Universidad Politécnica de Madrid

Escuela Universitaria de Informática

Departamento de Sistemas Inteligentes Aplicados

Trabajo Fin de Carrera

Localización probabilística en un robot con visión local

Mª Angeles Crespo Dueñas

Junio 2.003

Universidad Politécnica de Madrid

Escuela Universitaria de Informática

Departamento de Sistemas Inteligentes Aplicados

Trabajo Fin de Carrera

Localización probabilística en un robot con visión local

Director: José Mª Cañas Plaza
Ingeniero de Telecomunicación

Ponente: Fco Jaime Serradilla García
Doctor en Informática

Autora: Mª Angeles Crespo Dueñas

Junio 2.003

A mi padre

Agradecimientos

Al Departamento de Robótica de la Universidad Rey Juan Carlos, profesores y alumnos, por *adoptarme* y en especial a José Mª Cañas por su dedicación y paciencia.

A Francisco Serradilla del Departamento de Sistemas Inteligentes Aplicados de la Escuela Universitaria de Informática por aceptar ejercer de ponente y por ayudarme con los trámites burocráticos.

A Montserrat Parrilla del Departamento de Sistemas del Instituto de Automática Industrial por hacer posible este proyecto presentándome a José Mª y por estar siempre dispuesta a echar una mano.

A Angel Soriano del Instituto Mexicano del Petróleo y Pedro M. Talaván del Instituto Nacional de Estadística por su ayuda con *las matemáticas*.

A mis padres y hermanos por "estar" siempre ahí.

A mis amigo/as por su apoyo.

Índice

Índice de figuras

Índice de cuadros

1. Introducción

Uno de los problemas que se han abordado recientemente en el campo de la robótica móvil es el de la localización. La localización de robots móviles autónomos consiste en determinar la posición y orientación del robot dentro de su entorno. Para resolverlo se han desarrollado múltiples metodologías. En este trabajo fin de carrera se explorarán dos técnicas de localización probabilística para localizar un robot móvil en función de lo que observa con su cámara local.

Para comenzar este capítulo, se introduce una reseña histórica de cómo ha evolucionado la robótica tanto en sus orígenes industriales como en la robótica móvil, que es el área de investigación en la que se enmarca este proyecto. A continuación, se hace referencia al problema de la localización en sí y las herramientas que se utilizan para abordarlo. Posteriormente, se explica brevemente qué es la Robocup, por ser el escenario concreto en el que se aplicarán las técnicas desarrolladas en este proyecto, y se presentan los robots EyeBot y Aibo que pueden participar en diferentes categorías de esta competición de fútbol. Finalizamos el capítulo de introducción detallando la estructura del resto de la memoria.

1.1. Robótica

Los ancestros más antiguos de los robots datan del siglo XIII donde aparecieron los primeros autómatas, mecanismos cuyo objetivo era realizar tareas en forma mecánica de acuerdo a su diseño. Los autómatas de entonces estaban hechos de madera, como por ejemplo caballos colocados en serie que se desplazaban por la acción de la gravedad.

Durante los siglos XVII y XVIII se construyeron en Europa muñecos mecánicos muy ingeniosos que tenían algunas características de robots. Por ejemplo, Jacques de Vaucanson construyó varios músicos de tamaño humano a mediados del siglo XVIII. Se trataba de robots mecánicos diseñados para un propósito específico: la diversión. Hubo otras invenciones mecánicas durante la revolución industrial muchas de las cuales estaban dirigidas al sector de la producción textil, como por ejemplo la hiladora giratoria de Hargreaves (1770), la hiladora mecánica de Crompton (1779) y el telar mecánico de Cartwright (1785). En 1801, Joseph Maria Jacquard inventó un telar automático, el primero capaz de almacenar un programa y controlar una máquina. El "programa" determinaba el dibujo que aparecía en el telar. En 1805, Henri Maillardert construyó una muñeca mecánica (Figura 1) capaz de

hacer dibujos en la que una serie de levas se utilizaban como "el programa" para controlar el dispositivo en el proceso de escribir y dibujar.

Figura 1: Automata de Maillardert (1805)

Desde comienzos del siglo XX hasta la actualidad, la automatización ha ido conquistando terreno. Una obra checoslovaca publicada en 1921 por Karel Capek, denominada *Rossum's Universal Robots*, dio lugar al término robot, traducido de la palabra checa *Robota* que significa servidumbre o trabajador forzado.

Isaac Asimov, a quien se le atribuye el acuñamiento del término Robótica, propuso en 1950 en su libro *Yo, robot* sus tres leyes de la Robótica:

1. Un robot no puede hacer daño a un ser humano o, por medio de la inacción, permitir que un ser humano sea lesionado.

2. Un robot debe obedecer las órdenes recibidas de los seres humanos excepto si entran en conflicto con la primera ley.

3. Un robot debe protegerse a sí mismo siempre y cuando ésto no sea incompatible con las dos primeras leyes.

La robótica industrial nace de las exigencias prácticas en la producción, es un elemento importante de automatización encaminado a la reducción de costes. Los robots industriales operan con un tipo de programación relativamente sencilla en un entorno estable, determinado y con una movilidad restringida. La robótica tiene múltiples aplicaciones prácticas en el *mundo*

Figura 2: Robot que coloca bombones en una caja

real como los telemanipuladores (utilizados, por ejemplo, en cirugía) y los robots móviles teleoperados (utilizados, por ejemplo, en la detección de explosivos).

En 1954, en Estados Unidos se empezaron a patentar robots y en 1961, los científicos Engelberger (considerado como el padre de la Robótica) y Devol crearon el primer robot industrial: el Unimate 2000, utilizado en las líneas de ensamblaje de Ford Company para realizar procesos en serie y tareas de precisión en las que el hombre podía fallar. En 1973 los robots para el área de la industria ya estaban en el mercado y se formó la compañía ABB que es líder en robótica. En 1982, los japoneses crearon un brazo manipulador para usos industriales, el robot Scara (Selective Compliance Arm for Robotic Assembly), del que descienden los aún utilizados en la industria del automóvil para la limpieza de vehículos, tratado de superficie, soldado de diferentes partes y pintura. La Figura 2 muestra un ejemplo actual de un brazo robótico que detecta bombones con su cámara y los coloca en su hueco respectivo en una caja.

Las exploraciones espaciales son otro campo de aplicación de la robótica. Ya en 1976, la NASA envió los manipuladores Viking I y II a Marte. En 1997, se envía el primer vehículo robótico, el Sojourner, cuyo principal objetivo era demostar que pequeños vehículos autónomos pueden operar en Marte.

Por otra parte, el problema de comportamientos autónomos en robots móviles se investiga en las Universidades. En 1970 apareció el primer robot móvil, Shakey (Figura 3), al que se hace referencia como el primer robot

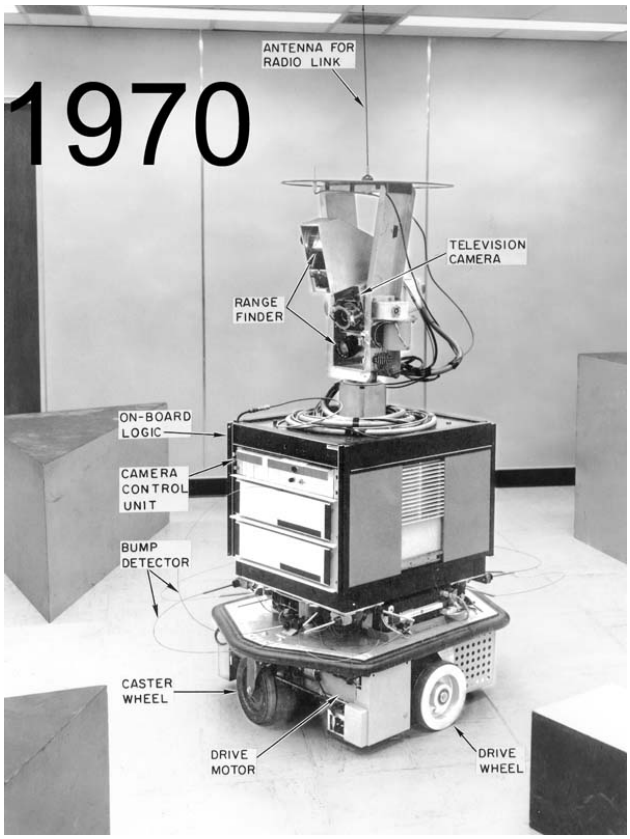

Figura 3: Robot Shakey

"inteligente". En 1983, Odexi Optics presentó robots móviles que trabajaban con sistemas de orugas o ruedas, pero que sólamente podían recorrer sitios planos. El siguiente paso fueron los robots con patas, y el último eslabón de la cadena son los robots androides o humanoides. La preocupación por la autonomía de los robots móviles es una parcela de investigación que continúa abierta hoy día.

En la actualidad, los robots también han hecho su aparición en el sector del entretenimiento. Robots como los *Lego Mindstorm*, un kit de construcción de robots que se vende en tiendas de juguetes y que permite diseñar y programar robots, o el *perrito Aibo* de Sony, que ha alcanzado gran éxito como juguete-mascota, son ejemplos de ello.

Como puede verse, el uso de robots va creciendo paulatinamente tanto en el entorno industrial como en el entorno de investigación, e incluso en el del entretenimiento, aumentando el número de aplicaciones a medida que aumenta su autonomía y funcionalidad.

Dentro del área de investigación de la robótica, las Universidades se han planteado como objetivo el conseguir que los robots desempeñen tareas de modo autónomo. Para conseguirlo, los robots están dotados de *procesadores*, *actuadores* y *sensores*. Estos dispositivos se han beneficiado de los progresos tecnológicos para mejorar su calidad y prestaciones:

- Los *sensores* son dispositivos electrónicos que permiten al robot medir alguna característica de su entorno o de sí mismo, informándose de lo que ocurre a su alrededor o dentro de él. De este modo, su comportamiento podrá ser sensible a las condiciones que le rodean, aunque la información proporcionada por los sensores es limitada, ruidosa e inexacta. Ejemplos de sensores son: sensores de luz, de temperatura, ultrasónicos, infrarrojos, láser, cámaras (que nosotros utilizaremos en el proceso de localización del robot), encoders o cuenta vueltas, sensores de contacto, de presión, etc...

- Los *actuadores* son dispositivos electrónicos, mecánicos o hidraúlicos que permiten al robot ejercer alguna acción sobre el medio o, simplemente, materializar un movimiento. Los actuadores hacen posible al robot influir sobre su entorno de modo activo, modificarlo o, sencillamente, desplazarse a través de él. Ejemplos de actuadores son: motores de continua, servos, válvulas, etc... En el caso de los robots móviles, éstos tienen capacidad de movimiento que en general se consigue con ruedas que los motores hacen girar o con patas desplazadas por los motores.

- Los *procesadores* ejecutan los programas del robot. Los programas leen los datos sensoriales y gobiernan los movimientos del robot, determinando su comportamiento. Estos programas establecen el enlace entre los datos sensoriales y las consignas de los actuadores (aunque este enlace se puede construir de forma inalterable, no es lo común) determinando el comportamiento del robot. Los procesadores utilizados en robótica móvil van desde microcontroladores específicos a ordenadores personales.

En las dos últimas décadas se han desarrollado múltiples trabajos de investigación. Algunas de las áreas cubiertas por dichos trabajos son:

1. Dotación de percepción sensorial evolucionada integrada en el sistema de control. Por ejemplo, dónde situar los sensores de contacto o una cámara, afecta a la información que el robot pueda recibir.

2. Búsqueda de nuevos diseños mecánicos y materiales más rígidos y ligeros. Por ejemplo, el diseño mecánico, condiciona enormemente los movimientos posibles en un robot con patas.

3. Incorporación de técnicas de inteligencia artificial (conceptos como planificación, uso de lógica, razonamiento probabilístico, teoría de la evidencia y lógica borrosa) que aumentan la autonomía del robot.

1.2. Localización

Uno de los problemas que se han abordado recientemente en el campo de la robótica móvil es el de la localización. La localización de robots móviles autónomos consiste en determinar la posición del robot dentro de su entorno, bien desde mapas concretos y mediante la utilización de sensores que le ayudan a determinar su posición en un cierto marco de referencia o bien mediante técnicas más actuales de localización y creación de mapas simultáneamente, SLAM (Simultaneous Localization and Mapping).

El problema de localización más sencillo es la *navegación a partir de una posición inicial conocida*. En este caso, la solución consiste en estimar la posición final del robot compensando los errores incrementales de odometría acumulados por sus encoders.

Más difícil de resolver es el problema de *localización global, donde se desconoce la posición inicial*. Aquí el robot ha de manejar múltiples hipótesis para determinar su posición con lo que los errores en las estimaciones son mayores que en el caso anterior.

Finalmente, existe un último problema de *localización de más de un robot* en el que un grupo de robots intenta localizarse y que es particularmente interesante si los robots pueden detectarse entre sí ya que esto añade dependencias estadísticas en las estimaciones locales de cada robot [Fox00].

1.2.1. Mapas

La característica fundamental de los robot móviles es precisamente esa, que se mueven. Para dotar a este movimiento de autonomía y que el robot sea capaz de tomar decisiones le interesa almacenar información del entorno. Los robots modelan su entorno mediante lo que se conoce como mapas, por ejemplo mapas de ocupación. Los mapas de ocupación permiten a los robots móviles tomar decisiones autónomas de movimiento: hacia dónde moverse,

cuánto girar, a qué velocidad deben avanzar, etc... Los robots necesitan estar localizados para construir y/o utilizar estos mapas.

Hay muchos tipos de mapas: En los mapas *globales* se representa toda el área de movimiento del robot mientras que en los *locales* sólo se representa el entorno próximo a la ubicación actual. Los mapas *topológicos* son grafos cíclicos con nodos (lugares relevantes) y arcos (pasajes entre nodos) que permiten planificar trayectorias pero no inferir distancias precisas. Los mapas *métricos* utilizan un sistema de coordenadas que permiten inferir distancias y ángulos. En los mapas *de elementos geométricos* se definen primitivas de percepción de esquinas, segmentos, etc... que se relacionan con las observaciones sensoriales y se realiza una estimación continua de la posición de esos elementos. En los mapas *de rejilla* se particiona el espacio en un mallado regular de celdillas; las observaciones sensoriales se relacionan con las celdillas y el estado de cada celdilla se estima continuamente.

1.2.2. Sensores de localización

La percepción es un elemento crítico en la resolución de los problemas de localización. A continuación se hablará de los sensores específicos utilizados para estimar la posición del robot. Los principales son los encoders.

Figura 4: Encoders, permiten estimar la posición del robot

Los *encoders* (Figura 4) nos devuelven pulsos o posición de las ruedas que nos permiten estimar la posición del robot calculando la diferencia entre

dos lecturas. Existen errores por desplazamiento, holguras, falta de precisión, etc... Los errores odométricos de los encoders pueden ser de dos tipos:

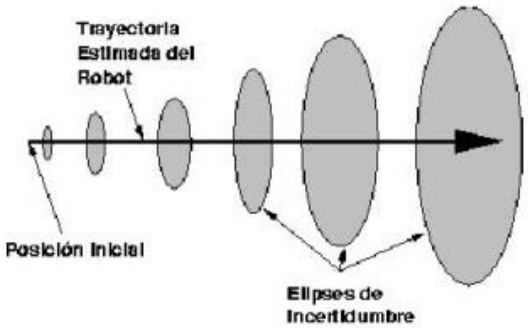

Figura 5: Elipse de error estimado (Tonouchi 1994)

1. *Errores sistemáticos* que dependen de las características del robot y sus sensores y que son importantes puesto que se acumulan (Figura 5). Estos errores son debidos a diferentes diámetros de ruedas, falta de alineamiento de las mismas, resolución limitada de los encoders y velocidad limitada de muestreo de los encoders. Los errores sistemáticos pueden corregirse con ruedas auxiliares, encoders adicionales o mediante calibración sistemática.

2. *Errores no sistemáticos* que no pueden predecirse y que son debidos a patinaje de las ruedas, rugosidad del suelo y objetos en el suelo. Los errores no sistemáticos pueden corregirse utilizando referencias mutuas (dos robots, uno parado), corrección interna (dos robots en movimiento) y navegación inercial (medir la aceleración en los tres ejes e integrar).

Corregir los errores odométricos (fundamentalmente los sistemáticos) es importante ya que con ello se hace más precisa la información de localización obtenida.

Un ejemplo de calibración para la correción de errores odométricos sistemáticos es *el test del cuadrado unidireccional* (Figura 6). Se posiciona el robot en (x_0, y_0, θ_0), junto a una pared y se le hace recorrer un cuadrado. Al final se miden las distancias desde tres puntos del robot a las paredes y se obtiene un error E_x, E_y, E_θ; el error puede deberse a diferentes diámetros (curva) y a errores de giro.

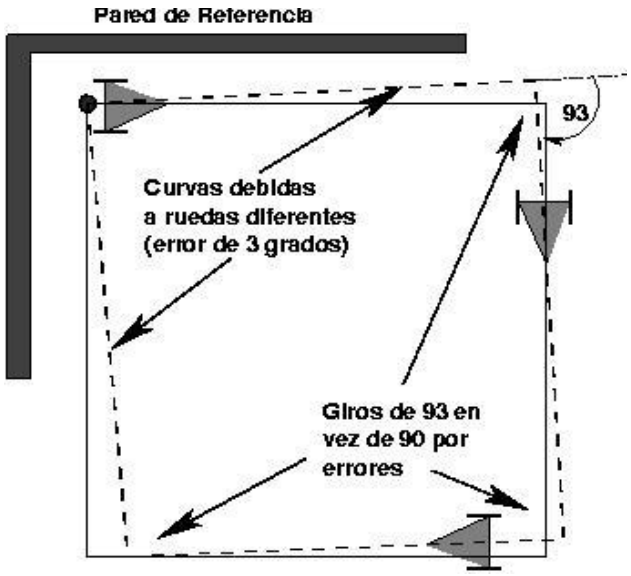

Figura 6: Test del cuadrado unidireccional

Una mejora al cuadrado unidireccional es el *test del cuadrado bidireccional* que evita que ciertos errores se compensen. En este caso hay que recorrer el cuadrado en los dos sentidos y conviene repetirlo varias veces.

Figura 7: El sensor GPS

Otro tipo de sensor utilizado para resolver el problema de localización es el GPS (Global Positioning System, Figura 7), útil para exteriores. Son

necesarios al menos cuatro satélites para obtener las coordenadas de posición y el tiempo. La configuración de estos sistemas está basada en un receptor a bordo del vehículo y un conjunto de estaciones transmisoras de ondas de radio operando desde satélites [González96]. El receptor permite calcular por triangulación la altitud, latitud y altura. Su principal ventaja es que proporciona la localización absoluta en un área suficientemente grande y sin requerir estructura alguna del entorno. Las fuentes de error del GPS son distintas a las de los encoders, los errores no son acumulativos y pueden deberse a errores del reloj (máximo 1m.), errores en la órbita (máximo 1m.), errores en la modelización de la propagación en la troposfera y la ionosfera (1m. y 10m., respectivamente), errores debidos a rebotes de la señal (0.5m.) y errores de usuario en el receptor o en la configuración del mismo (los más usuales y variables, de 1m. a cientos de km.).

1.2.3. Técnicas de Localización

Existen múltiples técnicas de localización que proporcionan otra aproximación al problema de localización sin utilizar sensores específicos de posición.

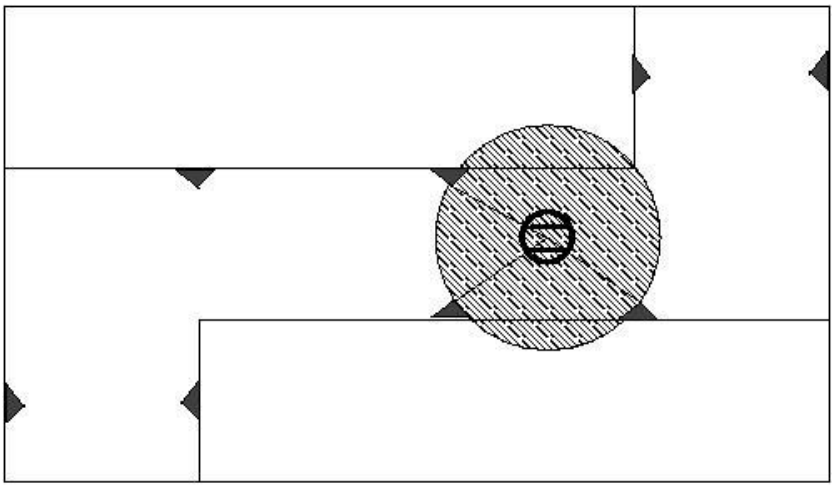

Figura 8: Ejemplo de localización con balizas (los triángulos representan las balizas)

Un ejemplo de estas técnicas de localización es la *localización con balizas* (Figura 8). Esta técnica permite localizar al robot geométricamente en un entorno restringido mediante el emplazamiento en el escenario de navegación de un determinado número de balizas (pueden ser códigos de barras, infrarrojos...) de posición conocida. Para calcular (x, y, θ) se utiliza la triangulación

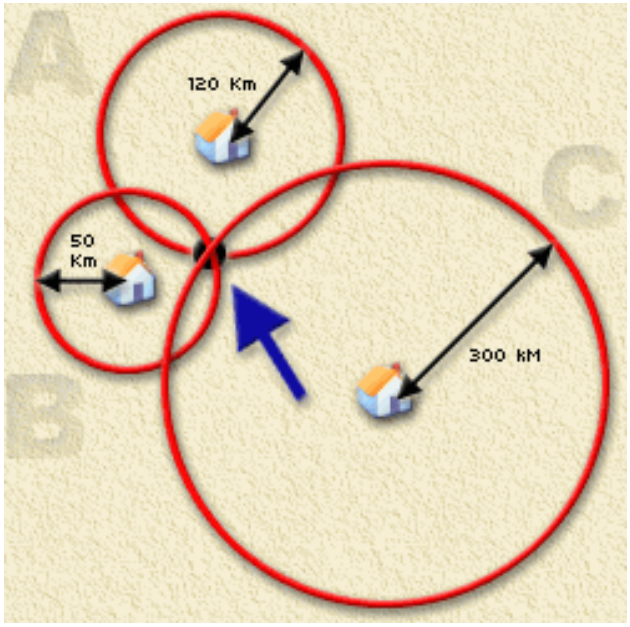

Figura 9: Localización por trilateración

y la trilateración. Mediante triangulación, (x, y, θ) se calculan basándose en el ángulo con que se "ven" las balizas; mediante trilateración (Figura 9), (x, y, θ) se calculan basándose en la distancia a las balizas.

Otra técnica es la de *scan matching*. Esta técnica consiste en comparar las lecturas de los sensores con un mapa global del entorno pero sin utilizar balizas. Se realizan una serie de lecturas (por ejemplo con un láser) tras realizar movimientos de traslación y rotación. Es necesario tener una estimación de la posición inicial del robot que se representa como una distribución gaussiana y se actualiza con las lecturas de los sensores. La ventaja es que se puede hacer una estimación muy precisa de la posición del robot de forma eficiente; su punto débil es que no es capaz de recuperarse de fallos considerables en la estimación de su posición.

Los filtros de Kalman son otra técnica utilizada para solucionar el problema de localización. Se trata de un estimador recursivo que fusiona información parcial e indirecta sobre localización permitiendo obtener una estimación de mínima varianza del estado del robot en cada instante de tiempo. Su principal limitación es que se trata de una técnica unimodal y exclusivamente gaussiana [Isard98]. Los entornos dinámicos y el ruido en las lecturas de los sensores tampoco son bien soportados por esta técnica.

La metodología conocida como *localización probabilística* es muy adecuada para la localización [Thrun00a] en mapas de interiores, ya que incorpora

incertidumbre en acciones y observaciones lo cual concuerda con la incertidumbre (errores odométricos) de las estimaciones de los sensores. La localización probabilística consiste en el proceso de determinar la probabilidad de que el robot se encuentre en una determinada posición dada una historia de lecturas de la información proporcionada por sus sensores y de una serie de acciones ejecutadas por el robot. A cada posible pose (posición y orientación) se le asocia una probabilidad, reflejando la verosimilitud de que sea la pose actual del robot. Esa estimación se actualiza con la incorporación de nuevas observaciones y nuevos movimientos comandados al robot. Esto nos permite localizar al robot incluso si se desconoce su posición inicial, tiene en cuenta los errores odométricos debidos a los sensores y permite representar situaciones ambiguas y resolverlas posteriormente. Su eficiencia dependerá del tamaño del mapa en el que se realiza la localización.

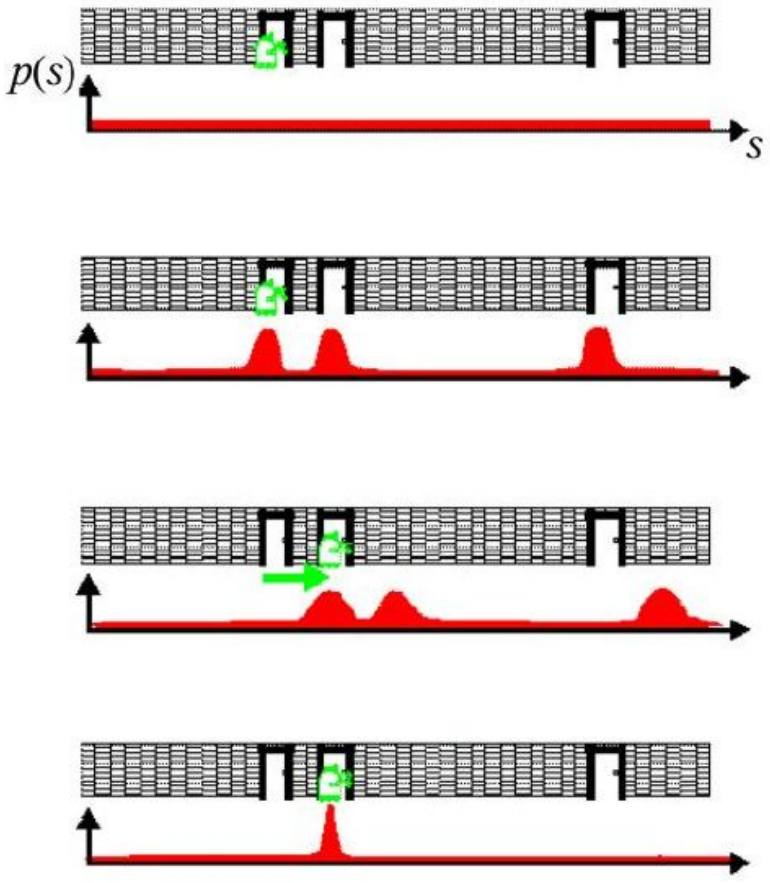

Figura 10: Localización probabilística

La idea intuitiva puede apreciarse en la Figura 10 [Fox99b]. En este caso,

se asume que se sitúa al robot en un espacio unidimensional en el que el sólo puede desplazarse horizontalmente y que desconoce su posición de partida; se representa este estado de incertidumbre como una distribución uniforme sobre todas las posibles posiciones. En el siguiente paso, suponemos que el robot detecta (haciendo uso de sus sensores) que se encuentra frente a una puerta; esto se refleja aumentando la verosimilitud en las zonas donde están las puertas y disminuyéndola en el resto de las posiciones. En los espacios próximos a las puertas, la probabilidad no es exactamente 0 sino un poco mayor para tener en cuenta el ruido incoporado por los sensores ya que las lecturas sensoriales pueden ser erróneas. La información disponible hasta el momento es insuficiente para determinar la posición del robot. Si ahora desplazamos el robot, la distribución se desplaza con el mismo como se refleja en el tercer paso. En este desplazamiento, se suavizan los valores para incorporar el ruido de actuación. En el último paso, el robot ha detectado otra vez que se encuentra frente a una puerta, haciendo uso nuevamente de sus sensores y esta evidencia multiplicada por la acumulada en el tercer paso, nos lleva a la conclusión de que el robot se encuentra muy probablemente situado frente a la segunda puerta. Nosotros aplicaremos este método para localizar al robot futbolista en el terreno de juego.

Un inconveniente de la localización probabilística es que se almacena y actualiza la distribución de probabilidades para todas las posibles posiciones para lo que se requiere mucho tiempo de cómputo. Esto hace que el proceso sea lento y no escalable a grandes entornos. Para paliar este problema, se utilizan *técnicas de muestreo*. Las técnicas de localización en mapas de ocupación han seguido una evolución significativa en los últimos años, siendo los algoritmos basados en filtros de partículas los que en la actualidad ofrecen la solución más efectiva a todos los problemas de localización. La idea de los algoritmos de filtros de partículas es representar la verosimilitud mediante un conjunto de muestras con distribución acorde a la verosimilitud $Bel(x) = \{x^{(i)}, p^{(i)}\}, i = 1, ..., m$ donde $x^{(i)}$ es una muestra y $p^{(i)}$ su probabilidad. En vez de calcular y almacenar todas las localizaciones posibles, se mantiene sólo un pequeño subconjunto representativo de muestras, lo que permite agilizar los cálculos de localización probabilística. Estos algoritmos también conocidos como *técnicas de MonteCarlo (MCL) o algortimos de condensación*, son un conjunto de técnicas de muestreo que permiten muestrear cualquier distribución. Nosotros aplicaremos este método para muestrear la verosimilitud de las localizaciones posibles del robot.

1.3. Robots con visión local en la Robocup

Una vez planteado el problema teórico de localización del que trata este proyecto, describiremos ahora el escenario concreto al cual está orientado, la Robocup. La Robocup [1] es una competición mundial en la que equipos compuestos por robots juegan al fútbol. Esta competición se inició en 1996 con la intención de incentivar el interés por la ciencia y la tecnología ya que aporta un escenario de investigación tanto para la robótica como para la inteligencia artificial.

El objetivo de la Robocup para el año 2050 es programar un equipo de robots humanoides autónomos que puedan ganar al equipo humano ganador del mundial de fútbol. Gracias a iniciativas de este tipo se fomenta la investigación de nuevas tecnologías que, para conseguir una serie de objetivos secundarios que se van marcando (y que están más al alcance de la tecnología actual), generan una serie de desarrollos que pueden aplicarse luego a otros campos. Basándose en el éxito conseguido por la Robocup de fútbol, se ha iniciado el proyecto *Robocup Rescue* para formentar la investigación y desarrollo en el campo de los rescates en desastres. El problema genérico que se pretende resolver en este caso es conseguir la colaboración entre agentes inteligentes en tiempo real en un entorno dinámico y competitivo.

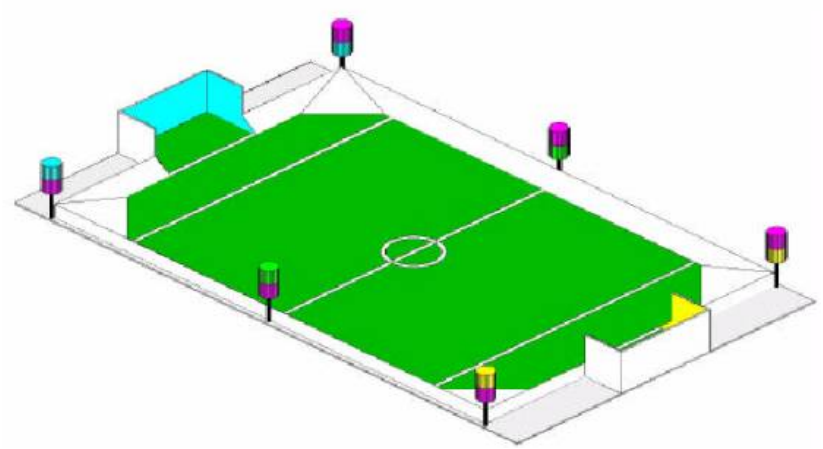

Figura 11: Campo de juego de la Robocup

La Robocup de fútbol se celebra anualmente y enfrenta a equipos de diferentes países y diferentes categorías: ligas de simulación, de robots pequeños, de robots medianos, de perritos Sony y de robots humanoides.

[1] http://www.robocup.org/

En este escenario, conocer la posición del robot en el terreno de juego es determinante para el comportamiento de cada jugador. Para facilitar la localización, se pueden situar balizas de colores en el campo de acuerdo con las normas de la Robocup (Figura 11). También puede hacerse uso de los cuenta vueltas de las ruedas y mover el robot a voluntad para recoger más datos.

1.3.1. El Robot EyeBot

Como ya se ha mencionado anteriormente, este trabajo fin de carrera está orientado a conseguir localizar un robot móvil en función de lo que observa con su cámara. El desarrollo realizado se puede aplicar a la Robocup en las categorías de robots pequeños, medianos y perritos Aibo de Sony.

Los robots EyeBot [2] son unos mini-robots móviles especialmente diseñados para la Robocup, en la categoría de robots móviles pequeños. En esta categoría, la localización se realiza normalmente haciendo uso de una cámara cenital. Dicha cámara capta una imagen que es analizada en un PC que comunica la posición a los jugadores a través de la radio. En este trabajo, por el contrario, se pretende que el robot sea capaz de orientarse autónomamente haciendo uso de sus recursos locales (sus motores, su cámara y su procesador).

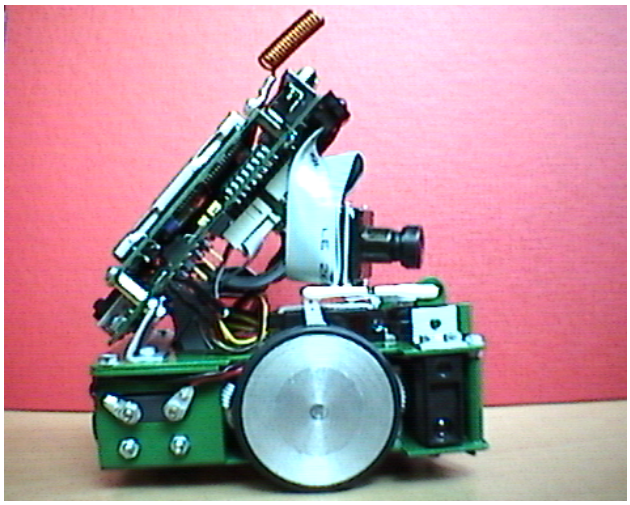

Figura 12: EyeBot

[2] http://www.ee.uwa.edu.au/∼braunl/eyebot

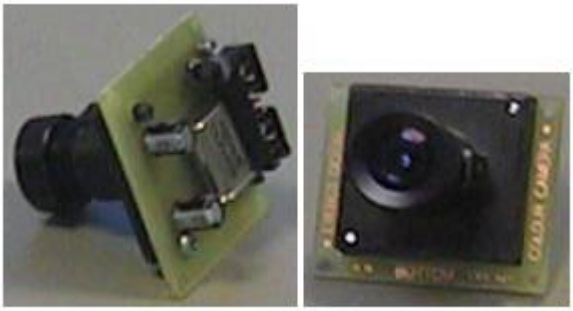

Figura 13: Vistas posterior y frontal de la cámara (EyeBot)

Los robots EyeBot tienen varios sensores: una cámara digital, tres sensores de infrarrojos y un par de *encoders* o cuenta vueltas (uno para cada motor); y varios actuadores: dos motores para mover las ruedas y dos servos, uno para el pateador y otro para la cámara. Los encoders devuelven un número de pulsos que indican el desplazamiento que ha realizado cada rueda con una resolución de un cuarto de vuelta. La cámara trabaja con 24 bits en color o en escala de grises, proporcionando una resolución de 80x60 pixels. Esta resolución es suficiente para la mayoría de las tareas que realiza el robot y permite un procesamiento rápido de la imagen. Lleva un servo asociado que le permite un giro horizontal o vertical, dependiendo del modelo, con el fin de facilitar la visión de los objetos deseados.

Como elemento de proceso incorpora un microprocesador Motorola 68332 a 35Mhz por lo que es conveniente la máxima optimización de los procesos a ejecutar en el mismo. Esta pequeña capacidad de proceso supone una limitación muy fuerte para ejecutar algoritmos de localización on-line, de ahí que se exploren técnicas de muestreo que aceleren dichos algortimos.

Para la localización del robot en este trabajo, se simularán *acciones* a partir de la información proporcionada por los encoders y *observaciones* a partir de la información proporcionada por la cámara digital.

1.3.2. El Robot AiBo

El perrito Aibo de Sony [3] es un robot de tamaño pequeño que también participa en la Robocup. Es otro robot en el que está previsto utilizar las técnicas de localización probabilística desarrolladas en este trabajo fin de carrera. Como en el caso del Eyebot, se pretende que el robot sea capaz de orientarse autónomamente haciendo uso de sus recursos locales.

Figura 14: Perrito Aibo de Sony

El robot Aibo está dotado de una cámara, un sensor de infrarrojos y cuatro patas con tres grados de libertad. Para comunicarse con el ordenador central utiliza conexión inalámbrica. La cámara trabaja en color a 20 fps. Las patas son más complicadas que las ruedas por lo que los errores acumulados por los encoders son mayores y la localización utilizando sólamente la información que éstos proporcionan no es posible. [Buschka00] describe una técnica de localización que utiliza lógica borrosa para la localización del robot Aibo en el campo de juego de la Robocup. En nuestro caso, no se utilizará lógica borrosa sino técnicas de localización probabilística que hacen uso de acciones e imágenes simuladas, información con la que se cuenta en esta plataforma.

[3] http://www.eu.aibo.com

1.4. Estructura de la memoria

El resto de la memoria está estructurada como se indica a continuación:

En el capítulo 2 se definirán los objetivos y requisitos a cubrir por este trabajo fin de carrera.

En el capítulo 3 presentamos los métodos probabilísticos empleados para solucionar el problema de localización y más en concreto la regla de Bayes utilizada en nuestro caso, detallando cómo se ha implementado y qué experimentos se han realizado para probar su funcionamiento.

En el capítulo 4, con estructura similar a la del capítulo 3, introducimos las técnicas de muestreo y el algoritmo "Condensation". El objetivo es agilizar los cómputos para que el proceso de localización se realice de manera eficiente.

El capítulo 5 resume las conclusiones obtenidas durante la realización de este proyecto. Se describirán los principales problemas identificados, la eficiencia y la precisión conseguidas con los dos algoritmos y se realizará una evaluación de las mismas.

Finalmente, se incluyen bibliografía y listados de los programas codificados en la fase de implementación.

2. Objetivos

Una de las líneas de trabajo del grupo de Robótica de la Universidad Rey Juan Carlos [4], en la que se encuadra este trabajo fin de carrera, consiste en programar un equipo de robots EyeBot y/o Aibo para competir en la Robocup. El objetivo de este trabajo fin de carrera es desarrollar algoritmos para conseguir localizar un robot móvil (que sepa estimar en qué posición está y con qué orientación) en función de lo que observa con su cámara y evaluar su rendimiento. Para localizarse el robot cuenta con los siguientes recursos: un mapa del entorno, que es el campo de juego de la Robocup; su cámara, que le proporcionará imágenes de ese entorno; y sus motores, que le permitirán desplazarse y girar.

Mediante la utilización de métodos de percepción probabilística, con y sin muestreo, se estimará la posición del robot en el campo de juego de la RoboCup (es decir, en el mapa que simula el mismo) en función de las observaciones simuladas que "ve" con la cámara local. Como primera aproximación se supondrá que el terreno de juego no tiene más jugadores u obstáculos.

Aunque en este trabajo no exigiremos implementación en el robot, sí se tendrán en cuenta las características del mismo para facilitar una futura implementación en un trabajo posterior.

El objetivo general se articula en tres puntos concretos: creación de un entorno simulado, implementación probabilística sin muestreo e implementación probabilística con muestreo.

1. En primer lugar se construirá un *entorno simulado*, fácilmente modificable, similar al escenario de la Robocup y en el que se probarán los algoritmos. Este entorno proporciona imágenes simuladas y permite simular el movimiento comandado. Se utilizará un mapa que simula el campo de juego de la Robocup y sobre el que se simularán dichas observaciones y actuaciones. El robot podrá trasladarse y rotar, y además se aplicará ruido a estas actuaciones reflejando el que existe en las actuaciones en el mundo real. Las observaciones serán imágenes simuladas. No se utilizan sensores reales, pero se incluirá ruido sensorial por lo que son una buena aproximación a la realidad.

2. Se realizará un estudio e implementación de una técnica de localización *probabilística sin muestreo*. En ella se explorará el funcionamiento frente a diferentes mapas, para observar los diferentes comportamientos del robot según el número de balizas, simetrías, etc... Se realizará un

[4] http://gsyc.escet.urjc.es/robotica/

estudio del modelo sensorial y de actuación, reglas de actualización / combinación de evidencias, cómo se comporta el robot en función del entorno, con qué precisión consigue localizarse y a qué coste.

3. Dado que el EyeBot sólo cuenta con un procesador a 35Mhz y la primera implementación parece ser demasiado costosa en él, se realizará una segunda implementación en la que se emplearán técnicas de Monte Carlo, *implementación probabilística con muestreo*, que agilicen los cálculos a realizar en el proceso de localización. Se realizará un estudio similar al explicado para la implementación probabilística sin muestreo y se compararán los resultados.

Un requisito de partida en este proyecto es su realización utilizando programación en ANSI-C y en un PC con sistema operativo GNU/Linux. El código es software libre, puede utilizarse y/o modificarse de acuerdo con la licencia GPL (General Public License) publicada por la Fundación de Software Libre.

3. Localización probabilística sin muestreo

La localización de robots autónomos es uno de los problemas principales de la robótica móvil. Se han desarrollado múltiples metodologías, presentadas en el capítulo 1, para resolver este problema. Una de estas técnicas es la localización probabilística. Como ya hemos visto anteriormente, ésta consiste en el proceso de determinar la probabilidad de que el robot se encuentre en una determinada posición dada una historia de lecturas de la información proporcionada por sus sensores y de una serie de movimientos realizados por el robot. Esto nos permite localizar al robot incluso si se desconoce su posición inicial, tiene en cuenta los errores odométricos debidos a los sensores y permite representar y resolver situaciones ambiguas. Su eficiencia dependerá del tamaño del mapa en el que se realiza la localización.

En este capítulo, se presentarán los fundamentos teóricos en los que se basa esta metodología, el entorno de simulación en el que se ha implementado, cómo se ha realizado dicha implementación, los experimentos realizados para estudiar su comportamiento y el rendimiento en el entorno simulado.

3.1. Fundamentos teóricos

Los algoritmos probabilísticos son los que han proporcionado mejores soluciones a problemas de localización global, aquella en la que se desconoce la posición inicial. También se han empleado con éxito en la construcción de mapas fiables en entornos muy amplios.

La idea fundamental de la localización probabilística es representar la información en forma de densidades de probabilidad de localización. A cada posible *pose* (posición y orientación) se le asocia una probabilidad, reflejando la verosimilitud de que sea la *pose* actual del robot. Esa estimación se actualiza con la incorporación de nuevas observaciones y nuevos movimientos comandados al robot.

Planteado de modo probabilístico, el problema de la localización consiste en estimar la distribución de la densidad de probabilidades de las posibles posiciones del robot condicionada a la información disponible: $P(position/obs(t))$. Como ya se explicó en el capítulo 1 (Figura 10), si el robot está observando una puerta con los sensores, es muy probable que esté situado frente a una de ellas. Por tanto, las posiciones situadas frente a las puertas en el mapa acumulan mayor verosimilitud que las intermedias del pasillo.

Como información de entrada, nuestro algoritmo localizador dispone no sólo de lecturas sensoriales, sino también de actuaciones que son los movi-

mientos ordenados a los motores. Además, se cuenta con un mapa conocido del entorno, similar al terreno de juego de la Robocup, por el que se desplaza y gira el robot. Una ventaja de utilizar el entorno probabilístico es que no requiere el uso de sensores específicos de localización (encoders, GPS...) sino que es capaz de utilizar sensores no específicos (visión, láser...) siempre y cuando las observaciones dependan en cierta medida de la posición del robot. Los sensores no específicos dan información indirecta y parcial sobre la localización actual del robot, en nuestro caso, se utilizarán las imágenes proporcionadas por la cámara local. En cuanto a las actuaciones, se reflejarán mediante un desplazamiento bidimensional de las evidencias acumuladas; el robot podrá desplazarse en x e y y rotar en θ.

En general, el problema de localización se divide en dos etapas que se ejecutan constantemente:

1. Se captura toda la información que proporciona una nueva lectura del sensor siguiendo un modelo sensorial y se incorpora el efecto de cierta orden al actuador siguiendo un modelo de actuación.

2. Con la información sensorial y de actuación se actualizan la evidencias acumuladas, materializando la fusión con la información anterior. A medida que el robot recibe nuevas observaciones sensoriales y nuevas actuaciones, su información se va acumulando, actualizando las probabilidades y haciéndolas evolucionar.

La *regla de Bayes* permite la fusión de información para la acumulación de evidencias. Contamos por tanto con un modelo sensorial ($p(o_t|x_t)$) y con un modelo de movimiento ($p(x_t|x_{t-1}, a_{t-1})$). Se asume un entorno *Markoviano*, es decir, la información pasada y futura es independiente si se conoce la posición actual. La idea clave es representar la información en forma de densidades de probabilidad para calcular la densidad de probabilidad posterior o verosimilitud $Bel(x_t) = P(position/obs(t), data(t-1))$ donde x_t indica la posición actual, obs(t) se corresponde con la observación actual y data(t-1) corresponde al conjunto de observaciones acumuladas hasta el instante t-1. [Thrun01] proporciona el detalle de la formulación matemática. Para su implemetación es necesario conocer tres distribuciones: *la verosimilitud inicial* (distribución uniforme si se desconoce la posición inicial), *el modelo de actuación* (probabilidad de que el robot se encuentre en una determinada posición tras un determinado movimiento, para lo que es necesario tener en cuenta los errores odométricos) y *el modelo sensorial* (probabilidad de que el robot se encuentre en una determinada posición dada una determinada lectura de sus sensores). La probabilidad acumulada puede seguir cualquier tipo de distribución espacial, incluso multimodal. Es importante resaltar, que los filtros

de Kalman no permiten este planteamiento multimodal y que, por lo tanto, no se utilizan en este tipo de escenarios.

En el apartado *implementación* de este capítulo se indicará el uso que se hace en este trabajo de la regla de Bayes para incorporar la información porporcionada por sensores y actuadores.

3.2. Entorno de simulación

Una vez introducida la parte teórica, en esta sección presentamos en detalle los elementos que intervienen en el entorno concreto de simulación que utilizaremos para estudiar los algortimos desarrollados. En la Figura 15 se muestra el esquema del entorno de simulación utilizado para la localización del robot.

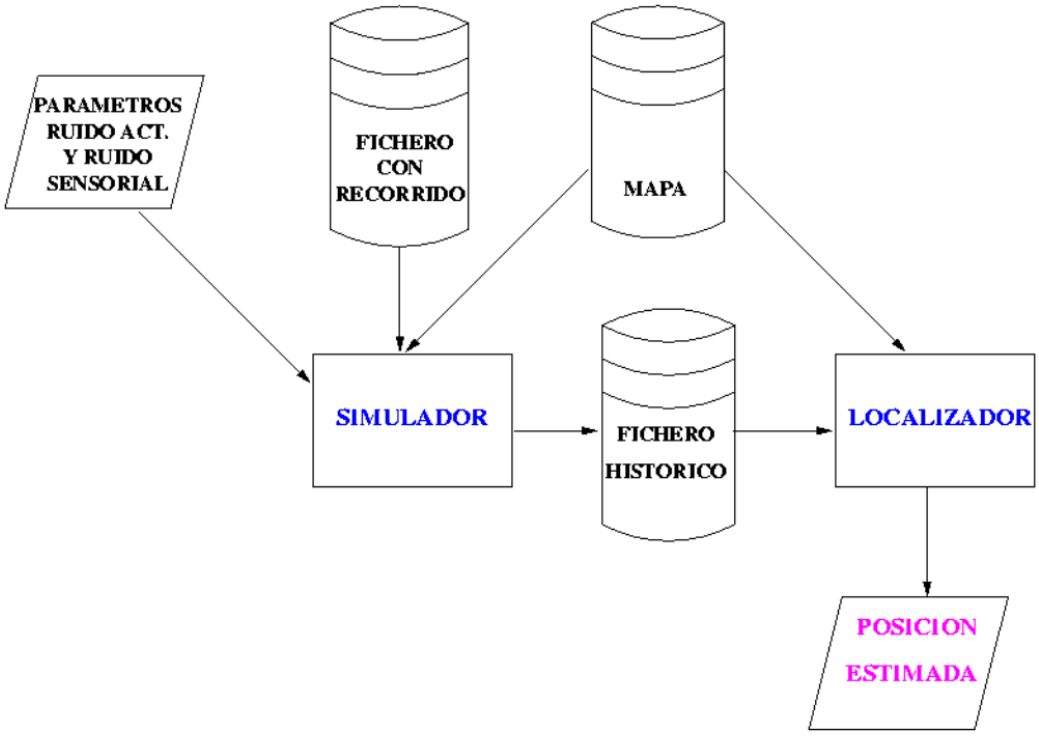

Figura 15: Elementos de nuestro proceso de localización

El Simulador proporcionará la información necesaria al algoritmo Localizador para un procesamiento *off-line*. Este proceso nos permite controlar los recorridos a voluntad y facilitar la comparativa entre la técnica probabilística sin muestreo y con muestreo, pero podría realizarse sin problema *on-line*.

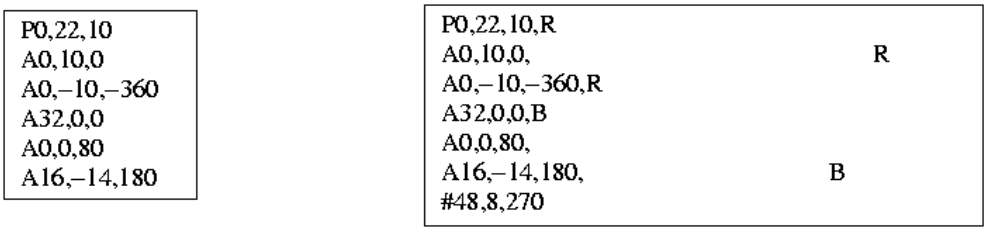

Figura 16: Ejemplos de ficheros utilizados por los algoritmos de localización: Recorrido (izda) e Histórico (dcha)

Los parámetros de entrada y los ficheros de entrada/salida al Simulador, que veremos posteriormente con más detalle, son los siguientes:

- *Mapa*: Como escenario simulado, se ha elegido un mapa conocido del campo de juego de la Robocup donde se simulan las balizas de colores (Figura 17).
- *Parámetros de ruido sensorial y ruido de actuación*: Para proporcionar un entorno lo más parecido al real, se dispone de dos parámetros de ruido sensorial y dos de ruido de actuación a partir de los cuales se modifican las imágenes proporcionadas por la cámara y las posiciones resultantes de aplicar los movimientos (desplazamientos en x, y y/o θ) respectivamente. Estos parámetros se describira con mas detalle en los apartados de *Modelo Sensorial y simulación de observaciones* y *Modelo de actuación y simulación de movimientos*.
- *Fichero con Recorrido*: Fichero de entrada al Simulador que contiene la posición inicial del robot y la secuencia de acciones a realizar. Un ejemplo se ilustra en la imagen izquierda de la Figura 16.
- *Fichero Histórico*: Fichero de salida del Simulador y de entrada al Localizador, su contenido es el del Fichero con Recorrido al que se han añadido las imágenes unidimensionales que simulan las capturadas por la cámara local (Figura 20) desde la posiciones recorridas. Un ejemplo se ilustra en la imagen derecha de la Figura 16.

El algoritmo Simulador lee el *Fichero con Recorrido* y comprueba sobre el *Mapa* de entrada qué balizas capta la cámara y en qué pixel. Este proceso se realiza primero para la posición inicial y después para las posteriores (resultantes de aplicar las acciones indicadas con el porcentaje de ruido de actuación solicitado). A continuación se aplica el ruido sensorial solicitado a la imagen calculada y se crean los registros en el *Fichero Histórico* de entrada al algoritmo Localizador.

3.2.1. Mapa

Para llevar a cabo nuestra simulación, se proporciona un sencillo mapa de entrada definido arbitrariamente como un array de 33 filas por 65 columnas en el que se pueden colocar un número variable de balizas para observar los diferentes comportamientos del robot según el número de balizas, simetrías, etc... Este mapa es utilizado por los algoritmos de simulación y localización.

```
L0,32,64,32
L0,0,64,0
L0,32,0,0
L32,32,32,0
B0,32,A
B32,32,B
B64,32,C
B,0,17,D
B64,17,E
B0,16,D
B64,16,E
B0,15,D
B64,15,E
B0,0,F
B32,0,G
B64,0,H
```

Cuadro 1: Ejemplo de fichero para la construcción de un mapa

El mapa de la Figura 17 se construye a partir del fichero que aparece en el Cuadro 1. Los mapas que hemos utilizado en los ejemplos se asemejan al campo de fútbol de la Robocup y se puede modificar fácilmente su tamaño, forma y el número y la posición de las balizas sin más que modificar el fichero. Las coordenadas *(x,y)* de inicio y fin de las líneas del campo van precedidas de la letra "L" (Línea) y se representarán como W = White (Figura 17). Las coordenadas de las balizas (12 en el ejemplo) van precedidas de la letra "B" (Baliza) y seguidas de cualquier letra mayúscula que identifica un color.

3.2.2. Fichero con Recorrido y Fichero Histórico

El formato del *Fichero con Recorrido* de entrada al Simulador, aparece ilustrado en la imagen izquierda de la Figura 16: el primer registro precedido

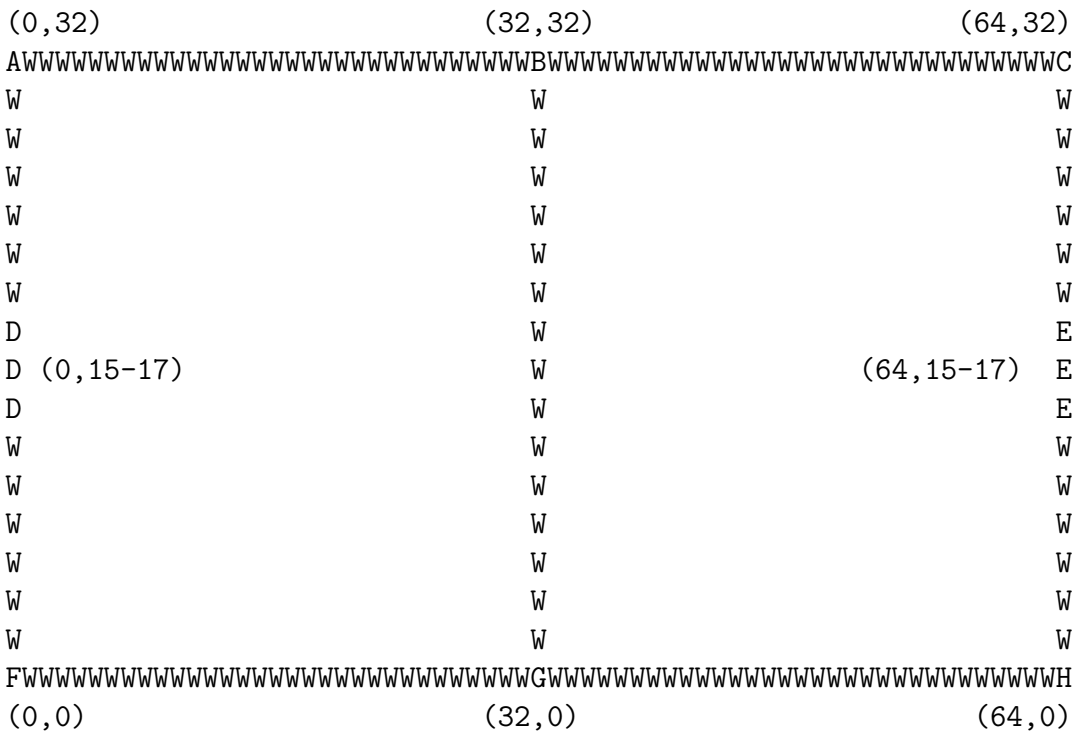

Figura 17: Ejemplo de Mapa con 12 balizas que simula el campo de juego del la Robocup

por la letra "P" (Posición) indica la posición inicial del robot (x, y, θ); los registros sucesivos, precedidos por la letra "A" (Actuación) indican las acciones a realizar, que son desplazamientos conseguidos enviando comandos a los motores del robot, en el mismo formato.

El formato del *Fichero Histórico* de salida del Simulador y de entrada al Localizador, aparece ilustrado en la imagen de la derecha de la Figura 16. El formato de los registros es similar al del *Fichero con Recorrido* a los que se añaden 80 caracteres que corresponden a las imágenes simuladas desde la posición inicial y sucesivas. La posición inicial se mantiene a modo informativo sólamente, se ignorará en el proceso ya que se pretende realizar una localización global. Como último registro y precedido del carácter "#" se añade al posición final también a modo informativo para compararla con la calculada por el Localizador.

3.2.3. Modelo sensorial y simulación de las observaciones

La cámara simulada se ha definido con campo visual de 45 ° y alcance de 25 centímetros (frente a los $33x65$ del tamaño del mapa), parámetros fácilmente modificables. Las imágenes proporcionadas por dicha cámara son unidimensionales de 80 pixels. Este modelo simplificado se ha elegido por ser suficiente para capturar toda la información necesaria ya que el robot se mueve en un plano. En la Figura 18 se representa el alcance de la cámara superpuesto al terreno de juego discretizado.

Figura 18: Campo visual de la cámara

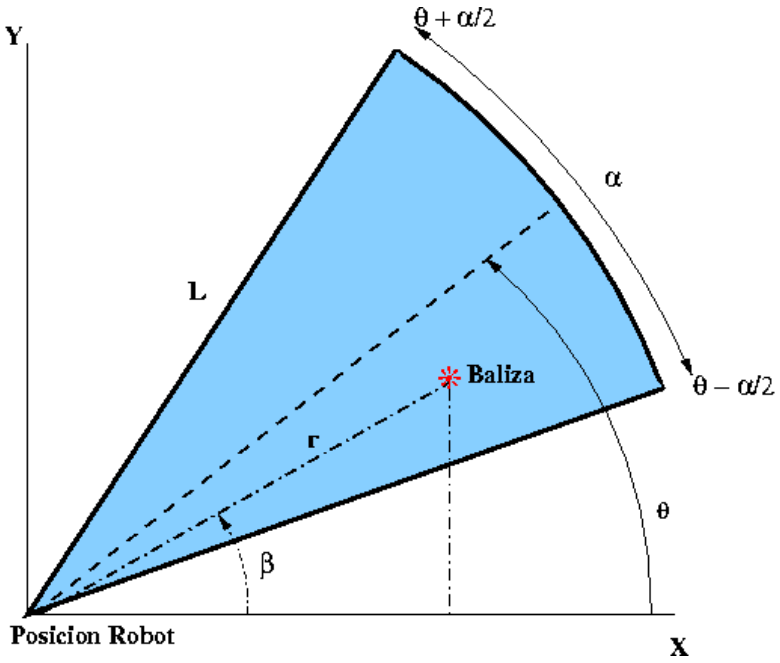

Figura 19: Detección de balizas

En nuestra simulación, el robot utilizará la cámara para detectar balizas de colores que le ayudarán a localizarse. Este proceso es fácilmente portable a la cámara del robot EyeBot que puede emplear filtros de color para identificar

las balizas. Además, la resolución proporcionada por la cámara del EyeBot es de $60x80$ con lo cual las dimensiones de las imágenes utilizadas también tienen una estrecha relación con la cámara real.

La Figura 19 indica cómo se realizan los cálculos para verificar si una *Baliza* se encuentra dentro del campo visual de la cámara. Esto nos permite generar la imagen simulada desde un punto determinado del campo de juego. El campo visual se define como un ángulo α (que en nuestro modelo es de 45 °) y un *Alcance L*. Sea P la *Posición del Robot* y b la posición de la *Baliza*. En primer lugar trasladamos P a $(0, 0)$ de tal forma que:

$$r = P - b = \sqrt{(b_y - P_y)^2 + (b_x - P_x)^2} \quad (1)$$

$$\beta = arctg\frac{b_y - P_y}{b_x - P_x} \quad (2)$$

La *Baliza b* se encuentra dentro del campo visual de la cámara si $r <$ L y $\beta \in [\theta$ - $\frac{\alpha}{2}$, $\theta + \frac{\alpha}{2}]$.

Si la baliza se encuentra dentro del campo visual de la cámara, el pixel de la imagen en el que se captura dicha baliza se calcula como:

$$pixel = 80 * \frac{((\theta + \alpha/2) - \beta)}{\alpha} \text{ para } \beta \leq (\theta + (\alpha/2)) \quad (3)$$

$$pixel = 80 - \frac{80 * (\beta - (\theta - \alpha/2))}{\alpha} \text{ para } \beta > (\theta + (\alpha/2)) \quad (4)$$

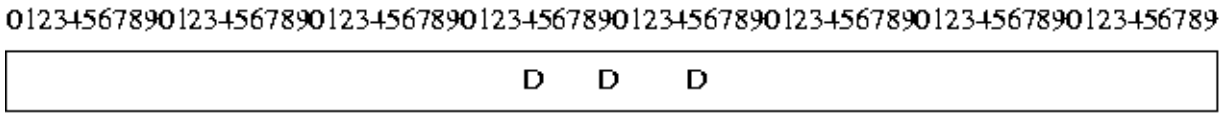

Figura 20: Ejemplo de imagen capturada por la cámara en la posición $(19, 16, 180)$

En la Figura 20 se representa un ejemplo de imagen de 80 pixels capturada por la cámara simulada en el mapa de la Figura 17 cuando el robot se encuentra en la posición de coordenadas $(x, y, \theta) = (19, 16, 180)$, es decir, situado en el mapa con coordenadas $(19, 16)$ y con la cámara orientada a 180 °. Desde esta posición, la cámara "ve" las 3 balizas que representan la portería del lado izquierdo del campo de "color" D en los pixels 34, 39 y 45.

Para proporcionar un entorno lo más parecido al *real*, se simula ruido sensorial que modifica la imagen *ideal*. Por un lado, puede eliminarse alguna

de las balizas existentes en la imagen *ideal* o incorporarse una nueva baliza (ruido de mutación que introduce falsos positivos o falsos negativos); por otro lado, las balizas de la imagen pueden sufrir un desplazamiento. Estas modificaciones son aleatorias y acordes a ciertas probabilidades que se le pasan al programa simulador como parámetros ($P_{mut.}$ y $P_{desp.}$ que explicaremos a continuación).

Para la mutación, por cada pixel en la imagen se calcula un número aleatorio entre 0.0 y 1.0, si el número obtenido es menor a la probabilidad solicitada ($P_{mut.}$), el pixel "se muta" (es decir, si contenía una baliza se elimina ésta y si no la contenía se incorpora una).

Para el desplazamiento se calcula un único número aleatorio entre 0.0 y 1.0 y, si el número obtenido es menor que la probabilidad solicitada ($P_{desp.}$), se desplaza toda la imagen un número aleatorio de pixels entre -10 y +10. La amplitud del desplazamiento sigue, por tanto, una distribución uniforme de entre -10 y +10 pixels.

3.2.4. Modelo de actuación y simulación de los movimientos

En el *Fichero con Recorrido*, las acciones posibles consisten en desplazamientos en x e y y rotaciones en θ. Si una acción incluye desplazamiento y rotación se realizará primero el desplazamiento y luego la rotación por lo que si se quiere girar primero, entonces deberán realizarse dos acciones: primero la rotación y luego el desplazamiento.

El ruido de actuación se simulará como ruido gaussiano (5) de media $\mu = 0$ y desviación estándar σ parametrizable. El programa simulador recibirá como parámetros de entrada dos porcentajes independientes: un porcentaje de error en x e y y otro porcentaje de error en θ de los que derivará la desviación estándar utilizada: σ_x,σ_y y σ_θ . Dichos porcentajes toman valores de 0 a 100. La nueva posición del robot tras una actuación a se calculará como (6).

$$ruido(x) = \frac{1}{\sigma} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{5}$$

$$Pos_{x,y,\theta}(t) = Pos_{x,y,\theta}(t-1) + (a_{x,y,\theta} * (1 + ruido(a_{x,y,\theta}))) \tag{6}$$

El área bajo la curva delimitada por las perpendiculares levantadas entre los puntos a y b de la imagen izquierda de la Figura 21 indican la probabilidad de que la variable X tome un valor cualquiera en ese intervalo. La imagen

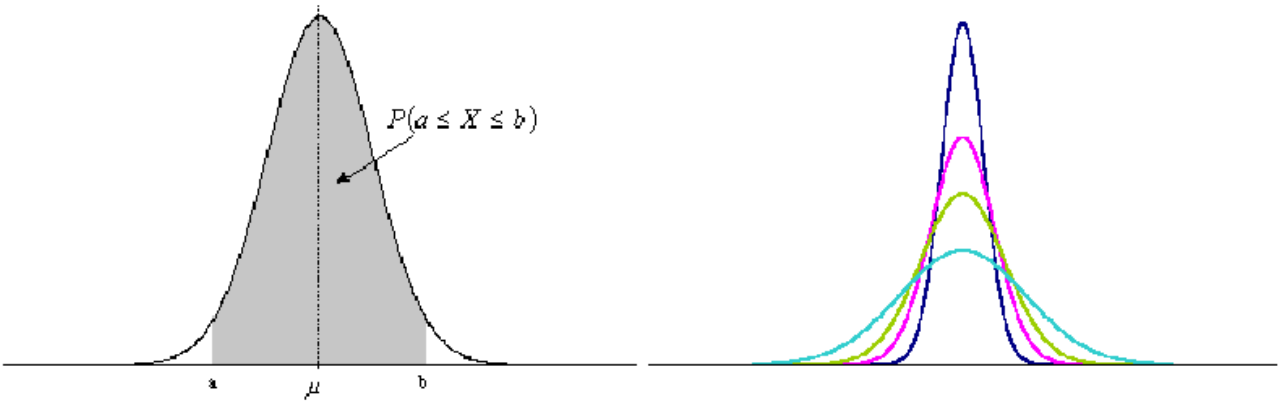

Figura 21: Campana de Gauss utilizada para simular el ruido de actuación

de la derecha muestra cómo varía la forma de la campana en función de la desviación estándar. Cuanto menor es σ, que como se ha indicado se obtiene a partir de los porcentajes de ruido pasados como parámetros al simulador, mayor es la probabilidad de errores de actuación pequeños.

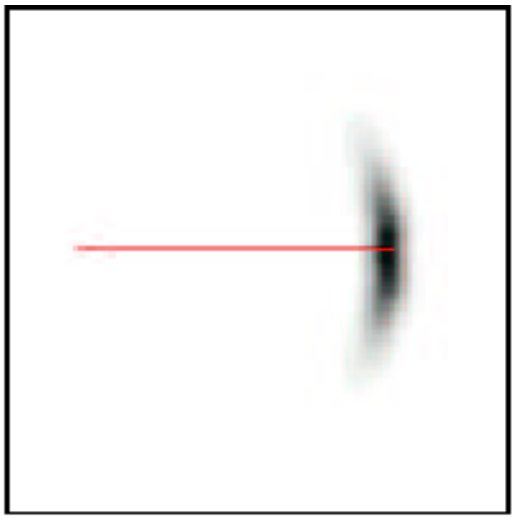

Figura 22: Ruido en el movimiento del robot

La imagen de la Figura 22 ilustra cómo afecta el ruido de actuación a la determinación de la posición del robot. Las zonas más oscuras representan las zonas más probables. A medida que el robot avanza, aumenta la incertidumbre en la pose debido a la acumulación del error.

3.3. Implementación del Algoritmo Localizador

El algoritmo Localizador lee del *Fichero Histórico* la imagen correspondiente a la posición inicial y genera un cubo de probabilidades para todas las poses posibles del mapa comparando la imagen leida con las calculadas para todas las posiciones. A continuación lee una acción y su imagen asociada, aplica la acción comandada al cubo de probabilidades existente y después combina esta información con la proporcionada por la nueva imagen mediante la regla de Bayes. La acumulación de evidencias aportadas por este proceso a medida que se van leyendo nuevos registros, hace aumentar la probabilidad de las posiciones más próximas a la real hasta permitir al robot discriminar el punto en el que está localizado.

3.3.1. Algoritmo

Las evidencias se almacenan en forma de probabilidades, utilizando la regla de Bayes para combinar las evidencias acumuladas con las aportadas por cada observación sensorial y cada actuación. Se utilizan tres "cubos de probabilidades" de dimensiones x, y y θ para representar todas las posiciones y orientaciones posibles en el campo. A cada posición (x, y, θ) del campo de fútbol se le asignará la probabilidad calculada y se almacenará en el "cubo" correspondiente:

1. Cubo 1: Acumulado histórico. Inicialmente, el Cubo 1 contendrá el valor 0.5 para todas las combinaciones (x, y, θ) ya que se desconoce la posición inicial y equiprobabilidad es una buena aproximación (aunque si se dispusiera de otra información a priori, se podría incorporar como una distribución no uniforme). Para instantes posteriores sus valores se calcularán aplicando la acción comandada al Cubo 3, es decir, desplazando en (x, y, θ) la información acumulada hasta el momento.

2. Cubo 2: Nuevas probabilidades a posteriori de las distintas posiciones condicionadas únicamente a la nueva imagen obtenida, $P(position/obs(t))$, independientemente de las imágenes anteriores. Para realizar este cálculo se recorren todas las posiciones del campo y se calcula la distancia entre la imagen real en t y la imagen teórica calculada; la probabilidad es inversamente proporcional a la distancia entre estas dos imágenes (Figura 23).

3. Cubo 3: Es la combinación de los cubos 1 y 2. Al aplicarle la actuación, se vuelca en el cubo 1 su resultado.

En una primera aproximación, para combinar los cubos 1 y 2, se realizaba una media de las probabilidades de ambos, lo cual proporcionaba buenos resultados. Para *formalizar* este proceso, se decidió utilizar la regla de Bayes para acumular evidencias ya que se comporta de forma robusta en la construcción de mapas [Thrun00a] y que también nos lleva a obtener los resultados deseados. El ratio (9) del que obtendremos las probabilidades del Cubo 3, se obtiene, por tanto, como producto de los ratios de las probabilidades de los Cubos 1 y 2 ([Cañas02]):

$$P_{position}(C_{(x,y,\theta)}, t) = P(position/obs(t), data(t-1)) \tag{7}$$

$$r_{map} = \frac{P_{position}}{1 - P_{position}} \tag{8}$$

$$r_{map}(C_{(x,y,\theta)}, t) = \frac{r_{obs}}{r_{apriori}} * r_{map}(C_{(x,y,\theta)}, t-1) \tag{9}$$

Al desconocer la posición inicial del robot, la verosimilitud asignada a todas las posiciones dentro del campo de fútbol inicialmente es 0.5. Cuando posteriormente, y como resultado de las actuaciones, se incorporan nuevas entradas en el cubo de probabilidades, las incertidumbres se inicializan a 0.005.

Al realizar la combinación de los cubos de probabilidades mediante la regla de Bayes (9), se definen como umbrales de saturación 0.999999 (máximo valor) y 0.000001 (mínimo valor). Estos umbrales se utilizan para evitar que las estimaciones se bloqueen cuando toman valores muy próximos a 0 o a 1 y que como consecuencia no reflejen la información aportada por nuevas evidencias [Cañas02].

3.3.2. Modelo probabilístico de observaciones

En nuestro modelo, la probabilidad de que una imagen haya sido vista desde una determinada posición se calcula como $p(position/obs) = e^{-d^2}$ donde d es la distancia entre la imagen teórica y la imagen observada. Dicha distancia se define como la diferencia en pixels entre las dos imágenes comparadas. Este modelo satisface el criterio intuitivo de que cuanto mayor es la distancia, menor es la probabilidad asignada a esa posición (Figura 23). No se trata de un modelo analítico sino de un modelo ad hoc (podrían utilizarse otros modelos) que proporciona valores entre 0 y 1, con mayor probabilidad a medida que las imágenes real y teórica (ideal desde esa posición) son más parecidas.

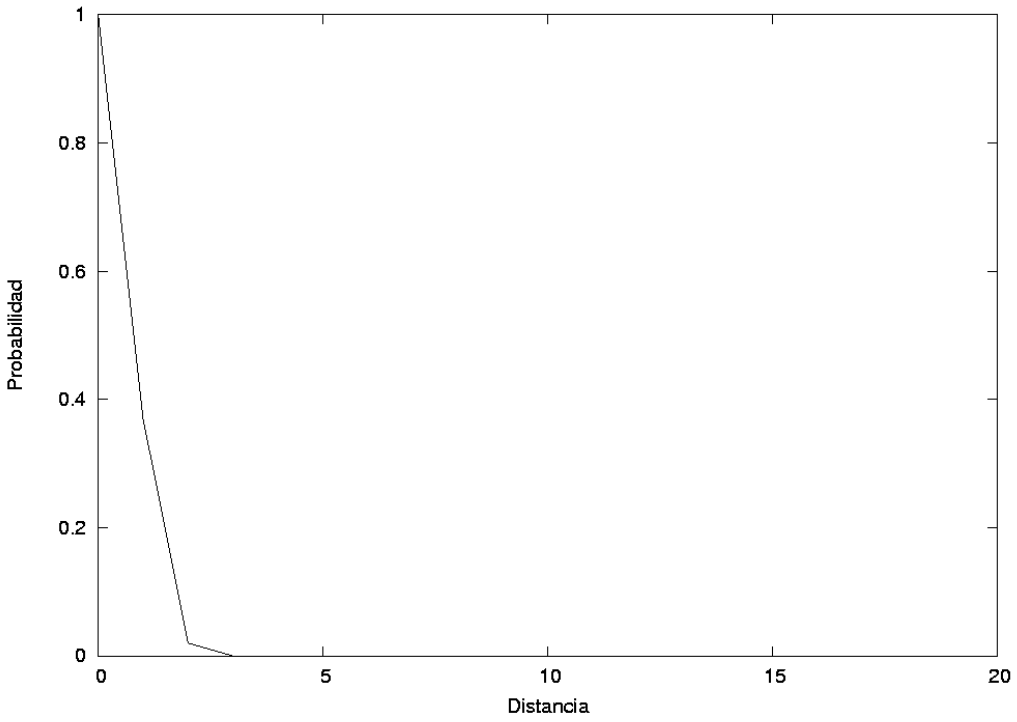

Figura 23: Verosimilitud de la posición en función de la distancia entre observaciones teórica y real

La función distancia se ha implementado comparando la imagen real, es decir, la imagen proporcionada por el Simulador, y la teórica, estimada por el Localizador para cada posición. Por cada baliza encontrada en la imagen real, se busca la más cercana que sea del mismo color en la imagen teórica, entre ellas hay una distancia de d_i pixels. La distancia $d_{12} = \sum_{i=1}^{b_r} d_i$ será la diferencia en pixels de las posiciones de las balizas en cada imagen, sumando las distancias para todas las balizas en la imagen. A continuación se realiza el proceso análogo recorriendo la imagen teórica para obtener la distancia $d_{21} = \sum_{j=1}^{b_t} d_j$. La distancia entre las dos imágenes se calcula como se indica en la fórmula (10) donde b_r es el número de balizas en la imagen real y b_t es el número de balizas en la imagen teórica. Si b_r y b_t son 0 entonces $d = 0$; si sólo una de las imágenes no contiene ninguna baliza, la distancia se calcula como $d = d_{21}/b_t$ para $b_r = 0$ y $d = d_{12}/b_r$ para $b_t = 0$.

$$d = ((d_{12}/b_r) + (d_{21}/b_t))/2 \tag{10}$$

El rango de valores para la distancia es de 0, para imágenes idénticas, a 80, para imágenes completamente diferentes. Como ya se ha indicado, el

cálculo de la verosimilitud se obtiene como e^{-d^2}.

En una versión previa, sólo se recorría la imagen teórica lo cual era menos preciso ya que las distancias calculadas entre ambas imágenes no son simétricas ($d_{12} \neq d_{21}$) puesto que el número de balizas de imagen real y teórica pueden ser diferentes. También se realizaron pruebas con diferentes modelos para calcular la verosimilitud como por ejemplo e^{-d} y $e^{-d^2/\sigma}$ (donde σ es una constante que permite suavizar la función). Para todas ellas el algoritmo permitía la correcta localización del robot, pero se eligió e^{-d^2} por ser la que mejor discriminaba la posición real entre las posibles asignándole una probabilidad mayor que a las otras.

3.4. Resultados experimentales

Se han realizado múltiples experimentos sobre diferentes mapas en los que se han variado la posición y el número de balizas para estudiar el comportamiento del algoritmo localizador probabilístico en distintas condiciones. Dichos experimentos se han realizado tanto sin ruido, en observaciones y acciones, como con ruido de diferente amplitud.

En primer lugar, se presenta un ejemplo de ejecución sobre el mapa más parecido al campo de juego de la Robocup (Figura 17) en una simulación sin ruido de actuación ni ruido sensorial. A continuación se presentarán otros dos ejemplos, también sin ruido, en los que se estudia el efecto de la simetría y del número de balizas en el mapa para el proceso de localización del robot. Finalmente se realizará una comparativa sobre el primer escenario incluyendo ruido.

Como se indicó anteriormente, los mapas se han discretizado definiéndolos como arrays de 33 filas por 65 columnas que representarán las posibles localizaciones en x e y del robot. La orientación θ se calculará en intervalos de 1°. Por lo tanto tenemos un total de $65x33x360 = 772,200$ posibles localizaciones.

En cuanto al rendimiento computacional, el tiempo de proceso varía en función del número de balizas puesto que desde cada nueva posición se verifica si se ve alguna de ellas. Para los mapas con 6 balizas, se necesitan 8 segundos por acción; en los mapas con 12 balizas, se necesitan 12 segundos; en los mapas con 18 balizas, son necesarios 18 segundos por acción. Las pruebas se han realizado en un PC con un procesador Pentium III a 1,133Mhz.

3.4.1. Ejemplo 1 - Ejecución típica

El Mapa de la Figura 17 se crea como el más parecido al campo de fútbol oficial de la Robocup. En él se utilizan 12 balizas en total: 6 balizas (4 se colocan una en cada esquina y otras 2 en los extremos opuestos de la mitad del campo) y dos porterías que simulamos como 3 balizas juntas a cada lado del campo. Cada portería es de un color y el resto de las balizas son todas diferentes entre sí.

Actuación No.	Acción	Posición	Imagen
0	—	40,16,0	E(35)E(40)E(44)
1	10,0,0	50,16,0	E(32)E(40)E(47)
2	0,0,45	50,16,45	C(33)
3	0,0,90	50,16,135	B(34)
4	-10,0,0	40,16,135	B(72)
5	-21,0,0	19,16,135	A(31)
6	0,0,45	19,16,180	D(45)D(39)D(34)
7	0,0,45	19,16,225	F(48)
8	0,0,65	19,16,290	G(6)

Cuadro 2: Fichero histórico, ejecución típica

La localización sobre este mapa es relativamente sencilla debido al número de balizas y a la ausencia de simetría gracias a los diferentes colores de las mismas.

La secuencia de acciones realizadas, nuevas posiciones e imágenes obtenidas se representan en el Cuadro 2. El algoritmo localizador recibirá información relativa a imágenes y acciones pero no las posiciones.

La Figura 24 ilustra la evolución de los *cubos de probabilidad* descritos anteriormente. La probabilidad se representa en tonos de grises, cuanto más claro el gris, mayor la probabilidad. Estas imágenes se generan para todas las posiciones del campo de fútbol en función de las estimaciones calculadas con la información disponible. Las coordenadas de estas imágenes son las mismas que las que se indican en los Mapas. En cada fila, pueden verse los cubos que representan el estado del robot tras incorporar la información relativa a una nueva observación o tras realizar una determinada acción. Se han elegido los ángulos en los que sabemos que se encuentra orientado para visualizar el corte transversal del cubo más significativo.

En la columna izquierda se representan los *cubos de probabilidad* a los que se aplicará la siguiente observación sensorial o acción. En la columna central

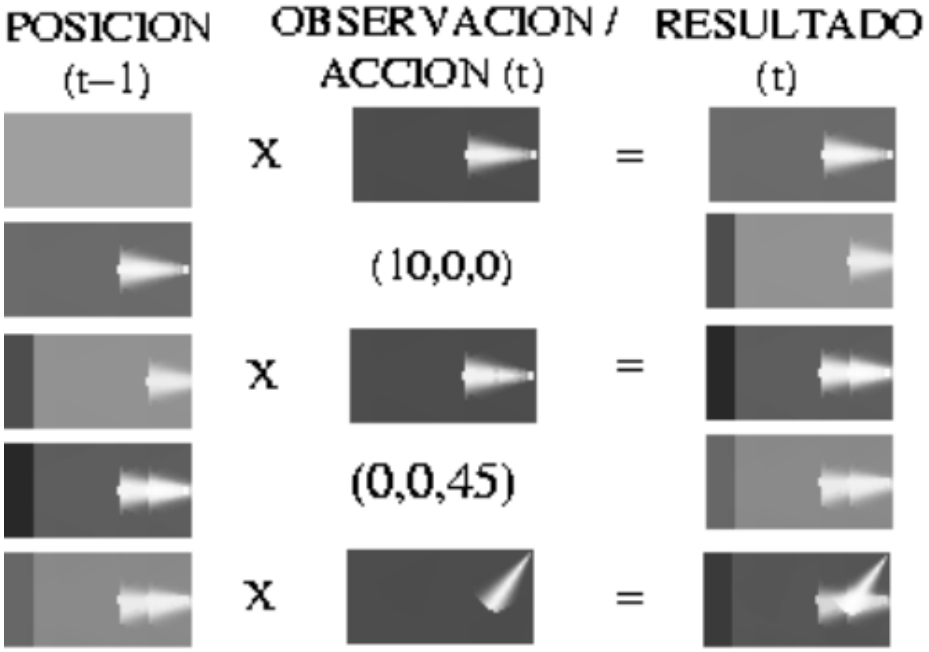

Figura 24: Cubos de Probabilidad, ejecución típica (Ejemplo 1)

se representa la nueva información a incorporar: imagen o acción; la imagen viene representada como un *cubo de probabilidades* calculado para la posición estimada actual (es la visualización bidimensional de $P(position_{x,y,\theta}/obs(t))$; la acción viene dada por un desplazamiento en x e y y/o una rotación en θ. En la columna derecha se representa el resultado de actualizar la columna de la izquierda con la información proporcionada por la columna central. La columna de la derecha se convierte en la columna izquierda de la siguiente iteración.

Se parte de un *cubo de probabilidades* inicializado a 0.5 (esquina superior izquierda de la Figura 24). La primera observación sensorial desde la posición inicial viene representada por el cubo central de la fila superior y la combinación de ambas por el cubo de la derecha. El corte transversal de la imagen corresponde a $\theta = 0\,^{\circ}$. A continuación, se realiza un desplazamiento en x, $(10, 0, 0)$; el cubo resultante de aplicar esta acción se representa en la columna derecha de la segunda fila. De momento, el robot sólo es capaz de saber que se encuentra en algún lugar de la parte derecha del campo. La nueva aportación sensorial desde la nueva posición y la nueva actuación (giro de $45\,^{\circ}$) le permiten reconfirmar su situación en la parte derecha del campo, pero aún no es capaz de estimar su posición. Ahora, el corte transversal de la imagen corresponde a $\theta = 45\,^{\circ}$. La tercera aportación sensorial combinada

con las evidencias acumuladas hasta el momento, le permiten seleccionar la posición $(50, 16, 45)$ -la correcta- como la más probable.

3.4.2. Ejemplo 2 - Efecto de la simetría

En este experimento vamos a estudiar el comportamiento del algoritmo localizador cuando aumenta la simetría en la distribución de balizas en el mapa.

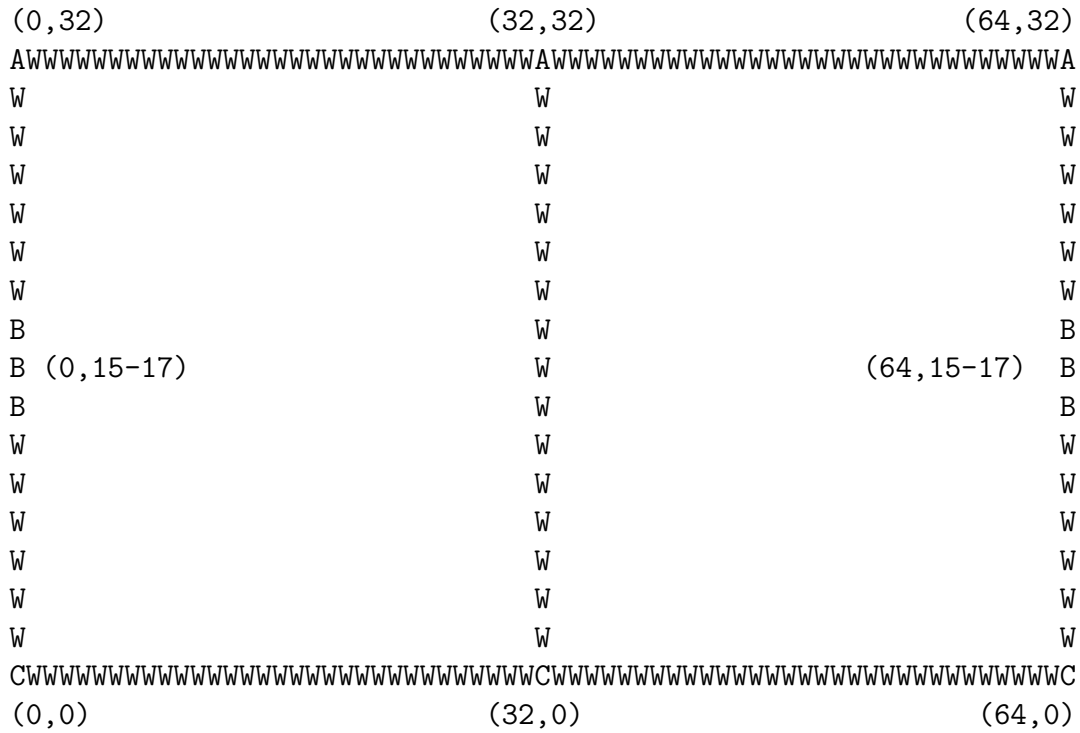

Figura 25: Mapa utilizado para el estudio del efecto de la simetría (Ejemplo 2)

El Mapa de la Figura 25 es exactamente igual al empleado en el Ejemplo 1 (Figura 17) pero utilizando simetría. La simetría se introduce usando sólo tres colores: uno para las 3 balizas situadas a un lado del campo, otro para las 3 balizas situadas al otro lado y finalmente 2 porterías del mismo color (simuladas por 3 balizas juntas cada una).

La secuencia de acciones realizadas, nuevas posiciones e imágenes obtenidas se representan en el Cuadro 3. El algoritmo localizador recibirá información relativa a imágenes y acciones pero no las posiciones. El *Fichero con*

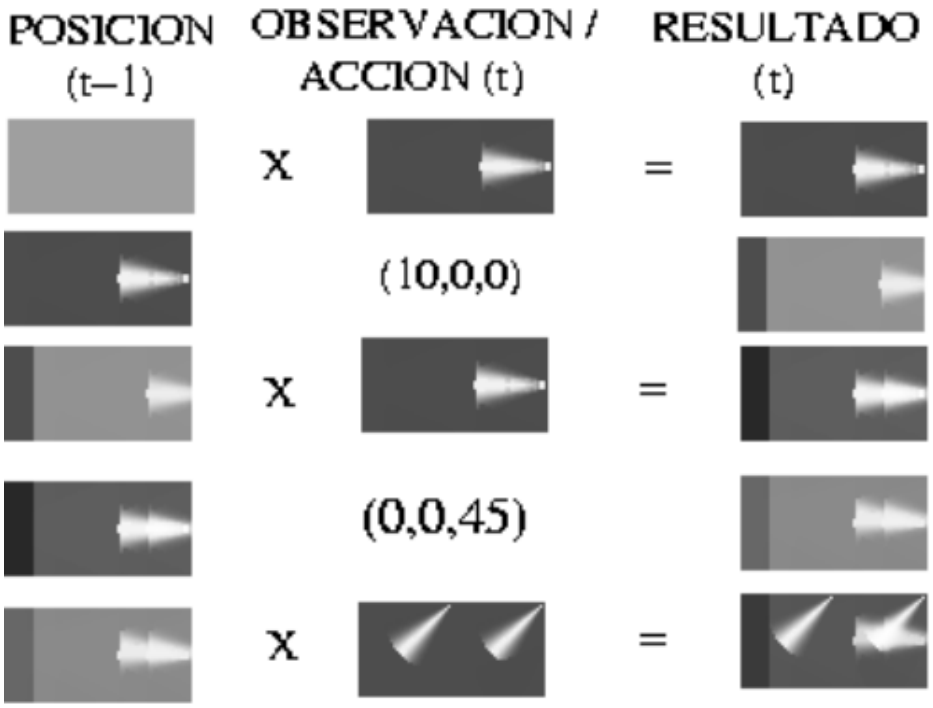

Figura 26: Cubos de Probabilidad, estudio del efecto de la simetría (Ejemplo 2)

Actuación No.	Acción	Posición	Imagen
0	—	40,16,0	B(35)B(40)B(44)
1	10,0,0	50,16,0	B(32)B(40)B(47)
2	0,0,45	50,16,45	A(33)
3	0,0,90	50,16,135	A(34)
4	-10,0,0	40,16,135	A(72)
5	-21,0,0	19,16,135	A(31)
6	0,0,45	19,16,180	B(45)B(39)B(34)
7	0,0,45	19,16,225	C(48)
8	0,0,65	19,16,290	C(6)

Cuadro 3: Fichero histórico, estudio del efecto de la simetría

Recorrido utilizado es idéntico al del Ejemplo 1, el *fichero histórico* ha variado en lo que a imágenes se refiere (como se aprecia si se compara con el Cuadro 2) al variar los colores de las balizas y las porterías.

La evolución de los *cubos de probabilidad* puede observarse en la Figura 26 que es bastante similar a la observada en la ejecución típica. La simetría

hace que el algoritmo tenga mayor dificultad para discriminar la posición, lo que se observa claramente en la tercera observación sensorial (imagen central inferior) comparada con la de la Figura 24.

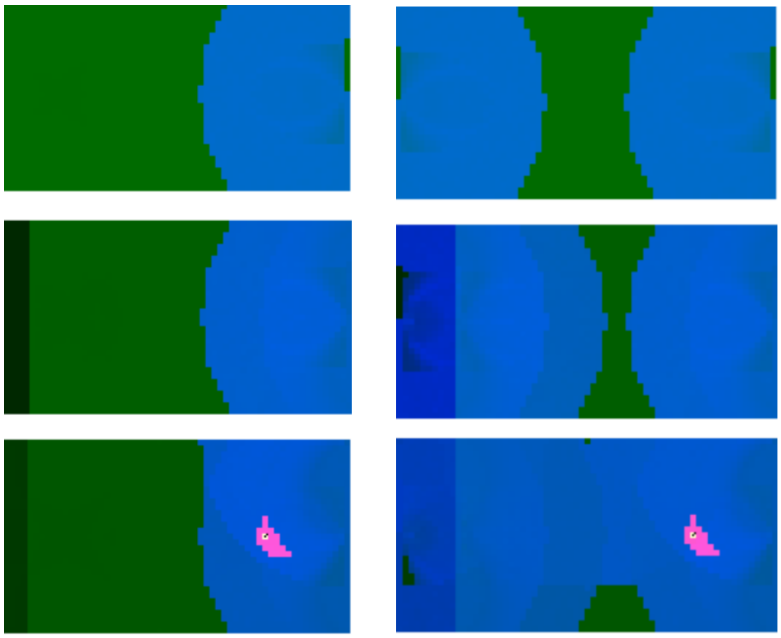

Figura 27: Comparativa de Cubos de Probabilidad acumulada sobre el mapa de la Robocup

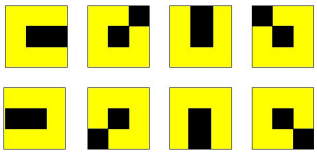

Figura 28: Visualización de la orientación más verosimil del Robot

En la Figura 27 se presenta una nueva forma de visualizar los *cubos de probabilidades*, acumulando la información probabilística obtenida desde todos los ángulos para cada posición (x, y) y representando el valor medio. La columna de la izquierda corresponde a la ejecución típica (Ejemplo 1) y la de la derecha al estudio del efecto de la simetría (Ejemplo 2). Se considerarán "TOP" aquellas posiciones con probabilidad superior a 0.997 y "BOTTOM" aquellas que estén por debajo de 0.5. Las posiciones por encima del umbral definido como TOP se muestran en rojo, las que se encuentran por debajo del umbral definido como BOTTOM se muestran en verde y el resto se muestran en azul. Para la posición (o posiciones) de mayor probabilidad se utiliza

el amarillo y se representa la orientación de máxima probabilidad con una aproximación de 45° como se indica en la Figura 28.

La secuencia de imágenes de la Figura 28 indica cómo se representa la orientación aproximada del Robot en el Campo de fútbol. La primera imagen indicaría que el robot tiene una orientación entre 0° y 44°, la segunda entre 45° y 89°, y así sucesivamente, en intervalos de 45°, hasta la última que sería entre 315° y 359° (es decir, para obtener esta representación, hemos convertido cada posición (x, y) a una nueva de $3x3$ transformando nuestro mapa de $65x33$ en uno de $195x99$ posiciones).

La primera fila de la Figura 27 se obtiene con la información aportada por la primera observación. Las imágenes sucesivas son el resultado de acumular las evidencias aportadas por una nueva acción y una nueva observación conjuntamente. La primera imagen indica que tras la primera observación, el robot sólo sabe que se encuentra en algún lugar de la parte derecha del campo; el efecto de la simetría en el Ejemplo 2 se nota en la reducción de la zonas menos probables (verdes) y el aumento de las posiciones "ambiguas" (espacios azules). La segunda imagen refleja la acumulación de evidencias tras una nueva acción y una nueva imagen que, de momento, no han aportado información suficiente para permitir la localización del robot en ninguno de los dos casos. La tercera imagen muestra como tras el giro de 45° y la observación desde ese punto, aumentan las probabilidades en la zona roja alcanzando el máximo en la zona amarilla que corresponde a la posición del robot $(50, 16, 45)$. En cualquier caso, el algoritmo es capaz de localizar el robot con el mismo número de acciones y observaciones en los dos ejemplos, aunque el Ejemplo 1 sigue siendo más restrictivo en el número de posiciones más probables.

3.4.3. Ejemplo 3 - Efecto del número de balizas

En este experimento vamos a estudiar el comportamiento del algoritmo localizador cuando disminuye el número de balizas en el mapa. Para ello, se utiliza el Mapa de la Figura 29 que contiene 6 balizas de diferentes colores colocadas simétricamente a ambos lados del mapa.

La secuencia de acciones realizadas, nuevas posiciones e imágenes obtenidas se representan en el Cuadro 4. De nuevo, el algoritmo localizador recibirá información relativa a imágenes y acciones pero no las posiciones.

La ejecución se desarrolla como puede verse en la Figura 30. Tras las dos primeras acciones, el robot considera como igualmente probables las siguientes posiciones: (0,32,10) -la correcta-, (3,29,23) y (13,29,55). Con la tercera

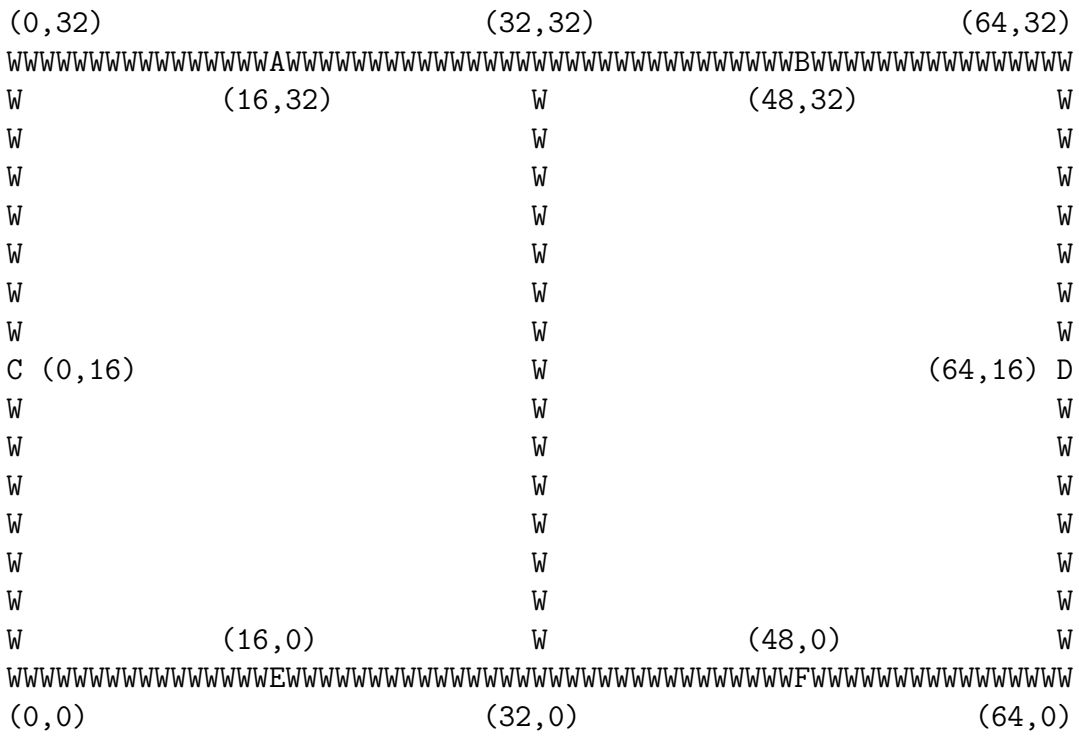

Figura 29: Mapa para estudiar el efecto del número de balizas (sin muestreo)

acción, reduce la incertidumbre a dos únicas posiciones: (16,22,90) - la correcta - y (29,19,135). A partir de la cuarta acción el robot está perfectamente localizado, certidumbre que mantiene hasta el final de la ejecución, tras 15 acciones realizadas. Es decir, el menor número de balizas ha hecho necesarias mayor número de acciones y observaciones para que el robot pudiera cerciorarse de su posición.

En la Figura 31 se representa la acumulación de evidencias para todos los ángulos y todas las posiciones del campo. Como puede verse, tras la observación inicial, lo único que puede deducirse es que el robot se encuentra en la parte izquierda del campo. Tras realizar dos acciones, el robot ya ha acumulado evidencias que han elevado considerablemente la probabilidad de las áreas representadas en rojo; apareciendo en amarillo las posiciones más probables y su orientación. Tras la cuarta acción el robot es capaz de estimar su posición con certeza (sólo una posición en amarillo).

Se han hecho multiples experimentos similares con 6, 12 y 18 balizas con y sin simetría. Para todos los experimentos realizados sin ruido, el robot

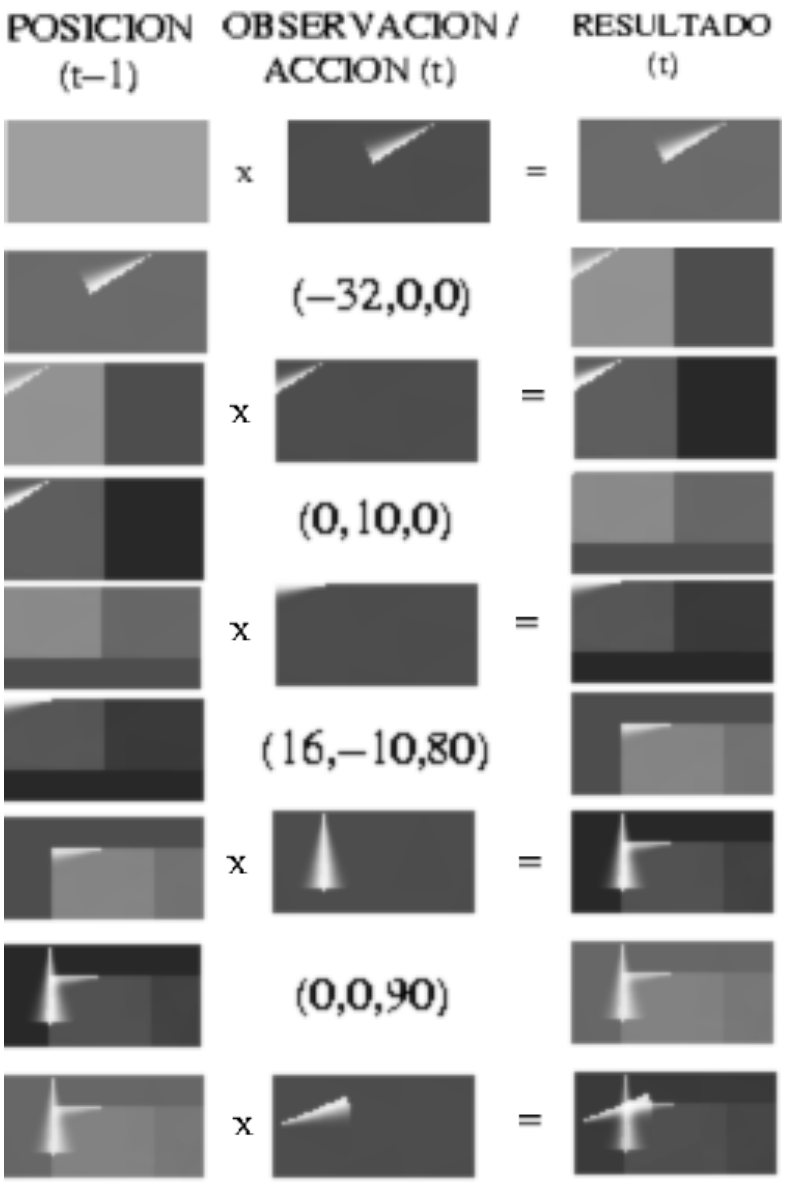

Figura 30: Cubos de Probabilidad, estudio del efecto del número de balizas (Ejemplo 3)

necesita ejecutar un mínimo de 2 acciones para o bien localizarse definitivamente o bien considerar la posición en que se encuentra como una de las más probables. En ningún caso se han necesitado más de 4 acciones para que el robot sepa con certeza cual es su posición. Una vez localizado, no se vuelve a perder aunque una acción no aporte información sensorial nueva (es decir, aunque desde la nueva posición no se vea ninguna baliza).

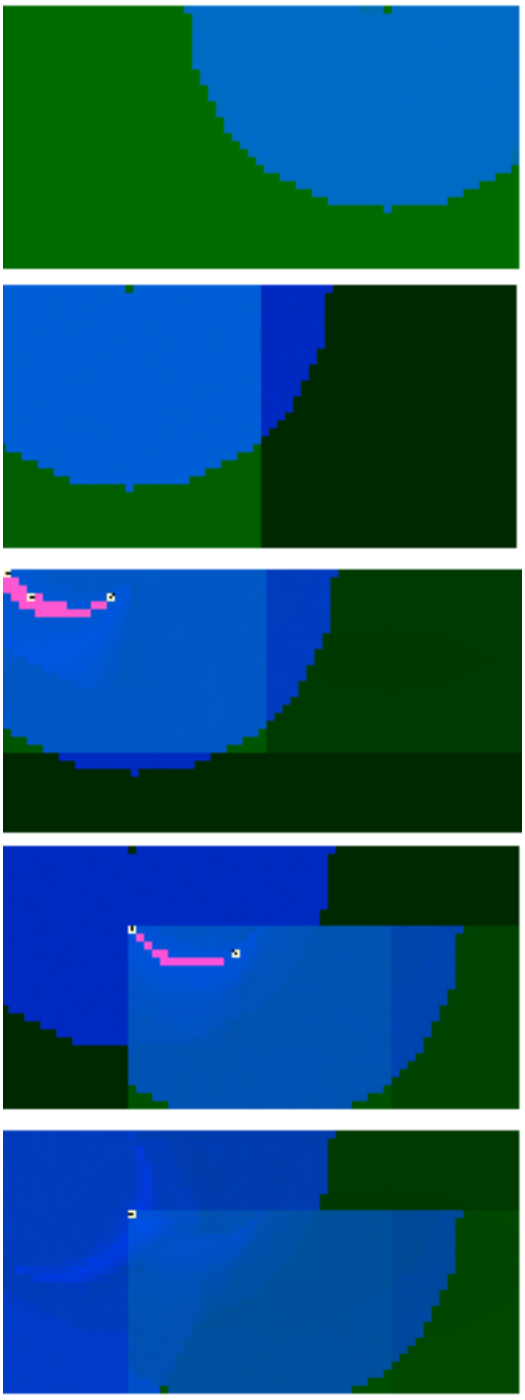

Figura 31: Cubos de Probabilidad acumulada, estudio del efecto del número de balizas (Ejemplo 3)

Acción No.	Acción	Posición	Imagen
0	—	32,22,10	B(0)
1	-32,0,0	0,22,10	A(0)
2	0,10,0	0,32,10	A(57)
3	16,-10,80	16,22,90	A(39)
4	0,0,90	16,22,180	C(3)
5	0,0,90	16,22,270	E(40)
6	33,0,0	49,22,270	F(44)
7	0,0,90	49,22,0	D(78)
8	0,0,90	49,22,90	B(29)
9	0,-19,180	49,3,270	F(72)
10	-1,5,0	48,8,270	F(40)
11	0,0,22	48,8,292	F(79)
12	0,0,-44	48,8,248	F(0)
13	0,0,-136	48,8,112	B(79)
14	0,0,-22	48,8,90	B(39)
15	0,0,-80	48,8,10	D(10)

Cuadro 4: Fichero histórico, estudio del efecto del número de balizas

3.4.4. Ejemplo 4 - Efecto del ruido sensorial y de actuación

A continuación, vamos a presentar otros experimentos que ilustran el rendimiento del algoritmo de localización en nuestro entorno cuando el simulador incorpora ruido sensorial y de actuación. Las pruebas con ruido se realizaron sobre el Mapa de la Figura 17 utilizado en la ejecución típica (Ejemplo 1). Este escenario con ruido es similar a las condiciones que se tendrán en el robot real, donde tanto las observaciones sensoriales como las actuaciones tienen asociada incertidumbre.

En primer lugar se realizaron pruebas que sólo incluían *ruido de actuación*. Como se explicó en el apartado de *Implementación* de este capítulo, el ruido de actuación se simula como ruido gaussiano y aplicado a la acción a realizar resulta en una nueva posición que es diferente a la que se obtendría sin ruido. Este ruido de actuación se refleja en que el robot recibe imágenes que no se corresponden a las tomadas desde la posición ideal (posición inicial + actuaciones puras). Desde esa nueva posición que incorpora el ruido de actuación, el algoritmo simulador genera la imagen a incluir en el fichero histórico de entrada al algoritmo de localización. Si como resultado de una acción *ruidosa* el simulador calcula una posición fuera del mapa utilizado ésta no se incluirá en el fichero histórico, quedando descartada.

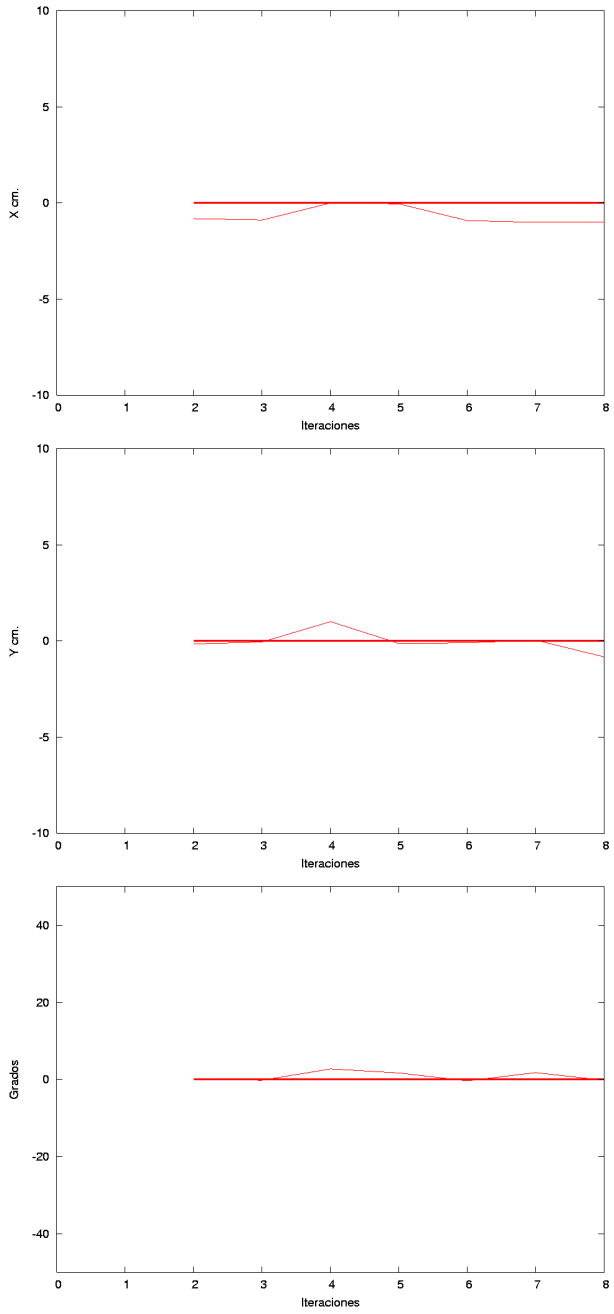

Figura 32: Errores en x, y y θ para ruido de actuación del 5 %

El ruido de actuación se varió entre un 5 % y un 60 % en los experimentos. El robot siempre fue capaz de estimar de modo aproximado su posición real tras 2 acciones, es decir, exactamente igual que en el escenario en el que no

se incluía ruido. La desviación máxima en x e y fue de 3cm. y de 9° en θ.

La Figura 32 muestra gráficamente los errores en centímetros para x e y y en grados para θ, respectivamente, para ruidos de actuación del 5 % en las tres dimensiones. El trazo grueso representa el comportamiento sin ruido. Sin ruido, el robot consigue localizarse tras 2 iteraciones con la máxima precisión que permite el mapa discretizado proporcionado (en nuestro caso celdillas de 1cm.); con ruido, en cambio, la precisión fluctúa en cada iteración, siendo la media inferior a 2cm. en x e y y de 5° en θ.

Act.#	Pos.Real	Prob. 0.1	Prob. 0.5	Prob. 0.9
0	(40,16,0)	————	————	————
1	(50,16,0)	————	————	(53,18,354)
2	(50,16,45)	(50,16,45)	(50,16,45)	(50,20,35)
3	(50,16,135)	(50,16,135)	————	(50,15,135)
4	(40,16,135)	(40,16,135)	————	————
5	(19,16,135)	(19,16,135)	(19,16,135)	(16,16,128)
6	(19,16,180)	(19,16,180)	(19,16,180)	(13,12,161)
7	(19,16,225)	(19,16,225)	(19,16,225)	(20,15,226)
8	(19,16,290)	(19,16,290)	(19,16,290)	(19,16,293)

Cuadro 5: Ejemplos de ruido sensorial de desplazamiento de la imagen

Act.#	Pos.Real	Prob. 0.005	Prob. 0.009	Prob. 0.01
0	(40,16,0)	————	————	————
1	(50,16,0)	————	————	————
2	(50,16,45)	(50,16,45)	(50,16,45)	————
3	(50,16,135)	(50,16,135)	(50,16,135)	————
4	(40,16,135)	(40,16,135)	————	(40,16,135)
5	(19,16,135)	(19,16,135)	(19,16,135)	(19,16,135)
6	(19,16,180)	(19,16,180)	(19,16,180)	(19,16,180)
7	(19,16,225)	(19,16,225)	(19,16,225)	(19,16,225)
8	(19,16,290)	(19,16,290)	(19,16,290)	(19,16,290)

Cuadro 6: Ejemplos de ruido sensorial de mutación

Después se realizaron pruebas que sólo incluían *ruido sensorial*. Como se explicó en el apartado de *Implementación* de este capítulo, el ruido sensorial se simula modificando la imagen *real*, bien desplazando las balizas ($P_{desp.}$) o

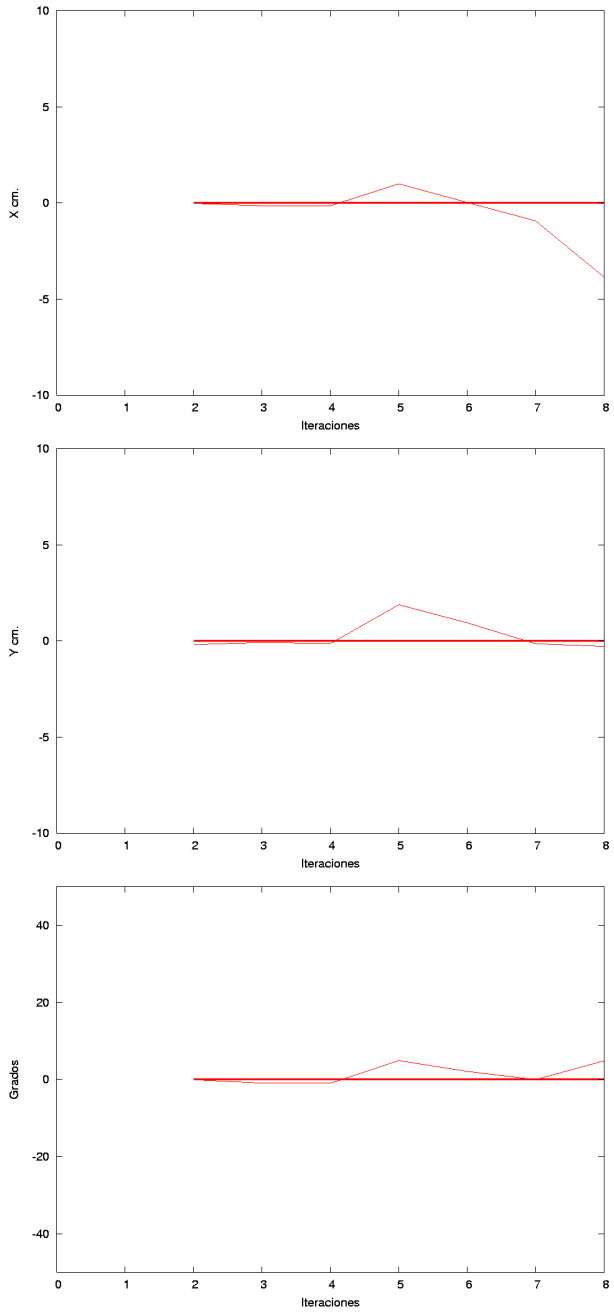

Figura 33: Errores en x, y y θ para ruidos de actuación y sensorial

bien eliminando/añadiendo balizas ($P_{mut.}$). En este caso, se realizaron pruebas variando $P_{desp.}$ como se muestra en el Cuadro 5. Para $P_{desp.} < 0{,}4$, la ejecución se realizó de forma idéntica a una ejecución sin ruido. A partir de

0.5 se pierde precisión y empezaron a detectarse casos en los que el robot se perdía después de haber estado localizado. A partir de 0.8 se pierde precisión con mayor frecuencia. Al aumentar la probabilidad de desplazamiento, perdemos precisión en la estimación, pero el robot sigue teniendo una idea bastante aproximada de su localización real.

Los desplazamientos de la imagen no afectan tanto a la estimación como el hecho de eliminar o añadir balizas que suelen ser el principal motivo por el que el robot no es capaz de discriminar cuál es su posición real entre todas las probables. El Cuadro 6 muestra ejemplos de ruido de mutación; para $P_{mut.} > 0{,}01$ el robot no es capaz de localizarse en la mayoría de los casos. El efecto del desplazamiento de la imagen es menos notable que el de la mutación, con el ruido de mutación, el robot se pierde con más facilidad incluso después de haberse localizado.

Finalmente, se realizaron pruebas que incluían *ruido sensorial y de actuación*. La Figura 33 muestra gráficamente los errores en centímetros para x e y y en grados para θ, respectivamente, en una ejecución típica para ruidos de actuación del 5 % en las tres dimensiones, $P_{desp.} = 0{,}1$ y $P_{mut.} = 0{,}001$. El trazo grueso representa el comportamiento sin ruido. Si se comparan estas figuras con las análogas para sólo ruido de actuación (Figura 32) se puede observar el incremento del error que se produce al incorporar el ruido sensorial.

Dado que el ruido gaussiano de actuación así como el ruido probabilístico sensorial son aleatorios, su efecto en el proceso de localización se estudia como la relación entre valores medios de ruido y de error de localización, y no entre valores puntuales. En los experimentos realizados, para porcentajes razonables de ruido de actuación (por debajo del 30 %) y probabilidades bajas de ruido sensorial ($P_{desp.} <= 0{,}2$ y $P_{mut.} <= 0{,}005$), los errores conseguidos son inferiores a 3cm para x e y y por debajo de 5 ° para θ para la mayoría de las ejecuciones. La precisión conseguida con estos niveles de ruido varía a medida que se le proporciona nueva información (imágenes o acciones) al algoritmo localizador y no aumenta necesariamente con la incorporación de nuevas evidencias.

4. Localización probabilística con muestreo

Un inconveniente de la localización probabilística presentada en el capítulo 3 es que se almacena y actualiza la distribución de probabilidades para todas las posibles posiciones del robot. Esto requiere mucho tiempo de cómputo y hace que el proceso sea lento y no escalable a grandes entornos. Además, es necesario discretizar el mapa lo cual limita la precisión de los resultados obtenidos.

Para reducir drásticamente el número de candidatos sin perder representatividad y utilizar un entorno continuo, se procede aquí a la utilización de técnicas iterativas de muestreo (técnicas de MonteCarlo). Esto nos permite agilizar los cálculos y, potencialmente, aumentar la precisión.

A continuación se introducen los fundamentos teóricos en los que está basado el método de MonteCarlo y cómo han sido implementados en nuestro caso. Finalmente, detallamos los experimentos realizados para probar su funcionamiento.

4.1. Fundamentos teóricos

Las técnicas de MonteCarlo son un conjunto de métodos de muestreo estadístico que investigadores y científicos han desarrollado durante los últimos 50 años para representar cualquier distribución con un número reducido de muestras sin perder representatividad. La metodología se desarrolló inicialmente en el campo de la física entre 1945 y 1955. En los años 80, estadísticos e informáticos desarrollaron una serie de algoritmos basados en estas técnicas que actualmente se utilizan en muchas ramas de la ciencia.

Uno de estos algoritmos, es el *algoritmo de condensación* también conocido como *filtro de partículas*. Este algoritmo nos permite muestrear la verosimilitud de las localizaciones posibles del robot, la cual estimaremos a partir de la secuencia de observaciones y movimientos del robot en un período de tiempo. La idea de los algoritmos de filtros de partículas es representar la verosimilitud mediante un conjunto de muestras con distribución acorde a la verosimilitud $Bel(x) = \{x^{(i)}, p^{(i)}\}, i = 1, ..., m$ donde $x^{(i)}$ es una muestra y $p^{(i)}$ su probabilidad.

Se tiene una población limitada de muestras que representan las posibles posiciones del robot. Las muestras se generan en cada instante a partir de las del instante anterior junto con las mediciones del entorno (en nuestro caso imágenes) y las acciones ejecutadas por el robot (desplazamiento en x, y, θ). Inicialmente, el conjunto de muestras se distribuye uniformemente de forma

aleatoria. Posteriormente, en cada instante de tiempo, se siguen tres pasos recursivamente:

- *Predicción*, se trasladan las muestras según la acción ejecutada y se estima la nueva posición de las muestras.
- *Actualización*, se calculan las probabilidades a posteriori de cada muestra, dada la última observación sensorial.
- *Remuestreo*, se genera un nuevo conjunto de muestras con distribución proporcional a la verosimilitud de las muestras anteriores.

[Sáez02] describe estos pasos en el algoritmo que se resume a continuación. Se parte del conjunto anterior de muestras en el instante t-1 (distribución uniforme), para construir un nuevo conjunto de muestras para el instante actual t:

$$M_{t-1} = \{(\varphi_{t-1}^1, \omega_{t-1}^1), (\varphi_{t-1}^2, \omega_{t-1}^2), ...(\varphi_{t-1}^n, \omega_{t-1}^n)\}$$

$$M_t = \{(\varphi_t^1, \omega_t^1), (\varphi_t^2, \omega_t^2), ...(\varphi_t^n, \omega_t^n)\}$$

1. **Fase de predicción:**
   Sea a_{t-1} la acción ejecutada por el robot en el instante t-1. Para cada muestra $\varphi_{t-1}^i \in M_{t-1}$ se predice su nuevo estado en el instante t, añadiendo un cierto error de movimiento:

   $$\widetilde{\varphi}_t^i = \varphi_{t-1}^i + a_{t-1} + (N(0, \sigma_x), N(0, \sigma_y), N(0, \sigma_\theta)), i = 1, 2, ...N$$

   De esta forma construimos un nuevo conjunto de N muestras $\widetilde{M}_t$ para el instante actual t.

2. **Fase de actualización de observaciones:**
   Sea v_t la observación realizada por el robot en el instante actual t. Para cada muestra $\varphi_t^i \in M_t$ se actualiza su probabilidad asociada $\widetilde{\omega}_t^i$ según la verosimilitud de que la observación v_t haya sido observada desde el estado $\widetilde{\varphi}_t^i$.

   $$\widetilde{\omega}_t^i = p(\widetilde{\varphi}_t^i | v_t), i = 1, 2, ...N$$

3. **Fase de remuestreo:**
   Construir un nuevo conjunto de N muestras M_t remuestreando con sustitución el conjunto $\widetilde{M}_t$, de forma que escoja cada muestra $\widetilde{\varphi}_t^i$ con probabilidad proporcional a la verosimilitud de la misma $\widetilde{\omega}_t^i$.

$$(\varphi_t^i, \omega_t^i) \longleftarrow \text{ Escoger muestra de } \widetilde{M}_t, i = 1, 2, ...N$$

   Normalizar las probabilidades ω_t^i de las muestras de M_t de forma que $\sum_{i-1}^{N} \omega_t^i = 1$. Para ello,

$$\omega_t^i \longleftarrow \frac{\omega_t^i}{\sum_{j=1}^{N} \omega_t^j}, i = 1, 2, ...N$$

A diferencia del método probabilístico sin muestreo, aquí no hay memoria explícita de las verosimilitudes del conjunto muestral en instantes anteriores ya que se recalculan para cada nuevo estado. La "historia" va implícitamente vinculada a la nueva distribución de las muestras que se concentran alrededor de las posiciones más probables en el instante anterior y que por tanto llevan a la convergencia de la población de muestras a la zona más probable que coincide con la posición estimada (próxima a la real) del robot.

4.2. Implementación

El algoritmo localizador (CONDENSATION) implementado en nuestro entorno parte de la información del simulador que se describió en el capítulo 3 (Figura 15). El simulador lee el *Fichero con Recorrido* y comprueba sobre el *Mapa* de entrada qué balizas capta la cámara y en qué pixel tanto para la posición inicial del robot como para las sucesivas. La secuencia de posiciones se obtienen como resultado de aplicar las acciones indicadas, añadiéndoles el porcentaje de ruido de actuación solicitado. A continuación se aplica el ruido sensorial solicitado a la imagen calculada y se crean los registros en el *Fichero Histórico* de entrada al algoritmo localizador. El simulador graba en el fichero histórico las imágenes observadas desde las sucesivas posiciones, añadiendo un cierto ruido sensorial. Este fichero histórico, con las imágenes y desplazamientos sucesivos del robot, es el fichero de entrada desde el cual el algoritmo localizador ha de estimar la posición real del robot.

Las fases del algoritmo localizador presentadas en los fundamentos teóricos se han implementado como se detalla a continuación:

4.2.1. Fase de predicción

En la fase de predicción se parte de una distribución uniforme de muestras en el campo de fútbol ya que se desconoce la posición inicial del robot. En las sucesivas iteraciones, las nuevas muestras se corresponden con las obtenidas en la fase de remuestreo de la iteración anterior tras aplicar la acción ejecutada por el robot añadiendo un cierto *error de movimiento*. Se aplican dos porcentajes de error: uno para (x, y) y otro para θ.

Es necesario recalcar que el *error de movimiento* utilizado en esta fase es diferente al ruido de actuación que se describió en el apartado de simulación del capítulo 3, aunque se genera de igual forma a partir de una distribución normal. Este error en el algoritmo localizador permite que el algoritmo no se bloquee si dentro del conjunto de muestras no se encuentra una cuyas coordenadas correspondan con las de la posición del robot. Sin error, el bloqueo se produciría cuando el conjunto muestral inicial no incorporase ninguna muestra representativa de la posición del robot. Al introducir cierto porcentaje de error, se incorporan muestras que pueden ser mejores o peores que las ya seleccionadas; las mejores ayudarán a que la convergencia se produzca correctamente en torno a la posición real del robot mientras que las peores serán desechadas en la siguiente fase de remuestreo.

4.2.2. Fase de actualización de observaciones

En la fase de actualización, se calcula la probabilidad asociada a cada muestra en base a la información proporcionada por el modelo sensorial en la última lectura sensorial. Como en el caso del método probabilístico sin muestreo, se realizaron pruebas con diferentes modelos para ajustar la probabilidad de que una imagen haya sido vista desde una determinada posición $p(\widetilde{\varphi}_t^i|v_t)$: e^{-d}, e^{-d^3}, e^{-d^2} y $e^{-d^2/\sigma}$, donde d es la distancia entre la imagen teórica y la imagen observada (v_t) calculada tal y como se explicó en la sección *Modelo probabilístico de observaciones* del capítulo 3. Los dos primeros modelos (e^{-d} y e^{-d^3})no proporcionaban los resultados deseados ya que no se conseguía que las muestras convergiesen en torno a la posición real del robot (como se verá en el Ejemplo 4). Los mejores resultados se obtuvieron para $e^{-d^2/256}$ (Figura 34).

Cuando no existe información sensorial, es decir, cuando no se aprecia ninguna baliza, pueden ocurrir 2 cosas:

1. si la muestra se encuentra en una posición desde donde el robot no ve nada, se les asigna probablidad 0.5 en vez de 1. Esto se hace así para relativizar su peso con respecto a situaciones más informativas;

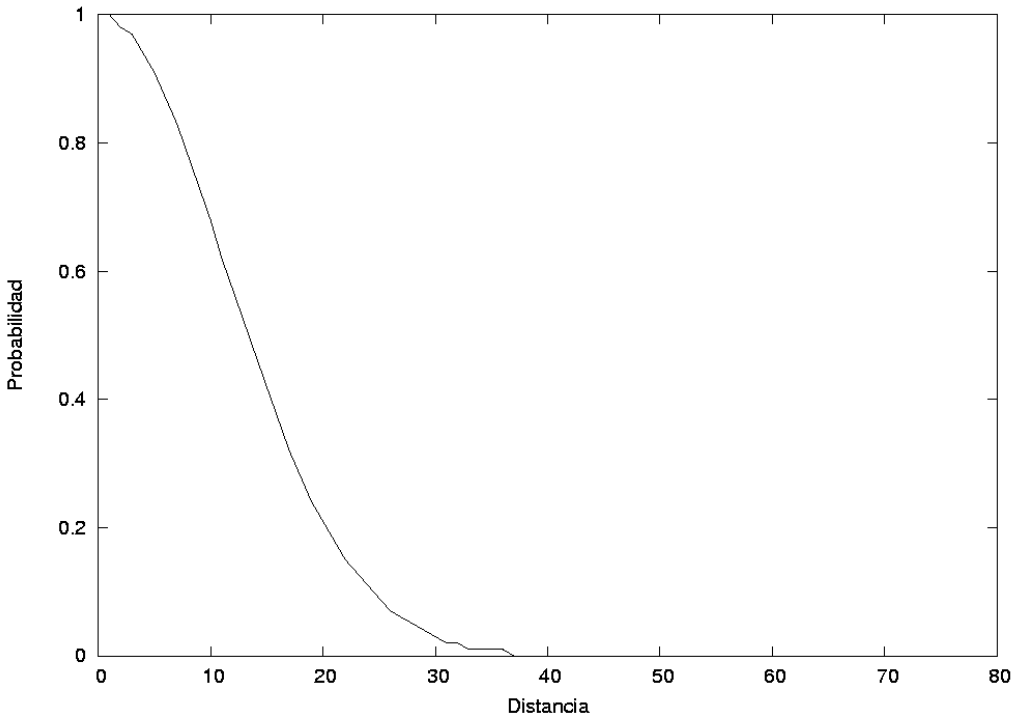

Figura 34: Verosimilitud de la posición en función de la distancia entre observaciones teórica y real

2. si la muestra está en una posición desde la que se ve algo se les asigna probabilidad 0.000001 en vez de 0. Esto se hace por la misma razón, aunque en este caso se sabe con certeza que el robot no puede estar en esa posición.

Del mismo modo que "sin muestreo" utilizamos umbrales para evitar la saturación típica asociada al uso de la regla de Bayes, "con muestreo" también se han definido umbrales que permiten una buena redistribución de las muestras en la fase de remuestreo: 0.95 (valor máximo) y 0.30 (valor mínimo).

4.2.3. Fase de remuestreo

En la fase de remuestreo se calcula un nuevo conjunto de muestras a partir de las existentes de forma que las nuevas muestras se concentren en las zonas de mayor probabilidad. De este modo, se garantiza la convergencia del conjunto muestral a la posición real del robot.

Para ello, se calcula la distribución acumulada (suma de las probabilidades del conjunto de muestras actual) y se generan números aleatorios uniformes entre los valores máximo y mínimo de dicha distribución acumulada. Se generan tantos números como muestras haya, seleccionando cada vez la muestra cuyo valor acumulado está inmediatamente por encima del número calculado. De esta forma, las muestras con mayor probabilidad serán las que se seleccionen mayor número de veces.

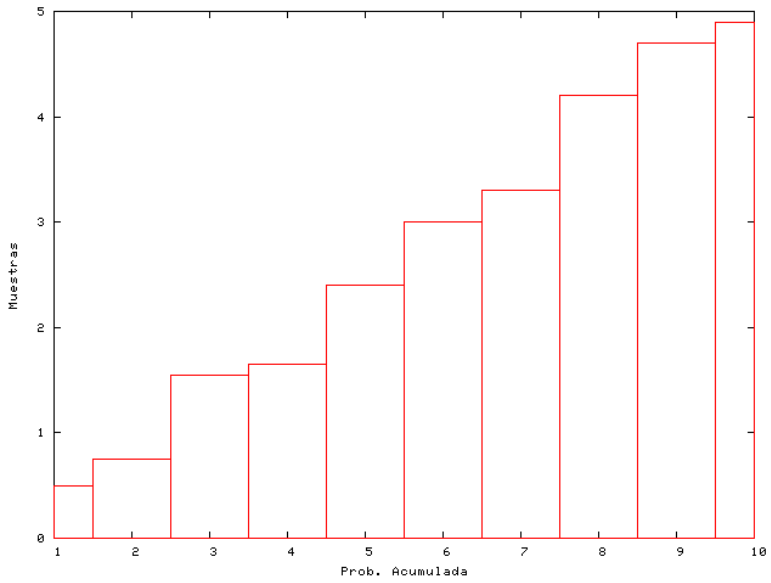

Figura 35: Probabilidades acumuladas del conjunto muestral

En la figura 35 se ilustra con un ejemplo cómo se realiza la implementación de esta fase para 10 muestras de probabilidades 0.5, 0.25, 0.8, 0.1, 0.75, 0.6, 0.3, 0.9, 0.5 y 0.2 (cuyos valores se van acumulando y representando en el eje y) para las muestras 1 a 10 respectivamente (representadas en el eje x). En este caso, se calcularían diez números aleatorios entre 0 y 4.9, que es la suma total de las probabilidades de las 10 muestras, y se elegiría la muestra del eje x cuyo valor corresponde al tramo en el que se encuentra dicho número sobre el eje y. Así, si el número aleatorio calculado fuese, por ejemplo, 1.3, se elegiría la muestra 3; si el valor fuese 3, se elegiría la muestra 6 y así sucesivamente. Como se puede ver, las muestras de mayor probabilidad están reflejadas en mayores tramos sobre el eje y y por lo tanto son elegidas un mayor número de veces que las de menor probabilidad.

4.3. Resultados experimentales

El funcionamiento del algoritmo CONDENSATION depende del ajuste adecuado de los diferentes parámetros que intervienen en el mismo y que se encuentran estrechamente ligados entre sí: número de muestras, error de movimiento, modelo de observaciones y entorno en el que se ejecuta. Los experimentos realizados estudian el efecto de estos parámetros en el algoritmo con objeto de determinar qué valores son los más adecuados para localizar a nuestro robot en nuestro entorno. Los 7 primeros ejemplos se realizaron sin ruido sensorial ni ruido de actuación en el Simulador, estudiándose el efecto de los mismos en el Ejemplo 8.

En los experimentos realizados, se ha observado que la localización se estabiliza tras 13-16 iteraciones para la mayoría de los casos, siendo 4 segundos el tiempo medio empleado para localizarse. En el método probabilístico sin muestreo, la localización era posible tras 2-4 iteraciones, sin embargo eran necesarios 12 segundos por iteración para la ejecución típica. Este tiempo lo hacía no escalable a entornos de mayor tamaño porque aumenta exponencialmente. Otro detalle significativo es que sin muestreo nos vemos obligados a discretizar el entorno limitando la precisión a las "celdillas" definidas, en cambio con muestreo contamos con un entorno continuo y conseguimos precisión de 1.9cm en x y 1.8 cm en y, es decir, conseguimos localizar al robot futbolista en el terreno de juego sólo con visión en 4 segundos.

Dado que el comportamiento del algoritmo depende de la población inicial de muestras, para una misma combinación de parámetros se pueden obtener diferentes resultados. Puesto que queremos que sea correcto independientemente de la distribución de la población inicial y para dar representatividad a los ejemplos detallados a continuación, se muestra el comportamiento más frecuente que se obtuvo realizando cada experimento 20 veces.

A continuación se presentan los siguientes estudios: Ejecución típica (marco de referencia para comparación con otros experimentos), Efecto del aumento del error de movimiento, Efecto de la disminución del número de muestras, Efecto del modelo probabilístico de observaciones, Efecto del número de balizas, Efecto del aumento del tamaño del campo, Efecto de la semilla y Efecto del ruido sensorial y de actuación.

4.3.1. Ejemplo 1 - Ejecución típica

Como ejecución típica se ha elegido un entorno similar al utilizado en la Ejecución típica sin muestreo: El mapa es el mismo (Figura 17), y el fichero histórico se ilustra en el Cuadro 7 (corresponde a una ampliación de

la secuencia de acciones del Cuadro 2 utilizado "sin muestreo"). La función distancia utilizada, $e^{-d^2/256}$, es la que dió mejores resultados en todos los experimentos realizados. Se utilizaron 2,000 muestras y error de movimiento 1.8 % por ser el caso en el que se consiguió mayor precisión en la localización.

Actuación No.	Acción	Posición	Imagen
0	——	50,16,135	B(34)
1	0,0,-90	50,16,45	C(33)
2	0,0,-45	50,16,0	E(32)E(40)E(47)
3	-10,0,0	40,16,0	E(35)E(40)E(44)
4	0,-8,225	40,8,225	G(40)
5	-10,0,45	30,8,270	G(15)
6	-11,0,-45	19,8,225	F(79)
7	0,8,0	19,16,225	F(48)
8	0,0,-45	19,16,180	D(45)D(39)D(34)
9	0,0,-45	19,16,135	A(31)
10	0,8,90	19,24,225	D(79)D(74)
11	-9,0,0	10,24,225	D(57)D(51)D(45)
12	-6,0,0	4,24,225	D(12)D(7)D(2)
13	0,0,-90	4,24,135	A(72)
14	6,0,0	10,24,135	A(28)
15	6,0,0	16,24,135	A(7)

Cuadro 7: Fichero histórico para ejemplos con muestreo

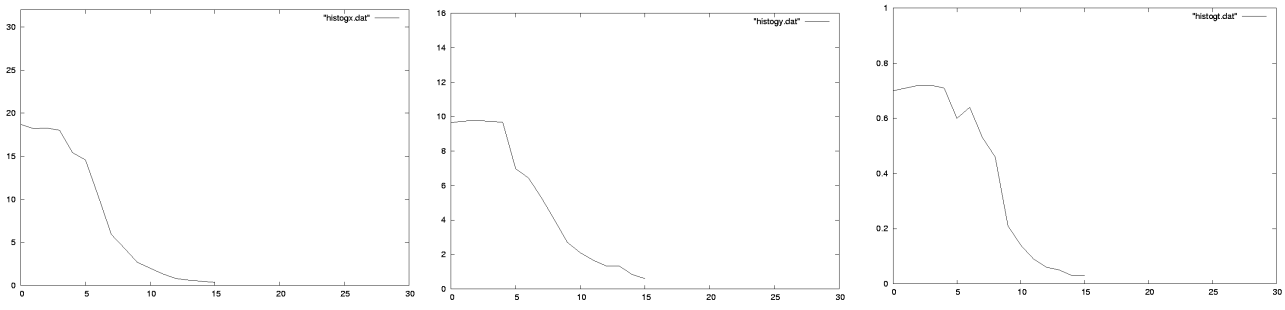

Figura 36: Ejecución típica: desviación típica x, y y $cos(\theta)$

Los resultados de la ejecución se ilustran en la Figuras 36, 37 y 38. En la Figura 36 se representa la desviación típica de las coordenadas x e y y la desviación típica en $cos(\theta)$, lo que nos indica el grado de dispersión de las muestras en torno a una posición candidata. La media de las desviaciones

típicas para las ejecuciones realizadas fue de 1.9cm en x y 1.8 cm en y. En el caso de θ, no se utiliza el ángulo directamente sino su coseno para evitar problemas en la medida debido a la discontinuidad existente entre $0\,^\circ$ y $360\,^\circ$.

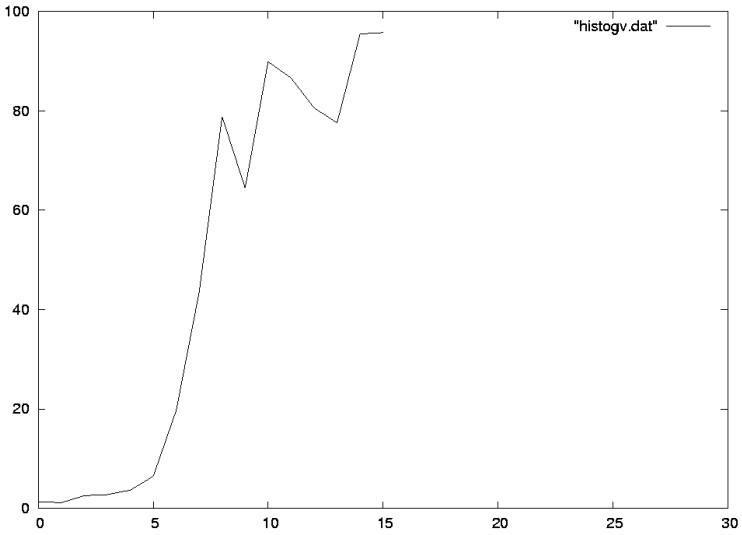

Figura 37: Ejecución típica: posiciones verosímiles

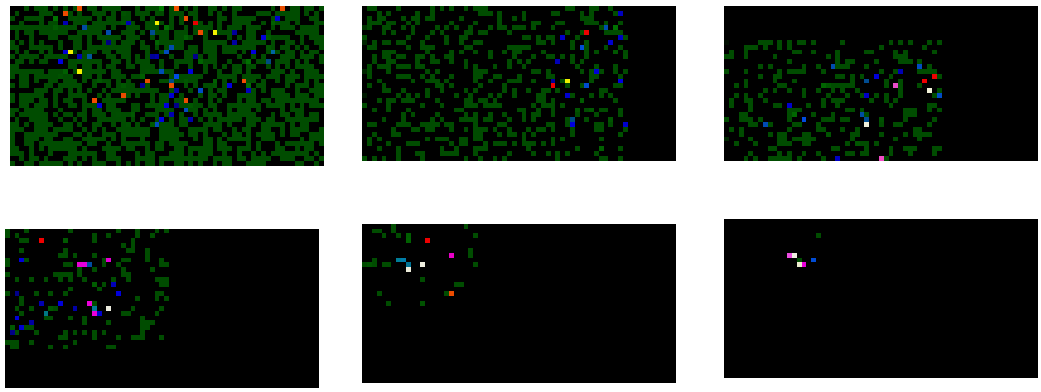

Figura 38: Ejecución típica: evolución del conjunto muestral

Una vez que sabemos que la población de muestras converge, hay que verificar que lo hace alrededor de la posición correcta. La Figura 37 representa el porcentaje de muestras verosímiles de acuerdo con la última observación sensorial para cada iteración, es decir, si las muestras convergen o no alrededor de posiciones verosímiles. Se consideran posiciones verosímiles aquellas cuya probabilidad a posteriori es superior a 0.8 . La media de posiciones verosímiles para las ejecuciones realizadas fue del 44.3 %.

En la Figura 38 se representa cómo evoluciona espacialmente la población de muestras en x e y. Los gráficos corresponden a las iteraciones 0, 3, 5, 8, 11 y 15. Al igual que se hizo en las representaciones gráficas del capítulo anterior, el color verde representa las muestras muy poco probables, el azul las muestras poco probables, el rojo las muestras muy probables y el amarillo las muestras de mayor probabilidad. Puede verse que se parte de dos mil muestras uniformemente distribuidas que finalmente se concentran en torno a la posición del robot.

Todas las representaciones gráficas nos indican, por tanto, que la convergencia va aumentando con las iteraciones pero que existen fluctuaciones en el número de posiciones verosímiles. Estos "picos" se deben a la calidad de la información sensorial que se incorpora en cada instante. Cuando una imagen puede ser observada desde muchas posiciones (lo cual sucede repetidamente en los ejemplos utilizados dado el sencillo modelo sensorial empleado) dependiendo de cómo haya ido evolucionando el conjunto muestral, se pueden producir fluctuaciones en dicho número en relación con la última observación que son más pronunciadas cuando el número de posiciones verosímiles del conjunto muestral es menor.

4.3.2. Ejemplo 2 - Efecto del error de movimiento

En este ejemplo vamos a estudiar el efecto del aumento del parámetro *error de movimiento* en el proceso de localización, para lo que se comparará una ejecución sobre el Mapa de la Figura 17, con el fichero histórico del Cuadro 7, con 2,000 muestras, función distancia $e^{-d^2/256}$ para el modelo probabilístico de observaciones y error de movimiento del 15 % con la Ejecución típica (15 % vs. 1.8 %).

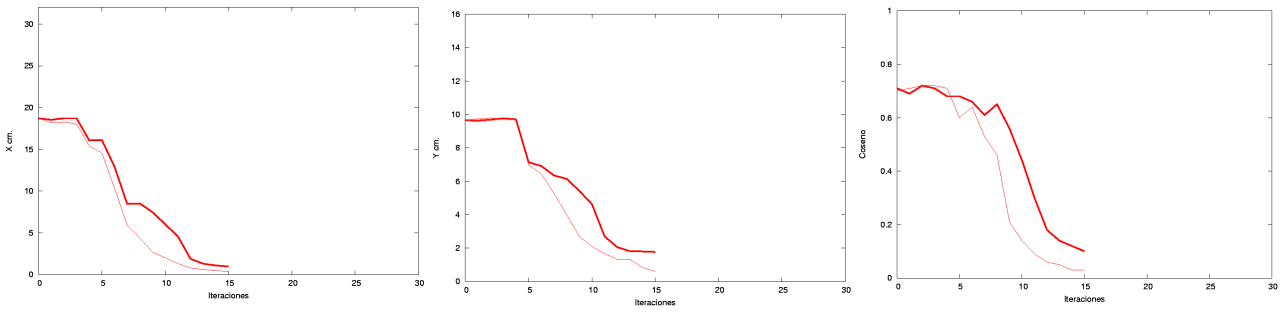

Figura 39: Efecto del error de movimiento: desviación típica x, y y $cos(\theta)$

En las Figuras 39 y 40 se representan con trazo grueso los resultados de este experimento y en trazo fino los resultados del experimento 1, lo que

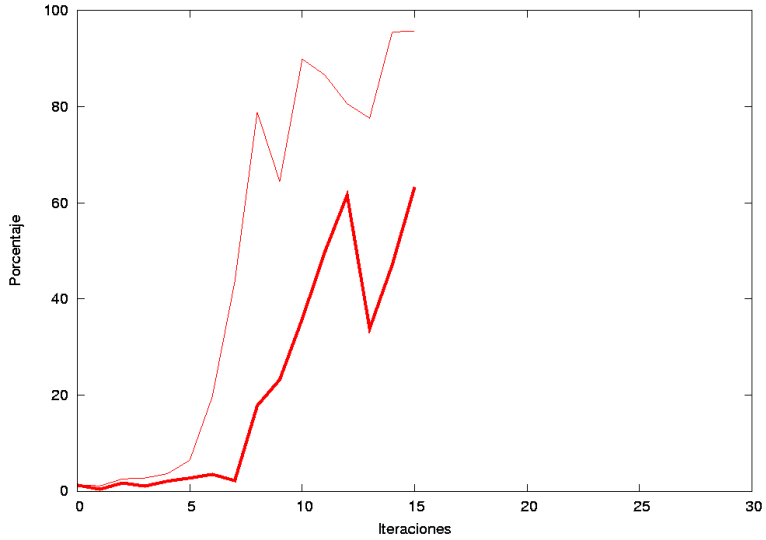

Figura 40: Efecto del error de movimiento: posiciones verosímiles

nos permite observar el efecto del aumento del error de movimiento. Por un lado se aprecia que las desviaciones típicas crecen ligeramente, es decir, la precisión conseguida es ligeramente inferior a la conseguida con menor error; por otro lado, se observa la disminución del número de posiciones verosímiles.

No obstante, con mayor error de movimiento se consigue que las variaciones entre ejecuciones sean menos pronunciadas y que el comportamiento sea más robusto en ejecuciones con ruido de actuación y sensorial o con menor número de muestras como veremos posteriormente. La media de las desviaciones típicas para las ejecuciones realizadas fue de 1.7cm en x y 2.4cm en y; la media de posiciones verosímiles fue del 58.1 %. Por tanto, cuanto mayor es el error de movimiento, menor es la precisión pero mayor es la estabilidad del algoritmo.

4.3.3. Ejemplo 3 - Efecto del número de muestras

En este ejemplo vamos a estudiar el efecto de la disminución del número de muestras en el proceso de localización para lo que se comparará la ejecución sobre el Mapa de la Figura 17, con el fichero histórico del Cuadro 7, con 500 muestras, función distancia $e^{-d^2/256}$ para el modelo probabilístico de observaciones y error de movimiento del 15 % con el Ejemplo 2 (500 muestras vs. 2,000 muestras).

Las Figuras 41 y 42 se obtienen por superposición de los resultados obtenidos en las ejecuciones de los Ejemplos 2 y 3 (2,000 muestras en trazo fino y

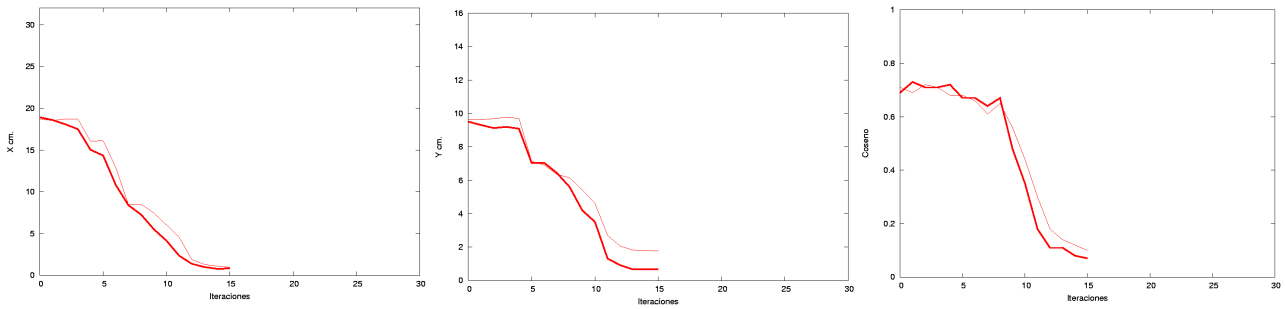

Figura 41: Efecto del número de muestras: desviación típica x, y y $cos(\theta)$

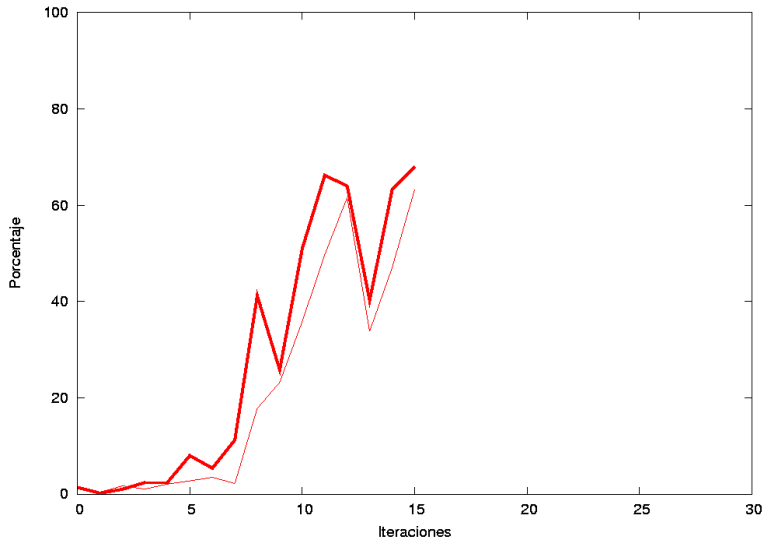

Figura 42: Efecto del número de muestras: posiciones verosímiles

500 muestras en trazo grueso, respectivamente). Podemos observar un comportamiento bastante similar en ambos casos tanto en la precisión obtenida como en el número de posiciones verosímiles comportándose de hecho algo mejor con menos muestras. Sin embargo, si reducimos el error de movimiento para aumentar la precisión, el robot no es capaz de localizarse utilizando sólo 500 muestras.

Por tanto, un mayor número de muestras no ayuda necesariamente a la convergencia ni a la obtención de mayor precisión en los resultados por encima de un determinado número pero sí conlleva un mayor coste computacional. Hemos comprobado, no obstante, que por debajo de un cierto umbral en el número de muestras, y ligado al resto de los parámetros, puede que no se consiga la convergencia. Por ejemplo, la ejecución típica con sólo un 1.8 % de

error de movimiento no localiza al robot con 500 muestras.

4.3.4. Ejemplo 4 - Efecto del modelo probabilístico de observaciones

En este ejemplo vamos a estudiar el efecto de la "función distancia" en el proceso de localización para lo que se comparará la ejecución sobre el Mapa de la Figura 17, con el fichero histórico del Cuadro 7, con 10,000 muestras, función distancia e^{-d^2} para el modelo probabilístico de observaciones y error de movimiento del 15 % con el Ejemplo 2 (e^{-d^2} y 10,000 muestras vs. $e^{-d^2/256}$ y 2,000 muestras).

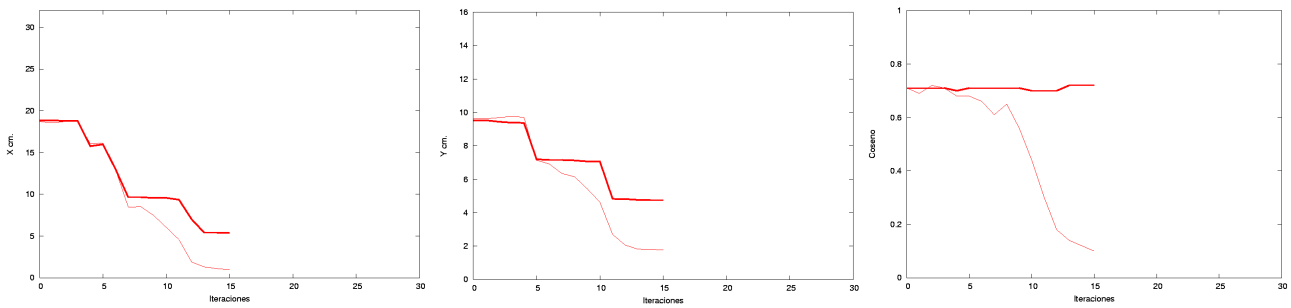

Figura 43: Efecto del modelo de observaciones: desviación típica x, y y $cos(\theta)$

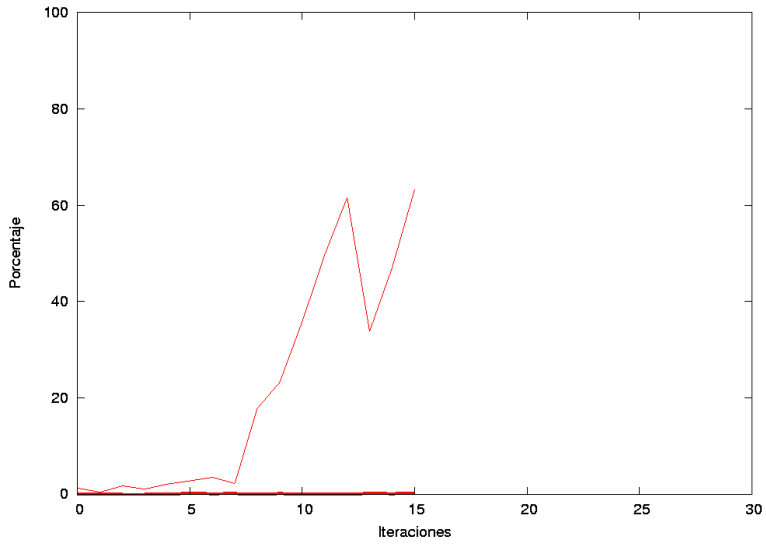

Figura 44: Efecto del modelo de observaciones: posiciones verosímiles

Las Figuras 43 y 44 se obtienen por superposición de los resultados obtenidos en las ejecuciones de los Ejemplos 2 y 4. Podemos observar que el conjunto muestral converge en x e y en ambos ejemplos, pero tiene dificultad en el ángulo en el Ejemplo 4 siendo el porcentaje de poses verosímiles muy reducido (inferior al 1 %). El impacto de la función elegida en el número de muestras necesarias queda demostrada al ser necesario utilizar 10,000 muestras en este caso para obtener resultados relativamente aceptables (con menos muestras el robot no consigue localizarse, es decir, el porcentaje de muestras verosímiles es 0 %). El número de poses verosímiles también puede incrementarse aumentando el error de movimiento pero entonces la precisión conseguida se ve aún más deteriorada, es decir, e^{-d^2} no es un buen modelo para nuestro entorno. Se realizaron experimentos similares con las funciones e^{-d} y e^{-d^3} pero tampoco se consiguió localizar al robot satisfactoriamente; como se mencionó anteriormente, la única función que ha permitido la localización del robot ha sido $e^{-d^2/256}$.

4.3.5. Ejemplo 5 - Efecto del número de balizas

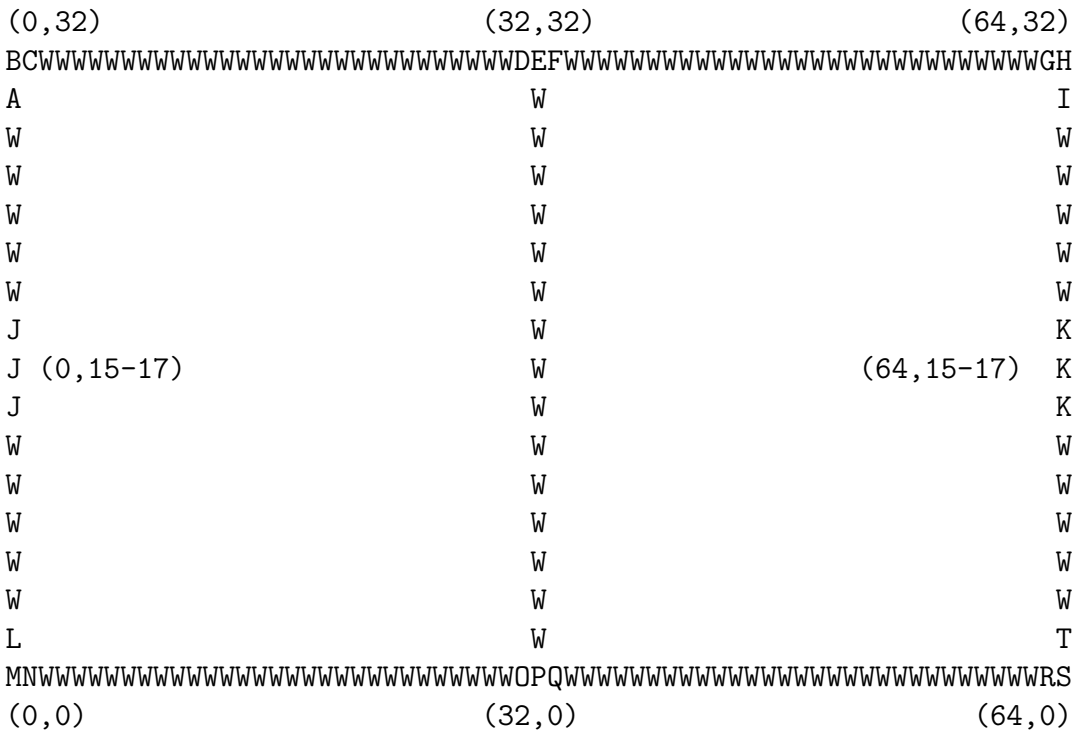

Figura 45: Mapa para estudiar el efecto del número de balizas (con muestreo)

En este ejemplo vamos a estudiar el efecto en el proceso de localización del número de balizas en el mapa proporcionado al robot para lo que se comparará la ejecución sobre el Mapa de la Figura 45 (24 balizas de diferentes colores excepto las de cada portería), con el fichero histórico del Cuadro 8, con 2,000 muestras, función distancia $e^{-d^2/256}$ para el modelo probabilístico de observaciones y error de movimiento del 1.8 %, con la Ejecución típica (24 balizas vs. 12 balizas).

Actuación No.	Acción	Posición	Imagen
0	——	50,16,135	D(31)E(34)F(36)
1	0,0,-90	50,16,45	G(29)H(33)I(36)
2	0,0,-45	50,16,0	M(32)N(40)O(47)
3	-10,0,0	40,16,0	M(35)N(40)O(44)
4	0,-8,225	40,8,225	S(45)T(40)U(33)
5	-10,0,45	30,8,270	S(27)T(15)U(3)
6	-11,0,-45	19,8,225	Q(79)R(77)
7	0,8,0	19,16,225	R(45)Q(48)P(51)
8	0,0,-45	19,16,180	J(45)K(39)L(34)
9	0,0,-45	19,16,135	A(28)B(31)C(34)
10	0,8,90	19,24,225	K(79)L(74)
11	-9,0,0	10,24,225	J(57)K(51)L(45)
12	-6,0,0	4,24,225	J(12)K(7)L(2)
13	0,0,-90	4,24,135	A(67)B(72)
14	6,0,0	10,24,135	A(22)B(28)C(34)
15	6,0,0	16,24,135	A(2)B(7)C(9)

Cuadro 8: Fichero histórico para efecto del número de balizas

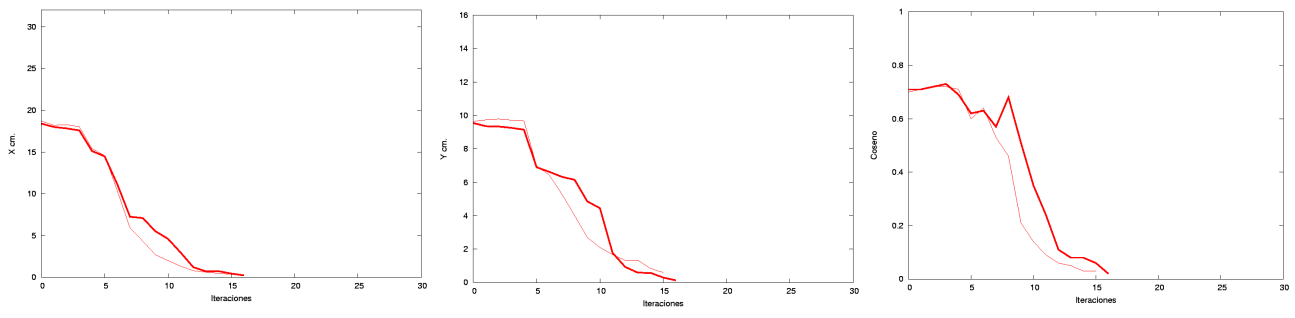

Figura 46: Efecto del número de balizas: desviación típica x, y y $cos(\theta)$

Las Figuras 46 y 47 se obtienen por superposición de los resultados obtenidos en las ejecuciones de los Ejemplos 1 y 5 (12 balizas en trazo fino y

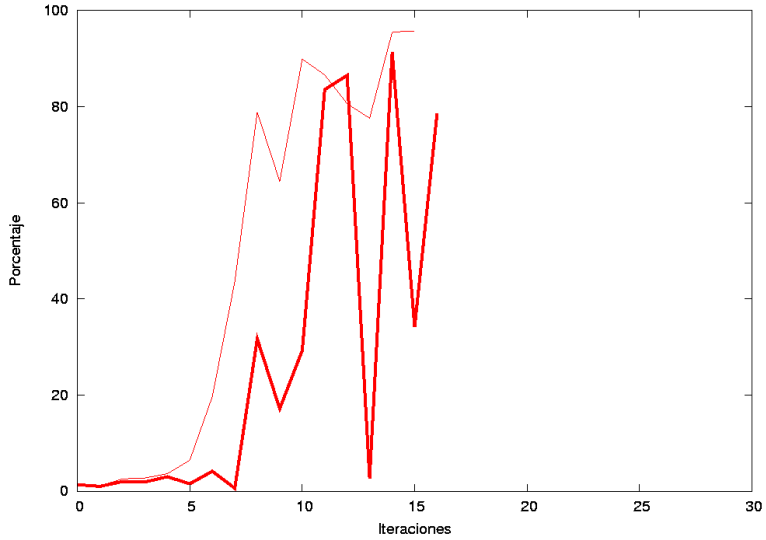

Figura 47: Efecto del número de balizas: posiciones verosímiles

24 balizas en trazo grueso, respectivamente). Podemos observar que el conjunto muestral converge de forma similar en x, y y $cos(\theta)$ y aunque existen fluctuaciones en el número de posiciones verosímiles, la localización se estabiliza a medida que aumenta el número de iteraciones obteniendo resultados similares sobre ambos mapas.

Por tanto, no obtenemos mejores resultados con más balizas. Hemos comprobado, no obstante, que si reducimos su número, el robot tiene mayores dificultades para localizarse y no lo consigue en el 100 % de los casos.

4.3.6. Ejemplo 6 - Efecto del tamaño del campo

En este ejemplo vamos a estudiar el efecto del tamaño del campo en el proceso de localización para lo que se comparará la ejecución sobre el Mapa de la Figura 17 al que se le ha duplicado el tamaño (aquí $66cmx130cm$), con el fichero histórico del Cuadro 9, con 2,000 muestras, función distancia $e^{-d^2/256}$ para el modelo probabilístico de observaciones y error de movimiento del 1.8 % con la Ejecución típica. Se cuadruplica el tamaño del campo y se duplica el alcance visual de la cámara para que todos los parámetros sean comparables con los de la Ejecución típica.

Las Figuras 48 y 49 se obtienen por superposición de los resultados obtenidos en las ejecuciones de los Ejemplos 1 y 6 (mapa de $33x65$ en trazo fino y mapa de $66x130$ en trazo grueso, respectivamente). Podemos observar que las desviaciones típicas en x y y son mucho mayores inicialmente en

Actuación No.	Acción	Posición	Imagen
0	——	100,32,135	B(34)
1	0,0,-90	100,32,45	C(33)
2	0,0,-45	100,32,0	E(32)E(40)E(47)
3	-20,0,0	80,32,0	E(35)E(40)E(44)
4	0,-16,225	80,16,225	G(40)
5	-20,0,45	60,16,270	G(15)
6	-22,0,-45	38,16,225	F(79)
7	0,16,0	38,32,225	F(48)
8	0,0,-45	38,32,180	D(45)D(39)D(34)
9	0,0,-45	38,32,135	A(31)
10	0,16,90	38,48,225	D(79)D(74)
11	-18,0,0	20,48,225	D(57)D(51)D(45)
12	-12,0,0	8,48,225	D(12)D(7)D(2)
13	0,0,-90	8,48,135	A(72)
14	12,0,0	20,48,135	A(28)
15	12,0,0	32,48,135	A(7)

Cuadro 9: Fichero histórico para efecto del tamaño del mapa

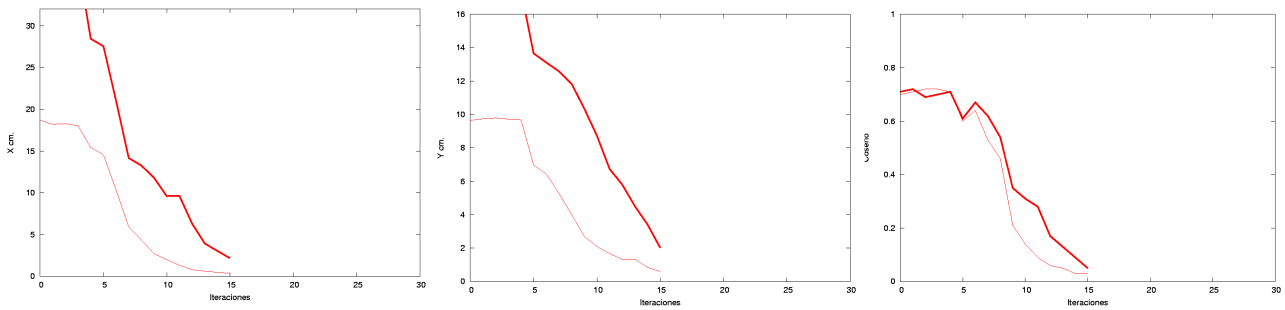

Figura 48: Efecto del tamaño del campo: desviación típica x, y y $cos(\theta)$

el Ejemplo 6, pero que al igual que en los ejemplos anteriores se reducen a medida que aumenta el número de iteraciones. Esto es lógico puesto que el campo es mayor y las muestras se distribuyen en un espacio más amplio. De hecho, podemos observar que ese aumento inicial no se produce en el caso del ángulo ya que éste mantiene sus dimensiones.

El número de posiciones verosímiles es menor y por ello se producen algunos "picos" como se explicó en el Ejemplo 1. Para este caso, se puede mejorar el número de poses verosímiles aumentando el error de movimiento lo cual repercute negativamente en la precisión conseguida.

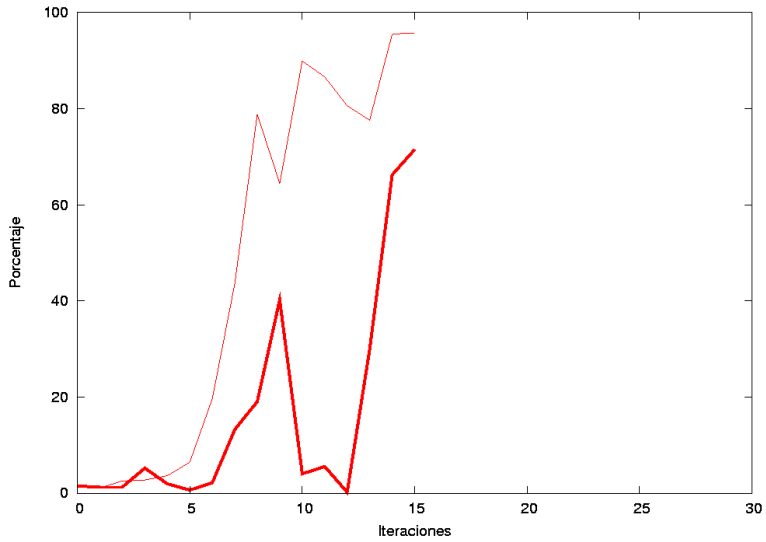

Figura 49: Efecto del tamaño del campo: posiciones verosímiles

Los resultados obtenidos en mapas de diferentes tamaños son proporcionalmente similares en el mismo número de iteraciones, a pesar de la mayor dispersión inicial de las muestras en campos mayores.

4.3.7. Ejemplo 7 - Efecto de la semilla

Nuestro objetivo es conseguir que el robot se localice sin conocer su posición inicial. No obstante, se realizaron pruebas en las que se incluía en el conjunto inicial una muestra cuyas coordenadas coincidían exactamente con la posición real del robot para comparar dichas ejecuciones con otras idénticas en las que todas las muestras eran aleatorias. Como en el Ejemplo 1, este ejemplo se realiza sobre el Mapa de la Figura 17, con el fichero histórico del Cuadro 7, con 2,000 muestras, función distancia $e^{-d^2/256}$ para el modelo probabilístico de observaciones y error de movimiento del 1.8 % (se fuerza una muestra con la posición inicial vs. se desconoce la posición inicial).

Las Figuras 50 y 51 se obtienen por superposición de los resultados obtenidos en las ejecuciones de los Ejemplos 1 y 7 (se desconoce la posición inicial en trazo fino y se fuerza una muestra con la posición inicial en trazo grueso, respectivamente). En ambos casos, las muestras convergen hacia la posición real del robot y puede observarse que las ejecuciones son bastante similares.

Aunque en algunas de las pruebas realizadas se obtuvieron resultados ligeramente mejores en cuanto al número de posiciones verosímiles cuando se

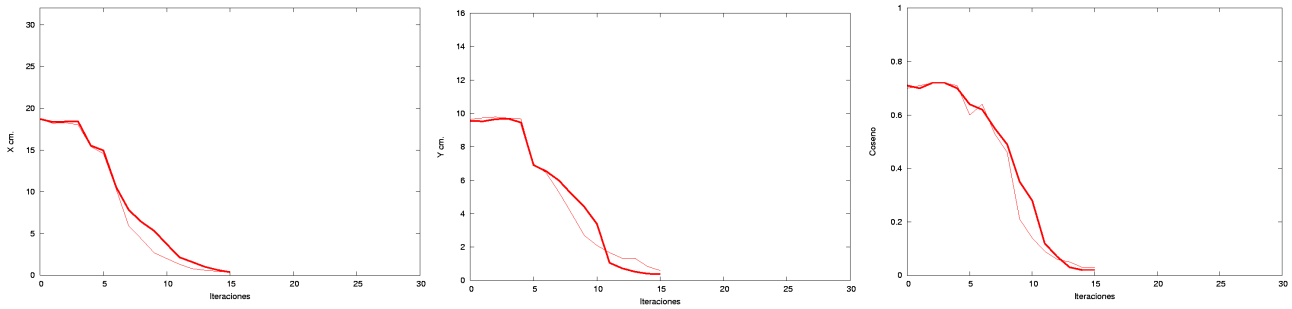

Figura 50: Efecto de la semilla: desviación típica x, y y $cos(\theta)$

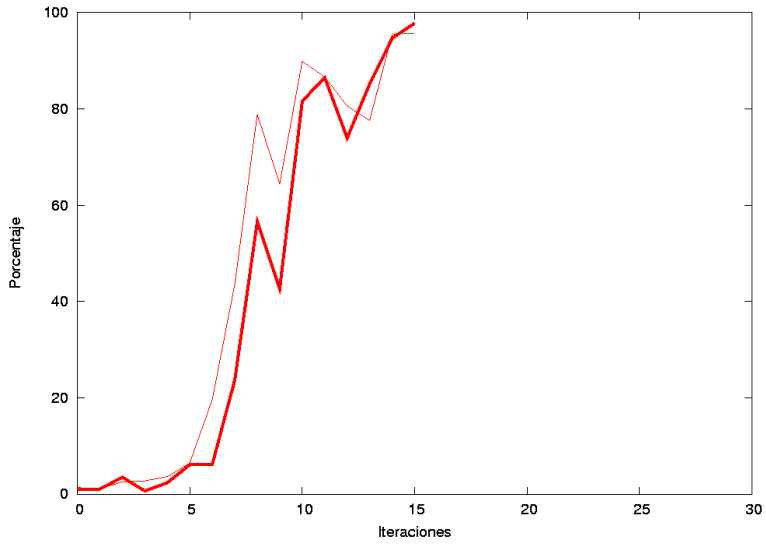

Figura 51: Efecto de la semilla: posiciones verosímiles

incluía la posición real como semilla, el rendimiento del algoritmo no depende de que se incluya la posición real en la población inicial; el robot se localiza aunque desconozca la posición inicial como se pretendía.

4.3.8. Ejemplo 8 - Efecto del ruido sensorial y de actuación

En este ejemplo vamos a estudiar el efecto del ruido sensorial y de actuación en el proceso de localización. Para ello se utilizó el Mapa de la Figura 17, con el fichero histórico del Cuadro 7, con 2,000 muestras, función distancia $e^{-d^2/256}$ para el modelo probabilístico de observaciones y errores de movimiento del 1.8 % y del 15 %.

En primer lugar se estudió el efecto del ruido de actuación. El Cuadro

10 muestra la media de las desviaciones típicas obtenidas en x e y así como la media del porcentaje de posiciones verosímiles para las ejecuciones realizadas sin ruido y con ruido de actuación de 10 % a 70 % utilizando error de movimiento del 1.8 %. Del mismo modo, el Cuadro 11 muestra la misma información utilizando error de movimiento del 15 %.

Ruido	x cm	y cm	% veros.
0 %	1.9	1.8	44.3
10 %	2.3	1.8	41.6
30 %	—	—	—-
60 %	—	—	—-
70 %	—	—	—-

Cuadro 10: Muestreo, efecto ruido de actuación para error de mvto 1.8 %

Ruido	x cm	y cm	% veros.
0 %	1.7	2.4	58.1
10 %	1.4	1.7	42.1
30 %	2.6	2.3	27.3
60 %	3.9	3.0	15.3
70 %	—	—	—-

Cuadro 11: Muestreo, efecto ruido de actuación para error de mvto 15 %

Los mejores resultados se obtienen con ruido de actuación del 10 % cuando el error de movimiento es del 15 %. Con ruido de actuación del 30 % y error de movimiento del 1.8 %, el robot ya no es capaz de localizarse; sí lo hace con error de movimiento del 15 % pero el numero de posiciones verosímiles en la última iteración (15) es sólo del 27.3 %. Por tanto, al aumentar el ruido de actuación, disminuye la precisión conseguida y el número de posiciones verosímiles hasta un cierto punto a partir del cual la localización no es posible. Como sucedía en los estudios sin ruido, el algoritmo se comporta de modo más estable para error de movimiento del 15 % que para el 1.8 % aunque ello suponga una disminución de la precisión conseguida en la localización.

Después se realizaron pruebas que sólo incluían ruido sensorial. Como se explicó en el apartado de *Implementación* del capítulo 3, el ruido sensorial se simula modificando la imagen *real*, bien desplazando las balizas ($P_{desp.}$) o bien

$P_{desp.}$	x cm	y cm	% veros.
0.0	1.7	2.4	58.1
0.1	1.8	2.1	55.0
0.5	2.0	2.3	38.0
0.9	2.9	2.5	16.6

Cuadro 12: Muestreo, efecto del ruido sensorial $P_{desp.}$

$P_{mut.}$	x cm	y cm	% veros.
0.0	1.7	2.4	58.1
0.0025	2.3	2.9	44.3
0.0050	3.1	3.6	16.8
0.0100	4.5	3.8	16.0

Cuadro 13: Muestreo, efecto del ruido sensorial $Pmut$.

eliminando/añadiendo balizas ($P_{mut.}$). En este caso, se realizaron pruebas variando $P_{desp.}$ como se muestra en el Cuadro 12. Se puede observar que tanto la precisión como el porcentaje de posiciones verosímiles disminuyen a medida que aumenta $P_{desp.}$. Los desplazamientos de la imagen no afectan tanto a la estimación como el hecho de eliminar o añadir balizas que suelen ser el principal motivo por el que el robot no es capaz de discriminar cuál es su posición real entre todas las probables. El Cuadro 13 muestra ejemplos de ruido de mutación; para $P_{mut.} > 0{,}01$ el robot no es capaz de localizarse en la mayoría de los casos.

Ruido Sens.	Ruido Desp.	x cm	y cm	% veros.
0.0 0.0	0 %	1.7	2.4	58.1
0.1 0.0025	10 %	2.9	2.4	29.2
0.1 0.0050	10 %	3.8	2.8	18.1
0.2 0.0025	10 %	3.8	3.0	32.8
0.2 0.0050	10 %	3.9	3.4	14.5

Cuadro 14: Muestreo, efecto ruido sensorial y de actuación

Finalmente, se realizaron pruebas que incluían *ruido sensorial y de actuación* conjuntamente. El cuadro 14 muestra ejemplos de ejecuciones típicas para el 10 % de ruido de actuación, $P_{desp.} = (0{,}1$ y $0{,}2)$ y $P_{mut.} = (0{,}0025$ y $0{,}0050)$. La tendencia es la misma que cuando se aplican los ruidos por

separado, a mayor ruido menor precisión y menor porcentaje de posiciones verosímiles. Los falsos positivos y negativos siguen siendo los que más dificultan el proceso de localización: en este caso, con $P_{mut.} = 0{,}0050$ el robot se pierde a veces y por encima de esa probabilidad el algoritmo es muy inestable.

Como ocurría con el método probabilístico sin muestreo, dado que el ruido gaussiano de actuación así como el ruido probabilístico sensorial son aleatorios, su efecto en el proceso de localización se estudia como la relación entre valores medios de error de localización y de porcentaje de posiciones verosímiles y no entre valores puntuales. En los experimentos realizados, se ha observado bastante tolerancia al ruido de actuación y al ruido sensorial de desplazamiento de la imagen siendo el ruido de mutación el que mayores problemas causa. En general, se confirma la intuición: cuanto mayor es el ruido, menor es la precisión y el porcentaje de poses verosímiles.

5. Conclusiones y trabajos futuros

Los objetivos de este proyecto se clasificaron en tres áreas: (1) creación de un *entorno simulado*, (2) implementación y estudio de la *localización probabilística sin muestreo* e (3) implementación y estudio de la *localización probabilística con muestreo*. A continuación se presentan las conclusiones del trabajo realizado agrupadas de igual forma.

Se ha diseñado un entorno simulado para probar el funcionamiento de los algoritmos probabilísticos desarrollados en este trabajo. Dicho entorno permite variar de modo muy sencillo los principales parámetros, lo cual ha sido muy útil para estudiar su efecto en el comportamiento de los algoritmos localizadores. El fichero con el mapa del entorno (que contiene las coordenadas de las líneas del campo y las de las balizas que corresponda como se vió en el capítulo 3) y el fichero con la secuencia de acciones a ejecutar por el robot (desplazamientos en x e y y rotaciones en θ) se proporcionan como entrada al simulador y son fácilmente editables para su modificación. Por otra parte, las dimensiones del mapa y las características del modelo sensorial (alcance y campo visual) son constantes definidas en el programa y cuyo valor es también fácilmente modificable. Finalmente, el ruido sensorial y de actuación se proporcionan como parámetros de entrada al invocar el programa y, por tanto, son variables para cada ejecución sin recompilación alguna.

El entorno simulado consta de varias partes significativas:

1. Se ha codificado un modelo sensorial de características similares a la cámara local del EyeBot para facilitar su portabilidad a este robot utilizando filtros de color. Sobre este modelo se permite la aplicación de ruido de observación para emular posibles desplazamientos aleatorios de las balizas y falsos positivos / falsos negativos incluyendo balizas inexistentes u ocultando las existentes.

2. Se ha desarrollado un modelo de actuación para simular los desplazamientos y traslaciones que permiten los motores del robot. Como en el modelo sensorial, también se incorpora la aplicación de ruido de actuación gaussiano.

3. Se ha automatizado la obtención de un fichero histórico que almacena las evidencias indirectas con las que contará el algoritmo localizador para encontrar la posición real del robot. Este fichero contiene la secuencia de acciones ejecutadas por el robot y las observaciones obtenidas desde las posiciones resultantes de aplicar dichas acciones (aplicando los

correspondientes ruidos de actuación y sensorial). Sirve para comparar de modo justo el rendimiento de los algoritmos localizadores con y sin muestreo pues ambos parten exactamente de los mismos datos.

Se ha implementado con éxito un algoritmo probabilístico sin muestreo que utiliza la regla de Bayes para acumular las evidencias aportadas por los modelos sensorial y de actuación y así localizar al robot.

Se ha comprobado que este algoritmo es bastante robusto y poco sensible al modelo probabilístico de observaciones, permitiendo la localización del robot con sólo 2-4 iteraciones. Su rendimiento computacional depende de las dimensiones del campo y del número de balizas del mismo: para el mapa más parecido al terreno de juego de la Robocup con dimensiones de $33x65$ cm son necesarios entre 24 y 48 segundos para localizar al robot utilizando un PC con procesador Pentium III a 1,133 Mhz, lo que le hace no escalable a entornos mayores ni ejecutable en el robot EyeBot dado su pequeño procesador. Por otra parte, este algoritmo requiere la discretización del mapa utilizado con lo que se limita la precisión al tamaño de la "celdilla" (en nuestro caso 1cm).

Además de este resultado típico, se ha estudiado el efecto de varios parámetros en el comportamiento de este algoritmo:

1. Se ha verificado el efecto de la simetría observándose que el algoritmo puede discriminar mejor la posición del robot cuando no existe simetría en las posiciones y colores de las balizas. En un escenario asimétrico, el número de posiciones verosímiles se reduce considerablemente, aunque el robot es capaz de localizarse con y sin simetría en el mismo número de iteraciones.

2. Se ha comprobado el efecto de la reducción del número de balizas y se verifica la intuición: el algoritmo también es capaz de localizar el robot aunque es necesario un mayor número de iteraciones para ello.

3. Se ha estudiado el efecto del ruido y se observa bastante tolerancia tanto al ruido de actuación como al ruido sensorial. Por separado, tolera sin perderse niveles de ruido del 60 % en actuación, $P_{mut} <= 0{,}4$ y cualquier $P_{desp.}$. Cuando se combinan ruidos sensorial y de actuación, los umbrales de tolerancia descienden al 30 % en cuanto a ruido de actuación y $P_{desp.} <= 0{,}2$ y $P_{mut} <= 0{,}001$ de ruido sensorial, con errores máximos de 3cm en x e y y $5\,^{\circ}$ en θ. La mutación causa más problemas al proceso de localización que el desplazamiento. También se observa que la precisión en cada instante varía ligeramente para cada

iteración dependiendo de lo discriminativa que sea la última imagen (cómo sea de informativa) más que de la historia acumulada hasta el momento.

En resumen, como indica [Thrun00a], la técnica probabilística sin muestreo ha demostrado ser muy adecuada para la localización incluso desconociendo la posición inicial del robot. Tolera incertidumbre en acciones y observaciones lo cual concuerda con la incertidumbre de los sensores y actuadores reales. Además, permite representar situaciones ambiguas y resolverlas (por ejemplo cuando existe simetría) dada su robustez. Su principal desventaja es que la eficiencia temporal depende del tamaño del mapa en el que se realiza la localización lo que hace que no sea escalable a grandes entornos.

Finalmente, se han implementado con éxito las tres fases (predicción, actualización y remuestreo) del algoritmo CONDENSATION que utiliza técnicas de muestreo para localizar al robot. Este algoritmo parte de una distribución uniforme de muestras en el campo de fútbol, es decir, sin conocimiento de la posición inicial del robot. En este caso, la localización es un fenómeno aleatorio por lo que la comprobación del funcionamiento del algoritmo se ha basado en múltiples pruebas con diferentes conjuntos iniciales de muestras.

Se ha comprobado que son necesarias entre 13 y 16 iteraciones (frente a las 2-4 requeridas por la implementación sin muestreo) para localizar al robot. No obstante, su rendimiento computacional es mucho mejor, consiguiendo la localización en 4 segundos donde la implementación sin muestreo requería entre 24 y 48 segundos. Además, este algoritmo ofrece resultados contínuos, en contraste con el universo discretizado que exige el probabilístico sin muestreo. De este modo, permite potencialmente una mayor precisión en los resultados si se realiza un adecuado ajuste de parámetros (número de muestras, error de movimiento, modelo probabilístico de observación, etc...). Puesto que no hemos superado con muestreo la precisión de celdilla conseguida sin muestreo, discretizar no supone en nuestro caso ninguna desventaja.

Además de este resultado típico, se ha estudiado el efecto de varios parámetros en el comportamiento de este algoritmo:

1. En cada iteración se ha incorporado un pequeño desplazamiento aleatorio para evitar el bloqueo del algoritmo si dentro del conjunto muestral no hay ninguna muestra con coordenadas lo suficientemente próximas a la posición del robot. Se ha estudiado cómo afecta este *error de movimiento* al comportamiento del algoritmo: a medida que aumenta, disminuye la precisión conseguida en la localización, pero aumenta la estabilidad del algoritmo, lo hace más robusto.

2. Se ha estudiado el comportamiento del algoritmo en función del modelo probabilístico de observaciones y se ha apreciado que es mucho más sensible al modelo que el algoritmo probabilístico sin muestreo. Sólo se ha conseguido localizar al robot de manera fiable con la función $e^{-d^2/256}$, que es mucho más suave que la utilizada en la implementación sin muestreo (e^{d^2}).

3. Se ha estudiado también el efecto del número de muestras y se ha observado que un mayor número no ayuda necesariamente a la convergencia ni a la obtención de mayor precisión de los resultados, aunque por debajo de un determinado umbral (500 muestras en nuestro entorno) no se consigue localizar al robot. Lo mismo sucede con el número de balizas, más balizas no ayudan necesariamente, pero el robot no consigue localizarse, o lo hace con mayor dificultad, por debajo de un determinado umbral (12 balizas). Es importante, por tanto, buscar un balance adecuado ya que estos aumentos conllevan mayor tiempo de cómputo pero no siempre mejores resultados.

4. También se estudió el efecto del aumento del tamaño del mapa, observándose que a pesar de la mayor dispersión inicial de las muestras en campos mayores, se obtienen unos resultados proporcionalmente similares con el mismo número de iteraciones.

5. Se comprobó que el robot es capaz de localizarse con igual precisión y con el mismo número de iteraciones se incluya o no en la población inicial una muestra de coordenadas iguales a la posición real del robot.

6. Finalmente, se estudió el efecto del ruido. Por separado, tolera niveles de ruido de actuación del 10 %, ruido sensorial de probabilidad 0.2 en desplazamiento y de probabilidad 0.01 en falsos positivos y negativos. Cuando se combinan ruido sensorial y de actuación, el algoritmo tolera menos ruido que por separado. En general, se confirma la intuición: cuanto mayor es el ruido, menor es la precisión en la población final y menor el porcentaje de poses verosímiles. La mayor estabilidad del algoritmo se obtuvo con "error de movimiento" del 15 %, ya que este parámetro ayuda a compensar el efecto del ruido sensorial y de actuación.

La técnica de localización probabilística con muestreo ha resultado ser eficaz para superar los inconvenientes de la técnica sin muestreo. Por un lado, permite agilizar los cómputos reduciendo el número de posiciones candidatas a un número limitado de muestras representativas. Esto evita la necesidad

de almacenar y actualizar todas las posibles posiciones y lo hace escalable a grandes entornos. Por otro lado, permite la utilización de un entorno continuo. Su principal desventaja es la necesidad de realizar un ajuste muy fino de todos los parámetros para que el algoritmo se comporte de manera robusta.

Queda abierto a trabajos futuros la implementación del método probabilístico para la localización en los robots reales, por ejemplo el EyeBot o el Aibo en el campo de la Robocup. Aunque este proyecto se ha realizado en un entorno simulado, se han tenido en cuenta las caracterísitcas de los robots reales para facilitar su futura implementación en ellos. Una posible referencia en esta línea son los trabajos de [Buschka00] utilizando lógica borrosa. Hay que recordar que no será posible implementar el método sin muestreo en el robot debido al excesivo consumo de CPU que requiere.

Otro aspecto a estudiar es la reducción progresiva del error de movimiento. Puesto que el algoritmo con muestreo más robusto se obtiene con un 15 % de error, podría reducirse progresivamente (cada x iteraciones) de tal forma que se consiga mayor precisión en los resultados finales una vez asegurada la convergencia de grano grueso.

Queda también por explorar el funcionamiento de estos algoritmos en un entorno en el que existen más robots y obstáculos ya que no se han tenido en cuenta en este trabajo. También pueden utilizarse otros sensores, no sólo visión, para estimar indirectamente la posición del robot en cierto entorno, como por ejemplo el láser, la potencia recibida en las tarjetas de red inalámbricas [Serrano03], etc...

Referencias

[Buschka00] Buschka, P., Saffiotti, A., Wasik, Z.:
Fuzzy Landmark-Based Localization for a Legged Robot.
Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS-2000) Takamatsu, Japan, 2000. pp1205-1210.

[Brandimarte] Brandimarte, P.:
Numerical Methods in Finance.
John Willey and Sons ISBN 0-471-39686-9.

[Cañas01] Cañas, J.M., Simmons Reid, García-Alegre, M.C.:
Detección probabilística de puertas con visión monocular activa.
Proceedings of II Workshop hispano-luso de Agentes Físicos WAF-2001, Universidad Rey Juan Carlos, March 2001.

[Cañas02] Cañas, J.M., García, L.:
Construcción de mapas dinámicos: comparativa.
Proceedings of III Workshop de Agentes Físicos WAF-2002, Universidad de Murcia, March 2002.

[Cañas03] Cañas, J.M., García, E., Matellán, V.:
Manual del Robot EyeBot, versión 2.5.
http://gsyc.escet.urjc.es/robotica/manualeyebot/manualeyebot.html Universidad Rey Juan Carlos, Enero 2003.

[Dellaert99] Dellaert, F., Burgard, W., Fox, D., Thrun, S.:
Using the condensation algorithm for robust, vision-based mobile robot localization.
In Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR) 1999.

[Fox98] Fox, D., Burgard, W.:
Active Markov Localization for Mobile Robots.
Robotics and Autonomous Systems, 1998.

[Fox99a] Fox, D., Burgard, W., Dellaert, F., Thrun, S.:
MCL Localization: Efficient position estimation for mobile robots.
In Proc. of the National Conference on Artificial Intelligence (AAAI) 1999.

[Fox99b] Fox, D., Burgard, W., Thrun, S.:
Markov Localization for Mobile Robots in Dynamic Environments.
Journal of Artifitial Intellingence Research 11 (1999) 391-427.

[Fox00] Fox, D., Thrun, S., Burgard, W., Dellaert, F.:
Particle Filters for Mobile Robot Localization.
In A. Doucet, N. de Freitas and N. Gordon, editors, Sequential Monte Carlo Methods in Practice.
Springer Verlag, New York 2000.

[Gallardo98] Gallardo, D., Escolano, F., Rizo, R., Colomina, O., Cazorla, M.A.:
Estimación bayesiana de características móviles mediante muestreo de la densidad a posteriori.
I Congrés Català d'Intel.ligència Artificial. Tarragona. Octubre de 1998.

[González96] González, J., Ollero, A.:
Estimación de la posición de un Robot Móvil.
Informática y Automática Vol. 29-4/1996.

[Gutmann98] Gutmann, J-S., Burgard, W., Fox, D., Konolife, K.:
An experimental comparison of localization methods.
Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'98).

[Gutmann02] Gutmman, J-S., Fox, D.:
An Experimental Comparison of Localization Methods Continued.
To appear in IEEE/RSJ International Conference on Inteligent Robots and Systems (IROS) 2002.

[Isard98] Isard, M., Blake, A.:
CONDENSATION - conditional density propagation for visual tracking.
Int. J. Computer Vision, in press (1998).

[Leonard90] Leonard, J., Cox, I.J.:
Dynamic Map Building for an Autonomous Mobile Robot.
1990 IEE.

[Liu] Liu, J.S.:
Monte Carlo Strategies in Scientific Computing.
Springer-Verlag ISBN 0-387-95230-6.

[Mackay96] Mackay, D.J.C.:
Introduction to Monte Carlo Methods.
Proceedings of a 1996 summer school, ed. M Jordan.

[Montemerlo02] Montemerlo, M., Thrun, S., Whittaker, W.:
Conditional Particle Filters for Simultaneous Mobile Robot Localization and People-Tracking.
IEEE International Conference on Robotics and Automation (ICRA) 2002.

[Robert] Robert, C.P., Casella, G. :
Monte Carlo Statistical Methods.
Springer-Verlag ISBN 0-387-98707-X.

[Sáez02] Sáez, J.M., Escolano, F.:
Localización global en mapas 3D basada en filtros de partículas.
III Workshop de Agentes Físicos, Murcia Marzo 2002.

[Serrano03] Serrano, O., Matellán, V., Cañas, J.M., Rodero, L.:
Robot Localization using wireless networks.
Informe Técnico en elaboración Departamento de Informática, Estadística y Telemática, 2003.
Universidad Rey Juan Carlos (Móstoles, Spain).

[Simmons95] Simmons, R., Koeing, S.:
Probabilistic Navigation in Partially Observable Environments.
Proceedings of the International Joint Conference on Artificial Intelligence. (IJCA,'95). July 1995.

[Thrun97] Thrun, S.:
Bayesian Landmark Learning for Mobile Robot Localization.
To appear in Machine Learning.
April 1997.

[Thrun00a] Thrun, S.:
Probabilistic Algorithms in Robotics.
AI Magazine, 21(4):93-109.
April 2000.

[Thrun00b] Thrun, S., Fox, D., Burgard, W.:
Monte Carlo Localization With Mixture Proposal Distribution.
Proceedings of the AAAI National Conference on Artificial Intelligence 2000.

[Thrun01] Thrun, S., Fox, D., Burgard, W., Dellaert, F.:
Robust Montecarlo Localization for Mobile Robots.
Artificial Intelligence Journal, 2001.

[Welch02] Welch, G., Bishop, G.:
An Introduction to the Kalman Filter.
UNC-Chapel Hill, TR 95-041, March 11, 2002.

A. Listados de programas

A.1. Simulador

A.1.1. Versión 3.5

```
/*                                                                    */
/* montecarlo-3.5-tgz                                                 */
/*                                                                    */
/* localiza_balizas.c  Uso 'localiza_balizas [nombre_mapa] [fichero_acciones] */
/*                         [ruido_traslacion][ruido_rotacion]         */
/*                         [prob_img_desp][prob_baliza]               */
/* Por defecto, nombre_mapa = "mi_mapa", fichero_acciones = "mi_fichero",     */
/* ruido_traslacion=0, ruido_rotacion=0, prob_img_desp=0 y prob_baliza=0      */
/*                                                                    */
/* Lee un mapa de FILAS*COLUMNAS y lo carga en un buffer, identificando líneas*/
/* y balizas. Se admiten un máximo de MAX_LINEAS líneas y MAX_BALIZAS balizas.*/
/* El fichero tiene dos tipos de registros:                           */
/*    Lxi,yi,xf,yf                                                    */
/*    Carácter L + coordenadas iniciales y finales para una línea     */
/*    Bx,y,C                                                          */
/*    Carácter B + coordenadas de la baliza + color (1 letra mayúscula)       */
/* Recibe como entrada la posición y orientación de la cámara y comprueba las */
/* balizas que se ven desde esa posición.                             */
/*                                                                    */
/* Identifica el campo visual dentro del mapa leido y el pixel en el que se   */
/* localiza la baliza visible a la cámara.                            */
/* Los parámetros de la cámara vienen dados por ALCANCE_C, CAMPO_VISUAL y     */
/* PIXELS_CAMARA (visión en una dimensión).                           */
/* Genera un fichero .pgm con el CAMPO_VISUAL dentro del mapa de entrada para */
/* la posición inicial.                                               */
/*                                                                    */
/* Lee un fichero con una posición inicial y una serie de acciones (MAX_MVTOS */
/* como máximo) de desplazamiento en "x", "y" y theta" y genera un fichero de */
/* salida con las acciones aplicadas y las imágenes obtenidas tras los        */
/* movimientos aplicados.                                             */
/* El fichero tiene dos tipos de registro:                            */
/*   Px,y,theta  Carácter P+posición inicial de la cámara             */
/*   Ax,y,theta  Carácter A+coordenadas y ángulo de desplazamiento    */
/*               Admite valores positivos y negativos pero descartará acciones*/
/*               que impliquen salirse del mapa                       */
/* Genera un fichero .pgm con el CAMPO_VISUAL dentro del mapa de entrada para */
/* cada una de las posiciones válidas resultantes.                    */
```

```
/* Se eliminan las limitaciones en el número de balizas contenidas en el mapa */
/* y en el número de acciones permitidas. Para ello se hace una lectura previa*/
/* de los ficheros de entrada, contabilizando las balizas y las acciones a    */
/* leer y se asigna memoria dinámicamente a los arrays que acumulan esta      */
/* información.                                                               */
/* Se incorpora ruido gaussiano de actuación y ruido en las imágenes en los   */
/* parámetros de entrada:                                                     */
/*  ruido_traslacion: porcentaje de ruido en x e y                            */
/*  ruido_rotacion: porcentaje de ruido en theta                              */
/*  prob_img_desp: prob. de que las balizas se desplacen de pixel en la imagen*/
/*  prob_baliza: prob. de que una baliza aparezca o desaparezca de un pixel   */
/*                                                                            */

#include <stdio.h>
#include <math.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <time.h>

#define FILAS         33
#define COLUMNAS      65
#define COORD_MAX     64
#define MAX_LINEAS     5
#define PIXELS_CAMARA 80
#define ALCANCE_C     25
#define CAMPO_VISUAL  45
#define PI             3.14159265
#define DEGTORAD     (PI / 180.0)
#define RADTODEG     (180.0 / PI)
#define RANGO_GAUSS 2001

/* Definición de estructuras de datos para balizas y posición de cámara */

typedef struct coordenada{
  double x;
  double y;
}Scoordenada;

typedef struct baliza{
  Scoordenada xy;
  char color;
}Sbaliza;
```

```
typedef struct posicion_camara{
  Scoordenada xy;
  double theta;
}Sposicion_camara;

typedef struct registro_salida{
  Sposicion_camara posicion_sal;
  char linea_sal[PIXELS_CAMARA+1];
}Sregistro_salida;


typedef struct ruido{
  double d;
  double gauss_d;
}Sruido;

/* Variables globales   */

FILE *mapa, *acciones;
char fin_fichero_m, fin_fichero_a;
char buffer_m[COLUMNAS][FILAS];
char linea_c[PIXELS_CAMARA+1];
int cbalizas, clineas, cacciones;
int ruido_t, ruido_r;
double prob_img_desp, prob_baliza;
Sregistro_salida o_file;
Sposicion_camara i_camara, *i_file, *npos_valida;
Sbaliza *balizas_m;
Sruido array_ruido_xy[RANGO_GAUSS], gaussiana_acumulada_xy[RANGO_GAUSS];
Sruido array_ruido_theta[RANGO_GAUSS], gaussiana_acumulada_theta[RANGO_GAUSS];

int inicializa_mapa() {

  int i, j;

  for (i=0; i<FILAS; i++)
    for (j=0; j<COLUMNAS; j++)
      buffer_m[j][i] = ' ';

  cbalizas = 0;

  return 0;
```

```
}

int lee_1coordenada(FILE *fichero){

  int i = 0, j, coordenada = 0;
  int posiciones[COORD_MAX];
  char aux, signo;
  char *p_aux;

  signo = '+';
  aux = fgetc(fichero);

  p_aux = &aux;
  while ((aux != '\n') && (aux != ',') && (aux != EOF)) {
    if (isdigit(aux)) {
      posiciones[i] = atoi(p_aux);
      i++;
      aux = fgetc(fichero);
    }
    else {
      if (aux == '-') {
signo = '-';
aux = fgetc(fichero);
      }
      else {
printf("LEE COORDENADA: Valor inesperado %c\n", aux);
exit(-1);
      }
    }
  }

  for (j=0; j<(i-1); j++)
    coordenada = coordenada + (posiciones[j]*pow(10,(i-j-1)));
  coordenada = coordenada + posiciones[(i-1)];
  if (signo == '-') coordenada = coordenada * (-1);

  return coordenada;
}

int salta_linea() {

  int xi, yi, xf, yf;
```

```
  xi = lee_1coordenada(mapa);
  yi = lee_1coordenada(mapa);
  xf = lee_1coordenada(mapa);
  yf = lee_1coordenada(mapa);

  if ((xi >= COLUMNAS) || (xf >= COLUMNAS ) || (yi >= FILAS) || (yf >= FILAS) ||
      (xi < 0) || (xf < 0 ) || (yi < 0) || (yf < 0)) {
    printf("SALTA LINEA: Coordenadas invalidas %d %d %d %d\n", xi, xf, yi, yf);
    exit(-1);
  }

  return 0;
}


int salta_baliza() {

  int x, y;
  char aux;

  x = lee_1coordenada(mapa);
  y = lee_1coordenada(mapa);

  if ((x >= COLUMNAS) || (y >= FILAS) || (x < 0 ) || (y < 0)) {
     printf("SALTA BALIZA: Coordenadas invalidas %d %d\n", x, y);
     exit(-1);
  }

  aux = fgetc(mapa);

  if (!(isupper(aux))) {
    printf ("SALTA BALIZA: Valor inesperado %c\n", aux);
    exit(-1);
  }

  return 0;
}

int cuenta_balizas() {

  char que_leo;

  fin_fichero_m = ' ';
```

```
  while (fin_fichero_m != 'F') {
    que_leo = fgetc(mapa);
    if (que_leo == EOF) fin_fichero_m = 'F';
    if (que_leo == 'L')
      salta_linea();
    if (que_leo == 'B') {
      cbalizas ++;
      salta_baliza();
    }
    if (que_leo == '\n');
    if ((que_leo != EOF) && (que_leo != 'L') &&
(que_leo != 'B') && (que_leo != '\n')) {
      printf("CUENTA BALIZAS: No puedo interpretar contenido del fichero %c\n",
que_leo);
      exit(-1);
    }
  }

  balizas_m = (Sbaliza *)malloc(sizeof(Sbaliza)*cbalizas);
  if (balizas_m == NULL) {
    printf("CUENTA BALIZAS: No hay memoria disponible para mapa de balizas\n");
    exit(-1);
  }

  return 0;
}

int lee_linea() {

  int xi, yi, xf, yf, x, y;

  xi = lee_1coordenada(mapa);
  yi = lee_1coordenada(mapa);
  xf = lee_1coordenada(mapa);
  yf = lee_1coordenada(mapa);

  if (xi == xf) {
    for(y=yi; y>=yf; y--){
      buffer_m[xi][y] = 'W';
    }
  }
```

```
  if (yi == yf) {
    for (x=xi; x<=xf; x++) {
      buffer_m[x][yi] = 'W';
    }
  }

  return 0;
}


int lee_baliza() {

  int x, y;
  char aux;

  x = lee_1coordenada(mapa); balizas_m[cbalizas-1].xy.x = x;
  y = lee_1coordenada(mapa); balizas_m[cbalizas-1].xy.y = y;

  aux = fgetc(mapa);

  buffer_m[x][y] = aux;
  balizas_m[cbalizas-1].color = aux;

  return 0;
}


int lee_mapa() {

  char que_leo;

  fin_fichero_m = ' ';

  while (fin_fichero_m != 'F') {
    que_leo = fgetc(mapa);
    if (que_leo == EOF) fin_fichero_m = 'F';
    if (que_leo == 'L') {
      clineas ++;
      if (clineas <= MAX_LINEAS)
lee_linea();
      else {
printf ("LEE MAPA: Excedido el maximo numero de lineas %d\n",MAX_LINEAS);
exit(-1);
```

```
      }
    }
    if (que_leo == 'B') {
      cbalizas++;
      lee_baliza();
    }
  }
  return 0;
}


int imprime_mapa() {

  int i, j;

  for (i=(FILAS-1); i>=0; i--){
    for (j=0; j<COLUMNAS; j++) printf("%c", buffer_m[j][i]);
    printf(" \n");
  }

  return 0;
}

int salta_posicion_inicial() {

  int x, y, theta;

  x = lee_1coordenada(acciones);
  y = lee_1coordenada(acciones);
  theta = lee_1coordenada(acciones);

  return 0;
}


int salta_1accion() {

  int x, y, theta;

  x = lee_1coordenada(acciones);
  y = lee_1coordenada(acciones);
  theta = lee_1coordenada(acciones);
```

```
  return 0;
}

int cuenta_acciones() {

  char que_leo;

  fin_fichero_a = ' ';

  while (fin_fichero_a != 'F') {
    que_leo = fgetc(acciones);
    if (que_leo == EOF) fin_fichero_a = 'F';
    if (que_leo == 'P') salta_posicion_inicial();
    if (que_leo == 'A') {
      cacciones++;
      salta_1accion();
    }
    if (que_leo == '\n');
    if ((que_leo != EOF) && (que_leo != 'P') && (que_leo != 'A') &&
(que_leo != '\n')) {
      printf("CUENTA ACCIONES: No puedo interpretar fichero de entrada %c\n",
que_leo);
      exit(-1);
    }
  }

  i_file = (Sposicion_camara *)malloc(sizeof(Sposicion_camara)*cacciones);
  if (i_file == NULL) {
    printf("CUENTA ACCIONES: No hay memoria disponible para i_file\n");
    exit(-1);
  }

  npos_valida = (Sposicion_camara *)malloc(sizeof(Sposicion_camara)*cacciones);
  if (npos_valida == NULL) {
    printf("CUENTA ACCIONES: No hay memoria disponible para npos_valida\n");
    exit(-1);
  }

  return 0;
}

int calcula_gaussiana_acumulada(int i) {
```

```
 gaussiana_acumulada_xy[i].d = array_ruido_xy[i].d;
 gaussiana_acumulada_theta[i].d = array_ruido_theta[i].d;

  if (i==0) {
    gaussiana_acumulada_xy[i].gauss_d = array_ruido_xy[i].gauss_d;
    gaussiana_acumulada_theta[i].gauss_d = array_ruido_theta[i].gauss_d;
  }
  else {
    gaussiana_acumulada_xy[i].gauss_d =
gaussiana_acumulada_xy[i-1].gauss_d+array_ruido_xy[i].gauss_d;
    gaussiana_acumulada_theta[i].gauss_d =
gaussiana_acumulada_theta[i-1].gauss_d+array_ruido_theta[i].gauss_d;
  }

  return 0;

}


double calcula_ruido(char dimension) {

  double temp;
  int k;

  if (dimension == 'C') {
    temp=((double)(rand()))/((double)(RAND_MAX+1.0));
    k=(int)(temp*RANGO_GAUSS-1);
    return gaussiana_acumulada_xy[k].d;
  }
  else {
    temp=((double)(rand()))/((double)(RAND_MAX+1.0));
    k=(int)(temp*RANGO_GAUSS-1);
    return gaussiana_acumulada_theta[k].d;
  }

  return 0;

}

int calcula_gaussianas() {

  double media_xy, desv_xy, extremos_xy;
  double media_theta, desv_theta, extremos_theta;
```

```
  int i;

  media_xy = 0;
  if (ruido_t == 0)
    /* Se utiliza en el ruido sensorial si no hay ruido de actuación */
    desv_xy = 0.1;
  else
    desv_xy = (double)(ruido_t)/100.00;
  extremos_xy = 3.0*desv_xy;

  media_theta = 0;
  if (ruido_r == 0)
    desv_theta = 0;
  else
    desv_theta = (double)(ruido_r)/100.00;
  extremos_theta = 3.0*desv_theta;

  for (i=0;i<RANGO_GAUSS;i++) {
    array_ruido_xy[i].d =
((extremos_xy+extremos_xy)*i/(RANGO_GAUSS-1)) - extremos_xy;
    array_ruido_xy[i].gauss_d = (1.00/(desv_xy*sqrt(2.00*PI)))*
      (exp(-(1.00/2.00)*(((array_ruido_xy[i].d-media_xy)/desv_xy)*
      ((array_ruido_xy[i].d-media_xy)/desv_xy))));
    array_ruido_theta[i].d = ((extremos_theta+extremos_theta)*i/(RANGO_GAUSS-1))
 - extremos_theta;
    array_ruido_theta[i].gauss_d = (1.00/(desv_theta*sqrt(2.00*PI)))*
      (exp(-(1.00/2.00)*(((array_ruido_theta[i].d-media_theta)/desv_theta)*
((array_ruido_theta[i].d-media_theta)/desv_theta))));
    calcula_gaussiana_acumulada(i);
  }

  return 0;

}

int inicializa_ifile() {

  int i;

  for (i=0; i<cacciones; i++) {
    i_file[i].xy.x = 0;
    i_file[i].xy.y = 0;
    i_file[i].theta = 0;
```

```
  }

  cacciones = 0;

  return 0;
}

int lee_posicion_inicial() {

  i_camara.xy.x = lee_1coordenada(acciones);
  i_camara.xy.y = lee_1coordenada(acciones);
  i_camara.theta = lee_1coordenada(acciones);

  return 0;
}

int verifica_accion_leida() {

  double nx, ny, ntheta;
  double ruido_x, ruido_y, ruido_theta;

  if (cacciones == 1) {
    nx = i_camara.xy.x + i_file[0].xy.x;
    ny = i_camara.xy.y + i_file[0].xy.y;
    ntheta = i_camara.theta  + i_file[0].theta;
  }
  else {
    nx = npos_valida[cacciones-2].xy.x + i_file[cacciones-1].xy.x;
    ny = npos_valida[cacciones-2].xy.y + i_file[cacciones-1].xy.y;
    ntheta = npos_valida[cacciones-2].theta  + i_file[cacciones-1].theta;
  }

  if (ruido_t != 0) {
    ruido_x = calcula_ruido('C');
    ruido_y = calcula_ruido('C');
  }
  else {
    ruido_x = 0;
    ruido_y = 0;
  }

  if (ruido_r != 0)
    ruido_theta = calcula_ruido('A');
```

```
  else
    ruido_theta = 0;

  nx = nx + ruido_x;
  ny = ny + ruido_y;
  ntheta = ntheta + ruido_theta;

  if (ntheta>360) ntheta = ntheta - 360;
  if (ntheta<0) ntheta = ntheta + 360;

  printf("VERIFICA ACCION LEIDA: cacciones = %d, x=%.2f, y=%.2f, theta= %.2f\n",
 cacciones, nx, ny, ntheta);

  if ((nx>(COLUMNAS-1)) || (ny>(FILAS-1)) || (nx<0) || (ny<0)) {
    printf("\n-- VERIFICA ACCION LEIDA: Accion %d invalida, x=%.2f e y=%.2f
fuera del mapa. Descartada.\n", cacciones, nx, ny);
    cacciones--;
  }
  else {
    npos_valida[cacciones-1].xy.x = nx;
    npos_valida[cacciones-1].xy.y = ny;
    npos_valida[cacciones-1].theta = ntheta;
  }

  return 0;
}

int lee_1accion() {
  int x, y, theta;

  x = lee_1coordenada(acciones); i_file[cacciones-1].xy.x = x;
  y = lee_1coordenada(acciones); i_file[cacciones-1].xy.y = y;
  theta = lee_1coordenada(acciones); i_file[cacciones-1].theta = theta;

  verifica_accion_leida();

  return 0;

}


int lee_ifile() {
```

```
  char que_leo;

  fin_fichero_a = ' ';

  while (fin_fichero_a != 'F') {
    que_leo = fgetc(acciones);
    if (que_leo == EOF) fin_fichero_a = 'F';
    if (que_leo == 'P') lee_posicion_inicial();
    if (que_leo == 'A') {
      cacciones++;
      lee_1accion();
    }
  }

  return 0;
}

int imprime_nuevas_posiciones() {

  int i;

  printf("\n Posicion inicial: x=%.2f, y=%.2f, theta=%.2f \n\n",
 i_camara.xy.x, i_camara.xy.y, i_camara.theta);

  for (i=0; i<cacciones; i++)
    printf(" Accion=%d: x=%.2f, y=%.2f, theta=%.2f \n",
   i, i_file[i].xy.x, i_file[i].xy.y, i_file[i].theta);
  printf("\n");

  for (i=0; i<cacciones; i++)
    printf(" Posicion=%d: x=%.2f, y=%.2f, theta=%.2f \n",
   i, npos_valida[i].xy.x, npos_valida[i].xy.y, npos_valida[i].theta);
  printf("\n");

  return 0;
}

int inicializa_linea_camara() {

  int i;

  for (i=0; i<PIXELS_CAMARA; i++) {
    linea_c[i] = ' ' ;
```

```
  }

  linea_c[i] = '\0';

  return 0;
}

int comprueba_si_ve_baliza(int i) {

  /* Función que calcula si una baliza es visible o no a la cámara. Se tratan  */
  /* las excepciones que se dan cuando el ángulo 0 está dentro del campo visual*/

  double db, thetab, cthetar, cthetamascvmedios, cthetamenoscvmedios;
  int bx, by, bcolor, cx, cy, pixel_baliza, cthetag;

  bx = balizas_m[i].xy.x;
  by = balizas_m[i].xy.y;
  bcolor = balizas_m[i].color;

  cx = i_camara.xy.x;
  cy = i_camara.xy.y;
  cthetag = i_camara.theta;

  cthetar = cthetag * DEGTORAD;

  cthetamascvmedios = cthetar + (CAMPO_VISUAL * DEGTORAD / 2);
  cthetamenoscvmedios = cthetar - (CAMPO_VISUAL * DEGTORAD / 2);

  /* Aquí se eliminan valores menores que 0 o mayores que 360 para los */
  /* ángulos comprendidos en el campo visual                           */
  if (cthetamascvmedios > (2*PI)) cthetamascvmedios -= (2*PI);
  if (cthetamenoscvmedios < 0) cthetamenoscvmedios += (2*PI);

  db = sqrt(((by - cy)*(by - cy)) + ((bx - cx)*(bx - cx)));

  /* Aquí se eliminan valores menores que 0 para el ángulo de la baliza */
  /* a detectar y el pixel de la baliza                                 */
  thetab = atan2((by - cy),(bx - cx));
  if (thetab < 0) thetab += (2*PI);

  /* Cálculo del pixel: Cuando thetab  > cthetamascvmedios, tenemos 0 dentro */
  /* del campo visual y por tanto discontinuidad en los angulos              */
  if (thetab  > cthetamascvmedios)
```

```
    pixel_baliza =
PIXELS_CAMARA-(PIXELS_CAMARA*(thetab-cthetamenoscvmedios)/
(CAMPO_VISUAL*DEGTORAD));
  else
    pixel_baliza =
PIXELS_CAMARA*(cthetamascvmedios-thetab)/(CAMPO_VISUAL*DEGTORAD);

  printf("COMPRUEBA SI VE BALIZA: bx = %d, by = %d, bcolor = %c\n",
bx, by, bcolor);
  printf("COMPRUEBA SI VE BALIZA: cx = %d, cy = %d, ctheta = %d\n",
cx, cy, cthetag);
  printf("COMPRUEBA SI VE BALIZA: db = %.8f, thetab = %.8f\n", db, (thetab*RADTODEG));
  printf("COMPRUEBA SI VE BALIZA: cthetamenoscvmedios = %.8f,
cthetamascvmedios = %.8f\n",
(cthetamenoscvmedios*RADTODEG), (cthetamascvmedios*RADTODEG));


  if ((db <= ALCANCE_C) && (db > 0) &&
      (((cthetamenoscvmedios <= cthetamascvmedios) &&
(cthetamenoscvmedios <= thetab) && (thetab <= cthetamascvmedios)) ||
       ((cthetamascvmedios <= cthetamenoscvmedios) &&
((thetab <= cthetamascvmedios) || (cthetamenoscvmedios <= thetab))))) {
      printf("** COMPRUEBA SI VE BALIZA: Ve baliza %d color = %c, pixel = %d\n",
 i, bcolor, pixel_baliza);
      linea_c[pixel_baliza] = bcolor;
  }
    else printf("COMPRUEBA SI VE BALIZA: No ve baliza %d \n", i);


  return 0;
}

int dibuja_vista_camara(int nfile) {

  double db, thetab, cthetar, cthetamascvmedios, cthetamenoscvmedios;
  int bx, by, cx, cy, cthetag, fichero;
  char header[70], nombre[50];
  unsigned char grey;

  cx = i_camara.xy.x;
  cy = i_camara.xy.y;
  cthetag = i_camara.theta;
```

```
  cthetar = cthetag * DEGTORAD;

  cthetamascvmedios = cthetar + (CAMPO_VISUAL * DEGTORAD / 2);
  cthetamenoscvmedios = cthetar - (CAMPO_VISUAL * DEGTORAD / 2);

  if (cthetamascvmedios > (2*PI)) cthetamascvmedios -= (2*PI);
  if (cthetamenoscvmedios < 0) cthetamenoscvmedios += (2*PI);

  /* Dibuja el campo visual dentro del mapa en un fichero .pgm. Para ello    */
  /* calcula para cada posición del buffer si es visible a la cámara en cuyo */
  /* caso lo pinta en blanco. Las posiciones no visibles aparecen en negro   */

  sprintf(nombre, "salida/vista_camara%d.pgm", nfile);
  fichero = open(nombre, O_CREAT | O_WRONLY, S_IRUSR | S_IWUSR);

  if ( fichero < 0) {
    printf ("DIBUJA VISTA CAMARA: No puedo abrir el fichero %s\n", &nombre[0]);
    exit (-1);
  }

  sprintf(header, "P5\n%d\n%d\n255\n", COLUMNAS, FILAS);
  write(fichero,&header[0],strlen(header));

  for (by=(FILAS-1); by>=0; by--)
    for (bx=0; bx<COLUMNAS; bx++) {
      db = sqrt(((by - cy)*(by - cy)) + ((bx - cx)*(bx - cx)));
      thetab = atan2((by - cy),(bx - cx));
      if (thetab < 0) thetab = thetab + (2*PI);
      if ((db <= ALCANCE_C) && (db > 0) &&
  (((cthetamenoscvmedios <= cthetamascvmedios) &&
  (cthetamenoscvmedios <= thetab) && (thetab <= cthetamascvmedios)) ||
  ((cthetamascvmedios <= cthetamenoscvmedios) &&
  ((thetab <= cthetamascvmedios) || (cthetamenoscvmedios <= thetab))))) {
grey = (unsigned char)255;
write(fichero,&grey,sizeof(unsigned char));
}
      else {
grey = (unsigned char)0;
write(fichero,&grey,sizeof(unsigned char));
      }
    }

  close(fichero);
```

```
return 0;
}

int ruido_a_imagen() {

  int i, npixels;
  double desp, temp;

  /*printf("Linea de entrada a ruido a imagen: ");
  for (i=0; i<80; i++) printf("%c", linea_c[i]);
  printf("\n"); */

  /* Si el número aleatorio calculado es menor que la probabilidad solicitada,
     desplazo la imagen */
  temp=((double)(rand()))/((double)(RAND_MAX+1.0));

  if (temp < prob_img_desp) {

    /* La imagen se desplaza aleatoriamente entre 1 y 10 pixels */
    npixels=(int)(temp*10);
    if (npixels == 0) npixels = 1;

    /* Esto es sólo para decidir si desplazo la imagen hacia la
       izquierda o hacia la derecha */
    desp = calcula_ruido('C');

    if (desp <= 0.0) {
      for (i=(PIXELS_CAMARA-npixels); i>=npixels; i--)
linea_c[i] = linea_c[i-npixels];
      for (i=(PIXELS_CAMARA-1);i>(PIXELS_CAMARA-npixels);i--)
linea_c[i] = ' ';
      for (i=(npixels-1);i>=0;i--) linea_c[i] = ' ';
    }
    if (desp > 0.0) {
      for (i=npixels; i<(PIXELS_CAMARA-npixels); i++)
linea_c[i] = linea_c[i+npixels];
      for (i=0;i<npixels;i++) linea_c[i] = ' ';
      for (i=(PIXELS_CAMARA-npixels);i<PIXELS_CAMARA;i++)
linea_c[i] = ' ';
    }
  }
```

```
  /* printf("Linea despues de desplazar imagen: ");
  for (i=0; i<80; i++) printf("%c", linea_c[i]);
  printf("\n");  */

  for (i=0; i<PIXELS_CAMARA; i++) {
    temp=((double)(rand()))/((double)(RAND_MAX+1.0));
    if (temp < prob_baliza) {
      if (linea_c[i] == ' ') linea_c[i] = 'X';
      else linea_c[i] = ' ';
    }
  }

  /* printf("Linea despues de poner/quitar balizas: ");
  for (i=0; i<80; i++) printf("%c", linea_c[i]);
  printf("\n"); */

   return 0;
}

int prepara_y_escribe_salida() {

  int i, j, k, fichero_salida;
  char linea_registro[100];

  fichero_salida = open("salida/accion_e_imagen.txt",
O_CREAT | O_WRONLY, S_IRUSR | S_IWUSR);
  if (fichero_salida < 0) {
    printf("PREPARA Y ESCRIBE SALIDA: No puedo abrir fichero posicion_e_imagen\n");
    exit (-1);
  }

  inicializa_linea_camara();
  for (i=0; i<cbalizas; i++) comprueba_si_ve_baliza(i);

  o_file.posicion_sal.xy.x = i_camara.xy.x;
  o_file.posicion_sal.xy.y = i_camara.xy.y;
  o_file.posicion_sal.theta = i_camara.theta;

  if ((prob_img_desp>0.00)||(prob_baliza>0.00)) ruido_a_imagen();

  for (k=0; k<=80; k++) o_file.linea_sal[k] = '\0';
  strncpy(o_file.linea_sal, &linea_c[0], 80);
  for (k=0; k<100; k++) linea_registro[k] = '\0';
```

```
  printf("\n== Registro Salida %.2f %.2f %.2f %s\n\n", o_file.posicion_sal.xy.x,
 o_file.posicion_sal.xy.y, o_file.posicion_sal.theta,
 &o_file.linea_sal[0]);
  sprintf(linea_registro, "P%d,%d,%d,%s", (int)(o_file.posicion_sal.xy.x),
(int)(o_file.posicion_sal.xy.y), (int)(o_file.posicion_sal.theta),
&o_file.linea_sal[0]);
  write(fichero_salida, &linea_registro[0], strlen(linea_registro));
  dibuja_vista_camara(0);

  for (j=0; j<cacciones; j++) {
    inicializa_linea_camara();
    i_camara.xy.x = npos_valida[j].xy.x;
    i_camara.xy.y = npos_valida[j].xy.y;
    i_camara.theta = npos_valida[j].theta;
    for (i=0; i<cbalizas; i++) comprueba_si_ve_baliza(i);

    o_file.posicion_sal.xy.x = i_file[j].xy.x;
    o_file.posicion_sal.xy.y = i_file[j].xy.y;
    o_file.posicion_sal.theta = i_file[j].theta;

    if ((prob_img_desp>0.00)||(prob_baliza>0.00)) ruido_a_imagen();

    for (k=0; k<=80; k++) o_file.linea_sal[k] = '\0';
    strncpy(o_file.linea_sal, &linea_c[0], 80);
    for (k=0; k<100; k++) linea_registro[k] = '\0';

    printf("\n== Registro Salida %.2f %.2f %.2f %s\n\n",
o_file.posicion_sal.xy.x, o_file.posicion_sal.xy.y,
o_file.posicion_sal.theta, &o_file.linea_sal[0]);
    sprintf(linea_registro, "\nA%d,%d,%d,%s", (int)(o_file.posicion_sal.xy.x),
 (int)(o_file.posicion_sal.xy.y), (int)(o_file.posicion_sal.theta),
 &o_file.linea_sal[0]);
    write(fichero_salida, &linea_registro[0], strlen(linea_registro));
    dibuja_vista_camara(j+1);
  }

  sprintf(linea_registro, "\n#%d,%d,%d", (int)(npos_valida[cacciones-1].xy.x),
 (int)(npos_valida[cacciones-1].xy.y),
 (int)(npos_valida[cacciones-1].theta));
  write(fichero_salida, &linea_registro[0], strlen(linea_registro));
  return 0;
```

```
}

/* Programa principal */


int main(int argc, char ** argv) {

  char *fichero_mapa, *fichero_acciones;

  int semilla;
  time_t aclock;

  time(&aclock);
  semilla = aclock;
  srand(semilla);

  /* Toma variables de entrada, las verifica y asigna adecuadamente */

  if ((argc != 1) && (argc !=  7)) {
    printf ("MAIN: Numero de argumentos erroneo %d\n", argc);
    printf ("MAIN: Uso 'localiza_balizas [nombre_mapa] [fichero_acciones]
 [ruido_t] [ruido_r] [prob_img_desp] [prob_baliza]'\n");
    exit(-1);
  }

  if (argc == 1) {
    fichero_mapa = "mi_mapa";
    fichero_acciones = "mi_fichero";
    ruido_t = 0; ruido_r = 0; prob_img_desp = 0; prob_baliza = 0;
  }
  else {
    fichero_mapa = argv[1];
    fichero_acciones = argv[2];
    ruido_t = atoi(argv[3]);
    ruido_r = atoi(argv[4]);
    prob_img_desp = atof(argv[5]);
    prob_baliza = atof(argv[6]);
  }

  if ((ruido_t<0) || (ruido_t>100) || (ruido_r<0) || (ruido_r>100) ||
      (prob_img_desp<0) || (prob_img_desp>1) || (prob_baliza<0) ||
      (prob_baliza>1)) {
    printf ("MAIN Error parámetros ruido: traslacion=%d, rotacion =%d
```

```
 desp imagen = %.5f io_baliza = %.5f\n",
 ruido_t, ruido_r, prob_img_desp, prob_baliza);
    exit(-1);
  }
  else
    printf ("MAIN Ruido: traslacion=%d, rotacion =%d  desp imagen = %.5f
io_baliza = %.5f\n", ruido_t, ruido_r, prob_img_desp, prob_baliza);

   /* Variables de control del número de líneas y balizas del mapa de entrada */
  cbalizas = 0; clineas = 0;

  /* Abre mapa de entrada, inicializa el buffer, carga el mapa leido y crea */
  /* un fichero con el campo visual dentro del mapa                          */
  mapa = fopen(fichero_mapa, "r");

  if (mapa == NULL) {
    printf ("MAIN: No puedo abrir el mapa %s\n", fichero_mapa);
    exit (-1);
  }

  cuenta_balizas();
  if (fclose(mapa) != 0) {
    printf ("MAIN: No puedo cerrar el mapa\n");
    exit (-1);
  }

  mapa = fopen(fichero_mapa, "r");
  if (mapa == NULL) {
    printf ("MAIN: No puedo abrir el mapa por segunda vez\n");
    exit (-1);
  }
  inicializa_mapa();
  lee_mapa();
  imprime_mapa();


  /* Variable de control del número de movimientos del fichero de acciones */
  cacciones = 0;

  /* Abre fichero de entrada e inicializa array de posiciones */
  acciones = fopen(fichero_acciones, "r");

  if (acciones == NULL) {
```

```
    printf ("MAIN: No puedo abrir el fichero de acciones %s\n",
fichero_acciones);
    exit (-1);
  }

  cuenta_acciones();
  if (fclose(acciones) != 0) {
    printf ("MAIN: No puedo cerrar el fichero de acciones\n");
    exit (-1);
  }

  acciones = fopen(fichero_acciones, "r");
  if (acciones == NULL) {
    printf ("MAIN: No puedo abrir el fichero de acciones por segunda vez\n");
    exit (-1);
  }

  calcula_gaussianas();

  inicializa_ifile();
  lee_ifile();
  imprime_nuevas_posiciones();

  /* Comprueba qué valizas capta la cámara y en qué pixel para la posición */
  /* inicial y posteriores, creando un registro en el fichero de salida    */
  /* con las coordenadas x e y, el ángulo posteriores a la acción y la     */
  /* imagen obtenida                                                       */

  prepara_y_escribe_salida();

  fcloseall();

  return 0;
}
```

A.2. Localizador

A.2.1. Versión 3.5 - Probabilístico sin muestreo

```
/*                                                                    */
/* montecarlo-3.5-tgz                                                 */
/*                                                                    */
/*      Uso 'verosimilitud [nombre_mapa] [nombre_accion_e_imagen]'    */
/*      Por defecto:                                                  */
/*          nombre_mapa = "mi_mapa" y nombre_imagen = "accion_e_imagen.txt"  */
/*                                                                    */
/*      Lee un mapa de FILAS*COLUMNAS y lo carga en un buffer,        */
/*      identificando líneas y balizas.                               */
/*      Se admiten un máximo de MAX_LINEAS líneas.                    */
/*      El fichero tiene dos tipos de registros:                      */
/*              Lxi,yi,xf,yf                                          */
/*              Bx,y,C                                                */
/*              Carácter B + coordenadas de la baliza + color         */
/*                                                  (1 letra mayúscula)  */
/*                                                                    */
/*      Lee una imagen inicial de PIXELS_CAMARA columnas y las compara con  */
/* las calculadas teóricamente, generando ficheros con las imágenes (en     */
/* grises) de las probabilidades de que la cámara esté posicionada en las   */
/* posiciones disponibles dentro del campo.                          */
/* Se generan 36 ficheros de nombre verosimilitud*.pgm donde 0 <= * < 360.  */
/* Cada uno de ellos contempla todas las posibles coordenadas x e y del campo */
/* pero un único ángulo de 0 a 360 en intervalos de 10 grados.       */
/*                                                                    */
/*      A continuación lee una acción y su imagen asociada y la compara con  */
/* la calculada teóricamente partiendo del cálculo realizado con la imagen  */
/* inicial y la acción indicada.                                     */
/*      Se generan cubos de probabilidades (para las coordenadas y ángulos  */
/* posibles en el mapa de entrada) y se van combinando para  llegar a un cubo */
/* final que debería reducir las localizaciones posibles con la información  */
/* acumulada.                                                        */
/*      El fichero accion_e_imagen.txt ha sido generado por el programa     */
/* localiza_baliza y parte de una imagen inicial en la que no se proporcionan */
/* las coordenadas y una serie de acciones e imágenes resultantes, en el    */
/* siguiente formato:                                                */
/*              Px,y,theta,imagen[PIXELS_CAMARA]                     */
/*              Posición inicial e imagen real                       */
/*              Ax,y,theta,imagen[PIXELS CAMARA]                     */
/*              Acciones sucesivas e imagenes generadas después de aplicarlas */
/* El programa localiza_baliza ya ha verificado que las acciones indicadas son*/
```

```
/* posibles dentro del mapa (habiéndose rechazado otras que no lo eran).      */
/* Se generaran 36 pgm ficheros como los indicados anteriormente para cada una*/
/* de "las verosimilitudes" resultantes de aplicar las acciones y combinar los*/
/* cubos de probabilidades.                                                   */
/* Además se crea un fichero .ppm y un fichero.dat con las "TOP 25"           */
/* probabilidades y un histograma con la curva de valores de probabilidades   */
/* tras cada acción, destacando el TOP1.                                      */
/* Se eliminan las limitaciones en el número de balizas contenidas en el mapa.*/
/* Para ello se hace una lectura previa del mapa y se contabiliza el número de*/
/* balizas contenidas en el mismo, asignándose dinámicamente memoria al array */
/* que acumula esta información.                                              */
/* Se mejoran las imágenes de los ficheros ppm incorporando la orientación    */
/* aproximada del robot. Para ello se representan 9 pixels por pixel.         */
/*                                                                            */

#include <stdio.h>
#include <math.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdlib.h>

#define FILAS         33
#define COLUMNAS      65
#define GRADOS_C     360
#define COORD_MAX     64
#define MAX_LINEAS     5
#define PIXELS_CAMARA 80
#define ALCANCE_C     25
#define CAMPO_VISUAL  45
#define PI             3.14159265
#define DEGTORAD     (PI / 180.0)
#define RADTODEG     (180.0 / PI)

/* Definición de estructuras de datos para balizas y posición de cámara */

typedef struct coordenada{
  double x;
  double y;
}Scoordenada;

typedef struct baliza{
  Scoordenada xy;
  char color;
```

```
}Sbaliza;

typedef struct posicion_camara{
  Scoordenada xy;
  double theta;
}Sposicion_camara;

typedef struct cubo_prob{
  double probabilidad[COLUMNAS][FILAS][GRADOS_C];
}Scubo_prob;

typedef struct top_prob{
  Sposicion_camara max_min_pos;
  double max_min_prob;
}Stop_prob;

/* Variables globales   */

FILE *mapa, *imagen;
char *fichero_mapa, *fichero_imagen;
char fin_mapa, fin_imagen;
char que_leo, imagen_en_blanco;
char buffer_m[COLUMNAS][FILAS];
char linea_c[PIXELS_CAMARA];
int cbalizas, clineas;
Sposicion_camara i_camara, accion;
char buffer_i[PIXELS_CAMARA];
Scubo_prob cubo_imagen0, cubo_imagen1, cubo_combinado;
int histograma_c[101];
Sbaliza *balizas_m;
double cubo_size;

int inicializa_mapa() {

  int i, j;

  for (i=0; i<FILAS; i++)
    for (j=0; j<COLUMNAS; j++)
      buffer_m[j][i] = ' ';

  cbalizas = 0;

  return 0;
```

```
}


int lee_1coordenada(FILE *fichero){

  int i = 0, j, coordenada = 0;
  int posiciones[COORD_MAX];
  char aux, signo;
  char *p_aux;

  signo = '+';
  aux = fgetc(fichero);
  p_aux = &aux;
  while ((aux != '\n') && (aux != ',') && (aux != EOF)) {
    if (isdigit(aux)) {
      posiciones[i] = atoi(p_aux);
      i++;
      aux = fgetc(fichero);
    }
    else {
      if (aux == '-') {
signo = '-';
aux = fgetc(fichero);
      }
      else {
printf("LEE COORDENADA: Valor inesperado %c\n", aux);
exit(-1);
      }
    }
  }

  for (j=0; j<(i-1); j++)
    coordenada = coordenada + (posiciones[j]*pow(10,(i-j-1)));
  coordenada = coordenada + posiciones[(i-1)];
  if (signo == '-') coordenada = coordenada * (-1);

  return coordenada;
}


int salta_linea() {

  int xi, yi, xf, yf;
```

```
  xi = lee_1coordenada(mapa);
  yi = lee_1coordenada(mapa);
  xf = lee_1coordenada(mapa);
  yf = lee_1coordenada(mapa);

  if ((xi >= COLUMNAS) || (xf >= COLUMNAS ) || (yi >= FILAS) || (yf >= FILAS) ||
      (xi < 0) || (xf < 0 ) || (yi < 0) || (yf < 0)) {
    printf("SALTA LINEA: Coordenadas invalidas %d %d %d %d\n", xi, xf, yi, yf);
    exit(-1);
  }

  return 0;
}


int salta_baliza() {

  int x, y;
  char aux;

  x = lee_1coordenada(mapa);
  y = lee_1coordenada(mapa);

  if ((x >= COLUMNAS) || (y >= FILAS) || (x < 0 ) || (y < 0)) {
     printf("SALTA BALIZA: Coordenadas invalidas %d %d\n", x, y);
     exit(-1);
  }

  aux = fgetc(mapa);

  if (!(isupper(aux))) {
    printf ("SALTA BALIZA: Valor inesperado %c\n", aux);
    exit(-1);
  }

  return 0;
}

int cuenta_balizas() {

  char que_leo;
```

```
  fin_mapa = ' ';

  while (fin_mapa != 'F') {
    que_leo = fgetc(mapa);
    if (que_leo == EOF) fin_mapa = 'F';
    if (que_leo == 'L')
      salta_linea();
    if (que_leo == 'B') {
      cbalizas ++;
      salta_baliza();
    }
    if (que_leo == '\n');
    if ((que_leo != EOF) && (que_leo != 'L') &&
(que_leo != 'B') && (que_leo != '\n')) {
      printf("CUENTA BALIZAS: No puedo interpretar contenido del fichero %c\n",
 que_leo);
      exit(-1);
    }
  }

  balizas_m = (Sbaliza *)malloc(sizeof(Sbaliza)*cbalizas);
  if (balizas_m == NULL) {
    printf("CUENTA BALIZAS: No hay memoria disponible para mapa de balizas\n");
    exit(-1);
  }

  return 0;
}

int lee_linea() {

  int xi, yi, xf, yf, x, y;

  xi = lee_1coordenada(mapa);
  yi = lee_1coordenada(mapa);
  xf = lee_1coordenada(mapa);
  yf = lee_1coordenada(mapa);

  if (xi == xf) {
    for(y=yi; y>=yf; y--){
      buffer_m[xi][y] = 'W';
    }
  }
```

```
  if (yi == yf) {
    for (x=xi; x<=xf; x++) {
      buffer_m[x][yi] = 'W';
    }
  }

  return 0;
}


int lee_baliza() {

  int x, y;
  char aux;

  x = lee_1coordenada(mapa); balizas_m[cbalizas-1].xy.x = x;
  y = lee_1coordenada(mapa); balizas_m[cbalizas-1].xy.y = y;

  aux = fgetc(mapa);

  buffer_m[x][y] = aux;
  balizas_m[cbalizas-1].color = aux;

  return 0;
}


int lee_mapa() {

  char que_leo;

  fin_mapa = ' ';

  while (fin_mapa != 'F') {
    que_leo = fgetc(mapa);
    if (que_leo == EOF) fin_mapa = 'F';
    if (que_leo == 'L') {
      clineas ++;
      if (clineas <= MAX_LINEAS)
lee_linea();
      else {
printf ("LEE MAPA: Excedido el maximo numero de lineas %d\n",MAX_LINEAS);
```

```
exit(-1);
      }
    }
    if (que_leo == 'B') {
      cbalizas++;
      lee_baliza();
    }
  }
  return 0;
}


int imprime_mapa() {

  int i, j;

  for (i=(FILAS-1); i>=0; i--){
    for (j=0; j<COLUMNAS; j++) printf("%c", buffer_m[j][i]);
    printf(" \n");
  }

  return 0;
}


int inicializa_cubo0(double unknown) {

  int i, j, k;

  for (i=(FILAS-1); i>=0; i--)
    for (j=0; j<COLUMNAS; j++)
      for (k=0; k<GRADOS_C; k++)
  cubo_imagen0.probabilidad[j][i][k]=unknown;

  return 0;
}


int inicializa_imagen() {

  int i;

  for (i=0; i<PIXELS_CAMARA; i++)
```

```
    buffer_i[i] = ' ';

  return 0;
}


int lee_imagen() {

  int i=0;
  char pixel;

  fin_imagen = ' ' ; imagen_en_blanco = 1;

  pixel = fgetc(imagen);
  if ((pixel == 'P') || (pixel == 'A')) {
    accion.xy.x = lee_1coordenada(imagen);
    accion.xy.y = lee_1coordenada(imagen);
    accion.theta = lee_1coordenada(imagen);
  }
  else {
    if (pixel != '#') {
      printf("LEE IMAGEN: Formato de imagen erroneo,
deberia ser posicion inicial 'P' o accion 'A', no %c\n", pixel);
      exit(-1);
    }
  }

  while ((pixel != '\n') && (pixel != EOF) && (pixel != '#')){
    pixel = fgetc(imagen);
    buffer_i[i] = pixel;
    if (pixel == ' ')
      imagen_en_blanco = 0;
    i++;
  }

  if ((i<PIXELS_CAMARA)&&(pixel!='#')) {
    printf("LEE IMAGEN: Formato de imagen erroneo,
solo %d pixels leidos (deben ser 80)\n", i);
    exit(-1);
  }

  if (pixel == EOF)  fin_imagen = 'F';
  if (pixel == '#') fin_imagen = 'f';
```

```
  return 0;
}


int desplaza_cubo_combinado() {

  /* Aplica acción al cubo combinado. Si existe desplazamiento en x, y y       */
  /* theta se realiza primero la traslación y luego la rotación, por lo tanto  */
  /* si se quiere realizar primero la rotación deberá especificarse mediante   */
  /* una acción de sólo rotación seguida de otra acción de traslación          */

  int i, j, k, x, y, theta;
  int desplaza_filas = 0, desplaza_columnas = 0, desplaza_angulos = 0;

  inicializa_cubo0(0.5/cubo_size);
  inicializa_histograma();

  printf("DESPLAZA CUBO COMBINADO: x=%.2f, y=%.2f, theta=%.2f \n",
 accion.xy.x, accion.xy.y, accion.theta);

  /* Las variables desplaza_* se utilizan para dar mayor legibilidad */
  /* y eficiencia al código */

  if (accion.xy.y != 0) desplaza_filas = 1;
  if (accion.xy.x != 0) desplaza_columnas = 1;
  if (accion.theta < 0)  accion.theta += GRADOS_C;
  if (accion.theta != 0) desplaza_angulos = 1;


  /* Desplaza filas y columnas. Si existe rotación, gira tras desplazarse. */

  if ((desplaza_filas == 1) && (desplaza_columnas == 1)) {
    if((accion.xy.y > 0) && (accion.xy.x > 0)) {
      printf("DESPLAZA FILAS Y COLUMNAS x e y>0  \n");
      for (y=accion.xy.y;y<FILAS;y++)
for (x=accion.xy.x; x<COLUMNAS; x++)
  for (theta=0; theta<GRADOS_C; theta++){
    i = y - (int)(accion.xy.y);
    j = x - (int)(accion.xy.x);
    k = theta - accion.theta;  if (k < 0) k+=GRADOS_C;
    cubo_imagen0.probabilidad[x][y][theta] =
      cubo_combinado.probabilidad[j][i][k];
```

```
  }
    }
    if((accion.xy.y > 0) && (accion.xy.x < 0)) {
      printf("DESPLAZA FILAS Y COLUMNAS x<0 e y>0  \n");
      for (y=accion.xy.y;y<FILAS;y++)
for (x=0; x<(COLUMNAS-(accion.xy.x*(-1))); x++)
          for (theta=0; theta<GRADOS_C; theta++){
    i = y - (int)(accion.xy.y);
    j = x - (int)(accion.xy.x);
    k = theta - accion.theta;  if (k < 0) k+=GRADOS_C;
    cubo_imagen0.probabilidad[x][y][theta] =
      cubo_combinado.probabilidad[j][i][k];
  }
    }
    if((accion.xy.y < 0) && (accion.xy.x > 0)) {
      printf("DESPLAZA FILAS Y COLUMNAS y<0 y x>0  \n");
      for (y=0;y<(FILAS-(accion.xy.y*(-1)));y++)
for (x=accion.xy.x; x<COLUMNAS; x++)
  for (theta=0; theta<GRADOS_C; theta++){
    i = y - (int)(accion.xy.y);
    j = x - (int)(accion.xy.x);
    k = theta - accion.theta;  if (k < 0) k+=GRADOS_C;
    cubo_imagen0.probabilidad[x][y][theta] =
      cubo_combinado.probabilidad[j][i][k];
  }
    }
    if((accion.xy.y < 0) && (accion.xy.x < 0)) {
      printf("DESPLAZA FILAS Y COLUMNAS x e y <0   \n");
      for (y=0;y<(FILAS-(accion.xy.y*(-1)));y++)
for (x=0; x<(COLUMNAS-(accion.xy.x*(-1))); x++)
  for (theta=0; theta<GRADOS_C; theta++){
    i = y - (int)(accion.xy.y);
    j = x - (int)(accion.xy.x);
    k = theta - accion.theta;  if (k < 0) k+=GRADOS_C;
    cubo_imagen0.probabilidad[x][y][theta] =
      cubo_combinado.probabilidad[j][i][k];
  }
    }
  }

  /* Desplaza sólo filas. Si existe rotación, gira tras desplazarse. */

  if ((desplaza_filas == 1) && (desplaza_columnas == 0)) {
```

```
    if (accion.xy.y > 0) {
      printf("DESPLAZA FILAS y>0 \n");
      for (y=accion.xy.y;y<FILAS;y++)
for (j=0; j<COLUMNAS; j++)
  for (theta=0; theta<GRADOS_C; theta++){
    k = theta - accion.theta; if (k < 0) k+=GRADOS_C;
    cubo_imagen0.probabilidad[j][y][theta] =
      cubo_combinado.probabilidad[j][y-(int)(accion.xy.y)][k];
  }
    }

    if (accion.xy.y < 0) {
      printf("DESPLAZA FILAS y<0 \n");
      for (y=0;y<(FILAS-(accion.xy.y*(-1)));y++)
for (j=0; j<COLUMNAS; j++)
  for (theta=0; theta<GRADOS_C; theta++) {
    k = theta - accion.theta; if (k < 0) k+=GRADOS_C;
    cubo_imagen0.probabilidad[j][y][theta] =
      cubo_combinado.probabilidad[j][y-(int)(accion.xy.y)][k];
  }
    }
  }


  /* Desplaza sólo columnas. Si existe rotación, gira tras desplazarse. */

  if ((desplaza_filas == 0) && (desplaza_columnas == 1)) {
    if (accion.xy.x > 0) {
      printf("DESPLAZA COLUMNAS x>0 \n");
      for (i=(FILAS-1); i>=0; i--)
for (x=accion.xy.x; x<COLUMNAS; x++)
  for (theta=0; theta<GRADOS_C; theta++) {
    k = theta - accion.theta;  if (k < 0) k+=GRADOS_C;
    cubo_imagen0.probabilidad[x][i][theta]
      = cubo_combinado.probabilidad[x-(int)(accion.xy.x)][i][k];
  }
    }
    if (accion.xy.x < 0) {
      printf("DESPLAZA COLUMNAS x<0 \n");
      for (i=(FILAS-1); i>=0; i--)
for (x=0; x<(COLUMNAS-(accion.xy.x*(-1))); x++)
  for (theta=0; theta<GRADOS_C; theta++) {
    k = theta - accion.theta; if (k < 0) k+=GRADOS_C;
```

```
    cubo_imagen0.probabilidad[x][i][theta] =
      cubo_combinado.probabilidad[x-(int)(accion.xy.x)][i][k];
  }
    }
  }


 /* Desplaza sólo ángulos, no existe traslación */

  if ((desplaza_filas == 0) && (desplaza_columnas == 0)
&& (desplaza_angulos == 1)) {
    printf("DESPLAZA ANGULO != 0 \n");
    for (i=(FILAS-1); i>=0; i--)
      for (j=0; j<COLUMNAS; j++)
for (theta=0; theta<GRADOS_C; theta++) {
  k = theta - accion.theta; if (k < 0) k+=GRADOS_C;
  cubo_imagen0.probabilidad[j][i][theta] =
    cubo_combinado.probabilidad[j][i][k];
}
  }
  return 0;
}


int inicializa_linea_camara() {

  int i;

  for (i=0; i<PIXELS_CAMARA; i++)
    linea_c[i] = ' ' ;

  return 0;
}


int comprueba_si_ve_baliza(int i) {

  /* Función que calcula si una baliza es visible o no a la cámara.   */

  double db, thetab, cthetar, cthetamascvmedios, cthetamenoscvmedios;
  int bx, by, bcolor, ccx, ccy, pixel_baliza, cthetag;

  bx = balizas_m[i].xy.x;
```

```
  by = balizas_m[i].xy.y;
  bcolor = balizas_m[i].color;

  ccx = i_camara.xy.x;
  ccy = i_camara.xy.y;
  cthetag = i_camara.theta;

  cthetar = cthetag * DEGTORAD;

  cthetamascvmedios = cthetar + (CAMPO_VISUAL * DEGTORAD / 2);
  cthetamenoscvmedios = cthetar - (CAMPO_VISUAL * DEGTORAD / 2);

  /* Aquí se eliminan valores menores que 0 o mayores que 360 para los  */
  /* ángulos comprendidos en el campo visual                            */
  if (cthetamascvmedios > (2*PI)) cthetamascvmedios -= (2*PI);
  if (cthetamenoscvmedios < 0) cthetamenoscvmedios += (2*PI);

  db = sqrt(((by - ccy)*(by - ccy)) + ((bx - ccx)*(bx - ccx)));

  /* Aquí se eliminan valores menores que 0 para el ángulo de la baliza a */
  /* detectar y el pixel de la baliza                                     */
  thetab = atan2((by - ccy),(bx - ccx));
  if (thetab < 0) thetab += (2*PI);

  /* Cálculo del pixel: Cuando thetab  > cthetamascvmedios, tenemos 0 dentro */
  /* del campo visual y por tanto discontinuidad en los angulos              */
  if (thetab  > cthetamascvmedios)
    pixel_baliza = PIXELS_CAMARA-(PIXELS_CAMARA*(thetab-cthetamenoscvmedios)/
(CAMPO_VISUAL*DEGTORAD));
  else
    pixel_baliza = PIXELS_CAMARA*(cthetamascvmedios-thetab)/
(CAMPO_VISUAL*DEGTORAD);

  if ((db <= ALCANCE_C) && (db > 0) &&
      (((cthetamenoscvmedios <= cthetamascvmedios) &&
(cthetamenoscvmedios <= thetab) && (thetab <= cthetamascvmedios)) ||
       ((cthetamascvmedios <= cthetamenoscvmedios) &&
((thetab <= cthetamascvmedios) || (cthetamenoscvmedios <= thetab))))) {
      linea_c[pixel_baliza] = bcolor;
  }

  return 0;
}
```

```
double calcula_distancia(char pixel_im1, char *imagen2, int i) {

  int pixel, iguales;
  double distancia, ascendente, descendente;

  distancia = 0;
  ascendente = 0; descendente = 0;

  if (i>0) {
    /* DESCENDENTE, recorre los pixels anteriores de la imagen calculada */
    /* buscando uno igual al dado si no lo encuentra añade la distancia  */
    /* máxima entre pixels                                               */
    iguales = 0; pixel = i-1;
    while ((iguales==0) && (pixel>=0)) {
      if (pixel_im1 == imagen2[pixel]) {
descendente = descendente + (i-pixel);
iguales = 1;
      }
      else pixel --;
    }
    if (iguales==0) descendente = descendente + PIXELS_CAMARA;
  }
  else descendente = descendente + PIXELS_CAMARA;

  if (i<PIXELS_CAMARA) {
    /* ASCENDENTE, recorre los pixels posteriores de la imagen calculada */
    /* buscando uno igual al dado si no lo encuentra añade la distancia  */
    /* máxima entre pixels                                               */
    iguales = 0; pixel = i+1;
    while ((iguales==0) && (pixel<PIXELS_CAMARA)) {
      if (pixel_im1 == imagen2[pixel]) {
ascendente = ascendente + (pixel-i);
iguales = 1;
      }
      else pixel ++;
    }
    if (iguales==0) ascendente = ascendente + PIXELS_CAMARA;
  }
  else ascendente = ascendente + PIXELS_CAMARA;

 if (descendente<ascendente) distancia = distancia + descendente;
```

```
 else distancia = distancia + ascendente;

 return distancia;
}


int compara_con_imagen_teorica(int vcx, int vcy, int vctheta, int n) {
  /* Calcula la distancia entre la imagen de entrada y la calculada según el */
  /* mapa. Si coinciden distancia = 0. Si no coinciden recorre la imagen     */
  /* calculada para ver si la coincidencia existe entre pixels anteriores o  */
  /* posteriores. La comparación sólo se realiza para posiciones distintas   */
  /* de blanco. La probabilidad obtenida del cálculo se representa en un     */
  /* fichero con valores de grises de 0 = MUY DIFERENTES a 255 = IGUALES     */

  int i, num_balizast, num_balizasr;
  double distancia_salr, distancia_salt, distancia_med, distancia_fin, probabilidad;

  inicializa_linea_camara();
  for (i=0; i<cbalizas; i++)
    comprueba_si_ve_baliza(i);

  distancia_salr = 0; distancia_salt = 0; num_balizast = 0; num_balizasr = 0;

  for (i=0; i<PIXELS_CAMARA; i++) {

    if ((buffer_i[i] != ' ') && (buffer_i[i] != linea_c[i])){
      num_balizasr ++;
      distancia_salr =
distancia_salr + calcula_distancia(buffer_i[i], &linea_c[0], i);
    }

    if ((linea_c[i] != ' ') && (buffer_i[i] != linea_c[i])){
      num_balizast ++;
      distancia_salt =
distancia_salt + calcula_distancia(linea_c[i], &buffer_i[0], i);
    }

  }

  if ((distancia_salr == 0) && (distancia_salt == 0)) distancia_med = 0;
  else {
    if (distancia_salr == 0)
      distancia_med = (distancia_salt/num_balizast);
```

```
    else {
      if (distancia_salt == 0)
distancia_med=(distancia_salr/num_balizasr);
      else
distancia_med=
((distancia_salr/num_balizasr)+(distancia_salt/num_balizast))/2.0;
    }
  }

  distancia_fin = distancia_med;

  probabilidad = exp(distancia_fin*distancia_fin*(-1));

  cubo_imagen1.probabilidad[vcx][vcy][vctheta] = probabilidad;

  return 0;
}

int calcula_cubo_combinado(int n) {

  int i, j, k, p;
  double ratio0, ratio1, ratioc, probabilidadc;

  for (i=(FILAS-1); i>=0; i--)
    for (j=0; j<COLUMNAS; j++)
      for (k=0; k<GRADOS_C; k++) {
cubo_imagen0.probabilidad[j][i][k] =
(0.999998 * cubo_imagen0.probabilidad[j][i][k]) + 0.000001;
cubo_imagen1.probabilidad[j][i][k] =
(0.999998 * cubo_imagen1.probabilidad[j][i][k]) + 0.000001;
ratio0 = cubo_imagen0.probabilidad[j][i][k] /
(1 - cubo_imagen0.probabilidad[j][i][k]);
ratio1 = cubo_imagen1.probabilidad[j][i][k] /
(1 - cubo_imagen1.probabilidad[j][i][k]);
ratioc = ratio0*ratio1;
probabilidadc = ratioc / (1+ratioc);
cubo_combinado.probabilidad[j][i][k] = probabilidadc;

p=(int)(cubo_combinado.probabilidad[j][i][k]*100);
histograma_c[p] = histograma_c[p] + 1;
      }

  return 0;
```

```
}


int crea_ficheros_pgm(int n) {

  char header[70], nombre[80];
  int x, y, theta;
  int fichero1, fichero2, fichero3;
  int gris;
  unsigned char grey;

  /* CUBO 0 (C0) = Imagen desplazada, CUBO 1 (C1) = Imagen Calculada, */
  /* CUBO COMBINADO (CC) = Media                                      */

  printf ("CREA_FICHERO_PGM: n=%d\n", n);

  for (theta = 0; theta < GRADOS_C; theta += 10) {

    sprintf(nombre,"pgms/%d/C0-%d.pgm", theta, n);
    fichero1 = open(nombre, O_CREAT | O_WRONLY, S_IRUSR | S_IWUSR);
    if ( fichero1 < 0) {
      printf ("CREA_FICHERO_PGM: No puedo abrir el fichero %s\n", &nombre[0]);
      exit (-1);
    }

    sprintf(nombre,"pgms/%d/C1-%d.pgm", theta, n);
    fichero2 = open(nombre, O_CREAT | O_WRONLY, S_IRUSR | S_IWUSR);
    if ( fichero2 < 0) {
      printf ("CREA_FICHERO_PGM: No puedo abrir el fichero %s\n", &nombre[0]);
      exit (-1);
    }

    sprintf(nombre,"pgms/%d/CC-%d.pgm", theta, n);
    fichero3 = open(nombre, O_CREAT | O_WRONLY, S_IRUSR | S_IWUSR);
    if ( fichero3 < 0) {
      printf ("CREA_FICHERO_PGM: No puedo abrir el fichero %s\n", &nombre[0]);
      exit (-1);
    }

    sprintf(header, "P5\n%d\n%d\n255\n", COLUMNAS, FILAS);
    write(fichero1,&header[0],strlen(header));
    write(fichero2,&header[0],strlen(header));
    write(fichero3,&header[0],strlen(header));
```

```
    for (y = (FILAS -1); y >= 0; y--) {
      for (x = 0; x < COLUMNAS; x++) {

gris = (int)(cubo_imagen0.probabilidad[x][y][theta]*255);
grey = (unsigned char) gris;
write(fichero1, &grey, sizeof(unsigned char));

gris = (int)(cubo_imagen1.probabilidad[x][y][theta]*255);
grey = (unsigned char) gris;
write(fichero2, &grey, sizeof(unsigned char));

gris = (int)(cubo_combinado.probabilidad[x][y][theta]*255);
grey = (unsigned char) gris;
write(fichero3, &grey, sizeof(unsigned char));

}
      }
    close(fichero1);
    close(fichero2);
    close(fichero3);
  }

  return 0;
}

int imprime_max_y_min(int n) {

  int x, y, theta, i, j;
  char header[70], nombre[80], dat_rec[80];
  int fichero, tplot_file;
  double rojo, verde, azul;
  double cubo_acum_top[COLUMNAS][FILAS], angulo[COLUMNAS][FILAS];
  int irojo, iverde, iazul, crojo, cverde, cazul;
  unsigned char red, green, blue;
  Stop_prob max_prob, min_prob;
  unsigned char grafico_tops[3][COLUMNAS][FILAS], salida[3][3*COLUMNAS][3*FILAS];
  Stop_prob top_prob[25], bot_prob[25];


  for (i=0; i<25; i++) {
    top_prob[i].max_min_prob = 0;
    bot_prob[i].max_min_prob = 1;
```

```
  top_prob[i].max_min_pos.xy.x = 0;
  bot_prob[i].max_min_pos.xy.x = 0;
  top_prob[i].max_min_pos.xy.y = 0;
  bot_prob[i].max_min_pos.xy.y = 0;
  top_prob[i].max_min_pos.theta = 0;
  bot_prob[i].max_min_pos.theta = 0;
}

max_prob.max_min_pos.xy.x = 99;
max_prob.max_min_pos.xy.y = 99;
max_prob.max_min_pos.theta = 999;
max_prob.max_min_prob = 0;
min_prob.max_min_pos.xy.x = 99;
min_prob.max_min_pos.xy.y = 99;
min_prob.max_min_pos.theta = 999;
min_prob.max_min_prob = 1;

for (y = (FILAS -1); y >= 0; y--)
  for (x = 0; x < COLUMNAS; x++) {
    cubo_acum_top[x][y] = 0;
    angulo[x][y] = 0;
  }

sprintf(nombre,"pgms/CC-FINAL%d.ppm",n);
fichero = open(nombre, O_CREAT|O_WRONLY, S_IRUSR|S_IWUSR);
if (fichero < 0) {
  printf ("IMPRIME MAX Y MIN: No puedo abrir el fichero %s\n", &nombre[0]);
  exit (-1);
}

sprintf(nombre,"dat/topplot%d.dat",n);
tplot_file = open(nombre, O_CREAT|O_WRONLY, S_IRUSR|S_IWUSR);
if (tplot_file < 0) {
  printf ("IMPRIME MAX Y MIN: No puedo abrir el fichero %s\n", &nombre[0]);
  exit (-1);
}

sprintf(header, "P6\n%d\n%d\n255\n", (COLUMNAS*3), (FILAS*3));
write(fichero,&header[0],strlen(header));

for (y = (FILAS -1); y >= 0; y--) {
  for (x = 0; x < COLUMNAS; x++) {
    crojo = 0; cverde = 0; cazul = 0; rojo = 0, verde = 0; azul = 0;
```

```
      for (theta = 0; theta < GRADOS_C; theta++) {
if (max_prob.max_min_prob < cubo_combinado.probabilidad[x][y][theta]) {
  max_prob.max_min_pos.xy.x = x;
  max_prob.max_min_pos.xy.y = y;
  max_prob.max_min_pos.theta = theta;
  max_prob.max_min_prob = cubo_combinado.probabilidad[x][y][theta];
}
if (min_prob.max_min_prob > cubo_combinado.probabilidad[x][y][theta]) {
  min_prob.max_min_pos.xy.x = x;
  min_prob.max_min_pos.xy.y = y;
  min_prob.max_min_pos.theta = theta;
  min_prob.max_min_prob = cubo_combinado.probabilidad[x][y][theta];
}
if (cubo_combinado.probabilidad[x][y][theta] >= 0.997) {
  if (cubo_acum_top[x][y] < cubo_combinado.probabilidad[x][y][theta])
    cubo_acum_top[x][y] = cubo_combinado.probabilidad[x][y][theta];
     rojo += cubo_combinado.probabilidad[x][y][theta];
  crojo +=1;
}
else {
  if (cubo_combinado.probabilidad[x][y][theta] < 0.50) {
    verde += cubo_combinado.probabilidad[x][y][theta];
    cverde += 1;
  }
  else {
    azul += cubo_combinado.probabilidad[x][y][theta];
    cazul += 1;
  }
}


irojo = (int)(rojo*255/crojo);
iverde = (int)(verde*255/cverde);
iazul=(int)(azul*255/cazul);

red = (unsigned char) irojo;
green = (unsigned char) iverde;
blue = (unsigned char) iazul;

grafico_tops[0][x][y] = red;
grafico_tops[1][x][y] = green;
grafico_tops[2][x][y] = blue;
```

```
sprintf(dat_rec, "%.2f %.2f %.8f\n",
(double)(x),(double)(y), (cubo_acum_top[x][y]));
write(tplot_file, &dat_rec[0], strlen(dat_rec));

for (i=0; i<25; i++) {
  if (cubo_combinado.probabilidad[x][y][theta]>top_prob[i].max_min_prob){
    for (j=24; j>i; j--) {
      top_prob[j].max_min_pos.xy.x = top_prob[j-1].max_min_pos.xy.x;
      top_prob[j].max_min_pos.xy.y = top_prob[j-1].max_min_pos.xy.y;
      top_prob[j].max_min_pos.theta = top_prob[j-1].max_min_pos.theta;
      top_prob[j].max_min_prob = top_prob[j-1].max_min_prob;
    }
    top_prob[i].max_min_pos.xy.x = x;
    top_prob[i].max_min_pos.xy.y = y;
    top_prob[i].max_min_pos.theta = theta;
    top_prob[i].max_min_prob = cubo_combinado.probabilidad[x][y][theta];
    i=25;
  }
}

for (i=0; i<25; i++) {
  if (cubo_combinado.probabilidad[x][y][theta]<bot_prob[i].max_min_prob){
    for (j=24; j>i; j--) {
      bot_prob[j].max_min_pos.xy.x = bot_prob[j-1].max_min_pos.xy.x;
      bot_prob[j].max_min_pos.xy.y = bot_prob[j-1].max_min_pos.xy.y;
      bot_prob[j].max_min_pos.theta = bot_prob[j-1].max_min_pos.theta;
      bot_prob[j].max_min_prob = bot_prob[j-1].max_min_prob;
    }
    bot_prob[i].max_min_pos.xy.x = x;
    bot_prob[i].max_min_pos.xy.y = y;
    bot_prob[i].max_min_pos.theta = theta;
    bot_prob[i].max_min_prob = cubo_combinado.probabilidad[x][y][theta];
    i=5;
  }
}
      }
    }
  }

  for (y = (FILAS -1); y >= 0; y--)
    for (x = 0; x < COLUMNAS; x++)
      for (theta = 0; theta < GRADOS_C; theta++) {
if (cubo_combinado.probabilidad[x][y][theta] == max_prob.max_min_prob) {
```

```
  printf("MAXIMA PROBABILIDAD Cubo %d x=%d y=%d theta=%d %.16f\n",
 n, x, y, theta, max_prob.max_min_prob);
  angulo[x][y] = theta;
  grafico_tops[1][x][y] = grafico_tops[0][x][y];
}
      }

  for (y = (FILAS - 1); y >= 0; y--)
    for (x = 0; x < COLUMNAS; x++) {
      for (i = 0; i < 3; i++) {
for (j = 0; j < 3; j++) {
  salida[0][(3*x)+j][(3*y)+i] = grafico_tops[0][x][y];
  salida[1][(3*x)+j][(3*y)+i] = grafico_tops[1][x][y];
  salida[2][(3*x)+j][(3*y)+i] = grafico_tops[2][x][y];
}
      }
      if ((cubo_acum_top[x][y] == max_prob.max_min_prob) && (max_prob.max_min_prob !=
salida[0][(3*x)+1][(3*y)+1] = 0;
salida[1][(3*x)+1][(3*y)+1] = 0;
salida[2][(3*x)+1][(3*y)+1] = 0;
if (((angulo[x][y] >= 0) && (angulo[x][y] < 45)) || (angulo[x][y] == 360)) {
  salida[0][(3*x)+2][(3*y)+1] = 0;
  salida[1][(3*x)+2][(3*y)+1] = 0;
  salida[2][(3*x)+2][(3*y)+1] = 0;
}
else {
  if ((angulo[x][y] >= 45) && (angulo[x][y] < 90)) {
    salida[0][(3*x)+2][(3*y)+2] = 0;
    salida[1][(3*x)+2][(3*y)+2] = 0;
    salida[2][(3*x)+2][(3*y)+2] = 0;
  }
  else {
    if ((angulo[x][y] >= 90) && (angulo[x][y] < 135)) {
      salida[0][(3*x)+1][(3*y)+2] = 0;
      salida[1][(3*x)+1][(3*y)+2] = 0;
      salida[2][(3*x)+1][(3*y)+2] = 0;
    }
    else {
      if ((angulo[x][y] >= 135) && (angulo[x][y] < 180)) {
salida[0][3*x][(3*y)+2] = 0;
salida[1][3*x][(3*y)+2] = 0;
salida[2][3*x][(3*y)+2] = 0;
      }
```

```
      else {
if ((angulo[x][y] >= 180) && (angulo[x][y] < 225)) {
  salida[0][3*x][(3*y)+1] = 0;
  salida[1][3*x][(3*y)+1] = 0;
  salida[2][3*x][(3*y)+1] = 0;
}
else {
  if ((angulo[x][y] >= 225) && (angulo[x][y] < 270)) {
    salida[0][3*x][3*y] = 0;
    salida[1][3*x][3*y] = 0;
    salida[2][3*x][3*y] = 0;
  }
  else {
    if ((angulo[x][y] >= 270) && (angulo[x][y] < 315)) {
      salida[0][(3*x)+1][3*y] = 0;
      salida[1][(3*x)+1][3*y] = 0;
      salida[2][(3*x)+1][3*y] = 0;
    }
    else {
      if ((angulo[x][y] >= 315) && (angulo[x][y] < 360)) {
salida[0][(3*x)+2][3*y] = 0;
salida[1][(3*x)+2][3*y] = 0;
salida[2][(3*x)+2][3*y] = 0;
      }
    }
  }
}
      }
    }
  }
}
      }
    }

  for (y = ((FILAS * 3)-1); y >= 0; y--)
    for (x = 0; x < (COLUMNAS*3); x++) {
      write(fichero, &salida[0][x][y], sizeof(unsigned char));
      write(fichero, &salida[1][x][y], sizeof(unsigned char));
      write(fichero, &salida[2][x][y], sizeof(unsigned char));
    }

  printf ("\n");
  for (i=0; i<25; i++)
```

```
    printf ("TOP 25 MAXIMA PROBABILIDAD %d x=%.2f y=%.2f, theta=%.2f,
Prob=%.16f\n", i, top_prob[i].max_min_pos.xy.x,
top_prob[i].max_min_pos.xy.y, top_prob[i].max_min_pos.theta,
top_prob[i].max_min_prob);

  for (i=0; i<25; i++)
    printf ("BOTTOM 25 MINIMA PROBABILIDAD %d x=%.2f y=%.2f, theta=%.2f,
Prob=%.16f\n", i, bot_prob[i].max_min_pos.xy.x,
bot_prob[i].max_min_pos.xy.y, bot_prob[i].max_min_pos.theta,
bot_prob[i].max_min_prob);

  close(fichero);
  close(tplot_file);

  return 0;
}


int inicializa_histograma() {

  int i;

  for (i=0; i<=100; i++)
    histograma_c[i] = 0;

  return 0;

}


int imprime_histograma(int n) {

  int j;
  double k;
  char nombre[80], dat_rec[80];
  int plot_file;

  sprintf(nombre,"dat/histog%d.dat",n);
  plot_file = open(nombre, O_CREAT|O_WRONLY, S_IRUSR|S_IWUSR);
  if (plot_file < 0) {
    printf ("IMPRIME HISTOGRAMA: No puedo abrir el fichero %s\n", &nombre[0]);
    exit (-1);
  }
```

```
  for (j=0; j<=100; j++) {
    k = (double)(j);
    sprintf(dat_rec, "%.2f %.2f\n", k,((double)(histograma_c[j])));
    write(plot_file, &dat_rec[0], strlen(dat_rec));
  }

  close(plot_file);

  return 0;
}




/* Programa Principal */

int main(int argc, char ** argv) {

 int mcx, mcy, mctheta, n;


  /* Toma variables de entrada, las verifica y asigna adecuadamente */


  if ((argc != 1) && (argc != 3)) {
    printf ("MAIN: Numero de argumentos erroneo %d\n", argc);
    printf ("MAIN: Uso 'verosimilitud [nombre_mapa] [nombre_accion_e_imagen]'\n");
    exit(-1);
  }

  if (argc == 1) {
    fichero_mapa = "mi_mapa";
    fichero_imagen = "accion_e_imagen.txt";
  }
  else {
    fichero_mapa = argv[1];
    fichero_imagen = argv[2];
  }

  /* Variables de control del número de líneas y balizas del fichero de entrada */
  cbalizas = 0; clineas = 0;
```

```
  /* Abre los ficheros de entrada, inicializa el buffer y carga el mapa leido */
  /* y crea un fichero con el campo visual dentro del mapa                    */
  mapa = fopen(fichero_mapa, "r");
  imagen = fopen(fichero_imagen, "r");

  if (mapa == NULL) {
    printf ("MAIN: No puedo abrir el mapa\n");
    exit (-1);
  }

  cuenta_balizas();
  if (fclose(mapa) != 0) {
    printf ("MAIN: No puedo cerrar el mapa\n");
    exit (-1);
  }

  mapa = fopen(fichero_mapa, "r");
  if (mapa == NULL) {
    printf ("MAIN: No puedo abrir el mapa por segunda vez\n");
    exit (-1);
  }
  inicializa_mapa();
  lee_mapa();
  imprime_mapa();

  if (imagen == NULL) {
    printf ("MAIN: No puedo abrir el fichero de imagenes y acciones\n");
    exit (-1);
  }

  n = 0;
  cubo_size = (double)(FILAS*COLUMNAS*GRADOS_C);
  inicializa_cubo0(0.5);
  inicializa_histograma();

  while (fin_imagen != 'F') {
    inicializa_imagen();
    lee_imagen();
    if (fin_imagen!='f') {
      if (n > 0) desplaza_cubo_combinado(n);
      for (mctheta = 0; mctheta < GRADOS_C; mctheta ++) {
i_camara.theta = mctheta;
for (mcy = (FILAS -1); mcy >= 0; mcy--) {
```

```
  i_camara.xy.y = mcy;
  for (mcx = 0; mcx < COLUMNAS; mcx++) {
    i_camara.xy.x = mcx;
    compara_con_imagen_teorica(mcx, mcy, mctheta, n );
  }
}
      }
      calcula_cubo_combinado(n);
      crea_ficheros_pgm(n);
      imprime_max_y_min(n);
      imprime_histograma(n);
      n++;
    }
    else fin_imagen='F';
  }

  fcloseall();

  return 0;
}
```

A.2.2. Versión 4.5 - Probabilístico con muestreo

```
/*                                                                      */
/* montecarlo-4.5-tgz                                                   */
/*                                                                      */
/*      Uso 'verosimilitud [nombre_mapa] [nombre_accion_e_imagen]'      */
/*      Por defecto:                                                    */
/*          nombre_mapa = "mi_mapa" y nombre_imagen = "accion_e_imagen.txt" */
/*                                                                      */
/*      Lee un mapa de FILAS*COLUMNAS y lo carga en un buffer,          */
/*      identificando líneas y balizas.                                 */
/*      Se admiten un máximo de MAX_LINEAS líneas.                      */
/*      El fichero tiene dos tipos de registros:                        */
/*              Lxi,yi,xf,yf                                            */
/*              Bx,y,C                                                  */
/*              Carácter B + coordenadas de la baliza + color           */
/*                                                  (1 letra mayúscula) */
/*                                                                      */
/*      Lee una imagen inicial de PIXELS_CAMARA columnas y las compara con */
/* las calculadas teóricamente, generando ficheros con las imágenes (en */
/* grises) de las probabilidades de que la cámara esté posicionada en las */
/* posiciones disponibles dentro del campo.                             */
/* Se generan 36 ficheros de nombre verosimilitud*.pgm donde 0 <= * < 360. */
/* Cada uno de ellos contempla todas las posibles coordenadas x e y del campo */
/* pero un único ángulo de 0 a 360 en intervalos de 10 grados.          */
/*                                                                      */
/*      A continuación lee una acción y su imagen asociada y la compara con */
/* la calculada teóricamente partiendo del cálculo realizado con la imagen */
/* inicial y la acción indicada.                                        */
/*      Se generan cubos de probabilidades (para las coordenadas y ángulos */
/* posibles en el mapa de entrada) y se van combinando para  llegar a un cubo */
/* final que debería reducir las localizaciones posibles con la información */
/* acumulada.                                                           */
/*      El fichero accion_e_imagen.txt ha sido generado por el programa */
/* localiza_baliza y parte de una imagen inicial en la que no se proporcionan */
/* las coordenadas y una serie de acciones e imágenes resultantes, en el */
/* siguiente formato:                                                   */
/*              Px,y,theta,imagen[PIXELS_CAMARA]                        */
/*              Posición inicial e imagen real                          */
/*              Ax,y,theta,imagen[PIXELS CAMARA]                        */
/*              Acciones sucesivas e imagenes generadas después de aplicarlas */
/* El programa localiza_baliza ya ha verificado que las acciones indicadas son*/
/* posibles dentro del mapa (habiéndose rechazado otras que no lo eran). */
```

```
/* Se generaran 36 pgm ficheros como los indicados anteriormente para cada una*/
/* de "las verosimilitudes" resultantes de aplicar las acciones y combinar los*/
/* cubos de probabilidades.                                                    */
/* Además se crea un fichero .ppm y un fichero.dat con las "TOP 25"            */
/* probabilidades y un histograma con la curva de valores de probabilidades    */
/* tras cada acción, destacando el TOP1.                                       */
/* Se eliminan las limitaciones en el número de balizas contenidas en el mapa.*/
/* Para ello se hace una lectura previa del mapa y se contabiliza el número de*/
/* balizas contenidas en el mismo, asignándose dinámicamente memoria al array */
/* que acumula esta información.                                               */
/* 4.x Incorpora muestreo                                                      */
/*                                                                             */

#include <stdio.h>
#include <math.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <time.h>

#define FILAS         33
#define COLUMNAS      65
#define GRADOS_C     360
#define COORD_MAX     64
#define MAX_LINEAS     5
#define PIXELS_CAMARA 80
#define ALCANCE_C     25
#define CAMPO_VISUAL  45
#define PI             3.14159265
#define DEGTORAD     (PI / 180.0)
#define RADTODEG     (180.0 / PI)
#define NMUESTRAS   2000
#define RANGO_GAUSS 2001


/* Definición de estructuras de datos para balizas y posición de cámara */

typedef struct coordenada{
  double x;
  double y;
}Scoordenada;

typedef struct baliza{
```

```
  Scoordenada xy;
  char color;
}Sbaliza;

typedef struct posicion_camara{
  Scoordenada xy;
  double theta;
}Sposicion_camara;

typedef struct cubo_prob{
  double probabilidad[COLUMNAS][FILAS][GRADOS_C];
}Scubo_prob;

typedef struct top_prob{
  Sposicion_camara max_min_pos;
  double max_min_prob;
}Stop_prob;

typedef struct ficheros {
  double prob_fichero;
  int n_ocurr;
}Sficheros;

typedef struct ruido{
  double d;
  double gauss_d;
}Sruido;

/* Variables globales   */

FILE *mapa, *imagen;
char *fichero_mapa, *fichero_imagen;
char fin_mapa, fin_imagen;
char que_leo, imagen_en_blanco;
char buffer_m[COLUMNAS][FILAS];
char linea_c[PIXELS_CAMARA];
int cbalizas, clineas;
Sposicion_camara i_camara, accion;
char buffer_i[PIXELS_CAMARA];
int histograma_c[101];
double dispx[50], dispy[50], dispt[50], pos_veros[50];
Sbaliza *balizas_m;
Sposicion_camara coord_muestra[NMUESTRAS], coord_muestra_temp[NMUESTRAS];
```

```
double muestra[NMUESTRAS], muestra_acumulada[NMUESTRAS];
Sruido array_ruido_xy[RANGO_GAUSS], gaussiana_acumulada_xy[RANGO_GAUSS];
Sruido array_ruido_theta[RANGO_GAUSS], gaussiana_acumulada_theta[RANGO_GAUSS];

int inicializa_mapa() {

  int i, j;

  for (i=0; i<FILAS; i++)
    for (j=0; j<COLUMNAS; j++)
      buffer_m[j][i] = ' ';

  cbalizas = 0;

  return 0;
}


int lee_1coordenada(FILE *fichero){

  int i = 0, j, coordenada = 0;
  int posiciones[COORD_MAX];
  char aux, signo;
  char *p_aux;

  signo = '+';
  aux = fgetc(fichero);
  p_aux = &aux;
  while ((aux != '\n') && (aux != ',') && (aux != EOF)) {
    if (isdigit(aux)) {
      posiciones[i] = atoi(p_aux);
      i++;
      aux = fgetc(fichero);
    }
    else {
      if (aux == '-') {
signo = '-';
aux = fgetc(fichero);
      }
      else {
printf("LEE COORDENADA: Valor inesperado %c\n", aux);
exit(-1);
      }
```

```
    }
  }

  for (j=0; j<(i-1); j++)
    coordenada = coordenada + (posiciones[j]*pow(10,(i-j-1)));
  coordenada = coordenada + posiciones[(i-1)];
  if (signo == '-') coordenada = coordenada * (-1);

  return coordenada;
}


int salta_linea() {

  int xi, yi, xf, yf;

  xi = lee_1coordenada(mapa);
  yi = lee_1coordenada(mapa);
  xf = lee_1coordenada(mapa);
  yf = lee_1coordenada(mapa);

  if ((xi >= COLUMNAS) || (xf >= COLUMNAS ) || (yi >= FILAS) || (yf >= FILAS) ||
      (xi < 0) || (xf < 0 ) || (yi < 0) || (yf < 0)) {
    printf("SALTA LINEA: Coordenadas invalidas %d %d %d %d\n", xi, xf, yi, yf);
    exit(-1);
  }

  return 0;
}


int salta_baliza() {

  int x, y;
  char aux;

  x = lee_1coordenada(mapa);
  y = lee_1coordenada(mapa);

  if ((x >= COLUMNAS) || (y >= FILAS) || (x < 0 ) || (y < 0)) {
     printf("SALTA BALIZA: Coordenadas invalidas %d %d\n", x, y);
     exit(-1);
  }
```

```
  aux = fgetc(mapa);

  if (!(isupper(aux))) {
    printf ("SALTA BALIZA: Valor inesperado %c\n", aux);
    exit(-1);
  }

  return 0;
}

int cuenta_balizas() {

  char que_leo;

  fin_mapa = ' ';

  while (fin_mapa != 'F') {
    que_leo = fgetc(mapa);
    if (que_leo == EOF) fin_mapa = 'F';
    if (que_leo == 'L')
      salta_linea();
    if (que_leo == 'B') {
      cbalizas ++;
      salta_baliza();
    }
    if (que_leo == '\n');
    if ((que_leo != EOF) && (que_leo != 'L') &&
(que_leo != 'B') && (que_leo != '\n')) {
      printf("CUENTA BALIZAS: No puedo interpretar contenido del fichero %c\n",
 que_leo);
      exit(-1);
    }
  }

  balizas_m = (Sbaliza *)malloc(sizeof(Sbaliza)*cbalizas);
  if (balizas_m == NULL) {
    printf("CUENTA BALIZAS: No hay memoria disponible para mapa de balizas\n");
    exit(-1);
  }

  return 0;
}
```

```
int lee_linea() {

  int xi, yi, xf, yf, x, y;

  xi = lee_1coordenada(mapa);
  yi = lee_1coordenada(mapa);
  xf = lee_1coordenada(mapa);
  yf = lee_1coordenada(mapa);

  if (xi == xf) {
    for(y=yi; y>=yf; y--){
      buffer_m[xi][y] = 'W';
    }
  }

  if (yi == yf) {
    for (x=xi; x<=xf; x++) {
      buffer_m[x][yi] = 'W';
    }
  }

  return 0;
}


int lee_baliza() {

  int x, y;
  char aux;

  x = lee_1coordenada(mapa); balizas_m[cbalizas-1].xy.x = x;
  y = lee_1coordenada(mapa); balizas_m[cbalizas-1].xy.y = y;

  aux = fgetc(mapa);

  buffer_m[x][y] = aux;
  balizas_m[cbalizas-1].color = aux;

  return 0;
}
```

```
int lee_mapa() {

  char que_leo;

  fin_mapa = ' ';

  while (fin_mapa != 'F') {
    que_leo = fgetc(mapa);
    if (que_leo == EOF) fin_mapa = 'F';
    if (que_leo == 'L') {
      clineas ++;
      if (clineas <= MAX_LINEAS)
lee_linea();
      else {
printf ("LEE MAPA: Excedido el maximo numero de lineas %d\n",MAX_LINEAS);
exit(-1);
      }
    }
    if (que_leo == 'B') {
      cbalizas++;
      lee_baliza();
    }
  }
  return 0;
}


int imprime_mapa() {

  int i, j;

  for (i=(FILAS-1); i>=0; i--){
    for (j=0; j<COLUMNAS; j++) printf("%c", buffer_m[j][i]);
    printf(" \n");
  }

  return 0;
}

int calcula_gaussiana_acumulada(int i) {

 gaussiana_acumulada_xy[i].d = array_ruido_xy[i].d;
 gaussiana_acumulada_theta[i].d = array_ruido_theta[i].d;
```

```
  if (i==0) {
    gaussiana_acumulada_xy[i].gauss_d = array_ruido_xy[i].gauss_d;
    gaussiana_acumulada_theta[i].gauss_d = array_ruido_theta[i].gauss_d;
  }
  else {
    gaussiana_acumulada_xy[i].gauss_d =
      gaussiana_acumulada_xy[i-1].gauss_d+array_ruido_xy[i].gauss_d;
    gaussiana_acumulada_theta[i].gauss_d =
      gaussiana_acumulada_theta[i-1].gauss_d+array_ruido_theta[i].gauss_d;
  }

  return 0;

}


double calcula_ruido(char dimension) {

  double temp, k;
  int nueva_muestra;

  if (dimension == 'C') {
    temp=((double)(rand()))/((double)(RAND_MAX+1.0));
    k=temp*gaussiana_acumulada_xy[RANGO_GAUSS-1].gauss_d;
    nueva_muestra = 0;
    if (gaussiana_acumulada_xy[nueva_muestra].gauss_d < k) {
      nueva_muestra ++;
      while ((nueva_muestra<RANGO_GAUSS) &&
(gaussiana_acumulada_xy[nueva_muestra].gauss_d <= k))
nueva_muestra++;
    }
    return gaussiana_acumulada_xy[nueva_muestra].d;
  }
  else {
    temp=((double)(rand()))/((double)(RAND_MAX+1.0));
    k=temp*gaussiana_acumulada_theta[RANGO_GAUSS-1].gauss_d;
    nueva_muestra = 0;
    if (gaussiana_acumulada_theta[nueva_muestra].gauss_d < k) {
      nueva_muestra ++;
      while ((nueva_muestra<RANGO_GAUSS) &&
     (gaussiana_acumulada_theta[nueva_muestra].gauss_d <= k))
nueva_muestra++;
```

```
    }
    return gaussiana_acumulada_theta[nueva_muestra].d;
  }



}

int calcula_gaussianas() {

  /* El error de movimiento en x,y y theta se calcula en porcentaje;
     se proporcionan dos valores de desviación uno para x e y otro para theta,
     calculando los extremos de la gausiana como tres veces dicho valor.
     Así para desv_xy = 0.05, los extremos de la función serán 0.15
     lo que equivale a un error del 15% */

  double media_xy, desv_xy, extremos_xy;
  double media_theta, desv_theta, extremos_theta;
  int i;

  media_xy = 0;
  desv_xy = 0.05;
  extremos_xy = 3.0*desv_xy;
  media_theta = 0;
  desv_theta = 0.05;
  extremos_theta = 3.0*desv_theta;

  for (i=0;i<RANGO_GAUSS;i++) {
    array_ruido_xy[i].d =
((extremos_xy+extremos_xy)*i/(RANGO_GAUSS-1)) - extremos_xy;
    array_ruido_xy[i].gauss_d = (1.00/(desv_xy*sqrt(2.00*PI)))*(exp(-(1.00/2.00)*
                                (((array_ruido_xy[i].d-media_xy)/desv_xy)*
                                ((array_ruido_xy[i].d-media_xy)/desv_xy))));
    array_ruido_theta[i].d =
((extremos_theta+extremos_theta)*i/(RANGO_GAUSS-1)) - extremos_theta;
    array_ruido_theta[i].gauss_d =
(1.00/(desv_theta*sqrt(2.00*PI)))* (exp(-(1.00/2.00)*
        (((array_ruido_theta[i].d-media_theta)/desv_theta)*
        ((array_ruido_theta[i].d-media_theta)/desv_theta))));
    calcula_gaussiana_acumulada(i);
  }

  return 0;
```

```
}

int genera_muestras(int i, int n) {

  double temp;
  double x, y, theta;

  temp=((double)(rand()))/((double)(RAND_MAX+1.0));
  x=temp*COLUMNAS;
  temp=((double)(rand()))/((double)(RAND_MAX+1.0));
  y=temp*FILAS;
  temp=((double)(rand()))/((double)(RAND_MAX+1.0));
  theta=temp*GRADOS_C;

  coord_muestra[i].xy.x = x;
  coord_muestra[i].xy.y = y;
  coord_muestra[i].theta = theta;

  return 0;
}


int inicializa_imagen() {

  int i;

  for (i=0; i<PIXELS_CAMARA; i++)
    buffer_i[i] = ' ';

  return 0;
}


int lee_imagen() {

  int i=0;
  char pixel;

  fin_imagen = ' ' ; imagen_en_blanco = 1;

  pixel = fgetc(imagen);
  if ((pixel = 'P') || (pixel = 'A')) {
```

```
    accion.xy.x = lee_1coordenada(imagen);
    accion.xy.y = lee_1coordenada(imagen);
    accion.theta = lee_1coordenada(imagen);
  }
  else {
    if (pixel != '#') {
      printf("LEE IMAGEN: Formato de imagen erroneo,
deberia ser posicion inicial 'P' o accion 'A', no %c\n", pixel);
      exit(-1);
    }
  }

  while ((pixel != '\n') && (pixel != EOF) && (pixel != '#')){
    pixel = fgetc(imagen);
    buffer_i[i] = pixel;
    if (isupper(pixel))
      imagen_en_blanco = 0;
    i++;
  }

  if ((i<PIXELS_CAMARA) && (pixel !='#') && (pixel != EOF)){
    printf("LEE IMAGEN: Formato de imagen erroneo,
solo %d pixels leidos (deben ser 80)\n", i);
    exit(-1);
  }

  if ((pixel == EOF) || (pixel == '#')) fin_imagen = 'F';

  return 0;
}


int desplaza_muestra(int i) {

  double ruido_x, ruido_y, ruido_theta;

 /* Aplica acción a la muestra combinada. */

   ruido_x = calcula_ruido('C'); ruido_y = calcula_ruido('C');

  if (accion.xy.x == 0)
    coord_muestra[i].xy.x = coord_muestra[i].xy.x+ruido_x;
  else
```

```
    coord_muestra[i].xy.x = coord_muestra[i].xy.x+(accion.xy.x*(1.0+ruido_x));

  if (accion.xy.y == 0)
    coord_muestra[i].xy.y = coord_muestra[i].xy.y+ruido_y;
  else
    coord_muestra[i].xy.y = coord_muestra[i].xy.y+(accion.xy.y*(1.0+ruido_y));

  ruido_theta = calcula_ruido('A');

  if (accion.theta == 0)
    coord_muestra[i].theta = coord_muestra[i].theta+ruido_theta;
  else
    coord_muestra[i].theta = coord_muestra[i].theta+(accion.theta*(1.0+ruido_theta));

  if (coord_muestra[i].theta < 0)
    coord_muestra[i].theta += GRADOS_C;
  if (coord_muestra[i].theta > GRADOS_C)
    coord_muestra[i].theta -= GRADOS_C;

  return 0;
}

int calcula_muestra_acumulada(int i) {

  if (i==0) muestra_acumulada[i] = muestra[i];
  else muestra_acumulada[i] = muestra_acumulada[i-1]+muestra[i];

  return 0;
}

int normaliza_muestras() {

  int i;

  for (i=0; i<NMUESTRAS; i++)
    muestra[i] = muestra[i] / muestra_acumulada[NMUESTRAS - 1];
}



int actualiza_histograma(int i) {

  int p;
```

```
  p=(int)(muestra[i]*100);
  histograma_c[p] = histograma_c[p] + 1;

}

int redistribuye_muestras(int n) {

  double temp, k;
  int i, j, nueva_muestra, contador, p;
  Sposicion_camara cuantas_muestras[NMUESTRAS];

  for (i=0; i<NMUESTRAS; i++) {
    temp=((double)(rand()))/((double)(RAND_MAX+1.0));
    k=temp*muestra_acumulada[NMUESTRAS-1];
    nueva_muestra = 0;
    if (muestra_acumulada[nueva_muestra] < k) {
      nueva_muestra ++;
      while ((nueva_muestra<NMUESTRAS) && (muestra_acumulada[nueva_muestra] <= k))
nueva_muestra++;
    }
    coord_muestra_temp[i].xy.x = coord_muestra[nueva_muestra].xy.x;
    coord_muestra_temp[i].xy.y = coord_muestra[nueva_muestra].xy.y;
    coord_muestra_temp[i].theta = coord_muestra[nueva_muestra].theta;
  }

  contador = 0;
  for (i=0; i<NMUESTRAS; i++) {
    if (contador == 0) {
      cuantas_muestras[contador].xy.x = coord_muestra_temp[i].xy.x;
      cuantas_muestras[contador].xy.y = coord_muestra_temp[i].xy.y;
      cuantas_muestras[contador].theta = coord_muestra_temp[i].theta;
      contador ++;
    }
    else {
      for (j=0;j<contador;j++)
if ((cuantas_muestras[j].xy.x == coord_muestra_temp[i].xy.x) &&
    (cuantas_muestras[j].xy.y == coord_muestra_temp[i].xy.y) &&
    (cuantas_muestras[j].theta == coord_muestra_temp[i].theta))
  j = contador + 10;
      if (j==contador) {
cuantas_muestras[contador].xy.x = coord_muestra_temp[i].xy.x;
cuantas_muestras[contador].xy.y = coord_muestra_temp[i].xy.y;
```

```
cuantas_muestras[contador].theta = coord_muestra_temp[i].theta;
contador ++;
      }
    }
    coord_muestra[i].xy.x = coord_muestra_temp[i].xy.x;
    coord_muestra[i].xy.y = coord_muestra_temp[i].xy.y;
    coord_muestra[i].theta = coord_muestra_temp[i].theta;

  }

  printf("Numero de muestras diferentes cubo %d = %d\n", n, contador);
  return 0;
}

int inicializa_linea_camara() {

  int i;

  for (i=0; i<PIXELS_CAMARA; i++)
    linea_c[i] = ' ' ;

  return 0;
}


int comprueba_si_ve_baliza(int i) {

  /* Función que calcula si una baliza es visible o no a la cámara.   */

  double db, thetab, cthetar, cthetamascvmedios, cthetamenoscvmedios;
  double bx, by, bcolor, ccx, ccy, pixel_baliza, cthetag;


  bx = balizas_m[i].xy.x;
  by = balizas_m[i].xy.y;
  bcolor = balizas_m[i].color;

  ccx = i_camara.xy.x;
  ccy = i_camara.xy.y;
  cthetag = i_camara.theta;

  cthetar = cthetag * DEGTORAD;
```

```
  cthetamascvmedios = cthetar + (CAMPO_VISUAL * DEGTORAD / 2);
  cthetamenoscvmedios = cthetar - (CAMPO_VISUAL * DEGTORAD / 2);

  /* Aquí se eliminan valores menores que 0 o mayores que 360 para los  */
  /* ángulos comprendidos en el campo visual                            */
  if (cthetamascvmedios > (2*PI)) cthetamascvmedios -= (2*PI);
  if (cthetamenoscvmedios < 0) cthetamenoscvmedios += (2*PI);

  db = sqrt(((by - ccy)*(by - ccy)) + ((bx - ccx)*(bx - ccx)));

  /* Aquí se eliminan valores menores que 0 para el ángulo de la baliza a */
  /* detectar y el pixel de la baliza                                     */
  thetab = atan2((by - ccy),(bx - ccx));
  if (thetab < 0) thetab += (2*PI);

  /* Cálculo del pixel: Cuando thetab  > cthetamascvmedios, tenemos 0 dentro */
  /* del campo visual y por tanto discontinuidad en los angulos              */
  if (thetab  > cthetamascvmedios)
    pixel_baliza = PIXELS_CAMARA-(PIXELS_CAMARA*(thetab-cthetamenoscvmedios)/
(CAMPO_VISUAL*DEGTORAD));
  else
    pixel_baliza = PIXELS_CAMARA*(cthetamascvmedios-thetab)/
(CAMPO_VISUAL*DEGTORAD);

  if ((db <= ALCANCE_C) && (db > 0) &&
      (((cthetamenoscvmedios <= cthetamascvmedios) &&
(cthetamenoscvmedios <= thetab) && (thetab <= cthetamascvmedios)) ||
       ((cthetamascvmedios <= cthetamenoscvmedios) &&
((thetab <= cthetamascvmedios) || (cthetamenoscvmedios <= thetab))))) {
    linea_c[(int)(pixel_baliza)] = bcolor;
  }

  return 0;
}


double calcula_distancia(char pixel_im1, char *imagen2, int i) {

  int pixel, iguales;
  double distancia, ascendente, descendente;

  distancia = 0;
  ascendente = 0; descendente = 0;
```

```
  if (i>0) {
    /* DESCENDENTE, recorre los pixels anteriores de la imagen calculada */
    /* buscando uno igual al dado si no lo encuentra añade la distancia  */
    /* máxima entre pixels                                               */
    iguales = 0; pixel = i-1;
    while ((iguales==0) && (pixel>=0)) {
      if (pixel_im1 == imagen2[pixel]) {
descendente = descendente + (i-pixel);
iguales = 1;
      }
      else pixel --;
    }
    if (iguales==0) descendente = descendente + PIXELS_CAMARA;
  }
  else descendente = descendente + PIXELS_CAMARA;

  if (i<PIXELS_CAMARA) {
    /* ASCENDENTE, recorre los pixels posteriores de la imagen calculada */
    /* buscando uno igual al dado si no lo encuentra añade la distancia  */
    /* máxima entre pixels                                               */
    iguales = 0; pixel = i+1;
    while ((iguales==0) && (pixel<PIXELS_CAMARA)) {
      if (pixel_im1 == imagen2[pixel]) {
ascendente = ascendente + (pixel-i);
iguales = 1;
      }
      else pixel ++;
    }
    if (iguales==0) ascendente = ascendente + PIXELS_CAMARA;
  }
  else ascendente = ascendente + PIXELS_CAMARA;

 if (descendente<ascendente) distancia = distancia + descendente;
 else distancia = distancia + ascendente;

 return distancia;
}


int compara_con_imagen_teorica(int m) {
  /* Calcula la distancia entre la imagen de entrada y la calculada según el */
  /* mapa. Si coinciden distancia = 0. Si no coinciden recorre la imagen     */
```

```
  /* calculada para ver si la coincidencia existe entre pixels anteriores o  */
  /* posteriores. La comparación sólo se realiza para posiciones distintas   */
  /* de blanco. La probabilidad obtenida del cálculo se representa en un     */
  /* fichero con valores de grises de 0 = MUY DIFERENTES a 255 = IGUALES     */

  int i, num_balizast, num_balizasr, ind;
  double distancia_salr, distancia_salt, distancia_med, distancia_fin, probabilidad;

  inicializa_linea_camara();
  for (i=0; i<cbalizas; i++)
    comprueba_si_ve_baliza(i);

  distancia_salr = 0; distancia_salt = 0; num_balizast = 0; num_balizasr = 0;

  for (i=0; i<PIXELS_CAMARA; i++) {

    if ((buffer_i[i] != ' ') && (buffer_i[i] != linea_c[i])){
      num_balizasr ++;
      distancia_salr = distancia_salr + calcula_distancia(buffer_i[i], &linea_c[0], i)
    }

    if ((linea_c[i] != ' ') && (buffer_i[i] != linea_c[i])){
      num_balizast ++;
      distancia_salt = distancia_salt + calcula_distancia(linea_c[i], &buffer_i[0], i)
    }

  }

  if ((distancia_salr == 0) && (distancia_salt == 0)) distancia_med = 0;
  else {
    if (distancia_salr == 0)
      distancia_med = (distancia_salt/num_balizast);
    else {
      if (distancia_salt == 0)
distancia_med=(distancia_salr/num_balizasr);
      else
distancia_med=((distancia_salr/num_balizasr)+(distancia_salt/num_balizast))/2.0;
    }
  }


   distancia_fin = distancia_med/16;
```

```
   probabilidad = exp(distancia_fin*distancia_fin*(-1));

  if (imagen_en_blanco == 1)
    probabilidad = (0.499999*probabilidad)+0.000001;
  else
    probabilidad = (0.65*probabilidad) + 0.30;

  muestra[m] = probabilidad;

  return 0;
}


int imprime_max_y_min(int n) {

  int i, x, y, theta;
  char header[70], nombre[80], dat_rec[80];
  int fichero, tplot_file;
  double cubo_acum_top[COLUMNAS][FILAS];
  int irojo, iverde, iazul;
  unsigned char red, green, blue;
  Stop_prob max_prob, min_prob;
  Sficheros Top[COLUMNAS][FILAS], Bottom[COLUMNAS][FILAS],
Middle[COLUMNAS][FILAS];

  max_prob.max_min_pos.xy.x = 99;
  max_prob.max_min_pos.xy.y = 99;
  max_prob.max_min_pos.theta = 999;
  max_prob.max_min_prob = 0;
  min_prob.max_min_pos.xy.x = 99;
  min_prob.max_min_pos.xy.y = 99;
  min_prob.max_min_pos.theta = 999;
  min_prob.max_min_prob = 1;

  for (y = (FILAS -1); y >= 0; y--) {
    for (x = 0; x < COLUMNAS; x++) {
      cubo_acum_top[x][y] = 0;
      Top[x][y].prob_fichero = 0;
      Top[x][y].n_ocurr = 0;
      Bottom[x][y].prob_fichero = 0;
      Bottom[x][y].n_ocurr = 0;
      Middle[x][y].prob_fichero = 0;
      Middle[x][y].n_ocurr = 0;
```

```
    }
  }

  sprintf(nombre,"pgms/CC-FINAL%d.ppm",n);
  fichero = open(nombre, O_CREAT|O_WRONLY, S_IRUSR|S_IWUSR);
  if (fichero < 0) {
    printf ("IMPRIME MAX Y MIN: No puedo abrir el fichero %s\n", &nombre[0]);
    exit (-1);
  }

  sprintf(nombre,"dat/topplot%d.dat",n);
  tplot_file = open(nombre, O_CREAT|O_WRONLY, S_IRUSR|S_IWUSR);
  if (tplot_file < 0) {
    printf ("IMPRIME MAX Y MIN: No puedo abrir el fichero %s\n", &nombre[0]);
    exit (-1);
  }

  sprintf(header, "P6\n%d\n%d\n255\n", COLUMNAS, FILAS);
  write(fichero,&header[0],strlen(header));

  for (i=0; i<NMUESTRAS; i++) {
    x = coord_muestra[i].xy.x;
    y = coord_muestra[i].xy.y;
    theta = coord_muestra[i].theta;

    if (max_prob.max_min_prob < muestra[i]) {
  max_prob.max_min_pos.xy.x = coord_muestra[i].xy.x;
  max_prob.max_min_pos.xy.y = coord_muestra[i].xy.y;
  max_prob.max_min_pos.theta = coord_muestra[i].theta;
  max_prob.max_min_prob = muestra[i];
    }
    if (min_prob.max_min_prob > muestra[i]) {
  min_prob.max_min_pos.xy.x = coord_muestra[i].xy.x;
  min_prob.max_min_pos.xy.y = coord_muestra[i].xy.y;
  min_prob.max_min_pos.theta = coord_muestra[i].theta;
  min_prob.max_min_prob = muestra[i];
    }
    if (muestra[i] >= 0.9) {
      if (cubo_acum_top[x][y] < muestra[i])
cubo_acum_top[x][y] = muestra[i];
      Top[x][y].prob_fichero += muestra[i];
      Top[x][y].n_ocurr ++;
    }
```

```
    else {
      if (muestra[i] < 0.5) {
Bottom[x][y].prob_fichero += muestra[i];
Bottom[x][y].n_ocurr ++;
      }
      else {
Middle[x][y].prob_fichero += muestra[i];
Middle[x][y].n_ocurr ++;
      }
    }
  }


  for (y = (FILAS -1); y >= 0; y--) {
    for (x = 0; x < COLUMNAS; x++) {
      if (Top[x][y].n_ocurr == 0)
irojo = 0;
      else
irojo = (int)(Top[x][y].prob_fichero*255/Top[x][y].n_ocurr);
      if (Bottom[x][y].n_ocurr == 0)
iverde = 0;
      else
iverde = (int)(Bottom[x][y].prob_fichero*255/Bottom[x][y].n_ocurr);
      if (Middle[x][y].n_ocurr == 0)
iazul = 0;
      else
iazul = (int)(Middle[x][y].prob_fichero*255/Middle[x][y].n_ocurr);
      red = (unsigned char) irojo;
      green = (unsigned char) iverde;
      blue = (unsigned char) iazul;
      if (cubo_acum_top[x][y] == max_prob.max_min_prob)
green = red;
      write(fichero, &red, sizeof(unsigned char));
      write(fichero, &green, sizeof(unsigned char));
      write(fichero, &blue, sizeof(unsigned char));
      sprintf(dat_rec, "%.2f %.2f %.8f\n",
      (double)(x), (double)(y), cubo_acum_top[x][y]);
      write(tplot_file, &dat_rec[0], strlen(dat_rec));
    }
  }

  printf("MAX PROB Cubo %d %.8f\n", n, max_prob.max_min_prob);
  printf("MIN PROB Cubo %d %.8f\n", n, min_prob.max_min_prob);
```

```
  for (i=0; i<NMUESTRAS; i++) {
    if (muestra[i] == max_prob.max_min_prob)
      printf("MAXIMA PROBABILIDAD Cubo %d x=%.2f y=%.2f theta=%.2f %.8f\n",
     n, coord_muestra[i].xy.x, coord_muestra[i].xy.y,
     coord_muestra[i].theta, max_prob.max_min_prob);
  }

  return 0;
}


int inicializa_histograma() {

  int i;

  for (i=0; i<=100; i++)
    histograma_c[i] = 0;

  return 0;

}


int imprime_histograma(int n) {

  int j;
  double k;
  char nombre[80], dat_rec[80];
  int plot_file;

  sprintf(nombre,"dat/histog%d.dat",n);
  plot_file = open(nombre, O_CREAT|O_WRONLY, S_IRUSR|S_IWUSR);
  if (plot_file < 0) {
    printf ("IMPRIME HISTOGRAMA: No puedo abrir el fichero %s\n", &nombre[0]);
    exit (-1);
  }
  for (j=0; j<=100; j++) {
    k = (double)(j);
    sprintf(dat_rec, "%.2f %.2f\n", k,((double)(histograma_c[j])));
    write(plot_file, &dat_rec[0], strlen(dat_rec));
  }
```

```
  return 0;
}

int calculos_estadisticos (int n) {

  int i, nposesv;
  double xmed, ymed, costmed, maxv, minv, desvtx, desvty, desvtcost, pposesv;

  xmed = 0, ymed = 0; costmed = 0;
  maxv = 0; minv = 1;
  nposesv = 0;

  for (i=0; i<NMUESTRAS; i++) {
    xmed = xmed + coord_muestra[i].xy.x;
    ymed = ymed + coord_muestra[i].xy.y;
    costmed = costmed + cos((coord_muestra[i].theta*DEGTORAD));
    if (muestra[i] < minv) minv = muestra[i];
    if (muestra[i] > maxv) maxv = muestra[i];
    if (muestra[i] > 0.8) nposesv++;
  }

  xmed = xmed / NMUESTRAS;
  ymed = ymed / NMUESTRAS;
  costmed = costmed / NMUESTRAS;

  desvtx = 0; desvty = 0; desvtcost = 0;

  for (i=0; i<NMUESTRAS; i++) {
    desvtx =
desvtx + ((coord_muestra[i].xy.x - xmed)*(coord_muestra[i].xy.x - xmed));
    desvty =
desvty + ((coord_muestra[i].xy.y - ymed)*(coord_muestra[i].xy.y - ymed));
    desvtcost =
desvtcost + ((cos((coord_muestra[i].theta*DEGTORAD)) - costmed)*
(cos((coord_muestra[i].theta*DEGTORAD)) - costmed));
  }

  desvtx = sqrt(((double)(1.0/NMUESTRAS))*desvtx);
  desvty = sqrt(((double)(1.0/NMUESTRAS))*desvty);
  desvtcost = sqrt(((double)(1.0/NMUESTRAS))*desvtcost);

  pposesv = ((double)(nposesv)) * 100.0 / ((double)(NMUESTRAS));
```

```
  printf("Cubo %d: Max Verosimilitud = %.8f Min Verosimilitud = %.8f\n",n,maxv,minv);
  printf("Media x = %.8f\n", xmed);
  printf("Media y = %.8f\n", ymed);
  printf("Media cost = %.8f\n", costmed);
  printf("Desv tipica x = %.8f\n", desvtx);
  printf("Desv tipica y = %.8f\n", desvty);
  printf("Desv tipica cost = %.8f\n", desvtcost);
  printf("Porcentaje Poses verosimiles = %.8f\n", pposesv);

  dispx[n] = desvtx; dispy[n] = desvty; dispt[n] = desvtcost;
  pos_veros[n] = pposesv;

  return 0;

}

int imprime_calculos(n) {

  int j;
  char nombre[80], dat_rec[80];
  int plot_file1, plot_file2, plot_file3, plot_file4;

  sprintf(nombre,"dat/histogx.dat");
  plot_file1 = open(nombre, O_CREAT|O_WRONLY, S_IRUSR|S_IWUSR);
  if (plot_file1 < 0) {
    printf ("IMPRIME CALCULOS: No puedo abrir el fichero %s\n", &nombre[0]);
    exit (-1);
  }

  sprintf(nombre,"dat/histogy.dat");
  plot_file2 = open(nombre, O_CREAT|O_WRONLY, S_IRUSR|S_IWUSR);
  if (plot_file2 < 0) {
    printf ("IMPRIME CALCULOS: No puedo abrir el fichero %s\n", &nombre[0]);
    exit (-1);
  }

  sprintf(nombre,"dat/histogt.dat");
  plot_file3 = open(nombre, O_CREAT|O_WRONLY, S_IRUSR|S_IWUSR);
  if (plot_file3 < 0) {
    printf ("IMPRIME CALCULOS: No puedo abrir el fichero %s\n", &nombre[0]);
    exit (-1);
  }
```

```
  sprintf(nombre,"dat/histogv.dat");
  plot_file4 = open(nombre, O_CREAT|O_WRONLY, S_IRUSR|S_IWUSR);
  if (plot_file4 < 0) {
    printf ("IMPRIME CALCULOS: No puedo abrir el fichero %s\n", &nombre[0]);
    exit (-1);
  }

  for (j=0; j<n; j++) {
    sprintf(dat_rec, "%d %.2f\n", j, dispx[j]);
    write(plot_file1, &dat_rec[0], strlen(dat_rec));
    sprintf(dat_rec, "%d %.2f\n", j, dispy[j]);
    write(plot_file2, &dat_rec[0], strlen(dat_rec));
    sprintf(dat_rec, "%d %.2f\n", j, dispt[j]);
    write(plot_file3, &dat_rec[0], strlen(dat_rec));
    sprintf(dat_rec, "%d %.2f\n", j, pos_veros[j]);
    write(plot_file4, &dat_rec[0], strlen(dat_rec));
  }

  return 0;

}


/* Programa Principal */

int main(int argc, char ** argv) {

  int i, n, semilla;
  time_t aclock;

  time(&aclock);
  semilla = aclock;
  srand(semilla);

  /* Toma variables de entrada, las verifica y asigna adecuadamente */


  if ((argc != 1) && (argc != 3)) {
    printf ("MAIN: Numero de argumentos erroneo %d\n", argc);
    printf ("MAIN: Uso 'verosimilitud [nombre_mapa] [nombre_accion_e_imagen]'\n");
    exit(-1);
  }
```

```
if (argc == 1) {
  fichero_mapa = "mi_mapa";
  fichero_imagen = "accion_e_imagen.txt";
}
else {
  fichero_mapa = argv[1];
  fichero_imagen = argv[2];
}

/* Variables de control del número de líneas y balizas del fichero de entrada */
cbalizas = 0; clineas = 0;

/* Abre los ficheros de entrada, inicializa el buffer y carga el mapa leido */
/* y crea un fichero con el campo visual dentro del mapa                     */
mapa = fopen(fichero_mapa, "r");
imagen = fopen(fichero_imagen, "r");

if (mapa == NULL) {
  printf ("MAIN: No puedo abrir el mapa\n");
  exit (-1);
}

cuenta_balizas();
if (fclose(mapa) != 0) {
  printf ("MAIN: No puedo cerrar el mapa\n");
  exit (-1);
}

mapa = fopen(fichero_mapa, "r");
if (mapa == NULL) {
  printf ("MAIN: No puedo abrir el mapa por segunda vez\n");
  exit (-1);
}
inicializa_mapa();
lee_mapa();
imprime_mapa();

if (imagen == NULL) {
  printf ("MAIN: No puedo abrir el fichero de imagenes y acciones\n");
  exit (-1);
}
```

```
  calcula_gaussianas();
  n = 0;
  for (i=0; i<NMUESTRAS; i++)
genera_muestras(i,n);

  while (fin_imagen != 'F') {

    inicializa_imagen();
    lee_imagen();
    if (fin_imagen!='F') {
      inicializa_histograma();

      if (n>0) {
redistribuye_muestras(n);
for (i=0; i<NMUESTRAS; i++)
  desplaza_muestra(i);
      }

      for (i=0; i<NMUESTRAS; i++) {
if ((coord_muestra[i].xy.x < COLUMNAS) && (coord_muestra[i].xy.y < FILAS)
 && (coord_muestra[i].xy.x >= 0) && (coord_muestra[i].xy.y >= 0 )) {
  i_camara.theta = coord_muestra[i].theta;
  i_camara.xy.y = coord_muestra[i].xy.y;
  i_camara.xy.x = coord_muestra[i].xy.x;
  compara_con_imagen_teorica(i);
}
else {
  muestra[i] = 0.000001;
}
calcula_muestra_acumulada(i);
actualiza_histograma(i);
      }

      imprime_max_y_min(n);
      imprime_histograma(n);
      calculos_estadisticos(n);

      n++;
    }
  }

  imprime_calculos(n);
```

```
  fcloseall();

  return 0;
}
```