UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

SISTEMA DE NAVEGACIÓN PARA UN ROBOT DE SERVICIO

TESIS

QUE PARA OPTAR POR EL GRADO DE:
DOCTOR EN INGENIERÍA
(COMPUTACIÓN)

PRESENTA:
MARCO ANTONIO NEGRETE VILLANUEVA

DIRECTOR DE TESIS:
DR. JESÚS SAVAGE CARMONA
FACULTAD DE INGENIERÍA, UNAM

Ciudad Universitaria, Cd. Mx. **Marzo de 2019**

… tú sabes que te dedico toda mi vida,
pero por si acaso no lo recuerdas,
este trabajo es para ti, Araceli.

Agradecimientos

Este trabajo se realizó con el apoyo del CONACYT, a través de la beca 233809, y de DGAPA-UNAM, a través del proyecto PAPIIT IG100818.

Agradezco sinceramente …

… a Lucrecia y Encarnación, mis padres, por el apoyo incondicional y por haberme puesto en este camino.

… a Ardia, mi hermana, quien me ha acompañado, guiado e inspirado, desde el primer día de kinder hasta esta etapa que está por concluir.

… a mi sobrina, Sofía, por alegrar mi vida con todas las sonrisas y abrazos que siempre tiene para mí.

… a Jesús, Jaime y José Luis, amigos y colegas, quienes me han enriquecido tanto personal como profesionalmente.

… a Sofía, Giles, Ale y Lizeth, con quienes he compartido este viaje llamado posgrado.

… a mis amigas Paulina, Jimena, Somara y Cassandra, con quienes he compartido ese otro viaje llamado psicología.

… a todos los miembros del Laborotario de Biorrobótica.

… a mi mentor, el Dr. Jesús Savage Carmona, por compartir sus conocimientos, por guiar mi quehacer académico y sobre todo, por su no tan paciente espera.

… a los miembros de mi jurado, Dr. Arturo Bouzas, Dr. Marco Morales, Dr. Ángel Kuri y Dr. Luis Pineda, por sus enriquecedoras recomendaciones.

… y a las estrellas, el azar y la selección natural, por haber transformado el hidrógeno en la hermosa persona que ahora inspira y motiva todos mis proyectos: Araceli, mi esposa.

Resumen

Los robots de servicio doméstico requieren de sistemas de navegación robustos que les permitan realizar sus tareas en ambientes dinámicos. Actualmente existen soluciones para la planeación de rutas, evasión de obstáculos, control de posición, mapeo y localización, sin embargo, hay poco trabajo reportado con respecto a sistemas de planeación de movimientos totalmente integrados y probados en ambientes reales.

En este trabajo se describe el sistema de navegación desarrollado para un robot de servicio doméstico. La propuesta se desarrolló tomando como guía el ViRBot, una arquitectura para desarrollar software para operar robots móviles autónomos. El sistema de navegación consta de un planeador de rutas basado en el algoritmo A*, donde se propone una función de costo para que las rutas calculadas sean más seguras. Para seguir estas rutas se implementaron leyes de control no lineales con base en el modelo cinemático del robot que, en conjunto con una máquina de estados, logran el seguimiento de la ruta con un determinado perfil de velocidad. La evasión de obstáculos se resolvió usando métodos basados en comportamientos: mediante un árbitro basado en prioridad que coordina tres comportamientos, se logró una evasión de obstáculos y se evitó el problema de los mínimos locales en los campos potenciales, usados en uno de los comportamientos. La cuantización vectorial es un algoritmo de agrupamiento que se utilizó para construir mapas de rutas. Los algoritmos de agrupamiento son computacionalmente costosos y esto dificulta su construcción en línea, sin embargo, mediante una implementación en paralelo, se redujo significativamente el tiempo de ejecución, lo que permitió utilizar las construcción de mapas de rutas como parte de los métodos de evasión de obstáculos.

Para la localización se implementó un Filtro de Kalman Extendido y la

extracción de líneas del ambiente, sin embargo, el filtrado sólo se lleva a cabo en ciertas regiones marcadas como confiables para localización. Las regiones en las que no se ejecuta la localización son aquellas con mucho desorden y en las que es difícil extraer líneas. En estas regiones, el robot basa la estimación de su posición sólo en la odometría. Esta estrategia está inspirada en los hallazgos hechos en la biología del comportamiento que indican que algunos insectos navegan usando marcas del ambiente, pero cuando la discrepancia entre lo que observan y lo que estiman es grande, entonces comienzan a navegar usando únicamente la integración de rutas.

El sistema desarrollado se comparó con el paquete de código abierto *nav2d* en pruebas tanto en simulación como con el robot real. Cuatro parámetros fueron utilizados para evaluar el desempeño general de ambos sistemas: la tasa de la distancia recorrida contra la distancia euclideana entre el punto inicial y el punto meta, la velocidad promedio, el número de colisiones y el número de puntos meta alcanzados. Se hicieron pruebas t-Student para determinar si hubo diferencias significativas. El sistema propuesto se desempeñó mejor de acuerdo con las diversas medidas obtenidas. Finalmente, el sistema también se probó en las tareas de la competencia @Home, cuyas pruebas están diseñadas para evaluar el desempeño general de robots de servicio doméstico.

Abstract

Domestic service robots require robust navigation systems in order to be able to perform their intended tasks in dynamic environments. Currently, there exist solutions for the problems of path planning, obstacle avoidance, position control, mapping and localization, nevertheless, there is a lack of work reported regarding fully integrated systems tested in real environments.

In this work we describe the navigation system that was developed for a domestic service robot. The proposal was developed following the ViRBot, an architecture for the development of software to operate autonomous mobile robots. The navigation system is composed of a path planner based on the A* algorithm, where it is proposed a cost function such that the calculated paths are safer. To track the paths, non linear control laws were implemented based on the kinematic model of the robot. These control laws, together with an augmented finite state machine, achieve the path tracking with a given speed profile. Obstacle avoidance was addressed using behavior based methods: using a priority based arbiter, which coordinates three behaviors, obstacle avoidance was achieved and also the problem of local minima was overwhelmed, which is a typical problem when using potential fields. Vector quantization is a clustering algorithm, which was used to build roadmaps. Clustering algorithms typically have high computational costs and thus, it is difficult to implement them for online roadmap building, nevertheless, with a parallel implementation, running time was significantly reduced, which allowed the use of online roadmap construction as a method for obstacle avoidance.

Line extraction from laser readings and the Extended Kalman Filter were used for localization, nevertheless, position filtering is performed only in certain regions marked as reliable for localization. The regions in which lo-

calization is not executed are those with messy environments, where it is difficult to extract lines. In these regions, robot estimates its position based only in odometry. This strategy is inspired in the findings made in the field of biology of behavior, which indicate that some insects navigate using landmarks but, when there is a big discrepancy between what ti is observed and what it is expected, they start to navigate using only path integration.

The proposed system was compared with the open source package *nav2d* with tests both simulated and with a real robot. Four parameters were used to evaluate the overall performance of both systems: the ratio of traveled distance to the euclidean distance from the start point to the goal point, the average speed, the number of collisions and the number of reached goal points. In order to determine if there were significant differences, several t-Student tests were made. The proposed system performed better according to the obtained measurements. Finally, the system was also tested in the task of the @Home competition for domestic service robots, whose tests are designed to evaluate the overall performance of these kind of robots.

Índice general

Índice de figuras

Capítulo 1

Introducción

La robótica es una disciplina que ha cobrado una gran importancia en los últimos años debido no sólo a los avances que ha tenido en cuanto a sus conceptos y metodologías, sino al impacto social que se hace cada vez más evidente. El desarrollo de técnicas de visión computacional, control automático, representación del conocimiento, toma de decisiones, entre otras, aunado al rápido aumento del poder de procesamiento de las computadoras, han logrado llevar a los robots de ser tan solo un tema de ciencia ficción, a una realidad muy próxima como parte de la vida diaria de los humanos.

Son numerosos los problemas en los que actualmente se pueden utilizar, o ya se están utilizando, robots de diversos tipos, desde teleoperados hasta totalmente autónomos. Las aplicaciones industriales fueron las primeras en las que los robots mostraron su utilidad, sin embargo, en la actualidad la manufactura es tan sólo una de un conjunto muy variado de usos: en medicina, agricultura, vigilancia, industria aeroespacial, rescate, etc. Por su aplicación, los robots se pueden clasificar en dos grandes grupos: robots industriales y robots de servicio. Por el nivel de intervención humana en la realización de una tarea, se pueden clasificar en robots teleoperados, semi-autónomos y autónomos.

El vertiginoso desarrollo de la robótica, que se hace patente tanto en el número de robots que existen actualmente como en la gran diversidad de tipos y aplicaciones, hace que definir qué es un robot sea una tarea complicada. Existen en la actualidad desarrollos tan diversos que una definición muy específica se arriesga a dejar fuera muchos sistemas que bien podrían llamarse robots, mientras que una definición laxa seguramente abarcaría sistemas que la mayoría de desarrolladores no consideraría como robots.

Debido a que la robótica es relativamente nueva, aún existen varias definiciones de *robot* que en general varían dependiendo del campo de aplicación o del área de investigación. De acuerdo con la Real Academia Española, un robot es una máquina o ingenio electrónico programable capaz de manipular objetos y realizar operaciones antes reservadas sólo a las personas. Latombe (1991) define un robot como un dispositivo mecánico versátil equipado con sensores y actuadores bajo el control de un sistema de cómputo. Dado que este trabajo se enfoca en robots móviles autónomos, una definición más adecuada es la de Arkin (1998), quien propone que un robot inteligente es una máquina capaz de extraer información de su ambiente y usar el conocimiento acerca de su mundo para moverse de manera segura y significativa, con un propósito específico. Esta definición es la que mejor se adapta a este trabajo y por ello será la definición utilizada.

1.1. Robots de servicio doméstico

De acuerdo con la Organización Internacional para la Normalización (ISO), un robot de servicio es un robot que realiza tareas útiles para los humanos o equipos excluyendo aplicaciones de automatización industrial (ISO, 2012). Esta norma también establece que un robot, en general, requiere de cierto grado de autonomía, la cual, en este contexto, se considera como la habilidad para realizar tareas basadas en el estado actual y en el sensado, sin intervención humana. El grado de autonomía en un robot de servicio varía de la autonomía parcial, que incluye la interacción humano-robot, a la autonomía completa, en la que no se requiere ninguna intervención humana.

La Federación Internacional de Robótica clasifica los robots de servicio en personales y profesionales (IFR, 2017). Ambos tipos se clasifican en categorías de acuerdo con el área de aplicación. Los robots de servicio personales incluyen a aquellos para tareas domésticas, robots de entretenimiento, para cuidado de adultos mayores, asistencia a personas con discapacidad, transportación personal y vigilancia y seguridad en hogares. Los robots de servicio profesionales incluyen, entre otros, robots de campo (por ejemplo, para agricultura, minería o aplicaciones espaciales), para inspección y mantenimiento, logística, aplicaciones médicas, rescate, seguridad y vehículos no tripulados (IFR, 2017). Aunque por definición un robot de servicio no tiene que ser necesariamente autónomo, entre mayor grado de autonomía tenga, mayor será su utilidad. Por ejemplo, en aplicaciones espaciales, dado al retraso que cau-

san las enormes distancias, la teleoperación no es factible y por lo tanto se requiere un robot completamente autónomo. En aplicaciones domésticas, si un robot tiene la tarea de limpiar una habitación, es deseable que lo haga sin necesidad de ayuda humana. Por ello, es deseable un alto grado de autonomía en robots de servicio doméstico.

1.2. Motivación

Para ser autónomo, un robot de servicio necesita varias habilidades: planeación de movimientos en ambientes dinámicos, reconocimiento y manipulación de objetos, interacción humano-robot, que puede incluir detección de humanos, reconocimiento de gestos y síntesis y reconocimiento de voz; planeación de acciones, entre otras. El problema de la planeación de movimientos se refiere a la habilidad del robot de localizarse a sí mismo, realizar un mapa del ambiente y planear rutas seguras de un punto a otro (Choset et al., 2005).

Un sistema de planeación de movimientos consiste en varios componentes que están interconectados entre sí y cuyo desempeño depende de otros módulos. Por ejemplo, seguir una ruta depende del planeador de movimientos y del control de bajo nivel y, a su vez, ambos dependen de la localización. Pero esta dependencia también se da en sentido contrario, puesto que, si el robot se mueve abruptamente (por alguna falla en el control de bajo nivel), será más probable que el sistema de localización falle. En principio, en un sistema de planeación de movimientos, se podrían probar todos los módulos de manera aislada y la confiabilidad del sistema completo se podría argumentar con la confiabilidad de cada componente. Sin embargo, esto es posible sólo si no hay realimentación entre los sistemas, de lo contrario, el desempeño del sistema completo podría ser diferente al desempeño de cada módulo individualmente.

Como se expresa en Amigoni et al. (2015), las competencias de robótica resultan útiles para probar y comparar diferentes algoritmos y sistemas ya que proporcionan una forma objetiva de evaluar el desempeño bajo condiciones controladas y replicables. Robocup@Home (Wachsmuth, Holz, Rudinac, y Ruiz-del Solar, 2015) y RoCKIn@Home (Amigoni et al., 2015) son ejemplos de competencias para robots de servicio doméstico donde se evalúan habilidades específicas así como el desempeño general. Contrario a los experimentos, donde se prueban hipótesis específicas, las competencias usualmente evalúan habilidades generales en robots. Además, probar un sistema en una competencia también ayuda a encontrar soluciones a problemas que solo emergen

en sistemas completamente integrados.

Existe una gran cantidad de trabajo reportado en la literatura científica sobre el problema de la planeación de movimientos para robots autónomos, sin embargo, éste es aún un problema abierto y desafiante cuando se trata de robots navegando en ambientes reales (Banino et al., 2018). Esto puede verse reflejado, por ejemplo, en la competencia Robocup@Home 2018, donde el libro de reglas contiene tres pruebas en las que la tarea a resolver está enfocada, entre otros problemas, en la navegación segura y la evasión de obstáculos, además, como se describe en Iocchi, Ruiz-del Solar, y van der Zant (2012), los robots de servicio doméstico aún tienen desempeños muy bajos en cuanto a tareas de navegación. La navegación autónoma está desarrollada al punto en que se pueden encontrar bibliotecas abiertas de software para implementar sistemas de navegación, pero estas bibliotecas generalmente implementan sólo habilidades básicas y aún se puede realizar bastante trabajo, por ejemplo, en la mejora de la planeación de rutas en ambientes difíciles como habitaciones muy desordenadas. El libro de reglas de la Robocup@Home también contempla pruebas en las que el robot tiene que evadir obstáculos "difíciles de ver", por ejemplo, una manzana, un vaso o una pieza de Lego. El echo de que competencias muy recientes incluyan tareas para probar navegación autónoma en robots de servicio doméstico, muestra que la navegación es aún un problema abierto.

1.3. Planteamiento del problema

Actualmente existen soluciones para habilidades aisladas en el problema de la planeación de movimientos pero aún hay una falta de soluciones cuando se trata de sistemas completamente integrados para navegación en ambientes domésticos reales. Habilidades como localización, mapeo y planeación de rutas están resueltas bajo ciertas consideraciones pero existe una falta de trabajo cuando se trata de sistemas que integren todas estas tecnologías.

Solucionar el problema de la planeación de movimientos implica el desarrollo de una gran cantidad de software. Dicho software debe ser extensible y de fácil mantenimiento para lograr la integración de los diferentes subsistemas. Por lo tanto, se requiere de una arquitectura de diseño.

El ambiente en el que los robots de servicio desempeñan sus tareas es altamente dinámico, por lo que el sistema de planeación de movimientos debe tener respuestas rápidas ante cambios inesperados. Sin embargo, debido

al creciente nivel cognitivo de las tareas a resolver, un robot de servicio doméstico debe tener también habilidades de planeación y predicción.

1.4. Hipótesis

El problema planteado y el desarrollo de este trabajo tienen como base las siguientes hipótesis:

- La navegación de un robot móvil es más segura si se planean rutas empleando la cercanía a los obstáculos como función de costo en un algoritmo de búsqueda, que si se emplean mapas de costo como se realiza comúnmente en robots de servicio.
- Se pueden construir mapas de rutas empleando técnicas de agrupamiento de datos.
- Los mapas de rutas pueden ser usados como método de evasión de obstáculos si se construyen lo suficientemente rápido.
- La combinación de métodos basados en comportamientos con métodos de planeación jerárquicos hacen más segura la navegación en robots móviles autónomos.
- Hallazgos hechos en el área de la biología del comportamiento sobre navegación en animales pueden ser útiles para el desarrollo de sistemas de navegación para robots móviles autónomos.

1.5. Contribuciones y publicaciones

Las principales contribuciones de este trabajo son:

- Propuesta de un módulo de planeación de rutas utilizando una función de costo que da como resultado rutas más seguras.
- Un método para construir mapas de rutas a partir de nubes de puntos.
- La implementación en paralelo de este método y las pruebas para su uso en evasión de obstáculos.

- Un método de localización con base en líneas en el ambiente y el Filtro de Kalman Extendido.

- El uso de hallazgos sobre navegación en animales para mejorar el sistema de navegación.

- Integración de los diferentes módulos en un sistema de planeación de movimientos con base en la arquitectura ViRBot.

Son resultado directo de esta tesis los siguientes artículos presentados en congresos internacionales:

- Negrete, M., Savage, J., Cruz, J. y Márquez, J. (2014) Behavior-Based Navigation System for a Service Robot with a Mechatronic Head. En *Congreso Latinoamericano de Control Automático*, Cancún, México.

- Negrete, M., Savage, J., Cruz, J. y Márquez, J. (2014) Parallel Implementation of Roadmap Construction for Mobile Robots using RGB-D Cameras. En *Open German-Russian Workshop on Pattern Recognition and Image Understanding*, Koblenz, Alemania.

También son resultado directo los siguientes artículos publicados en revistas:

- Negrete, M., Savage, J. y Contreras, L. (2018) A motion planning system for a domestic service robot. *SPIIRAS Proceedings*, Vol. 5, No. 60. pp 5-38.

- Negrete, M., Savage, J. y Contreras, L. (2018) A motion planning system for a domestic service robot using de ViRBot architecture. *International Journal of Advanced Robotics Systems*. (Aceptado para su publicación).

- Savage, J., Rosenbluth, D., Matamoros, M., Negrete, M., Contreras, L., Cruz, L., Martell, R., Estrada, H.E. y Okada, H. (2019) Semantic Reasoning in Service Robots Using Expert Systems. *Robotics and Autonomous Systems*. Vol. 114, pp 77-92.

1.6. Descripción del documento

Este trabajo está organizado de la siguiente manera: en el capítulo 2 se presentan los conceptos básicos sobre navegación en robots autónomos que serán utilizados a lo largo de este trabajo y se hace una descripción breve de la arquitectura ViRBot (acrónimo de *Virtual-Real Robot*), que se utilizó para el desarrollo del sistema propuesto. Al mismo tiempo, en este capítulo se hace una revisión del trabajo relacionado para mostrar las aportaciones de esta propuesta. En los capítulos 3 al 6 se describe el sistema de planeación de movimientos en sí. En el capítulo 3 se explica el módulo de planeación de rutas y el control de bajo nivel empleado para seguir dichas rutas. En el capítulo 4 se describe el cartógrafo, módulo de la arquitectura ViRBot encargado de construir la representación del ambiente y de determinar la posición del robot dentro de dicho ambiente. El capítulo 5 está dedicado a explicar el módulo de los métodos basados en comportamientos. En el módulo de localización se empleó una estrategia que estuvo inspirada en hallazgos hechos en el área de la biología del comportamiento sobre navegación en animales. En el capítulo 6 se explican estos hallazgos y la forma en que se utilizaron en el sistema de navegación propuesto. Se realizaron pruebas tanto en simulación como con dos robots reales: Justina y HSR, el primero, desarrollado en el Laboratorio de Biorrobótica y el segundo, desarrollado por la empresa Toyota. Se realizaron varias pruebas en el Laboratorio, sin embargo, las pruebas más importantes, fueron las tareas de la competencia Robocup@Home. Las características de los robots Justina y HSR, así como los resultados, se describen en el capítulo 7. Finalmente, en el capítulo 8 se dan las conclusiones y se plantea el trabajo futuro.

Capítulo 2

Antecedentes

En este capítulo se hace una revisión de los conceptos básicos que serán utilizados en el resto del documento. Primero, en la sección 2.1, se revisan los tres paradigmas de la robótica: jerárquico, reactivo e híbrido. Estos tres paradigmas se describen con base en la relación entre las tres primitivas de la robótica, sensar, planear y actuar, que también se revisan en este capítulo. También se discute cómo estos paradigmas están relacionados con el sistema de planeación de movimientos que se propone. En la sección 2.2 se revisa en sí el problema de la planeación de movimientos, las tareas que éste involucra y aquellas que se abordan en este trabajo. La sección 2.3 está dedicada a describir el ViRBot, un arquitectura para desarrollar software para robots móviles. Se explican a grandes rasgos cada una de las capas que lo componen, los módulos en cada capa, y se enfatizan aquellos relacionados con el sistema propuesto. En la sección 2.4, se discute el trabajo relacionado y las contribuciones de este trabajo.

2.1. Paradigmas y primitivas de la robótica

En general, las tareas que puede realizar un robot se pueden clasificar en tres grandes conjuntos conocidos como primitivas de la robótica: sensar, planear y actuar (Murphy, Murphy, y Arkin, 2000). A continuación se da una breve definición de cada una de ellas.

Sensar. Esta primitiva se refiere a la extracción de información del ambiente ya sea interno o externo del robot. Un transductor es un dispositivo que convierte una forma de energía en otra y éste se convierte en sensor

cuando la información que produce es útil para que el robot realice su tarea.

Planear. Se refiere a la generación de subtareas y toma de decisiones con base en la información obtenida de los sensores y/o de alguna representación del ambiente previamente generada. La planeación genera directivas que son enviadas a los actuadores. El hardware necesario para realizar planeación puede ser cualquier dispositivo para procesar información: CPU's, GPU's, microcontroladores, DSP's, entre otros.

Actuar. Esta primitiva se refiere a la modificación del ambiente por alguno de los dispositivos del robot. Estos dispositivos se conocen como actuadores y los comandos que se les envían pueden ser generados directamente de la información de los sensores o bien, de las directivas creadas en la etapa de planeación.

La forma en que las primitivas se interconectan unas con otras define a los llamados *paradigmas de la robótica*: jerárquico o tradicional, reactivo e híbrido (Bekey, 2005). En los siguientes párrafos se describen algunas de sus características.

Jerárquico. En este paradigma, las tres primitivas se realizan de forma secuencial, como se muestra en la figura 2.1. El hecho de tener una etapa de planeación antes de la actuación, hace que este paradigma tenga las siguientes características:

- Fuerte dependencia de una representación interna del ambiente.
- La latencia es grande en comparación con el paradigma reactivo, es decir, puede haber retrasos significativos entre los estímulos de entrada y la respuesta del robot.
- El costo computacional en general es alto.
- Debido a que se tiene una representación interna del ambiente, este paradigma tiene altas capacidades de predicción.
- El robot puede resolver tareas de alto nivel cognitivo.

Un ejemplo clásico de este paradigma es la arquitectura del robot Shakey (Nilsson, 1969). Este software estaba compuesto por un sistema de sensado que traducía la imagen de la cámara en una representación interna del mundo. La parte de planeación tomaba el modelo interno del mundo y se generaba un plan para alcanzar una meta. El ejecutor tomaba el plan y enviaba las

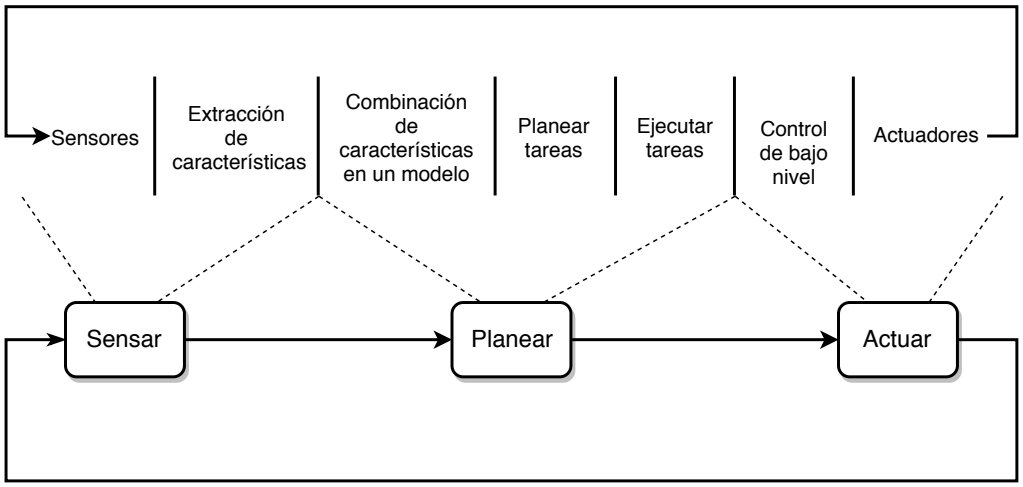


Figura 2.1: Ejemplo de las diferentes etapas en un sistema desarrollado bajo el paradigma jerárquico. Cada etapa se ejecuta de manera secuencial tomando como entradas las salidas de la etapa anterior (Murphy et al., 2000).

acciones correspondientes a los actuadores. La figura 2.1 muestra un ejemplo de sistema robótico bajo el paradigma jerárquico.

Reactivo. En este paradigma, las primitivas sensar y actuar están conectadas directamente sin una etapa de planeación de por medio, como se muestra en la figura 2.2. Sus principales características son las siguientes:

- No se requiere de una representación interna del ambiente.
- El tiempo de respuesta es menor en comparación con el paradigma jerárquico, es decir, se tienen latencias cortas.
- El costo computacional es bajo.
- La capacidad de predicción es limitada.
- En general, no se pueden resolver tareas de alto nivel cognitivo.

Este paradigma fue expuesto inicialmente por R. Brooks (1986) como una contrapropuesta al paradigma jerárquico. En este trabajo, Brooks afirma que no se necesita de una representación del mundo para lograr un comportamiento inteligente en los robots y que, el único "modelo" del mundo que se requiere es el mundo mismo (R. A. Brooks, 1991).

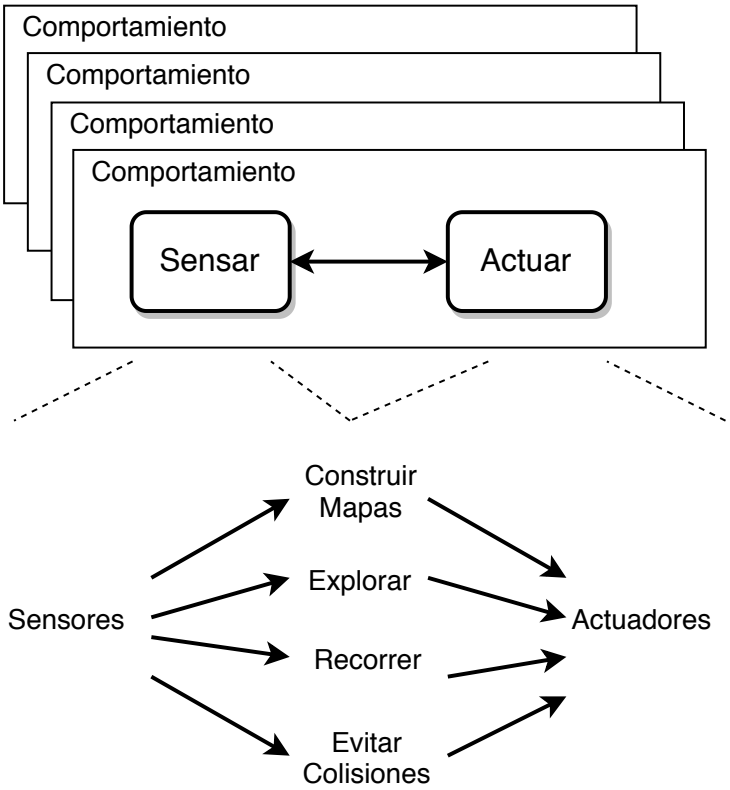


Figura 2.2: Ejemplo de un sistema bajo el paradigma reactivo. El sensar y actuar están directamente conectados formando un comportamiento. El sistema completo consiste en varios de estos comportamientos.

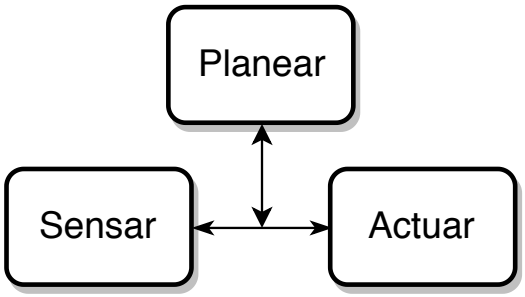


Figura 2.3: En el paradigma híbrido las primitivas se relacionan como planear y luego sensar-actuar (Murphy et al., 2000).

Híbrido. Este paradigma surge para utilizar las ventajas de los dos paradigmas previos, es decir, usar comportamientos reactivos para obtener respuestas rápidas (lo que le permite al robot desenvolverse en ambientes dinámicos) y al mismo tiempo, usar representaciones internas para resolver tareas de alto nivel cognitivo. La forma en que se relacionan las primitivas en este paradigma se puede resumir en planear y luego sensar-actuar. La figura 2.3 muestra la relación entre las tres primitivas en el paradigma híbrido.

El sistema de navegación desarrollado en este trabajo corresponde al paradigma híbrido. Como se mencionó en la sección 1.3, un robot de servicio doméstico se desenvuelve en un ambiente dinámico mientras resuelve tareas complejas. Por ejemplo, para planear un ruta se requiere de un mapa, es decir, de una representación interna, pero, si se presenta un obstáculo inesperado mientras el robot se mueve, se requiere de una respuesta rápida para evadirlo. Como se verá en la sección 2.3, la arquitectura ViRBot comprende módulos tanto reactivos como jerárquicos.

2.2. Tareas en el problema de la planeación de movimientos

El problema de la planeación de movimientos comprende cuatro tareas principalmente: navegación, cobertura, localización y mapeo (Choset et al., 2005). Para estas cuatro tareas es necesario tener una descripción o representación de los puntos en el espacio que ocupa el robot. A esta descripción se le llama *configuración* y el *espacio de configuraciones* es el conjunto de todas las configuraciones posibles que puede tener el robot.

La navegación es el problema de encontrar un movimiento libre de colisiones desde una configuración inicial a una final. La cobertura se refiere al problema de mover un sensor o una herramienta de modo que se asegure que se cubren todos los puntos de un espacio determinado. La localización consiste en determinar la configuración del robot dado un mapa y un conjunto de lecturas de los sensores. El mapeo consiste en la exploración y sensado de un ambiente desconocido de modo que se obtenga una representación de dicho ambiente que sea útil para alguna de las otras tareas. Al problema de la localización y mapeo simultáneos se le conoce como SLAM (por sus siglas en inglés) (G. Dissanayake, Huang, Wang, y Ranasinghe, 2011). En este trabajo se abordarán los problemas de navegación, mapeo y localización.

2.2.1. Características del robot

Para poder construir un planeador de movimientos es necesario conocer primero las características del robot a utilizar. La primera de ellas es el número de *grados de libertad* que se refiere al número de parámetros independientes que definen la configuración del robot. En el caso de un robot móvil que sólo se mueve en el plano se tienen tres grados de libertad: dos coordenadas de posición (x, y) y una orientación θ. Los espacios de configuración pueden describirse empleando variedades y los grados de libertad definen la *forma* de dicha variedad (Katrakazas, Quddus, Chen, y Deka, 2015).

Otra característica importante son las restricciones de movimiento. Si el robot se puede mover en cualquier dirección en el espacio de configuraciones (en ausencia de obstáculos) se habla de un robot omnidireccional. Si existen restricciones de velocidad, como en el caso de un automóvil, que sólo se puede desplazar en una dirección determinada por la orientación de las llantas delanteras, entonces se habla de un robot con restricciones *no holonómicas*. Es importante aclarar que una restricción es no holonómica cuando sólo se puede representar en términos de velocidades pero no de posición. Por ejemplo, un robot diferencial sólo se puede mover con una velocidad perpendicular al eje que une las llantas, sin embargo, con la planeación adecuada, es posible alcanzar cualquier configuración(LaValle, 2006). El caso contrario es la restricción de movimiento al plano XY que se puede expresar como $\dot{z} = 0$ (en términos de la velocidad) o como $z = Constante$ (en términos de la posición).

Finalmente, es importante tomar en cuenta si el modelo del robot será dinámico o solamente cinemático. En el primer caso las señales de entrada pueden ser fuerzas o pares y en el segundo, se asume que las velocidades se

pueden manipular arbitrariamente y por lo tanto pueden considerarse como señales de entrada.

En este trabajo se emplearon únicamente modelos cinemáticos para el diseño de las leyes de control, sin embargo, dado que se utilizó un control realimentado, esta aproximación es suficiente para lograr que el robot se mueva sobre la ruta deseada. Se emplearon tanto modelos holonómicos como no holonómicos. El robot Justina (ver sección 7.1) tiene una base omnidireccional y para el problema de la localización fue necesario el modelo sin restricciones no holonómicas, sin embargo, las leyes de control se diseñaron considerando sólo un robot diferencial. En la sección 3.1 se explica el porqué de estas consideraciones.

2.2.2. Características de los algoritmos

Una vez que se ha definido la tarea a realizar y con base en las características del robot, puede elegirse un algoritmo para resolver determinado problema. Si lo que nos interesa es que el robot ejecute una tarea en un tiempo mínimo, con el menor gasto de energía posible o recorriendo la distancia más corta, entonces se requiere de un algoritmo *óptimo* (Paden, Čáp, Yong, Yershov, y Frazzoli, 2016).

El *costo computacional* de un algoritmo se refiere a la cantidad de recursos necesarios para resolverlo, es decir, cantidad de memoria y tiempo de ejecución. La complejidad se expresa como función de la cantidad de datos de entrada y se suele clasificar como exponencial, polinomial, logarítmica, factorial, etc, dependiendo de la función que pueda acotar ya sea el peor caso o un promedio de los distintos casos. Los datos de entrada pueden ser el número de grados de libertad o el número de estados en un planeador, por ejemplo .

Otra característica importante de los algoritmos es si son *completos* o no. Se dice que un algoritmo es completo si garantiza encontrar la solución al problema si es que ésta existe. Algunas veces, para disminuir la complejidad de un algoritmo se maneja una completitud para una resolución dada, es decir, el algoritmo garantiza encontrar una solución si se maneja un cierto nivel de discretización (LaValle, Branicky, y Lindemann, 2004). Un ejemplo de esto se verá en la sección 3.3, en el que una ruta se puede calcular sólo si el espacio navegable se discretiza.

Una última característica que está más relacionada con la implementación del planeador que con los algoritmos en general, es aquella que se refiere al

momento en que se realiza la planeación. Si el planeador construye un plan previo a la ejecución se dice que éste es *fuera de línea*. Si el planeador actualiza el plan conforme lo ejecuta, entonces se dice que es *en línea* (Choset et al., 2005). El sistema de navegación propuesto incluye un planeador en línea, pues, como se explica en el capítulo 5, la ruta se puede modificar durante la ejecución si es que se detecta un obstáculo.

2.3. La arquitectura ViRBot

Para desarrollar robots de servicio doméstico, se requiere tanto de un hardware mínimo como también de una arquitectura a seguir para integrar la gran cantidad de software necesario para operar este tipo de robots autónomos (Kortenkamp y Simmons, 2008). Para realizar las tareas requeridas, un robot de servicio doméstico debe tener desde controles de bajo nivel, para mover diferentes actuadores de acuerdo con un valor deseado, hasta planeación de tareas de alto nivel, para entender, planear y ejecutar comandos dados por los humanos. Diseñar robots de servicio requiere de mucho más que sólo fusionar y coordinar tecnologías de tendencia actual en aprendizaje de máquina, visión computacional, control, navegación autónoma e interacción humano-máquina. Un robot de este tipo interactúa con un ambiente siempre cambiante en el que las condiciones "óptimas" de un laboratorio de investigación casi nunca se dan. Por estas razones, se vuelve obligatorio seguir una arquitectura de diseño que guíe el desarrollo e integración de todo el software requerido.

ViRBot es una arquitectura para diseñar, organizar, integrar y probar software para robots de servicio autónomos (Savage et al., 2008). En esta arquitectura, la operación de un robot de servicio está dividida en cuatro capas generales: entrada, planeación, manejo de conocimiento y ejecución, cada una de las cuales consta a su vez de varios subsistemas (ver figura 2.4). Cada subsistema tiene una función específica que contribuye a la operación final del robot. Esta arquitectura tiene características similares a aquellas presentadas en la arquitectura del agente *Interrap* (Müller, 1996). En la figura 2.4 se muestran los módulos involucrados en el sistema de navegación propuesto.

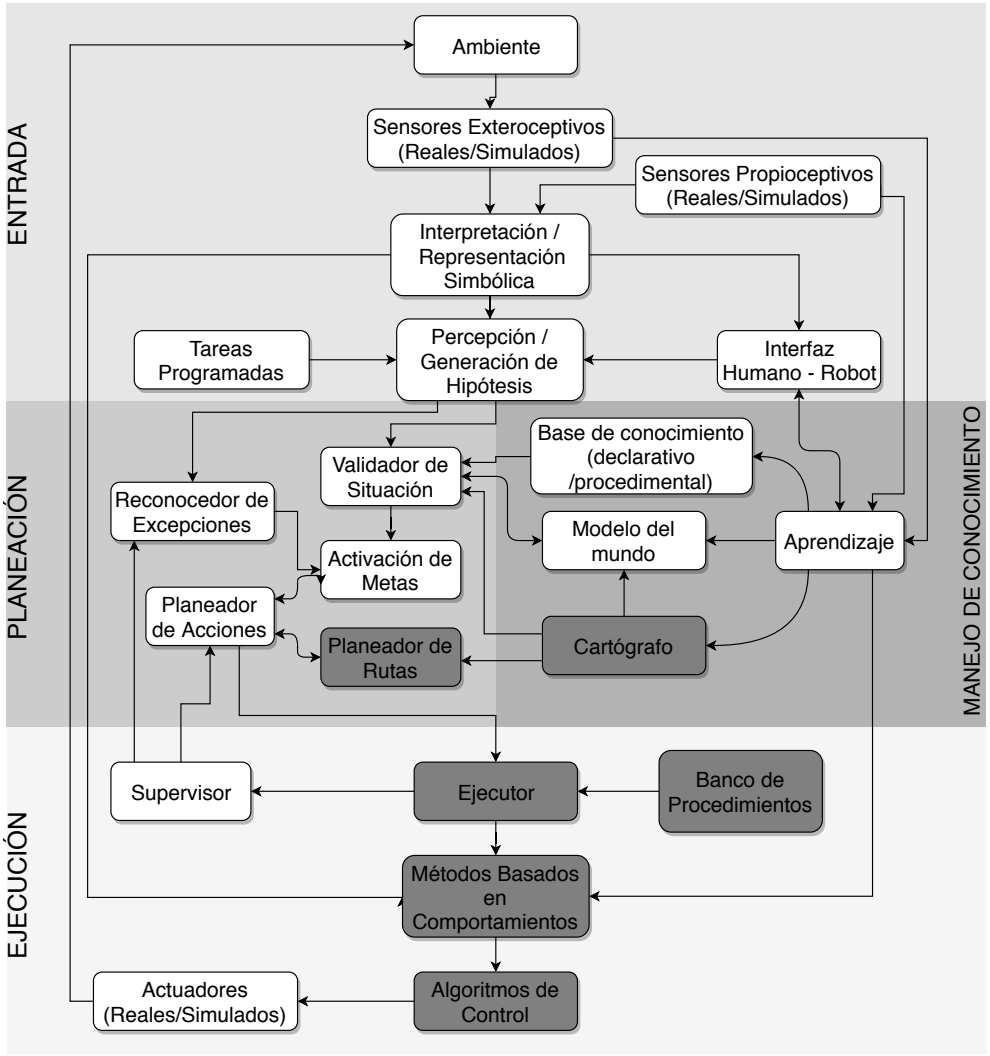


Figura 2.4: La arquitectura ViRBot: varias capas y subsistemas se integran para operar robots de servicio. Los módulos resaltados son aquellos involucrados en esta propuesta.

Capa de entrada

Esta capa comprende todos los sensores, propioceptivos y exteroceptivos. Cada sensor tiene también un modo de simulación de modo que es posible usar un robot virtual para probar todos los algoritmos usando datos simulados. Esta capa también incluye la información proveniente del sistema de interacción humano-robot, que puede consistir en los comandos de voz o gestos reconocidos. Diferentes técnicas de procesamiento digital de señales se aplican a los datos de todos los sensores para obtener representaciones simbólicas que después son usadas para crear un conjunto de creencias. Además de esto, la capa de entrada incluye un conjunto de tareas previamente programadas, debido a que éstas fijan un conjunto de metas que son usadas para planear acciones.

Capa de planeación

En cooperación con la capa de manejo del conocimiento, las creencias generadas en la capa de entrada se validan en esta capa y se crea un reconocimiento de situación. Dado dicho reconocimiento, se activan un conjunto de metas. El planeador de acciones se encarga de encontrar la secuencia de operaciones físicas necesarias para lograr las metas fijadas. El módulo de planeación de movimientos, como su nombre lo indica, se encarga de calcular los movimientos que los actuadores (base móvil y manipuladores) deben realizar para llevar a cabo la secuencia de operaciones generada por el planeador de acciones. Si durante la ejecución de un plan sucede algo no considerado, el reconocedor de excepciones tratará de resolver el problema fijando nuevas metas y replaneando la secuencia de operaciones.

Capa de manejo del conocimiento

Esta capa construye y maneja el conocimiento del robot tanto declarativo como procedimental. El cartógrafo está a cargo de tres tareas: mantener una representación del ambiente usando mapas geométricos y mapas de rutas, construir dicha representación usando SLAM (siglas en inglés de localización y mapeo simultáneos) y algoritmos de agrupamiento, y localizar al robot usando Filtros de Kalman y Localización de Monte Carlo Adaptable. Esta capa también incluye una base de conocimiento tanto procedimental como declarativo. El primero consiste en un sistema experto que utiliza un sistema

basado en reglas escrito en lenguaje CLIPS (un lenguaje lógico desarrollado por la NASA). El conocimiento declarativo comprende patrones de objetos, rostros, localidades predefinidas, regiones en el ambiente y posiciones de personas y objetos. Diversos algoritmos de aprendizaje son utilizados para entrenar dichos patrones y también para generar nuevos comportamientos.

Capa de ejecución

El primer módulo de esta capa es el ejecutor. Éste toma un conjunto de procedimientos predefinidos para transformar los planes generados por el módulo de planeación de acciones en secuencias de acciones simples. El ejecutor informa el resultado de la ejecución de cada secuencia predeterminada al supervisor, que revisa que todas las acciones y movimientos se lleven a cabo de acuerdo con el plan. El banco de procedimientos es un conjunto de máquinas de estados finitas aumentadas usadas para resolver parcialmente problemas específicos tales como tomar un objetos, preguntar y memorizar un rostro, entre otros similares. Una vez que el ejecutor genera secuencias de acciones simples, un conjunto de comportamientos se usa para hacer frente a problemas no previstos por el planeador de acciones, como la evasión de un obstáculo desconocido. Los comportamientos generan señales que son tomadas como valores deseados por los algoritmos de control de bajo nivel y, finalmente, las salidas de los controladores son usadas para mover los actuadores.

El ViRBot y el sistema propuesto

La arquitectura ViRBot es una abstracción de las tareas e interacciones entre los distintos componentes del software usado para operar un robot de servicio. Un programa corriendo en el robot puede realizar más de una tarea de la arquitectura ViRBot y del mismo modo, un módulo del ViRBot podría estar implementado en más de un programa. El sistema de navegación propuesto es parte del planeador de rutas, el cartógrafo, el banco de procedimientos, los comportamientos y los algoritmos de control. La figura 2.4 muestra, resaltados en gris oscuro, aquellos subsistemas involucrados en esta propuesta. En los siguientes capítulos se describe cada módulo del sistema propuesto y su correspondiente parte en la arquitectura ViRBot.

2.4. Trabajo relacionado

2.4.1. Arquitecturas para robots de servicio

El ViRBot es un arquitectura que proporciona lineamientos para el desarrollo y organización del software para la operación de robots de servicio doméstico. Sin embargo, existen otras propuestas sobre la forma en que puede abordarse el desarrollo de robots de servicio. En L. A. Pineda, Rodríguez, Fuentes, Rascon, y Meza (2015) se propone un análisis sobre robots de servicio con base en tres niveles: especificación funcional, nivel de dispositivos y algoritmos y nivel de implementación. Estos tres niveles son análogos a aquellos propuestos por David Marr para la visión computacional: nivel de teoría computacional, nivel de representación y algoritmos y nivel de implementación. En L. Pineda et al. (2017) se hace una revisión de los trabajos hechos en estos tres niveles en tres robots: Justina (Savage et al., 2019), Markovito (Avilés-Arriaga et al., 2009) y Golem (L. A. Pineda et al., 2015). En este marco conceptual, la presente propuesta formaría parte de los niveles de dispositivos y algoritmos y de implementación. Los algoritmos de control, planeación de rutas y localización de esta propuesta forman parte del nivel de dispositivos y algoritmos. Estos, a su vez, proporcionan las herramientas necesarias para que en el robot Justina (ver sección 7.1) el nivel funcional se pueda abordar utilizando sistemas expertos para el razonamiento semántico (Savage et al., 2019). El nivel de implementación se refiere a las plataformas de hardware así como a los programas de apoyo como drivers de sensores y actuadores y sistemas operativos. En esta propuesta se aborda este nivel, pues se analiza el uso de programación en paralelo para aumentar la rapidez de ejecución de los algoritmos de agrupamiento utilizados en la construcción de mapas de rutas (ver sección 4.3.2).

2.4.2. Sistemas de navegación completos

La mayor parte de la literatura relacionada con la planeación de movimientos reporta problemas y experimentos aislados, tales como algoritmos para mapeo, algoritmos para planeación de rutas, localización o evasión de obstáculos; sin embargo, existen pocos trabajos que reporten sistemas de planeación de movimientos completamente implementados y probados para robots de servicio doméstico. Ejemplos de estos trabajos son los robots Markovito (Avilés-Arriaga et al., 2009), LISA (Seib, Memmesheimer, y Paulus,

2016) y Cosero (Stückler, Schwarz, y Behnke, 2016).

Avilés-Arriaga et al. (2009) proponen un sistema de navegación basado en una arquitectura de tres niveles de software: funcional, ejecución y decisión. Este sistema de planeación de movimientos usa programación dinámica para la planeación de rutas, filtros de partículas y marcas geométricas (esquinas y líneas) para localización, celdas de ocupación para la representación del ambiente e incluyen información semántica para una mejor planeación de movimientos. Este sistema está basado en una arquitectura bien definida, de forma similar al trabajo aquí presentado.

El trabajo de Stückler et al. (2016) está enfocado en el diseño y construcción de un robot de servicio cognitivo. Para construir un mapa, su sistema de planeación de movimientos utiliza mapeo y localización simultáneos con base en un filtro de partículas *Rao-Blackwellized* y la localización adaptable de Monte Carlo para la estimación de la posición. Los autores de este trabajo usan mallas 3D de *surfels* de las cuales extraen mapas de navegación en 2D analizando la capacidad de desplazamiento en el mapa. El diseño de su software también está basado en una arquitectura con cuatro capas: tareas, subtareas, acción y percepción, y sensoriomotriz. Contrario a la arquitectura usada en este trabajo, los autores no separan la planeación de tareas del manejo de conocimiento.

Seib et al. (2016) desarrollaron un robot de servicio autónomo basado en ROS llamado LISA. Su sistema de planeación de movimientos también usa filtros de partículas para localización y mapeo. Para la planeación de rutas emplean el algoritmo A* y celdas de ocupación y, de forma similar a este trabajo, las rutas calculadas son postprocesadas para obtener rutas más suaves. Para el seguimiento de la ruta suavizada, los autores emplean puntos de referencia a lo largo de dicha ruta. Como principal beneficio se menciona la estrecha integración con la interfaz gráfica de usuario. Contrario a este trabajo, los autores no mencionan el uso de comportamientos reactivos para lidiar con obstáculos no previstos.

2.4.3. Localización y mapeo

Para cumplir sus tareas, los robots de servicio necesitan una representación del ambiente (un mapa ya sea geométrico, topológico o semántico) y su configuración actual dentro de ese ambiente. El problema de construir un mapa mientras se estima la posición del robot al mismo tiempo se conoce como SLAM (siglas en inglés de localización y mapeo simultáneos) y

es comúnmente resuelto empleando Filtro de Kalman (M. G. Dissanayake, Newman, Clark, Durrant-Whyte, y Csorba, 2001) o filtros de partículas (Grisetti, Stachniss, y Burgard, 2007). Trabajos más recientes han abordado este problema empleando cámaras y técnicas de compresión de datos,por ejemplo, Contreras y Mayol-Cuevas (2017) y Contreras y Mayol-Cuevas (2015). Para resolver sólo el problema de la localización, una de las técnicas más usadas en el desarrollo de robots de servicio es la Localización Adaptativa de Monte Carlo (AMCL por sus siglas en inglés) (Thrun, Burgard, y Fox, 2005).

Los mapas de rutas son un tipo de mapas topológicos útiles para la navegación de robots móviles en ambientes estructurados. Existen varias técnicas para la construcción de este tipo de mapas cuando se dispone de una representación geométrica de los objetos en el ambiente, por ejemplo, diagramas de Voronoi (Latombe, 1991), mapas de visibilidad (Lozano-Pérez y Wesley, 1979) o métodos probabilísticos (Kavraki, Svestka, Latombe, y Overmars, 1996). En este trabajo se describe cómo se construyen mapas de rutas usando información adquirida con cámaras RGB-D y técnicas de cuantización vectorial. Un trabajo similar se describen en Savage et al. (2016). La cuantización vectorial tiene un costo computacional muy alto y el procesamiento puede ser muy lento. En esta propuesta, el tiempo de ejecución se redujo significativamente mediante la implementación en paralelo de dichos algoritmos.

2.4.4. Evasión de obstáculos

Los campos potenciales artificiales (Khatib, 1986) son una técnica para planeación de rutas y evasión de obstáculos. Se basan en la construcción de una función potencial con un mínimo global en el punto meta y máximos locales en los objetos que se desean evadir. Si el robot se mueve en sentido contrario al gradiente de dicha función, el robot puede alcanzar el punto meta y evadir los obstáculos al mismo tiempo. Tienen la desventaja de presentar mínimos locales, lo que los hace un algoritmo no completo para planeación de rutas, sin embargo, como se explicará en la sección 5.2, esto se puede solucionar empleando técnicas de la robótica basada en comportamientos.

Los mapas de costo son también una técnica comúnmente usada para la evasión de obstáculos, por ejemplo, el trabajo de Lu, Hershberger, y Smart (2014) usa un esquema de mapas de costo en capas para planear rutas en ambientes dinámicos. Sin embargo, un mapa de costo requiere de una representación geométrica del ambiente, mientras que los campos potenciales

pueden ser puramente reactivos. Como se discutirá más adelante, los mapas de costos resultan poco efectivos cuando los robots tienen que navegar en espacios reducidos. En Lefèvre, Vasquez, y Laugier (2014) se hace una revisión de diversos métodos para evasión de obstáculos y detección de riesgo de colisión en navegación autónoma.

2.4.5. El uso de modelos psicológicos

El uso de modelos psicológicos se remonta al trabajo de Braitenberg (1986) donde se propone construir robots inteligentes con base en la interconexión de sensores y actuadores para dar al robot comportamientos tales como agresión o previsión. Un ejemplo más reciente es el trabajo de Karpov (2014), quien desarrolló robots móviles capaces de interactuar con el ambiente de una forma inteligente empleando la Teoría de Simonov de las Emociones. El autor utiliza el componente temperamental de este modelo para fijar un balance entre inhibición y excitación en el conjunto de comportamientos del robot. El resultado de este trabajo mostró que la emulación de emociones puede ser útil para los robot en ambientes complejos con características desconocidas. También hay trabajos en los que se mejora el desempeño del robot mediante un enfoque neurofisiológico. Un ejemplo es el trabajo de Zeno, Patel, y Sobh (2017), quienes desarrollaron un sistema de navegación emulando la dinámica de las células de navegación y percepción del espacio de un roedor.

Kurup y Lebiere (2012) explican por qué los robots autónomos modernos deberían utilizar arquitecturas cognitivas (con modelos de atención, memoria y percepción desarrollados en el campo de la psicología) para mejorar su desempeño, sin embargo, hasta el conocimiento del autor de este trabajo, no hay trabajos en los que dichos modelos psicológicos sean aplicados a robots de servicio doméstico. En esta propuesta se describe la utilización de algunos hallazgos sobre navegación en animales para mejorar el desempeño del sistema de planeación de movimientos.

2.4.6. El paquete *nav2d* de ROS

La plataforma ROS (siglas en inglés del Sistema Operativo para Robots) es un sistema de código abierto para comunicar procesos que provee la funcionalidad que comúnmente se necesita en el desarrollo de software para robots

móviles autónomos, tales como paso de mensajes y manejo de paquetes (Quigley, Gerkey, y Smart, 2015). ROS también proporciona varios paquetes con algoritmos para implementar las habilidades más comunes requeridas en los robots autónomos: percepción de objetos y personas, representación del conocimiento y navegación. *nav2d* es un paquete de código abierto para navegación en 2D (`http://wiki.ros.org/nav2d`) cuyas principales características, de acuerdo con su página web, son la evasión de obstáculos puramente reactiva, un planeador de rutas simple y un SLAM basado en grafos que permite a múltiples robots construir un mapa de manera cooperativa. Este paquete fue usado para comparar el desempeño del sistema propuesto.

Capítulo 3

Planeación de Rutas

En este capítulo se explica cómo en el sistema propuesto se planean y ejecutan rutas. En la arquitectura ViRBot, una vez que se ha determinado una secuencia de acciones para realizar una tarea, el planeador de rutas se encarga de calcular los movimientos necesarios. Se comienza por describir, en la sección 3.1, el control de bajo nivel, consistente en un control no lineal diseñado para una base diferencial. En las secciones 3.2 y 3.3 se explica la forma en que, con base en una representación del ambiente y algoritmos de búsqueda en grafos, se planea la ruta en sí. Finalmente, en la sección 3.4 se describe una máquina de estados empleada para lograr un perfil de velocidad y obtener así un movimiento más suave.

3.1. Control de bajo nivel

En el resto de este capítulo y en los siguientes se explica cómo el sistema propuesto representa el ambiente, planea una ruta y emplea comportamientos para evadir obstáculos, sin embargo, todos estos algoritmos requieren de leyes de control que garanticen que todos los movimientos planeados sean realizados correctamente. Para ese propósito, se emplearon leyes de control no lineales.

Considere una base omnidireccional como la que se muestra en la figura 3.1, donde la configuración está determinada por tres variables $[x, y, \theta]$ que corresponden a la pose del robot en el plano. Si sólo se considera la parte cinemática y suponiendo que no hay deslizamiento, el modelo del robot está dado por:

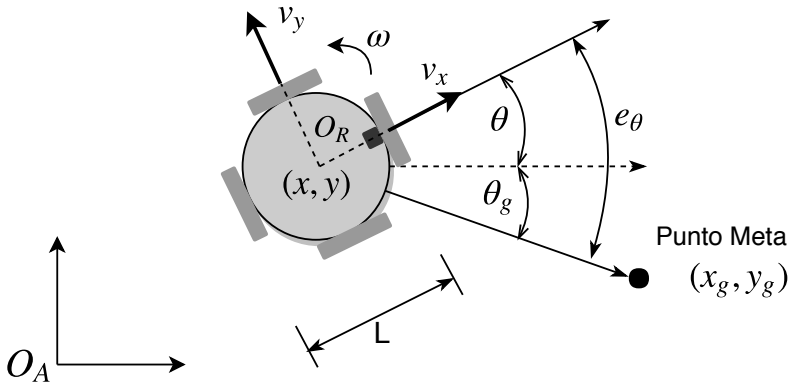


Figura 3.1: Base móvil omnidireccional con diámetro L. Se considera que (x, y, θ) es la pose del robot y (v_x, v_y, ω) son las señales de entrada. Se desea alcanzar el punto (x_g, y_g).

$$\dot{x} = v_x \cos\theta - v_y \sin\theta \tag{3.1}$$
$$\dot{y} = v_x \sin\theta + v_y \cos\theta \tag{3.2}$$
$$\dot{\theta} = \omega, \tag{3.3}$$

donde v_x y v_y son las componentes de la velocidad lineal con respecto al sistema O_R (sistema de referencia que se mueve junto con el robot), es decir, son las velocidades "frontal" y "lateral" del robot; ω es la velocidad angular. v_x, v_y y ω son consideradas como señales de entrada.

En este modelo la parte dinámica no se toma en cuenta, es decir, la configuración $[x, y, \theta]$ se considera también como el vector de estados. Se asume también que las señales v_x, v_y y ω se pueden fijar arbitrariamente. Esto no sucede en el sistema real, donde la señal de entrada es el voltaje que se fija en las terminales de los motores, sin embargo, se puede suponer que las respuestas mecánica y eléctrica de los cuatro motores son lo suficientemente rápidas y por lo tanto, se puede fijar una velocidad angular deseada en cada motor en un tiempo muy corto.

El objetivo de las leyes de control es determinar las señales v_x, v_y y ω de modo que se pueda garantizar que el robot alcanzará la posición deseada (x_g, y_g) incluso ante la presencia de incertidumbres (tales como las dinámicas no modeladas) y perturbaciones. Aunque la base móvil es omnidireccional, en este trabajo los movimientos se hacen considerando una base diferencial. Esto

es debido a que la mayoría de los sensores están al frente del robot, por lo que resulta útil que el frente siempre apunte hacia la dirección de movimiento. En el robot de servicio Justina (ver sección 7.1), la omnidireccionalidad se utiliza en tareas donde se requieren movimientos laterales finos, por ejemplo, si al intentar tomar un objeto éste se encuentra fuera del espacio de trabajo del manipulador. Estas situaciones están fuera del alcance de este trabajo.

Considere el esquema mostrado en la figura 3.1. Sea θ_g el ángulo deseado, que corresponde al ángulo del vector de error de posición $[x_g - x, y_g - y]$, calculado como

$$\theta_g = \text{atan2}\left(y_g - y, x_g - x\right),$$

de donde se puede calcular el error de ángulo

$$e_\theta = \theta_g - \theta = \text{atan2}\left(y_g - y, x_g - x\right) - \theta, \tag{3.4}$$

Es importante notar que e_θ, al igual que cualquier otra medida angular, debe estar siempre en el intervalo $(-\pi, \pi]$. Si la diferencia dada por (3.4) resulta en un ángulo mayor que π o menor que $-\pi$, éste de corregirse para lograr el comportamiento deseado y que las leyes de control funcionen correctamente.

Para modelar un robot omnidireccional como uno diferencial, basta con hacer $v_y = 0$. Suponiendo que se tiene una base móvil cuyo modelo está dado por (3.1)-(3.3), entonces las leyes de control

$$v_x = v_{max} e^{-\frac{e_\theta^2}{\alpha}} \tag{3.5}$$

$$\omega = \omega_{max}\left(\frac{2}{1 + e^{-\frac{e_\theta}{\beta}}} - 1\right) \tag{3.6}$$

con $v_{max} > 0$, $\omega_{max} > 0$, $\alpha > 0$ y $\beta > 0$, garantizan que el robot alcanzará la posición deseada $(x_g, y_g)^T$. Dadas las señales de control v_x, v_y y ω, las cuatro velocidades de los motores (izquierdo v_{xl}, derecho v_{xr}, frontal v_{yf} y trasero v_{yr}) se pueden calcular como:

$$\begin{aligned} v_{xl} &= v_x - \frac{L}{2}\omega, \qquad v_{yf} = v_y + \frac{L}{2}\omega \\ v_{xr} &= v_x + \frac{L}{2}\omega, \qquad v_{yr} = v_y - \frac{L}{2}\omega \end{aligned}$$

donde L es el diámetro del robot (ver figura 3.1).

Las leyes de control (3.5)-(3.6) tienen cuatro parámetros de diseño. v_{max} y ω_{max} corresponden a las velocidades lineal y angular, respectivamente, que el robot puede alcanzar durante el movimiento. Ambos parámetros, en principio, se pueden fijar arbitrariamente, sin embargo, en una implementación real, están limitados por las capacidades de los motores.

Para una mejor comprensión de los parámetros α y β, considere la figura 3.2. Se puede observar que la constante α determina qué tan rápido decrece la velocidad lineal v cuando el error de ángulo e_θ crece. Una α pequeña hará que la velocidad lineal decrezca muy rápido, es decir, el robot no avanzará sino hasta que esté "apuntando" directamente hacia la posición deseada; en otras palabras, primero girará hasta que el error de ángulo sea muy pequeño y entonces se moverá hacia el punto meta. La constante β determina qué tan rápido se incrementa la velocidad angular ω cuando el error de ángulo crece. En general, una β pequeña hará que el robot se mantenga siempre "apuntando" hacia la posición deseada, es decir, se tendrá un mejor seguimiento de rutas, sin embargo, si β es demasiado pequeña, se pueden presentar oscilaciones no deseadas en el movimiento. En Cruz (2013) se describe un método basado en algoritmos genéticos que puede usarse para sintonizar estas constantes. Los valores usados en este trabajo se describen en el capítulo 7.

3.2. Representación del ambiente

El primer paso para resolver la tarea de navegación es la planeación de una ruta con base en un mapa construido previamente. Como se mencionó en la sección 2.2, un mapa es una representación del ambiente que contiene información espacial útil para tomar decisiones. En este trabajo se emplearon mapas basados en celdas de ocupación y mapas de rutas. Los primeros son creados utilizando el paquete abierto *gmapping* que se incluye como parte de la plataforma ROS (ver sección 7.2) y los mapas de rutas se construyen como se explica en la sección 4.3.

Los mapas basados en celdas de ocupación fueron originalmente propuestos por Elfes (1989) y consisten en una discretización de una porción acotada del espacio continuo en celdas rectangulares que pueden estar o no ocupadas. A cada celda se le asigna un número que representa la probabilidad de que dicha celda esté ocupada. La tarea principal de un algoritmo que cree un mapa de celdas de ocupación es determinar la probabilidad posterior de que

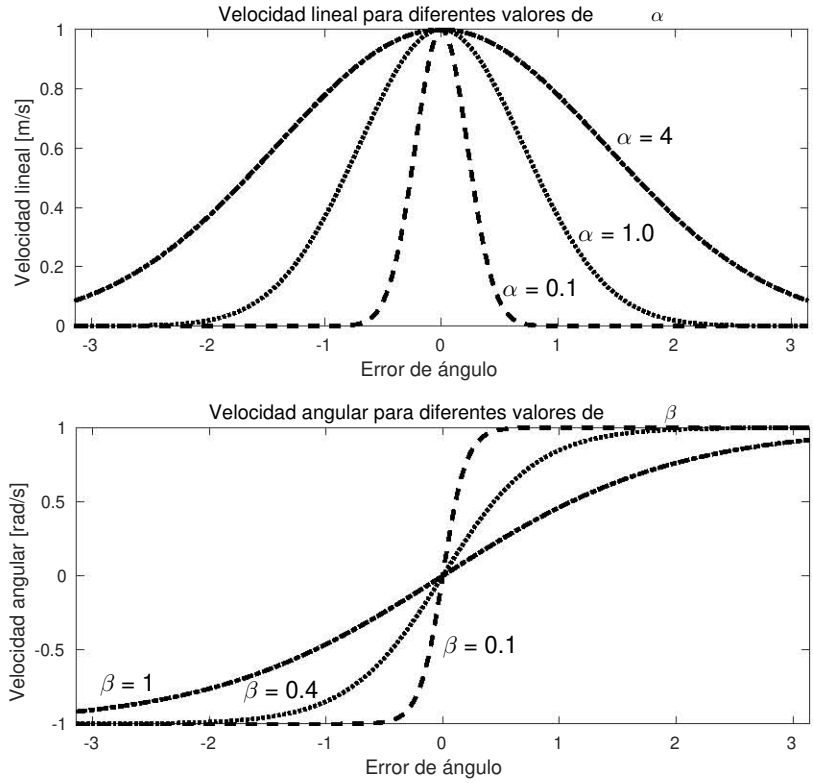


Figura 3.2: Velocidades lineal y angular para diferentes valores de α y β

una celda esté ocupada dado un conjunto de lecturas y una ruta recorrida por el robot (Thrun et al., 2005).

Las principales ventajas de usar mapas de celdas de ocupación son que su construcción no depende de ningún tipo de características predefinidas, se tiene acceso en tiempo constante a la información de todas las celdas y la capacidad de representar áreas desconocidas. Las principales desventajas son los posibles errores de discretización y que, en general, requieren de mucha memoria (Burgard, Hebert, y Bennewitz, 2016). En la figura 3.3 se muestra un ejemplo de mapa de celdas de ocupación.

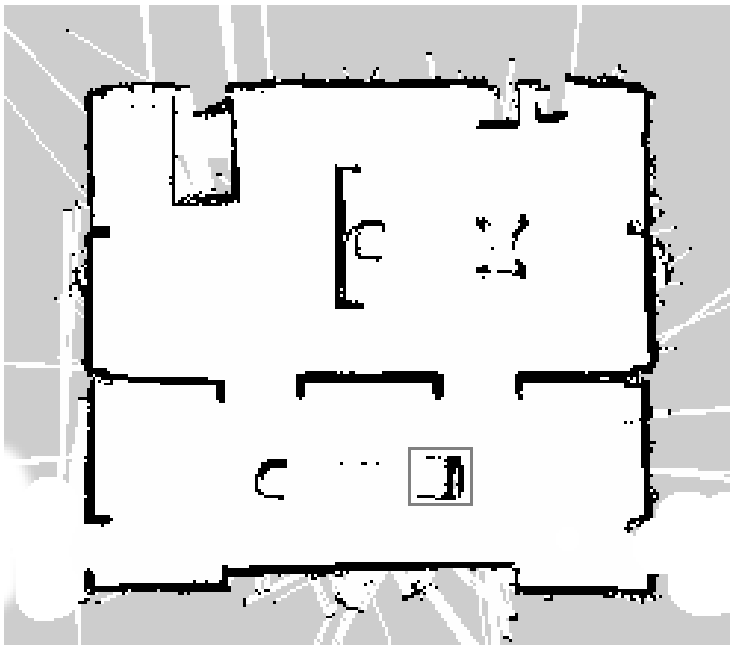
Por otra parte, los mapas de rutas son un tipo de representación en la que se mapean las propiedades de conectividad del espacio libre. Estos mapas de rutas tienen la forma de un grafo donde cada vértice corresponde a un punto en el espacio de configuración y cada borde es una ruta (Kavraki y LaValle, 2008). Para que un grafo G sea considerado como mapa de rutas debe satisfacer dos propiedades: accesibilidad y conectividad. La accesibilidad se refiere a que, para cualesquiera puntos q_s (inicial) y q_g (meta), ambos en el espacio libre C_{free}, si existe una ruta que conecte q_s y q_g, entonces deben existir rutas que conecten a q_s con algún nodo $q_s\prime \in G$ y a q_g con algún nodo $q_g\prime \in G$. La conectividad se refiere a que la existencia de una ruta que conecte a q_s y q_g implica la existencia de una ruta que conecte a $q_s\prime$ y $q_g\prime$.

En la sección 4.3.2 se describe el método propuesto para construir mapas de rutas a partir de imágenes capturadas con cámaras RGB-D. La implementación en paralelo de esta construcción permite su uso como método para evasión de obstáculos.

3.3. Planeación de rutas mediante A*

En la arquitectura ViRBot, la capa de planeación incluye al planeador de rutas. Este módulo toma información del planeador de acciones y de la representación del ambiente que mantiene el cartógrafo para calcular rutas seguras.Si el ambiente está representado con celdas de ocupación, el problema de la planeación de rutas se puede resolver aplicando algoritmos de búsqueda en grafos. En este caso, cada celda representa un nodo en el grafo y se considera que está conectado sólo con las celdas vecinas que pertenecen al espacio libre. Para determinar los nodos vecinos, se puede usar conectividad cuatro u ocho.

A* es un algoritmo de búsqueda que explora la ruta con el menor costo



(a)

(b)

Figura 3.3: Ejemplos de celdas de ocupación. **a)** A pesar de ser una discretización, con una buena resolución la representación puede ser bastante exacta. **b)** Parte del mapa ampliada donde se aprecia la discretización: las partes blancas son celdas libres, las negras, ocupadas y las grises, aquellas sin información.

esperado. Es muy similar al algoritmo de Dijkstra y se diferencia de éste por la implementación de una heurística para elegir el siguiente nodo a expandir, lo hace más rápido. El aspecto más importante es el diseño de la función de costo $g(n)$, que define el peso de cada nodo (González, Pérez, Milanés, y Nashashibi, 2016). Para un determinado nodo n, el costo esperado $f(n)$ se calcula como

$$f(n) = g(n) + h(n)$$

donde $g(n)$ es el costo de la ruta desde el nodo inicial hasta el nodo n, y $h(n)$ es una función heurística que determina *un costo* que se esperaría obtener desde el nodo n hasta el nodo meta. Una condición necesaria es que el costo esperado $h(n)$ debe subestimar el costo real, es decir, se debe cumplir que

$$h(n) \leq g(n) \quad \forall\, n \in \mathit{Graph}$$

3.3.1. Función de costo para rutas más seguras

El algoritmo A* encuentra una ruta que es óptima con respecto a la función de costo $g(n)$. Esta función debe diseñarse de acuerdo con aquello que sea necesario en la navegación del robot: recorrer la distancia más corta, la más rápida o la más eficiente energéticamente, por mencionar algunos ejemplos. En esta propuesta, el riesgo de colisionar con los obstáculos del mapa se toma como parte de dicha función de costo, de modo que el robot siempre navegará a través de las rutas más seguras. Para lograr esto, se define la función de costo como

$$g(n) = d(n) + r(n), \tag{3.7}$$

donde $d(n)$ es la distancia acumulada desde el nodo inicio hasta el nodo actual n y $r(n)$ es un valor inversamente proporcional a la distancia desde el nodo n al obstáculo más cercano. Dado que se tiene una representación mediante celdas de ocupación, a cada celda se le puede asignar un valor (además del nivel de ocupación) que indique la cercanía al obstáculo más cercano. De este modo, el cálculo de cercanías se puede hacer fuera de línea en una etapa de preprocesamiento.

La figura 3.4 muestra una comparación entre las rutas calculadas con y sin la componente $r(n)$ de la función de costo; es decir, las rutas resultantes son, en el primer caso, las más seguras y en el segundo, las más cortas. Como se puede observar en la parte de arriba de la figura, si sólo se toma en cuenta

la distancia como función de costo, la ruta calculada tiene posiciones junto a las paredes, dado que esta es la distancia más corta. La parte de abajo de la figura muestra la ruta considerando la cercanía a los obstáculos como parte de la función de costo y, como puede verse, las rutas calculadas son más seguras. En el capítulo 7 se explican las pruebas que se realizaron para evaluar la seguridad de las rutas calculadas.

Resultados similares se pueden obtener si primero se aumenta el tamaño de los obstáculos en el mapa, sin embargo, con este enfoque el planeador puede llegar a fallar al calcular rutas a través de espacios reducidos. Los mapas de costo son un enfoque similar pero, como se mostrará en la sección de resultados, tienen el mismo problema de no poder calcular rutas a través de espacios reducidos.

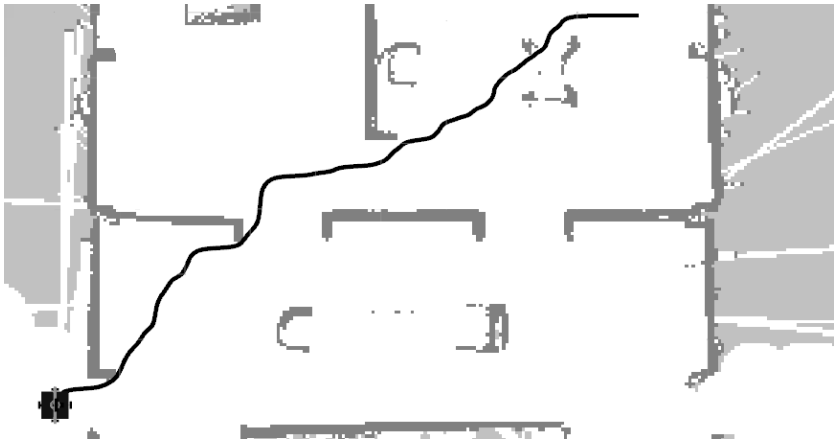
La función heurística usada en este trabajo es la distancia de Manhattan desde la celda actual hasta la celda meta. Después de que se planea la ruta, ésta se procesa para obtener una ruta más suave que permita tener leyes de control también suaves y así evitar daños a los motores de la base móvil.

3.3.2. Suavizado de la ruta por descenso del gradiente

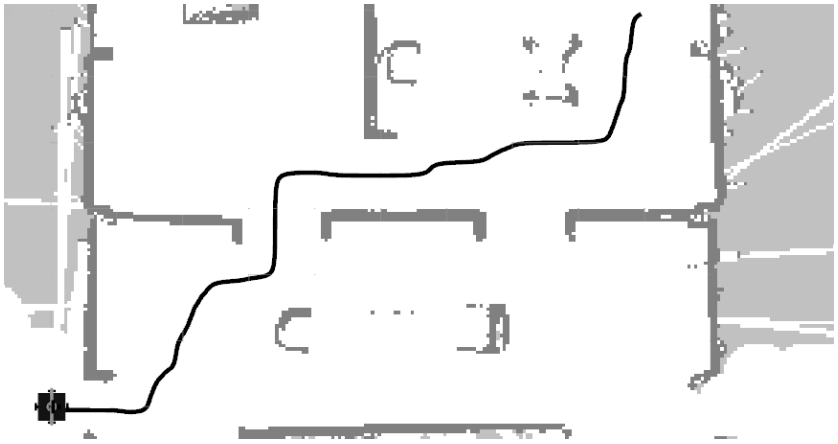
Dado que la ruta que arroja A* se calcula a partir de celdas de ocupación, las vueltas siempre serán ángulos rectos, en caso de que se haya utilizado conectividad cuatro, o bien vueltas a $\pi/8$, en caso de que se haya utilizado conectividad ocho. En cualquier caso, no es deseable tener "esquinas" en las rutas por varias razones: la primera de ellas es que la función de posición no es diferenciable en dichas esquinas y, además, este tipo de vueltas ocasionan cambios bruscos en las señales de control, lo que finalmente ocasiona daños a los actuadores del robot. Por otro lado, las restricciones de movimiento en algunos tipos de bases (como es el caso de los automóviles) pueden impedir ejecutar vueltas en ángulo recto.

Por lo anterior, es conveniente suavizar la ruta calculada por A* de modo que la nueva ruta sea lo suficientemente parecida a la original, pero al mismo tiempo, lo suficientemente suave para evitar curvas muy pronunciadas. La figura 3.5 muestra un ejemplo de suavizado con dos casos extremos: la ruta de la figura 3.5a es una ruta igual a la original y la figura 3.5b es una ruta sin vueltas pero que ha dejado de ser una ruta útil. La figura 3.5c es un *promedio ponderado* de las dos anteriores.

Para el suavizado de las rutas se han propuesto diversas soluciones. La más simple es combinar líneas rectas y curvas, sin embargo, el cambio brusco



(a)



(b)

Figura 3.4: Comparación de rutas. **a)** Función de costo sólo con distancia. **b)** Distancia más cercanía a los obstáculos.

de curvaturas puede ocasionar sobrepasos en el control y desgaste, o incluso falla, en sistemas mecánicos (Elbanhawi, Simic, y Jazar, 2015). Las Espirales de Euler también han sido una propuesta. Su característica principal es que la curvatura se incrementa de forma lineal con respecto a la longitud del arco, sin embargo, este tipo de curvas requieren ser aproximadas con polinomios de alto orden, lo que dificulta su uso en aplicaciones en tiempo real (McCrae y Singh, 2009). Las curvas de Bézier son una propuesta comúnmente usada en el área de diseño asistido por computadora, sin embargo, éstas tienen la desventaja de que el orden de la curva aumenta conforme aumenta el número de puntos de control. Si se desea mantener un orden bajo, es necesario conectar varias curvas, sin embargo, esto trae nuevamente el problema de discontinuidades en la curvatura (Elbanhawi et al., 2015).

Las curvas *spline* son un tipo de curvas definidas a intervalos por polinomios. Una curva B-spline es un tipo de spline que forma parte de un conjunto base, es decir, cualquier spline de grado n se puede expresar como una combinación lineal de B-splines del mismo orden n. Las curvas spline son otra forma de suavizar rutas y tienen la ventaja, contra las curvas de Bézier, de que su orden no aumenta conforme aumenta el número de puntos de control. Sin embargo, dada una ruta con m puntos, el número de parámetros a ajustar sí aumenta conforme aumenta m, pues la spline que suaviza la ruta total debe ser expresada como una combinación lineal de curvas B-spline Lepetič, Klančar, Škrjanc, Matko, y Potočnik (2003). Los métodos hasta aquí descritos para suavizado de rutas utilizan polimonios para aproximar las curvas. El método usado en este trabajo toma un enfoque basado en funciones de costo y, a diferencia de los métodos mencionados, sólo requiere de dos parámetros de diseño para obtener la ruta suavizada.

Para poder obtener una ruta como la de la figura 3.5c, se utiliza una función de costo que toma en cuenta el parecido con la ruta original y el *nivel de suavizado* que se desea. Estos dos criterios están en conflicto: entre más suavizado, menor será el parecido con la ruta original y viceversa, es decir, la función de costo será grande con mucho suavizado y también es muy grande si la ruta es muy parecida a la original. La idea es que dicha función de costo tenga un mínimo en un punto intermedio de modo que este punto corresponda a una ruta como la de la figura 3.5c.

Las funciones cuadráticas son una buena opción como función de costo ya que tienen un mínimo global y además son diferenciables. Para el suavizado

de la ruta se utiliza la función de costo

$$V = \frac{1}{2}\alpha \sum_{i=1}^{n} (p_i - q_i)^2 + \frac{1}{2}\beta \sum_{i=1}^{n-1} (p_i - p_{i+1})^2 \tag{3.8}$$

donde $Q = \{q_1 \dots q_n\}$ son todos los puntos $q = [x_q\, y_q]$ que forman la ruta original calculada con A* y $P = \{p_1 \dots p_n\}$ son los puntos $p = [x_p\, y_p]$ de la ruta ya suavizada. La constante $\alpha \geq 0$ es un parámetro de diseño para indicar cuánto peso se le quiere dar al parecido de la ruta original con la suavizada y $\beta \geq 0$ indica el peso que se le da la distancia entre un punto y el siguiente, lo que está relacionado con el suavizado en sí. Con $\alpha = 0$ se obtendría una línea recta, como la figura 3.5b, y con $\beta = 0$ se obtendría una ruta igual a la original, como en la figura 3.5a.

La ruta suavizada se obtiene encontrando el argumento que minimiza la función 3.8, sin embargo, dada la cantidad tan grande de variables, es difícil encontrarlo analíticamente. Una forma más sencilla es mediante el método del descenso del gradiente, que consiste en mover el argumento pequeñas cantidades proporcionales al gradiente de la función V y en sentido contrario a éste. Dado que el cambio entre cada iteración es proporcional a ∇V, se puede asumir que cuando el cambio en el argumento es menor que una tolerancia, entonces se ha alcanzado el mínimo.

La ecuación 3.8 toma como argumentos las posiciones tanto de la ruta original como de la suavizada, sin embargo, dado que sólo varían los puntos de la nueva ruta, el gradiente ∇V está dado por:

$$\Big[\underbrace{\alpha(p_1 - q_1) + \beta(p_1 - p_2)}_{\frac{\partial V}{\partial p_1}}, \dots, \underbrace{\alpha(p_i - q_i) + \beta(2p_i - p_{i-1} - p_{i+1})}_{\frac{\partial V}{\partial p_i}}, \dots, \underbrace{\alpha(p_n - q_n) + \beta(p_n - p_{n-1})}_{\frac{\partial V}{\partial p_n}}\Big] \tag{3.9}$$

Nótese que se está derivando con respecto a p (puntos de la ruta suavizada), no con respecto a q (puntos de la ruta original). Cada punto de la ruta tiene coordenadas (x, y), por lo que el algoritmo de suavizado se tiene que aplicar para ambas coordenadas.

El algoritmo 1 contiene los pasos en pseudocódigo para implementar el descenso del gradiente para el suavizado de una ruta. La constante $\delta > 0$ debe ser lo suficientemente pequeña para evitar inestabilidad en el algoritmo, sin embargo, se debe considerar que entre más pequeña sea ésta, mayor será el costo computacional. En la figura 3.5 se observa un ejemplo del resultado del suavizado con diferentes valores de α y β.

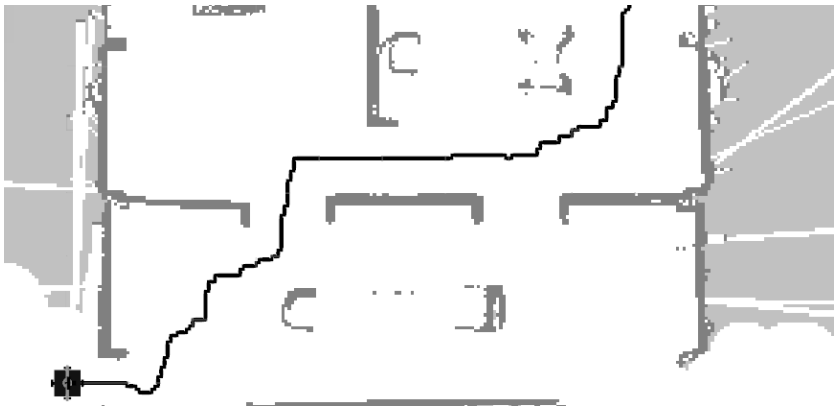
Algoritmo 1: Descenso del gradiente para suavizado de rutas.

Datos:
Conjunto de puntos $Q = \{q_1 \dots q_i \dots q_n\}$ de la ruta original
Parámetros α y β
Tolerancia ϵ
Resultado:
Conjunto de puntos $P = \{p_1 \dots p_i \dots p_n\}$ de la ruta suavizada

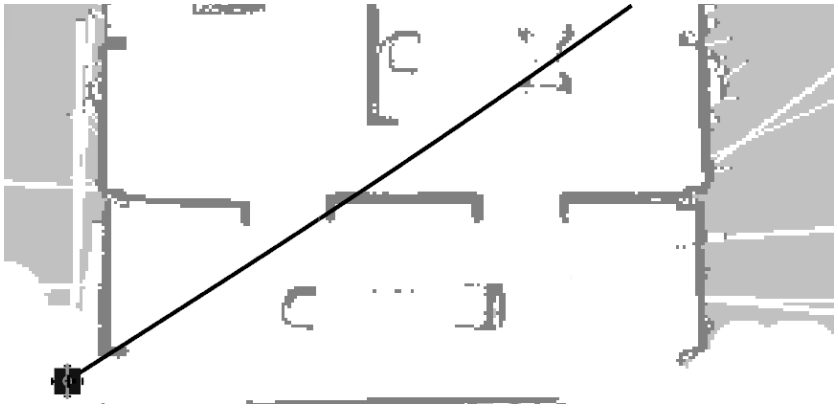
para cada $p_i \in P$ **hacer**
  $p_i \leftarrow q_i$
fin
mientras $\|\nabla V(p_i)\| > \epsilon$ **hacer**
  $x_1^p \leftarrow \alpha(x_1^p - x_1^q) + \beta(x_1^p - x_2^p$
  $y_1^p \leftarrow \alpha(y_1^p - y_1^q) + \beta(y_1^p - y_2^p$
  para cada $i \in [2, n-1]$ **hacer**
    $x_i^p \leftarrow \alpha(x_i^p - x_i^q) + \beta(2x_i^p - x_{i+1}^p - x_{i-1}^p)$
    $y_i^p \leftarrow \alpha(y_i^p - y_i^q) + \beta(2y_i^p - y_{i+1}^p - y_{i-1}^p)$
  fin
  $x_n^p \leftarrow \alpha(x_n^p - x_n^q) + \beta(x_n^p - x_{n-1}^p)$
  $y_n^p \leftarrow \alpha(y_n^p - y_n^q) + \beta(y_n^p - y_{n-1}^p)$
fin
regresar P

3.4. Generación de un perfil de velocidad

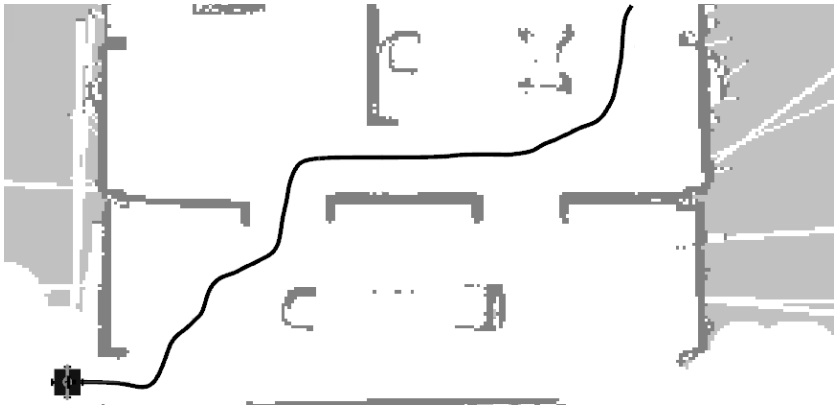
En el ViRBot, el banco de procedimientos es un conjunto de máquinas de estados que se utilizan para llevar a cabo tareas simples, tales como tomar un objeto (una vez que la posición ha sido determinada empleando algoritmos



(a) Suavizado con $\beta = 0$



(b) Suavizado con $\alpha = 0$



(c) Suavizado con $\alpha = 0.9$ y $\beta = 0.01$

Figura 3.5: Ejemplos de suavizado de rutas con diferentes valores de α y β.

de visión computacional), preguntar y memorizar un nombre, navegar hasta un punto dado siguiendo una ruta, entre otros. Este conjunto de máquinas de estados pertenece a la capa de ejecución en la arquitectura ViRBot y sirven para llevar a cabo la secuencia de tareas generada por el planeador de acciones. En esta sección se verá cómo se utiliza una máquina de estados para seguir la ruta calculada por el planeador de rutas.

Las leyes de control (3.5)-(3.6) tienen la desventaja de que sólo dependen del error de ángulo, es decir, el robot no desacelera conforme se aproxima al punto meta. Esto causa que el robot presente fuertes oscilaciones cuando se encuentra en una vecindad alrededor de la posición deseada. Una forma de resolver este problema es ejecutar la ley de control sólo si la distancia al punto meta

$$d = \sqrt{(x_g - x_r)^2 + (y_g - y_r)^2}$$

es mayor que una tolerancia ϵ:

$$\begin{bmatrix} v_l \\ v_r \end{bmatrix} = \begin{cases} \begin{bmatrix} v + \frac{L}{2}\omega \\ v - \frac{D}{2}\omega \end{bmatrix}, & \text{si} \quad d > \epsilon \\ \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, & \text{en otro caso} \end{cases}$$

En este caso, el robot se detendrá abruptamente cuando el error de distancia sea menor que ϵ, lo cual tampoco es deseable, pues el robot debería desacelerar lentamente conforme se aproxima al punto meta. También es deseable que el robot incremente su velocidad lentamente al inicio del movimiento, lo cual no está considerado aún, ya que, de acuerdo con (3.5)-(3.6), el robot puede tener una velocidad lineal v_{max} desde el inicio del movimiento.

Por otro lado, las leyes de control (3.5)-(3.6) están diseñadas para alcanzar un punto meta, sin embargo, la tarea a realizar no es el seguimiento de un punto sino de una ruta compuesta por una secuencia de puntos. La aceleración y desaceleración puede lograrse parametrizando la ruta completa con respecto al tiempo. Comúnmente esto se hace ajustando un polinomio de quinto orden, o mayor, si se requieren mayores cambios de velocidad y aceleración. Expresar las posiciones x y y como una función del tiempo es fácil cuando sólo se requieren movimientos en línea recta, sin embargo, en un robot de servicio doméstico que se mueve en un ambiente dinámico, la

rutas pueden ser muy complejas y la parametrización con respecto al tiempo puede resultar poco factible.

Una forma fácil de hacer que el robot acelere y desacelere, o en general, obtener un perfil de velocidad, es mediante el uso de una máquina de estados finita aumentada (AFSM por sus siglas en inglés) que establezca los valores de v_{max} en las leyes de control (3.5)-(3.6). Sea v_{sm} la nueva velocidad lineal máxima, de modo que ahora (3.5)-(3.6) quedan de la forma:

$$v = v_{sm} e^{-\frac{e_\theta^2}{\alpha}}, \tag{3.10}$$

$$\omega = \omega_{max} \left(\frac{2}{1 + e^{-\frac{e_\theta}{\beta}}} - 1 \right), \tag{3.11}$$

En términos generales, la máquina de estados para el seguimiento de rutas se encarga de fijar el punto (x_g, y_g) y la velocidad máxima v_{sm} con el que se calcularán las velocidades de los motores de acuerdo con (3.10)-(3.11). Considerando que la ruta a seguir está compuesta de N puntos $(x_i^p, y_i^p) \quad i \in [0, N-1]$, la posición deseada para las leyes de control se elige como el punto (x_i^p, y_i^p) que está "enfrente" del robot, esto es, un punto de la ruta cuya posición está a una distancia pequeña de la posición actual. Para este trabajo, se consideró un punto a 0.3 m enfrente del robot.

La figura 3.6 muestra la máquina de estados usada para determinar v_{sm} y la posición deseada para las leyes de control. Esta máquina de estados genera tres fases de movimiento. En la primera (fase de aceleración), v_{sm} se incrementa poco a poco hasta alcanzar el valor final v_{max}, que es la velocidad de crucero. La segunda fase corresponde a la velocidad crucero. En esta etapa, v_{max} se mantiene constante y la velocidad lineal v es casi constante y disminuye sólo cuando el robot va a dar vuelta. Cuando el robot está "cerca" del último punto de la ruta, es decir, cuando la distancia r es menor que una constante r_d, se comienza la fase de desaceleración, en la que v_{sm} se asigna proporcional al error de distancia; esto hace que el movimiento del robot sea parecido al de un control de posición proporcional. Finalmente, cuando la distancia r al punto meta es menor que una tolerancia ϵ, el robot se detiene. La posición deseada para las leyes de control se va seleccionando del conjunto de puntos que componen la ruta. Cuando la distancia a la posición deseada es menor que 0.3 m, entonces ésta se cambia por el siguiente punto en la ruta. Los valores de r_d y Δv se determinan dependiendo de qué tan rápido se desea que el robot acelere y desacelere.

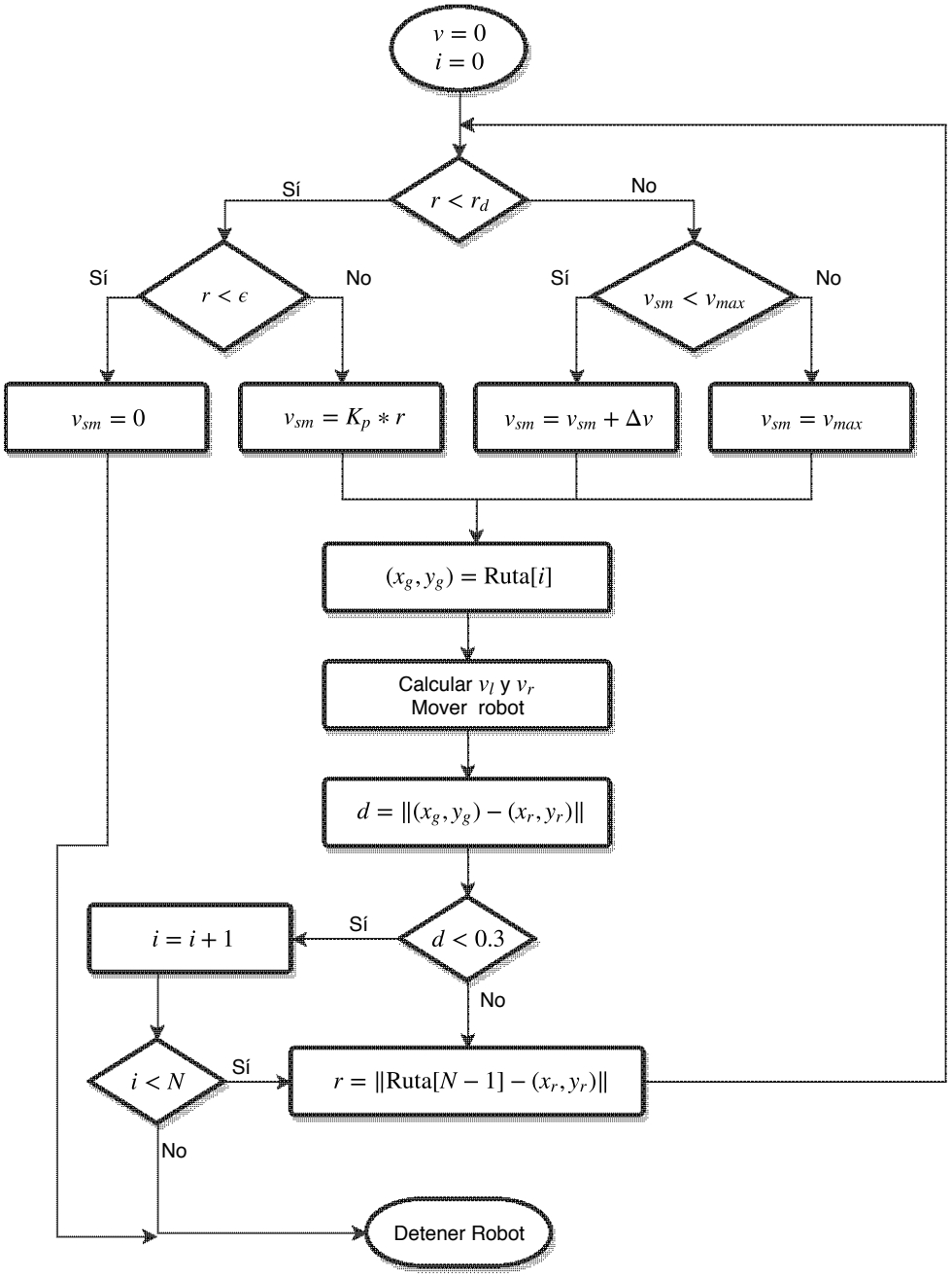
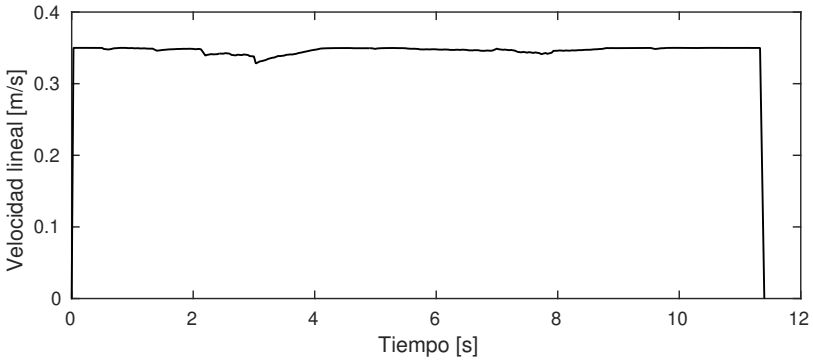
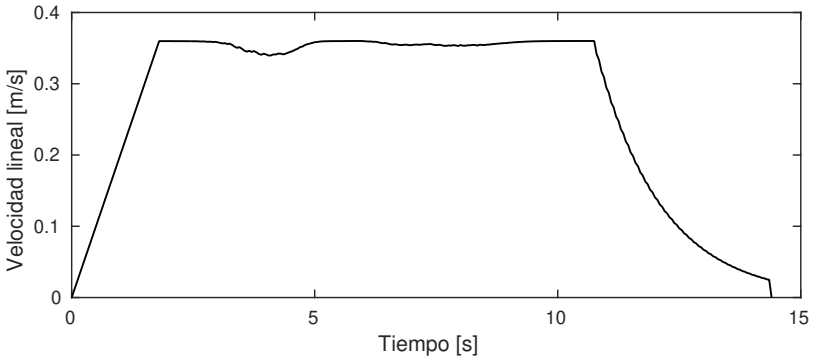


Figura 3.6: Máquina de estados utilizada para generar un perfil de velocidad.



(a) Señal de control v sin perfil de velocidad.



(b) Señal de control v usando un perfil de velocidad.

(c) Ruta seguida por el robot.

Figura 3.7: Comparación entre las señales de control usando **a)** sólo las ecuaciones (3.5)-(3.6) y **b)** la máquina de estados de la figura 3.6. La figura **c)** muestra la ruta seguida por el robot.

En la figura 3.7 se muestra una comparación entre la señal de control v generada sin perfil de velocidad y la señal v generada mediante la máquina de estados propuesta. En ambos casos la velocidad lineal máxima v_{max} fue de 0.35 m/s, sin embargo, se puede observar que, utilizando la máquina de estados, la velocidad se incrementa suavemente al inicio del movimiento, entre 0 y 1.5 [s]. Entre los 1.5 y 11 [s], la velocidad lineal v se mantiene cerca de v_{max}, las variaciones que presenta se deben a las curvas que presenta la ruta en las que, de acuerdo con las leyes de control, la velocidad lineal disminuye levemente y la velocidad angular aumenta. Desde los 11 [s] y hasta el final del movimiento se observa una desaceleración con una curva de tipo exponencial. Esto se debe a que la velocidad v_{sm} se asigna proporcional al error de distancia.

Hasta este momento, para la planeación y seguimiento de rutas se ha considerado que se conoce la posición del robot. En el siguiente capítulo se explica cómo se estima dicha posición.

Capítulo 4

Localización y mapeo

En la arquitectura ViRBot, el cartógrafo es el módulo encargado de construir y mantener una representación del ambiente, además de determinar la posición del robot dentro de dicho ambiente. En este capítulo se explica la forma en que se construyen los mapas y los algoritmos de localización empleados. Primero, en la sección 4.1 se explica la forma en que se calcula la odometría, necesaria para los algoritmos de localización, pues representa una estimación del movimiento con base en el modelo cinemático. En la sección 4.2 se explica la localización mediante el Filtro de Kalman Extendido. Finalmente, en la sección 4.3 se explica el procedimiento para construir mapas de rutas empleando técnicas de cuantización vectorial.

4.1. Odometría

La odometría se refiere a la estimación de la posición únicamente con base en la integración de las velocidades de los motores. La odometría funciona muy bien para estimar pequeños desplazamientos, sin embargo, dado que la estimación se basa en una integración, cualquier pequeño error en los parámetros del modelo de movimiento resultará en un error en la estimación que crece indefinidamente. Esto hace necesario algoritmos de localización como los que se describirán en las secciones siguientes, sin embargo, ambos necesitan de la odometría como primera estimación de la posición. Es por ello que en esta sección se describe cómo se calcula.

Una primera aproximación de odometría es simplemente resolver numéricamente las ecuaciones diferenciales (3.1)-(3.3) con v_x, v_y y ω calculados de

acuerdo con (3.10)-(3.11). Sin embargo, si se dispone de encoders en los motores, se puede hacer una mejor estimación con base en la distancia recorrida por cada llanta.

Considérese una base omnidireccional con cuatro motores perpendiculares como la que se muestra en la figura 4.1. Los motores derecho e izquierdo producen un movimiento "frontal" mientras que el frontal y trasero, producen un movimiento "lateral". La suma de ambos desplazamientos resulta en el movimiento omnidireccional. Para calcular el desplazamiento total con respecto a un sistema de referencia O_{map} primero se calculan los desplazamientos con respecto a un sistema O_r que está montado sobre el robot y luego se aplica una transformación homogénea.

Dada una base omnidireccional de diámetro L y con una posición actual $P_k = (x_k, y_k, \theta_k)$, la posición P_{k+1} después de un tiempo Δt se puede obtener a partir de las distancias recorridas por cada llanta. Considere el esquema de la figura 4.1 y las siguientes variables y sistemas de referencia:

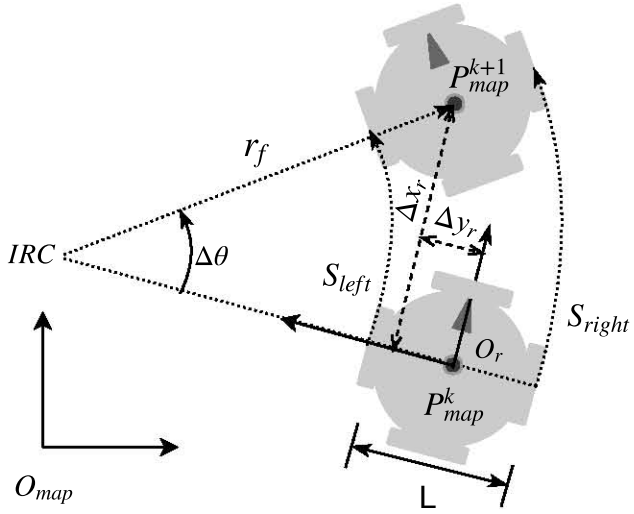
- S_{left}, S_{right}, S_{front} y S_{rear}: distancias recorridas por las llantas izquierda, derecha, frontal y trasera respectivamente.
- L: diámetro del robot (distancia entre ejes de las llantas).
- O_A: sistema de referencia absoluto.
- O_R: sistema de referencia montado sobre el centro del robot.

El incremento en la posición se puede calcular como la suma del incremento producido por las llantas izquierda y derecha (desplazamiento frontal) más el incremento producido por las llantas frontal y trasera (desplazamiento lateral). El desplazamiento frontal se puede calcular como:

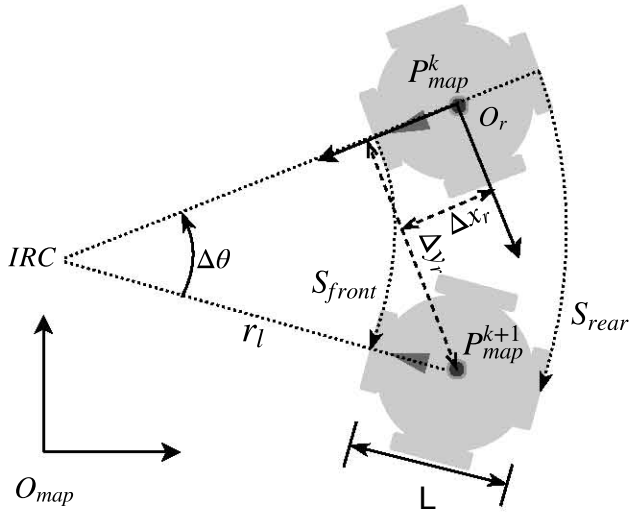
$$\begin{aligned} \Delta\theta &= \frac{S_{right} - S_{left}}{L} \\ \Delta x_r &= r_f \sin\Delta\theta \\ \Delta y_r &= r_f \left(1 - \cos\Delta\theta\right) \end{aligned}$$

donde

$$r_f = \frac{S_{left} + S_{right}}{2\Delta\theta}$$



(a) Desplazamiento causado por las llantas izquierda y derecha.



(b) Desplazamiento causado por las llantas delantera y trasera.

Figura 4.1: Variables usadas para el cálculo de la odometría.

es el radio de la curva descrita debida al desplazamiento frontal, es decir, la distancia del robot al centro instantáneo de rotación (IRC por sus siglas en inglés). El desplazamiento lateral se puede calcular como:

$$\begin{aligned} \Delta\theta &= \frac{S_{front} - S_{rear}}{L} \\ \Delta x_r &= r_l\,(1 - \cos\Delta\theta) \\ \Delta y_r &= r_l \sin\Delta\theta \end{aligned}$$

con $r_l = \frac{S_{rear}+S_{front}}{2\Delta\theta}$ la distancia al IRC de la curva producida por el desplazamiento lateral. El incremento en el ángulo $\Delta\theta$ es el mismo para ambos movimientos, por lo que se puede promediar el valor de ambas estimaciones. El valor de Δx_r y Δy_r se puede obtener con la suma de ambos desplazamientos:

$$\begin{aligned} \Delta\theta &= \frac{S_{right} - S_{left} + S_{front} - S_{rear}}{2L} \\ r_f &= \frac{S_{left} + S_{right}}{2\Delta\theta} \\ r_l &= \frac{S_{rear} + S_{front}}{2\Delta\theta} \\ \Delta x_r &= r_f \sin\Delta\theta + r_l\,(1 - \cos\Delta\theta) \\ \Delta y_r &= r_f\,(1 - \cos\Delta\theta) + r_l \sin\Delta\theta \end{aligned}$$

Estos desplazamientos están expresados con respecto al sistema O_r, por lo que para calcular la posición con respecto a O_map basta con hacer una transformación:

$$x_{k+1} = x_k + \Delta x_r \cos\theta_k - \Delta y_r \sin\theta_k \tag{4.1}$$

$$y_{k+1} = y_k + \Delta x_r \sin\theta_k + \Delta y_r \cos\theta_k \tag{4.2}$$

$$\theta_{k+1} = \theta_k + \Delta\theta \tag{4.3}$$

Para obtener la odometría de una base diferencial, basta con considerar $S_{rear} = 0$ y $S_{front} = 0$. En el caso del robot de servicio utilizado en este trabajo, las distancias S_{right}, S_{left}, S_{rear} y S_{front} se obtienen a partir de las lecturas de los encoders colocados en el eje de cada motor, de acuerdo con

$$S = K_{MPP} P$$

donde P es el número de pulsos contados y K_{MPP} es una constante que indica el número de metros recorridos por cada pulso. Esta constante se puede obtener a partir de la medida de la llanta y las especificaciones del encoder, pero es más confiable si se obtiene experimentalmente a partir de varias pruebas de movimiento.

4.2. Filtro de Kalman Extendido

El Filtro de Kalman Extendido (EKF por sus siglas en inglés) es un método de localización local, es decir, la estimación inicial de la posición debe ser cercana a la posición real del robot. Si esta condición no se cumple, el EKF no convergerá a la posición real del robot. El EKF considera que se tiene un modelo del sistema y un modelo de observación, es decir, un modelo que relaciona los estados del sistema con las mediciones realizadas. El EKF sirve para estimar los estados del sistema cuando se tiene ruido que se adiciona tanto al modelo (llamado ruido de proceso) como a las señales medidas (llamado ruido de medición). La estimación con EKF consiste de dos etapas principales: predicción y actualización. En la primera etapa se estiman los estados de acuerdo con el modelo, es decir, como si no se tuviera ruido de proceso. En la etapa de actualización se corrigen las estimaciones con base en la diferencia entre las salidas estimadas y las medidas.

Para poder localizar al robot empleando el EKF primero se requiere de una medición. En este trabajo, dicha medición se hace extrayendo segmentos de recta a partir de las lecturas de un sensor láser que luego se comparan con un mapa previamente construido. La diferencia entre las rectas observadas y las del mapa sirve para calcular la posición del robot. Esta medición se filtra usando un EKF y con ello se localiza al robot.

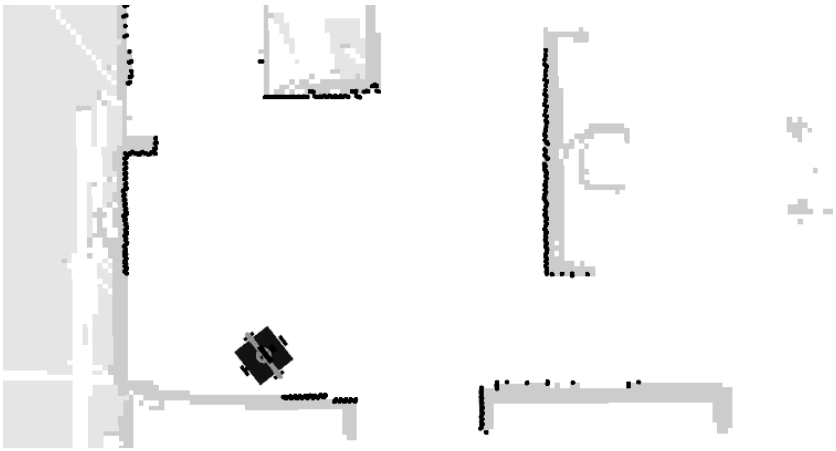
4.2.1. Extracción de líneas

Los segmentos de línea son extraídos de las lecturas del sensor láser usando un algoritmo basado en el trabajo de Yan et al. (2012). Éste consiste de dos partes principales: agrupar los puntos de acuerdo con su cercanía y calcular la ecuación de la recta mediante análisis de componentes principales (PCA por sus siglas en inglés). El algoritmo 2 muestra los pasos para la extracción de líneas y en la figura 4.2 se muestra el resultado de dicho algoritmo.

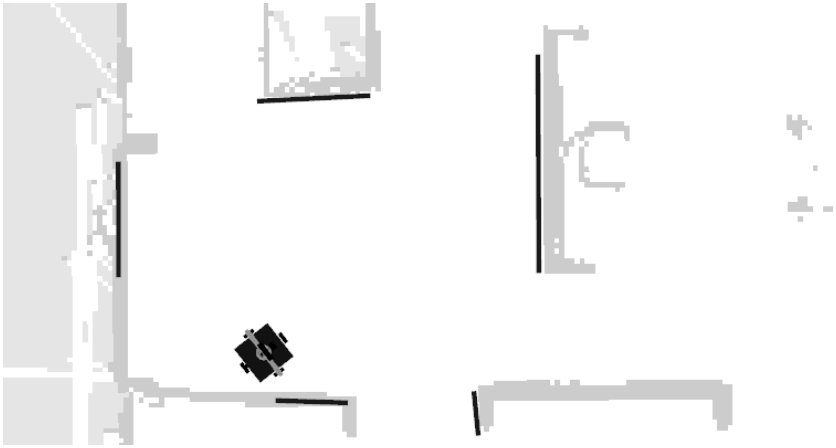
Algoritmo 2: Extracción de líneas a partir de las lecturas del sensor láser.

Datos:
Conjunto de n lecturas del láser $P = \{p_0 \dots p_i \dots p_{n-1}\}$
Resultado:
Conjunto de segmentos de línea L

$L \leftarrow \emptyset$
$i \leftarrow 1$
mientras $i < n$ **hacer**
  $L \leftarrow L \cup \{L_j\}$ //Nueva línea L_j
  $C_j \leftarrow \emptyset$ //Conjunto de puntos p_i asociados a L_j
  mientras $\|p_i - p_{i-1}\| < K_1$ **hacer**
    $C_j \leftarrow C_j \cup \{p_i\}$
    $i \leftarrow i + 1$
  fin
  Calcular la ecuación de L_j con el primero y último puntos de C_j
fin
$best\_fit \leftarrow$ False
mientras $\neg best\_fit$ **hacer**
  $best\_fit \leftarrow$ True
  para cada $L_j \in L$ **hacer**
    Del conjunto C_j asociado a L_j, encontrar el punto p_f más alejado de L_j
    si $d(p_f, L_j) > K_2$ **entonces**
      Dividir L_j en dos nuevas líneas L_{j1} y L_{j2} tomando P_f como punto de quiebre.
      Dividir el conjunto C_j en los correspondientes C_{j1} y C_{j2}
      $best\_fit =$ False
    fin
  fin
fin
para cada $L_j \in L$ **hacer**
  Calcular la ecuación de L_j usando PCA y los puntos asociados C_j
fin



(a) Lecturas del sensor láser.



(b) Líneas extraídas.

Figura 4.2: Resultado de la extracción de líneas utilizando el algoritmo 2

4.2.2. Cálculo de la posición

La posición del robot se calcula promediando las diferencias entre las líneas conocidas y las líneas observadas. Para empatar un segmento de línea observado L_o con un segmento conocido L_k, se propone la función de pseudo distancia:

$$d(L_o, L_k) = d_{pf} + d_{pi} + d_{nf} + d_{ni} \tag{4.4}$$

La distancia $d(L_o, L_k)$ es la suma de las distancias entre los extremos de ambos segmentos y la distancia de los extremos del segmento observado a la línea que contiene al segmento conocido. Como se muestra en la figura 4.3, considere que el segmento conocido L_k está definido por los puntos extremos

$$L_k : \overline{p_{ki}p_{kf}}, \quad p_{ki} = (x_{ki}, y_{ki}), \quad p_{kf} = (x_{kf}, y_{kf})$$

y el segmento observado L_o, por los puntos

$$L_o : \overline{p_{oi}p_{of}}, \quad p_{oi} = (x_{oi}, y_{oi}), \quad p_{of} = (x_{of}, y_{of})$$

También considere que la línea que contiene al segmento conocido está definida por la ecuación

$$L_k : A_k x + B_k y + C_k = 0$$

y la línea que contiene al segmento observado, por la ecuación

$$L_o : A_o x + B_o y + C_o = 0$$

Las distancias d en la ecuación 4.4 están dadas por:

$$\begin{aligned} d_{pf} &= \|p_{of} - p_{kf}\| \\ d_{pi} &= \|p_{oi} - p_{ki}\| \\ d_{nf} &= \frac{|A_k x_{kf} + B_k y_{kf} + C_k|}{\sqrt{A_k^2 + B_k^2}} \\ d_{ni} &= \frac{|A_k x_{ki} + B_k y_{ki} + C_k|}{\sqrt{A_k^2 + B_k^2}} \end{aligned}$$

Un segmento de línea observado L_o se considera empatado con un segmento conocido L_k si la distancia $d(L_o, L_k)$ es menor que la distancia entre L_o y cualquier otro segmento conocido.

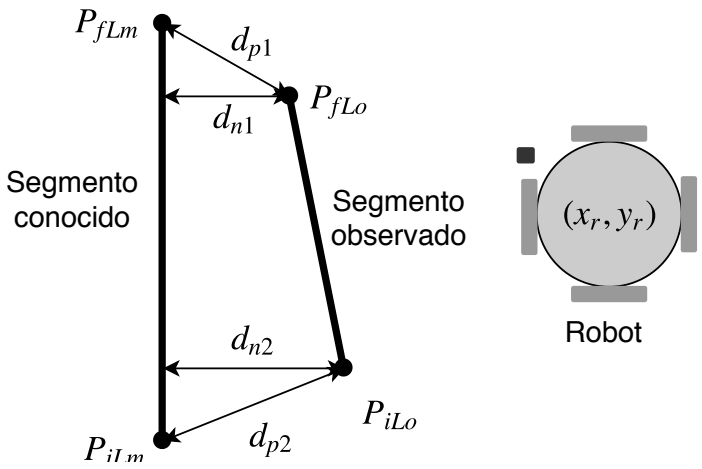


Figura 4.3: Empatado de un segmento observado con uno conocido.

Se asume que todos los objetos en el ambiente real (mesas, escritorios, paredes) están en una orientación ortogonal o cercana a ésta y, por lo tanto, se espera que cada segmento observado tenga un ángulo cercano ya sea a cero o a noventa grados. Un segmento observado se considera vertical si

$$|B_o/A_o| < 0.6$$

y horizontal si

$$|A_o/B_o| < 0.6$$

Si el segmento observado no es vertical ni horizontal, entonces se considera como una observación errónea y no se toma en cuenta.

Para cada par (L_k, L_o) (segmento conocido, segmento observado) se calcula un error de posición $\left(e_{ko_x}, e_{ko_y}\right)$ y un error de ángulo e_{ko_θ}. Si L_o es vertical, se usan las siguientes ecuaciones:

$$e_{ko_x} = \left(x_{k_f} - x_{o_f} + x_{k_i} - x_{o_i}\right)/2 \quad (4.5)$$
$$e_{ko_y} = 0 \quad (4.6)$$
$$e_{ko_\theta} = \pi/2 - \operatorname{atan2}\left(A_o, -B_o\right) \quad (4.7)$$

Y si L_o es horizontal:

$$e_{ko_x} = 0 \quad (4.8)$$
$$e_{ko_y} = \left(y_{k_f} - y_{o_f} + y_{k_i} - y_{o_i}\right)/2 \quad (4.9)$$
$$e_{ko_\theta} = -\operatorname{atan2}\left(A_o, -B_o\right) \quad (4.10)$$

Finalmente, un error promedio ponderado e_p se obtiene a partir de todos los pares empatados y la posición medida P_{Rm} se determina como

$$P_{Rm} = P_R + e_p \tag{4.11}$$

donde P_R es la estimación actual de la posición del robot y

$$e_p = \sum_i w_i e_{ko_i} \tag{4.12}$$

donde e_{ko_i} es el error de posición i-ésimo calculado de acuerdo con (4.5)-(4.10) y w_i es un peso inversamente proporcional a la distancia entre la línea conocida y la línea observada, calculado como

$$w_i = \frac{d_i}{d_0 \cdots + d_i + \cdots + d_n} \tag{4.13}$$

con d_i, la distancia calculada de acuerdo con (4.4). Es decir, el error de posición determinado por cada par de líneas empatado es proporcional a la discrepancia. En otras palabras, entre más difiera una línea observada con su correspondiente conocida, menos se tomará en cuenta para la corrección de posición. En el capítulo 6 se explica cómo se usan estas discrepancias para determinar regiones confiables para realizar la localización.

4.2.3. Filtrado de la posición

El Filtro de Kalman se puede aplicar a sistemas continuos y discretos. En este trabajo, se optó por la versión discreta dado el periodo de muestreo de 100 [ms], que es el tiempo transcurrido entre un cálculo de la odometría y otro y corresponde también con el periodo de muestreo del sensor láser. De las ecuaciones (3.1)-(3.3), se puede obtener un modelo discreto usando una aproximación de las derivadas:

$$x_{k+1} = x_k + \Delta t \left(v_x \cos\theta_k - v_y \sin\theta_k\right) + \nu_1 \tag{4.14}$$

$$y_{k+1} = y_k + \Delta t \left(v_x \sin\theta_k + v_y \cos\theta_k\right) + \nu_2 \tag{4.15}$$

$$\theta_{k+1} = \theta_k + \Delta t\omega + \nu_3 \tag{4.16}$$

donde

$$X = [x\ y\ \theta]^T$$

es el vector de estados, Δt es el paso de muestreo, v_x, v_y y ω, las velocidades frontal, lateral y angular, respectivamente (consideradas como señales de entrada) y

$$\nu = \begin{bmatrix} \nu_1 & \nu_2 & \nu_3 \end{bmatrix}^T$$

es ruido gaussiano sin correlación temporal, media cero y matriz de covarianza Q. El modelo de observación, considerando ruido en las mediciones, está dado por

$$Z_k = X_k + \omega_k$$

donde ω_k es también ruido gaussiano, sin correlación temporal, media cero y matriz de covarianza R. Los estados medidos Z_k se obtienen de acuerdo con (4.11) como se describen en la sección 4.2.2. La estimación de la posición consiste de los siguientes pasos:

Predicción. Con base en el modelo cinemático, el siguiente estado y las salidas se predicen considerando que no hay ruido tanto en el modelo de transición de estados como en el modelo de observación:

$$\begin{aligned} \hat{X}_{(k+1|k)} &= \hat{F}(X_{(k|k)}, u_{(k)}) \\ \hat{Z}_{(k+1|k)} &= \hat{X}_{(k+1|k)} \\ P_{(k+1|k)} &= J_{(k)} P_{(k|k)} J_{(k)}^T + Q \end{aligned}$$

donde P es la matriz de covarianza del error de estimación y J es el Jacobiano de la función F con respecto a X, que para el modelo (4.14)-(4.16) y considerando únicamente una base diferencial, está dado por

$$J = \begin{bmatrix} 1 & 0 & -\Delta t v_x \sin\theta \\ 0 & 1 & \Delta t v_x \cos\theta \\ 0 & 0 & 1 \end{bmatrix}$$

La matriz P es la covarianza del error de estimación, cuya norma decrece conforme se realiza el proceso de estimación. Sólo es necesario asignar un valor inicial para esta matriz. En este trabajo, la matriz se inició en la identidad.

Actualización. Con base en el error de observación y la covarianza del error de estimación, el siguiente estado estimado se calcula de acuerdo con:

$$\begin{aligned} S_{(k+1)} &= H_{(k+1)} P_{(k|k+1)} H_{(k+1)}^T + R \\ K_{(k+1)} &= P_{(k+1|k)} H_{(k+1)}^T S_{(k+1)}^{-1} \\ \hat{X}_{(k+1|k+1)} &= \hat{X}_{(k+1|k)} + K_{(k+1)} \left(Z_{(k+1)} - \hat{Z}_{(k+1|k)} \right) \\ P_{(k+1|k+1)} &= \left(I - P_{(k+1)} H_{(k+1)} \right) P_{(k+1|k)} \end{aligned}$$

donde H es el Jacobiano del modelo de observación que, en este caso, es la identidad debido a que se considera que las señales medidas son el mismo vector de estados. La matriz resultante K se conoce como ganancia de Kalman. Para este trabajo, se emplearon las matrices

$$Q = \begin{bmatrix} 0.001 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0.001 \end{bmatrix}$$

$$R = \begin{bmatrix} 0.02 & 0 & 0 \\ 0 & 0.02 & 0 \\ 0 & 0 & 0.04 \end{bmatrix}$$

y un tiempo de muestreo $\Delta t = 0.1$ [s].

En la etapa de predicción, la estimación de la posición con base en el modelo cinemático no se calcula resolviendo las ecuaciones en diferencias (4.14)-(4.16), sino tomando la odometría como dicha predicción. De hecho, la odometría es en sí una estimación basada sólo en el modelo cinemático. En la etapa de actualización, las mediciones $Z_{(k+1)}$ son aquellas determinadas por (4.11) como se explicó en la sección 4.2.2.

Es importante mencionar que la estimación de la posición no se lleva a cabo todo el tiempo. Esto se debe a que en ambientes muy desordenados es difícil realizar la extracción de líneas de forma confiable y el aumento de falsos positivos puede llevar a errores grandes en la estimación de la posición lo que puede producir que el Filtro de Kalman ya no sea viable para la localización. Por esto, la estimación de la posición sólo se realiza en ciertas regiones marcadas como confiables. Si el robot está fuera de esas regiones, entonces la posición se estima únicamente con base en la odometría. Esta característica del sistema de navegación está inspirada en ciertos hallazgos hechos en la biología del comportamiento sobre navegación en animales. En el capítulo 6 se explican estos hallazgos y la forma en que se utilizaron para mejorar la localización.

Localización Adaptable de Monte Carlo

La Localización Adaptable de Monte Carlo es un método probabilístico usado para determinar la posición del robot con base en filtros de partículas que se usan para seguir la posición en 2D contra un mapa conocido. Está implementado en *amcl*, un paquete de ROS de código abierto. La base teórica

está explicada en Thrun et al. (2005) y los detalles de la implementación se pueden encontrar en la documentación en línea del paquete. En este trabajo, el paquete *amcl* se usó en conjunto con el Filtro de Kalman Extendido para estimar la posición del robot.

4.3. Construcción de mapas de rutas mediante algoritmos de agrupamiento

Los mapas de rutas son útiles para la navegación de robots móviles en ambientes estructurados. Si dichos mapas de rutas se construyen los suficientemente rápido y con base en información extraída de los sensores del robot, entonces se pueden usar para evasión de obstáculos. Si no se dispone de una representación geométrica del ambiente, se puede obtener por métodos de cuantización vectorial y, con base en esta representación, es posible construir un mapa de rutas. A grandes rasgos, el proceso para construir dicho mapa es el siguiente: se procesa la nube de puntos adquirida de una cámara RGB-D y se separa el espacio libre del ocupado. Luego, tanto el espacio libre como el ocupado se agrupan. Los centroides y tamaños de los grupos correspondientes al espacio ocupado son usados como una representación geométrica de los objetos en el ambiente. Los centroides del espacio libre son usados como nodos del mapa de rutas.

Las cámaras RGB-D proveen información compuesta por una imagen RGB y una nube de puntos que representa la posición en el espacio de cada píxel de la imagen capturada. En este trabajo se utilizó únicamente la información espacial, que está dada por un conjunto R de tripletas de la forma

$$r_j = \left(x_{screen_j}, y_{screen_j}, d_j\right)$$

donde $(x_{screen_j}, y_{screen_j})$ es la posición del píxel en la imagen y d_j es la distancia sobre la línea de vista de la cámara al objeto observado. La nube de puntos S expresada en coordenadas cartesianas

$$s_j = (x_j, y_j, z_j)$$

con respecto al plano de la cámara, se puede obtener con la transformación

$$s_j = Mr_j, \tag{4.17}$$

donde M es la matriz de parámetros intrínsecos de la cámara RGB-D, la cual se puede obtener por diversos métodos, por ejemplo, el que se describe en Jin, Lei, y Geng (2014).

Dado que la cámara está montada en la cabeza del robot, que cuenta con movimientos *pan* y *tilt*, es conveniente aplicar una transformación homogénea para expresar los puntos con respecto a la base móvil. La transformación homogénea está dada por

$$p_j = \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 & H_x \\ \sin\theta & \cos\theta & 0 & H_y \\ 0 & 0 & 1 & H_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} s_j \tag{4.18}$$

donde $(\theta,\ \phi)$ son los ángulos *pan* y *tilt* de la cabeza, respectivamente, con $\theta > 0$ cuando la cabeza gira hacia la izquierda y $\phi > 0$ cuando gira hacia abajo. (H_x, H_y, H_z) es la posición de la cabeza con respecto al centro del robot.

Después de la transformación, un punto p_j se considera como espacio libre si su componente en z es menor que una constante K_h y, como espacio ocupado, de otro modo.

4.3.1. Cuantización Vectorial

La cuantización vectorial (Linde, Buzo, y Gray, 1980) se usa comúnmente para compresión de datos en telecomunicaciones y procesamiento digital de señales. En el campo de la robótica, se usa también para comprimir datos y obtener un conjunto más pequeño pero significativo. En este trabajo, se usó cuantización vectorial para agrupar el espacio libre y ocupado y, con base en los grupos resultantes, construir un mapa de rutas.

Dada una nube de puntos P, es decir, un conjunto de n_v vectores

$$p_j = (x_j, y_j, z_j),\ j = 1, ..., n_v$$

que representan una posición en el espacio, se puede encontrar un conjunto de centroides que representen estos vectores. Una colección del centroides es llamada *codebook*, está diseñada a partir de una larga secuencia de entrenamiento y es representativa de todos los vectores p_j a ser codificados. El *codebook* se crea con el algoritmo Linde-Buzo-Gray (Linde et al., 1980) como se explica en el algoritmo 3.

Algoritmo 3: Cuantización Vectorial con el algoritmo Linde-Buzo-Gray.

Datos:
Nube de puntos $P = \{p_j | p_j = (x_j, y_j, z_j), j \in [1, n_v]\}$
El tamaño deseado del codebook L_d
Resultado: Codebook $D = \{C_{L_m}\}$

$m \leftarrow 1$ //Iteración actual
$L_m \leftarrow 1$ //Número actual de centroides
//El primer centroide es el promedio de todos los puntos
$C_1 \leftarrow \frac{1}{n_v} \sum_{j=1}^{n_v} p_j$
$D_m \leftarrow \{C_1\}$ //Codebook inicial
mientras $L_m < L_d$ **hacer**
  para cada $C_i \in D_m$ **hacer**
    $\psi \leftarrow$ perturbación de pequeña magnitud
    Obtener dos nuevos centroides sumando $\pm\psi$
  fin
  //D_{m+1} ahora contiene L_{m+1} nuevos centroides
  $L_{m+1} \leftarrow 2L_m$
  //Conjunto de grupos para cada centroide
  $R \leftarrow \{R_k\}, k \in [1, L_{m+1}]$
  $\overline{d_t} \leftarrow \infty$ //Cambio promedio en los centroides
  mientras $\overline{d_t} > \epsilon$ **hacer**
    para cada $p_j \in P$ **hacer**
      Asignar p_j al grupo R_k más cercano cuyo centroide es $C_k \in D_{m+1}$.
    fin
    para cada $\forall R_k \in R$ **hacer**
      Recalcular el centroide C_k promediando todos los vectores $p_j \in R_k$
    fin
    Calcular $\overline{d_t}$ como la distancia media entre C_k y su valor en la iteración previa
  fin
fin

El tamaño deseado L_d del *codebook* se elige con un equilibrio entre las limitaciones en el tiempo de cómputo para permitir una operación en tiempo real y la precisión deseada. En este trabajo, se utilizó

$$|\psi| = 0.01 \quad \epsilon = 0.03 \quad L_d = 64$$

El algoritmo LBG se aplica para agrupar tanto el espacio libre como el ocupado y se calculan un total de 128 centroides, 64 para el espacio libre y 64 para el ocupado. La figura 4.4 muestra los grupos resultantes.

4.3.2. Construcción de mapas de rutas

Después de que los espacios libre y ocupando han sido agrupados, el mapa de rutas se construye siguiendo el algoritmo 4. En este algoritmo $e(v, v\prime)$ representa el borde entre los nodos v y $v\prime$ y

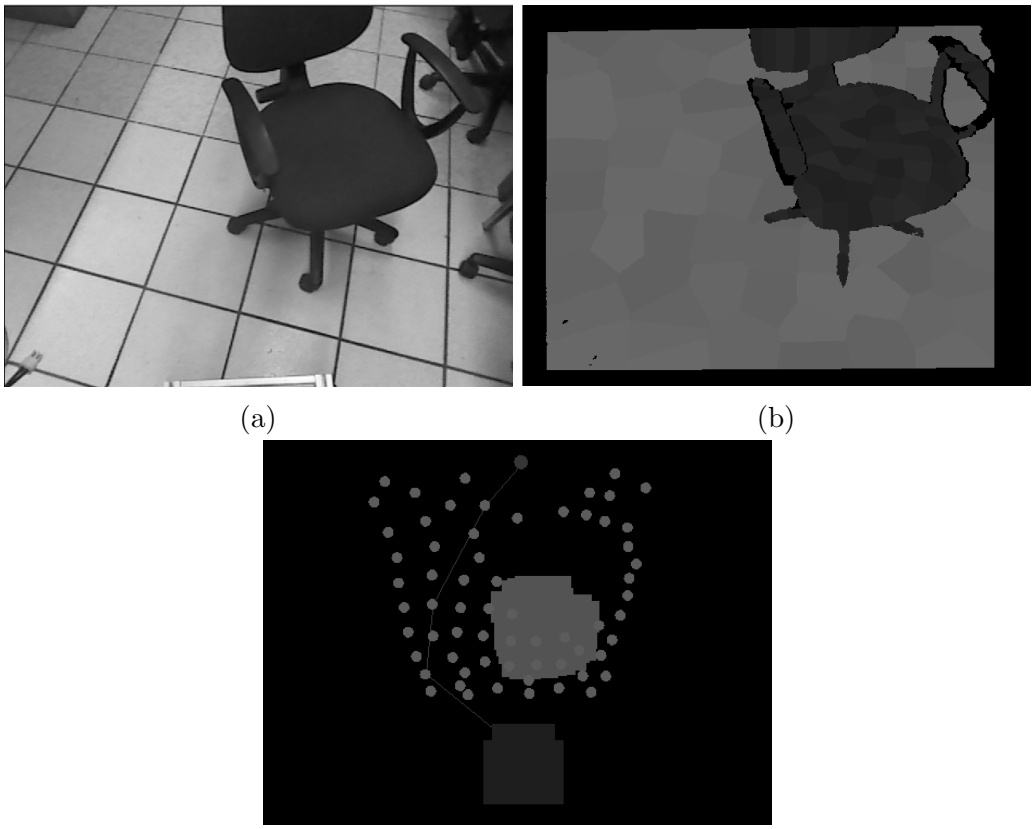
$$Vis(v, v\prime, p)$$

es una función que determina si es posible alcanzar el nodo v desde el nodo $v\prime$, en línea recta, sin chocar con el obstáculo cuyo centroide es p. NV es el valor que devuelve la función Vis cuando el nodo v no es visible desde el nodo $v\prime$.

La figura 4.4 muestra la imagen original, los espacios libre y ocupado agrupados y el mapa de rutas resultante. En general, los algoritmos de agrupamiento tienen un alto costo computacional, lo que resulta en latencias grandes que harían poco factible utilizarlos como un método en línea para evasión de obstáculos. Sin embargo, en este trabajo la cuantización vectorial se implementó en paralelo permitiendo así obtener latencias mucho más cortas. Los detalles de esta implementación se dan en Negrete, Savage, Cruz, y Márquez (2014).

4.3.3. Implementación en paralelo

La transformación de la nube de puntos al horizonte canónico, la separación del espacio libre del ocupado y el agrupamiento fueron implementados en paralelo utilizando CUDA, un conjunto de herramientas diseñadas específicamente para desarrollar aplicaciones paralelas en GPUs NVidia. El algoritmo para construir el mapa de rutas a partir de los espacios cuantizados se implementó de forma serial.



(a) (b)

(c)

Figura 4.4: **a)** Imagen original capturada por la cámara RGB-D. **b)** Los grupos del espacio libre están en gris claro y los ocupados, en gris oscuro. Las regiones en negro corresponden a los punto sin información de profundidad. **c)** La representación resultante del ambiente. Los puntos representan los nodos (centroides del espacio libre) y los rectángulos en gris claro, los obstáculos del ambiente.

Algoritmo 4: Construcción del mapa de rutas a partir del espacio agrupado.

Datos:
//Centroides del espacio ocupado cuantizado
$P = \{p_i\}$
//Centroides del espacio libre cuantizado
$C = \{c_j\}$
Resultado:
Mapa de rutas $G(V, E)$ con:
E: Los bordes del mapa de rutas
V: Los nodos del mapa de rutas

$E \leftarrow \emptyset$
$V \leftarrow C$
$i \leftarrow 0$
para cada $v \in V$ **hacer**
  para cada $v\prime \in V$ **hacer**
    si $e(v, v\prime) \notin V \wedge (Vis(v, v\prime, p) \neq NV \ \forall p \in P)$ **entonces**
      $E \leftarrow E \cup \{e(v, v\prime)\}$
    fin
  fin
fin

Tiempo en serie [s]	Tiempo en paralelo [s]
1.341	0.078
1.373	0.078
1.341	0.078
1.388	0.077
1.404	0.078
1.357	0.078

Tabla 4.1: Comparación de tiempos de procesamiento para las implementaciones en serie y paralelo.

En este trabajo, se utilizó un GPU Nvidia Quadro 2000, el cual cuenta con 192 núcleos, 1.25 GHz de velocidad y 1 GB de memoria RAM. Se utilizó la versión 6.0 de CUDA para el procesamiento en paralelo, las bibliotecas OpenNI versión 1.5 para la captura de datos de la cámara RGB-D y las bibliotecas OpenCV 2.4.8 para operaciones básicas sobre imágenes (colorear los espacios resultantes, despliegue en pantalla, entre otras).

Para comparar la efectividad de la implementación en paralelo, el agrupamiento también se implementó en serie. La tabla 4.1 muestra los tiempos necesarios para procesar un cuadro de vídeo tanto para la implementación en serie como la paralela. Se puede observar que la implementación en paralelo es significativamente más rápida que la serial. La media de tiempo para la implementación en paralelo fue de 78 [ms] y fue hasta 18 veces más rápida que la implementación en serie. Con este tiempo, es posible utilizar la creación de mapas de rutas a partir de nubes de puntos como algoritmo de evasión de obstáculos.

La construcción de mapas de rutas será utilizada en conjunto con los métodos basados en comportamientos como se explica en el siguiente capítulo.

Capítulo 5

Comportamientos reactivos

Como explica en la sección 2.3, una vez que se genera una secuencia de acciones a realizar, el ejecutor se encarga de llevarlas a cabo empleando métodos basados en comportamientos. En este capítulo se explica la forma en que se implementa el subsistema de los métodos basados en comportamientos de la arquitectura ViRBot. El sistema de navegación propuesto implementa los comportamientos necesarios para ejecutar los movimientos generados en la capa de planeación. Se comienza por describir el enfoque de la robótica basada en comportamientos, sus características y las ventajas de mezclarla con el paradigma jerárquico. Después se describen los campos potenciales artificiales, empleados para implementar uno de los comportamientos. Por último, se describe la forma en que se integran los tres comportamientos desarrollados.

5.1. La robótica basada en comportamientos

Como se mencionó en el capítulo 2, el jerárquico y el reactivo son dos paradigmas de la robótica, cada uno con sus ventajas y desventajas. El paradigma jerárquico supone una representación del ambiente y esto proporciona a los robots altas capacidades de predicción y, por lo tanto, la capacidad de resolver tareas más complejas; sin embargo, esto también implica un alto costo computacional y respuestas relativamente lentas, lo que dificulta hacer frente a cambios rápidos en el ambiente. Por el contrario, el paradigma reactivo no depende de una presentación del ambiente y su respuesta es rápida, lo que lo hace una mejor opción para trabajar en ambientes dinámicos pero

disminuye las capacidades de predicción.

La robótica basada en comportamientos afirma que la inteligencia en un robot se puede obtener como una propiedad emergente de un conjunto lo suficientemente grande de comportamientos interactuando unos con otros (Arkin, 1998). Un comportamiento es un par estímulo-respuesta y, dado que no hay planeación de por medio, la respuesta es rápida. Puesto que un comportamiento representa una "conexión directa" de los sensores con los actuadores, es posible tener varios comportamientos tratando de controlar los actuadores del robot. Una solución a este problema es el uso de un árbitro que se encargue de seleccionar un comportamiento con base en una jerarquía o la amplitud de la respuesta, o bien, que produzca una respuesta mediante la ponderación de varios comportamientos.

Mezclar ambos paradigmas permite tomar las ventajas de cada uno y es el caso de la arquitectura ViRBot. La representación del ambiente y los algoritmos de planeación permiten al robot realizar movimientos complejos mientras que los métodos basados en comportamientos le permiten hacer frente a cambios rápidos en el ambiente, como obstáculos no considerados en el mapa. En este trabajo, una vez que se planea una ruta, ésta se ejecuta con un enfoque basado en comportamientos. Existen tres comportamientos corriendo de forma concurrente: *Go-to-Goal-Point*, *Avoid-Obstacle* y *Collision-Risk*. Para integrar los tres comportamientos, se utilizó un árbitro basado en prioridades donde el primer comportamiento tiene la prioridad más baja y el último, la más alta.

5.2. Campos potenciales artificiales

En la sección 3.3 se explicó cómo planear una ruta cuando se tiene una representación del ambiente y la sección 3.1 se dedicó al diseño de leyes de control que permitan seguir la ruta planeada. Sin embargo, ¿qué sucede cuando el robot tiene que evadir obstáculos que no están en el mapa y que no fueron contemplados en la planeación de la ruta?

Un forma de evadir obstáculos es mediante campos potenciales artificiales (Khatib, 1986).Una función potencial es una función real diferenciable $U : R^n \rightarrow R$ que puede verse como una función de energía potencial y que, por lo tanto, su gradiente $\nabla U(q)$, donde q es la posición del robot, representa fuerza.

El gradiente de la función, dado por

$$\nabla U(q) = \left[\frac{\partial U}{\partial q_1}, \ldots, \frac{\partial U}{\partial q_n}\right] \tag{5.1}$$

es un vector que apunta en la dirección de máximo cambio de U. Si esta función potencial se diseña de modo que tenga un mínimo en la posición meta y máximos locales en la posición de cada obstáculo que se desee evadir, entonces se puede mover al robot mediante el algoritmo de descenso del gradiente y esto hará que el robot continúe moviéndose hasta encontrar un punto q^* en el cual $\nabla U(q^*) = 0$.

Los valores q que satisfacen $\nabla U(q^*) = 0$ son llamados puntos críticos y pueden ser máximos locales, mínimos locales o puntos silla. Para determinar la naturaleza de un punto crítico, se puede utilizar la matriz Hessiana, dada por

$$H(q) = \begin{bmatrix} \frac{\partial^2 U}{\partial q_1^2} & \cdots & \frac{\partial^2 U}{\partial q_1 \partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 U}{\partial q_n \partial q_1} & \cdots & \frac{\partial^2 U}{\partial q_n^2} \end{bmatrix} \tag{5.2}$$

Si $H(q)$ evaluada en q^* es no singular, significa que q^* es un punto crítico aislado. Si $H(q^*)$ es positiva definida, entonces q^* es un mínimo local, si es negativa definida, se trata de un máximo local, y en otro caso, q^* es un punto silla. Si $H(q^*)$ es singular, entonces significa que q^* no es un punto aislado, es decir, que la función potencial U es "plana" en la región que contiene a q^*.

Si el robot se mueve siempre en sentido contrario al gradiente, en algún momento llegará a un mínimo local. Puede suceder que la condición inicial del robot corresponda a un máximo local o a un punto silla, en cuyo caso el robot no se moverá, pues en estos puntos el gradiente es cero. Esta situación es muy improbable, además, cualquier perturbación hará que el robot "se mueva" de ese punto y entonces se dirigirá al mínimo local, es decir, los puntos silla y los máximos son puntos inestables.

5.2.1. Diseño mediante campos atractivos y repulsivos

Como se expuso previamente, la función potencial $U(q)$ debe diseñarse de modo que sea diferenciable, que tenga un mínimo en el punto meta, máximos

locales en cada obstáculo, y todos sus puntos críticos deben ser aislados. Una forma sencilla de lograrlo es diseñando por separado dos campos: uno atractivo y otro repulsivo. La idea es que el punto meta debe generar una "fuerza atractiva" mientras que cada obstáculo debe generar una "fuerza repulsiva" (Oroko y Nyakoe, 2014).

Algunas veces no es necesario diseñar primero el campo potencial y luego obtener su gradiente, sino que se puede diseñar directamente la fuerza correspondiente al gradiente. La fuerza atractiva se debe diseñar de modo que siempre se aleje del punto meta y las fuerzas repulsivas siempre deben tener una dirección que apunte hacia el obstáculo que se desea evadir, además, las fuerzas repulsivas deben crecer conforme la distancia a dicho obstáculo disminuye y ser muy pequeñas o cero, si el robot está lo suficientemente lejos.

Es importante recordar que las fuerzas tanto atractiva como repulsiva corresponden al gradiente de la función potencial, es decir, representan la dirección de máximo cambio, sin embargo, lo que se desea es alcanzar un mínimo, por lo que al aplicar el algoritmo de descenso del gradiente, el robot se moverá en sentido contrario a estas fuerzas.

Una desventaja de generar campos potenciales mediante la suma de un campo atractivo y uno repulsivo es la posibilidad de tener mínimos locales. Esto se puede solucionar con la generación de funciones potenciales mediante otros métodos como el algoritmo de *wavefront*. Este algoritmo no tiene el problema de los mínimos locales pero tienen la desventaja de necesitar de una discretización del espacio.

5.2.2. Movimiento por descenso del gradiente

Una vez diseñado el campo potencial y calculado el respectivo gradiente, sólo basta con mover al robot de acuerdo con el algoritmo 5. La constante $\alpha > 0$ debe ser lo suficientemente pequeña para evitar oscilaciones pero sin que ello conlleve un costo computacional muy alto.

Aunque en el algoritmo se indica que el robot seguirá moviéndose mientras la magnitud del gradiente sea mayor que cero, en una implementación real se fija una tolerancia y cuando la magnitud es menor que ésta, se detiene al robot.

Finalmente, para que el robot se mueva hacia cada punto q_i generado por el descenso del gradiente, se pueden utilizar las leyes de control (3.5)-(3.6).

Algoritmo 5: Descenso del gradiente para mover al robot a través de un campo potencial.

Datos:
Posición inicial q_s, posición final q_g, posiciones q_{oi} de los obstáculos
Resultado:
Secuencia de puntos $\{q_0, q_1, q_2, \dots\}$

$q_0 \leftarrow q_s$
$i \leftarrow 0$
mientras $\|\nabla U(q_i)\| > 0$ **hacer**
  $q_{i+1} \leftarrow q_i - \alpha \nabla U(q_i)$
  $i \leftarrow i + 1$
fin

5.3. Combinación de comportamientos

5.3.1. El comportamiento *Go-To-Goal-Point*

Este comportamiento toma como señales de entrada las posiciones deseadas calculadas por el planeador de rutas (ver sección 3.3) y la posición del robot estimada por los algoritmos de localización (ver capítulo 4). La salida de este comportamiento son las velocidades de las llantas calculadas de acuerdo con las leyes ce control (3.10)-(3.11) explicadas en el capítulo 3. Se considera que el comportamiento está activo siempre que hay una nueva ruta a seguir.

5.3.2. El comportamiento *Avoid-Obstacles*

En el sistema propuesto se utilizaron campos potenciales artificiales como método de evasión de obstáculos. El campo potencial se diseñó empleando campos atractivos y repulsivos. Dado que el movimiento estará guiado por el gradiente ∇U y no por la función U en sí, es más sencillo diseñar directamente el gradiente ∇U, es decir, diseñar las fuerzas de atracción y repulsión. En esta propuesta se utilizaron las fuerzas propuestas por Arambula y Padilla (2011) dadas por

$$F_a = \zeta \frac{(q - q_g)}{\|q - q_g\|}, \qquad \zeta > 0 \tag{5.3}$$

$$F_r = \begin{cases} \eta \left(\sqrt{\frac{1}{d} - \frac{1}{d_0}} \right) \frac{q_{oi} - q}{d} & \text{si} \quad d < d_0 \\ 0 & \text{en otro caso} \end{cases} \tag{5.4}$$

donde q es la posición del robot, q_g es el punto meta, q_{oi} es la posición del i-ésimo obstáculo y $d = \|q_{oi} - q\|$ es la distancia del robot al obstáculo i. Estas fuerzas repulsivas tienen tres parámetros de diseño: $\zeta > 0$, $\eta > 0$ y $d_0 > 0$. La constante ζ define qué tan grande es la fuerza de atracción y η, qué tan grande es la de repulsión. d_0 es la distancia de influencia, esto es, aquellos objetos que estén a una distancia del robot mayor que d_0 producirán una fuerza de repulsión de cero y no afectarán el movimiento del robot.

La fuerza resultante se calcula como la suma de la fuerza atractiva y la media de todas las fuerzas repulsivas:

$$F = F_a + \frac{1}{n} \sum_{i=1}^{n} F_r \tag{5.5}$$

Una vez calculado el gradiente, el robot se mueve utilizando el algoritmo 5 (descenso del gradiente). La constante α debe ser lo suficientemente pequeña para evitar oscilaciones en el movimiento, considerando también que entre más pequeña se tendrá un costo computacional más alto. Finalmente, el robot se mueve hacia el punto q_i utilizando las leyes de control expuestas en el el capítulo 3.

Diseñar la función potencial U utilizando campos atractivos y repulsivos tiene el problema de los mínimos locales, es decir, el robot podría detenerse en algún punto (mínimo local) debido a que el gradiente ∇U en ese punto sea cero sin que éste sea el punto meta. Sin embargo, este problema se puede resolver si los comportamientos se coordinan correctamente, como se explicará en las siguientes secciones.

Es importante notar que, aunque el algoritmo 5 considera las posiciones de todos los obstáculos q_{oi}, no es necesario tener una representación del ambiente que contenga dichas posiciones. Dado que el robot contiene sensores de distancia, particularmente, un sensor láser, cada lectura se considera como un obstáculo y por lo tanto los campos potenciales pueden calcularse directamente de las lecturas de los sensores. En la figura 5.1 se muestra un

ejemplo del comportamiento *Avoid-Obstacles* donde el movimiento se ejecuta de acuerdo con las ecuaciones (5.3)-(5.5).

5.3.3. El comportamiento *Collision-Risk*

El objetivo de este comportamiento es detener al robot si se detecta un riesgo de colisión. Dado que el sensor láser con que cuenta el robot sólo genera lecturas en un plano, no es posible detectar obstáculos que no cruzan dicho plano, es por ello que se utilizó una cámara RGB-D montada en la cabeza del robot. El criterio para determinar si existe un riesgo de colisión es la detección de un obstáculo en frente del robot, pero sólo si éste se moverá hacia adelante. Para determinar la presencia de un obstáculo, se cuenta el número de píxeles en la imagen RGB-D que tienen una coordenada z mayor que un umbral y que están dentro de un área rectangular en frente del robot. Si el conteo excede un umbral K_obs, entonces se considera que hay un obstáculo. Considere la figura 5.2, el cuadro gris claro frente a la base móvil es la región de detección definida por la caja envolvente $[(x_{min}, y_{min}), (x_{max}, y_{max})]$. La detección de riesgo de colisión se lleva a cabo de acuerdo con el algoritmo 6.

La salida de este comportamiento es siempre cero dado que está pensado para detener al robot en caso de un riesgo de colisión, es decir, las velocidades de las llantas que genera este comportamiento son siempre cero. Se considera que el comportamiento está activado en cualquier momento que se cumplan las condiciones descritas en el algoritmo 6.

5.3.4. El árbitro

La figura 5.3 muestra un diagrama de bloques del movimiento basado en comportamientos. El árbitro selecciona el comportamiento activado que tenga la mayor prioridad. El comportamiento *Go-To-Goal-Point* se activa cada vez que se calcula una nueva ruta que se desea seguir. *Avoid-Obstacles* se activa cuando la fuerza repulsiva calculada es mayor que cero, es decir, hay obstáculos cuya distancia al robot es menor que d_0. El comportamiento *Collision-Risk* está activo cuando se detecta un riesgo de colisión. Con este esquema, el robot es capaz de evadir obstáculos inesperados y también se resuelve el problema de los mínimos locales en los campos potenciales, ya que cuando un obstáculo está enfrente del robot, lo que podría producir la condición del mínimo local, el comportamiento de riesgo de colisión se activa, el robot se detiene y entonces una nueva ruta se calcula usando un nuevo mapa

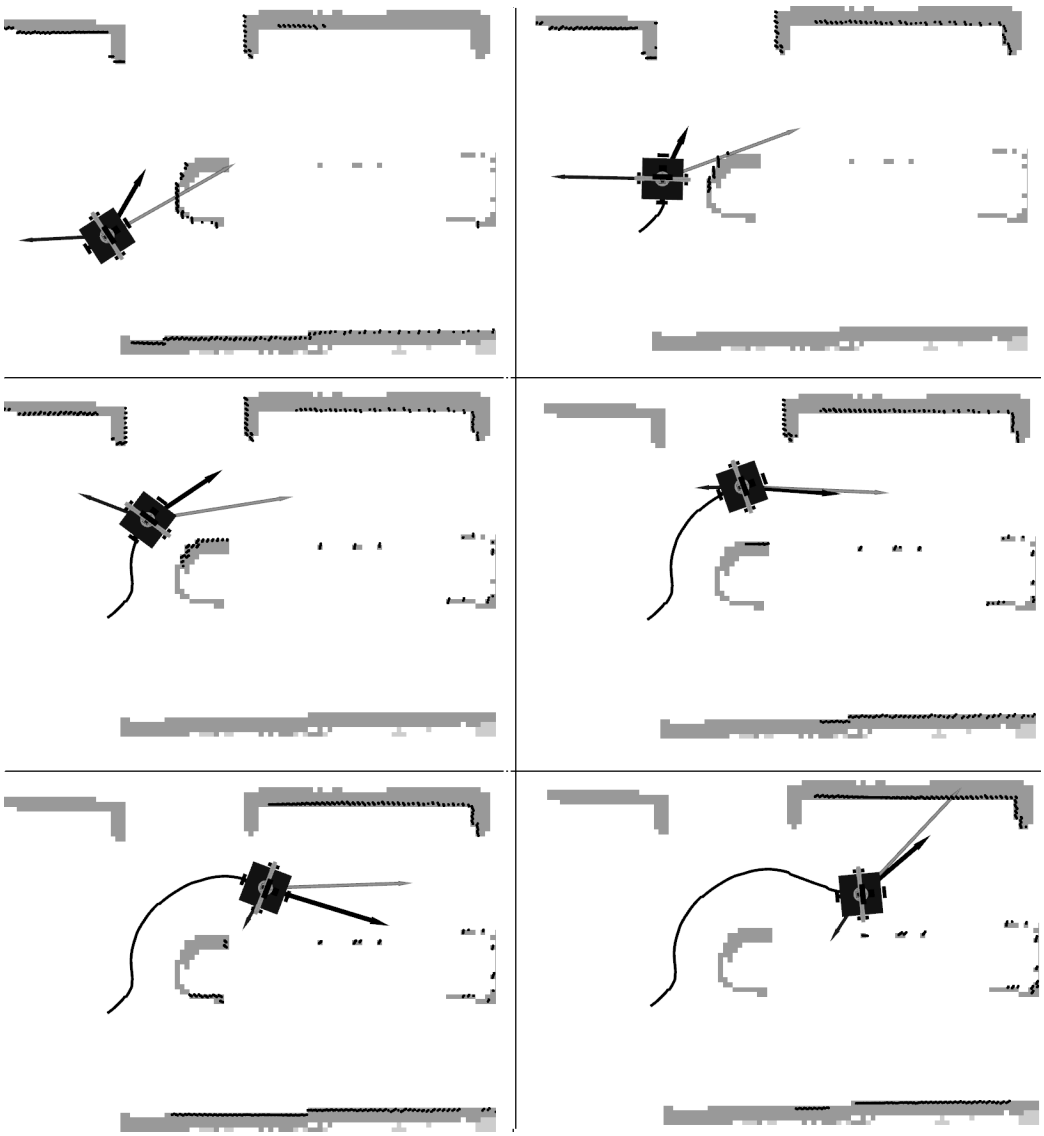


Figura 5.1: Las fuerzas repulsiva (flecha chica negra), atractiva (flecha chica gris) y resultante (flecha negra grande) con las constantes $\eta = 6.0$, $\zeta = 2.0$ y $d_0 = 1.5$, para diferentes posiciones. Los puntos negros son las lecturas del sensor láser y la línea negra detrás del robot es la ruta descrita.

Algoritmo 6: Detección de riesgo de colisión

Datos:
Nube de puntos $P = \{p_j | p_j = (x_j, y_j, z_j), j \in [1, n]\}$
Señales de control v y ω
Número mínimo de puntos $K_{obstacle}$
Mínima velocidad lineal v_{fwd} para considerar que el robot se mueve hacia enfrente
Resultado:
Indicador de riesgo de colisión

$i \leftarrow 0$
Riesgo de colisión $\leftarrow$ Falso
para cada $p_j \in P$ **hacer**
  si $x_j > x_{min} \wedge x_j < x_{max} \wedge y_j > y_{min} \wedge y_j < y_{max}$ **entonces**
    $i \leftarrow i + 1$
  fin
fin
si $v > v_{fwd} \wedge i > K_{obstacle}$ **entonces**
  //El número de puntos es mayor que el umbral y el robot se mueve hacia enfrente
  Riesgo de colisión $\leftarrow$ Verdadero
fin
Regresar "riesgo de colisión"

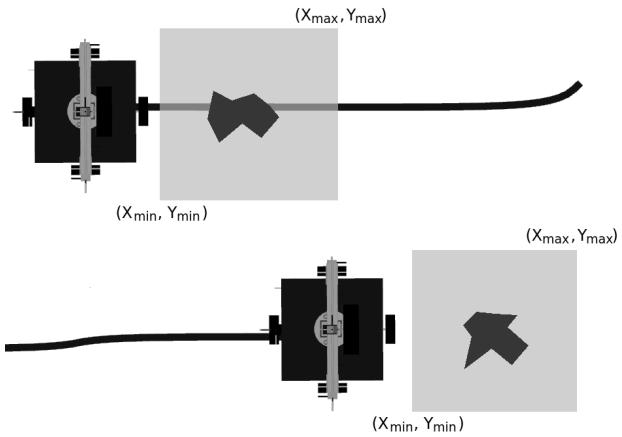


Figura 5.2: **Arriba:** se detecta riesgo de colisión, dado que el obstáculo está sobre la ruta del robot. **Abajo:** aunque el obstáculo está frente al robot, éste no está sobre la ruta, por lo tanto, no hay riesgo de colisión.

construido como se describe en la sección 4.3.2. En el capítulo 7 se describen los resultados obtenidos con este esquema basado en comportamientos para evasión de obstáculos.

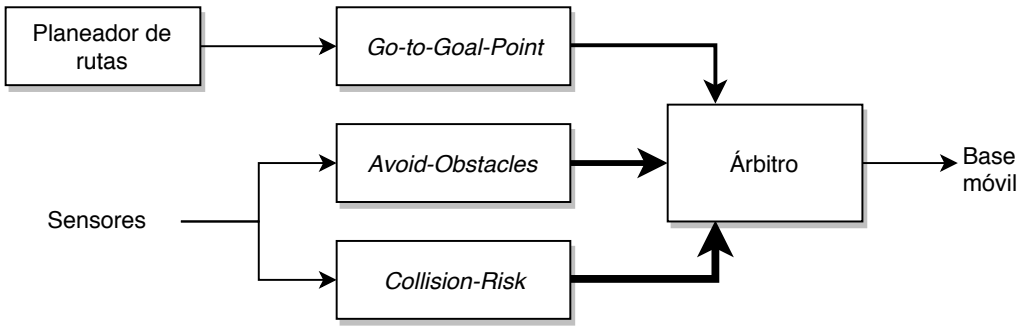


Figura 5.3: Árbitro basado en prioridades. El grosor de la línea indica la prioridad de cada comportamiento.

Capítulo 6

Aplicación de modelos cognitivos

La psicología cognitiva y las ciencias de la computación (más específicamente, la inteligencia artificial) son dos disciplinas con una amplia región de intersección debido a la naturaleza de sus objetos de estudio. Revisando algunos de sus conceptos y premisas básicas, se puede ver que la psicología cognitiva supone que la mente humana se puede considerar como un sistema computacional que actúa, a través de un conjunto de procesos, sobre estructuras de datos (Pashler y Medin, 2004). La inteligencia artificial se define como el estudio del comportamiento inteligente logrado por medios computacionales (Brachman y Levesque, 2004). En esta última definición se encuentran conceptos como *comportamiento* e *inteligente* que son objeto de estudio de la psicología cognitiva. De forma similar, *procesos* y *estructuras de datos*, que son parte de la definición de psicología cognitiva, se encuentran como objetos de estudio de las ciencias de la computación. Como se afirma en Gershman, Horvitz, y Tenenbaum (2015), la psicología cognitiva y la ciencias de la computación se pueden enriquecer mutuamente compartiendo conceptos, hallazgos y metodologías.

La psicología comparativa es el campo que se encarga de estudiar las similitudes y diferencias entre el comportamiento en humanos y animales. El campo que estudia más específicamente los procesos cognitivos en el comportamiento humano y no humano es la cognición comparativa (Zentall y Wasserman, 2012). Percepción, atención, aprendizaje, memoria y categorización son algunos de los conceptos que se estudian en animales dentro de la cognición comparativa y sus hallazgos han hecho importantes contribu-

ciones al estudio de la cognición humana. Los conceptos utilizados en este trabajo son aquellos relacionados con la cognición espacial, es decir, cómo los animales navegan en un determinado ambiente.

En este capítulo se presentan algunos conceptos relacionados con la navegación en animales como son la integración de la información, uso de marcas en el ambiente, integración de rutas y navegación basada en mapas. Posteriormente se explica cómo se utilizaron estos conceptos para mejorar la localización en el sistema de navegación para el robot de servicio. Finalmente se hace una discusión sobre otros conceptos relacionados y cómo se podrían utilizar para mejorar otras tareas en el sistema de navegación propuesto.

6.1. Conceptos sobre navegación en animales

6.1.1. Integración de información para localización

La integración de información se refiere a la forma en que un organismo pondera o discrimina dos o más fuentes de información para realizar una estimación espacial. Cheng, Shettleworth, Huttenlocher, y Rieser (2007) proponen que la integración se realiza de una forma bayesiana, donde el peso asignado a cada fuente al realizar la ponderación es inversamente proporcional a la magnitud de la incertidumbre estimada para esa fuente. Esto se puede expresar como

$$Y = W_{x0}X_0 + W_{x1}X_1 + ... + W_{xn}X_n \tag{6.1}$$

donde X_i es cada una de las fuentes de información y W_i es el peso de cada una de ellas calculado como

$$W_i = \frac{\sigma_i^2}{\sigma_0^2 + \sigma_1^2 + ... + \sigma_n^2} \tag{6.2}$$

donde σ_i^2 es la varianza asociada a la fuente X_i que se toma como medida de incertidumbre. De (6.1) y (6.2) se puede observar que entre más grande es la varianza, es decir, mayor incertidumbre, menor peso se le otorga a esa fuente de información.

En general, se dan tres casos de integración de información para realizar estimaciones espaciales:

- Dos fuentes actuales se combinan para hacer una estimación.

- Una fuente actual se combina con información previa.
- Una fuente actual se combina con información categórica que puede o no estar derivada de experiencia previa.

Como ejemplo de la primera se tiene el *efecto ventrílocuo*, en el que se integran dos fuentes de información: la vista, correspondiente al movimiento de la boca, y el oído, correspondiente a la voz de la persona. Si el muñeco está cerca de la boca de la persona que habla, ambos estímulos se percibirán como provenientes de la misma fuente, es decir, se ponderarán de manera similar, y se tendrá la ilusión de que el muñeco habla. Sin embargo, si la voz está lo suficientemente alejada del muñeco, es decir, existe una discrepancia grande entre los estímulos, entonces se empezarán a percibir como dos fuentes distintas y la ilusión desaparecerá.

El experimento del astronauta intentando atrapar una pelota es un ejemplo del caso dos. Aquí la fuente de información actual es la posición observada de la pelota y la fuente previa es la estimación de la posición de la pelota con base en la experiencia previa correspondiente a las estimaciones aprendidas en la tierra. En general, los astronautas tenderán a sobrestimar la velocidad de la pelota.

El tercer caso se ejemplifica con el experimento de pedir a una persona que recuerde la posición de un punto dentro de un círculo. En general, la estimación de la posición tenderá a estar cerca del centro de alguno de los cuatro cuadrantes que dividen un círculo.

6.1.2. Integración de rutas

La integración de rutas se refiere a la capacidad que tienen los animales de estimar la distancia que han recorrido únicamente con base en el tiempo, es decir, sin hacer correcciones de la estimación con base en claves o marcas observadas en el ambiente. Esta capacidad se ha observado en experimentos con roedores en los que las claves dadas al animal son movidas de su posición original. Cuando este desplazamiento es muy grande, se observa que el animal comienza a ignorar las claves y logra llegar a la meta aun cuando la discrepancia en éstas es muy grande. El flujo óptico, el esfuerzo requerido para recorrer cierta distancia y la percepción de luz polarizada del cielo están entre los mecanismos usados por animales para realizar la integración de rutas (Cheng, 2012).

La integración de rutas puede considerarse como un sistema de *backup*. Cuando la discrepancia en las marcas es grande, el animal comenzará a usar sólo las estimaciones temporales. Expresado en la forma de la ecuación (6.1), marcas e integración de rutas corresponderían a dos fuentes de información X_0 y X_1 con varianzas σ_0^2 y σ_1^2. Cuando las claves del ambiente discrepan mucho, se tiene una σ_0^2 muy grande y por lo tanto, de acuerdo con (6.2), se tendrá un peso W_{x0} muy pequeño.

6.2. Aplicación en robots de servicio

6.2.1. Integración de información en el robot Justina

El sistema de navegación del robot Justina (ver sección 7.1) utiliza el sensor láser para extraer marcas del ambiente y utiliza dichas marcar para localizarse. Cabe mencionar que un problema actual muy fuerte en la robótica es el de la visión computacional. En estos días sigue siendo complicado reconocer un objeto, localizarlo en el ambiente y distinguirlo de otros parecidos. Esta es una limitante al momento de querer implementar un sistema de navegación que esté basado principalmente en la observación de marcas del ambiente, como proponen Cheng et al. (2007) que los animales lo hacen. Por el momento, las claves más sencillas de extraer a partir de las lecturas de los sensores, son claves de naturaleza geométrica: planos, líneas, esquinas, espigas y otras. El robot Justina utiliza la extracción de líneas a partir de las lecturas del láser para localizarse (ver sección 4.2.

Del sensor láser se pueden extraer múltiples líneas y, dependiendo de los objetos del ambiente, éstas pueden ser muy confiables o bien, ser sólo ruido. Ponderando la posición calculada a partir de cada línea, se puede estimar la posición del robot. Cada línea observada con el sensor es comparada con las líneas almacenadas en un mapa construido previamente. La distancia entre una línea observada y una conocida se usa como medida de incertidumbre, por lo que, entre más alejadas estén entre sí, de acuerdo con (6.1), menor será el peso asignado a esas líneas. Como se explica en el capítulo 4, la posición del robot se estima como

$$P_{Rm} = P_R + \sum_i w_i e_{ko_i} \tag{6.3}$$

donde P_R es la estimación actual de la posición del robot, e_{ko_i} es el error de posición i-ésimo calculado de acuerdo con (4.5)-(4.10) y w_i es un peso

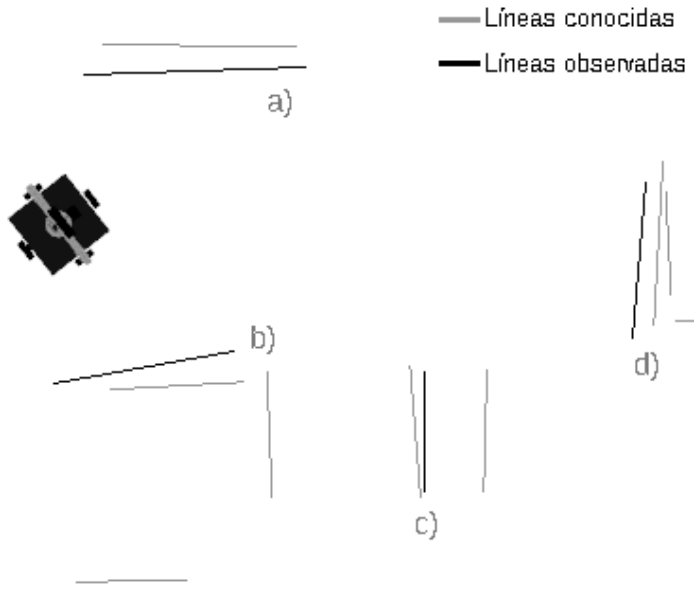


Figura 6.1: Ejemplo de líneas observadas y conocidas para la estimación de posición

inversamente proporcional a la distancia entre la línea conocida y la línea observada, calculado como

$$w_i = \frac{d_i}{d_0 \cdots + d_i + \cdots + d_n} \tag{6.4}$$

donde d_i es la distancia entre la línea observada y la línea conocida calculada de acuerdo con (4.4). Es decir, la posición se estima ponderando los pares de líneas empatados de acuerdo con la discrepancia entre lo observado y lo que se espera observar (líneas conocidas). La figura 6.1 muestra un ejemplo del conjunto de marcas conocidas y observadas.

6.2.2. Uso de la odometría en lugar de marcas

Como se mencionó anteriormente, se ha observado en animales que cuando existe mucha discrepancia en las claves del ambiente, estos comienzan a usar integración de rutas para navegar en lugar de las claves (Cheng et al., 2007). En el caso del robot Justina, se hace algo muy similar. Dentro de la representación geométrica del ambiente se incluyen regiones marcadas como

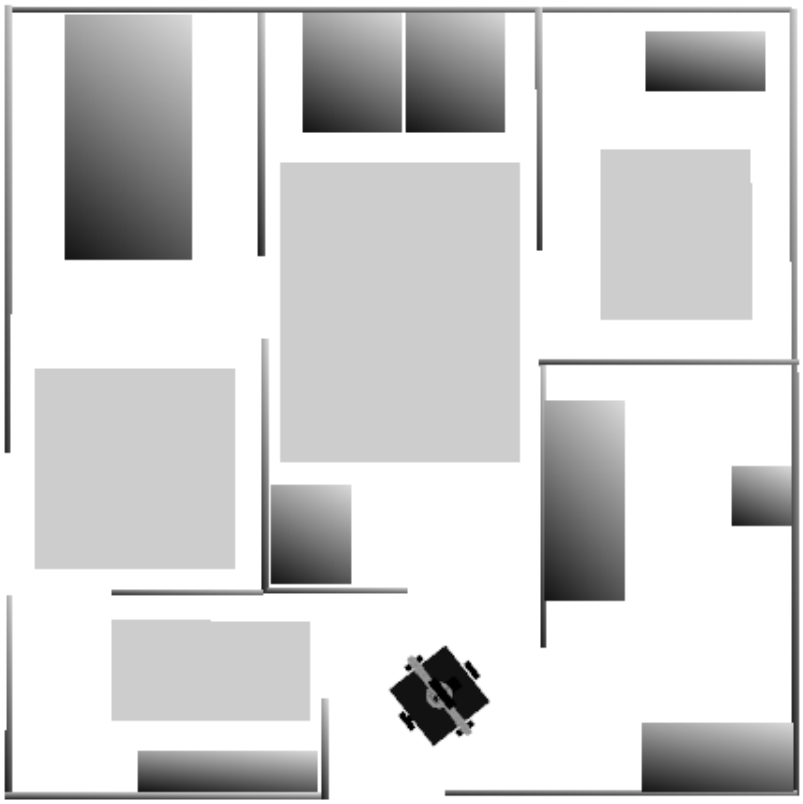


Figura 6.2: Los bloques en gris oscuro difuminado son objetos en el ambiente (paredes, muebles) y las regiones en gris claro son aquellas en las que es confiable estimar la posición con Filtro de Kalman.

"buenas" para localizar al robot. El que una región sea buena o no para la navegación se decide *a priori* mediante pruebas experimentales. En la figura 6.2 se puede observar el mapa geométrico de una casa. Las regiones color gris oscuro indican los lugares en los que el robot puede "confiar" en la extracción de líneas. Si el robot está fuera de estas zonas, basará su navegación únicamente en la odometría, que es el equivalente robótico de la integración de rutas.

De forma similar, aunque el robot esté dentro de una región marcada como buena para localización, si la incertidumbre en la extracción de marcas, calculada de acuerdo con (4.11) y (6.4), es muy grande, entonces, de forma similar a lo descrito en Cheng et al. (2007), el robot usará la odometría como

sistema de *backup*. Como se mencionó en la sección 4.2, la estimación de la posición con el Filtro de Kalman no se lleva a cabo todo el tiempo, sino que se tienen regiones marcadas como confiables

6.2.3. Mapas geométricos o claves del ambiente

Existe una discusión, en lo que se refiere a la navegación en animales, sobre si estos usan una representación geométrica o un mapa cognitivo (como se discute en Cheng, Huttenlocher, y Newcombe (2013)), o bien basan su navegación únicamente en el reconocimiento de marcas del ambiente. En Gallistel (1998) se menciona que si los animales tienen una representación simbólica del ambiente, en algún momento deberán hallarse los sustratos neuronales que permitan esta representación, del mismo modo en que en una computadora los transistores son el sustrato material de las operaciones simbólicas que ésta realiza.

En el caso de la navegación en robots móviles el enfoque que se ha tomado es el de la representación geométrica. Técnicas muy usadas como el mapeo y localización simultáneos, mapas topológicos, mapas de visibilidad y otros (Choset et al., 2005), tienen como objetivo la construcción o uso de algún tipo de representación geométrica del ambiente. Esto también es debido a la dificultad que todavía existe para el reconocimiento de lo que en robótica se conoce como *marcas naturales*, esto es, reconocer objetos comunes (muebles, puertas, pasillos, cuartos, etc) y distinguirlos de otros parecidos. Debido a esta dificultad, es complicado diseñar un sistema de navegación confiable que esté basado únicamente la observación de claves del ambiente.

6.3. Discusión sobre la aplicación en otros sistemas del robot

En la sección anterior se explicó cómo se utilizaron ciertos hallazgos hechos en el área de la cognición comparativa para mejorar la estrategia de localización en el sistema de planeación de movimientos propuesto. En esta sección se discuten algunos otros conceptos que se hallan implícitamente utilizados en los robots de servicio con el fin de sentar bases para futuras aplicaciones de los modelos cognitivos en el desarrollo de robots de servicio doméstico.

6.3.1. Representaciones estructuradas y sistemas expertos

Un conjunto de conceptos estudiados en la psicología cognitiva y que pueden encontrarse en el desarrollo de robots móviles autónomos es el de aquellos relacionados con la representación del conocimiento. Éste se define como la forma en que la información es almacenada y utilizada en los procesos psicológicos (Pashler y Medin, 2004). Esta teoría asume que la mente es un tipo de sistema computacional y que las personas manipulan información por medio de procesos que actúan sobre estructuras de datos.

De acuerdo con Brachman y Levesque (2004), en las ciencias de la computación, la representación del conocimiento es la rama de la inteligencia artificial que estudia la manipulación de símbolos, que representan un conjunto de proposiciones, para generar nuevas representaciones de nuevas proposiciones.

Para que una representación pueda clasificarse como tal, es necesario que existan cuatro elementos:

- Un mundo representado
- Un mundo que representa
- Un conjunto de reglas de representación
- Un conjunto de procesos para extraer y hacer uso de la información contenida en la representación

En general, existen cuatro tipos de representaciones (Pashler y Medin, 2004):

- Modelos espaciales
- Modelos por características
- Redes semánticas
- Representaciones estructuradas

Los modelos espaciales suponen que el mundo que representa es un espacio geométrico donde cada eje de coordenadas es una dimensión psicológica como color, forma, etc. Este tipo de representaciones son muy utilizadas para modelar similitudes entre los conceptos. Cada concepto corresponde a un punto

en el espacio y la similitud se calcula con una métrica r de Minkowski, dada por

$$\left[\sum_i \left(|x_{i1} - x_{i2}|\right)\right]^{\frac{1}{r}} \tag{6.5}$$

donde x_{i1} y x_{i2} son puntos en el espacio que representan dos conceptos diferentes. Una de las ventajas de este tipo de representaciones es que se pueden construir de manera sistemática a partir de juicios dados por participantes además del hecho de que existen formas de expresar la representación en forma matemática. Entre las principales desventajas se encuentra el hecho de que las personas relacionan más una lámpara y una silla, por ejemplo, que una lámpara con la luna, a pesar de que existen más similitudes como la forma y el brillo.

Los modelos por características proponen que los conceptos del mundo representado están definidos por conjuntos de características en el mundo representante. Podrían verse como una versión discreta de los modelos espaciales. Las similitudes entre conceptos se determinan a partir de las operaciones básicas de la teoría de conjuntos: unión, intersección y resta.

Entre las ventajas de este tipo de representaciones se tiene que dan acceso explícito a las características que hacen que dos elementos sean similares, mientras que las representaciones espaciales, no. Además explican, mediante el *conjunto de características comunes*, juicios de similitud que en una representación espacial violarían la propiedad del triángulo, como el caso de la pelota-luna-lámpara. También se apoyan en la evidencia neurológica de que el sistema visual extrae características de la imagen de entrada. Las principales desventajas son que no hay un método para determinar qué información debería incluirse en las características y que asume que todas las características son independientes, por lo que este modelo no puede dar información sobre las relaciones entre las características.

El tercer tipo de representaciones, las redes semánticas, proponen representar la información mediante un grafo dirigido, donde cada nodo equivale a un concepto, y cada vértice, a la relación existente entre un par de conceptos. En la figura 6.3 se muestra un ejemplo de red semántica.

El último tipo de representaciones, las estructuradas, pueden considerarse como una generalización de las redes semánticas, en donde las relaciones ya no son sólo binarias sino múltiples. Por ejemplo, el concepto `compact-disc-player` tendría varios conceptos asociados:

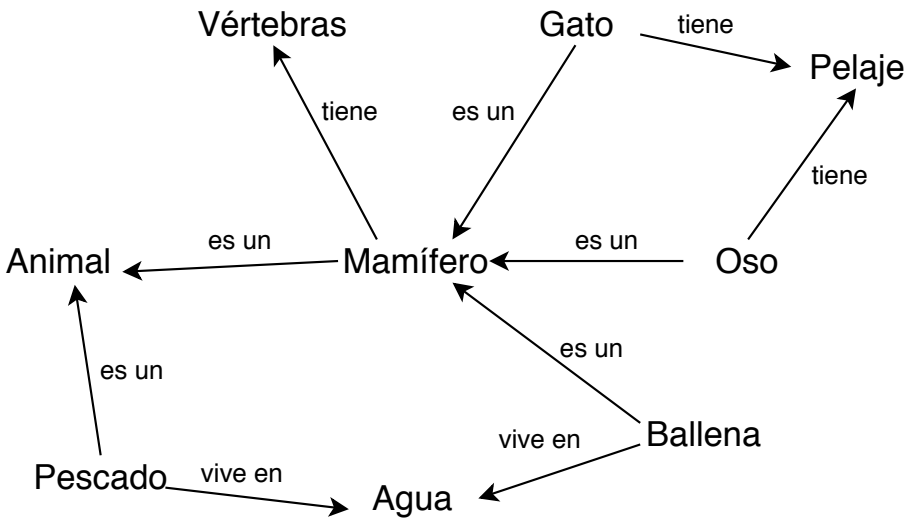


Figura 6.3: Ejemplo de red semántica

```
compact-disc-player:
  Color: Black}
  Function: Play music}
  Has-Parts: Disc-holder, Buttons, Volume-control...}
  Used-with: compact-disc}
```

Las representaciones estructuradas tienen la gran ventaja de poder utilizarse en la construcción de sistemas expertos y sistemas basados en reglas, que son construidos a partir de conjuntos muy grandes de reglas de producción, es decir, de pares SI-ENTONCES (Jackson, 1998).

6.3.2. Representaciones espaciales y reconocimiento de objetos

Una habilidad importante con que debe contar un robot de servicio es el reconocimiento y localización de objetos. Este proceso generalmente consta de dos grandes pasos: la etapa de entrenamiento y la etapa de prueba. En la etapa de entrenamiento se buscan las características más representativas de los objetos que se intentan reconocer. En la fase de prueba, las características

detectadas se comparan con la base de datos del robot para determinar el objeto que está siendo observado.

En la tarea de reconocimiento de objetos, las representaciones espaciales juegan un papel importante, debido a que, como se mencionó, cuentan con un equivalente matemático, lo que las hace fáciles de implementar en una computadora. En este caso, cada eje en el espacio representante corresponde a la magnitud de alguna característica. Existen muchas formas de cuantificar las características de un objeto. Un algoritmo muy usado en la actualidad es la trasformación de características invariante a escala, o SIFT por sus siglas en inglés. Con este algoritmo a cada objeto se le extrae un conjunto de puntos característicos, llamados *keypoints*, y éstos se cuantifican, a su vez, mediante un conjunto de 128 valores.

De hecho, SIFT es una combinación de una representación espacial y una por características. Es una representación espacial porque, para determinar si un *keypoint* de la imagen observada corresponde o "se parece" a un *keypoint* almacenado en la memoria del robot, se utiliza una métrica como aquella dada por (6.5), esto es, cada punto es almacenado en un espacio métrico de 128 dimensiones y la similitud se determina por la distancia entre dos de ellos. Es también una representación por características porque cada objeto representado en la memoria del robot consta de un conjunto de *keypoints*. Para determinar si se está observando un objeto, se realiza una intersección entre el conjunto de *keypoints* observados y el conjunto de *keypoints* correspondientes a un objeto.

En la figura 6.4 se muestran las fases de entrenamiento y prueba, a la izquierda y derecha, respectivamente, de la transformada SIFT. Los puntos blancos son aquellos observados que resultaron ser "parecidos" (con base en la métrica r) a algún punto almacenado en la memoria del robot. Los puntos unidos por una línea gris claro son los que están en la intersección de los conjuntos *keypoints*-observados y *keypoints*-de-un-objeto.

Las representaciones estructuradas también son muy usadas en la robótica pero esta vez en el área de planeación de acciones. Como se mencionó anteriormente, son fácilmente implementables gracias a que pueden usarse con la notación de cálculo de predicados y existen lenguajes de programación que usan esta misma notación tales como CLIPS y Prolog (llamados lenguajes lógicos). Los sistemas expertos son sistemas construidos con conjuntos grandes de pares SI-ENTONCES que utilizan representaciones estructuradas para definir los antecedentes y consecuentes en un sistema de producciones. Un programa escrito en lenguaje lógico ejecutará las instrucciones contenidas en

Figura 6.4: Ejemplo de uso del algoritmo SIFT

el consecuente siempre que se presenten las condiciones establecidas en el antecedente. Por ejemplo, las producciones

```
(defrule InferLocation_Livingroom
    (or (and (sofa)(tv))
        (and (sofa)(center_table))
        (and (tv)(center_table))
    )
    =>
    (assert (update_location_livingroom))
)

(defrule InferLocation_Kitchen
    (or (and (stove)(fridge))
        (and (dishes)(fridge))
        (and (stove)(dishes))
    )
    =>
    (assert (update_location_kitchen))
)
```

definen el conjunto de condiciones que deben cumplirse para actualizar la posición del robot. De acuerdo con el ejemplo, si se observan cualesquiera dos

de tres objetos, *sofa*, *tv* o *center_table*; entonces se concluye que la posición actual es *livingroom*. De forma similar, si se observan *stove*, *dishes* o *fridge*, se concluye que la ubicación es *kitchen*. El robot Justina utiliza un sistema experto de este tipo como planeador de acciones.

Estos conceptos se presentan con la finalidad de dar contexto al trabajo futuro que se discutirá en el capítulo 8.

Capítulo 7

Pruebas y resultados

En este capítulo se presentan los resultados del desempeño general del sistema. La efectividad de cada uno de lo módulos se ha mostrado en el correspondiente capítulo, por ello, a continuación se muestran sólo las pruebas relacionadas con el desempeño del sistema completo. Se comienza por describir a los robots de servicio Justina y *Human Support Robot*, que fueron las plataformas en las que se realizaron todas las pruebas. Posteriormente se describen los resultados tanto en simulación como con los robots reales. El desempeño de la propuesta se comparó con el paquete *nav2d* mediante varias pruebas estadísticas. Finalmente, se presentan resultados sobre el desempeño del sistema en el contexto de la competencia internacional de robótica Robocup@Home. Estos últimos resultados son importantes dado que en esta competencia el robot se desenvuelve en ambientes más parecidos a los que se tendrían en una aplicación real en ambientes domésticos.

7.1. Los robots de servicio Justina y *Human Support Robot*

Justina es un robot de servicio doméstico construido en el Laboratorio de Biorrobótica de la Facultad de Ingeniería de la UNAM y cuyo software está desarrollado bajo la arquitectura ViRBot (ver sección 2.3). Este robot y sus predecesores han participado en varias competencias nacionales e internacionales en las que se evalúa el desempeño general de este tipo de robots, tales como la liga Robocup@Home (Wachsmuth et al., 2015) y el RoCKIn@Home (Amigoni et al., 2015). En estas competencias, Justina ha llevado a cabo

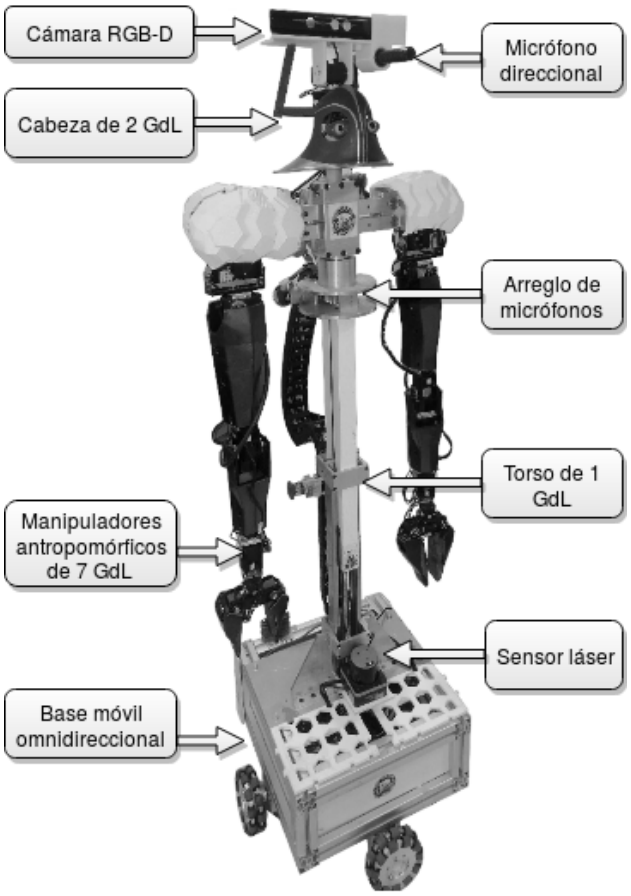


Figura 7.1: El robot de servicio doméstico Justina.

varias tareas como limpiar una mesa, servir bebidas, entre otros comandos dados por humanos.

Justina cuenta con varios sensores: una cámara RGB-D y una cámara RGB, dos sensores láser, uno al frente y otro en la parte posterior, un micrófono direccional y un arreglo de micrófonos, así como encoders en cada motor. Los actuadores de Justina consisten en una base móvil omnidireccional, dos brazos antropomórficos de siete grados de libertad, un torso que permite cambiar la altura del robot, y una cabeza con movimientos pan y tilt. La figura 7.1 muestra la disposición de los sensores y actuadores mencionados.

El robot *Human Support Robot* (HSR) es una plataforma desarrollada por

Figura 7.2: El robot de servicio doméstico HSR de Toyota.

la empresa Toyota para fines de investigación en robots de servicio doméstico (`https://www.toyota-global.com/innovation/partner_robot/`). Es un producto terminado aunque aún no se encuentra disponible para su venta al público, estando su uso restringido a las instituciones y grupos de trabajo que la misma empresa autoriza.

En la figura 7.2 se muestra al robot HSR y la disposición de su hardware. Los actuadores con que cuenta este robot son similares a los del robot Justina: una base móvil omnidireccional, un torso, un brazo de cinco grados de libertad, una cabeza con movimientos de pan y tilt y una herramienta de succión que permite tomar objetos planos ligeros como hojas de papel. Con respecto a los sensores, este robot cuenta con una cámara RGB-D, un sensor láser, dos cámaras RGB en la cabeza para aplicaciones de visión estereoscópica, una cámara en el brazo, detectores de impacto en la base móvil y encoders en todos los motores.

Los robots Justina y HSR sirvieron como plataformas de pruebas para el sistema propuesto.

7.2. La plataforma ROS

ROS (del inglés Robot Operating System) es una plataforma de código abierto que provee la funcionalidad comúnmente necesaria en el desarrollo de software para robots móviles autónomos, como paso de mensajes y manejo de paquetes (Quigley et al., 2015). El sistema de navegación propuesto se desarrolló siguiendo la arquitectura ViRBot y la implementación se realizó utilizando las herramientas provistas por ROS.

ROS puede describirse en dos niveles conceptuales: el sistema de archivos y el grafo de procesos. El sistema de archivos se refiere a la forma en que se organizan los recursos en disco y el grafo de procesos a la forma que interactúan los programas desarrollados en tiempo de ejecución.

En cuanto al sistema de archivos, son importantes los siguientes conceptos:

- ***Workspace:*** Se refiere a las carpetas que contienen paquetes de ROS.
- **Paquete:** Es la principal unidad de organización de software en ROS. Puede contener nodos, bibliotecas, datasets, archivos de configuración, entre otros.
- **Manifiesto:** Definido por el archivo `package.xml` en cada paquete. Provee metadatos acerca de cada paquete.
- **Mensaje:** Archivos con extensión `.msg`. Definen estructuras de datos para el paso de mensajes en ROS.
- **Servicio:** Archivos con extensión `.srv`. Definen estructuras de tipo *request-response*. Utilizan mensajes para dicha definición.

El grafo de procesos es una red *peer-to-peer* de procesos. Los componentes básicos son:

- **Roscore:** Inicializa el sistema ROS: un máster + *rosout* + un servidor de parámetros.
- **Nodos:** Es simplemente un ejecutable que usa los estándares de ROS para comunicarse con otros nodos.
- **Tópicos:** Algo similar a una variable cuyo contenido puede ser compartido entre todos los nodos mediante un patrón de publicación y suscripción.

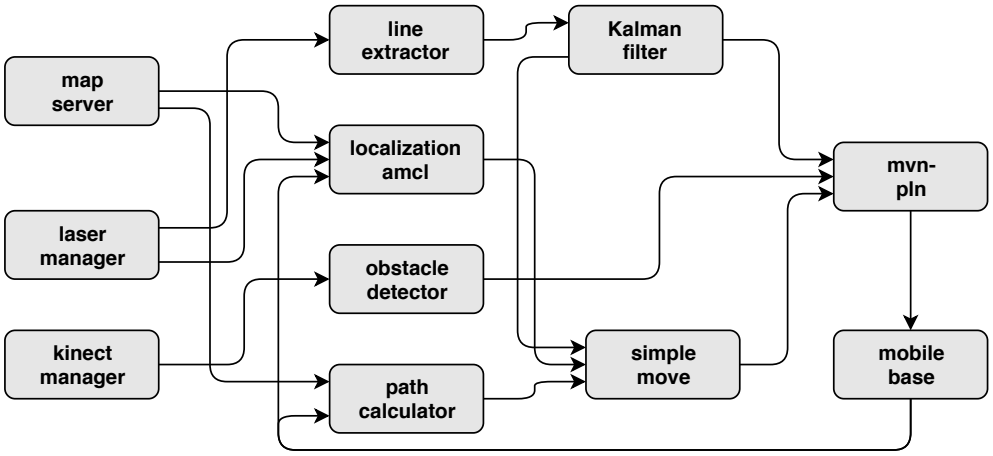


Figura 7.3: Implementación del sistema propuesto en ROS. Los cuadros representan nodos y las flechas, la comunicación entre ellos, empleando ya sea tópicos o servicios.

- **Servicios:** Otra forma de comunicar nodos pero con un patrón de petición y respuesta. Puede verse como una función que puede ser llamada por cualquier nodo.
- **Servidor de parámetros:** Es un diccionario compartido. Todos los nodos pueden leer y escribir parámetros en tiempo de ejecución.

La figura 7.3 muestra un diagrama de bloques de los nodos en los que se implementó el sistema propuesto.

7.3. Resultados en simulación

Para probar el desempeño de la propuesta, se comparó con el paquete *nav2d*. Para ello se generaron diez mapas artificiales consistentes en polígonos con formas y posiciones aleatorias. Además, para cada mapa se generaron cien puntos aleatorios dentro del espacio libre para utilizarlos como puntos meta en la ejecución de movimientos. La figura 7.4 muestra ejemplos de los mapas artificiales generados y los puntos aleatorios. Para los resultados en simulación también se usaron dos mapas obtenidos de ambientes reales: del Laboratorio de Biorrobótica y de la arena @Home de la Robocup 2018 (mostrado en la figura 7.7).

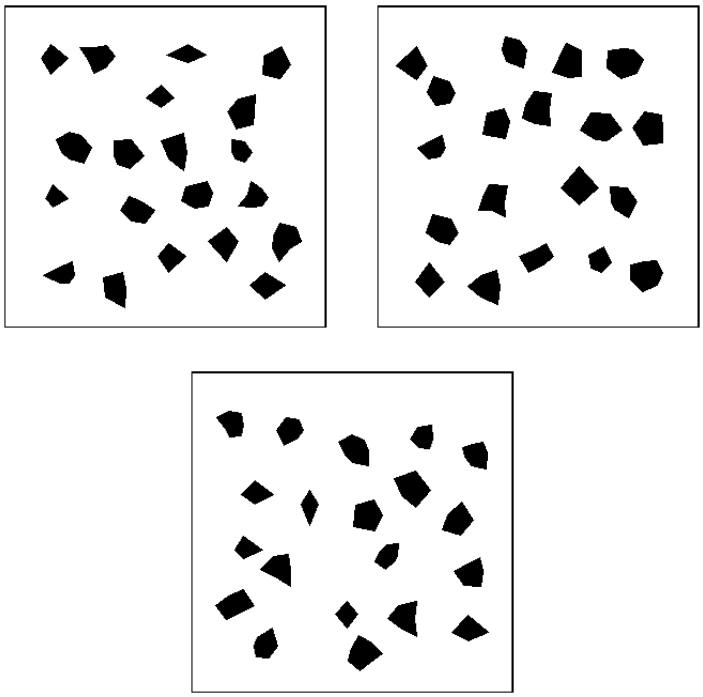


Figura 7.4: Ejemplos de los mundos usados para probar la planeación de rutas y los puntos meta aleatorios.

El paquete *nav2d* se utilizó con todas las constantes y los parámetros en sus valores por defecto, excepto para la máxima velocidad lineal, la cual se fijó en 0.7 m/s para ser la misma que aquella usada en las leyes de control (3.10)-(3.11); y el *map inflation radius* (ver documentación del paquete) cuyo valor se fijó en 0.2 que fue el valor más grande que permitió al paquete *nav2d* calcular rutas a través de las puertas del ambiente.

El control de bajo nivel se implementó utilizando las constantes $\alpha = 0.6$, $\beta = 0.09$, $\omega_{max} = 1.0$ y $V_{max} = 0.7$. La planeación de rutas se implementó de acuerdo con los descrito en el capítulo 3. Para la detección de riesgo de colisión se empleó la caja envolvente dada por $x_{min} = 0.3$, $y_{min} = -0.25$, $x_{max} = 0.9$, $y_{max} = 0.25$ con constantes $K_{obstacle} = 30$, $K_{\Pi} = 0.05$ y $v_{fwd} = 0.1$. Para los campos potenciales artificiales se emplearon las constantes $\zeta = 1.0$, $\eta = 5.0$ y $d_0 = 0.8$.

La línea base (el paquete *nav2d*) y la propuesta fueron probados con 100 puntos aleatorios para cada mapa. La comparación se hizo con base en

tres parámetros: la tasa de la distancia viajada a la distancia euclideana (TDEDR), velocidad promedio en metros por segundo (AMpS) y número de colisiones (NoC). TDEDR se calcula como el cociente de la distancia total recorrida entre la distancia euclidiana del punto inicial al punto meta. Este parámetro sirve para medir qué tan eficiente es una ruta en cuanto a la distancia recorrida. AMpS se calcula como la distancia total recorrida entre el tiempo total que le tomó al robot recorrer dicha distancia. Nótese que no corresponde al tiempo de cálculo de trayectoria sino al tiempo de recorrido para seguir dicha trayectoria. Este tiempo se puede ver afectado por un mal control de posición o por maniobras de evasión de obstáculos innecesarias. Finalmente, el parámetro NoC es un conteo del número de veces que el robot chocó con algún obstáculo. Puesto que estos son resultados en simulación, se pueden detectar las colisiones sin que sea necesario detener al robot. Como se describirá en la siguiente sección, este parámetro fue diferente para las pruebas con el robot real, en las que no es permisible que el robot colisione con obstáculos.

La tabla 7.1 muestra la media y la desviación estándar del TDEDR para los 100 puntos aleatorios para cada mapa. Para determinar si hay una diferencia significativa entre ambos sistemas de navegación se realizó una prueba t-Student de dos colas. La última columna muestra los estadísticos y el valor p para estas pruebas. De forma similar, las tablas 7.2 y 7.3 muestran la media, desviación estándar y parámetros de la prueba t-Student para los parámetros AMpS y NoC respectivamente.

La tabla 7.1 muestra que el TDEDR fue mayor para el sistema propuesto que para la línea base, es decir, la distancia viajada usando el paquete *nav2d* fue significativamente menor que la distancia usando el sistema propuesto en 11 de 12 ambientes probados (usando una confianza del 95 %). Esto se debe a la función de costo descrita en la sección 3.3. Dado que la cercanía a los obstáculos es considerada como parte de la función de costo, las rutas en general son más largas, puesto que el robot tiende a moverse lejos de los obstáculos. La figura 7.5 muestra una comparación de las rutas calculadas por el sistema propuesto y por el paquete *nav2d*. Se pueden obtener rutas más seguras con el paquete *nav2d* si se incrementa el radio de inflación de obstáculos, sin embargo, esto provoca que dicho paquete no pueda calcular rutas a través de espacios reducidos, como las puertas, por ejemplo. A pesar de las distancias más largas, el robot alcanzó los puntos meta en tiempos más cortos, como se puede ver en la tabla 7.2, donde se muestra que el parámetro AMpS (metros por segundo promedio) fue significativamente mayor para el

	Propuesta		Nav2d		Significancia	
Mundo	Media	Desv. Est.	Media	Desv. Est.	t	valor p
Biorobotics L.	1.29	0.39	1.18	0.20	2.45	0.01534
@Home arena	1.56	0.98	1.19	0.18	3.73	0.00030
Random 1	1.31	0.35	1.12	0.20	4.41	1.88E-5
Random 2	1.26	0.27	1.13	0.22	3.76	0.00022
Random 3	1.30	0.24	1.10	0.13	7.55	3.4E-12
Random 4	1.31	0.26	1.11	0.15	6.62	5.5E-10
Random 5	1.27	0.22	1.13	0.21	4.43	1.54E-5
Random 6	1.24	0.21	1.11	0.17	4.73	4.22E-6
Random 7	1.24	0.18	1.10	0.13	6.19	3.83E-9
Random 8	1.22	0.27	1.15	0.22	1.96	0.05032
Random 9	1.22	0.27	1.13	0.26	2.44	0.01540
Random 10	1.26	0.22	1.08	0.14	7.01	5.6E-11

Tabla 7.1: Tasa entre la distancia viajada y la distancia euclidiana (TDEDR) para la propuesta y la línea base (el paquete *Nav2d*) con ambientes simulados. Se evaluaron cien rutas para cada mapa.

sistema propuesto que para la línea base.

Para ambos sistemas de navegación, se contó el número de colisiones (NoC), considerando como tales cualquier superposición de alguna parte de la proyección 2D de la base móvil con alguna celda del mapa. La tabla 7.3 muestra las estadísticas de NoC para las cien rutas para cada mapa. Como se puede ver, el NoC con el sistema propuesto fue significativamente menor para todos los mapas, esto como resultado de las rutas más seguras calculadas por el planeador de rutas. La figura 7.5 muestra una comparación de las rutas calculadas por la línea base y por el sistema propuesto.

7.4. Resultados con el robot real

De forma similar a los resultados en simulación, se probó el sistema propuesto y el paquete *nav2d* con veinte puntos aleatorios en el Laboratorio de Biorrobótica. Por razones de seguridad, una vez que se detectó que el robot iba a chocar contra algún obstáculo o incluso sólo rozarlo, se detuvo por completo. Por ello, contrario a las pruebas en simulación, en este caso no se tiene una medida del número de colisiones (Noc) sino una medida del número de

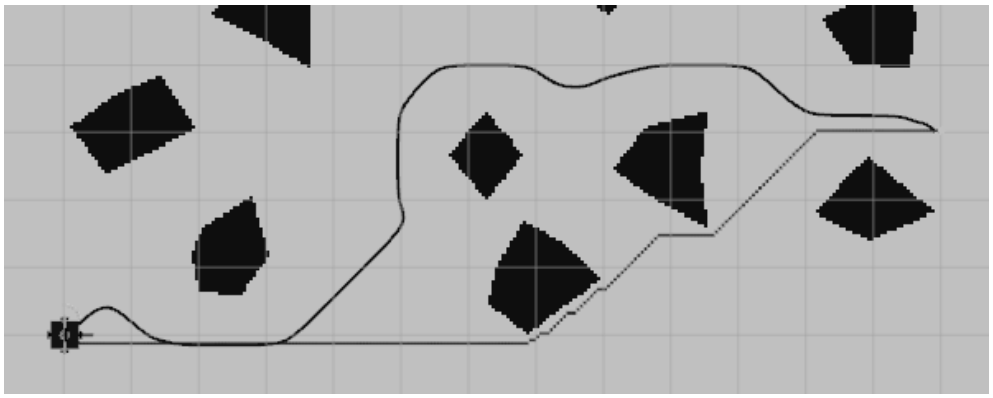


Figura 7.5: Comparación de rutas. **Abajo:** ruta calculada por el paquete *nav2d*. **Arriba:** ruta calculada por el sistema propuesto. Esta última es más segura.

	Propuesta		Nav2d		Significancia	
Mundo	Media	D. Est.	Media	D. Est.	t	valor p
Biorobotics L.	0.43	0.107	0.20	0.042	20.08	<2.2E-16
@Home arena	0.43	0.109	0.19	0.048	19.62	<2.2E-16
Random 1	0.41	0.110	0.20	0.046	17.98	<2.2E-16
Random 2	0.43	0.099	0.20	0.040	21.61	<2.2E-16
Random 3	0.46	0.091	0.20	0.033	22.81	<2.2E-16
Random 4	0.43	0.092	0.21	0.047	21.73	<2.2E-16
Random 5	0.42	0.086	0.20	0.037	23.42	<2.2E-16
Random 6	0.43	0.090	0.21	0.036	23.14	<2.2E-16
Random 7	0.44	0.091	0.20	0.034	24.64	<2.2E-16
Random 8	0.42	0.113	0.20	0.035	18.37	<2.2E-16
Random 9	0.42	0.111	0.20	0.046	18.32	<2.2E-16
Random 10	0.43	0.089	0.21	0.036	23.09	<2.2E-16

Tabla 7.2: Promedio de Metros por Segundo (AMpS) para la propuesta y la línea base (el paquete *Nav2d*) para las escenas simuladas. Cien rutas aleatorias fueron evaluadas por cada mapa.

	Propuesta		Nav2d		Significancia	
Mundo	Media	D. Est.	Media	D. Est.	t	valor p
Biorobotics L.	0.22	0.48	1.96	1.76	-9.51	3.7E-16
@Home arena	0.43	0.97	2.70	1.82	-11.00	<2.2E-16
Random 1	0.10	0.30	1.88	1.66	-10.55	<2.2E-16
Random 2	0.08	0.31	2.19	1.61	-12.90	<2.2E-16
Random 3	0.13	0.34	1.91	1.62	-10.74	<2.2E-16
Random 4	0.08	0.27	2.27	1.68	-12.86	<2.2E-16
Random 5	0.05	0.22	1.85	1.59	-11.20	<2.2E-16
Random 6	0.07	0.25	2.08	1.64	-12.12	<2.2E-16
Random 7	0.15	0.48	2.40	1.73	-12.50	<2.2E-16
Random 8	0.14	0.38	2.43	2.10	-10.71	<2.2E-16
Random 9	0.15	0.39	1.99	1.50	-11.82	<2.2E-16
Random 10	0.07	0.26	1.93	1.68	-10.92	<2.2E-16

Tabla 7.3: Número de colisiones (NoC) para la propuesta y la línea base (el paquete *Nav2d*) para los ambientes simulados. Se evaluaron cien rutas aleatorias por cada mapa.

puntos alcanzados (NoRGP). Para todos los puntos meta, se pusieron objetos enfrente del robot para verificar el desempeño de la evasión de obstáculos. Todos los parámetros y constantes fueron los mismos que para las pruebas en simulación. De los 20 puntos meta, con el sistema propuesto se alcanzaron 18, sin chocar ni tocar ningún obstáculo inesperado. Con el paquete *nav2d* sólo se alcanzaron 14 puntos.

La figura 7.6 muestra el desempeño del enfoque basado en comportamientos para la evasión de obstáculos. En la parte de arriba se muestra al robot al inicio del movimiento. En este punto, los comportamientos *Go-To-Goal-Point* y *Avoid-Obstacles* están activados. La imagen del centro muestra el momento en el que se detecta un riesgo de colisión y el robot se detiene. Aunque la persona estaba enfrente del robot, lo que pudo causar un mínimo local, el robot fue capaz de alcanzar la meta debido a la activación del comportamiento *Collision-Risk* y el consecuente cálculo de una nueva ruta, como se muestra en la parte de abajo de la figura.

La tabla 7.4 muestra las estadísticas para los parámetros TDEDR y AMpS para ambos sistemas de navegación. Como se puede ver, en el mundo real con obstáculos inesperados, la distancia media recorrida usando el paquete *nav2d*

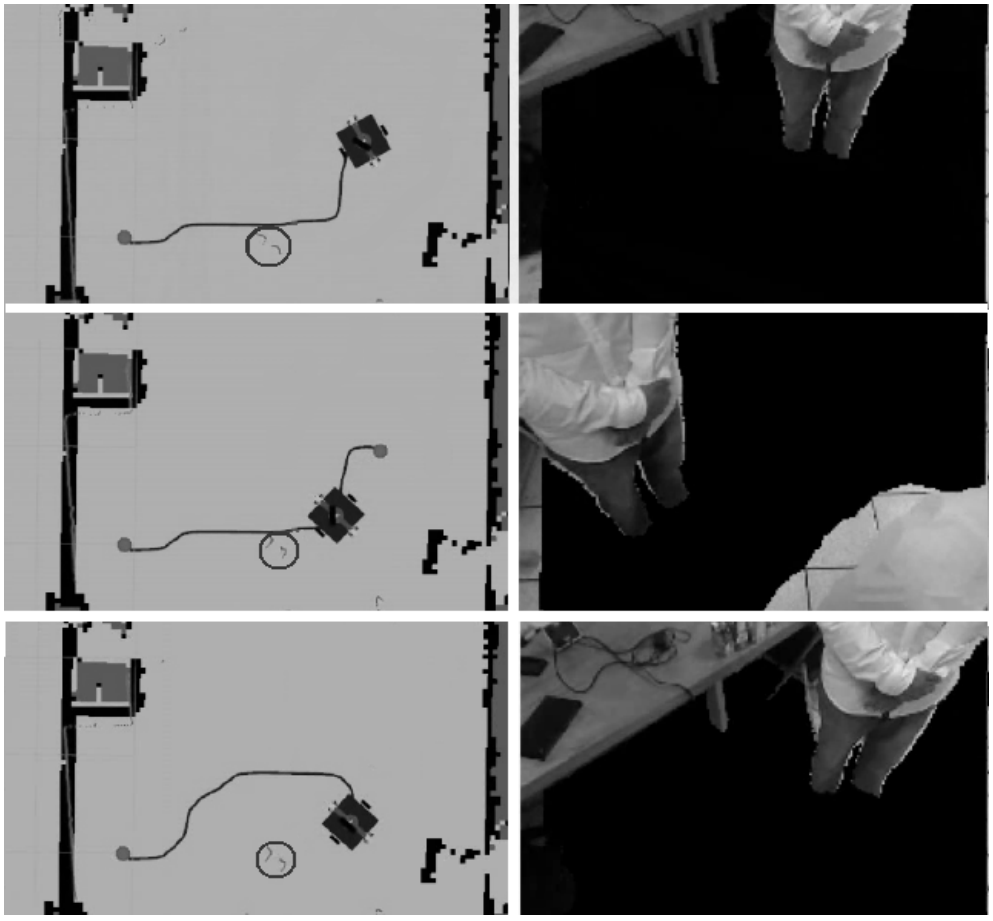


Figura 7.6: Evasión de obstáculos basada en comportamientos. **Arriba:** el robot comienza el movimiento. **Centro:** se detecta un riesgo de colisión. **Abajo:** se calcula una nueva ruta.

	Propuesta		Nav2d		Significancia	
Mundo	Media	Desv. Est.	Media	Desv. Est.	t	valor p
TDEDR	1.06	0.337	1.36	0.365	-2.6376	0.01253
AETPM	0.19	0.052	0.17	0.054	1.4153	0.1661
NoRGP	18		14			

Tabla 7.4: Estadísticas para TDEDR y AMpS para el sistema propuesto y la línea base. NoRGP es simplemente el conteo del número de puntos meta alcanzados.

fue significativamente mayor que la distancia con el sistema propuesto. Esto significa que el sistema propuesto evade obstáculos de forma más eficiente dado que viajó menores distancias. El parámetro AMpS no tuvo diferencia significativa, lo que es en sí un buen indicador: el sistema propuesto recorrió rutas más cortas lo que, a una velocidad media igual, significa menores tiempos de recorrido.

7.5. Competencias de robots de servicio

Para promover el desarrollo de robots móviles autónomos, existen competencias internacionales como Robocup (Ferrein y Steinbauer, 2016) y RoCKIn (Amigoni et al., 2015). Ambas competencias tienen la ventaja de proporcionar un problema estándar donde un amplio rango de tecnologías y enfoques pueden ser integrados, examinados y evaluados. Como se afirma en Iocchi, Holz, Ruiz-del Solar, Sugiura, y Van Der Zant (2015), las competencias son útiles cuando es necesario evaluar el desempeño general de un sistema completo y no sólo probar hipótesis aisladas. Por estas razones, el robot Justina y sus predecesores han participado en el Robocup@Home desde 2006 y en 2018 se participó también con el robot HSR. El sistema de planeación de movimientos descrito en este trabajo fue probado satisfactoriamente en la última edición de esta competencia, celebrada en Montreal, Canadá. En la liga @Home, la navegación fue necesaria en 6 de 7 pruebas (van Beek, Holz, Matamoros, Rascon, y Wachsmuth, 2018). En todas ellas, Justina y HSR realizaron satisfactoriamente todos los movimientos.

En la figura 7.7 se observa el mapa de la arena @Home donde se realizaron varias pruebas. La figura 7.8 muestra al robot Justina durante la prueba *Help Me Carry*. A grandes rasgos, en esta prueba el robot debía seguir a un

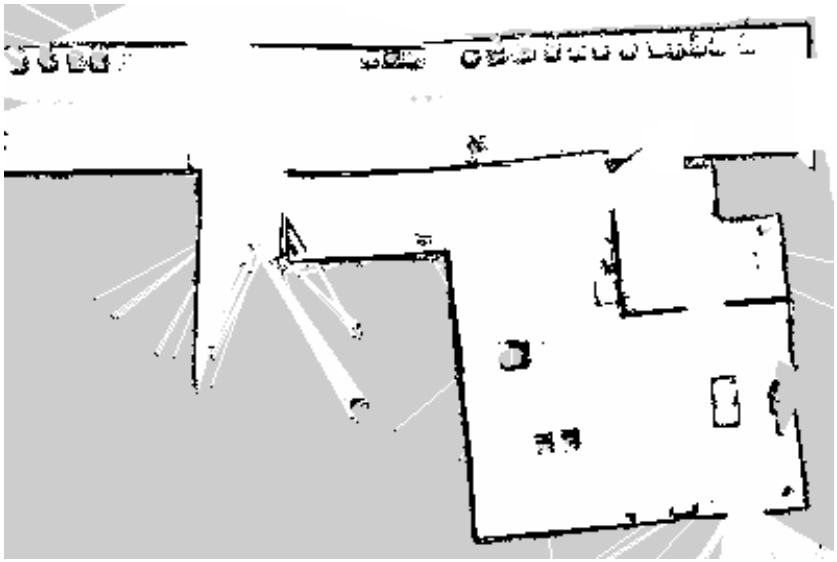


Figura 7.7: La arena @Home de la Robocup 2018. Aunque se observa un claro desvío en el mapa durante el proceso de SLAM, el sistema se desempeñó correctamente debido a la navegación activa.

humano hasta un punto determinado donde debía recoger una bolsa y llevarla de regreso a un punto dentro de la casa. Como se especifica en el libro de reglas, esta prueba está centrada en la navegación segura y la evasión de obstáculos. Esta última habilidad se evaluó colocando un objeto pequeño en la ruta del robot. La figura 7.8 muestra el momento en que Justina detectó el riesgo de colisión, se activó el comportamiento *Collision-Risk* y se calculó una nueva ruta.

Otra de las pruebas importantes durante la competencia fue la de *Restaurant*. En esta prueba el robot debe detectar que un comensal lo está llamando, acercarse, tomar la orden, regresar al lugar designado como bar, tomar los objetos correspondientes y llevarlos hasta el lugar del comensal que los pidió. Al igual que la prueba *Help Me Carry*, como lo indica el libro de reglas, esta prueba está centrada en la navegación robusta, entre varias otras características. El principal reto radica en el hecho de que se trata de un ambiente desconocido y altamente dinámico, pues se realiza en un restaurante real o algún comercio similar, por ello, en esta prueba, los comportamientos juegan un papel escencial. La figura 7.9 muestra al robot Justina durante esta prueba, en el momento en que se aproxima al comensal que ordenará alimentos.

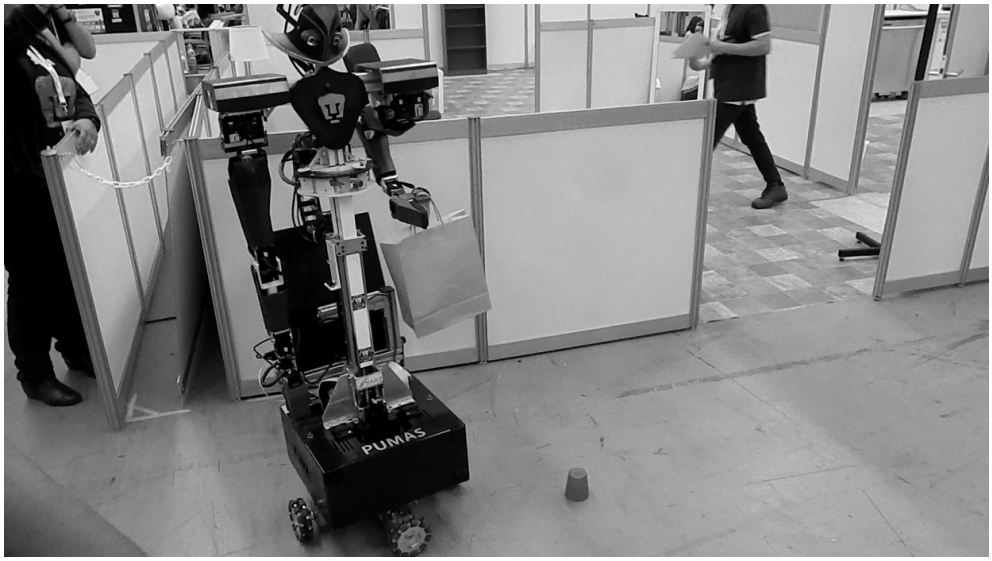


Figura 7.8: Evasión de obstáculos durante la prueba *Help Me Carry*.

La prueba GPSR (*General Purpose Service Robot*) tiene como fin evaluar todas las habilidades básicas con que debe contar un robot de servicio. Un operador selecciona aleatoriamente un comando de un conjunto muy grande de posibilidades y el robot debe ser capaz de planear las acciones necesarias para realizar dicho comando. El objetivo de esta prueba es evaluar la planeación de acciones, sin embargo, de manera implícita, se evalúa la navegación de los robots, pues si esta parte falla, el robot no podrá completar prácticamente ninguna tarea. En la figura 7.10 se muestra al robot HSR durante la prueba *GPSR*.

Figura 7.9: El robot Justina durante la prueba *Restaurant*.

Figura 7.10: El robot HSR durante la prueba *GPSR*.

Capítulo 8

Discusión

8.1. Conclusiones

Se desarrolló un sistema de planeación de movimientos tomando al ViRBot como arquitectura de diseño. El sistema desarrollado incluyó la localización, construcción de mapas, evasión de obstáculos y planeación y seguimiento de rutas, todas ellas tareas fundamentales en el problema de la planeación de movimientos, como se explicó en el capítulo 2.

El sistema se implementó en los robots Justina y HSR utilizando las herramientas provistas por la plataforma ROS. Se realizaron pruebas para evaluar el desempeño tanto en simulación como en ambientes reales, comparándolo con el paquete *nav2d*. Como parámetros de comparación se emplearon la tasa de la distancia recorrida contra la distancia euclidiana (TDEDR), los metros por segundo promedio (AMpS), el número de colisiones (NoC) y el número de puntos meta alcanzados (NoRGP). Las distancias recorridas usando el sistema propuesto fueron significativamente más largas, en las pruebas en simulación, que las distancias recorridas usando el paquete *nav2d*, de acuerdo con el parámetro TDEDR. Esto se debe a la función de costo utilizada en el algoritmo A* que resulta en rutas más largas pero más seguras. Esta seguridad de las rutas se verificó con el número de colisiones (NoC) significativamente menor para el sistema propuesto que para la línea base, en el caso de las pruebas en simulación, y con el mayor número de puntos meta alcanzados, en el caso de las pruebas con el robot real. Contrario a las pruebas en simulación, las distancias recorridas utilizando el sistema propuesto fueron significativamente más cortas, lo que se debe a una evasión de obstáculos

más eficiente, consecuencia del uso de comportamientos.

Se comprobó que el uso de la función de costo propuesta para el algoritmo A* permitió la planeación de rutas más seguras. Esto se comprobó al comparar el desempeño con el paquete *nav2d* que utiliza mapas de costos, técnica más comúnmente usada en la navegación en robots de servicio doméstico. El calcular la función de costo de manera inversamente proporcional a los objetos más cercanos, como parte del algoritmo de búsqueda, resultó en un mejor desempeño con respecto a la línea base.

La construcción de mapas de rutas en línea resultó viable como método para la evasión de obstáculos y esto se logró gracias a la implementación en paralelo. Como se mostró en el capítulo de resultados, la evasión de obstáculos fue satisfactoria, tanto en las pruebas en laboratorio como en las pruebas en competencia.

La estrategia de estimar la posición del robot mediante Filtro de Kalman Extendido sólo cuando éste se encuentra en regiones que han sido clasificadas como confiables para localización, fue satisfactoria. El robot logró mantenerse localizado incluso después de pasar por espacios desordenados donde la extracción de líneas no es posible o es poco confiable, esto debido a que en estas regiones el robot estima su posición sólo con base en la odometría. Esta estrategia estuvo inspirada en la integración de rutas e integración de la información, ambas características de la navegación en insectos. Como se discutió anteriormente, esta aplicación puede extenderse a otros hallazgos de la cognición comparativa que permitan mejorar el sistema de planeación de movimientos.

Como se planteó en la introducción de este trabajo, probar el sistema como un todo integrado es importante cuando hay realimentación entre los distintos componentes involucrados y que, por lo tanto, la confiabilidad del sistema no puede basarse en la confiabilidad de los componentes aislados. El desempeño del sistema propuesto se probó en varias competencias específicamente diseñadas para evaluar robots autónomos completamente integrados. Las pruebas *Help Me Carry*, *Restaurant* y *GPSR* de la Robocup@Home, con los robots Justina y HSR, se presentaron como ejemplos del buen desempeño del sistema de navegación en estas competencias.

8.2. Trabajo Futuro

Como trabajo futuro se plantea una comparación más extensiva con otras arquitecturas para robots se servicio en condiciones similares. Se realizará una documentación más extensiva de este sistema con la finalidad de que otros grupos de investigación puedan usarlo como paquete abierto, similar a *nav2d*.

La integración de técnicas de visión computacional también es parte del trabajo que se plantea como continuación de esta propuesta. La localización con base ya no sólo en líneas sino en marcas naturales, tales como muebles u objetos del hogar, puede hacer que la estimación de la posición sea más robusta. Como se discutió en el capítulo 6, varios autores afirman que animales como los insectos navegan basándose más en marcas del ambiente que en representaciones geométricas internas, sin embargo, un sistema de esta naturaleza aún es poco confiable en un robot de servicio doméstico debido al estado del arte en visión computacional, sin embargo, combinando técnicas de esta área con la navegación autónoma, se puede comenzar a desarrollar un sistema de planeación de movimientos más parecido al de un animal.

Para una mejor ejecución de los movimientos, se plantea desarrollar leyes de control con base ya no sólo en el modelo cinemático de la base móvil, sino con base en un modelo dinámico para tomar en cuenta efectos de inercia y fricción. Se plantea desarrollar leyes de control con base en técnicas más recientes como la pasividad o los modos deslizantes.

Todas las pruebas presentadas en este trabajo fueron hechas en interiores. La luz solar afecta considerablemente a sensores como el láser y las cámaras RGB-D, por lo que el uso en exteriores del sistema propuesto aún no es posible. Como parte del trabajo a realizar, se plantea desarrollar algoritmos de evasión de obstáculos que no requieran de la cámara RGB-D, así como la implementación de algoritmos de localización y mapeo que sólo requieran de imágenes RGB.

Finalmente, se espera poder utilizar varios de los módulos desarrollados en la planeación de movimientos para el robot de servicio no necesariamente relacionados con la navegación, por ejemplo, en la manipulación de objetos. Se plantea extender los módulos de planeación de rutas y representación del ambiente para aplicarlos en el problema de la manipulación de objetos.

Referencias

Amigoni, F., Bastianelli, E., Berghofer, J., Bonarini, A., Fontana, G., Hochgeschwender, N., ... others (2015). Competitions for benchmarking: task and functionality scoring complete performance assessment. *IEEE Robotics & Automation Magazine*, *22*(3), 53–61.

Arambula, F., y Padilla, M. (2011). Autonomus robot navigation using adaptive potential fields. *Mathematical and Computer Modelling*, *40*, 1141-1156.

Arkin, R. C. (1998). *Behavior-based robotics*. MIT press.

Avilés-Arriaga, H. H., Sucar, L. E., Morales, E. F., Vargas, B. A., Sánchez, J., y Corona, E. (2009). Markovito: A flexible and general service robot. En D. Liu, L. Wang, y K. C. Tan (Eds.), *Design and control of intelligent robotic systems* (p. 401-423).

Banino, A., Barry, C., Uria, B., Blundell, C., Lillicrap, T., Mirowski, P., ... others (2018). Vector-based navigation using grid-like representations in artificial agents. *Nature*, 1.

Bekey, G. A. (2005). *Autonomous robots: from biological inspiration to implementation and control*. MIT press.

Brachman, R., y Levesque, H. (2004). *Knowledge representation and reasoning*. Elsevier.

Braitenberg, V. (1986). *Vehicles: Experiments in synthetic psychology*. MIT press.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, *2*(1), 14–23.

Brooks, R. A. (1991). Intelligence without representation. *Artificial intelligence*, *47*(1-3), 139–159.

Burgard, W., Hebert, M., y Bennewitz, M. (2016). World modeling. En *Springer handbook of robotics* (pp. 1135–1152). Springer.

Cheng, K. (2012). Arthropod navigation: Ants, bees, crabs, spiders finding

their way. En *The oxford handbook of comparative cognition (2 ed.).* Oxford University Press.

Cheng, K., Huttenlocher, J., y Newcombe, N. S. (2013). 25 years of research on the use of geometry in spatial reorientation: a current theoretical perspective. *Psychonomic bulletin & review*, *20*(6), 1033–1054.

Cheng, K., Shettleworth, S. J., Huttenlocher, J., y Rieser, J. J. (2007). Bayesian integration of spatial information. *Psychological bulletin*, *133*(4), 625.

Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., y Thrun, S. (2005). *Principles of robot motion: theory, algorithms, and implementation.* MIT press.

Contreras, L., y Mayol-Cuevas, W. (2015). Trajectory-driven point cloud compression techniques for visual slam. En *Intelligent robots and systems (iros), 2015 ieee/rsj international conference on* (pp. 133–140).

Contreras, L., y Mayol-Cuevas, W. (2017). O-poco: Online point cloud compression mapping for visual odometry and slam. En *Robotics and automation (icra), 2017 ieee international conference on* (pp. 4509–4514).

Cruz, J. (2013). *Optimización de campos potenciales para navegación mediante algoritmos genéticos* (Tesis de Master no publicada). Universidad Nacional Autónoma de México.

Dissanayake, G., Huang, S., Wang, Z., y Ranasinghe, R. (2011). A review of recent developments in simultaneous localization and mapping. En *Industrial and information systems (iciis), 2011 6th ieee international conference on* (pp. 477–482).

Dissanayake, M. G., Newman, P., Clark, S., Durrant-Whyte, H. F., y Csorba, M. (2001). A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on robotics and automation*, *17*(3), 229–241.

Elbanhawi, M., Simic, M., y Jazar, R. N. (2015). Continuous path smoothing for car-like robots using b-spline curves. *Journal of Intelligent & Robotic Systems*, *80*(1), 23–56. doi: 10.1007/s10846-014-0172-0

Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation. *Computer*(6), 46–57.

Ferrein, A., y Steinbauer, G. (2016). 20 years of robocup. *KI-Künstliche Intelligenz*, *30*(3-4), 225–232.

Gallistel, C. (1998). Symbolic processes in the brain: The case of insect navigation. En D. Scarborough, S. Sternberg, y D. Osherson (Eds.),

An invitation to cognitiver science. vol 4. The MIT Press.

Gershman, S. J., Horvitz, E. J., y Tenenbaum, J. B. (2015). Computational rationality: A converging paradigm for intelligence in brains, minds, and machines. *Science*, *349*(6245), 273–278.

González, D., Pérez, J., Milanés, V., y Nashashibi, F. (2016). A review of motion planning techniques for automated vehicles. *IEEE Trans. Intelligent Transportation Systems*, *17*(4), 1135–1145.

Grisetti, G., Stachniss, C., y Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE transactions on Robotics*, *23*(1), 34–46.

IFR. (2017). *Service robots.* Descargado de `https://ifr.org/service-robots`

Iocchi, L., Holz, D., Ruiz-del Solar, J., Sugiura, K., y Van Der Zant, T. (2015). Robocup@ home: Analysis and results of evolving competitions for domestic and service robots. *Artificial Intelligence*, *229*, 258–281.

Iocchi, L., Ruiz-del Solar, J., y van der Zant, T. (2012). Domestic service robots in the real world. *Journal of Intelligent & Robotic Systems*, *66*(1), 183–186.

ISO. (2012). *Robots and robotic devices – vocabulary* (ISO n.º 8373-2012). Geneva, Switzerland: International Organization for Standardization.

Jackson, P. (1998). *Introduction to expert systems*. Addison-Wesley Longman Publishing Co., Inc.

Jin, B., Lei, H., y Geng, W. (2014). Accurate intrinsic calibration of depth camera with cuboids. En *European conference on computer vision* (pp. 788–803).

Karpov, V. (2014). Robot's temperament. *Biologically Inspired Cognitive Architectures*, *7*, 76 - 86. Descargado de `http://www.sciencedirect.com/science/article/pii/S2212683X13000972` doi: https://doi.org/10.1016/j.bica.2013.11.004

Katrakazas, C., Quddus, M., Chen, W.-H., y Deka, L. (2015). Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, *60*, 416–442.

Kavraki, L. E., y LaValle, S. M. (2008). Motion planning. En *Springer handbook of robotics* (pp. 109–131). Springer.

Kavraki, L. E., Svestka, P., Latombe, J.-C., y Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, *12*(4),

566–580.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, *5*(1), 90–98.

Kortenkamp, D., y Simmons, R. (2008). Robotic systems architectures and programming. En *Springer handbook of robotics* (pp. 187–206). Springer.

Kurup, U., y Lebiere, C. (2012). What can cognitive architectures do for robotics? *Biologically Inspired Cognitive Architectures*, *2*, 88 - 99. doi: https://doi.org/10.1016/j.bica.2012.07.004

Latombe, J. C. (1991). *Robot motion planning*. Kluwer Academic.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

LaValle, S. M., Branicky, M. S., y Lindemann, S. R. (2004). On the relationship between classical grid search and probabilistic roadmaps. *The International Journal of Robotics Research*, *23*(7-8), 673–692.

Lefèvre, S., Vasquez, D., y Laugier, C. (2014). A survey on motion prediction and risk assessment for intelligent vehicles. *ROBOMECH journal*, *1*(1), 1.

Lepetič, M., Klančar, G., Škrjanc, I., Matko, D., y Potočnik, B. (2003). Time optimal path planning considering acceleration limits. *Robotics and Autonomous Systems*, *45*(3), 199 - 210. doi: https://doi.org/10.1016/j.robot.2003.09.007

Linde, Y., Buzo, A., y Gray, R. M. (1980). An algorithm for vector quantizer design. *Communications, IEEE Transactions on*, *28*(1), 84–95.

Lozano-Pérez, T., y Wesley, M. A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, *22*(10), 560–570.

Lu, D. V., Hershberger, D., y Smart, W. D. (2014). Layered costmaps for context-sensitive navigation. En *Intelligent robots and systems (iros 2014), 2014 ieee/rsj international conference on* (pp. 709–715).

McCrae, J., y Singh, K. (2009). Sketching piecewise clothoid curves. *Computers & Graphics*, *33*(4), 452 - 461. doi: https://doi.org/10.1016/j.cag.2009.05.006

Müller, J. P. (1996). *The design of intelligent agents: a layered approach* (Vol. 1177). Springer Science & Business Media.

Murphy, R., Murphy, R. R., y Arkin, R. C. (2000). *Introduction to ai robotics*. MIT press.

Negrete, M., Savage, J., Cruz, J., y Márquez, J. (2014). Parallel implementation of roadmap construction for mobile robots using rgb-d cameras.

OGRW2014, 184.

Nilsson, N. (1969). A mobile automaton: An application of artificial intelligence techniques. En *Proc of the first international joint conference on artificial intelligence (ijcai-69)* (p. 509-520).

Oroko, J., y Nyakoe, G. (2014). Obstacle avoidance and path planning schemes for autonomous navigation of a mobile robot: a review. En *Proceedings of sustainable research and innovation conference* (pp. 314–318).

Paden, B., Čáp, M., Yong, S. Z., Yershov, D., y Frazzoli, E. (2016). A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, *1*(1), 33–55.

Pashler, H., y Medin, D. (2004). *Stevens handbook of experimental psychology, memory and cognitive processes* (Vol. 2). John Wiley & Sons.

Pineda, L., Sucar, E., Savage, J., Aceves, A., Becerra, H., Fuentes, G., ... Rodríguez, A. (2017). Robótica de servicio. En *La computación en méxico por especialidades académicas* (p. 127-166). Asociación Mexicana de Computación.

Pineda, L. A., Rodríguez, A., Fuentes, G., Rascon, C., y Meza, I. V. (2015). Concept and functional structure of a service robot. *International Journal of Advanced Robotic Systems*, *12*(2), 6.

Quigley, M., Gerkey, B., y Smart, W. D. (2015). *Programming robots with ros: a practical introduction to the robot operating system.* O'Reilly Media, Inc.

Savage, J., Contreras, L., Figueroa, I., Pacheco, A., Bermudez, A., Negrete, M., ... Rivera, C. (2016). Construction of roadmaps for mobile robots' navigation using rgb-d cameras. En *Intelligent autonomous systems 13* (pp. 217–229). Springer.

Savage, J., LLarena, A., Carrera, G., Cuellar, S., Esparza, D., Minami, Y., y Peñuelas, U. (2008). Virbot: a system for the operation of mobile robots. En *Robocup 2007: Robot soccer world cup xi* (pp. 512–519). Springer.

Savage, J., Rosenblueth, D. A., Matamoros, M., Negrete, M., Contreras, L., Cruz, J., ... Okada, H. (2019). Semantic reasoning in service robots using expert systems. *Robotics and Autonomous Systems*, *114*, 77 - 92. doi: https://doi.org/10.1016/j.robot.2019.01.007

Seib, V., Memmesheimer, R., y Paulus, D. (2016). A ros-based system for an autonomous service robot. En *Robot operating system (ros)* (p. 215-252). Springer.

Stückler, J., Schwarz, M., y Behnke, S. (2016). Mobile manipulation, tool use, and intuitive interaction for cognitive service robot cosero. *Frontiers in Robotics and AI*, *3*, 58.

Thrun, S., Burgard, W., y Fox, D. (2005). *Probabilistic robotics*. MIT press.

van Beek, L., Holz, D., Matamoros, M., Rascon, C., y Wachsmuth, S. (2018). *Robocup@home 2018: Rules and regulations.* `http://www.robocupathome.org/rules/2018_rulebook.pdf`.

Wachsmuth, S., Holz, D., Rudinac, M., y Ruiz-del Solar, J. (2015). Robocup@ home-benchmarking domestic service robots. En *Aaai* (pp. 4328–4329).

Yan, R., Wu, J., Wang, W., Lim, S., Lee, J., y Han, C. (2012). Natural corners extraction algorithm in 2d unknown indoor environment with laser sensor. En *Control, automation and systems (iccas), 2012 12th international conference on* (pp. 983–987).

Zeno, P. J., Patel, S., y Sobh, T. M. (2017). A novel neurophysiological based navigation system. *Biologically Inspired Cognitive Architectures*, *22*, 67 - 81. doi: https://doi.org/10.1016/j.bica.2017.09.002

Zentall, T. R., y Wasserman, E. A. (2012). *The oxford handbook of comparative cognition*. Oxford University Press.