

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

Copyright ©2007 by Paloma de la Puente Yusty

Índice general

I Preliminares	1
1 Introducción	2
1.1 Preámbulo	2
1.2 Antecedentes	5
1.2.1 Proyectos previos	5
1.2.2 Robots	10
1.3 Marco del proyecto	12
1.4 Objetivos del proyecto	12
1.5 Alcance del proyecto	12
2 Estado del Arte	14
2.1 Navegación, planificación de trayectorias y control reactivo . .	14
2.1.1 Control de movimiento	14
2.1.2 Aptitudes para la navegación: planificación y reacción .	15
2.2 Localización y construcción de mapas con un robot móvil . .	17
2.2.1 Tipos de mapas	19
2.2.2 SLAM con técnicas probabilísticas	23
2.2.3 Localización probabilística basada en mapas	25
2.2.4 Filtro de Kalman	27
2.2.5 Filtro Extendido de Kalman	29
II Desarrollo	32
3 Arquitectura del sistema	33
3.1 Plataformas de desarrollo y ejecución	33
3.2 Bibliotecas empleadas	34
3.2.1 Standard Template Library (STL)	35
3.2.2 Microsoft Foundation Class Library (MFC)	35
3.2.3 Open Graphics Library (OpenGL)	36
3.2.4 Biblioteca Mathematics	36

3.2.5	Biblioteca Aria	36
3.3	Funciones auxiliares utilizadas	37
3.4	Estructura de clases	38
3.4.1	La clase CRawLaserData	40
3.4.2	La clase CProcLaserData	41
3.4.3	La clase CPosData	43
3.4.4	La clase CRobotData	44
3.4.5	La clase CSocketNode	46
3.4.6	La clase CRobotDataReal	46
4	Localización y mapas	48
4.1	Aplicación del EKF a la localización del robot: descripción del algoritmo	48
4.2	Consideraciones sobre el coste computacional de la localización	51
4.3	Algoritmo de borrado de puntos dinámicos del mapa	52
4.4	La clase CKalman_Loc	54
4.4.1	KalmanPos	54
4.4.2	KalmanUpdate	54
4.4.3	GuardarMapa	55
4.4.4	LeerMapa	55
4.4.5	PuntoEnPol	55
4.5	Otras funciones de la clase CProcLaserData	56
4.5.1	CalculaPol	56
4.5.2	Split	56
4.5.3	ConviertePol	56
4.5.4	PolAng	57
4.6	Pruebas y resultados	57
4.6.1	Pruebas en modo <i>fichero</i>	57
4.6.2	Pruebas en modo <i>real</i>	66
5	Control de movimiento, planificación de trayectorias y control reactivo	68
5.1	Control de movimiento	68
5.2	Planificación de trayectorias	76
5.3	Control Reactivo	78
5.4	La clase CMoveControl	81
5.4.1	DefineDest	81
5.4.2	Dist2Tray	82
5.4.3	FindPoint	83
5.4.4	GetCommand	83
5.4.5	DefineTrajectory	84

5.4.6	<code>SmoothTrajectory</code>	84
5.4.7	<code>ComputeElasticTray</code>	85
5.4.8	<code>DivideTrajectory</code>	85
5.5	Pruebas y resultados	86
5.5.1	Pruebas en modo <i>simulación</i>	86
5.5.2	Pruebas en modo <i>real</i>	90
6	Integración	98
6.1	La clase <code>CRobot</code>	98
6.1.1	<code>ProcessData</code>	99
6.2	La clase <code>CControl3GUIDlg</code>	100
6.3	Interfaz de usuario	101
6.3.1	Funcionalidad de los botones del diálogo	101
6.3.2	Selección de opciones mediante ratón o teclado	105
6.4	Pruebas y resultados	106
6.4.1	Pruebas en modo <i>simulación</i>	106
III	Información Complementaria	114
7	Conclusiones y trabajos futuros	115
7.1	Conclusiones	115
7.2	Líneas futuras	117
A	Transformaciones relativas	118
B	EKF para medidas de distancia	120
C	Hardware utilizado	123
C.1	Plataforma móvil B21r	123
C.2	Plataforma móvil Pioneer P3-AT de MobileRobots/ActivMedia Robotics	125
C.3	Escáner láser LMS200	127
D	Estructura de Descomposición del Proyecto (EDP)	129

Índice de figuras

1.1	Robot cortacésped Automower Husqvarna	3
1.2	Robot aspirador RoboMaxx	4
1.3	Rover sobre la superficie de Marte	4
1.4	Proyecto Panorama	5
1.5	Proyecto RM-III	6
1.6	Proyecto EVS	7
1.7	Proyecto MobiNet	7
1.8	Proyecto Blacky	8
1.9	Proyecto WebFair	8
1.10	Proyecto Guido	9
1.11	Proyecto Urbano	9
1.12	Robot ROBUTER	10
1.13	Robot BLACKY	11
1.14	Robot Urbano	11
1.15	Robot Guido	12
2.1	Mapas métricos geométricos	20
2.2	Descomposición en celdillas exactas	21
2.3	Descomposición en celdillas fijas	21
2.4	Mapa métrico discretizado	22
2.5	Mapas topológicos basados en el diagrama de Voronoi	22
2.6	Mapas topológico y semántico	22
2.7	Trayectorias odométrica y corregida mediante localización de Markov en un mapa de ocupación de celdillas [1]	26
2.8	Representación topológica de un entorno de oficina	26
3.1	Diagrama de clases del sistema	39
3.2	Línea de fichero correspondiente a datos del láser	41
3.3	Línea de fichero correspondiente a datos de odometría	43
4.1	Representación de la ecuación de medida del robot	50

4.2	Ángulos a medir para ver si un punto está dentro de un polígono no convexo	53
4.3	Primer experimento con datos del laboratorio	58
4.4	Segundo experimento con datos del laboratorio	59
4.5	Tercer experimento con datos del laboratorio	60
4.6	Cuarto experimento con datos del laboratorio	61
4.7	Primer experimento con datos de Intel	63
4.8	Segundo experimento con datos de Intel	64
4.9	Tercer experimento con datos de Intel	65
4.10	Primer experimento con datos del Museo de las Ciencias Príncipe Felipe de Valencia	66
4.11	Segundo experimento con datos del Museo de las Ciencias Príncipe Felipe de Valencia	67
5.1	Velocidad de avance y giro en el sistema de referencia local del robot	68
5.2	Control de la orientación del robot	69
5.3	Nuevos errores incorporados al regulador	72
5.4	Diferentes trayectorias entre dos puntos	73
5.5	Medida de la distancia del robot al segmento de trayectoria en el que se halla	74
5.6	Ángulo a regular para que el robot se aproxime a la trayectoria definida en dos casos diferentes	75
5.7	Cálculo de coordenadas de los puntos del arco cuando $\text{erro_ang} > 0$	78
5.8	Cálculo de coordenadas de los puntos del arco cuando $\text{erro_ang} < 0$	79
5.9	Trayectorias suaves conseguidas	80
5.10	Definición del punto p y otras magnitudes para el punto de la trayectoria $i = 0$	81
5.11	Los obstáculos del tramo 1 no deben afectar al 2 ni los del tramo 4 a los tramos 3 y 5.	82
5.12	Desplazamiento que sufre cada punto de la trayectoria	82
5.13	Diagrama de flujo simplificado de parte del proceso efectuado en <code>GetCommand</code>	84
5.14	Puntos de un segmento de la trayectoria original y del resultado de la llamada a <code>DivideTrajectory()</code>	86
5.15	Primer experimento de planificación de trayectorias y control de movimiento	87
5.16	Segundo experimento de planificación de trayectorias y control de movimiento	88

5.17	Tercer experimento de planificación de trayectorias y control de movimiento	92
5.18	Cuarto experimento de planificación de trayectorias y control de movimiento	93
5.19	Experimento previo para el control reactivo	94
5.20	Aumento de la máxima desviación permitida	95
5.21	Primer experimento de control reactivo	95
5.22	Primer experimento de movimiento con robot real	96
5.23	Segundo experimento de movimiento con robot real	97
6.1	Interfaz de usuario	101
6.2	Deformación de trayectorias definidas en el laboratorio de DISAM-UPM	109
6.3	Deformación de trayectorias en un mapa obtenido con borrado de puntos dinámicos	110
6.4	Trayectoria deformada en el camino de ida y nuevas deformaciones en el camino de vuelta	111
6.5	Primer experimento para ver el comportamiento del sistema con el robot Pioneer	112
6.6	Segundo experimento para ver el comportamiento del sistema con el robot Pioneer	113
A.1	Composición de transformaciones relativas	118
A.2	Inversión de una transformación relativa	119
B.1	Corrección de la posición en un movimiento circular uniforme mediante medidas de distancia a las observaciones	121
B.2	Corrección de la posición en un movimiento circular uniforme mediante las coordenadas de las observaciones	122
C.1	Plataforma móvil B21r de iRobot	123
C.2	Plataforma móvil Pioneer P3-AT	125
C.3	Panel de control	125
C.4	Láser LMS200 de SICK	127
D.1	Estructura de Descomposición del Proyecto	131

Índice de cuadros

5.1	Regulador inicial para el control de movimiento	70
5.2	Regulador para el control de movimiento	71
C.1	Características del robot B21r de iRobot	124
C.2	Características del robot Pioneer P3-AT de MobileRobots . .	126
C.3	Características del láser SICK LMS-200	128

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 impulsión 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 datos necesarios. Aunque no son estrictamente imprescindibles, los robots móviles suelen incorporar sensores proprioceptivos 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 estereoeceptivos 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 (figura 1.1) suelen utilizar algún medio que delimita 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 (figura 1.2) 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



Figura 1.2: Robot aspirador RoboMaxx

y tienen una cabeza y un cuello para que las cámaras puedan tener una mejor perspectiva (figura 1.3). Estas cámaras se utilizan para la construcción de mapas que les permitan guiarse por el territorio así como para enviar imágenes a la Tierra. Estos robots también cuentan con 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 robots 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 los

apartados 1.2.1, página 8, y 1.2.2, página 10, así como el Anexo ??), 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).



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 (figura 1.5). 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).

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 (figura 1.7). 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 1.6: Proyecto EVS



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 (figura 1.8). 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



Figura 1.8: Proyecto Blacky



Figura 1.9: Proyecto WebFair

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 1.9).

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 (figura 1.10). 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 (figura 1.11). El módulo de navegación fue



Figura 1.10: Proyecto Guido



Figura 1.11: Proyecto Urbano

desarrollado por Diego Rodríguez-Losada en el marco de su Tesis Doctoral [4].

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, conciencia 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 (figura 1.12). 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).

URBANO

Plataforma B21r del fabricante iRobot (figura 1.14). 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 LMS200 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 1.13: Robot BLACKY



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 ??.

GUIDO

Guido (figura 1.15) 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.

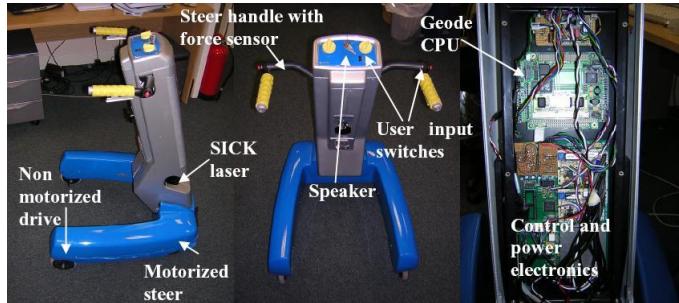


Figura 1.15: Robot Guido

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. La eficiencia de este sistema y el grado de conocimiento del entorno que proporcione han de permitir que el robot lleve a cabo sus tareas con precisión y flexibilidad. 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 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 se ha empleado hasta el momento la parte del proyecto relativa al control de movimiento y planificación de trayectorias 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.

La estructura de descomposición del proyecto (EDP), la programación temporal seguida para llevar a cabo las tareas correspondientes a las distintas fases del mismo así como la estimación del presupuesto se incluyen en los apéndices.

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 existe 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, \dots, 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 sobre un mapa del entorno puede deberse a errores en la localización, a imprecisiones en el mapa utilizado o a obstáculos dinámicos no contemplados en dicho mapa.

Algunas soluciones (por ejemplo la mostrada en [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” y 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. Un poco más en profundidad, 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 muchos casos no puede efectuarse la correspondencia de forma determinista y ha de emplearse una formulación probabilista. Algunas mejoras para abordar este problema y descartar las asociaciones erróneas se han presentado en [?].
- 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 (capítulo 7).

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.

Este tipo de mapas es ampliamente utilizado en entornos estructurados

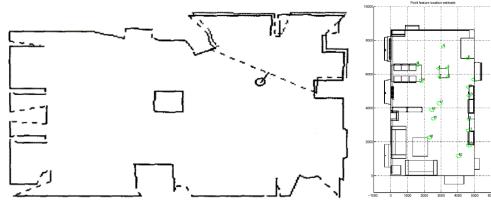


Figura 2.1: Mapas métricos geométricos

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.

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.

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.

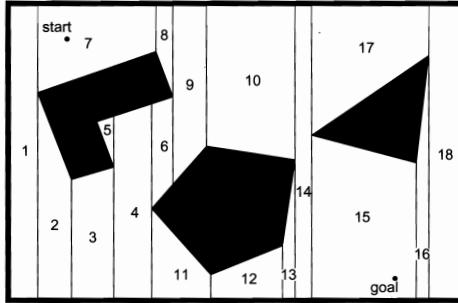


Figura 2.2: Descomposición en celdillas exactas

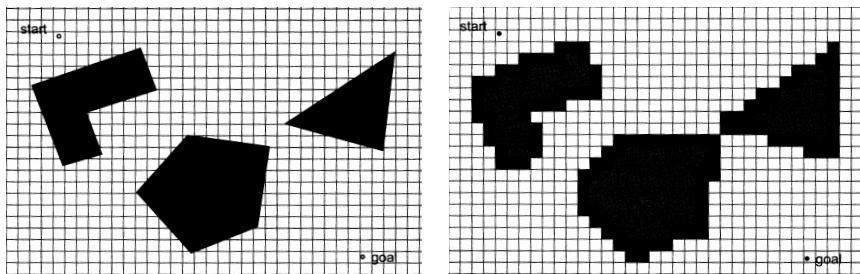


Figura 2.3: Descomposición en celdillas fijas

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.

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 2.4: Mapa métrico discretizado

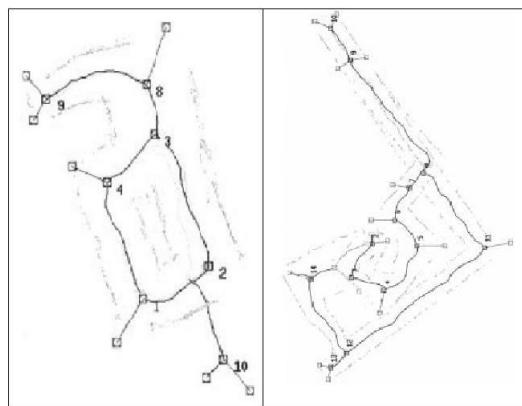


Figura 2.5: Mapas topológicos basados en el diagrama de Voronoi

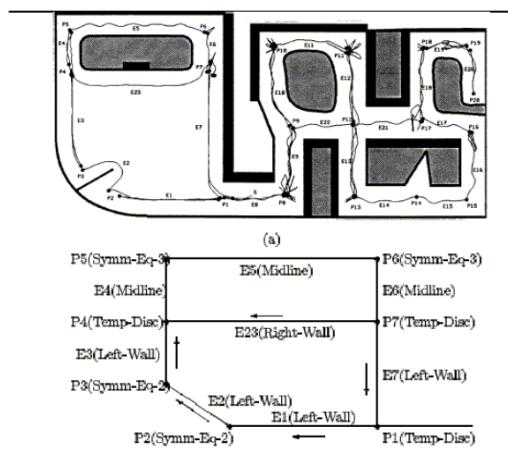


Figura 2.6: Mapas topológico y semántico

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:

$$u_1, z_1, u_2, z_2 \dots u_t, z_t \quad (2.1)$$

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

$$x_t \sim s_t, m_t \quad (2.2)$$

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.

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

$$\begin{aligned} z^t &= z_1, z_2 \dots z_t \\ u^t &= u_1, u_2 \dots u_t \end{aligned} \quad (2.4)$$

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:

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

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

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

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:

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

Con lo que finalmente queda:

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

Esta expresión 2.8 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, $p(s_t | u_t, s_{t-1})$, y la de las medidas efectuadas por el sistema estereoeceptivo, $p(z_t | s_t, m_t)$.

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 todos 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. La solución adoptada se encuadra dentro de la categoría de métodos de Máxima Probabilidad Incremental. La idea básica del algoritmo así llamado consiste en construir un único mapa según van llegando los datos, sin mantener ninguna noción sobre la incertidumbre del mismo o de la posición del robot en cada instante. Únicamente se determinan la posición y el mapa más probables en cada momento. La principal ventaja del algoritmo es su mayor simplicidad y el menor tiempo de procesamiento y capacidad de almacenamiento que requiere frente a otras soluciones. Sin embargo, no presenta un buen comportamiento a la hora de cerrar bucles dado que al no guardarse ninguna información sobre la incertidumbre del mapa, una estimación realizada no puede corregirse a partir de datos posteriores. Una posible forma de hacer frente a este problema es utilizar algoritmos híbridos. Los algoritmos híbridos ofrecen una aproximación intermedia entre la obtención del mapa más probable sin mantener una noción de la incertidumbre del mismo ni de la posición del robot (algoritmo de Máxima Probabilidad Incremental), y la estimación de la función de probabilidad del mapa completo (solución SLAM-EKF), que conserva la totalidad de la información sobre la incertidumbre. En definitiva lo que hacen este tipo de algoritmos es mantener la incertidumbre en la posición del robot de una u otra forma y estimar el mapa más probable simplificando la ecuación 2.8 para la localización en un mapa dado.

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.

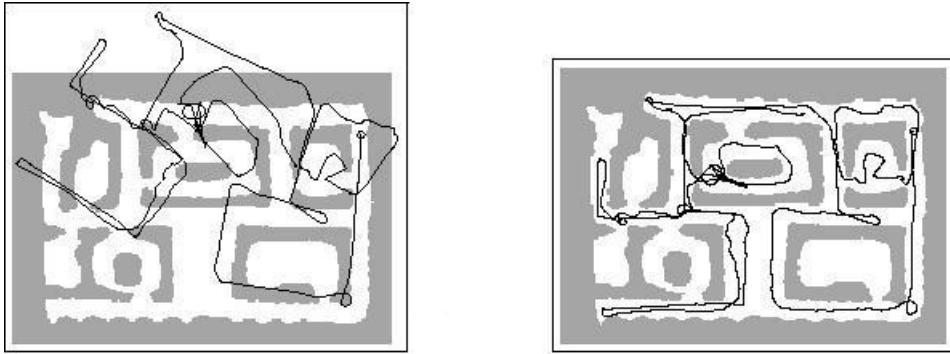


Figura 2.7: Trayectorias odométrica y corregida mediante localización de Markov en un mapa de ocupación de celdillas [1]

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].

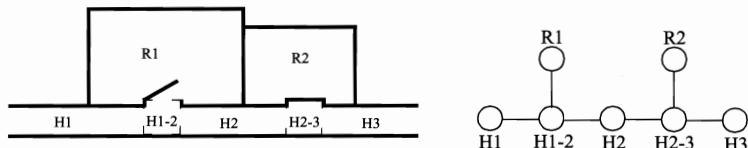


Figura 2.8: Representación topológica de un entorno de oficina

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 extendida 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 definen 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 alta-

mente 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:

$$z_k = H_k x_k + \rho_m \quad (2.9)$$

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:

$$x_k = Ax_{k-1} + Bu_{k-1} + \rho_p, \quad (2.10)$$

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_p 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:

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

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:

$$\tilde{x}_k - x_k = A(\hat{x}_{k-1} - x_{k-1}) - \rho_p \quad (2.12)$$

y la matriz de covarianza de x_k queda:

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

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

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

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

con

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

$$S_k = R_k + H_k \tilde{P}_k H_k^T \quad (2.17)$$

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:

$$z = h(x) + \rho_m \quad (2.18)$$

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

cuyas ecuaciones linealizadas en la estimación más reciente son:

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

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

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

Etapa de predicción

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

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

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

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

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

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 $H_k \leftarrow \frac{\delta h}{\delta x} |_{\hat{x}_{k-1}}$:

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

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

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

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:

$$h(x, z) + \rho_m = c, \quad (2.28)$$

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 estereoeceptivas.

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

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

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

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

$$K_k = \tilde{P}_k \left(\frac{\delta h}{\delta x} |_{\tilde{x}_k, z_k} \right)^T S_k^{-1} \quad (2.31)$$

$$S_k = \left(\frac{\delta h}{\delta z} |_{\tilde{x}_k, z_k} \right) R_k \left(\frac{\delta h}{\delta z} |_{\tilde{x}_k, z_k} \right)^T + \left(\frac{\delta h}{\delta x} |_{\tilde{x}_k, z_k} \right) \tilde{P}_k \left(\frac{\delta h}{\delta x} |_{\tilde{x}_k, z_k} \right)^T \quad (2.32)$$

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:

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

Parte II

Desarrollo

Capítulo 3

Arquitectura del sistema

La arquitectura del sistema describe la estructura global del mismo. Incluye su división en componentes, el reparto de responsabilidades entre dichos componentes, la colaboración entre ellos y el flujo de control.

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 con el protocolo *telnet* a través de cable Ethernet.

Para más información acerca de estos robots y del resto de hardware utilizado puede consultarse el Anexo C.

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`, `Bidirectional- 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*. Se encuentra publicada bajo licencia GPL.

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 matriz identidad de las dimensiones que queremos
- `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.

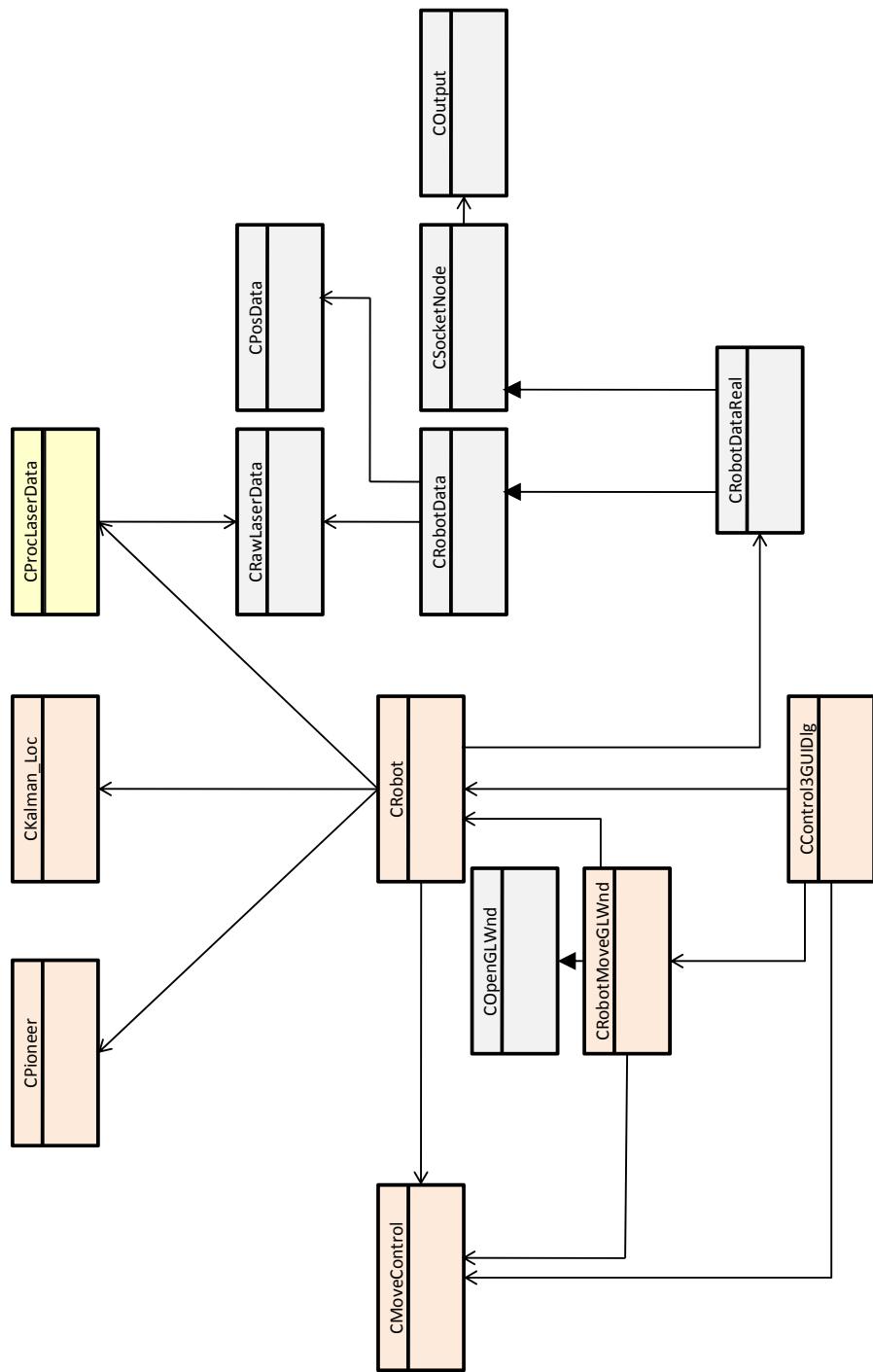


Figura 3.1: Diagrama de clases del sistema

En el diagrama no se muestran las dependencias de las bibliotecas anteriormente descritas. Como puede observarse, se ha realizado un diseño modular con relaciones claras entre sus elementos. 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.

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:

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.

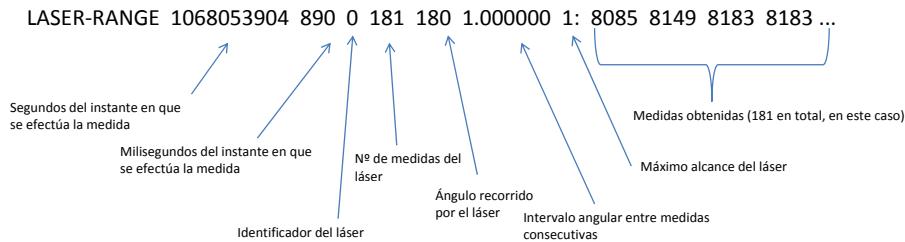


Figura 3.2: Línea de fichero correspondiente a datos del láser

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°) 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:

$$\begin{aligned} \text{med_x}[i] &= \text{off_x} + f\text{med}[i]\cos\text{_alfa}[istep] \\ \text{med_y}[i] &= \text{off_y} + f\text{med}[i]\sin\text{_alfa}[istep] \\ \text{med_z}[i] &= \text{off_z}, \end{aligned}$$

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° en 0, 5° o de 1° en 1°. 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:

POS 1068053904 879840: 13.221452 -1.697673 -0.346679 0.000000 0.000000

Segundos del instante en que se efectúa la medida

Milisegundos del instante en que se efectúa la medida

Coordenada x de la posición

Coordenada y de la posición

Coordenada θ de la posición

Velocidad de avance

Velocidad de giro

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." "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 180°, que las medidas se toman cada 1° y que el valor de todas ellas es `laser_data.med[i] = 8000` (límite de alcance).

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**).

Capítulo 4

Localización y mapas

El sistema de localización realizado no incorpora la construcción simultánea de mapas de los tipos descritos en el capítulo de Estado del Arte sino que elabora y luego va actualizando un mapa de puntos a partir de las observaciones correspondientes a las medidas del láser. Son los puntos que ya forman parte del mapa los que se utilizan para la localización en cada instante. Se trata de un método de *máxima probabilidad incremental* basado en el filtro extendido de Kalman.

4.1 Aplicación del EKF a la localización del robot: descripción del algoritmo

En el problema que se trata, el estado queda definido por la posición del robot, $[x_R, y_R, \theta_R]^T$. Las medidas de la odometría (incrementales, luego referenciadas al sistema de coordenadas local del robot) constituyen las entradas al sistema,

$$\vec{u} = \begin{pmatrix} u_x \\ u_y \\ \theta_u \end{pmatrix},$$

en cada instante (se suprimen sus subíndices de tiempo para simplificar la notación; siempre se utilizan las últimas medidas disponibles). De esta forma, el modelo de estado identificable con 2.19 es:

$$\vec{x}_{R_k} = \vec{x}_{R_{k-1}} \oplus \vec{u} + \vec{\rho}_\rho, \quad (4.1)$$

siendo \oplus el operador composición de transformaciones relativas.

Etapa de predicción De acuerdo con esto, la predicción del estado dada

por 2.22 será:

$$\begin{pmatrix} \tilde{x}_{R_k} \\ \tilde{y}_{R_k} \\ \tilde{\theta}_{R_k} \end{pmatrix} = \begin{pmatrix} \hat{x}_{R_{k-1}} + u_x \cos \hat{\theta}_{R_{k-1}} - u_y \sin \hat{\theta}_{R_{k-1}} \\ \hat{y}_{R_{k-1}} + u_x \sin \hat{\theta}_{R_{k-1}} + u_y \cos \hat{\theta}_{R_{k-1}} \\ \hat{\theta}_{R_{k-1}} + \theta_u \end{pmatrix} \quad (4.2)$$

Para la matriz de covarianza de x_k la ecuación 2.23 varía ligeramente ya que lo que se conoce no es la matriz de covarianza del proceso conjunto, Q_p , sino la covarianza del ruido presente en la odometría \vec{u} . Si esta variable gaussiana se representa por $\vec{u}_k \sim N(\hat{\vec{u}}_k, Q)$ (tomaremos un valor constante para la covarianza del ruido en el incremento de odometría medido a partir de los datos de los encoders) la predicción de la covarianza del estado será:

$$\tilde{P}_k = F_x \hat{P}_{k-1} F_x^T + F_u Q F_u^T, \quad (4.3)$$

donde $F_x = \frac{\delta f}{\delta x} |_{\tilde{x}_k}$ y $F_u = \frac{\delta f}{\delta u} |_{\tilde{x}_k}$ (se emplea la mejor estimación del estado disponible hasta el momento).

El resultado del cálculo de estas matrices jacobianas se incluye en el Anexo A.

El algoritmo se inicia partiendo de $\hat{x}_0 = 0$, $\hat{P}_0 = 0$.

Etapa de corrección Las medidas, observaciones realizadas por el láser, se relacionan con el estado de forma implícita mediante composición con él y comparación con los puntos del mapa que se utilice. Como puede verse en la figura 4.1, la expresión que liga idealmente el estado con la observación i por medio del punto del mapa j asociado a ella será:

$$\vec{h}_{ij} = \vec{x}_R \oplus \vec{o}_i - \vec{l}_j = \vec{0}, \quad (4.4)$$

Aplicando la definición del operador \oplus (Anexo A), 4.4 queda:

$$\vec{h}_{ij} = \begin{pmatrix} -x_{lj} + x_R + o_{ix} \cos \theta_R - o_{iy} \sin \theta_R \\ -y_{lj} + y_R + o_{ix} \sin \theta_R + o_{iy} \cos \theta_R \end{pmatrix} = \vec{0}, \quad (4.5)$$

como puede obtenerse a partir de la figura 4.1.

Denotando $H_{x_{ijk}} = \frac{\delta \vec{h}_{ij}}{\delta \vec{x}} |_{\tilde{x}_k, z_k}$ y $H_{z_{ijk}} = \frac{\delta \vec{h}_{ij}}{\delta \vec{z}} |_{\tilde{x}_k, z_k}$ para cada iteración k, la matriz de covarianza de una innovación individual viene dada por 2.32:

$$S_{ijk} = H_{x_{ijk}} \tilde{P}_k H_{x_{ijk}}^T + H_{z_{ijk}} R H_{z_{ijk}}^T \quad (4.6)$$

Como puede verse en 4.6, la covarianza en las medidas del láser también se tomará como constante.

El cálculo de la distancia de Mahalanobis de la innovación h_{ijk} mediante esta matriz S_k permitirá realizar la asociación de cada observación i al punto

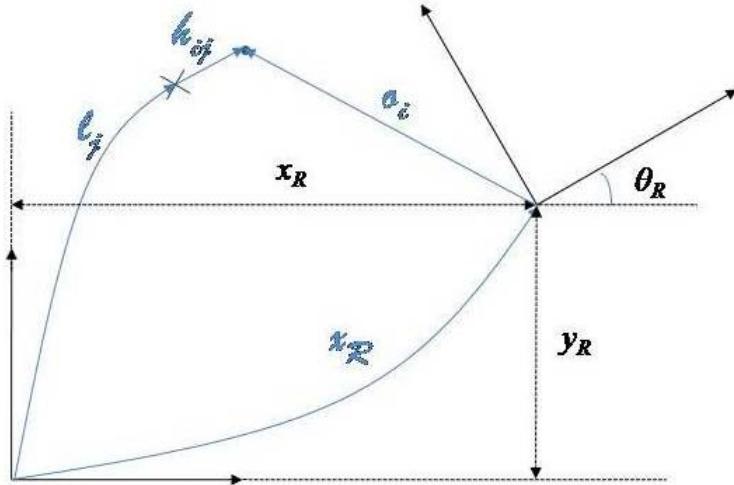


Figura 4.1: Representación de la ecuación de medida del robot

del mapa j para el cual esa distancia sea mínima. Si dicha asociación supera el test de Mahalanobis, se tendrá en cuenta para corregir la posición del robot, en caso contrario se podrá añadir la observación como nuevo punto del mapa. De entre todas las matrices $h_{ij}, H_{x_{ij}}$ y $H_{z_{ij}}$ calculadas para cada observación asociada a un punto del mapa se utilizarán únicamente las correspondientes a dicho punto. Estas matrices se denominarán $h_{i\min}, H_{xi\min}, H_{zi\min}$.

Según se van asociando observaciones, se tienen más medidas que utilizar. Por ello, las dimensiones de las matrices h, H_x, H_z y R irán creciendo al irse añadiendo datos. Su tamaño final en cada iteración determinará las dimensiones de S y K .

Para un número t de observaciones asociadas en la iteración k , se tiene:

$$\begin{aligned}\dim(h_k) &= 2t \times 1, \\ \dim(H_{x_k}) &= 2t \times 3, \\ \dim(H_{z_k}) &= 2t \times 2t, \\ \dim(R_k) &= 2t \times 2t\end{aligned}$$

Con lo que la matriz S global calculada a partir de ellas por medio de 2.32 será:

$$S_k = H_{x_k} \tilde{P}_k H_{x_k}^T + H_{z_k} R H_{z_k}^T \quad (4.7)$$

y tendrá dimensión $(2t \times 2t)$.

La matriz de Kalman del sistema, viene dada por 2.31:

$$K_k = \tilde{P}_k H_{x_k}^T S_k^{-1} \quad (4.8)$$

por lo que su tamaño será $(3 \times 2t)$.

Finalmente, los valores corregidos de la posición del robot y su covarianza se obtienen a partir de la predicción de acuerdo con 2.29 y 2.30:

$$\hat{x}_k = \tilde{x}_k - K_k h_k \quad (4.9)$$

$$\hat{P}_k = (I - K_k H_{x_k}) \tilde{P}_k \quad (4.10)$$

4.2 Consideraciones sobre el coste computacional de la localización

El coste computacional de la etapa de corrección del algoritmo puede llevar a tiempos de procesamiento altos, lo que conduce a un comportamiento del sistema poco eficaz. A continuación se realiza un análisis del mismo y se identifican las principales fuentes de retardo en el cómputo de la posición corregida.

Para n puntos en el mapa y t observaciones adquiridas por el láser, el coste computacional de la etapa de asociación de datos es de orden $O(nt)$.

El cálculo de la matriz de ganancia de Kalman se realiza mediante 4.8. Al tener la matriz $\tilde{P}_k H_{x_k}^T$ dimensión $(3 \times 2t)$ y S dimensión $(2t \times 2t)$, el coste computacional para hallar la matriz K será $O(t^3)$.

Sin variar el coste computacional, puede verse que una forma sencilla de reducir el tiempo de cálculo consiste en reducir el número de observaciones a tener en cuenta. Las observaciones que se encuentran muy alejadas del robot (distancias superiores a 6m) directamente son ignoradas. El láser proporciona medidas a intervalos de 1° (181 medidas en total) o bien a intervalos de 0.5° (361 medidas en total). Utilizando sólo una de cada dos observaciones asociadas se logra una reducción de tiempo considerable sin que se aprecien pérdidas en los resultados de la localización.

Las operaciones que inicialmente consumían más tiempo en el procesamiento de las observaciones eran la declaración de las matrices h_{ij} , $H_{x_{ij}}$ y $H_{z_{ij}}$ para todas las innovaciones y el ir ampliando las matrices h , H_x , H_z y R por filas o por columnas a medida que se asociaban más datos, con la consiguiente reserva de memoria cada vez.

En relación al primer punto, se optó por realizar una única declaración al principio de cada proceso de corrección e inicializar las componentes de las matrices que permanecen constantes para todas las observaciones.

Respecto al segundo punto, también se hace una reserva inicial de memoria para el tamaño máximo de las matrices h y H_x y según se van asociando observaciones se van incluyendo componentes en ellas. Las matrices H_z de las sucesivas asociaciones se van guardando en un vector de la STL.

La última mejora introducida se centra en la obtención de la matriz S (definida al comienzo de la etapa de corrección sin especificación de tamaño) y se basa en el hecho de que H_z y R son matrices diagonales por bloques, por lo que tienen muchas componentes nulas. El primer término de S se calcula a partir de la matriz $Pht = \tilde{P}_k H_{x_k}^T$, que puede ser posteriormente utilizada en el cálculo de K . A continuación se va sumando a cada bloque (2×2) de la diagonal principal el producto $H_z R H_z^T$ de cada observación, efectuándose así la operación únicamente sobre las componentes que sufren alguna modificación.

4.3 Algoritmo de borrado de puntos dinámicos del mapa

Para eliminar los puntos que se añadieron al mapa en un momento dado pero dejan de corresponder a asociaciones con las medidas del láser (dejan de observarse), se construye un polígono a partir de estas medidas de modo que los puntos que quedan dentro del mismo pueden ser borrados.

La obtención del polígono se hace de forma recursiva, empleándose en líneas generales el siguiente procedimiento:

- Se toma el segmento que une la primera observación con la última.
- Se mira cuál es la observación intermedia que está más separada del segmento.
- Si esta separación es suficiente, se repite el proceso entre la primera observación y la más separada y entre ésta y la última.
- Cuando un segmento no tiene ninguna observación a más distancia que el umbral establecido, se añaden sus extremos como vértices del polígono.

Cuando alguna observación está fuera del alcance de medida, la subdivisión se realiza entre la primera observación y la anterior a aquélla y entre la

observación siguiente a la medida lejana y la última observación tomada. En el caso de que una observación esté muy separada de la anterior, se realiza el procedimiento de subdivisión para el bloque de puntos previo a la separación y después para el bloque de puntos siguientes a esa separación.

A partir del polígono determinado en cada instante, se eliminan aquellos puntos de su interior que no están muy próximos a la frontera. Dada la forma en que se construye el polígono, esta última condición es necesaria como medida de precaución para no borrar puntos del mapa indebidamente.

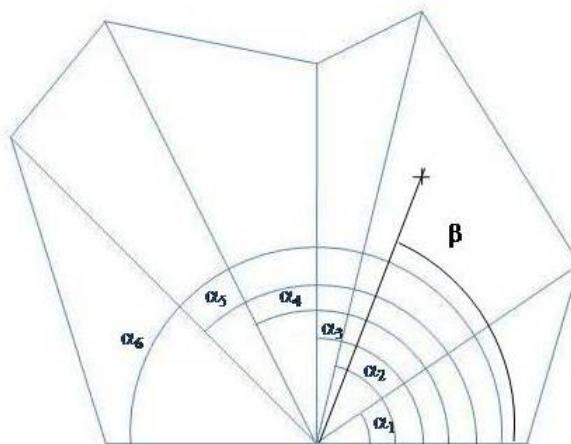


Figura 4.2: Ángulos a medir para ver si un punto está dentro de un polígono no convexo

El polígono será, por lo general, no convexo. El algoritmo utilizado para ver si un punto se encuentra en su interior requiere medir los ángulos $\alpha_1, \alpha_2, \dots, \alpha_v$, indicados en la figura 4.2. Para cada punto del mapa (tras calcular sus coordenadas en el sistema de referencia local en el láser), se halla el valor del ángulo β y se mira entre qué dos alphas está comprendido (α_1 y α_2 en el caso de la figura). Con esto se seleccionan dos vértices que servirán para ver si el punto del mapa está dentro del polígono. Se mide la distancia del robot al punto del mapa y se compara con la distancia entre el robot y el vértice de los dos anteriores que sea más cercano al mismo(criterio conservador). Si resulta ser menor, será que el punto se encuentra dentro del polígono.

De este modo, para cada punto del mapa sólo ha de calcularse un ángulo. Esto permite una velocidad de ejecución aceptable.

4.4 La clase CKalman_Loc

Esta clase se ha creado para gestionar las operaciones relacionadas con el tratamiento de los mapas de puntos que se utilizan y, fundamentalmente, la localización del robot en cada instante. Aparte del constructor, contiene las funciones que se describen brevemente a continuación.

4.4.1 KalmanPos

```
void KalmanPos(float inc_ odom_x, float inc_ odom_y, float inc_ odom_theta)
```

En esta función se realizan los cálculos correspondientes a la fase de predicción del algoritmo de localización.

- **inc_odom_x**: incremento de odometría medido sobre el eje x del sistema local en el robot
- **inc_odom_y**: incremento de odometría medido sobre el eje y del sistema local en el robot
- **inc_odom_theta**: variación incremental odométrica en la orientación del robot

La posición resultante se almacena en una variable miembro de tipo **Matrix** llamada **pos_robot_kalman**.

4.4.2 KalmanUpdate

```
void KalmanUpdate(const std::vector<Point2D>& v)
```

Esta función efectúa la fase de corrección de la localización a partir de la posición obtenida mediante la predicción y de las observaciones que se le pasan.

- **v**: vector de la STL que contiene los puntos correspondientes a las observaciones realizadas por el láser

El resultado se guarda nuevamente en la variable **pos_robot_kalman**.

Inicialmente el mapa (vector de la STL que almacena objetos de la clase **Point2D**) se encontrará vacío si no se utiliza uno ya realizado. Como ya se ha explicado, las observaciones que no se asocian a ningún punto del mapa se añaden como puntos del mismo si así se selecciona mediante la variable miembro **actualiza** (controlable desde el cuadro de diálogo mediante un *Check box*)

Para cada uno de los puntos del mapa se comprueba si están en el interior del polígono de observaciones mediante 4.4.5. En caso afirmativo, y si así lo indica la variable miembro `borrar`, se eliminan del vector `mapa`. Esto es posible porque el polígono es previamente calculado en cada iteración mediante un objeto de la clase `CProcLaserData` perteneciente a la clase `CRobot`. Sus puntos se almacenan en un vector de la STL al cual apunta un puntero miembro de `CKalmanLoc`, `pol`.

4.4.3 GuardarMapa

void GuardarMapa(LPTSTR path)

Esta función sirve para guardar un fichero de puntos del mapa de forma que si se dispone de un buen mapa no sea necesario realizar uno nuevo. También resulta útil para examinar los mapas generados o para agilizar algunas pruebas.

- `path`: *path* en el que guardar un fichero de texto que contendrá dos columnas con las coordenadas de los puntos de un mapa

4.4.4 LeerMapa

void LeerMapa(LPTSTR path)

Esta función se emplea para descargar un mapa ya creado (por medio de 4.4.3, generalmente).

- `path`: *path* de un fichero de texto ya existente en el que se definen los puntos de un mapa mediante dos columnas de coordenadas

A medida que se van leyendo puntos se guardan en el vector `mapa`.

Para que las dos funciones anteriores se ejecuten adecuadamente en Visual Studio .NET 2005 en los *Settings* o *Propiedades* del proyecto no debe marcarse la utilización del juego de caracteres Unicode.

4.4.5 PuntoEnPol

int PuntoEnPol(int k)

Esta función sirve para determinar si un punto del mapa se encuentra en el interior del polígono al que apunta el puntero `pol`.

- **k**: índice del punto del mapa cuya pertenencia al polígono se quiere determinar

Si el punto *k* del vector `mapa` está en el interior del polígono de observaciones de acuerdo con los criterios establecidos, se devuelve un 1 para que sea eliminado del mapa. En caso contrario la función devuelve un 0.

4.5 Otras funciones de la clase `CProcLaserData`

4.5.1 `CalculaPol`

```
void CalculaPol()
```

Esta función se utiliza para crear un polígono envolvente de las observaciones realizadas por el láser. Inicializa el número de puntos que lo forman a 0 y después hace una llamada a la función recursiva `Split`, que se describe a continuación.

4.5.2 `Split`

```
void Split(int primer, int segun)
```

Función recursiva que permite ir obteniendo los segmentos que formarán el polígono con las propiedades mencionadas.

- **primer**: índice de la observación a partir de la cual se van a hallar nuevos segmentos
- **segun**: índice de la observación hasta la cual se calculan segmentos

Se trata de la implementación del proceso descrito al principio de 4.3.

Estas dos funciones no han sido realizadas dentro del proyecto. Se incluyen en este capítulo en lugar de en el anterior para que pueda comprenderse mejor su utilización.

4.5.3 `ConviertePol`

```
void ConviertePol(Matrix pos)
```

Esta función calcula las coordenadas de los vértices del polígono en el sistema de referencia global.

- **pos**: posición del robot. Se emplea la mejor estimación disponible en cada iteración.

Se utiliza principalmente para trazar el dibujo del polígono y para medir distancias de los puntos del mapa a los lados del mismo.

4.5.4 PolAng

```
void PolAng(Matrix pos)
```

Método que se emplea para calcular los ángulos α de la figura 4.2.

- **pos**: posición del robot. Se emplea la mejor estimación disponible en cada iteración.

Se aplica un coeficiente de acercamiento de los vértices del polígono al robot y posteriormente se calcula el ángulo α de cada vértice. Estos ángulos se almacenan en un vector **angulos** de la STL variable miembro de la clase.

4.6 Pruebas y resultados

Las pruebas relativas a la localización y a la construcción de mapas de puntos se han efectuado en los modos fichero y real. En simulación las medidas del láser se consideran con alcance constante, por lo que no aportan información que permita la localización.

4.6.1 Pruebas en modo *fichero*

En estas pruebas, se han empleado datos de odometría y del láser obtenidos con diferentes robots y en diferentes entornos por Diego Rodríguez-Losada y otros investigadores que han trabajado con él. Se ha estudiado el funcionamiento del algoritmo implementado por medio de ficheros que contienen esa información sensorial sin procesar. En ellos se alternan líneas del tipo representado en la figura 3.3 (datos de la odometría) y líneas del tipo representado en la figura 3.2 (datos de medidas del láser). Estas pruebas resultan de mucha utilidad ya que utilizan gran variedad de datos y permiten evaluar el funcionamiento del sistema en entornos muy distintos y con medidas de diferentes características.

Utilización de datos tomados en el laboratorio de DISAM-UPM

En este caso, los datos empleados contienen 181 medidas del láser cada vez. Debe recordarse que, por motivos de coste computacional, el algoritmo de localización emplea una de cada dos de ellas. Como estos datos fueron obtenidos con el robot Urbano, no es necesario modificar el *offset* del láser utilizado en el programa.

1. En esta prueba se han utilizado los siguientes parámetros:

- desviación típica del ruido de la odometría introducida en el filtro de Kalman: $\sigma_{odom} = 0,1$
- desviación típica del ruido de las medidas del láser: $\sigma_{med} = 0,3$
- ruido adicional: $\sigma_{extra} = 0$
- borrado de puntos dinámicos del mapa: *borrar = false*
- incorporación de nuevos puntos al mapa: *actualiza = true*

Resultados:

En la siguiente figura se puede ver el resultado de la ejecución del programa con las condiciones indicadas. Las sucesivas posiciones del robot según los datos de odometría disponibles serían las representadas en color verde, mientras que la trayectoria corregida mediante el filtro de Kalman es la mostrada en color rojo. Los puntos del mapa creado aparecen en color negro.

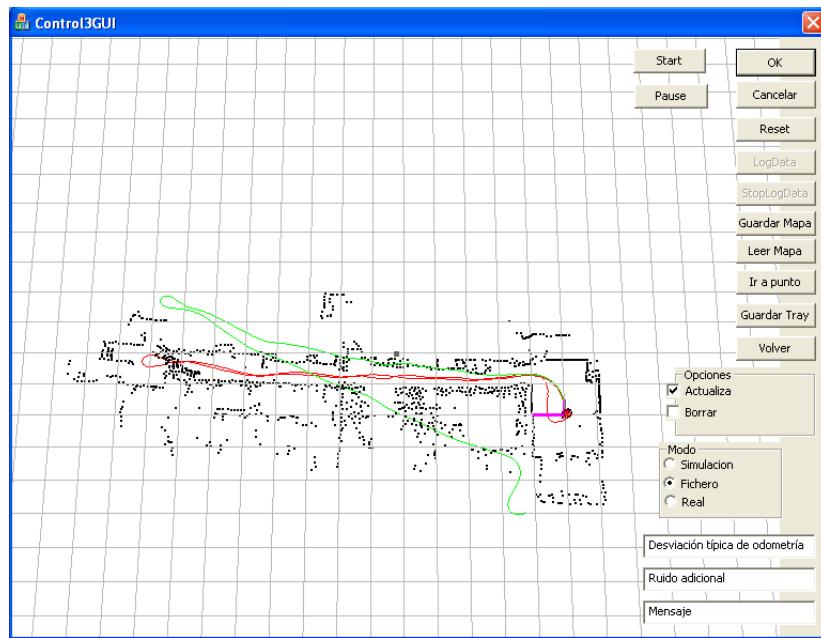


Figura 4.3: Primer experimento con datos del laboratorio

Se puede apreciar el buen funcionamiento del algoritmo, que permite situar el robot adecuadamente en el mapa y ofrece un valor de posición final muy similar al de la posición de partida del robot.

El tiempo total empleado en la obtención del mapa y en el cálculo de la trayectoria corregida es: 3.54min.

2. En esta prueba se han utilizado los siguientes parámetros:

- desviación típica del ruido de la odometría introducida en el filtro de Kalman: $\sigma_{odom} = 0,3$
- desviación típica del ruido de las medidas del láser: $\sigma_{med} = 0,3$
- ruido adicional: $\sigma_{extra} = 0$
- borrado de puntos dinámicos del mapa: *borrar* = *false*
- incorporación de nuevos puntos al mapa: *actualiza* = *true*

Resultados:

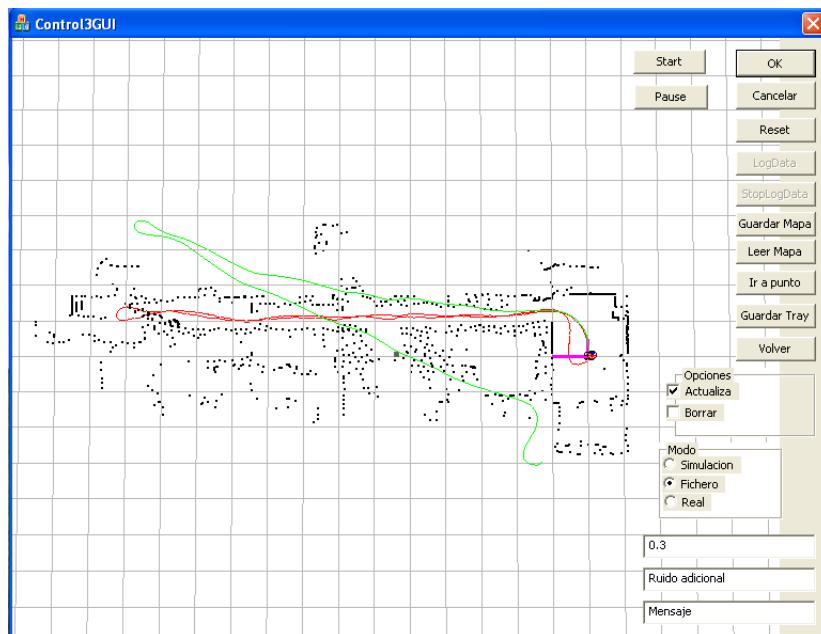


Figura 4.4: Segundo experimento con datos del laboratorio

Al no ser excesivamente malos los datos de la odometría, no se logran demasiadas mejoras en la localización tras otorgar una mayor importancia a las medidas del láser frente a dichos datos. Sin embargo, puede verse que la parte izquierda del mapa obtenido en este caso es más precisa.

El tiempo total empleado en la obtención del mapa y en el cálculo de la trayectoria es: 3.22min. La disminución del tiempo de ejecución es resultado de la mejor asociación de observaciones a puntos del mapa, que hace que éste tenga un menor tamaño.

3. En esta prueba se han utilizado los siguientes parámetros:

- desviación típica del ruido de la odometría introducida en el filtro de Kalman: $\sigma_{odom} = 0,3$
- desviación típica del ruido de las medidas del láser: $\sigma_{med} = 0,3$
- ruido adicional: $\sigma_{extra} = 0$
- borrado de puntos dinámicos del mapa: *borrar* = *true*
- incorporación de nuevos puntos al mapa: *actualiza* = *true*

Resultados:

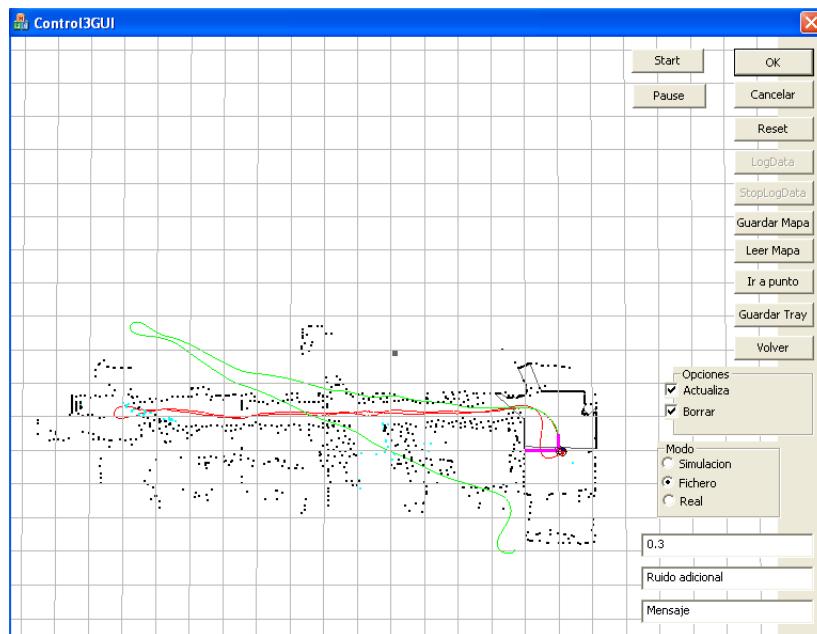


Figura 4.5: Tercer experimento con datos del laboratorio

Puede verse que cuando se tomaron estos datos había algunas personas moviéndose cerca del recorrido realizado por el robot. Esta opción de ejecución permite borrar del mapa los puntos correspondientes a las posiciones que fueron ocupando (puntos de color cyan en la figura). La localización no resulta afectada de manera significativa por el borrado de dichos puntos. En la figura queda representado el polígono envolvente de las medidas del láser correspondiente a la última posición del robot.

El tiempo total empleado en la obtención del mapa y en el cálculo de la trayectoria corregida es: 3.63min. El cálculo de los puntos del polígono, el de los ángulos necesarios para aplicar el algoritmo de borrado y el análisis de los

puntos del mapa no asociados a cada observación en las sucesivas iteraciones (llamadas al método `KalmanUpdate`) provocan un cierto retardo en la ejecución del programa. Este tiempo, no obstante, es considerablemente inferior al correspondiente a otros algoritmos para examinar si un punto pertenece o no a un polígono (algoritmo de Jordan, algoritmo radial...). Además, cada observación nueva ha de compararse con un número menor de puntos del mapa, con lo que en algunos casos el retardo puede quedar prácticamente compensado.

4. En esta prueba se han utilizado los siguientes parámetros:

- desviación típica del ruido de la odometría introducida en el filtro de Kalman: $\sigma_{odom} = 0,6$
- desviación típica del ruido de las medidas del láser: $\sigma_{med} = 0,3$
- ruido adicional: $\sigma_{extra} = 0,005$
- borrado de puntos dinámicos del mapa: `borrar = false`
- incorporación de nuevos puntos al mapa: `actualiza = true`

Resultados:

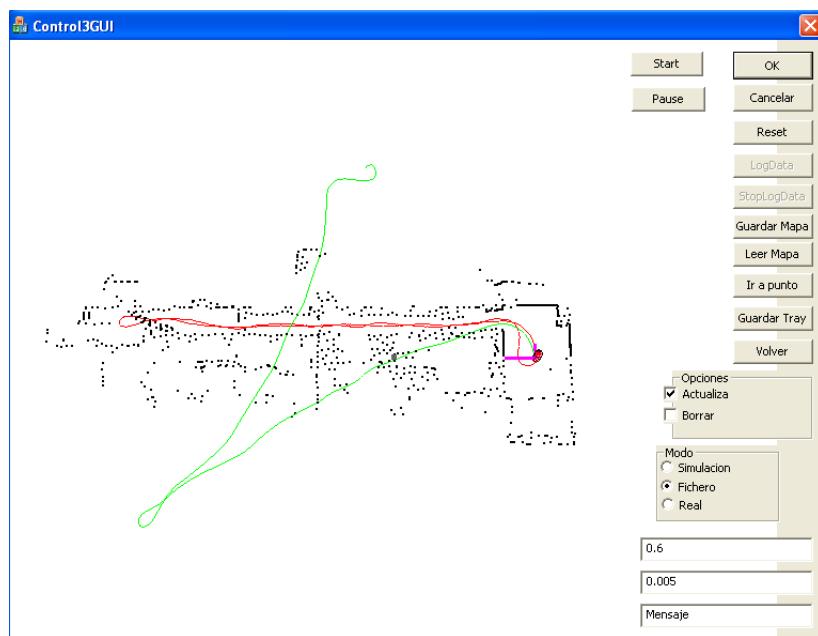


Figura 4.6: Cuarto experimento con datos del laboratorio

Al inyectarse algo de ruido añadido en las medidas odométricas, la posición del robot estimada en base a dichas medidas está muy alejada de la real.

Sin embargo, mediante el algoritmo de localización se sigue obteniendo un buen resultado con sólo volver a incrementar el parámetro considerado en el EKF como varianza del ruido de la odometría.

El tiempo total empleado en la obtención del mapa y en el cálculo de la trayectoria corregida es: 3.15min. Tras tenerse en cuenta que la varianza del ruido de la odometría debe ser algo superior, se mejora la asociación de datos y con ello disminuye el tiempo de procesamiento.

Utilización de datos de Intel

Con estos datos, el número de medidas del láser disponibles en cada momento es 361. Esto repercute en el tiempo de procesamiento.

1. En esta prueba se han utilizado los siguientes parámetros:

- desviación típica del ruido de la odometría introducida en el filtro de Kalman: $\sigma_{odom} = 0,1$
- desviación típica del ruido de las medidas del láser: $\sigma_{med} = 0,3$
- ruido adicional: $\sigma_{extra} = 0$
- borrado de puntos dinámicos del mapa: *borrar = false*
- incorporación de nuevos puntos al mapa: *actualiza = true*

Resultados:

Se puede observar que las medidas odométricas presentan un alto grado de error. Por ello, es muy probable que la varianza introducida en el filtro de Kalman para dichas medidas sea demasiado pequeña y, por lo tanto, la elaboración del mapa no sea la correcta.

El tiempo total empleado en la obtención del mapa y en el cálculo de la trayectoria corregida es: 6.8min.

2. En esta prueba se han utilizado los siguientes parámetros:

- desviación típica del ruido de la odometría introducida en el filtro de Kalman: $\sigma_{odom} = 0,8$
- desviación típica del ruido de las medidas del láser: $\sigma_{med} = 0,3$
- ruido adicional: $\sigma_{extra} = 0$
- borrado de puntos dinámicos del mapa: *borrar = false*
- incorporación de nuevos puntos al mapa: *actualiza = true*

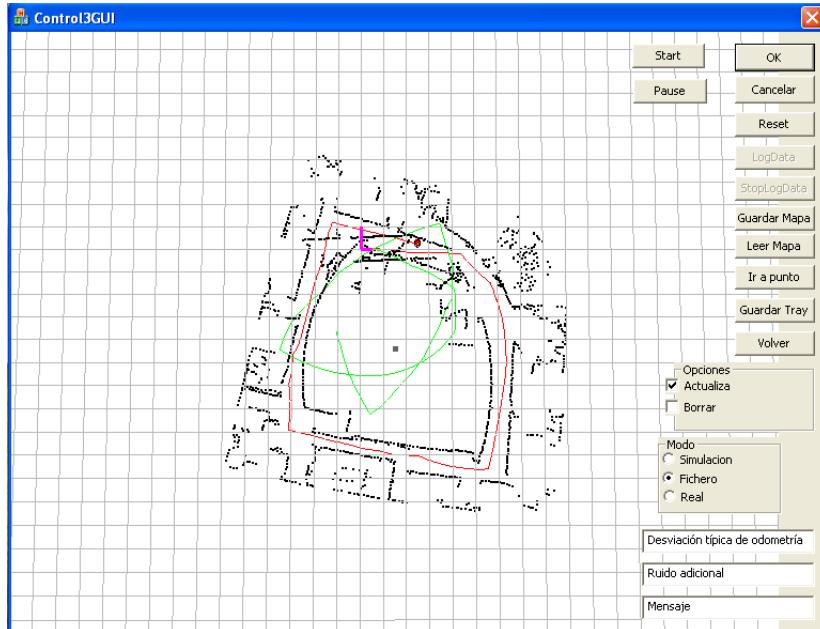


Figura 4.7: Primer experimento con datos de Intel

Resultados:

En este caso, la varianza del ruido de las medidas de odometría estimada para la utilización del filtro de Kalman resulta más adecuada y, como puede observarse, conduce a un resultado excelente a pesar de la poca calidad de los datos odométricos disponibles.

El tiempo total empleado en la obtención del mapa y en el cálculo de la trayectoria corregida es: 5.71min.

3. En esta prueba se han utilizado los siguientes parámetros:

- desviación típica del ruido de la odometría introducida en el filtro de Kalman: $\sigma_{odom} = 1$
- desviación típica del ruido de las medidas del láser: $\sigma_{med} = 0,3$
- ruido adicional: $\sigma_{extra} = 0$
- borrado de puntos dinámicos del mapa: *borrar = true*
- incorporación de nuevos puntos al mapa: *actualiza = true*

Resultados:

El tiempo total empleado en la obtención del mapa y en el cálculo de la trayectoria corregida es: 6.61min.

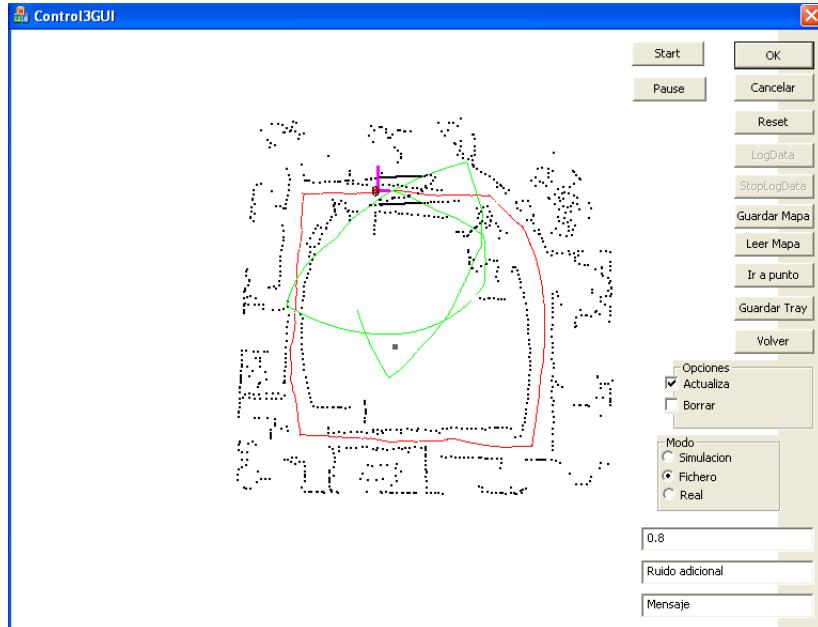


Figura 4.8: Segundo experimento con datos de Intel

Utilización de datos del Museo de las Ciencias Príncipe Felipe de Valencia

Estos datos incluyen 181 medidas del láser en cada momento. Por esta razón su procesamiento resulta más ágil que el del caso anterior. No obstante, dada la gran extensión del mapa elaborado y la exhaustiva exploración del entorno para la realización del mismo, el tiempo total consumido es notablemente superior por serlo también la duración del recorrido de toma de datos.

1. En esta prueba se han utilizado los siguientes parámetros:

- desviación típica del ruido de la odometría introducida en el filtro de Kalman: $\sigma_{odom} = 0,9$
- desviación típica del ruido de las medidas del láser: $\sigma_{med} = 0,3$
- ruido adicional: $\sigma_{extra} = 0$
- borrado de puntos dinámicos del mapa: *borrar = false*
- incorporación de nuevos puntos al mapa: *actualiza = true*

Resultados:

El tiempo total empleado en la obtención del mapa y en el cálculo de la trayectoria corregida es: 26.54min.

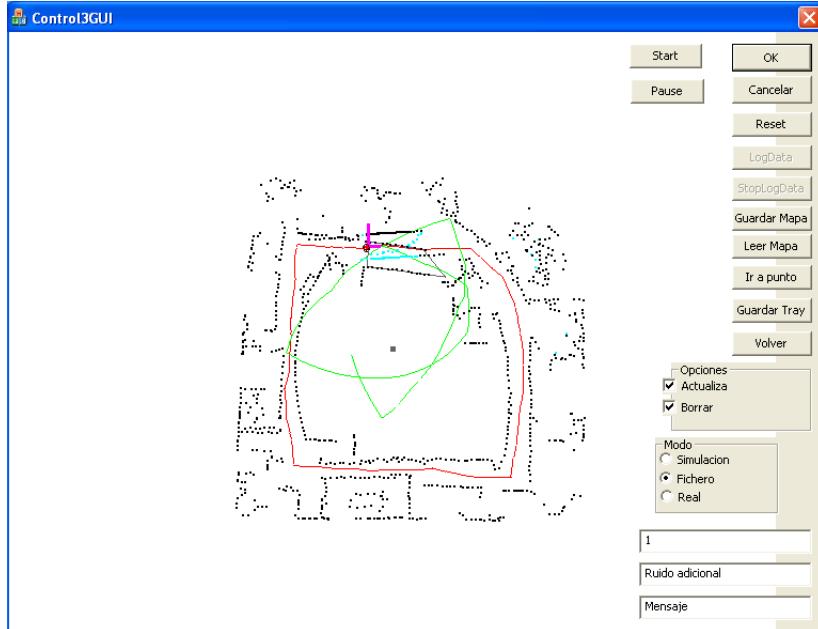


Figura 4.9: Tercer experimento con datos de Intel

2. En esta prueba se han utilizado los siguientes parámetros:

- desviación típica del ruido de la odometría introducida en el filtro de Kalman: $\sigma_{odom} = 0,9$
- desviación típica del ruido de las medidas del láser: $\sigma_{med} = 0,3$
- ruido adicional: $\sigma_{extra} = 0$
- borrado de puntos dinámicos del mapa: $borrar = true$
- incorporación de nuevos puntos al mapa: $actualiza = true$

Resultados:

El tiempo total empleado en la obtención del mapa y en el cálculo de la trayectoria corregida es: 36.01min.

Como puede observarse, la presencia de personas en el museo afecta a la construcción del mapa y a la localización. Esto puede corregirse mediante el borrado de puntos del mapa, con lo que se logra un resultado realmente bueno. A pesar de la gran longitud de la trayectoria seguida, con el correspondiente incremento de errores en las medidas de los encoders, el sistema mantiene una correcta estimación de la posición del robot hasta el final del recorrido efectuado.

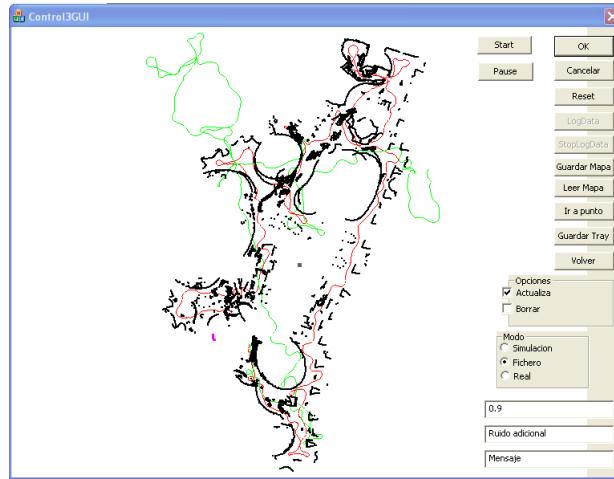


Figura 4.10: Primer experimento con datos del Museo de las Ciencias Príncipe Felipe de Valencia

4.6.2 Pruebas en modo *real*

Como ya se ha mencionado, en estas pruebas se ha trabajado con el robot móvil Urbano. A continuación se muestran algunos de los mapas obtenidos en el laboratorio del departamento llevando al robot en modo teleoperado. Los mapas realizados mediante movimiento planificado, empleando el control diseñado, se incluyen en el capítulo 6.

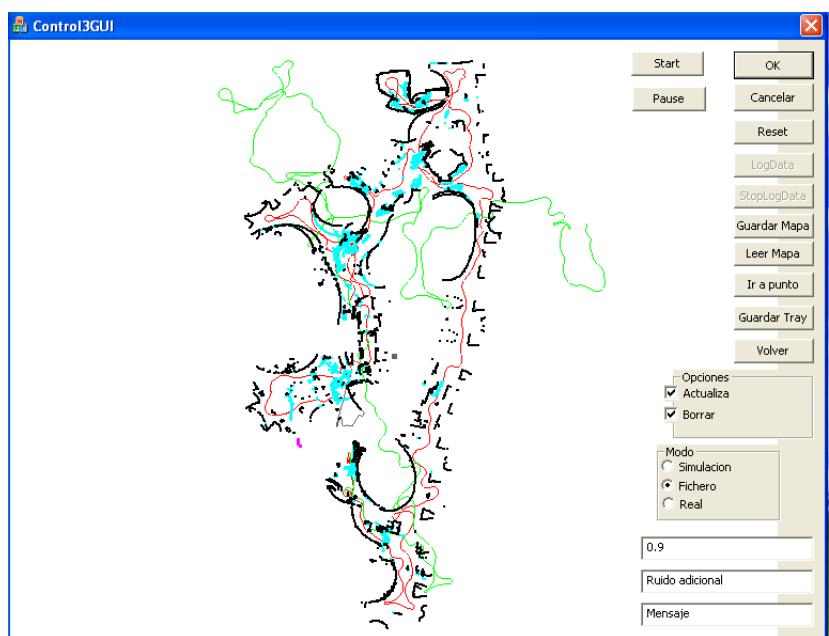


Figura 4.11: Segundo experimento con datos del Museo de las Ciencias Príncipe Felipe de Valencia

Capítulo 5

Control de movimiento, planificación de trayectorias y control reactivo

5.1 Control de movimiento

Las variables sobre las que se actúa para controlar el movimiento del robot son las velocidades de avance y giro. Sus sentidos son los indicados en la figura 5.1:

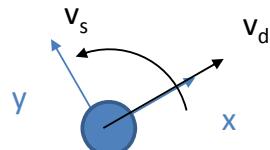


Figura 5.1: Velocidad de avance y giro en el sistema de referencia local del robot

La medida de ángulos se realiza en todo momento en el intervalo [-PI,PI]. Para efectuar la conversión a este rango se utiliza la función `AngRango`.

Como se expuso en el capítulo de Estado del Arte, resulta conveniente utilizar un regulador con realimentación que emplee información sobre la posición del robot en cada instante para obtener unos valores de velocidad que permitan llegar al objetivo dado.

Para que el robot se traslade mirando de frente la mayor parte del tiempo, se busca minimizar la diferencia entre su orientación y la del vector que lo

une con el punto de destino. Al mismo tiempo el robot deberá irse acercando a dicho punto de destino.

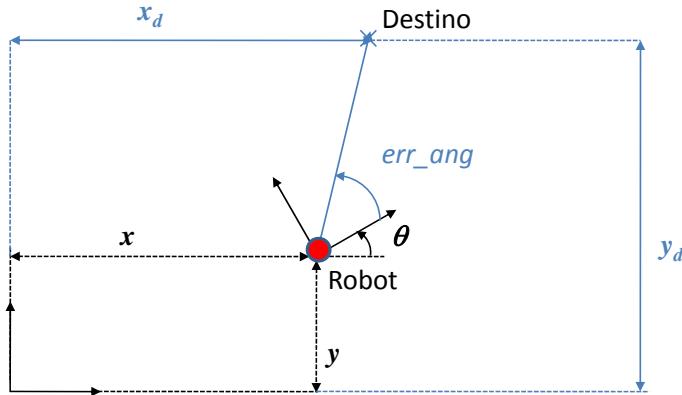


Figura 5.2: Control de la orientación del robot

Como puede verse en la figura 5.2, el ángulo resultante de esa diferencia de orientaciones se llamará *err_ang*. En función de su valor se distinguen cuatro casos principales en los cuales se utilizaron en una primera aproximación las ganancias indicadas a continuación:

donde todas las magnitudes están expresadas en el Sistema Internacional salvo cuando se especifica lo contrario.

Al aumentar la duración del ciclo de tareas cuando se utiliza el robot real Urbano, los comandos de velocidad enviados de acuerdo con la tabla 5.1 llegaban con retraso y el robot daba algunos bandazos. Para solucionar este problema se modificó el regulador de forma que se contemplara también la orientación de la trayectoria en el tramo en que se encuentra el robot en cada instante y la orientación en el tramos siguiente, a modo de control predictivo. Los nuevos ángulos sobre los que también se actúa se denominarán *err_ang2* y *err_ang3*, respectivamente, y se muestran en la figura ??.

La acción de control resultante es una combinación lineal de las acciones sobre estos tres ángulos. Se ha utilizado un peso de 0.3 para regular *err_ang*, un peso de 0.2 para regular *err_ang2* y un peso de 0.5 para regular *err_ang3*. El regulador final obtenido utiliza los valores y ganancias que aparecen en la tabla 5.1. Con el robot Pioneer los pesos son los mismos y se mantienen las ganancias del cuadro 5.1.

donde de nuevo las magnitudes se expresan en el S.I. a no ser que se especifique lo contrario. *p1*, *p2* y *p3* son los pesos indicados (0.3, 0.2 y 0.5).

El sistema funciona como una máquina de estados. Cuando el robot está lo

- $170^\circ < \text{err_ang}(\text{°})$

Urbano:

$$v_d = 0$$

$$v_s = 30\text{err_ang}$$

Pioneer:

$$v_d = 0$$

$$v_s = 0,65\text{err_ang}$$

- $45^\circ < \text{err_ang}(\text{°}) < 170^\circ$

Urbano:

$$v_d = 2$$

$$v_s = 40\text{err_ang}$$

Pioneer:

$$v_d = 0,025$$

$$v_s = 0,7\text{err_ang}$$

- $15^\circ < \text{err_ang}(\text{°}) < 45^\circ$

Urbano:

$$v_d = 6$$

$$v_s = 34\text{err_ang}$$

Pioneer:

$$v_d = 0,1$$

$$v_s = 0,7\text{err_ang}$$

- $\text{err_ang}(\text{°}) < 15^\circ$

Urbano:

$$v_d = 10$$

$$v_s = 26\text{err_ang}$$

Pioneer:

$$v_d = 0,4$$

$$v_s = 0,6\text{err_ang}$$

Cuadro 5.1: Regulador inicial para el control de movimiento

- $160^\circ < err_ang(\circ) < 180^\circ$

Urbano:

$$v_d = 0$$

$$v_s = 30err_ang$$

- $45^\circ < err_ang(\circ) < 160^\circ$

Urbano:

$$v_d = 3$$

$$v_s = p1 \times 30err_ang + p2 \times 30err_ang2 + p3 \times 30err_ang3$$

- $15^\circ < err_ang(\circ) < 45^\circ$

Urbano:

$$v_d = 5$$

$$v_s = p1 \times 26err_ang + p2 \times 26err_ang2 + p3 \times 26err_ang3$$

- $err_ang(\circ) < 15^\circ$

Urbano:

$$v_d = 10$$

$$v_s = p1 \times 20err_ang + p2 \times 26err_ang2 + p3 \times 26err_ang3$$

Cuadro 5.2: Regulador para el control de movimiento

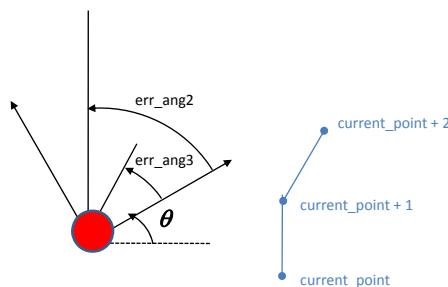


Figura 5.3: Nuevos errores incorporados al regulador

suficientemente cerca del punto de destino se considera que ha llegado y se establece como destino el siguiente punto de la trayectoria definida. Cuando la distancia al último punto de la misma es menor que 0.5m, actúa un regulador proporcional para que la velocidad de avance del robot vaya disminuyendo gradualmente a medida que se aproxima el final de su recorrido.

Si por algún motivo el robot se halla separado de la trayectoria calculada entre dos puntos, este controlador lo llevaría directamente hacia el punto de destino en línea recta. Esto podría ocasionar el choque con algún obstáculo. Por ello, se utiliza también un regulador que hace que el robot se acerque a la trayectoria definida antes de dirigirse hacia el siguiente punto a alcanzar. Esta situación se muestra en la figura 5.4:

El algoritmo empleado en este caso requiere medir en primer lugar la distancia de la posición del robot a la recta que une los dos puntos de la trayectoria entre los que se encuentra. Esta distancia tendrá signo positivo o negativo dependiendo de a qué lado de la trayectoria se encuentre el robot (figura 5.5).

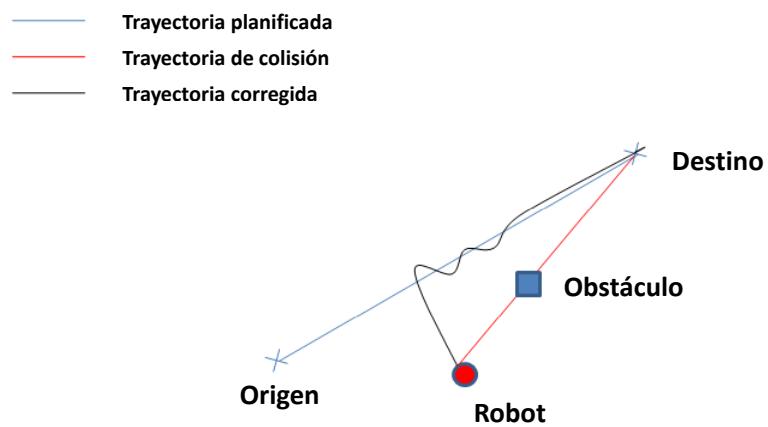
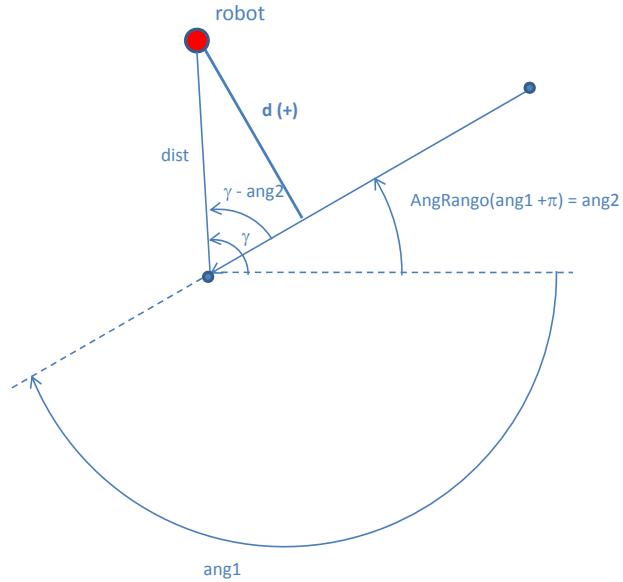


Figura 5.4: Diferentes trayectorias entre dos puntos

$$d = \text{dist} \times \sin(\gamma - \text{ang2}) > 0$$



$$d = \text{dist} \times \sin(\gamma - \text{ang2}) < 0$$

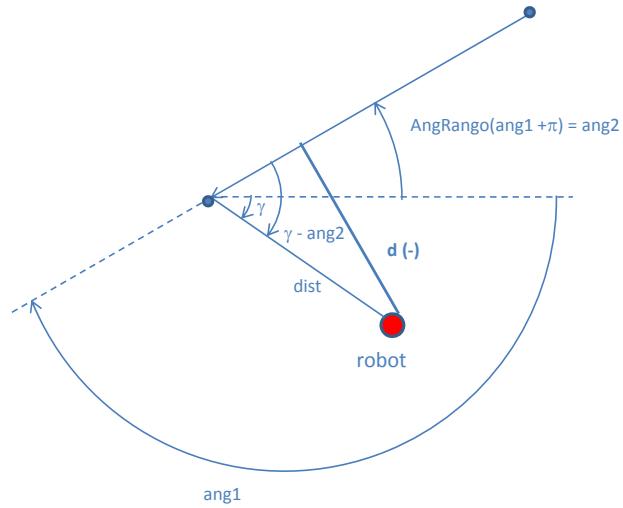
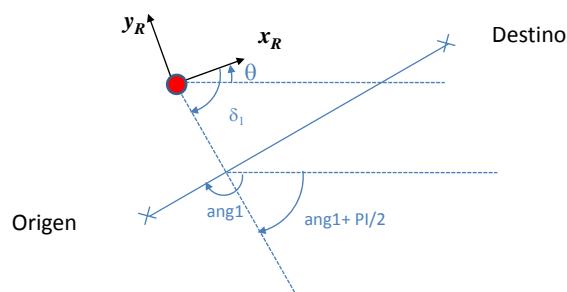
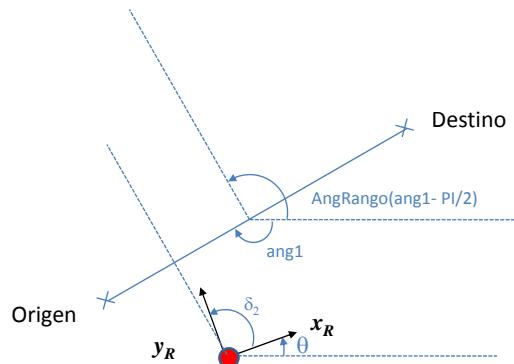


Figura 5.5: Medida de la distancia del robot al segmento de trayectoria en el que se halla

Si el robot está más lejos de la trayectoria que lo establecido por un cierto valor límite, se realizará un control que tienda a dirigirlo perpendicularmente hacia la misma. Para ello se utilizan unos valores de ganancia muy similares a los indicados en la tabla anterior, pero el ángulo que se mide es aquél que separa la orientación del robot con la que debería tener para acercarse perpendicularmente al segmento de trayectoria correspondiente. El modo en que se calcula este ángulo en dos casos en que el robot está situado a un lado u otro de un mismo segmento se muestra en la figura 5.6.



$$\delta_1 = \text{ang1} + \pi/2 - \theta$$



$$\delta_2 = \text{AngRango}(\text{ang1} - \pi/2) - \theta$$

Figura 5.6: Ángulo a regular para que el robot se aproxime a la trayectoria definida en dos casos diferentes

Otro aspecto que se tiene en cuenta en el diseño del regulador es el hecho de que el robot pueda pasarse de algún punto de la trayectoria al hallarse muy próximos unos puntos a otros y ser la velocidad alta. En ese caso resultaría absurdo que el robot regresara a dicho punto para seguir la secuencia exacta. Es más conveniente que se dirija hacia aquél que, estando relativamente cerca del destino teórico, sea más cercano al robot. Lo mismo ocurre si la tolerancia que determina si se ha llegado o no a un punto es excesivamente pequeña. Para ello, cada vez que se van a calcular unas nuevas velocidades se busca si en la trayectoria hay algún punto entre los cinco siguientes al punto de destino que se encuentre a menos distancia de la posición del robot que éste.

5.2 Planificación de trayectorias

Como se ha visto en la sección anterior, el control de movimiento del robot precisa disponer de un conjunto de puntos de paso que se vayan definiendo como destinos sucesivos, conformando la trayectoria que ha de seguirse. El alcance de este proyecto no incluye la generación automática de dichos puntos de paso a partir de un destino final dentro de un mapa. Lo que puede hacerse es seleccionar la serie de puntos que determina la trayectoria mediante uso del ratón sobre la interfaz gráfica en la que se ve un mapa o parte de uno. Otra posibilidad para obtener los puntos de la trayectoria consiste en llevar el robot hacia un sitio en modo teleoperado e ir guardando su posición cada vez que recorre una cierta distancia de forma que puede regresar al punto del que partió de manera autónoma.

Cuando los puntos que dan lugar a la trayectoria están bastante separados conviene suavizar los cambios de dirección en la misma. El algoritmo utilizado para ello se explica a continuación.

Mientras sea posible, para cada punto de la trayectoria se toma el siguiente a él como base o punto intermedio a suprimir en caso necesario. Se definen dos vectores que van desde la base hasta el punto anterior y hasta el punto siguiente a ella, respectivamente, y se mide el ángulo que los separa. Si este ángulo, que llamaremos `erro_ang`, es menor que 20° o superior a 160° no ha de redondearse la trayectoria; simplemente se pasa al siguiente de sus puntos. En caso contrario se halla su bisectriz para situar sobre ella el centro del arco de circunferencia que servirá para suavizar la trayectoria en el punto base. A partir del centro se irán determinando puntos de forma que su distancia a él sea igual al radio y que queden uniformemente repartidos sobre un arco tangente a los dos segmentos de trayectoria que se unen. El cálculo de los ángulos que permiten calcular las coordenadas de los puntos del arco se basa en un ángulo auxiliar definido como `alfa_ref = AngRango(PI - ang_bis)` y

es diferente según *erro_ang* sea positivo o negativo, dado que esto condiciona el sentido en que deben ir creciendo dichos ángulos.

En la figura 5.7 se muestra un caso en el que *erro_ang* es mayor que cero. Para la obtención de los sucesivos puntos el valor de *ang_inc* se va incrementando en 0.1 rad mientras que al sumarse a α_1 no se sobreponga el valor de α_2 . Estos dos ángulos vienen dados por:

$$\alpha_1 = \text{AngRango}(\alpha_{ref} - (\frac{\pi}{2} - \frac{\text{erro_ang}}{2}))$$

$$\alpha_2 = \text{AngRango}(\alpha_{ref} + (\frac{\pi}{2} - \frac{\text{erro_ang}}{2}))$$

Así, para la trayectoria de 5.7 los valores son:

$$\text{erro_ang} = 100^\circ$$

$$\alpha_{ref} = \text{AngRango}(\pi - (-130^\circ)) = \text{AngRango}(310^\circ) = -50^\circ$$

$$\alpha_1 = \text{AngRango}(-50^\circ - 90^\circ + 50^\circ) = \text{AngRango}(-90^\circ) = -90^\circ$$

$$\alpha_2 = \text{AngRango}(-50^\circ + 90^\circ - 50^\circ) = \text{AngRango}(-10^\circ) = -10^\circ$$

Si *erro_ang* es menor que cero, los ángulos extremos mediante los cuales se calculan las coordenadas de los puntos del arco son:

$$\alpha_1 = \text{AngRango}(\alpha_{ref} + (\frac{\pi}{2} + \frac{\text{erro_ang}}{2}))$$

$$\alpha_2 = \text{AngRango}(\alpha_{ref} - (\frac{\pi}{2} + \frac{\text{erro_ang}}{2}))$$

Ahora el ángulo α_{dec} , figura 5.8, se va decrementando 0.1 rad hasta que su suma con α_1 se hace menor que α_2 .

En el caso concreto representado en 5.8 los ángulos que se muestran toman los siguientes valores:

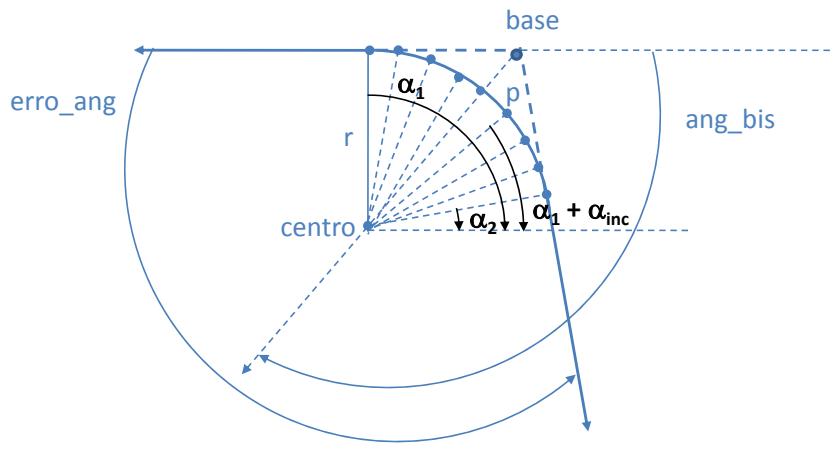
$$\text{erro_ang} = -100^\circ$$

$$\alpha_{ref} = \text{AngRango}(\pi - (130^\circ)) = \text{AngRango}(50^\circ) = 50^\circ$$

$$\alpha_1 = \text{AngRango}(50^\circ + 90^\circ - 50^\circ) = \text{AngRango}(-90^\circ) = 90^\circ$$

$$\alpha_2 = \text{AngRango}(50^\circ - 90^\circ + 50^\circ) = \text{AngRango}(-10^\circ) = 10^\circ$$

A continuación se muestra el resultado de aplicar el sistema implementado a diferentes trayectorias. Como puede verse, el color negro representa la trayectoria original y el azul, la trayectoria suavizada. El valor del radio utilizado es 1m.



$$\text{centro} = (x_c, y_c)$$

$$x_p = x_c + r \cos(\alpha_1 + \alpha_{inc})$$

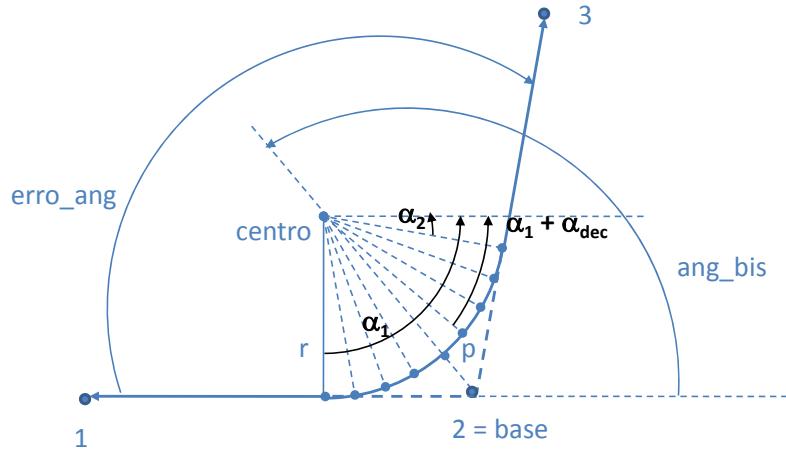
$$y_p = y_c - r \sin(\alpha_1 + \alpha_{inc})$$

Figura 5.7: Cálculo de coordenadas de los puntos del arco cuando $\text{erro_ang} > 0$

5.3 Control Reactivo

Una trayectoria planificada y modificada de acuerdo con los apartados anteriores puede requerir nuevos cambios si se detectan obstáculos cercanos a ella. Este aspecto se contempla en el proyecto a través de un algoritmo que desvía los puntos de la trayectoria cercanos a los objetos. Para cada punto i de la trayectoria nominal se siguen los mismos pasos. En primer lugar, se define un punto p sobre la perpendicular al segmento que une ese punto de la trayectoria con el siguiente de forma que su distancia a dicho punto de la trayectoria sea 1m (ver figura 5.10). El vector que une el punto de la trayectoria con el punto p se denominará v_1 .

Se define un vector v_2 con origen en el punto de la trayectoria considerado y extremo los sucesivos puntos tomados como obstáculos. La proyección ortogonal de este vector sobre v_1 define un punto que llamaremos p_2 . La distancia entre éste y el punto de la trayectoria se denota como d , que puede



$$\text{centro} = (x_c, y_c)$$

$$x_p = x_c + r \cos(\alpha_1 + \alpha_{\text{dec}})$$

$$y_p = y_c - r \sin(\alpha_1 + \alpha_{\text{dec}})$$

Figura 5.8: Cálculo de coordenadas de los puntos del arco cuando $\text{erro_ang} < 0$

adoptar un signo u otro dependiendo de a qué lado de la trayectoria se encuentre el objeto. La distancia entre p_2 y el obstáculo correspondiente se mide mediante d_1 .

En la deformación de la trayectoria sólo se utilizan los obstáculos que proporcionan un valor de d_1 suficientemente pequeño, puesto que la influencia de los objetos cercanos a tramos más avanzados de aquella podría llevar a malos resultados.

A través de un valor de d corregido al tenerse en cuenta el radio del robot, se guardan para cada punto de la trayectoria las dos distancias mínimas a algún obstáculo a cada uno de los dos lados de la misma. Estas distancias se llamarán $\text{min_int}[i]$ y $\text{max_int}[i]$ respectivamente. Sus valores están inicializados con la máxima desviación que se le permite al robot, para el caso en que no haya ningún obstáculo en alguno de los lados y otras situaciones similares. Si después de recalcular dichos valores se cumple que $\text{min_int} > \text{max_int}$, el robot no puede evitar los obstáculos sin superar la máxima desviación es-

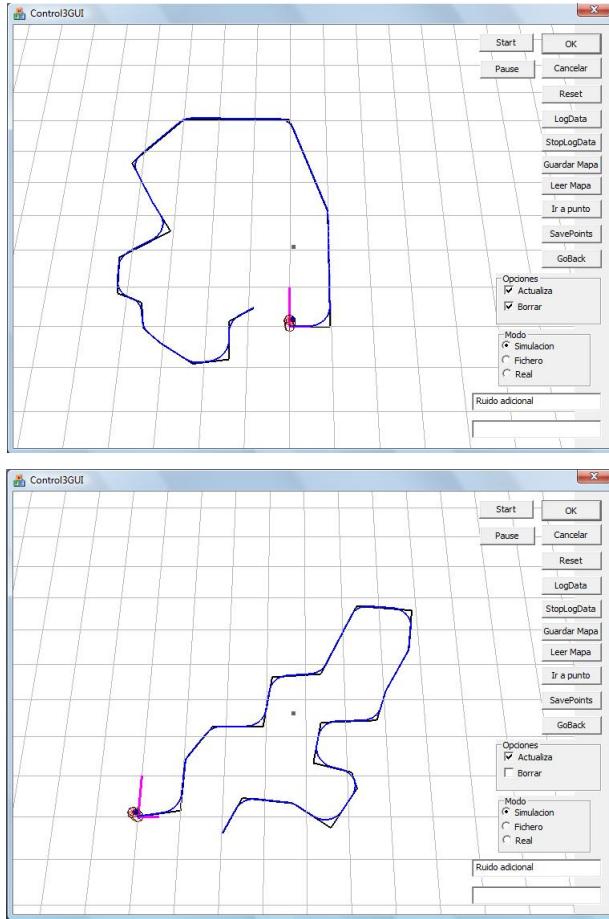


Figura 5.9: Trayectorias suaves conseguidas

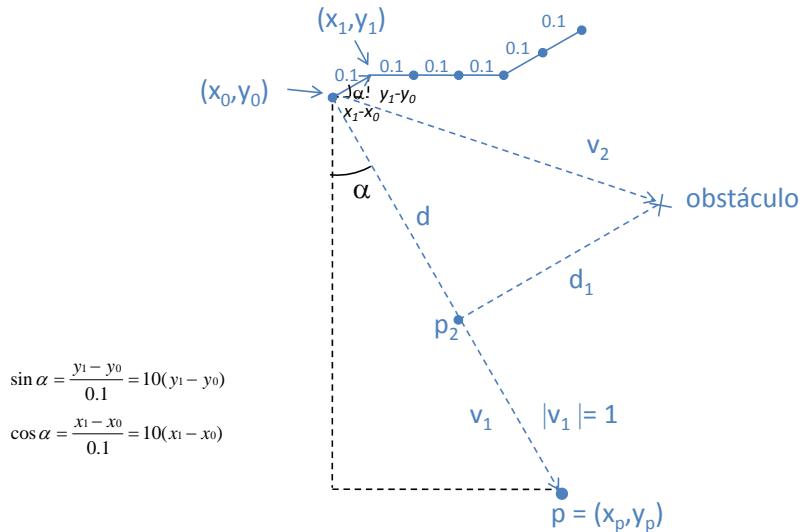
tablecida y se dice que se encuentra en estado de bloqueo.

Haciendo la media entre esos dos valores se obtiene el desplazamiento, error [i], que debe aplicarse sobre el punto para que quede a la misma distancia de cada uno de los límites hallados para ambos lados (figura 5.12).

Con los puntos intermedios de la trayectoria lo que se hace es promediar el valor del error en el punto considerado con los valores del mismo en los puntos anterior y siguiente:

$$\text{error2}[i] = \frac{\text{error}[i-1] + \text{error}[i] + \text{error}[i+1]}{3}$$

De este modo se busca deformar la trayectoria más suavemente.



$$\begin{aligned}
x_p &= x_0 + \sin \alpha = x_0 + 10(y_1 - y_0) \\
y_p &= y_0 - \cos \alpha = y_0 - 10(x_1 - x_0)
\end{aligned}$$

Figura 5.10: Definición del punto p y otras magnitudes para el punto de la trayectoria $i = 0$

5.4 La clase CMoveControl

En esta clase se llevan a cabo las tareas de control de movimiento, planificación de trayectorias y control reactivo que se han descrito. Las principales funciones que se utilizan para ello son las siguientes:

5.4.1 DefineDest

```
void DefineDest(float x_d, float y_d, float tol)
```

Esta función sirve para definir el punto hacia el que tiene que dirigirse el robot.

- x_d : coordenada x del punto de destino.

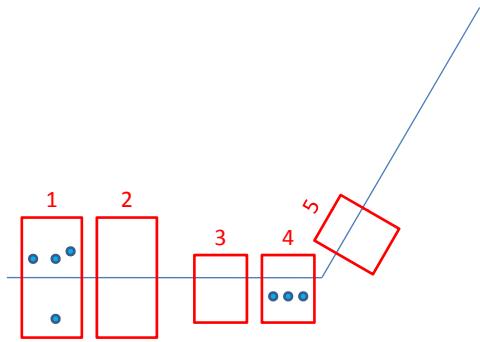


Figura 5.11: Los obstáculos del tramo 1 no deben afectar al 2 ni los del tramo 4 a los tramos 3 y 5.

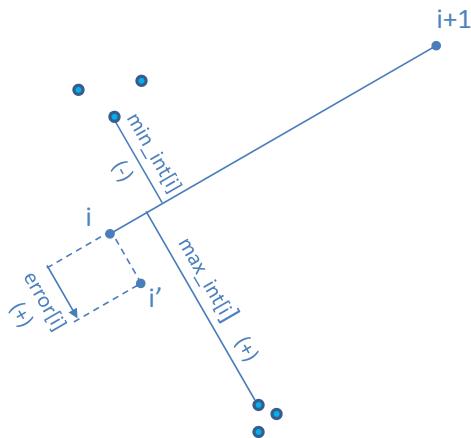


Figura 5.12: Desplazamiento que sufre cada punto de la trayectoria

- y_d : coordenada y del punto de destino.
- tol : distancia máxima al punto de destino para que se considere que éste ha sido alcanzado. Generalmente se utiliza una tolerancia de 50mm.

Se guardan los argumentos en variables miembro, para que sus valores sean utilizados por el control de movimiento, y se establece el estado como NO_LLEGADO mediante una macro con este nombre.

5.4.2 Dist2Tray

```
float Dist2Tray()
```

Esta función se utiliza para hallar la distancia del robot al segmento que une los puntos de la trayectoria entre los que se encuentra e indicar a qué lado

del mismo está, de modo que pueda realizarse el control de acercamiento a la trayectoria descrito en 5.1.

El cálculo de la distancia se lleva a cabo mediante los ángulos mostrados en la figura 5.5, con la fórmula que aparece en ella. El valor que se devuelve es el de d .

5.4.3 FindPoint

```
int FindPoint()
```

Esta función se utiliza para ver cuál es el próximo punto de la trayectoria hacia el que debe ir el robot. Es útil en el caso de que haya pasado demasiado rápidamente por algún punto, para evitar que tenga que retroceder (ver última parte de 5.1).

Se busca entre los cinco puntos de la trayectoria siguientes al de índice `current_point` dentro del vector `trajectory` aquel que esté más cerca del robot. Se devuelve el valor de su índice.

5.4.4 GetCommand

```
int GetCommand(float* vd, float* vs)
```

La función de este método consiste en calcular las velocidades de avance y giro con los que se va a mover el robot en cada momento. El paso de parámetros se realiza por referencia.

- `vd`: puntero a la velocidad de avance.
- `vs`: puntero a la velocidad de giro.

En primer lugar se mira si es necesario corregir el punto de destino con una llamada a la función `FindPoint()` y si es así se hacen los correspondientes cambios. A continuación se determina el estado, viendo si la distancia entre la posición del robot y el punto de destino es menor que la tolerancia.

En la figura se muestra el diagrama de flujo simplificado de los pasos que se realizan a partir de este punto.

Si el estado es `NO_LLEGADO`, se determinan las velocidades mediante el procedimiento indicado en 5.1 a partir de la posición del robot (disponible en toda la clase mediante las variables miembro `x` e `y`) y del punto guardado como destino. Si el estado es `LLEGADO` se da valor nulo a ambas velocidades y se incrementa una variable llamada `current_point` que mide de este modo el índice del último punto de la trayectoria al que se ha llegado. Si efectivamente hay una trayectoria definida, se llama a la función `DefineDest` para guardar

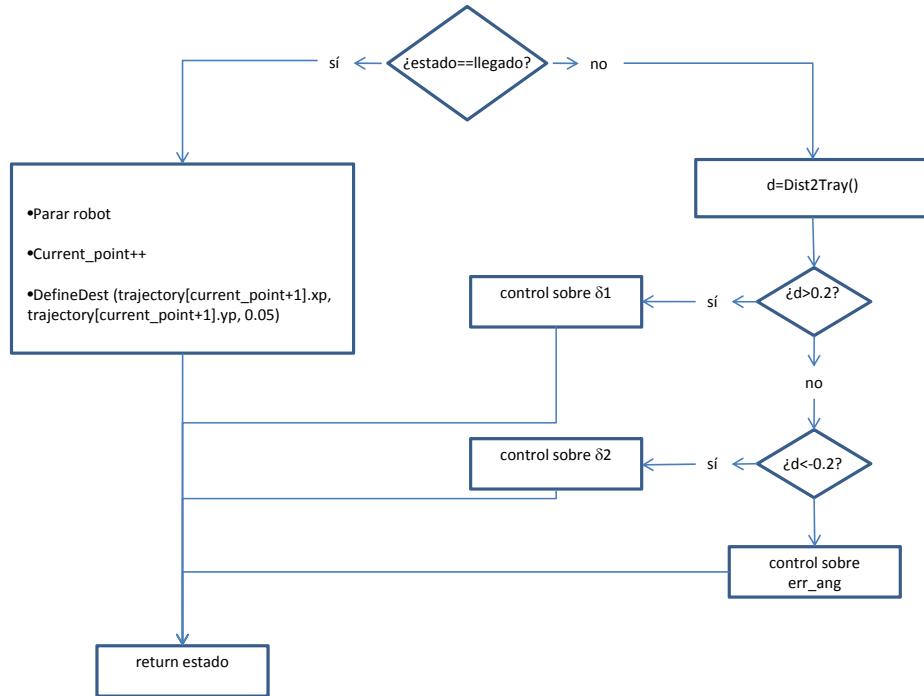


Figura 5.13: Diagrama de flujo simplificado de parte del proceso efectuado en `GetCommand`

como destino el siguiente punto de la misma (`current_point + 1`). Los puntos de la trayectoria han de estar almacenados en un vector de la STL, vector `trajectory`, miembro de la clase y definido para contener objetos `CPoint2D`. El valor de retorno es el estado.

5.4.5 DefineTrajectory

```
void DefineTrajectory(void)
```

Esta función se emplea para indicar que hay una trayectoria definida e inicializar el punto por el que debe empezarse a recorrerla (`current_point = 0`).

5.4.6 SmoothTrajectory

```
std::vector<Point2D> SmoothTrajectory(float r)
```

Esta función se utiliza para suavizar la trayectoria.

- **r**: radio del arco con el que se suavizan los ángulos de la trayectoria.

Los puntos de la trayectoria a suavizar se hallan guardados en un vector dinámico de nombre **tray_prev** e inicialmente se copian en otro vector dinámico variable local de la función (vector **smooth_tray**) para realizar en éste las modificaciones oportunas y no perder los puntos de la trayectoria original (de cara a la representación gráfica, principalmente). Esta función es la implementación del algoritmo descrito en 5.2. Se devuelve el vector **smooth_tray**, que contiene los puntos de la trayectoria suave obtenida.

5.4.7 ComputeElasticTray

```
std::vector<Point2D> ComputeElasticTray()
```

Esta función sirve para deformar la trayectoria de forma que se aleje de los obstáculos (puntos 2D guardados en el vector de la STL **v_puntos**, variable miembro de la clase).

Los puntos de la trayectoria que se deforma son los almacenados en el vector **nom_tray** de la STL, que se copian en una variable local de nombre **ret_tray**. Con estos puntos se siguen los pasos indicados en el algoritmo de 5.3. En caso de que al calcularse alguno de los nuevos puntos se produzca bloqueo (**min_int[i] > max_int[i]**), se devuelve la trayectoria **ret_tray** con los puntos hallados hasta ese momento. Si no se da situación de bloqueo, el vector **ret_tray** se devuelve al final de la función y tendrá tantos puntos como la trayectoria inicial. El resultado de esta llamada será asignado al vector **trajectory** para que pueda aplicarse el control de movimiento.

5.4.8 DivideTrajectory

```
int DivideTrajectory()
```

Esta función permite obtener una trayectoria igual a la inicial (**nom_tray**) pero con el requisito de que cada uno de sus puntos no diste más de 0.1m del siguiente.

Se utiliza un vector de la STL, variable local de la función, en el que se guarda cada punto de la trayectoria seguido de tantos otros puntos como sea necesario para que el segmento que va de aquél al próximo punto de **nom_tray** cumpla la condición dada. El número de puntos intermedios que se añaden será $n = \frac{dist}{0,1} - 1$, siendo *dist* la longitud del segmento inicial. Finalmente se iguala el vector **nom_tray** al vector con los nuevos puntos.

La utilidad de esta función reside en la necesidad de tener los puntos de la trayectoria suficientemente juntos para que el control reactivo que se imple-

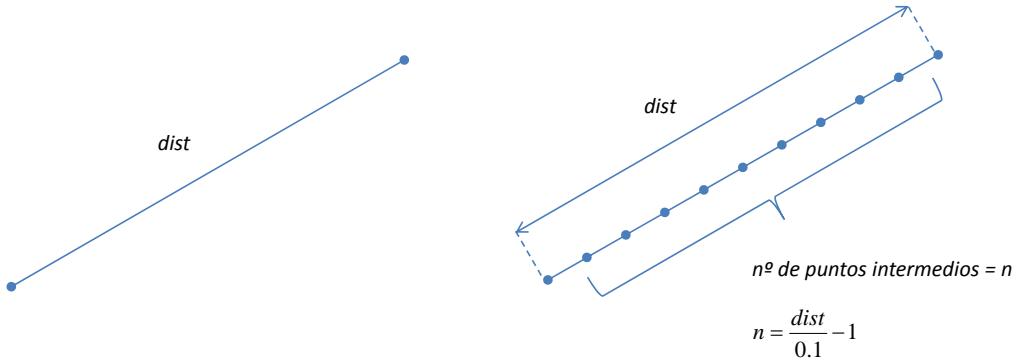


Figura 5.14: Puntos de un segmento de la trayectoria original y del resultado de la llamada a `DivideTrajectory()`

menta a través de `ComputeElasticTray()` sea efectivo. Si no fuera así, el filtro que se realiza mediante d_1 no permitiría esquivar muchos de los obstáculos.

5.5 Pruebas y resultados

Todas las funcionalidades de control de movimiento, planificación de trayectorias y control reactivo del proyecto han sido probadas inicialmente mediante simulación gráfica. Esto permite la depuración y mejora del comportamiento del programa de una forma más cómoda. Las pruebas con robots reales requieren más espacio y pueden conllevar mayores riesgos si no se han llevado a cabo previamente en simulación. Además, es preciso cargar baterías, llevar el ejecutable al ordenador portátil que se utiliza, realizar la conexión *telnet* e introducir las contraseñas correspondientes... por lo que resultan bastante más lentas. El control de movimiento, sin embargo, sí que fue probado en modo real en fases anteriores del proyecto ya que la respuesta del robot ante las velocidades aplicadas puede ser algo diferente en uno y otro caso y resultaba conveniente ajustar bien los parámetros para no seguir trabajando con un regulador inadecuado.

5.5.1 Pruebas en modo *simulación*

Control de movimiento y planificación de trayectorias

1. Seguimiento de una trayectoria planificada a priori

En este caso, lo que se ha hecho es definir los puntos de la trayectoria inicial mediante doble click con el botón derecho del ratón sobre la inter-

faz gráfica. Según se van añadiendo puntos se va suavizando la trayectoria. Cuando se ha completado la definición de la trayectoria deseada se indica que el robot debe seguirla y éste comienza a moverse sobre la misma. En la figura puede verse en color azul la trayectoria resultante tras la planificación, en verde la trayectoria seguida de acuerdo con las medidas de la odometría y en rojo la trayectoria seguida según la localización. Como las medidas del láser en modo simulación se hallan a 8000m del robot (ver método **Simulate**, de 3.4.3), no se realiza asociación de datos ni se corrige la posición, de modo que las trayectorias dibujadas en verde y rojo serán necesariamente iguales. La única diferencia reside en el hecho de que se han representado a diferentes alturas.

Resultados:

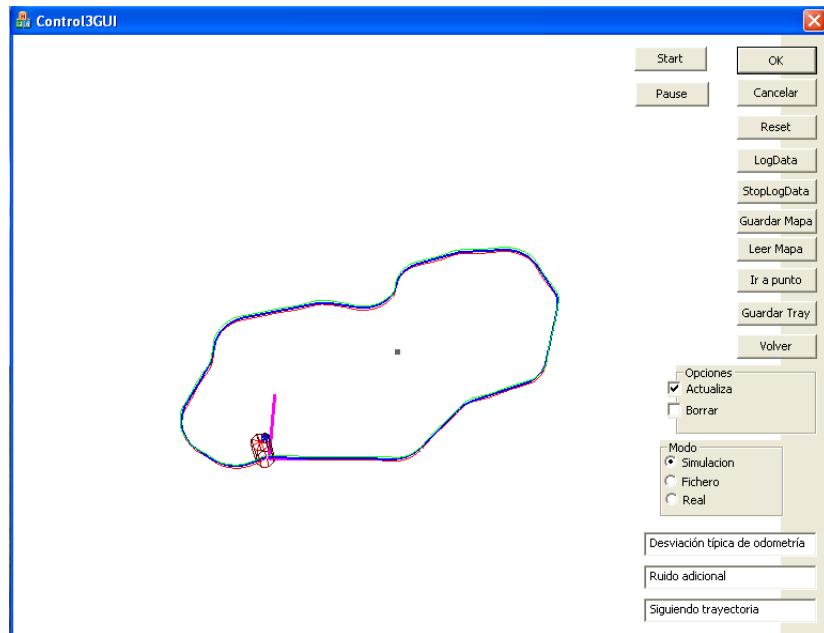


Figura 5.15: Primer experimento de planificación de trayectorias y control de movimiento

Como puede verse, la trayectoria seguida por el robot es prácticamente idéntica a la planificada, lo que muestra el buen diseño del controlador.

2. Seguimiento de una trayectoria planificada a priori con el robot Pioneer P3AT

Esta es una de las pruebas que se realizó con el control elaborado para el robot Pioneer P3AT de MobileRobots/Activmedia. En ella se utiliza MobileSim, un software de simulación proporcionado por los fabricantes para la experimentación con Aria. MobileSim está construido sobre el simulador

Stage (creado por Richard Vaughan, Andrew Howard y otros como parte del proyecto Player/Stage), con algunas modificaciones por parte de MobileRobots. Utilizando la clase **SimpleConnector** de la biblioteca Aria, la conexión se inicia por defecto con el simulador.

Resultados:

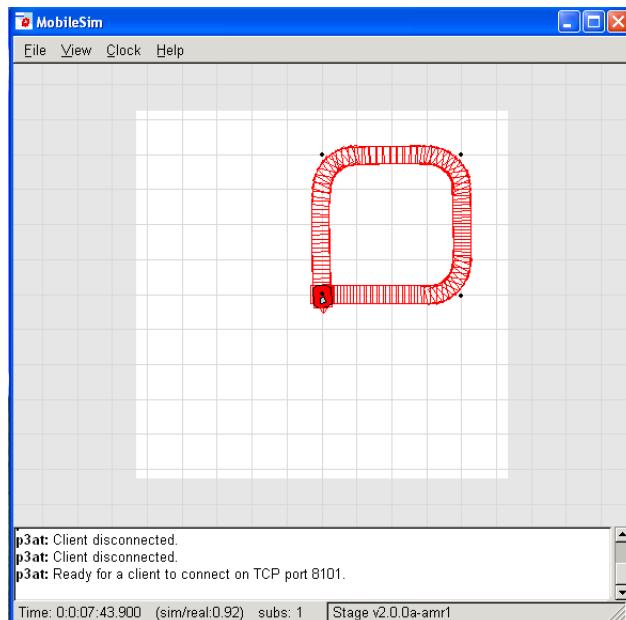


Figura 5.16: Segundo experimento de planificación de trayectorias y control de movimiento

La trayectoria inicial definida en este caso es la formada por los puntos marcados en negro. Puede observarse que el control proporciona de nuevo un buen resultado.

3. Seguimiento de una trayectoria planificada dinámicamente

En esta prueba se generan nuevos puntos de la trayectoria a medida que el robot se acerca a su próximo destino. En algunos casos, al llegar el robot a un punto y posteriormente añadir otro punto después de aquél, el algoritmo que suaviza la trayectoria hace que ésta deje de pasar por el punto en el que se encontraba el robot. Si se produce esta circunstancia, entra en acción el controlador para evitar desvíos sobre la trayectoria planificada. El buen funcionamiento del mismo puede verse con claridad en las figuras 5.17 y 5.18:

Resultados:

En la segunda figura, por ejemplo, el robot llega al punto B procedente del A y, seguidamente, se añade a la trayectoria el punto C. El algoritmo para suavizar trayectorias redondea entonces la trayectoria que habría de

estar formada por A, B y C, pero el robot ya se encuentra en B. El regulador inicial conduciría al robot directamente hacia C pero las mejoras introducidas hacen que el robot se aproxime primero a la trayectoria planificada.

4. Seguimiento del camino de vuelta después de que el robot ejecute una trayectoria.

En este caso, se define una trayectoria mediante el ratón o moviendo al robot por teleoperación con el teclado (sección 6.3.2). Cuando finaliza el seguimiento de la misma, el robot es capaz de regresar de forma autónoma al punto de partida si así se le indica. En la primera figura que se muestra, la trayectoria inicial está definida mediante el ratón mientras que en la segunda se utiliza el modo teleoperado. La trayectoria dibujada en color verde es la correspondiente a la odometría durante la ida y la que aparece en rosa es la odometría del camino de vuelta.

Resultados:

Control reactivo

1. Obtención de trayectorias deformadas ante la presencia de obstáculos cercanos a una trayectoria definida En esta prueba se define primeramente una trayectoria y a continuación se crean obstáculos más o menos cercanos a la misma mediante doble click con el botón izquierdo del ratón sobre el punto en el que debe situarse cada uno de ellos. En el momento en que se añade un obstáculo, la trayectoria se deforma mediante el algoritmo previamente explicado. En este caso no se han incorporado obstáculos nuevos durante el movimiento del robot sobre la trayectoria, por lo que no se trata de una aplicación de control reactivo sino sólo de una prueba del algoritmo de deformación.

Resultados:

Como puede verse, los obstáculos se han representado en color cyan. La trayectoria se deforma de un modo suave en los puntos cercanos a ellos sin superar la máxima desviación permitida, establecida en 0.15m. Los obstáculos suficientemente alejados de la trayectoria no suponen ninguna modificación en la misma. En la segunda figura hay un obstáculo que afecta a dos tramos diferentes de la trayectoria y ambas deformaciones se realizan correctamente. Con este valor de la máxima desviación permitida no pueden esquivarse obstáculos que se encuentren sobre la trayectoria a seguir. En este caso, para el robot Urbano, entrarían en acción los mecanismos de control reactivo de bajo nivel (comandos de parada y giro). Si se utiliza en su lugar un valor de 0.6m, las deformaciones son mayores y pueden evitarse los obstáculos que supondrían un choque directo (figura ??). El problema que conlleva esta opción es que en el caso de entornos densos (como son la mayoría de los entornos de

interiores), las deformaciones son excesivas hacia uno y otro lado y se pierde suavidad en la trayectoria.

2. Trayectoria seguida por el robot ante la aparición de un obstáculo durante su movimiento sobre una trayectoria

En este caso se ha llevado el robot a un punto por medio de una trayectoria definida sobre la interfaz gráfica y se le ha indicado que regrese a su posición inicial. Durante el camino de vuelta, se ha creado un obstáculo en la trayectoria teórica que debería seguir el robot para ver su respuesta simulada en tiempo real.

Resultados: En la figura 6.3 se ha plasmado en color verde la trayectoria seguida por el robot durante la ida (trayectoria teórica de vuelta) y en colores azul y magenta la trayectoria real efectuada ante la presencia del obstáculo.

5.5.2 Pruebas en modo *real*

Cuando se utiliza el sistema con el robot real las llamadas a `GetCommand` se realizan cada 300ms aproximadamente. Como se verá en los siguientes resultados, el seguimiento de la trayectoria es algo menos preciso que en las pruebas realizadas en simulación.

Control de movimiento y planificación de trayectorias

1. Seguimiento de una trayectoria planificada a priori En estas pruebas lo que se ha hecho es definir una trayectoria con el ratón, al igual que en el modo simulación, y ver cómo el robot se va dirigiendo hacia los correspondientes puntos de la misma. La longitud de las trayectorias efectuadas está limitada por el entorno de experimentación, una zona del laboratorio no muy amplia y con objetos cercanos que disminuyen el área explorable.

Resultados: En la figura ?? se muestra en color azul la trayectoria nominal seleccionada y en colores verde y rojo la seguida por el robot.

En el segundo caso se aprecia que el mantenimiento de la dirección de la trayectoria prevalece frente a la acción de acercar el robot a la misma, pero esto evita que la trayectoria resultante sea más ondulada.

2. Seguimiento del camino de vuelta después de que el robot ejecute una trayectoria. La forma de proceder en este caso es la misma que en el modo de simulación. **Resultados:** En la figura 5.23 se muestra en color verde la trayectoria durante la ida y en color rosa la correspondiente al camino de vuelta.

Como puede observarse, en este caso aparecen mayores desviaciones respecto a la trayectoria definida. La desviación producida al dar la vuelta el robot hace que éste tarde un poco en coger la dirección correcta, pero las

acciones de control sobre err_ang2 y err_ang3 permiten que los cambios no sean demasiado bruscos. Si la trayectoria fuera más larga podría verse que la estabilización es bastante rápida.

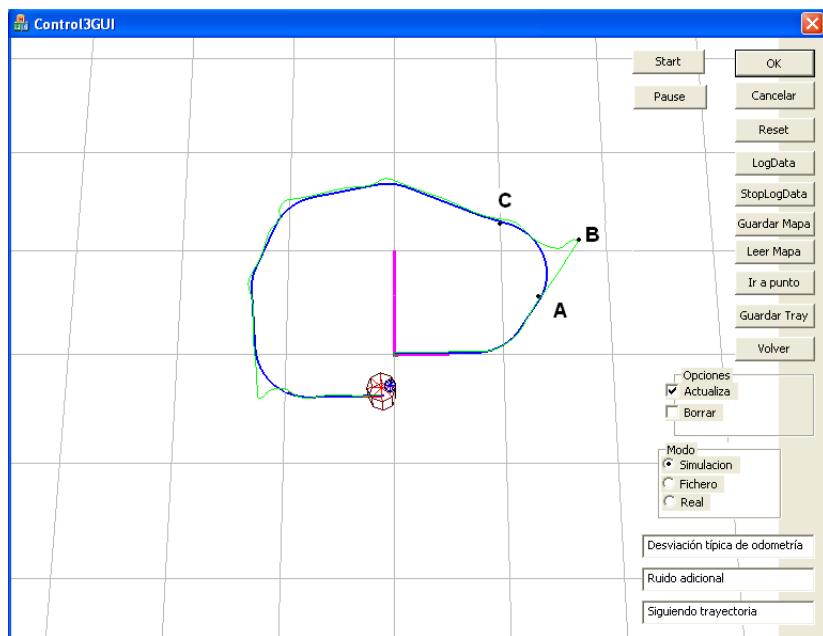
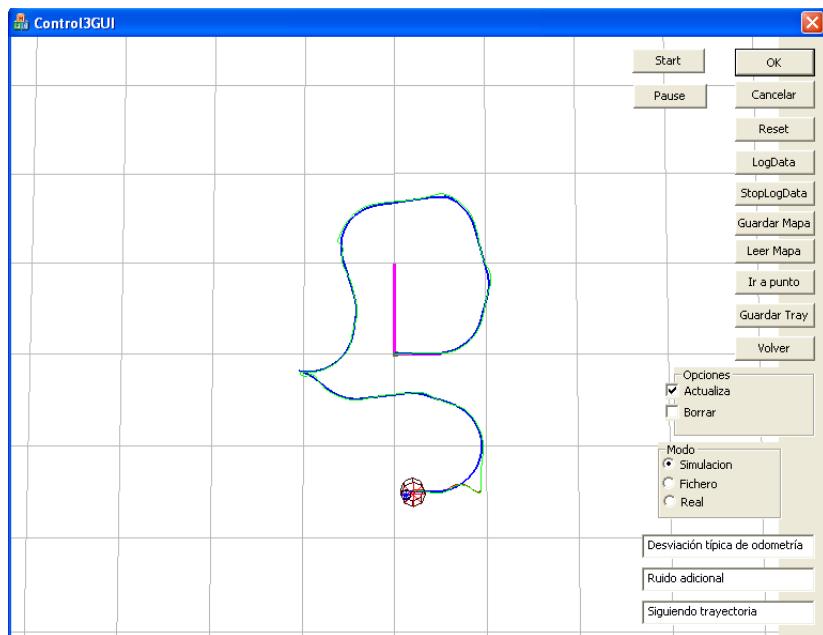


Figura 5.17: Tercer experimento de planificación de trayectorias y control de movimiento

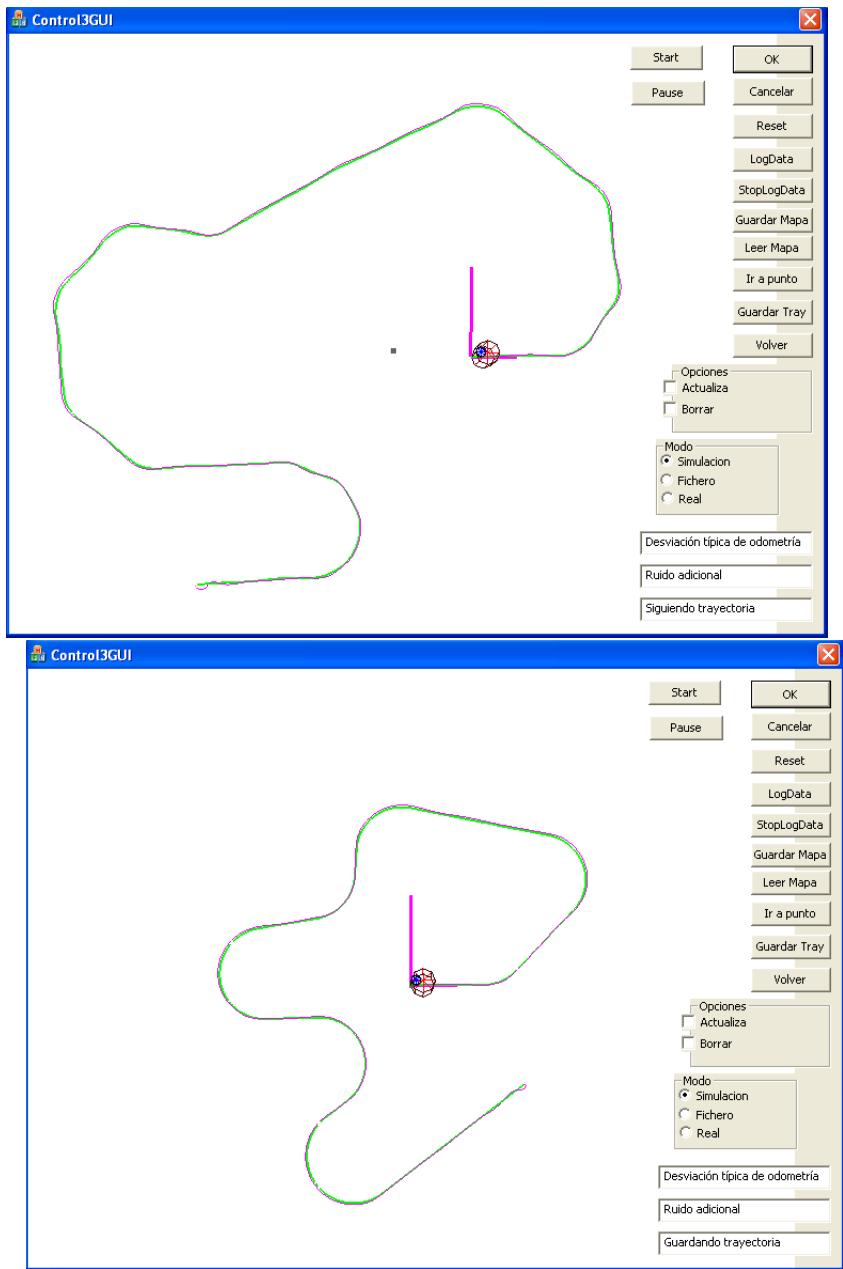


Figura 5.18: Cuarto experimento de planificación de trayectorias y control de movimiento

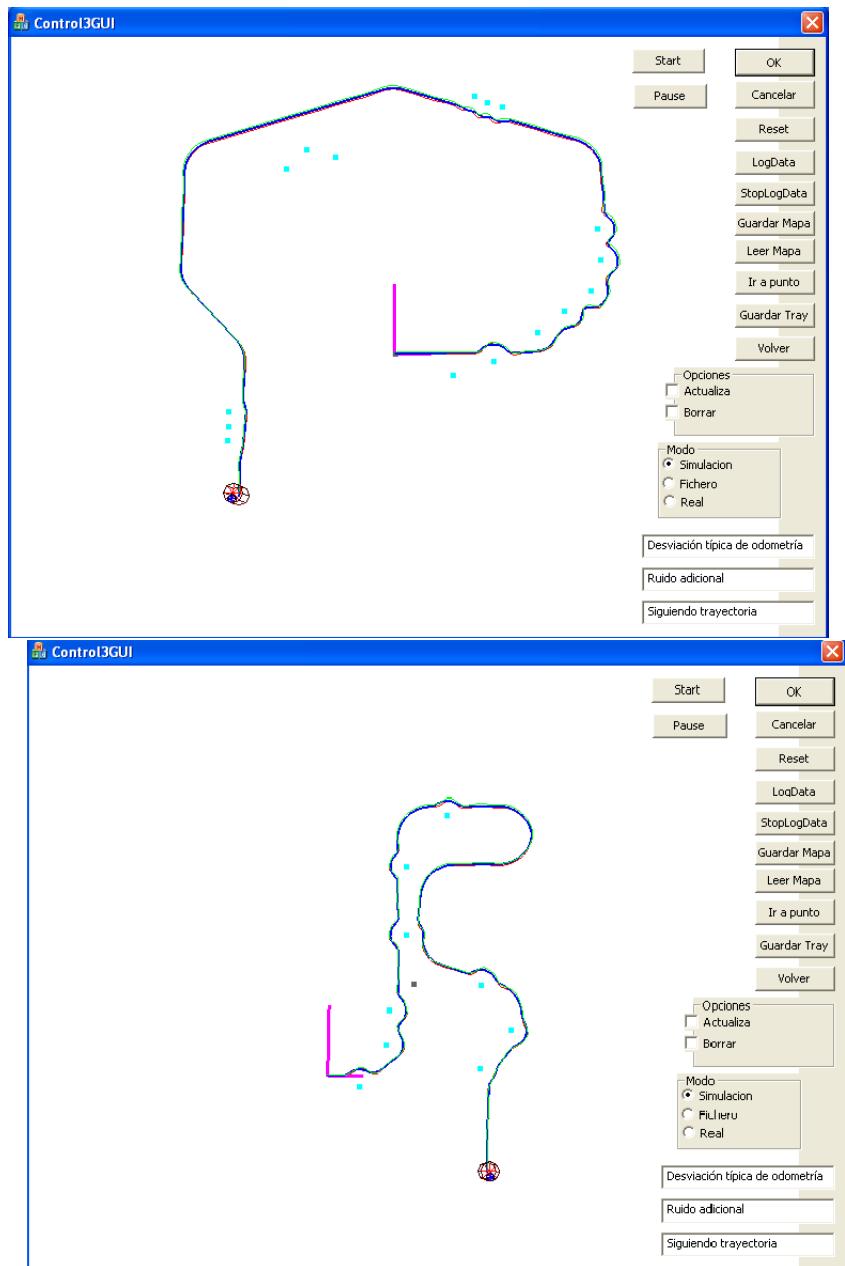


Figura 5.19: Experimento previo para el control reactivo

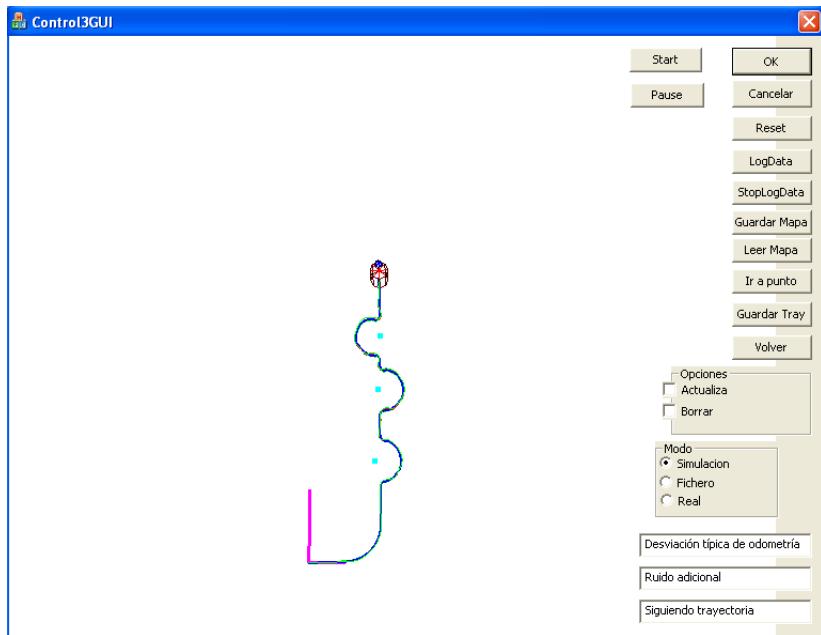


Figura 5.20: Aumento de la máxima desviación permitida

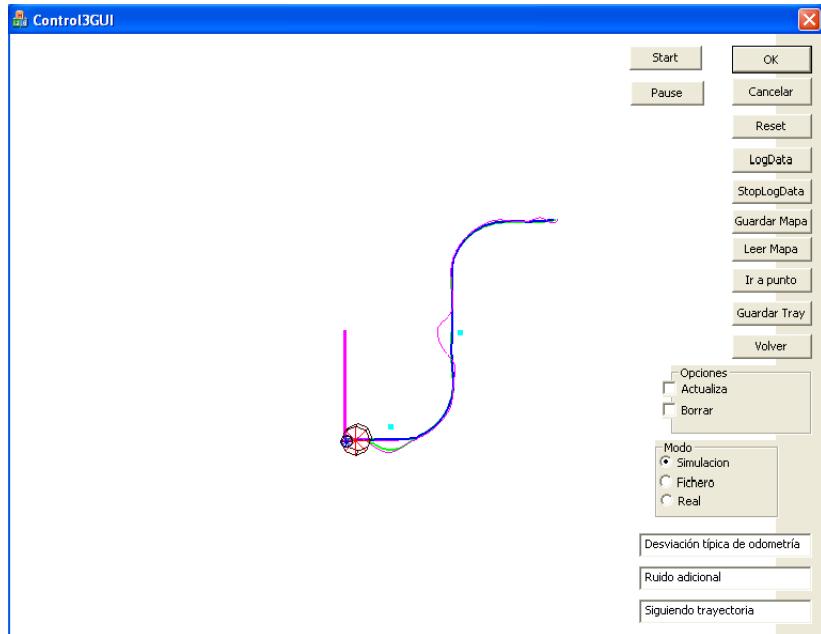


Figura 5.21: Primer experimento de control reactivo

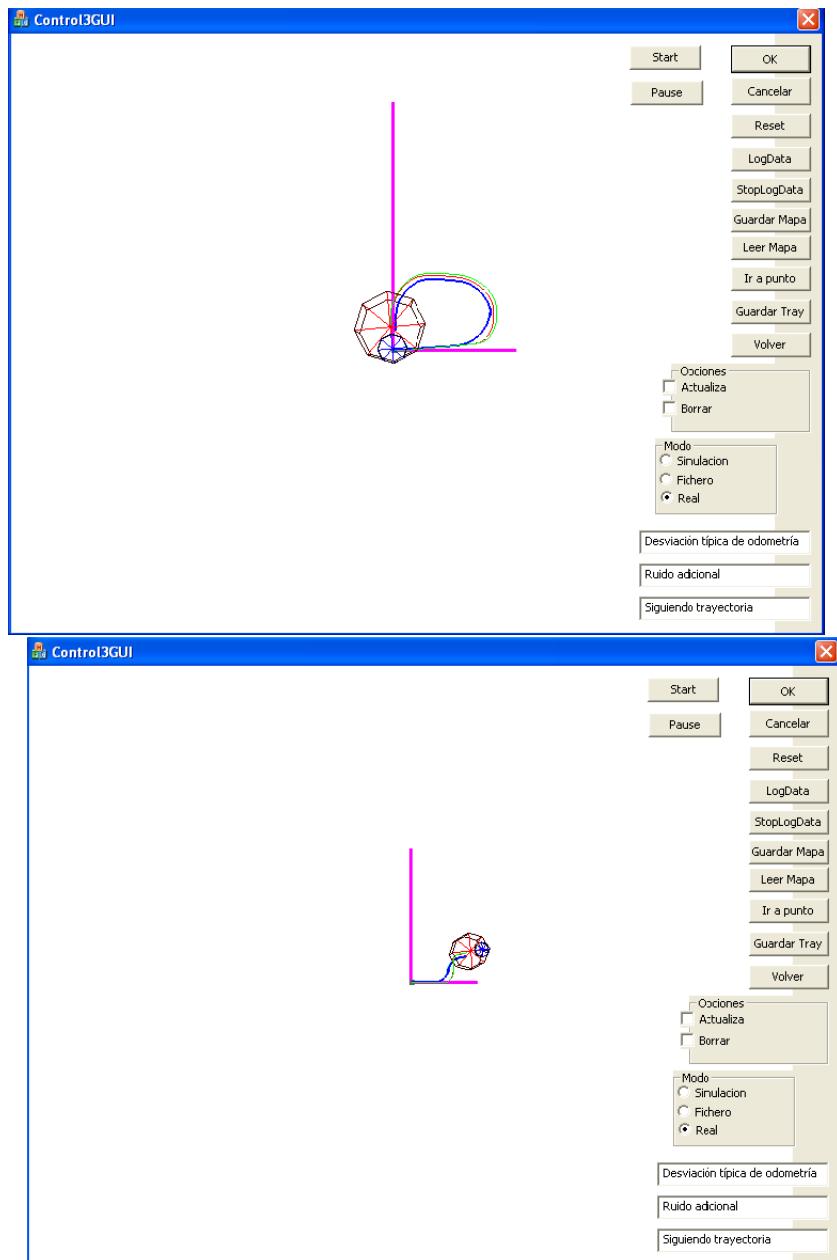


Figura 5.22: Primer experimento de movimiento con robot real

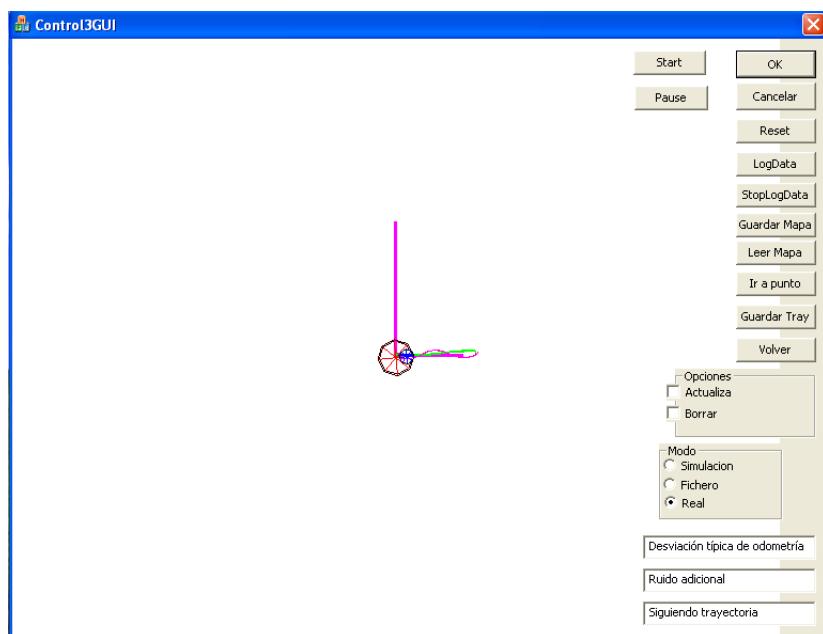


Figura 5.23: Segundo experimento de movimiento con robot real

Capítulo 6

Integración

El funcionamiento del sistema requiere la integración de sus distintos componentes. El robot debe combinar su capacidad para localizarse correctamente en el entorno con el seguimiento de una trayectoria adecuada, resultante de la planificación y el control reactivo. La localización resulta imprescindible para el buen funcionamiento del control y, particularmente, para situaciones en que el robot quede alejado de la trayectoria definida (como en la figura 5.4).

La interacción entre el módulo de localización y el de control de movimiento se lleva a cabo mediante la clase `CRobot`, como puede apreciarse en el diagrama 3.1. Desde las clases `CRobotGLWnd` y `CControl3GUID1g` se realiza la transferencia de información desde la interfaz gráfica al resto del programa. En esta última se tiene el objeto `g1_wnd`, de la clase `CRobotGLWnd`, el objeto `robot`, de la clase `CRobot` y el objeto `control`, de la clase `CMoveControl`. Desde `robot` se tiene un puntero al objeto `control1`. Desde `g1_wnd` se tienen punteros a `robot` y a `control` (para simplificar las llamadas a sus métodos y no tener que utilizarlos a través del puntero al objeto `robot`).

6.1 La clase `CRobot`

El papel de esta clase consiste en integrar:

- la obtención de los datos de odometría y del láser
- la estimación o/y corrección de la posición mediante el filtro de Kalman y la actualización del mapa
- el paso de la posición corregida y del mapa de obstáculos al módulo de control de movimiento

Para ello se dispone de algunas variables miembro importantes:

- `robotdata`, de la clase `CRobotDataReal`
- `proc_laser_data`, de la clase `CProc_Laser_Data`
- `loc`, de la clase `CKalman_Loc`
- `control`, puntero a un objeto de la clase `CMoveControl` (que habrá de apuntar al objeto `control` miembro del diálogo `CControl3GUID1g`)

También posee una variable puntero a un objeto de tipo `CPioneer`, para las pruebas de control que se han hecho con el robot P3AT.

6.1.1 ProcessData

```
int ProcessData()
```

Es la función principal de la clase. Sirve para integrar los componentes que se han indicado anteriormente.

En primer lugar se actualizan los datos odométricos y del láser mediante `robotdata.UpdateData()`.

Si se han obtenido datos de la odometría (valor de retorno `ODOM_DATA` o `DATA_ODOM LASER`) se calcula el incremento que ha de utilizarse para la etapa de predicción del algoritmo de localización a partir de la transformación inversa de la posición odométrica previa y su composición con la posición odométrica recién obtenida. La nueva posición se guarda como posición antigua para la siguiente llamada a la función. En caso de que se haya añadido un ruido adicional (a través del cuadro de diálogo), se inyecta éste en el incremento de odometría mediante composición con aquél. La posición odométrica con el ruido extra se guarda en un vector de la STL para su representación gráfica. Por último se calcula la predicción de la posición odométrica mediante `loc.KalmanPos` con los argumentos correspondientes al incremento de odometría calculado.

Si se han obtenido datos del láser (`robotdata.UpdateData()` ha devuelto `LASER_DATA` o `DATA_ODOM LASER`) lo primero que se hace es procesar la información del mismo que estará disponible en la variable de la clase `CRawLaserData` perteneciente a `robotdata` para que en `proc_laser_data` se tenga el vector `v` con los puntos correspondientes a las medidas del láser (llamada a `proc_laser_data.DefineData(robotdata.laser_data)`). Seguidamente se realizan las operaciones necesarias para calcular los puntos del polígono correspondiente a esa serie de medidas del láser mediante el objeto `proc_laser_data`.

Se obtienen también los ángulos α de la figura 4.2 y se pasan éstos y la información del polígono a variables miembro del objeto `loc`. Se calcula la corrección de la posición estimada por medio de `loc.KalmanUpdate`, pasándole como argumento el vector `v` de la variable `proc_laser_data`.

La posición resultante de esta actualización se guarda en un vector de la STL para representar en la interfaz gráfica la trayectoria corregida mediante el algoritmo de localización y poder compararla con la trayectoria basada únicamente en los datos de la odometría.

Esta nueva posición es la que ha de utilizar el control de movimiento. Como las variables `x` y `y` de la clase `CMoveControl` son públicas, basta con asignarles el valor correspondiente a la posición corregida con el filtro de Kalman.

Lo último que se hace es pasarle a la variable `control` el vector con aquellos puntos del mapa (actualizado en la llamada a `KalmanUpdate`) suficientemente cercanos al robot como vector que contiene los obstáculos a evitar por el control reactivo y efectuar la deformación de la trayectoria mediante `control.ComputeElasticTray()`.

El valor de retorno es 1 si se han actualizado datos de odometría o del láser y 0 en caso contrario.

Esta función va a ejecutarse a modo de bucle, dentro de un *timer* que se define en la clase `CControl13GUIDlg`.

6.2 La clase `CControl13GUIDlg`

Esta clase hereda de la clase pública `CDialog`, perteneciente a las MFC. En ella comienza el hilo principal de ejecución del programa. Desde esta clase se crea el cuadro de diálogo, al que han de añadirse los diferentes botones para realizar el paso de información del usuario a los distintos componentes del sistema. En ella se tiene la variable `g1_wnd` para crear la ventana gráfica y asociarla a él. Como se ha mencionado, proporciona el modo de controlar el ciclo de tareas que ha de realizar el robot mediante un temporizador o *timer* que permite la repetición de una serie de acciones periódicamente. La variable `robot`, de acuerdo con lo visto, es necesaria para obtener la posición corregida y el mapa y hacer posible su utilización desde el control de movimiento en cada momento. La variable `control` se emplea para facilitar el acceso a aquél.

6.3 Interfaz de usuario

Aunque ya se ha mostrado en otras figuras, a continuación se presenta el aspecto de la interfaz de usuario desarrollada. En esta imagen se ve tal y como aparece al iniciarse la ejecución del programa (salvo los colores de fondo y cuadrícula, que por defecto son negro y magenta pero pueden cambiarse pulsando en el teclado la letra 'F'). Seguidamente se describirán sus propiedades y la utilidad de sus botones.

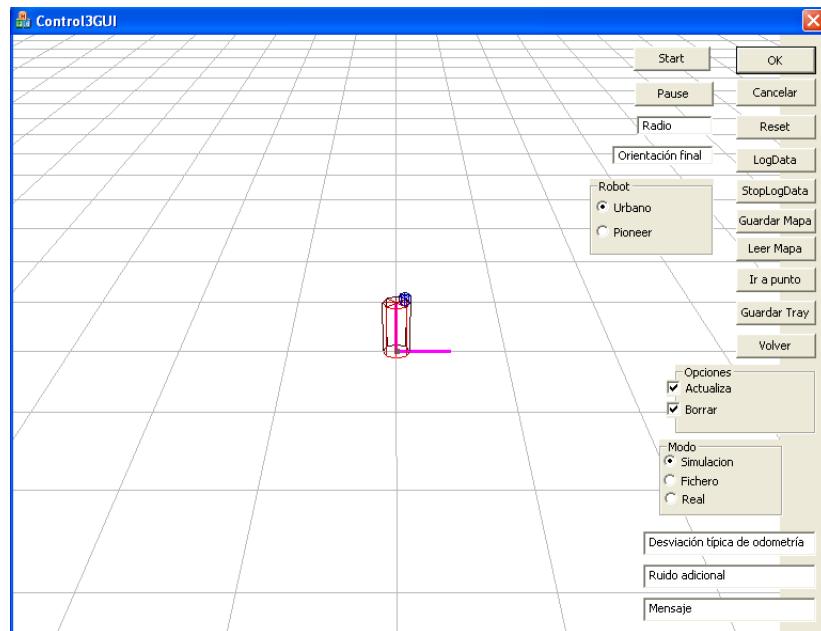


Figura 6.1: Interfaz de usuario

6.3.1 Funcionalidad de los botones del diálogo

OK y Cancelar

Siempre que se pulse alguno de estos botones se cierra la interfaz de usuario y finaliza la ejecución del programa.

Reset

Se emplea para anular la definición de una trayectoria y de una serie de obstáculos realizados mediante el ratón.

Guardar Mapa

Se utiliza para guardar los puntos de un mapa en un fichero. Ha de ser pulsado tras finalizar el recorrido del robot por el entorno que se desee representar. Al ser marcado se abre el típico cuadro de Windows *Guardar Como* para seleccionar el directorio y el nombre del fichero. De esta forma, no tienen que realizarse nuevos mapas cada vez sino que pueden ser leídos mediante *Leer mapa*:

LeerMapa

Muestra un cuadro de Windows tipo *Abrir* para establecer como mapa los puntos leídos de un fichero creado mediante *Guardar Mapa*.

Ir a punto

Se emplea para que el robot siga una trayectoria definida sobre la interfaz gráfica mediante el ratón.

Guardar Tray

Permite guardar la trayectoria que sigue el robot, para que luego pueda regresar al punto de partida. Ha de pulsarse en el momento en que se desee que empiece a guardarse el camino, normalmente antes de que comience el movimiento del robot.

Volver

Cuando se pulsa este botón el robot emprende el camino de regreso por la misma trayectoria seguida hasta el momento.

Opciones

Dentro de este grupo de botones de tipo *check-box* se ofrecen algunas alternativas de funcionamiento del sistema:

- Actualiza: permite añadir o no puntos al mapa en función de las medidas del láser. Si no se ha leído ningún mapa mediante *Leer Mapa* y no se marca esta opción, el mapa permanecerá vacío y no podrá efectuarse la localización. Puede variarse la condición elegida a lo largo del funcionamiento del sistema.

- Borrar: sirve para establecer si se han de borrar los puntos dinámicos mediante el algoritmo del polígono envolvente o no. En caso afirmativo se dibuja sobre la pantalla el polígono correspondiente en cada momento y los puntos que han sido borrados aparecen en color cyan.

Modo

Este grupo de botones *radio-box* se emplea para seleccionar el tipo de conexión que se desea.

- Simulación: establece el funcionamiento del sistema en modo simulación
- Fichero: establece el funcionamiento del sistema en modo fichero
- Real: establece el funcionamiento del sistema en modo real

LogData

Permite guardar en un fichero los datos de odometría y del láser mediante la llamada al método `StartLogData` de la clase `CRobotData`.

StopLogData

Se utiliza para cerrar el fichero anterior y de este modo dejar de registrar los datos. Básicamente realiza una llamada a `StopLogData` de la clase `CRobotData`.

Lógicamente, los dos botones anteriores no se encuentran activados si el modo de funcionamiento es tipo fichero.

Desviación típica de odometría

En esta *edit box* puede introducirse el valor de la desviación típica en los datos de la odometría que se ha de utilizar en el filtro de Kalman. Si en ella no se escribe ningún valor, se emplea una desviación típica de 0.1.

Ruido adicional

El valor que se introduzca en esta *edit box* será inyectado como ruido adicional a la odometría. Permite evaluar el funcionamiento del algoritmo de localización con distintas calidades de los datos odométricos. Si no se escribe nada en el cuadro, el valor por defecto es 0.

Mensaje

En esta *edit box* se muestra información sobre la última acción que se le ha pedido que realice al robot (*Guardando trayectoria*, *Siguiendo trayectoria*)...

Start

Con este botón se inicia un temporizador para que se ejecute el ciclo de tareas del robot cada cierto tiempo (periodo de 10ms en los modos simulación y fichero y de 150ms en modo real). Conviene utilizarlo una vez se han seleccionado los parámetros de funcionamiento.

Pause

Este botón mata el temporizador y con ello dejan de realizarse las actualizaciones del ciclo, por lo que el sistema se mantiene estático hasta que vuelva a lanzarse el ciclo de tareas pulsando de nuevo el botón *Start*.

Radio

El valor que se introduzca en esta *edit box* se utilizará como radio de curvatura para suavizar la trayectoria. Si no se escribe nada en ella el valor por defecto es 1m.

Orientación final

Esta *edit box* se utiliza si se desea concretar una orientación del robot en el punto final de la trayectoria. Si se ha pulsado la opción de *Volver* la orientación final será por defecto la que tenía el robot en el momento en que se comenzó a guardar la trayectoria. En caso contrario el robot mantendrá la orientación con la que llegue al destino.

Robot

Este grupo de botones se utiliza para escoger el tipo de robot que se va a emplear.

- Urbano: opciones correspondientes al robot Urbano (B21r de iRobot).
- Pioneer: opciones correspondientes al robot Pioneer (P3AT de Activ-Media Robotics).

Las principales diferencias entre seleccionar uno u otro robot se hallan en el modo en que se realiza la conexión, en la forma en que se envían los comandos de velocidad y se recibe la información de la odometría y del láser y en los parámetros del control de movimiento. La separación de ambos casos se realiza, por lo tanto, en las clases `CControl3GUID1g` (conexión en el botón *Start* y envío de comandos en el *timer*), `CRobot` (obtención de las medidas del sistema odométrico y de las observaciones proporcionadas por el láser) y `CMoveControl` (regulador para seguir la trayectoria).

6.3.2 Selección de opciones mediante ratón o teclado

Por medio del ratón pueden realizarse los siguientes cambios relativos a la visualización:

- Acercamiento y alejamiento del punto de vista mediante giro de la rueda del ratón en uno u otro sentido. El mismo resultado se obtiene moviendo el ratón con el botón derecho pulsado.
- Variación del punto de vista manteniendo apretado el botón izquierdo del ratón mientras se mueve.
- Desplazamiento del origen de coordenadas del sistema de referencia global sobre el plano representado en la interfaz gráfica pulsando simultáneamente el botón izquierdo del ratón y el botón *Ctrl* del teclado y arrastrando el cursor.
- Desplazamiento del origen de coordenadas de sistema global sobre el eje z del mismo al mover el ratón y mantener pulsados el botón derecho y la tecla *Ctrl*.
- Creación y representación de obstáculos mediante doble click con el botón izquierdo del teclado. Si existe una trayectoria definida esta acción provoca la deformación de la misma.
- Definición de puntos de una trayectoria. Si no se trata del primer punto de la misma, se obtiene la trayectoria redondeada inmediatamente después de haberse añadido el nuevo punto.

A través del uso único del teclado pueden hacerse más modificaciones:

- Cambio del tipo de representación de 2D a 3D y viceversa con la tecla correspondiente a la letra 'P'

- Cambio del color del fondo y algunos otros colores con la tecla correspondiente a la letra 'F'
- Ocultar o mostrar la cuadrícula que muestra el plano del movimiento con la tecla correspondiente a la letra 'G'
- Mostrar o no mostrar la trayectoria definida, la trayectoria modificada y los obstáculos mediante las teclas correspondientes a las letras 'T', 'E' y 'O' respectivamente
- Aumentar y disminuir el tamaño de las celdas de la cuadrícula por medio de las teclas correspondientes a las letras 'M' y 'L'
- Movimiento del robot en modo teleoperado. La letra 'W' se utiliza para aumentar la velocidad de avance; la 'S', para disminuir la velocidad de avance; la 'D', para incrementar la velocidad de giro (sentido de las agujas del reloj, figura 5.1); la 'A' para disminuir la velocidad de giro (o incrementarla en sentido antihorario) y la barra de espaciado se emplea para hacer nulas las velocidades del robot y que éste se detenga.

6.4 Pruebas y resultados

En estas pruebas se pone de manifiesto la interacción entre las dos partes principales del sistema desarrollado.

6.4.1 Pruebas en modo *simulación*

Como ya se ha visto, en este modo de funcionamiento no se realiza localización del robot por no haber disponibilidad de medidas del láser para el robot Urbano. Sin embargo, sí que permite evaluar la deformación de trayectorias en entornos densos en obstáculos mediante el uso de mapas.

Movimiento sobre trayectoria deformada ante los obstáculos definidos por los puntos de un mapa

A continuación se utilizan mapas obtenidos por los métodos descritos en el capítulo 4 de modo que sus puntos sean los que sirvan para deformar la trayectoria. La trayectoria inicial se dibuja en color azul y la deformada, en color verde. Se ha tomado un rango de 1m a la posición del robot para considerar los puntos del mapa como obstáculos. La máxima desviación permitida se ha establecido en 0.3m para evitar deformaciones excesivas y no entrar en situación de bloqueo con demasiada facilidad.

Aquí puede verse la utilidad del borrado de puntos dinámicos del mapa. En los casos anteriores, hay obstáculos que impiden el avance del robot en una cierta zona (situación de bloqueo). Dichos obstáculos, sin embargo, no son puntos reales del mapa sino que corresponden probablemente a sucesivas posiciones de una persona en el momento de la toma de datos. Si el mapa hubiera sido obtenido con la opción de borrado activada, como resultaría conveniente, el robot podría llegar hasta el final del recorrido planificado:

El número de obstáculos considerados en este tipo de situaciones es significativamente alto, por lo que la trayectoria deformada difiere de la original. En caso de que se desee que el robot regrese al punto de origen de su movimiento se empleará como trayectoria nominal sobre la que realizar los cambios oportunos la trayectoria seguida en el camino de ida, siendo ésta una trayectoria deformada en base a la trayectoria nominal inicial. En la figura 6.4 se muestra en color azul la trayectoria seguida durante la ida (trayectoria nominal del camino de vuelta) y en color rosa la trayectoria modificada durante el camino de regreso.

Construcción de un mapa mediante movimiento controlado del robot y deformación de la trayectoria ante los puntos del mismo con el robot Pioneer

Con este robot, el simulador proporcionado por el fabricante permite disponer de las medidas ficticias del láser en un entorno hipotético representado en el mismo por medio de un mapa geométrico de líneas. Así, puede verse el comportamiento del sistema desarrollado a la hora de construir el mapa de puntos de un entorno como el dado por aquél mapa. En la figura 6.6 se muestran los resultados de un primer experimento. En ella pueden apreciarse algunos detalles interesantes.

Como primera conclusión podría destacarse la insuficiencia de un valor de 0.1 en la estimación del ruido de la odometría, lo que afecta a la construcción del mapa y hace que se creen paredes dobles en algunos casos. La posición corregida por el filtro aparece en color rojo, mientras que la de la odometría es la que se dibuja en color verde. Al final del recorrido empieza a apreciarse la desviación en la posición odometrífica. La trayectoria azul es la nominal, definida por el usuario mediante el ratón, y la trayectoria deformada no se muestra, pero es bastante similar a la trayectoria real seguida(trayectoria roja). Puede verse que se deforma adecuadamente en los puntos cercanos a las paredes o muebles.

En un segundo experimento se mejoró la estimación del ruido de la odometría (estableciéndose en un valor de 0.5), con lo que aumenta la calidad del mapa construido. Posiblemente se obtendrían mejores resultados incre-

mentando algo más dicho valor. Los colores empleados son los mismos que en el caso anterior, pero sólo se representa el último tramo de la trayectoria nominal definida. De nuevo se puede observar la deformación de la trayectoria en las cercanías de los obstáculos y el efecto de la localización, más fácilmente apreciable en observación simultánea en tiempo real de ambos simuladores.

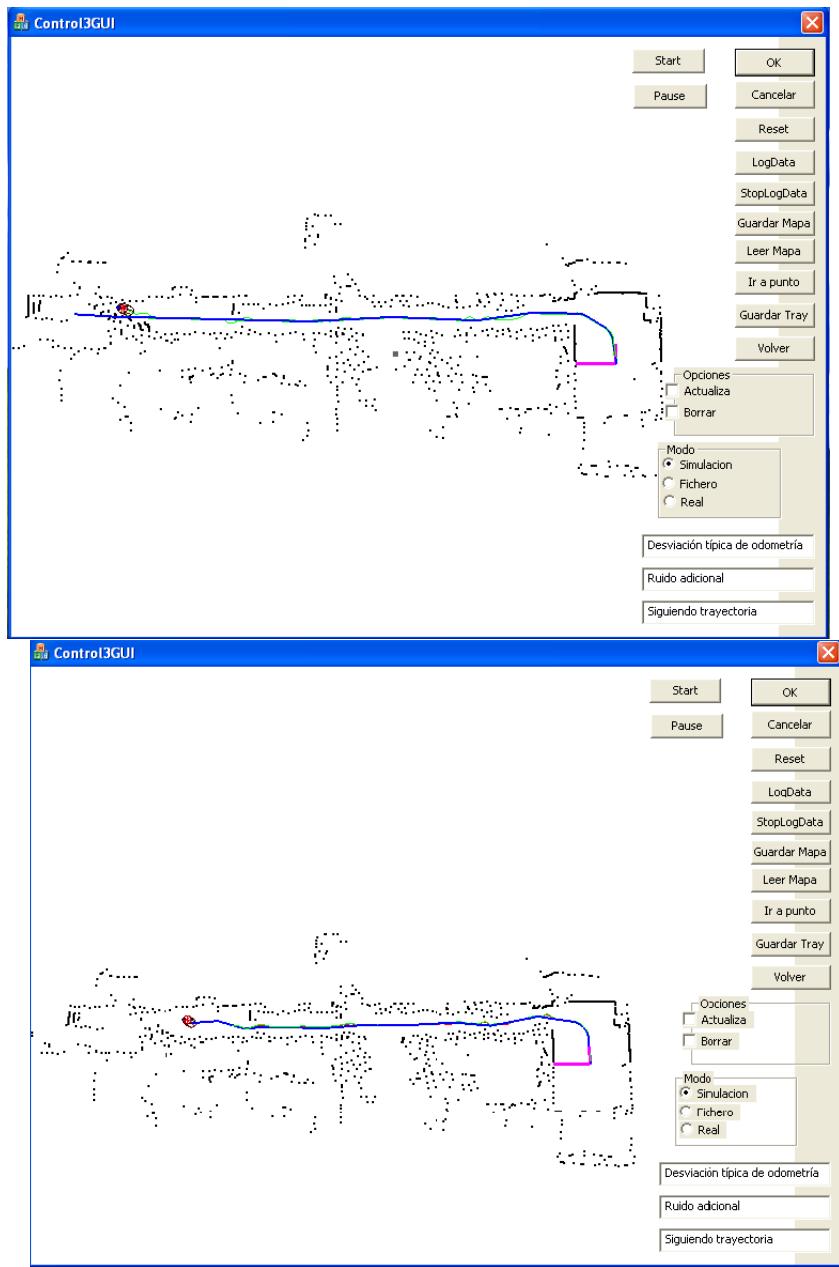


Figura 6.2: Deformación de trayectorias definidas en el laboratorio de DISAM-UPM

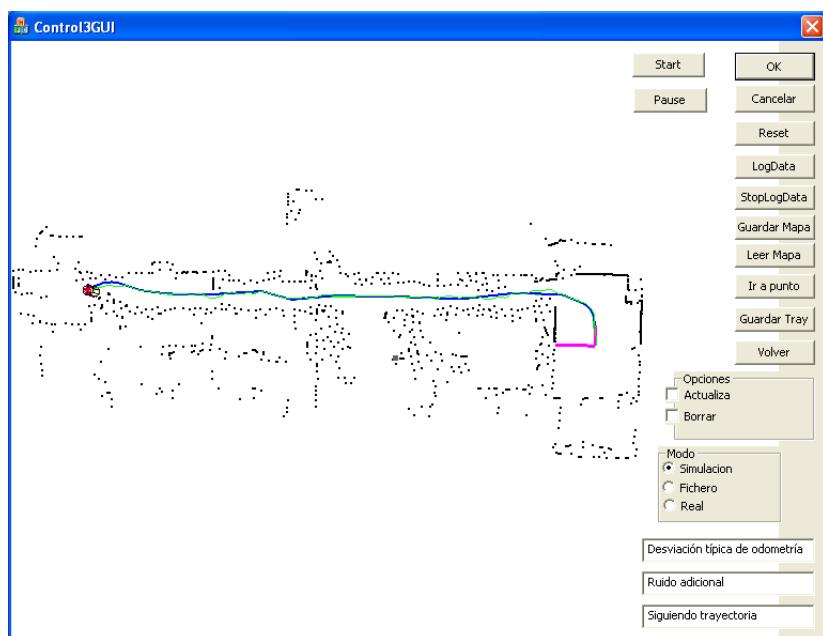


Figura 6.3: Deformación de trayectorias en un mapa obtenido con borrado de puntos dinámicos

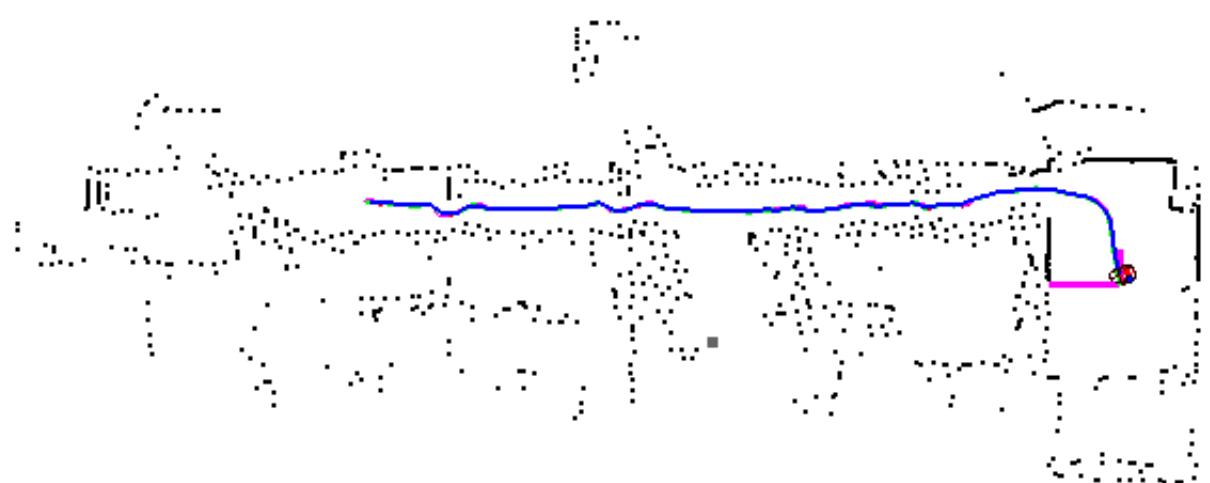


Figura 6.4: Trayectoria deformada en el camino de ida y nuevas deformaciones en el camino de vuelta

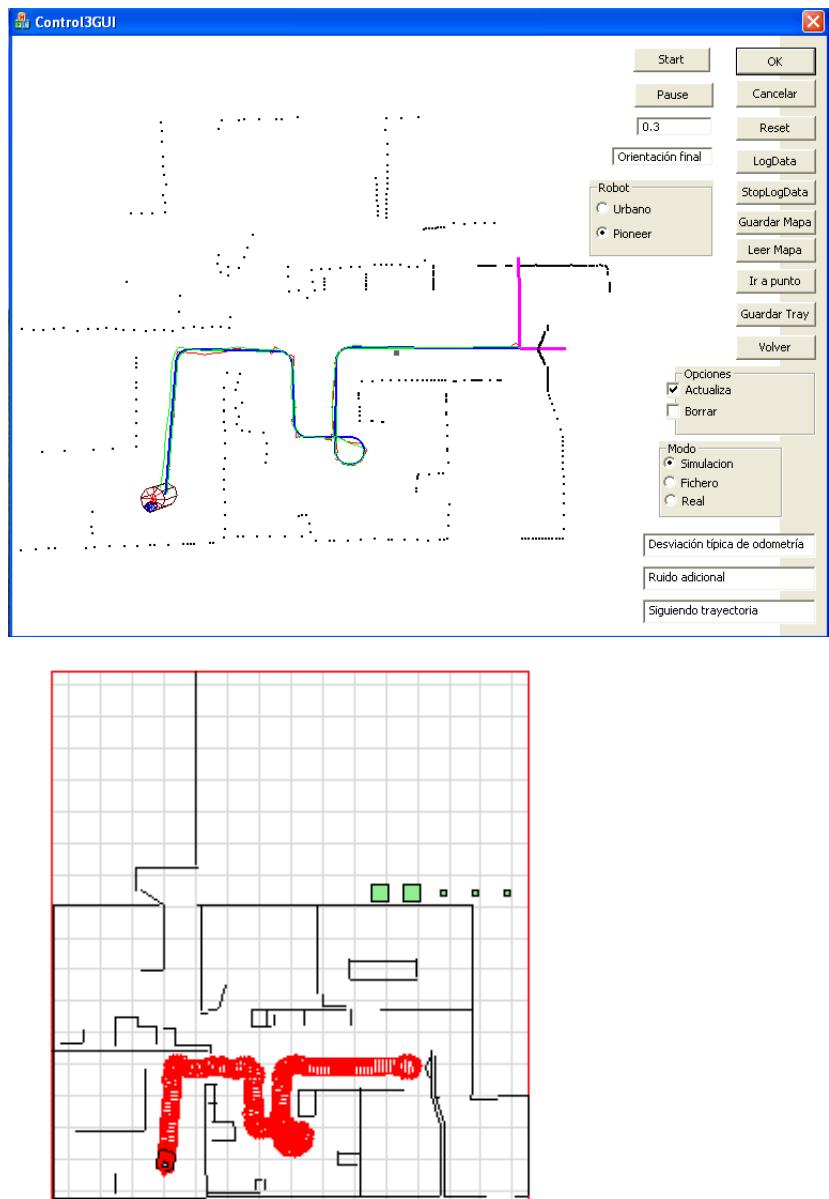


Figura 6.5: Primer experimento para ver el comportamiento del sistema con el robot Pioneer

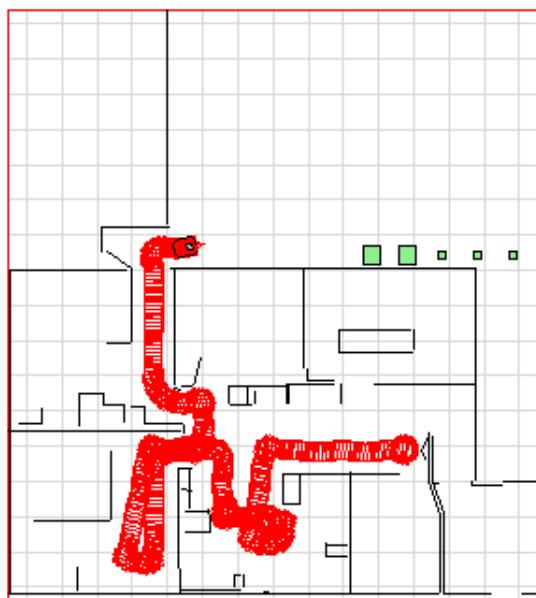
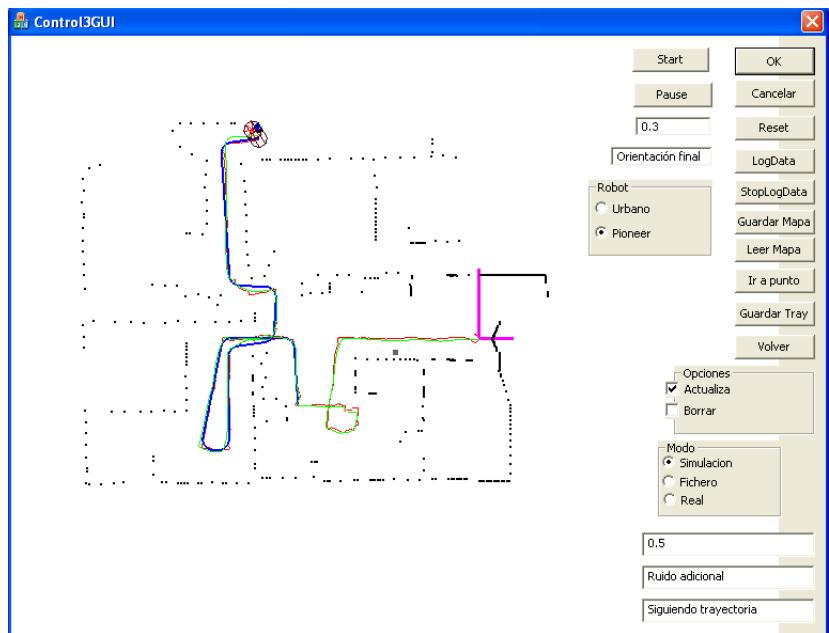


Figura 6.6: Segundo experimento para ver el comportamiento del sistema con el robot Pioneer

Parte III

Información Complementaria

Capítulo 7

Conclusiones y trabajos futuros

En este capítulo se realiza una reflexión sobre los resultados alcanzados y se presenta una recopilación de posibles avances a desarrollar en próximas líneas de trabajo.

7.1 Conclusiones

En el desarrollo del presente proyecto se han abordado diferentes facetas de la problemática actual en la navegación de robots móviles, fundamentalmente dentro de entornos de interiores. Se ha construido un sistema robusto y fácilmente utilizable por diferentes plataformas robóticas mediante sencilla adaptación del envío de los comandos de bajo nivel y la recepción de los datos de los sensores (odometría y láser). Se dispone de una interfaz gráfica que permite evaluar el comportamiento del sistema de una forma sencilla, incorporando también todos los botones necesarios para la selección de opciones. El funcionamiento puede ser en modo de simulación, basado en la utilización de datos del estado del robot y del entorno almacenados en un fichero en experimentos previos, o en operación con los robots reales.

A continuación se exponen los resultados, conclusiones y aportaciones más relevantes de cada una de las dos partes principales que constituyen el proyecto.

- **Control de movimiento, planificación de trayectorias y control reactivo**

Esta primera componente del proyecto abarca el diseño de un sistema de control que permite al robot desplazarse sobre una trayectoria suavizada en los puntos de paso definidos, de modo que antes de llegar ex-

actamente hasta uno de dichos puntos comienza a orientarse hacia el siguiente. Este módulo de control garantiza que el robot no choca contra ningún obstáculo, ya que se sigue una estrategia de alto nivel en la que la trayectoria nominal anterior se deforma en tiempo real para apartarse de ellos y, además, el regulador implementado impide que el robot se separe de esta nueva trayectoria ya libre de cualquier objeto detectado por el sensor láser. En la aplicación particular aquí realizada para el robot Urbano, la presencia de obstáculos a alturas inferiores a la posición del láser sobre el robot no se refleja en la trayectoria deformada obtenida; en este caso actúan los sensores de ultrasonidos del robot y mediante control reactivo de bajo nivel basado en comandos de cambio de velocidad de avance o giro se evita colisión alguna, regresando después el robot a la última trayectoria obtenida. En caso de que el robot no pueda salvar algún obstáculo o no pueda hacerlo sin desviarse más de la cuenta, el robot se para y se encontrará en situación de bloqueo, a la espera de nuevas órdenes o de la definición de otro camino.

• Localización y mapas

Para que el control en bucle cerrado no se vea perjudicado por los errores acumulados en los datos de los encóders sobre la posición del robot en cada instante, se ha utilizado un algoritmo de localización basado en el filtro extendido de Kalman (EKF). Se trata de una aplicación innovadora, al utilizar directamente los mismos puntos detectados por el láser tanto para la construcción de un mapa del entorno como para corregir la estimación de la posición del robot en cada momento. El problema que presenta consiste precisamente en que no se tiene en cuenta la incertidumbre del propio mapa, aunque este hecho es generalizado en gran parte de los trabajos realizados en este ámbito. La asociación de datos se realiza sólo tras su aceptación mediante el test de Mahalanobis, con un intervalo de confianza del 95 % en la distribución χ^2 con 3 grados de libertad. Los resultados obtenidos con los datos tomados en las instalaciones de Intel y en el Museo de las Ciencias Príncipe Felipe de Valencia son especialmente buenos. Respecto al primer caso, resultaría prácticamente imposible cerrar un bucle de este tipo a partir de datos procedentes exclusivamente de la odometría, y más si éstos tienen tan poca calidad. El borrado de puntos antiguos permite mejorar la localización en la llegada a la zona cercana al punto de partida. Respecto al caso del museo, cabe destacar la gran extensión del área explorada así como la coincidencia casi exacta de patrones en partes

del mapa elaboradas al cabo de tiempos muy distintos y a pesar de la complejidad geométrica del entorno. La posibilidad de borrar aquellos puntos del mapa que dejen de observarse en las inmediaciones del robot mejora la fiabilidad de los mapas construidos. Con ello pueden evitarse errores en la asociación de datos y se impide que el robot tenga que esquivar obstáculos ya inexistentes.

La interacción de ambos subsistemas permite una navegación precisa y segura del robot en entornos complejos, incluso con presencia de obstáculos dinámicos en los mismos. Todas las tareas que han de desempeñarse suponen una carga computacional relativamente elevada, lo que ocasionaba problemas en la ejecución, sobre todo con los robots reales. Se ha tratado de mejorar el código para minimizar el tiempo de procesamiento, lográndose finalmente unos buenos resultados al respecto.

7.2 Líneas futuras

Los principales puntos para caracterizar el planteamiento establecido en la evolución futura del proyecto realizado son los siguientes:

- Utilización del sistema de localización para el robot Pionner P3At tan pronto como éste disponga de un láser que le permita obtener información sobre su entorno.
- Extensión de los modelos geométricos 2D a 3D mediante la utilización de una muñeca que incline el láser. Probablemente será necesario emplear nuevos métodos de procesamiento de datos para gestionar el incremento en el número de medidas.
- Definición de trayectorias iniciales mediante el seguimiento de una persona o mediante comandos enviados a Urbano por voz.
- Dotar al robot de una autonomía más completa, incorporando conductas que le permitan seguir un proceso de aprendizaje por sí mismo.

Apéndice A

Transformaciones relativas

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

$$\vec{x}_{ij} = [x_{ij} \ y_{ij} \ \theta_{ij}]^T \quad (\text{A.1})$$

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

$$\vec{x}_{ik} = \vec{x}_{ij} \oplus \vec{x}_{jk} = \begin{pmatrix} x_{ij} + x_{jk}\cos\theta_{ij} - y_{jk}\sin\theta_{ij} \\ y_{ij} + x_{jk}\sin\theta_{ij} + y_{jk}\cos\theta_{ij} \\ \theta_{ij} + \theta_{jk} \end{pmatrix} \quad (\text{A.2})$$

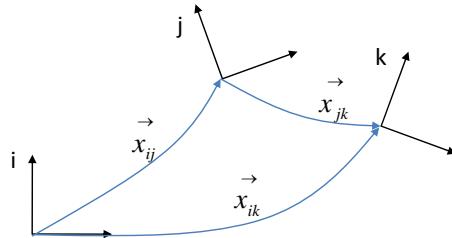


Figura A.1: Composición de transformaciones relativas

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

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

La jacobiana respecto a la segunda transformación será:

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

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

$$\vec{x}_{ji} = \ominus \vec{x}_{ij} = \begin{pmatrix} -x_{ij}\cos\theta_{ij} - y_{ij}\sin\theta_{ij} \\ x_{ij}\sin\theta_{ij} - y_{ij}\cos\theta_{ij} \\ -\theta_{ij} \end{pmatrix} \quad (\text{A.5})$$

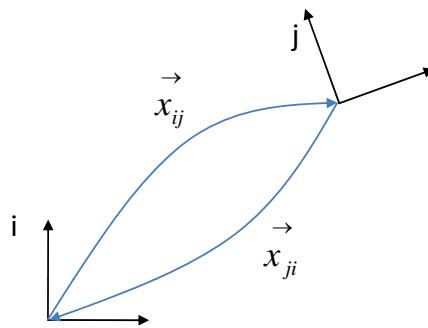


Figura A.2: Inversión de una transformación relativa

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:

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

Apéndice B

EKF para medidas de distancia

En el caso de utilizar solamente medidas de distancia a los puntos del mapa, sin que se conozca su orientación respecto a la posición del robot, la ecuación de medida del EKF es unidimensional en lugar de tener dos componentes:

$$h_{ij} = \sqrt{(x_R - x_{lj})^2 + (y_R - y_{lj})^2} - d_i = 0, \quad (\text{B.1})$$

donde d_i es cada medida de distancia dada por algún sensor estereoceptivo. Se omiten nuevamente los subíndices correspondientes al número de iteración para simplificar la notación.

De este modo, las matrices $H_{x_{ij}}$ y $H_{z_{ij}}$ quedan:

$$H_{x_{ij}} = \left[\frac{x - x_{lj}}{\sqrt{(x - x_{lj})^2}}, \frac{y - y_{lj}}{\sqrt{(y - y_{lj})^2}}, 0 \right]^T \quad H_{z_{ij}} = -1$$

La covarianza de una innovación individual quedará:

$$s_{ij} = H_{x_{ij}}^T \tilde{P} H_{x_{ij}} + H_{z_{ij}}^2 \sigma_d^2 \quad (\text{B.2})$$

con σ_d^2 como varianza en las medidas de distancia. Puede apreciarse que la dimensión de este producto es 1. De este modo, las asociaciones se realizarán minimizando la distancia de Mahalanobis, que en este caso resulta ser $\frac{h_{ij}^2}{s}$.

La matriz h estará formada por aquellos valores de h_{ij} que hayan minimizado esa distancia para cada observación. Si el número de observaciones asociadas es t , tendrá dimensiones $(t \times 1)$. La matriz H_x tendrá como filas las matrices H_{x_i} correspondientes a cada asociación realizada. Sus dimensiones serán $(t \times 3)$. La matriz H_z será la opuesta de la matriz identidad de dimensión $(t \times t)$. La matriz R también será diagonal, con elementos iguales a σ_d^2 .

Así, la matriz S tendrá dimensiones $(t \times t)$ y K , $(3 \times t)$.

Por último, quedará:

$$\hat{x} = \tilde{x} - Kh \quad (\text{B.3})$$

$$\hat{P} = (I - KH_x)\tilde{P} \quad (\text{B.4})$$

Estas ecuaciones fueron implementadas en Matlab sobre una base de código elaborado por Diego Rodríguez-Losada para ver los resultados proporcionados en la localización del robot a partir de un fichero con datos de observaciones. En la figura B.1 se muestra en color verde la trayectoria odometrífica correspondiente a un movimiento circular teórico (mostrado en color azul). En rojo se representa la trayectoria corregida mediante esta aplicación del EKF. Puede verse que la información proporcionada por la distancia a los puntos resulta insuficiente para corregir los errores acumulados por la odometría y, al cabo de un cierto tiempo, el robot terminaría perdido.

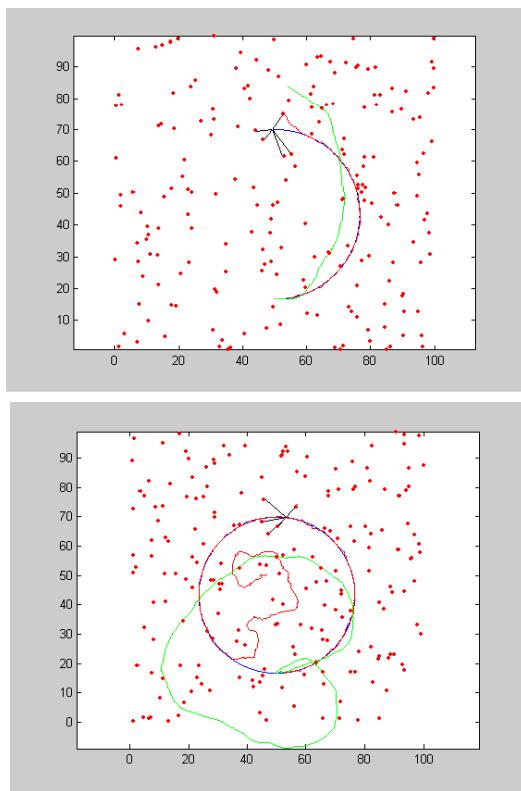


Figura B.1: Corrección de la posición en un movimiento circular uniforme mediante medidas de distancia a las observaciones

Con la información completa de las coordenadas de los puntos observados la asociación de datos se realiza correctamente y esto no sucede, como se

puede apreciar en la figura B.2. Se concluye por este motivo que no resulta conveniente utilizar la simplificación desarrollada aquí, aunque podría haber sido de utilidad y, en cualquier caso, sirvió para asentar conocimientos sobre el EKF.

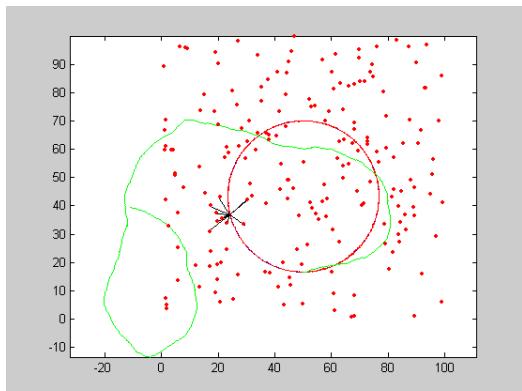


Figura B.2: Corrección de la posición en un movimiento circular uniforme mediante las coordenadas de las observaciones

Apéndice C

Hardware utilizado

Aunque ya se han introducido algunas descripciones del hardware empleado, a continuación se añade algo más de información sobre sus características.

C.1 Plataforma móvil B21r

La plataforma más usada en los experimentos del proyecto corresponde al modelo B21r de la empresa iRobot. Se trata de una plataforma holonómica cuyo sistema motriz es de tipo synchro-drive con cuatro ruedas directrices y motrices. Los sensores de infrarrojos y de ultrasonidos que posee no se han utilizado por su menor precisión, fiabilidad y rango respecto al escáner láser LMS200 (sección C.3).



Figura C.1: Plataforma móvil B21r de iRobot

En la parte superior del robot hay dos pulsadores rojos que sirven para activar los frenos del robot en caso necesario. Pueden quitarse pulsando nuevamente alguno de estos botones o a través del panel de control del robot girando el botón negro situado junto a él. Las principales características de este robot se presentan en la siguiente tabla:

Característica	B21r
Diámetro	52cm
Altura	106cm
Distancia al suelo	2.54cm
Peso	122.5kg
Capacidad de carga	90kg
Baterías	4x12V estancas plomo-ácido
Autonomía	6 h
Sistema Motriz	Synchro-drive 4 ruedas
Motores	4 servomotores 24V
Ruedas	Goma maciza
Diámetro ruedas	11cm
Ancho ruedas	3cm
Sistema de giro	Synchro-drive 4 ruedas
Radio máxima curvatura	0cm
Radio giro	0cm (robot holonómico)
Máxima velocidad avance	0.9m/seg.
Máxima velocidad de giro	167°/seg.
Terreno	Interiores planos y uniformes
Resolución avance	1mm
Resolución giro	0.35°

Cuadro C.1: Características del robot B21r de iRobot

El robot tiene un PC base con Red Hat Linux. También dispone de un PC secundario con sistema operativo Windows. El acceso de bajo nivel al hardware del robot lo llevan a cabo unos programas servidores incluidos en el software proporcionado por iRobot (denominado *Mobility* y basado en el estándar de programación distribuida CORBA). El único de estos servidores que se ha utilizado es el b21server (alias base), encargado del manejo de los motores y de la obtención de medidas de los sensores.

C.2 Plataforma móvil Pioneer P3-AT de MobileRobots/ActivMedia Robotics

Este robot no ha podido ser utilizado en la mayor parte de las pruebas por no disponer de sensor láser para detectar obstáculos, construir mapas y situarse correctamente en su entorno de operación. Sin embargo, la utilización del sistema implementado será prácticamente inmediata tan pronto como se tenga un láser instalado.

Se trata de una plataforma utilizable en todo tipo de terrenos (AT son las siglas de *All Terrain*) También permite un alto grado de carga. Posee tracción a las cuatro ruedas y la realización de giros se basa en el deslizamiento. Esto repercute en la estimación de la odometría, resultando ser bastante peor que la proporcionada por el B21r.



Figura C.2: Plataforma móvil Pioneer P3-AT

El cuerpo del robot es de aluminio y su parte delantera se puede desmontar con facilidad por medio de unos tornillos. En la plataforma superior del robot está situado el panel de control, que permite acceder al microcontrolador y dispone de algunos LEDs indicadores de estado. En él también se haya el puerto serie RS-232 utilizado para establecer la conexión con el equipo en el que se ejecuta el software de control desarrollado.

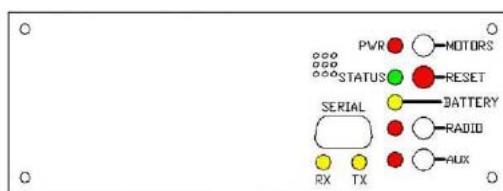


Figura C.3: Panel de control

A continuación se resumen en una tabla las principales características de este robot:

Característica	Pioneer P3-AT
Largo	50cm
Ancho	49cm
Alto	26cm
Distancia al suelo	8cm
Peso	12kg
Carga útil	32kg
Cuerpo	1.6mm aluminio pintado h
Baterías	12V estanca, plomo-ácido
Autonomía	4-8 h
Sistema Motriz	4 ruedas motrices
Ruedas	Neumáticas nylon
Diámetro ruedas	21.5cm
Ancho ruedas	8.8cm
Sistema de giro	Deslizamiento diferencial
Radio máxima curvatura	40cm
Radio giro	0cm
Máxima velocidad avance	1.2m/seg.
Máximo escalón	10cm
Máximo hueco	15.2cm
Máxima pendiente	40
Terreno	Asfalto, tierra, césped, etc.
Encoders	500pulsos
Procesador	Hitachi H8S

Cuadro C.2: Características del robot Pioneer P3-AT de MobileRobots

C.3 Escáner láser LMS200

Este escáner láser se ha utilizado como único sensor estereoceptivo debido a sus elevadas prestaciones en comparación con las de los sensores de infrarrojos y ultrasonidos. Se encuentra instalado en la parte superior del robot Urbano (a 1.2m de altura), desplazado 16.8cm hacia delante con respecto al centro de la plataforma B21r. El fabricante es la empresa alemana SICK.



Figura C.4: Láser LMS200 de SICK

Utiliza un haz láser infrarrojo de clase I (inofensivo para el ojo humano incluso en tiempos de exposición prolongados) para la obtención de medidas de distancia con gran precisión y rapidez. El barrido es de 180° y las medidas tomadas son prácticamente independientes de la reflectancia de los objetos.

Presenta varias opciones de funcionamiento que permiten variar el alcance, la precisión y el número de medidas en cada barrido del láser.

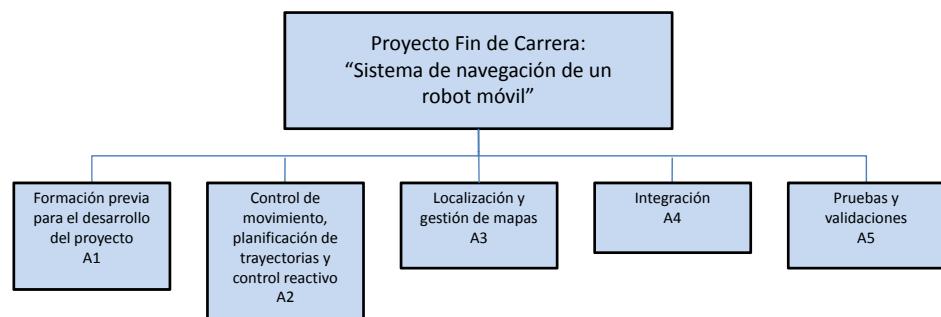
La siguiente tabla contiene sus principales características:

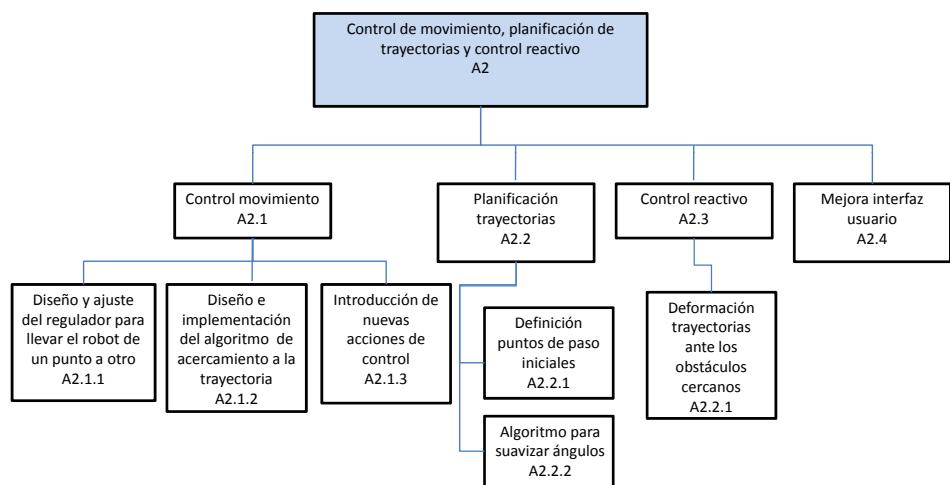
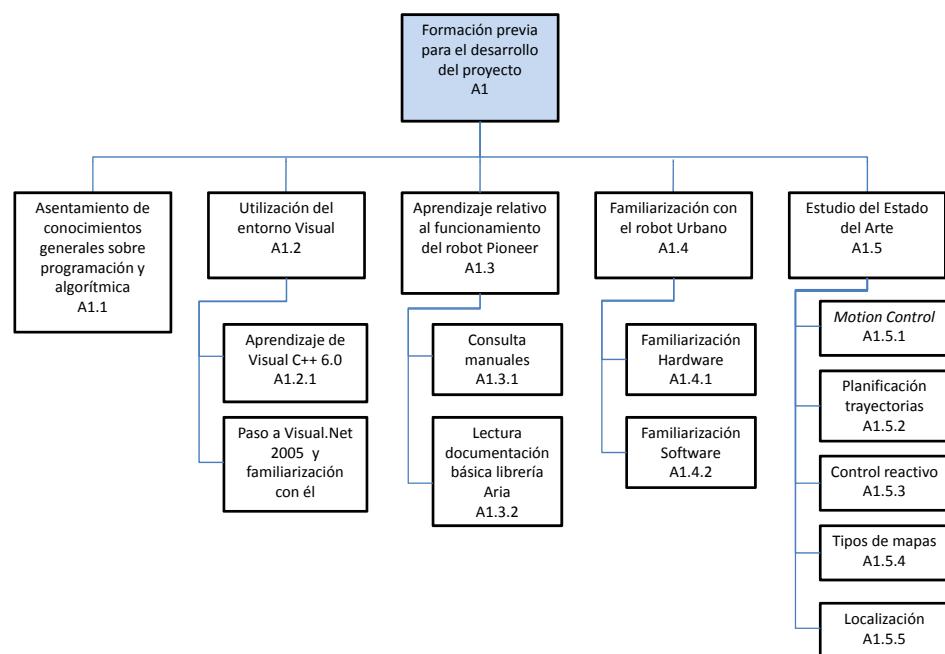
Característica	LMS-200
Resolución angular	1°/0.5°/0.25°
Tiempo de respuesta	13ms/26ms/53ms
Resolución	10mm
Error sistemático (modo mm)	±15mm
Error estadístico (1 Sigma)	5mm
Clase láser	1 (eye-safe)
Máxima distancia	80m
Interfase de datos	RS422/RS232
Velocidad transferencia	9.6/19.2/38.4/500 kBaud
Alimentación	24V DC ± 15 %
Consumo	20 W
Peso	4.5kg
Ancho	155mm
Alto	210mm
Profundo	156mm

Cuadro C.3: Características del láser SICK LMS-200

Apéndice D

Estructura de Descomposición del Proyecto (EDP)





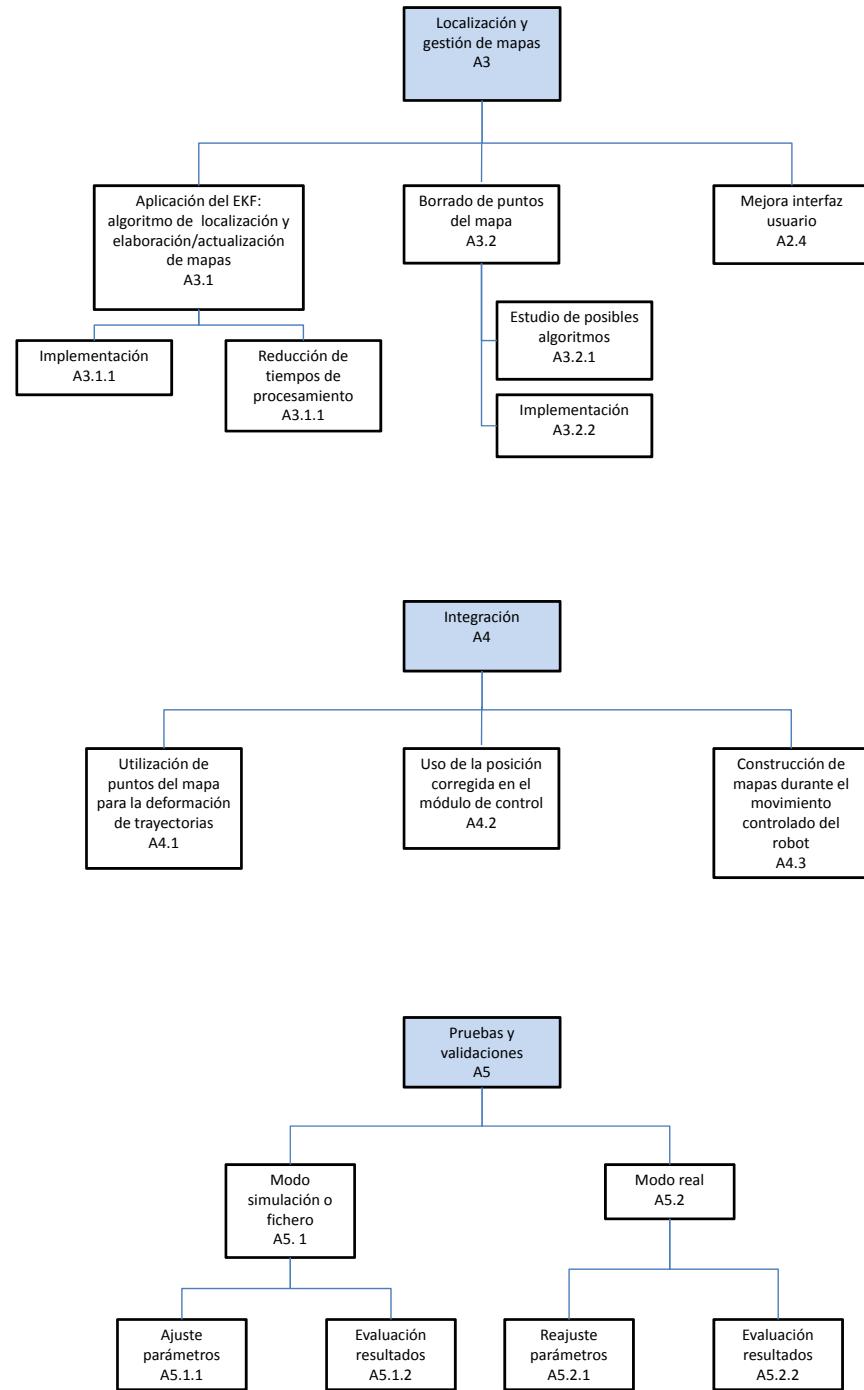


Figura D.1: Estructura de Descomposición del Proyecto

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] Álvaro Arranz, Jorge Baliñas, Sebastián Bronte, Josué García, Daniel González, Javier Gutiérrez, Ángel Llamazares, Fernando Rojas, and Víctor Sanz. Aplicaciones de robots móviles. Technical report, Universidad de Alcalá, 2006.
- [3] Diego Rodríguez-Losada, Fernando Matía, Agustín Jiménez, Ramón Galán, and Gerard Lacev. Implementing map based navigation in Guido, the robotic smartwalker. In *IEEE International Conference on Robotics and Automation*, 2005.
- [4] Diego Rodríguez-Losada. *SLAM Geométrico en Tiempo Real para Robots Móviles en Interiores basado en EKF*. PhD thesis, Universidad Politécnica de Madrid, ETSI Industriales, 2004.
- [5] Diego Rodriguez-Losada, Fernando Matia, Agustin Jimenez, Ramon Galan, and Gerard Lacev. 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 Bruyninx. *Kalman Filters: A Tutorial*, 1999.

Programas empleados

- Microsoft Visual C++ 6.0 y Visual Studio .NET 2005 (Visual Studio 8) para el desarrollo del software.
- Editor de L^AT_EX WinEdt para la elaboración del presente documento.
- Microsoft Project para la obtención de figuras y diagramas de planificación del proyecto.
- Microsoft Office PowerPoint para la creación de figuras y esquemas generales a incluir en esta documentación.
- Subversion como controlador de versiones.