Informe Final
Aprendizaje bioinspirado: Módulos para el TAM-WG

Pablo Scleidorovich
Juan Andrés Sanguinetti

Tutor:
Gonzalo Tejera

Instituto de Computación
Facultad de Ingeniería - Universidad de la República
Montevideo - Uruguay

9 de Noviembre 2015

Resumen

En este proyecto se comienza estudiando el estado del arte respecto a aprendizaje bioinspirado y SLAM.

Primero se estudia aprendizaje bioinspirado, centrándose específicamente en el problema de Spatial Learning (SL - navegación espacial). En base a este estudio, se propone realizar una implementación de uno de los modelos encontrados, el TAM-WG. Este modelo es combinación de otros dos llamados Taxon Affordance Model (TAM) y World Graph (WG). En ambos, se utilizan técnicas de aprendizaje por refuerzo (RL - reinforcemente learning) para hacer que un agente aprenda conductas de navegación espacial que imiten la forma en que aprenden los roedores.

Debido a la complejidad encontrada para realizar dicha implementación, el alcance del proyecto se termina acotando a la resolución de algunos de los problemas encontrados. En particular, el proyecto se termina centrando en desarrollar un algoritmo de SLAM métrico y mecanismos que muestren cómo los resultados obtenidos, pueden ser utilizados para resolver el problema original. Más especificamente, se desarrollan dos algoritmos que generan polígonos convexos, que representen la región del espacio en la que se encuentre el robot. Para ello, uno utiliza únicamente la información sensada en un momento en particular, mientras que el otro aprovecha además la información en el mapa, generando la partición del mismo en componentes convexas.

Como los algoritmos de estados convexos solo pretenden ser una prueba de concepto, es necesario continuar desarrollandolos para que puedan ser usados correctamente. Los resultados obtenidos en el proyecto muestran cómo la obtención de estos polígonos podría ser utilizada para definir los estados y acciones en los algoritmos de RL en los modelos TAM y WG. A diferencia de los estados en los modelos originales, estos tendrían la ventaje de proveer un conjunto reducido de acciones posibles por estado, disminuyendo de esta forma el tiempo necesario para el aprendizaje.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Motivación en Aprendizaje Bioinspirado

Los organismos biológicos se han caracterizado a lo largo de los años por haber evolucionado para actuar en un mundo con cambios rápidos y de alta incertidumbre, riqueza indefinida y una disponibilidad limitada de información. En cambio los sistemas industriales, operan generalmente en entornos cerrados y con escasa o nula incertidumbre. Sin embargo, los artefactos que quieran trabajar en el mundo real, deben ser capaces de hacer frente a situaciones de incertidumbre y reaccionar rápidamente ante los cambios en el entorno. Es por esto que es importante para los investigadores, desarrollar máquinas que posean al menos algunas de las propiedades de los organismos biológicos, tales como la adaptabilidad, robustez, versatilidad y agilidad. En este sentido los sistemas naturales son una gran fuente de inspiración.

Grandes aportes han sido dados a la humanidad en base a la imitación de los sistemas biológicos. Uno de ellos ha sido la creación de las aeronaves, hecho inspirado en el vuelo de los pájaros. Este es un ejemplo claro de lo que se puede avanzar en lo tecnológico al imitar a la naturaleza (en su aspecto más amplio). Hoy en día se ven aviones de toda índole, con capacidades de volar sorprendentes, incluso algunas de ellas no conocidas en especies animales. No hay duda que a través del estudio e imitación de la Biología se puede enriquecer a las ciencias tecnológicas, así como se lo está haciendo con la Robótica. Sin embargo, se podría decir que este enriquecimiento es mutuo o bidireccional, ya que a través de los modelos robóticos bioinspirados se pueden validar modelos biológicos, o incluso retroalimentarlos. No hay que perder de vista que la Biología al fin y al cabo es una ciencia, y como ciencia también se basa en modelos, los cuales en muchos casos no se ajustan satisfactoriamente a la realidad (el organismo animal en este caso). Por lo tanto el estudio e implementación de modelos computacionales bioinspirados tiene también la ventaja de nutrir a la Biología.

Lamentablemente, el hecho de copiar sistemas biológicos de manera completa en la mayoría de los casos no es factible (debido a la complejidad que implica sintetizar cada detalle), o de poco interés (ya que los animales deben cumplir con muchas restricciones que no se aplican a los robots, como cuidar el metabolismo, deshacerse de parásitos, etc), o la solución tecnológica es superior a la disponible en la naturaleza (por ejemplo, no existe aun un equivalente biológico para la rueda). De cualquier manera, resolver algunos de los principios de los sistemas biológicos que pueden ser de alguna forma utilizados como fuente de inspiración en el diseño de robots, puede ser de gran importancia. De hecho en los últimos años, soluciones biológicamente inspiradas han tenido mucho éxito en resolver problemas complejos. En este sentido es importante destacar, que los sistemas a los que no se les da inteligencia desde afuera, sino que por el contrario los mismos la adquieren a través del aprendizaje, son los que han obtenido mejores resultados[1].

El problema que pretende investigar este proyecto es el problema del aprendizaje espacial, es decir, cómo hacer que un robot aprenda de forma automática conductas de comportamiento espacial que sean útiles para desempeñar su tarea cualquiera sea ella. Para ello, en particular interesan los modelos bioinspirados, los cuales investigan cómo los animales aprenden

estas conductas, para luego hacer los modelos computacionales que intenten imitarlos.

1.2. Motivación SLAM

Si bien son muy comunes hoy en día los casos en que un individuo se encuentra con un dispositivo móvil con GPS, y puede ubicarse fácilmente a la vez de conocer el mapa del entorno, también existen muchas situaciones en las que un individuo o un robot autónomo se encuentran en un lugar desconocido, sin tecnología que le resuelva ni la ubicación en dicho entorno, ni un mapa del mismo. El siguiente ejemplo nos puede servir para entender el proceso implicado en SLAM. Considérese un simple robot: un conjunto de ruedas, conectadas con un motor y con una cámara. Además también un actuador, esto es, un dispositivo físico que nos permita controlar la velocidad y la dirección de dicho robot. Ahora imaginemos que el robot es colocado en un lugar inaccesible con fin de construir un mapa de dicho lugar, y que aquél es manejado remotamente por un operador. En este escenario el operador le permitiría al robot explorar el territorio, y a través de la información obtenida por la cámara, el operador entendería dónde se encuentran los objetos circundantes y cómo el robot está orientado con referencia a ellos. En este ejemplo lo que el operador ha hecho es tal y cual SLAM (Simultaneous Localization and Mapping). Determinar la ubicación de los objetos dentro del ambiente circundante es una instancia "Mapping", y establecer la posición del robot con respecto a dichos objetos es precisamente "Localization".

Tal como se observa en el ejemplo, SLAM busca paralelamente resolver la ubicación y construir un mapa del entorno sin información previa.

Por otra parte, este proyecto de Aprendizaje Bioinspirado, el cuál se basó fuertemente desde un principio en la aplicación de algoritmos de Aprendizaje por Refuerzo, necesitaba para cualquier testeo práctico respetable, de un algoritmo SLAM que resolviera el problema de la ubicación y el armado del mapa del entorno (SLAM).

1.3. Objetivos y alcance del proyecto

Dentro de los objetivos del proyecto se encuentran:

- Relevar el estado del arte en aprendizaje bio-inspirado centrándose en algoritmos de aprendizaje por refuerzo.
- Diseño e implementación de un algoritmo de aprendizaje sobre alguna de las plataformas robóticas disponibles.
- Investigar y/o desarrollar métricas de evaluación que, en lo posible, permitan medir la eficiencia de la solución del sistema implementado.

El alcance final del proyecto abarca los siguientes puntos:

- Relevamiento del estado del arte en aprendizaje bio-inspirado, y SLAM.
- Diseño e implementación de un algoritmo de SLAM métrico.
- Diseño e implementación de un algoritmo de particionamiento del espacio, para ser utilizado en un futuro algoritmo de aprendizaje.

1.4. Organización del trabajo

El presente documento se ha dividido en tres partes con el fin de ordenar y clarificar el trabajo realizado, de forma que resulte más fácil y comprensible su lectura. Notar que el capítulo 1 es una introducción a todas las partes.

Parte I - Marco Teórico

El objetivo de esta parte del documento es mostrar un resumen de la revisión del estado del arte relevada y analizada durante la etapa de investigación.

Capítulo 2 - Describe el problema original de estudio del proyecto, que es Spatial Learning, concepto que refiere al aprendizaje de conductas espaciales por parte de un agente las cuales le permiten al mismo realizar una tarea en un entorno dado.

Capítulo 3 - Se presenta el problema de SLAM resuelto. En una primer instancia se explica los motivos por los cuales no fue implementado el algoritmo de SL y las motivaciones detrás del algoritmo SLAM realizado.

Parte II - Implementación

Capítulo 4 - En este capítulo primero se intenta dar una descripción de cómo se descompone el algoritmo SLAM diseñado, comentando algunas de las hipótesis que fueron utilizadas y los motivos detrás de ellas. Luego, cada sección subsiguiente detalla una de las componentes del algoritmo.

Capítulo 5 - En este capítulo se describen los algoritmos pTAM y pWG, algoritmos de división de mapas en regiones convexas. Se ha de tener en cuenta que estos módulos son simplemente una prueba de concepto para mostrar cómo se podría usar el mapa desarrollado en SLAM.

Capítulo 6 - En este capítulo se describe los módulos físicos implementados (robot y laberinto), así como su uso y funcionamiento.

Parte III - Evaluación

Capítulo 7 - Describe la estrategia de experimentación realizada, así como las pruebas y los resultados obtenidos en la evaluación de las soluciones propuestas.

Capítulo 8 - Es el capítulo final de este informe. Presenta las conclusiones más importantes que se desprenden de este proyecto. También se trazan líneas a seguir como posibles trabajos a futuro.

Parte I

Marco Teórico

Capítulo 2

Problema original: Spatial Learning (SL)

Spatial Learning hace referencia al aprendizaje de conductas espaciales de un agente, las cuales le permiten al mismo realizar una tarea en un entorno dado. Este problema carece de definición formal, pero dicho de otra forma en palabras sencillas, equivale a aprender conductas de navegación, que le permitan al agente moverse de forma eficaz en su entorno para cumplir sus objetivos.

En las etapas iniciales de este proyecto, el objetivo era resolver este problema. Sin embargo, debido a la complejidad que presenta, se terminó por resolver únicamente un módulo del mismo. Si bien al final no se resolvió el problema entero, se encuentra la necesidad de presentarlo aquí para entender la motivación detrás de este proyecto, así como las decisiones tomadas.

2.1. Perspectiva Biológica

El aprendizaje bioinspirado intenta recrear los mecanismos por los cuales los animales aprenden. Es por eso que en esta sección se comienza presentando algunas de las regiones y células más importantes del cerebro en la teoría de cognición espacial. Luego se procede a comentar los experimentos que se realizan en el área.

2.1.1. El cerebro y el procesamiento de la información

La cantidad de estructuras cerebrales que participan en el proceso de cognición espacial son muchas y no son completamente comprendidas. Por ello, en esta sección no se pretende realizar un informe completo de estas estructuras sino dar una visión general de ellas.

Antes de comenzar, se mencionarán algunas cuestiones que aparecen al considerar que un animal utilice un mapa interno como guía para el proceso de cognición espacial. Dichas cuestiones, no se plantean con el fin de contestarlas, sino con el de intentar agrupar funcionalmente a las estructuras cerebrales, así como de facilitar su explicación e introducción.

Si los animales utilizan un "mapa", algunas de las cuestiones que sería pertinente preguntar sería: ¿Cómo lo construyen?, ¿Qué información utilizan?, ¿Cómo lo almacenan?, ¿Cómo se ubican en el mapa?, ¿Cómo lo utilizan para que les sea de utilidad ?, etc.

Para ir contestando un poco a estas preguntas, en [2] se habla de múltiples niveles de organización espacial. Es decir se habla de que se utilizan múltiples mapas en conjunto para la navegación en el espacio. El modelo que presentan en su artículo (TAM-WG que se documenta en detalle en la sección adecuada), a grandes rasgos puede ser dividido en 4 módulos. Uno de ubicación espacial (SLAM), uno de navegación a nivel local (modelo TAM), uno de navegación a nivel global (modelo WG) y uno de control (sistemas motivacionales en los modelos TAM y WG).

En el módulo de control se habla de que los drives (motivaciones) guían a los movimientos y al aprendizaje. Las motivaciones incluyen aspectos como el miedo, el hambre, la sed, etc. Estas le sirven al animal para definir un estado interno que, junto a la memoria de los lugares donde estas necesidades fueron satisfechas, pueden ser utilizadas para definir la toma de decisiones. Los módulos relacionados del cerebro incluyen al sistema sensorial (que modifica a las motivaciones - por ejemplo, el ver comida hace que aumente el hambre), al hipotálamo y al estriado ventral.

El módulo de ubicación espacial, le permite al animal precisamente lo que dice su nombre, ubicarse en el espacio. Se encarga de crear un mapa y utilizarlo para representar la posición actual del animal. Las estructuras asociadas incluyen al hipocampo, las células de dirección de la cabeza y a las células grilla.

El módulo de navegación global codifica lugares de interés y le permite al animal realizar "planificación" a grandes rasgos, es decir, se realiza la planificación sin tener en cuenta los detalles. Por ejemplo, al utilizar una vía de trenes la navegación global planificaría una secuencia de estaciones a las que se debe ir para llegar al destino sin considerar detalles de cómo cambiar de trenes en una misma estación. Este módulo está asociado a las células de lugar, a la corteza prefrontal y al estriado ventral.

Finalmente, el módulo de navegación local permite moverse con información local es decir, sin usar información proveniente de la ubicación espacial. A modo de ejemplo, al moverse entre estaciones de tren se utiliza la navegación global, pero al caminar por la plataforma del tren se utiliza la navegación local. Los módulos relacionados incluyen a la corteza parietal posterior.

Head Direction Cells

Las neuronas de dirección de la cabeza [3] son neuronas cuya activación depende de la relación de la dirección de la cabeza del animal con respecto al ambiente en un plano horizontal. Cada una de estas neuronas son similares a una brújula ya que tiene una dirección asociada para la cual se activan, y su activación no depende del lugar. A modo de ejemplo, una neurona de dirección de la cabeza con dirección asociada norte, tendrá máxima activación cuando la cabeza del animal apunte en la dirección norte (en el plano horizontal) y esto será independiente de donde se encuentre el animal. Por otro lado, estas neuronas difieren de una brújula, en que su activación no se ve producida por el campo magnético de la tierra, sino que se activa según puntos de referencia y señales de movimiento propias del animal como por ejemplo señales vestibulares y propioceptivas.

Se debe observar que la activación de las células de dirección de la cabeza depende únicamente de la cabeza del animal, no depende de la orientación del cuerpo del animal ni de la orientación relativa entre la cabeza y el cuerpo. Además, estas neuronas se activan sin importar si el animal está quieto o en movimiento y son independientes del comportamiento que esté teniendo el animal.

Dentro de un mismo ambiente, las propiedades de activación de estas neuronas se mantienen estables durante muchos días o incluso meses.

Grid Cells

Las células grilla o grid cells[4] son neuronas cuyo patrón de activación, el cual está modulado según el lugar, define un arreglo triangular que cubre toda la superficie disponible de un espacio abierto bidimensional. Se piensa que las células grilla forman una parte esencial en el sistema de navegación métrico de coordenadas del cerebro. Además, han llamado mucho la atención gracias a que su patrón de activación, el cual forma una estructura similar a un cristal, no depende del mundo exterior sino que es completamente creado por el sistema nervioso.

La estructura rígida generada y su independencia con el mundo exterior sugiere que la posición, señalada por la activación de estas células, debe ser integrada a partir de las señales de velocidad y dirección sin referencia alguna al ambiente exterior. Esto es lo que se conoce

como integración de trayectorias (path integration - PI). Si bien la integración de trayectorias determina la estructura básica de la matriz del patrón de activación de estas células, la estabilidad de los vértices de la malla y de la orientación de la malla con respecto al ambiente implica que la grilla debe estar asociada con bordes geométricos y puntos de referencia. Bajo ciertas condiciones estas asociaciones pueden sobrescribir el proceso actual de integración de trayectorias. Cómo las señales de integración de trayectorias son integradas con señales sensoriales externas, no ha sido determinado todavía.

Place Cells

Las células de lugar o place cells[5] son células piramidales del hipocampo descubiertas por O'Keefe [6], que exhiben patrones de activación altamente confiables que codifican la posición espacial del animal en un espacio dado.

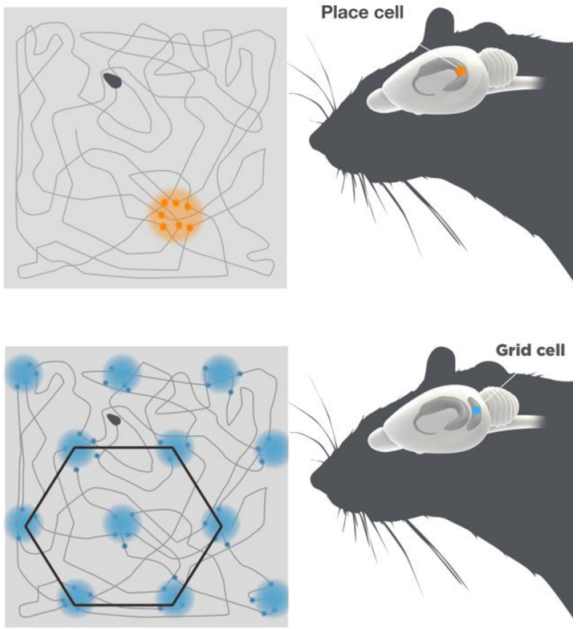

Figura 2.1.1 – Place Cells and Grid Cells
Las células de lugar, descubiertas por John O'Keefe, residen en el hipocampo del cerebro y se activan cuando una rata se encuentra en un determinado lugar. En la corteza entorrinal, las células grilla, descubiertas por May-Britt Moser y Edvard Moser, se disparan a intervalos regularmente espaciados cuando un animal se mueve a través del espacio, formando un patrón hexagonal.[7]

Hipocampo

El hipocampo[5] es parte del prosencéfalo ubicado en el lóbulo temporal medial. Es crítico en relación a la formación de memoria que puede ser utilizada conscientemente. El hipocampo funciona como el principal motor de búsqueda del cerebro. Permite buscar de manera rápida y eficiente los recuerdos depositados en la neocorteza. El hipocampo está altamente vinculado a corteza entorrinal y es fuertemente influenciado por neuromoduladores subcorticales.

La mayoría de los estudios sobre el hipocampo han sido realizados en roedores y han dado fruto a la teoría de navegación espacial.

Hipotálamo

El hipotálamo[8] es una pequeña área en la base del cerebro que, si bien solo pesa aproximadamente 4 de los 1400 gramos de un cerebro humano adulto, realiza una gran cantidad de funciones que son vitales para la supervivencia de un individuo. En general, el hipotálamo actúa como integrador y regulador de las funciones necesarias para la vida, y en particular contiene funcionalidad relacionada con la regulación del hambre, sed, sexo y temperatura [9].

En Swanson (2000, 2005)[10, 11] se postula que además influencia el comportamiento a través de sus conexiones con el ganglio basal y la corteza prefrontal. En Arbib & Bonaiuto 2012[2] se utiliza la salida del hipotálamo como un mecanismo para regular la velocidad de aprendizaje de un animal.

Corteza Prefrontal

La corteza prefrontal es la corteza cerebral que cubre la parte frontal del lóbulo frontal. Esta región está implicada en el planeamiento del comportamiento cognitivo complejo, de la expresión de la personalidad, de la toma de decisiones y de la moderación de las conductas sociales [12]. Se considera que la actividad básica de esta región del cerebro es orquestar los pensamientos y acciones de un individuo acorde a sus metas [13]. Esta región se relaciona con habilidades para diferenciar pensamientos contradictorios, discernir entre bien y mal, mejor y peor, igual y diferente y prever consecuencias futuras de actividades actuales. Debido a esto, en Arbib & Bonaiuto 2012[2] se compara esta región con el módulo WG, módulo de planeamiento de acciones a largo plazo. Daño en la corteza prefrontal puede significar en déficits de concentración, orientación, capacidad de abstracción, juicio, capacidades de resolución de problemas y comportamiento social inadecuado.

2.1.2. Experimentación en roedores[14]

En la literatura bioinspirada, el problema de SL se aborda estudiando cómo los animales aprenden los caminos por los que se mueven para lograr sobrevivir. Más en particular, se suele estudiar roedores y su comportamiento en laberintos.

En el estudio de estos animales, se observa que ellos deben utilizar un sistema de orientación mediante el cual desenvolverse sin problemas en ambientes complejos, y alcanzar así sus objetivos propuestos. Ellos deben guiarse por su entorno, e ir creándose un "mapa virtual interno" al que acceder para volver sobre sus pasos, o evitar lugares ya visitados que no aporten ningún beneficio. Las roedores suelen evitar lugares abiertos, y tienden a desplazarse por esquinas y recovecos. Además, ellos siempre establecen un lugar que sirve tanto de refugio como de "reseteo" para establecer el "punto cero" en las incursiones que realizan al medio. Este comportamiento de los roedores es altamente adaptativo en cuanto a depredación se refiere ya que, al evadir los lugares abiertos se quedan menos expuestos a posibles peligros. Por otro lado, su complejo sistema de navegación está sustentado por múltiples fuentes de información que se debe integrar y, en algunos casos, discernir cuáles son las más confiables. Los tipos de información de la que se valen los roedores se clasifican en dos tipos: información externa e interna. La información interna procede principalmente del sistema vestibular que les informa sobre la dirección que tiene su cabeza, o la velocidad a la que se mueven. La información externa es principalmente la que proviene del sistema visual (aunque también integra la que viene del sistema sensorial, olfativo y auditivo). Este último tipo de información les es muy útil a la hora de establecer puntos de referencia, y así facilitar la memorización de un determinado entorno. Para poder establecer el mapa virtual interno correctamente, y acceder a él y al contexto en el cual se encuentra dicho mapa, los roedores deben integrar ambos tipos de información, aunque a veces no disponen de la información externa, en cuyo caso se ven obligados a utilizar los mapas generados únicamente con la información vestibular. Este último sistema, si bien da pie a algunos errores, es bastante fiable.

Respecto a los experimentos que se realizan en los roedores, para investigar su comportamiento, lo que se suele hacer es depositarlos en laberintos con comida oculta, la cual deben

encontrar. Si bien los tipos de experimentos de este estilo y sus objetivos son diversos, en general lo que se observa es cómo va cambiando el comportamiento del roedor a medida que pasa el tiempo y/o las iteraciones del experimento en respuesta a alguna variable de control. Para ejemplificar un poco mejor las experimentaciones que se realizan sobre roedores, a continuación primero se presentan algunos de los laberintos más usuales y luego algunas de sus utilizaciones.

Laberinto tipo T

Este laberinto, como lo dice su nombre, tiene forma de T. En este el roedor es colocado en la base de la T. Al final del corredor, el roedor debe decidir si doblar a la izquierda o a la derecha. La comida es depositada al final de uno de estos brazos.

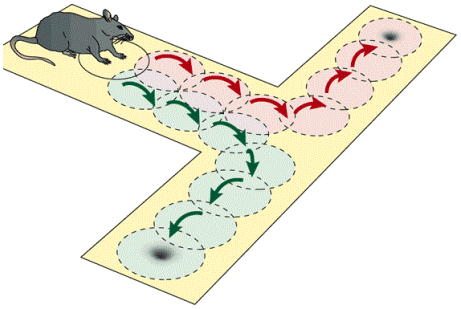

Figura 2.1.2 – Laberinto tipo T.
En la figura se observa la decisión que enfrenta el roedor en el laberinto tipo T.[15]

Laberinto de brazos radiales

Los laberintos de brazos radiales, son laberintos que disponen de N brazos conectados en forma radial como se observa en la siguiente figura. La comida puede disponerse en uno o varios de los brazos del mismo. A su vez el roedor se coloca inicialmente en uno de los brazos o en el centro del laberinto. La posición inicial de este puede variar.

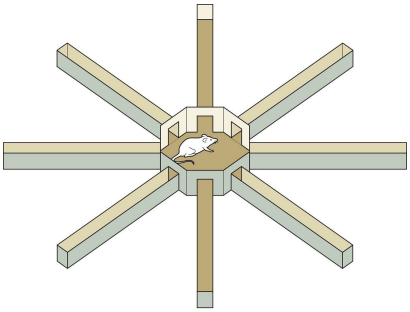

Figura 2.1.3 – Laberinto de 8 brazos radiales.[16]

Laberinto Acuático de Morris

El laberinto de agua de Morris[17] consiste en una piscina cilíndrica de agua la cual es lo suficientemente honda para que el roedor no de pie. En esta, se encuentra una plataforma escondida la cual el roedor debe encontrar. La posición inicial y la de la plataforma pueden variar.

Figura 2.1.4 – Laberinto acuático de morris.
En la figura, dentro del cuadrante superior izquierdo de la piscina se puede observar la plataforma escondida.[18]

Usos

El caso más sencillo consiste en poner al roedor en el laberinto y ver como mejora su desempeño con el transcurso del tiempo. Tomando como ejemplo el laberinto de agua de Morris, un roedor es colocado sucesivamente en una posición inicial fija del laberinto para que busque la plataforma escondida. Cada iteración del experimento termina cuando encuentra dicha plataforma. Para evaluar y analizar el desempeño del roedor existen varios mecanismos. Sin hacer una lista extensiva estos incluyen: medir el tiempo total del experimento, medir la distancia total recorrida, analizar el recorrido realizado, analizar el tiempo o porcentaje de tiempo que el roedor estuvo en cada zona, etc. Además, se suelen usar iteraciones de control que son iteraciones en las cual se ha quitado la recompenza del roedor (en el caso de Morris, se quita la plataforma) y se lo mantiene buscando durante una cierta cantidad de tiempo para evaluar el progreso/comportamiento del roedor.

A partir del caso más sencillo, se han ido desarrollando diversas variantes a los experimentos. A modo de ejemplo, la posición del roedor y/o ubicación de la comida se pueden variar de forma aleatoria o secuencial, se puede usar marcas para ayudar al roedor a orientarse, se puede comparar el desempeño de los roedores bajo el efecto de diversas condiciones físicas como por ejemplo ratones sanos vs. enfermos o drogados, etc.

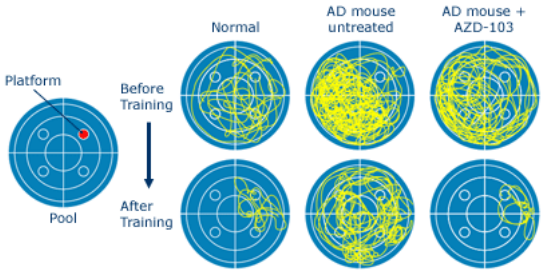

Figura 2.1.5 – Laberinto de Morris y alzheimer.
En la figura se muestra el recorrido que realizan tres tipos de roedores en el laberinto acuático de Morris. La imagen a la izquierda muestra la ubicación de la plataforma en la piscina, la fila superior los recorridos antes del entrenamiento, y la inferior después. Las tres columnas de izquierda a derecha corresponden a un roedor sano, a uno con alzheimer y a uno con alzheimer bajo tratamiento.[19]

Algo a tener en cuenta es que cuando se quiere evaluar el comportamiento del roedor frente a ciertos estímulos, se ha de tener cierto cuidado con la creación, uso y mantenimiento de los laberintos. A modo de ejemplo, si lo que se evalúa es la creación de mapas a partir de la información visual, se deberá tener cuidado de que no haya otro tipo de información como pueda ser la olfativa. Siguiendo con el ejemplo, para esto se utiliza comida inodora y se limpia de forma adecuada el laberinto en cada iteración.

Tarea T-Maze reversal

Para ejemplificar mejor todo esto, se expone la tarea T-maze reversal, la cual es un experimento particular donde se compara el desempeño de roedores sanos vs. roedores con lesiones de fórnix. La tarea fue presentada en el articulo [20], y en el [2] se presenta un modelo computacional de SL (uno de los objetos centrales de estudio de este proyecto) que imita el comportamiento observado de los roedores en este experimento.

EL mismo consiste en entrenar a los roedores en el laberinto tipo T poniéndoles comida en el brazo derecho. Cuando los roedores alcanzan cierto desempeño la comida empieza a ser depositada en el extremo opuesto. Para evaluar el desempeño, lo que se hace es intercalar una iteración de control en el laberinto radial de 8 brazos cada cierta cantidad de iteraciones. La métrica utilizada es el porcentaje de tiempo que el roedor pasa en cada brazo durante las iteraciones de control. La idea detrás de este experimento es observar que tan rápido pueden los roedores re-aprender el camino a la comida. En la evaluación se utiliza el laberinto de 8 brazos ya que en el tipo T, debido a que es demasiado sencillo, no es posible apreciar como varía la preferencia en las acciones durante el re-aprendizaje. Si se usa el tipo T como evaluación, lo que se observa es que tanto el comportamiento de los roedores lastimados como el de los sanos cambia de forma relativamente abrupta, pasando a preferir el brazo izquierdo. Sin embargo, al evaluarlo en el de 8 brazos, lo que se observa es el mismo cambio de preferencia en los roedores sanos y un cambio gradual de preferencia desde el brazo derecho al brazo izquierdo en los roedores lastimados.

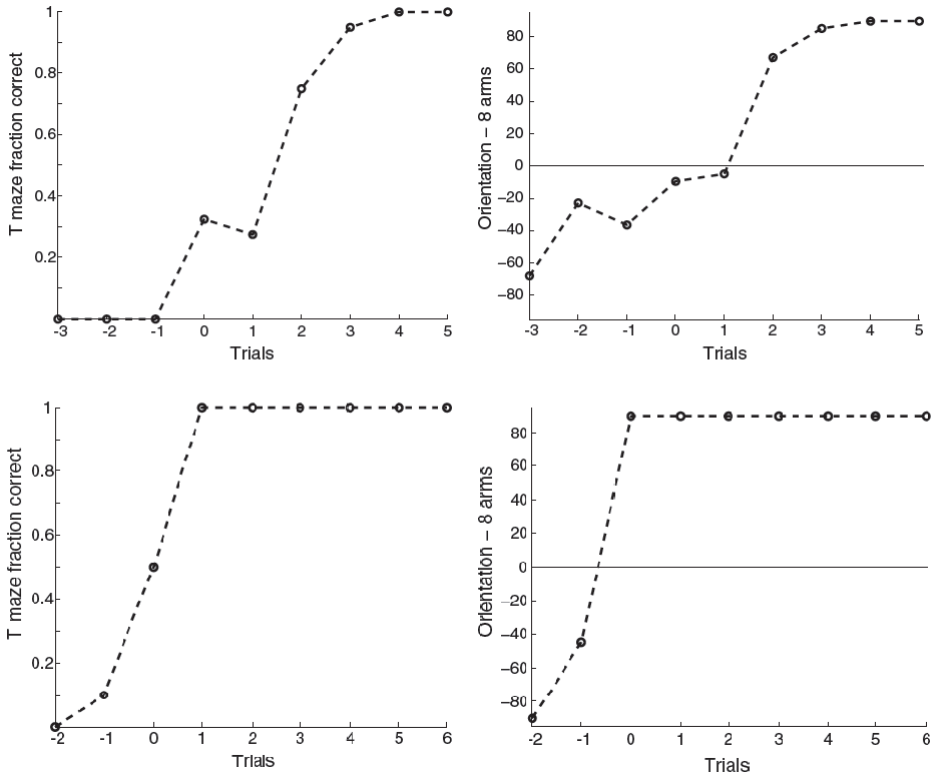

Figura 2.1.6 – Resultado de roedores en tarea T-maze Reversal.
En la figura se muestra gráficas que evalúan el desempeño de roedores simulados tanto sanos como lesionados en la tarea T-maze reversal. Las gráficas para el desempeño en roedores reales es similar a esta. La primera fila corresponde a los roedores lesionados y la segunda a los sanos. La primer columna muestra como varía el desempeño en el tiempo y la segunda muestra como varía la dirección preferida en el tiempo en laberinto de 8 brazos. Las direcciones son representadas por un enumerado entre -80 y 80, y se corresponden con las direcciones $-\pi$ y π respectivamente.[2]

2.2. Perspectiva computacional

Los modelos computacionales de spatial learning tienen dos objetivos. En primer lugar buscan crear modelos que embeban en los robots las capacidades de navegación que tienen ciertos seres vivos. Si bien se está progresando en esta área, los robots aún no son capaces de navegar de igual forma que los animales. Dentro de los problemas que se encuentran se debe mejorar la efectividad, la eficiencia y la generalidad de los algoritmos existentes. Es de especial interés realizar algoritmos que rompan la hipótesis del mundo cerrado. Esta hipótesis asume que el modelo del mundo que tiene el robot contiene todo lo que este necesita saber, no pueden haber sorpresas. Si la hipótesis del mundo cerrado es violada no se garantiza su correcto funcionamiento. De esto resultan dos problemas. El primero es que es altamente fácil olvidarse de poner todos los detalles necesarios. El segundo es que usualmente ni siquiera es posible considerar todos los escenarios que se puedan presentar. De esta forma, programar un robot que funcione en una amplia variedad de escenarios es excesivamente difícil [21]. Por este motivo, observando que los animales son capaces de realizar estas tareas en un mundo abierto, es razonable querer entender y copiar cómo ellos las realizan.

En segundo lugar, las implementaciones bioinspiradas de SL ayudan a las fuentes de conocimiento bioinspiradas en diversas formas. Primeramente, estos algoritmos sirven para "validar" los modelos generados existentes (se utiliza comillas ya que un modelo nunca puede ser validado, solo invalidado), es decir para sugerir que las hipótesis tomadas pueden ser correctas. A su vez, además de "validar/invalidar" modelos, la experimentación con modelos

computacionales puede servir para indicar carencias sugiriendo preguntas y/o caminos de investigación a seguir.

Por lo general, todos los modelos computacionales de SL cuentan con ciertos mecanismos en común. Dos de los más importantes son la implementación de un sistema de motivaciones y el uso de algoritmos de Reinforcement Learning (RL - algoritmos de aprendizaje a base de prueba y error). El sistema de motivaciones tiene como objetivo simular el estado interno de un animal como puede ser el tener hambre y/o tener miedo. Estas motivaciones son las que guiarán el comportamiento del robot. Por ejemplo si el estado prevalecente en el robot es el hambre, el robot basará sus decisiones en base a la búsqueda de comida, mientras que si prevalece el miedo, la conducta prevalecente será la de escape. Por el otro lado, los algoritmos de RL proveen una forma natural de guiar el aprendizaje. Lo que se busca es reforzar las conductas que hayan tenido un efecto favorable en el estado del robot. A modo de ejemplo, si en un determinado momento el robot se encuentra en la posición X con hambre y al doblar derecha encuentra comida, se buscará reforzar la conducta "doblar derecha en X cuando esté con hambre" para que esta sea repetida en un futuro con la esperanza de volver a encontrar comida.

Dentro de los modelos existentes, en 1977 Arbib y Lieblich [22] crean el WG (World Graph), modelo en el cual un grafo representa las posiciones de interés para el robot dentro de un espacio. Para la toma de decisiones se desarrolla un sistema de motivaciones, aunque sin las herramientas adecuadas se sugiere dos formas en las que el sistema podría ser utilizado para guiar el movimiento. Posteriormente, con el advenimiento de los algoritmos de aprendizaje por refuerzo, se le agrega uno de estos algoritmos al modelo WG como mecanismo natural para resolver esta problemática. En 1988 Guazzelli, Corbacho, Bota y Arbib [23] implementan el modelo TAM (Taxon Affordance Model) y lo acoplan al WG para formar el TAM-WG. El TAM es otro modelo que utiliza a un sistema motivacional junto a un algoritmo de RL para guiar el movimiento de un agente. Sin embargo, a diferencia del WG, el objetivo de este modelo es la navegación local (a corto plazo) no global, utiliza información de carácter más bien métrico que topológico. En [2] Bonaiuto y Arbib realizan mejoras al modelo TAM-WG en el cual se le asocia al modelo ideas mejoradas de ejecutabilidad (capacidad de ejecutar una acción) y deseabilidad (deseo de realizar una acción) . En el viejo modelo TAM-WG, la deseabilidad y ejecutabilidad eran combinadas de forma aditiva mientras que en el nuevo modelo estas se combinan de forma multiplicativa. Para explicarlo de forma sencilla, la idea es que antes si una acción tenía probabilidad 0 de ser realizada pero una deseabilidad grande, esta acción podía ser elegida. Con el cambio introducido, al combinarse multiplicativamente, las acciones con baja ejecutabilidad tendrán poca o nula probabilidad de ser elegidas. Por ejemplo, si la ejecutabilidad es 0, al multiplicar por 0 esa acción será invalidada.

En este trabajo, antes de desviarse a SLAM se trabajó principalmente con el modelo TAM-WG mejorado por lo que principalmente se citan trabajos asociados, sin embargo, cabe destacar que estos no son los únicos trabajos en el área. A modo de ejemplo en el 2000 Arleo y Gerstner realizan un modelo que tiene por objetivo la simulación de células de lugar. En su trabajo, ellos utilizan un sistema motivacional similar al WG y usan RL como método de aprendizaje. Los algoritmos de RL utilizan un estado, mientras que en el WG los estados son los nodos del grafo, en este otro modelo los estados son representados por las células de lugar.

A continuación se presenta el modelo TAM-WG más a fondo y algunas de las herramientas utilizadas más importantes.

2.2.1. Reinforcement Learning[24]

Como se comentó en la sección anterior, un algoritmo de SL consiste en una implementación de un algoritmo de RL por lo cual, antes de presentar al modelo TAM-WG parece adecuado agregar una sección que introduzca estos algoritmos.

Reinforcement Learning es un área de machine learning inspirada por la psicología conductista. Dicha área estudia cómo un agente que pretende alcanzar una meta aprende en base a prueba y error interactuando con su ambiente. Es decir, estudia cómo un agente toma

una decisión en base a los resultados de sus experiencias pasadas. RL se distingue de otros áreas de machine learning ya que no se le dice al agente que acciones debe tomar sino que éste debe aprender cuáles son las mejores probando cada una de ellas.

A grandes rasgos, un algoritmo de RL lo que busca es encontrar la mejor política que maximice la recompensa que conlleva seguirla. Una política es básicamente una función que a cada situación que se pueda presentar le asigna una decisión a tomar. Para lograrlo, lo que se hace es esencialmente probar las diferentes decisiones de forma empírica y de ahí elegir las que hayan dado mejores recompensas en el pasado.

Para que un algoritmo de RL funcione adecuadamente, los principales problemas que se deben resolver son los siguientes:

1. **Trade-off entre explotación y exploración**.

   Debido a la peculiaridad de prueba y error de los algoritmos de RL, uno de los desafíos con los que se debe lidiar que no aparece en otros tipos de aprendizaje, es el trade-off entre explotación y exploración. Exploración se le llama a la acción de probar ya sea decisiones nuevas, o aquellas con las que se disponga de poca información, con el fin de evaluar sus resultados. Explotación se le llama a la acción de elegir las acciones tomadas para las cuales se maximicen las expectativas de recompenza según las experiencias pasadas. Si siempre se explora nunca se aprovecha el conocimiento generado. Si siempre se explota, no es posible hallar mejores soluciones. De ahí surge el problema del trade-off.

2. **Recompensas tardías.**

   Otro problema con el que deben lidiar los algoritmos de RL es el de la recompensa tardía y la necesidad de la generalización. El primer problema hace referencia al hecho de que muchas veces la recompensa que otorga una decisión no es obtenida hasta mucho después de haber sido realizada. A modo de ejemplo, un estudiante que decide entrar en la facultad no ve una recompensa económica hasta luego de haber terminado la carrera. El otro problema hace referencia al hecho de que un agente sólo puede visitar una porción finita del conjunto de pares estado-acción. Debido a que el conjunto completo no tiene por qué ser finito, surge la necesidad de tener un mecanismo de generalización que permita inferir la recompensa de los pares no visitados.

Ejemplo: Un robot móvil debe recolectar basura en diferentes habitaciones y luego regresar para dejar la basura y recargarse. Para ello, cada vez que el robot llegue a la estación a recargarse se podría dar como recompensa positiva la cantidad de basura que este haya traído. De esta forma, en algún momento en que el robot esté funcionando, este podría encontrarse con la decisión de si entrar a una nueva habitación en busca de más basura para recolectar o si empezar a buscar su camino de regreso a la estación de recarga, cosa que debería decidir basándose en que recompensas recibió (es decir si pudo llegar a la base o no) en situaciones similares en el pasado tanto habiendo decidido entrar como no.

Formulación del problema

El problema de RL pretende ser una abstracción directa del problema de aprender interactuando con el ambiente para alcanzar una meta. El objeto que toma las decisiones es denominado el agente. Todo aquello que no sea el agente es denominado el ambiente. Ambos interactúan continuamente en tiempos discretos. El agente tomando decisiones y el ambiente respondiendo a esas acciones y presentando nuevas situaciones. De esta interacción surgen además las recompensas numéricas obtenidas por el agente que se tratan de maximizar a lo largo del tiempo.

A un mapa que a cada estado le asigne una distribución de probabilidades a sus acciones se le denomina una política. El objetivo de un agente es hallar la política que maximice una medida de la recompensa esperada. Esta medida es conocida como el retorno descontado.

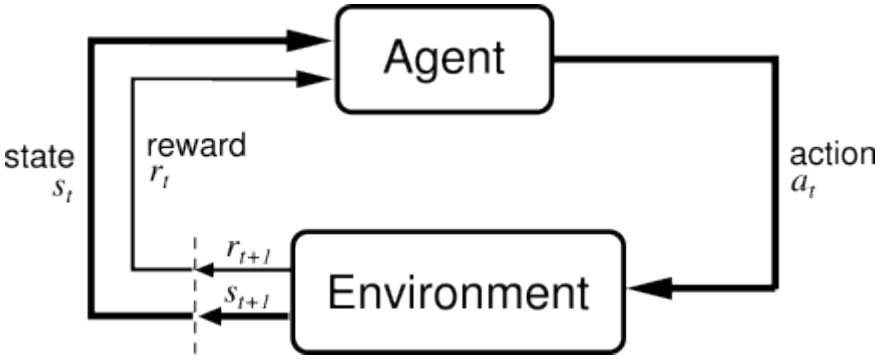

Figura 2.2.1 – Esquema básico de RL.
La figura muestra la interacción entre el agente y el ambiente en un algoritmo de RL. El agente realiza una acción la cual modifica el estado generando tanto uno nuevo como una recompenza.[24]

Una formulación más precisa del problema puede encontrarse en [24].

2.2.2. Un modelo de SL: TAM-WG[2, 23]

Como se mencionó anteriormente el modelo TAM-WG es la integración de los modelos TAM y WG. Ambos son similares en que utilizan un sistema motivacional junto a RL para proveer un algoritmo de aprendizaje espacial, sin embargo, ambos difieren en su finalidad.

En este trabajo se presenta los conceptos generales del modelo y se explica superficialmente como funciona el algoritmo. Por más detalles ver las referencias asociadas.

World Graph (WG)

El objetivo del modelo WG es proveer un algoritmo que permita planificar la navegación con un nivel de abstracción relativamente alto. Para ello, el modelo realiza una especie de mapa topológico del mundo representado por un grafo similar al mapa de una red de subterráneos. Así como en el grafo del subterráneo los nodos representan estaciones y las aristas caminos entre ellas, en el WG los nodos representan lugares de interés y las aristas caminos entre ellos. De esta forma, como en el ejemplo del subterráneo en el cual el mapa permite planificar la ruta entre estaciones, el WG permite planificar la navegación a gran escala de la siguiente manera. Por ejemplo, el mapa podría prever que en la estación X se haga un cambio de linea de la A a la B, pero no indicaría como moverse en la estación para realizar dicho cambio.

En este modelo, los nodos del grafo son utilizados como estados para el algoritmo de RL. Debido a que los estados codifican lugares, los nodos simulan parcialmente a las neuronas de lugar. La diferencia entre ambos es que las neuronas de lugar cubren toda el área a representar de un determinado lugar mientras que los nodos en el grafo representan posiciones particulares. Por ejemplo, mientras que las neuronas de lugar codificarían toda el área dentro de una jaula para roedores, el WG quizás podría tener nodos únicamente para el bebedero y para la rueda.

Según la descripción que se da en Arbib & Bonaiuto 2012[2], al iniciar el algoritmo, el grafo se inicializa con un único nodo representando la posición actual. Luego, dada una posición cualquiera, esta puede ser reconocida como un nodo en el grafo, como un lugar sin interés, o como un lugar de interés que aún no es parte del grafo. En el último caso se agrega un nuevo nodo para representar la nueva posición y se crea una arista entre él y el nodo anterior. En particular, en la implementación observada, los lugares de interés son los lugares donde

cambian los posibilidades de movimientos como por ejemplo el fin de un brazo, o la unión de varios. Visto de otra forma, estos son los lugares donde el roedor se ve obligado a tomar una nueva decisión de a donde ir.

Una característica importante a tener en cuenta que difiere del TAM es que este modelo le asigna una posición a cada nodo o, equivalentemente, a cada estado (ya que los estados representan situaciones de interés en el espacio, e interesa saber dónde surgen las mismas). Además, como los estados son posiciones, estos son independientes de la dirección de la cabeza, cosa que no pasa en el TAM como se verá en breve.

Una observación importante a tener en cuenta es que la asignación de una posición a los estados hace necesario que se cuente con un módulo que permita obtener la posición en el algoritmo. En el artículo de referencia esto se hace mediante SLAM, o más específicamente simulan un módulo de SLAM otorgándole al algoritmo la posición global con ruido y señalando la existencia de RATSLAM, un algoritmo de SLAM bioinspirado. Algo a tener en cuenta es que tanto SLAM como WG generan mapas y sirven para localizarse por medio de la ubicación de puntos clave, sin embargo, el objetivo del WG no es localizarse sino aprender a navegar en el espacio.

La necesidad de contar con un módulo de localización, así como las similitudes encontradas entre SLAM y el WG son en parte la causa de que el foco de este proyecto haya pasado de SL a SLAM. Más información respecto a este tema se verá en el siguiente capitulo.

TAM

El modelo TAM (Taxon Affordance Model), hace uso de los affordances. Un affordance es básicamente un potencial perceptible del entorno para una acción. Por ejemplo, una pared representa una imposibilidad de continuar, mientras que una puerta abierta representa un potencial para atravesar la pared (por el hueco de la puerta). En el caso del TAM, los affordances de interés son los de movimiento, es decir lo que interesa es el potencial de movimiento en cada dirección. El objetivo de este modelo es usar los affordances tanto para reconocer situaciones similares como para limitar los movimientos posibles en cada situación. Con los affordances, esto último es posible de forma natural. Por ejemplo si hay una pared en frente del agente, los affordances indicarían no es posible continuar con lo cual no tendría sentido considerar esa dirección como una dirección posible de movimiento.

En el TAM, las situaciones reconocibles representan los estados para el algoritmo de RL. Dos situaciones son consideradas como la misma únicamente si presentan los mismos affordances. En particular, esto significa que son independientes de la posición y dependientes de la dirección de la cabeza. Por ejemplo en el laberinto de 8 brazos radiales, cuando el actor se posiciona en el centro de un brazo mirando hacia el centro del laberinto los affordances que se obtienen son los mismos sin importar el brazo. Por este motivo el estado será el mismo para cualquiera de los ocho lugares. De igual forma, considerese ahora al robot en el centro de un pasillo pero mirando en dos direcciones diferentes. En la primera mirando hacia la pared y en la segunda mirando en la dirección del pasillo. En una de las dos situaciones el agente tiene affordances para adelante y atrás pero no para los costados. En la otro, se tienen affordances para los costados pero no para adelante y atrás. Por este motivo cada situación será modelada en un estado diferente. En comparación con el WG, en el primer ejemplo cada brazo generaría un estado diferente ya que la posición también lo es, y en el segundo el estado sería el mismo.

Sistema Motivacional

El sistema motivacional utilizado en el TAM-WG considera dos tipos de motivaciones: las aversivas y las apetitivas. Las apetitivas son aquellas que tienden a aumentar con el pasaje del tiempo mientras que las aversivas son las que tienden a disminuir. Por ejemplo, el hambre es una motivación apetitiva ya que cuanto más tiempo se pase sin comer más se incrementará el hambre. Por el contrario, el miedo es una motivación aversiva ya que a medida que pasa el tiempo desde una situación que haya generado este sentimiento, el mismo tenderá a bajar.

En el TAM-WG, las motivaciones son modeladas como un real acotado inferior y superiormente. En cada iteración cada una es actualizada acorde a 3 factores: el pasaje del tiempo, estímulos y condiciones especiales. El tiempo incrementa o decrementa gradualmente a cada motivación acorde a si son aversivas o apetitivas. Las condiciones y estímulos son eventos que modifican la motivación de forma directa e indirecta respectivamente. En el caso del hambre la distinción es relativamente sencilla. El evento "comer" es una condición ya que satisface al hambre de forma directa. Paralelamente, sentir olor a comida es un estímulo ya que aumenta el hambre de forma indirecta. En el modelo, una condición varía a una motivación de forma directamente proporcional a su "tamaño", por ejemplo cuando más comida se coma más disminuirá el hambre. Por el otro lado, un estímulo hace variar a una motivación de forma proporcional a la variación del estímulo. Por ejemplo si de repente se siente olor a comida, el hambre aumenta, pero por más que se siga sintiendo el olor, el hambre no seguirá aumentando por ese motivo salvo el olor se intensifique.

Algoritmos de RL

El modelo TAM-WG más que ser un único algoritmo de RL, es la integración de varios. En particular se utiliza un algoritmo de RL para TAM y otro por cada drive en el WG. Estos algoritmos son todos integrados en uno combinando las políticas generadas en una sola para realizar el proceso de selección de acciones. Esto se puede realizar ya que las acciones a tomar son las mismas para todos los estados, estas son avanzar en una de N direcciones equiespaciadas predefinidas. Realmente en el WG la acción sería ir hacia otro estado, sin embargo esto se transforma en moverse en una dirección.

Los algoritmo de RL utilizados en el TAM-WG siguen una arquitectura de estilo actor-critic. En esta arquitectura se considera que hay un crítico que evalúa las acción realizadas por el actor. En base a las criticas (rewards) obtenidas, el actor va llevando una función de preferencias para cada par estado/acción. Estas preferencias son utilizadas junto con una política para asignar una distribución de probabilidad a las acciones de cada estado y así, usando esta distribución, poder elegir la siguiente acción a tomar.

Para cada algoritmo de RL en el WG, lo que se utiliza como reward es el valor de las condiciones del drive correspondiente multiplicados por el drive normalizados entre 0 y 1. Por ejemplo para el algoritmo de RL asociado al hambre, el reward vendría dado por el valor de la condición "comer" multiplicado por el valor del drive normalizado "hambre". De esta forma, si se come con mucha hambre el reward será mayor que si se come cuando no se tiene. El resultado de esto es que la velocidad de aprendizaje queda regulada por el nivel de los drives. En el caso del TAM, que hay un único algoritmo de RL para todos los drives, el reward utilizado es la suma de todos los rewards individuales para cada drive.

Resultados de TAM-WG

El principal resultado del modelo TAM-WG es poder replicar la conducta de los roedores sanos y lesionados en la tarea T-maze reversal. Los roedores sanos fueron simulados mediante el modelo TAM-WG completo, y los roedores lesionados fueron simulados eliminando la componente WG del algoritmo. Corriendo el algoritmo en un simulador geométrico, el resultado fue la reproducción del comportamiento observado en los roedores reales.

Además de lo anterior, otros resultados no tan importantes son los siguientes. En primer lugar se observa cómo funciona el algoritmo al utilizar más de un drive. Para ello se probo poner al roedor simulado en un laberinto lineal, donde en un determinado sector se le proporciona una descarga y un poco más adelante se coloca la comida. El resultado al colocar al roedor virtual en el extremo opuesto de la comida es una oscilación. Al acercarse a la zona de la descarga la componente del miedo persevera sobre la componente del hambre y se aleja, pero al alejarse el miedo disminuye hasta que nuevamente el drive del hambre toma control y regresa. En segundo lugar, se observó que pasa al entrenar al roedor simulado en un laberinto, y luego de que aprenda el recorrido a la comida, bloquearle un sector de ese camino. El

resultado fue que se encontró un nuevo camino a la comida, pero el reaprendizaje era muy lento.

Capítulo 3

Problema Resuelto - SLAM para la división del espacio

En esta sección, en una primer instancia se explican los motivos por los cuales no fue implementado el algoritmo de SL y las motivaciones detrás del algoritmo SLAM realizado. Además, se plantea el marco teórico para SLAM.

3.1. Pasando de SL a SLAM

En las etapas iniciales del proyecto se pretendía desarrollar el algoritmo TAM-WG siguiendo como referencia el artículo "Múltiple levels of spatial organization"[2]. Sin embargo, se encontraron los siguientes problemas:

- En primer lugar, fue necesario desarrollar un algoritmo de RL. El mismo pretendía imitar los mecanismos expuestos en la referencia, sin embargo, como no se contaba con suficiente información, esto no fue posible. A modo de ejemplo, el artículo menciona que para el proceso de toma de decisiones se utiliza una red neuronal DNF ("Dynamic neural field"). Sin embargo, no se pudo obtener especificaciones de la misma, ni en el artículo, ni en sus referencias, ni contactando a los autores ya que no se recibió respuesta por parte de ellos. Por este motivo, intentando seguir la referencia, se desarrollo un algoritmo de RL remplazando los módulos con los que no se contara con suficiente información. En particular el algoritmo implementado utilizaba (siguiendo la referencia) 100 acciones por estado correspondientes a 100 direcciones de movimientos posibles. Esto hacía que el algoritmo desarrollado aprendiera demasiado lento. De esta forma, se decidió que para que el algoritmo funcionara, se debería poder discretizar las acciones (movimientos) posibles, lo que permitiría incrementar la velocidad de aprendizaje.
- En el modelo TAM-WG se necesita un módulo de localización. En el artículo se menciona que el algoritmo toma como entrada la salida de uno de SLAM.
- En el modelo WG, es necesario contar con un módulo encargado de generar y mantener el grafo del mundo. Para crear dicho módulo, es necesario definir lo que representa un nodo. En la referencia esto lo definen como una posición o situación de interés, y dicen que, en el caso de los laberintos, estas posiciones están dadas por los lugares en donde cambian las posibilidades de movimiento. A modo de ejemplo, en el laberinto de 8 brazos podría haber un nodo para el centro y uno para el extremo de cada brazo. El problema con esta definición es que es demasiado vaga ya que, realmente, las posibilidades de movimiento varían de forma continua. Esto puede observarse mejor en la figura 3.1.1, en la que se observa cómo estas varían a medida que un robot se mueve por un pasillo. Intentando formalizar estas ideas, se pensó que los conjuntos convexos capturan en alguna medida el concepto de "tener las mismas posibilidades de movimiento". Esto se

debe a que es posible moverse entre todo par de puntos del polígono en linea recta sin abandonar la región. De esta forma se pensó que si se contara con un mapa métrico del espacio, el algoritmo de creación de estados podría verse reducido a reconocer estas áreas.

Analizando todo esto, la idea a la que se llegó es que desarrollando un algoritmo de SLAM métrico en conjunto con otro de obtención de polígonos convexos, todos estos problemas podrían ser solucionados. Las posibilidades de movimiento en el modelo TAM podrían ser discretizadas hallando el polígono convexo que represente la región del espacio en la que se encuentra el robot. Las acciones estarían dadas por las aristas del polígono que no representen paredes. Por otra parte, la utilización de estos convexos junto a la información en el mapa podría usarse para definir donde cambian las posibilidades de movimiento (es decir los estados del modelo WG).

De esta forma, observando las ventajas que podría tener, se comenzó a incursionar en SLAM. La meta original era luego continuar con el modelo TAM-WG, pero por las dificultades presentadas para resolver el problema de SLAM, se terminó acotando el alcance del proyecto.

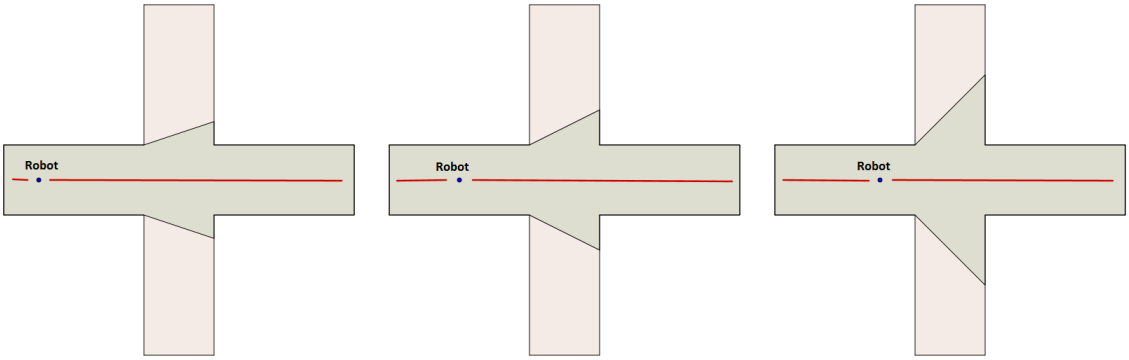

Figura 3.1.1 – Variación de posibilidades de movimiento.
En la figura se observa como van cambiando las posibilidades de movimiento a medida que avanza el robot por el laberinto. La linea roja se agrega para destacar como varía la distancia que se puede mover el robot hacia atrás y hacia adelante. A su vez, se observa que a medida que el robot se acerca al centro, este va pudiendo realizar giros más y más pronunciados hacia los costados.

3.2. Especificación del problema[21, 25, 26]

SLAM es una sigla que representa “Localización y Mapeo Simultáneo”. Cuando se habla de "mapeo", se hace referencia al problema de integrar cierta información que es recogida por los sensores, y que debe ser representada de una forma dada. Es una representación subjetiva de como el robot ve el mundo. En este sentido, toman gran relevancia tanto la representación del entorno como la interpretación de lo obtenido por los sensores. Por otra parte, "localización", trata de estimar la posición de un robot relativa a un mapa. En este sentido se puede hablar de “Pose tracking” cuando se conoce la posición inicial del robot, y se debe estimar la posición en el tiempo, y se habla de “Global localization” cuando no se tiene a priori ningún conocimiento de la posición inicial del robot.

En el correr de los años, los investigadores intentaron resolver el problema de la localización de diversas maneras. Las primeras soluciones simplemente ignoraban los errores de localización, lo que si bien presentaba ventajas de simplicidad, eliminaba completamente la posibilidad de planificación de caminos. A partir de esto también surgieron los sistemas reactivos de navegación, los cuáles se mencionarán más adelante, con la filosofía de “avanzar hasta llegar a determinado lugar”. Otro enfoque es el de los mapas topológicos, también mencionados más adelante, que poseen información simbólica para la localización en algunos

puntos como puertas de entrada, pero que no necesitan de localización continua. Lamentablemente, en situaciones reales es muy difícil disponer de puntos de referencia distinguibles, lo cual es un problema para este tipo de enfoques que puede ser resuelto agregando puntos de referencia artificiales, o encontrándolos en el entorno a través de la detección de características geométricas muy notorias. Otras técnicas buscan juntar los datos recibidos por los sensores con un mapa a priori, usando Arboles de Interpretación[25] o estructuras similares, con la dificultad latente de que es muy raro que la información que brindan los sensores se encuentre de forma amena para integrar en el mapa. Para esto la mayoría de los sistemas se rigen por la premisa de moverse cortas distancias, construir un mapa local e ir integrándolo paso a paso con el mapa global que se llevaba hasta entonces, producto de las integraciones anteriores[21]. El problema de SLAM trata de construir un mapa, y simultáneamente ubicar al robot en y con ese mapa. En muchos sentidos se parece al "Problema del huevo y la gallina", ya que para ubicarse se necesita un buen mapa, y para construir un buen mapa se necesita saber precisamente dónde se está ubicado[26].

3.3. Tipos de SLAM [27, 28]

3.3.1. Online vs Offline

Un algoritmo de SLAM se considera Offline cuando el robot en una primera instancia navega sobre el entorno para recolectar datos, y en una segunda instancia, se realiza el armado del mapa con los datos recopilados. Por otra parte, se considera SLAM Online cuando el procesamiento de navegación del robot se realiza simultáneamente con el de localización y creación del mapa.

3.3.2. Mapa Topológico vs Mapa Métrico

Desde que la localización es un problema fundamental en la robótica, se han desarrollado y discutido muchos enfoques, que pueden ser clasificados en tres tipos: Métricos, Topológicos e Híbridos. Los enfoques métricos son útiles cuando se necesita conocer con precisión la ubicación exacta del robot en términos de coordenadas métricas (ej. en coordenadas cartesianas). La mayoría de los enfoques utilizan mapas métricos. Los enfoques topológicos por otra parte representan el estado del robot de una forma más cualitativa. El problema de localizar un robot usando un mapa topológico es equivalente a determinar el nodo en el grafo que se corresponde con la posición actual del robot. En los últimos años algunos investigadores han integrado el paradigma métrico y el paradigma topológico obteniendo una representación híbrida [29].

3.3.3. Estáticos vs. Dinámicos

Se habla de SLAM estáticos cuando se puede asumir que no habrán cambios en el entorno con el correr del tiempo. La mayoría de las investigaciones sobre SLAM asumen entornos estáticos. Por otra parte, cuando se habla de SLAM dinámicos cuando se deben contemplar cambios en el entorno, como personas o lugares que se mueven (ej. Google Driverless Car).

3.3.4. Monorobot vs. Multirobot

Como el nombre lo dice, se entiende por SLAM monorobot, cuando se utiliza un solo robot, y por SLAM multirobot cuando se involucra más de uno comunicándose entre sí. En este último caso, la complejidad de los algoritmos aumenta (un ejemplo de experimento Multirobot utilizando un algoritmo descentralizado se puede encontrar en [30]).

3.4. Proceso SLAM[31]

En el proceso SLAM se pueden identificar múltiples partes:

1. Extracción de landmarks (o características)
2. Asociación de datos
3. Estimación del estado
4. Actualización de estado y características

Cada una de estas partes se puede resolver por diferentes caminos.

Las landmarks son características que pueden ser fácilmente observables y distinguibles en el ambiente. Estas son utilizadas por el robot para obtener su ubicación. A continuación se detallan algunas propiedades deseables que deberían cumplirse en lo que se refiere a los landmark:

- Un landmark debería ser visto desde muchas posiciones y ángulos dentro del ambiente.
- Un landmark debería ser único, de manera de poder identificarlo fácilmente.
- Debe haber una cantidad considerable de landmarks dentro del ambiente circundante, ya que de lo contrario el robot podría pasar mucho tiempo sin reconocer alguno, y por consecuencia, correr el riesgo de perderse.
- Un landmark debería ser estacionario, de lo contrario no tendría sentido registrarlo para luego intentar ubicarse en referencia a él.

Un ejemplo claro de esto podría ser la Estatua de la Libertad. Por tener la particularidad de ser vista desde muchos lugares (tierra, mar y aire), y como además es única, la misma podría utilizarse como un landmark.

3.4.1. Ejemplo de extracción de características mediante sensor láser.

La extracción de características es un aspecto crucial en SLAM, ya que se confía en el uso de puntos de referencia para determinar la posición del agente. La decisión de cuál algoritmo utilizar depende de varios aspectos, como por ejemplo el tipo de sensores que se utilicen, o las particularidades de las características a ser reconocidas. A continuación, se presenta un ejemplo utilizando un sensor láser.

Para ver esto, observese la figura 3.4.1. En la parte superior de la misma se puede ver el retrato fotográfico de una sala, con una línea punteada indicando el plano sobre el cuál el sensor láser ha tomado las medidas. Por otro lado, en la parte inferior, se muestra una representación gráfica del barrido láser que realizó el sensor en dicha sala. En la imagen obtenida se detectan dos tipos de características que son de vital importancia en este caso. Ellas son los puntos de ruptura y los segmentos. Los puntos de ruptura se refieren a discontinuidades en el proceso de sensado, que muchas veces se dan por la presencia de objetos que se encuentran por delante de otros más lejanos. Detectar estas rupturas permite clasificar los puntos en grupos llamados clusters. Por otra parte se le llama segmentos a los conjuntos de puntos consecutivos que se unen formando una línea recta, y que son el resultado de escanear superficies planas del entorno como mesas, paredes, etc.

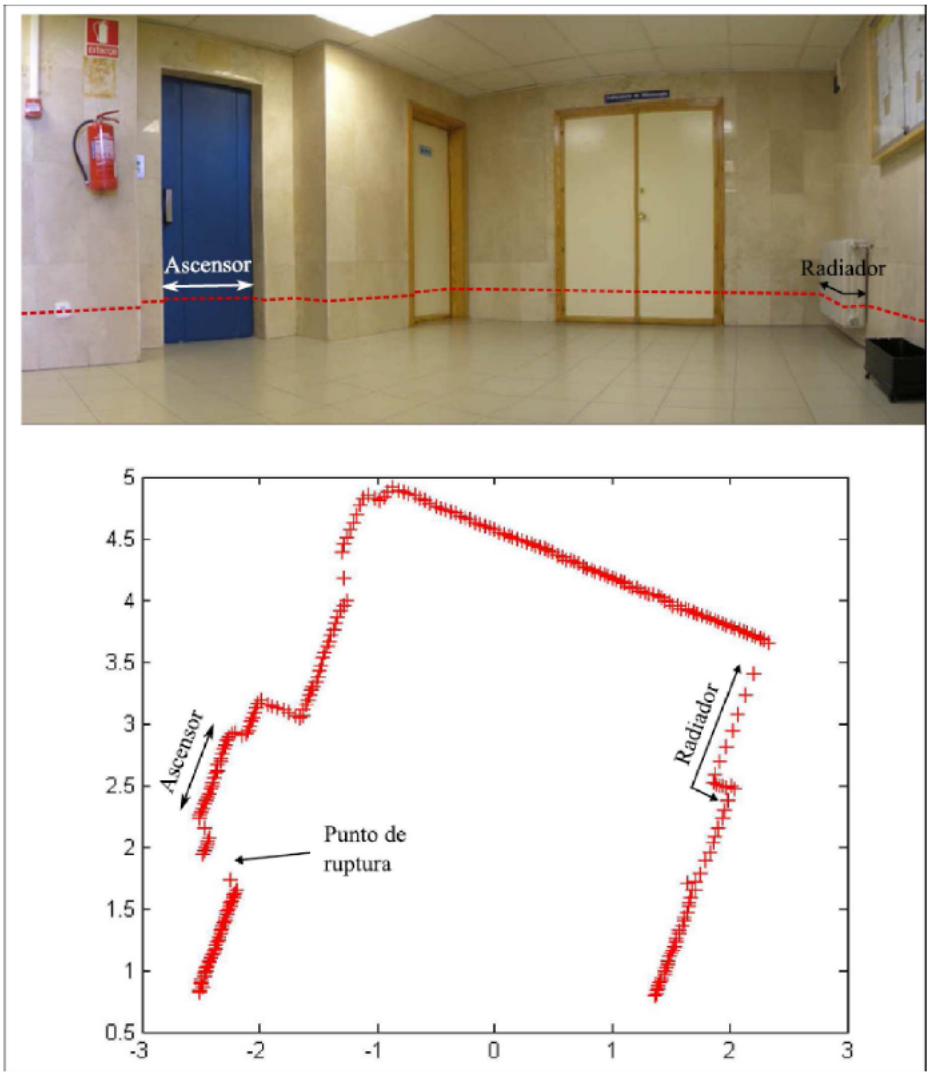

Figura 3.4.1 – Barrido láser.
Fotografía de un escenario real junto con su correspondiente barrido láser**[28].**

3.4.2. Enfoque Probabilista

Los algoritmos que siguen este enfoque resuelven el problema SLAM utilizando probabilidades para el control de errores en los sensores del robot. Dos de los más utilizados son: EKF SLAM, y Fast-SLAM. Un estudio detallado y comparación de performance de los mismos, se puede encontrar en [32], que resume las siguientes conclusiones:

- Para ambos algoritmos el tiempo de ejecución se ve afectado por la cantidad de landmarks que se consideren en el ambiente.
- Fast-SLAM ofrece una solución más escalable, es capaz de incorporar más cantidad de landmarks sin la penalización en la eficiencia como pasa en el caso de EKF SLAM.
- A pesar de la escalabilidad extra, Fast-SLAM ofrece mucha menor precisión en el mapa.
- Fast-SLAM ofrece una solución con más resistencia, ya que es menos probable encontrar fallas como consecuencia de un error en la etapa de asociación de datos.

El objetivo general de estos métodos, es encontrar la función de probabilidad que modela la posición del robot y del mapa del entorno en un momento dado.

Para esto se definen las siguientes variables estocásticas:

- x_i variable que modela la posición del robot en el instante de tiempo i.
- m_i modela el mapa en el tiempo i.
- z_i modela la observación realizada en el tiempo i.
- u_i modela los actuadores del robot en el tiempo i, por ejemplo velocidad de cada motor, distancia a recorrer, etc.

El tiempo t se toma como un conjunto discreto de instantes, que suelen coincidir con los momentos en los que el robot recibe información. Los subíndices en las variables estocásticas definidas se corresponden con el instante de tiempo al que corresponden estas variables. Tomando en cuenta lo anterior, la probabilidad a estimar para solucionar el problema de SLAM se formula como:

- $p\left(x_1 : t \mid z_1 : t, u_1 : t\right)$

Es decir, la distribución de probabilidad de estado $x_i : t$ que se adapta mejor a las observaciones $z_i : t$ y actuadores $u_i : t$. En conclusión, para resolver el problema de SLAM de esta forma, se deberá estimar la función de probabilidad de la variable x_i en cada instante de tiempo de $_1$a t. Para realizar dicha estimación, se utilizan Redes Bayesianas [21, 27].

3.5. Dificultades de SLAM

3.5.1. El Sensado

El sensado es una de las grandes fuentes de error en el SLAM. Los sensores están limitados en lo qué pueden sensar, y en cómo lo pueden hacer. Todos los tipos de sensores presentan ruido estocástico, lo que los hace poco confiables. Además, como en muchos de los casos el sensado se realiza con el robot en movimiento, el error aumenta en gran medida.

A continuación se presentan los sensores más utilizados.

3.5.1.1. Sensor de distancia láser

Es de los sensores más utilizados en el área de la Robótica. Utiliza las técnicas (TOF, Time of Flight) y Phase-Shift (Corrimiento de fase). Un ejemplo de un SLAM que utiliza un sensor de distancia láser Hokuyo se puede encontrar en [33].

VENTAJAS:

- Es rápido, de hecho en la mayoría de los casos su medida se puede considerar casi instantánea.
- La precisión es bastante buena, varía en el rango de los 2,5 cm en equipos económicos como el Lidar que se utiliza en el laboratorio, a menos de 1mm en equipos más caros.
- La resolución angular que presenta supera la de sonar.
- Los datos obtenidos pueden ser interpretados directamente como el alcance hacia dicho objeto en una dirección específica.

DESVENTAJAS:

- Los equipos que permiten una visión de 360 grados son caros (por encima de los 400 dólares en USA). Se puede conseguir sensores láser de luz pulsada con precios que van desde los 75 dólares, con rangos de1 cm de apreciación. Sin embargo, para obtener una visión 360 grados en un plano con estos últimos, se debe girar el láser y tomar diferentes medidas.
- Algunos materiales como el vidrio y el cristal son transparentes para el láser.
- Normalmente, los datos son limitados a un plano, aunque existen formas de obtener datos 3D.

En la figura 3.5.1 se presentan algunos sensores láser que se encuentran disponibles comercialmente y que utilizan el método TOF [28].

NOMBRE	MODULACIÓN	MEDIDA (RANGO)	PRESICIÓN	MEDIDA (TIEMPO)
LaserTech Impulse 100LR	Pulso (900 nm)	0-575m	3 cm @ 50 m, obstáculo blanco	0.3-0.7 s
Riegl FG21-HA	Pulso (904 nm)	2-600 m	+/- 5 cm	0.1-1 s
Riegl LD90-3100VHS-FLP	Pulso	2-200 m (reflejo 80%)	+/- 2.5 cm	0.5 ms
LaserOptronix LDM500 MIL	Pulso	0-999 m	+/- 1 m	-
Leica DISTO pro	Onda senoidal	0.3-100m	+/- 3 mm	0.5-4 s
LaserOptronix PH30	Onda senoidal	0-30m	+/- 5 mm	-

Figura 3.5.1 – Comparativa de sensores láser comercializados[28]

3.5.1.2. Sonar

Este tipo de sensor también ha sido muy utilizado en la Robótica debido a su bajo costo. Su principio de funcionamiento consiste en utilizar energía acústica para efectuar medidas de distancia. Al igual que los sensores láser, los sonares utilizan la técnica TOF [28]para realizar mediciones. Su uso puede ser bueno en ambientes bajo agua. Un ejemplo de un SLAM que utiliza este tipo de sensores se puede encontrar en [34].

VENTAJAS:

- Muy económico.
- Poseen alto rango de precisión (normalmente del 1 % del valor real).
- Puede operar en cualquier condición de luminosidad.
- Son sencillos de controlar. Muchos artículos de investigación los utilizan por lo que los algoritmos de control están al alcance de la mano.

DESVENTAJAS:

- Sufren del problema de eco débil y crosstalk. El primero sucede cuando el poder de la señal recibida es muy bajo, y se necesita mucho tiempo para alcanzar el umbral de detección. Por otro lado, el segundo se da cuando varios sonares envían una misma onda, y puede pasar que se reciban las ondas enviadas por otro sensor.

- Resolución angular pobre. Debido a su bajo precio esto se suele solucionar combinando varios sonares. En este caso son factibles problemas de interferencia de sonido, que deben ser tenidos en cuenta.

Debido a las limitaciones del sonar, utilizar únicamente estos sensores es difícil en la práctica. Sin embargo existen técnicas que permiten extraer información geométrica de las características en un grado aceptable [35].

3.5.1.3. Visión

Estos tipos de sensores utilizan imágenes en color, o escalas de grises para extraer información útil. Un ejemplo de SLAM que utiliza una cámara como sensor se puede encontrar en [34].

VENTAJAS:

- Gran cantidad de información.
- Capacidad de obtener información 3D sobre el ambiente.
- Las cámaras son sensores pasivos, no emiten ni luz ni sonido.

DESVENTAJAS:

- Se requiere gran poder de cómputo para extraer la información de imágenes.
- La visión puede ser perjudicada por la luminosidad.

3.5.2. Manejo de Incertidumbre

Como se mencionó anteriormente, en SLAM conviven los problemas de crear un mapa y localizarse simultáneamente. Para localizarse es necesario usar un mapa, y para crear el mapa es necesario localizarse. De esta forma, la posición acumula error en el mapa y viceversa, por lo que es necesario manejar dicho error correctamente para que no se acumule de forma divergente.

3.5.3. Actuadores

Los actuadores son los responsables de ejecutar el movimiento del robot, de la forma que lo indique la unidad de control. En navegación, la estimación de la posición a través del cálculo de la rotación de las ruedas es llamada odometría. La realidad marca que dicho cálculo no es preciso ya que presenta una desviación no despreciable, la cual es generada por dos tipos de errores: errores sistemáticos (que son debido a las imperfecciones en la estructura del robot), y errores no sistemáticos (debidos a causas externas, por ejemplo el deslizamiento del robot debido a la superficie donde transita, la intervención del humano, etc.). Si bien los errores del tipo sistemático pueden ser compensados en cierto grado [36], los no sistemáticos son más difíciles de solucionar. El uso de sensores internos más avanzados como giroscopios o acelerómetros puede reducir este margen de error pero no eliminarlo. La única manera de solucionar este problema es utilizando sensores exteroceptivos [37], es decir aquellos sensores que basan sus medidas en la correspondencia con objetos externos.

3.5.4. Reconocimiento de lugares

El robot debe reconocer lugares por los que ya pasó, para luego ubicarlos en el mapa de forma correcta. Por otra parte también debe tener la precaución de no confundir lugares similares e identificarlos como el mismo si efectivamente no lo son.

3.6. Navegación [21]

La navegación es una de las funciones más desafiantes a desarrollar en un robot, en parte porque involucra prácticamente todo acerca de la robótica y la inteligencia artificial: sensado, actuación, planificación, arquitecturas, hardware, eficiencia computacional, y resolución de problemas. A su vez, es una actividad crítica para todo robot que quiera desarrollar movilidad. Si bien todos los robots reactivos poseen comportamientos para moverse a través del entorno sin colisionar, la navegación refiere a un concepto mucho más intencional y requiere deliberación, es decir que el robot tiene que ser capaz de planificar cómo llegar a un determinado lugar. A partir de esto surgen dos categorías de técnicas: Navegación Topológica y Navegación Métrica, también conocidas como Navegación Cualitativa y Navegación Cuantitativa respectivamente. Las funciones de navegación pueden ser definidas por cuatro preguntas:

1. ¿HACIA DÓNDE ESTOY YENDO?: Normalmente esta parte está predeterminada por un humano o por algún planificador, es por esto que muchos roboticistas no la consideran parte de la navegación, y asumen que el robot ha sido dirigido hacia un objetivo en particular
2. ¿CUÁL ES LA MEJOR FORMA DE LLEGAR AL OBJETIVO?: Es el problema de la "Planificación de caminos", y una de las más importantes en la navegación. En este sentido se tiene dos tipos de planificación de caminos: cualitativos y cuantitativos.
3. ¿DÓNDE HE ESTADO?: Construir un mapa de lo que se ha recorrido es parte del trabajo del robot.
4. ¿DÓNDE ESTOY?: Para poder construir un mapa de los lugares recorridos, el robot tiene que poder localizarse.

3.6.1. Memoria espacial

La memoria espacial, que es una estructura de datos que permite mapear el espacio recorrido, fundamental para responder la tercer y cuarta pregunta, puede ser representada de dos formas:

1. REPRESENTACIÓN DE RUTA (CUALITATIVA): Expresan el espacio en términos de conexiones entre puntos de referencia. Por ejemplo: "Siga derecho hasta que vea el cartel de Pare, luego doble a la derecha, hasta que llegue a la esquina..." En este sentido son dependientes de la perspectiva, ya que a veces no es tan trivial el hecho de reconocer las señales, especialmente para robots. También es importante destacar que las instrucciones dadas son egocéntricas por lo que asumen que el agente las va siguiendo a cada paso.
2. REPRESENTACIÓN SEGÚN DISPOSICIÓN (O MÉTRICA): Son lo contrario a las representaciones de ruta. Es dar un mapa del entorno, por eso es que se llama también representación métrica, ya que los mapas suelen estar en escala. En oposición a la representación de ruta, es una perspectiva global del entorno, y no una vista egocéntrica del mismo, por lo que no depende de la perspectiva del agente (es independiente de la orientación y la posición).

De acuerdo a las características de este trabajo de investigación, el mismo se acerca más hacia un enfoque métrico, por lo que se analizará un poco más en detalle estos modelos en la subsección siguiente.

Si bien al igual que en la navegación topológica se tiene el objetivo de encontrar un camino hacia el objetivo, hay algunas diferencias significativas entre los dos modelos de navegación. En primer lugar los métodos métricos por lo general favorecen técnicas que generan un valor óptimo, según alguna medida de "qué es mejor", mientras que los métodos cualitativos suelen encontrar una ruta con puntos de referencia identificables. Otra diferencia, es que los métodos métricos, por lo general suelen descomponer el camino en "subobjetivos", que consisten

en puntos de paso, que suelen ser lugares fijos o coordenadas (x,y). Vale la pena destacar, que estos puntos de paso suelen ser puntos de referencia significativos en el mundo matemático, pero a diferencia de la navegación topológica, no necesariamente tienen que tener alguna correspondencia con puntos de referencia del mundo real. En general, los métodos métricos son más compatibles con la deliberación, mientras que los métodos cualitativos se manejan mejor con sistemas más reactivos. Como función deliberativa, los métodos métricos deben afrontar montones de dificultades, como la representación del entorno, manejar cambios dinámicamente, y cierta complejidad computacional.

3.6.2. Métodos métricos

Los planificadores métricos de caminos poseen dos componentes: La representación del espacio (estructuras), y el algoritmo. La primer tarea de estos planificadores, es particionar el espacio en una estructura amena para la planificación de caminos. Para esta tarea utilizan una gran variedad de técnicas para representar el mundo, de las cuales, si bien se destaca alguna en popularidad, como es el caso de "Regular grids", no existe una técnica dominante sobre las otras. En general la intención de todas ellas es de representar solamente las características más destacadas, o la configuración más relevante de objetos relevantes para la navegación en el espacio de interés.

3.6.2.1. Configuración del espacio

En [21], al espacio real en donde se encuentran los robots se le denomina "World Space", y a la estructura de datos que se utiliza para representarlo se le denomina espacio de configuración (o "CSpace"). Dada la definición anterior se considera que un buen espacio de configuración, es aquel que reduce al mínimo posible el número de dimensiones que el planificador tiene que tomar en cuenta para representar adecuadamente toda la información necesaria. Por ejemplo, es fácil ver que si uno se maneja en un plano bidimensional, la dimensión z es innecesaria y podría no ser tenida en cuenta.

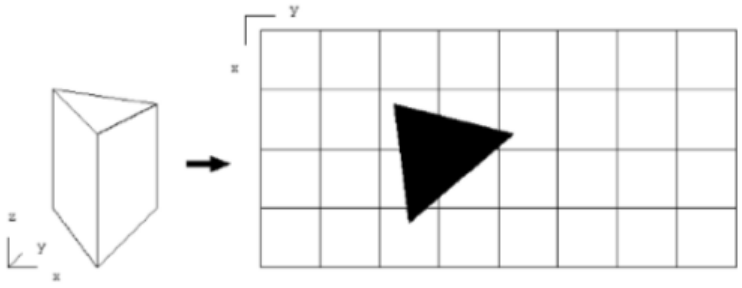

Figura 3.6.1 – Transformación de un objeto a un espacio de dos dimensiones.[21]

La figura 3.6.1 muestra la transformación de un objeto a un espacio de dos dimensiones. Cabe destacar que la mayoría de los algoritmos métricos de planificación de caminos asumen dos dimensiones. En este sentido para la planificación de caminos, se suele asumir al robot como un objeto redondeado, de tal manera que la orientación no sea algo importante.

3.6.3. Diferentes representaciones Cspace

Dada la enorme cantidad de diferentes representaciones existentes, se detallará solamente las más representativas. Básicamente las diferencias entre las mismas radican en las diferentes formas de particionar el espacio libre. Cada espacio abierto que no se encuentra ocupado por un objeto es espacio abierto, donde el robot tiene la libertad de moverse sin chocar contra ningún objeto.

3.6.3.1. Meadow Maps

Muchos de los algoritmos de planificación de caminos asumen desde el inicio la disponibilidad de un mapa preciso del entorno, que luego será procesado de alguna forma, y a través de algunos algoritmos, llevado a alguna representación Cspace adecuada. Meadow Maps son un ejemplo de una representación Cspace de este tipo. Básicamente, transforman el espacio libre en polígonos convexos, los cuales tienen la característica de que si el robot se encuentra en cualquier punto del polígono, y toma la decisión de ir en línea recta hacia cualquier otro punto del mismo, no saldrá nunca del polígono en el trayecto. En otras palabras, el polígono representa para el robot una región segura para moverse, por lo que el gran desafío se remite en elegir la mejor serie de polígonos para transitar el espacio. En la práctica los Meadow Maps no son muy utilizados en el área de robótica pero sirven para ver de forma simple los espacios de configuración y luego planificar un camino sobre el.

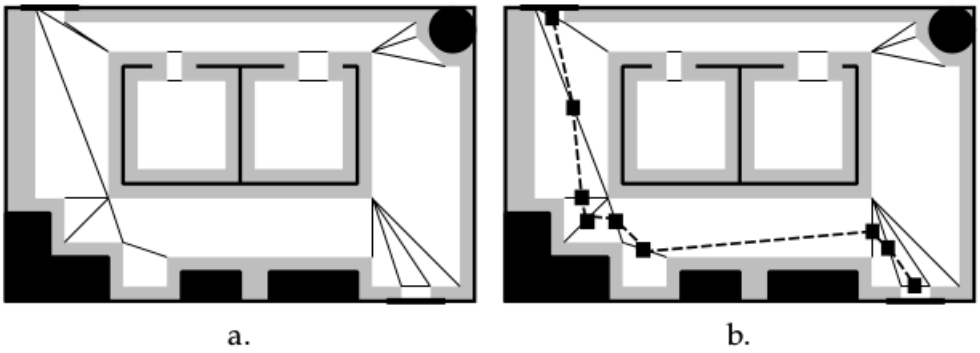

Figura 3.6.2 – **Meadow Maps**[21]

Como se ve en las figuras anteriores el algoritmo consta de pocos pasos:

- En primer lugar se parte de un diseño métrico del entorno.
- A partir de ese momento se expanden todos los obstáculos para considerar al robot como un punto.
- Luego se particiona el espacio libre en polígonos convexos considerando los segmentos de línea entre pares de puntos con alguna característica interesante, como pueden ser esquinas, puertas, o límites de objetos.
- Así que la cuestión se convierte en cómo especificar puntos candidatos en el polígono. Una solución es encontrar el punto medio de cada segmento de línea que limita otro polígono. De esa forma esos puntos serían los nodos de un grafo, y las líneas que los unen las aristas del mismo. El algoritmo puede entonces determinar qué caminos tomar.

3.6.3.2. Grafos de Voronoi

Son un mecanismo de representación de un espacio de configuración, y de creación de un grafo simultáneamente. Estos son capaces de construir un mapa topológico a medida que exploran un entorno nuevo. La idea es generar una línea, equidistante de todos los puntos, que se llamará la Arista de Voronoi, y como se ve en la figura 3.6.3, la línea pasa por el centro de todas las entradas y pasillos. Asimismo el punto donde se encuentran varias Aristas de Voronoi, es llamado Vértice de Voronoi. Vale la pena mencionar que existe a menudo una correspondencia física entre los vértices y el espacio libre que puede ser sensado, por lo que resulta fácil para un robot seguir un camino generado por un Grafo de Voronoi, dado que siempre se mantendrá equidistante de todos los objetos.

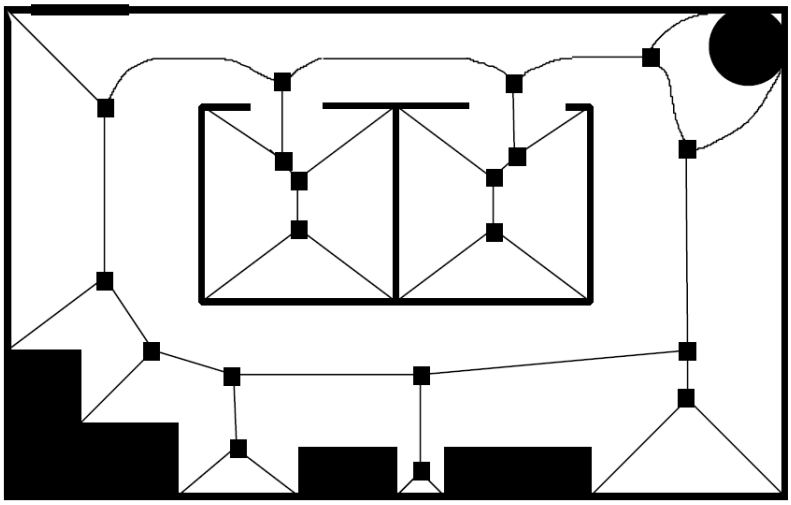

Figura 3.6.3 – **Grafo de un Grafo de Voronoi Generalizado (GVG)** [21]

3.6.3.3. Regular Grids

Es otra forma de particionar el espacio. Consiste en superponer una cuadrícula cartesiana de dos dimensiones con el espacio, como se muestra en la figura 3.6.4.

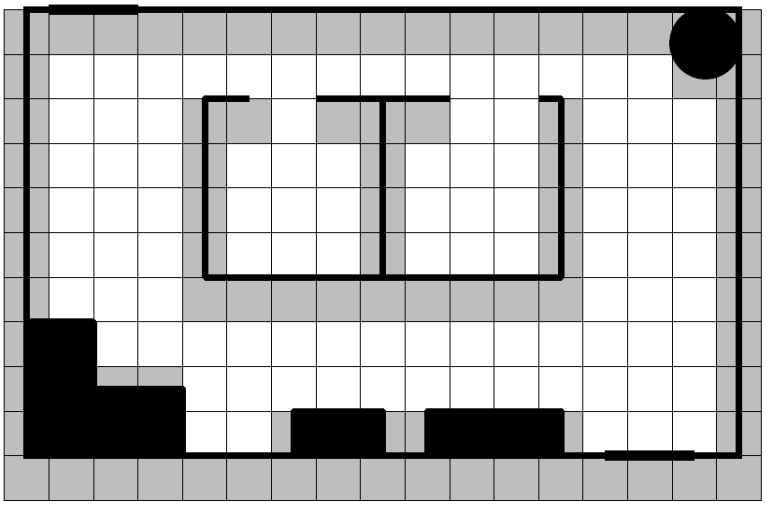

Figura 3.6.4 – **Regular Grids**[21]

La idea de fondo es manejar una cuadrícula de ocupación, en la que un cuadrado que contiene parte de un objeto se marca como ocupado, y en caso contrario se marca como libre. Por otra parte, al centro de cada cuadrado de la cuadrícula se lo ve como un nodo de un grafo altamente conectado. Asimismo se llaman grafo 8-conectado si se permiten arcos diagonales entre los nodos, y grafo 4-conectado en el caso contrario. Este tipo de representación, tiene como actividad clave definir un tamaño adecuado de cuadrado. Por un lado si los cuadrados son muy grandes, el espacio libre se verá reducido, y por otro lado cuanta mayor granularidad se desee tener, más costoso será el algoritmo computacionalmente hablando, ya que el número de nodos se verá aumentado proporcionalmente.

3.6.3.4. Árboles cuaternarios

Son una variante de los Regular grids, que busca solucionar el problema de desperdicio de espacio libre que se sufre en la versión anterior. Dicho problema se soluciona manejando recursivamente la división de los cuadrados, y subdividiéndolos dinámicamente en 4 cuando un objeto ocupa solo una parte de ellos. Conceptualmente se busca que los cuadrados ocupados estén realmente ocupados, y no malgasten espacio libre limitando la capacidad de movimiento del robot.

Parte II

Implementación

Como se mencionaba parcialmente en el marco teórico, no se terminó realizando el modelo TAM-WG sino que se implementó un algoritmo de SLAM cuyo objetivo era servir luego para generar el grafo del WG. Por las dificultades encontradas en SLAM, el proyecto se limitó a resolver este problema. Además, para mostrar cómo se podría utilizar este módulo para generar los estados del WG, se implementó un algoritmo que divide al espacio en regiones convexas.

A continuación, en el capítulo 4 se presenta el algoritmo de SLAM diseñado, en el 5 el de la división del espacio en regiones convexas, y en el 6 se describe la implementación de los módulos físicos (es decir del robot y el laberinto).

Capítulo 4

Algoritmo SLAM diseñado

En este capítulo, primero se intenta dar una descripción de cómo se descompone el algoritmo en general, comentando algunas de las hipótesis que fueron utilizadas y los motivos detrás de ellas. Luego, cada sección subsiguiente detalla una de las componentes del algoritmo.

4.1. Descripción General

El algoritmo de SLAM diseñado es un SLAM métrico. En este se construye un mapa que pretende serle fiel a la realidad utilizando información sensorial láser. El algoritmo fue diseñado específicamente para laberintos de roedores como por ejemplo el de 8 brazos, en los que se cuenta con un ambiente estático, libre de obstáculos y de paredes planas. Además, se ha de tener en cuenta que, originalmente, se asumía que el sensado era realizado con una frecuencia relativamente alta (varias veces por segundo). De ser así, el desplazamiento del robot entre iteraciones sería relativamente pequeño, y la posición obtenida por el algoritmo de SLAM una iteración podría ser usada como una buena estimación al comienzo de la próxima.

Si bien existen sensores que permiten utilizar la última hipótesis, estos tienen un costo relativamente alto y escapan al presupuesto del proyecto. Con el sensor utilizado, obtener las medidas necesarias por cada iteración insume alrededor de 30 segundos. De esta forma, entre iteración e iteración, fue necesario hacer que el robot se desplace una distancia suficientemente grande (alrededor de 40 cm) para que el robot pueda recorrer todo el laberinto en un tiempo razonable. Por este motivo, la posición obtenida por el algoritmo en el tiempo anterior no pudo ser usada como un buen estimativo de la nueva y fue necesario incluir un módulo de integración de trayectorias (PI - path integration). No obstante de ello, el algoritmo fue implementado para no usar PI por lo que este módulo no se explica, y en este capítulo se asumirá que, al principio de cada iteración, el robot cuenta con una aproximación de su posición y dirección de la cabeza.

Como se puede ver en el siguiente pseudocódigo, a grandes rasgos, el algoritmo consta de 4 grandes pasos: sensado y extracción de características, identificación de características, ubicación/orientación del robot y de las características extraídas, integración de la información.

Algorithm 1 SLAM

```
1: Sensar y extraer características.
2: Inicializar posición robot.
3: Crear mapa inicial posicionando las características.
4: while true do
5:     Sensar y extraer características.
6:     Identificar características.
7:     Ubicación y orientación del robot y características extraídas.
8:     Integración información.
```

- En el primer paso lo que se hace es obtener la información sensorial y luego buscar características en ella que puedan ser fácilmente reconocibles por el robot. La información sensorial utilizada consta de la distancias del robot a las paredes, medidas mediante un sensor láser. Por otro lado, las características extraídas son segmentos orientados formados a partir de la nube de puntos obtenida. Estos intentan representar las porciones de paredes observadas. La idea detrás de esto, es que reconocer una pared permite orientar fácilmente la dirección de la cabeza y reconocer dos paredes permite ubicar al robot en el espacio. Esto se verá mejor en la sección correspondiente.
- En el segundo paso, teniendo en cuenta una aproximación de la posición y orientación del robot, una vez extraídas las características del espacio observado, estas deben ser identificadas comparándolas contra las características previamente halladas y registradas en el mapa.
- En el tercer paso, las características reconocidas son utilizadas para refrescar (calcular) la posición y orientación del robot en el mapa.
- Finalmente, hecho lo anterior, se procede a integrar la información nueva a la vieja. Es decir, se agregan al mapa los segmentos que no fueron reconocidos y se mejora las posiciones, direcciones y extremos de los que sí lo fueron. De esta forma se amplía y mejora la información del mapa generado, con lo cual, si el algoritmo funciona bien, se tenderá a disminuir los futuros error cometidos en el paso tres y a aumentar la zona en donde el robot puede ubicarse de esta manera.

Algo a tener en cuenta, es que en el paso tres es necesario reconocer al menos dos segmentos no paralelos, o un segmento con al menos un vértice representando una esquina real. De otra forma no sería posible calcular la posición del robot o inclusive tampoco la dirección de la cabeza (si no se reconociera ni un segmento). Si bien este problema puede darse, en este proyecto se ignora ya que rara vez ocurre. Los motivos por los que el problema podría darse son dos. Uno de ellos se debe a que el algoritmo falle en el segundo paso y no reconociera características que sí se debería reconocer. Esto se podría dar por ejemplo si se tiene mucho error en la aproximación de la posición u orientación, y/o en la información de los segmentos (ya sea los observados o los del mapa). En el proyecto, la única causa observada de esto se debe al error dado por el precario módulo de PI.

El otro motivo por el que podría darse sería si realmente no se observaran suficientes paredes no paralelas que puedan ser reconocidas. Por ejemplo, si se estuviera en un pasillo muy largo solo se distinguirían dos paredes paralelas con lo cual sería posible calcular la dirección de la cabeza pero no la ubicación. A su vez, si se estuviera en el medio de una habitación lo suficientemente grande, quizás no se podría distinguir ninguna pared y de esta forma no se podría derivar ningún resultado. En el proyecto, bajo las hipótesis utilizadas y con los laberintos utilizados, este problema se puede dar casi únicamente en pasillos muy largos. El otro caso en el que podría pasar sería si el robot se moviera demasiado de una iteración a la otra. En la nueva el iteración el robot podría llegar a ver solamente paredes nuevas, y de esta forma no reconocer ninguna. No obstante, se ha de tener en cuenta que el algoritmo fue diseñado para moverse poco en cada iteración. De respetarse esto, si el robot

no tiene problemas sensoriales, siempre debería poder reconocer al menos un segmento ya que en un ambiente estático una pared no puede dejar de verse instantáneamente.

Por los motivos anteriores, cuando el problema se presenta en esta implementación, este simplemente es ignorado abortando la iteración y continuando a la siguiente. Como se verá más adelante, esta política tuvo malos resultados en una única ocasión, en la cual se tuvo el problema por desplazarse mucho de una iteración a la otra.

Finalmente, cabe destacar que en ambientes donde esto pudiera ser un problema más crítico, el algoritmo podría mejorarse utilizando un mejor módulo de de PI e integrando su salida de mejor manera, y/o cambiando el sensor láser utilizado por uno (o varios) que tengan capacidades de medir distancias en varias direcciones a la vez.

4.2. Características utilizadas y su detección.

4.2.1. Uso de segmentos como características.

Como se mencionó anteriormente, las características que se utilizan en este algoritmo, son segmentos que representan porciones de paredes. La motivación para esto vino de las siguientes fuentes.

En primer lugar, se ha de tener en cuenta los ambientes en los que debe trabajar el robot. Estos son laberintos de paredes planas que simulan los utilizados en algunos experimentos con roedores, como por ejemplo el laberinto tipo T y el de 8 brazos. Observar que en estos ambientes no hay otros elementos además de las paredes. Debido a estos motivos, originalmente surgió la idea de que las esquinas podrían ser características fácilmente reconocibles, que permitieran al robot calcular su posición y orientación. Para ello, sería necesario poder hallarlas de alguna forma, con lo cual, se consideró que una buena idea sería almacenar en el mapa la información de las paredes. Además, otra ventaja que tendría esto, es que, permitiría realizar un mapa métrico del espacio el cual podría ser usado luego en el WG para detectar los estados (regiones donde las posibilidades de movimiento son "las mismas").

Finalmente, la idea de las esquinas evolucionó un poco. Se observó que las esquinas simplemente son intersecciones de dos rectas, y que en realidad, dadas dos rectas no paralelas cualesquiera, es posible generar puntos imaginarios que sean igualmente útiles para ubicarse. Esto permite aumentar bastante la cantidad de puntos que se pueden utilizar.

De esta forma, por todos estos motivos, se decidió que las características a utilizar serían segmentos.

4.2.2. Algoritmo de extracción se segmentos.

Dado que se decidió extraer segmentos de la nubes de puntos obtenidas por el sensor láser, la siguiente cuestión que surge es: ¿Cómo se debe realizar? Para entender mejor esto, primero se analizará una cuestión de conectividad, luego se profundizará en cómo extraer las rectas, y finalmente se comentará la información extra que se le adjunta a los segmentos, así como la estructura de datos utilizada.

4.2.2.1. Cuestiones de conectividad.

Una de las primeras consideraciones a tener en cuenta es que, para que un conjunto de puntos conforme una pared, estos han de estar conectados. Que un conjunto de puntos consecutivos de la nube estén alineados, no basta para indicar que pertenecen a una misma pared. Como los puntos observados son un conjunto discreto, a priori, no es posible garantizar que no haya huecos entre ellos. He aquí que el problema radica: a partir del sensado realizado, no es posible derivar la información de conectividad para un entorno cualquiera, con lo cual, surge la necesidad de definir algún criterio que permita definir cuándo dos puntos están conectados. Para ello, se realiza la siguiente definición:

Se dice que dos puntos x, y están conectados si solo si $\exists x_1, ..., x_n$ tal que $x_1 = x$, $x_n = y$, y $\forall i < n :: d(x_i, x_{i+1}) < K$ donde d es la función distancia euclideana y K es una constante.

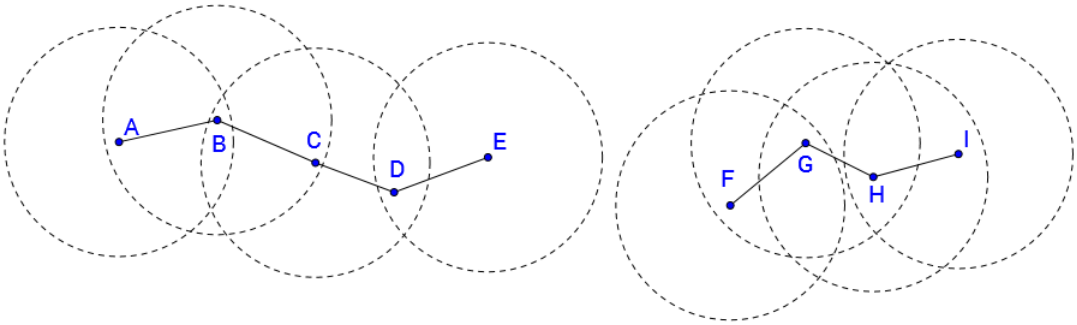

Figura 4.2.1 – Conexión de puntos discretos.
En la figura se muestra un ejemplo de conexión. Los conjuntos de puntos $\{A, B, C, D, E\}$ y $\{F, G, H, I\}$ forman las componentes conexas. Cualquier par de puntos dentro de una misma componente están conectados, pero pares entre diferentes no.

Para que la definición tenga sentido, la constante K tiene que ser lo suficientemente pequeña para no considerar unidos a huecos que el robot pueda atravesar. A su vez, tiene que ser lo suficientemente grande para que la definición sea de utilidad.

El mínimo límite superior de K viene dado por las dimensiones del robot. Si dos puntos consecutivos están a menor distancia que eso, los puntos pueden ser considerados como unidos ya que el robot no puede atravesar por ese camino[1].

Por otro lado, las características del ambiente permiten incrementar ese límite. Por ejemplo, si se sabe que los pasillos de un laberinto son de 0.5 metros, dos puntos consecutivos que estén a menor distancia han de pertenecer a la misma pared.

De esta forma, por los motivos antedichos, si no se tuviera ruido se cumpliría que $K_{optimo} = max\{K_{robot}, K_{ambiente}\}$.

Si la resolución de la nube de puntos[2] fuese demasiado baja, el problema que se tendrá, es que será difícil decir (por este método) que dos puntos consecutivos estén unidos. Esto se debe a que las distancias entre puntos tenderán a ser mayores. De esta forma, si la resolución es muy chica, este mecanismo se vuelve obsoleto. En el proyecto, para una resolución de 100 puntos (3.6 grados entre pares) y laberintos con pasillos de 0.4 y 0.5 metros, se utilizó $K = 0{,}25$.

Utilizando esta definición, es posible dividir a la nube en componentes conexas. Esto simplifica el problema de encontrar los segmentos de paredes. En vez de buscarlos en la nube original, se los puede buscar en cada "sub-nube"[3] por separado. Además, como las "sub-nubes" son componentes conexas, se sabrá que, todos los segmentos obtenidos en cada una estarán conectados entre sí formando un camino de paredes. Por otro lado, observar también que si dos puntos x_i y x_j dentro de una componente forman parte de una misma pared, entonces todos los puntos intermedios deben formar parte de ella.

A continuación se presenta el algoritmo de particionamiento en componentes conexas.

[1] Si bien esto mezcla la idea de un segmento como posibilidad de movimiento con la de representación de pared, esto se hace intencionalmente por motivos prácticos.

[2] La resolución de la nube de puntos es la cantidad de puntos que tiene. A mayor resolución se tiene una mayor cantidad de puntos y una menor cantidad de grados entre puntos consecutivos.

[3] Una “sub-nube” es un subconjunto de puntos de la nube original.

Algorithm 2 **División en componentes conexas.**

Entrada: $\boldsymbol{X = \{x_1, ..., x_n\}}$ nube de puntos ordenada en forma antihoraria.
Salida: $\boldsymbol{K_1, ..., K_m}$ las componentes conexas de $\boldsymbol{X}$

OBS: Los índices son módulo $\boldsymbol{n}$.

Buscar i / $d(x_i, x_{i+1}) > K$
if $\nexists\, i$ **then**
    $K_1 = X$
else
    $i_0 = i + 1$
    $j = i_0$
    $m = 1$
    $K_m = \{x_j\}$
    while $j + 1 \neq i_0$ **do**
        if $d(x_j, x_{j+1}) < K$ **then**
            Agregar x_{j+1} a K_m
        else
            $m = m + 1$
            $K_m = \{x_{j+1}\}$
        $j = j + 1$
Devolver las componente creadas.

4.2.2.2. Encontrando las paredes, problema e idea original.

A partir de las últimas observaciones en la sección anterior, se puede ver que, el problema de hallar los segmentos de paredes se reduce a particionar cada componente conexa de la siguiente forma:

Sea $K = \{x_1, ..., x_n\}$ una componente conexas con los puntos ordenados en sentido antihorario. Entonces la componente puede dividirse en subcomponentes
$$P_1 = \{x_1, ..., x_{i_1}\}, P_2 = \{x_{i_1+1}, ..., x_{i_2}\}, ..., P_k = \{x_{i_{k-1}+1}, ..., x_n\}$$
en donde cada una representa una pared distinta y se compone de los puntos pertenecienes a la misma.

Observar que en la descomposición anterior, en ausencia de ruido, los puntos de una componente están alineados formando una recta ya que las paredes son planas. A su vez, dos componentes consecutivas no pueden estar alineadas ya que serían la misma pared, y además, deben intersectarse en una esquina real[4] ya que ambas están conectadas.

En ausencia de ruido, realizar esta nueva división es un problema trivial. Basta recorrer los puntos e ir agrupando los que estén alineados. Cuando un nuevo punto no esta alineado a los anteriores se comienza una nueva componente. La única consideración a tener en cuenta en este caso es que dos puntos siempre están alineados.

Bajo presencia de ruido, el problema se complica más. Los puntos de una pared dejan de estar alineados, y a veces, puntos de paredes distintas pasan a estarlo. De esta forma, la posibilidad de resolver este problema depende de la intensidad y el tipo de ruido que tengan los datos.

4.2.2.3. Adaptando Hough para nubes de puntos

El algoritmo de Hough, que se presenta con más detalle en el apéndice matemático, es un algoritmo que permite extraer rectas de una imagen. Como se observa en el apéndice, el

[4]Se usa la palabra real para contrastar contra las esquinas creadas artificialmente. Estas últimas se crean intersectando rectas de paredes no paralelas que no están conectadas en la realidad

algoritmo de Hough halla rectas en las imágenes considerando la función de rectas de un punto ($f(\theta) = \vec{x_1} \cdot e^{i\theta}$) para varios puntos. Las gráficas generadas de aquellos que estén alineados se intersectarán en uno o dos puntos dependiendo si se toma el dominio completo de la función o solo la mitad. Cuando se tiene puntos de varias rectas, se generan intersecciones que representan rectas falsas. Para detectar las verdaderas, la idea del algoritmo es buscar los puntos del espacio con mayor densidad de gráficas pasando por ellos.

Cuando no se tiene ruido, para hacer esto bastaría calcular las intersecciones de todo par de gráficas y contar cuantas veces se repita cada una. Sin embargo, para el caso con ruido esto no tiene sentido. El problema es que las gráficas de los diversos puntos de una misma recta, en vez de intersectarse todas en un único punto, cada par podrá intersectarse en uno diferente. Esto puede observarse en la figura 4.2.2.

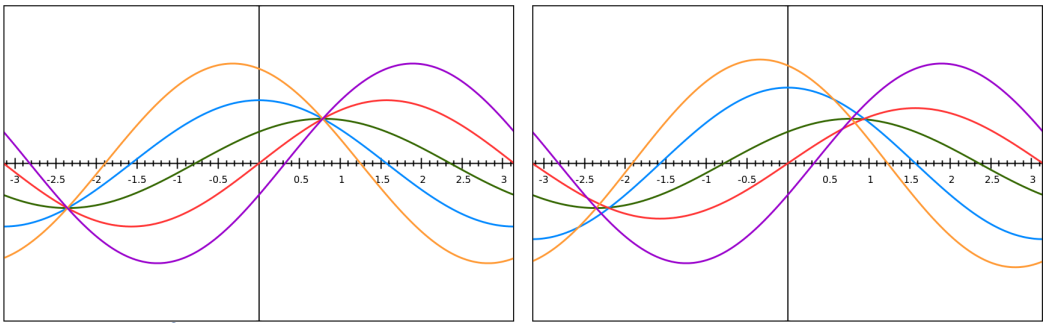

Figura 4.2.2 – Houghs con vs. sin ruido.
En la figura se muestra las funciones de rectas para 5 puntos alineados antes y después de agregar ruido.

Por el hecho anterior, y dado que que el espacio de búsqueda es denso, lo que se hace es dividir al espacio en bins[5] y contabilizar la cantidad de gráficas que pasan por cada uno de ellos. Los bins pueden formar una partición del espacio de búsqueda o pueden estar superpuestos. La función que a cada bin la asigna el número anterior es un intento de aproximar la función "densidad de gráficas por punto". Cuanto más alta sea la densidad de un bin, más probable será que la recta que represente sea real.

[5]Los bins son una estructura computacional de datos que representan una región rectangular de un espacio y permiten consultas eficientes sobre la información que albergan.

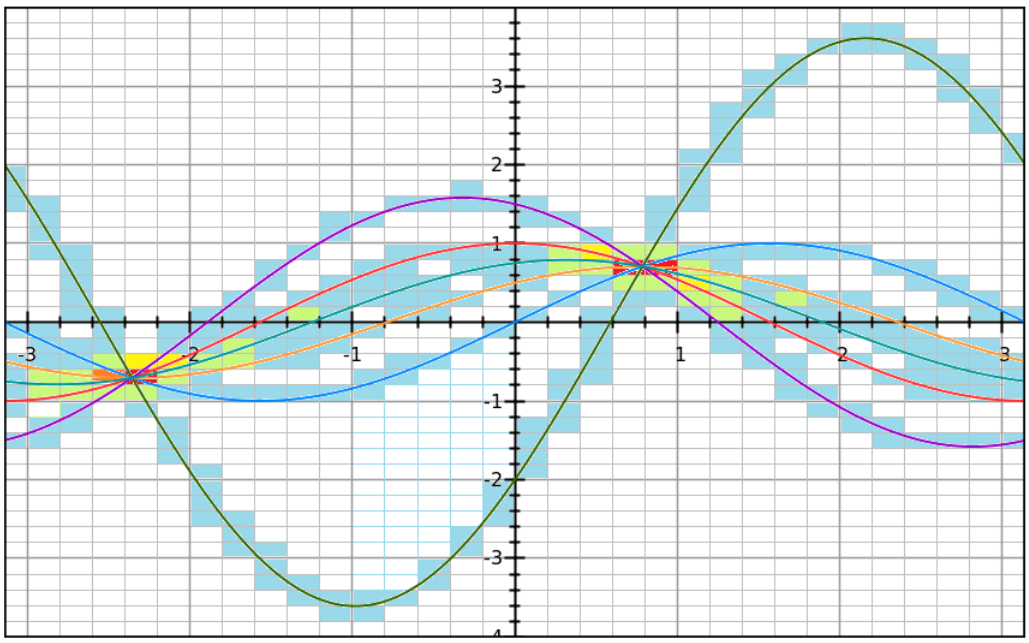

Figura 4.2.3 – Ejemplo de binning.
En la figura se ve la división en bins para el algoritmo de Hough. Las gráficas representan las funciones de rectas para 6 puntos alineados. El color de los bins indica la cantidad de gráficas que contiene. Blanco indica 0, celeste 1, verde 2, amarillo 3, naranja 5 y rojo 6 (no se ven bins con 4 puntos).

A continuación se presenta un esbozo del algoritmo que discretiza el espacio en bins y aproxima la función de densidad para una nube de puntos. Por comodidad se asumirá que los bins particionan el espacio, aunque, como se comentó, no es necesario que sea una partición. Además, solo se usarán bins que representen las distancias negativas. Esto se debe a que si (θ, d) representa una recta, $(\theta + 180, -d)$ también representa la misma. La decisión de tomar d negativa se debe a que esto orienta las normales hacia el origen de coordenadas (posición del robot). Dicha orientación es compatible con la orientación antihoraria de los segmentos que se usa en las distintas secciones ya que, si se aplica la regla de la mano derecha a una recta orientada de esa forma, su normal apuntará hacia el centro.

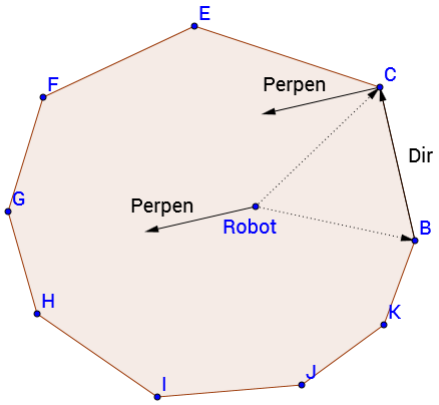

Figura 4.2.4 – Orientación de la nube de puntos.
La figura representa al robot y a una nube de puntos obtenida por el sensado. La nube está orientada en sentido anti-horario. Usando la regla de la mano derecha con el vector Dir se obtiene el vector perpendicular $Perpen$. Como la perpendicular apunta hacia el centro, la recta BC tiene distancia signada negativa.

Algorithm 3 Cálculo de la función de densidad

Entrada: $X = \{\vec{x_1}, ..., \vec{x_n}\}$ nube de puntos.
Salida: $f : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ la función de densidad de gráficas por bin

Constantes: $\Delta =$ tamaño de los bins en el eje de las distancias.
$\Theta =$ tamaño de los bins en el eje tita.

Descripción: La función de salida es solo una aproximación. Para cada vector $\vec{x_i}$ se calcula su función de rectas para un conjunto discreto del dominio (un punto por cada centro en el eje tita de los bins), y se contabiliza en los bins apropiados.

1: Inicializar f en 0 para todo bin
2: $M := \frac{2\pi}{\Theta}$ es la máxima cantidad de bins en tita
3: **for** $i = 1, n$ **do**
4: **for** $j = 1, M$ **do**
5: $\theta_{CentroBin} = j \cdot \Theta - \frac{\Theta}{2}$
6: $d = \vec{x_i} \cdot e^{i\theta_{CentroBin}}$ distancia de la gráfica de $\vec{x_i}$ en θ
7: **if** $d < 0$ **then**
8: $k = round(\frac{d}{\Delta})$ índice para la distancia d
9: $f(j, k) \Leftarrow f(j, k) + 1$
10: Retornar f

Observación: El esbozo presentado es solo una aproximación de la función densidad ya que realmente solo se tiene en cuenta un conjunto finito de puntos de cada gráfica. Cuantos más puntos se tomen de ellas más próximo se estará al valor de la función real. Además, también se debería de tener cuidado de no contar cada gráfica más de una vez por bin.

4.2.2.4. Agregando el modelado del ruido

En esta sección se introducirán mecanismos de soporte para el ruido en el algoritmo 3. Para ello, se comenzará explicando un mecanismo básico al cual se la irá introduciendo varias mejoras.

Antes de modelar el ruido, primero se ha de tener en cuenta qué tipos se puede tener. Debido a que la información sensada consiste de la medición de la distancia para ciertas direcciones, cada dato obtenido tiene ruido tanto en el ángulo (dirección), como en la distancia. Si bien, por completitud, en esta sección se analizan ambos tipos, en la práctica solo se agregó soporte para el ruido en la distancia.

En general, en todos los casos, el soporte para el ruido se agrega modificando la forma de conteo en el algoritmo 3. Es decir, lo que se modifica es el contenido de su bloque de código que comienza en el paso 7 . Cada punto, en vez de afectar únicamente al bin en que yace, también afectará a los bins circundantes en un radio. Esto es conocido con el nombre de estimación de la densidad mediante kernels [38]. La forma más sencilla de esto es, dado un radio para el índice tita y uno para d, se suma 1 a cada bin que quede en el vecindario. Si r_{tita} y r_d son los radios para los índices tita y d respectivamente, el bloque en el algoritmo sería modificado de la siguiente manera:

1: **if** $d < 0$ **then**
2: $k = round(\frac{d}{\Delta})$ índice para la distancia d
3: **for** $a = j - r_{tita}, j + r_{tita}$ **do**
4: **for** $b = k - r_d, k + r_d$ **do**
5: $f(a, b) \Leftarrow f(j, k) + 1$

El principal problema del mecanismo anterior es que no tiene en cuenta las características reales del error.

Por ejemplo, para el láser utilizado, según su manual, el error máximo es de 2.5 cm. Ahora, suponiendo que la distancia para la dirección 0 grados es de 100 cm y que no se cometen errores en la dirección de la cabeza, el punto real observado debería caer en el segmento $[(97{,}5, 0), (102{,}5, 0)]$. Para cada punto $(d, 0)$ en el segmento, su función de rectas será $f(\theta) = d \cdot cos(\theta)$. Si se grafican todas juntas, como se observa en la figura 4.2.5, se puede observar que la figura obtenida es una especie de banda sinusoidal de altura variable en el eje de las distancias.

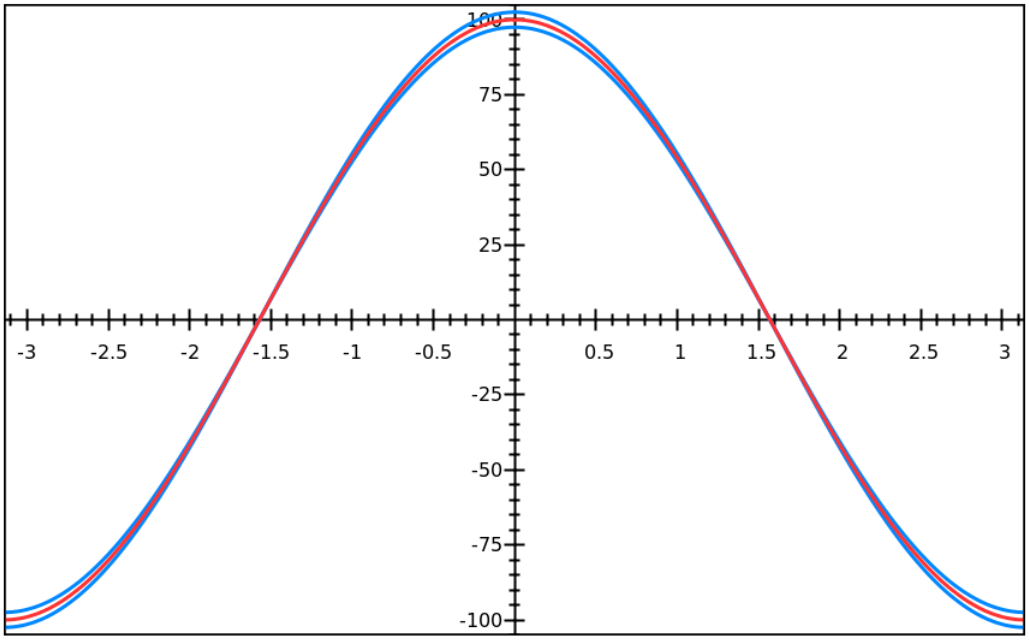

Figura 4.2.5 – Función de rectas de un punto con error en su norma.
En la figura se ven graficadas las funciones de recta para los puntos $(97{,}5, 0), (100, 0), (102{,}5, 0)$. Se puede observar que la altura de la banda varía según tita.

En particular esto implica que en el algoritmo anterior el radio r_d debería ser variable según $\vec{x_j}$ y θ. La fórmula para obtener r_d bajo estas hipótesis viene dada por la siguiente ecuación:

$$r_d = |round(\frac{\frac{max\_error\_d}{\|\vec{x_j}\|} \cdot \vec{x_j} \cdot e^{i\theta}}{\Delta})|$$

La fórmula proviene de restarle la distancia original a la distancia máxima dada por el error máximo. Está ultima se puede obtener aumentando la norma del punto observado en una cantidad igual al tamaño del error máximo ($x_j + max\_error\_d \cdot \frac{x_j}{\|x_j\|}$) y luego aplicándole la función de rectas. Utilizando esta idea en el algoritmo anterior, el bloque pasaría a verse de la siguiente forma:

1: **if** $d < 0$ **then**
2: $\quad r_d = |round(\frac{\frac{max\_error\_d}{\|\vec{x_i}\|} \cdot \vec{x_i} \cdot e^{i\theta_{CentroBin}}}{\Delta})|$
3: $\quad k = round(\frac{d}{\Delta})$ índice para la distancia d
4: $\quad$ **for** $a = j - r_{tita}, j + r_{tita}$ **do**
5: $\quad\quad$ **for** $b = k - r_d, k + r_d$ **do**
6: $\quad\quad\quad f(a, b) \Leftarrow f(j, k) + 1$

Volviendo a utilizar como ejemplo al láser LidarLite, otra consideración a tener en cuenta es la siguiente. Si bien se comete un supuesto error máximo de $\pm$2.5cm, el error no se distribuye uniformemente en el intervalo. Una distribución distribución gaussiana sería más próxima a la distribución real. De esta forma, si el láser indica 100cm, es bastante más probable que el valor real sea más próximo a 100cm que a 97.5cm o a 102.5cm.

Considerando este hecho parece tener sentido que en vez de sumar 1 a cada bin del vecindario, se sume un peso variable que dependa de la distancia del punto original de la gráfica al centro del bin. El resultado que se tendrá es que las rectas más próximas a la gráfica del punto (gráfica del medio de la banda en la figura 4.2.6) sean más probable que las más alejadas.

La distribución de pesos según la distancia (es decir la función kernel) debería estar relacionada a la distribución del error cometido por el láser. Sin embargo, por facilidad, en este trabajo simplemente se probó funciones polinomiales de 1er, 2^{o} y 3er orden de la forma $c - x^n$, donde $x = \frac{k}{r_d}$ en el algoritmo anterior y c es una constante. En el proyecto, utilizando ruido gaussiano con desviación estándar de entre 3 y 6 cm los mejores resultados se obtuvieron usando las funciones $3{,}8 - x$ y $2{,}8 - x^2$.

Llegado este punto, el próximo error que se debería analizar es el de la dirección. Para ello, asúmase primero que la distancia no tiene error. Recordar que la gráfica de rectas para un punto es una sinusoide cuya amplitud es igual a la norma de dicho punto. Observar que como el punto no cambia de norma, lo único que cambia es la fase de la función (ángulo para el cual se maximiza la sinusoide). Prosiguiendo como con el error en la distancia, si se grafican todas las gráficas juntas, lo que se obtendrá es otra banda sinusoidal. En este caso, observando el ancho de la banda en el eje tita, a diferencia de lo que pasaba con el error en la distancia, se verificará es constante. Esto se debe a que toda gráfica es una traslación entre $\pm max\_error\_\theta$ en el eje tita de la gráfica original.

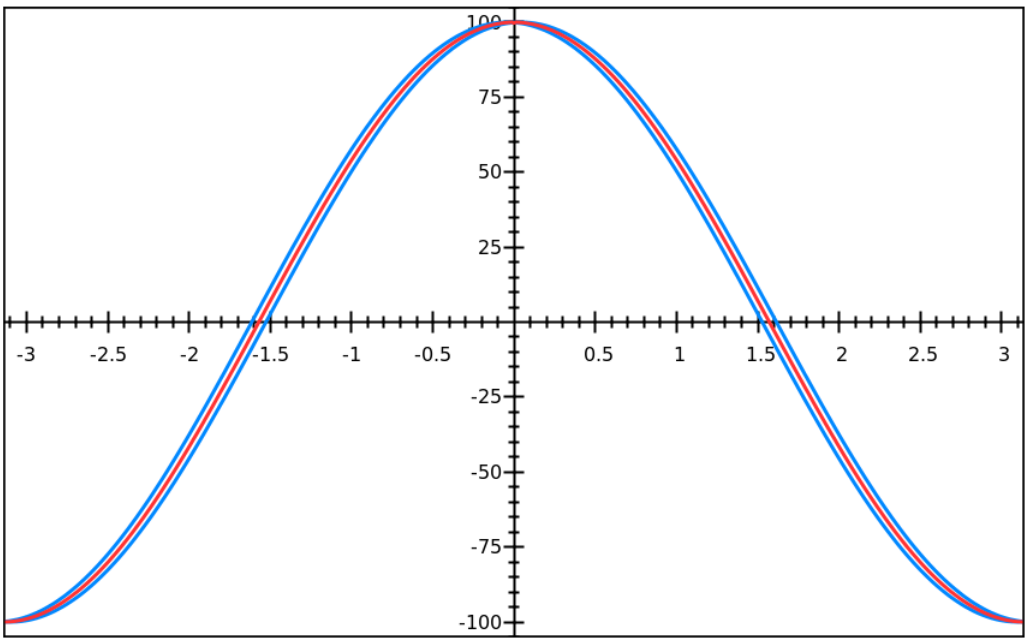

Figura 4.2.6 – Función de rectas de un punto con error en su ángulo.
En la figura se puede ver la función de rectas graficada para los puntos $100e^{\frac{2,5}{180}\pi i}, 100e^{\frac{0}{180}\pi i}, 100e^{\frac{-2,5}{180}\pi i}$. Se puede observar que el ancho de la banda es constante en tita.

Teniendo en cuenta esta diferencia con el error en la distancia, es posible proceder ahora exactamente de la misma forma considerando la distribución del error en tita.

Finalmente, para tener un modelado completo del error, sería necesario integrar ambos errores. Un posible mecanismo para ello sería considerarlos como independientes si bien esto no es cierto. La ventaja de este método sería que es de relativa sencillez implementarlo. Por ejemplo, se podría usar para el ángulo un kernel similar al de la distancia, y luego el peso para un bin en particular podría ser la suma de ambos kernels.

4.2.2.5. Diseñando el algoritmo

Teniendo una idea más clara de cómo calcular la función de densidad para una nube de puntos y conociendo como manejar el error, es posible proceder a extraer los segmentos de la nube. Para ello podría ser posible continuar con el algoritmo de Hough para encontrar las rectas en la nube, sin embargo, el camino tomado diverge un poco de él.

Una vez obtenida la función densidad, es necesario evaluar sus resultados de alguna forma. El algoritmo de Hough lo que propone es utilizar un umbral en la densidad. Todos aquellos bins que sobrepasen dicho valor representaran las rectas obtenidas. De esta forma, se generaran demasiadas rectas, pudiendo cada punto pertenecer a varias de ellas. Este mecanismo no es adecuado ya que, por el contrario, lo que se desea es que cada punto esté asociado a una sola recta (salvo eventualmente si son extremos de un segmento, en cuyo caso pueden pertenecer a dos).

Si bien existen diversos acercamientos al problema, en el proyecto se utilizó el siguiente. Dada la función densidad, se elije el bin de valor máximo. Si son varios, lo que se hace es generar un conjunto de bins recursivamente y luego hallar el bin que represente el centro de masa. Para generar dicho conjunto, se comienza tomando uno de los bins de valor máximo. Luego, en cada recursión, se adjuntan otros bins de valor máximo que sean adyacentes a uno del conjunto. La idea de esto es que si hay varias islas de bins sólo se utiliza una de ellas. Una vez obtenido el bin, este pasará a representar una de las rectas (segmentos) a utilizarse.

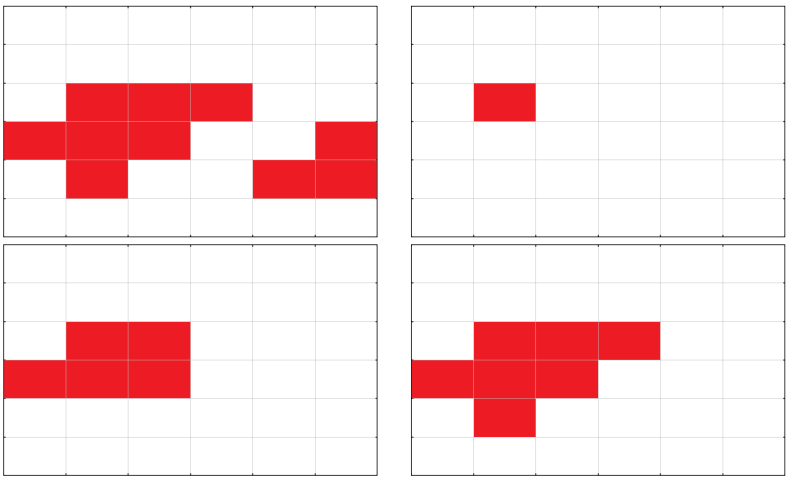

Figura 4.2.7 – Obtención de una isla de bins máximos.
La figura muestra el proceso para quedarse con un solo bloque de bins de valor máximo. La imagen de arriba a la izquierda muestra los bins originales. En la de su derecha se elige un bin. Luego en los siguientes dos pasos (linea inferior), se agregan los adyacentes en cada paso.

A continuación, los puntos cuyas gráficas pasan por el bin obtenido son particionados[6] en componentes conexas, usando los conceptos de la sección 4.2.2.1. Aquella componente que tenga mayor cantidad de índices pasará a formar uno de los segmentos obtenidos. Si x_i y x_j son los extremos del segmento y hay algún punto entre ellos que no pertenezca al segmento, estos son agregados. Con los puntos restantes se vuelva a calcular la función de densidad y se comienza nuevamente[7].

Una vez extraídos todos los segmentos, estos son ordenados de forma antihoraria utilizando el índice de su primer punto. Este paso se hace por comodidad, ya que segmentos consecutivos en una componente conexa han de estar conectados de forma directa, es decir, el último punto de un segmento y el primero del siguiente formarán un camino conexo entre ambos. Si ambos segmentos representan paredes diferentes esto significa que extrapolando sus rectas es posible hallar la esquina entre las paredes (asumiendo sean diferentes, a continuación se habla del tema).

Finalmente, algo a tener en cuenta es que, debido al ruido, a veces los puntos de una misma pared son partidos en varios segmentos. Este efecto no es algo deseado ya que la pared no se representa correctamente. Para intentar prevenirlo, una vez ordenados los segmentos estos son recorridos observando cuanto varía la dirección de segmentos consecutivos, si no varía mucho los segmentos son unidos.

A continuación se presenta el algoritmo completo de extracción de segmentos. A lo anterior se agrega un paso extra para extraer información de conexión será usada luego.

[6]Para no tener que buscar nuevamente los puntos cuyos gráficos perteneces al bin, en el algorítmo del cálculo de la función densidad se puede agregar un array a cada bin que controle dicha información.

[7]Si para cada bin se lleva el array de puntos mencionado en la nota anterior, en vez de reejecutar el algoritmo, a cada bin se le puede remover y descontar los puntos que hayan sido utilizados en el segmento generado.

Algorithm 4 Extracción de características

Entrada: $X = \{\vec{x_1}, ..., \vec{x_n}\}$ nube de puntos.

Salida: $S = S_1, S_2, ..., S_k$ Segmentos de la forma $S_i = \{\vec{x_{a_i}}, ..., \vec{x_{b_i}}\}$.
$M_1, ..., M_k$ Arrays de conexión de la forma $M_i = [m_{i_1}, m_{i_2}]$. m_{i_1} y m_{i_2} son booleanos que indican si el segmento S_i está conectado al S_{i-1} y S_{i+1} respectivamente.

Constantes: T = diferencia de máxima para que dos segmentos consecutivos sean condiderados como uno solo.

Descripción: El algoritmo extrae los segmentos de paredes detectados en la nube. Se intenta que cada pared se corresponda con a lo sumo un segmento. Los segmentos de salida están ordenados de forma antihoraria.

1: Dividir X en $K_1, ..., K_l$ sus componentes conexas ordenadas.
2: **for all** K_i **do**
3: Setear $R = K_i$ puntos restantes para armar segmentos.
4: Setear $L_i = \{\}$ lista vacía de segmentos para K_i.
5: **while** $\#R > 1$ **do**
6: Hallar la función de densidad f para la nube R.
7: Hallar el bin de valor máximo: $bin = maxarg(f)$.[8]
8: Hallar el conjunto de puntos P cuyas gráficas estén en bin.
9: Obtener S la componente conexa de mayor cardinal en P.
10: Agregar S a L_i.
11: Calcular puntos restantes: $R \leftarrow R - S$.
12: Ordenar los segmentos de L_i en sentido antihorario.
13: **for** $j = 1, \#L_i - 1$ **do**
14: **if** $|dir(S_j) - dir(S_{j+1})| < T$ **then**
15: Unir S_j y S_{j+1} reemplazando en L_i adecuadamente.
16: $n = \#S + 1$ índice en S del siguiente elemento a agregar.
17: Agregar los segmentos de L_i a S: $S \leftarrow S + L_i$.
18: Primer y último segmento agregado no están conectados al anterior y siguiente respectivamente: $m_{n_1} = false$, $m_{(\#S)_2} = false$
19: **for all** $n \leq j < \#S$ **do** $m_{j_2} = true$
20: **for all** $n < j \leq \#S$ **do** $m_{j_1} = true$

En el algoritmo anterior, por simplicidad se omitió comentar algunos detalles del mismo. En particular estos tienen que ver con la creación de aristas. Debido a la presencia de ruido, no es deseable construir una recta con muy pocos puntos ya que no es raro encontrarse ocasiones en las que el error de todos los puntos coinciden para hacer aristas con mucho error o inclusive falsas. Particularmente, en el trabajo se pidió que todo segmento creado tenga al menos 4 puntos. A su vez, tampoco es deseable crear segmentos que sean demasiado cortos, es decir todos los puntos observados estén muy próximos entre si. En este caso, la probabilidad de que el segmento esté mal construido es muy alto, en especial en presencia de ruido. Para prevenir esto, se pidió que todo segmento tenga al menos 0.25 metros.

4.2.2.6. Problemas, observaciones y mejoras.

El algoritmo de Houghs, como desarrollado en este proyecto, presenta algunos problemas.

1. El primero de estos problemas es que, debido a que el algoritmo presenta características greedy, las aristas extraídas no representan el mejor conjunto que se podría obtener. Esto se debe a que, en la elección de los segmentos, en cada paso se intenta obtener aquel que tenga mayor cantidad de puntos. De esta forma, los puntos de los bordes,

pertenecientes a paredes adyacentes a los segmentos creados inicialmente, tenderán a incluirse a dichos segmentos. Como resultado, se puede degradar la regresión de los segmentos iniciales y se cuenta con menos puntos para detectar los otros. Para resolver este problema, se podría agregar un paso extra al algoritmo, tal que analice mejor los bordes de los segmentos y tome una mejor decisión.

2. Por el otro lado, si bien el algoritmo intenta obtener el "segmento greedy", no siempre lo logra. El problema se da ya que Hough realmente busca rectas y no segmentos. Cuando existen diversos segmentos orientados en el mismo sentido, todos ellos quedan representados bajo un mismo bin. Para observarlo mejor, supóngase se tiene tres segmentos, dos de los cuales pertenecen a una misma recta con 10 puntos cada uno. En este caso, el bin asociado a estos segmentos será el mismo y tendrá aproximadamente 20 puntos. Si el segmento restante perteneciese a otra recta y tuviese 12 puntos este debería ser el próximo segmento a construirse. De esta forma, como la densidad en el primer bin es mayor a la del segundo, el segmento a crearse será uno de los primeros dos. Este problema se visualiza en la figura 4.2.8.

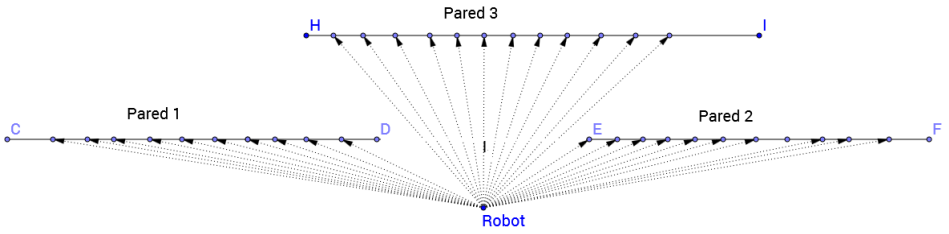

Figura 4.2.8 – Ejemplo del orden en la creación de segmentos.
En la figura se observa al robot, a 3 paredes y a la nube de puntos que observa el robot. La pared 1 y 2 tienen 10 puntos de la nube cada una, la pared 3 tiene 12. Idealmente el algoritmo por ser greedy debería primero crear el segmento con los puntos de la pared 3 y luego los segmentos de las paredes 1 y 2.

3. Por último, el tercer problema detectado con la metodología del algoritmo tiene que ver con su costo computacional. Si se desea tener precisión en las rectas obtenidas es necesario trabajar con una grilla fina de bins. Esto hace que la cantidad de operaciones y la cantidad de memoria requerida aumenten rápidamente con la precisión. A modo de ejemplo, si se tiene paredes que pueden estar a una distancia máxima de 3 metros y se quiere una precisión de 1cm y 1 grado, la grilla generada tendrá aproximadamente 108,000 bins (100*360). Si se quiere una precisión de 1mm y 0.1 grados entonces la grilla tendrá aproximadamente 10,800,000 bins (1000*3600). Si a demás se le suma el hecho de que hay que recalcular los bins luego de obtener cada segmento, si no se tiene cuidado la memoria puede ser agotada, o se puede enlentecer demasiado el algoritmo.

En el proyecto, este último problema se tuvo constantemente presente ya que sus efectos eran fácilmente notorios. Particularmente, esto obligó a cambiar la versión del lenguaje de programación utilizada y a desarrollar varias mejoras ya que, de lo contrario, el debugging consumía demasiado tiempo. Para tener una noción de esto, una iteración de este algoritmo llegó a demorar más de un segundo (ejecutándolo varias veces para cada componente conexa). Considerando que en una ejecución completa del algoritmo de SLAM podía llegar a tener 1000 iteraciones eso equivalía aproximadamente a 17 minutos de ejecución en total(1000 segundos). A continuación se presentan algunas posibles mejoras que pretenden solucionar el problema(algunas de las cuales fueron realizadas en el proyecto):

- **Límite del espacio de búsqueda(θ, d):** Dada una nube de puntos es posible limitar el espacio de búsqueda acotando el máximo valor de d. Para ello, se puede utilizar como máximo la mayor norma de los puntos de la nube. Esto se debe a que si una recta pasa

por un punto x, la distancia del origen a la recta debe ser menor o igual a la distancia del origen al punto x.

- **Binning en dos pasos:** Cuando se halla la función de densidad para una componente conexa por primera vez, a priori no se conoce qué direcciones puedan tener las rectas. De esta forma se hace necesario recorrer todo el espacio de búsqueda en el eje tita. De esta forma una mejora posible es realizando el binning en dos etapas.
  Primero se realiza una vez con una grilla gruesa admitiendo mucho error. El objetivo de esto es encontrar una aproximación inicial a los segmentos. Luego en el segundo paso, se vuelve a realizar el binning pero utilizando una grilla más fina y admitiendo menos error. Este segundo binning no se realiza para la nube de puntos original, sino que se realiza una vez por cada segmento potencial obtenido en la primer iteración. Como para cada segmento se tiene una dirección candidata es posible acotar el espacio de búsqueda en tita.
  Durante el proyecto, se observó que para que esto de buenos resultados, a cada segmento inicial se le debe agregar algunos puntos antes y después de su comienzo y fin. A demás, se encontró que si el origen de coordenadas se traslada al centro de masa de la nube, los resultados obtenidos mejoran [9], y es posible acotar bastante el espacio en d. Esto último se debe a que el centro de masa de un segmento está dentro del mismo segmento, con lo cual, la nueva distancia al origen se vuelve 0 teóricamente.

- **Estructuras de datos y órden operacional:** Como se mencionó antes, fue necesario cambiar la versión del lenguaje de programación debido a los costos tanto computacionales como de memoria. En particular, un factor importante a tener en cuenta es la estructura de datos utilizada. Originalmente se utilizaba Lua. En Lua, todo tipo de datos complejos es representado mediante tablas. Estas son de extrema practicidad y por su generalidad facilitan mucho la programación. El problema es que tienen una huella de memoria relativamente alta y se tiene poco control sobre esto. De esta forma, al utilizar una grilla de aproximadamente 10 millones de elementos, la computadora se queda sin memoria. Observando esto, se decidió cambiar la versión de Lua de la estándar a Lua-jit. Lua-jit tiene un módulo llamado ffi que permite compilar código C y en particular usar sus tipos de estructuras. Al cambiar la versión fue posible implementar los bins como un simple array de floats, reduciendo altamente la memoria y los costos computacionales relacionados al manejo de la estructura.
  Por otro lado, relacionado a esto, para reducir los costos computacionales es importante cuidar el orden de las operaciones. Por ejemplo, si el array de la tabla de bins esta ordenado por filas, es preferible recorrer el array en este sentido que hacerlo por columnas. El objetivo de esto es intentar mantener un alto cache hit ratio para que se pueda aprovechar al máximo las ventajas de la memoria cache.

- **Arquitecturas vectoriales, gpgpu:** Una posible ventaja que puede tener el algoritmo es que es altamente paralelizable. Por ejemplo dado un punto, todo bin podría calcular en paralelo su distancia al punto y luego aumentar su contador en base al valor obtenido. De esta forma, si el algoritmo fuera adaptado para realizar operaciones vectoriales o para trabajar en un gpu, el tiempo de ejecución se vería reducido drásticamente.

4.3. Estructura de los segmentos y el mapa.

En esta sección, en la primer parte se describe la información que debe ser calculada a partir de la extracción de segmentos realizada en el paso anterior. Esta parte se agrega a esta sección ya que, si bien corresponde a la anterior, conceptualmente está más vinculada a la representación de la información y su utilización con el mapa.

Finalmente, en la segunda parte de esta sección se describe cómo se almacenan los datos en el mapa y que estructuras útiles se le agregan al mismo.

[9] Esto se debe a que los bins representan un conjunto finito de rectas y se tiene mayores problemas de presición cuanto más alejada este la recta del origen.

4.3.1. Representación de segmentos.

Una vez obtenida la división de puntos en segmentos se debe decidir de qué forma se deben representarán y que información se debe incluir. La motivación para tomar estas decisiones está condicionada por cómo se manejará luego la información, es decir cómo será utilizada. De esta forma, el objetivo de esta sección es proveer una estructura de datos para los segmentos que sea de utilidad y practicidad para las subsiguientes secciones. Todas las estructuras presentadas en esta sección son compatibles con la estructura del mapa. Es decir, el mapa se construirá con las mismas estructuras.

4.3.1.1. Nomenclatura

A partir de ahora en más, se comenzará a hablar de nodos, vértices y aristas izquierdas y derechas. Este vocabulario tomará sentido a partir de la orientación antihoraria impuesta en los objetos. En general, el término izquierdo se asociará con la idea de que un objeto se encuentra antes que otro en este orden, ídem para el termino derecha. Por ejemplo dado un segmento, el primer vértice dado por el orden será llamado vértice izquierdo, mientras que el segundo será el vértice derecho.

De igual forma, los términos siguiente y anterior serán utilizados para representar lo mismo. En este caso, dado un segmento se hablará de su segmento siguiente y anterior.

4.3.1.2. De nube de puntos a segmentos.

Antes de comenzar a hablar de la representación de los segmentos, se ha de tener en cuenta que el algoritmo de extracción de características representa cada segmento como un conjunto ordenado de puntos. Sin embargo, estos puntos tienen ruido con lo cual el segmento no está bien definido. Para definirlo es necesario definir sus dos vértices.

Para realizar esto, el primer paso es realizar algún tipo de regresión lineal sobre los puntos. A priori cualquier regresión es adecuada, sin embargo, en el proyecto, los mejores resultados se obtuvieron al aplicar el algoritmo de hough nuevamente. En este caso, el algoritmo se termina al encontrar el mejor bin ya que esta representará la recta buscada y no se busca dividir una nube de puntos. Observar que en los casos que no fue necesario unir segmentos en el paso 15 del algoritmo de extracción de características, la recta que se obtiene es la misma. Por este motivo, si esta información es almacenada en la extracción de características, no es necesario recalcularla.

Obtenidas las regresiones, el siguiente paso es definir los vértices de los segmentos. Para los segmentos que estén conectados a otro, uno de sus extremos puede ser definido como la intersección de las rectas dadas por ambos. Para los que no tengan conexiones, sus extremos pueden ser definidos proyectando el primer y/o último punto de la nube del segmento. Esta proyección puede ser realizada de dos formas. Una es de forma perpendicular, la otra es proyectando en dirección al origen de coordenadas. La motivación de la segunda se debe a que el extremo del segmento debería representar el punto intersección del rayo láser con la pared. La figura 4.3.1 ilustra las dos formas de proyectar los extremos.

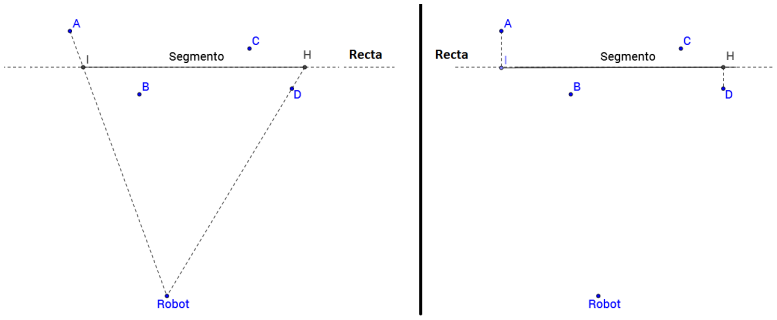

Figura 4.3.1 – Formas de proyectar un punto sobre una recta.
En la figura se ve el robot y a los cuatro puntos (A, B, C, D) que forman su nube de puntos. La recta punteada horizontalmente representa la obtenida por la regresión. En la imagen de la derecha los extremos del segmento son obtenidos proyectando perpendicularmente los puntos A y B. En la de la izquierda, los extremos son obtenidos proyectando los mismos puntos pero proyectando en la dirección del robot.

4.3.1.3. Representación segmentos.

Para representar los segmentos, se utiliza dos formas. Ambas son equivalentes, por lo que si se modifica una de las representaciones se deberá tener la consistencia de modificar la otra. Se almacena ambas representaciones para evitar tener que estar recalculando los valores continuamente (ya que ambas se usan frecuentemente).

Una de las dos representaciones es una de las formas más comunes. Se mantiene una lista de puntos y cada segmento es un puntero a dos puntos.

La otra representación utilizada, es la representación polar de segmentos que se presenta en detalle el apéndice matemático. La misma consiste de una 4-upla de la forma (θ, y, x_1, x_2) donde θ es la dirección del segmento e y, x_1, x_2 son las coordenadas de los vértices en la base $\{e^{i\theta}, e^{i(\theta+\frac{\pi}{2})}\}$. En particular, la coordenada x_1 pertenecerá al primer vértice izquierdo y la coordenada x_2 al derecho. Como se veía en el marco teórico, la coordenada y es compartida por ambos vértices.

4.3.1.4. Esquinas

Dado un vértice de un segmento, algo que será de interés conocer es cuándo el vértice representa una esquina real. Para ello se distinguen dos casos.

El primer caso, el más sencillo es cuando el vértice es intersección de otros dos segmentos. En este caso el vértice es una esquina.

El caso más complicado se da cuando el vértice no es intersección de dos segmentos. En este caso, como lo que se observa de la pared es un conjunto discreto de puntos, no se tiene información suficiente para indicar donde es el fin de la misma (salvo se deduzca a partir de un historial de observaciones). No obstante, en algunos casos se podrá dar algo de información extra respecto a la esquina. Para ver esto considérese el siguiente ejemplo.

Sea S un segmento formado por los vértices ordenados V_1, V_2, y sea x_n el último punto de la nube del segmento. Supongase además que no está conectado al siguiente. Si la norma del punto x_{n+1} es mayor que la de V_2, mediante trigonometría es posible decir a qué distancia máxima se encuentra la esquina del segmento. Para ello basta prolongar el segmento e intersectarlo con la recta que pasa por el origen y el punto x_{n+1}. El segmento no puede ser más largo que eso ya que de lo contrario el punto x_{n+1} no sería visible (porque el laser rebotaría en el segmento y no en ese punto). La figura 4.3.2 muestra mejor este hecho.

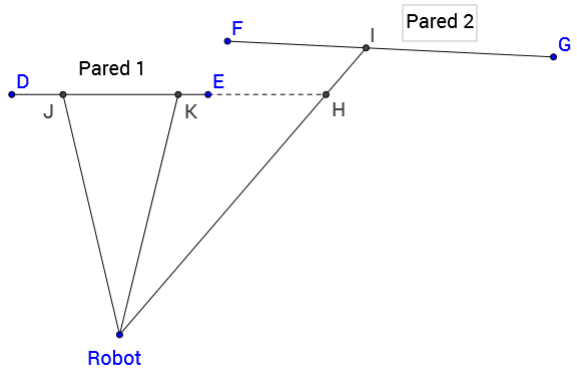

Figura 4.3.2 – Máximo error en las esquinas.
La figura muestra al robot junto a dos paredes y tres haces láser. El primer y segundo haz intersectan la primer pared en los puntos J y K. El tercer haz intersecta la segunda pared en el punto I. El punto H muestra donde intersectaría el tercer haz si la primer pared fuese lo suficientemente larga como para intersectarlo. De esta forma,se observa que el punto K (vértice de la pared uno) debe estar contenido en el segmento KH.

A partir de esta idea, si la máxima distancia a la esquina es chica, se puede considerar que el vértice V_2 es la esquina teniendo en cuenta la cota del error.

4.3.2. Estructura del mapa.

El mapa se estructura de forma casi idéntica a los segmentos. En si, el mapa es una recopilación de los segmentos observados en diferentes iteraciones, al cual, por cuestiones de facilitar su uso, se le agrega algunas estructuras extra. Los detalles son explicados a continuación.

4.3.2.1. Representación de segmentos en el mapa.

La representación de los segmentos en el mapa y la vista en la sección 4.2 difieren en dos aspectos. El primero, es que, por facilidad de uso, a cada arista y a cada punto se los coloca dentro de un array y se les asigna un número identificador que corresponde a su posición en el array.

4.3.2.2. Hash de aristas

El hash de aristas es un hash que organiza todas las aristas del mapa según su ángulo. El objetivo de este es que dado un ángulo se pueda buscar rápidamente todas las aristas cuya dirección coincida con él. Esto será de especial utilidad a la hora de realizar la identificación de segmentos ya que dado un segmento no será necesario compararlo contra todos los del mapa. En el proyecto, el hash cuenta con 360 índices, uno para cada grado. Cada posición es un array (utilizado en forma de conjunto) que alberga a todos los segmentos con la misma entrada en el hash sin tener consideraciones extra en el orden. Observar que si una arista cambia de dirección es necesario cambiarla de índice en el hash.

4.3.2.3. Array de memoria

El array de memoria simplemente es un array que guarda información de los segmentos observados en la iteración anterior. Cada segmento tiene una entrada en el array en la que se guarda un puntero al mismo y el ángulo de sus puntos inicial y final como fueron observados (es decir, relativos a la dirección de la cabeza del robot).

Como se verá en la siguiente sección, el objetivo de guardar este array es proveer un mecanismo sencillo para la orientación inicial.

4.4. Identificación de segmentos, orientación y ubicación .

Esta sección explica cómo un conjunto de segmentos extraídos del sensado son identificados (reconocidos) en el mapa. Es decir, dado un segmento extraído en la iteración actual, se ha de evaluar si se corresponde con uno observado previamente o no. En el caso afirmativo, se debe decir cuál es entre todos los segmentos (paredes) registrados en el mapa. Por otro lado, dado que está relacionado al reconocimiento, en esta sección también se explica cómo se realiza la orientación y la ubicación.

Si bien en la sección 4.1 se muestra que en el algoritmo de SLAM diseñado primero se identifican los segmentos y luego se realiza la ubicación/orientación, en realidad, estos pasos se ven intercalados de la siguiente forma:

1. **Identificación inicial.**
2. **Orientación inicial.**
3. **Identificación general -primera iteración.**
4. **Orientación.**
5. **Posicionamiento.**
6. **Identificación general -segunda iteración.**

El primer paso permite identificar segmentos extraídos en la iteración actual comparándolos contra los almacenados en el array de memoria. El objetivo es permitirle al robot orientarse de forma "rápida" (segundo paso del algoritmo), para así disminuir los errores que se puedan cometer en el paso 3. En dicho paso, se continúa identificando segmentos pero esta vez utilizando el hash de aristas.

En el paso 4, considerando los nuevos segmentos identificados, se recalcula la dirección de la cabeza con el objetivo de tener una mejor estimación, y luego, en el paso 5, se calcula la posición.

Finalmente, en el paso 6, se repite el paso 3 con la esperanza de que al mejorar la estimación, sea posible identificar más segmentos. Si bien podría ser buena idea iterar los pasos 3 al 5 hasta que el conjunto de segmentos reconocidos no varie de un paso al otro, no se encontró necesario agregar esta complejidad extra.

4.4.1. Identificación Inicial

La identificación inicial de segmentos es un paso diseñado para permitir al robot orientarse utilizando el array de memoria. La idea es que los segmentos observados en la iteración actual y la anterior han de tener un alto índice de coincidencia (asumiendo se itera cada poco tiempo). Es decir, si el robot se movió poco, las paredes que observa deben ser esencialmente las mismas.

Este paso lo que hace es crear una tabla de identificación, donde a cada arista observada se le asocia un puntero que indica con que arista del mapa se corresponde. Si no se corresponde con ninguna se le asigna un puntero nulo. Para ello, dada una arista de la iteración actual, se buscan todas las aristas en el array de memoria cuya dirección difiera de la dirección de la arista actual en no más de una constante.

Antes de continuar con el algoritmo, observar que para poder comparar las direcciones, tanto las aristas observadas como las del hash han de tener el mismo marco de referencia. De esta forma, como el ángulo de las aristas observadas es relativo a la dirección de la cabeza, para compararlas contra las del mapa, es necesario contar con una estimación de este valor. Como se mencionó anteriormente, en la desarrollo inicial de este proyecto se asumía el

algoritmo ejecutaría varias veces por segundo, con lo cual, el robot no podría moverse mucho entre iteraciones. De esta forma, se utilizaba la dirección de la cabeza en el tiempo anterior para estimar la actual, en cuyo caso la constante del algoritmo indicaba el máximo ángulo que el robot podía girar entre iteraciones. No obstante, como la hipótesis no resultó ser cierta, fue necesario incluir PI. De esta forma la constante pasó a ser una cota del error cometido en la integración.

Siguiendo con el algoritmo, una vez obtenido el conjunto de segmentos con dirección similar, si dicho conjunto es vacío, a la arista actual se le asigna un puntero nulo. Si el conjunto tiene un solo elemento, se asignara el puntero a dicho segmento. En otro caso, el puntero será no nulo, pero se debe contar con un mecanismo de decisión. Para ello, utilizando la información guardada en el array de memoria, para cada arista del conjunto obtenido, se calcula el ángulo coincidente entre ella y la observada. Si el ángulo de coincidencia es 0, entonces se considerara el ángulo de distancia entre ambos segmentos. El puntero elegido será el asociado al segmento de mayor coincidencia o al de menor distancia (si la mayor coincidencia es 0). Para explicar mejor esto considérese la figura 4.4.1.

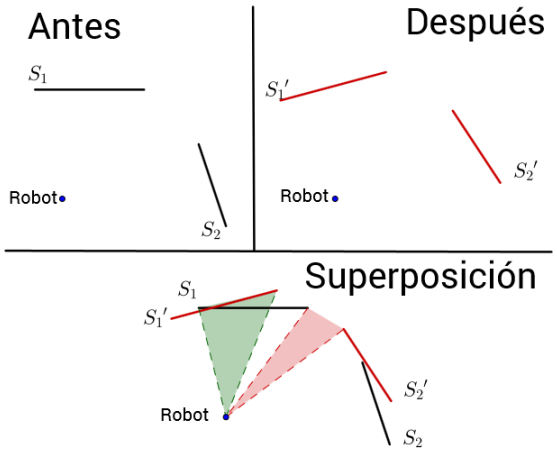

Figura 4.4.1 – Ángulo de coincidencia y ángulo de distancia.

En la figura, en la fila superior se muestra las paredes que observa el robot antes y después de girar θ grados. En la fila inferior se superponen ambas observaciones. El área verde indica el ángulo de coincidencia entre el segmento 1 antes y después del giro (S_1 y S_1'). Como el ángulo de coincidencia entre el segmento 2 después del giro y el segmento 1 antes del giro (S_2' y S_1) es de 0 grados, el área roja representa el ángulo de distancia entre los segmentos.

4.4.2. Orientación y posicionamiento

En el algoritmo desarrollado, tanto la posición como la orientación se calculan utilizando ideas similares. Para realizar el cálculo, ambos utilizan un conjunto de características identificadas en el mapa. A partir de ellas, los resultados se deducen considerando la diferencia entre posiciones/orientaciones reales de las características y las observadas.

Como el posicionamiento requerirá haber calculado previamente la dirección de la cabeza, se comenzará analizando esto último primero. A su vez, por simplicidad, se partirá explicando como realizar el cálculo utilizando un solo segmento y luego se explicará la idea para segmentos múltiples.

4.4.2.1. Orientación con un solo segmento

Considérese que en un determinado momento se tiene un segmento observado O que ha sido identificado como el segmento S del mapa. Tanto el segmento O como S tendrán una

orientación θ_O y θ_S respectivamente. Observar que O haya sido identificado como S significa que son la misma pared. De esta forma, θ_S es la orientación en el mapa y θ_O es la orientación del segmento relativa a la dirección de la cabeza. Luego, si HD es la dirección de la cabeza se tendrá que $\theta_S = \theta_O + HD$. Despejando, de esta ecuación resulta:

$$HD = \theta_S - \theta_O{}^{10}$$

La figura 4.4.2 ilustra el resultado.

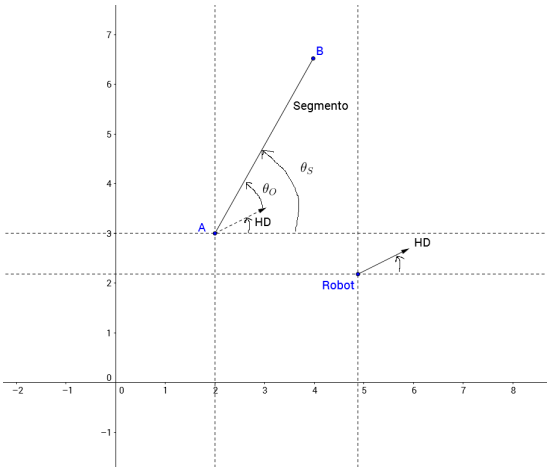

Figura 4.4.2 – Orientación con un solo segmento.
En la figura se observa a al robot y a una pared (segmento) posicionados dentro de un mapa. El vector HD indica la orientación del robot. La dirección del segmento en coordenadas locales y globales esta dado por los ángulos θ_O, θ_S respectivamente. La relación entre ambas esta dada por la ecuación $\theta_S = HD + \theta_O$.

4.4.2.2. Orientación usando múltiples segmentos

Supongase ahora que los segmentos observados identificados son $O_1, ..., O_n$. Si $S_1, ..., S_n$ son sus equivalentes en el mapa, utilizando la ecuación $HD = \theta_S - \theta_O$, es posible obtener un HD_i para todo par O_i, S_i. De esta forma, cada segmento sugiere una dirección de la cabeza diferente, y el nuevo problema consiste en obtener una sola medida a partir ellas. Para resolver el problema se puede proceder de varias maneras como por ejemplo tomando promedios o medianas[11].

En el caso particular de este proyecto los mejores resultados se obtuvieron considerando el promedio ponderado, eliminando los valores extremos.

Como ponderador de cada HD_i se utilizó a su valor correspondiente $min(longitud(O_i), longitud(S_i))$. La idea, si bien no del todo correcta, es que cuanto mayor sea la longitud del segmento, mejor será la estimación de su dirección ya que, el error en su regresión, tenderá a ser menor.

Para eliminar valores extremos se hace uso la mediana y un radio. Únicamente se promedian los HD_i tal que $d(HD_i, Mediana(HD_i))| < R$ en donde d es la función distancia entre ángulos[12]. En el proyecto el radio utilizado R es una constante y equivale a 5 grados.

4.4.2.3. Posicionamiento

Una vez hallada la dirección de la cabeza, siguiendo procesos similares, es posible hayar la posición del robot en el mapa. Para ello, en vez de considerar segmentos se considerarán puntos de referencia.

[10] Observar que la igualdad es entre ángulos con lo cual $0 = 2\pi$.
[11] Observar que los datos promediados son ángulos, con lo cual se debe tener en cuenta su caracter modular a la hora de hacer los cálculos.
[12] Observar que la distancia entre ángulos está dada por la siguiente función: $d(\theta_1, \theta_2) = min(|\theta_1 - \theta_2|, 2\pi - |\theta_1 - \theta_2|)$.

Antes de comentar cuales son los puntos de referencia y sus especificidades, primero se procederá a comentar como se utilizan. Para ello, supongase que X es el punto de referencia observado y P es su correspondiente en el mapa. De forma similar a la sección $4.4.2.1$, la posición del robot en el mapa queda dada por la siguiente ecuación:

$$Pos = P - X$$

Observar que en la ecuación anterior, el vector X debe estar rotado conforme a la dirección de la cabeza. Por ejemplo, supongase que el ángulo de la dirección de la cabeza es de 45 grados. Si el punto referencia se observa en la posición $(1, 0)$ relativa a las coordenadas y orientación del robot, antes de utilizarlo en la ecuación anterior anterior, el punto debe ser rotados 45 grados. De esta forma el vector a utilizar será el $(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2})$.

En la figura 4.4.3 se ilustra la ecuación.

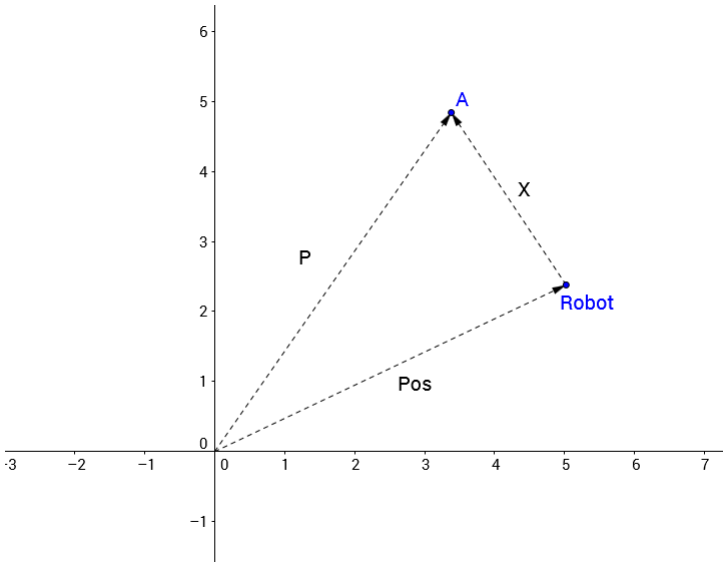

Figura 4.4.3 – Posicionamiento con un punto.
En la figura se observa al robot y a un punto de referencia A ubicados en el mapa. Las coordenadas locales y globales de A están dadas por los vectores X y P respectivamente. La relación entre ellos está dada por la ecuación $P = Pos + X$.

Cuando se tienen múltiples puntos de referencia los métodos para obtener la posición son equivalentes a los vistos en la sección $4.4.2.2$. En este caso, en el proyecto simplemente se utilizó el promedio de los posiciones obtenidas por cada punto de referencia.

Finalmente, para que esta sección este completa, es necesario decir como obtener los puntos de referencia. Para ello se utilizan dos ideas. Una, como se mencionó anteriormente, es intersectar rectas no paralelas. Para ello, cada par de segmentos identificados no paralelos generará un punto de referencia[13]. Originalmente, las coordenadas locales de estos puntos eran calculadas como la intersección de las rectas obtenidas por la prolongación de los segmentos observados. A su vez las coordenadas en el mapa estaban dadas prolongando los segmentos correspondientes en el mapa. El problema de esto es que pequeñas variaciones en los ángulos de los segmentos puede cambiar considerablemente los puntos obtenidos, en especial si la intersección de los segmentos se da en un punto muy alejado. Para solucionar este problema, se decidió cambiar la forma en que se calcula las coordenadas locales. Para ello lo que se hizo fue intersectar las rectas que pasen por los centros de masa de la nube de puntos de los segmentos observados con la dirección dada por los segmentos del mapa. La figura 4.4.4 ilustra mejor estos cálculos.

[13]Debido a la presencia de errores, dos segmentos son considerados como no paralelos si el ángulo entre ellos es mayor que una constante. El valor utilizado de dicha constante es de 20 grados.

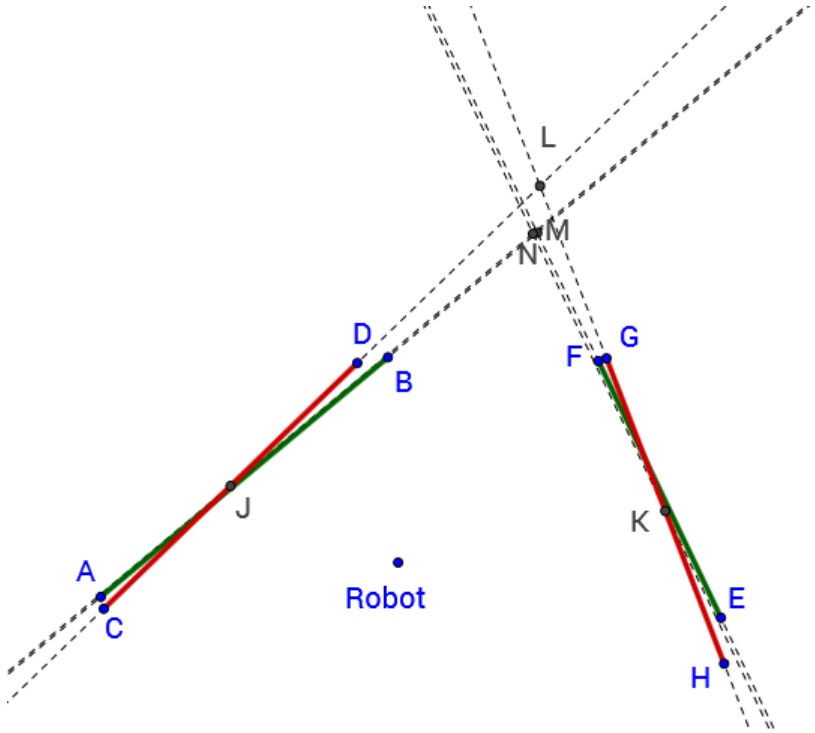

Figura 4.4.4 – Dos formas de posicionamiento.
En la figura se observa al robot en una posición del mapa junto a dos paredes observadas. Los segmentos rojos representan a las paredes como observadas en la iteración actual (resultado de la regresión lineal). Los verdes representan a las paredes según la información almacenada en el mapa. Los puntos J y K son los centros de masa de los segmentos observados. Los puntos M y L surgen de prolongar e intersectar los segmentos verdes y rojos respectivamente. El punto N surge de intersectar la recta paralela a AB que pasa por J, con la recta paralela a EF que pasa por K.Observar que La diferencia entre los puntos N y M es mucho menor que la diferencia entre M y L.

La otra idea utilizada para obtener puntos de referencia se aplica cuando no es posible generar puntos mediante la primer forma. Es decir cuando no existen al menos dos rectas identificadas no paralelas. En estos casos se toma como referencia a los vértices de los segmentos que sean considerados como esquinas. El problema es que como dichas esquinas realmente solo son una aproximación, estas tiendan a tener mayor error. Por este motivo, estos puntos de referencia solo son usados como un respaldo cuando no se tienen puntos mediante el otro método.

4.4.3. Identificación General

Nuevamente, así como en la identificación inicial de segmentos, el propósito de la identificación general es producir una tabla de correspondencia entre los segmentos observados y los segmentos del mapa.

Para realizar la identificación en este paso, dado un segmento se utiliza el hash de aristas para buscar los segmentos del mapa con direcciones similares y así restringir el dominio de búsqueda. Para intentar encontrar el segmento asociado de forma rápida, primero se comienza buscando en la entrada del hash correspondiente ángulo del segmento en cuestión. Si el segmento no se identifica dentro de ese conjunto, se procede a buscar en los adyacentes y así sucesivamente hasta alcanzar un radio de búsqueda máximo[14]. En cuyo caso, se detiene la búsqueda asumiendo el segmento observado no se corresponde con ninguno del mapa y asociándole un puntero nulo en la tabla de correspondencia.

[14]En este trabajo el radio es de 40 grados

Para decidir si el segmento observado se corresponde con un segmento del hash siendo revisado se piden dos condiciones. La primera es que la distancia de los vértices del segmento observada a la recta dada por el otro no sea mayor que una constante. Esto en particular implica que quede contenido en una banda de espesor fijo que tenga por centro a la recta. La segunda condición pide que al menos una de las proyecciones de los vértices sobre la recta quede contenida o próxima al segmento del hash, o en su defecto, que el segmento dado por la proyección de ambos vértices contenga al del hash.

Estas condiciones se observan en la figura 4.4.5.

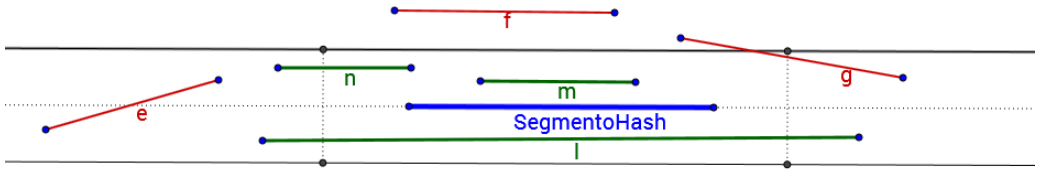

Figura 4.4.5 – Condiciones para la identificación general de segmentos.
En la figura se ejemplifica las condiciones que se piden para que un segmento observado sea identificado con uno del hash. Las rectas superior e inferior indican la tolerancia de la primer condición. Los segmentos punteados verticales indican la tolerancia de la segunda. Para pasar la primer condición un segmento debe tener sus dos vértices dentro de la banda (segmentos e, l, m, n). Para además pasar la segunda, al menos uno de los vértices debe quedar en el cuadrilátero delimitado por las rectas y los segmentos (segmentos n, m), o sino, la proyección sobre la recta debe contener a SegmentoHash (segmento l).

Una vez que una arista es identificada, se utiliza su información de conexión para identificar la mayor cantidad de segmentos posibles. Para ello, si tanto el segmento observado como su correspondiente en el mapa tienen un segmento siguiente, entonces se asume que estos son el mismo. Si ambos tienen un segmento anterior, se procede de igual forma.

4.5. Integración de la información

Una vez realizada la identificación de segmentos, posicionamiento y orientación se procede a integrar la nueva información al mapa. Las aristas y puntos que no fueron identificados simplemente son agregados al mapa y la información de los que si fueron debe ser actualizada. La información a actualizarse puede separarse en 4 categorías. La primera es información de conexión del segmento, la segunda es orientación y posición de la recta a la que pertenece el segmento, la tercera es bordes del segmento y la cuarta es información dependiente.

4.5.1. Información de conexión

El primer tipo de información a actualizar es la de conexión. Esta información esta embebida tanto en segmentos como en vértices. Los vértices almacenan punteros a sus segmentos izquierdo y derecho, mientras que los segmentos almacenan punteros a su sucesor y antecesor. Esta información se ve modificada únicamente cuando se agregan conexiones. Esto puede deberse a dos motivos. El primero es que se haya encontrado un nuevo segmento que sea sucesor o predecesor de otro previamente registrado en el mapa. En este caso la información correspondiente debe ser actualizada sin más consideraciones.

El segundo motivo, puede deberse a que, dos segmentos que previamente se desconocía estuvieran unidos pasen a estarlo. Este caso es un poco más complicado ya que dos puntos que solían ser considerados como diferentes pasan a ser el mismo punto. Para ello se debe resolver el alias eliminando a uno de los puntos y uniendo la información de ambos.de forma tal que no queden incoherencias.

4.5.2. Información de rectas

En este trabajo, para actualizar un segmento lo que se hace es llevar un control del centro de masa de los puntos observados del segmento, de su dirección y de la cantidad de puntos que han sido observados del mismo. Cuando se tiene una nueva observación de dicho segmento el centro de masa y la dirección son actualizados mediante una suma ponderada usando como peso a la cantidad de puntos. Hecho esto se procede sumar los pesos y a almacenar el nuevo valor.

La idea detrás de esto es llevar un punto de control el cual sea altamente probable que pertenezca al segmento. Dicho punto es el centro de masa del segmento.

4.5.3. Información de bordes

Una vez que se actualiza la recta a la que pertenece el segmento se procede a actualizar los vértices. Estos son actualizados de dos maneras acorde al tipo. Si un vértice es la intersección de dos segmentos se deberá recalcular la intersección. En caso contrario, nuevamente se se dividen en dos tipos. Vértices que representan esquinas y vértices que no. En ambos casos el proceso es casi igual. Se comienza proyectando el vértice viejo y el de la nueva observación sobre la recta recién obtenida. Luego, para los vértices que no son esquinas se elije la proyección que agrande más al segmento. Para los vértices que si lo son, se hace lo mismo pero únicamente si el error de considerar al vértice como esquina se achica. Si lo hace, entonces se modifica tanto el vértice como este valor.

4.5.4. Información dependiente

La información dependiente es toda la información que debe ser actualizada como consecuencia de los pasos anteriores para mantener una estructura consistente de datos. A modo de ejemplo, si se modifico la dirección del segmento es probable que sea necesario cambiar su ubicación en el hash de aristas del mapa. Otros valores a modificar incluyen el vector perpendicular del segmento, su coordenada y, etc.

Capítulo 5

Algoritmo de estados convexos.

En este capítulo primero se habla de la motivación detrás del algoritmo y luego se procede a comentar su implementación. Se ha de tener en cuenta que el módulo desarrollado es simplemente una prueba de concepto para mostrar cómo se podría usar el mapa desarrollado en SLAM. También se ha de tener en cuenta que esta sección se basa en definiciones y demostraciones matemáticas que son adjuntadas en un apéndice.

5.1. Motivación

En las etapas iniciales de la fase de implementación del proyecto, como se mencionó anteriormente, se pretendía implementar el modelo TAM-WG utilizando como referencia los artículos [2, 23]. En dicho artículo se menciona que los estados en el WG representan lugares de interés en el mapa. Estos pueden incluir lugares donde se aplique refuerzo (en el algoritmo de RL) o que requieran un cambio de decisión en la planificación actual de movimiento. Por otra parte, los estados en el TAM representan situaciones de interés determinadas principalmente por las capacidades de movimiento. Estas situaciones son independientemente del lugar en donde surjan. A modo de ejemplo, una situación podría ser estar parado en un pasillo del laberinto de brazos radiales mirando hacia el centro. Esta situación representará el mismo estado independientemente del brazo en que se esté.

En base a lo anterior, si se fuese a implementar el modelo TAM-WG, sería necesario definir mejor lo que es un lugar o situación de interés en los modelos WG y TAM respectivamente. Más puntualmente, se debería contar con dos mecanismos. Uno de ellos debería distinguir las capacidades de movimiento que tenga el agente (por ejemplo, en un pasillo solo se puede ir hacia adelante o atrás) y proveer una forma de comparar dichas capacidades. El otro mecanismo debería permitir distinguir las regiones del espacio en las que se deba cambiar la planificación (por ejemplo, al llegar al final de un pasillo). Observar que esto último es equivalente a encontrar las regiones donde "las capacidades de movimiento cambien". El problema de esto es que el cambio no se da de forma discreta sino continua. A modo de ejemplo, cuando se está en la mitad del "pasillo vertical" del laberinto tipo T, solo se puede moverse hacia adelante o hacia atrás. Sin embargo, a medida que se llega a la intersección de la T se va pudiendo moverse en direcciones cada vez más y más oblicuas hacia los costados, hasta que finalmente es posible doblar en 90 grados.

Luego de analizar el problema, se sugirió que encontrar regiones convexas del espacio delimitadas por las paredes podría resolver gran parte del problema de forma natural. En el WG, las aristas de las regiones indicarían los lugares donde es necesario cambiar la dirección de movimiento. Esto se debe a que, para moverse de un punto de la región a otro, como esta es convexa, no hace falta cambiar la dirección de movimiento una vez que haya sido seleccionada. Si estas regiones se representan en un mapa, el resultado es equivalente a los Meadow Maps vistos en el marco teórico. Por otra lado, en el TAM, si se obtiene el polígono de la región convexa en la que se encuentra el robot, es posible definir las capacidades de movi-

miento utilizando las aristas del polígono. Estas estarán dadas por el conjunto de direcciones hacia las aristas que no representen paredes. A dichas aristas se les denominará portales. Nuevamente, observar que esta definición no está completa. Como un portal es un segmento y no un punto, no queda claro cuál es la dirección hacia el portal. Algunas opciones para ello podrían incluir a la dirección perpendicular al segmento, o la dirección hacia su centro. La figura 5.1.1 ilustra mejor el concepto de los portales. Observar que el resultado no es exactamente el mismo que el descrito en el artículo de referencia.

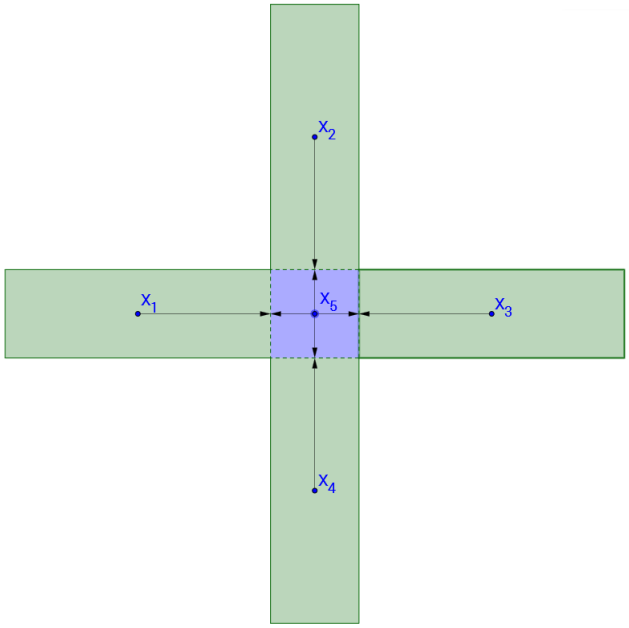

Figura 5.1.1 – Uso de polígonos para la discretización de movimientos posibles.
En la figura se muestra un laberinto de 4 brazos radiales dividido en regiones convexas. Los segmentos punteados representan los portales. Los vectores desde las posiciones X_1, X_2, X_3, X_4, X_5 representan las capacidades de movimiento en dichas posiciones.

5.2. Problema a resolver pTAM y pWG

El problema a resolver puede dividirse en dos instancias diferentes. Una consiste en hallar el polígono convexo que represente la región en la que se encuentre el robot. Para ello se utiliza únicamente la información sensorial. El objetivo final de esta instancia sería proveer el polígono convexo al modelo TAM (que no utiliza ni mapas ni información de la posición), para luego poder ser usado en la distinción de movimientos posibles.

La otra instancia del problema también consiste en obtener un polígono convexo que represente la región donde se encuentre el robot. Sin embargo, en este caso, para ello también se utiliza la información del mapa, el cual se pretende particionar en polígonos convexos. Una vez que un área esté particionada se espera que los polígonos obtenidos en iteraciones subsiguientes sean alguno de los registrados en el mapa. El objetivo final sería poder usar estos polígonos en la determinación de los estados en el WG. Para ello, los estados podrían definirse como los polígonos en sí mismos (regiones donde "las decisiones son constantes") o como las aristas que unan polígonos (regiones donde se debe tomar una decisión).

Por cuestiones de comodidad, a ambos problemas se los denominará como pTAM (polígonos TAM) y pWG (polígonos WG) respectivamente.

5.2.1. Dificultades

Tanto el problema pTAM como el pWG presentan al menos las siguientes dificultades:

- **Información parcial**

La primera dificultad es que para armar los polígonos no se cuenta con información completa del entorno. Solo se cuenta con una "visión local" del mismo representada por una nube discreta de puntos, o, eventualmente, por las aristas extraídas a partir de ella. En el caso del pWG se cuenta además con la información del mapa, pero esta no estará completa hasta una vez finalizado el mapa.

- **Ruido en los datos**

La segunda dificultad es que, no solo la información es incompleta sino que además posee ruido. Esto en particular puede implicar en que se obtengan aristas falsas o torcidas cosa que afectaría la creación y, en el caso del pWG, la manutención de los polígonos.

- **Múltiples formas de dividir el espacio**

La tercera dificultad plantea la siguiente incógnita: ¿Cómo dividir el espacio? El problema es que dado un espacio (o una observación del mismo), es posible dividirlo de varias formas en regiones convexas. Tomando como ejemplo a un cuadrado, este puede ser dividido en 3 formas posibles mediante sus vértices. Una de ellas es considerando al cuadrado entero como una única división (ya que un cuadrado es convexo). Las otras dos es dividiendo al cuadrado en dos triángulos por medio de una sus dos diagonales. En este caso trivial, sería deseable que el cuadrado no sea dividido ya que, tener más polígonos, significa tener que tomar más decisiones durante la navegación.

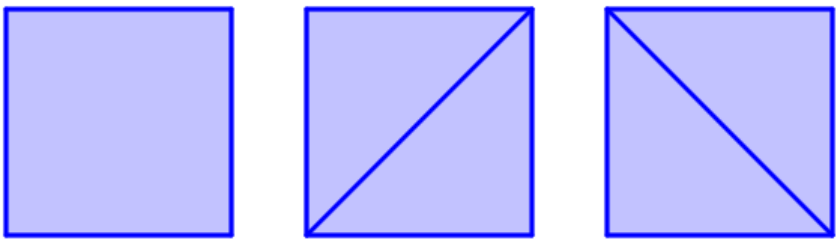

Figura 5.2.1 – Formas de dividir un cuadrado en convexos utilizando sus vértices.

En otros casos, este problema no es tan trivial . Por ejemplo considérese el polígono de la siguiente figura con forma similar a una "L". Asumiendo que se deseara dividirlo usando la menor cantidad de polígonos posibles, existe exactamente cuatro formas de hacerlo (uniendo el punto X a alguno de los Y). A simple vista, ninguna parece tener mejores propiedades que las otras.

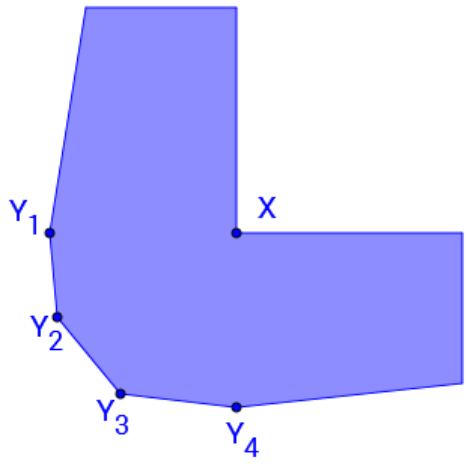

Figura 5.2.2 – Polígono similar a una "L" con un solo vértice convexo.

La existencia de varias formas de dividir el espacio, hace necesario elegir una. Más específicamente, es necesario diseñar un mecanismo de decisión. De este mecanismo, en particular, se desea que el resultado sea invariante de la orientación del robot. Además, idealmente, si en un determinado punto el resultado del algoritmo es el polígono P, sería deseable que al ejecutar el algoritmo desde cualquier otro punto de P el resultado sea el mismo. La idea de estas dos condiciones es la siguiente. Imagínese que el robot se encuentra en un pasillo que conecta dos habitaciones cuadradas como se ve en la figura 5.2.3.

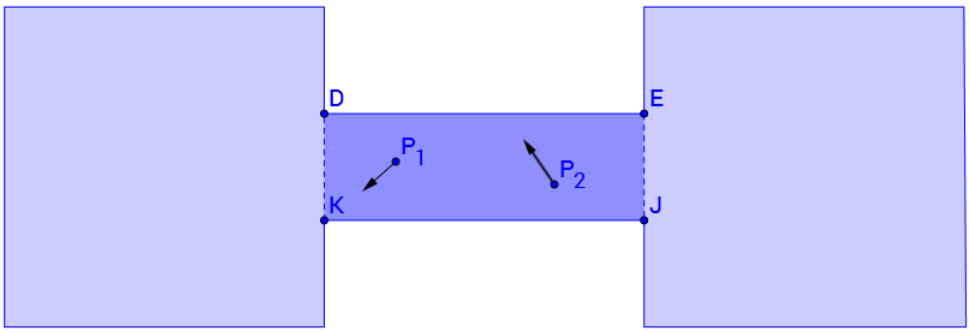

Figura 5.2.3 – Propiedades deseables de los convexos a extraer.
En la figura se muestra al robot en dos posiciones (P_1 y P_2) de un pasillo que conecta dos habitaciones cuadradas. El área sombreada oscura representa el polígono convexo que se desearía obtener desde dichas posiciones.

Independientemente de la posición u orientación en que se encuentre el robot en el pasillo, el polígono resultante debería ser el mismo (el polígono que represente al pasillo).

5.3. Algoritmo implementado

Para resolver las dos instancias del problema, la solución propuesta primero intenta resolver el problema pTAM, y luego, utilizando su salida junto a la información del mapa se pretende resolver el pWG. Para ello, se hace fuerte uso de las herramientas matemáticas que se presentan en el apéndice A, en especial de la proposición "Recorte de no convexos 2" y de

la que dice que un polígono es convexo si solo si todos sus vértices lo son. Además, se usa libremente las definiciones presentadas en él.

Antes de ver cómo se utilizan estas herramientas en la solución se ha de tener en cuenta las siguientes observaciones.

- Los laberintos en los que se trabaja son de paredes planas. Si el mapa resultante no tiene errores, este debería formar un polígono no complejo ya sea con o sin agujeros.
- La nube de puntos obtenida del sensado forma un polígono gráfica polar. Esto es trivialmente cierto ya que para cada dirección se obtiene una distancia con el sensor láser. De esta forma, se puede definir una función f tal que a dichas direcciones le asigne la distancia dada por el sensor laser. Luego, para extenderla al conjunto $[-\pi, \pi]$ basta con realizar la interpolación lineal de los puntos.

Estas observaciones implican que es posible utilizar las proposiciones anteriores sobre los datos sensados. Si bien, realmente el algoritmo se aplicará al polígono generado dado por los segmentos extraídos en SLAM, las hipótesis seguirán siendo válidas ya que las rectas que se extraen de la gráfica polar no pueden estar alineadas con el origen de coordenadas. De lo contrario se contaría con varios puntos que están prácticamente alineados y por tanto la nube no sería una gráfica polar.

5.3.1. pTAM

En esta sección se describe el algoritmo implementado para resolver el problema pTAM. El algoritmo toma como entrada la nube de puntos sensada y a los segmentos extraídos en SLAM[1]. La salida pretende ser un polígono convexo que represente la región en la que se encuentra el robot y que tenga las propiedades deseables mencionadas en la sección 5.2.1. Debido a los ambientes en los que se trabaja, se asume siempre se pueden observar paredes con las que se pueda construir un tal polígono. Esto se debe a que los ambientes se constituyen principalmente de pasillos estrechos por donde circula el roedor/robot.

La idea del algoritmo consiste en etiquetar los vértices de los segmentos como convexos o no convexos[2] y, en base a las etiquetas, utilizando las proposiciones mencionadas anteriormente, recortar el polígono. Esto se hace recursivamente hasta que todos los vértices sean convexos.

Algorithm 5 Selección polígono pTAM

Entrada: X = nube de puntos sensada.
S = conjunto de segmentos extraidos en SLAM.

Salida: Polígono convexo pTAM

1: Crear poligono inicial utilizando los segmentos en S
2: Etiquetar vértices del poligono
3: **while** Existen vértices no convexos **do**
4: Recortar polígono
5: Etiquetar vértices
6: Retornar polígono obtenido

A continuación se detalla mejor únicamente el recorte de polígonos. El primer paso no se detalla, ya que simplemente consiste en crear una lista de punteros a los vértices de los segmentos que represente al polígono siendo creado. Por otro lado, el algoritmo de etiquetado de vértices se detalla en el apéndice de algoritmos.

[1] Notar que dichos segmentos no dependen de la dirección de la cabeza ni la posición.

[2] Un vertice de un polígono se dice convexo cuando el ángulo interior en dicho vértice es menor o igual a 180 grados. Más información se puede ver en el apéndice matemático.

5.3.1.1. Algoritmo de Recortes

Una vez que los vértices del polígono fueron etiquetados, se comienza a recortar el polígono por medio de los vértices no convexos. Supóngase que dichos vértices son $Y_1, ..., Y_k$, supóngase además que están ordenados en forma antihoraria y que O es el origen de coordenadas. El problema se divide en dos instancias:

- **$\forall i$ el ángulo $ang(Y_i, O, Y_{i+1}) < 180$:**

En este caso, utilizando la proposición de recortes de polígonos 2, se retorna el polígono $\{Y_1, ..., Y_k\}$.

- **$\exists i$ tal que $1 \leq i \leq k$ y el ángulo $ang(Y_i, O, Y_{i+1}) \geq 180$:**

En este caso, utilizando un reordenamiento de los índices podemos suponer i=k. Entonces el ángulo $\theta = ang(Y_k, O, Y_1) \geq 180$. Supóngase los vértices convexos ordenados entre Y_k e Y_1 son $X_1, ..., X_m$ con $X_1 = Y_1$ y $X_m = Y_1$. Lo que se hará será construir el mayor polígono convexo posible comenzando desde Y_k. Para ello se comienza con el polígono $\{X_1, X_2, X_3\}$ y en cada paso se agrega un vértice hasta llegar a X_m inclusive. Para agregar un vértice se debe controlar que el polígono resultante siga siendo convexo. Para ello, si el polígono en construcción es $\{X_1, ..., X_i\}$, al agregar X_{i+1} se debe revisar que los los vértices X_1 y X_{i+1} sean convexos en $\{X_1, ..., X_{i+1}\}$. Además, la arista $[X_1, X_{i+1}]$ no debe intersectar ningún segmento del polígono siendo recortado. En la figura 5.3.1 se ilustra mejor la creación de este convexo.

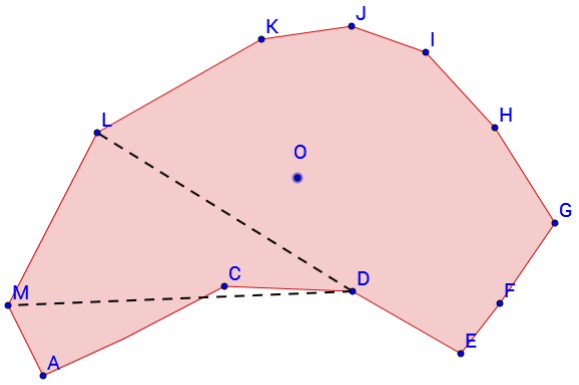

Figura 5.3.1 – Creación de convexos a partir de un punto.
En la figura se ejemplifica la creación de un convexo a partir de un punto. El punto O representa el origen de coordenadas (posición del robot). Los vértices no convexos son C y D, y $ang(D, O, C) > 180$. De esta forma se comienza creando el triángulo $\{D, E, F\}$. Luego se agregan vértices uno a uno hasta llegar a L. No es posible agregar el vértice M ya que D dejaría de ser convexo, inclusive si lo fuese, $[D, M]$ intersecta a $[C, A]$ por lo que tampoco se podría agregar.

Una vez obtenido el mayor convexo a partir de Y_k, supóngase que sea $\{X_1, ..., X_i\}$, se debe observar si contiene al origen. Para ello se revisa que el ángulo $ang(X_i, O, X_1) \geq 180$. En caso afirmativo, se retorna este polígono, en otro caso se retorna el polígono $\{Y_1, ..., Y_k, X_{i+1}, ..., X_m\}$.

Algo a tener en cuenta, es que la correctitud del algoritmo se basa en asumir que el polígono siendo recortado contiene al origen de coordenadas. De esta forma, en la explicación anterior, recordando que $\theta \geq 180$, se tiene que $m \geq 3$, de lo contrario se rompería la hipótesis. Finalmente, el hecho de que la hipótesis sea cierta se basa en que el robot/roedor trabaja en ambientes cerrados con lo cual siempre está rodeado de paredes.

5.3.2. pWG

El algoritmo para resolver el problema pWG se construye utilizando el polígono obtenido en pTAM. Así como en el algoritmo de SLAM la idea era llevar un conjunto de segmentos en

el mapa los cuales luego eran reconocidos y mejorados, en el pWG se trabaja de la misma manera. Una vez que se extrae un segmento se pasa por un proceso de identificación. Si es identificado la información del polígono es actualizada, en caso contrario el polígono es agregado al mapa. Los algoritmos enunciados en esta sección deberían ser ejecutados luego de las actualizaciones realizadas en SLAM.

5.3.2.1. Cambios estructurales en el mapa

Tanto para reconocer, como para almacenar los polígonos en el mapa, su estructura debe ser modificada. Para ello, se realizan los siguientes cambios:

- Se agrega un array de polígonos al mapa.
- A cada vértice del mapa se le agrega un array de punteros a los polígonos que pertenece.
- Se crea una tabla en el mapa que, a todo par de vértices orientados que conformen una arista de un polígono les asigna un puntero a dicho polígono. Esta tabla será denominada hash de polígonos.
- Se crea una estructura para almacenar polígonos. La estructura es un array de punteros a vértices.

5.3.2.2. Modificación de actualización de la información en SLAM

Como fueron agregadas estructuras al mapa, la modificación de los segmentos en SLAM tendrá repercusiones en las estructuras agregadas. En particular hay dos tipos de cambios que tienen efectos.

- **Eliminación de aliases de vértices.**

Cuando dos aristas que antes no se intersectaban pasan a intersectarse, los dos vértices anteriores pasan a ser uno solo. De esta forma, se debe actualizar todos los polígonos a los que pertenecen, y el hash de polígonos para reflejar este cambio. Además, los arrays de polígonos a los que pertenecen los puntos se deben unir en uno solo.

- **Modificación de la posición de los vértices.**

La modificación de la posición de los vértices hace que cambie la forma de los polígonos. Eventualmente, estos podrían dejar de ser convexos, en cuyo caso habría que particionar dichos polígonos. En este trabajo esto no se realiza ya que no es esencial para el funcionamiento y esta parte del proyecto sólo es una prueba de concepto.

5.3.2.3. Identificación de polígonos

Dado un polígono, la idea para identificarlo consiste en utilizar los vértices que hayan sido reconocidos durante SLAM (es decir los vértices de las aristas que fueron reconocidas). Para ello, lo que se hace es ver si alguna de las aristas orientadas de P se encuentra en el hash de polígonos. Este paso puede ser realizado en cualquier momento luego de realizada la identificación de segmentos.

Algorithm 6 Identificación de polígonos

Entrada: $P = \{x_1, ..., x_n\}$ = polígono obtenido en pTAM.

Salida: Puntero al polígono correspondiente en el mapa. En caso de que no exista se devuelve un puntero nulo.

```
1: for i = 1, n do
2:     j = i + 1
3:     if x_i, x_j son vertices que fueron reconocidos en SLAM then
4:         if hashPoligonos[x_i][x_j] no es nulo then
5:             Devolver hashPoligonos[x_i][x_j]
6: Devolver puntero nulo
```

5.3.2.4. Actualización de polígonos en el mapa.

Una vez realizada la identificación, si se determina que el polígono no es ninguno de los existentes, este es agregado al mapa completando todas las estructuras necesarias.

Si el polígono fue reconocido, entonces se procede a integrar la información en cuatro pasos

- **Eliminación de aliases.**

Como se mencionaba anteriormente, el mapa puede tener aliases de puntos debido a que no se intersectan dos aristas que debieran estarlo. La existencia de aliases hace que no pueda utilizarse información de conexión entre aristas usada en el siguiente paso (unión de polígonos). Por este motivo, primero se desea eliminar posibles aliases de los vértices del polígono obtenido en pTAM. Para ello, estos son comparados contra los del polígono identificado del mapa y sus adyacentes. Para que dos puntos sean considerados como uno, se pide que la distancia entre ambos sea menor que la constante de conexión presentada en SLAM.

Algorithm 7 Eliminación de aliases

Entrada: $P = \{x_1, ..., x_n\}$ = polígono obtenido en pTAM.
$Q = \{y_1, ..., y_m\}$ = polígono identificado del mapa.

Constantes: K = Constante de conexión

```
1: HuboCambios = true
2: while HuboCambios do
3:     HuboCambios = false
4:     for i = 1, n do
5:         for j = 1, m do
6:             if d(x_i, y_j) < K then
7:                 EliminarAlias(x_i,y_j)
8:                 HuboCambios = true
9:             else if y_j tiene siguiente y d(x_i, sig(y_j)) < K then
10:                EliminarAlias(x_i,sig(y_j))
11:                HuboCambios = true
12:            else if y_j tiene anterior y d(x_i, ant(y_j)) < K then
13:                EliminarAlias(x_i,ant(y_j))
14:                HuboCambios = true
```

En el algoritmo anterior, faltaría decir como se realiza la eliminación del alias, pero esta consiste únicamente en tomar los segmentos a los que pertenecen dichos puntos, intersectarlos y modificar toda la información asociada para reflejar este cambio. En particular, tanto en el polígono obtenido en pTAM como en su asociado, el puntero al vértice debe ser modificado para apuntar a la nueva intersección.

- **Unión de polígonos.**

Una vez resueltos lo aliases se procede a unir ambos polígonos. Para ello se procede en dos pasos. Primero se utiliza la información de conexión para unirlos y luego se recorren los vértices del polígono pTAM insertándolos en el del mapa. Más específicamente, en el segundo paso se parte de una arista en común y se recorre ambos polígonos al mismo tiempo. En cada paso se pone primero el punto que haga al ángulo convexo.

A continuación se presenta el algoritmo para unir los polígonos usando herramientas de adyecencia. Luego se da una idea general de como terminar la unión.

Algorithm 8 **Unión por adyacentes**

Entrada: $P = \{x_1, ..., x_n\}$ = polígono obtenido en pTAM.
$Q = \{y_1, ..., y_m\}$ = polígono identificado del mapa.

```
1: HuboCambios = true
2: while HuboCambios do
3:     HuboCambios = false
4:     for i = 1, n do
5:         if x_i ∉ Q then
6:             for j = 1, m do
7:                 if x_i = sig(y_j) then
8:                     Q ⇐ {y_1, ..., y_j, x_i, y_{j+1}, ..., y_m}
9:                     HuboCambios = true
10:                    Break
11:                else if x_i = ant(y_j) then
12:                    Q ⇐ {y_1, ..., y_{j-1}, x_i, y_j, ..., y_m}
13:                    HuboCambios = true
14:                    Break
```

Antes de proceder a comentar el próximo paso, se debe tener en cuenta lo siguiente. Los vértices de un polígono pueden ser particionados en componentes conexas utilizando la información de conexión de los vértices. De esta forma, una vez finalizado el algoritmo anterior, si x es un vértice que se encuentra tanto en el polígono pTAM como en el del mapa, entonces toda su componente conexa en pTAM ha de estar incluida en el polígono del mapa. De igual forma, si un vértice en pTAM no pertenece al polígono del mapa, ningún otro de su componente puede estar incluido. De lo contrario, por la parte anterior, toda la componente lo estaría, lo que es absurdo. De esta forma, agregar los puntos restantes equivale a insertar en el polígono del mapa las componentes conexas del polígono pTAM que aún no estén incluidas.

Para realizar esta tarea se puede proceder mediante un algoritmo de estilo similar a merge-sort. Para ello se parte de una componente conexa que ya esté agregada y luego se van agregando el resto de las componentes utilizando el siguiente método como ordenamiento. Para ello, primero se define un orden en los puntos y luego se extiende a las componentes. Sean x un punto que partenezca tanto al polígono A como al polígono B. Sea y_A e y_B el siguiente punto en A y B respectivamente. Se dirá que $y_A < y_B$ si y_B se encuentra contenido en el semiplano positivo dado por el segmento orientado $[x, y_A]$ sin estar alineado al segmento. Si los tres puntos estan alineados se dira $y_A < y_B$ si $dist(x, y_A) < dist(x, y_B)$. Para ejemplificar mejor este concepto tanto como para ver como se extiende a las componentes, observese las figuras. 5.3.2 y 5.3.3.

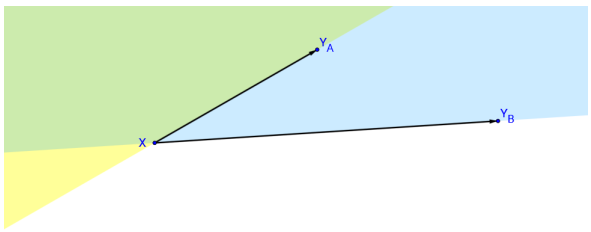

Figura 5.3.2 – Orden de vértices.
En la figura se ejemplifica como ordenar los vértices de dos segmentos que compartan un punto. El semiplano superior dado por $[X, Y_B]$ es pintado de color celeste, mientras que el semiplano superior del segmento $[X, Y_A]$ es pintado de amarillo. La sección de color verde corresponde a la intersección de ambos semi planos. Como se puede observar, Y_A pertenece al semiplano superior de $[X, Y_B]$ con lo cual $Y_B < Y_A$

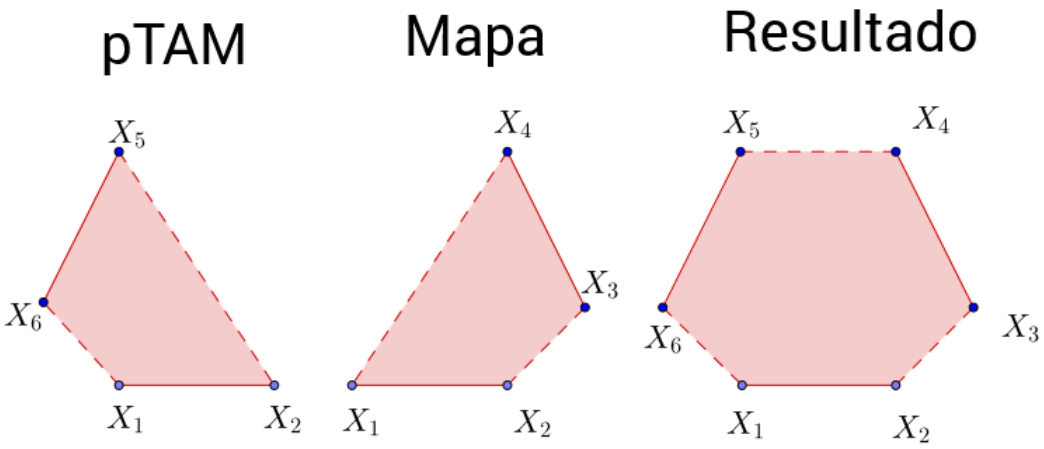

Figura 5.3.3 – Unión de polígonos. En la figura se observa al polígono obtenido pTAM, a su poligono correspondiente en el mapa y al resultado de combinar ambos. Las lineas continuas indican que los vértices están comunicados por un segmento (es decir que hay una pared entre ellos), mientras que las lineas punteadas indican lo contrario. En la figura se observan tres componentes conexas correspondiente con los segmentos $[X_1, X_2]$, $[X_3, X_4]$ y $[X_5, X_6]$. La componente común en ambos polígonos es la primera. Para ordenar la segunda y tercer componente se aplica la definición de orden a los vértices X_2, X_3 y X_5. Como X_5 está contenido en el semiplano positivo dado por el segmento orientado $[X_2, X_3]$ se tiene que $X_3 < X_5$ y por tanto $[X_3, X_4] < [X_5, X_6]$.

- **Recorte de polígonos.**

Una vez unidos los polígonos, el polígono resultante puede que ya no sea convexo. Por este motivo el polígono se recorta hasta que vuelva a ser convexo. Esto se realiza de igual forma que en pTAM, salvo que en el etiquetado inicial no se cuenta con una nube de puntos para considerar. Ocasionalmente, al terminar los recortes, el resultado puede ser un subpolígono convexo del polígono original antes de efectuar la unión. En dicho caso, si el polígono original sigue siendo convexo, entonces se descartan los cambios ya que se quiere dividir en la menor cantidad de convexos posibles.

- **Actualización de información dependiente**

Para concluir esta sección, el último paso que se debe realizar es actualizar toda la información relacionada al polígono resultante. Para ello se debe modificar toda la información dependiente para que el mapa (base de datos) quede en un estado consistente. Para ello, es

necesario modificar el hash de polígonos eliminando las aristas que ya no existan y agregando las nuevas. También es necesario reflejar los cambios en los vértices modificados del polígono. Si un vértice fue sacado/agregado, la referencia al polígono también debe ser sacada/agregada a la lista de polígonos del vértice.

Capítulo 6

Robot y laberinto

En esta sección se describe los módulos físicos implementados (robot y laberinto), así como su uso y funcionamiento. Todos los módulos responden a los siguientes factores:

- Necesidades impuestas por los algoritmos implementados.
- Presupuesto disponible.
- Disponibilidad de materiales.

6.1. Robot

6.1.1. Requerimientos

Los requerimientos del robot pueden dividirse en las siguientes categorías:

- **Sensado**

El robot debe ser capaz de medir la distancia a las paredes para 100 direcciones distintas (una cada 3.6 grados).[1]

- **Movimiento**

Imitando al artículo de referencia, el robot debe ser capaz de realizar navegación diferencial.

- **Comunicación**

Como se verá más adelante, el programa de control ejecuta de forma distribuida, con lo cual el robot debe ser capaz de comunicarse con una PC.

- **Cómputo**

El robot funciona como un dispositivo esclavo del programa principal. El procesamiento de la información se realiza principalmente en una PC. De esta forma, se tiene necesidades de cómputo bajas.

6.1.2. Hardware

Teniendo en cuenta los requerimientos del robot y los materiales disponibles, el robot implementado cuenta con:

[1] Se utilizan 100 direcciones ya que, al comienzo del proyecto se utilizaba como referencia al artículo **??** en el cual usaban como entrada estos datos y se heredó el número.

- **Una SBC Fox G20[39].** La placa cuenta con un procesador Atmel ARM9 de 400Mhz, 256KB de memoria flash para el bootloader, almacenamiento externo mediante una memoria microSD con direccionamiento hasta 16GB, dos puertos huésped USB 2.0, un puerto dispositivo USB, un puerto Ethernet 10/100, dos puertos seriales de 3.3V, buses para I2C y SPI, etc.
- **Memoria microSD de 8GB.**
- **Dos servomotores Dinamixel AX12**[40] **y una rueda loca** para el movimiento. Los motores AX12 tienen un ángulo operacional de 300 grados, resolución de 0.29 grados, capacidad para rotación continua y velocidad angular máxima de 59RPM. El voltaje operacional es de 12V, y las corrientes mínima y máxima son 50 y 900mA respectivamente. El protocolo de comunicación es un protocolo serial half-duplex asincrónico, el cual permite conectar hasta 254 motores en serie.
- **Un dispositivo USB2Dynamixel.** Permite la comunicación de los motores a un puerto USB.
- **Dos servomotores Dinamixel AX12 y un sensor láser Lidar Lite v1[41]** para el sensado. El sensor Lidar Lite v1 es un sensor de distancia láser económico con un coste de aproximadamente 100 US$ a la fecha. El dispositivo tiene soporte para comunicación mediante I2C o PWM y permite obtener la distancia a un objeto con un rango operacional entre 0 y 60 metros en menos de 0.02 segundos y apreciación de 1cm. Según sus especificaciones, el sensor tiene un error máximo de 2.5cm en el peor caso. Para poder obtener las distancias en 100 direcciones en los 360 grados, el sensor LidarLite fue montado sobre dos servomotores. Fue necesario utilizar dos ya que el rango operacional de cada uno es de 300 grados.
- **Un adaptador wireless VB279-USB WiFi Link** para la comunicación entre la placa y la PC.
- **Una fuente switcheada**
- **Pack de 8 baterías AA**
- **Piezas bioloid[42] y piezas diseñadas en acrílico**
- **Tuercas, tonillos, etc**.

En las siguientes imágenes se muestra el robot construido.

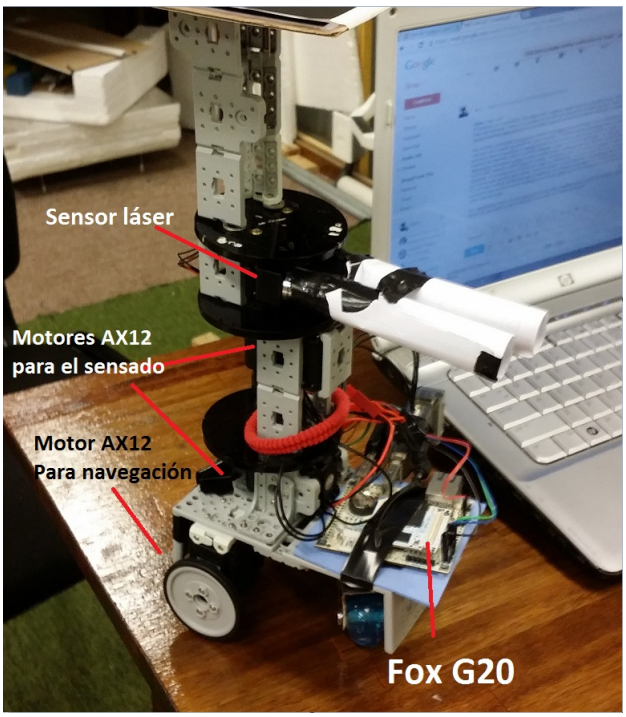

Figura 6.1.1

Figura 6.1.2 – Robot - vista completa.
En la figura se muestra el robot casi en su versión final. La única diferencia es la longitud del "largavista" de papel utilizado, que es más largo que en la versión final. En la imagen, le estructura vertical que sobre sale, es el montaje de los dos servomotores y el sensor láser. Dicha estructura es prolongada para colocar una marca de trackeo a 35cm del piso, la cual es utilizada en el sistema de evaluación[2]. El primer servomotor de la estructura se encuentra en su base, debajo del primer disco de acrílico, y el segundo, se encuentra debajo del siguiente disco. El sensor láser es el objeto entremedio de los dos discos superiores observados.

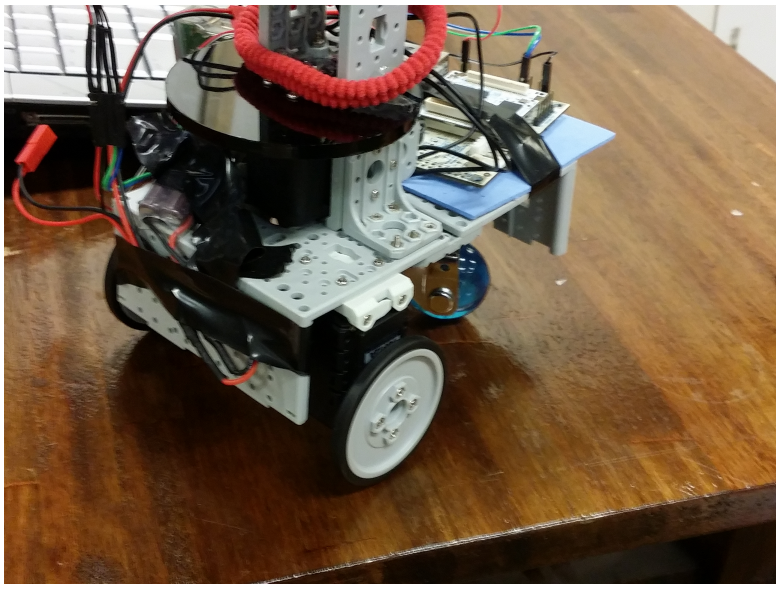

Figura 6.1.3 – Robot - Navegación diferencial.

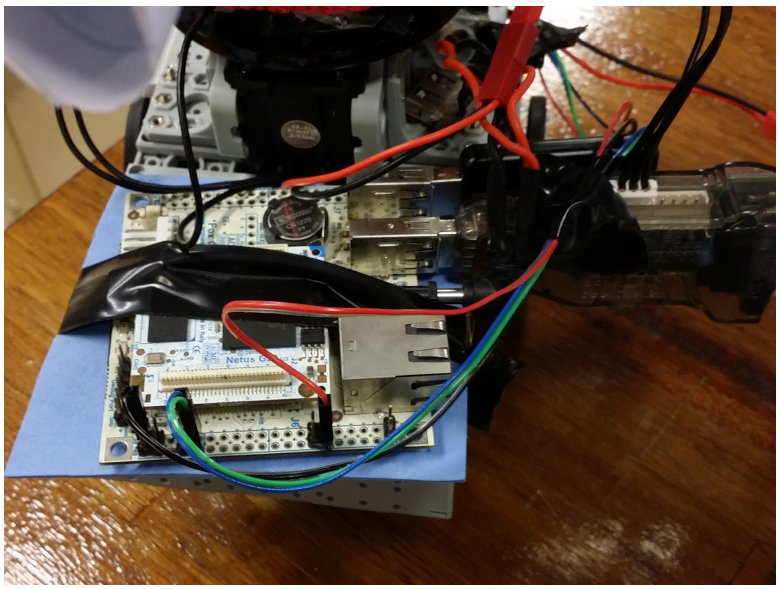

Figura 6.1.4 – Robot - SBC.
En la imagen se puede apreciar a la SBC FOX G20 y sus conexiones. Los cuatros cables conectados a los GPIO son los utilizados para la comunicación I2C con el sensor láser.

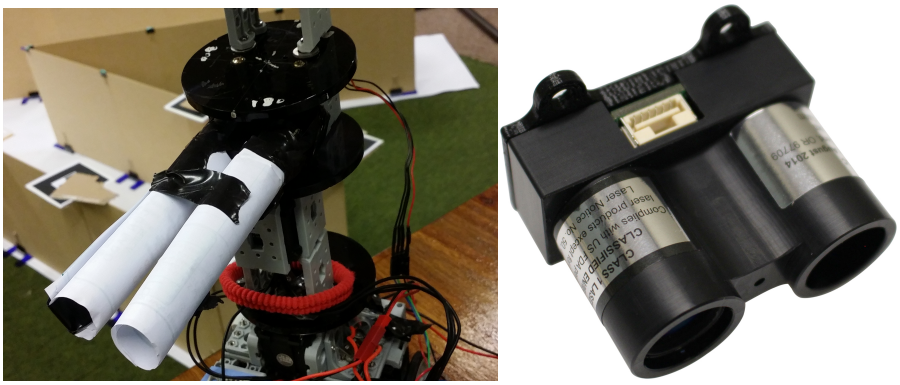

Figura 6.1.5 – Robot - Sensor láser.

6.1.2.1. Sensor Lidar Lite v1

Antes de proceder con la siguiente sección, se dedica un espacio para hablar del sensor láser y explicar mejor el motivo de los "largavistas" de papel.

Si bien el sensor láser Lidar Lite v1 en sus especificaciones indica que el sensor comete un error máximo de 2.5cm, en la práctica esto no resultó ser cierto. Luego de calibrado el instrumento, al probar obtener medidas desde el centro de un laberinto cuadrado de 80cm de lado, se encontró que el error cometido podía alcanzar los 10cm. Considerando que la longitud máxima (del centro a un vértice del cuadrado) es de 56cm, el error representa un mínimo aproximado de 18% del valor real. Por otro lado, se encontró que el error en el cuadrado era "geométrico", es decir, el error dependía de la geometría del espacio. Esto se puede observar mejor en la figura 6.1.6.

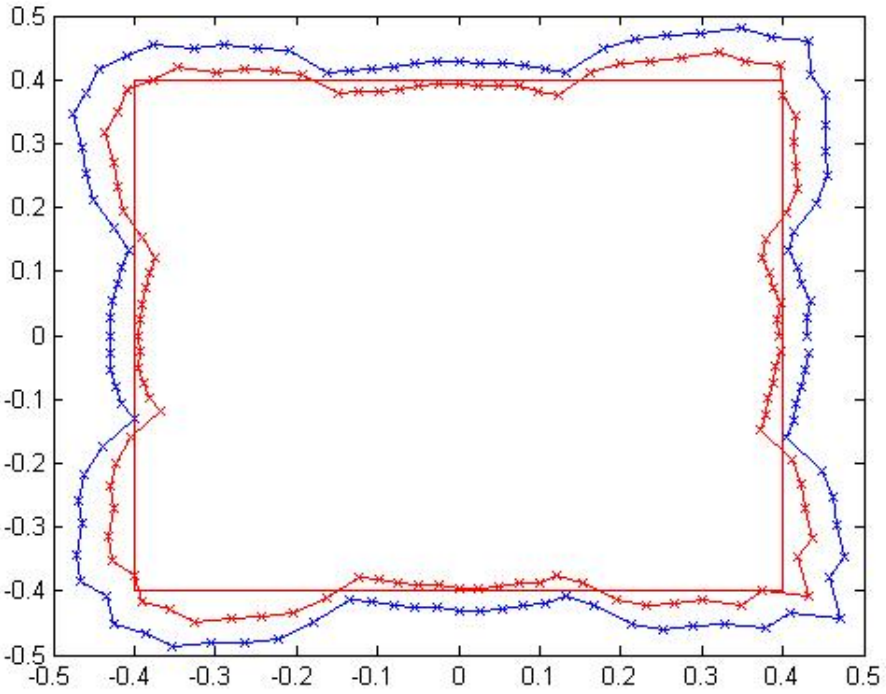

Figura 6.1.6 – Error antes de utilizar el largavista.
En la figura se observa la nube de puntos obtenida por el robot desde el centro del cuadrado. Las nubes azul y roja son las obtenidas antes y después de la calibración respectivamente. Esta se realiza sumando/restando la misma constante a todas las distancias obtenidas por el sensor láser. La constante se determina hallando el valor para el cual se minimiza el error medio de las medidas.

A partir de esta observación se realizó una investigación en la cual, en un foro del producto, se encontró que un usuario obtenía medidas erróneas al utilizar el sensor en ambientes cerrados. El usuario reportaba que la causa se debía a reflexiones internas en el ambiente y comentaba que el problema se resolvía al agregarle una especie de largavista al sensor. En el mensaje, adjunto, se incluía un archivo para impresión 3d creado por él[43].

Al obtener esta información, se volvió a realizar la prueba del cuadrado pero esta vez utilizando un largavista de papel de aproximadamente 8cm de largo. El resultado se puede observar en la figura 6.1.7.

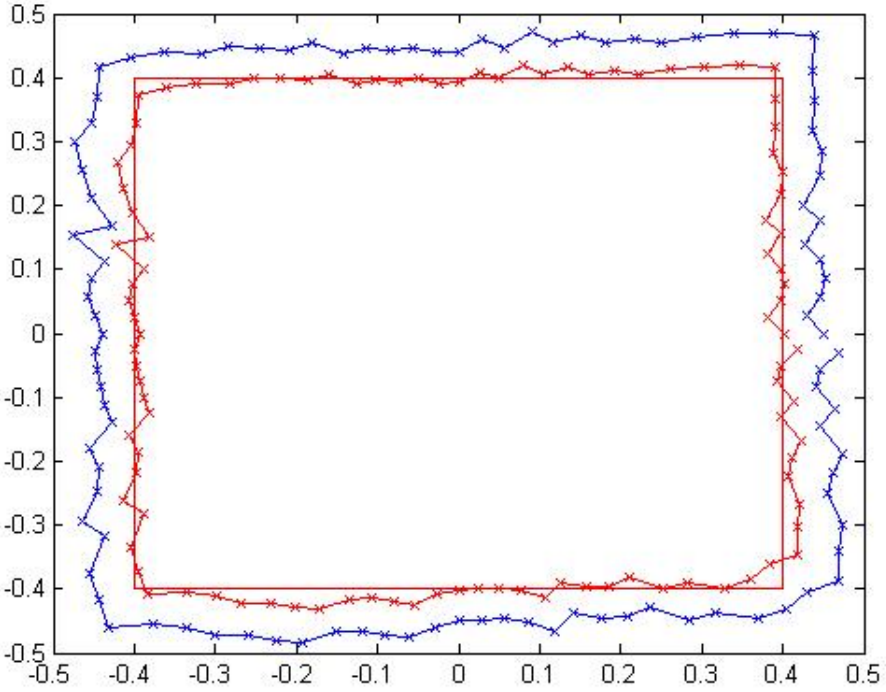

Figura 6.1.7 – Error cuando se utiliza el largavista.
En la figura se observan la nubes de puntos obtenidas por el robot luego de agregar el largavista. Nuevamente, las nubes azul y roja son las obtenidas antes y después de la calibración respectivamente. Comparando contra la figura 6.1.6, se puede observar que los puntos se ajustan notoriamente mejor al cuadrado.

Finalmente, el sensor fue probado en el laberinto que se usaría para los experimentos (laberinto de 8 brazos radiales, con brazos de 1.2m de largo y 0.4m de ancho - ver parte III del informe). La pruebas realizadas al sensor consistían en graficar las nubes de puntos obtenidas por el robot desde distintas posiciones del laberinto. Las variables de control eran el material de los largavistas y su longitud. Se probaron longitudes desde 12 cm hasta 3 cm aproximadamente. Los materiales probados fueron cilindros de papel (hechos a mano), tubos de PBC, y plástico PLA blanco (en el largavistas impreso del archivo). Se encontró que los mejores resultados se obtenía utilizando los tubos de papel. Se cree que esto está relacionado a lo que menciona el usuario en su post, que dice que la mayoría de los plásticos son invisibles para el láser, lo que hace necesario revestir el largavista con pintura adecuada. Por otro lado, en el estudio realizado se encontró que los mejores resultados se obtenían al usar largavistas de entre 4 y 6cm. A continuación se presentan algunas de las imágenes obtenidas en diferentes casos.

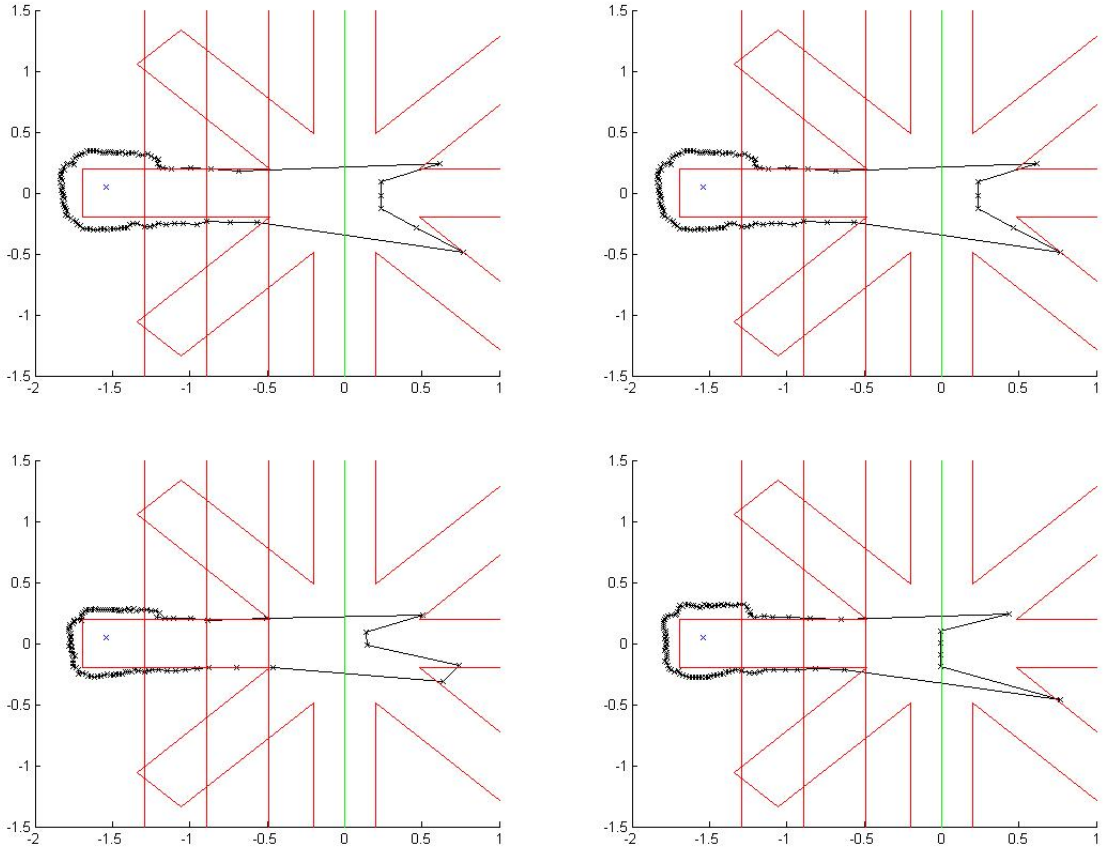

Figura 6.1.8 – Error del láser variando el largo del largavista.
En la figura, en la imagen de arriba a la derecha se observa la nube de puntos sin usar un largavistas. A su izquierda se utiliza un largavista de 12cm. En la fila inferior, se utilizan largavistas de 6 y 3 cm respectivamente. La "x" en cada imagen se corresponde con la posición del robot y los ejes están medidos en metros.

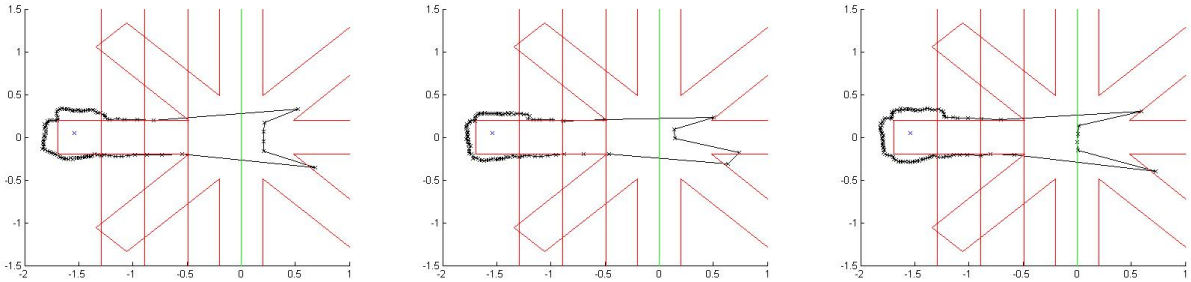

Figura 6.1.9 – Error del láser variando el material del largavista.
En la figura, de izquierda a derecha, se observan las nubes de puntos obtenidas con los largavistas de igual longitud de PBC, papel y PLA respectivamente. La "x" en cada imagen se corresponde con la posición del robot y los ejes están medidos en metros.

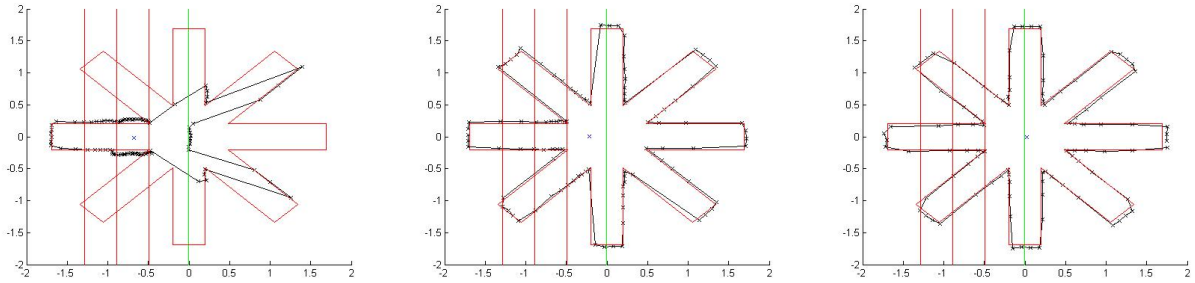

Figura 6.1.10 – Error del láser variando la posición.
En la figura, se observa la nube de puntos obtenida para distintas posiciones del laberinto utilizando un largavista de papel de aproximadamente 5cm de largo. La "x" en cada imagen se corresponde con la posición del robot y los ejes están medidos en metros.

Algo que puede observarse en las figuras anteriores es que, por lo general, el láser tiene mayor error para los puntos cercanos. Pasados los 40cm aproximadamente, el error comienza a estar en el rango reportado por el manual del dispositivo. Finalmente, en algunas de las imágenes se observan "puntos fantasmas", es decir, puntos donde no hay ningún objeto cercano. Estos puntos siempre se dan en las mismas posiciones del robot en laberinto y para las mismas direcciones. No se tiene certeza de la causa de ese error.

6.1.3. Software

6.1.3.1. Sistema operativo y lenguaje de programación.

El sistema operativo utilizado en el robot es EmDebian 'Wheezy' Grip 7.4. El lenguaje de programación es Lua 4.2.

Para generar la microSD booteable, se siguió la documentación ofrecida en[44]. Para montar el rootfs se utilizó la imagen provista en [45]. La imagen del kernel provisto en esa misma página no tiene soporte para I2C ni tampoco para el adaptador WiFi link, por lo que fue necesario recompilar el kernel agregando los dos módulos necesarios. Para ello, se utilizó el tutorial provisto en [46].

El sistema operativo utilizado en el PC es windows 7. Por motivos de eficiencia computacional, en vez de utilizar Lua como lenguaje de programación, se utiliza Luajit 2.0.2.

6.1.3.2. Distribución de módulos

Principalmente por cuestiones de reproducibilidad, el algoritmo implementado se ejecuta de manera off-line. Es decir, primero se utiliza el robot para relevar un conjunto de datos, los cuales luego son utilizados como entrada del algoritmo principal. Esto permite ver como se altera el resultado para un mismo conjunto de datos al cambiar el algoritmo. De esta forma, el algoritmo principal del robot es ejecutado en una PC.

Para relevar los datos utilizando el robot, se cuenta con dos módulos. Un módulo de control que ejecuta desde una PC y un módulo esclavo que ejecuta en el robot e implementa una interfaz para permitir su uso. Ambos módulos se comunican mediante TCP. El módulo de control permite a un usuario controlar los movimientos del robot indicando en cada momento las velocidades de cada rueda. A su vez permite indicarle al robot cuando realizar el sensado. Tanto los movimientos del robot como los valores sensados son almacenados por el módulo principal en archivos de texto para su posterior uso.

A continuación se enumeran las funciones provistas por la interfaz del robot.

- ***velocidades*(*izq*, *der*)**: Permite setear las velocidades de las ruedas izquierda y derecha del robot. Devuelve una marca de tiempo que indica cuando se envió el comando a los motores. Esta marca es utilizada para implementar la integración de trayectorias.

- ***distancia*(*dir*)**: Devuelve la distancia sensada en la dirección representada por ***dir***, que toma valores entre 1 y 100. El valor 1 indica la dirección -180 grados y cada índice sucesivo suma 3.6 grados hasta llegar al valor 100 que indica la dirección 177.4 grados. Notar que las direcciones -180 y 180 grados son la misma.
- ***distancias*360()**: Devuelve la distancia sensada para las 100 direcciones de la función anterior.
- ***exit*()**: Función para finalizar la ejecución. Hace "halt" del sistema.

6.2. Laberinto modular

6.2.1. Motivación

Pensando en la realización de pruebas para el algoritmo SLAM, y a futuro, en diferentes pruebas con algoritmos de RL, se manejaron diferentes posibilidades de laberintos, todos ellos de paredes planas, siendo estos algunos de los que se encuentran más frecuentemente en los experimentos realizados con roedores. Para esto, se plantearon determinadas características que tenía que tener el mismo, para que se pudieran realizar las pruebas de forma óptima y similar a las que se venían realizando en el simulador. Las mismas son:

- **Precisión:** Como el algoritmo de SLAM implementado tiene un enfoque métrico, era de interés saber cuán bien el robot construye el mapa. De esta forma, para poder evaluar con precisión, se deseaba que los laberintos construidos coincidieran con su diseño. Para ello, cuanto mayor sea la precisión en los ángulos y las distancia, mejor será la evaluación.
- **Modularidad**: Esta necesidad surgía de los siguientes motivos. En primer lugar, no se sabía qué espacio se iba a disponer para utilizar. Por ello se deseaba que se pudiera cambiar las dimensiones del laberinto para adaptarse al espacio, y que sea fácilmente trasladable. Además, como todavía no se tenía determinado con qué laberinto se iba a trabajar, se planteó construir un conjunto de piezas que sirvieran para construir varios laberintos.

6.2.2. Solución

Para esto se utilizaron placas rectangulares en MDF de 35cm x 40 cm de lados, y 3 mm de espesor para que fueran las paredes del laberinto. A la vez, se realizó el diseño de piezas que sirvieran como "perfiles" y que sostuvieran dichas paredes. Además, el diseño permitiría reutilizar las piezas en los diferentes laberintos, respetando sus ángulos y medidas. El diseño constaba de tres tipos de piezas:

- Piezas para ángulos de 90 grados entre paredes.
- Pieza para ángulos de 45 grados entre paredes.
- Piezas para la extensión de paredes.

Las piezas fueron cortadas en acrílico con la cortadora láser de la Facultad de Ingeniería, y el diseño de las mismas fue realizado en Corel.

Luego se utilizaron aprieta papeles, que fueron pegados con silicona a las piezas en acrílico de forma de lograr los ángulos pretendidos.

En la figura 6.2.2 se ve el proceso de creación de las piezas de ángulos de 90 y 180 grados.

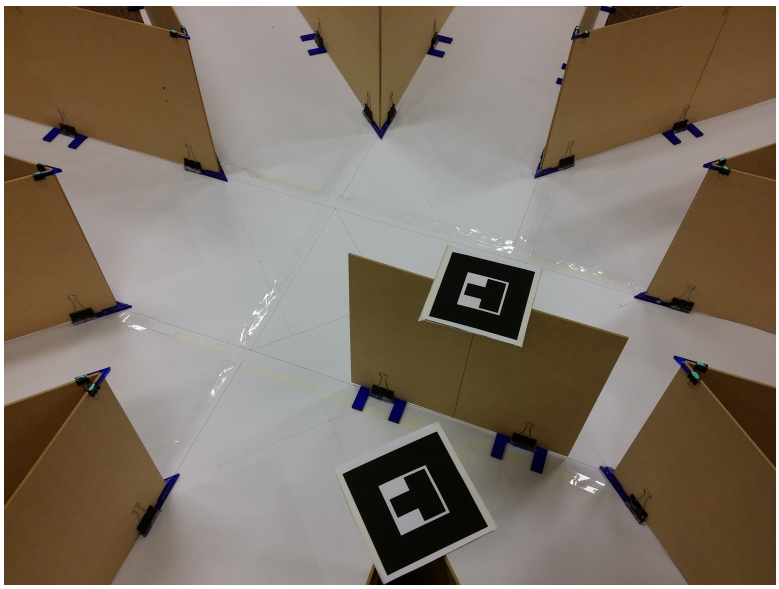

Figura 6.2.1 – Piezas de 45 grados en laberinto radial.
En la figura se observa el centro del Laberinto Radial armado, en el cual se destacan las placas de MDF colocadas sobre las piezas de 45 grados formando el centro del laberinto.

Figura 6.2.2 – Creación del laberinto, piezas de 90 y extensoras.

Figura 6.2.3 – Laberinto Radial Completo.
Armando en la sala de Robótica de la Facultad de Ingeniería.

Figura 6.2.4 – Laberinto Radial Completo desde adentro.

Parte III

Evaluación

Capítulo 7

Experimentación y resultados

7.1. Ambientes de experimentación

Debido a que realizar las pruebas con el robot real conlleva una gran cantidad de tiempo, las pruebas fueron realizadas en dos ambientes. Se realizó una corta con el robot real, y luego el resto en ambientes simulados.

7.1.1. Ambiente de simulación

Durante este proyecto se utilizó como plataforma de simulación a V-Rep[47].

El simulador de robots V-REP, que cuenta con un entorno de desarrollo integrado, se basa en una arquitectura de control distribuido: cada objeto / modelo puede ser controlado de forma individual ya sea a través de un plug-in, un nodo de ROS, un cliente de API remoto, o una solución personalizada. Esto hace a V-REP muy versátil e ideal para aplicaciones multi-robots. Los controladores pueden ser escritos en C / C ++, Python, Java, Lua, Matlab, Octave o Urbi. V-REP se utiliza para el rápido desarrollo de algoritmos, simulaciones de automatización de fábrica, prototipado y verificación rápidos, la robótica relacionada con la educación, etc. Durante este proyecto, se utilizó el lenguaje de programación Lua, tanto para el ambiente simulado como para el ambiente real.

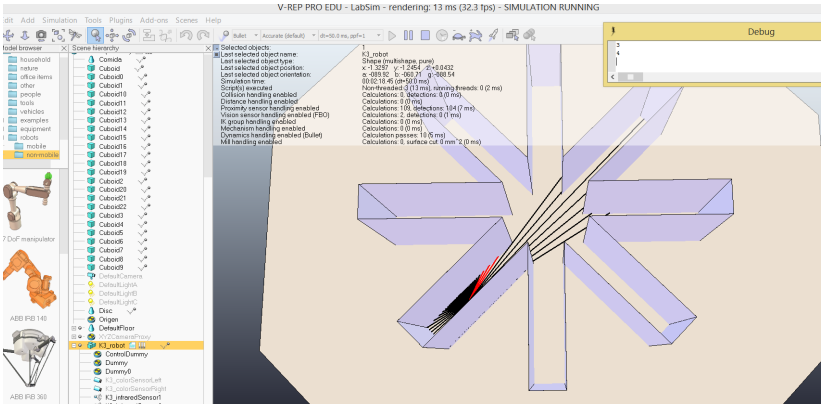

Figura 7.1.1 – Simulación en V-Rep

7.1.2. Ambiente real

Para la prueba con el robot real, se decidió montar el laberinto de 8 brazos radiales utilizando las piezas diseñadas en la sección 6.2. Los brazos del laberinto medían 120 cm de largo y 40 cm de ancho. El robot utilizado es el descrito en la sección 6.1.

Para poder evaluar los resultados del algoritmo implementado, se dispuso de un cámara posicionada sobre el escenario. Poniendo una marca sobre el robot y utilizando las librerías de ArToolkit[48], se pretendía obtener la posición y dirección de la cabeza del robot en cada momento. Los resultados de esto no fueron satisfactorios. En ocasiones, la dirección de la cabeza podía estar rotada en 90, 180 y 270 grados, es decir, suponiendo el valor real era 35 grados, los valores que se obtenían eran cercanos a uno de los siguientes: 35, 125, 215 y -65 grados. A su vez, si bien el error en la posición comenzaba siendo del orden de los centímetros luego de la calibración, a medida que el tiempo avanzaba este aumentaba pudiendo alcanzar valores entre 20 y 30cm. Por esto motivo, utilizando los valores de la cámara como base, la posición y dirección de la cabeza fueron corregidas manualmente, haciendo coincidir la nube de puntos obtenida por el robot con la forma del laberinto.

Figura 7.1.2 – Escenario real de simulación.
En la esquina superior derecha de la figura, con un recuadro rojo se ve señalada la cámara encargada de registrar el experimento. La misma se encuentra ubicada de forma tal de visualizar todo el laberinto.

7.2. Pruebas realizadas de SLAM.

7.2.1. Plan de pruebas

Como se mencionó anteriormente, realizar pruebas con el robot conlleva mucho tiempo. Por esto motivo, solo se realizó una prueba con el robot para ver como se comportaba el algoritmo en un "ambiente real". El resto de las pruebas fueron realizadas a través del simulador. Las mismas pueden ser divididas en dos categorías. La primera incluye pruebas de tolerancia a errores cometidos ya sea por el láser o por la integración de trayectorias. La segunda categoría incluye pruebas que intentan evaluar cómo se comportaría el algoritmo en el ambiente real. Para ello, antes de efectuar estas pruebas, es necesario caracterizar los errores cometidos por el robot. Una vez encontrado un error que se asuma similar al error real, se procede a realizar pruebas más exhaustivas.

7.2.2. Resultados con el robot real

Se procedió a realizar una corrida en el laberinto radial de 8 brazos, tomando medidas con el láser cada intervalos de no más de 30-40 cm, y almacenando además los datos obtenidos por la integración de trayectorias. El objetivo era realizar una corrida offline del algoritmo SLAM con datos reales. Durante las pruebas se detectaron dos tipos de errores con características geométricas a tener en cuenta, generados por la medición del sensor láser:

- Se generan puntos fantasma en determinados lugares del laberinto, que se repiten en las sucesivas corridas. Los mismos son ignorados en la ejecución del algoritmo, por no cumplir con la restricción de cantidad de puntos mínima para formar una arista. Ver figura 7.2.1
- Se observa que las paredes cercanas al robot tenían mayor error en la medición de distancias del láser, que paredes más alejadas. Ver figura 7.2.2.

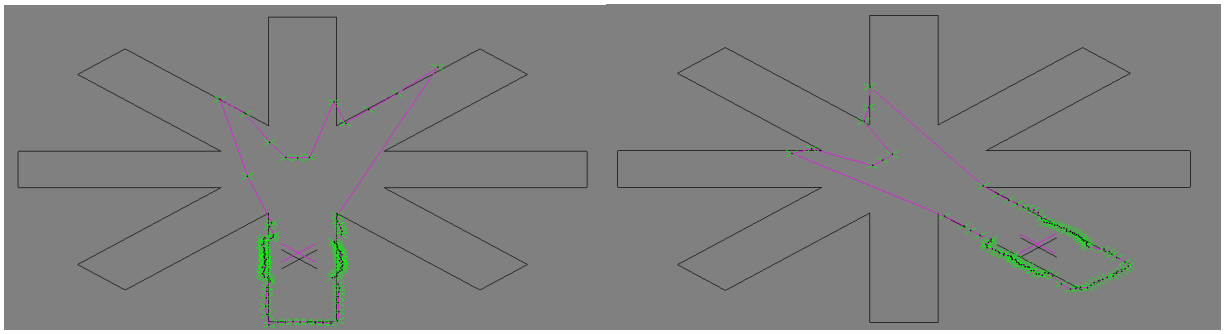

Figura 7.2.1 – Puntos Fantasma
En la figura se puede observar los puntos fantasmas alrededor del centro de laberinto. Estos puntos aparecen cuando el robot se encuentra en alguno de los brazos del laberinto.

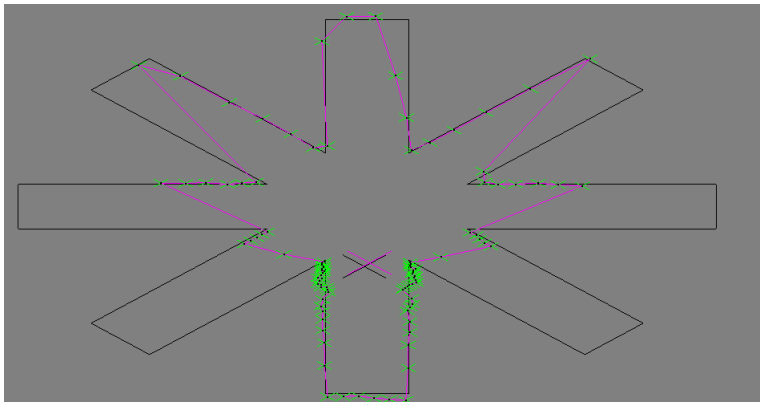

Figura 7.2.2 – Error geométrico
En la figura se observa como los puntos cercanos a la posición del robot tienen mayor error que los puntos alejados.

Como resultado de la ejecución del algoritmo SLAM sobre datos reales, se obtuvo la figura 7.2.3. Las métricas de evaluación utilizadas en este experimento fueron el error en la dirección de la cabeza y la posición. Los resultados pueden ser observados en la tabla

Resultados obtenidos con el robot real						
	Error Posición (cm)			Error Dirección de la cabeza (º)		
	Promedio	Máx.	Std. Dev.	Promedio	Máx.	Std. Dev.
Resultados	2.9	6.8	1.4	0.6	2.5	0.5

Cuadro 7.1 – Resultados de la prueba con el robot real

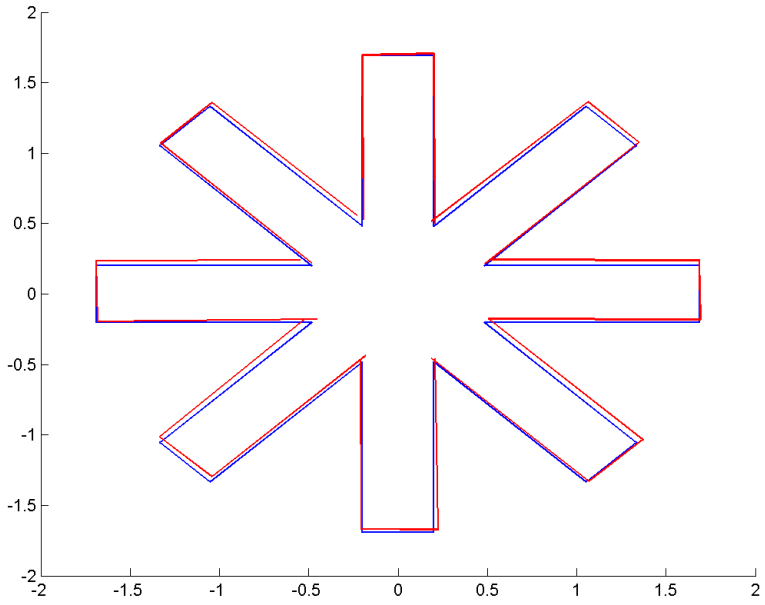

Figura 7.2.3 – Resultado del mapeo del laberinto real.
En la figura, en azul se muestra el mapa real del laberinto y en rojo se muestra el mapa obtenido por el algoritmo.

7.2.3. Tolerancia al error

Para evaluar la tolerancia del algoritmo frente a los distintos tipos de error, se generó una recorrida del robot por un laberinto similar al real pero en el simulador. Los datos extraídos de esta forma (posición del robot, dirección de la cabeza y medidas del sensor láser) carecen de ruido, con lo cual, inyectándolo manualmente en las ejecuciones off-line, es posible estudiar el comportamiento de los distintos tipos por separado. Se evalúan tres tipos de ruido:

- **Error en la dirección de la cabeza dada por la integración de trayectorias**
- **Error en la posición del robot dada por la integración de trayectorias**
- **Error en las medidas del sensor láser**

Como métricas de evaluación, se utiliza el error máximo y el error medio en la posición y dirección de la cabeza cometido por el algoritmo de SLAM. Estos resultados son el promedio de ejecutar el algoritmo con 10 semillas diferentes de números aleatorios. Para tener una linea base de comparación de los resultados, antes de comenzar, primero se ejecuta el algoritmo con los datos antes de inyectar ruido y luego se procede de la siguiente forma:

1. Se agrega ruido uniforme de distinta magnitud en la dirección de la cabeza dada por la integración de trayectorias observando el comportamiento del algoritmo. La magnitud es incrementada hasta que el algoritmo falle.

2. Se procede de igual forma que en el paso anterior, pero agregando ruido uniforme a la posición. El ruido se inyecta sumándole un $\Delta\vec{v}$ con dirección entre 0 y 2π, y norma acotada por la intensidad del ruido.
3. Se prueba el algoritmo utilizando ambos errores a la vez.
4. Se prueba el algoritmo agregando ruido gaussiano acotado a las medidas dadas por el sensor láser. En este caso, no se utilizan otros tipos de ruido.

Cabe destacar que los errores inyectados en la integración de trayectorias tienen magnitud constante. Esto se debe a que, entre cada iteración del algoritmo de SLAM, el robot se mueve a lo sumo 40-50cm con lo cual el error cometido se ve acotado. Idealmente, el error no debería ser uniforme y su magnitud debería ser proporcional a la longitud de la trayectoria realizada, sin embargo, estas asunciones simulan el peor caso posible.

7.2.3.1. Resultados

Dirección de la cabeza El algoritmo soporta ruido uniforme en la dirección de la cabeza hasta una magnitud de ± 20 grados, manteniendo sus resultados constantes hasta dicha magnitud. Con magnitudes mayores el error crece abruptamente y el algoritmo deja de ser útil. Esto se debe a que en el mismo se maneja una tolerancia de 25 grados en el reconocimiento de aristas. Para que una arista pueda ser reconocida como otra, la diferencia entre sus direcciones debe diferir menos de esa cantidad. A raíz de esto, errores mayores a 20 grados pueden generar confusión entre aristas, ya que, el laberinto utilizado (laberinto de 8 brazos radiales) presenta simetría cada 45 grados. De esta forma, si una arista esta rotada 20 grados, como la tolerancia es de 25 el robot podría perder noción del brazo en que se encuentra.

Ruido inyectado en dirección de la cabeza obtenido de PI						
Magnitud (º)	Error Posición (cm)			Error Dirección de la cabeza (º)		
	Promedio	Máx.	Std. Dev.	Promedio	Máx.	Std. Dev.
<20	0.3	0.5	0.2	0	0	0
20	21.8	116.4	34.7	16.7	45.1	21.8

Cuadro 7.2 – Resultados del primer análisis de ruido en PI

Posición Así como en la dirección de la cabeza, el ruido en la posición no afecta al algoritmo mientras la magnitud sea menor que 30cm. Nuevamente esto se debe a que, para reconocer aristas, una de las condiciones que se pide es que la distancia entre ambas sea menor que una constante (fijada en 30 cm). Pasada la misma, el robot puede dejar de reconocer las aristas y perderse.

Ruido inyectado en la posición obtenido de PI						
Magnitud (cm)	Error Posición (cm)			Error Dirección de la cabeza (º)		
	Promedio	Máx.	Std. Dev.	Promedio	Máx.	Std. Dev.
<30	0.3	0.5	0.2	0	0	0
30	33.4	116.3	36.5	25.7	46	22.0

Cuadro 7.3 – Resultados del segundo análisis de ruido en PI

Combinación de la dirección de la cabeza y la posición. Se combinó las magnitudes máximas de error en los pasos anteriores. El resultado se mantiene invariable, es decir, que el algoritmo soporta simultáneamente hasta 20 grados y 20 cm de error.

Medidas del sensor láser. El ruido en el sensor láser sí tiene consecuencias más notorias en el algoritmo. La cota del ruido gaussiano fue variada entre 0 (sin ruido) y 8 cm, aumentando de uno en uno en cada paso. Al llegar a la magnitud 8, el algoritmo falla en en la extracción de características (algoritmo de Hough modificado). A continuación se muestra la tabla de resultados.

Ruido inyectado en las medidas obtenidas por el láser						
Magnitud (cm)	Error Posición			Error Dirección de la cabeza		
	Promedio	Máx.	Std. Dev.	Promedio	Máx.	Std. Dev.
0	0.3	0.5	0.2	0	0	0
1	0.5	1.1	0.2	0.1	0.4	0.1
2	1.1	2.5	0.6	0.4	1.1	0.3
3	1.3	3.6	0.6	0.3	0.7	0.2
4	1.8	4.0	0.8	0.5	1.6	0.4
5	2.5	6.1	1.4	1.0	2.2	0.6
6	2.4	6.3	1.6	1.3	3.7	0.9
7	5.6	13.9	3.5	2.4	5.1	1.3
8	Falla					

Cuadro 7.4 – Resultado del análisis de ruido en las medidas del sensor láser.
Los datos se extrajeron promediando los resultados obtenidos de 10 corridas para cada magnitud de ruido.

7.2.4. Elección de ruido para el resto de las otras pruebas

7.2.4.1. Ruido en PI

Del análisis de la sección 7.2.3, se observa que sin ruido en el láser, el algoritmo soporta hasta 20 grados en la dirección de la cabeza. ya que, luego de ese umbral, es posible confundir aristas. Si se considera que al agregar ruido en el láser, las rectas que se obtendrán estarán torcidas (respecto a su orientación real), para que el algoritmo funcione, es necesario que el error en la dirección de la cabeza sumado al de las rectas sea menor que 20 grados. De esta forma, para dar algo de margen a que funcione el algoritmo, la magnitud de este error fue tomada en 10 grados.

Por otra parte, la magnitud del error de la posición fue tomada en 20cm. Esta decisión fue realizada teniendo en cuenta que el algoritmo soporta hasta 30cm y que el robot no se mueve más de 50cm entre iteraciones. De moverse 50cm, un error de 20cm representa un 40 %, lo que es un error considerable.

7.2.4.2. Ruido en el láser.

Como se mencionó anteriormente, el error que presenta el láser en el laberinto de 8 brazos radiales tiene características geométricas. Esta condición hace que sea altamente difícil replicar dicho error. De esta forma, si bien no es la mejor idea, se optó por agregarle ruido gaussiano acotado al sensor láser. Para elegir la magnitud del ruido, se compararon los resultados del algoritmo obtenidos por el robot real con los obtenidos en la sección 7.2.3. Los resultados más próximos se producen para las magnitudes entre 5 y 6cm, con lo cual, debido a que luego se probaría el algoritmo en laberintos más grandes, se decidió por la menor de las dos magnitudes (5cm).

Comparación resultados robot real y ruido láser						
	Error Posición (cm)			Error Dirección de la cabeza (º)		
	Promedio	Máx.	Std. Dev.	Promedio	Máx.	Std. Dev.
Robot real	2.9	6.8	1.4	0.6	2.5	0.5
Ruido láser 5 cm	2.5	6.1	1.4	1.0	2.2	0.6
Ruido láser 6 cm	2.4	6.3	1.6	1.3	3.7	0.9

Cuadro 7.5 – Elección de ruido en el láser

7.2.5. Evaluación general del SLAM

Para probar el algoritmo de SLAM más exhaustivamente, se generaron 12 recorridas del robot en un laberinto de 8 brazos radiales (esta vez con brazos de 2m de longitud y 0.5m de ancho), y 11[1] recorridas en uno de 10 brazos paralelos (con pasillos de iguales dimensiones al anterior, ver imagen 7.2.5). Cada recorrida realizada cubre entre 3 y 4 veces la totalidad del laberinto correspondiente y cuenta con entre 300 y 600 posiciones diferentes de donde se realizó el sensado. A su vez, cada recorrida se ejecuta 10 veces utilizando el ruido dado en la sección 7.2.4, variando la semilla de números aleatorios en cada iteración.

7.2.5.1. Métricas de evaluación.

Para evaluar el desempeño obtenido en ambos laberintos, nuevamente se utiliza como métrica el error máximo y el error medio de la posición y dirección de la cabeza. De cada ejecución se obtienen estos cuatro valores y luego, para cada métrica se haya su valor máximo, promedio y desviación estándar. Los resultados pueden ser observados en los cuadros 7.6 y 7.7. El cuadro correspondiente al laberinto de 10 brazos paralelos resume la información de las 110 ejecuciones totales realizadas. El cuadro correspondiente al laberinto de 8 brazos radiales solo resume la información de 100 de las 120 ejecuciones totales. En las otras 20 ejecuciones el algoritmo falló. Más de esto se habla en la sección 7.2.5.3.

Error en el laberinto de 8 brazos radiales			
ERROR	Promedio	Máx.	Std. Dev.
Pos - Máx (cm)	8.89	31.9	4.4
Pos – Promedio (cm)	3.7	13.9	1.92
HD – Máx (º)	2.9	7.3	0.9
HD – Promedio (º)	1.0	4.7	0.7

Cuadro 7.6 – Error en el laberinto de 8 brazos radiales.
En el cuadro se muestra el promedio, el máximo y la desviación estándar de las cuatro métricas obtenidas en cada ejecución del algoritmo de SLAM diseñado en el laberinto de 8 brazos radiales.

Error en el laberinto de 10 brazos paralelos			
ERROR	Promedio	Máx.	Std. Dev.
Pos - Máx (cm)	22.0	61.1	13.0
Pos – Promedio (cm)	8.3	21.7	4.8
HD – Máx (º)	3.8	7.7	1.3
HD – Promedio (º)	1.4	3.8	0.8

Cuadro 7.7 – Error en el laberinto de 10 brazos paralelos.
En el cuadro se muestra el promedio, el máximo y la desviación estándar de las cuatro métricas obtenidas en cada ejecución del algoritmo de SLAM diseñado en el laberinto de 10 brazos paralelos.

7.2.5.2. Efectos del ruido en los mapas obtenidos

Para ver como el ruido en los datos afecta el mapa resultante, para una recorrida específica de cada laberinto se muestran algunos de los mapas obtenidos al variar la semilla de números aleatorios. Esto puede dar una idea de la incidencia que tiene el ruido en el algoritmo y la capacidad que tiene este para manejarlo.

[1] Originalmente también eran 12, pero uno de los archivos generados estaba corrupto

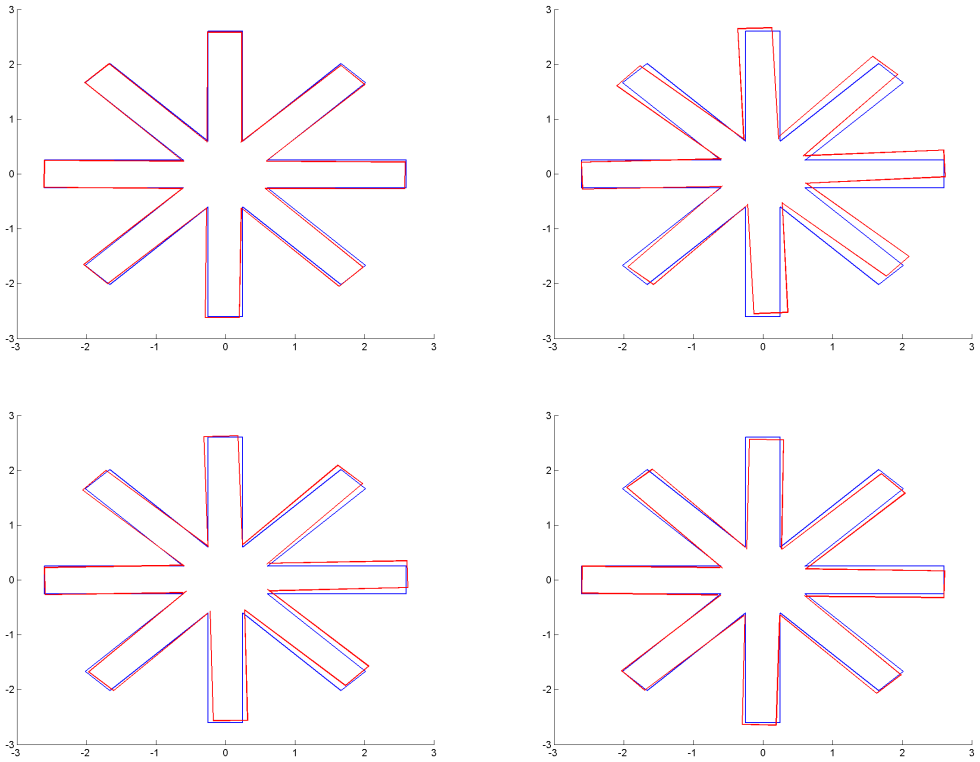

Figura 7.2.4 – Laberinto radial de 8 brazos - Diferentes semillas
En la figura se muestra el mapa resultante de una misma recorrida del laberinto de 8 brazos radiales, utilizando 4 semillas de números randómicos diferentes. En azul se muestran los mapas reales y en rojo los obtenidos por el algoritmo.

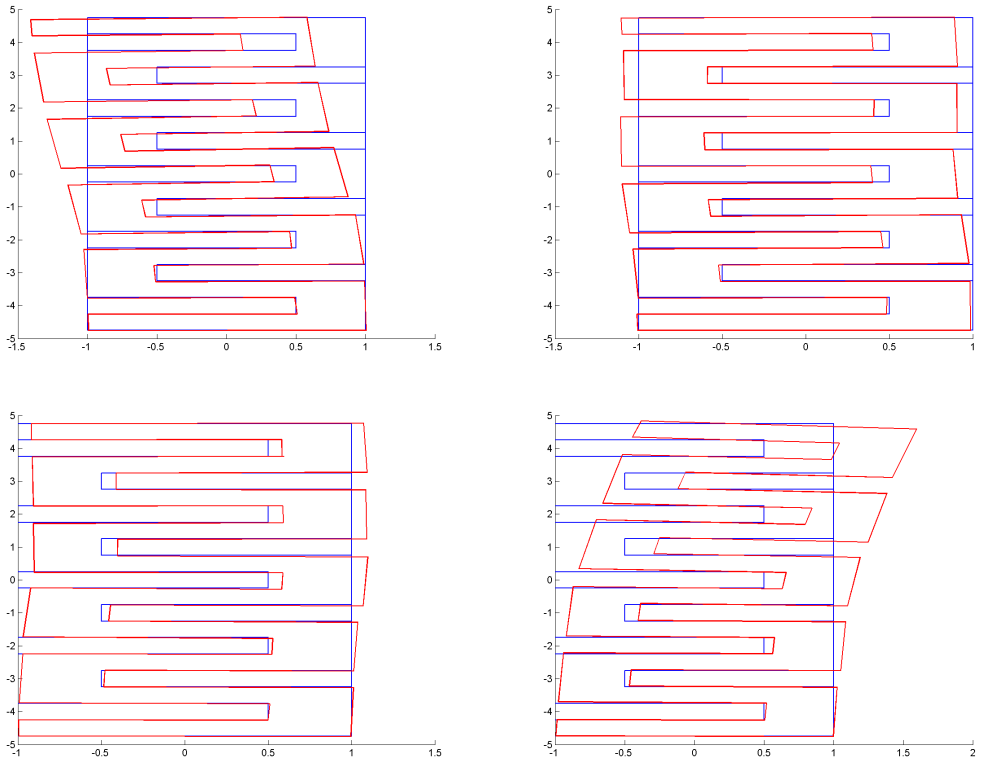

Figura 7.2.5 – Laberinto de 10 brazos paralelos - Diferentes semillas
En la figura se muestra el mapa resultante de una misma recorrida del laberinto de 10 brazos paralelos, utilizando 4 semillas de números randómicos diferentes. En azul se muestran los mapas reales y en rojo los obtenidos por el algoritmo.

En la figura 7.2.4 y en especial en la 7.2.5, se puede observar que el ruido en los datos tiene un gran efecto sobre los resultados del algoritmo. Los efectos son más notorios para el algoritmo de 10 brazos paralelos ya que la distancia que debe ser recorrida desde el inicio del laberinto (en la parte inferior) hasta el fin (en la parte superior) es aproximadamente 30 metros (2 metros por recorrer un brazo y 1 metro para pasar de uno a otro). En cambio, en el laberinto de brazos radiales, para viajar entre dos puntos cualquiera del laberinto se debe recorrer a lo sumo 5 metros aproximadamente (2 metros por brazo y uno para cambiar de brazo). De esta forma, los errores se acumulan más en el laberinto de 10 brazos paralelos que en el de 8 radiales.

7.2.5.3. Ejecuciones fallidas

EL algoritmo de SLAM diseñado falló en 20 ejecuciones, es decir, se vio interrumpido por un error. Estas 20 ejecuciones se corresponden con 2 recorridas particulares (que son ejecutadas con 10 semillas de números aleatorios diferentes) del laberinto de 8 brazos radiales. El fallo se vio dado ya que en una iteración el robot ve únicamente aristas nuevas. De esta forma, no es capaz de reconocer ninguna y el algoritmo falla ya que se asumía esto no puede pasar. Esto se solucionaría si el robot pudiera sensar y ejecutar el algoritmo desde posiciones intermedias entre la de la iteración del fallo y la anterior.

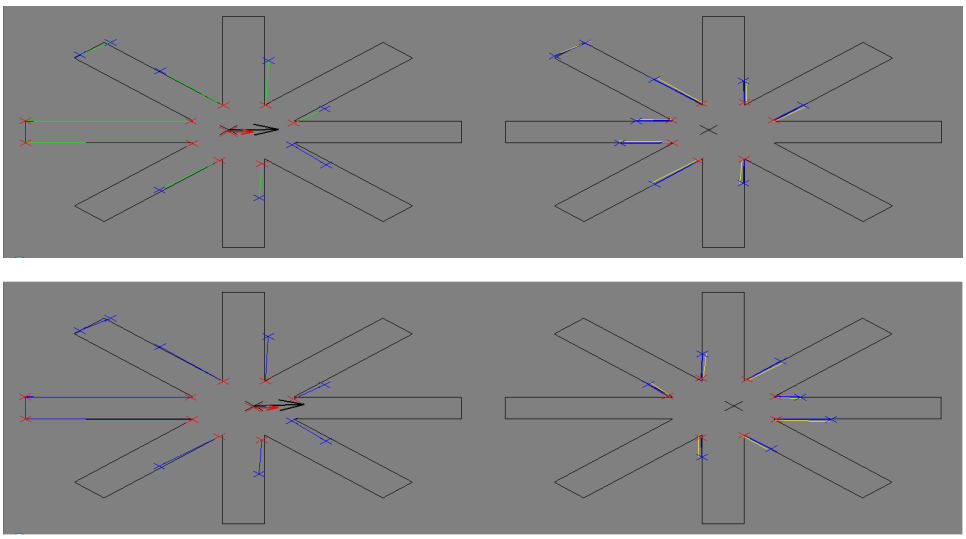

Figura 7.2.6 – Error de ubicación
En la figura superior se muestra una iteración del algoritmo SLAM, y en la figura inferior se muestra la misma ejecución pero una iteración después. En la parte izquierda de ambas imágenes, en color verde y azul se muestran las aristas pertenecientes al mapa. En la parte derecha, se muestra las aristas observadas en dicha iteración. Notar que las observadas en la segunda iteración no se corresponden con ninguna del mapa con lo cual se genera un fallo ya que el robot no es capaz de ubicarse con la información disponible.

7.2.6. Pruebas de evolución del error

Este tipo de pruebas se generaron para poder evaluar el comportamiento del algoritmo a través del tiempo. Esto involucró realizar recorridas del laberinto que cubran su totalidad varias veces. Por temas prácticos, se procedió a obtener esta iteración larga, concatenando 10 veces una de las corridas anteriores del algoritmo (las cuales recorrían el laberinto 3 o 4 veces), generando así una recorrida que transitara entre 30 y 40 veces cada brazo del laberinto. Esto fue posible ya que en las recorridas realizadas, se pretendió comenzar y terminar en el mismo punto previendo que de esa forma podrían ser concatenadas.

Los resultados de este experimento muestran que en el algoritmo diseñado los errores en la posición y en la dirección de la cabeza se mantienen relativamente constantes en el tiempo. Para ello, se grafican las posiciones desde las que se realizó el sensado y se le asigna un color acorde al error obtenido por el algoritmo en dicha posición. El resultado es que el error depende más de la zona que del tiempo.

Algo interesante que se observa en el experimento es que, una vez que el mapa está completo este no sufre grandes cambios, al menos no para la cantidad de vueltas dadas en el experimento. Y, por tanto, los errores tampoco sufren grandes cambios. En las figuras 7.2.7, 7.2.8, 7.2.9 y 7.2.10 se observa mejor los resultados.

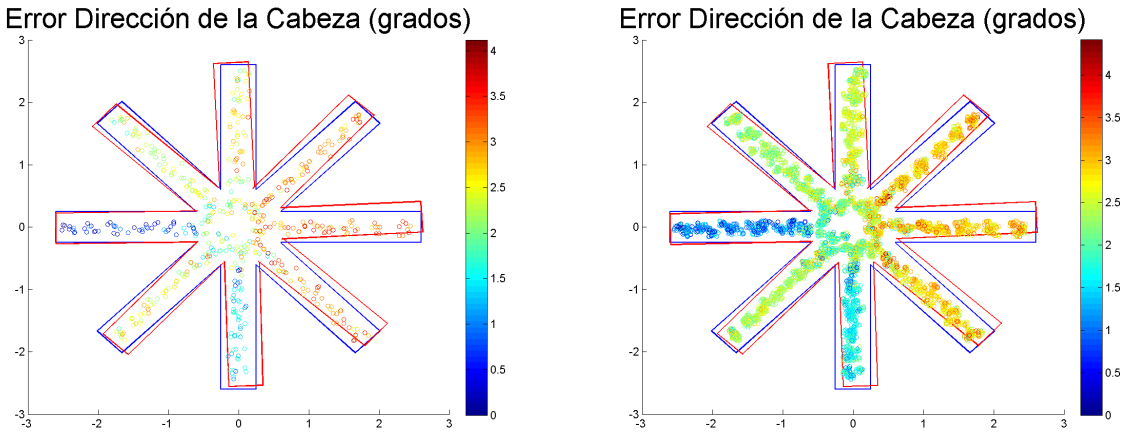

Figura 7.2.7 – Error en la dirección de la cabeza por zonas
En la figura se muestra el error en la dirección de la cabeza cometido desde diferentes ubicaciones. La imagen a la izquierda corresponde a una recorrida corta, mientras que la imagen a la derecha corresponde a la concatenación de dicha recorrida 10 veces. En ambas ejecuciones se utiliza la misma semilla de ruido.

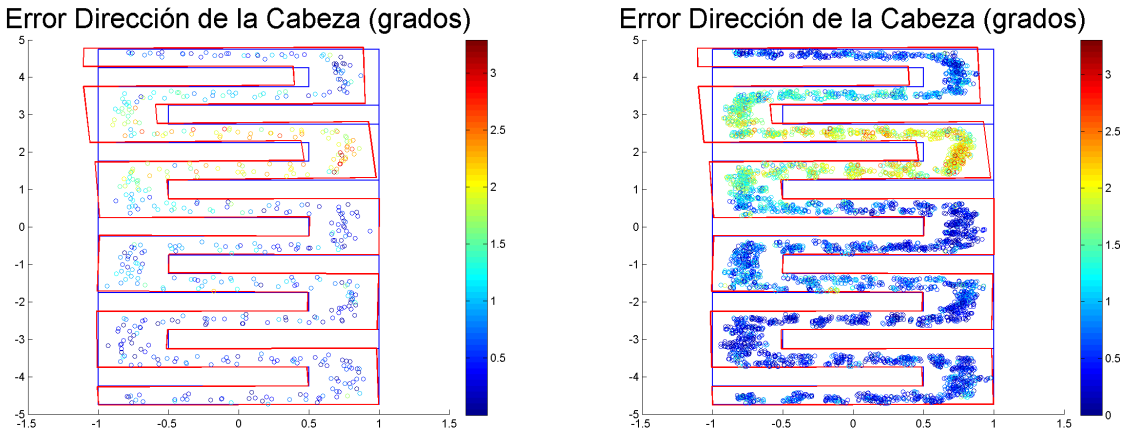

Figura 7.2.8 – Error en la dirección de la cabeza por zonas
En la figura se muestra el error en la dirección de la cabeza cometido desde diferentes ubicaciones. La imagen a la izquierda corresponde a una recorrida corta, mientras que la imagen a la derecha corresponde a la concatenación de dicha recorrida 10 veces. En ambas ejecuciones se utiliza la misma semilla de ruido.

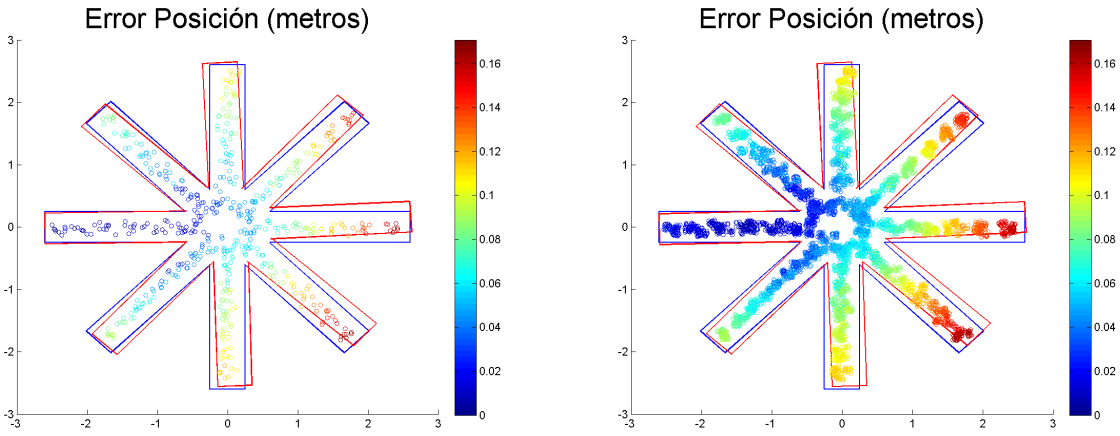

Figura 7.2.9 – Error en la posición por zonas
En la figura se muestra el error en la posición cometido desde diferentes ubicaciones. La imagen a la izquierda corresponde a una recorrida corta, mientras que la imagen a la derecha corresponde a la concatenación de dicha recorrida 10 veces. En ambas ejecuciones se utiliza la misma semilla de ruido.

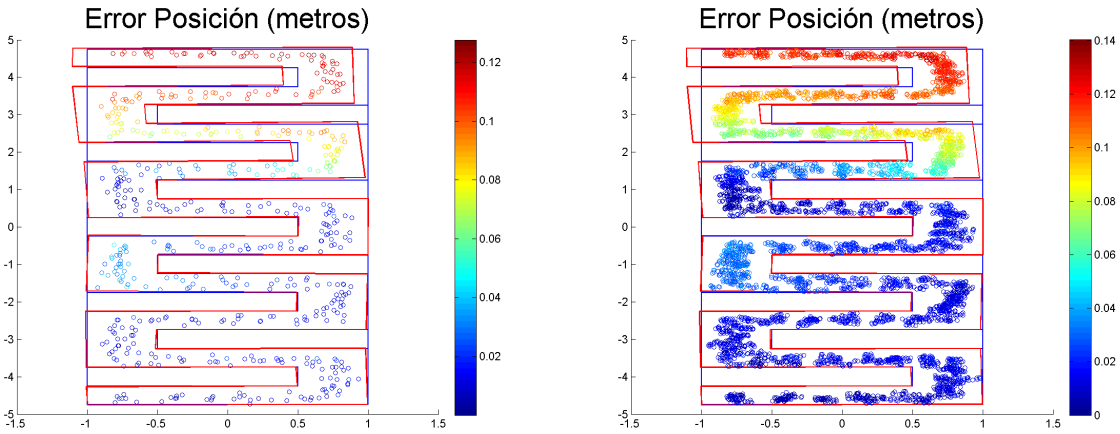

Figura 7.2.10 – Error en la posición por zonas
En la figura se muestra el error en la posición cometido desde diferentes ubicaciones. La imagen a la izquierda corresponde a una recorrida corta, mientras que la imagen a la derecha corresponde a la concatenación de dicha recorrida 10 veces. En ambas ejecuciones se utiliza la misma semilla de ruido.

7.3. Resultados del algoritmo de división del espacio.

Como se mencionó anteriormente, el objetivo del algoritmo de división del espacio es realizar una prueba de concepto. En esta, se pretende mostrar como el algoritmo de SLAM diseñado puede ser utilizado para obtener polígonos convexos que le sean de utilidad a los algoritmos TAM y WG. Se debe considerar que, al ser esta parte una prueba de concepto para la cual se dedicó un tiempo mínimo, el algoritmo aún requiere de muchas mejoras.

La evaluación de este algoritmo consiste en observar los polígonos obtenidos en los algoritmos pTAM y pWG para cuatro laberintos (laberinto de 8 brazos radiales, 10 brazos paralelos, tipo t y dona). Particularmente se observa si los polígonos obtenidos tienen las características deseadas mencionadas en el capítulo 5. En particular, para evaluar el pWG, se muestra los

polígonos finales obtenidos, mientras que para el pTAM , como no hay polígonos finales, se muestran los principales tipos de polígonos obtenidos durante la ejecución del algoritmo.

7.3.1. Resultados pTAM

A continuación se muestran los diferentes tipos de polígonos obtenidos por el algoritmo pTAM. Primero se muestra los polígonos por el algoritmo funcionando correctamente y luego se muestran los casos en los que no.

7.3.1.1. Funcionamiento correcto del algoritmo pTAM

En esta sección se muestra los resultados obtenidos por el algoritmo pTAM cuando este funciona correctamente. Como los polígonos generados se repiten en los diferentes escenarios (laberintos) ya que las situaciones son similares, solo se muestran las clases que sean más representativas.

De las ejecuciones realizadas del algoritmo es posible hacer algunas observaciones, las cuales se ven ejemplificadas en la figura 7.3.1.

Observaciones:

- En primer lugar, se comprueba que el algoritmo, como es de esperarse, efectivamente obtiene polígonos convexos.
- En segundo lugar, tomando como ejemplo a las imágenes a y f, se puede observar que desde las dos posiciones en que se encuentra el robot en el laberinto (las cruces negras o las rojas de mayor tamaño en la imagen[2]), los polígonos que se obtienen son diferentes. Si bien esto es de esperarse, idealmente, como ambas posiciones se encuentran en la misma región del laberinto, el polígono obtenido debería ser el mismo. Esta claro esto no es posible ya que el resultado depende de las aristas extraídas que dependen, entre otros factores, de la posición del robot en el laberinto.
- En tercer lugar, en especial en los casos que se tiene un solo punto no convexo los resultados generan zonas que no son las "divisiones lógicas" que se realizan del espacio. A modo de ejemplo, en la imagen b, la "división lógica" elegiría el brazo inferior como el polígono convexo y no el seleccionado por el algoritmo. Nuevamente, si esto podría significar un problema para el uso del polígono en el TAM, no es posible decirlo de antemano.

[2]Las cruces negras representan la posición real, las cruces rojas de mayor tamaño representan la posición obtenida por el algoritmo de SLAM.

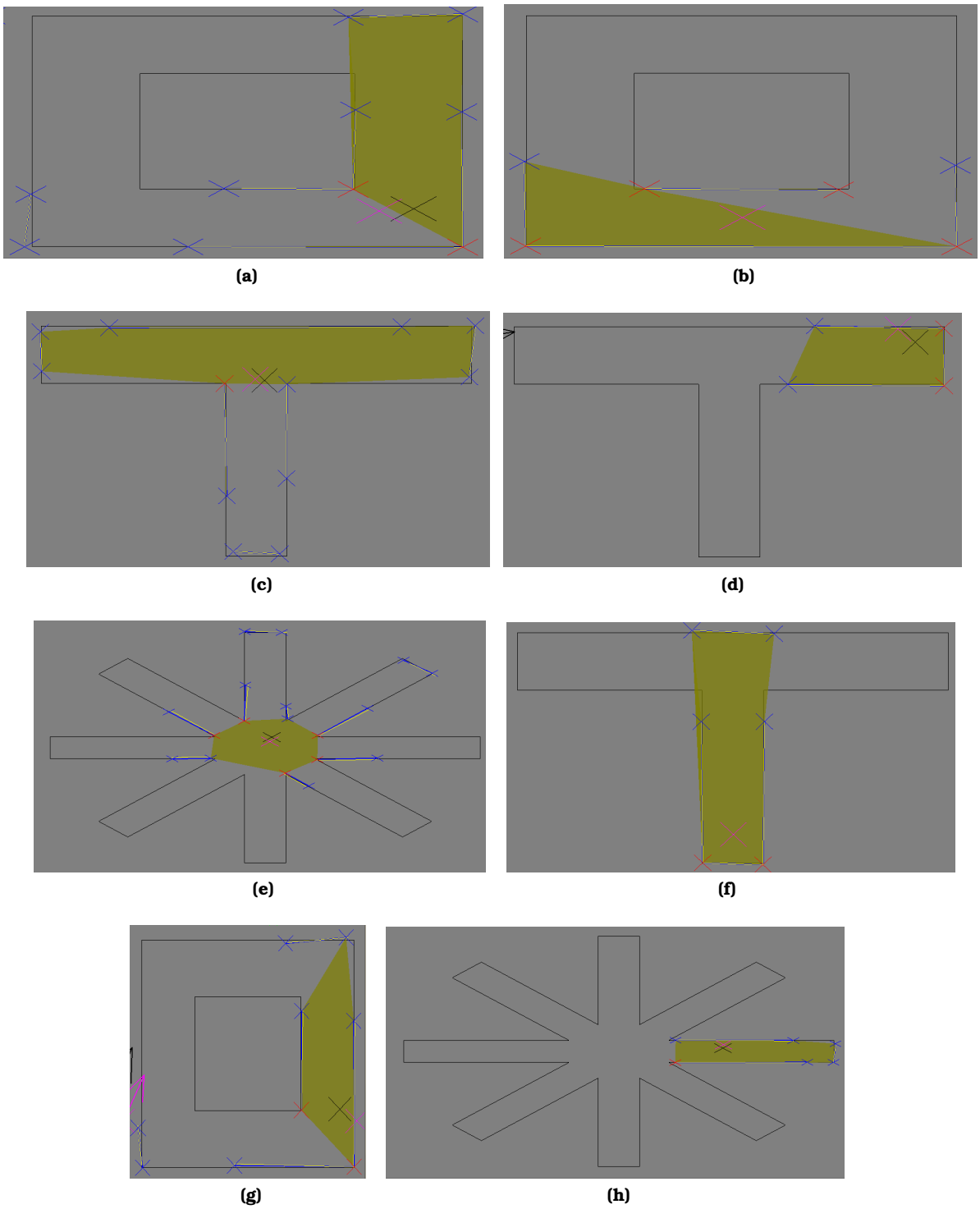

Figura 7.3.1 – Polígonos más representativos de pTAM.
En la figura se observan las clases de polígonos más comunes obtenidas por el algoritmo pTAM. Estas situaciones se ven repetidas en las diferentes ambientes en los que ejecuta el robot. En las imágenes, las cruces chicas rojas representan puntos en el mapa que son considerados como esquinas de una pared. Las chicas negras son los bordes de los segmentos que aún no se sabe si se corresponden con una esquina o no. De las cruces grandes, la negra representa la posición real del robot, la violeta es la posición dada por PI y la roja es la dada por el algoritmo de SLAM. Si alguna no puede ser observada es porque está superpuesta con otra.

7.3.1.2. Funcionamiento incorrecto del algoritmo pTAM

En algunas ocasiones, el algoritmo pTAM obtuvo como resultado polígonos que no contuvieran a la posición del robot, o que no representaran una porción del laberinto. Este problema principalmente a que los polígonos se crean a partir de los segmentos extraídos en el algoritmo de Hough modificado. Cuando el algoritmo es incapaz de crear aristas ya sea porque los datos tienen demasiado ruido o porque no se cuenta con una resolución de puntos suficiente, el polígono resultante se ve afectado.

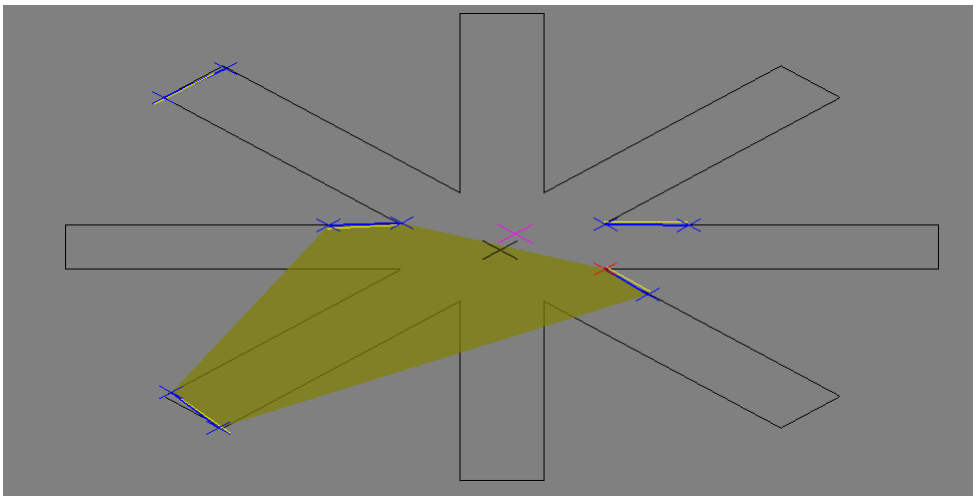

Figura 7.3.2 – Errores en pTAM.
En la imagen se observa el polígono obtenido por el algoritmo pTAM cuando el algoritmo de Hough no genera las aristas necesarias. En las imagen, las cruces chicas rojas representan puntos en el mapa que son considerados como esquinas de una pared. Las chicas negras son los bordes de los segmentos que aún no se sabe si se corresponden con una esquina o no. De las cruces grandes, la negra representa la posición real del robot, la violeta es la posición dada por PI y la roja es la dada por el algoritmo de SLAM. Si alguna no puede ser observada es porque está superpuesta con otra.

7.3.2. Resultados pWG

A continuación se muestran las particiones obtenidas en el algoritmo pWG en los diferentes laberintos para los casos que funcionó correctamente el algoritmo.

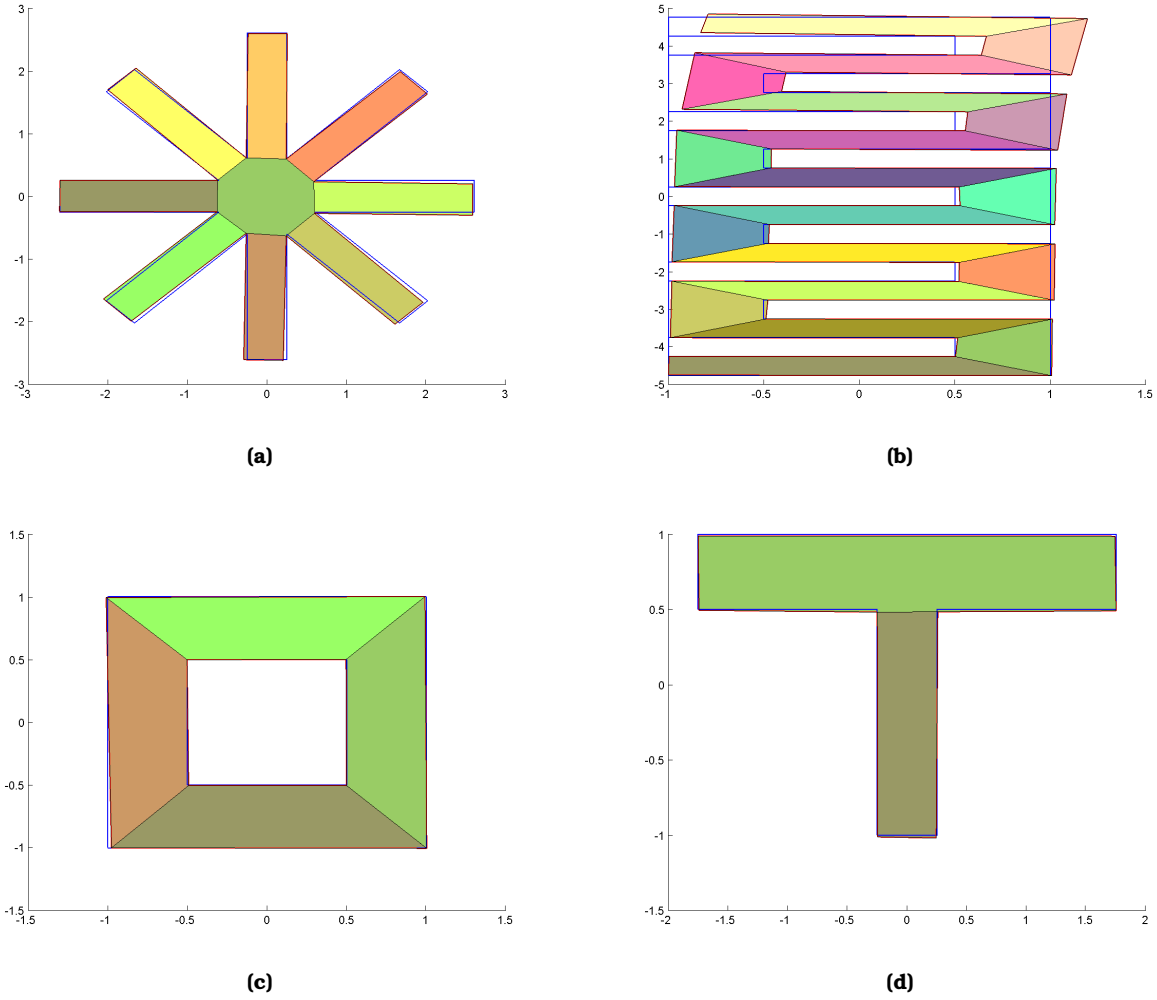

Figura 7.3.3 – Resultados pWG
En las figuras a,b,c y d se muestran las particiones finales de polígonos en el algoritmo pWG para los laberintos de 8 brazos radiales, 10 brazos paralelos, dona y tipo T respectivamente. En la imagen solo se muestra las particiones cuando el algoritmo funcionó correctamente.

Como se observa en la figura 7.3.3, en los casos en que el algoritmo funcionó correctamente, los polígonos resultantes conforman una partición del mapa en convexos. Estos polígonos cumplen con la característica deseada de que el espacio no se divida innecesariamente. A modo de ejemplo, el laberinto tipo T se divide únicamente en dos polígonos, cantidad mínima para particionar la T en convexos.

7.3.2.1. Errores en pWG

Los casos en que el algoritmo pWG ejecutó incorrectamente se debe principalmente a dos causas. Una de ellas es la presencia de aliases ya sea de puntos o aristas en el mapa.[3] La segunda cause de errores, son los errores provenientes de los módulos que utiliza el pWG, en particular los errores provenientes del pTAM. Nuevamente, como, el módulo implementado simplemente es una prueba de concepto no se implementó tolerancia a errores de este tipo.

Los errores encontrados en el modelo fueron diversos. Para nombrar algunas de ellas, se encontraron divisiones del mapa que no conforman particiones, oscilaciones en las particio-

[3]Recordar que un punto o arista tiene un alias si es representdo por dos objetos diferentes en el mapa que no son reconocidos como el mismo.

nes generadas, polígonos cuyas aristas atraviesan paredes, regiones del espacio que se ven representadas por más de un polígono, etc.

A continuación se presenta una imagen que ejemplifica dos de estos casos.

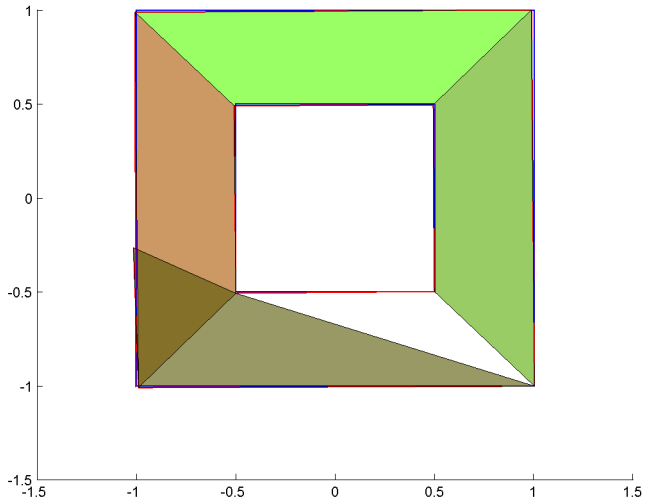

Figura 7.3.4 – Error en pWG

En la figura se observa el resultado del algoritmo pWG para una de la ejecuciones realizadas en el laberinto con forma de dona. El resultado no conforma una partición del espacio. En total se observan 5 polígonos (4 trapezoides y 1 triángulo) que no constituyen una partición. La linea vertical del triángulo está conformada por el alias de un segmento de la pared izquierda.

Capítulo 8

Conclusiones, observaciones y trabajo futuro.

8.1. Generales

Este trabajo estuvo orientado desde su origen por dos objetivos. En primer lugar, el de relevar el estado del arte en aprendizaje bio-inspirado centrándose preferentemente en algoritmos de aprendizaje por refuerzo. En segundo lugar, por el de diseñar e implementar un algoritmo de aprendizaje sobre alguna de las plataformas robóticas disponibles.

Durante el desarrollo del primer objetivo, se pudo detectar que la información disponible es de difícil acceso, y que la misma presenta carencias en detalles relevantes al momento de intentar implementar una solución similar. De esta forma, la tarea adquirió una dimensión que excedía al alcance del proyecto, por lo que se limitó a crear una estructura de módulos que dejaran encaminada la futura implementación del modelo TAM-WG.

A diferencia de SL, al estudiar el estado del arte de SLAM, se descubrió que existe mucha información disponible y muy variada, por lo que el desafío en ese sentido, es lograr encontrar una referencia que se adapte bien a la metodología deseada y a las características del hardware disponible, ya sean sensores, actuadores, etc.

8.2. SLAM, pTAM y pWG

El algoritmo de SLAM implementado, diseñado para trabajar en laberintos de paredes planas, permite construir un mapa métrico del espacio así como también ubicarse en él. Los algoritmos pTAM y pWG muestran además como puede ser usada esta información para generar una partición convexa del espacio.

A partir de las experimentaciones realizadas se obtienen las conclusiones expuestas en las siguientes secciones.

8.2.1. Principales causas de error en SLAM

De las secciones que evalúan la tolerancia del ruido y sus efectos en el mapa, se observa que el resultado del algoritmo se ve fuertemente influenciado por el introducido en el sensor láser.

En particular, en la primera de las dos secciones, se puede observar cómo el incremento del ruido en el láser va degradando los resultados en las métricas de evaluación obtenidas hasta el punto, en que el algoritmo deja de funcionar. Por otro lado, en la otra sección, se observa que, especialmente en el laberinto de 10 brazos paralelos, al cambiar la semilla de ruido, el mapa resultante varía de forma muy notoria.

Teniendo en cuenta esto, los resultados podrían mejorarse de las siguientes formas.

- **Disminuyendo el error en el láser**
- **Incrementando la resolución del láser** - Es decir, disminuir el ángulo para el cual se toman medidas láser. Por ejemplo, en vez de tomar una medida cada 3.6 grados, se podría tomar una cada 1 grado. De esta forma, se obtendría una mayor cantidad de puntos por recta observada disminuyendo así los errores en la regresión realizada.
- **Mejorando el algoritmo de extracción de segmentos -** Posibles mejoras de esto se comentaron en la sección 4.2.2.6

8.2.2. Evolución del mapa en el tiempo.

Como se observaba en la sección 7.2.6, una de las cosas que se puede apreciar es que una vez creado el mapa, este no mejora con el tiempo, o al menos no lo hace para los experimentos efectuados. Este hecho indica que no se tiene un buen método de integración de la información ya que sería deseable que, al ir promediando observaciones, el mapa generado mejore (o al menos cambie).

Como se mencionó en los capítulos de implementación, la forma actual de integrar la información de los segmentos consiste en ir calculando el centro de masa de sus puntos, e ir promediando las direcciones bajo las que fue observado.

En vista de que lo anterior no logra el efecto deseado se propone un cambio al método que se deja como trabajo a futuro. Para definir la dirección de una arista, se podría generar un árbol de dependencia entre ellas, en el cual cada nodo (salvo la raíz) almacenaría la dirección relativa a su padre. Este valor sería modificado únicamente cuando se observan ambas aristas al mismo tiempo, en cuyo caso, sería reemplazado mediante algún promedio ponderado entre el valor del nodo actual y el valor observado, o algún mecanismo semejante. Al modificar el valor de un nodo, se debería actualizar la dirección de su segmento correspondiente y el de todos sus descendientes usando la fórmula $theta_{hijo} = \theta_{padre} + \theta_{relativo}$. De esta forma, si bien puede que sea un poco más costosa computacionalmente, se esperaría que esto haga que los ángulos entre segmentos tiendan a sus valores reales.

8.2.3. pTAM y pWG

A partir de los resultados vistos en el capítulo 7, queda claro que los algoritmos pTAM y pWG requieren aún de varias mejoras. En particular los problemas observados provienen de dos fuentes.

La primera de ellas se ejemplifica en la figura 7.3.2. En ella se observa que el algoritmo de Hough fue incapaz de extraer todas las aristas que podrían ser vistas desde la posición en que se encuentra el robot. Para ese caso, el resultado es un polígono cuyas aristas atraviesan las paredes del laberinto, pero en otros casos, también se observó que el polígono obtenido puede no contener a la posición del robot. Cualquiera de estos casos produce comportamientos inesperados en la partición del mapa o en los polígonos obtenidos por el pTAM.

Por otro lado, la segunda fuente de problemas es la existencia de aliases de puntos en el mapa. Es decir cuando un punto es representado por más de un objeto sin darse cuenta que son el mismo. El problema que esto trae aparejado es que el algoritmo de reconocimiento de polígonos hace uso de una tabla que, dados dos puntos orientados, se devuelve el polígono al que pertenecen. De esta forma, si se usa un alias para indexar la tabla, el reconocimiento puede fallar.

Antes de que los algoritmos puedan ser usados correctamente, los problemas mencionados han ser resueltos. Como el objetivo de ellos en el proyecto solo era realizar una prueba concepto, el planteo de correcciones y mejores análisis del algoritmo se dejan como trabajo futuro.

8.3. Próximos pasos

Como se mencionó anteriormente, el objetivo real del proyecto era implementar un algoritmo de TAM-WG. Para esto, se desarrollaron tres algoritmos (SLAM métrico, pTAM, pWG) con intenciones de usarlos luego para resolver el problema original. Debido al tiempo limitado en que se enmarca el proyecto de grado y debido a las dimensiones de los problemas abarcados, se terminó optando por acotar el alcance del proyecto únicamente a estos problemas.

Teniendo esto en cuenta, se tiene dos grandes pasos a seguir. El primer paso sería seguir desarrollando y mejorando los algoritmos pTAM y pWG. En particular, para que los algoritmos puedan ser utilizados en el TAM-WG, primero sería necesario corregir los errores mencionados en las secciones anteriores. Finalizado este paso, se debería proceder a implementar el algoritmo TAM-WG. Para ello, podrían implementarse los modelos TAM y WG por separado parra luego unirlos en uno.

Apéndice A

Herramientas matemáticas

En esta sección se pretende presentar algunas herramientas matemáticas que fueron utilizadas en el desarrollo del algoritmo. Algunas herramientas no fueron utilizadas para SLAM sino en en un algoritmo de división del espacio, sin embargo, por cuestiones de ordenamiento todas se presentan en esta sección.

A.1. Ecuaciones de rectas y propiedades

A.1.1. Ecuación general de la recta

De álgebra linea se sabe que cualquier recta en $\mathbb{R}^2$puede ser escrita mediante la ecuación: $aX + bY = c$, o equivalentemente en su versión vectorial: $\vec{v} \cdot \vec{x} = c$, en donde $\vec{v} := (a, b)$ y $\vec{x} := (X, Y)$

A.1.2. Ecuación polar de la recta

A partir de la ecuación general se puede observer que dividiendo toda la ecuación por $\|\vec{v}\|$ se obtiene $\vec{p} \cdot \vec{x} = d$ en donde $\vec{p} := \frac{\vec{v}}{\|\vec{v}\|}$ y $d := \frac{c}{\|\vec{v}\|}$. Observar que en esta forma se tiene $\|\vec{p}\| = 1$ por lo que existe θ tal que $\vec{p} = (cos(\theta), sin(\theta))$, y abusando un poco la notación podemos escribir $\vec{p} = e^{i\theta}$. La ecuación $\vec{x}e^{i\theta} = d$ es lo que se conoce como ecuación polar o ecuación normal de la recta.

Esta ecuación tiene varias propiedades e implicancias importantes dentro de las que se comentan y demuestran algunas a continuación.

Propiedad 1 - Toda recta tiene exactamente dos ecuaciones polares. Si (d, θ) son los parámetros de una de las ecuaciones de la recta, es fácil observar la otra viene dada por $(-d, \theta + 180)$ y que no hay más posibilidades. Esto es trivial multiplicando la ecuación polar por -1, observando $e^{i(\theta+180)} = -e^{i\theta}$, y observando que como $\|\vec{p}\| = 1$, no es posible multiplicar por constantes diferentes de 1 y -1.

Propiedad 2 - El vector $\vec{p}$ es perpendicular a la recta. Para demostrar esto basta tomar dos puntos diferentes $\vec{x}_1$, $\vec{x}_2$ pertenecientes a ella y observar que $\vec{v} := \vec{x}_1 - \vec{x}_2$ es la dirección de la recta. Luego se tiene $\vec{p} \cdot \vec{v} = \vec{p} \cdot (\vec{x}_1 - \vec{x}_2) = \vec{p} \cdot \vec{x}_1 - \vec{p} \cdot \vec{x}_2 = d - d = 0$, lo que demuestra $\vec{v}$ y $\vec{p}$ son perpendiculares.

Propiedad 3 - El punto $d\vec{p}$ es la proyección perpendicular del origen en la recta. Para probar esto, observese que como $\vec{p} \cdot (d\vec{p}) = d\|\vec{p}\|^2 = d \cdot 1 = d$ se tiene que el punto $d\vec{p}$ pertenece a la recta ya que cumple con su ecuación. Observar además que $d\vec{p}$ es un escalar por $\vec{p}$, lo que implica que también pertenece a la recta de dirección $\vec{p}$ que pasa por el origen. De esta forma, utilizando la propiedad anterior, se tiene que $d\vec{p}$ es intersección de la recta $\vec{p} \cdot \vec{x} = d$ con

la perpendicular que pasa por el origen, lo que en otras palabras significa es la proyección perpendicular.

Propiedad 4 - d es la distancia signada del origen a la recta. Esta propiedad se desprende trivialmente de la anterior recordando que la distancia de un punto a una recta es la distancia de ese punto a su proyección perpendicular en la recta. De esta forma se tiene $\|d\vec{p} - 0\| = |d| \cdot \|\vec{p}\| = |d|$.

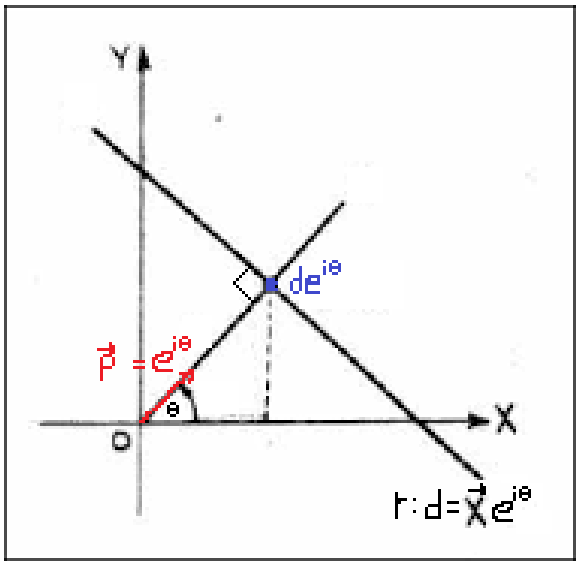

Figura A.1.1 – Ecuación polar de una recta y propiedades.
En la figura se puede observar las propiedades anteriores aplicadas a la recta r.

Propiedad 5 - La ecuación polar establece una función sobreyectiva entre $[0, 2\pi)$ x $\mathbb{R}$ y las rectas en $\mathbb{R}^2$. Esta propiedad es una consecuencia directa de la ecuación. Dado una dupla (θ, d) la función la asigna la recta $\vec{x}e^{i\theta} = d$. La sobreyectividad se debe a que para toda recta existen parámetros (θ, d).

Propiedad 6 - Función rectas de un punto - Dado $\vec{x}_1 \in \mathbb{R}^2$, todas las rectas que pasan por el punto quedan representadas por el grafo de la función $f : [0, 2\pi] \to \mathbb{R}$ con $f(\theta) := \vec{x}_1 \cdot e^{i\theta}$. Nuevamente, esta es una propiedad trivial. Basta recordar que el grafo de una funcion f se define como el conjunto $\{(x, f(x)) \ / \ x \in dm(f)\}$. Esta claro que todo punto del grafo representa una recta utilizando la correspondencia de la propiedad 5, y esta pasa por $\vec{x}_1$ por la definición de f. A su vez, dada un recta que pase por $\vec{x}_1$ si θ es la dirección de la recta perpendicular entonces, por la propiedad 2, $d = \vec{x}_1 \cdot e^{i\theta} = f(\theta)$.

Es importante observar que si bien la función es sobreyectiva, no es inyectiva ya que toda recta que pase por $\vec{x}_1$ tendrá dos representaciones en el grafo de f. Para hacer la función inyectiva se debería acotar el dominio a un itervalo $[\theta_0, \theta_0 + 180]$.

Si se utliza las coordenadas polares del punto, no es difícil observar que la función puede ser expresada como una sinusoide. Si $\vec{x}_1 = \rho e^{i\phi}$ entonces su función de rectas sera $f(\theta) = \rho \cdot cos(\theta - \phi)$. La demostración es muy sencilla: $f(\theta) = \vec{x_1} \cdot e^{i\theta} = (\rho \cdot (cos(\phi), sin(\phi)) \cdot (cos(\theta), sin(\theta)) = \rho \cdot (cos(\phi)cos(\theta) + sin(\phi)sin(\theta)) = \rho \cdot cos(\theta - \phi)$. La última igualdad se deduca de la propiedad del coseno para la súma/resta.

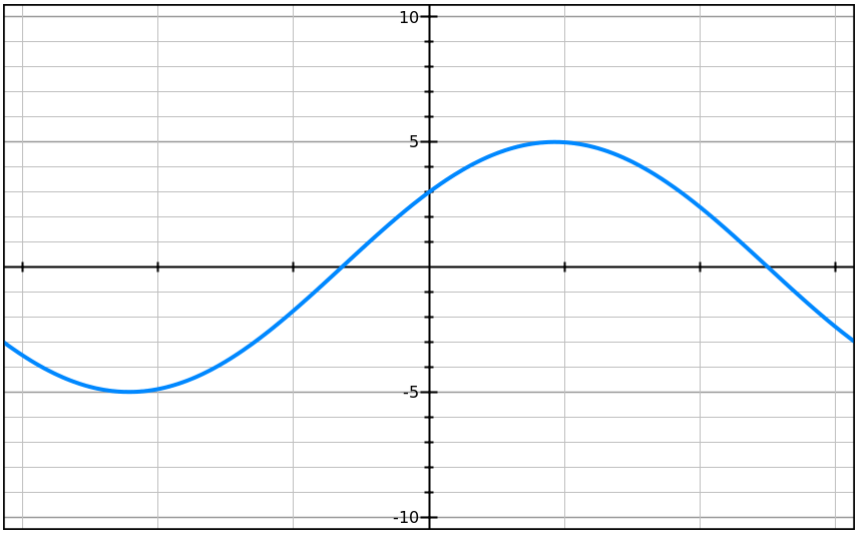

Figura A.1.2 – Gráfica de la función de rectas.
En esta figura se puede observar como queda la función de rectas para el punto . El eje representa a entre y el eje representa a .

A.1.3. Representación normal o polar de segmentos.

La representación normal o polar de segmentos es un nombre inventando en el proyecto que se le dio a la representación utilizada. El nombre se debe a que la representación utiliza los conceptos de la ecuación polar de la recta. Esta representación tendrá la ventaja de ser sencilla y facilmente manipulable. En ella un segmento orientado $\vec{x}_1, \vec{x}_2$ es representado como una tupla (θ, y, x_1, x_2) donde θ es la dirección del vector $\vec{x}_2 - \vec{x}_1$, $y := e^{i(\theta+\frac{\pi}{2})} \cdot \vec{x}_1$, $x_1 := e^{i\theta} \cdot \vec{x}_1$, y $x_2 := e^{i\theta} \cdot \vec{x}_2$. Notar que θ es la dirección del segmento y los vectores auxiliares $e^{i\theta}$ y $e^{i(\theta+\frac{\pi}{2})}$ forman una base ortonormal de $\mathbb{R}^2$ que coinciden con la dirección del segmento y su perpendicular. Luego, las coordenadas y, x_1 y x_2 son las coordenadas de los puntos en esa base.

Para rotar el segmento desde el origen de coordenadas simplemente basta hacer el cambio $\theta \leftarrow (\theta + \Delta\theta)$. Y, para trasladarlo, θ se mantiene constante y se hace el cambio $(y, x_1, x_2) \leftarrow ((y, x_1, x_2) + \vec{\Delta v}(e^{i(\theta+\frac{\pi}{2})}, e^{i\theta}, e^{i\theta})$. Si se desea realizar una rotación sobre un punto arbitrario $\vec{x}$, esta se puede hacer en tres partes, primero se traslada para que el origen quede en $\vec{x}$, luego se rota y finalmente se deshace el traslado. Los resultados serán: $\theta \leftarrow (\theta + \Delta\theta)$ y $(y, x_1, x_2) \leftarrow ((y, x_1, x_2) - \vec{x}(e^{i(\theta+\frac{\pi}{2})}, e^{i\theta}, e^{i\theta}) + \vec{x}(e^{i(\theta+\frac{\pi}{2}+\Delta\theta)}, e^{i(\theta+\Delta\theta)}, e^{i(\theta+\Delta\theta)})$.

Otra ventaja de esta representación es que permite ubicar fácilmente otros puntos en relación al segmento. Por ejemplo, es fácil saber si estan en que semiplano estan, si se encuentran más adelante o más atras en la dirección del segmento, etc. A modo de ejemplo, esta representación permite saber fácilmente si un punto cae dentro de un poligono convexo. Basta fijarse que el punto siempre quede dentro del semiplano interior de cada segmento.

A.2. Transformada de Hough[49, 50]

Originalmente, la "Transformada de Hough" es una herramienta que permite detectar rectas en una imagen. La misma fue generalizada luego en lo que se conoce como "Transformada de Hough Generalizada". Con esta última es posible detectar diversos tipos de curvas o figuras. Esta una técnica muy robusta frente al ruido y la existencia de huecos en la frontera de los objetos. Si bien en este proyecto las rectas fueron extraídas a partir de nubes de puntos más que de imágenes, los algoritmos son similares y comparten las mismas ideas.

A.2.1. Idea general

La transformada de Hough hace uso de la ecuación polar de las rectas. La idea es que si se tienen varios puntos de una misma recta, todas las funciones de rectas asociadas a esos puntos se intersectarán en exactamente 1 o 2 puntos dependiendo si el dominio integra la mitad del período o el período completo. La figura A.2.1 ilustra mejor este hecho.

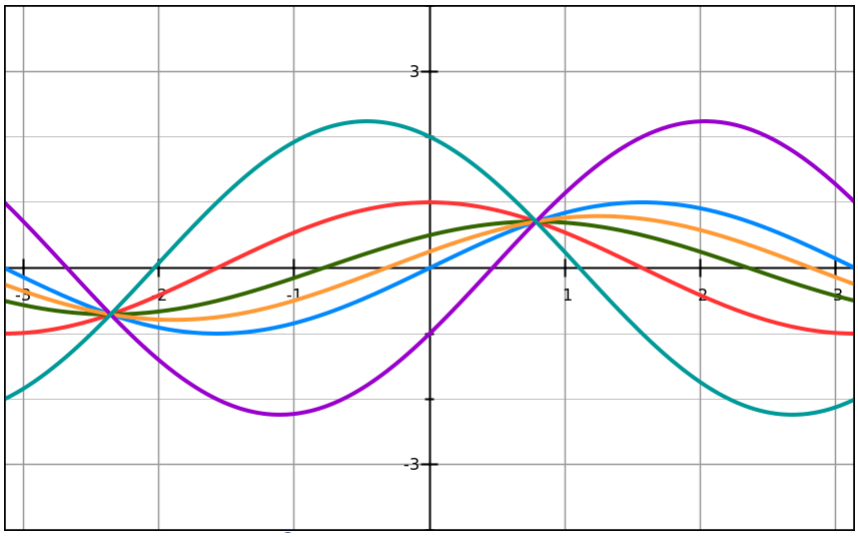

Figura A.2.1 – Transformada de Hough - Una recta sin ruido.
La figura muestra la funciones de rectas asociadas a los puntos $(2,-1)$, $(1,0)$, $(0{,}5,0{,}5)$, $(0{,}25,0{,}75)$, $(0,1)$, y $(-1,2)$. Estos conforman la recta $X+Y=1$, o equivalentemente, $\vec{x}\cdot e^{i\frac{\pi}{4}}=\frac{\sqrt{2}}{2}$. Como es de esperase, las curvas se intersectan en los puntos $(\frac{\pi}{4},\frac{\sqrt{2}}{2})$ y $(\frac{-3\pi}{4},-\frac{\sqrt{2}}{2})$ que representan la recta anterior.

Resolver estas gráficas no presenta grandes problemas cuando se tiene una sola recta, sin embargo, el problema se complica cuando se tienen varias o cuando los puntos tienen ruido.

Dados dos puntos, sus gráficas correspondientes se intersectarán en dos posiciones ya que dos puntos son suficientes para definir una recta. De esta forma cuando se está en el primer caso problemático las gráficas se intersectan en muchos puntos que no representan las rectas originales. A modo de ejemplo considerese 3 puntos no alineados definiendo únicamente dos rectas. Si se observaran las gráficas, se encontraría 6 intersecciones y 3 rectas posibles. Con esta información no sería posible decir cuales eran las rectas reales.

Similarmente, el otro problema al ingresar ruido modifica los lugares donde se intersectan las curvas. Esto hace que las curvas estén deformadas y corridas y dificulta la decisión de donde se intersectan realmente.

En un caso real se tiene ambos problemas lo cual dificulta aún más la extracción de las rectas. Para que este problema sea resoluble lo que se necesita es que se cuente con muchas más muestras formando rectas reales que formando rectas falsas. Volviendo al caso del triangulo, si en las dos rectas reales se tienen 10 puntos en cada una en vez de solamente 2, entonces ahora si será posible distinguir cuales son las rectas reales ya que las falsas tendrán aproximadamente 2 puntos cada una mientras que los reales tendrán cerca de 10 (dependiendo si se cuenta con ruido o no). De esta forma, lo que se usa como estrategia para extraer las rectas es buscar los centros de mayor densidad de la gráfica y utilizar algún umbral para definir cuando una recta es verdadera o falsa.

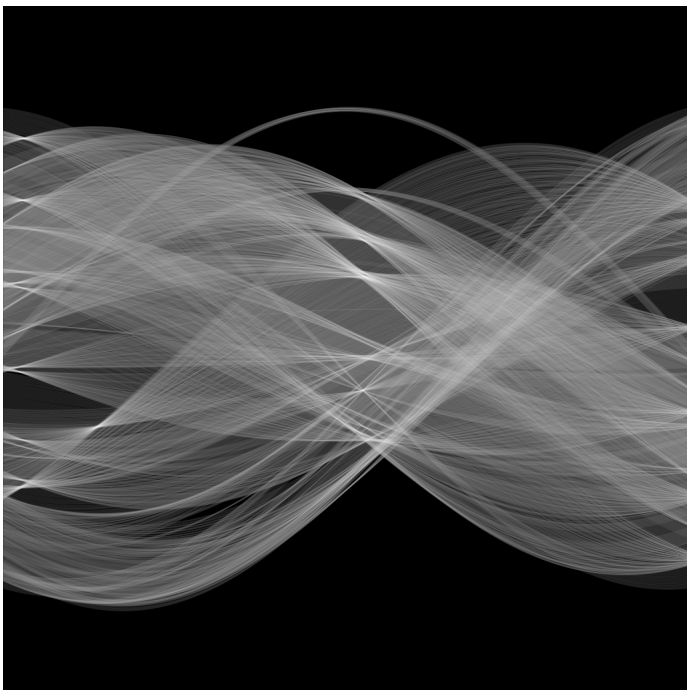

Figura A.2.2 – Transformada de Hough - Múltiples rectas con ruido.
La figura muestra un caso de la Transformada de Hough en el que se cuenta con varias lineas y ruido. Los regiones más luminosas indican que la densidad es mayor en ellas.[51]

A.3. Convexidad

Debido a que lo que se presenta son herramientas matemáticas, esta sección se estructura como una aglomeración ordenada de definiciones y proposiciones.

A.3.1. Definiciones

Definición: Convexidad en $\mathbb{R}^2$

Un conjunto $C \in \mathbb{R}^2$ se dice convexo si para todo $x_1, x_2 \in C$ se tiene el segmento $[x_1, x_2] \subset C$.

En particular, en este proyecto que un conjunto sea convexo implicará que el agente pueda trasladarse de forma directa de un punto al otro sin la necesidad de cambiar de dirección (tomar nuevas decisiones) una vez que se haya seteado el rumbo.

Definición: Polígonos simples, complejos y con agujeros.

Un polígono simple se define como una curva cerrada de segmentos ordenados tal que ningún par de segmentos no adjuntos se intersectan. Si se intersectan al polígono se le llama complejo. Un polígono se dice que tiene un agujero cuando la región exterior no es conexa. Estos polígonos no son simples ya que no se definen como una curva cerrada. Más bien son un polígono al que se le ha restado un conjunto de poligonos contenidos en su interior. A modo de ejemplo, supongamos que x_1, x_2, x_3, x_4 sea un cuadrado. Ese polígono es un polígono simple ya que forman una curva cerrada y sus lados no adyacentes no se autointersectan, además no contiene agujeros ya que la región exterior es conexa. Por el otro lado, utilizando los mismos vértices, el polígono x_1, x_3, x_2, x_4 no es simple ya que los lados $[x_1, x_3]$ y $[x_2, x_4]$ se intersectan en el centro del cuadrado. Finalmente, si al cuadrado se le quita otro cuadrado concéntrico más pequeño la figura resultante será un polígono simple con un agujero.

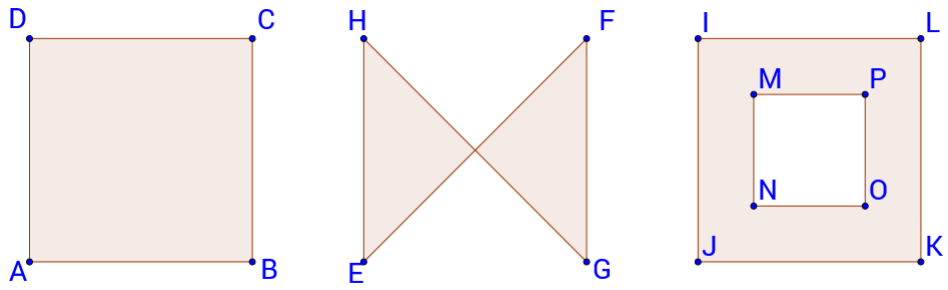

Figura A.3.1 – Tipos de polígonos.
En la figura, el polígono de la izquierda es un polígono simple,. el del medio es complejo, y el de la derecha es un polígono con un agujero.

Definición: Conjunto Gráfica Polar

Un conjunto $P \in \mathbb{R}^2$ se dice que es una gráfica polar $\iff \exists x \in P$ / la frontera de P puede escribirse cómo una gráfica polar con centro de coordenadas en x.

En particular los conjuntos convexos y la nube de puntos que obtendrá el robot serán gráficas polares.

Definición: Ángulo y vértice convexo.

Se dice que un ángulo es convexo si solo si es menor o igual a 180 grados. En el caso de un polígono, se dice que uno de sus vértices es convexo, si el ángulo interior en ese punto lo es.

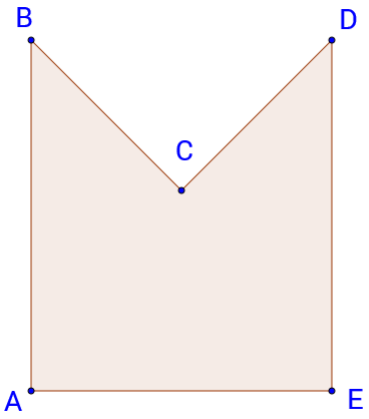

Figura A.3.2 – Vértices convexos vs no convexos.
En la figura, los vértices A, B, D y E son convexos mientras que el vértice C es no convexo.

A.3.2. Proposiciones

Proposición: Un polígono simple C es convexo $\iff \forall x_i, x_j$ vértices con $i \neq j$, la diagonal $[x_i, x_j] \subset C$

El directo es cierto por la definición de convexidad. Para demostrar el recíproco tomar $p, q \in C$ / $[p, q] \not\subset C$. Por continuidad, esto implica $\exists [p', q'] \subset [p, q]$ / $p', q' \in C$ y $(p', q') \cap C = \phi$. De esta forma se tiene que p', q' pertenecen a dos segmentos diferentes del polígono. Supongamos estos sean $[x_1, x_2], [x_3, x_4]$. Observar que cambiando el orden de los vertices se puede asumir

$[x_1, x_2, x_3, x_4]$ es un cuadrilatero simple con lo que $[p', q'] \subset [x_1, x_2, x_3, x_4]$ y como las aristas de este cuadrilatero son diagonales de C, y C no tiene agujeros se tiene $[p', q'] \subset C$ lo que es absurdo.

Proposición: Toda diagonal interna de un polígono simple convexo, divide al convexo en 2 polígonos también convexos.

Antes de deomostrar la proposición, una diagonal interna es una diagonal que no este contenida en el borde.Sea $d = [x_i, x_j]$ una tal diagonal. Si el poligono es $p = [x_1, ..., x_i, ..., x_j, ..., x_n]$, se tiene que d divide a p en los subpoligonos $[x_1, ..., x_i, x_j, ..., x_n]$ y $[x_i, x_{i+1}, ..., x_{j-1}, x_j]$. Ambos de estos polígonos son convexos por la proposición anterior.

Proposición: Si C es un polígono simple convexo y $[x_1, x_2]$ es una arista, C estará contenido en uno de los semiplanos generados por el segmento.

Esta proposición se puede demostrar mediante inducción en la cantidad de vértices (eliminando aquellos que tengan ángulo 180 grados, se asume no se tiene esos vértices). Para $n = 3$ esto es trivialmente cierto. Para el caso $n = 4$ supongase por absurdo que x_4 no esta contenido en el semiplano dado por la segmento $[x_1, x_2]$ y el punto x_3. Dividase el el otro semiplano en las siguientes secciones como se ve en la figura A.3.3.

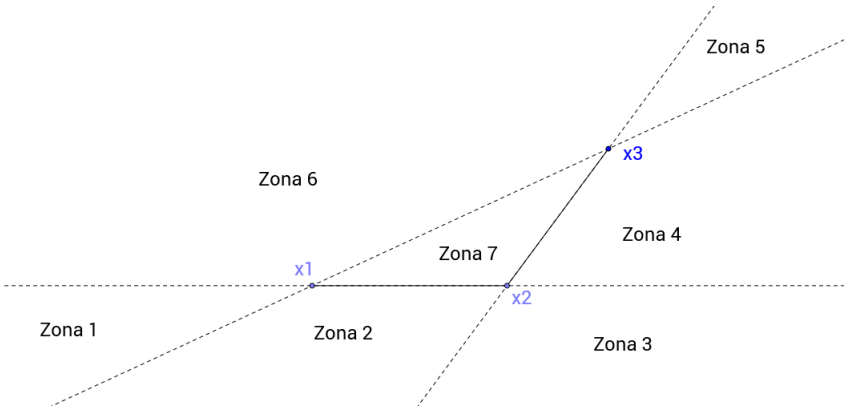

Figura A.3.3 – División de zonas.

Como se ve en dicha figura, x_4 no puede estar la zona 3 ya que de lo contrario $[x_1, x_3] \not\subset C$. Idem en la zona 1 con el segmento $[x_2, x_4]$. Finalmente, tampoco puede estar en la zona 3 ya que de lo contrario $[x_3, x_4]$ intersectaría con $[x_1, x_3]$ concluyendo el absurdo. Para el caso general $n \to n + 1$, basta considerar los poligonos $[x_1, x_2, ..., x_n]$ y $[x_1, x_2, ..., x_{n-1}, x_{n+1}]$. De la proposición anterior se tiene que ambos son convexos y por tanto se puede usar la hipótesis inductiva sobre ambos. De esta forma, la proposición quedaría probada observando que tienen al punto x_3 en común ya que $n \geq 4$ (esto implíca que todos los puntos quedan en el mismo semiplano que x_3).

Proposición: Un polígono simple es convexo $\iff$ todos sus vértices son convexos.

Para demostrar esto, primero supongamos P es simple convexo y que $p_n \in P$ es no convexo. Por definición el ángulo p_{n-1}, p_n, p_{n+1} exterior no es convexo. De esta forma se tiene que $[p_{n-1}, p_{n+1}] \not\subset P$ de lo contrario P_n pertenecería al interior de P lo que es absurdo.

La demostración del recíproco se puede realizar por inducción en la cantidad de nodos de un polígono (nuevamente, se asume no hay ángulos llanos). En el caso base $n = 3$ la proposición se cumple trivialmente ya que todo triángulo es convexo y como sus ángulos suman 180, todos deben ser menores que eso. Para el caso $n \to n + 1$, lo que se hace es dividir al poligono en dos. Si $P = [x_1, ..., x_{n+1}]$, dividimos en $P_1 = [x_1, ..., x_n]$ y $P_2 = [x_n, x_{n+1}, x_1]$. Observar

que P_2 es un triángulo y que P_1 conserva los mismos ángulos que P salvo en los vértices x_n y x_1 que es donde se removió el triángulo. Observar que el nuevo ángulo en estos vértices ha de ser menor y por tanto siguen siendo puntos convexos. Usando la hipótesis inductiva P_1 es convexo. Usando una de las proposiciones anteriores se sabe que $\forall d \in P_1$ diagonal, se tiene que $d \subset P_1$. Para demostrar P es convexo es suficiente mostrar que las diagonales restantes están contenidas en P. Estas son de la forma $[x_i, x_{n+1}]$. Si definimos $d_i := [x_i, x_{n+1}]$, $d_1, d_n \subset P$ trivialmente. Sea d_i con $i \neq 1, n, n+1$. Observar que d_i debe intersectar al segmento $[x_n, x_{n+1}]$. Esto se demuestra aplicandole la proposición anterior a los lados $[x_{n-1}, x_n], [x_n, x_1], [x_1, x_2]$ al polígono P_1. De esta forma se tendrá que el punto x_i esta contenido en la intersección de los tres semiplanos dados por esos segmentos. De esta forma, $\exists x$ / $d_i \cap [x_n, x_{n+1}] = x$. Observar $d_i = [x_i, x] \cup [x, x_n + 1]$, que $[x, x_n + 1] \subset P_2$ y que $[x_i, x] \subset P_1$ ya que P_1 es convexo. Esto implica $d_i \subset P$ lo que finaliza la prueba.

Proposición - Recorte de no convexos 1:
Hipótesis: Sea P un polígono gráfica polar con x su centro. Sea x_i, x_j dos vértices no convexos, no consecutivos orientados en sentido antihorario respecto de c, tal que todos los vértices intermedios sean convexos y el ángulo antihorario x_i, x, x_j menor a 180.
Tesis: el poligono $[x_i, ..., x_j]$ es convexo y no contiene a x.

Para demostrar esto basta observar que el polígono auxiliar $[x, x_i, ..., x_j]$ es convexo. Para ello observar que x es un vértice convexo por hipótesis, idem para todo vértice entre x_i y x_j.Si se observa x_i, x_j también son convexos la tesis queda demostrada por la proposición anterior. Para ver esto último basta observar que los puntos x_{i+1} y x_{j-1} caen en la intersecciónde los semiplanos dados por las aristas $[x, x_i]$ y $[x, x_j]$ ya que P es una gráfica polar. Esto se observa mejor en la figura A.3.4.

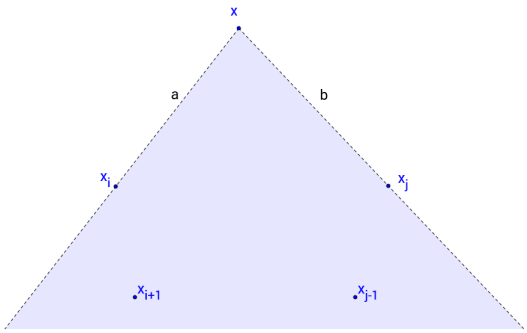

Figura A.3.4 – Orden en gráfica polar.
La imágen muestra a los vértices x_i, x_{i+1}, x_{j-1} y x_j de un polígono gráfica polar junto a su centro x.

Proposición - Recorte de no convexos 2:
Hipótesis: Sea P un polígono gráfica polar con c su centro y $p_1, ..., p_n$ sus vértices no convexos ordenados de form antihoraria tal que $\forall i$ el ángulo antihorario p_i, x, p_{i+1} es menor a 180.
Tesis: El polígono formado por los puntos no convexos es una gráfica polar de centro también x.

La demostración surge de recortar sucesivamente a P por sus vértices no convexos utilizando la proposición anterior.

Apéndice B

Algoritmos

B.1. Algoritmo de etiquetado de vértices

El algoritmo de etiquetado puede dividirse en dos. Un algoritmo para el etiquetado inicial y otro para las siguientes iteraciones de la recursión del algoritmo pTAM.

El algoritmo inicial etiqueta a los vértices del polígono teniendo en cuenta no solo el ángulo en dicho vértice, sino también a la información extra aportada por la nube de puntos. A priori esto parece un poco extraño ya que la definición de vértice convexo solo depende del ángulo en dicho punto. El motivo por el que se utiliza la nube es que el polígono generado por los segmentos extraídos depende de la resolución de puntos que se utilice . Como la nube de puntos es discreta, algunas paredes pueden no tener suficientes puntos para generar un segmento en el algoritmo de extracción. Al tener en cuenta los puntos de la nube se obtiene mayor información sobre el valor de la etiqueta en el polígono completo. Para observar mejor esto, considérese la siguiente figura.

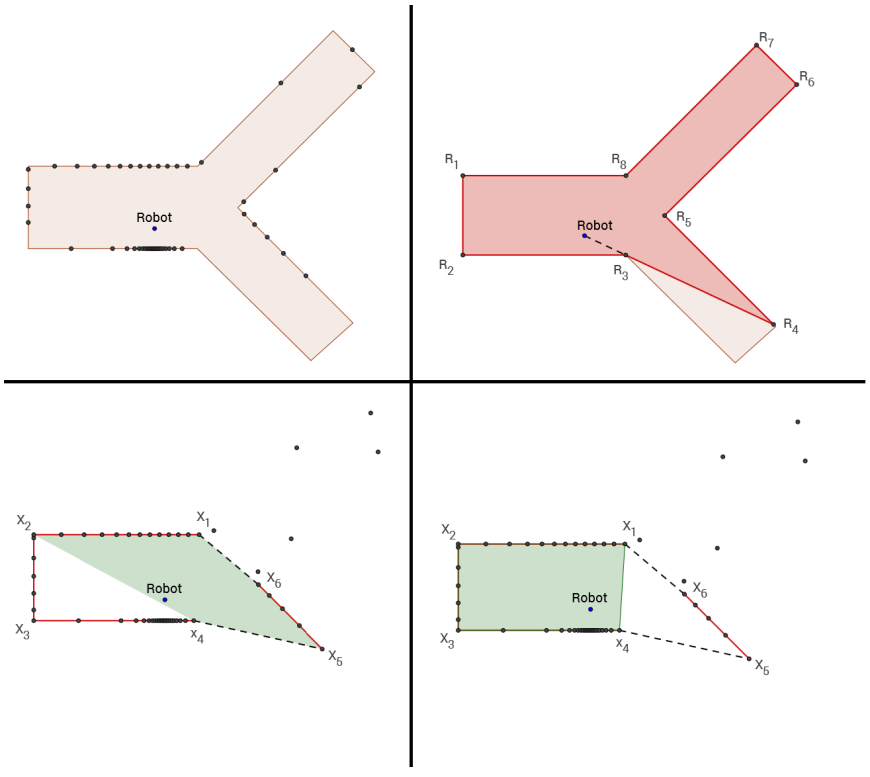

Figura B.1.1 – Extracción de polígonos - dependencia del etiquetado.

En la figura B.1.1 se muestra al robot en una posición de un laberinto con forma de "Y". En la parte superior izquierda se muestra la nube de puntos que se observa bajo una resolución de 7.2 grados (50 puntos). A su derecha, el polígono rojo $[R_1, ..., R_8]$ muestra lo que vería el robot si tuviese resolución infinitesimal. Es decir, si pudiera obtener la distancia a las paredes para toda dirección. Observar que en el polígono rojo los vértices R_3, R_4, R_5 son no convexos. En la fila inferior se muestra los segmentos creados a partir de la nube, los puntos sueltos están muy dispersos como para formar otros segmentos. En la parte izquierda se muestra el convexo que resultaría del algoritmo de recortes (sección 5.3.1.1) si los puntos X_1 y X_6 fuesen etiquetados acorde a su ángulo en el polígono $[X_1, ..., X_6]$. Notar que en dicho caso ambos puntos serían convexos. En la parte derecha se muestra como quedaría el resultado etiquetando usando la nube de puntos. En este caso, tanto X_1, como X_6 son etiquetados como no convexos.

A continuación se presenta el algoritmo. Como el etiquetado inicial y el general difieren muy poco, ambos son resumidos en un solo algoritmo.

Algorithm 9 Etiquetado de un vértice

Entrada: $\{x_1, ..., x_n\}$ = la nube de puntos sensada.
V_{i-1}, V_i, V_{i+1} = tres vértices consecutivos del polígono.

Salida: Etiqueta **Convexo** o **No Convexo** para el vértice V_i

OBS: Recordar que los puntos están ordenados en sentido antihorario. De esta forma, si x, y son dos puntos, es posible decir $x \leq y$.

```
1: Calcular θ_i el ángulo interno V_{i-1}, V_i, V_{i+1}
2: if θ_i > 180 then
3:     Devolver No Convexo
4: else
5:     if not Etiquetado inicial then
6:         Devolver Convexo
7:     else
8:         convexoPorIzquierda = true
9:         convexoPorDerecha = true
10:        if {V_{i-1}, V_i} no forman un segmento extraído then
11:            Buscar x ∈ {x_1, ..., x_n} tal que V_{i-1} ≤ x ≤ V_i y x no esté en el semiplano interno
    delimitado por {V_{i-1}, V_i}
12:            if ∃x then
13:                convexoPorIzquierda = false
14:        if {V_i, V_{i+1}} no forman un segmento extraído then
15:            Buscar x ∈ x_1, ..., x_n tal que V_i ≤ x ≤ V_{i+1} y x no esté en el semiplano interno
    delimitado por {V_i, V_{i+1}}
16:            if ∃x then
17:                convexoPorIzquierda = false
18:        if convexoPorIzquierda and convexoPorDerecha then
19:            Devolver Convexo
20:        else
21:            Devolver No Convexo
```

Para ver como actúan y se comparan ambos algoritmos, considérese el vértice X_1 en la figura B.1.1. En este caso, los vértices V_{i-1}, V_i, V_{i+1} serían X_6, X_1, X_2 respectivamente. El ángulo interno es convexo, por tanto, si no es el etiquetado inicial se retorna la etiqueta correspondiente y el algoritmo finaliza. Si lo es, entonces se revisa la nube de puntos. $[X_6, X_1]$ no es un segmento extraído con lo cual se busca si existen puntos de la nube que estén entre medio de los vértices y queden fuera del semiplano interno. En la figura B.1.1 exiten 6 puntos que

cumplen esta condición (obsrevar los puntos sueltos entre X_1 y X_6). De esta forma, el vértice X_1 no es convexo por izquierda y por tanto recibe la etiqueta acorde.

Observar que la condición especial en el etiquetado inicial deja de ser necesaria en las siguientes iteraciones ya que el polígono es recortado. De esta forma, los "puntos extra" (puntos no utilizados en la formación de segmentos) son recortados también y solo pasan a ser de interés un subconjunto de vértices del polígono inicial.

Finalmente, por simplicidad en la explicación, en el algoritmo anterior no se incluyó tolerancia al ruido. Para ello, es necesario modificar los pasos 2, 11 y 15. En los últimos dos pasos, si el punto no está del semiplano adecuado, se toma una distancia máxima para considerarse como incluido. Por otro lado, la condición en 2 pasa a ser similar a las anteriores. Se dice el ángulo es no convexo si el vértices V_{i+1} queda en el semiplano exterior dado por el segmento $[V_{i-1}, V_i]$. Similarmente se podría haber utilizado la condición $\theta_i > 180 + \epsilon$, pero la tolerancia generada no es buena ya que admite poco error para puntos cercanos a V_i, y mucho cuando se alejan. Esto se evidencia en la figura B.1.2.

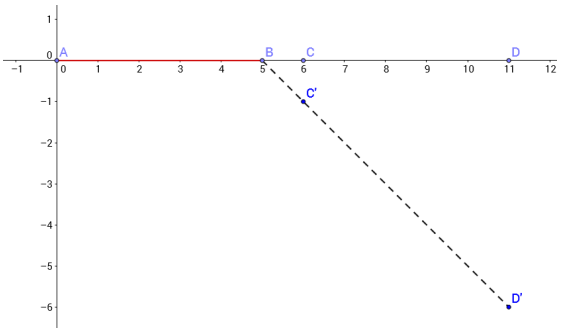

Figura B.1.2 – Error en etiquetado.
En la figura se observa la tolerancia a error para considerar a un vértice como convexo utilizando la condición $\theta_i > 180 + \epsilon$. Si $\epsilon = 45$, el punto C solo puede tener un error vertical de 1 unidad (punto C') para que el angulo A, B, C sea convexo. A su vez, D puede tener a los sumo 6 (punto D').

Bibliografía

[1] Turgay Temel. *System and Circuit Design for Biologically-Inspired Intelligent Learning*. IGI Global, 2010.

[2] Michael A Arbib and James J Bonaiuto. Multiple levels of spatial organization: World graphs and spatial difference learning. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems*, 20(4):287–303, August 2012. ISSN 1059-7123. doi: 10.1177/1059712312449545. URL http://dx.doi.org/10.1177/1059712312449545.

[3] Jeffrey Taube. Head direction cells. *Scholarpedia*, 4(12):1787, 2009.

[4] Edvard Moser and May-Britt Moser. Grid cells. *Scholarpedia*, 2(7):3394, 2007.

[5] Gyorgy Buzsaki. Hippocampus. *Scholarpedia*, 6(1):1468, 2011.

[6] John O'keefe and Lynn Nadel. *The hippocampus as a cognitive map*, volume 3. Clarendon Press Oxford, 1978.

[7] https://www.sciencenews.org/article/neuroscientists-garner-nobel-discovering-brain%E2%80%99s-%E2%80%98inner-gps%E2%80%99.

[8] Clifford Saper. Hypothalamus. *Scholarpedia*, 4(1):2791, 2009.

[9] LW Swanson and GJ Mogenson. Neural mechanisms for the functional coupling of autonomic, endocrine and somatomotor responses in adaptive behavior. *Brain Research Reviews*, 3(1):1–34, 1981.

[10] Larry W Swanson. Cerebral hemisphere regulation of motivated behavior. *Brain research*, 886(1):113–164, 2000.

[11] Larry W Swanson. Anatomy of the soul as reflected in the cerebral hemispheres: neural circuits underlying voluntary control of basic motivated behaviors. *Journal of Comparative Neurology*, 493(1):122–131, 2005.

[12] Yaling Yang and Adrian Raine. Prefrontal structural and functional brain imaging findings in antisocial, violent, and psychopathic individuals: a meta-analysis. *Psychiatry Research: Neuroimaging*, 174(2):81–88, 2009.

[13] Earl K Miller, David J Freedman, and Jonathan D Wallis. The prefrontal cortex: categories, concepts and cognition. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 357(1424):1123–1136, 2002.

[14] http://alojoptico.us.es/portaleto/nav_espacial_rata/comportamiento.html.

[15] http://www.nature.com/nrn/journal/v1/n1/fig_tab/nrn1000_041a_F6.html.

[16] Randy J Nelson. *An introduction to behavioral endocrinology* . Sinauer Associates, 2005.

[17] Richard GM Morris. Morris water maze. *Scholarpedia*, 3(8):6315, 2008.

[18] https://ffiproject.wordpress.com/investigation/experiences-with-mices/.

[19] http://www.transitiontherapeutics.com/technology/alzheimers.php.

[20] J O'keefe. Spatial memory within and without the hippocampal system. *Neurobiology of the Hippocampus*, 375:403, 1983.

[21] Robin R. Murphy. *An Introduction to AI Robotics (Intelligent Robotics and Autonomous Agents)*. A Bradford Book, 1st edition, November 2000. ISBN 0262133830. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/0262133830.

[22] M Arbib and ISRAEL Lieblich. Motivational learning of spatial behavior. *Systems neuroscience*, pages 221–239, 1977.

[23] Alex Guazzelli, Mihail Bota, Fernando J Corbacho, and Michael A Arbib. Affordances. motivations, and the world graph theory. *Adaptive Behavior*, 6(3-4):435–471, 1998.

[24] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[25] Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-23957-4. doi: 10.1007/978-3-540-30301-5. URL http://dx.doi.org/10.1007/978-3-540-30301-5.

[26] URL https://openslam.org/.

[27] Federico Andrade, Martin Llofriu, Gonzalo Tejera, and Facundo Benavides. Slam estado del arte. *Facultad de ingeniería*, 2013.

[28] Josué Enríquez Zárate. Diseño y construcción de un rangefinder.ªplicado a un robot móvil. Master's thesis, Universidad autónoma de Mexico, 2010.

[29] Adriana Tapus. *Topological SLAM - Simultaneous Localization and Mapping with Fingerprints of Places*. PhD thesis, ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE, 2005.

[30] KeithY.K. Leung, TimothyD. Barfoot, and HughH.T. Liu. Decentralized cooperative slam for sparsely-communicating robot networks: A centralized-equivalent approach. *Journal of Intelligent & Robotic Systems*, 66(3):321–342, 2012. ISSN 0921-0296. doi: 10.1007/s10846-011-9620-2. URL http://dx.doi.org/10.1007/s10846-011-9620-2.

[31] Soren Riisgaard and Morten Rufus Blas. Slam for dummies a tutorial approach to simultaneous localization and mapping by the 'dummies', 2005.

[32] Megan R Naminski. An analysis of simultaneous localization and mapping (slam) algorithms. *Macalester College*, 2013.

[33] Victor Narváez, Francisco Yandún, David Pozo, Luis Morales, and Jorge Rosero. *Diseño e implementación de un sistema de Localización y Mapeo Simultáneos (SLAM) para la Plataforma Robótica ROBOTINO*. PhD thesis, Escuela Politécnica Nacional, Facultad de Ingeniería Eléctrica y Electrónica, Quito, Ecuador, 2014.

[34] Rodrigo Francisco Munguía-Alcalá, R. F.a-Alcalá and Antoni Grau-Saldes. Slam con mediciones angulares: método por triangulación estocástica. *Ingeniería, investigación y tecnología*, 14:257 – 274, 06 2013. ISSN 1405-7743. URL http://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S1405-77432013000200010&nrm=iso.

[35] O. Wijk and H.I. Christensen. Triangulation-based fusion of sonar data with application in robot pose tracking. *Robotics and Automation, IEEE Transactions on*, 16(6):740–752, Dec 2000. ISSN 1042-296X. doi: 10.1109/70.897785.

[36] J. Borenstein and Liqiang Feng. Measurement and correction of systematic odometry errors in mobile robots. *Robotics and Automation, IEEE Transactions on*, 12(6):869–880, Dec 1996. ISSN 1042-296X. doi: 10.1109/70.544770.

[37] Guido Zunino. Simultaneous localization and mapping for navigation in realistic environments. In *In Proc. IEEE Conference*, pages 67–72, 2002.

[38] http://courses.cs.tamu.edu/rgutier/csce666_f13/.

[39] http://www.acmesystems.it/FOXG20.

[40] http://www.trossenrobotics.com/dynamixel-ax-12-robot-actuator.aspx.

[41] http://lidarlite.com/docs/v2/.

[42] https://en.wikipedia.org/wiki/Robotis_Bioloid.

[43] http://www.thingiverse.com/thing:769066/#files.

[44] http://www.acmesystems.it/microsd_format.

[45] http://www.trossenrobotics.com/dynamixel-ax-12-robot-actuator.aspx.

[46] http://www.acmesystems.it/compile_linux_2_6_38.

[47] http://www.coppeliarobotics.com/.

[48] http://artoolkit.org/.

[49] http://iaci.unq.edu.ar/materias/vision/archivos.htm.

[50] http://www4.ujaen.es/~satorres/pract_vision.htm.

[51] http://www.jayrambhia.com/blog/hough-transform/.