por lo que la localización no será eficiente para el futuro trabajo de navegación autónoma. Es decir, podría obtenerse un buen mapa, pero el robot dependería únicamente de los datos que pudiera percibir de su ambiente para localizarse, teniendo encoders instalados y con el software necesario para su lectura y transformación en datos de odometría. Esto representa una subutilización de los recursos disponibles, y partiendo de la premisa de la eficiencia, esto es inaceptable, por lo que se decidió descartar Google Cartographer.

Siguiendo con los lineamientos dados por la bibliografía [67] se seleccionó en definitiva Gmapping como algoritmo para SLAM.

Se adjunta el código de Slam_gmapping, nodo principal en el mapeo. Este se encuentra en el anexo A.

3.1.4 Localización

AMCL

El paquete recibe su nombre por sus siglas en inglés, Aproximación de Localización de Monte Carlo. Es un método de localización probabilístico para un robot que se mueve sobre un plano 2D, de un mapa ya conocido.

El método de Monte Carlo fue inicialmente propuesto en 1970 por Handschin [68] con propósitos generales de estimación de estados. Luego para 1993, Gordon et al. [69] propusieron una forma independiente de seguimiento de objetos. Con esta aplicación y las nuevas tecnologías que permitían velocidades de procesamiento mucho más rápidas, se hizo posible generar desarrollos novedosos en esta área.

El algoritmo parte de lo que se conoce como la suposición de Markov, esta asume que los procesos con los que se trabajarán serán estocásticos y que no tendrán en cuenta las muestras pasadas, es decir que la distribución de probabilidad de una variable aleatoria sujeta a esta suposición dependerá únicamente de su valor presente [70]. Para este caso particular la suposición dicta que el único estado en el entorno que afecta sistemáticamente a las lecturas de los sensores es la ubicación del robot.

La presunción de Markov tiende a ser fácilmente quebrantada en ambientes donde existe interacción humana, pues cualquier tipo de movimiento humano dentro del área en la que los sensores toman mediciones generará oclusiones que harán parecer que el entorno en el que se encuentra el robot es diferente del mapa que se le proporciono inicialmente. No obstante, el método puede ser usado para ambientes no muy alterados con respecto al plano inicial.

La técnica de Monte Carlo se fundamenta en un filtro de partículas que toma N partículas cada vez que el robot ejecuta un movimiento. Como es bien sabido, los filtros de partículas tienen un costo computacional bastante alto, y por esto el método lo compensa utilizando muestreo proporcional a la probabilidad, es decir que tomará más muestras en donde estas sean realmente importantes [71].

3.1.5 Navegación

Navigation Stack

Este es el paquete por defecto de ROS para navegación. Este toma información de odometría, lecturas de los sensores y una posición deseada, y entrega comandos de velocidad seguros para el robot [72].

El concepto general es bastante sencillo, la dificultad se encuentra al intentar hacer un paquete genérico para los diferentes tipos de robots, con los diferentes tipos de sensores existentes, es por esto que el prerrequisito básico del paquete es que el robot trabaje con ROS.

Este paquete cuenta con 3 opciones de planificador global, cada uno independiente del otro, de forma tal que es posible seleccionarlos y compararlos.

El primero de ellos es *carrot planer,* el más simple de los 3 métodos. Este revisa si entre la posición actual y la meta existe algún obstáculo, de ser así establece una meta parcial. Es muy útil cuando se pretende acercar lo más posible el robot a la meta. No se sugiere para ambientes muy complejos.

El segundo método es *navfn,* este se basa en en el algoritmo de Dijkstra, el cual consiste en evaluar los diferentes caminos y escoger el que tenga menor costo, o en otras palabras el que sea más corto.

Y finalmente *global_planer.* Este está diseñado como un complemento versátil de *navfn*. Este puede utilizar algoritmos como A*, aproximación cuadrática y ruta de red [45].

Además del planeador global existe un planeador local, este es mencionado como *DWA_local_planner* haciendo referencia al método de Aproximación por ventana dinámica, que se explicó en la sección de 2.4.

De manera conjunta con el planeador local está el mapa de costos. Este se compone de 3 capas: la capa estática, la capa de obstáculos y una opcional llamada capa de inflación. En la capa estática se tendrá el resultado directo del SLAM, en la capa de obstáculos se encontrarán los obstáculos tanto 2D como 3D, y en la capa de inflación se encuentran los obstáculos inflados para proveer información adicional a la función de costos. Sobre este mapa de costos es que el planeador de trayectorias actuará.

Todos los resultados, tanto parciales como globales, son fácilmente visualizables desde Rviz, y desde allí mismo es posible establecer la meta a alcanzar por el robot. Existe también un método para introducir la meta desde la ventana de comandos, pero resulta más práctico y eficiente hacerlo desde la interfaz de Rviz.

Se adjunta el código de move_base, el nodo principal encargado del desplazamiento de la plataforma. Este se encuentra en el anexo B.

3.2 Hardware

En esta sección se presentarán los diferentes elementos utilizados durante el desarrollo del proyecto. Se establecerán las principales características de cada uno y su aporte.

3.2.1 Pioneer 3DX

Es un robot de la marca MobiliRobots®, diferencial de dos ruedas, pequeño y ligero, diseñado para uso en interiores. En la Figura 12 se puede apreciar el robot como viene de fábrica, este cuenta con sonares, encoders, baterías, un microcontrolador con ARCOS firmware y un computador embebido [73]. Es un robot fácil de usar, lo suficientemente robusto para la aplicación.

Figura 12 Pioneer 3DX. Imagen tomada de [73]

El robot cuenta con un cuerpo hecho en aluminio (44x28x22cm), dos ruedas de 16,5 cm de diámetro, las cuales van asociadas cada una a un motor independiente con una relación de transmisión de 38,3:1, y encoders de 500 ticks.

El robot puede moverse a una velocidad máxima de 1.6 m/s. A velocidades menores puede llevar cargas hasta de 23 kg [74].

Una de las grandes ventajas de este robot es que permite acceder a los drivers de los motores usando comandos básicos en ROS.

3.2.2 Hokuyo 04LX-UG01

Es un sensor laser asequible y preciso, ampliamente utilizado en diferentes aplicaciones de robótica. Toma medidas hasta de 5,6 m en 240°, con una exactitud de ± 30 mm. Es liviano (160 g) y de bajo consumo (2,5W) [75]. Como se puede apreciar en la Figura 13, solo requiere un cable, a través del cual se envían y reciben datos, y además se alimenta el sensor.

Figura 13 Hokuyo 04LX-UG01. Imagen tomada de [75]

3.2.3 ZED

Es una cámara 3D para sensado de profundidad, motion tracking y mapeo 3D en tiempo real. Cuenta con una cámara doble de 4MP, capaz de grabar videos 3D de 2K en 110°, con una buena sensibilidad de poca luz [76]. Es compacta, sencilla y de fácil conexión (Figura 14).

Figura 14 ZED Camera. Imagen tomada de [76]

3.2.4 Spatial Advanced Navigation (IMU/GPS)

Es un sistema de navegación en miniatura (Figura 15), asistido por GPS, que proporciona posición, velocidad, aceleración y orientación de manera precisa. Combina acelerómetros calibrados por temperatura, giroscopios, magnetómetros y un sensor de presión con un receptor del sistema global de navegación por satélite (GNSS) avanzado, con acceso a 4 satélites. Estos se combinan en un algoritmo de fusión que finalmente entrega una navegación y orientación precisa y confiable [77].

Figura 15 Spatial Advanced Navigation. Imagen tomada de [77].

3.2.5 Pc Dell Inspiron 15-7559

Computador personal portátil (Figura 16) con CPU de 64 bits, procesador core i 7 de sexta generación, 16 Gb de memoria RAM, tarjeta gráfica NVIDIA GTX 960M, con Ubuntu 16.04 [78].

Se escogió trabajar con este PC porque cumple a cabalidad con los requisitos de procesamiento y visualización que el proyecto requiere, tomando en cuenta los diferentes tipos de SLAM existentes.

Figura 16 Dell Inspiron 15-7559. Imagen tomada de [78]

4 Simulación

En este capítulo se relata el proceso que se llevó a cabo para conseguir una simulación del entorno de trabajo desde Gazebo, y como correr los nodos de SLAM y Navegación.

Para poder simular en Gazebo, lo primero que se debe hacer es obtener el modelo 3D del robot con el que se va a trabajar. En este paquete están contenidos diferentes tipos de archivos, los más comunes son:

World: Es un archivo que contiene toda la información sobre el entorno en el que se va a desenvolver la simulación. Es posible crear uno desde 0, agregando luces, piso y obstáculos, o descargar uno ya hecho. Para este caso se utiliza uno de los mundos más comunes, este se llama Willow Garage (Figura 17), que es un entorno 3D de las oficinas donde empezó ROS y OpenCV. Este es un archivo tipo .world y suele ser utilizado a través de un lanzador (.launch)

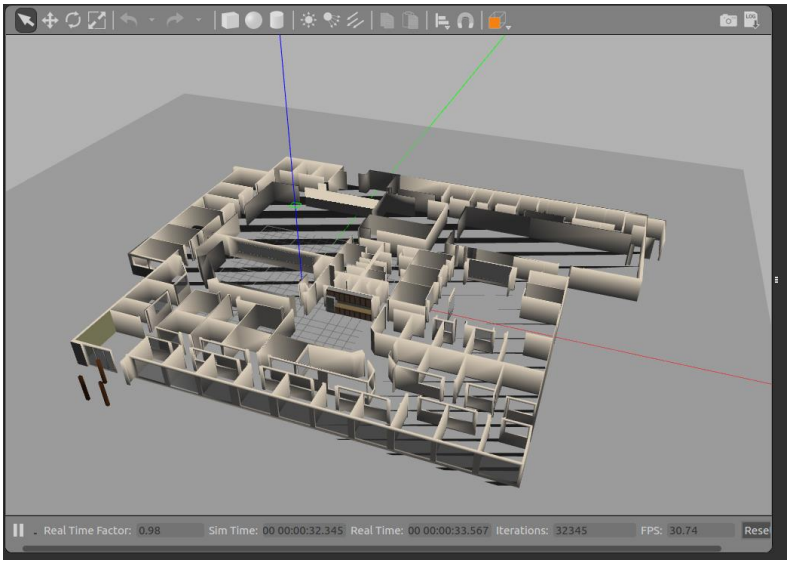


Figura 17 Modelo 3D de Willow Garage en Gazebo. Imagen tomada por el autor

Model: En él se definen cada una de las partes del robot y como se relacionan unas con otras; su geometría, longitud, amplitud, altura, masa y las uniones entre unas y otras. Adicionalmente desde aquí se incluyen 3 archivos que tendrán las especificaciones de el simulador (mybot.gazebo), la definición de cada color o material (materials.xacro) y algunos aspectos matemáticos como momentos de

inercia, que facilitan la descripción del robot (macro.xacro). El resultado puede apreciarse en la Figura 18.

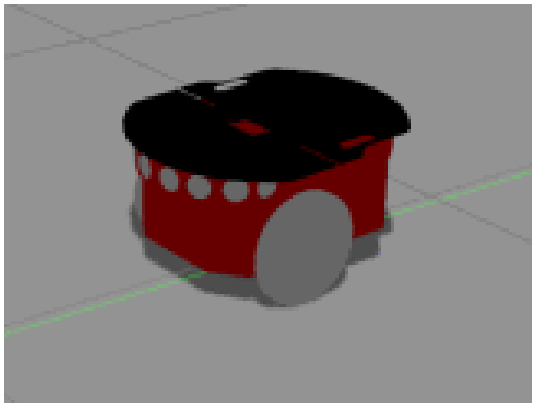


Figura 18 Modelo 3D de Pioneer 3DX en Gazebo. Imagen tomada por el autor

Con estos archivos se consigue visualizar el robot dentro del entorno escogido, pero para poder moverlo es necesario generar archivos de control. Lo que se utiliza normalmente son controladores PID. Estos también pueden ser agregados al mismo lanzador en el que se ejecutan el mundo y el modelo del robot.

Una vez se tienen todos estos archivos ejecutándose es posible publicar mensajes de velocidad a través de la ventana de comandos, y que el robot simulado los reciba como ordenes de movimiento.

Lo siguiente para el proceso de simulación es agregar sensores al robot, de forma tal que este pueda percibir su entorno. Para ello es necesario agregar el modelo del sensor al del robot. Para este caso el Hokuyo ya está definido en Gazebo, por lo que basta con agregar una ruta hasta el archivo de descripción de las funciones del sensor genérico, y en el archivo del modelo especificar los valores del rango, la frecuencia de operación, la localización del sensor y los vínculos con el robot para que se mueva solidario a este. Es importante especificar el marco de referencia en el que se publicaran los datos del sensor.

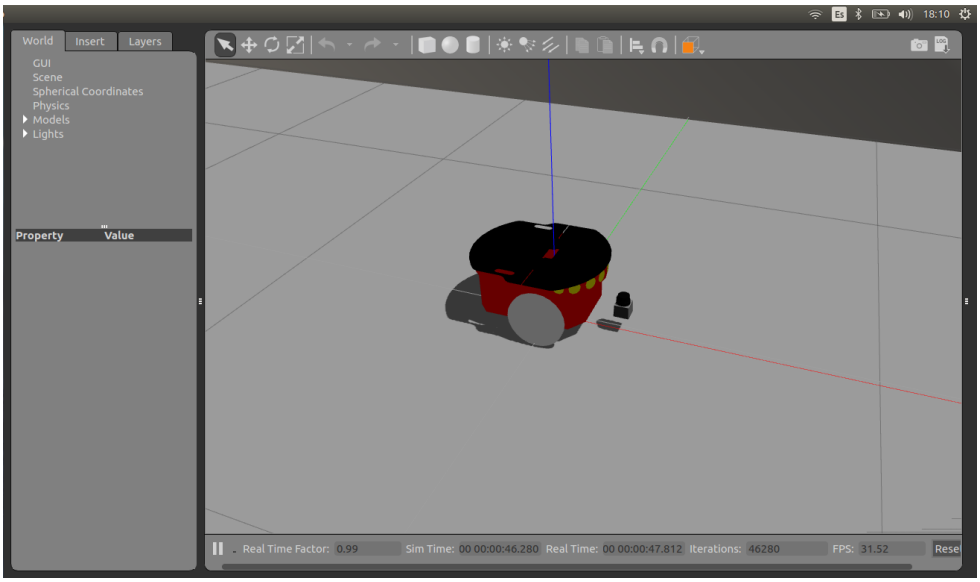


Figura 19 Pioneer 3DX con sensor laser Hokuyo, en Gazebo. Imagen tomada por el autor

Luego de que comprobar que el sensor aparece dentro de la simulación en Gazebo (Figura 19), podemos verificar que se estén publicando datos dentro del tópico que recién se creó. Adicionalmente podemos también contrastar el mapa de Willow Garage con un gráfico de los datos del Laser en Rviz (Figura 20).

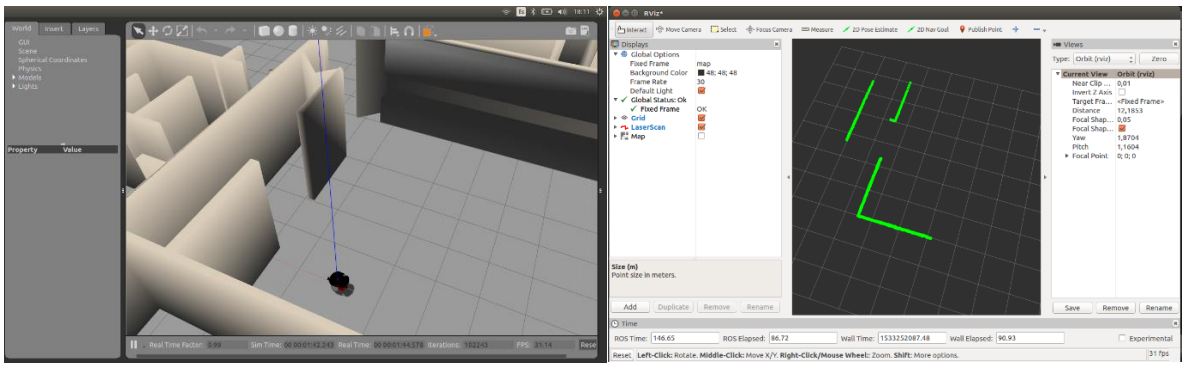


Figura 20 Lectura del sensor laser en entorno simulado. Imagen tomada por el autor

Una vez finalizado el tema del sensor laser, es necesario generar un dato de odometría. Este es un tema bastante sencillo de trabajar en Gazebo. Dado que el robot es del tipo diferencial es posible utilizar el plugin differencial_drive_controller. A este se le deben pasar como parámetros las junturas que dan lugar a los ejes de ambas las ruedas, y la separación entre las mismas, adicionalmente se debe especificar cuál será el tópico que dirigirá al carro, para ajustarnos a los estándares se utilizará cmd_vel. Nuevamente es importante especificar en donde serán publicados los datos. Por tratarse de odometría se deben hacer dos tipos de publicaciones, una en un tópico comúnmente nombrado como odom y una trasformación dinámica entre los marcos de referencia base_link y odom.

Con la odometría y los datos del láser ya es posible generar un mapa utilizando Gmapping. Con tal fin se hace la instalación del paquete y se especifica que el tiempo que se utilizara no será el de ROS, sino que será el dado por Gazebo, esto se hace con el parámetro use_sim_time. Luego de ello ya es posible correr el nodo de slam_gmapping (Figura 21).

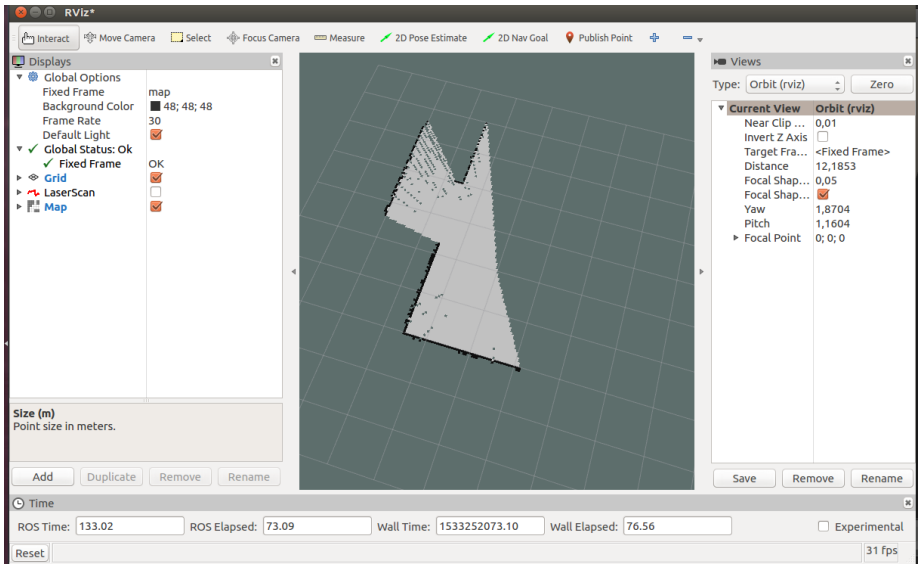


Figura 21 Mapa publicado por Gmapping. Imagen tomada por el autor

Ahora que el robot puede crear mapas de lo que está percibiendo debe asegurarse que pueda moverse autónomamente desde un punto hasta otro. Para conseguirlo se utiliza el paquete Navigation Stack. Se deben generar los diferentes archivos de configuración. Para la simulación es posible dejar la mayoría de los parámetros por defecto, con estos el robot consigue generar trayectorias simples y alcanzar metas dentro del espacio donde se encuentra, a pesar de ello, al momento de atravesar un espacio estrecho (una puerta) presenta dificultades. Sin embargo, el funcionamiento es lo suficientemente bueno como para pasar al modelo real y corregir los errores en él.

En la Figura 22 se presenta una prueba de navegación dentro del entorno de Willow Garage. En la parte izquierda se puede observar que se establece una meta en un punto arbitrario, del cual el robot no posee ninguna información. En la parte central se puede observar como el vehículo alcanza la meta propuesta. Y en la parte derecha se ve como se propone una nueva meta y el vehículo se encamina hacia ella.

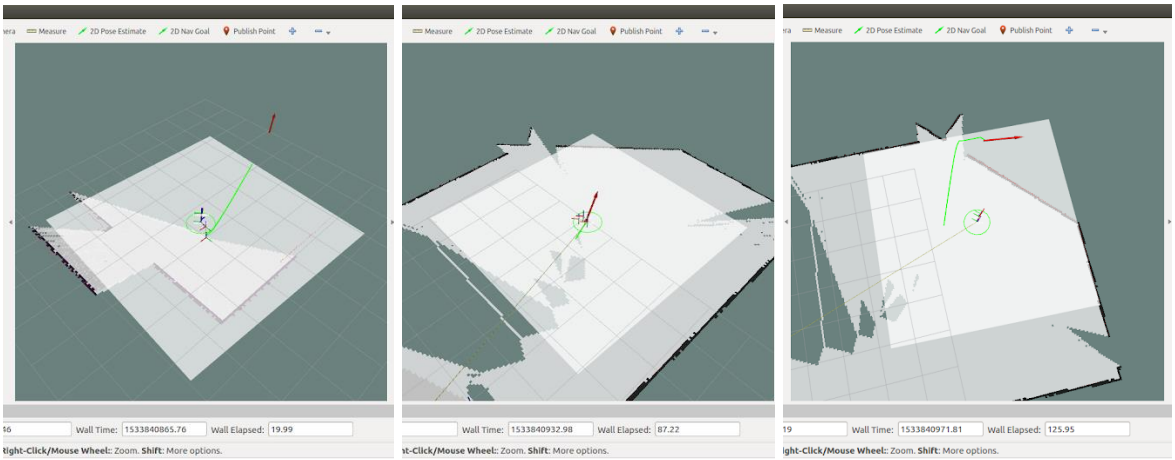


Figura 22 Prueba de Navegación en entorno simulado. Imagen tomada por el autor

A grandes rasgos es posible decir que Gazebo es un entorno muy útil para realizar simulaciones, tanto de sensoria, como de funcionamiento de los algoritmos. Con estas prácticas se consiguen evitar errores y posibles daños en el robot o los sensores.

5 Puesta en Marcha, Pruebas y Resultados

En este capítulo se presenta la metodología utilizada para el desarrollo del proyecto, así mismo como las pruebas realizadas y sus correspondientes resultados.

Inicialmente se planteó la posibilidad de trabajar con la plataforma AmigoBot (Figura 23), un robot diferencial de la empresa MobileRobots®, pequeño y versátil, que cuenta con sonares integrados. Este trabaja con un SDK de Pioneer llamado ARIA (Advanced Robot Interface for Applications) que le permite controlar de manera dinámica y sencilla velocidad, rumbo y diferentes parámetros de movimiento. Adicionalmente también recibe información acerca de la posición estimada (odometría), lecturas de los sonares y demás datos operativos actuales enviados desde el robot [79].

Figura 23 AmigoBot. Imagen tomada de [80]

Esta plataforma fue descarta pues debido a su tamaño era complicado montar todos los sensores sin afectar su estabilidad. Además, sus ruedas pequeñas dificultan su movilidad en terrenos escabrosos.

Como alternativa se propuso trabajar con el Pioneer 3DX (Figura 12), robot más robusto, de la misma familia y que trabaja con la misma configuración (diferencial) y usando el mismo SDK, y adicionalmente cuenta con un pc embebido. Por tanto, era posible suponer que trasladar lo trabajado hacia este no debería implicar mayores complicaciones. Sin embargo, problemas con permisos para acceder a los drivers de los motores desde el computador embebido, retrasó un par de semanas el proceso. Luego de pruebas sencillas se consiguió publicar mensajes de velocidad a través de la ventana de comandos que son leídos por un complemento de ARIA llamado RosAria. Con ello se tuvo una gran dificultad, pues el comando es extenso y por tanto digitarlo toma tiempo, y hacerlo cuando el carro se encuentra en movimiento es complejo. Por ello el siguiente paso fue utilizar un nodo para

teleoperación, con el que es posible guiar al robot con sencillos comandos de teclado o incluso utilizando un joystick.

Todo lo anterior se realizó desde el computador embebido en el Pioneer 3DX. Este es un Mamba de 2,26 GHz, con un procesador Intel core 2 Duo, una CPU de 32 bits, 4 Gb de RAM y una tarjeta gráfica Intel GMA 4500 HD. Para iniciar los nodos de RosAria y de teleoperación se conecta el pc a una pantalla, luego cuando se inicia el movimiento se desconecta, y se utiliza solamente el teclado.

Una vez se consiguió un movimiento continuo y controlado se procedió a montar los sensores sobre el robot. Se decide ubicar el sensor laser lo más cercano al suelo posible, pues este es capaz de detectar obstáculos que estén muy próximos a él, así que se tomarán sus lecturas como base para el SLAM. De igual forma el estar cercano al suelo permite que el sensor detecte obstáculos más pequeños que el robot, garantizando así que la carcasa no se estrellara con obstáculos que puedan dañarlo.

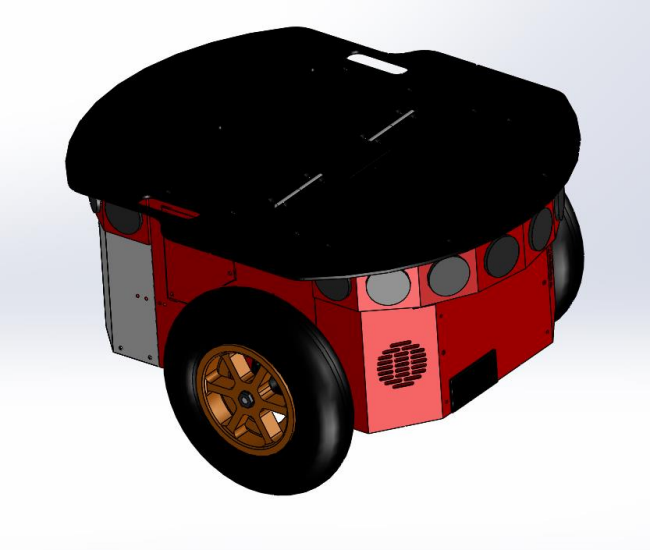


Figura 24 Modelo CAD del Pioneer 3DX. Imagen tomada por el autor

Basados en el modelo CAD del robot (Figura 24) se diseñó la plataforma de la Figura 25 para dar soporte y protección al sensor.

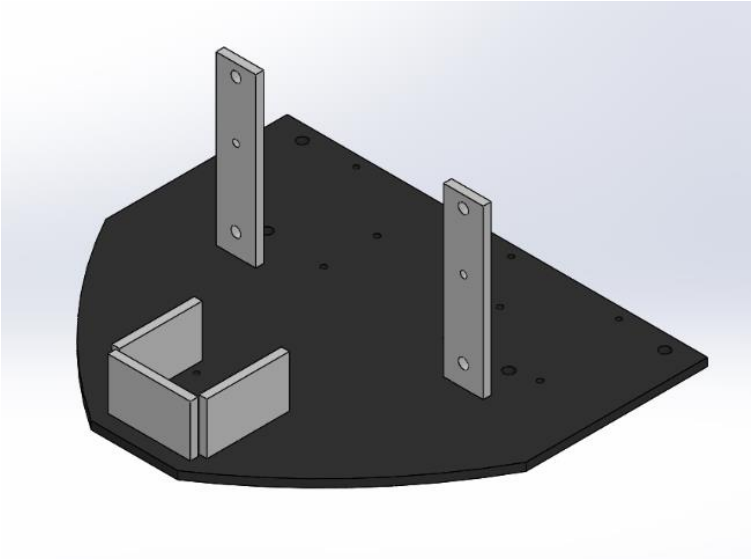


Figura 25 Base para Hokuyo. Imagen tomada por el autor

Por otra parte, es necesario mantener la central inercial lo más cerca posible al centro de masa del robot, y adicionalmente se debe agregar una antena a por lo menos 50cm de distancia del GPS. Teniendo esto en cuenta se diseñó la base que se puede ver en la Figura 26.

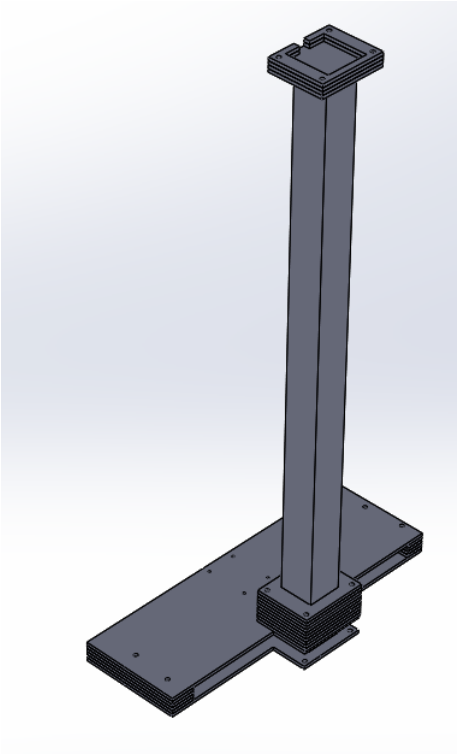


Figura 26 Base para Spatial Advanced Navigation. Imagen tomada por el autor

La integración de las bases antes mencionadas se puede observar en la Figura 27.

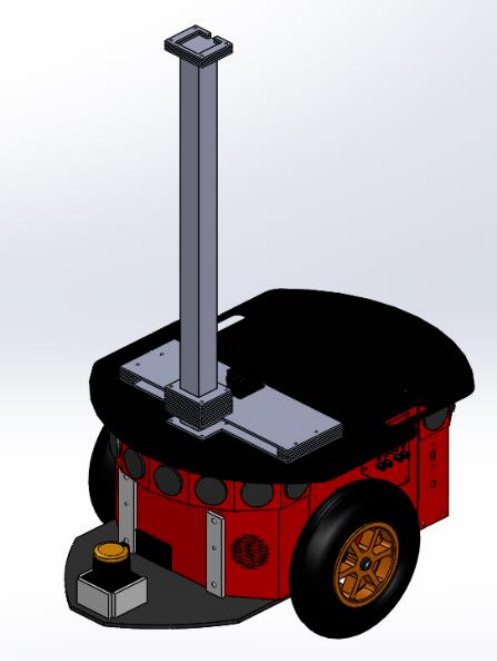


Figura 27 Modelo CAD del robot con los sensores laser e IMU con sus respectivas bases. Imagen tomada por el autor

En cuanto a la base para la cámara ZED, se utilizó una que el grupo de investigación GIDAM de la universidad Militar Nueva Granada previamente había diseñado y fabricado con acrílico. Este garantiza la sujeción de las cámaras al soporte buscando que quede lo más ajustado posible para evitar vibraciones.

Con los sensores ya montados y luego de haber simulado, se procedió a lanzar los nodos respectivos para cada sensor. Al hacerlo se hizo evidente que era necesaria una tarjeta gráfica de mayor capacidad que la Intel GMA 4500 HD para obtener datos de la ZED. Por tal motivo se decidió utilizar un DELL Inspiron 15-7559, que cuenta con una GTX 960M, y demás características explicadas en la sección 3.2.5, que hacen que sea mucho más eficiente para el trabajo.

Se diseñó también una base para este computador (Figura 28) que evite riesgos potenciales, principalmente debidos a posibles choques o que el robot deba desplazarse en un ambiente muy inclinado en el que el computador pueda caerse.

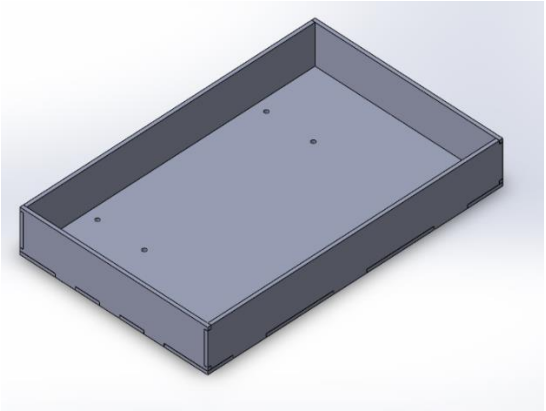


Figura 28 Base para Pc Dell Inspiron 15-7559. Imagen tomada por el autor

Al unir todas las partes el robot se ve así (Figura 29)

Figura 29 Robot completo. Imagen tomada por el autor

Inicialmente se planteó la posibilidad de utilizar los 2 computadores paralelamente, de forma tal que el Mamba tomará los datos de los sensores y que el procesamiento fuese hecho por el Dell. La comunicación entre los dos ser realizó por medio de un router que generaba una red local a la que se conectaban ambos computadores.

El computador Mamba debía correr el nodo de RosAria, que da acceso a los drivers de los motores y a la información de odometría, y el nodo de URG que da acceso a la información entregada por el Hokuyo. Mientras que el Dell debería correr los nodos de Zed wrapper, que inicializa las cámaras con todos sus parámetros, el nodo de advanced navigation, que entrega toda la información ya procesada de la central

inercial y del GPS, y el nodo de Google Cartographer, que tomaría los datos seleccionados y generaría un mapa.

Utilizando el comando ssh de ROS fue posible correr los nodos del Mamba desde el Dell haciendo uso de la conexión inalámbrica existente entre los dos por estar conectados a la misma red local generada por el router.

Este plan inicial tuvo una serie de complicaciones. El primer problema fue el alcance del router, pues a pesar de seguir las indicaciones del fabricante no se conseguía un alcance superior a los 7 metros, lo cual es una distancia insuficiente para los fines planteados. Se genero un apantallamiento de la señal emitida por el router utilizando un material conductor con forma cónica, esto con el fin de intentar dirigir la señal, sin embargo, con esto solo se logró extender el alcance aproximadamente 50 cm. Para dar solución a este problema se decidió crear una red local desde el Dell como Hotspot, lo que permite la comunicación entre los dos computadores que estarán siempre a escasos centímetros de distancia.

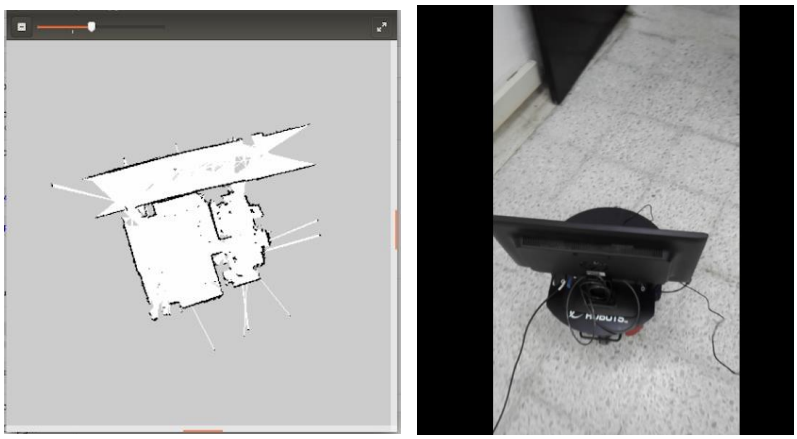
Otra dificultad importante se presentó en Google Cartographer, como se explicó en la sección 0, por lo que se decidió cambiar de algoritmo de SLAM, utilizando uno más sencillo (Gmapping), para luego robustecerlo en la parte de navegación.

La ZED entrega datos de odometría visual, que pretendían ser usados para compararse con la odometría entregada por RosAria, pero los datos entregados por la ZED son fuertemente afectados por factores ambientales, en especial la luz, por lo que se decide no utilizarlos.

Al cambiar el paquete que realizaría el mapeo, fue necesario ignorar los datos de la ZED para SLAM, pues este método trabaja con datos únicamente 2D, de forma que una nube de puntos 3D no tiene cabida.

El mayor problema se presentó cuando se intentó correr el nodo de Gmapping, pues este no generaba ningún tipo de mapa a pesar de tener los datos requeridos, con las configuraciones necesarias según los autores. Revisando en detalle el código y utilizando la ayuda del compilador de ROS, se hizo evidente que el problema no estaba en los datos ingresados sino en el manejo del paquete. Luego de una inspección profunda en la bibliografía y en los foros de ayuda de ROS, además de una serie de pruebas, se pudo concluir que al utilizar ssh se tienen dos parámetros de Ros time que generan incompatibilidades en la lectura de los datos. La manera más sencilla de solucionarlo es correr todos los nodos en un solo computador.

Por practicidad con el tema de conexiones internas del robot se decidió utilizar el computador Mamba. Desde este fue posible correr el nodo de SLAM, obteniendo mapas claros y que corresponden al entorno en el que se desplaza el robot (Figura 30 a).



a) b)

Figura 30 a) Mapa generado desde el Mamba en Lab de Robótica UMNG b) Robot con pantalla alámbrica. Imagen tomada por el autor

Se realizaron pruebas en el tercer piso de los bloques C, D y E de la sede Calle 100 de la Universidad Militar, un espacio de pasillos sobre los que es posible realizar un recorrido circular, lo que permite evaluar la capacidad del algoritmo para cerrar bucles (Loop-Closure). En la Figura 31 se puede apreciar como el algoritmo cierra a la perfección el ciclo, pero añadiendo un ángulo en la parte inferior izquierda. Los triángulos grises que se ven a lo largo de diferentes tramos son debidos a falta de información, pues el alcance del láser no llega hasta el otro extremo del pasillo y la información de lo que tiene justo en frente es desconocida.

Figura 31 Prueba de Loop-Closure. Imagen tomada por el autor

Los mapas tomados con el Mamba entregaron los resultados esperados, no obstante, el manejo de una pantalla conectada alámbricamente a la corriente (Figura 30 b), la capacidad de procesamiento para el manejo de navegación y la escasa capacidad de la GPU, hicieron que se optara por desmontar el computador embebido y poner en su lugar el computador Dell.

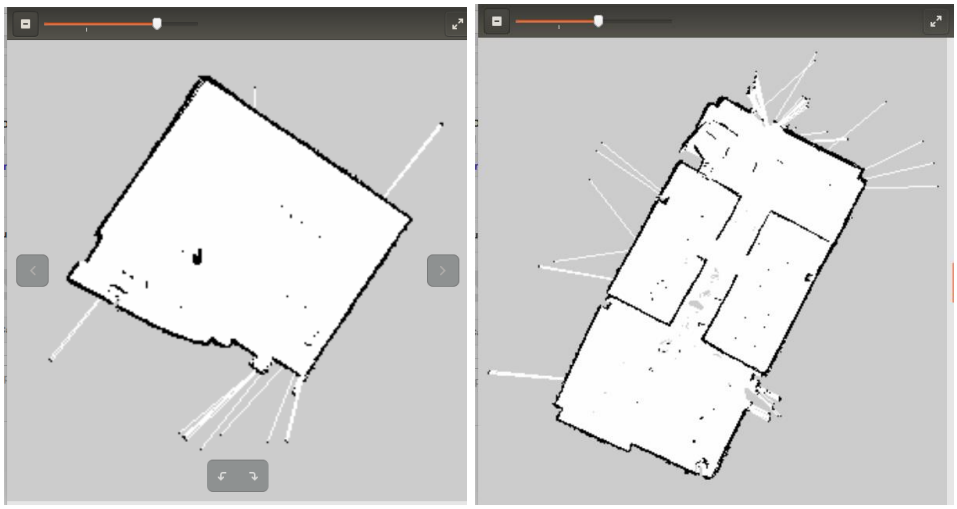
Una vez habilitados los permisos para poder manipular los drivers de los motores desde este nuevo computador fue posible correr el nodo de SLAM, implementando además control remoto utilizando un control Sony DualShock 3 (Figura 32) que se conecta por medio de Bluetooth al Pc y envía comandos a más de 20 metros.

Figura 32 Control Sony DualShock 3 utilizado para dirigir al robot. Imagen tomada por el autor

Se realizaron pruebas mapeando diferentes entornos (Figura 33). De estas pruebas se pueden resaltar dos riesgos importantes: 1) el láser tiene problemas con los objetos que reflejan. Bien sean pinturas oscuras y brillantes o vidrios translucidos, el láser no los detecta como obstáculos. 2) Objetos demasiado delgados, como patas de sillas o mesas, no siempre son tomados como obstáculos.

Adicionalmente se puede decir que el algoritmo interpreta algunos puntos indeterminados en el láser como obstáculos en el máximo rango de este, por lo que en la mayoría de los mapas se pueden apreciar puntos que no concuerdan con el entorno real. Sin embargo, estos no tienen afectan en la navegación.

Otro punto para destacar es el comportamiento del algoritmo en espacios donde el láser realiza mediciones muy similares, como pasillos largos y sin distinciones en puntos intermedios. Allí el algoritmo da prioridad a las lecturas del láser antes que a las de la odometría, asumiendo que las ruedas se están deslizando y el vehículo se encuentra detenido en el espacio.



a) b)

Figura 33 a) Mapa de un aula de clase UMNG, calle 100 b) Mapa de Salas de Estudio UMNG, calle 100. Imagen tomada por el autor

Se realizó una prueba tomando un mapa a la máxima velocidad del robot (1,2 m/s) para observar que tan afectados se ven los resultados por este parámetro. Al comparar la Figura 30 a, con la Figura 34 puede verse como la cantidad de puntos que no tienen concordancia con el mapa real aumento. Pero, como se menciono anteriormente, estos puntos no afectan la navegación, por lo que se puede decir que no hay ningún inconveniente en que el robot opere en su máxima velocidad.

Figura 34 Mapa tomado a velocidad máxima del Pioneer 3DX en Lab de Robótica UMNG. Imagen tomada por el autor

Finalizada la etapa de SLAM, es posible pasar a la etapa de guiado. Para ella se tomaron en consideración varios tipos de algoritmos, tomando en cuenta que en la literatura no está muy clara la diferencia entre guiado y navegación; algunos autores las definen como iguales, mientras que otros definen al guiado como una parte de la navegación. Finalmente, como se explica en la sección 2.4, se optó por la aproximación de ventana dinámica, con procesos de optimización en el planeamiento de trayectorias. Esta técnica es aplicada en su totalidad en el paquete de Navigation Stack.

Este paquete ofrece la posibilidad de implementar la técnica, con todos sus parámetros configurables, de forma tal que cada usuario puede aplicar un mismo método acomodado a sus necesidades y a sus objetivos.

Al generar todos los parámetros de configuración se presentaron ciertas dificultades, como la no convergencia de las trayectorias, la aparente obstrucción de vías despejadas, el choque con obstáculos, la inclusión de datos tipo PointCloud2 y la falta de localización sobre los mapas generados.

La convergencia en las trayectorias pudo ser solucionada al darle una mayor ponderación a la importancia de que el robot siga la trayectoria generada por el planeador global, sobre el local. Esto hace que, a pesar de los obstáculos dinámicos existentes, el robot siempre intente acercarse a la meta.

En cuanto a la dificultad del robot de pasar por lugares estrechos, se pudo destacar la influencia de la inflación de los obstáculos y del robot. Si el robot mide aproximadamente 40cm de diámetro y se infla este 10 cm radialmente, se necesitará un especio de 60 cm para que el robot pueda pasar, pero si

adicionalmente se inflan también los obstáculos 10 cm, y el robot pretende pasar por una puerta, requerirá un espacio mínimo de 80cm para poder pasar, lo que representa el doble de espacio del realmente requerido.

Además, el algoritmo, al calcular la ventana dinámica, utiliza un parámetro de tiempo de simulación, es decir cuánto tiempo simulara cada una de las velocidades dentro de la ventana para decidir si son admisibles o no. Tiempos de simulación muy cortos harán que se generen muy pocas combinaciones de posibles velocidades por lo que el paso por lugares estrechos se dificulta aún más. Mientras que tiempos de simulación demasiado grandes harán que el costo computacional se eleve considerablemente.

El choque con obstáculos se debe a la falta de visibilidad de los mismos. Inicialmente el algoritmo se corrió únicamente utilizando el sensor laser, y se pretendió agregar las cámaras estereoscópicas. El nodo encargado de su funcionamiento entrega los datos provenientes del sensado profundo en un mensaje del tipo PointCloud2, este parece tener problemas con el mapa de costos, pues todo punto mostrado en la nube de puntos es tomado como obstáculo sin importar su altura (Figura 35). Como solución parcial para este problema se propone el uso de la nube de puntos entregada por los sonares incorporados en el robot, lo cual brinda información adicional acerca de la altura de los obstáculos.

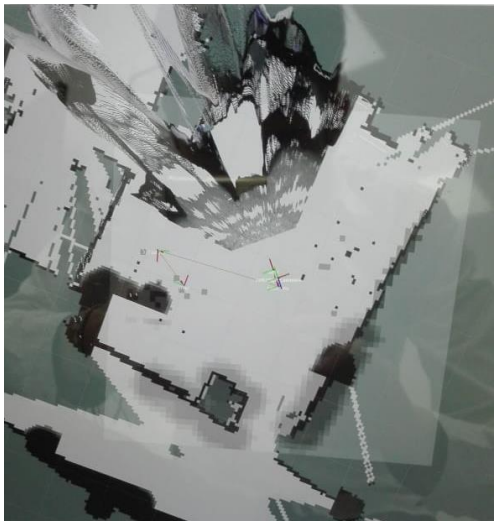


Figura 35 Mapa de costos incluyendo la información de la nube de puntos de la ZED. Imagen tomada por el autor

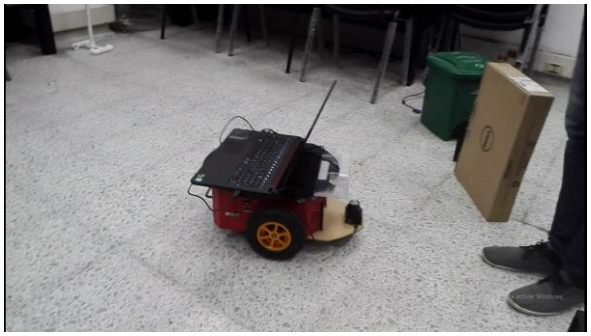
Finalmente, de la localización dentro de los mapas se puede decir que esta funciona bastante bien mientras se está empezando a generar el mapa, más cuando se acerca a la meta esta empieza a variar, produciendo deslizamientos entre el marco de referencia odom con respecto a map. Esto provoca que el robot continúe moviéndose a pesar de haber llegado a la meta, y al alejarse de la misma trata de corregir, generando nuevos deslizamientos entre los marcos de referencia. Se planteó como una posible solución generar parámetros dinámicos, es decir que

cuando el robot este cerca de la meta, el algoritmo se comporte de manera distinta a como lo haría lejos de ella, pero para implementar esta solución seria necesario modificar el código base del algoritmo. Por tanto, se propuso identificar el problema como un mínimo local, y solucionarlo dando una tolerancia al mínimo global, es decir, se estableció un error admisible con respecto a la meta global, de forma tal que el robot se detenga cuando la distancia hasta la meta sea tolerable.

Para calcular el valor del error admisible fue necesario establecer que tan lejos estaba el robot de alcanzar la meta. Debido a que la meta se define de manera relativa a la posición del robot, y los ambientes en que se desarrollaron las pruebas son poco estructurados, el calculo del error se hizo de manera aproximada utilizando herramientas graficas del software RVIZ.

Al promediar los errores de 10 pruebas se obtuvo un valor de 0.45 m, tomando en cuenta esto y que el radio del robot es de 0.2 m aproximadamente, se establece como mínimo global admisible inicialmente un valor de 0.5 m, sin embargo, al realizar pruebas experimentales se evidenció que este puede disminuir hasta 0.4 m.

Se realizo una prueba a la evasión de obstáculos dinámicos. En ella se dejó para el robot un solo espacio posible por el que transitar, y en dicho espacio se ubicó una persona (Figura 36 a), luego de que el robot evaluara que no existía ruta posible, la persona en medio dio un paso aparte (Figura 36Figura 37 b) y el robot consiguió cruzar (Figura 36 c).



a)

b)

c)

Figura 36 Prueba evasión de obstáculos dinámicos. Imagen tomada por el autor

Debido a que el proyecto pretende ser base de un proyecto más robusto y enfocado a labores de agricultura en espacios abiertos, se implementó también un filtro extendido de Kalman que permite la fusión sensórica con datos de IMU y GPS para obtener un dato de odometría más certero y menos susceptible a los deslizamientos de las ruedas en suelos lodosos o superficies demasiado lisas.

Para su correcta implementación se utilizó el paquete de Robot Localization, el cual cuenta con un nodo para la transformación de latitud y longitud al sistema UTM, y con un nodo de fusión sensórica.

La primera aproximación obtenida del EKF arrojo datos incoherentes, por lo que se detallaron los datos de IMU y GPS, y se revisó en profundidad la información de Advanced Navigation, notando una sugerencia importante de los autores, que consiste en calibrar los sensores antes de su uso. Se realizaron pruebas antes y después de la calibración para corroborar su importancia.

El dato entregado por el GPS es un dato de tipo satelital, que contiene Latitud, Longitud y Altitud. Para este estudio se omite la altitud, pues los datos se toman en un plano 2D, en el que las variaciones de altura son insignificantes, y adicionalmente los datos de altitud entregados por un GPS no son del todo fiables debido a la forma en que estos dispositivos adquieren las señales.

En la Tabla 1 se presentan los resultados obtenidos.

Tramo	Distancia Real (m)	Distancia Calibrada (m)	Distancia Sin Calibrar (m)	Error Absoluto Calibrado	Error Relativo Calibrado (m)	Error Absoluto Sin Calibrar	Error Relativo Sin Calibrar (m)
1	8,35	7,9511	5,0217	4,7768%	0,3989	39,8604%	3,3283
2	10	6.0010	5,3065	39.9901%	3,9990	46,9354%	4,6935
3	3	2,4678	2,5996	17,7406%	0,5322	13,3478%	0,4004

Tabla 1 Datos experimentales tomados del GPS antes y después de calibrar

Es evidente que existe una mejoría, sin embargo, el error final luego de la calibración continúa teniendo una magnitud importante. Parte del error se puede atribuir la conversión a UTM, pues este está diseñado para entregar medidas a nivel del mar y las medidas fueron tomadas a 2600 msnm aproximadamente. Para el tramos 2 se asume que existe algún tipo de oclusión hacia los satélites, esta puede ser un edificio o simplemente una nube.
En la Figura 37 puede apreciarse más fácilmente la diferencia entre los datos calibrados y sin calibrar. El recorrido calibrado se acerca mucho más al realizado, sin cruces ni retrocesos.

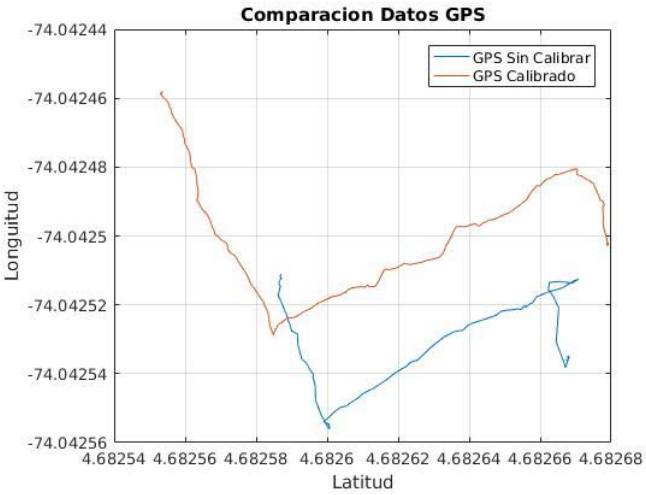
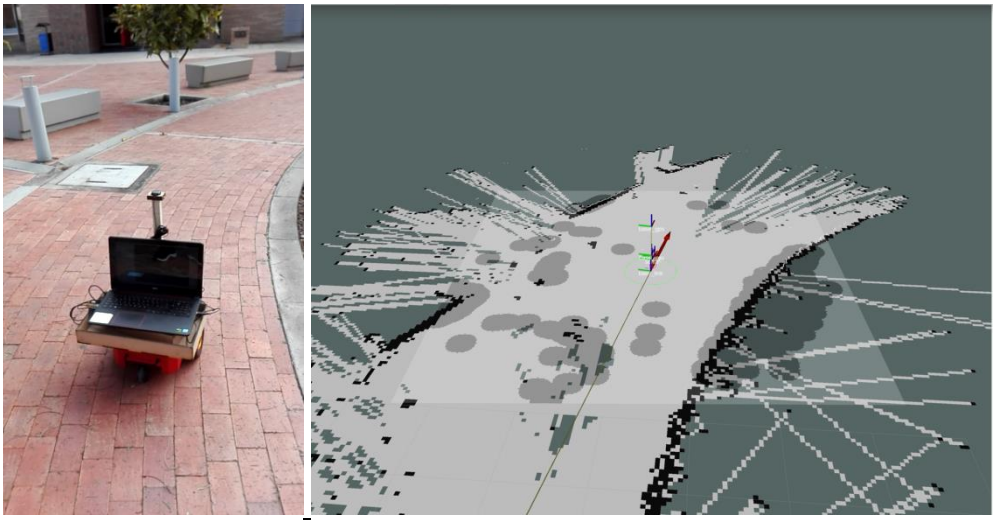


Figura 37 Grafica de contraste entre los datos de GPS antes y después de la calibración. Imagen tomada por el autor

Se realizaron pruebas en exteriores verificando el funcionamiento del algoritmo, con la persistencia de las dificultades en la localización y en el mapeo (Figura 38). Con esta prueba se pudo comprobar el correcto funcionamiento del nodo de EKF para la fusión sensórica, y adicionalmente se obtuvo un resultado particular, pues en entornos en donde el láser no percibe ningún obstáculo se repite el comportamiento visto en pasillos. El algoritmo asume que las ruedas se están deslizando y que el vehículo está detenido, y por esto el resultado en la parte superior del mapa (Figura 38 c) se ve distorsionado.



a) b)

c)

Figura 38 a) Robot en entorno abierto b) Vista de Rviz del algoritmo corriendo c) Mapa final. Imagen tomada por el autor

Los parámetros que se modificaron de los valores por defecto del algoritmo, y con los cuales se obtuvieron los resultados antes mencionados se presentan en la Tabla 2.

Parámetro	Función	Valor	Unidades
obstacle_range	Distancia máxima a la que el sistema insertará obstáculos	2.5	m
cost_factor	Factor multiplicativo del costo de cada píxel	0.55	-
neutral_cost	Define que tan pronunciadas serán las curvas para evitar un obstáculo	66	-
lethal_cost	Determina el riego que toman las trayectorias. Valores bajos harán que considere todas las trayectorias como "muy peligrosas" y las descarte	253	-
sim_time	Tiempo en el que se simularan las distintas velocidades en la ventana dinámica	5	seg
xy_goal_tolerance	Tolerancia de error en distancia respecto a la meta	0.3	m
yaw_goal_tolerance	Tolerancia de error en ángulo respecto a la meta	0.15	rad
robot_radius	Radio de la circunferencia en la que se encuentra inscrito el robot	0.23	m
inflation_radius	Radio de seguridad, con el que se inflaran los obstáculos	0.4	m
static_map	Define si se utiliza un mapa previo o no	false	-

Tabla 2 Parámetros de Navigation Stack

A manera de resumen se puede decir que a lo largo de la implementación y las pruebas se presentaron varias dificultades, las mas importantes relacionadas con la implementación del algoritmo de SLAM y con la comunicación entre los diferentes dispositivos. La solución para estas dificultades se dio realizando una conexión directa del PC Dell al robot Pioneer, evitando problemas de sincronización en los tiempos de ROS en dos maquinas diferentes, y permitiendo que todos los dispositivos estén conectados a una sola unidad de procesamiento.

En este capítulo se pudo apreciar el desarrollo del proyecto paso a paso, desde su implementación hasta sus pruebas finales. Como resultado se obtuvo un algoritmo claramente capaz de realizar mapas de su ambiente y de guiar al robot en entornos sencillos.

6 Conclusiones y Trabajos Futuros

El presente trabajo de grado tuvo como objetivo principal la navegación y mapeo en entornos desconocidos, por parte de un robot móvil diferencial. Para su correcto desarrollo se realizó una amplia búsqueda bibliográfica, en la que se encontró que el problema de mapeo y localización simultanea se viene planteando hace más de 20 años, por lo que la soluciones que se han propuesto son muchas y muy distintas.

Es de resaltar que el proyecto en su totalidad fue desarrollado con ROS, lo que facilito enormemente la labor, pues su diseño modular permite trabajar cada parte del trabajo por separado, corregir errores y unir a un todo. Esto sin contar además con la gran cantidad de información disponible del manejo de las herramientas útiles para SLAM y Navegación, desde código abierto.

ROS ofrece la posibilidad de acceder a la información de múltiples maneras y de ajustarla según las necesidades de cada usuario. Cuenta con un visualizador (Rviz) eficaz, y que a la vez permite la interacción con los datos. Cuenta además con un poderoso simulador (Gazebo) capaz de generar lecturas de distintos tipos de sensores en ambientes creados. La interacción entre estas 3 herramientas, ROS, Rviz y Gazebo, generan en conjunto la capacidad de simular entornos con escasas diferencias del mundo real.

En cuanto a la implementación del algoritmo de SLAM es importante recalcar la importancia del uso de la dinámica del robot en este, para facilitar los procesos de navegación posteriores. También cabe destacar que existen muchos factores que intervienen en el correcto desarrollo de un mapa, por ejemplo, si se usan métodos visuales la luz ambiente afectara en gran magnitud medidas como la odometría. Otro ejemplo claro de los factores que pueden afectar es la similitud entre ciertas partes del entorno, estas afectan en gran medida la localización. Si se trabaja con LandMarks estas similitudes pueden provocar que el robot asuma que se encuentra en un lugar diferente. Por otra parte, si se utilizan métodos de aproximación por cuadricula de ocupación y se transita por lugares sin evidentes diferencias se corre el riesgo de que el algoritmo asuma deslizamiento en las ruedas y por tanto empiece a superponer el mapa en el mismo lugar, aunque el robot avance.

Para nuestro caso particular, en el trabajo con la Spatial Advanced Navigation, se encontró muy relevante la calibración del instrumento antes de cada toma de datos. Esto en implementaciones de mayor magnitud puede generar problemas.

Para finalizar, en el campo de la navegación, se realizó una importante aproximación, se consiguió evadir obstáculos estáticos y dinámicos, mientras que se buscaba alcanzar una meta propuesta, esto sin interrumpir en ningún momento la labor de mapeo.

El primer objetivo a futuro es corregir los problemas a nivel de visión 3D e implementar diferentes técnicas para poder comparar y establecer cuál es la mejor para este caso de estudio.

El proyecto finaliza con una clara orientación a ser fácilmente portable, por lo que se espera poder llevarlo a una plataforma de mayor magnitud (Figura 39) que fue construida en el marco del proyecto INV-ING-2637: Aproximación a la autonomía de una plataforma robótica dedicada a labores de agricultura de precisión.

Se espera que este robot sea capaz de navegar autónomamente y a la par lleve a cabo labores de agricultura tales como riego, abono, y control de plagas.

Figura 39 Robot Ceres. Desarrollo del proyecto de investigación ING-26-37 de la UMNG. Imagen tomada por el autor

Bibliografía

[1] R. A. d. l. L. Española, «Diccionario de la lengua Española,» octubre 2014. [En línea]. Available: http://dle.rae.es/?id=WYRlhzm. [Último acceso: 27 mayo 2017].

[2] J. J. C. Robotica, Ciudad de Mexico: Pearson, 2006.

[3] G. Lazea y A. E. Lupu, «Aspects on path planning for mobile robots,» *TEMPUS M-JEP,* pp. 19-23, 1997.

[4] M. Aguilera Hernández, M. Bautista y J. Iruegas, «Diseño y Control de Robots Móviles,» Instituto Tecnológico de Nuevo Laredo , Nuevo Laredo, 2007.

[5] V. A. M. M. y F. W. , «A NEW TECHNIQUE IN MOBILE ROBOT SIMULTANEOUS LOCALIZATION AND MAPPING,» *Revista Controle & Automação/,* vol. 17, nº 2, 2006.

[6] E. C. Quiroga, L. M. Martin y A. U. Caisedo, «La estereoscopía, métodos y aplicaciones en diferentes áreas del conocimiento,» *Rev. Cient. Gen. José María Córdova,* pp. 201-219, 2015.

[7] J. B. L. M. P. y E. C. Bravo, «Segmentación y parametrización de líneas en datos láser 2D basado en agrupamiento por desplazamiento de media,» *Revista Tecnura,* vol. 17, nº 37, pp. 84-98, 2013.

[8] H. Parra, L. H. Rios y M. Bueno, «NAVEGACIÓN DE ROBOTS MÓVILES MEDIANTE COMPORTAMIENTOS UTILIZANDO LOGICA DIFUSA,» *Scientia et Technica,* vol. XIII, nº 34, pp. 79-84, 2007.

[9] R. Carrazo y A. Cipriano , «Desarrollo de un sistema de navegación para robots móviles, basado en logica difusa,» *XV Congreso de la Asociasión Chilena de Control Automatico,* 2002.

[10] J. Godjevac, «A Learning Procedure for a Fuzzy System: Application to Obstacle Avoidance,» *International Symposium on fuzzy logic,* pp. 142 - 148, 1995.

[11] J. Godjevac, «Comparative study of fuzzy, control neural network control and neuro-fuzzy control,» ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE, Lausana, 1995.

[12] E. Sierra, A. Hossian y G. Monte, *CONTROLADORES INTELIGENTES APLICADOS A ROBÓTICA MÓVIL,* Cordoba.

[13] C. Becker, J. Salas, K. Tokusei y J.-C. Latornbe, «Reliable Navigation Using Landmarks,» *IEEE lnternatlonal Conference on Robotics and Automation,* pp. 401-406, 1995.

[14] F. E. SISTLER, «Robotics and Intelligent Machines in Agriculture,» *IEEE JOURNAL OF ROBOTICS AND AUTOMATION,* vol. 3, nº 1, pp. 3-6, 1987.

[15] P. C. Jhonson, Farm Inventions in the Making of America, Wallace Homestead Book Co, 1976.

[16] M. Jhonson, "Automation in citrus sorting and packing", Chicago: Chicago Press, 1985.

[17] G. Kranzler, «Applying digital image processing in agriculture,» *Agricultural Engineering,* vol. 66, pp. 11-13, 1985.

[18] H. Hwang y F. Sistler, «"The implementation of a robotic manipulator on a pepper transplanting machine,» de *CAD/CAM, Robotics Automat*, St Louis, 1985.

[19] N. Kondo y N. Kawamura, *Methods of detecting fruit by visual sensor attached to manipulator,* Kyoto: Laboratory of Agricultural Machinery, Kyoto University, 1990.

[20] N. Kawamura, «Japan's technology farm,» *Robotics and Intelligent Machines in Agriculture,* pp. 52-62, 1983.

[21] A. Ollero Baturone, Robotica: Manipuladores y robots móviles, Barcelona: Alfaomega, 2001.

[22] S. Ortigoza, J. R. García Sánchez, V. R. Barrientos Sotelo y M. A. Molina Vilchis, «UNA PANORÁMICA DE LOS ROBOTS MÓVILES,» *Telematique,* vol. 6, nº 3, pp. 1-14, 2007.

[23] H. Secchi, «Una Introducción a los Robots Móviles,» *AADECA,* pp. 1-78, 2008.

[24] F. Candelas Herías y J. Corrales Ramón , Comparativa algoritmos fusion MoCap-UWB, San Vicente: Universidad de Alicante, 2008.

[25] D. Gallardo López, APLICACIÓN DEL MUESTREO BAYESIANO EN ROBOTS MÓVILES: ESTRATEGIAS PARA LOCALIZACIÓN Y ESTIMACIÓN DE MAPAS DEL ENTORNO, San vicente: UNIVERSIDAD DE ALICANTE, 1999.

[26] D. Salmond y N. Gordon, An introduction to particle filters, Stellenbosch University, 2005.

[27] R. E. Kálmán, «A New Approach to Linear Filtering and Prediction Problems,» *Journal of Basic Engineering,* vol. 82, nº D, pp. 35-45, 1960.

[28] H. W. Sorenson, «Least-squares estimation: from Gauss to Kalman,» *IEEE spectrum,* vol. July, pp. 63-68, 1970.

[29] «A Comparitive Study Of Kalman Filter, Extended Kalman Filter And Unscented Kalman Filter For Harmonic Analysis Of The Non-Stationary Signals,» *International Journal of Scientific & Engineering Research,* vol. 3, nº 7, pp. 1-9, 2012.

[30] M. E. Aranda Romasanta, Estudio y aplicación del Filtro de Kalman en fusión de sensores en UAVs, Sevilla: Universidad de Sevilla, 2017.

[31] L. Turner, An Introduction to Particle Filtering, Lancaster University, 2013.

[32] A. M. Castro Pescador, Fusión Sensórica INS/ GPS para Navegación en Plataformas Moviles, Bogotá: UNIVERSIDAD MILITAR NUEVA GRANADA, 2013.

[33] A. Doucett, N. de Freitas, K. Murphy y S. Russell, «Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks,» de *UNCERTAINTY IN ARTIFICIAL INTELLIGENCE PROCEEDINGS*, San Francisco, 2000.

[34] «Rao-Blackwellized Monte Carlo Data Association for Multiple Target Tracking,» de *Seventh International Conference on Information Fusion*, Stockholm , 2004.

[35] S. Riisgaard y M. Rufus Blas, SLAM for Dummies.

[36] G. Dissanayake, H. Durrant-Whyte y T. Bailey, «A Computationally Efficient Solution to the Simultaneous Localisation and Map Building (SLAM) Problem,» de *Proceedings of the 2000 IEEE Intemational Conference on Robotics & Automation* , San Francisco, CA , 2006.

[37] B. S. LEMUS QUIROGA, J. F. CLAVIJO MAYORGA y E. M. YOSA VELÁSQUEZ, RECONSTRUCCIÓN TRIDIMENSIONAL DE UN AMBIENTE NO ESTRUCTURADO Y ESTÁTICO USANDO UNA PLATAFORMA MÓVIL TELE OPERADA MEDIANTE ROS Y SU REPRESENTACIÓN EN UN ENTORNO VIRTUAL, Bogota: Universidad Militar Nueva Granada, 2017.

[38] B. Williams, M. Cummins, J. Neira, P. Newman, I. Reid y J. Tardos, *A comparison of loop closing techniques in,* Zaragoza, Oxford: Universidad de Zaragoza, Spain, University of Oxford, UK.

[39] S. Stramigioli, D. Dresscher, J. F. Broenink y M. Poel, «Reuse-oriented SLAM Framework using Component-based Approach,» University of Twente , Enschede, 2017.

[40] M. J. a. W. G. F. a. P. D. Milford, «RatSLAM: a hippocampal model for simultaneous localization and mapping,» *Robotics and Automation, 2004.*

Proceedings. ICRA'04. 2004 IEEE International Conference, vol. 1, pp. 403--408, 2004.

[41] U. G. Villaseñor Carrillo, M. A. González Aguirre, A. Sotomayor Olmedo, E. Gorrostieta Hurtado, J. C. Pedraza Ortega, J. Vargas Soto y S. Tovar Arriaga, «Desarrollo de un sistema de navegación para robots móviles mediante diferentes patrones de comportamientos,» de *VIII Congreso Internacional sobre Innovación y Desarrollo Tecnológico*, Cuernavaca Morelos, 2010.

[42] R. Carrasco y A. Cipriano, «Sistema de guiado para un robot móvil, basado en lógica difusa,» de *XV Congreso de la Asociación Chilena de Control Automatio*, Santiago, 2002.

[43] H. Korrapati, Loop closure for topological mapping and navigation with omnidirectional images, Universit´e Blaise Pascal - Clermont-Ferrand II, 2013.

[44] T. Duckett y U. Nehmzow, Performance Comparison of Landmark Recognition Systems for Navigating Mobile Robots, American Association for Artificial Intelligence, 2000.

[45] O. Brock y 0. Khatib, «High-speed Navigation Using the Global Dynamic Window Approach,» de *International Conference on Robotics & Automation* , Detroit, 1999.

[46] S. Thrun y e. al., «Map Learning and High-Speed Navigation in RHINO,» de *Fifteenth National Conference on Artificial Intelligence*, Madison, 1998.

[47] D. Fox, W. Burgard y S. Thrun, «The Dynamic Window Approach to Collision Avoidance,» *IEEE Robotics and Automation Magazine,* vol. 4, nº 1, pp. 23-33, 1997.

[48] Open Source Robotics Foundation, «ROS,» [En línea]. Available: http://www.ros.org/. [Último acceso: 2018 Julio 21].

[49] F. Ferland, D. Letourneau, A. Aumont, J. Fremy, M.-A. Legault, M. Lauria y F. Michaud, «Natural Interaction Design of a Humanoid Robot,» *Journal of Human-Robot Interaction,* vol. 1, nº 2, pp. 118-134, 2012.

[50] ROS Answers, «PR2 drifts over time in Gazebo,» Askbot version 0.7.58, 22 Agosto 2012. [En línea]. Available: https://answers.ros.org/question/42011/pr2-drifts-over-time-in-gazebo/. [Último acceso: 08 Agosto 2018].

[51] T. Moore, «robot_localization wiki,» Sphinx, 2016. [En línea]. Available: http://docs.ros.org/kinetic/api/robot_localization/html/index.html. [Último acceso: 23 Julio 2018].

[52] T. Moore, «navsat_transform_node,» Sphinx , 2016. [En línea]. Available: http://docs.ros.org/kinetic/api/robot_localization/html/navsat_transform_node.html. [Último acceso: 23 Julio 2018].

[53] T. Moore y D. Stouch, «A Generalized Extended Kalman Filter Implementation for the Robot Operating System,» de *Advances in Intelligent Systems and Computing 13*, Springer, 2016, pp. 335-348.

[54] Google, «Cartographer,» Read the Docs, 2018. [En línea]. Available: https://google-cartographer.readthedocs.io/en/latest/. [Último acceso: 24 julio 2018].

[55] W. Hess, D. Kohler, H. Rapp y D. Andor, «Real-Time Loop Closure in 2D LIDAR SLAM,» de *IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, 2016.

[56] S. Kohlbrecher, O. von Stryk, J. Meyer y U. Klingauf, «A flexible and scalable SLAM system with full 3D motion estimation,» de *IEEE International Symposium on Safety, Security, and Rescue Robotics*, Kyoto, 2011 .

[57] S. Kohlbrecher, «hector_mapping,» Creative Commons Attribution 3.0, 21 08 2012. [En línea]. Available: http://wiki.ros.org/hector_mapping. [Último acceso: 24 julio 2018].

[58] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, U. Klingauf y O. von Stryk, «Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots,» de *Robot World Cup XVII*, Eindhoven, 2014.

[59] G. Grisetti, C. Stachniss y W. Burgar, «GMapping,» BSD-3-Clause, [En línea]. Available: https://openslam-org.github.io/gmapping. [Último acceso: 24 Julio 2018].

[60] G. Grisetti, C. Stachniss y W. Burgard, «Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling,» de *IEEE International Conference on Robotics and Automation*, 2005.

[61] K. Konolige, G. Grisetti, R. Kummerle, B. Limketkai y R. Vincent, «Efficient Sparse Pose Adjustment for 2D Mapping,» de *IEEE/RSJ International Conference on Intelligent Robots and Systems* , Taipei, 2010.

[62] J. Machado Santos, D. Portugal y R. Rocha, «An Evaluation of 2D SLAM Techniques Available in Robot Operating System,» de *11th IEEE Int. Symp. on Safety, Security, and Rescue Robotics*, Linköping, 2013.

[63] B. Steux y O. El Hamzaoui, «CoreSLAM : a SLAM Algorithm in less than 200 lines of C code,» de *11th International Conference on Control, Automation, Robotics and Vision, ICARCV*, Singapore, 2010.

[64] L. Carlone, R. Aragues, J. Castellanos y B. Bona, «A Linear Approximation for Graph-based Simultaneous Localization and Mapping,» de *Robotics: Science and Systems*, Los Angeles, 2011.

[65] R. Gomez Ojeda, D. Zuñiga-Noël, F.-A. Moreno, D. Scaramuzza y J. Gonzalez-Jimenez, «PL-SLAM: a Stereo SLAM System through the Combination of Points and Line Segments,» de *IEEE International Conference on Robotics and Automation*, Singapur, 2017.

[66] M. J. Milford y G. F. Wyeth, «SeqSLAM: Visual Route-Based Navigation for Sunny Summer Days and Stormy Winter Nights,» de *IEEE International Conference on Robotics and Automation*, Saint Paul, 2012.

[67] A. T. Coroiu y O. Hinton, A Platform for Indoor Localisation, Mapping, and Data Collection using an Autonomous Vehicle, Lund, Suecia: Lund University, 2017.

[68] J. E. HANDSCHINJ, «Monte Carlo Techniques for Prediction and Filtering of Non-Linear Stochastic Processes,» *Automatica,* vol. 6, pp. 555-563, 1970.

[69] N. J. Gordon, D. J. Salmond y A. F. M. Smith, «Novel approach to nonlinear/non-Gaussian Bayesian state estimation,» *IEE PROCEEDINGS-F,* vol. 140, nº 2, pp. 107 - 113, 1993.

[70] D. Fox , «Markov Localization,» 19 11 1999. [En línea]. Available: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume11/fox99a-html/node2.html. [Último acceso: 27 Julio 2018].

[71] D. Fox, W. Burgard, F. Dellaert y S. Thrun, «Monte Carlo Localization: Efficient Position Estimation for Mobile Robots,» de *Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence*, Orlando, 1999.

[72] M. Gruhler, «navigation,» Commons Attribution 3.0, 10 03 2017. [En línea]. Available: http://wiki.ros.org/navigation. [Último acceso: 31 Julio 2018].

[73] MobileRobots, «Pioneer 3 - DX,» 2011. [En línea]. Available: http://www.mobilerobots.com/Libraries/Downloads/Pioneer3DX-P3DX-RevA.sflb.ashx. [Último acceso: 16 Julio 2018].

[74] F. Espinosa, M. Salazar , D. Pizarro y F. Valdes, «Electronics Proposal for Telerobotics Operation of P3-DX Units,» de *Remote and Telerobotics*, Madrid, Nicolas Mollet, 2010, pp. 1-16.

[75] Hokuyo Automatic Co, «Hokuyo Products,» 2014. [En línea]. Available: https://www.hokuyo-aut.jp/search/single.php?serial=166. [Último acceso: 16 Julio 2018].

[76] Stereo Labs, «The camera that senses space and motion,» 2018 Stereolabs, 2018. [En línea]. Available: https://www.stereolabs.com/zed/. [Último acceso: 16 Julio 2018].

[77] Advanced Navigation, «Spatial,» 2018. [En línea]. Available: http://www.advancednavigation.com.au/product/spatial. [Último acceso: 16 Julio 2018].

[78] DELL, «Inspiron 15 Gaming,» 2018. [En línea]. Available: https://www.dell.com/en-us/shop/dell-laptops/inspiron-15-gaming/spd/inspiron-15-7559-laptop. [Último acceso: 30 Julio 2018].

[79] Adept MobileRobots LLC, «ARIA,» Omron, 25 Enero 2018. [En línea]. Available: http://robots.mobilerobots.com/wiki/ARIA. [Último acceso: 08 Agosto 2018].

[80] Adept MobileRobots, «Mobile Robots,» 2011. [En línea]. Available: http://www.mobilerobots.com/Libraries/Downloads/AmigoBot-AMGO-RevA.sflb.ashx. [Último acceso: 08 Agosto 2018].

Anexos

A. Código de Slam_Gmapping

```
1. /*
2.  * slam_gmapping
3.  * Copyright (c) 2008, Willow Garage, Inc.
4.  *
5.  * THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS
   CREATIVE
6.  * COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS
   PROTECTED BY
7.  * COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER
   THAN AS
8.  * AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.
9.  *
10.       * BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU
   ACCEPT AND AGREE TO
11.       * BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS
   YOU THE RIGHTS
12.       * CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH
   TERMS AND
13.       * CONDITIONS.
14.       *
15.       */
16.
17.      /* Author: Brian Gerkey */
18.      /* Modified by: Charles DuHadway */
19.
20.
21.      /**
22.
23.      @mainpage slam_gmapping
24.
25.      @htmlinclude manifest.html
26.
27.      @b slam_gmapping is a wrapper around the GMapping SLAM
   library. It reads laser
28.      scans and odometry and computes a map. This map can be
29.      written to a file using e.g.
30.
31.        "rosrun map_server map_saver static_map:=dynamic_map"
32.
33.      <hr>
34.
35.      @section topic ROS topics
36.
37.      Subscribes to (name/type):
38.      - @b "scan"/<a
   href="../../sensor_msgs/html/classstd__msgs_1_1LaserScan.html">sens
   or_msgs/LaserScan</a> : data from a laser range scanner
39.      - @b "/tf": odometry from the robot
40.
```

```
41.
42.      Publishes to (name/type):
43.      - @b "/tf"/tf/tfMessage: position relative to the map
44.
45.
46.      @section services
47.       - @b "~dynamic_map" : returns the map
48.
49.
50.      @section parameters ROS parameters
51.
52.      Reads the following parameters from the parameter server
53.
54.      Parameters used by our GMapping wrapper:
55.
56.      - @b "~throttle_scans": @b [int] throw away every nth laser
   scan
57.      - @b "~base_frame": @b [string] the tf frame_id to use for
   the robot base pose
58.      - @b "~map_frame": @b [string] the tf frame_id where the
   robot pose on the map is published
59.      - @b "~odom_frame": @b [string] the tf frame_id from which
   odometry is read
60.      - @b "~map_update_interval": @b [double] time in seconds
   between two recalculations of the map
61.
62.
63.      Parameters used by GMapping itself:
64.
65.      Laser Parameters:
66.      - @b "~/maxRange" @b [double] maximum range of the laser
   scans. Rays beyond this range get discarded completely. (default:
   maximum laser range minus 1 cm, as received in the the first
   LaserScan message)
67.      - @b "~/maxUrange" @b [double] maximum range of the laser
   scanner that is used for map building (default: same as maxRange)
68.      - @b "~/sigma" @b [double] standard deviation for the scan
   matching process (cell)
69.      - @b "~/kernelSize" @b [int] search window for the scan
   matching process
70.      - @b "~/lstep" @b [double] initial search step for scan
   matching (linear)
71.      - @b "~/astep" @b [double] initial search step for scan
   matching (angular)
72.      - @b "~/iterations" @b [int] number of refinement steps in
   the scan matching. The final "precision" for the match is
   lstep*2^(-iterations) or astep*2^(-iterations), respectively.
73.      - @b "~/lsigma" @b [double] standard deviation for the scan
   matching process (single laser beam)
74.      - @b "~/ogain" @b [double] gain for smoothing the likelihood
75.      - @b "~/lskip" @b [int] take only every (n+1)th laser ray for
   computing a match (0 = take all rays)
76.      - @b "~/minimumScore" @b [double] minimum score for
   considering the outcome of the scanmatching good. Can avoid
   'jumping' pose estimates in large open spaces when using laser
   scanners with limited range (e.g. 5m). (0 = default. Scores go up
```

```
   to 600+, try 50 for example when experiencing 'jumping' estimate
   issues)
77.
78.      Motion Model Parameters (all standard deviations of a
   gaussian noise model)
79.      - @b "~/srr" @b [double] linear noise component (x and y)
80.      - @b "~/stt" @b [double] angular noise component (theta)
81.      - @b "~/srt" @b [double] linear -> angular noise component
82.      - @b "~/str" @b [double] angular -> linear noise component
83.
84.      Others:
85.      - @b "~/linearUpdate" @b [double] the robot only processes
   new measurements if the robot has moved at least this many meters
86.      - @b "~/angularUpdate" @b [double] the robot only processes
   new measurements if the robot has turned at least this many rads
87.
88.      - @b "~/resampleThreshold" @b [double] threshold at which the
   particles get resampled. Higher means more frequent resampling.
89.      - @b "~/particles" @b [int] (fixed) number of particles. Each
   particle represents a possible trajectory that the robot has
   traveled
90.
91.      Likelihood sampling (used in scan matching)
92.      - @b "~/llsamplerange" @b [double] linear range
93.      - @b "~/lasamplerange" @b [double] linear step size
94.      - @b "~/llsamplestep" @b [double] linear range
95.      - @b "~/lasamplestep" @b [double] angular step size
96.
97.      Initial map dimensions and resolution:
98.      - @b "~/xmin" @b [double] minimum x position in the map [m]
99.      - @b "~/ymin" @b [double] minimum y position in the map [m]
100.     - @b "~/xmax" @b [double] maximum x position in the map [m]
101.     - @b "~/ymax" @b [double] maximum y position in the map [m]
102.     - @b "~/delta" @b [double] size of one pixel [m]
103.
104.     */
105.
106.
107.
108.     #include "slam_gmapping.h"
109.
110.     #include <iostream>
111.
112.     #include <time.h>
113.
114.     #include "ros/ros.h"
115.     #include "ros/console.h"
116.     #include "nav_msgs/MapMetaData.h"
117.
118.     #include "gmapping/sensor/sensor_range/rangesensor.h"
119.     #include "gmapping/sensor/sensor_odometry/odometrysensor.h"
120.
121.     #include <rosbag/bag.h>
122.     #include <rosbag/view.h>
123.     #include <boost/foreach.hpp>
124.     #define foreach BOOST_FOREACH
125.
```

```
126.     // compute linear index for given map coords
127.     #define MAP_IDX(sx, i, j) ((sx) * (j) + (i))
128.
129.     SlamGMapping::SlamGMapping():
130.       map_to_odom_(tf::Transform(tf::createQuaternionFromRPY( 0,
   0, 0 ), tf::Point(0, 0, 0 ))),
131.       laser_count_(0), private_nh_("~"), scan_filter_sub_(NULL),
   scan_filter_(NULL), transform_thread_(NULL)
132.     {
133.       seed_ = time(NULL);
134.       init();
135.     }
136.
137.     SlamGMapping::SlamGMapping(ros::NodeHandle& nh,
   ros::NodeHandle& pnh):
138.       map_to_odom_(tf::Transform(tf::createQuaternionFromRPY( 0,
   0, 0 ), tf::Point(0, 0, 0 ))),
139.       laser_count_(0),node_(nh), private_nh_(pnh),
   scan_filter_sub_(NULL), scan_filter_(NULL), transform_thread_(NULL)
140.     {
141.       seed_ = time(NULL);
142.       init();
143.     }
144.
145.     SlamGMapping::SlamGMapping(long unsigned int seed, long unsig
   ned int max_duration_buffer):
146.       map_to_odom_(tf::Transform(tf::createQuaternionFromRPY( 0,
   0, 0 ), tf::Point(0, 0, 0 ))),
147.       laser_count_(0), private_nh_("~"), scan_filter_sub_(NULL),
   scan_filter_(NULL), transform_thread_(NULL),
148.       seed_(seed), tf_(ros::Duration(max_duration_buffer))
149.     {
150.       init();
151.     }
152.
153.
154.     void SlamGMapping::init()
155.     {
156.       // log4cxx::Logger::getLogger(ROSCONSOLE_DEFAULT_NAME)-
   >setLevel(ros::console::g_level_lookup[ros::console::levels::Debug]
   );
157.
158.       // The library is pretty chatty
159.       //gsp_ = new GMapping::GridSlamProcessor(std::cerr);
160.       gsp_ = new GMapping::GridSlamProcessor();
161.       ROS_ASSERT(gsp_);
162.
163.       tfB_ = new tf::TransformBroadcaster();
164.       ROS_ASSERT(tfB_);
165.
166.       gsp_laser_ = NULL;
167.       gsp_odom_ = NULL;
168.
169.       got_first_scan_ = false;
170.       got_map_ = false;
171.
172.
```

```
173.
174.       // Parameters used by our GMapping wrapper
175.       if(!private_nh_.getParam("throttle_scans",
   throttle_scans_))
176.         throttle_scans_ = 1;
177.       if(!private_nh_.getParam("base_frame", base_frame_))
178.         base_frame_ = "base_link";
179.       if(!private_nh_.getParam("map_frame", map_frame_))
180.         map_frame_ = "map";
181.       if(!private_nh_.getParam("odom_frame", odom_frame_))
182.         odom_frame_ = "odom";
183.
184.       private_nh_.param("transform_publish_period",
   transform_publish_period_, 0.05);
185.
186.       double tmp;
187.       if(!private_nh_.getParam("map_update_interval", tmp))
188.         tmp = 5.0;
189.       map_update_interval_.fromSec(tmp);
190.
191.       // Parameters used by GMapping itself
192.       maxUrange_ = 0.0;  maxRange_ = 0.0; // preliminary default,
   will be set in initMapper()
193.       if(!private_nh_.getParam("minimumScore", minimum_score_))
194.         minimum_score_ = 0;
195.       if(!private_nh_.getParam("sigma", sigma_))
196.         sigma_ = 0.05;
197.       if(!private_nh_.getParam("kernelSize", kernelSize_))
198.         kernelSize_ = 1;
199.       if(!private_nh_.getParam("lstep", lstep_))
200.         lstep_ = 0.05;
201.       if(!private_nh_.getParam("astep", astep_))
202.         astep_ = 0.05;
203.       if(!private_nh_.getParam("iterations", iterations_))
204.         iterations_ = 5;
205.       if(!private_nh_.getParam("lsigma", lsigma_))
206.         lsigma_ = 0.075;
207.       if(!private_nh_.getParam("ogain", ogain_))
208.         ogain_ = 3.0;
209.       if(!private_nh_.getParam("lskip", lskip_))
210.         lskip_ = 0;
211.       if(!private_nh_.getParam("srr", srr_))
212.         srr_ = 0.1;
213.       if(!private_nh_.getParam("srt", srt_))
214.         srt_ = 0.2;
215.       if(!private_nh_.getParam("str", str_))
216.         str_ = 0.1;
217.       if(!private_nh_.getParam("stt", stt_))
218.         stt_ = 0.2;
219.       if(!private_nh_.getParam("linearUpdate", linearUpdate_))
220.         linearUpdate_ = 1.0;
221.       if(!private_nh_.getParam("angularUpdate", angularUpdate_))
222.         angularUpdate_ = 0.5;
223.       if(!private_nh_.getParam("temporalUpdate",
   temporalUpdate_))
224.         temporalUpdate_ = -1.0;
```

```
225.       if(!private_nh_.getParam("resampleThreshold",
   resampleThreshold_))
226.         resampleThreshold_ = 0.5;
227.       if(!private_nh_.getParam("particles", particles_))
228.         particles_ = 30;
229.       if(!private_nh_.getParam("xmin", xmin_))
230.         xmin_ = -100.0;
231.       if(!private_nh_.getParam("ymin", ymin_))
232.         ymin_ = -100.0;
233.       if(!private_nh_.getParam("xmax", xmax_))
234.         xmax_ = 100.0;
235.       if(!private_nh_.getParam("ymax", ymax_))
236.         ymax_ = 100.0;
237.       if(!private_nh_.getParam("delta", delta_))
238.         delta_ = 0.05;
239.       if(!private_nh_.getParam("occ_thresh", occ_thresh_))
240.         occ_thresh_ = 0.25;
241.       if(!private_nh_.getParam("llsamplerange", llsamplerange_))
242.         llsamplerange_ = 0.01;
243.       if(!private_nh_.getParam("llsamplestep", llsamplestep_))
244.         llsamplestep_ = 0.01;
245.       if(!private_nh_.getParam("lasamplerange", lasamplerange_))
246.         lasamplerange_ = 0.005;
247.       if(!private_nh_.getParam("lasamplestep", lasamplestep_))
248.         lasamplestep_ = 0.005;
249.
250.       if(!private_nh_.getParam("tf_delay", tf_delay_))
251.         tf_delay_ = transform_publish_period_;
252.
253.     }
254.
255.
256.     void SlamGMapping::startLiveSlam()
257.     {
258.       entropy_publisher_ = private_nh_.advertise<std_msgs::Float6
   4>("entropy", 1, true);
259.       sst_ = node_.advertise<nav_msgs::OccupancyGrid>("map", 1, t
   rue);
260.       sstm_ = node_.advertise<nav_msgs::MapMetaData>("map_metadat
   a", 1, true);
261.       ss_ = node_.advertiseService("dynamic_map", &SlamGMapping::
   mapCallback, this);
262.       scan_filter_sub_ = new message_filters::Subscriber<sensor_m
   sgs::LaserScan>(node_, "scan", 5);
263.       scan_filter_ = new tf::MessageFilter<sensor_msgs::LaserScan
   >(*scan_filter_sub_, tf_, odom_frame_, 5);
264.       scan_filter_-
   >registerCallback(boost::bind(&SlamGMapping::laserCallback, this,
   _1));
265.
266.       transform_thread_ = new boost::thread(boost::bind(&SlamGMap
   ping::publishLoop, this, transform_publish_period_));
267.     }
268.
269.     void SlamGMapping::startReplay(const std::string & bag_fname,
   std::string scan_topic)
270.     {
```

```
271.       double transform_publish_period;
272.       ros::NodeHandle private_nh_("~");
273.       entropy_publisher_ = private_nh_.advertise<std_msgs::Float6
   4>("entropy", 1, true);
274.       sst_ = node_.advertise<nav_msgs::OccupancyGrid>("map", 1, t
   rue);
275.       sstm_ = node_.advertise<nav_msgs::MapMetaData>("map_metadat
   a", 1, true);
276.       ss_ = node_.advertiseService("dynamic_map", &SlamGMapping::
   mapCallback, this);
277.
278.       rosbag::Bag bag;
279.       bag.open(bag_fname, rosbag::bagmode::Read);
280.
281.       std::vector<std::string> topics;
282.       topics.push_back(std::string("/tf"));
283.       topics.push_back(scan_topic);
284.       rosbag::View viewall(bag, rosbag::TopicQuery(topics));
285.
286.       // Store up to 5 messages and there error message (if they
   cannot be processed right away)
287.       std::queue<std::pair<sensor_msgs::LaserScan::ConstPtr,
   std::string> > s_queue;
288.       foreach(rosbag::MessageInstance const m, viewall)
289.       {
290.         tf::tfMessage::ConstPtr cur_tf = m.instantiate<tf::tfMess
   age>();
291.         if (cur_tf != NULL) {
292.           for (size_t i = 0; i < cur_tf->transforms.size(); ++i)
293.           {
294.             geometry_msgs::TransformStamped transformStamped;
295.             tf::StampedTransform stampedTf;
296.             transformStamped = cur_tf->transforms[i];
297.             tf::transformStampedMsgToTF(transformStamped,
   stampedTf);
298.             tf_.setTransform(stampedTf);
299.           }
300.         }
301.
302.         sensor_msgs::LaserScan::ConstPtr s = m.instantiate<sensor
   _msgs::LaserScan>();
303.         if (s != NULL) {
304.           if (!(ros::Time(s->header.stamp)).is_zero())
305.           {
306.             s_queue.push(std::make_pair(s, ""));
307.           }
308.           // Just like in live processing, only process the
   latest 5 scans
309.           if (s_queue.size() > 5) {
310.             ROS_WARN_STREAM("Dropping old scan:
   " << s_queue.front().second);
311.             s_queue.pop();
312.           }
313.           // ignoring un-timestamped tf data
314.         }
315.
316.         // Only process a scan if it has tf data
```

```
317.         while (!s_queue.empty())
318.         {
319.           try
320.           {
321.             tf::StampedTransform t;
322.             tf_.lookupTransform(s_queue.front().first-
   >header.frame_id, odom_frame_, s_queue.front().first->header.stamp,
   t);
323.             this->laserCallback(s_queue.front().first);
324.             s_queue.pop();
325.           }
326.           // If tf does not have the data yet
327.           catch(tf2::TransformException& e)
328.           {
329.             // Store the error to display it if we cannot process
   the data after some time
330.             s_queue.front().second = std::string(e.what());
331.             break;
332.           }
333.         }
334.       }
335.
336.       bag.close();
337.     }
338.
339.     void SlamGMapping::publishLoop(double transform_publish_perio
   d){
340.       if(transform_publish_period == 0)
341.         return;
342.
343.       ros::Rate r(1.0 / transform_publish_period);
344.       while(ros::ok()){
345.         publishTransform();
346.         r.sleep();
347.       }
348.     }
349.
350.     SlamGMapping::~SlamGMapping()
351.     {
352.       if(transform_thread_){
353.         transform_thread_->join();
354.         delete transform_thread_;
355.       }
356.
357.       delete gsp_;
358.       if(gsp_laser_)
359.         delete gsp_laser_;
360.       if(gsp_odom_)
361.         delete gsp_odom_;
362.       if (scan_filter_)
363.         delete scan_filter_;
364.       if (scan_filter_sub_)
365.         delete scan_filter_sub_;
366.     }
367.
368.     bool
```

```
369.     SlamGMapping::getOdomPose(GMapping::OrientedPoint& gmap_pose,
    const ros::Time& t)
370.     {
371.       // Get the pose of the centered laser at the right time
372.       centered_laser_pose_.stamp_ = t;
373.       // Get the laser's pose that is centered
374.       tf::Stamped<tf::Transform> odom_pose;
375.       try
376.       {
377.         tf_.transformPose(odom_frame_, centered_laser_pose_,
   odom_pose);
378.       }
379.       catch(tf::TransformException e)
380.       {
381.         ROS_WARN("Failed to compute odom pose, skipping scan
   (%s)", e.what());
382.         return false;
383.       }
384.       double yaw = tf::getYaw(odom_pose.getRotation());
385.
386.       gmap_pose = GMapping::OrientedPoint(odom_pose.getOrigin().x
   (),
387.                                           odom_pose.getOrigin().y
   (),
388.                                           yaw);
389.       return true;
390.     }
391.
392.     bool
393.     SlamGMapping::initMapper(const sensor_msgs::LaserScan& scan)
394.     {
395.       laser_frame_ = scan.header.frame_id;
396.       // Get the laser's pose, relative to base.
397.       tf::Stamped<tf::Pose> ident;
398.       tf::Stamped<tf::Transform> laser_pose;
399.       ident.setIdentity();
400.       ident.frame_id_ = laser_frame_;
401.       ident.stamp_ = scan.header.stamp;
402.       try
403.       {
404.         tf_.transformPose(base_frame_, ident, laser_pose);
405.       }
406.       catch(tf::TransformException e)
407.       {
408.         ROS_WARN("Failed to compute laser pose, aborting
   initialization (%s)",
409.                  e.what());
410.         return false;
411.       }
412.
413.       // create a point 1m above the laser position and transform
   it into the laser-frame
414.       tf::Vector3 v;
415.       v.setValue(0, 0, 1 + laser_pose.getOrigin().z());
416.       tf::Stamped<tf::Vector3> up(v, scan.header.stamp,
417.                                           base_frame_);
418.       try
```

```
419.       {
420.         tf_.transformPoint(laser_frame_, up, up);
421.         ROS_DEBUG("Z-Axis in sensor frame: %.3f", up.z());
422.       }
423.       catch(tf::TransformException& e)
424.       {
425.         ROS_WARN("Unable to determine orientation of laser: %s",
426.                  e.what());
427.         return false;
428.       }
429.
430.       // gmapping doesnt take roll or pitch into account. So
   check for correct sensor alignment.
431.       if (fabs(fabs(up.z()) - 1) > 0.001)
432.       {
433.         ROS_WARN("Laser has to be mounted planar! Z-coordinate
   has to be 1 or -1, but gave: %.5f",
434.                          up.z());
435.         return false;
436.       }
437.
438.       gsp_laser_beam_count_ = scan.ranges.size();
439.
440.       double angle_center = (scan.angle_min + scan.angle_max)/2;
441.
442.       if (up.z() > 0)
443.       {
444.         do_reverse_range_ = scan.angle_min > scan.angle_max;
445.         centered_laser_pose_ = tf::Stamped<tf::Pose>(tf::Transfor
   m(tf::createQuaternionFromRPY(0,0,angle_center),
446.
    tf::Vector3(0,0,0)), ros::Time::now(), laser_frame_);
447.         ROS_INFO("Laser is mounted upwards.");
448.       }
449.       else
450.       {
451.         do_reverse_range_ = scan.angle_min < scan.angle_max;
452.         centered_laser_pose_ = tf::Stamped<tf::Pose>(tf::Transfor
   m(tf::createQuaternionFromRPY(M_PI,0,-angle_center),
453.
    tf::Vector3(0,0,0)), ros::Time::now(), laser_frame_);
454.         ROS_INFO("Laser is mounted upside down.");
455.       }
456.
457.       // Compute the angles of the laser from -x to x, basically
   symmetric and in increasing order
458.       laser_angles_.resize(scan.ranges.size());
459.       // Make sure angles are started so that they are centered
460.       double theta = - std::fabs(scan.angle_min - scan.angle_max)
   /2;
461.       for(unsigned int i=0; i<scan.ranges.size(); ++i)
462.       {
463.         laser_angles_[i]=theta;
464.         theta += std::fabs(scan.angle_increment);
465.       }
466.
```

```
467.       ROS_DEBUG("Laser angles in laser-frame: min: %.3f max: %.3f
   inc: %.3f", scan.angle_min, scan.angle_max,
468.                 scan.angle_increment);
469.       ROS_DEBUG("Laser angles in top-down centered laser-frame:
   min: %.3f max: %.3f inc: %.3f", laser_angles_.front(),
470.                 laser_angles_.back(),
   std::fabs(scan.angle_increment));
471.
472.       GMapping::OrientedPoint gmap_pose(0, 0, 0);
473.
474.       // setting maxRange and maxUrange here so we can set a
   reasonable default
475.       ros::NodeHandle private_nh_("~");
476.       if(!private_nh_.getParam("maxRange", maxRange_))
477.         maxRange_ = scan.range_max - 0.01;
478.       if(!private_nh_.getParam("maxUrange", maxUrange_))
479.         maxUrange_ = maxRange_;
480.
481.       // The laser must be called "FLASER".
482.       // We pass in the absolute value of the computed angle
   increment, on the
483.       // assumption that GMapping requires a positive angle
   increment.  If the
484.       // actual increment is negative, we'll swap the order of
   ranges before
485.       // feeding each scan to GMapping.
486.       gsp_laser_ = new GMapping::RangeSensor("FLASER",
487.                                              gsp_laser_beam_count
   _,
488.                                              fabs(scan.angle_incr
   ement),
489.                                              gmap_pose,
490.                                              0.0,
491.                                              maxRange_);
492.       ROS_ASSERT(gsp_laser_);
493.
494.       GMapping::SensorMap smap;
495.       smap.insert(make_pair(gsp_laser_->getName(), gsp_laser_));
496.       gsp_->setSensorMap(smap);
497.
498.       gsp_odom_ = new GMapping::OdometrySensor(odom_frame_);
499.       ROS_ASSERT(gsp_odom_);
500.
501.
502.       /// @todo Expose setting an initial pose
503.       GMapping::OrientedPoint initialPose;
504.       if(!getOdomPose(initialPose, scan.header.stamp))
505.       {
506.         ROS_WARN("Unable to determine inital pose of laser!
   Starting point will be set to zero.");
507.         initialPose = GMapping::OrientedPoint(0.0, 0.0, 0.0);
508.       }
509.
510.       gsp_->setMatchingParameters(maxUrange_, maxRange_, sigma_,
511.                                   kernelSize_, lstep_, astep_,
   iterations_,
512.                                   lsigma_, ogain_, lskip_);
```

```
513.
514.       gsp_->setMotionModelParameters(srr_, srt_, str_, stt_);
515.       gsp_->setUpdateDistances(linearUpdate_, angularUpdate_,
   resampleThreshold_);
516.       gsp_->setUpdatePeriod(temporalUpdate_);
517.       gsp_->setgenerateMap(false);
518.       gsp_->GridSlamProcessor::init(particles_, xmin_, ymin_,
   xmax_, ymax_,
519.                                     delta_, initialPose);
520.       gsp_->setllsamplerange(llsamplerange_);
521.       gsp_->setllsamplestep(llsamplestep_);
522.       /// @todo Check these calls; in the gmapping gui, they use
523.       /// llsamplestep and llsamplerange intead of lasamplestep
   and
524.       /// lasamplerange.  It was probably a typo, but who knows.
525.       gsp_->setlasamplerange(lasamplerange_);
526.       gsp_->setlasamplestep(lasamplestep_);
527.       gsp_->setminimumScore(minimum_score_);
528.
529.       // Call the sampling function once to set the seed.
530.       GMapping::sampleGaussian(1,seed_);
531.
532.       ROS_INFO("Initialization complete");
533.
534.       return true;
535.     }
536.
537.     bool
538.     SlamGMapping::addScan(const sensor_msgs::LaserScan& scan,
   GMapping::OrientedPoint& gmap_pose)
539.     {
540.       if(!getOdomPose(gmap_pose, scan.header.stamp))
541.          return false;
542.
543.       if(scan.ranges.size() != gsp_laser_beam_count_)
544.         return false;
545.
546.       // GMapping wants an array of doubles...
547.       double* ranges_double = new double[scan.ranges.size()];
548.       // If the angle increment is negative, we have to invert
   the order of the readings.
549.       if (do_reverse_range_)
550.       {
551.         ROS_DEBUG("Inverting scan");
552.         int num_ranges = scan.ranges.size();
553.         for(int i=0; i < num_ranges; i++)
554.         {
555.           // Must filter out short readings, because the mapper
   won't
556.           if(scan.ranges[num_ranges - i - 1] < scan.range_min)
557.             ranges_double[i] = (double)scan.range_max;
558.           else
559.             ranges_double[i] = (double)scan.ranges[num_ranges - i
    - 1];
560.         }
561.       } else
562.       {
```

```
563.         for(unsigned int i=0; i < scan.ranges.size(); i++)
564.         {
565.           // Must filter out short readings, because the mapper
   won't
566.           if(scan.ranges[i] < scan.range_min)
567.             ranges_double[i] = (double)scan.range_max;
568.           else
569.             ranges_double[i] = (double)scan.ranges[i];
570.         }
571.       }
572.
573.       GMapping::RangeReading reading(scan.ranges.size(),
574.                                      ranges_double,
575.                                      gsp_laser_,
576.                                      scan.header.stamp.toSec());
577.
578.       // ...but it deep copies them in RangeReading constructor,
   so we don't
579.       // need to keep our array around.
580.       delete[] ranges_double;
581.
582.       reading.setPose(gmap_pose);
583.
584.       /*
585.       ROS_DEBUG("scanpose (%.3f): %.3f %.3f %.3f\n",
586.                 scan.header.stamp.toSec(),
587.                 gmap_pose.x,
588.                 gmap_pose.y,
589.                 gmap_pose.theta);
590.                 */
591.       ROS_DEBUG("processing scan");
592.
593.       return gsp_->processScan(reading);
594.     }
595.
596.     void
597.     SlamGMapping::laserCallback(const sensor_msgs::LaserScan::Con
   stPtr& scan)
598.     {
599.       laser_count_++;
600.       if ((laser_count_ % throttle_scans_) != 0)
601.         return;
602.
603.       static ros::Time last_map_update(0,0);
604.
605.       // We can't initialize the mapper until we've got the first
   scan
606.       if(!got_first_scan_)
607.       {
608.         if(!initMapper(*scan))
609.           return;
610.         got_first_scan_ = true;
611.       }
612.
613.       GMapping::OrientedPoint odom_pose;
614.
615.       if(addScan(*scan, odom_pose))
```

```
616.       {
617.         ROS_DEBUG("scan processed");
618.
619.         GMapping::OrientedPoint mpose = gsp_-
   >getParticles()[gsp_->getBestParticleIndex()].pose;
620.         ROS_DEBUG("new best pose: %.3f %.3f %.3f", mpose.x,
   mpose.y, mpose.theta);
621.         ROS_DEBUG("odom pose: %.3f %.3f %.3f", odom_pose.x,
   odom_pose.y, odom_pose.theta);
622.         ROS_DEBUG("correction: %.3f %.3f %.3f",
   mpose.x - odom_pose.x, mpose.y - odom_pose.y,
   mpose.theta - odom_pose.theta);
623.
624.         tf::Transform laser_to_map = tf::Transform(tf::createQuat
   ernionFromRPY(0, 0, mpose.theta), tf::Vector3(mpose.x,
   mpose.y, 0.0)).inverse();
625.         tf::Transform odom_to_laser = tf::Transform(tf::createQua
   ternionFromRPY(0, 0, odom_pose.theta), tf::Vector3(odom_pose.x,
   odom_pose.y, 0.0));
626.
627.         map_to_odom_mutex_.lock();
628.         map_to_odom_ = (odom_to_laser * laser_to_map).inverse();
629.         map_to_odom_mutex_.unlock();
630.
631.         if(!got_map_ || (scan-
   >header.stamp - last_map_update) > map_update_interval_)
632.         {
633.           updateMap(*scan);
634.           last_map_update = scan->header.stamp;
635.           ROS_DEBUG("Updated the map");
636.         }
637.       } else
638.         ROS_DEBUG("cannot process scan");
639.     }
640.
641.     double
642.     SlamGMapping::computePoseEntropy()
643.     {
644.       double weight_total=0.0;
645.       for(std::vector<GMapping::GridSlamProcessor::Particle>::con
   st_iterator it = gsp_->getParticles().begin();
646.           it != gsp_->getParticles().end();
647.           ++it)
648.       {
649.         weight_total += it->weight;
650.       }
651.       double entropy = 0.0;
652.       for(std::vector<GMapping::GridSlamProcessor::Particle>::con
   st_iterator it = gsp_->getParticles().begin();
653.           it != gsp_->getParticles().end();
654.           ++it)
655.       {
656.         if(it->weight/weight_total > 0.0)
657.           entropy += it->weight/weight_total * log(it-
   >weight/weight_total);
658.       }
659.       return -entropy;
```

```
660.     }
661.
662.     void
663.     SlamGMapping::updateMap(const sensor_msgs::LaserScan& scan)
664.     {
665.       ROS_DEBUG("Update map");
666.       boost::mutex::scoped_lock map_lock (map_mutex_);
667.       GMapping::ScanMatcher matcher;
668.
669.       matcher.setLaserParameters(scan.ranges.size(), &(laser_angl
   es_[0]),
670.                                  gsp_laser_->getPose());
671.
672.       matcher.setlaserMaxRange(maxRange_);
673.       matcher.setusableRange(maxUrange_);
674.       matcher.setgenerateMap(true);
675.
676.       GMapping::GridSlamProcessor::Particle best =
677.              gsp_->getParticles()[gsp_->getBestParticleIndex()];
678.       std_msgs::Float64 entropy;
679.       entropy.data = computePoseEntropy();
680.       if(entropy.data > 0.0)
681.         entropy_publisher_.publish(entropy);
682.
683.       if(!got_map_) {
684.         map_.map.info.resolution = delta_;
685.         map_.map.info.origin.position.x = 0.0;
686.         map_.map.info.origin.position.y = 0.0;
687.         map_.map.info.origin.position.z = 0.0;
688.         map_.map.info.origin.orientation.x = 0.0;
689.         map_.map.info.origin.orientation.y = 0.0;
690.         map_.map.info.origin.orientation.z = 0.0;
691.         map_.map.info.origin.orientation.w = 1.0;
692.       }
693.
694.       GMapping::Point center;
695.       center.x=(xmin_ + xmax_) / 2.0;
696.       center.y=(ymin_ + ymax_) / 2.0;
697.
698.       GMapping::ScanMatcherMap smap(center, xmin_, ymin_, xmax_,
   ymax_,
699.                                     delta_);
700.
701.       ROS_DEBUG("Trajectory tree:");
702.       for(GMapping::GridSlamProcessor::TNode* n = best.node;
703.           n;
704.           n = n->parent)
705.       {
706.         ROS_DEBUG("  %.3f %.3f %.3f",
707.                   n->pose.x,
708.                   n->pose.y,
709.                   n->pose.theta);
710.         if(!n->reading)
711.         {
712.           ROS_DEBUG("Reading is NULL");
713.           continue;
714.         }
```

```
715.          matcher.invalidateActiveArea();
716.          matcher.computeActiveArea(smap, n->pose, &((*n-
   >reading)[0]));
717.          matcher.registerScan(smap, n->pose, &((*n->reading)[0]));
718.        }
719.
720.        // the map may have expanded, so resize ros message as well
721.        if(map_.map.info.width != (unsigned int) smap.getMapSizeX()
    || map_.map.info.height != (unsigned int) smap.getMapSizeY()) {
722.
723.          // NOTE: The results of ScanMatcherMap::getSize() are
   different from the parameters given to the constructor
724.          //       so we must obtain the bounding box in a
   different way
725.          GMapping::Point wmin = smap.map2world(GMapping::IntPoint(
   0, 0));
726.          GMapping::Point wmax = smap.map2world(GMapping::IntPoint(
   smap.getMapSizeX(), smap.getMapSizeY()));
727.          xmin_ = wmin.x; ymin_ = wmin.y;
728.          xmax_ = wmax.x; ymax_ = wmax.y;
729.
730.          ROS_DEBUG("map size is now %dx%d pixels (%f,%f)-(%f,
   %f)", smap.getMapSizeX(), smap.getMapSizeY(),
731.                    xmin_, ymin_, xmax_, ymax_);
732.
733.          map_.map.info.width = smap.getMapSizeX();
734.          map_.map.info.height = smap.getMapSizeY();
735.          map_.map.info.origin.position.x = xmin_;
736.          map_.map.info.origin.position.y = ymin_;
737.          map_.map.data.resize(map_.map.info.width * map_.map.info.
   height);
738.
739.          ROS_DEBUG("map origin: (%f, %f)",
   map_.map.info.origin.position.x, map_.map.info.origin.position.y);
740.        }
741.
742.        for(int x=0; x < smap.getMapSizeX(); x++)
743.        {
744.          for(int y=0; y < smap.getMapSizeY(); y++)
745.          {
746.            /// @todo Sort out the unknown vs. free vs. obstacle
   thresholding
747.            GMapping::IntPoint p(x, y);
748.            double occ=smap.cell(p);
749.            assert(occ <= 1.0);
750.            if(occ < 0)
751.              map_.map.data[MAP_IDX(map_.map.info.width, x, y)] = -
   1;
752.            else if(occ > occ_thresh_)
753.            {
754.              //map_.map.data[MAP_IDX(map_.map.info.width, x, y)] =
   (int)round(occ*100.0);
755.              map_.map.data[MAP_IDX(map_.map.info.width, x,
   y)] = 100;
756.            }
757.            else
```

```
758.             map_.map.data[MAP_IDX(map_.map.info.width, x,
   y)] = 0;
759.         }
760.       }
761.       got_map_ = true;
762.
763.       //make sure to set the header information on the map
764.       map_.map.header.stamp = ros::Time::now();
765.       map_.map.header.frame_id = tf_.resolve( map_frame_ );
766.
767.       sst_.publish(map_.map);
768.       sstm_.publish(map_.map.info);
769.     }
770.
771.     bool
772.     SlamGMapping::mapCallback(nav_msgs::GetMap::Request  &req,
773.                               nav_msgs::GetMap::Response &res)
774.     {
775.       boost::mutex::scoped_lock map_lock (map_mutex_);
776.       if(got_map_ && map_.map.info.width && map_.map.info.height)
777.       {
778.         res = map_;
779.         return true;
780.       }
781.       else
782.         return false;
783.     }
784.
785.     void SlamGMapping::publishTransform()
786.     {
787.       map_to_odom_mutex_.lock();
788.       ros::Time tf_expiration = ros::Time::now() + ros::Duration(
   tf_delay_);
789.       tfB_->sendTransform( tf::StampedTransform (map_to_odom_,
   tf_expiration, map_frame_, odom_frame_));
790.       map_to_odom_mutex_.unlock();
791.     }
```

B. Código Move_base

```
1. /*********************************************************************
   ***
2. *
3. * Software License Agreement (BSD License)
4. *
5. *  Copyright (c) 2008, Willow Garage, Inc.
6. *  All rights reserved.
7. *
8. *  Redistribution and use in source and binary forms, with or
   without
9. *  modification, are permitted provided that the following
   conditions
10.      *  are met:
11.      *
12.      *   * Redistributions of source code must retain the above
   copyright
13.      *     notice, this list of conditions and the following
   disclaimer.
14.      *   * Redistributions in binary form must reproduce the above
15.      *     copyright notice, this list of conditions and the
   following
16.      *     disclaimer in the documentation and/or other materials
   provided
17.      *     with the distribution.
18.      *   * Neither the name of the Willow Garage nor the names of
   its
19.      *     contributors may be used to endorse or promote products
   derived
20.      *     from this software without specific prior written
   permission.
21.      *
22.      *  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
   CONTRIBUTORS
23.      *  "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
   BUT NOT
24.      *  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
   FITNESS
25.      *  FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
   THE
26.      *  COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
   INDIRECT,
27.      *  INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
   (INCLUDING,
28.      *  BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
   SERVICES;
29.      *  LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
   HOWEVER
30.      *  CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
   CONTRACT, STRICT
31.      *  LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
   ARISING IN
32.      *  ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
   OF THE
```

```
33.      *  POSSIBILITY OF SUCH DAMAGE.
34.      *
35.      * Author: Eitan Marder-Eppstein
36.      *         Mike Phillips (put the planner in its own thread)
37.      *********************************************************
   ********/
38.      #include <move_base/move_base.h>
39.      #include <cmath>
40.
41.      #include <boost/algorithm/string.hpp>
42.      #include <boost/thread.hpp>
43.
44.      #include <geometry_msgs/Twist.h>
45.
46.      #include <tf2_geometry_msgs/tf2_geometry_msgs.h>
47.
48.      namespace move_base {
49.
50.        MoveBase::MoveBase(tf2_ros::Buffer& tf) :
51.          tf_(tf),
52.          as_(NULL),
53.          planner_costmap_ros_(NULL),
   controller_costmap_ros_(NULL),
54.          bgp_loader_("nav_core", "nav_core::BaseGlobalPlanner"),
55.          blp_loader_("nav_core", "nav_core::BaseLocalPlanner"),
56.          recovery_loader_("nav_core", "nav_core::RecoveryBehavior"
   ),
57.          planner_plan_(NULL), latest_plan_(NULL),
   controller_plan_(NULL),
58.          runPlanner_(false), setup_(false), p_freq_change_(false),
   c_freq_change_(false), new_global_plan_(false) {
59.
60.          as_ = new MoveBaseActionServer(ros::NodeHandle(), "move_b
   ase", boost::bind(&MoveBase::executeCb, this, _1), false);
61.
62.          ros::NodeHandle private_nh("~");
63.          ros::NodeHandle nh;
64.
65.          recovery_trigger_ = PLANNING_R;
66.
67.          //get some parameters that will be global to the move
   base node
68.          std::string global_planner, local_planner;
69.          private_nh.param("base_global_planner", global_planner,
   std::string("navfn/NavfnROS"));
70.          private_nh.param("base_local_planner", local_planner,
   std::string("base_local_planner/TrajectoryPlannerROS"));
71.          private_nh.param("global_costmap/robot_base_frame",
   robot_base_frame_, std::string("base_link"));
72.          private_nh.param("global_costmap/global_frame",
   global_frame_, std::string("/map"));
73.          private_nh.param("planner_frequency",
   planner_frequency_, 0.0);
74.          private_nh.param("controller_frequency",
   controller_frequency_, 20.0);
75.          private_nh.param("planner_patience",
   planner_patience_, 5.0);
```

```
76.          private_nh.param("controller_patience",
   controller_patience_, 15.0);
77.          private_nh.param("max_planning_retries",
   max_planning_retries_, -1);  // disabled by default
78.
79.          private_nh.param("oscillation_timeout",
   oscillation_timeout_, 0.0);
80.          private_nh.param("oscillation_distance",
   oscillation_distance_, 0.5);
81.
82.          //set up plan triple buffer
83.          planner_plan_ = new std::vector<geometry_msgs::PoseStampe
   d>();
84.          latest_plan_ = new std::vector<geometry_msgs::PoseStamped
   >();
85.          controller_plan_ = new std::vector<geometry_msgs::PoseSta
   mped>();
86.
87.          //set up the planner's thread
88.          planner_thread_ = new boost::thread(boost::bind(&MoveBase
   ::planThread, this));
89.
90.          //for commanding the base
91.          vel_pub_ = nh.advertise<geometry_msgs::Twist>("cmd_vel",
   1);
92.          current_goal_pub_ = private_nh.advertise<geometry_msgs::P
   oseStamped>("current_goal", 0 );
93.
94.          ros::NodeHandle action_nh("move_base");
95.          action_goal_pub_ = action_nh.advertise<move_base_msgs::Mo
   veBaseActionGoal>("goal", 1);
96.
97.          //we'll provide a mechanism for some people to send goals
   as PoseStamped messages over a topic
98.          //they won't get any useful information back about its
   status, but this is useful for tools
99.          //like nav_view and rviz
100.         ros::NodeHandle simple_nh("move_base_simple");
101.         goal_sub_ = simple_nh.subscribe<geometry_msgs::PoseStampe
   d>("goal", 1, boost::bind(&MoveBase::goalCB, this, _1));
102.
103.         //we'll assume the radius of the robot to be consistent
   with what's specified for the costmaps
104.         private_nh.param("local_costmap/inscribed_radius",
   inscribed_radius_, 0.325);
105.         private_nh.param("local_costmap/circumscribed_radius",
   circumscribed_radius_, 0.46);
106.         private_nh.param("clearing_radius", clearing_radius_,
   circumscribed_radius_);
107.         private_nh.param("conservative_reset_dist",
   conservative_reset_dist_, 3.0);
108.
109.         private_nh.param("shutdown_costmaps",
   shutdown_costmaps_, false);
110.         private_nh.param("clearing_rotation_allowed",
   clearing_rotation_allowed_, true);
```

```
111.         private_nh.param("recovery_behavior_enabled",
   recovery_behavior_enabled_, true);
112.
113.         //create the ros wrapper for the planner's costmap... and
   initializer a pointer we'll use with the underlying map
114.         planner_costmap_ros_ = new costmap_2d::Costmap2DROS("glob
   al_costmap", tf_);
115.         planner_costmap_ros_->pause();
116.
117.         //initialize the global planner
118.         try {
119.           planner_ = bgp_loader_.createInstance(global_planner);
120.           planner_-
   >initialize(bgp_loader_.getName(global_planner),
   planner_costmap_ros_);
121.         } catch (const pluginlib::PluginlibException& ex) {
122.           ROS_FATAL("Failed to create the %s planner, are you
   sure it is properly registered and that the containing library is
   built? Exception: %s", global_planner.c_str(), ex.what());
123.           exit(1);
124.         }
125.
126.         //create the ros wrapper for the controller's costmap...
   and initializer a pointer we'll use with the underlying map
127.         controller_costmap_ros_ = new costmap_2d::Costmap2DROS("l
   ocal_costmap", tf_);
128.         controller_costmap_ros_->pause();
129.
130.         //create a local planner
131.         try {
132.           tc_ = blp_loader_.createInstance(local_planner);
133.           ROS_INFO("Created local_planner %s",
   local_planner.c_str());
134.           tc_-
   >initialize(blp_loader_.getName(local_planner), &tf_,
   controller_costmap_ros_);
135.         } catch (const pluginlib::PluginlibException& ex) {
136.           ROS_FATAL("Failed to create the %s planner, are you
   sure it is properly registered and that the containing library is
   built? Exception: %s", local_planner.c_str(), ex.what());
137.           exit(1);
138.         }
139.
140.         // Start actively updating costmaps based on sensor data
141.         planner_costmap_ros_->start();
142.         controller_costmap_ros_->start();
143.
144.         //advertise a service for getting a plan
145.         make_plan_srv_ = private_nh.advertiseService("make_plan",
    &MoveBase::planService, this);
146.
147.         //advertise a service for clearing the costmaps
148.         clear_costmaps_srv_ = private_nh.advertiseService("clear_
   costmaps", &MoveBase::clearCostmapsService, this);
149.
150.         //if we shutdown our costmaps when we're deactivated...
   we'll do that now
```

```
151.         if(shutdown_costmaps_){
152.           ROS_DEBUG_NAMED("move_base","Stopping costmaps
   initially");
153.           planner_costmap_ros_->stop();
154.           controller_costmap_ros_->stop();
155.         }
156.
157.         //load any user specified recovery behaviors, and if that
   fails load the defaults
158.         if(!loadRecoveryBehaviors(private_nh)){
159.           loadDefaultRecoveryBehaviors();
160.         }
161.
162.         //initially, we'll need to make a plan
163.         state_ = PLANNING;
164.
165.         //we'll start executing recovery behaviors at the
   beginning of our list
166.         recovery_index_ = 0;
167.
168.         //we're all set up now so we can start the action server
169.         as_->start();
170.
171.         dsrv_ = new dynamic_reconfigure::Server<move_base::MoveBa
   seConfig>(ros::NodeHandle("~"));
172.         dynamic_reconfigure::Server<move_base::MoveBaseConfig>::C
   allbackType cb = boost::bind(&MoveBase::reconfigureCB, this, _1,
   _2);
173.         dsrv_->setCallback(cb);
174.       }
175.
176.       void MoveBase::reconfigureCB(move_base::MoveBaseConfig &con
   fig, uint32_t level){
177.         boost::recursive_mutex::scoped_lock l(configuration_mutex
   _);
178.
179.         //The first time we're called, we just want to make sure
   we have the
180.         //original configuration
181.         if(!setup_)
182.         {
183.           last_config_ = config;
184.           default_config_ = config;
185.           setup_ = true;
186.           return;
187.         }
188.
189.         if(config.restore_defaults) {
190.           config = default_config_;
191.           //if someone sets restore defaults on the parameter
   server, prevent looping
192.           config.restore_defaults = false;
193.         }
194.
195.         if(planner_frequency_ != config.planner_frequency)
196.         {
197.           planner_frequency_ = config.planner_frequency;
```

```
198.           p_freq_change_ = true;
199.         }
200.
201.         if(controller_frequency_ != config.controller_frequency)
202.         {
203.           controller_frequency_ = config.controller_frequency;
204.           c_freq_change_ = true;
205.         }
206.
207.         planner_patience_ = config.planner_patience;
208.         controller_patience_ = config.controller_patience;
209.         max_planning_retries_ = config.max_planning_retries;
210.         conservative_reset_dist_ = config.conservative_reset_dist
    ;
211.
212.         recovery_behavior_enabled_ = config.recovery_behavior_ena
    bled;
213.         clearing_rotation_allowed_ = config.clearing_rotation_all
    owed;
214.         shutdown_costmaps_ = config.shutdown_costmaps;
215.
216.         oscillation_timeout_ = config.oscillation_timeout;
217.         oscillation_distance_ = config.oscillation_distance;
218.         if(config.base_global_planner != last_config_.base_global
    _planner) {
219.           boost::shared_ptr<nav_core::BaseGlobalPlanner> old_plan
    ner = planner_;
220.           //initialize the global planner
221.           ROS_INFO("Loading global planner %s",
    config.base_global_planner.c_str());
222.           try {
223.             planner_ = bgp_loader_.createInstance(config.base_glo
    bal_planner);
224.
225.             // wait for the current planner to finish planning
226.             boost::unique_lock<boost::recursive_mutex> lock(plann
    er_mutex_);
227.
228.             // Clean up before initializing the new planner
229.             planner_plan_->clear();
230.             latest_plan_->clear();
231.             controller_plan_->clear();
232.             resetState();
233.             planner_-
    >initialize(bgp_loader_.getName(config.base_global_planner),
    planner_costmap_ros_);
234.
235.             lock.unlock();
236.           } catch (const pluginlib::PluginlibException& ex) {
237.             ROS_FATAL("Failed to create the %s planner, are you
    sure it is properly registered and that the \
238.                       containing library is built? Exception:
    %s", config.base_global_planner.c_str(), ex.what());
239.             planner_ = old_planner;
240.             config.base_global_planner = last_config_.base_global
    _planner;
241.           }
```

```
242.         }
243.
244.         if(config.base_local_planner != last_config_.base_local_p
   lanner){
245.           boost::shared_ptr<nav_core::BaseLocalPlanner> old_plann
   er = tc_;
246.           //create a local planner
247.           try {
248.             tc_ = blp_loader_.createInstance(config.base_local_pl
   anner);
249.             // Clean up before initializing the new planner
250.             planner_plan_->clear();
251.             latest_plan_->clear();
252.             controller_plan_->clear();
253.             resetState();
254.             tc_-
   >initialize(blp_loader_.getName(config.base_local_planner), &tf_,
   controller_costmap_ros_);
255.           } catch (const pluginlib::PluginlibException& ex) {
256.             ROS_FATAL("Failed to create the %s planner, are you
   sure it is properly registered and that the \
257.                       containing library is built? Exception:
   %s", config.base_local_planner.c_str(), ex.what());
258.             tc_ = old_planner;
259.             config.base_local_planner = last_config_.base_local_p
   lanner;
260.           }
261.         }
262.
263.         last_config_ = config;
264.       }
265.
266.       void MoveBase::goalCB(const geometry_msgs::PoseStamped::Con
   stPtr& goal){
267.         ROS_DEBUG_NAMED("move_base","In ROS goal callback,
   wrapping the PoseStamped in the action message and re-sending to
   the server.");
268.         move_base_msgs::MoveBaseActionGoal action_goal;
269.         action_goal.header.stamp = ros::Time::now();
270.         action_goal.goal.target_pose = *goal;
271.
272.         action_goal_pub_.publish(action_goal);
273.       }
274.
275.       void MoveBase::clearCostmapWindows(double size_x, double si
   ze_y){
276.         geometry_msgs::PoseStamped global_pose;
277.
278.         //clear the planner's costmap
279.         getRobotPose(global_pose, planner_costmap_ros_);
280.
281.         std::vector<geometry_msgs::Point> clear_poly;
282.         double x = global_pose.pose.position.x;
283.         double y = global_pose.pose.position.y;
284.         geometry_msgs::Point pt;
285.
286.         pt.x = x - size_x / 2;
```

```
287.         pt.y = y - size_y / 2;
288.         clear_poly.push_back(pt);
289.
290.         pt.x = x + size_x / 2;
291.         pt.y = y - size_y / 2;
292.         clear_poly.push_back(pt);
293.
294.         pt.x = x + size_x / 2;
295.         pt.y = y + size_y / 2;
296.         clear_poly.push_back(pt);
297.
298.         pt.x = x - size_x / 2;
299.         pt.y = y + size_y / 2;
300.         clear_poly.push_back(pt);
301.
302.         planner_costmap_ros_->getCostmap()-
   >setConvexPolygonCost(clear_poly, costmap_2d::FREE_SPACE);
303.
304.         //clear the controller's costmap
305.         getRobotPose(global_pose, controller_costmap_ros_);
306.
307.         clear_poly.clear();
308.         x = global_pose.pose.position.x;
309.         y = global_pose.pose.position.y;
310.
311.         pt.x = x - size_x / 2;
312.         pt.y = y - size_y / 2;
313.         clear_poly.push_back(pt);
314.
315.         pt.x = x + size_x / 2;
316.         pt.y = y - size_y / 2;
317.         clear_poly.push_back(pt);
318.
319.         pt.x = x + size_x / 2;
320.         pt.y = y + size_y / 2;
321.         clear_poly.push_back(pt);
322.
323.         pt.x = x - size_x / 2;
324.         pt.y = y + size_y / 2;
325.         clear_poly.push_back(pt);
326.
327.         controller_costmap_ros_->getCostmap()-
   >setConvexPolygonCost(clear_poly, costmap_2d::FREE_SPACE);
328.       }
329.
330.       bool MoveBase::clearCostmapsService(std_srvs::Empty::Reques
   t &req, std_srvs::Empty::Response &resp){
331.         //clear the costmaps
332.         boost::unique_lock<costmap_2d::Costmap2D::mutex_t> lock_c
   ontroller(*(controller_costmap_ros_->getCostmap()->getMutex()));
333.         controller_costmap_ros_->resetLayers();
334.
335.         boost::unique_lock<costmap_2d::Costmap2D::mutex_t> lock_p
   lanner(*(planner_costmap_ros_->getCostmap()->getMutex()));
336.         planner_costmap_ros_->resetLayers();
337.         return true;
338.       }
```

```
339.
340.
341.       bool MoveBase::planService(nav_msgs::GetPlan::Request &req,
   nav_msgs::GetPlan::Response &resp){
342.         if(as_->isActive()){
343.           ROS_ERROR("move_base must be in an inactive state to
   make a plan for an external user");
344.           return false;
345.         }
346.         //make sure we have a costmap for our planner
347.         if(planner_costmap_ros_ == NULL){
348.           ROS_ERROR("move_base cannot make a plan for you because
   it doesn't have a costmap");
349.           return false;
350.         }
351.
352.         geometry_msgs::PoseStamped start;
353.         //if the user does not specify a start pose, identified
   by an empty frame id, then use the robot's pose
354.         if(req.start.header.frame_id.empty())
355.         {
356.             geometry_msgs::PoseStamped global_pose;
357.             if(!getRobotPose(global_pose, planner_costmap_ros_)){
358.               ROS_ERROR("move_base cannot make a plan for you
   because it could not get the start pose of the robot");
359.               return false;
360.             }
361.             start = global_pose;
362.         }
363.         else
364.         {
365.             start = req.start;
366.         }
367.
368.         //update the copy of the costmap the planner uses
369.         clearCostmapWindows(2 * clearing_radius_, 2 * clearing_ra
   dius_);
370.
371.         //first try to make a plan to the exact desired goal
372.         std::vector<geometry_msgs::PoseStamped> global_plan;
373.         if(!planner_->makePlan(start, req.goal,
   global_plan) || global_plan.empty()){
374.           ROS_DEBUG_NAMED("move_base","Failed to find a plan to
   exact goal of (%.2f, %.2f), searching for a feasible goal within
   tolerance",
375.               req.goal.pose.position.x,
   req.goal.pose.position.y);
376.
377.           //search outwards for a feasible goal within the
   specified tolerance
378.           geometry_msgs::PoseStamped p;
379.           p = req.goal;
380.           bool found_legal = false;
381.           float resolution = planner_costmap_ros_->getCostmap()-
   >getResolution();
382.           float search_increment = resolution*3.0;
```

```
383.           if(req.tolerance > 0.0 && req.tolerance < search_increm
   ent) search_increment = req.tolerance;
384.           for(float max_offset = search_increment; max_offset <=
   req.tolerance && !found_legal; max_offset += search_increment) {
385.             for(float y_offset = 0; y_offset <= max_offset && !fo
   und_legal; y_offset += search_increment) {
386.               for(float x_offset = 0; x_offset <= max_offset && !
   found_legal; x_offset += search_increment) {
387.
388.                 //don't search again inside the current outer
   layer
389.                 if(x_offset < max_offset-1e-
   9 && y_offset < max_offset-1e-9) continue;
390.
391.                 //search to both sides of the desired goal
392.                 for(float y_mult = -1.0; y_mult <= 1.0 + 1e-
   9 && !found_legal; y_mult += 2.0) {
393.
394.                   //if one of the offsets is 0, -1*0 is still 0
   (so get rid of one of the two)
395.                   if(y_offset < 1e-9 && y_mult < -1.0 + 1e-
   9) continue;
396.
397.                   for(float x_mult = -1.0; x_mult <= 1.0 + 1e-
   9 && !found_legal; x_mult += 2.0) {
398.                     if(x_offset < 1e-9 && x_mult < -1.0 + 1e-
   9) continue;
399.
400.                     p.pose.position.y = req.goal.pose.position.y
   + y_offset * y_mult;
401.                     p.pose.position.x = req.goal.pose.position.x
   + x_offset * x_mult;
402.
403.                     if(planner_->makePlan(start, p,
   global_plan)){
404.                       if(!global_plan.empty()){
405.
406.                         //adding the (unreachable) original goal
   to the end of the global plan, in case the local planner can get
   you there
407.                         //(the reachable goal should have been
   added by the global planner)
408.                         global_plan.push_back(req.goal);
409.
410.                         found_legal = true;
411.                         ROS_DEBUG_NAMED("move_base", "Found a
   plan to point (%.2f, %.2f)", p.pose.position.x, p.pose.position.y);
412.                         break;
413.                       }
414.                     }
415.                     else{
416.                       ROS_DEBUG_NAMED("move_base","Failed to find
   a plan to point (%.2f, %.2f)", p.pose.position.x,
   p.pose.position.y);
417.                     }
418.                   }
419.                 }
```

```
420.               }
421.             }
422.           }
423.         }
424.
425.         //copy the plan into a message to send out
426.         resp.plan.poses.resize(global_plan.size());
427.         for(unsigned int i = 0; i < global_plan.size(); ++i){
428.           resp.plan.poses[i] = global_plan[i];
429.         }
430.
431.         return true;
432.       }
433.
434.       MoveBase::~MoveBase(){
435.         recovery_behaviors_.clear();
436.
437.         delete dsrv_;
438.
439.         if(as_ != NULL)
440.           delete as_;
441.
442.         if(planner_costmap_ros_ != NULL)
443.           delete planner_costmap_ros_;
444.
445.         if(controller_costmap_ros_ != NULL)
446.           delete controller_costmap_ros_;
447.
448.         planner_thread_->interrupt();
449.         planner_thread_->join();
450.
451.         delete planner_thread_;
452.
453.         delete planner_plan_;
454.         delete latest_plan_;
455.         delete controller_plan_;
456.
457.         planner_.reset();
458.         tc_.reset();
459.       }
460.
461.       bool MoveBase::makePlan(const geometry_msgs::PoseStamped& g
    oal, std::vector<geometry_msgs::PoseStamped>& plan){
462.         boost::unique_lock<costmap_2d::Costmap2D::mutex_t> lock(*
    (planner_costmap_ros_->getCostmap()->getMutex()));
463.
464.         //make sure to set the plan to be empty initially
465.         plan.clear();
466.
467.         //since this gets called on handle activate
468.         if(planner_costmap_ros_ == NULL) {
469.           ROS_ERROR("Planner costmap ROS is NULL, unable to
    create global plan");
470.           return false;
471.         }
472.
473.         //get the starting pose of the robot
```

```
474.         geometry_msgs::PoseStamped global_pose;
475.         if(!getRobotPose(global_pose, planner_costmap_ros_)) {
476.           ROS_WARN("Unable to get starting pose of robot, unable
   to create global plan");
477.           return false;
478.         }
479.
480.         const geometry_msgs::PoseStamped& start = global_pose;
481.
482.         //if the planner fails or returns a zero length plan,
   planning failed
483.         if(!planner_->makePlan(start, goal,
   plan) || plan.empty()){
484.           ROS_DEBUG_NAMED("move_base","Failed to find a  plan to
   point (%.2f, %.2f)", goal.pose.position.x, goal.pose.position.y);
485.           return false;
486.         }
487.
488.         return true;
489.       }
490.
491.       void MoveBase::publishZeroVelocity(){
492.         geometry_msgs::Twist cmd_vel;
493.         cmd_vel.linear.x = 0.0;
494.         cmd_vel.linear.y = 0.0;
495.         cmd_vel.angular.z = 0.0;
496.         vel_pub_.publish(cmd_vel);
497.       }
498.
499.       bool MoveBase::isQuaternionValid(const geometry_msgs::Quate
   rnion& q){
500.         //first we need to check if the quaternion has nan's or
   infs
501.         if(!std::isfinite(q.x) || !std::isfinite(q.y) || !std::is
   finite(q.z) || !std::isfinite(q.w)){
502.           ROS_ERROR("Quaternion has nans or infs... discarding as
   a navigation goal");
503.           return false;
504.         }
505.
506.         tf2::Quaternion tf_q(q.x, q.y, q.z, q.w);
507.
508.         //next, we need to check if the length of the quaternion
   is close to zero
509.         if(tf_q.length2() < 1e-6){
510.           ROS_ERROR("Quaternion has length close to zero...
   discarding as navigation goal");
511.           return false;
512.         }
513.
514.         //next, we'll normalize the quaternion and check that it
   transforms the vertical vector correctly
515.         tf_q.normalize();
516.
517.         tf2::Vector3 up(0, 0, 1);
518.
```

```
519.         double dot = up.dot(up.rotate(tf_q.getAxis(),
   tf_q.getAngle()));
520.
521.         if(fabs(dot - 1) > 1e-3){
522.           ROS_ERROR("Quaternion is invalid... for navigation the
   z-axis of the quaternion must be close to vertical.");
523.           return false;
524.         }
525.
526.         return true;
527.       }
528.
529.       geometry_msgs::PoseStamped MoveBase::goalToGlobalFrame(cons
   t geometry_msgs::PoseStamped& goal_pose_msg){
530.         std::string global_frame = planner_costmap_ros_-
   >getGlobalFrameID();
531.         geometry_msgs::PoseStamped goal_pose, global_pose;
532.         goal_pose = goal_pose_msg;
533.
534.         //just get the latest available transform... for accuracy
   they should send
535.         //goals in the frame of the planner
536.         goal_pose.header.stamp = ros::Time();
537.
538.         try{
539.           tf_.transform(goal_pose_msg, global_pose,
   global_frame);
540.         }
541.         catch(tf2::TransformException& ex){
542.           ROS_WARN("Failed to transform the goal pose from %s
   into the %s frame: %s",
543.               goal_pose.header.frame_id.c_str(),
   global_frame.c_str(), ex.what());
544.           return goal_pose_msg;
545.         }
546.
547.         return global_pose;
548.       }
549.
550.       void MoveBase::wakePlanner(const ros::TimerEvent& event)
551.       {
552.         // we have slept long enough for rate
553.         planner_cond_.notify_one();
554.       }
555.
556.       void MoveBase::planThread(){
557.         ROS_DEBUG_NAMED("move_base_plan_thread","Starting planner
   thread...");
558.         ros::NodeHandle n;
559.         ros::Timer timer;
560.         bool wait_for_wake = false;
561.         boost::unique_lock<boost::recursive_mutex> lock(planner_m
   utex_);
562.         while(n.ok()){
563.           //check if we should run the planner (the mutex is
   locked)
564.           while(wait_for_wake || !runPlanner_){
```

```
565.             //if we should not be running the planner then
   suspend this thread
566.             ROS_DEBUG_NAMED("move_base_plan_thread","Planner
   thread is suspending");
567.             planner_cond_.wait(lock);
568.             wait_for_wake = false;
569.           }
570.           ros::Time start_time = ros::Time::now();
571.
572.           //time to plan! get a copy of the goal and unlock the
   mutex
573.           geometry_msgs::PoseStamped temp_goal = planner_goal_;
574.           lock.unlock();
575.           ROS_DEBUG_NAMED("move_base_plan_thread","Planning...");
576.
577.           //run planner
578.           planner_plan_->clear();
579.           bool gotPlan = n.ok() && makePlan(temp_goal, *planner_p
   lan_);
580.
581.           if(gotPlan){
582.             ROS_DEBUG_NAMED("move_base_plan_thread","Got Plan
   with %zu points!", planner_plan_->size());
583.             //pointer swap the plans under mutex (the controller
   will pull from latest_plan_)
584.             std::vector<geometry_msgs::PoseStamped>* temp_plan =
   planner_plan_;
585.
586.             lock.lock();
587.             planner_plan_ = latest_plan_;
588.             latest_plan_ = temp_plan;
589.             last_valid_plan_ = ros::Time::now();
590.             planning_retries_ = 0;
591.             new_global_plan_ = true;
592.
593.             ROS_DEBUG_NAMED("move_base_plan_thread","Generated a
   plan from the base_global_planner");
594.
595.             //make sure we only start the controller if we still
   haven't reached the goal
596.             if(runPlanner_)
597.               state_ = CONTROLLING;
598.             if(planner_frequency_ <= 0)
599.               runPlanner_ = false;
600.             lock.unlock();
601.           }
602.           //if we didn't get a plan and we are in the planning
   state (the robot isn't moving)
603.           else if(state_==PLANNING){
604.             ROS_DEBUG_NAMED("move_base_plan_thread","No
   Plan...");
605.             ros::Time attempt_end = last_valid_plan_ + ros::Durat
   ion(planner_patience_);
606.
607.             //check if we've tried to make a plan for over our
   time limit or our maximum number of retries
```

```
608.             //issue #496: we stop planning when one of the
   conditions is true, but if max_planning_retries_
609.             //is negative (the default), it is just ignored and
   we have the same behavior as ever
610.             lock.lock();
611.             planning_retries_++;
612.             if(runPlanner_ &&
613.                (ros::Time::now() > attempt_end || planning_retrie
   s_ > uint32_t(max_planning_retries_))){
614.               //we'll move into our obstacle clearing mode
615.               state_ = CLEARING;
616.               runPlanner_ = false;  // proper solution for issue
   #523
617.               publishZeroVelocity();
618.               recovery_trigger_ = PLANNING_R;
619.             }
620.
621.             lock.unlock();
622.           }
623.
624.           //take the mutex for the next iteration
625.           lock.lock();
626.
627.           //setup sleep interface if needed
628.           if(planner_frequency_ > 0){
629.             ros::Duration sleep_time = (start_time + ros::Duratio
   n(1.0/planner_frequency_)) - ros::Time::now();
630.             if (sleep_time > ros::Duration(0.0)){
631.               wait_for_wake = true;
632.               timer = n.createTimer(sleep_time, &MoveBase::wakePl
   anner, this);
633.             }
634.           }
635.         }
636.       }
637.
638.       void MoveBase::executeCb(const move_base_msgs::MoveBaseGoal
   ConstPtr& move_base_goal)
639.       {
640.         if(!isQuaternionValid(move_base_goal-
   >target_pose.pose.orientation)){
641.           as_-
   >setAborted(move_base_msgs::MoveBaseResult(), "Aborting on goal
   because it was sent with an invalid quaternion");
642.           return;
643.         }
644.
645.         geometry_msgs::PoseStamped goal = goalToGlobalFrame(move_
   base_goal->target_pose);
646.
647.         //we have a goal so start the planner
648.         boost::unique_lock<boost::recursive_mutex> lock(planner_m
   utex_);
649.         planner_goal_ = goal;
650.         runPlanner_ = true;
651.         planner_cond_.notify_one();
652.         lock.unlock();
```

```
653.
654.         current_goal_pub_.publish(goal);
655.         std::vector<geometry_msgs::PoseStamped> global_plan;
656.
657.         ros::Rate r(controller_frequency_);
658.         if(shutdown_costmaps_){
659.           ROS_DEBUG_NAMED("move_base","Starting up costmaps that
   were shut down previously");
660.           planner_costmap_ros_->start();
661.           controller_costmap_ros_->start();
662.         }
663.
664.         //we want to make sure that we reset the last time we had
   a valid plan and control
665.         last_valid_control_ = ros::Time::now();
666.         last_valid_plan_ = ros::Time::now();
667.         last_oscillation_reset_ = ros::Time::now();
668.         planning_retries_ = 0;
669.
670.         ros::NodeHandle n;
671.         while(n.ok())
672.         {
673.           if(c_freq_change_)
674.           {
675.             ROS_INFO("Setting controller frequency to %.2f",
   controller_frequency_);
676.             r = ros::Rate(controller_frequency_);
677.             c_freq_change_ = false;
678.           }
679.
680.           if(as_->isPreemptRequested()){
681.             if(as_->isNewGoalAvailable()){
682.               //if we're active and a new goal is available,
   we'll accept it, but we won't shut anything down
683.               move_base_msgs::MoveBaseGoal new_goal = *as_-
   >acceptNewGoal();
684.
685.               if(!isQuaternionValid(new_goal.target_pose.pose.ori
   entation)){
686.                 as_-
   >setAborted(move_base_msgs::MoveBaseResult(), "Aborting on goal
   because it was sent with an invalid quaternion");
687.                 return;
688.               }
689.
690.               goal = goalToGlobalFrame(new_goal.target_pose);
691.
692.               //we'll make sure that we reset our state for the
   next execution cycle
693.               recovery_index_ = 0;
694.               state_ = PLANNING;
695.
696.               //we have a new goal so make sure the planner is
   awake
697.               lock.lock();
698.               planner_goal_ = goal;
699.               runPlanner_ = true;
```

```
700.               planner_cond_.notify_one();
701.               lock.unlock();
702.
703.               //publish the goal point to the visualizer
704.               ROS_DEBUG_NAMED("move_base","move_base has received
   a goal of x: %.2f, y: %.2f", goal.pose.position.x,
   goal.pose.position.y);
705.               current_goal_pub_.publish(goal);
706.
707.               //make sure to reset our timeouts and counters
708.               last_valid_control_ = ros::Time::now();
709.               last_valid_plan_ = ros::Time::now();
710.               last_oscillation_reset_ = ros::Time::now();
711.               planning_retries_ = 0;
712.             }
713.             else {
714.               //if we've been preempted explicitly we need to
   shut things down
715.               resetState();
716.
717.               //notify the ActionServer that we've successfully
   preempted
718.               ROS_DEBUG_NAMED("move_base","Move base preempting
   the current goal");
719.               as_->setPreempted();
720.
721.               //we'll actually return from execute after
   preempting
722.               return;
723.             }
724.           }
725.
726.           //we also want to check if we've changed global frames
   because we need to transform our goal pose
727.           if(goal.header.frame_id != planner_costmap_ros_-
   >getGlobalFrameID()){
728.             goal = goalToGlobalFrame(goal);
729.
730.             //we want to go back to the planning state for the
   next execution cycle
731.             recovery_index_ = 0;
732.             state_ = PLANNING;
733.
734.             //we have a new goal so make sure the planner is
   awake
735.             lock.lock();
736.             planner_goal_ = goal;
737.             runPlanner_ = true;
738.             planner_cond_.notify_one();
739.             lock.unlock();
740.
741.             //publish the goal point to the visualizer
742.             ROS_DEBUG_NAMED("move_base","The global frame for
   move_base has changed, new frame: %s, new goal position x: %.2f, y:
   %.2f", goal.header.frame_id.c_str(), goal.pose.position.x,
   goal.pose.position.y);
743.             current_goal_pub_.publish(goal);
```

```
744.
745.             //make sure to reset our timeouts and counters
746.             last_valid_control_ = ros::Time::now();
747.             last_valid_plan_ = ros::Time::now();
748.             last_oscillation_reset_ = ros::Time::now();
749.             planning_retries_ = 0;
750.           }
751.
752.           //for timing that gives real time even in simulation
753.           ros::WallTime start = ros::WallTime::now();
754.
755.           //the real work on pursuing a goal is done here
756.           bool done = executeCycle(goal, global_plan);
757.
758.           //if we're done, then we'll return from execute
759.           if(done)
760.             return;
761.
762.           //check if execution of the goal has completed in some
   way
763.
764.           ros::WallDuration t_diff = ros::WallTime::now() - start
   ;
765.           ROS_DEBUG_NAMED("move_base","Full control cycle time:
   %.9f\n", t_diff.toSec());
766.
767.           r.sleep();
768.           //make sure to sleep for the remainder of our cycle
   time
769.           if(r.cycleTime() > ros::Duration(1 / controller_frequen
   cy_) && state_ == CONTROLLING)
770.             ROS_WARN("Control loop missed its desired rate of
   %.4fHz... the loop actually took %.4f seconds",
   controller_frequency_, r.cycleTime().toSec());
771.         }
772.
773.         //wake up the planner thread so that it can exit cleanly
774.         lock.lock();
775.         runPlanner_ = true;
776.         planner_cond_.notify_one();
777.         lock.unlock();
778.
779.         //if the node is killed then we'll abort and return
780.         as_-
   >setAborted(move_base_msgs::MoveBaseResult(), "Aborting on the goal
   because the node has been killed");
781.         return;
782.       }
783.
784.       double MoveBase::distance(const geometry_msgs::PoseStamped&
    p1, const geometry_msgs::PoseStamped& p2)
785.       {
786.         return hypot(p1.pose.position.x - p2.pose.position.x,
   p1.pose.position.y - p2.pose.position.y);
787.       }
788.
```

```
789.       bool MoveBase::executeCycle(geometry_msgs::PoseStamped& goa
   l, std::vector<geometry_msgs::PoseStamped>& global_plan){
790.         boost::recursive_mutex::scoped_lock ecl(configuration_mut
   ex_);
791.         //we need to be able to publish velocity commands
792.         geometry_msgs::Twist cmd_vel;
793.
794.         //update feedback to correspond to our curent position
795.         geometry_msgs::PoseStamped global_pose;
796.         getRobotPose(global_pose, planner_costmap_ros_);
797.         const geometry_msgs::PoseStamped& current_position = glob
   al_pose;
798.
799.         //push the feedback out
800.         move_base_msgs::MoveBaseFeedback feedback;
801.         feedback.base_position = current_position;
802.         as_->publishFeedback(feedback);
803.
804.         //check to see if we've moved far enough to reset our
   oscillation timeout
805.         if(distance(current_position,
   oscillation_pose_) >= oscillation_distance_)
806.         {
807.           last_oscillation_reset_ = ros::Time::now();
808.           oscillation_pose_ = current_position;
809.
810.           //if our last recovery was caused by oscillation, we
   want to reset the recovery index
811.           if(recovery_trigger_ == OSCILLATION_R)
812.             recovery_index_ = 0;
813.         }
814.
815.         //check that the observation buffers for the costmap are
   current, we don't want to drive blind
816.         if(!controller_costmap_ros_->isCurrent()){
817.           ROS_WARN("[%s]:Sensor data is out of date, we're not
   going to allow commanding of the base for
   safety",ros::this_node::getName().c_str());
818.           publishZeroVelocity();
819.           return false;
820.         }
821.
822.         //if we have a new plan then grab it and give it to the
   controller
823.         if(new_global_plan_){
824.           //make sure to set the new plan flag to false
825.           new_global_plan_ = false;
826.
827.           ROS_DEBUG_NAMED("move_base","Got a new plan...swap
   pointers");
828.
829.           //do a pointer swap under mutex
830.           std::vector<geometry_msgs::PoseStamped>* temp_plan = co
   ntroller_plan_;
831.
832.           boost::unique_lock<boost::recursive_mutex> lock(planner
   _mutex_);
```

```
833.           controller_plan_ = latest_plan_;
834.           latest_plan_ = temp_plan;
835.           lock.unlock();
836.           ROS_DEBUG_NAMED("move_base","pointers swapped!");
837.
838.           if(!tc_->setPlan(*controller_plan_)){
839.             //ABORT and SHUTDOWN COSTMAPS
840.             ROS_ERROR("Failed to pass global plan to the
   controller, aborting.");
841.             resetState();
842.
843.             //disable the planner thread
844.             lock.lock();
845.             runPlanner_ = false;
846.             lock.unlock();
847.
848.             as_-
   >setAborted(move_base_msgs::MoveBaseResult(), "Failed to pass
   global plan to the controller.");
849.             return true;
850.           }
851.
852.           //make sure to reset recovery_index_ since we were able
   to find a valid plan
853.           if(recovery_trigger_ == PLANNING_R)
854.             recovery_index_ = 0;
855.         }
856.
857.         //the move_base state machine, handles the control logic
   for navigation
858.         switch(state_){
859.           //if we are in a planning state, then we'll attempt to
   make a plan
860.           case PLANNING:
861.             {
862.               boost::recursive_mutex::scoped_lock lock(planner_mu
   tex_);
863.               runPlanner_ = true;
864.               planner_cond_.notify_one();
865.             }
866.             ROS_DEBUG_NAMED("move_base","Waiting for plan, in the
   planning state.");
867.             break;
868.
869.           //if we're controlling, we'll attempt to find valid
   velocity commands
870.           case CONTROLLING:
871.             ROS_DEBUG_NAMED("move_base","In controlling state.");
872.
873.             //check to see if we've reached our goal
874.             if(tc_->isGoalReached()){
875.               ROS_DEBUG_NAMED("move_base","Goal reached!");
876.               resetState();
877.
878.               //disable the planner thread
879.               boost::unique_lock<boost::recursive_mutex> lock(pla
   nner_mutex_);
```

```
880.               runPlanner_ = false;
881.               lock.unlock();
882.
883.               as_-
   >setSucceeded(move_base_msgs::MoveBaseResult(), "Goal reached.");
884.               return true;
885.             }
886.
887.             //check for an oscillation condition
888.             if(oscillation_timeout_ > 0.0 &&
889.                 last_oscillation_reset_ + ros::Duration(oscillati
   on_timeout_) < ros::Time::now())
890.             {
891.               publishZeroVelocity();
892.               state_ = CLEARING;
893.               recovery_trigger_ = OSCILLATION_R;
894.             }
895.
896.             {
897.              boost::unique_lock<costmap_2d::Costmap2D::mutex_t> l
   ock(*(controller_costmap_ros_->getCostmap()->getMutex()));
898.
899.             if(tc_->computeVelocityCommands(cmd_vel)){
900.               ROS_DEBUG_NAMED( "move_base", "Got a valid command
   from the local planner: %.3lf, %.3lf, %.3lf",
901.                                cmd_vel.linear.x,
   cmd_vel.linear.y, cmd_vel.angular.z );
902.               last_valid_control_ = ros::Time::now();
903.               //make sure that we send the velocity command to
   the base
904.               vel_pub_.publish(cmd_vel);
905.               if(recovery_trigger_ == CONTROLLING_R)
906.                 recovery_index_ = 0;
907.             }
908.             else {
909.               ROS_DEBUG_NAMED("move_base", "The local planner
   could not find a valid plan.");
910.               ros::Time attempt_end = last_valid_control_ + ros::
   Duration(controller_patience_);
911.
912.               //check if we've tried to find a valid control for
   longer than our time limit
913.               if(ros::Time::now() > attempt_end){
914.                 //we'll move into our obstacle clearing mode
915.                 publishZeroVelocity();
916.                 state_ = CLEARING;
917.                 recovery_trigger_ = CONTROLLING_R;
918.               }
919.               else{
920.                 //otherwise, if we can't find a valid control,
   we'll go back to planning
921.                 last_valid_plan_ = ros::Time::now();
922.                 planning_retries_ = 0;
923.                 state_ = PLANNING;
924.                 publishZeroVelocity();
925.
```

```
926.                 //enable the planner thread in case it isn't
   running on a clock
927.                 boost::unique_lock<boost::recursive_mutex> lock(p
   lanner_mutex_);
928.                 runPlanner_ = true;
929.                 planner_cond_.notify_one();
930.                 lock.unlock();
931.               }
932.             }
933.             }
934.
935.             break;
936.
937.           //we'll try to clear out space with any user-provided
   recovery behaviors
938.           case CLEARING:
939.             ROS_DEBUG_NAMED("move_base","In clearing/recovery
   state");
940.             //we'll invoke whatever recovery behavior we're
   currently on if they're enabled
941.             if(recovery_behavior_enabled_ && recovery_index_ < re
   covery_behaviors_.size()){
942.               ROS_DEBUG_NAMED("move_base_recovery","Executing
   behavior %u of %zu", recovery_index_, recovery_behaviors_.size());
943.               recovery_behaviors_[recovery_index_]-
   >runBehavior();
944.
945.               //we at least want to give the robot some time to
   stop oscillating after executing the behavior
946.               last_oscillation_reset_ = ros::Time::now();
947.
948.               //we'll check if the recovery behavior actually
   worked
949.               ROS_DEBUG_NAMED("move_base_recovery","Going back to
   planning state");
950.               last_valid_plan_ = ros::Time::now();
951.               planning_retries_ = 0;
952.               state_ = PLANNING;
953.
954.               //update the index of the next recovery behavior
   that we'll try
955.               recovery_index_++;
956.             }
957.             else{
958.               ROS_DEBUG_NAMED("move_base_recovery","All recovery
   behaviors have failed, locking the planner and disabling it.");
959.               //disable the planner thread
960.               boost::unique_lock<boost::recursive_mutex> lock(pla
   nner_mutex_);
961.               runPlanner_ = false;
962.               lock.unlock();
963.
964.               ROS_DEBUG_NAMED("move_base_recovery","Something
   should abort after this.");
965.
966.               if(recovery_trigger_ == CONTROLLING_R){
```

```
967.                 ROS_ERROR("Aborting because a valid control could
   not be found. Even after executing all recovery behaviors");
968.                 as_-
   >setAborted(move_base_msgs::MoveBaseResult(), "Failed to find a
   valid control. Even after executing recovery behaviors.");
969.               }
970.               else if(recovery_trigger_ == PLANNING_R){
971.                 ROS_ERROR("Aborting because a valid plan could
   not be found. Even after executing all recovery behaviors");
972.                 as_-
   >setAborted(move_base_msgs::MoveBaseResult(), "Failed to find a
   valid plan. Even after executing recovery behaviors.");
973.               }
974.               else if(recovery_trigger_ == OSCILLATION_R){
975.                 ROS_ERROR("Aborting because the robot appears to
   be oscillating over and over. Even after executing all recovery
   behaviors");
976.                 as_-
   >setAborted(move_base_msgs::MoveBaseResult(), "Robot is
   oscillating. Even after executing recovery behaviors.");
977.               }
978.               resetState();
979.               return true;
980.             }
981.             break;
982.           default:
983.             ROS_ERROR("This case should never be reached,
   something is wrong, aborting");
984.             resetState();
985.             //disable the planner thread
986.             boost::unique_lock<boost::recursive_mutex> lock(plann
   er_mutex_);
987.             runPlanner_ = false;
988.             lock.unlock();
989.             as_-
   >setAborted(move_base_msgs::MoveBaseResult(), "Reached a case that
   should not be hit in move_base. This is a bug, please report it.");
990.             return true;
991.         }
992.
993.         //we aren't done yet
994.         return false;
995.       }
996.
997.       bool MoveBase::loadRecoveryBehaviors(ros::NodeHandle node){
998.         XmlRpc::XmlRpcValue behavior_list;
999.         if(node.getParam("recovery_behaviors", behavior_list)){
1000.          if(behavior_list.getType() == XmlRpc::XmlRpcValue::Type
   Array){
1001.            for(int i = 0; i < behavior_list.size(); ++i){
1002.              if(behavior_list[i].getType() == XmlRpc::XmlRpcValu
   e::TypeStruct){
1003.                if(behavior_list[i].hasMember("name") && behavior
   _list[i].hasMember("type")){
1004.                  //check for recovery behaviors with the same
   name
```

```
1005.                   for(int j = i + 1; j < behavior_list.size(); j+
   +){
1006.                     if(behavior_list[j].getType() == XmlRpc::XmlR
   pcValue::TypeStruct){
1007.                       if(behavior_list[j].hasMember("name") && be
   havior_list[j].hasMember("type")){
1008.                         std::string name_i = behavior_list[i]["na
   me"];
1009.                         std::string name_j = behavior_list[j]["na
   me"];
1010.                         if(name_i == name_j){
1011.                           ROS_ERROR("A recovery behavior with the
   name %s already exists, this is not allowed. Using the default
   recovery behaviors instead.",
1012.                                     name_i.c_str());
1013.                           return false;
1014.                         }
1015.                       }
1016.                     }
1017.                   }
1018.                 }
1019.                 else{
1020.                   ROS_ERROR("Recovery behaviors must have a name
   and a type and this does not. Using the default recovery behaviors
   instead.");
1021.                   return false;
1022.                 }
1023.               }
1024.               else{
1025.                 ROS_ERROR("Recovery behaviors must be specified
   as maps, but they are XmlRpcType %d. We'll use the default recovery
   behaviors instead.",
1026.                   behavior_list[i].getType());
1027.                 return false;
1028.               }
1029.             }
1030.
1031.             //if we've made it to this point, we know that the
   list is legal so we'll create all the recovery behaviors
1032.             for(int i = 0; i < behavior_list.size(); ++i){
1033.               try{
1034.                 //check if a non fully qualified name has
   potentially been passed in
1035.                 if(!recovery_loader_.isClassAvailable(behavior_li
   st[i]["type"])){
1036.                   std::vector<std::string> classes = recovery_loa
   der_.getDeclaredClasses();
1037.                   for(unsigned int i = 0; i < classes.size(); ++i
   ){
1038.                     if(behavior_list[i]["type"] == recovery_loade
   r_.getName(classes[i])){
1039.                       //if we've found a match... we'll get the
   fully qualified name and break out of the loop
1040.                       ROS_WARN("Recovery behavior specifications
   should now include the package name. You are using a deprecated
   API. Please switch from %s to %s in your yaml file.",
```

```
1041.                           std::string(behavior_list[i]["type"]).c
   _str(), classes[i].c_str());
1042.                       behavior_list[i]["type"] = classes[i];
1043.                       break;
1044.                     }
1045.                   }
1046.                 }
1047.
1048.                 boost::shared_ptr<nav_core::RecoveryBehavior> beh
   avior(recovery_loader_.createInstance(behavior_list[i]["type"]));
1049.
1050.                 //shouldn't be possible, but it won't hurt to
   check
1051.                 if(behavior.get() == NULL){
1052.                   ROS_ERROR("The ClassLoader returned a null
   pointer without throwing an exception. This should not happen");
1053.                   return false;
1054.                 }
1055.
1056.                 //initialize the recovery behavior with its name
1057.                 behavior-
   >initialize(behavior_list[i]["name"], &tf_, planner_costmap_ros_,
   controller_costmap_ros_);
1058.                 recovery_behaviors_.push_back(behavior);
1059.               }
1060.               catch(pluginlib::PluginlibException& ex){
1061.                 ROS_ERROR("Failed to load a plugin. Using default
   recovery behaviors. Error: %s", ex.what());
1062.                 return false;
1063.               }
1064.             }
1065.           }
1066.           else{
1067.             ROS_ERROR("The recovery behavior specification must
   be a list, but is of XmlRpcType %d. We'll use the default recovery
   behaviors instead.",
1068.                 behavior_list.getType());
1069.             return false;
1070.           }
1071.         }
1072.         else{
1073.           //if no recovery_behaviors are specified, we'll just
   load the defaults
1074.           return false;
1075.         }
1076.
1077.         //if we've made it here... we've constructed a recovery
   behavior list successfully
1078.         return true;
1079.       }
1080.
1081.       //we'll load our default recovery behaviors here
1082.       void MoveBase::loadDefaultRecoveryBehaviors(){
1083.         recovery_behaviors_.clear();
1084.         try{
1085.           //we need to set some parameters based on what's been
   passed in to us to maintain backwards compatibility
```

```
1086.           ros::NodeHandle n("~");
1087.           n.setParam("conservative_reset/reset_distance",
   conservative_reset_dist_);
1088.           n.setParam("aggressive_reset/reset_distance",
   circumscribed_radius_ * 4);
1089.
1090.           //first, we'll load a recovery behavior to clear the
   costmap
1091.           boost::shared_ptr<nav_core::RecoveryBehavior> cons_clea
   r(recovery_loader_.createInstance("clear_costmap_recovery/ClearCost
   mapRecovery"));
1092.           cons_clear->initialize("conservative_reset", &tf_,
   planner_costmap_ros_, controller_costmap_ros_);
1093.           recovery_behaviors_.push_back(cons_clear);
1094.
1095.           //next, we'll load a recovery behavior to rotate in
   place
1096.           boost::shared_ptr<nav_core::RecoveryBehavior> rotate(re
   covery_loader_.createInstance("rotate_recovery/RotateRecovery"));
1097.           if(clearing_rotation_allowed_){
1098.             rotate->initialize("rotate_recovery", &tf_,
   planner_costmap_ros_, controller_costmap_ros_);
1099.             recovery_behaviors_.push_back(rotate);
1100.           }
1101.
1102.           //next, we'll load a recovery behavior that will do an
   aggressive reset of the costmap
1103.           boost::shared_ptr<nav_core::RecoveryBehavior> ags_clear
   (recovery_loader_.createInstance("clear_costmap_recovery/ClearCostm
   apRecovery"));
1104.           ags_clear->initialize("aggressive_reset", &tf_,
   planner_costmap_ros_, controller_costmap_ros_);
1105.           recovery_behaviors_.push_back(ags_clear);
1106.
1107.           //we'll rotate in-place one more time
1108.           if(clearing_rotation_allowed_)
1109.             recovery_behaviors_.push_back(rotate);
1110.         }
1111.         catch(pluginlib::PluginlibException& ex){
1112.           ROS_FATAL("Failed to load a plugin. This should not
   happen on default recovery behaviors. Error: %s", ex.what());
1113.         }
1114.
1115.         return;
1116.       }
1117.
1118.       void MoveBase::resetState(){
1119.         // Disable the planner thread
1120.         boost::unique_lock<boost::recursive_mutex> lock(planner_m
   utex_);
1121.         runPlanner_ = false;
1122.         lock.unlock();
1123.
1124.         // Reset statemachine
1125.         state_ = PLANNING;
1126.         recovery_index_ = 0;
1127.         recovery_trigger_ = PLANNING_R;
```

```
1128.         publishZeroVelocity();
1129.
1130.         //if we shutdown our costmaps when we're deactivated...
   we'll do that now
1131.         if(shutdown_costmaps_){
1132.           ROS_DEBUG_NAMED("move_base","Stopping costmaps");
1133.           planner_costmap_ros_->stop();
1134.           controller_costmap_ros_->stop();
1135.         }
1136.       }
1137.
1138.       bool MoveBase::getRobotPose(geometry_msgs::PoseStamped& glo
   bal_pose, costmap_2d::Costmap2DROS* costmap)
1139.       {
1140.         tf2::toMsg(tf2::Transform::getIdentity(),
   global_pose.pose);
1141.         geometry_msgs::PoseStamped robot_pose;
1142.         tf2::toMsg(tf2::Transform::getIdentity(),
   robot_pose.pose);
1143.         robot_pose.header.frame_id = robot_base_frame_;
1144.         robot_pose.header.stamp = ros::Time(); // latest
   available
1145.         ros::Time current_time = ros::Time::now();  // save time
   for checking tf delay later
1146.
1147.         // get robot pose on the given costmap frame
1148.         try
1149.         {
1150.           tf_.transform(robot_pose, global_pose, costmap-
   >getGlobalFrameID());
1151.         }
1152.         catch (tf2::LookupException& ex)
1153.         {
1154.           ROS_ERROR_THROTTLE(1.0, "No Transform available Error
   looking up robot pose: %s\n", ex.what());
1155.           return false;
1156.         }
1157.         catch (tf2::ConnectivityException& ex)
1158.         {
1159.           ROS_ERROR_THROTTLE(1.0, "Connectivity Error looking up
   robot pose: %s\n", ex.what());
1160.           return false;
1161.         }
1162.         catch (tf2::ExtrapolationException& ex)
1163.         {
1164.           ROS_ERROR_THROTTLE(1.0, "Extrapolation Error looking up
   robot pose: %s\n", ex.what());
1165.           return false;
1166.         }
1167.
1168.         // check if global_pose time stamp is within costmap
   transform tolerance
1169.         if (current_time.toSec() - global_pose.header.stamp.toSec
   () > costmap->getTransformTolerance())
1170.         {
1171.           ROS_WARN_THROTTLE(1.0, "Transform timeout for %s. " \
```

```
1172.                                 "Current time: %.4f, pose stamp:
   %.4f, tolerance: %.4f", costmap->getName().c_str(),
1173.                                 current_time.toSec(),
   global_pose.header.stamp.toSec(), costmap-
   >getTransformTolerance());
1174.           return false;
1175.         }
1176.
1177.         return true;
1178.       }
1179.     };
```