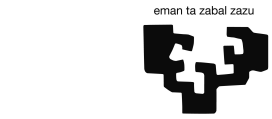

Máster en Ingeniería Computacional y Sistemas Inteligentes

Proyecto Fin de Máster

Mejora de estrategias de navegación en el robot de servicio kTBot/Teknibot

---

Autor
Iker Lluvia Hermosilla

Dirigido por
Dr. Basilio Sierra Araujo y Dr. Ander Ansuategi Cobo

2017

Agradecimientos

Antes de comenzar quiero agradecer a mi familia por el apoyo que me ha dado a lo largo toda de mi vida, especialmente a mi Ama. Sin ellos no podría haber aprendido lo que sé.

También me gustaría dar las gracias a IK4-TEKNIKER por ofrecerme la oportunidad de trabajar con ellos y vivir una muy buena experiencia. Sobretodo a Ander por guiarme y prestarme su ayuda cuando lo he necesitado.

Doy gracias a la facultad en general por haberme dado unos estudios de calidad y en particular a Basi ya que es él quien me ha orientado en una parte importante de mi formación con la que estoy muy contento.

Y, por último, a la comunidad open source por su actitud cooperativa compartiendo sus conocimientos con transparencia.

Eskerrik asko.

Resumen

La idea de este proyecto es volver a poner en marcha un robot en el que parte del software que utiliza se ha quedado obsoleto. Éste corresponde al área de la robótica de servicio, puesto que está diseñado para hacer de asistente a las personas haciendo de guía en entornos interiores a la vez que ofrece información relevante para los visitantes. Aunque el principal cambio en la migración consista en la actualización del sistema operativo y el framework utilizado para el gobierno del robot, se pretende modificar todos los demás paquetes y librerías que se utilicen de forma que hagan uso de sus ultimas versiones que serán, en principio, las que mejores resultados obtengan. Asímismo, se quiere incorporar nuevas funcionalidades o mejoras en las diferentes fases del robot como la navegación, el rendimiento general o la comunicación. Pese a que esto último no abarque gran parte del trabajo realizado, es el principal motivo que le ha dado vida al proyecto.

Este trabajo está dividido en cinco capítulos. El primero de ellos hace una introducción al mismo explicando las motivaciones del proyecto y los objetivos o retos que se plantean. Después se hace un breve repaso a la literatura, donde se describen los principales algoritmos utilizados para abordar los problemas que surgen en la navegación autónoma. En tercer lugar se especifican la mayoría de los componentes electrónicos incorporados en el sistema que hacen posible un correcto funcionamiento del robot. A continuación se explica en qué consiste el desarrollo del proyecto en sí y los procedimientos llevados a cabo para realizar tanto la migración como la mejora del software. Para finalizar, se concluye el trabajo con las deducciones, lecciones e ideas que han surgido en la realización de este proyecto y las cambios que se quieren realizar en un futuro con el fin de hacer evolucionar al robot.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Introducción

La robótica se ha convertido en la raíz de muchas investigaciones debido a la infinidad de aplicaciones que ofrece. En la industria es donde ha tenido mayor repercusión, ya que muchas de las tareas que antes eran realizadas por personas están siendo sustituidas por robots, que las cumplen con un gran nivel de eficiencia. Sin embargo, el éxito de los robots domésticos se ha demorado debido a la dificultad de controlar y adaptarse a los entornos altamente cambiantes con los que se encuentran.

En este proyecto se aborda un problema de la robótica de servicio, que se podría decir que está a caballo entre las dos áreas mencionadas, ya que el entorno y muchas de las tareas son predecibles pero dispone de la interacción con personas creando demasiada incertidumbre en la toma de decisiones, lo que da lugar a un gran número de situaciones críticas que gestionar. Y es que para que los robots móviles puedan ejercer de manera satisfactoria sus tareas es necesario cumplir una serie de premisas, tales como:

- Posicionamiento absoluto en el área de trabajo
- Navegación en entornos dinámicos y desconocidos
- Percepción del entorno para una evitación de obstáculos robusta
- Recuperación de situaciones conflictivas
- Interacción persona-robot
- Seguridad

Desde un punto de vista científico, hay métodos que dan solución a estas cuestiones pero ninguno lo suficientemente fiable como para no fallar en un entorno real, ya que son probados en condiciones de laboratorio sin tanta incertidumbre ni situaciones inesperadas como las que se dan realizando la labor para los que están destinados. Existe, por tanto, la necesidad de seguir investigando con el objetivo de conseguir técnicas robustas que puedan dar una respuesta efectiva ante cualquier problemática que se dé, por anómala que sea. Además, la presencia humana será constante y los robots no sólo tiene que reconocerla, sino que deben actuar consecuentemente haciendo además que la convivencia entre ambos sea natural evitando situaciones que puedan incomodar a las personas.

1.2. Descripción

El objetivo del proyecto es poner en marcha Teknibot, un robot de servicio móvil y autónomo que incluya funcionalidades básicas especialmente enfocadas a actividades de asistencia en entornos interiores (e.g.. hogar, residencias, hospitales, museos, oficinas, lugares de trabajo, etc.).
Los retos planteados en el proyecto se centran en:

- Integrar robots en entornos a priori desconocidos, continuamente cambiantes y con una importante presencia humana.
- Crear robots capaces de interaccionar de forma natural con las personas, detectando su presencia y adecuando su comportamiento en consonancia a la situación.

Para ello se desarrollan tecnologías en tres niveles:

- La navegación autónoma. Dotar al robot con la capacidad de desplazarse de forma autónoma en entornos dinámicos, planificando y ejecutando trayectorias, evitando obstáculos y dando una respuesta eficaz a cualquier situación compleja.
- La interacción persona-robot. Poder detectar personas a su alrededor e interactuar con ellas con la posibilidad de atender a ciertas necesidades que estas puedan tener.
- Fiabilidad y seguridad implementando metodologías para el análisis de los complementos propios del robot. Este es un aspecto crítico en la robótica ya que un comportamiento inadecuado en uno de ellos puede suponer una mala percepción del entorno y/o una respuesta errónea ante las circunstancias dadas.

Sobre estas capacidades tecnológicas se han desarrollado las funcionalidades de Teknibot adecuándolas al contexto de uso en este caso el centro tecnológico IK4-TEKNIKER.

1.2.1. IK4-TEKNIKER

IK4-TEKNIKER es un centro tecnológico integrado por personas con vocación y compromiso por impulsar la capacidad innovadora de sus clientes e incrementar su capital tecnológico para mejorar su competitividad de forma sostenible, a través de la generación y aplicación del conocimiento científico-tecnológico. La definición y despliegue de los valores que definen la personalidad del centro, interna y externamente, son la orientación a resultados y al

Figura 1.1: kTBot en el museo Eureka.

cliente, la excelencia, la confianza y el compromiso. Constituido jurídicamente como Fundación privada sin ánimo de lucro, en cuyo patronazgo y junta está mayoritariamente representada la industria.

En la actualidad IK4-TEKNIKER forma parte de la Alianza Tecnológica IK4, de la que también son socios los otros centros tecnológicos: Azterlan, Ceit, Cidetec, Gaiker, Ideko, Ikerlan, Lortek y Vicomtech. La alianza IK4 fue constituida en 2005 según un modelo federal por el que sus integrantes comparten estrategias y combinan capacidades sin renunciar a su soberanía. La suma de las capacidades de los nueve centros a través de una estrategia común proporciona a la alianza la flexibilidad precisa para adaptarse a las características de todas las empresas. Es decir, la capacita tecnológicamente para dar una respuesta global a las necesidades de la gran empresa al tiempo que potencia su capacidad para mantener una relación de cercanía, proximidad y compromiso con las pymes. IK4 es hoy un referente en el entorno europeo, situándose entre las principales corporaciones científico-tecnológicas privadas del continente. Cabe destacar el papel protagonista de IK4 en el VII Programa Marco de la Unión Europea, habiendo tomado parte en más de 200 proyectos de investigación, de los cuales ha liderado 63. IK4 se sitúa en una posición ideal para establecer relaciones a largo plazo como aliado estratégico de la empresa en materia de innovación. Esto hace de IK4 un referente en participación en proyectos de investigación impulsados por las administraciones públicas.

1.2.2. kTBot

El robot desarrollado es una copia de kTBot, un robot capaz de navegar de manera autónoma en Eureka! Zientzia Museoa haciendo tareas de guía para los visitantes del museo y realizando presentaciones de contenidos en las diferentes salas. Acompaña a los grupos hasta las salas y aporta curiosidades científicas y mensajes relacionados con los contenidos del museo. También ofrece asistencia en tareas de recepción y acompañamiento de visitantes.

1.2.3. Eureka!

Eureka Zientzia Museoa, Obra Social de Kutxa, es un recurso cultural y educativo al servicio de nuestra sociedad.

Situado en el parque tecnológico Miramón de San Sebastián, en un entorno natural privilegiado, Eureka Zientzia Museoa es un museo interactivo donde la información se presenta de forma atractiva, con un nuevo estilo de comunicación a través de la manipulación de objetos y la realización de

experimentos. Eureka Zientzia Museoa pretende proporcionar un entorno estimulante para la participación en actividades relacionadas con el mundo de la ciencia y la técnica. Se podría definir como un centro de divulgación científica accesible a todas las edades, como complemento de los programas educativos para la comprensión de los principios científicos y como recurso para familias e individuos en su tiempo de ocio, dirigido a su desarrollo personal.

1.3. Objetivos

El objetivo del proyecto es poner en funcionamiento un robot similar al ya mencionado kTBot, aumentando su fiabilidad, mejorando sus funcionalidades y añadiendo algunas nuevas. Se pretende realizar una implementación más robusta, que ofrezca un mayor control y seguridad respecto a terceros que intenten acceder al sistema. También es importante que su programación sea fácilmente adaptable a futuras modificaciones ya que el área en el que nos encontramos está en continuo crecimiento y poder incorporar lo último en tecnología mejorará su rendimiento. Otro de los objetivos del proyecto es rediseñar la estructura de la implementación, encapsulando cada función con la finalidad de facilitar su gestión y simplificar su migración a otros entornos. Asímismo se requiere que dichas funciones sean igualmente parametrizables aumentando la rapidez con la que se puedan integrar en otros proyectos.

1.3.1. Rehabilitación del robot

En anteriores apartados hemos visto que partimos de una base sin la necesidad de partir desde cero, sin embargo, gran parte de esa base ha quedado obsoleta. Algunos protocolos han cambiado y, en consecuencia, hay que adaptar los controladores de los componentes para que éstos convivan. Otros de los drivers simplemente ha dejado de recibir soporte técnico con lo cual su funcionamiento puede no ser el esperado, de modo que hay que encontrar una alternativa que se adecue a nuestros intereses. Otra parte del código simplemente ha evolucionado y, fieles a nuestros principios, queremos añadir las mejores aportaciones al mismo lo que incluye un gran número de nuevos parámetros. Esto conlleva un nuevo abanico de posibilidades de funcionar los cuales deberemos estudiar y aplicar el más conveniente en cada caso. Aparte de estos cambios habituales en lo que a una actualización de un sistema informático se refiere, hay dos puntos a tener en cuenta, los cuales nos van a condicionar implementación en su totalidad; Ubuntu y ROS.

Ubuntu

El sistema operativo que gestiona los recursos de hardware y provee servicios a los programas de aplicación de software es Ubuntu. Es bien conocido que esta distribución basada en Debian GNU/Linux ofrece robustez y soporte debido a sus lanzamientos regulares y su continua versión LTS (*Long Term Support*). Además, al ser distribuido como software libre tenemos acceso al código fuente y la libertad de estudiar cómo trabaja el programa pudiendo modificarlo para que haga lo que nosotros queramos, lo que es de vital importancia en este proyecto ya que queremos un control absoluto del mismo. Por otra parte, al ser un producto de código abierto hay una gran comunidad que trabaja con él facilitando la resolución de los problemas que puedan surgir. Esta forma de trabajar hace honor al nombre del producto puesto que, curiosamente, *ubuntu* es una regla ética sudafricana y se traduce como 'si todos ganan, tú ganas' o 'todo lo que es mio, es para todos'.

Concretamente, la migración que realizamos es de la versión Ubuntu 10.04 LTS Lucid Lynx (abril de 2010) a la 16.04 LTS Xenial Xerus (abril de 2016). Pese a sus muchas ventajas frente a otros sistemas operativos, la realidad es que es un requisito que debemos cumplir debido a que nuestra mayor prioridad es utilizar ROS para el manejo del robot.

ROS

ROS (*Robot Operating System*) es un entorno de trabajo para el desarrollo de robots que provee la funcionalidad de un sistema operativo, con un modelo distribuido [2]. Se monta sobre Ubuntu, aunque ha habido algún intento sin mucho éxito de hacerlo multiplataforma. ROS fue creado en 2007 por el Laboratorio de Inteligencia Artificial de Stanford para dar soporte al conocido STAIR (*STanford Artificial Intelligence Robot*). Sin embargo, era cuestión de tiempo que este proyecto se expandiera debido al aumento de la oferta y demanda de las unidades robóticas, cada una con sus propios firmware y procedimientos para poder ser controladas. ROS pretende ser un *middleware* robótico genérico y compatible con todos los modelos. De esta forma los desarrolladores pueden focalizarse en aprender y/o mejorar las técnicas que ofrezcan un mejor rendimiento de los robots en lugar de perder horas entendiendo cual es el protocolo para trabajar con un modelo en concreto. Como se puede deducir, ROS también es software libre bajo términos de licencia BSD (*Berkeley Software Distribution*), la cual permite libertad para uso comercial e investigador. Al igual que en el caso anterior, hay una gran comunidad realizando aportaciones continuamente, tanto corrigiendo

errores como creando nuevas funcionalidades, lo que hace posible un avance eficiente de la tecnología. En este caso la migración corresponde a actualizar ROS Diamondback (marzo de 2011) a ROS Kinetic Kame (mayo de 2016).

Otros

Además de ROS y Ubuntu, hay otros software de terceros que debemos actualizar como es el caso de Qt u OpenCV, que veremos más adelante.

1.3.2. Nuevas funcionalidades

Aunque gran parte del proyecto consiste en la actualización del software y en ella ya vienen implícitas muchas mejoras, queremos dedicarle cierto tiempo a añadir nuevas funcionalidades que creemos pueden aportar un mayor grado de robustez y estabilidad en la práctica.

kTBot tiene varios puntos claramente mejorables, tanto en la localización como en la navegación como en la interacción persona-robot. En primer lugar, para que el robot mantenga una posición relativamente precisa la anterior también debe haberlo sido, en caso de que suceda a una ubicación errónea o desconocida su corrección será costosa o imposible. En segundo lugar, cuando el robot está navegando y los obstáculos son los esperados no existe incertidumbre en que alcanzará su destino con éxito. Sin embargo, dificultades que impidan continuar con la trayectoria establecida pueden anularla completamente, obligando al robot a entrar en una fase de recuperación poco 'inteligente' y demasiado sorprendente para los usuarios, lo que genera cierto grado de desconfianza en el sistema. Por último, la interacción persona robot está limitada a una comunicación mediante una pantalla táctil incorporada a la máquina y varios comandos de voz que ésta pueda dar, luego convendría ofrecer una mayor versatilidad.

Autolocalización

El mayor problema que tiene kTBot es localizarse por primera vez al inicializar el sistema y una vez que éste se ha perdido, esto es, el robot mantiene correctamente la trayectoria pero tiene dificultades cuando ésta se desvía o desconoce. La solución propuesta para resolver la primera de las cuestiones es determinar cierto punto inicial conocido para el robot de manera que siempre empiece en esa misma zona después de un reinicio del sistema. En cuanto al segundo problema se refiere, no dispone de un método que dé salida al mismo de modo que todos los esfuerzos se centran en reducir la probabilidad de que

se dé esa casuística. Se pretende establecer un método que sepa localizarse con más o menos exactitud en todo momento, independientemente del estado que le preceda.

Recuperación

El procedimiento del que dispone a modo de recuperación al no ser posible, o el robot así lo determina, alcanzar el destino actual es comenzar una rotación sobre sí mismo y esperar que el impedimento ya no lo sea o se encuentre otro itinerario exitoso. En el caso de que ninguna condición se cumpla, volverá a rotar 360º sobre sí mismo para dar por finalizada la tarea al no haber encontrado una alternativa válida. La idea es que el nuevo sistema de recuperación se centre más en la situación actual del robot, el problema en concreto que impide alcanzar el destino y proceda con una actuación acorde. De todos modos, en el ámbito de uso en el que nos encontramos, quizá lo más efectivo sea priorizar la imagen que proyectamos a los usuarios de 'inteligencia' y no el hecho de llegar al destino en sí. Son aspectos en los que la fase experimental es de gran ayuda y que veremos más adelante.

Interacción persona-robot

Como ya hemos mencionado anteriormente, kTBot incorpora una pantalla táctil con la que podemos comunicarnos con él. Ésta dispone de una interfaz de usuario que nos facilita mandarle órdenes, como enviarlo a un destino, cambiar el modo de operación, configurar ciertos ajustes, etc. aunque para algunas opciones necesitamos ser administradores. Además, el robot también tiene un altavoz que le permite compartir información acústica y por el que podemos recibir avisos. Aparte de esta vía de comunicación, sería interesante que pudiéramos hacerlo de forma remota y basta con contemplar el entorno en el que nos encontramos para deducir que la manera más cómoda, útil y práctica es tener la posibilidad de hacerlo a través de un smartphone. Para ello hay que reestructurar la lógica del sistema contemplando cuestiones que antes no eran necesarias.

Capítulo 2

Estado del arte

2.1. Introducción

En este capítulo veremos los avances que se han logrado hasta el momento y los algoritmos que mejores resultados obtienen en las tareas que nos corresponden. Aunque este proyecto no tenga un espíritu puramente investigador, es interesante conocer las soluciones que existen en la literatura para poder hacer una comparativa y utilizar la que más se asemeje a nuestras características. Como en otros muchos casos, no encontraremos una respuesta universal que sea la más adecuada para todas las situaciones, de modo que el estudio de las mismas es primordial para deducir qué método es el más adecuado en cada momento. Siendo Teknibot un robot autónomo, dividiremos la navegación en tres áreas principales: mapping, localización y path planning [3].

2.2. Mapping

El mapeo robótico o *robot mapping* ha sido objeto de muchas investigaciones en el área de la inteligencia artificial durante más de dos décadas. Dicha tarea aborda el problema de adquirir un modelo espacial de entornos físicos a través de sensores y es vista como una de las más importantes a la hora de construir un robot móvil realmente autónomo. Pese a los grandes avances realizados, todavía siguen apareciendo fuertes retos. En la actualidad, disponemos de métodos robustos para el mapping de entornos estáticos, estructurados y de limitadas dimensiones. Sin embargo, un eficiente mapeo desestructurado, dinámico y de grandes dimensiones es aún un problema abierto a la investigación [1].

La mayoría de los mejores algoritmos que podemos encontrar en la literatura tienen algo en común: todos son probabilísticos. Éstos emplean modelos probabilísticos del robot y su entorno convirtiendo las medidas obtenidas de los sensores en mapas. La razón del éxito de estas técnicas se basa en el hecho que el mapeo robótico está caracterizado por la incertidumbre y el ruido sensorial [4].
Durante los últimos años, el interés en los métodos Bayesianos ha aumentado considerablemente [5] y es que el principio básico que subyace en cada uno de los algoritmos es el teorema de Bayes:

$$p(H|D) = \frac{p(D|H)p(H)}{p(D)}, \tag{2.1}$$

donde $p(D|H)$ es la probabilidad del dato D dada la hipótesis H, $p(H)$ es

	Kalman	Lu/Milios	EM	Incremental ML	Hybrid	Occupancy Grids	Multi-Planar Maps	Dogma
Representation	landmark locations	point obstacles	point obstacles	landmark locations or grid maps	point obstacles	occupancy grids	objects and polygons	occupancy grids
Uncertainty	posterior poses and map	posterior poses and map	maximum likelihood map	(local) maximum likelihood map	maximum likelihood map	posterior map	maximum likelihood map	posterior map
Convergence	strong	no	weak?	no	no	strong	weak	weak
Local Minima	no	yes	yes	yes	yes	no	yes	yes
Incremental	yes	no	no	yes*	yes	yes	no	no
Requires Poses	no	no	no	no	no	yes	yes	yes
Sensor Noise	Gaussian	Gaussian	any	any	any	any	Gaussian	any
Can map cycles	yes	no	yes	no	yes, but not nested	n/a	n/a	n/a
Map dimensionality	$\sim 10^3$	unlimited	unlimited	unlimited	unlimited	unlimited	unlimited	unlimited
Correspondence	no	yes	yes	yes	yes	yes	yes	yes
Handles raw data	no	yes	yes	yes	yes	yes	yes	yes
Dynamic env's	limited	no	no	no	no	limited	no	yes

Figura 2.1: Un intento de comparar los principales algoritmos de mapeo [1].

la probabilidad a priori de que la hipótesis sea cierta, $p(H|D)$ es la probabilidad a posteriori de que la hipótesis sea cierta dado el dato D, y $p(D) = \sum_H p(D|H)p(H)$ es la probabilidad del dato medido sobre todas las hipótesis.
Lo ideal sería detallar los diferentes algoritmos y sus variaciones. Sin embargo, queda fuera del alcance del proyecto de modo que haremos una comparación superficial de las propiedades más relevantes [6, 7] mediante la figura 2.1.

2.3. Localización

La localización de un robot móvil consiste en estimar la pose del robot (posición y orientación) relativa a su entorno. El problema de localización es clave en la robótica móvil ya que de su precisión dependerá el error de muchas de las acciones que se realizan. Dependiendo de las circunstancias, éste se divide en tres subproblemas, siendo el que más atención ha recibido en la literatura el seguimiento de la posición [8, 9, 10]. Aquí se conoce la pose inicial, y el objetivo es compensar el error incremental de la odometría del robot. Algoritmos para el tracking de la posición a menudo asumen restric-

ciones en el tamaño del error y en la incertidumbre del robot. Un mayor reto supone la localización global [11, 12], donde el robot desconoce la posición inicial y debe deducirla él mismo. Dicha cuestión es más compleja ya que no se puede asumir que el error en la estimación del robot sea pequeña. En consecuencia, el sistema debería ser capaz de considerar múltiples hipótesis distintas al mismo tiempo. Por último, y todavía más difícil, se encuentra el problema del robot secuestrado [13, 14], el cual un robot perfectamente localizado es trasladado a otro lugar sin que pueda percibirlo. Este problema se diferencia del de la localización global en que el robot cree saber con exactitud el punto en el que se encuentra tras el cambio. El problema del robot secuestrado es utilizado frecuentemente para medir su capacidad de recuperación ante un fallo de localización descomunal. Es de añadir, que estos problemas son particularmente complejos en entornos dinámicos, e.g. si el robot opera en entornos con personas a su alrededor alterando las mediciones de los sensores [15, 16].

La inmensa mayoría de los algoritmos existentes resuelven el problema del tracking de la posición. Sin embargo, los hay que intentan dar respuesta las tres cuestiones como es el caso del algoritmo probabilístico llamado *Monte Carlo localization* (MCL) o localización por filtro de partículas, utilizado en este proyecto. MCL intenta solucionar la localización global y el problema del robot secuestrado adaptándose a las distribuciones de ruido aleatorias con lo que evita la necesidad de extraer características de la información sensorial. La idea principal de MCL es representar la creencia de un conjunto de ejemplos (también llamados partículas) creada a partir de la distribución posterior sobre las poses del robot. En otras palabras, en vez de realizar la aproximación de forma paramétrica como en el caso de los filtros de Kalman y los algoritmos de localización de Markov [17, 18], MCL lo hace representando una colección aleatoria de partículas con pesos asociados que aproximan la distribución deseada. En el contexto de la localización, dicha representación tiene varias ventajas con respecto a anteriores estrategias. Una de ellas es que se puede adaptar a características arbitrarias del sensor, dinámicas de movimiento y distribuciones de ruido. Otra es que gestionan muy bien el coste computacional pudiendo adaptarse a los recursos disponibles en cada momento, y el código puede ejecutarse en diferentes ordenadores aunque tengan una gran diferencia de velocidad sin modificación alguna del código. Además, los filtros de partículas son fáciles de implementar lo que los hace un ejemplo atractivo para la localización robótica. Sin embargo, también existen ciertos inconvenientes procedentes de la naturaleza estocástica de la aproximación. Entre otros, si el conjunto de ejemplos es pequeño, un robot bien localizado puede perder su posicionamiento debido a que MCL no puede

generar una muestra en la ubicación correcta. El algoritmo también puede no ajustarse al problema del robot secuestrado debido a que no haya ninguna partícula restante próxima a la nueva posición del robot. Por último, MCL podría ser ineficaz en casos en los que los sensores son tan precisos que el ruido generado es nulo o casi. Es por estos motivos por los que es de vital importancia conocer todos los parámetros del algoritmo y poder ajustarlos a nuestras necesidades concretas [19].

2.4. SLAM

En cuanto a la realización del mapa se refiere, la práctica más utilizada y que mejor resultados obtiene es la denominada SLAM, del inglés *Simultaneous Localization And Mapping*. Esta técnica consiste en adquirir un mapa espacial den entorno a le vez que se localiza el robot en base a ese modelo. Como es fácilmente deducible, el problema SLAM es uno de los más importantes si no el más en lo que a la robótica autónoma se refiere [20]. El proceso que se lleva a cabo es el siguiente: un robot móvil deambula por un entorno desconocido, partiendo desde un punto con coordenadas conocidas. Su movimiento es dudoso, haciendo cada vez más difícil determinar sus coordenadas globales. Mientras deambula, el robot percibe el entorno a través de los sensores con lo que genera el mapa y obtiene su localización en el mismo simultáneamente.

Formalmente, SLAM es descrito en la terminología probabilística [21]. Denominemos al tiempo con t, y a la localización del robot con x_t. Para robots móviles en una superficie plana, x_t es un vector tridimensional, siendo dos de ellas la posición en el plano y la tercera el valor rotacional para su orientación. La secuencia de ubicaciones, o trayectoria, viene dada por

$$X_T = x_0, x_1, x_2, ...x_T. \tag{2.2}$$

Aquí T es un instante de tiempo (T puede ser ∞). La posición inicial x_0 es conocida; las demás posiciones no pueden ser percibidas.

La odometría proporciona información relativa entre dos puntos consecutivos. Tomemos u_t como la odometría que representa el movimiento entre el tiempo $t-1$ y el tiempo t; dicha información se debe obtener de los encoders de las ruedas del robot. Luego la secuencia

$$U_T = u_1, u_2, u_3, ...u_T. \tag{2.3}$$

representa el movimiento relativo del robot. En el caso de que el ruido fuera nulo, U_T sería suficiente para deducir X_T de la posición inicial x_0. Sin em-

bargo, las mediciones odométricas son ruidosas y las técnicas de cálculo de trayectorias inevitablemente distan de la realidad.

Finalmente, el robot percibe los objetos en el entorno. Llamemos m al mapa real del entorno, el cual puede estar compuesto por puntos de referencia, objetos, superficies, etc. y m describe sus ubicaciones. Generalmente se asume que el mapa del entorno m es invariante al tiempo (i.e., estático).

Las medidas del robot establecen información entre las características m y las posiciones del robot x_t. Asumiendo que el robot obtiene exactamente una medición en cada instante de tiempo, la secuencia de mediciones viene dada por

$$Z_T = z_1, z_2, z_3, ...z_T. \tag{2.4}$$

El problema SLAM consiste en recuperar un modelo del mundo m y una secuencia de puntos X_T en base a la odometría y las mediciones realizadas.

2.5. Planificación de trayectorias

La planificación de trayectorias o *path planning* es otro de los conceptos principales en lo que a la navegación autónoma se refiere y su objetivo es encontrar el camino óptimo entre dos puntos. Óptimo se refiere generalmente al más corto, pero también puede ser el más recto, el que menos frena o cualquiera que la aplicación requiera. Para poder planificar una trayectoria es necesario un mapa del entorno y que el robot sea consciente de su localización con respecto al mismo. Se asume que el robot cumple ambos requisitos y, además, puede evitar obstáculos temporales en su camino.

Existen dos formas de representar un entorno en el ordenador: la aproximación continua y la discreta [22]. En una aproximación discreta, el mapa está dividido por secciones del mismo tamaño (e.g. una rejilla o mapa hexagonal) o diferentes (e.g., provincias de un estado). Estos últimos son denominados mapas topológicos. Los mapas discretos son fácilmente representables mediante grafos, donde las secciones equivalen a vértices (también llamados 'nodos') que están conectados por aristas si el robot puede navegar de un vértice a otro. Computacionalmente, el grafo se almacena en una lista o matriz de adyacencia. Una aproximación continua requiere definir detalladamente los obstáculos internos y sus límites externos, comúnmente en forma de polígono, donde las trayectorias son codificadas como secuencias de

números reales. Los mapas discretos son la representación predominante en la robótica.

Actualmente el mapa más utilizado es el *occupancy grid map* traducido como malla, rejilla o red de ocupación. Aquí, el entorno es discretizado en cuadrados de tamaño variable (e.g., 1cm x 1cm) en los cuales se marcan los obstáculos pudiendo estar cada uno ocupado o libre. En una malla probabilística, los cuadrados también pueden tomar valores entre 0 y 1 que representan la probabilidad de que haya un obstáculo. Esta posibilidad es particularmente importante cuando la posición del sensor que detecta el objeto es dudosa. Una de las desventajas de los mapas en malla es la cantidad de memoria que necesitan y el alto coste computacional para recorrer las estructuras de datos con un gran número de vértices y/o aristas. La solución a este problema es utilizar mapas topológicos para representar grandes secciones del mapa. No existe una solución óptima y lo eficiente suele ser combinar diferentes modos de representación para obtener una solución. También existe una combinación entre mapas continuos y discretos. Por ejemplo, los mapas de carreteras para sistemas GPS que almacenan los datos topológicamente asociando unas coordenadas GPS específicas a cada vértice.

Algorithm 1 Algoritmo general de búsqueda en un grafo

```
1: Q.insertar(x_0) y marcar x_0 como visitado
2: while Q no esté vacía do
3:     if x ∈ X_Dest then return Éxito
4:     end if
5:     for all u ∈ U(x) do
6:         x' ← f(x, u)
7:         if x' no está visitado then
8:             Marcar x' como visitado Q.insertar(x')
9:         else
10:            Resolver x'
11:        end if
12:    end for
13: end while
14: return Fracaso
```

A continuación se muestran los algoritmos más utilizados en recorrido de grafos [23, 24], siendo cada uno de ellos una variación del pseudocódigo del algoritmo 1.

2.5.1. Búsqueda en anchura

En un recorrido en anchura Q es una cola FIFO o *First-In First-Out*, donde el siguiente punto seleccionado será el primero en ser añadido a la lista. De modo que todas las trayectorias con k pasos sin siempre contempladas antes que alguna con $k+1$ pasos. Su coste computacional es $O(|V|+|E|)$, en el cual $|V|$ y $|E|$ son el número de vértices y aristas, respectivamente, asumiendo que todas las operaciones básicas como determinar si un punto ha sido visitado se realizan en tiempo constante.

2.5.2. Búsqueda en profundidad

En este caso Q es representada como una pila LIFO (*Last-In First-Out*, al contrario que en el método anterior. En este caso el último estado insertado en Q es el primero en considerarse, haciendo que no sea adecuado para mapas discretos, ya que puede ser completamente ineficiente para destinos próximos al origen, sobretodo en el hipotético caso de que el mapa fuera infinito. ya que no terminaría. Su coste computacional es el mismo que en el caso anterior, $O(|V|+|E|)$.

2.5.3. Dijkstra

Hasta ahora, no ha habido preferencia al escoger una acción u otra en la búsqueda. Sin embargo, el algoritmo de Dijkstra determina el camino más corto dado un vértice origen al resto de los vértices en un grafo con pesos en cada arista. Para ello, utiliza una cola de prioridad que es ordenada en base a una función de coste $C : X \rightarrow [0, \infty]$. Para cada estado x, $C*(x)$ es el coste óptimo desde el estado inicial x_0 calculado a través de los costes $l(e)$ de todas las combinaciones de aristas que unen x_0 con x. Si el coste no es óptimo, dicho valor es almacenado en $C(x)$. La función de coste es calculada incrementalmente durante el recorrido. Al iniciar el proceso, $C*(x_0) = 0$ y el resto son desconocidos, para luego ser actualizados tal que $C(x') = C(x)+l(e)$ donde e es el coste de la arista que va desde x a x'. $C*(x)$ es actualizado con $C(x)$ solo cuando sea necesario, es decir, cuando el nuevo valor mejore el anterior. Una vez termine el algoritmo, el grafo estará completo y la distancia óptima desde el origen será conocida. El coste computacional del algoritmo de Dijkstra es $O(|E|+|V|log|V|)$ pudiendo ser menor dependiendo el caso. Para información más detallada sobre su eficiencia consultar [25].

2.5.4. A*

El algoritmo de búsqueda A* o A-estrella es una extensión del algoritmo de Dijkstra que pretende reducir el número total de estados que explora incorporando una estimación heurística de la ejecución del objetivo dado un estado. Así, utiliza una función de evaluación $f(n) = g(n) + h'(n)$, donde $h'(n)$ representa el valor heurístico para ir desde el nodo n hasta el final, y $g(n)$ el coste real para llegar a dicho nodo desde el inicio. El algoritmo es una combinación entre búsquedas del tipo primero en anchura con primero en profundidad: mientras que $h'(n)$ tiende a primero en profundidad, $g(n)$ tiende a primero en anchura. De este modo, se cambia de camino de búsqueda cada vez que existen nodos más prometedores, para lo que utiliza una cola de prioridad ordenada en base al valor de la función $f(n)$.

A* es un algoritmo completo ya que en caso de existir una solución, siempre dará con ella. Y el tiempo que tarde para conseguirlo dependerá de la calidad de la estimación heurística. En el caso peor, la complejidad será exponencial mientras que en el caso mejor el objetivo se puede alcanzar en tiempo lineal. Para garantizar la optimización del algoritmo, la función $h'(n)$ debe ser heurística admisible, esto es, que no sobrestime el coste real de alcanzar el nodo objetivo. Además, si para todo nodo n del grafo se cumple $g(n) = 0$, nos encontramos ante una búsqueda voraz. O, si por el contrario, $h'(n)$ es quien toma ese valor, A* pasa a ser una búsqueda de coste uniforme no informada.

Capítulo 3

Herramientas utilizadas

3.1. Introducción

En este apartado veremos algunos de los componentes hardware que incorpora el robot, su utilización y posibles alternativas. En la figura 3.1 se muestran los sensores y actuadores principales. Existen otros componentes que no se mencionan en este trabajo, no por carecer de importancia sino porque su uso es condición obligatoria para el correcto funcionamiento y coordinación de todo el conjunto. De modo que no es necesario resaltarlos ya que su empleo viene dado implícitamente en las siguientes secciones.

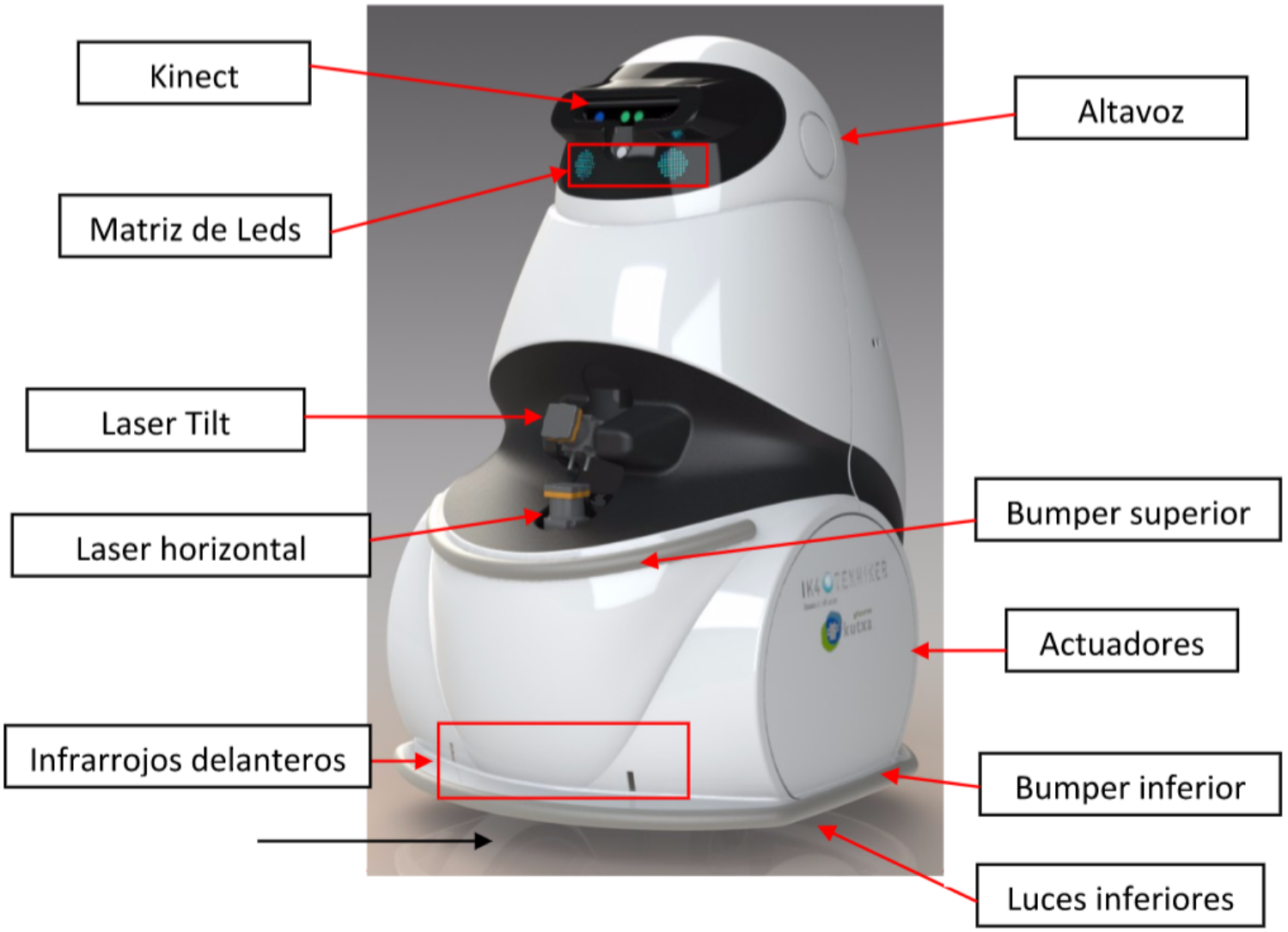

Figura 3.1: Vista delantera del robot y sus elementos.

3.2. Segway

Para poder desplazarse, el robot se encuentra sobre una plataforma Segway, empresa estadounidense fundada en 1999 por Dean Kamen. El modelo que nosotros utilizamos es el Segway Robotic Movility Platform^{TM} 200, que es una plataforma robótica móvil basada en el autobalanceado Segway Human Transporter (HT).

Segway RMP es más rápido, más barato, y más ágil que otras plataformas comparables. También es robusta, su radio de giro es nulo y puede soportar grandes pesos [26]. En lo que a robótica se refiere, tiene un movimiento no holonómico, esto es, no es omnidireccional, ya que la cantidad de grados de libertad controlables es menor que la cantidad de grados de libertad totales. En la figura 3.2 podemos ver algunas de sus especificaciones. Existen nuevas versiones similares a ésta aunque se puede utilizar cualquier otra plataforma móvil, siempre y cuando sea compatible con los requisitos del proyecto.

RMP 200/ATV

TOP SPEED	10 mph (16 km/h) 14.7 ft/s (4.4 m/s)
RANGE	12-15 mi (19-24 km)
TRACK WIDTH	21 in (53 cm)
TIRE DIAMETER	19 in (48 cm)
FOOTPRINT	25 x 24 in (64 x 61 cm) ATV: 30 x 24 in (76 x 61 cm)
TURNING RADIUS	zero degrees
TURNING ENVELOPE	30 in (76 cm)
TOTAL WEIGHT	140 lbs (64 kg) ATV: 157 lbs (71 kg)
DRIVE BATTERIES	Two Li-Ion Packs
MAX. RUN TIME	~8 hrs
MAX. PAYLOAD	200 lbs (91 kg)
RECHARGE TIME	~8 hrs (from empty)
CONTROL MODE	2WD Balance Mode or Tractor Mode
DATA UPDATE RATE	100 Hz
OPERATING TEMP. RANGE	14°F to 122°F (-10°C to 50°C)

Figura 3.2: Especificaciones de Segway RMP 200.

3.3. Bumper

El robot está dotado de unos parachoques o *bumpers* que detectan cuándo éste colisiona con algún objeto. Dispone de dos en la parte delantera (a baja y media altura) y uno en la trasera, aunque con un buen sistema de control de obstáculos ninguno de ellos debería activarse salvo en situaciones excepcionales.

3.4. Hokuyo UTM-30LX

Como se puede ver en la figura 3.1 el robot tiene dos láseres en la parte delantera, los cuales permiten medir distancias. Su funcionamiento consiste en lanzar un abanico de rayos en un plano controlando sus rebotes y estimando la distancia en función al tiempo transcurrido, con la técnica llamada *time-of-light*. Uno de ellos está unido a un servomotor que le permite inclinarse realizando un barrido vertical, posibilitando obtener mucha más información e incluso crear una nube de puntos. La figura 3.4 muestra las especificaciones de este láser (figura 3.3), pero si se requiere una mayor precisión o rango de acción podemos optar por utilizar un Velodyne LiDAR, por ejemplo.

Figura 3.3: Sensor láser Hokuyo UTM-30LX.

Product Name	Scanning Laser Range Finder	
Model	**UTM-30LX**	**UTM-30LN**
Light Source	Laser Semiconductor λ = ~~870nm~~ , 905nm ③ Laser Class 1	
Supply Voltage	12VDC ± 10%	
Supply Current	Max: 1A, Normal : 0.7A	
Power Consumption	Less than 8W	
Detection Range and Detection Object	Guaranteed Range: 0.1 ~ 30m (White Kent Sheet) Maximum Range : 0.1 ~ 60m Minimum detectable width at 10m : 130mm (Vary with distance)	
Accuracy	Under 3000lx : White Kent Sheet: ±30mm*1 (0.1m to 10m) Under 100000lx : White Kent Sheet: ±50mm*1 (0.1m to 10m)	
Measurement Resolution and Repeated Accuracy	1mm 0.1 – 10m : σ <10mm, 10 – 30m : σ <30mm (White Kent Sheet) Under 3000lx : σ < 10mm*1 (White Kent Sheet up to 10m) ⑧ Under 100000lx : σ < 30mm*1 (White Kent Sheet up to 10m) ⑧	
Scan Angle	270°	
Angular Resolution	0.25° (360°/1440)	
Scan Speed	25ms (Motor speed : 2400rpm)	
Interface	USB Ver2.0 Full Speed (12Mbps)	
Output	Synchronous Output 1- Point	Detection Output 1- Point ⑤
LED Display ⑦	Green : Power supply Red : Normal Operation (Continuous), Malfunction (Blink)	Power supply Object detection inside area (Continuous) Malfunction (Blink)
Ambient Condition (Temperature, Humidity)	-10°C ~ +50°C Less than 85%RH (Without Dew, Frost)	
Storage Temperature	-25~75°C	
Environmental Effect	Measured distance will be shorter than the actual distance under rain, snow and direct sunlight*2.	
Vibration Resistance	10 ~ 55Hz Double amplitude 1.5mm in each X, Y, Z axis for 2hrs. 55 ~ 200Hz 98m/s^2 sweep of 2min in each X, Y, Z axis for 1hrs.	
Impact Resistance	196m/s^2 In each X, Y, Z axis 10 times.	
Protective Structure	Optics: IP64	
Insulation Resistance	10MΩ DC500V Megger	
Weight	210g (Without cable)	
Case	Polycarbonate	
External Dimension (W×D×H)	60mm×60mm×87mm ⑥ MC-40-3127	

Figura 3.4: Especificaciones del telémetro láser Hokuyo UTM-30LX.

3.5. Kinect

En la parte superior del robot hay una cámara Kinect versión 1 con una cierta inclinación hacia arriba para abarcar una mayor parte de nuestra región de interés. La Kinect de Microsoft se compone de una barra con varios sensores unida a una base motorizada (figura 3.5). Suele ser clasificada como una cámara RGB-D porque proporciona una imagen en color con el sensor de color CMOS 640x480 (640x480@30fps) y otra en profundidad con el sensor infrarrojo (IR) CMOS 1280x1024 (640x480@30fps). Su campo de visión es de 58º en horizontal, 45º en vertical y 70º en diagonal, siendo el rango de operación del sensor de profundidad de entre 0.8 y 3.5 metros, con una resolución de 1 centímetro. Además, para mejorar la calidad de esta última dispone de un proyector infrarrojo en la misma barra. También lleva incorporados cuatro micrófonos para la obtención de información de audio. En cuanto al control de inclinación, la base está motorizada y se apoya en un acelerómetro para su sensorización, y posiblemente para la estabilización de la imagen.

Figura 3.5: Kinect v1 de Microsoft.

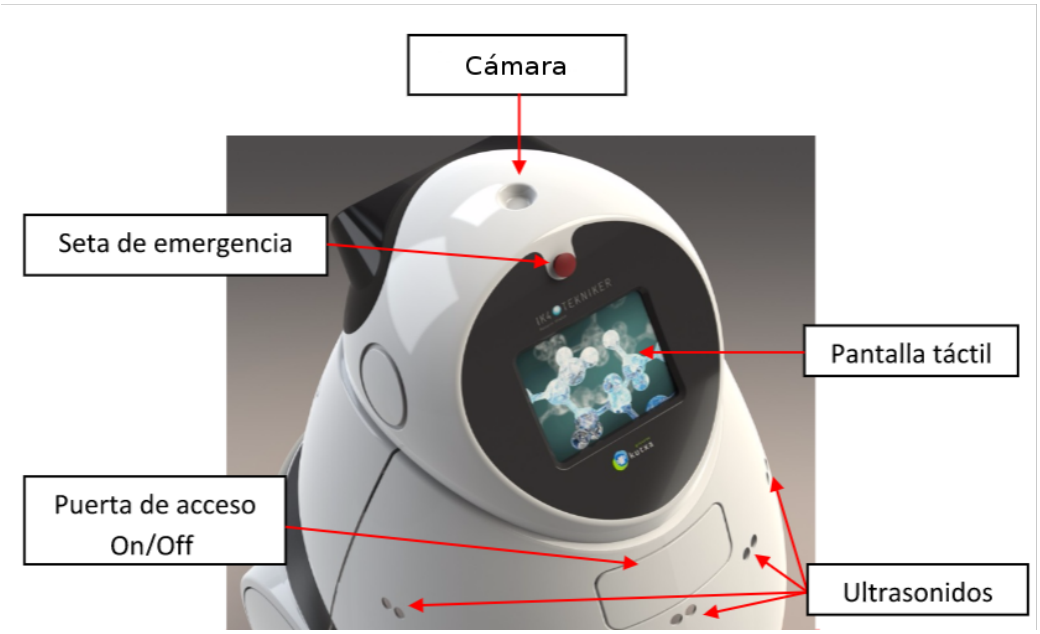

Figura 3.6: Vista trasera del robot y sus elementos.

3.6. Ultrasonidos

Hasta ahora hemos visto que tanto los láseres como la cámara Kinect están enfocados hacia adelante ya que, por lo general, el robot se desplaza en ese dirección. Sin embargo, hay algunas ocasiones en las que Teknibot debe realizar pequeños movimientos hacia atrás y, lógicamente, se debe tener cierta sensorización al respecto. Es por esto que el robot dispone de siete telémetros ultrasónicos en la parte trasera formando una media luna horizontal a 60 centímetros de la base aproximadamente (3.6). El modelo utilizado es el SRF08 con un rango máximo de 11 metros y un tiempo ciclo de hasta 65 milisegundos, aunque las características no son tan relevantes ya que la mayoría de sensores de este tipo cumplen los requisitos de este proyecto. Éstos están conectados en serie con una comunicación vía el bus I2C, aunque lo ideal es una conexión en paralelo para aumentar la eficiencia.

3.7. Infrarrojos

A ambos lados de la base existen cuatro sensores infrarrojos (figura 3.1), dos en la parte delantera y dos en la trasera, para la detección de obstáculos cercanos a mínima altura ya que sus mediciones en distancia tienen un rango

entre 10 y 80 centímetros. El modelo utilizado es el GY2Y0A21YK ó 2Y0A21 de SHARP, compuesto por un detector de posición sensitivo, un diodo emisor de infrarrojos y circuito para el procesado de señal.

3.8. Cámara

En la parte superior de la cabeza del robot está colocada una cámara vertical IDS UI-5240CP-NIR-GL (figura 3.7) que facilita una imagen monocroma de 1.32 megapíxeles a 60 fps mediante un sensor CMOS (figura 3.8).

Figura 3.7: Cámara industrial IDS UI-5240CP-NIR.

El motivo por el cual la cámara está orientada hacia arriba es contribuir al proceso de localización mediante la obtención de características situadas en el techo. Hay que recordar que el robot está diseñado principalmente para entornos interiores ya que en exteriores esta técnica no es de gran ayuda. Sin embargo, nos encontramos en un contexto industrial de modo que la distancia de observación de dichas referencias puede ser considerable y, en consecuencia, la imagen que se obtiene debe ser lo mejor posible. Además, teniendo en cuenta las personas que se hallan continuamente en el área de trabajo del robot puede haber muchos curiosos que se acerquen al mismo añadiendo grandes cantidades de ruido e incluso llegando a colapsar los sensores. La cúspide de la cabeza es, a priori, el punto con mayor libertad en estas circunstancias y el cual puede evitar un error catastrófico en el posicionamiento.

3.9. Servomotor

Como hemos mencionado en la sección del telémetro láser Hokuyo, uno de ellos está acoplado a un servomotor que le permite cambiar su orientación sobre el eje horizontal y obtener así información en tres dimensiones. El motor

Tipo de sensor	CMOS Mono
Sistema de obturador	Global Shutter / Rolling Shutter / Global Start Shutter
Characteristic	Lineal
Método de lectura del sensor	Progressive scan
Clase de píxeles	SXGA
Resolución	1,31 Mpx
Resolución (h x v)	1280 x 1024 Pixel
Relación de aspecto	5:4
CAD	10 bit
Profundidad de color (camára)	12 bit
Clase de sensor óptico	1/1,8"
Superficie optica	6,784 mm x 5,427 mm
Diagonal del sensor óptico	8,69 mm (1/1,84")
Tamaño de píxel	5,3 µm

Figura 3.8: Especificaciones del sensor de la cámara IDS UI-5240CP-NIR-GL.

utilizado es el RX-28 HN07-N101 de Dynamixel con un par motor de 2.5 N.m a 14.8 V y un mínimo ángulo de control de aproximadamente 0.29º x 1024, controlando la posición a través de un potenciómetro.

3.10. Matriz de LEDs + Arduino

Con la finalidad de dar cierta sensación de 'vida' al robot, en la parte delantera de su cabeza se han añadido dos matrices de LEDs 8x8 con 6 posibles colores diferentes (figura 3.1), controladas mediante un microcontrolador Arduino.

3.11. Pantalla táctil

En cuanto a la interacción, Teknibot dispone de una pantalla táctil en la parte superior trasera (figura 3.6) que facilitará la comunicación, ya que mediante ella, principalmente, se mostrarán y enviarán las acciones disponibles en cada momento. El modelo en concreto es la pantalla LCD de 8.4.ªPR-A0840-ORBD de la empresa Alphadisplay. Además, es de gran ayuda en la etapa de desarrollo ya que nos permite el control total de la máquina. De lo contrario sería necesario conectar una pantalla externa o cierto método de control remoto.

3.12. Altavoces

A ambos laterales de la cabeza se encuentran las aberturas para que salga el sonido del altavoz/altavoces, que permitirán ofrecer información sonora en los diversos estados del robot. Es por esta vía, por ejemplo, por donde se anunciarán las charlas informativas y curiosidades de los lugares que se visiten.

3.13. Baterías

Para la alimentación de todos estos componentes el robot consta de cuatro baterías independientes, dos de ellas para la tracción y otras dos para los demás elementos. Todas son del tipo Li-Ion (ion de litio) las cuales ofrecen una gran relación energía almacenada/peso. Ofrecen 72 voltios de corriente continua, y disponen de un conversor que les permite suministrar 5, 12 o 24 V para el hardware que así lo requiera. En la parte inferior trasera del robot hay una puerta para recargar las baterías mediante un cargador externo. Tanto el tiempo de carga total como el de descarga es de 8 horas aproximadamente, aunque hay que tener en cuenta su deterioro y pérdida de capacidad con el paso de los años.

Figura 3.9: Joystick Logitech Extreme 3D Pro.

3.14. PC

Por último, lógicamente es necesario un ordenador o 'cerebro' que se encargue de gestionar todos estos componentes, haciendo un uso eficiente de los mismos mediante el software que desarrollemos. Actualmente dicha tarea le corresponde a un Dell con 3.8 GiB de memoria RAM y el procesador Intel Core i7-2600 de 8 núcleos trabajando a una frecuencia de 3.40 GHz.

3.15. Joystick

En las primeras etapas del desarrollo del robot es imprescindible tener una herramienta para controlar su movimiento ya que agiliza mucho ciertas tareas, como por ejemplo el mapping. En este proyecto se utiliza el joystick Extreme 3D Pro de Logitech (figura 3.9), conectado mediante USB. Pese a que ofrece una amplia variedad de acciones, muchas de ellas no se tienen en cuenta ya que su utilización es exclusivamente para mover las ruedas.

Capítulo 4

Desarrollo

4.1. Introducción

Pese a que seleccionar los elementos necesarios para cumplir con los objetivos del proyecto y que éstos puedan convivir correctamente ya es un gran logro, es necesario un software que los gestione y dote al conjunto de cierto nivel de inteligencia. En este capítulo veremos el desarrollo que hace que eso sea posible, donde se encuentra el núcleo el desarrollo del proyecto en sí. Está dividido en tres partes: rehabilitación del robot, utilidades o funcionalidades y fase experimental.

4.2. Rehabilitación

Se denomina rehabilitación del robot al proceso de restitución o restauración del mismo de manera que vuelva al estado original. Pero en este caso no es solo eso puesto que se sustituyen los componentes obsoletos y/o defectuosos, se actualiza el software utilizado a su última versión y, además, se incorporan nuevas mejoras. Cabe mencionar que todos los archivos son modificados utilizando la convención de programación de C++ y traduciendo todos los nombres y comentarios al inglés con la finalidad de facilitar su lectura y comprensión. También se añade la funcionalidad de informes de diagnóstico de ROS donde se cree conveniente, para llevar un mejor control de fallos en tiempo de ejecución.

4.2.1. Ubuntu

Ubuntu es el sistema operativo, el software principal en el que se apoyarán todos los programas y con el que se conectarán todos los componentes electrónicos.

Objetivo

La versión utilizada originalmente es la 10.04 LTS *Lucid Lynx* (04/2010) y se quiere instalar Ubuntu 16.04 LTS *Xenial Xerus* (04/2016) por ser la última versión disponible con soporte a largo plazo.

Progreso

Ha sido actualizado a la última versión sin ningún inconveniente mediante una memoria USB.

4.2.2. ROS

ROS es el framework para el desarrollo de software que provee las funcionalidades de un sistema operativo y mediante el cual se gestiona todo, o casi todo, el funcionamiento del robot.

Objetivo

La versión utilizada originalmente es Diamondback (03/2011) y se quiere instalar ROS Kinetic Kame (05/2016) por ser la última versión disponible, aunque recientemente es el lanzamiento de la versión Lunar Loggerhead (05/2017).

Progreso

La instalación y configuración del entorno para el correcto funcionamiento de ROS Kinetic no supone ningún problema. Sin embargo, esta acción tiene consecuencias más adelante a la hora de compilar y ejecutar los paquetes necesarios ya que su estructura ha cambiado. Por ejemplo, el archivo *manifest.xml* ya no existe y en su lugar se utiliza *package.xml*, el formato del *CMakeLists.txt* es diferente, la forma de generar mensajes de ROS varía, etc.

4.2.3. Pantalla táctil

La pantalla táctil es el único modo de comunicación por defecto, de modo que lo primero es hacer que ésta lea correctamente nuestros comandos.

Objetivo

El driver a poner en marcha es EETI eGalax eGTouch v2.5.5814, que sirve tanto para máquinas de 32 como de 64 bits, y después utilizarlo para calibración táctil.

Progreso

Antes de ejecutar el script de instalación, es necesario conectar el controlador. Después no hay más que acceder a la carpeta en la que se encuentra el archivo comprimido y ejecutar desde la terminal:

```
$ tar -xzvf eGTouch_v2.5.5814.L-x.tar.gz
$ sudo eGTouch_v2.5.5814.L-x/setup.sh
```

Una vez finalizado ejecutamos

```
$ eCalib
```

y seguimos las indicaciones para la calibración. Al completar el proceso la pantalla táctil funciona eficazmente.

4.2.4. Gestor de las baterías

Las baterías no pueden bajar de un cierto umbral de tensión ya que podrían estropearse, de modo que es necesario llevar un control continuo de las mismas.

Objetivo

Instalar un daemon o demonio que se ejecute en todo momento en el sistema operativo de manera que apague el robot cuando alguna de las baterías baje de 69 V.

Progreso

En primer lugar se instalan los drivers de Linux de Measurement Computing y se configura el archivo 61-mcc.rules. Si las reglas no pueden cargarse de manera automática se puede abrir la terminal y ejecutar:

```
$ sudo udevadm control --reload-rules
```

Podemos forzar manualmente a udev a activar sus reglas con:

```
$ sudo udevadm trigger
```

Después viene el turno del controlador de la tarjeta MC USB-1208FS, que es la que se comunica directamente con las baterías recibiendo y/o enviando información. Éste antes utilizaba la librería *asio 1.10.2*, pero ahora se encuentra dentro de *boost* luego hay que actualizar su uso en el código y modificar las dependencias correspondientes.

En segundo lugar, se crean los programas de se quieren ejecutar continuamente en el sistema operativo en segundo plano para poder llamarlos desde los ficheros *.service* se se iniciarán al encender el ordenador. La forma de hacerlo cambió con Ubuntu 15.04, antes se utilizaba el sistema *upstart* y ahora es *systemd* el encargado de dicha tarea. De modo que se crean los dos archivos para luego añadirlos en la carpeta */etc/systemd/system*:

- **mc1208fs.service**: se encarga de comunicarse con las baterías y ofrecer diversas funcionalidades para su gestión.

- **check_battery.service**: hará uso del demonio anterior para consultar el nivel de carga de las baterías y apagar el ordenador en caso de que éste baje de 69 V.

La información recopilada por ambos es escrita en el archivo */var/log/syslog* y se puede ver en vivo el nivel de estado de las baterías mediante el siguiente comando:

```
$ tail -f /var/log/syslog | grep 'check_battery'
```

Lo ideal es poder representar dicha información gráficamente en la barra de estado de la pantalla, como en el caso de teléfonos y tabletas.

Por defecto al inicializar *mc1208fs* con *systemd* hay un problema en el que ocasionalmente dicha tarea no se lleva a cabo correctamente. Para evitarlo se ha creado un archivo PID para decirle a *systemd* que ese proceso no se termina tras hacer un fork.

4.2.5. Drivers

El siguiente proceso que se debe llevar a cabo es la instalación y configuración de los drivers de todos las herramientas hardware que se utilizan.

Objetivo

De forma que se pueda obtener información de los sensores y enviar acciones a los actuadores, es necesario poner en funcionamiento los siguientes paquetes:

- **dynamixel_motor**: conjunto de paquetes para la interacción de la línea de servomotores Robotis Dynamixel.
- **infrared**: hacer funcionar los infrarrojos.
- **joy**: obtiene información del joystick.
- **laser_snapshotter**: en base al ángulo y velocidad de movimiento del servomotor, se comunica con el Hokuyo unido a éste para generar una nube de puntos (el tipo de ROS *sensor_msgs::PointCloud2*).
- **libsegwayrmp**: librería que facilita todas las funcionalidades para el manejo y control de la plataforma Segway.
- **mc1208fs**: control de la tarjeta MC USB-1208FS.

- **rainbowduino_control**: paquete para la modificación de las matrices de LEDs.
- **robotis_controller**: hace uso de *dynamixel_motor* para controlar el servomotor de la forma deseada.
- **segway_rmp**: hace uso de *libsegwayrmp* para ofrecer las funcionalidades adaptadas al caso de uso de este proyecto.
- **serial**: es una librería multiplataforma para la interacción con los puertos tipo RS-232.
- **teknigazer**: se conecta con la cámara vertical para la identificación de marcadores y la publicación de dicha información.
- **teleop**: hace uso de *joy* para gestionar los comandos del joystick y actuar en consecuencia. Hace posible la teleoperación del robot. Quizá esto represente una funcionalidad, pero se ha decidido incorporarlo a esta sección porque no es común en el uso habitual del robot más que para ciertos movimientos puntuales de mantenimiento.
- **urg_node**: paquete para la comunicación de los telémetros Hokuyo mediante su librería *urg_c*.
- **usbi2c**: utiliza un módulo de comunicación de USB a I2C, unido a los sensores de ultrasonidos SRF08. Permite la usuario transmitir el rango de ultrasonidos mediante un *publisher* de ROS.

Progreso

Debido a que los dispositivos pueden verse reflejados en el ordenador con diferentes identificadores y no siempre los mismos, es recomendable definir unas reglas udev. Udev es el gestor de dispositivos que usa el kernel de Linux y su función es controlar los archivos de dispositivo en */dev*, manejando todas las acciones del espacio de usuario al agregar o quitar dispositivos. Para ello se debe crear un archivo en la carpeta */etc/udev/rules.d* con el formato adecuado. Por ejemplo, para el caso de los láseres Hokuyo creamos el fichero *10-sensors.rules* y añadimos lo siguiente:

SUBSYSTEMS==üsb", KERNEL=="ttyACM[0-9]*", ATTRS{manufacturer}=="Hokuyo Data Flex for USB", ATTRS{product}==ÜRG-Series USB Driver", PROGRAM=/opt/ros/kinetic/lib/urg_node/getID /dev/ttyACM%n q", SYMLINK+="sensors/hokuyo_%c", MODE="0666"

Lo que hará que cuando se conecte un dispositivo por vía usb con un nombre asociado *ttyACM[0-9]** con los datos de fabricante y producto correspondientes, se ejecute el programa *getID* con el nombre asociado como parámetro y la respuesta de éste se utilizará para crear un link simbólico en la ruta */dev/sensors*. Además, tendrá todos los permisos para todos los usuarios. De esta forma se puede contactar con un Hokuyo específico utilizando su identificador.

Pese a que el paquete *urg_node* se puede instalar mediante su binario (.deb), de momento éste no incluye el ejecutable *getID*. Es por esta razón por la que se ha descargado el código fuente, para compilarlo y copiar después el ejecutable creado en la ruta de instalación del paquete en /opt. Sin embargo, hay que tener cuidado con esto ya que al limpiar el workspace en el que se compila, por alguna razón que no llegamos a comprender el ejecutable deja de funcionar en modo *root*, que es como se ejecuta desde la regla udev, alegando que falta cierta librería dinámica. Esto hace que el láser Hokuyo no se identifique con el nombre esperado y, en consecuencia, nuestro programa no pueda comunicarse con él. La solución más sencilla para evitar este problema es volver a compilar el paquete y copiar el nuevo ejecutable en la ruta deseada.

Es de añadir que para el correcto funcionamiento del paquete *segway_rmp* no se pueden cargar los módulos kernel *ftdi_sio* y *usbserial*. De ser así tendremos que ejecutar el comando *rmmod* dándole como parámetro cada uno de ellos. Sin embargo, los demás sensores USB necesitan el módulo *ftdi_sio* para ser representados en */dev/ttyUSB** con lo que existe cierta incompatibilidad. La manera de evitar este conflicto es configurar el paquete *segway_rmp* para utilizar el puerto serie en vez de FTDI2XX, es por eso que se utiliza la librería *libsegwayrmp* y no *libftd2xx*. Es importante conocer las dependencias de los diferentes paquetes y compilarlos en el orden adecuado. En el caso de *libsegwayrmp* tenemos que tener cuidado si lo metemos dentro del workspace puesto que al no ser un paquete de ROS, es necesario incluir un CATKIN_IGNORE en su raíz para evitar futuros errores de compilación.

4.2.6. Relación espacial de los elementos

La información dada por los sensores es tratada mediante el método correspondiente en base a su procedencia y utilidad, para luego comunicarse con los demás elementos y comenzar una cadena de toma de decisiones. Para que esa información esté debidamente representada en un marco global, es

necesario disponer de un punto de referencia común y relacionar toda esa información espacial respecto a éste. Es por esta razón por la que el sistema debe conocer la transformación, i.e. traslación y rotación, entre el origen de coordenadas de cada uno de sus sensores y/o actuadores.

Objetivo

Publicar un árbol de sistemas de referencia que represente el robot y sus elementos, de forma que la combinación de información espacial sea eficaz y se asemeje a la realidad.

Progreso

Se han medido físicamente las distancias entre los diferentes elementos para luego poder publicarlas mediante un *tf_broadcaster*. Este paquete crea y publica un árbol de transformaciones en base a las mediciones obtenidas en un fichero XML, que luego cualquier nodo puede consultar. En este caso en particular, la raíz del árbol de transformaciones es la base del robot, que se encuentra en el punto central de la parte inferior de la plataforma (figura 4.1). En la tarea para la que está pensada el robot se puede admitir un error en el posicionamiento de hasta 20 cm aproximadamente, es por ello que no se ha realizado un postprocesado de los datos o una comprobación de la calidad de los mismos. Sin embargo, para obtener la mayor precisión posible es recomendable hacer una calibración extrínseca de todos los sensores, tarea que se pretende realizar en un futuro cercano.

4.2.7. Mapeo

Una vez que el robot tiene conocimiento y control de él mismo, es el momento de añadir información sobre el mundo. Recordemos que es primordial que Teknibot sepa en todo momento su posicionamiento, ya que de ello dependen muchas de las demás tareas que realiza. Por lo tanto la representación del entorno se debe ajustar en la mayor medida posible a la realidad, o, mejor dicho, a cómo interpretan los sensores la realidad.

Objetivo

Realizar un mapeo de la zona de trabajo legible por el robot.

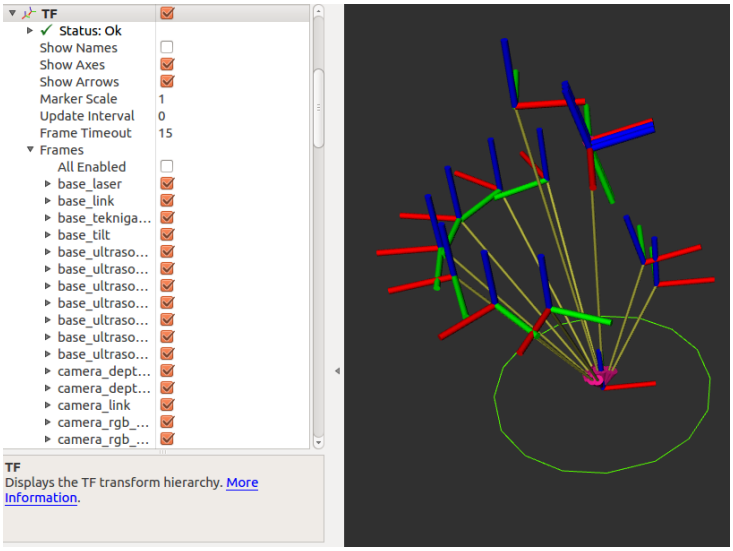

Figura 4.1: Relación espacial (tf) de los elementos del robot.

Progreso

Hemos optado por hacer uso del nodo disponible en los repositorios de ROS *gmapping*, ya que implementa un algoritmo de filtro de partículas con buenos resultados [27] y es ampliamente parametrizable. Como datos de entrada se utiliza el láser Hokuyo estático, el inferior, que se encuentra a 50 cm del suelo. Para este proceso se mueve el robot mediante el joystick lentamente y sin giros bruscos, haciendo un recorrido cerrado del área de interés y repitiendo parte de la etapa inicial, con la finalidad de facilitar el trabajo de cerrar el bucle o *loop closure* y mejorar así la calidad del mapa.

4.2.8. Navegación

El mapa del mundo ya es disponible, de modo que ahora hay que saber moverse por él. Para la localización se utiliza el algoritmo AMCL [19], que ya dispone de una implementación con su mismo nombre en los repositorios de ROS. En este apartado también se incluye la evitación de obstáculos.

Objetivo

Una vez que el robot esté localizado, poder mandarle a un punto en el mapa al que el robot vaya si es posible realizando un recorrido lógico y sin colisionar con los obstáculos que se encuentre en el camino.

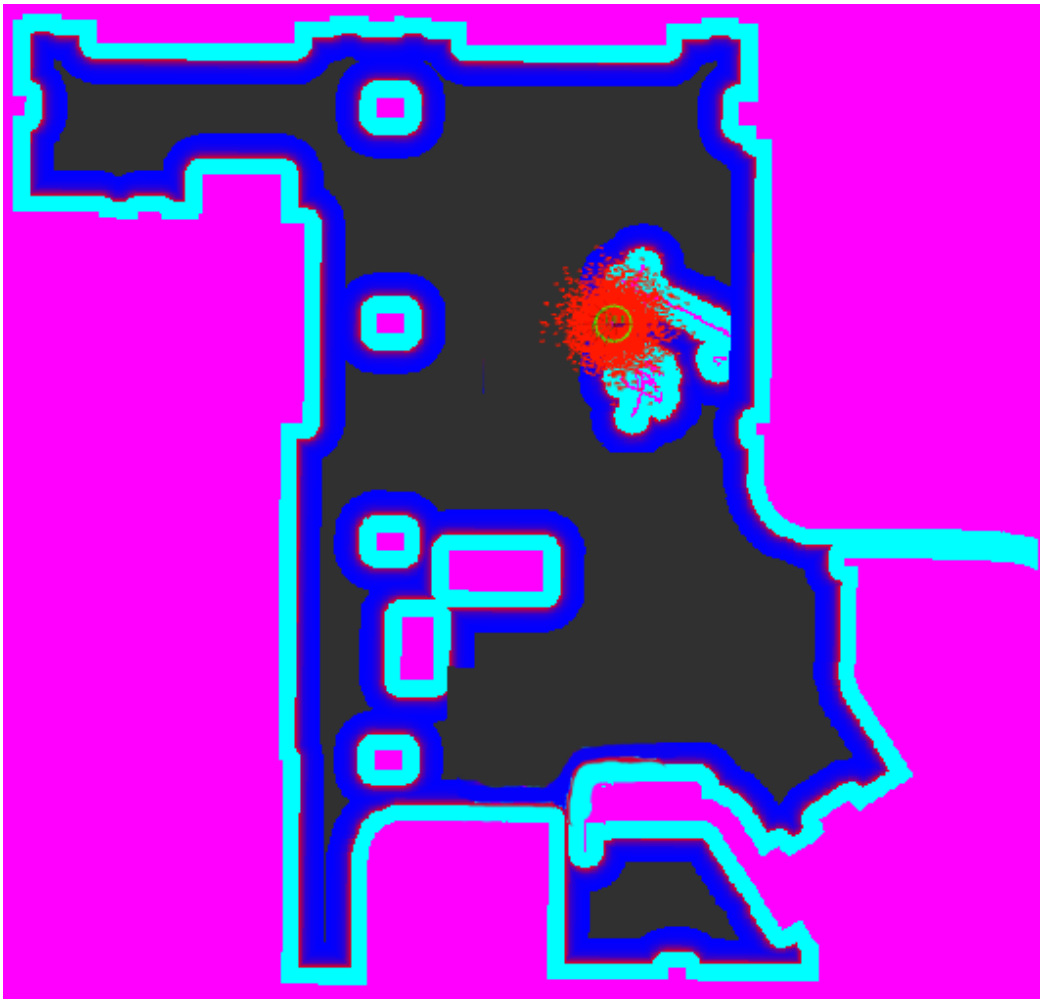

Figura 4.2: Vista en Rviz del mapa de la zona. El color rosa representa el obstáculo y los azules su inflación. El círculo verde es la base del robot. Los puntos rojos son la dispersión de las partículas del algoritmo AMCL.

Progreso

En primer lugar, la localización aproximada del robot viene dada ya que el punto de partida es conocido o se le asigna mediante comando. Después se le envía una ubicación objetivo con respecto al origen de coordenadas del mapa creado anteriormente. En el transcurso del movimiento el robot hace uso de la odometría y del mismo láser con el que ha creado el mapa para mantener su localización. En lo que a los obstáculos se refiere, estos son detectados gracias a los sensores infrarrojos, ultrasonidos y el Hokuyo unido al servomotor. Tanto los obstáculos estáticos, esto es, los que están representados en el

mapa y no varían, como los dinámicos son representados en un *costmap*. En él se les aplica una inflación igual al radio útil del robot para poder representar éste mediante un punto y reducir el coste computacional del cálculo de trayectorias. Además, cuando el robot calcula que un punto es alcanzable pero algún punto del recorrido no puede avanzar, entra en un modo de recuperación durante un cierto tiempo e intentar explorar otras opciones o esperar que la calculada previamente está disponible de nuevo.

4.2.9. Interfaz de usuario

La última parte correspondiente a la migración es la interfaz de usuario y los modos de operación disponibles en el robot correspondientes a la comunicación persona-robot.

Objetivo

Poner en marcha la interfaz gráfica que se muestra en la pantalla táctil que controla la gestión de los diferentes estados del robot.

Progreso

El problema principal que hay a la hora de compilar el paquete *tekni-bot_manager* correspondiente a la interfaz gráfica es que la versión mínima que utiliza ROS Kinetic es Qt 5.3 y la implementación está para Qt 4. Se quiere utilizar la última versión disponible de todo el software que se utilice, ya que en principio su funcionamiento será óptimo, de modo que hay que modificar el paquete para utilizar Qt5, adaptando la estructura del código, las dependencias y las instrucciones dadas a CMake para la compilación. Una vez hecho esto, existen nuevas incompatibilidades con librerías como OpenCV que hay que corregir y reajustar el proyecto para que cada paquete funcione correctamente y puedan convivir conjuntamente.

4.3. Mejoras

Esta sección corresponde a las mejoras que se han intentado incorporar al sistema, siendo algunas efectivas y otras no. Este puede que sea el apartado más relevante para los lectores con cierto espíritu investigador, ya que el estudio de la literatura y el nivel de conocimiento para decidir las innovaciones a incorporar son más exigentes. A pesar de todo, muchas de las ideas quedan fuera del alcance del proyecto debido a la falta de tiempo pero se pretende trabajar con ellas y algunas se verán en el apartado de trabajo futuro. Por

Figura 4.3: Ventana principal de la interfaz de usuario donde se escoge el modo de operación.

lo tanto los avances logrados son, de momento, menores pero que aun así mejoran en cierta medida el rendimiento general del robot.

4.3.1. Filtrado del láser

En primer lugar, el láser Hokuyo superior detecta colisiones con la propia carcasa del robot que no interesan. Debido al movimiento del mismo, si el ángulo es mayor que un valor concreto el propio robot llega a entrar en su campo de observación mostrándose como un obstáculo. Esta es una situación que no puede darse bajo ningún concepto, ya que por cómo se ha programado el algoritmo de navegación intenta evitar el obstáculo calculando una nueva ruta pero nunca puede encontrarla al ser el obstáculo permanente.

Objetivo

Es importante que la detección del telémetro sea eficaz y solo muestre los cuerpos externos. Con lo cual es necesario hacer un tratamiento de los

datos y obtener únicamente la información relevante y que permita llegar al destino de forma eficiente.

Progreso

La primera solución que se puede ocurrir, y la que se realizaba en kTBot, es reducir el ángulo de movimiento vertical del abanico láser, no llegando nunca a entrar en contacto con la carcasa. Sin embargo, en los primeros momentos en que el cuerpo del robot entra en el rango del telémetro éste no lo cubre completamente. Por lo tanto reducir su grado de acción puede suponer una pérdida de información importante. Otra opción es hacer un filtrado del láser y descartar información que esté demasiado cerca. Pero simplemente con esto el recorrido que realiza el telémetro puede ser excesivo, pese a que no se detecten obstáculos innecesarios que generen un comportamiento ilógico. Eso conlleva a que el tiempo que se tarda en generar la nube de puntos en cada ciclo del Hokuyo aumente, reduciendo la precisión en la detección de obstáculos. En virtud de ello, la solución óptima es una combinación de ambas. Primero se decidirá en qué límites inferiores y superiores del movimiento de láser comienza la carcasa a abarcar demasiado campo de visión, o lo que es lo mismo, hasta que límites la información obtenida del entorno es suficiente. Una vez calculado ambos puntos, mediante un filtrado del láser se limpian las colisiones correspondientes a la propia carcasa del robot.

4.3.2. Detección de obstáculos

Hasta ahora las herramientas que se ha utilizado para la detección de obstáculos, principalmente, son los dos telémetros Hokuyo. Es cierto que los infrarrojos detectan objetos cercanos a muy baja altura y los ultrasonidos los que se encuentran detrás del robot, pero el comportamiento del robot depende de éstos en casos aislados. Luego nos centraremos en los sensores láser de distancia frontales. El inferior únicamente detecta los objetos en un plano horizontal de 50 cm sobre la base del robot y el superior, al estar en continuo movimiento, abarca obstáculos en un mayor número de alturas pero pro este motivo se ve reducida su frecuencia útil.

Objetivo

Ya que el robot dispone de una cámara Kinect, la idea es añadirla al conjunto de sensores utilizados para la detección de obstáculos.

Progreso

La tarea de usar la nube de puntos generada por la Kinect como vía para detectar obstáculos es sencilla. Sin embargo, hay varios puntos que se deben tener en cuenta. El primero de ellos, que la Kinect está orientada hacia arriba, donde generalmente las distancias a la que se detectan los cuerpos son mayores. Esta inclinación supone un problema a la hora de realizar la limpieza y esta cuestión se verá en el siguiente apartado. Sin embargo, este mismo motivo puede ayudar al robot a evitar ciertas situaciones en las que el recorrido calculado le obligue a pasar debajo de elementos peligrosos. O, en su lugar, zonas por las que los humanos no pueden pasar cómodamente ya que, recordemos, el robot está pensado para hacer de guía y llevar a los clientes por diferentes lugares. Otro punto es que el tratamiento de una nube de puntos demanda un mayor coste computacional que el de un sensor de una o dos dimensiones y, lógicamente, se quiere llevar un control de los obstáculos con una frecuencia similar a la que el sensor es capaz de detectarlos.

4.3.3. Limpieza de obstáculos

Hasta ahora cuando nos hemos referido a obstáculos en todo momento hemos mencionado su detección, pero no debemos olvidar que ésta no puede ser eficaz sin una buena limpieza. La limpieza o, dicho de otra forma, el hecho de afirmar que el obstáculo detectado anteriormente ya no existe, se realiza mediante el trazado de rayos o *raytracing*. En el algoritmo *raytracing* se determinan las superficies próximas en la escena trazando rayos desde el observador (sensor) hasta su rango máximo a través del plano de la imagen. Se calcula la intersección del rayo con el primer objeto y éste determina la distancia de la primera colisión potencial. En cuanto a la limpieza, el rayo debe llegar a un punto más lejano que en la etapa previa, lo que significará que el obstáculo detectado anteriormente no existe y ahora hay uno en el nuevo punto de intersección. En el caso peor, el haz llegara hasta el punto más lejano que puede alcanzar. En nuestro escenario de trabajo estas distancias pueden ser demasiado grandes por la potencia de las herramientas que utilizamos y, en menor medida, por las dimensiones del entorno, lo que conlleva a un mayor coste computacional. Esto implica un mayor retardo en la limpieza de los obstáculos manteniéndose representados en el mapa aun cuando ya no existen.

Objetivo

Debido a que el ordenador lleva muchos procesos a cabo al mismo tiempo, hay que optimizar el coste computacional de todas las tareas que se realizan. De modo que el objetivo es realizar la limpieza de obstáculos con mayor rapidez y sin perder eficacia, claro está.

Progreso

La primera solución llevada a cabo es, como en el caso de los láseres, es realizar un filtrado de la nube de puntos obtenida por la Kinect limitándola en los tres ejes (horizontal, vertical y longitudinal) y dejando de contemplar los cuerpos más allá de esos valores. Este tamiz apenas significa una mejora en los tiempos de ejecución ya que el *raytracing* igualmente tiene que llegar hasta su rango máximo y, de hecho, en algunas casos estos tiempos son peores.

La segunda opción es utilizar los datos de los sensores únicamente para el marcado de obstáculos y simular otra fuente para la limpieza. Asímismo, se realiza un nodo que genere una nube de puntos que represente el espacio de trabajo que se desea contemplar, esto es, representa una especie de habitación cúbica y el marcaje no tiene en cuenta los objetos más allá de esas paredes. En este caso, el proceso de limpieza es mucho más rápido ya que las distancias máximas que alcanza el *raytracing* son relativamente pequeñas. Sin embargo, para abarcar el mayor número de rayos posibles la nube de puntos que hay que publicar es muy densa esto es, tiene un gran número de puntos, lo que implica una pérdida de rendimiento considerablemente mayor a la mejora lograda. Luego no es factible.

El tercer método planteado para dar salida al problema es la limpieza del mismo mapa. Ya se ha mencionado en alguna ocasión que el mundo se representa en un mapa de rejillas, y este a su vez tiene varias capas que denominamos *costmap*. El encargado de representar los obstáculos dinámicos es el *costmap* local, y no es más que otro mapa de rejilla permanente alrededor de la base del robot. Esta vez no se tiene en cuenta la detección de obstáculos en sí y la atención se centra en el estado del mapa. Por esta razón, se ha implementado un programa que acceda a este mapa y vacíe, limpie, desocupe... todas las casillas del mismo con una cierta frecuencia. En nuestro caso, en base a los tiempos necesarios para ejecución de otras tareas del robot, la limpieza se realiza a 2 Hz con lo que no es necesario utilizar los sensores para ello y que su función sea exclusivamente la de marcaje.

4.3.4. 3D Mapping

Como hemos mencionado anteriormente, la reconstrucción del mapa se hace mediante el telémetro Hokuyo inferior consiguiendo un mapa lo suficientemente realista como para poder simular la realidad. Sin embargo, el láser obtiene información únicamente de las partículas que están en su mismo plano horizontal, a 50 cm de la base, y el robot colisiona con cuerpos que se encuentran hasta a 140 cm. El mapa de rejilla generado dispone menos celdas ocupadas de las convenientes ya que no contempla los obstáculos en todo ese rango y, en consecuencia, el número de obstáculos dinámicos a detectar aumenta cuando son conocidos a priori. Esto hace que aumente considerablemente la probabilidad que la primera trayectoria global calculada a la hora de intentar alcanzar un destino varíe.

Objetivo

El objetivo es realizar un mapping de la zona de trabajo teniendo en cuenta todos los elementos que se encuentren entre la base y la parte más alta del robot, i.e. con los que puede colisionar, y generar el mapa de rejilla en base a todos ellos. Esta mejora evitará en muchas ocasiones tener que recalcular la trayectoria reduciendo los recursos computacionales necesarios en pleno recorrido.

Progreso

En base a las herramientas que disponemos, la elección ha sido utilizar la cámara Kinect para la realización de esta tarea.

En principio la idea ha sido construir una malla de ocupación 3D mediante un octomap. Un octomap se basa en un octree y provee estructuras de datos y algoritmos de mapeo particularmente diseñados para el área de la robótica. Aunque para el caso de uso de este proyecto es innecesario mantener tanta cantidad de información debido a que los movimientos del robot son en dos dimensiones y nunca en el eje vertical.

Posteriormente se ha optado por generar una nube de puntos de los objetos entre 0 y 140 cm para luego proyectarlos en el plano horizontal que representa el suelo. No obstante, para ello es necesario un filtro de los datos para quedarnos únicamente con los correspondientes a ese rango pese a que esa información descartada puede resultar útil. El mapa generado no se ve alterado por considerar cuerpos más allá de la altura del robot puesto que nunca serán representados en el mapa en el cual se basarán los algoritmos

de navegación. Aunque pueden ser de gran ayuda a la hora de intentar posicionar el robot, ya que disponemos de una mayor cantidad de información relevante con la que comparar las mediciones de los sensores. Además, por este mismo motivo se utiliza la imagen RGB-D en vez de solamente la nube de puntos, esto es, una imagen que suministra tanto información espacial en 3 dimensiones como de color aprovechando toda la información que facilita la Kinect.

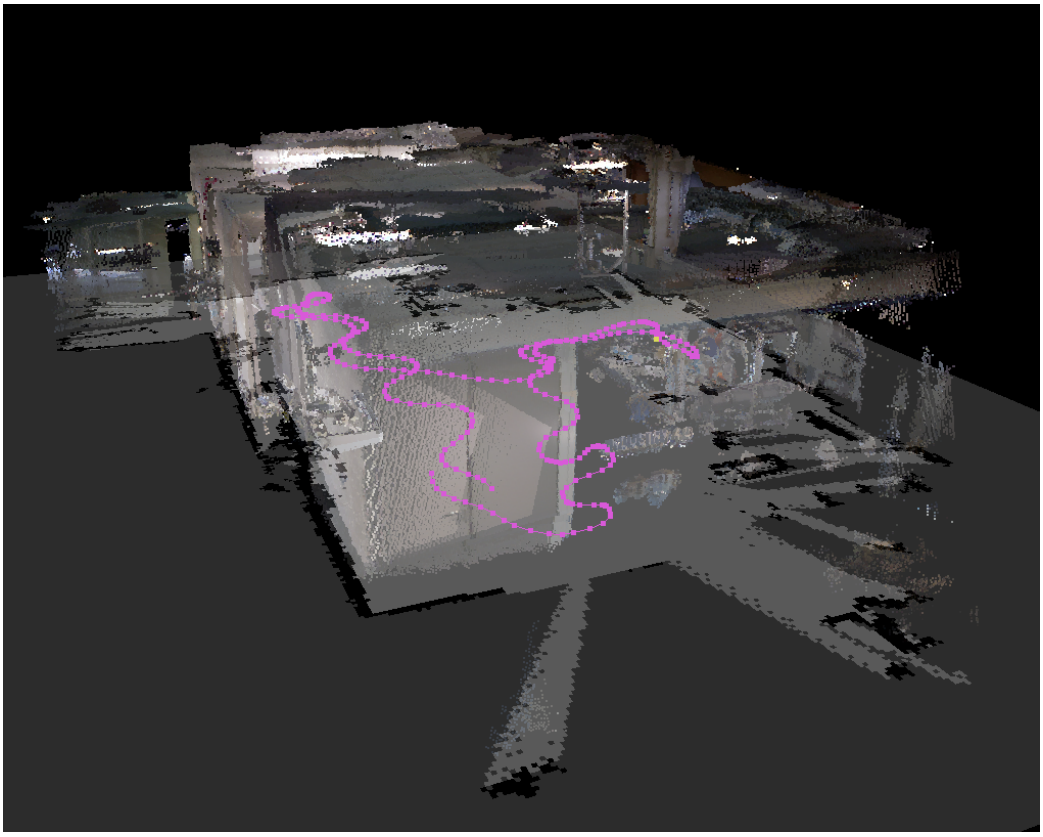

Figura 4.4: Mapping en 3 dimensiones del entrono de trabajo, con la trayectoria realizada para ello en rosa.

4.3.5. Localización

En base al trabajo realizado en la sección anterior, ahora disponemos de un modelo en tres dimensiones de la zona de trabajo lo que aumenta el número de técnicas que podemos emplear para la aproximación de la localización.

Objetivo

Se quiere abordar el problema del posicionamiento desde otras perspectivas, intentando explotar todos los recursos disponibles.

Progreso

En primer lugar y, al igual que en la mejora anterior, se ha intentado hacer el proceso de localización gracias el octomap generado en la fase de mapping. Inicialmente los resultados no son favorables debido a que hay un gran número de parámetros que hay que adecuar a nuestra situación específica y, al optar por otros métodos de representación del mapa debido al coste que este supone, rápidamente se ha descartado la posibilidad de localizarse mediante un octomap.

En segundo lugar, se ha decidido probar la técnica llamada *image registration* o registro de la imagen adaptada a una nube de puntos. Este método se ha implementado utilizando las librerías OpenCV y PCL (*Point Cloud Library*) pero los resultados han sido suficientemente precisos en un demasiado bajo 20 % de los casos, lo que ha hecho que se descarte esta técnica como modo de localización.

A continuación, con la ayuda de las dos librerías mencionadas anteriormente, se ha implementado la técnica de *surface matching*. Teniendo en cuenta que la escena se ha generado en el proceso de mapeo, este método consiste en varias fases que son:

- **Generación del modelo 3D**: esta fase corresponde a la generación del modelo en tres dimensiones que representa el objeto. En este caso en vez de ser el modelo de un objeto en particular es el entorno de trabajo completo.
- **Extracción de puntos característicos**: se extraen los puntos característicos del los cuerpos en el campo de visión, también llamados PPF (*Point Pair Features*).
- **Entrenamiento del modelo**: se entrena el modelo 3D obtenido para luego poder comprarlo con la escena.
- **Estimación de la pose**: mediante los puntos característicos obtenidos tanto del modelo como de la escena, se hace un clustering dónde se obtiene una aproximación de la pose en la que está el modelo.

- **Pose final**: tras estimar la pose se realiza un refinamiento de la misma para que su representación sea lo mas precisa posible. En este caso se utiliza el algoritmo ICP (*Iterative Closest Point*) que minimiza la diferencia entre dos nubes de puntos.

Como lo que nos interesa es conocer la localización del robot con respecto al mapa, es necesario aplicar una cadena de transformaciones a la pose final calculada. Pese a que los resultados de este procedimiento son relativamente buenos, no se tiene en cuenta el estado anterior del robot ni la información odométrica y son aspectos que ofrecen una gran cantidad de información relevante. Luego es recomendable explorar otras opciones.

En cuarto y último lugar, se ha decidido combinar las diferentes fuentes de información disponibles como la odometría, el láser y la imagen RGB-D para la localización gracias al proyecto de código abierto llamado RTAB-Map. Éste dispone de un amplio número de parámetros que se pueden modificar y ajustar en base a las condiciones en las que se encuentra el robot. En las pruebas realizadas en el laboratorio, su funcionamiento es muy eficaz ya que localización es precisa y constante, y en muchos casos resuelve el problema de la autolocalización y el problema del robot secuestrado. Es de añadir que en dichas pruebas en nivel de ruido del entorno era bajo ya que no había personas o elementos dinámicos en las proximidades de los sensores.

4.3.6. Modificación de parámetros

Debido a que el robot dispone de varias formas de calcular su localización en el mapa, surge la necesidad que crear un sistema que los gestione y decida cual es el más preciso en cada momento. Para ello el primer requisito es que exista la posibilidad de cambiar el método que proporcione el posicionamiento en tiempo real. Y no solo eso, si no que es recomendable que todos los parámetros que dependan del entorno puedan ser modificados en cualquier momento con el fin de mejorar la calidad del posicionamiento.

Objetivo

Modificar las implementaciones de los diferentes algoritmos de localización para que los parámetros de éstos ofrezcan la posibilidad de adaptarlos en base a las mediciones que los sensores estén obteniendo en cada instante.

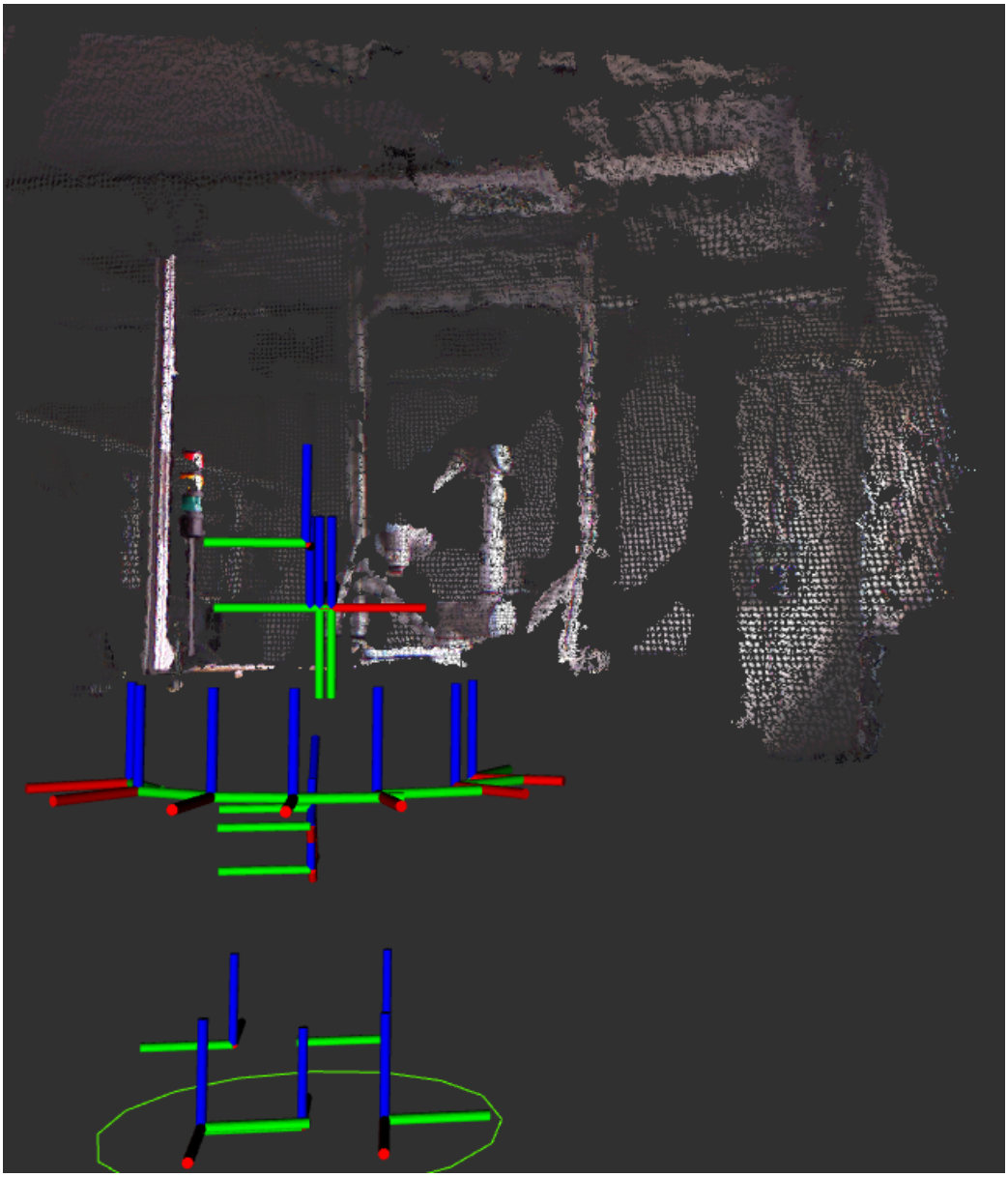

Figura 4.5: Estructura del robot y en frente la nube de puntos obtenida por la Kinect.

Progreso

ROS ofrece un paquete llamado *dynamic_reconfigure* que proporciona los medios para cambiar los parámetros de un nodo sin tener que reiniciarlo. De manera que el código de gestiona teknigazer, amcl y RTAB-Map ha sido modificado y ahora se pueden modificar sus parámetros a través en tiempo

de ejecución.

4.3.7. Interacción

Dado que el robot dispone de conexión inalámbrica a Internet mediante un módem USB y hasta ahora éste solo se utiliza para recibir cierto feedback de sus comportamientos. Es interesante que exista la opción de comandar el robot mediante esta vía de comunicación, qué menos al encontrarnos en la era del *Internet of things*.

Objetivo

La idea es disponer de una aplicación móvil, por ejemplo, con la que poder enviar destinos, cancelarlos, modificar algunos estados del robot, etc. También se quiere mostrar en la pantalla de dicho dispositivo externo información sobre la situación del robot como el mapa, la localización, la información captada por los sensores, etc.

Progreso

Se han añadido nuevas funciones para la recepción de ciertas acciones. Estando ambos sistemas conectados en una misma red wifi, el dispositivo se conecta mediante TCP a la IP del robot y manda las tramas correspondientes. Luego estas son traducidas en acceso a los medios de ROS (topics, servicios, acciones) y gestionadas en el robot, devolviendo un feedback y ofreciendo una gran parte de la información que se muestra en la propia pantalla táctil. En las pruebas que se han realizado la comunicación se lleva a cabo satisfactoriamente pero la gestión de la misma está todavía en una fase inicial y hay que reestructurar el sistema para mantener una buena coordinación.

Capítulo 5

Conclusión

5.1. Conclusión

Los avances conseguidos y los procedimientos vistos a lo largo de este proyecto parecen prometedores. El comportamiento del robot es en muchos casos satisfactorio ya que consigue alcanzar sus objetivos. Sin embargo, en las situaciones que el nivel de ruido sensorial es elevado el robot tiene comportamientos que distan mucho de lo que la sociedad espera de un ser inteligente. Es bien sabido que para que la robótica avance, es necesario una robusta estructura hardware convenientemente adaptada al problema que se plantea y un sistema de gestión eficiente que logre el máximo rendimiento de dichos componentes. Esto permitirá utilizar algoritmos más potentes consiguiendo una mejor síntesis de la información obtenida por los sensores, lo que conlleva al tratamiento de un mayor porcentaje de información relevante. Unos sensores de mayor precisión permiten que la percepción de la realidad sea mucho más precisa, en consecuencia el robot tendrá un mayor control del entorno y de sí mismo. Además, un ordenador más potente permitirá la realización de más o mejores algoritmos aumentando la capacidad de procesamiento, evitando problemas de retardos como los que han surgido en la realización del proyecto. No obstante, éstos elementos tienen un consumo de energía más elevado y requieren de mayores fuentes de alimentación. En definitiva, las diferentes partes del robot deben evolucionar conjuntamente y de forma uniforme.

5.2. Trabajo futuro

Pese a que Teknibot tenga un funcionamiento aceptable, todavía le falta un largo camino que recorrer. Es en este apartado donde veremos las ideas que han surgido a lo largo del proyecto pero que no se han podido llevar a cabo todavía y se intentarán incorporar más adelante.

Como hemos comentado en la mejora de la interacción con el robot, tal y como está implementada la logística del sistema actualmente no permite una buena coordinación de diferentes modos de control del robot. De modo que el primer cambio que se pretende realizar es reestructurar toda la lógica del sistema para que sea posible la convivencia de N fuentes de control.

La segunda mejora que se quiere incorporar es la coordinación de N fuentes de localización. Actualmente existen tres formas de que el robot pueda conocer su posicionamiento en el mapa, pero la elección de una u otra es muy simple. Por lo tanto disponer de un 'cerebro' que decida cual de ellas utilizar

en cada situación específica marcaría una gran diferencia.

Otro punto crítico del robot es la recuperación ante un problema en la navegación. Actualmente adopta un comportamiento rotacional muy poco inteligente, eficaz en muy pocas ocasiones y que genera cierta desconfianza a la gente. Un robot guía orientado a las personas como es éste, no solo debe resolver los problemas que se le plantean con éxito sino que debe crear en los individuos un sentimiento de seguridad de que así lo va a ser. Por ésta razón es necesario adoptar otros comportamientos ante un imprevisto en la navegación como lo es un obstáculo que impida alcanzar el destino o una pérdida en la localización. Se han propuesto ideas como la de buscar puntos característicos en el entorno o comunicarse con los seres de alrededor para que éstos le ayuden, pero todavía hay que realizar un estudio en profundidad.

En cuanto a los componentes hardware se refiere, se pretende incorporar una unidad de medición inercial o IMU (del inglés *inertial measurement unit*) para corregir la desviación de la odometría. En la línea de la percepción del entorno una mejora relevante sería la utilización de una omnicámara para un control en 360º del área de trabajo. En un futuro más lejano cuando la precisión y robustez del sistema lo permitan también se quiere incorporar más actuadores, e.g. un brazo robótico, para ofrecer un mayor número de servicios.

Por último, y quizá ya independiente del campo de la robótica. Se intentará añadir una funcionalidad para la creación de modelos 3D precisos del entrono en un formato normalizado para su posterior uso en entornos de simulación, por ejemplo.

Bibliografía

[1] S. Thrun *et al.*, "Robotic mapping: A survey," *Exploring artificial intelligence in the new millennium*, vol. 1, pp. 1–35, 2002.

[2] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, 2009.

[3] C. Stachniss, *Robotic mapping and exploration*, vol. 55. Springer, 2009.

[4] E. B. Olson, "Robust and efficient robotic mapping," 2008.

[5] J. M. Bernardo and A. F. Smith, "Bayesian theory," 2001.

[6] U. Raschke and J. Borenstein, "A comparison of grid-type map-building techniques by index of performance," in *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pp. 1828–1832, IEEE, 1990.

[7] W. Burgard, C. Stachniss, G. Grisetti, B. Steder, R. Kümmerle, C. Dornhege, M. Ruhnke, A. Kleiner, and J. D. Tardós, "A comparison of slam algorithms based on a graph of relations," in *The 2009 IEEE/RSJ International Conference on intelligent Robots and Systems, October 11-15, 2009 St. Louis, USA*, pp. 2089–2095, IEEE conference proceedings, 2009.

[8] H.-T. Cho and S. Jung, "Neural network position tracking control of an inverted pendulum an xy table robot," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 2, pp. 1210–1215, IEEE, 2003.

[9] O. L. Casanova *et al.*, "Robot position tracking using kalman filter," 2008.

[10] W. Burgard, D. Fox, and D. Hennig, "Fast grid-based position tracking for mobile robots," in *KI-97: Advances in Artificial Intelligence*, pp. 289–300, Springer, 1997.

[11] P. Jensfelt and S. Kristensen, "Active global localization for a mobile robot using multiple hypothesis tracking," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 5, pp. 748–760, 2001.

[12] S. Se, D. Lowe, and J. Little, "Global localization using distinctive visual features," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 1, pp. 226–231, IEEE, 2002.

[13] S. Koenig, A. Mudgal, and C. Tovey, "A near-tight approximation lower bound and algorithm for the kidnapped robot problem," in *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pp. 133–142, Society for Industrial and Applied Mathematics, 2006.

[14] A. Majdik, M. Popa, L. Tamas, I. Szoke, and G. Lazea, "New approach in solving the kidnapped robot problem," in *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, pp. 1–6, VDE, 2010.

[15] W. Burgard, A. B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun, "Experiences with an interactive museum tour-guide robot," *Artificial intelligence*, vol. 114, no. 1-2, pp. 3–55, 1999.

[16] S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Haehnel, C. Rosenberg, N. Roy, *et al.*, "Probabilistic algorithms and the interactive museum tour-guide robot minerva," *The International Journal of Robotics Research*, vol. 19, no. 11, pp. 972–999, 2000.

[17] G. Welch and G. Bishop, "An introduction to the kalman filter," 1995.

[18] D. Fox, W. Burgard, and S. Thrun, "Markov localization for mobile robots in dynamic environments," *Journal of Artificial Intelligence Research*, vol. 11, pp. 391–427, 1999.

[19] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," *AAAI/IAAI*, vol. 1999, no. 343-349, pp. 2–2, 1999.

[20] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[21] S. Thrun and J. J. Leonard, "Simultaneous localization and mapping," in *Springer handbook of robotics*, pp. 871–889, Springer, 2008.

[22] R. Bohlin, *Robot path planning*. Chalmers University of Technology, 2002.

[23] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[24] N. Correll, "Introduction to autonomous robots," 2014.

[25] B. V. Cherkassky, A. V. Goldberg, and T. Radzik, "Shortest paths algorithms: Theory and experimental evaluation," *Mathematical programming*, vol. 73, no. 2, pp. 129–174, 1996.

[26] H. G. Nguyen, J. Morrell, K. D. Mullens, A. B. Burmeister, S. Miles, N. Farrington, K. M. Thomas, and D. W. Gage, "Segway robotic mobility platform," in *Optics East*, pp. 207–220, International Society for Optics and Photonics, 2004.

[27] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.