UNIVERSIDAD POLITÉCNICA DE MADRID

**DEPARTAMENTO DE AUTOMÁTICA
INGENIERÍA ELECTRÓNICA
E INFORMÁTICA INDUSTRIAL**

**División de Ingeniería de Sistemas
y Automática (DISAM)**

Sistema de navegación de un robot móvil

AUTOR: Paloma de la Puente Yusty
TUTOR: Diego Rodríguez-Losada González

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS INDUSTRIALES
José Gutiérrez Abascal, 2
28006 Madrid

Madrid, noviembre de 2007

Índice general

Índice de figuras

Parte I

Preliminares

Capítulo 1

Introducción

1.1 Preámbulo

La robótica móvil es un campo de investigación relativamente reciente que actualmente ocupa importantes líneas de trabajo en laboratorios y centros técnicos de todo el mundo. Su desarrollo supone la integración de numerosas disciplinas entre las que se encuentran la automática, la electrónica, la ingeniería mecánica, la informática, la inteligencia artificial, la estadística ...

Para ser considerado como tal, un robot móvil precisa de un sistema de locomoción que le permita desplazarse. Existen numerosas posibilidades al respecto, ya que podemos encontrar robots que andan, saltan, reptan o se mueven sobre ruedas y también robots aéreos o submarinos cuyo sistema motriz se basa en impulsion fluidomecánica. La mayor parte de estos mecanismos de locomoción surge como imitación de los modos de movimiento que se pueden observar en la naturaleza. La principal excepción es la rueda, ampliamente utilizada por sus buenas características sobre suelo plano.

Dentro de los robots móviles, los robots autónomos son aquellos que no necesitan la intervención humana para mantener el sentido de la orientación y la capacidad de navegación. Como indica Smithers, la idea principal del concepto de autonomía se obtiene de su etimología: *auto* (propio) y *nomos* (ley o regla). Los sistemas automáticos se autorregulan pero no son capaces de generar las leyes que deben tratar de seguir sus reguladores. A diferencia de ellos, los sistemas autónomos han de poder determinar sus estrategias, o leyes, de conducta y modificar sus pautas de comportamiento al mismo tiempo que operan en el entorno. Por lo tanto, queda claro que la autonomía añade requisitos sobre el concepto de automatismo.

Los robots autónomos deben ser capaces de adquirir información sobre el entorno. Para ello han de contar con sensores que proporcionen las medidas y da-

tos necesarios. Aunque no son estrictamente imprescindibles, los robots móviles suelen incorporar sensores propioceptivos que suministran medidas relativas a su estado: velocidad, incremento de posición y aceleraciones. Los encoders situados en las ruedas del robot, por ejemplo, registran el número de revoluciones de las mismas y permiten obtener la llamada posición odométrica a partir de la estimación del movimiento relativo incremental. Aunque esta aproximación presenta una buena precisión a corto plazo, la acumulación de errores a medida que aumenta el recorrido causa grandes diferencias con la posición real. Por este y otros motivos hace falta disponer de sensores estereoceptivos para percibir los aspectos externos al robot. Los sensores de este tipo más comunes en robótica móvil avanzada son los de ultrasonidos, infrarrojos, los láseres y las cámaras. Asimismo, el robot ha de poseer computadores o microprocesadores para el procesamiento de la información y la ejecución de los algoritmos de control del sistema de navegación.

Los robots móviles presentan la gran ventaja de poder emplear sus habilidades particulares allí donde sea necesario. Esta es la clave de su creciente demanda comercial. En los últimos años se han multiplicado sus aplicaciones en entornos industriales, militares, domésticos y de seguridad [2].

Algunos ejemplos de robots de este tipo son los cortacésped, los robots aspiradora, los *Rovers* de Marte, los robots guía ...

Los robots cortacésped suelen utilizar algún medio que delimite la superficie a recorrer (como puede ser un cable que sirva de frontera para cerrar un recinto). Una vez señalada el área de césped a cortar, hacen un barrido irregular de toda la superficie.

Figura 1.1: Robot cortacésped Automower Husqvarna

Los robots aspiradores son capaces de realizar la limpieza de todo tipo de suelos. Suelen constar de dos ruedas traseras motrices y una rueda delantera directriz. Hay modelos básicos que se mueven de forma aleatoria, sin cubrir normalmente la totalidad del espacio a limpiar, y modelos avanzados que incorporan las últimas técnicas de mapeo para lograr una alta eficiencia de funcionamiento.

Los *rovers* constan de un cuerpo central que se desplaza sobre seis ruedas y tienen una cabeza y un cuello para que las cámaras puedan tener una mejor perspectiva. Estas cámaras se utilizan para la construcción de mapas que les permitan

Figura 1.2: Robot aspirador RoboMaxx

guiarse por el territorio así como para enviar imágenes a la Tierra. Estos robots también cuentan con un un brazo robótico y un gran número de sistemas térmicos, además de los sistemas de baterías y de comunicaciones.

Figura 1.3: Rover sobre la superficie de Marte

Los rovers *Spirit* y su "gemelo"*Opportunity* aterrizaron en el planeta rojo el 4 y el 25 de enero de 2004, respectivamente. Tienen una capacidad de movilidad mucho mayor que su antecesor el rover *Pathfinder*.

Los robot guía son un tipo de robots móviles que han de tener amplio conocimiento sobre su entorno. Sus requisitos primordiales están orientados a la interacción con las personas, pero también es imprescindible en ellos un buen sistema de navegación que les permita desplazarse de forma autónoma en exposiciones, ferias y museos. Uno de estos robots es *Urbano* (ver 1.2.1, 1.2.2, Anexo B), un robot inteligente que ha sido empleado en el presente proyecto.

1.2 Antecedentes

En el grupo de Control Inteligente de la Universidad Politécnica de Madrid, en el cual se ha realizado este proyecto, trabaja un equipo de profesores e ingenieros con elevada experiencia en el campo de los robots móviles.

1.2.1 Proyectos previos

Los proyectos dentro de esta área que más significativamente han contribuido al nivel alcanzado se describen a continuación por orden de antigüedad. En la siguiente sección se describirá el hardware de los robots empleados.

Panorama (1989-1993)

Financiado por la Unión Europea (Proyecto ESPRIT 2483), consiste en la elaboración de un sistema avanzado de percepción y navegación para vehículos industriales dentro de entornos parcialmente estructurados y parcialmente conocidos. El proyecto estuvo orientado a demostrar la viabilidad de sistemas de transporte autónomos que pudieran sustituir a vehículos controlados por conductor en un amplio rango de aplicaciones.

Figura 1.4: Proyecto Panorama

RM-III (1994-1996)

Sistema central de control para un sistema industrial de transporte basado en múltiples robots móviles autónomos dotados de inteligencia. Los vehículos tienen un alto grado de autonomía que les permite evitar obstáculos y buscar en cada caso el mejor camino a seguir. Los obstáculos pueden ser tanto fijos como móviles, con lo que la dificultad aumenta. Un componente adicional en la autonomía

de los vehículos lo constituye su capacidad para gestionar el nivel de carga de sus baterías, trasladándose y conectándose automáticamente a los puntos de suministro de energía al considerarlo conveniente. Este proyecto fue financiado por la Comisión Interministerial de Ciencia y Tecnología y contó con la participación de UPM-DISAM y de la Universidad Carlos III.

Figura 1.5: Proyecto RM-III

EVS (1996-1999)

Desarrollo de un entorno de ingeniería para facilitar el diseño y la implementación de sistemas autónomos distribuidos con aplicación en robótica móvil y en procesos continuos. Este entorno proporciona herramientas para el prototipado rápido y la experimentación, por medio de un sistema de realidad virtual que modela el sistema en desarrollo y su entorno.

Figura 1.6: Proyecto EVS

Mobinet (1996-2000)

Trabajo científico de un amplio conjunto de investigadores europeos en el diseño de un prototipo de robot móvil inteligente y completamente autónomo, con alta capacidad de maniobrabilidad y manipulación. Destinado a servir de ayuda a personas discapacitadas, la interacción con el usuario se lleva a cabo a través de comandos de muy alto nivel. El proyecto fue financiado por la Unión Europea bajo programa TMR (Training and Mobility of Researchers).

Figura 1.7: Proyecto MobiNet

Blacky (2000-2001)

Diseño de un prototipo de robot móvil para navegación en entornos parcialmente estructurados y con un elevado número de personas.El objetivo del proyecto es desarrollar un robot capaz de moverse en entornos complicados, tipo recintos feriales, e interaccionar con las personas que allí se encuentren. Desde el punto de vista del control reactivo, los obstáculos no sólo no son fijos, sino que su movimiento es totalmente imprevisible. Desde el punto de vista de la localización, se empleaba un sistema de marcas fijas en las paredes. El robot se encuentra con bastante frecuencia totalmente rodeado por personas, lo que complicaba la determinación de su posicionamiento.

WebFair (2001-2004)

Proyecto Europeo Ref: IST-2000-29456.

Desarrollo y validación de un sistema de tele-presencia basado en robots móviles, capaz de facilitar el acceso de individuos a grandes exhibiciones y ferias comerciales a través de internet. La construcción de mapas y demás pruebas se realizaron en el Castillo de Belgioso, Italia, dedicado a la celebración de exposiciones.

Figura 1.8: Proyecto Blacky

Figura 1.9: Proyecto WebFair

Guido (2004)

Sistema de navegación robusto para un robot de la empresa Haptica que sirva de ayuda a la movilidad de personas débiles o con problemas de visión. Un punto a destacar entre las mejoras introducidas es la utilización de la información proporcionada por los bordes de segmentos del mapa, lo cual evita redundancias y permite así disminuir el coste computacional [3]. Muy buenos resultados en casos reales empleando un ordenador portátil auxiliar.

Urbano(2001-2004)

Proyecto de investigación nacional Ref: DPI2001-3652C0201.

Proyecto financiado por el Ministerio de Ciencia y Tecnología y supervisado por la Ciudad de las Artes y las Ciencias de Valencia (CACSA), donde el robot se ha probado con éxito. El módulo de navegación fue desarrollado por Diego Rodríguez-Losada en el marco de su Tesis Doctoral,[4]

Figura 1.10: Proyecto Guido

Figura 1.11: Proyecto Urbano

RobInt (2004-2007)

Ref: DOI 2004-007908C0201.

Modelado, diseño de una tecnología de diseño e implementación de comportamientos inteligentes en robots guía. Se trata de un proyecto financiado por el Ministerio de Ciencia y Tecnología que da continuidad al proyecto Urbano introduciendo mejoras en los módulos de navegación, consciencia y emociones, conocimiento y diálogo.

1.2.2 Robots

ROBUTER

Fabricado por Robosoft, consiste en una plataforma con elevada capacidad sensorial pensada para trabajo de experimentación. El modo de locomoción es diferencial y cuenta con un computador principal de Motorola con sistema operativo Albatros. Tiene también un computador secundario Sun Sparc. Los sensores de proximidad que posee son 24 sensores de ultrasonidos. También presenta capacidad de visión y Wireless Ethernet. Este robot se utilizó en los proyectos RM-III y EVS.

Figura 1.12: Robot ROBUTER

BLACKY

Plataforma MRV-4 de Denning Branch Int. Robotics dotada de un PC Pentium con sistema operativo Linux y un láser rotatorio para la localización con balizas reflectantes. También posee 24 sensores de ultrasonidos y Ethernet Wireless radio. Es el robot utilizado en el proyecto del mismo nombre.

Figura 1.13: Robot BLACKY

URBANO

Plataforma B21r del fabricante iRobot. El computador base es un PC Pentium con sistema operativo Linux; el computador secundario es un PC Pentium con Windows. Dispone de un láser SICK LM300 para la localización. También posee 24 sensores de ultrasonidos e infrarrojos y Ethernet Wireless radio. En DISAM-UPM se han desarrollado una cabeza y un brazo con los que el robot puede hacer gestos e interaccionar con personas. Este robot se ha empleado en los proyectos WebFair, Urbano y RobInt.

Figura 1.14: Robot Urbano

Dado que este robot ha sido asimismo utilizado en el desarrollo de este proyecto, puede consultarse más información sobre él en el Anexo B.

GUIDO

Guido es un robot de cuatro ruedas con dos motores para hacer girar las dos delanteras, pero ha de ser empujado para moverse [5]. Tiene un sensor láser para percibir el entorno y en el manillar hay un sensor de medida de fuerza para detectar el comando de giro. El computador de a bordo es un PC104 300 MHz Geode con 32 MB de disco duro y 32 MB de RAM, con sistema operativo Haptica-TinyDCLinux. Tiene disponible un puerto Ethernet para las comunicaciones y se alimenta a 24V suministrados por cuatro baterías.

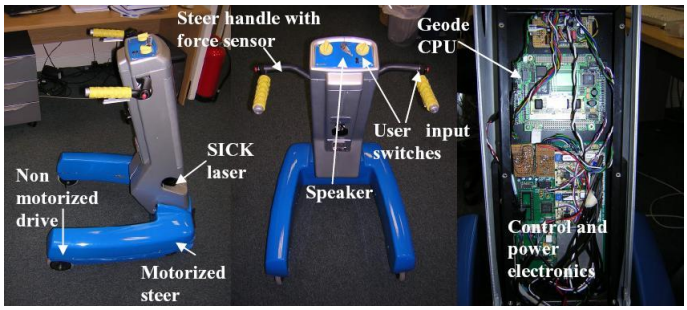

Figura 1.15: Robot Guido

1.3 Marco del proyecto

Este proyecto se ha llevado a cabo bajo beca de colaboración concedida por el Ministerio de Educación y Ciencia en el Departamento de Automática, Ingeniería Electrónica e Informática Industrial de la Escuela Técnica Superior de Ingenieros Industriales de la Universidad Politécnica de Madrid, dentro del grupo de Control Inteligente.

1.4 Objetivos del proyecto

El objetivo del proyecto consiste en la realización de un sistema de navegación para un robot móvil. Se ha trabajado en control de movimiento, planificación de trayectorias, control reactivo, localización y construcción de mapas.

1.5 Alcance del proyecto

Los principales elementos de trabajo en este proyecto son los siguientes:

- **Formación previa.** En primer lugar se requería un asentamiento de conocimientos sobre programación y algorítmica, el aprendizaje de C++ y Visual C++. Posteriormente fue necesaria la comprensión de la base de software de navegación existente. Otra parte importante era la puesta en funcionamiento del robot Pioneer P3AT y el estudio de las librerías de la distribución Aria.

- **Desarrollo del control de movimiento.** Programación de un regulador proporcional con planificación de ganancia, empleando tanto los parámetros de Urbano como los correspondientes al Pioneer. Generación de una trayectoria suave en los puntos de paso de una trayectoria definida. Algoritmo para evitar desviaciones sobre la trayectoria programada. Control reactivo: deformación de la trayectoria por la presencia de obstáculos y paredes cercanas al camino inicial a seguir por el robot.

- **Localización y construcción de mapas.** Estudio del filtro de Kalman y del filtro extendido de Kalman (EKF). Aplicación a la la localización del robot a partir de los puntos de medida proporcionados por el láser. Modificación del modelo para el caso de disponer únicamente de medidas de distancia, que no se utiliza debido a su peor comportamiento. Programación de las fases de predicción y corrección del algoritmo. Análisis del coste computacional y reducción del tiempo de procesamiento. Obtención y actualización de mapas de puntos a partir de los datos registrados por el láser. Utilización de mapas ya realizados para efectuar la localización. Borrado de los puntos correspondientes a obstáculos dinámicos del mapa que se encuentren cercanos a la posición del robot en cada momento. Con el Pioneer sólo es utilizable la parte del proyecto relativa al control de movimiento, planificación de trayectorias y control reactivo ya que al no poderse utilizar el láser no era posible realizar tareas de localización.

- **Integración.** Funcionamiento conjunto e intercambio de información entre los componentes anteriormente indicados.

- **Pruebas.** Pruebas de funcionamiento en modo simulación y con los robots reales. Corrección, depuración y posibles mejoras a partir de los resultados de dichas pruebas.

  Paralelamente se iba desarrollando un cuadro de diálogo para facilitar la ejecución y la selección de opciones en la misma.

Capítulo 2

Estado del Arte

2.1 Navegación, planificación de trayectorias y control reactivo

Un aspecto esencial de un robot móvil es su capacidad de navegación. Sin entrar en profundidad en la cinemática implicada en el movimiento de diferentes tipos de robots móviles, ha de tenerse en cuenta que ésta limita significativamente las posibles configuraciones que puede adoptar un robot para moverse entre dos determinados puntos. El término *trayectoria* se diferencia del de *camino* en que incorpora la dimensión tiempo. Así, mientras que en un camino se tiene una serie discreta de puntos o configuraciones, en una trayectoria se asocia a cada uno de ellos una velocidad. No obstante, cuando se hace referencia a la planificación, es corriente el empleo de ambos conceptos indistintamente.

2.1.1 Control de movimiento

El desplazamiento del robot entre dos puntos se realiza mediante lo que se conoce como *motion control*. Existen diferentes técnicas para afrontar esta tarea, distinguiéndose entre control en bucle abierto o control con realimentación[6].

Control en bucle abierto

Consiste en el seguimiento de una trayectoria definida por un perfil de puntos o velocidades en función del tiempo. El mayor inconveniente que presenta es la alta probabilidad de que el robot no llegue a su destino si por algún motivo se encuentra en un lugar distinto al previamente esperado para un cierto momento. Además, precalcular una trayectoria posible no resulta sencillo en muchos casos.

Otra de sus desventajas es que suele basarse en movimientos de giro o avance puros, y con frecuencia da lugar a movimientos algo bruscos.

Control con realimentación

Una manera más apropiada de tratar el control de movimiento de un robot móvil es el control mediante *feedback* de su posición. Con un controlador de este tipo, la planificación de trayectorias se reduce a la obtención de una serie de puntos intermedios que determinen el camino a seguir entre los puntos origen y destino.

2.1.2 Aptitudes para la navegación: planificación y reacción

En navegación de alto nivel juegan un papel fundamental las habilidades de planificación y reacción del robot. Ambos sistemas son igualmente importantes y han de integrarse adecuadamente. De nada sirve que el robot disponga de un conjunto de acciones a realizar para alcanzar un objetivo dado (plan), si durante la ejecución del mismo se encuentra con condiciones no previstas a las que no sabe cómo hacer frente. Tampoco tiene sentido que el robot sea capaz de reaccionar ante diferentes circunstancias si no puede guiarse para llegar al destino deseado. La solución teórica ideal lleva a la fusión de ambos conceptos, de modo que constantemente se elabora o actualiza un plan que reaccione en tiempo real a la información más inmediata que se tenga para describir el entorno.

Se dice que el sistema de un robot es completo si y sólo si para todos los posibles problemas que puedan encontrarse (diferentes estados de partida, mapas o destinos...), cuando exista alguna trayectoria entre el estado inicial y el final, el sistema llegará al estado final. Por lo tanto, si un sistema es incompleto significa que existe al menos un problema que tiene solución para el cual el sistema no encuentra solución. Siendo una propiedad enormemente difícil de conseguir, a menudo se renuncia a la completitud por razones de coste computacional.

Planificación de trayectorias

La planificación de trayectorias se lleva a cabo en la mayoría de los casos en una representación formal denominada *espacio de configuración*. Para un robot con k grados de libertad, cada uno de sus posibles estados o configuraciones se describirá con k valores reales: $q_1, q_2..., q_k$. El vector que contenga esas k coordenadas podrá considerarse como un punto en un espacio de dimensión k que recibe el nombre de espacio de configuración C del robot. En el caso de que el robot se mueva en las cercanías de algún obstáculo, han de evitarse las colisiones con él, lo cual puede complicarse mucho en el espacio físico. Sin embargo, en el espacio de configuración la resolución del problema es directa si se define el *espacio de*

configuración de obstáculos O como el subespacio de C en el que el robot choca contra algún objeto. El subespacio libre en el que el robot puede moverse con seguridad será la diferencia $F = C - O$.

Generalmente, se considera al robot móvil como un simple punto. En consecuencia, el espacio de configuración es fácilmente manejable por ser plano con ejes x e y. Una importante consideración a realizar es el hecho de que al reducirse el robot a un punto, habrá que aumentar el tamaño de los obstáculos en la medida del radio del robot.

Una vez introducido este concepto, se distinguen tres estrategias principales para efectuar la planificación de trayectorias básica:

- *Mapa de líneas*: basado en identificar rutas en el espacio libre. Destacan los métodos del *Gráfico de visibilidad* y del *Diagrama de Voronoi*.

- *Descomposición en celdillas*: a partir de una mapa de celdillas se clasifican éstas en libres y ocupadas y se crea un grafo de conectividad entre celdas libres adyacentes (ver 2.2.1).

- *Campo de potencial*: consiste en imponer una adecuada función matemática sobre el espacio. Permite obtener la *fuerza* sobre el robot (que se traducirá en una velocidad) como resultado de potenciales de atracción y repulsión.

Las trayectorias generadas por estos algoritmos pueden ser mejoradas en relación a distintos criterios. Por ejemplo, puede resultar interesante suavizar los ángulos en las trayectorias previas mediante arcos de circunferencia (como se implementa en el presente proyecto) o interpolación cuadrática. Estas técnicas de perfeccionamiento son también un componente relevante del término planificación de trayectorias.

Control reactivo

El control reactivo tiene como principal propósito evitar que el robot pueda chocar con algún obstáculo en el seguimiento de la trayectoria previamente hallada. Para ello existen diversos algoritmos que modifican estas trayectorias en función de los objetos detectados por las medidas que le llegan al robot procedentes de los sensores estereoceptivos. La presencia de obstáculos en una trayectoria obtenida mediante planificación puede deberse a errores en la localización, a imprecisiones en el mapa utilizado o a obstáculos dinámicos no contemplados en el mapa.

Algunas soluciones (por ejemplo [7]) consisten en calcular un conjunto de comandos de giro o de variación de la velocidad a aplicar al robot en función de lo que éste percibe a través de sus sensores. También se ha afrontado la tarea de esquivar obstáculos mediante controladores basados en lógica borrosa.

Un nuevo enfoque, efectivo y genérico, es el que se presenta en [8]. A partir de un camino inicial libre de obstáculos, se trata de ir deformándolo iterativamente cuando se encuentran obstáculos próximos, siguiendo para ello un criterio de optimización. El camino deformado debe apartarse de los obstáculos, cumplir las restricciones cinemáticas de movimiento (lo que complica significativamente el problema en el caso de robots no holónomos) y mantener las mismas configuraciones inicial y final que el camino original. En el artículo se establece un marco teórico en el que la deformación del camino inicial se modela como un sistema dinámico dentro del algoritmo que controla el proceso.

También cabe mencionar que existen otros algoritmos de control reactivo que podrían considerarse más avanzados por tener en cuenta la velocidad de los obstáculos detectados. No obstante, emplean modelos excesivamente sencillos del robot y los obstáculos que no suelen conducir a un buen comportamiento práctico.

Por último, hay trabajos recientes que se centran en resolver el problema en escenarios complejos, con un alto grado de ocupación o dinámicos. Para ello, se basan en estrategias de ”divide y vencerás” posteriormente aplican diferentes técnicas de control reactivo. En [9] se explica una metodología que, a nivel simbólico, consiste en identificar situaciones para posteriormente aplicar una determinada acción en cada caso. Una vez definidas las posibles situaciones (a partir de las posiciones relativas de robot, obstáculos y destino) así como las acciones asociadas a las mismas, se realiza una implementación geométrica particular.

2.2 Localización y construcción de mapas con un robot móvil

Uno de los mayores problemas que conciernen a la navegación de robots móviles consiste en la determinación de su localización con un elevado grado de precisión. Para cumplir con el objetivo de autonomía no basta con situar el robot en un sistema de referencia global sino que es imprescindible conocer su posición relativa tanto respecto a posibles obstáculos móviles como respecto al modelo estático de su entorno. Para ello, existen diferentes alternativas utilizando mapas que o bien se introducen previamente al robot o bien se elaboran a medida que el mismo se mueve por un determinado lugar. Esta segunda opción empieza a desarrollarse a finales de los años 80. En un principio se desacoplaron los problemas de construcción del mapa y de la localización del robot en el mismo; pronto se descubriría que la solución rigurosa requiere tratar ambos aspectos al mismo tiempo en lo que se conoce como SLAM (Simultaneous Localization and Mapping). La solución al problema SLAM está considerada como pieza clave en la búsqueda de la completa autonomía de un robot móvil y concentra los principales esfuerzos de

investigación de grupos de robótica móvil de todo el mundo que han conseguido importantes avances y continúan intentando resolver dicho problema.

El origen de las dificultades en la localización y la construcción de mapas se deriva de la existencia de ruido en las medidas de los sensores y en las limitaciones en el rango de las mismas. Entrando un poco más en detalle, los principales factores que impiden que el proceso sea más sencillo son los siguientes:

- Las observaciones se obtienen con respecto al sistema de referencia propio del robot, cuya posición viene afectada de un cierto grado de incertidumbre inherente a la odometría. Así, la imprecisión en las observaciones se añade a la ya existente en la posición del robot y se complica extremadamente la minimización de los errores.

- En multitud de casos es necesaria la representación de mapas de tamaño grande, lo cual supone un mayor coste computacional y una mayor imprecisión en la odometría según aumentan los desplazamientos del robot.

- Los entornos suelen ser dinámicos, sobre todo en el caso de robots guía o domésticos. Si se considera que las observaciones corresponden a puntos fijos representados con carácter permanente en el mapa se simplifica el problema, pero se tienen discrepancias con la realidad. Una aproximación parcialmente eficiente consiste en ir borrando objetos transitorios del mapa a lo largo del tiempo, tratándolos a modo de ruido. El rápido avance de la visión artificial y el desarrollo de dispositivos sensores que diferencien entre obstáculos móviles y estructuras estáticas dentro de un marco de referencia móvil permitirán notables mejoras en este aspecto.

- La asociación de las observaciones con los objetos del mapa puede resultar compleja si éstos son parecidos entre sí. En ciertos casos no puede efectuarse la correspondencia de forma determinista y ha de emplearse una formulación probabilista.

- Los entornos son tridimensionales pero contemplar este aspecto introduce un mayor grado de complejidad y ha de tenerse en cuenta que generalmente los sensores están configurados para una percepción plana horizontal. Aunque existen algunos trabajos en la representación de mapas tridimensionales, la mayor parte de los algoritmos más empleados hasta el momento aceptan la hipótesis de bidimensionalidad del entorno. El paso a modelos en tres dimensiones es uno de los próximos objetivos a seguir.

2.2.1 Tipos de mapas

Cuanto mayor sea la complejidad del mapa a emplear, mayor será la complejidad computacional de su construcción, de la localización y de la navegación. La precisión del mapa vendrá condicionada en cada caso por la precisión que se necesite en el movimiento del robot y por la precisión de los sensores de que se disponga.

Se establecen tres niveles posibles de representación en la definición de un mapa: geométrico, topológico y semántico. Una representación completa del entorno debería incorporarlos todos pero con frecuencia las soluciones halladas se centran en uno sólo de ellos y, a lo sumo, incluyen uno o los dos restantes como mera información extra que pueda mejorar la navegación.

El nivel métrico consiste en representar las coordenadas y propiedades de los objetos del mapa. Este modelo puede a su vez ser geométrico, en el que se representan elementos discretos del entorno de modo que almacenan su parametrización geométrica, o discretizado, en el que la ocupación del entorno se analiza mediante una división del mismo. Así, los mapas métricos geométricos representan objetos con determinadas características geométricas y diferentes grados de complejidad dependiendo de las capacidades sensoriales y de extracción de información. En la figura 2.1 se muestran dos mapas geométricos que reflejan propiedades características de los entornos de interiores. En el mapa de la izquierda se representan las paredes mediante segmentos y en el de la derecha puede apreciarse que las esquinas y objetos delgados, como patas de mobiliario, se representan como puntos.

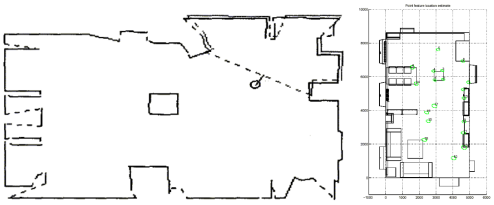

Figura 2.1: Mapas métricos geométricos

Este tipo de mapas es ampliamente utilizado en entornos estructurados debido principalmente a la facilidad de visualización que ofrecen y la compacidad que presentan. Otra ventaja fundamental es la filtración de los objetos dinámicos al hacerse la extracción previa de características del entorno. Los sensores necesarios para construir estos mapas no pueden generar mucho ruido, puesto que han de permitir distinguir los diferentes elementos del entorno. Otro inconveniente a resaltar es su incapacidad para proporcionar un modelo completo del espacio que rodea al robot. Los puntos que no se identifican como características geométricas del mundo real son eliminados, con lo que para ganar en robustez y compacidad se

pierde información de los sensores. Esta limitación afecta a tareas como la planificación de trayectorias y la exploración de entornos, reduciendo consiguientemente la utilidad de estos mapas en la navegación de robots móviles.

En los mapas métricos discretizados,se utiliza la información de los sensores sin segmentar y se construye una función de densidad de probabilidad de ocupación del espacio. Como ésta no puede cubrir todo el espacio de forma continua, se efectúa una descomposición en celdillas y se asigna una probabilidad a que cada una esté ocupada o libre. Esta división puede ser exacta, manteniendo las fronteras de los obstáculos como bordes de las celdillas, o mediante celdillas de dimensiones fijas que se reparten por todo el espacio [6]. En las figuras 2.2 y 2.3 pueden verse ejemplos de ambos tipos de descomposición. En la división en celdillas fijas se aprecia que un estrecho paso entre dos obstáculos puede perderse con esta representación.

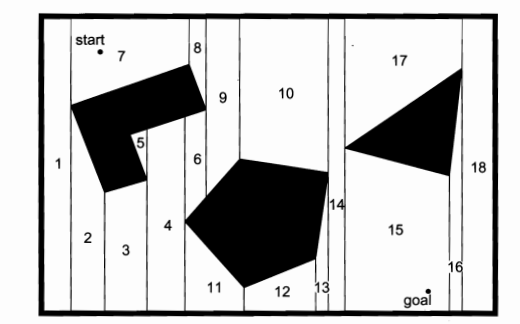

Figura 2.2: Descomposición en celdillas exactas

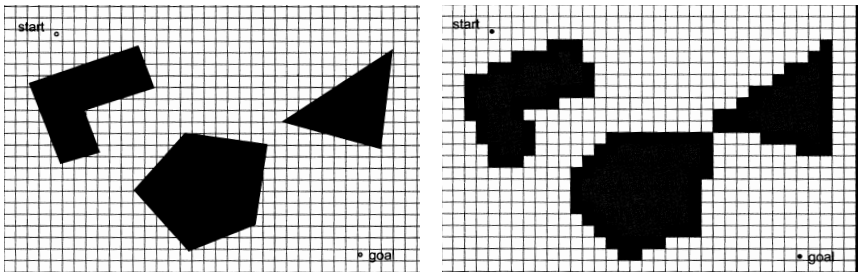

Figura 2.3: Descomposición en celdillas fijas

En este caso no se analiza la pertenencia de cada celdilla a un objeto individual, por lo que aunque el espacio esté discretizado se logra su representación de forma continua. En la figura 2.4 se puede ver un mapa discretizado o de ocupación de celdillas de un entorno con formas irregulares que haría complicada la representación geométrica.

Figura 2.4: Mapa métrico discretizado

Este tipo de mapas puede precisar de una alta capacidad de almacenamiento, tanto mayor cuanta más resolución se requiera. Por otra parte, permite representaciones continuas y completas incluso a partir de datos de sensores con mucho ruido como los de ultrasonidos, lo que los hace especialmente prácticos.

El nivel de representación topológico se centra en definir nodos (en ocasiones llamados *lugares distintivos*) y las conexiones entre ellos. Los mapas topológicos a menudo se elaboran a partir de mapas geométricos pero también pueden obtenerse directamente. Un concepto importante en estos modelos es el de nodos adyacentes. Los nodos adyacentes son aquellos que el robot puede alcanzar sucesivamente sin pasar por ningún otro nodo intermedio. Dentro de este nivel son muy comunes las representaciones basadas en el Diagrama de Voronoi o el Gráfico de Voronoi Generalizado (GVG)como las que se presentan en [10] y se muestran en la figura 2.5.

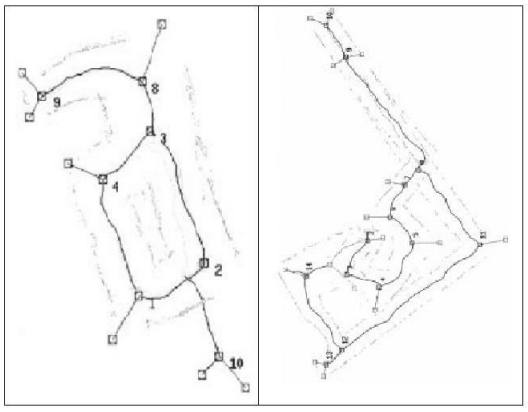

Figura 2.5: Mapas topológicos basados en el diagrama de Voronoi

Los mapas topológicos tienen un grado de compacidad incluso mayor que

los mapas métricos geométricos. Sin embargo, esta representación depende en gran medida de las capacidades sensoriales del robot y da lugar a mapas menos realistas.

La característica fundamental del nivel semántico frente al topológico es que toda la información geométrica se elimina por completo. El resultado es una representación de los lugares más significativos del entorno y sus conexiones, denotados mediante simples etiquetas lingüísticas. En la figura 2.6 se observan los mapas topológico y semántico de un entorno de interiores obtenidos en [11].

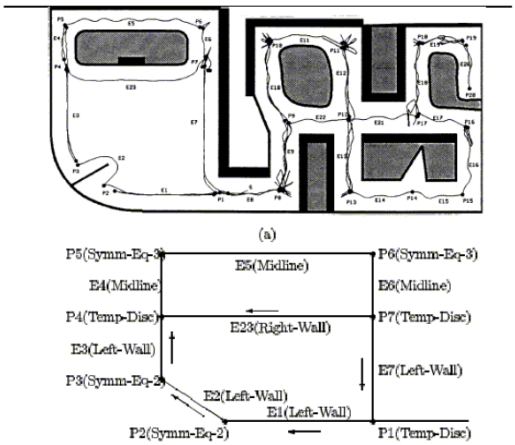

Figura 2.6: Mapas topológico y semántico

De estos tres niveles, el más utilizado es, sin duda, el métrico. Su principal atractivo es la gran riqueza de su representación, lo que permite una navegación del robot más robusta. Los otros dos niveles de representación suelen construirse a partir del anterior y se utilizan para tareas de planificación de alto nivel. Su compacidad los hará interesantes en entornos estructuralmente complejos o de tamaño superior a los explorados hasta el momento.

2.2.2 SLAM con técnicas probabilísticas

En cualquier modelo matemático de un sistema existen diversas fuentes de incertidumbre. En los sistemas dinámicos, además, puede haber perturbaciones que afecten al control. Por último, como ya se ha mencionado, los sensores no pueden suministrar información completa y exacta. Estas son algunas de las limitaciones por las que los modelos deterministas resultan insuficientes y por las que se llega al planteamiento de modelos estocásticos.

Desde los años 90 las principales técnicas de construcción de mapas y localización se han fundamentado en la teoría de la probabilidad y, más concretamente, en

el teorema de Bayes. Los algoritmos que analizan la solución al problema SLAM desde esta perspectiva son claramente los que mejores resultados han tenido. La formulación probabilística rigurosa del problema SLAM se conoce comúnmente como Filtro de Bayes y puede considerarse como una generalización temporal del teorema del mismo nombre. El planteamiento se expone a continuación[4].

En la construcción de un mapa por un robot móvil se tienen dos tipos de medidas: las que provienen de los sensores propioceptivos (odometría) y las de los sensores estereoceptivos (observaciones). Se puede suponer sin pérdida de generalidad que estas medidas llegan secuencial y alternativamente en el tiempo:

$$u_1, z_1, u_2, z_2 ... u_t, z_t \tag{2.1}$$

Donde u representa un desplazamiento relativo del robot y z una medida de los sensores externos. El subíndice indica el instante de tiempo al que corresponde cada medida, siendo t el instante actual. El estado del sistema en el instante t típicamente vendrá dado por la posición del robot s_t y la posición de los objetos del mapa m_t

$$x_t \sim s_t, m_t \tag{2.2}$$

El teorema de Bayes permite conocer la probabilidad de dicho estado condicionada a los datos recogidos hasta ese instante t. Hay que destacar que esta función de probabilidad representa cualquier solución probabilista al problema SLAM.

$$p(x_t \mid z^t, u^t) = p(s_t, m_t \mid z^t, u^t) = \eta p(z_t \mid s_t, m_t, z^{t-1}, u^t) p(s_t, m_t \mid z^{t-1} u^t) \tag{2.3}$$

$$\begin{aligned} z^t &= z_1, z_2 ... z_t \\ u^t &= u_1, u_2 ... u_t \end{aligned} \tag{2.4}$$

Aceptando que el único estado que existe es el definido por s_t y m_t (hipótesis de Markov), la función de probabilidad puede escribirse nuevamente:

$$p(x_t \mid z^t, u^t) = p(s_t, m_t \mid z^t, u^t) = \eta p(z_t \mid s_t, m_t) p(s_t, m_t \mid z^{t-1} u^t) \tag{2.5}$$

El segundo factor puede expresarse en base al teorema de la probabilidad total como:

$$p(s_t, m_t \mid z^{t-1}, u^t) = \int \int p(s_t, m_t \mid z^{t-1}, u^t, s_{t-1}, m_{t-1}) p(s_{t-1}, m_{t-1} \mid z^{t-1}, u^t) ds_{t-1} dm_{t-1} \tag{2.6}$$

Aplicando otra vez la hipótesis de Markov y el hecho de que es lógico pensar que el movimiento del robot es independiente del mapa, se tiene:

$$p(s_t, m_t \mid z^{t-1}, u^t) = \int p(s_t \mid u_{t-1}, s_{t-1}) p(s_{t-1}, m \mid z^{t-1}, u^{t-1}) ds_{t-1} \qquad (2.7)$$

Con lo que finalmente queda:

$$p(s_t, m \mid z^t, u^t) = \eta p(z_t \mid s_t, m_t) \int p(s_t \mid u_t, s_{t-1}) p(s_{t-1}, m \mid z^{t-1}, u^{t-1}) ds_{t-1} \qquad (2.8)$$

Esta expresión eq:final es la que se conoce como Filtro de Bayes en el problema SLAM. El problema así planteado puede abordarse recursivamente si se conocen en cada instante las medidas sensoriales de ese instante (con su probabilidad) junto con la función de probabilidad del estado en el instante anterior. Hacen falta, por lo tanto, dos funciones de probabilidad correspondientes a datos sensoriales: la relativa a la odometría, $p(s_t \mid u_t, s_{t-1})$, y la de las medidas efectuadas por el sistema estereoceptivo, $p(z_t \mid s_t, m_t)$.

Sin embargo, la ecuación del Filtro de Bayes no puede resolverse ni implementarse directamente sino que es preciso realizar ciertas simplificaciones o suposiciones que dan lugar a los diferentes algoritmos existentes.

El estudio de estos algoritmos queda fuera del alcance de este proyecto ya que en él no se hace SLAM propiamente dicho, como se verá en el capítulo correspondiente.

2.2.3 Localización probabilística basada en mapas

Entre las técnicas probabilistas de localización destacan especialmente dos tipos de localización: *localización de Markov* y *localización basada en el filtro de Kalman*. El primero de ambos métodos utiliza una distribución de probabilidad explícitamente especificada sobre todas las posibles posiciones del robot en el mapa. No requiere que el robot tenga una posición inicial conocida y permite la relocalización a partir de situaciones ambiguas. Sin embargo, para actualizar la probabilidad de todas las posiciones del espacio de estado hace falta que éste tenga una representación discretizada (mapas de celdillas, mapas topológicos...).

En 1994 se celebró el concurso *1994 American Association for Artificial Intelligence (AAAI) National Robot Contest*, en el que se proporcionaba a los robots participantes un mapa topológico imperfecto del entorno con el cual tenían que conseguir llegar a una determinada habitación establecida como meta. El robot Rinho empleaba en dicho concurso localización de Markov a partir de la construcción de un mapa de celdillas. En la figura 2.7 pueden verse sus resultados.

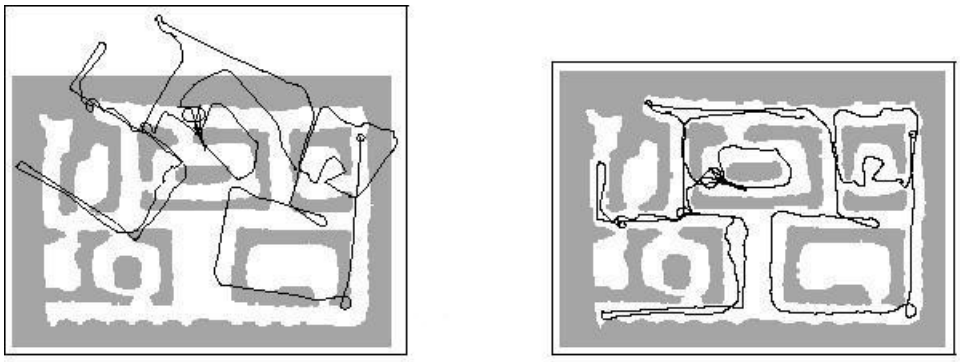

Figura 2.7: [1] Trayectorias odométrica y corregida mediante localización de Markov en un mapa de ocupación de celdillas

Las técnicas de localización de Markov con mapas topológicos son difíciles de aplicar en entornos no estructurados. En otro tipo de casos, sin embargo, puede ser más adecuado. El ganador de la competición de robots móviles de la AAAI de 1994, llamado Dervish, empleaba localización probabilística de Markov sobre representación topológica del entorno. En la figura 2.8 se muestra un ejemplo de representación topológica de un entorno de oficinas similar al del concurso. Dervish detectaba las puertas cerradas y las puertas abiertas, guardando la información para relacionarla con la conectividad de nodos del mapa. Posteriores desarrollos pueden encontrarse en [12] y [13].

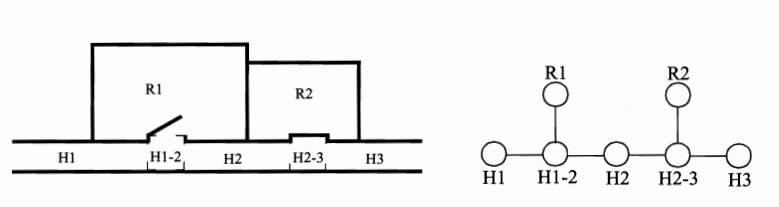

Figura 2.8: Representación topológica de un entorno de oficina

Otro tipo de localización basada en el método de Markov es la *localización de Monte Carlo*. En ella, inicialmente se toma al azar un elevado número de posibles configuraciones hipotéticas para el robot. Con las medidas de los sensores se va actualizando la probabilidad de que cada una de esas configuraciones sea la configuración del robot en ese instante en base a modelos estadísticos mediante aplicación del teorema de Bayes. De un modo similar, cada movimiento incremental del robot se incorpora al cálculo de probabilidades mediante el modelo estadístico de la medida del movimiento. Las configuraciones cuya probabilidad resulta ser muy baja se sustituyen por otras configuraciones aleatorias del espacio

de estado.

El filtro de Kalman emplea una representación de densidad de probabilidad Gaussiana de la posición del robot y la asociación de datos para la localización. Es interesante el hecho de que el proceso de localización del filtro de Kalman es el resultante si se aplica al modelo de Markov la suposición de que la incertidumbre en la posición del robot sigue una distribución normal.

Las principales ventajas de la localización mediante filtro de Kalman son su precisión y eficiencia cuando se conoce la posición de partida en el movimiento del robot, el poder aplicarse con modelos de representación del entorno continuos (mapas geométricos) y las menores necesidades de tiempo y memoria respecto a la localización de Markov. El mayor inconveniente que presenta es la posibilidad de que el robot quede completamente perdido si el error en su movimiento es demasiado grande (como resultado del choque con un obstáculo, por ejemplo).

2.2.4 Filtro de Kalman

La localización basada en el filtro de Kalman es la más extendido en la literatura y en implementaciones prácticas y debido a sus buenas propiedades, apropiadas para la mayor parte de aplicaciones, es la que se ha utilizado en el desarrollo de este proyecto.

Conceptos introductorios

El filtro de Kalman es un algoritmo recursivo óptimo para procesar información[14]. Combina la totalidad de la información disponible, ponderándola según su grado de incertidumbre, para realizar la estimación de las variables que definan el estado del sistema. El funcionamiento del filtro requiere el conocimiento de la dinámica del sistema, así como de los modelos estadísticos del ruido en las medidas de los sensores y de la incertidumbre inicial del modelo del sistema. Al tratarse de un algoritmo recursivo, cada estimación se efectúa a partir de la anterior y de la nueva información disponible, sin que sea preciso almacenar todos los datos previos.

El filtro de Kalman permite minimizar el error en la estimación de las variables de interés cuando el modelo es lineal y la incertidumbre del sistema y de las medidas de los sensores es ruido blanco gaussiano. En esta situación, la función de densidad de probabilidad de cada variable a analizar condicionada a las medidas tomadas es tal que la media, la moda y la mediana coinciden, lo que evita cualquier posible conflicto a la hora de determinar cuál es la mejor estimación. Las hipótesis aceptadas pueden parecer altamente restrictivas pero hacen posible la resolución matemática del problema y se acercan bastante bien a la realidad

en la mayoría de los casos. En otros, sin embargo, han de contemplarse algunas variaciones y resulta de utilidad el llamado filtro extendido de Kalman (EKF).

Ecuaciones para sistemas dinámicos

A continuación se muestran las ecuaciones que definen el comportamiento del filtro de Kalman aplicado a sistemas dinámicos[15].

La relación entre cada una de las medidas tomadas en un instante y el estado del sistema es del tipo:

$$z_k = H_k x_k + \rho_m \tag{2.9}$$

Con z vector de medidas, x vector de estado y ρ_m ruido gaussiano en las medidas, con media nula y matriz de covarianza R_k ó R si ésta es constante en el tiempo.

Se considera que la evolución del sistema sigue el siguiente modelo de estado lineal:

$$x_k = Ax_{k-1} + Bu_{k-1} + \rho_\rho, \tag{2.10}$$

donde A es la matriz de estado; B, la matriz de entrada; u_{k-1}, el vector de entrada en el instante k-1 y ρ_ρ representa la incertidumbre del proceso (con matriz de covarianza Q_p, o $Q_{p_{k-1}}$ si es variable con el tiempo).

Etapa de predicción Así, la *predicción del estado*, $\tilde{x}_k$, vendrá dada por:

$$\tilde{x}_k = A\hat{x}_{k-1} + Bu_{k-1}, \tag{2.11}$$

siendo $\hat{x}_{k-1}$ la estimación del estado en el instante k-1.

Si la entrada u se conoce con exactitud, el error de predicción dado por la diferencia entre 2.11 y 2.10 será:

$$\tilde{x}_k - x_k = A(\hat{x}_{k-1} - x_{k-1}) - \rho_\rho \tag{2.12}$$

y la matriz de covarianza de x_k queda:

$$\tilde{P}_k = A\hat{P}_{k-1}A^T + Q_{p_{k-1}} \tag{2.13}$$

Etapa de corrección La solución de mínimos cuadrados entre las medidas y las medidas esperadas es:

$$\hat{x}_k = \tilde{x}_k + K_k(z_k - H_k\tilde{x}_k) \tag{2.14}$$

$$\hat{P}_k = (I - K_k H_k)\tilde{P}_k \tag{2.15}$$

con

$$K_k = \tilde{P}_k H_k^T S_k^{-1} \tag{2.16}$$

$$S_k = R_k + H_k \tilde{P}_k H_k^T \tag{2.17}$$

Estas ecuaciones constituyen el llamado filtro *dinámico* de Kalman. La matriz K_k se denomina *ganancia de Kalman*. La diferencia entre las medidas en k y sus valores esperados, $v_k = z_k - H_k \tilde{x}_k$, se conoce como la *innovación* del proceso y la matriz S corresponde a su matriz de covarianza.

El filtro de Kalman *estático* es un caso particular del anterior y se obtiene a partir de estas ecuaciones haciendo $A = I$, $B = 0$ y $Q_p = 0$.

2.2.5 Filtro Extendido de Kalman

La principal aportación de este algoritmo respecto al del filtro de Kalman convencional es su extensión a sistemas no lineales. También permite incorporar los casos en los que la relación entre el estado y las medidas de los sensores externos no puede definirse de forma explícita. La convergencia del EKF depende de diversos factores como pueden ser la estimación inicial, las no linealidades de las ecuaciones, el orden en que se procesan las medidas...De este modo, no existe prueba formal de la convergencia del algoritmo, sino tan solo ciertos tests de consistencia que evalúan el comportamiento del mismo en cada caso.

Ecuaciones para sistemas no lineales

Las ecuaciones del medida 2.9 y de estado 2.10, frecuentemente presentan carácter no lineal:

$$z = h(x) + \rho_m \tag{2.18}$$

$$x_k = f(x_{k-1}, u_{k-1}) + \rho_\rho, \tag{2.19}$$

cuyas ecuaciones linealizadas en la estimación más reciente son:

$$z = h(\tilde{x}_k) + \frac{\delta h}{\delta x}|_{\tilde{x}_k}(x - \tilde{x}_k) + \rho_m \tag{2.20}$$

$$x_k = f(\hat{x}_{k-1}, u_{k-1}) + \frac{\delta f}{\delta x}\,|_{\hat{x}_{k-1}}\,(x_{k-1} - \hat{x}_{k-1}) + \rho_\rho \tag{2.21}$$

Con ello, las ecuaciones del EKF se obtienen de la siguiente manera:
Etapa de predicción

$$\tilde{x}_k = f(\hat{x}_{k-1}, u_{k-1}) \tag{2.22}$$

Sustituyendo A por $\frac{\delta f}{\delta x}\,|_{\hat{x}_{k-1}}$ en 2.13:

$$\tilde{P}_k = (\frac{\delta f}{\delta x}\,|_{\hat{x}_{k-1}})\hat{P}_{k-1}(\frac{\delta f}{\delta x}\,|_{\hat{x}_{k-1}})^T + Q_{p_{k-1}} \tag{2.23}$$

Etapa de corrección Empleando la ecuación de medida no lineal para realizar la predicción de las medidas, 2.14 pasa a ser:

$$\hat{x}_k = \tilde{x}_k + K_k(z_k - h(\tilde{x}_k)), \tag{2.24}$$

donde se aprecia que la innovación es, en este caso, $v_k = z_k - h(\tilde{x}_k)$.

Para completar el algoritmo, 2.15, 2.16, 2.17 se modifican mediante $H_k \leftarrow \frac{\delta h}{\delta x} \mid_{\hat{x}_{k-1}}$:

$$\hat{P}_k = (I - K_k \frac{\delta h}{\delta x} \mid_{\hat{x}_{k-1}})\tilde{P}_k \tag{2.25}$$

$$K_k = \tilde{P}_k(\frac{\delta h}{\delta x} \mid_{\hat{x}_{k-1}})^T S_k^{-1} \tag{2.26}$$

$$S_k = R_k + (\frac{\delta h}{\delta x} \mid_{\hat{x}_{k-1}})\tilde{P}_k(\frac{\delta h}{\delta x} \mid_{\hat{x}_{k-1}})^T \tag{2.27}$$

Ecuaciones para el caso de ecuación de medida en forma implícita

No siempre es posible despejar z en la forma de 2.9. En su lugar, a veces sólo se dispone de ecuaciones implícitas del tipo:

$$h(x, z) + \rho_m = c, \tag{2.28}$$

con c vector de constantes. En este caso, las ecuaciones anteriores varían del siguiente modo:

Etapa de predicción No se ve afectada, ya que en ella no influyen las medidas estereoceptivas.

Etapa de corrección Al ser ahora la innovación $v_k = c - h(\tilde{x}_k, z_k)$, la ecuación 2.24 se transforma en:

$$\hat{x}_k = \tilde{x}_k + K_k(c - h(\tilde{x}_k, z_k)) \tag{2.29}$$

En las expresiones 2.15, 2.16, 2.17 ha de efectuarse el cambio $H_k \leftarrow \frac{\delta h}{\delta x} \mid_{\tilde{x}_k, z_k}$. La matriz R_k ha de sustituirse por $(\frac{\delta h}{\delta z} \mid_{\tilde{x}_k, z_k})R_k(\frac{\delta h}{\delta z} \mid_{\tilde{x}_k, z_k})^T$.

$$\hat{P}_k = (I - K_k \frac{\delta h}{\delta x} \mid_{\tilde{x}_k, z_k})\tilde{P}_k \tag{2.30}$$

$$K_k = \tilde{P}_k(\frac{\delta h}{\delta x} \mid_{\tilde{x}_k, z_k})^T S_k^{-1} \tag{2.31}$$

$$S_k = (\frac{\delta h}{\delta z} \mid_{\tilde{x}_k, z_k})R_k(\frac{\delta h}{\delta z} \mid_{\tilde{x}_k, z_k})^T + (\frac{\delta h}{\delta x} \mid_{\tilde{x}_k, z_k})\tilde{P}_k(\frac{\delta h}{\delta x} \mid_{\tilde{x}_k, z_k})^T \tag{2.32}$$

Una forma de comprobar que una innovación es consistente con el modelo consiste en calcular la llamada distancia de Mahalanobis de la misma y determinar si este valor de la adecuación de la medida al estado se encuentra dentro del intervalo de confianza escogido. El cuadrado de la distancia de Mahalanobis se calcula mediante $v_k^T S_k^{-1} v_k$ y también recibe el nombre de *Normalised Innovation Squared*, NIS_k. Este parámetro tiene distribución χ^2 con tantos grados de libertad como medidas independientes haya en el vector de medidas z_k, con lo que la decisión de aceptar o rechazar la medida se hará en base a:

$$v_k^T S_k^{-1} v_k < \chi^2_{dim(v_k),\alpha=confianza} \tag{2.33}$$

Parte II

Desarrollo

Capítulo 3

Arquitectura del sistema

3.1 Plataformas de desarrollo y ejecución

En la realización de este proyecto se ha utilizado una estación de trabajo PC con una configuración de software acorde con las prácticas habituales del grupo de robótica móvil:

- Sistema Operativo Windows XP®
- Lenguaje de programación C++ sobre el entorno de desarrollo MS Visual C++ 6.0® y posteriormente sobre Visual Studio .NET 2005® (Visual Studio 8).

Con ello se consigue la compatibilidad con otros trabajos del grupo y se tiene una buena portabilidad del desarrollo. También se ha comprobado que el paso al sistema operativo Windows Vista® es directo y no plantea ningún tipo de problema.

Algunas de las ventajas del lenguaje C++ son su buena capacidad de cálculo, el hecho de que es un lenguaje orientado a objetos y que se trata de un lenguaje de alto nivel con flexibilidad y potencia expresiva. Todo ello permite reducir el tamaño y la complejidad del código. Además, al tratarse de un lenguaje compilado, presenta una buena eficiencia en tiempos de ejecución frente a los lenguajes interpretados. También cabe destacar que gran parte de los entornos de programación distribuidos por los fabricantes de robots móviles están escritos en este lenguaje. Por supuesto, también es compatible con plataformas de desarrollo libre (como la herramienta Player/Stage/Gazebo, que proporciona software gratuito para el desarrollo de aplicaciones con robots y sensores sin imponer un lenguaje de programación determinado).

Respecto al sistema operativo, el uso de Visual Studio impone utilizar Windows. Este entorno simplifica la creación y compilación de código y, sobre todo,

permite el tratamiento de cuadros de diálogo e interfaces gráficas para aplicaciones Windows de un modo sencillo. No obstante, son muy abundantes los trabajos en robótica móvil desarrollados con Linux® (Player & Stage, por ejemplo, no se puede emplear en Windows).

Se han realizado versiones iniciales en modo consola (sobre todo cuando se empleaba simultáneamente el simulador *MobileSim* del robot Pioneer) y finalmente se ha optado por aplicaciones de tipo MFC (ver 3.2.2 basadas en cuadros de diálogo.

El modo de operación puede ser de tipo *simulación*, *fichero* o *real*. La simulación permite observar en pantalla el movimiento teórico de un robot ficticio ante instrucciones que se le dan por medio del ratón o del teclado. Con el modo fichero se muestra de forma similar el comportamiento del sistema al utilizar unos datos de odometría y de medidas del láser guardados previamente en un fichero. El modo real es el que se emplea con el robot verdadero y también permite visualizar los resultados sobre la interfaz gráfica.

Respecto a la plataforma de ejecución para el modo real, en diferentes fases del ciclo de vida del proyecto se han utilizado los robots Pioneer y Urbano. A este último corresponde la implementado el sistema completo y con él se ha llevado a cabo la mayor parte de las pruebas.

En el caso del Pioneer P3AT, la puesta en funcionamiento se ha basado en comunicación con cable serie entre un computador portátil y el robot.

Como se ha visto en la introducción, el computador base de Urbano es un PC Pentium con sistema operativo GNU/Linux. También cuenta con un computador secundario que es un PC Pentium con Windows XP. La conexión entre la plataforma de desarrollo y el robot se realiza con un ordenador portátil mediante conexión *telnet* a través de cable Ethernet.

3.2 Bibliotecas empleadas

Para el desarrollo del software del proyecto se ha hecho uso de varias bibliotecas existentes.

3.2.1 Standard Template Library (STL)

Biblioteca de C++ que incluye clases contenedoras, algoritmos e iteradores. Las clases contenedoras son patrones (*templates*) que permiten almacenar objetos de tipos muy variados. Los iteradores son una generalización de los punteros y sirven para acceder a los elementos almacenados en los contenedores. Así, se dice que la STL es una biblioteca *genérica*, ya que sus componentes están altamente parametrizados.

La STL incluye un gran número de algoritmos para manipular los datos guardados en los contenedores. Las funciones que implementan estos algoritmos deberán tomar como argumentos tipos de datos coherentes con las operaciones a realizar. El conjunto de requisitos que debe tener un tipo de dato recibe el nombre de *concepto*. En la STL se definen los siguientes conceptos para clasificar los iteradores: `OutputIterator`, `InputIterator`, `ForwardIterator`, `Bidireccional-Iterator`, `RandomAccessIterator`. Algunos de ellos son *refinamientos* de otros, lo que implica una relación similar a la herencia en la programación orientada a objetos.

La clase `vector` es una clase contenedora de tipo secuencial y, como cualquier *template*, puede ser particularizada para contener diferentes tipos de objetos. Se trata de una extensión de los vectores o *arrays* disponibles en C++, con la ventaja de que el número de elementos almacenados puede variar dinámicamente, realizándose la gestión de memoria de forma automática. Esta clase ha sido ampliamente utilizada en el proyecto.

3.2.2 Microsoft Foundation Class Library (MFC)

La biblioteca de clases base de Microsoft encapsula gran parte de la API de Windows en clases de C++. Constituye una importante herramienta para desarrollar aplicaciones Windows con entornos de ventanas, menús, etc. Proporciona, entre otras cosas, un gran número de clases para manejar distintos tipos de ventanas predefinidas y para incluir los controles más comunes en cuadros de diálogo.

En el tipo de proyecto empleado se crea una interfaz gráfica constituida por un cuadro de diálogo principal que hereda de la clase pública `CDialog`. Un editor de recursos disponible en Visual Studio facilita la forma de añadir botones al mismo y la declaración de funciones asociadas a los diferentes eventos que pueden tener lugar sobre ellos.

3.2.3 Open Graphics Library (OpenGL)

Biblioteca de funciones estándar para el dibujo de gráficos 2D y 3D. Permite variar los colores, el tamaño y la posición de los objetos, el grosor de las líneas, los focos de luz, los puntos de vista, etc.

En este proyecto se ha utilizado la clase `COpenGLWnd`, implementada por Diego Rodríguez-Losada, como clase base de una nueva clase llamada `CRobotMoveGLWnd`. `COpenGLWnd` hereda de la clase `CWnd` de las MFC y permite crear una ventana con fondo negro sobre el que se dibuja una cuadrícula de color magenta y los ejes de un sistema de referencia global para el movimiento sobre un plano. También incluye diversas funcionalidades para cambiar el punto de vista por medio del ratón o el teclado y sirve para dibujar diferentes tipos de robots. En la

clase `CRobotMoveGLWnd` se maneja el dibujo de trayectorias, obstáculos, puntos del mapa, polígonos... Se ha reprogramado la función virtual `OnKeyDown` de la clase `COpenGLWnd` para modificar las opciones de visualización desde el teclado y para hacer funcionar al robot en modo teleoperado. Desde ella se realiza también el desplazamiento del dibujo del robot de acuerdo con el movimiento del mismo.

3.2.4 Biblioteca Mathematics

Biblioteca desarrollada por Diego Rodríguez-Losada para facilitar el tratamiento de la geometría y distintos tipos de operaciones. En el proyecto se utiliza principalmente para la definición de puntos y segmentos (clases `Point2D` y `Segment2D`). También se dispone de los ficheros `maths.h`, `maths.cpp`, `matrix.h`, `matrix.cpp`, del mismo autor, para definir matrices y operar con ellas.

3.2.5 Biblioteca Aria

Aria (Advanced Robot Interface for Applications) es una interfaz para el uso de los robots de la firma MobileRobots.Inc que puede usarse sobre las APIs de Linux y Win32. Integra software para establecer las comunicaciones, para controlar la velocidad y la orientación de los robots, para la utilización de diferentes sensores y accesorios... Se trata de una biblioteca escrita en C++, pero también puede ser ampliamente utilizada en Java o Phyton mediante *wrappers*.

En el proyecto se han utilizado únicamente las funciones básicas que permiten la conexión y desconexión sencilla del robot, el envío de comandos de movimiento de bajo nivel (velocidades de avance y giro) y la lectura de los datos proporcionados por los encóders. La mayor parte de estas funciones se encapsulan dentro de la clases `ArRobot` y `ArSimpleConnector`.

El equivalente a las funciones indicadas ha sido implementado en la clase `CPioneer` para no tener dependencias de Aria fuera de dicha clase. Además, se utilizan como variables miembro punteros a void, que primeramente se inicializan a NULL para después hacer un cast a vectores que apuntan a los objetos correspondientes (`ArRobot`, `ArSimpleConnector`). Con esto se evita que al ejecutarse la aplicación sea necesario tener instalado el software de Aria.

3.3 Funciones auxiliares utilizadas

A lo largo del desarrollo del proyecto se han creado algunas funciones de carácter auxiliar para realizar ciertas operaciones. Su declaración y contenido se pueden encontrar en los ficheros `tools.h` y `tools.cpp` Las más destacadas son:

- `AngRango`: se emplea para convertir un ángulo dado en su equivalente dentro del intervalo [-PI, PI]
- `ErrAng`: sirve para calcular el ángulo perteneciente a [-PI, PI] que separa un ángulo de partida de un ángulo de destino por el camino más corto
- `Comp`: se utiliza para realizar la composición de dos transformaciones relativas (Ver Anexo A)
- `J1`: sirve para calcular la matriz jacobiana de una composición de dos transformaciones relativas respecto de la primera variable (Ver Anexo A)
- `J2`: se emplea para obtener la matriz jacobiana de una composición de dos transformaciones relativas respecto de la segunda variable (Ver Anexo A)
- `eye`: sirve para crear una matiz identidad de las dimensiones que queramos
- `InvTrans`: permite calcular la inversión de una transformación relativa entre dos sistemas de referencia (Ver Anexo A)

Las dos primeras funciones resultan de utilidad en el control, la planificación de trayectorias y el control reactivo. El resto se utilizan principalmente para la localización y en el tratamiento de los mapas.

3.4 Estructura de clases

En la figura 3.1 se puede ver el diagrama UML de las clases que conforman el sistema.

En el diagrama no se muestran las dependencias de las bibliotecas anteriormente descritas. Las clases que han sido implementadas en el presente proyecto aparecen en color rosa. El color gris se ha utilizado para las clases que existían previamente. La clase `CProcLaserData` forma parte de este último grupo, pero ha sido ligeramente modificada.

A continuación se explica el uso que se ha hecho de estas clases mientras que las clases nuevas serán tratadas en posteriores capítulos, después de que se estudien los algoritmos incorporados.

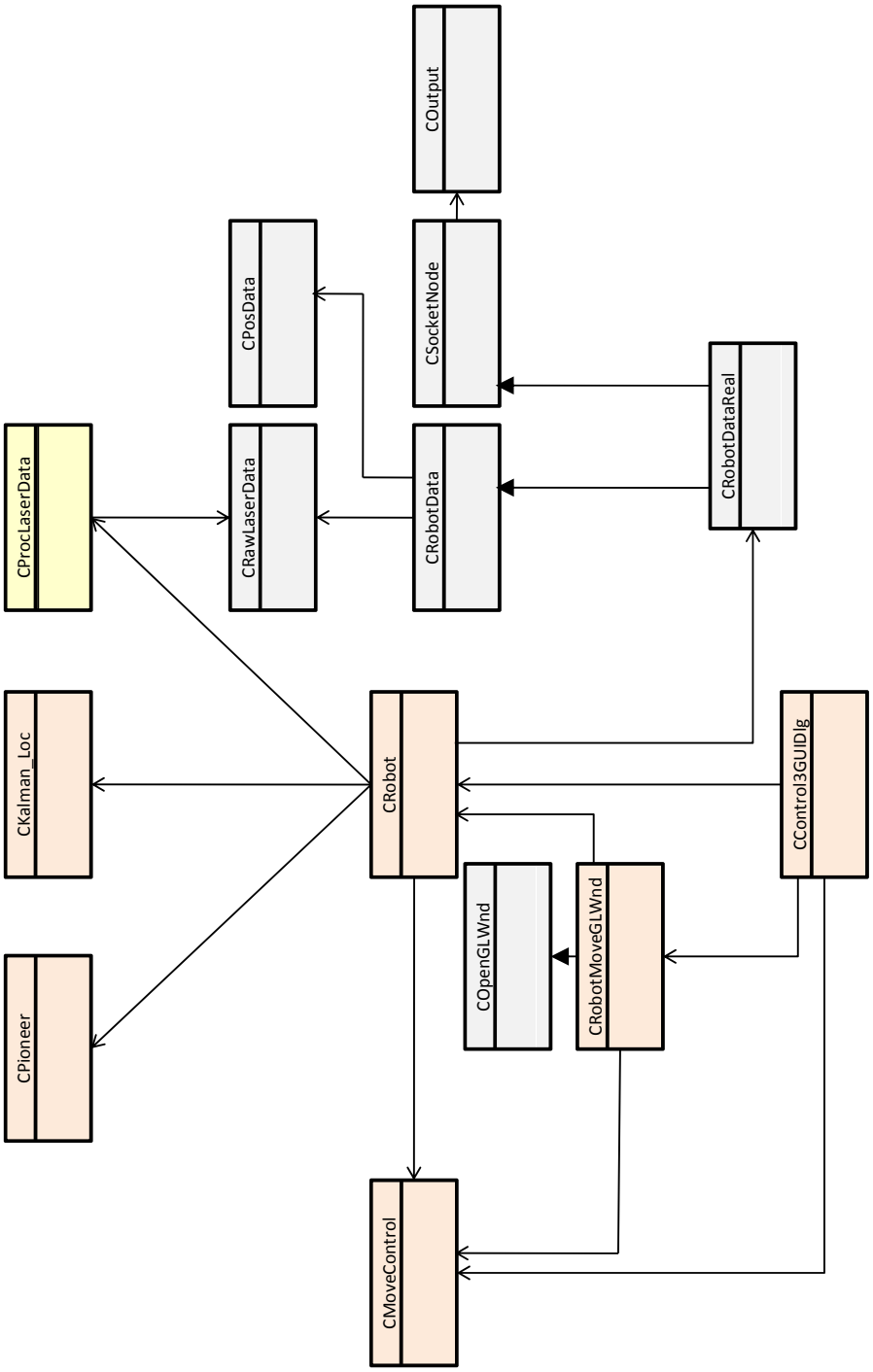

Figura 3.1: Diagrama de clases del sistema

3.4.1 La clase `CRawLaserData`

Esta clase se encarga principalmente de leer y escribir ficheros con los datos procedentes del láser y de ajustar dicha información al formato correspondiente para ser enviada o tras ser recibida por un socket. También define una macro con el máximo número de medidas que puede proporcionar el láser, 361. Sus métodos son los siguientes:

Load

`int Load(FILE* source_file)`
Esta función se emplea para leer de un fichero los datos correspondientes a las medidas del láser.

- `source_file`: fichero del que se lee la información del láser

Los resultados obtenidos sobre el instante en que se efectúa la medida, el identificador del láser, el número de medidas, el ángulo recorrido por éste y su resolución y el máximo alcance del láser se almacenan en variables con nombres representativos: `time_stamp_seconds`, `time_stamp_useconds`, `id_laser`, `num_med`, `scan_angle`, `scan_resolution`, `max_range`...

Estos datos han de ir seguidos de dos puntos, :, y detrás han de aparecer las medidas sucesivas de las distancias asociadas a cada ángulo, que se van guardando en el vector de enteros `med`. Todos los valores deben ir separados por un espacio en blanco para ser leídos correctamente. En el ejemplo de la figura 3.2 puede verse mejor cómo ha de ser el formato de una línea del fichero:

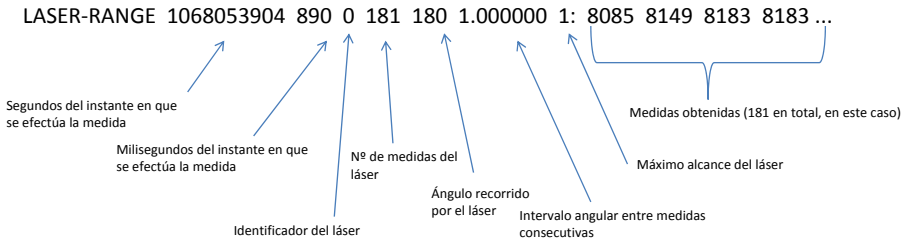

Figura 3.2: Línea de fichero correspondiente a datos del láser

Si hay algún fallo en la lectura de los parámetros o de las medidas se devuelve -1. Si el fichero tiene el formato adecuado y se lee sin ningún problema se devuelve 0.

Save

`void Save(FILE* log_file)`
Esta función se emplea para escribir en un fichero los datos correspondientes a las medidas del láser.

- `log_file`: fichero en el que se escribe la información del láser

Se escribe el término LASER_RANGE seguido del contenido de las variables `time_stamp_seconds`, `time_stamp_useconds`, `id_laser`, `num_med`, `scan_angle`, `scan_resolution)` y `max_range` y dos puntos (':'). Después se escriben los valores de las diferentes medidas almacenadas en el vector `med`. Todos los datos van separados por un espacio en blanco. En resumen, una llamada a esta función escribe una línea de fichero con el formato mostrado en 3.2.

Serialize

`int Serialize(char* cad,int& l,int size) const`

DeSerialize

`int DeSerialize(char* cad,int& l,int size)`

Estas dos funciones no se han utilizado directamente, por lo que no se entra en detalles sobre su funcionamiento.

3.4.2 La clase `CProcLaserData`

Se trata de una clase para el procesamiento y manejo de la información del láser. Sus métodos más utilizados son:

SetOffset

`void SetOffset(float offx, float offy, float offz)`
Esta función se emplea para establecer el offset del láser, es decir, indica la posición del mismo respecto al sistema de referencia local del robot.

- `offx`: desplazamiento del láser sobre la coordenada x del sistema de referencia local del robot
- `offy`: desplazamiento del láser sobre la coordenada y del sistema de referencia local del robot

- `offz`: desplazamiento del láser sobre la coordenada z del sistema de referencia local del robot

En el robot Urbano el offset del láser viene dado por (0.168, 0.0, 1.2) y en el Pioneer P3AT los valores son (0.0, 0.0, 0.4).

Estos parámetros se pasan a las variables miembro `off_x`, `off_y` y `off_z`. Además, en ella se calculan las razones trigonométricas seno y coseno de los ángulos correspondientes a todas las diferentes medidas que pueden tomarse (intervalos de $0,5^{\text{o}}$) y se guardan en sendos vectores `cos_alfa` y `sen_alfa`, de tamaño igual al máximo número de medidas posibles.

DefineData

`int DefineData(const CRawLaserData& d)`
Permite calcular las coordenadas de los puntos correspondientes a las medidas del láser.

- `d`: objeto de la clase `CRawLaserData` cuyos datos se van a procesar

A partir del alcance de las medidas realizadas se ajusta el valor de las distancias hasta los obstáculos sobre cada ángulo de medida (almacenadas en el vector `med`) y el resultado se guarda en un vector de enteros de nombre `fmed`. Con el offset del láser y dichas medidas de distancia se obtienen las coordenadas de cada punto i detectado mediante trigonometría:
med_x$[i]$ = off_x + $fmed[i]$cos_alfa$[istep]$
med_y$[i]$ = off_y + $fmed[i]$sen_alfa$[istep]$
med_z$[i]$ = off_z,
donde el valor de la variable `step` puede ser 1 ó 2, viniendo dado por el número de medidas. De este modo se toman los ángulos de $0,5^{\text{o}}$ en $0,5^{\text{o}}$ o de 1^{o} en 1^{o}. Los puntos 2D que así se obtienen (la coordenada z se mantiene constante) se almacenan en el vector de la STL `v`, miembro de esta clase.

La función devuelve un 0 si el número de medidas es distinto de 181 o de 361 (en cuyo caso no se realiza procesamiento de las medidas, ya que ha de haber algún error) y un 1 cuando el número de medidas es el adecuado y, por lo tanto, se han efectuado todas las operaciones de la definición de datos.

3.4.3 La clase `CPosData`

Esta clase es similar a la clase progCRawLaserData pero se encarga del tratamiento de la información referente a la odometría. Dispone de variables miembro públicas en las que almacenar la posición odométrica y las velocidades del robot (`posx`, `posy`, `posth`, `vel_drive`, `vel_steer`). Sus métodos son:

Load

`int Load(FILE* source_file)`
Esta función se emplea para leer de un fichero los datos correspondientes a las medidas de odometría.

- `source_file`: fichero del que se lee la información de la odometría

Los resultados obtenidos sobre el instante en que se efectúa la medida, la coordenadas x, y y z de la posición, la velocidad de avance y la velocidad de giro se almacenan en las variables correspondientes: `pos_temp`, `pos_time_ms`, `posx`, `posy`, `posth`, `vel_drive` y `vel_steer`.

En el ejemplo de la figura 3.3 se muestra cómo ha de ser el formato de una línea del fichero:

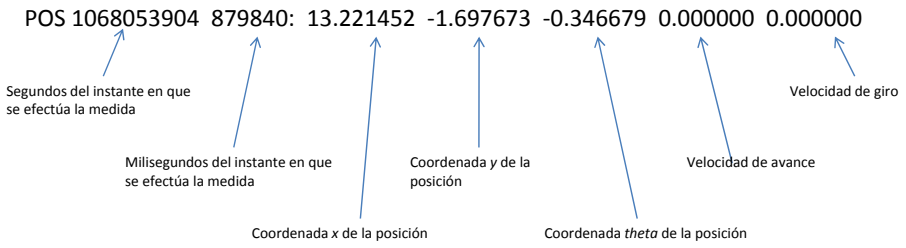

Figura 3.3: Línea de fichero correspondiente a datos de odometría

Si hay algún fallo en la lectura de los parámetros o de las medidas se devuelve -1. Si el fichero tiene el formato adecuado y se lee sin ningún problema se devuelve 0.

Save

`void Save(FILE* log_file)`
Esta función se emplea para escribir en un fichero los datos correspondientes a las medidas de posición.

- `log_file`: fichero en el que se escribe la información de la odometría

Se escribe el término POS seguido del contenido de las variables `pos_temp` y `pos_time_ms` y dos puntos (':'). Después se escriben los valores de `posx`, `posy`, `posth`, `vel_drive` y `vel_steer`. Todos los datos van separados por un espacio en blanco. En resumen, una llamada a esta función escribe una línea de fichero con el formato mostrado en 3.3.

Serialize

```
int Serialize(char* cad,int& l,int size) const
```

DeSerialize

```
int DeSerialize(char* cad,int& l,int size)
```

Al igual que en la clase `CRawLaserData`, estas dos funciones no se han utilizado directamente, por lo que no se entra en detalles sobre su funcionamiento.

3.4.4 La clase `CRobotData`

Esta clase se emplea para definir en cuál de los tres modos posibles se ha de conectar el robot y para procesar la información que permite su funcionamiento en los modos *simulación* y *fichero*. En ella se definen una serie de macros para indicar tipos de conexión (`SIMU_CONN`, `FILE_CONN`, `REAL_CONN`) , tipos de datos (`DATA_CONN_ERROR`, `DATA_NONE`, `DATA_ODOM`, `DATA_LASER`, `DATA_ODOM_LASER`, `DATA_ODOM_INIT`) y tipos de estado (`STATUS_NOT_INIT`, `STATUS_CONNECTION_ERROR`, `STATUS_CONNECTED`. Dentro de la clase se tienen dos variables miembro para el manejo de los datos de la odometría y del láser: `odom_data`, de la clase `CPosData`, y `laser_data`, de la clase `CRawLaserData`. Las principales funciones que se utilizan son las siguientes:

Init

```
bool Init(int typ,const char* source)
```

Esta función se define como `virtual` (para que pueda reescribirse en clases que hereden de ésta) y se emplea para establecer el modo de funcionamiento del robot. En caso de que el modo sea tipo fichero también se abre el fichero del que han de leerse los datos.

- `typ`: modo de funcionamiento del robot
- `source`: nombre del fichero de datos o IP del robot real al que ha de conectarse el sistema

En primer lugar se copia el parámetro `source` en la cadena `source_name`, variable miembro de la clase. Si la conexión es de tipo fichero, se abre el fichero de nombre `source_name` en modo lectura. Si el fichero no existe la función devuelve `false`. En caso contrario se guarda el modo de funcionamiento en la variable entera `type`, también miembro de la clase, con los posibles valores `FILE_CONN` (1), `SIMU_CONN` (2) o `SIMU_REAL` (3) y la función devuelve `true`.

SpeedValues

`void SpeedValues(float vel_drive, float vel_steer)`
Esta función se emplea para ajustar las velocidades al formato y rango adecuados.

- `vel_drive`: velocidad de avance que se le quiere dar al robot
- `vel_steer`: velocidad de giro que se le quiere dar al robot

Si la velocidad de avance es superior a 100 se le da valor 100 y si es menor que 0 se le da valor 0. Con la velocidad de giro se hace lo mismo en el intervalo [-100,100]. Ambos resultados se convierten a tipo `char` y se almacenan en las variables de clase `com_drive` y `com_steer`.

LoadData()

`int LoadData()`
Esta función se emplea para leer de un fichero los datos correspondientes a las medidas de odometría o del láser.
En líneas generales, se lee la primera palabra de la línea del fichero abierto en la llamada a `Init` en la que estemos y se mira si es igual a las cadenas "POS.º "LASER-RANGE". En el primer caso, se llama a la función `Load` sobre el objeto `pos_data` y en el segundo, a la función `Load` del objeto `laser_data`.
Se devuelve el tipo de datos obtenido (`DATA_CONN_ERROR`, `DATA_ODOM`, `DATA_LASER`...).

Simulate

`int Simulate()`
Esta función se emplea para actualizar la información de la odometría y del láser en el modo simulación.
Respecto a la odometría, su actualización se realiza mediante composición de la posición odométrica anterior (accesible mediante `odom_data`) con un vector de incrementos definido a partir de las velocidades `com_drive` y `com_steer`. Esta nueva posición obtenida se emplea para actualizar las variables `posx`, `posy` y `posth` de `odom_data`, de forma que queda disponible para la siguiente llamada.
En lo relativo al láser, en la simulación se establece que el número de medidas es 181, que el ángulo que se cubre es de 180º, que las medidas se toman cada 1º y que el valor de todas ellas es `laser_data.med[i]` $= 8000$.
El valor que se devuelve es siempre `DATA_ODOM_LASER`.

UpdateData

`int UpdateData()`
Esta función es de tipo `virtual` y se emplea para actualizar los datos de la odometría y del láser en los modos simulación y fichero.
Si el tipo de conexión es `FILE_CONN` se efectúa una llamada a la función `LoadData`.
Si el tipo de conexión es `SIMU_CONN` se llama a la función `Simulate`. En estos casos se devuelve el resultado de estas llamadas. Por defecto se devolverá `DATA_CONN_ERROR` (tras realizarse la actualización del estado a `STATUS_CONNECTION_ERROR`).

Otras funciones de la clase que se utilizan para guardar en un fichero los datos de la odometría y del láser son `StartLogData` (crea y abre el fichero en el que escribir los datos), `Log` (que a su vez llama a las funciones `Save` de las variables `odom_data` o `laser_data` según corresponda) y `StopLogData`.

3.4.5 La clase CSocketNode

Es la clase encargada de establecer las comunicaciones con el robot real mediante el uso de sockets. No se ha utilizado directamente.

3.4.6 La clase CRobotDataReal

Como se puede ver en el diagrama 3.1, esta clase hereda de `CRobotData`y `CSocketNode`. Es la clase necesaria para el funcionamiento del robot en modo real. Sus métodos más importantes son los siguientes:

Init

`bool Init(int typ,const char* source)`
Esta función se emplea para establecer el modo de funcionamiento del robot y efectuar la conexión en el caso de modo real.

- `typ`: modo de funcionamiento del robot
- `source`: nombre del fichero de datos o IP del robot real al que ha de conectarse el sistema

En ella se hace una llamada a la función `Init` de la clase `CRobotData`, de forma que sólo es necesario añadir la funcionalidad de establecer conexión en modo real. En ese caso, se leen del parámetro `source` la dirección IP y el puerto de escucha del servidor con el que se ha de realizar la conexión. Una vez se tienen estos datos, se inicia un socket cliente y se lanza el thread que se encarga de la comunicación entre éste y el servidor. Para ello se utilizan funciones de `CSocketNode`. El valor de retorno es siempre `true`.

TransferData

`int TransferData()`
Esta función se emplea para enviar los valores de velocidad al robot y recibir de él la información sobre la odometría y las medidas del láser.
Los datos de velocidad se envían en el formato "TransferData 100 -10", donde 100 sería la velocidad de avance y -10, la velocidad de giro. Los datos de odometría y del láser que se reciben se pasan a las variables `odom_data` y `laser_data` por medio de sus métodos `DeSerialize`, de modo que quedan procesados para su utilización en el resto del programa.

UpdateData

`int UpdateData()`
Esta función se emplea para actualizar los datos de la odometría y del láser en el modo real.
Si el tipo de conexión es `REAL_CONN` se efectúa una llamada a la función `TransferData` y posteriormente a `Log` para guardar los datos obtenidos en un fichero (siempre que previamente se haya empleado `StartLogData`). En caso contrario (`SIMU_CONN` o `FILE_CONN`) se llama a la función `Update` de la clase `CRobotData`.
Se devuelve el tipo de datos obtenido (`DATA_CONN_ERROR`, `DATA_ODOM`, `DATA_LASER`, `DATA_ODOM_LASER` o `DATA_NONE`).

Apéndice A

Transformaciones relativas

La transformación relativa entre el sistema de referencia i y el sistema de referencia j viene dado por

$$\vec{x_{ij}} = [x_{ij} y_{ij} \theta_{ij}]^T \tag{A.1}$$

El operador composición $\oplus$ entre dos transformaciones relativas se define como:

$$\vec{x_{ik}} = \vec{x_{ij}} \oplus \vec{x_{jk}} = \begin{pmatrix} x_{ij} + x_{jk} cos\theta_{ij} - y_{jk} sin\theta_{ij} \\ y_{ij} + x_{jk} sin\theta_{ij} + y_{jk} cos\theta ij \\ \theta_{ij} + \theta_{jk} \end{pmatrix} \tag{A.2}$$

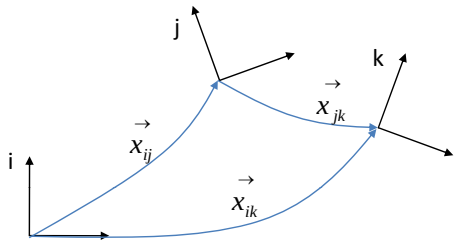

Figura A.1: Composición de transformaciones relativas

Si se calcula su jacobiana respecto de la primera transformación se obtiene:

$$F1(x_{ij}, x_{jk}) = \frac{\delta x_{ik}}{\delta x_{ij}} = \begin{pmatrix} 1 & 0 & -x_{jk} sin\theta_{ij} - y_{jk} cos\theta_{ij} \\ 0 & 1 & x_{jk} cos\theta_{ij} - y_{jk} sin\theta ij \\ 0 & 0 & 1 \end{pmatrix} \tag{A.3}$$

La jacobiana respecto a la segunda transformación será:

$$F2(x_{ij}, x_{jk}) = \frac{\delta x_{ik}}{\delta x_{jk}} = \begin{pmatrix} cos\theta_{ij} & -sin\theta_{ij} & 0 \\ sin\theta_{ij} & cos\theta_{ij} & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{A.4}$$

El operador inversión $\ominus$ de la transformación relativa entre el sistema de coordenadas i y el sistema de coordenadas j se define como:

$$\vec{x_{ji}} = \ominus \vec{x_{ij}} = \begin{pmatrix} -x_{ij}cos\theta_{ij} - y_{ij}sin\theta_{ij} \\ x_{ij}sin\theta_{ij} - y_{ij}cos\theta_{ij} \\ -\theta_{ij} \end{pmatrix} \tag{A.5}$$

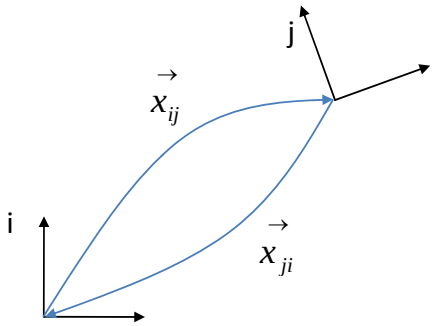

Figura A.2: Inversión de una transformación relativa

Si x_{ij}, x_{jk} y x_{ik} son tres transformaciones relativas tales que $x_{ik} = x_{ij} \oplus x_{jk}$ entonces se verifican las siguientes ecuaciones:

$$\begin{aligned} x_{ik} &\neq x_{jk} \oplus x_{ij} \\ x_{ij} &= x_{ik} \ominus x_{jk} \\ x_{jk} &= \ominus x_{ij} \oplus x_{ik} \\ \ominus x_{ik} = \ominus(x_{ij} \oplus x_{jk}) &= \ominus x_{ij} \oplus (\ominus x_{jk}) \end{aligned} \tag{A.6}$$

Bibliografía

[1] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[2] Alvaro Arranz, Jorge Baliñas, Sebastian Bronte, Josue Garcia, Daniel Gonzalez, Javier Gutierrez, Angel Llamazares, Fernando Rojas, and Victor Sanz. Aplicaciones de robots moviles. Technical report, Universidad de Alcala, 2006.

[3] Diego Rodriguez-Losada, Fernando Matia, Agustin Jimenez, Ramon Galan, and Gerard Lacey. Implementing map based navigation in guido, the robotic smartwalker. In *IEEE International Conference on Robotics and Automation*, 2005.

[4] Diego Rodriguez-Losada. *SLAM Geometrico en Tiempo Real para Robots Moviles en Interiores basado en EKF*. PhD thesis, Universidad Politecnica de Madrid, ETSI Industriales, 2004.

[5] Diego Rodriguez-Losada, Fernando Matia, Agustin Jimenez, Ramon Galan, and Gerard Lacey. Guido, the robotic smartwalker for the frail visually impaired. In *First International Congress on Domotics, Robotics and Remote Assistance for All.DRT4ALL'05*, 2005.

[6] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. MIT Press, 2004.

[7] W. Feiten, R.Bauer, and G. Lawitzky. Robust obstacle avoidance in unknown and cramped environments. In *Proc. IEEE Int. Conf. Robotics and Automation*, 1994.

[8] F. Lamiraux, D. Bonnafous, and O.Lefebvre. Reactive path deformation for nonholonomic mobile robots. *IEEE Transactions on Robotics*, 20(6):967–977, December 2004.

[9] Javier Minguez and Luis Montano. Nearness diagram (nd) navigation: Collision avoidance in troublesome scenarios. *IEEE Transactions on Robotics and Automation*, 20(1), February 2004.

[10] Howie Choset and Keiji Nagatani. Topological simultaneous localization and mapping (SLAM): Toward exact localization without explicit localization. *IEEE Transactions on Robotics and Automation*, 17(2):125–137, April 2001.

[11] B. J. Kuipers and Y.T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Journal of Robotics and Autonomous Systems*, 8:47–63, 1991.

[12] R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *Proc. of the International Joint Conference on Artificial Intelligence*, 1995.

[13] L.P. Kaelbling, A.R. Cassandra, and J.A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1996.

[14] Peter S. Maybeck. *Stochastic Models, Estimation, and Control*, volume 1. Academic Press, 1979.

[15] Joris De Schutter, Jan De Geeter, Tine Lefebvre, and Herman Bruyninckx. *Kalman Filters: A Tutorial*, 1999.

[16] Javier Minguez, Javier Osuna, and Luis Montano. A "divide and conquer"strategy based on situations to achieve reactive collision avoidance in troublesome scenarios. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[17] Jesus Gonzalez. Simulacion y control de un robot movil. Proyecto fin de carrera, DISAM, 2004.

[18] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, 1988.

[19] Helmut Kopka and Patrick W.Daly. *Guide to LaTeX*. Addison Wesley, 2004.

[20] Teach yourself visual c++ in 21 days: `http://newdata.box.sk/bx/c`.

[21] Mobilerobots.inc: `http://www.mobilerobots.com`.

[22] Departamento de automatica de la etsii: http://www.disam.upm.es.

[23] Ayuda oficial de microsoft para la programacion: http://www.msdn.microsoft.es.

[24] Player project: http://www.playerstage.sourceforge.net.