Departamento de Sistemas Telemáticos y Computación
Universidad Rey Juan Carlos


# Tesis Doctoral

# Aportaciones a la auto-localización visual de robots autónomos con patas

Francisco Martín Rico
francisco.rico@urjc.es

*Directores de Tesis:*
Vicente Matellán Olivera
vicente.matellán@urjc.es
Jose María Cañas Plaza
josemaria.plaza@urjc.es

**Francisco Martín Rico**

Departamento de Sistemas Telemáticos y Computación
Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos
Av. Tulipán s/n
Móstoles 28933 Madrid

*Correo electrónico:* francisco.rico@urjc.es
*Web:* http://gsyc.es/ fmartin

Autorizamos la defensa de la tesis doctoral *Aportaciones a la auto-localización visual de robots autónomos con patas* cuyo autor es D. Francisco Martín Rico.

Vicente Matellán Olivera  José María Cañas Plaza

Móstoles, 25 de Junio de 2008

TESIS DOCTORAL: Aportaciones a la auto-localización visual de robots autónomos con patas

AUTOR: D. Francisco Martín Rico
DIRECTORES: Dr. Vicente Matellán Olivera
Dr. José María Cañas Plaza

El tribunal nombrado para juzgar la Tesis arriba indicada, compuesto por los siguientes doctores:

PRESIDENTE:

VOCALES:

SECRETARIO:

acuerda otorgarle la calificación de:

Móstoles, de de

El Secretario del Tribunal

*A mi familia*

# Agradecimientos

Al final de este camino recorrido me gustaría agradecer a todas las personas la ayuda que me han prestado, día a día, hasta llegar al final. Ha sido un camino largo y duro, que se ha prolongado más de lo que yo hubiese querido. Espero poder agradecerles algún día la paciencia y comprensión que me han brindado.

Mis primeras palabras de gratitud son para tí, Laura, porque te he pedido más de lo que te he dado y, a tu manera, me has ayudado a andar este camino. Nos hemos quedado muy cerca del final... Esta tesis es en parte también tuya porque la hemos sufrido juntos.

También me gustaría agradecer a mis padres y a mi hermana su cariño y su infinita paciencia. Ellos me han motivado a llegar hasta aquí y me han empujado cuando las piernas flaqueaban. Nunca le agradeceré suficiente a mi madre que me hiciera comprender que no hay meta que no pudiera alcanzar.

Quisiera agradecerles también su trabajo, tiempo y, sobre todo, su comprensión a Vicente y José María. Sus valiosos comentarios son un tesoro que me ayudan que cada día intente ser mejor en lo que hago. Gracias.

También quisiera agradecer a toda la gente de GSyC por hacerme sentir como en mi segundo hogar. Nunca hubiera imaginado compañeros mejores.

También quisiera agradecer a todos los integrantes del equipo TeamChaos por su trabajo en el software desarrollado en esta tesis. En especial, al Dr. Humberto Martínez Barberá, por su trabajo, sus valiosos comentarios y consejos.

Gracias a todos.

# Resumen

La capacidad de auto-localización es fundamental en robótica móvil. Esta capacidad permite que un robot se pueda desplazar por un entorno entre puntos globalmente definidos. También permite que el robot realice tareas que dependan de su posición en el mundo. Si un robot móvil no contara con esta información, su funcionamiento se vería muy limitado.

Esta tesis se enfoca en resolver este problema en robots con patas que usan la visión como principal fuente de información. El objetivo fundamental de esta tesis es aportar soluciones al problema de la auto-localización de robots móviles, haciendo hincapié en los métodos aplicados en el desarrollo de estas soluciones.

En concreto, en esta tesis se proponen varias soluciones que combinan algoritmos markovianos difusos y uno o varios Filtros Extendidos de Kalman. Las combinaciones que se proponen toman las mejores características de ambos algoritmos y consiguen una auto-localización confiable, robusta y ligera.

En esta tesis también se describen los métodos que se han aplicado al desarrollar las soluciones de auto-localización. Asimismo, se describen las herramientas necesarias para aplicar estos métodos.

Las soluciones que se proponen en esta tesis se implementado en el robot AIBO. Éste es un robot móvil con aspecto de perro, que se desplaza mediante 4 patas y recibe la información del entorno principalmente de una cámara situada en su cabeza. El entorno donde se han aportado los resultados más relevantes de esta tesis es la competición de la Robocup, donde equipos de robots compiten entre ellos de manera autónoma en un juego similar al ”fútbol”. Este entorno es un escenario muy exigente donde se pueden probar y contrastar las tecnologías relacionadas con la robótica móvil.

Estas soluciones se integran en el software del equipo *TeamChaos* y se han usado satisfactoriamente en la edición del German Open celebrada en Hannover en el año 2007.

# Abstract

The ability of self-localization is crucial in mobile robotics. This ability allows a robot to move in an environment between points globally defined. It also allows the robot to perform tasks that depends on its position in the world. If a mobile robot does not know this information, its operation would be very limited.

This thesis focuses on resolving this problem in robots with legs which use vision as the primary source of information. The main goal of this thesis is to provide solutions to the problem of self-localization in mobile robots, with emphasis on the methods used in developing these solutions.

Specifically, in this thesis several solutions are proposed that combine fuzzy markovian algorithms and one or more Extended Kalman Filters. The combinations proposed take the best features of both algorithms and get a reliable, robust and lightly self-locating.

This thesis also describes the methods to be applied to develop solutions for self-localization. It describes the needed tools to implement these methods.

The solutions proposed in this thesis are implemented in the AIBO robot. This is a mobile robot with dog appearance which moves using 4 legs and receives information from the environment using mainly camera allocated in his head. The environment where most relevant results of this thesis have been provided is the Robocup competition, where teams of robots compete among them independently in a game similar to ”soccer”. In this complicated environment several technologies related to mobile robotics can be tested and contrasted.

These solutions are integrated into the software of *TeamChaos* and they have been used satisfactorily in the edition of the German Open held in Hanover in 2007.

# ÍNDICE GENERAL

# ÍNDICE DE FIGURAS

# CAPÍTULO 1

# Introducción

La Robótica es un campo de investigación en el que se han producido grandes avances en los últimos años. Cada vez es más común la aplicación de robots en tareas peligrosas (desactivación de bombas), poco accesibles para el ser humano (exploración espacial) o monótonas y fatigosas (cadenas de montaje industriales).

La Robótica Móvil se encuentra dentro del campo de la investigación de la Robótica y se centra en aquellos robots que tienen la capacidad de desplazarse por su entorno, que en muchos casos comparte con el ser humano. Debido a esto surgen varios problemas que deben resolverse (navegación, interacción hombre-robot, etc.).

La auto-localización es uno de estos problemas, y trata de dotar a un robot móvil de la capacidad de conocer su posición en su entorno. Esta capacidad se usa para la navegación o la realización de tareas dependientes de la posición del robot.

Las posibles soluciones al problema de la auto-ocalización se ven fuertemente condicionadas al tipo de sensores (ultrasonidos, láser, cámaras, etc.) y al tipo de actuadores que se emplean en el desplazamiento (ruedas, hélices, patas, etc.).

El objetivo de esta tesis es aportar soluciones al problema de la auto-localización visual de un robot autónomo que usa como principales actuadores motrices un conjunto de patas. Un matiz diferenciador entre esta propuesta y trabajos relacionados es

que está enfocada desde el punto de vista ingenieril, poniendo énfasis en los métodos y las herramientas necesarias para resolver este problema.

Éste capítulo prentede dar contexto a esta tesis. En la sección 1.1 se introduce el concepto de robótica profundizando posteriormente, en la sección 1.2, en el campo de la robótica móvil, presentando los robots históricamente más relevantes en el campo relacionado con este trabajo. La robótica móvil presenta una serie de problemas que se consideran clásicos y se resumen en la sección 1.3. En particular, y como tema central de la tesis, se presentan los aspectos fundamentales del problema de la autolocalización en la sección 1.4. En la sección 1.5 se presentan los objetivos de esta tesis, así como las características del problema a resolver.

## 1.1. Robótica

La *Robótica* es el área encaminada a diseñar y construir aparatos y sistemas capaces de realizar tareas propias de un ser humano. Las tecnologías en las que se apoya podrían resumirse en: la mecánica, la electrónica y la informática, aunque admite la aplicación de múltiples disciplinas de ingeniería.

Existen multitud de definiciones de los términos *Robótica* y *Robot*, dos términos íntimamente ligados. A continuación enumeramos algunas de las más destacadas:

- Según la **Real Academia de la Lengua Española**, la *Robótica* es "*la técnica que aplica la informática al diseño y empleo de aparatos que, en sustitución de personas, realizan operaciones o trabajos, por lo general en instalaciones industriales*". Asimismo, define *Robot* como "*la máquina o ingenio electrónico programable, capaz de manipular objetos y realizar operaciones antes reservadas sólo a las personas*".

- Según la **wikipedia** [1], la *Robótica* es "*la rama de la tecnología, que estudia el diseño y construcción de máquinas capaces de desempeñar tareas repetitivas o peligrosas para el ser humano*". La misma fuente define a un *Robot* como un "*dispositivo electrónico y generalmente mecánico, que desempeña tareas*

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

*automáticamente, ya sea de acuerdo a supervisión humana directa,o a través de un programa predefinido o siguiendo un conjunto de reglas generales*".

- Según la **Asociación de Industrias Robóticas (RIA)**[2], un *Robot* es "*un manipulador reprogramable multifuncional diseñado para mover materiales, partes, herramientas o dispositivos especializados, a través de movimientos programados variados para la realización de una variedad de tareas*". Esta definición es muy limitada, ya que no contempla la posibilidad de que un robot se desplace.

- Quizás una de las definiciones más ajustadas del término *robot móvi* para el objetivo de esta tesis la realiza **Arkin** [Ark98] según el cual "*un robot (inteligente) es una máquina capaz de extraer información del entorno y de usar el conocimiento que posee sobre el mundo para moverse de forma segura y con un propósito*".

Básicamente, un robot es una computadora equipada con sensores para obtener información de su entorno y con una serie de actuadores que le permite interaccionar con él. Esta computadora se programa para reaccionar, mediante sus actuadores, conforme a la información sensorial que percibe.

Los robots realizan tareas peligrosas, repetitivas o tediosas, que de otra manera tendría que realizar el ser humano. Son capaces de realizar una tarea monótona continuamente y hacerlo con gran precisión, como es el caso de los brazos robóticos en entornos industriales. En la exploración submarina o espacial (figura 1.1) también se usan robots para realizar tareas que pondrían en peligro a seres humanos.

Los robots también tienen su sitio en el ámbito doméstico. Cada vez es más posible que haya robots a nuestro alrededor que sean mascotas o que nos ayuden en las tareas cotidianas del hogar. Un ejemplo de este último caso es el robot Roomba[3], un robot móvil que realiza tareas de limpieza en el hogar y que se ha comercializado con gran éxito en los últimos años.

Una de las divisiones mas generales de robot es la de robots industriales y robots móviles. Los robots industriales más extendidos son los robots manipuladores

[2] http://www.roboticsonline.com/
[3] http://www.irobot.com/

Figura 1.1: MARS Rover.

dotados con brazos robóticos. Estos robots modifican el entorno que le rodea, que normalmente se encuentra altamente estructurado y delimitado. Carecen de capacidad de desplazamiento (generalmente están anclados en una posición) o se encuentra muy limitada. Sus movimientos son sumamente precisos y normalmente precalculados, aunque en algunos casos están equipados con sensores para ayudarles en la manipulación de los elementos de su entorno.

Los robots móviles son aquellos que poseen mecanismos que les permiten desplazarse por su entorno. Están equipados de sensores que les permiten percibir los elementos que están a su alrededor para poder realizar tareas de navegación y de interacción con su entorno. En este tipo de robots se ha centrado esta tesis, por lo que en la siguiente sección se describen en profundidad las características y los problemas que presentan este tipo de robots.

## 1.2. Robótica Móvil

Los robots móviles se caracterizan por su capacidad de desplazarse por el entorno que les rodea, no estando anclados a una posición física fija. Este tipo de robots recibe actualmente gran atención y es el principal foco de investigación en el campo de la Robótica. La práctica totalidad de las universidades más importantes tiene uno o más grupos que enfocan su investigación en el campo de la Robótica Móvil. Existen

Figura 1.2: UAV MQ-8B Fire Scout (izquierda) y UV Bluefin-12(derecha).

robots móviles en la industria, en el ejército o en cuerpos de seguridad. También hay robots móviles que se comercializan para entretenimiento o labores agrícolas, p.e.

Los robots móviles son intuitivamente clasificables atendiendo al entorno donde se desplazan. Hay robots aéreos, acuáticos y terrestres. Existen también un tipo de robots usados en misiones por las agencias espaciales. Estos robots se usan en la exploración de superficies de otros planetas, donde sustituyen a los astronautas debido al enorme riesgo que entrañan estas misiones. Estos robots son asimilables a los robots terrestres, ya que los sensores y la forma de desplazarse son similares.

Los robots aéreos (imagen izquierda de la figura 1.2) son normalmente llamados *vehículos aéros no tripulados* (*unmanned aerial vehicles, UAVs*). Pueden ser controlados por control remoto o volar autónomamente. Actualmente se usan principalmente por el ejército en tareas de reconocimiento. También hay UAVs en tareas civiles tales como la lucha contra incendios, reconocimiento en escenarios de desastres, etc.

Los robots acuáticos (imagen derecha de la figura 1.2) son llamados *vehículos autónomos submarinos* (*autonomous underwater vehicle, AUV*). Los AUVs son robots submarinos no tripulados que se alimentan por baterías o celdas de combustible (dispositivo de conversión electroquímico) y realizan tareas bajo el agua hasta profundidades de 6000 metros, típicamente. Se usan principalmente en investigaciones oceanográficas y en la construcción e inspección de construcciones submarinas (conductos, cables y prospecciones submarinas).

Los robots móviles terrestres tienen un campo de aplicación mucho más amplio que los tipos que acabamos de describir. Pueden realizar tareas de reconocimiento en campo abierto, labores agrícolas, tareas de navegación en entornos de oficina o incluso realizar tareas de teleasistencia, limpieza o entretenimiento en hogares. Un robot

móvil terrestre puede operar en entornos *interiores* y *exteriores*. Existen grandes diferencias que influyen en la operación del robot. Los entornos interiores generalmente presentan unas condiciones de luz controladas, la superficie es plana y el entorno está estructurado. Los entornos de oficinas son un ejemplo de este tipo de entorno donde se han realizado multitud de estudios en robótica móvil.

En los entornos exteriores, en cambio, no están controladas las condiciones de luz. Además, la superficie puede presentar irregularidades y el robot puede verse afectado por factores climatológicos (viento, lluvia, etc.).

Este trabajo se centra en los robots móviles terrestres que operan en entornos interiores, tanto en entornos de oficina como en entornos altamente ingenierizados, como el campo de la RoboCup, que se describirán próximamente.

Los robots móviles terrestres se pueden clasificar dependiendo del tipo de actuadores que usen para desplazarse en *robots con ruedas* y *robots con patas*. Los robots con ruedas tienen varias ventajas. En primer lugar, son más fáciles de controlar, ya que cada rueda tiene como máximo un grado de libertad y un robot tiene como máximo dos grados de libertad. En segundo lugar, estos robots son más robustos, estables (con más de 2 ruedas) y veloces. Este tipo de robot tiene como principal desventaja la necesidad de que la superficie en la que se desplaza no tenga desniveles muy bruscos.

Los robots con patas, como el que se ha usado en esta tesis, se desplazan mediante un movimiento coordinado de cada una de las partes que componen las patas. Los robots que cuentan únicamente con dos patas se denominan bípedos y son estáticamente inestables. Su movimiento es muy complejo, ya que debe mantener el equilibrio mientras se desplaza para no caerse. Los robots que cuentan con más patas son más estables y el mantenimiento del equilibrio deja de ser crítico.

Los robots móviles terrestres pueden estar equipados con una gran variedad sensores. Los sensores son muy importantes en el campo de la Robótica, y aún más en el campo de la Robótica Móvil Autónoma debido que el robot se ha de desplazar por su entorno. Para realizar esta labor de manera autónoma es imprescindible que sea capaz de obtener del entorno toda la información que sea relevante. De otra manera el robot se vería seriamente limitado.

Los sensores miden magnitudes físicas del entorno (luminosidad, sonido, temperatura, distancia, etc.). Esta información, por estar midiendo magnitudes del mundo

real, esta sujeta a todo tipo de incertidumbres y hace que sea imprecisa, ambigua, contradictoria y que contenga gran cantidad de ruido. El robot debe ser capaz de interpretar estos valores para que le sean de utilidad.

Existe una enorme variedad de sensores que pueden formar parte de un robot móvil. No existe el sensor perfecto, y todos tienen sus ventajas y sus inconvenientes. A continuación se enumeran algunos de los más relevantes y se describen sus características:

**Brújulas** Miden el componente horizontal del campo magnético terrestre y así determinan la orientación absoluta del robot. Son útiles en tareas de navegación, aunque su principal desventaja es que pueden verse afectadas por objetos metálicos cercanos.

**Odómetros** Son sensores propioceptivos que miden la posición y el movimiento de partes del robot. Se basan en detectar, mediante células fotoeléctricas, el dibujo que se encuentra en unos pequeños discos situados en el eje de giro de un elemento móvil. Se usan generalmente para calcular la odometría en robots con ruedas. Estos sensores pueden sufrir de errores sistemáticos (imprecisiones en el cálculo del diámetro de las ruedas), o esporádicos (deslizamiento de las ruedas).

**Sensores Táctiles** Estos sensores detectan el contacto con algún elemento del entorno. Muchos son capaces de medir la magnitud de la presión que se produce en el sensor y no solo la presencia o no de presión. Es habitual que se usen para recubrir partes del robot y detectar así el contacto con obstáculos. Aún así, el uso de estos sensores debe ser el último recurso para evitar obstáculos, ya que es preferible que los obstáculos se detecten antes de tocarlos.

**Sensores de infrarrojos** Estos sensores son dispositivos electrónicos capaces de medir la radiación electromagnética infrarroja de los cuerpos en su campo de visión. Se basan en emitir un haz de luz en el espectro infrarrojo y medir la cantidad reflejada por los obstáculos cercanos. En teoría, cuanta mayor cantidad de radiación mida, menor es la distancia a un obstáculo. Uno de sus inconvenientes es que dependen de la reflectividad de la superficie de los obstáculos. Si el

obstáculo es completamente negro, es posible que sea difícil de detectar. Además, no tienen mucho alcance y se ven afectados por las condiciones luminosas del entorno.

**Sensores de ultrasonidos** Los ultrasonidos son ondas acústicas cuya frecuencia está por encima del límite perceptible por el oído humano. Los sensores de ultrasonidos se basan en emitir este tipo de ondas y detectar las que son reflejadas por los obstáculos cercanos. Un análisis de este reflejo aporta información de la distancia al obstáculo. El principal problema de este tipo de sensores es su resolución angular, ya que los pulsos emitidos tienen una forma cónica. Además, dependiendo de la forma del objeto, es posibles que la onda reflejada en un obstáculo no pueda retornar al sensor.

**Sensores láser** Este sensor se basa en la emisión de haces de luz y la detección de los reflejos en los obstáculos del entorno. Suelen ser muy precisos y fiables. Se están haciendo muy populares en el campo de la Robótica Móvil. Su principal desventaja es su elevado precio y coste energético.

**Cámaras** Las cámaras son los sensores más ricos del que dispone un robot. Una cámara produce una imagen, que no es más que una matriz de puntos que codifican la intensidad y el tipo de luz que refleja un objeto. La principal dificultad de este sensor es que la imagen ha de ser procesada para extraer información de ella. Este análisis de la imagen no es sencillo la mayor parte de las ocasiones y necesita de gran capacidad de cómputo.

**GPS** Este dispositivo es capaz de calcular su posición en relación a un eje de referencias absoluto. Se basa en detectar las señales que producen 24 satélites geosincrónicos que orbitan la tierra dos veces al día en una órbita muy precisa y transmiten la información a la tierra. Los dos principales inconvenientes son la precisión (alrededor de 10 metros) y que no se puede usar en entornos de interiores.

Para aportar contexto a este campo, a continuación se presentarán una serie de robots que han pasado a la historia por el avance que supusieron para el campo de la robótica móvil, y que tienen influencia de un modo u otro en el presente trabajo.

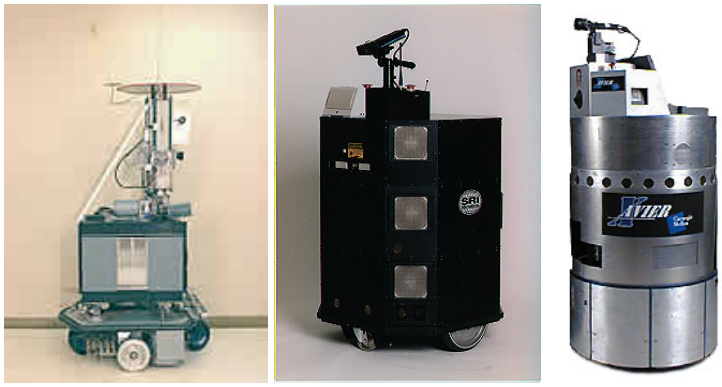

Figura 1.3: De izquierda a derecha el Robot shakey (1972), el Robot Flakey (1991) y elRobot Xavier (1993).

El primer robot que se presenta es Shakey [Nil84] (robot situado en la izquierda en la figura 1.3). Éste es el primer robot móvil que ”razonaba sus acciones”. Fue desarrollado entre 1966 y 1972 en el Stanford Research Institute (ahora SRI International). Este robot usa modelos simbólicos de Inteligencia Artificial (IA) para navegar en entornos de oficina, organizando sus tareas en sensar, pensar y actuar. Esto hace que sea lento en la reconstrucción íntegra del mundo y en el cálculo de los movimientos a realizar. Cuenta con ruedas para desplazarse, una cámara, sensores de distancia y detectores de choque. Este robot no tiene un funcionamiento completamente autónomo, y se conecta a computadoras DEC PDP-10 y PDP-15 por medio de conexiones de radio para determinar sus siguientes acciones.

El sucesor de Shakey en el SRI es el robot Flakey [SRK93] (robot situado en el centro de la figura 1.3), creado en 1984. Es un robot semicilíndrico que se desplaza gracias a dos ruedas motrices independientes. Como sensores, dispone de un anillo de 12 sensores de ultrasonido, odómetros en las ruedas, una cámara y un sensor láser, que le ofrece una rica información de profundidad en un área cercana al robot.

En este robot se desarrollaron técnicas de navegación autónoma. Para llevar a cabo las tareas de navegación, usaba técnicas basadas en comportamientos que se combinaban mediante lógica borrosa.

Otro robot que ha tenido gran influencia en los trabajos posteriores sobre robótica móvil es Xavier [SGH$^+$97] (robot situado en la derecha en la figura 1.3), de-

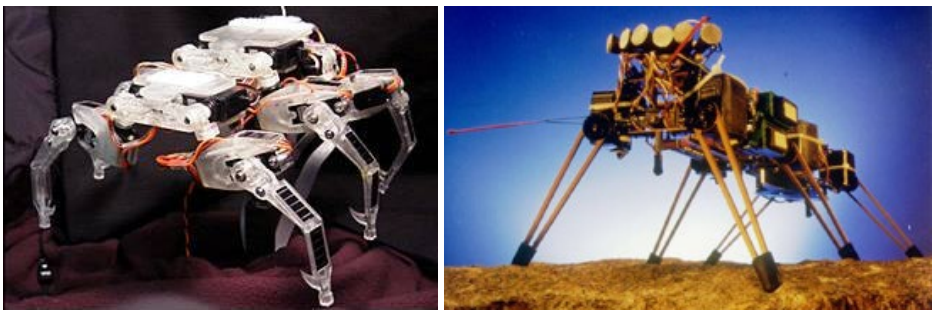

Figura 1.4: Robots araña (izquierda) y robot Genghis (derecha).

sarrollado en la universidad Carnegie Mellon desde 1993. El principal objetivo era desarrollar un robot móvil completamente autónomo que pudiera operar de manera fiable durante largos periodos de tiempo, aprender y adaptarse a su entorno y que pudiera interaccionar con la gente de una forma socialmente aceptable.

Xavier es un robot formado por una base cilíndrica de 61 centímetros de diámetro y se desplaza por medio de 4 ruedas. Cuenta con sensores de contacto, un anillo de sensores de ultrasonido alrededor del cilindro, un láser y una cámara. Tiene dos computadoras conectadas y cuenta con una tarjeta inalámbrica para comunicarse.

Ésta es una referencia muy significativa para esta tesis, pues sirvió para desarrollar técnicas probabilísticas que constituyen la base de parte de los trabajos de auto-localización realizados en esta tesis.

En los últimos años, los robots con patas se han generalizado, siendo la evolución natural de los robots con ruedas. La motivación de construir robots con patas aparece por el deseo de crear robots parecidos a los seres vivos y por sus características, que permiten desplazarse de manera mas efectiva en sitios que son inaccesibles a los robots con ruedas (escaleras, pendientes, etc.). Los primeros esfuerzos por crear robots similares a seres vivos se centran en robots con multitud de patas, parecidos a insectos, como se puede observar en la figura 1.4. Estos robots tienen gran estabilidad y la coordinación de las patas se realiza con una secuencia iterativa de posiciones precalculadas. Su movimiento surge a partir de la coordinación de todas las articulaciones del robot y es más complejo que los robots con ruedas, en los que el movimiento

Figura 1.5: Robot Genibo.

se produce simplemente incrementando la velocidad de giro de los motores de las ruedas.

En la evolución de los robots desde los robots con ruedas hasta los robots más actuales han surgido robots con patas similares a animales, que están fuera de ámbitos académicos o industriales y realizan funciones de compañía. Estos robots mascotas interactúan con el ser humano y le sirven de compañía. Este es el caso del robot AIBO de Sony o Genibo[4] (figura 1.5).

Estos robots con múltiples patas suponen un paso hacia el diseño de humanoides. La principal característica de los humanoides es su aspecto, semejante al ser humano. Además, su diseño se orienta a que tengan características similares al ser humano: suelen estar estructurados en cabeza, tronco y extremidades; usan como sensor principal la visión mediante cámaras situadas en posiciones cercanas a los ojos; etc.

Una de las principales características de un humanoide es que son robots bípedos. El hecho de contar con dos patas tiene como consecuencia que su movimiento sea aún mas complejo. Ya no tiene la estabilidad con la que cuentan los robots con varias patas. Además, en su movimiento el robot ha de mantener el equilibrio para no caerse. Por ahora la mayor parte de los robots se desplazan lentamente sin levantar ambos pies del suelo, situación que se produce en el ser humano cuando salta o corre. El hecho de que un robot separe los dos pies del suelo y mantenga posteriormente el equilibrio es un problema complejo de solucionar.

[4] http://genibo.dasarobot.com/english/

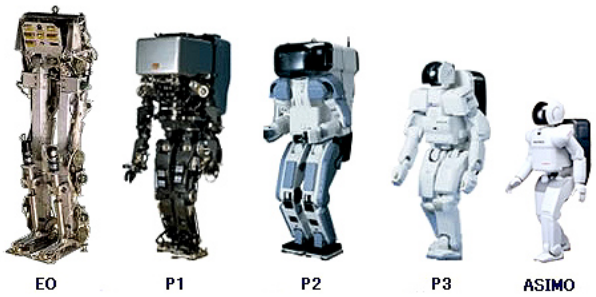


Figura 1.6: Evolución del robot Asimo entre 1993 y 2007.


Figura 1.7: Humanoide de Toyota (izquierda), QRIO(centro) y robot Nao (derecha).

Uno de los robots humanoides más famosos es Asimo[5], fabricado por Honda. Este proyecto comenzó en 1986, y los resultados más llamativos se obtuvieron principalmente entre 1993 y 2007 (figura 1.6), aunque su desarrollo sigue aún vivo hoy en día, existiendo una última versión capaz de correr a 6 km/h. Otros desarrollos de humanoides que se pueden destacar son el humanoide creado por Toyota, el robot QRIO o el robot Nao (figura 1.7). El robot Nao es especialmente interesante en el contexto de esta tesis, por ser la plataforma elegida para sustituir al robot AIBO en la RoboCup. Se trata de un robot que aún se encuentra en una fase inicial de desarrollo. Cuenta con varios sensores entre los que destaca una cámara VGA. Usa un sistema operativo Linux que se ejecuta en un procesador Geode.

A lo largo de esta sección hemos repasado las principales características de los robots móviles. También hemos descrito los elementos que los componen. En la últi-

[5]http://world.honda.com/ASIMO/

ma parte hemos presentado una serie de robots para ilustrar la evolución de los robots móviles, desde los primeros robots basados en ruedas y sensores de ultrasonidos, hasta los más actuales con forma humana y que basan su percepción en la visión. Este tipo de robots humanoides marcan la tendencia actual y se prevé que se desarrollen enormemente en el futuro.

En la siguiente sección se presentan las áreas activas de la Robótica Móvil, que tratan de resolver los problemas clásicos de este tipo de robots. Estos problemas derivan de su capacidad de desplazarse por su entorno.

## 1.3. Áreas activas de investigación en Robótica Móvil

La Robótica Móvil es un campo de investigación muy amplio donde actualmente se concentran gran parte de las investigaciones en Robótica. Dentro de este campo existen varias cuestiones abiertas. Estas cuestiones corresponden a cada uno de los elementos que hacen posible que un robot móvil se desplace por un entorno e interactúe con él. Alrededor de estas cuestiones se han formado áreas activas de investigación.

La primera taxonomía en que se pueden dividir estas áreas de investigación son las *cuestiones hardware* y las *cuestiones software*. Las *cuestiones hardware* están relacionadas con el diseño y la construcción de los robots móviles, así como los sensores y actuadores que pueden formar parte de él. Construir un robot implica decisiones como qué sistema de locomoción se ha de usar: ruedas, hélices, aletas, patas, etc. y cada una de estas opciones tendrá sus cuestiones particulares (cuántas patas, qué tipo de ruedas, etc.). Otra decisión importante cuando se diseña el hardware de un robot es con qué tipo y cuántos sensores se montarán en el robot: cámaras, láser, infrarrojos. Unir todo esto y hacerlo adecuado al entorno y tarea a realizar forma parte de los problemas hardware relacionados con la robótica móvil.

En esta investigación partimos de un robot comercial en el que no tenemos capacidad de decisión sobre los elementos que lo componen. Nos enfocamos en dotar a robots con patas de comportamientos y habilidades autónomas. Por esta razón, las cuestiones relativas a la construcción de robots no son objeto de nuestro interés. En

las cuestiones software, relacionadas con las habilidades de alto nivel de los robots, es donde se centra este trabajo.

Las *cuestiones software* abordan cuestiones de nivel cognitivo que persiguen dotar al robot de comportamientos y habilidades "inteligentes". Una mejor y más completa solución de estos problemas permite un mayor grado de autonomía y complejidad en las tareas que un robot lleva a cabo.

Como hemos comentado, esta investigación se centra en las cuestiones software de la Robótica Móvil. Un robot móvil ha de se capaz de desplazarse por su entorno e interactuar con él. Para que un robot sea capaz de realizar estas tareas, hay una serie de cuestiones que se han de tener en cuenta. Este conjunto de cuestiones se puede resumir, a nuestro juicio, en:

**Interacción hombre-máquina** Un robot móvil cuyo entorno deba compartirlo con un ser humano, ha de ser capaz de convivir con él de manera sociable. Debe ser capa de *interactuar* con él para relacionarse, obtener comandos de actuación y mostrar información. Esta interacción hombre-máquina ha de ser de tal forma que el ser humano sea capaz de aceptar la comunicación con el robot y entenderla de una manera agradable.

**Representación de entorno** Un robot móvil se desplaza por su entorno para realizar una tarea concreta. En algunos casos no necesita conocer el entorno que le rodea para realizar su labor. Por ejemplo, el robot *Roomba* es un robot móvil doméstico cuya tarea es la aspirar la suciedad de una habitación. Este robot no tiene una representación interna del entorno donde realiza su labor. Únicamente realiza movimientos aleatorios de avance y giro, a la vez que evita obstáculos.

En otros casos, las tareas que debe realizar dependen de su posición en el entorno. En este caso, es necesario tener una representación interna, o *mapa*. En el mapa se representan los elementos que un robot debe tener en cuenta para realizar su labor (paredes, muebles, obstáculos, objetivos, etc.). Por ejemplo, una de las tareas del robot Xavier es llevar el correo de un despacho a otro en un entorno de oficinas. Para realizar esta labor, el robot debe mantener un mapa del entorno donde se representan los distintos despachos. En este mapa, el

robot debe asociar cada despacho con una persona para poder identificar a qué posición debe desplazarse.

La representación interna del entorno y la construcción de mapas son cuestiones abiertas en Robótica Móvil. Existen multitud de soluciones a esta cuestión, aunque no hay ninguna que satisfaga todas las necesidades que pueda tener un robot móvil. La representación del entorno depende del uso que se va a hacer de él. En algunos casos es adecuada una *representación métrica* del entorno. Este tipo de mapas (figura 1.8) representan el entorno a partir de un eje de referencia. Todas las posiciones asociadas a cualquier elemento del entorno se representan como una coordenada respecto a este eje de referencia. Este tipo de representación tiene la ventaja de ser muy general.

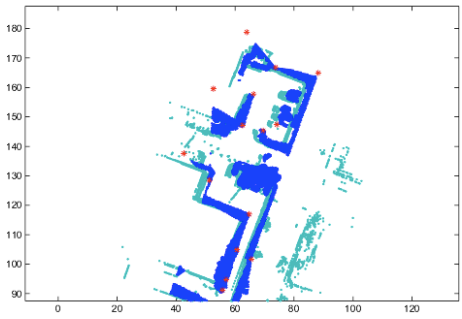


Figura 1.8: Mapa métrico de un entorno de interiores generado por un robot.

Aunque los mapas métricos son adecuados en gran cantidad de aplicaciones, en otros casos se requiere representar en el mapa las conexiones entre porciones del entorno con valor semántico sin necesitar gran precisión. En estos casos se emplean una *representación topológica*. Este tipo de mapas (figura 1.9) dividen el entorno en zonas con similares características y con significado semántico para el robot. La principal ventaja es su adaptación a la tarea que realiza el robot. Por otro lado, esta representación suele ser muy poco general.

También existen representaciones del entorno en forma de *rejilla*. Esta representación discretiza el entorno en un conjunto de celdas de similares dimensiones. Esta representación es útil cuando se aplican algoritmos que necesitan

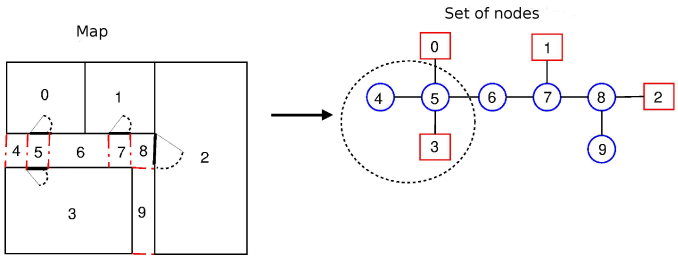


Figura 1.9: Mapa topológico de un entorno de interiores. El espacio es dividido en zonas, no necesariamente iguales, dependiendo de sus características.

disponer de un conjunto de estados discreto, y se desea que la representación sea más general que en caso de las representaciones *topológicas*. La primera ventaja de este tipo de representaciones es que la posición de cada celda de la rejilla es fácilmente traducible a coordenadas métricas respecto a un eje de referencia. Otra ventaja es que el número de estados se puede controlar estableciendo el tamaño de cada celda. La parte negativa es que, la mayor parte de las ocasiones, hay que decidir entre contar un gran precisión o un conjunto reducido de estados. Hay mucho algoritmos cuya carga computacional crece exponencialmente con el número de estados, con lo que es necesario mantener un conjunto reducido.

Aunque la mayor parte de los trabajos en Robótica Móvil usan representaciones *estáticas* del entorno construidas con anterioridad a la operación del robot, en algunos la representación es *dinámica*. Esta representación dinámica del entorno se construye (parcial o totalmente) durante la operación del robot con los elementos que percibe a través de sus sensores. El campo que se ocupa de la formación de este tipo de mapas y su aplicación en el cálculo de la posición del robot se denomina *SLAM* (*Simultaneous Localization And Mapping*), que describiremos a continuación.

**Navegación** La capacidad principal de un robot móvil es la de desplazarse por su entorno. Es por esta razón que existen varias cuestiones abiertas alrededor de la habilidad de navegación de un robot móvil. El robot ha de ser capaz de desplazarse por su entorno sin colisionar con los obstáculos estáticos (paredes, mesas,

columnas, etc.) ni con los elementos dinámicos del entorno (seres humanos u otros robots).

Podemos dividir esta cuestión en dos: *navegación global* y *navegación local*. La navegación global es la habilidad de, habiendo determinado la posición de un robot móvil, moverse a un punto determinado del entorno. Este punto de destino se puede fijar por un operador externo o ser calculado por el robot durante su operación. Existen multitud de estudios que tratan de aportar métodos de navegación global: Grafos de Voronoi, VFF, técnicas de gradiente, etc.

La navegación local es la habilidad de determinar los obstáculos que hay alrededor del robot e reaccionar a ellos de forma correcta. Entre los métodos de navegación local están los de ventana dinámica, curvatura-velocidad, basados en flujo óptico, etc.

Es habitual que un robot combine navegación local y global para alcanzar posiciones globales del entorno evitando en su recorrido a los obstáculos que se encuentre en su trayectoria.

**Auto-localización** Para poder realizar tareas de navegación global, es imprescindible el conocimiento de la posición del robot en el entorno. La habilidad de un robot móvil de determinar su posición en un entorno se denomina *auto-localización*. Para determinar su posición, es esencial que el robot utilice la información sensorial (odometría, distancia a obstáculos, detección de balizas, etc.) y la representación del entorno.

La habilidad de auto-localización puede ser *local* o *global*. La auto-localización local, también denominada *tracking*, trata de actualizar una única estimación de la posición de un robot usando su información sensorial. Esto es suficiente en multitud de aplicaciones, pero el robot no puede recuperarse de situaciones de completo desconocimiento de su posición, o de estimaciones erróneas.

Por otro lado, la auto-localización global, también denominado *problema del secuestro* por algunos autores, aborda la situación en la cual el robot no conoce su posición inicial o se puede recuperar de situaciones de estimación errónea.

Esta habilidad, en general, es capaz de poder mantener varias hipótesis sobre la posición del robot.

**SLAM** La cuestión que abordan las técnicas de SLAM es la de situar a robot móvil en una posición desconocida de un entorno desconocido, o parcialmente conocido, y que el robot sea capaz de construir incrementalmente un mapa consistente de este entorno mientras determina su posición en él. La solución al problema del SLAM se considera por muchos investigadores en el campo de la robótica móvil como el ”Santo Grial”, ya que podría aportar a un robot capacidades aún más autónomas.

Existen multitud de estudios que abordan la cuestión de la auto-localización. De hecho, esta tesis pretende aportar soluciones a ella. En la siguiente sección se describirán los aspectos fundamentales que influyen en este problema para el caso particular de los robots con patas. Asimismo, en el capítulo 2 se presentan los trabajos relevantes que han tratado de aportar soluciones a esta cuestión.

## 1.4. Auto-localización en robots móviles

En la sección anterior ya hemos definido la auto-localización de un robot móvil como la habilidad de determinar su posición dentro de un entorno. Esta habilidad es esencial para que un robot móvil, que realice tareas dependientes de su posición o la de los objetos que deba manipular, sea realmente autónomo. Si un robot móvil no conoce donde está, es difícil que pueda determinar la siguiente acción que debe realizar.

Existen gran cantidad de aplicaciones robóticas donde es vital que un robot móvil conozca su posición en un entorno. Un ejemplo es el caso del *Grand Challenge*[6]. En esta competición se pretende que un vehículo autónomo (figura 1.10) sea capaz de recorrer grandes distancias autónomamente. Para llevar a cabo el recorrido no se usa un robot convencional, sino un vehículo comercial equipado con sensores láser, cámaras, GPS, etc. Este vehículo lleva a cabo una navegación global usando un sistema de GPS para conocer su posición, y usa el resto de sus sensores para ayudar

[6]http://www.darpa.mil/grandchallenge/

en el cálculo de su posición y realizar tareas de navegación local para evitar posibles obstáculos. El sucesor del *Grand Challenge* es el *Urban Challenge*, que se desarrolla en circuitos urbanos.


Figura 1.10: Vehículo autónomo usado en el *Grand Challenge*.

Otro ejemplo de una aplicación donde un robot móvil necesita de la información de localización fue el caso del robot Minerva [TBB+99]. Este robot realiza la labor de guía en el *"Museo Nacional de Historia Americana Smithsonian"*. Para llevar a cabo esta labor, el robot debe navegar de un punto a otro del museo evitando a los posibles obstáculos, principalmente personas. Para poder realizar un navegación entre distintos puntos del museo necesita conocer su posición, por lo que debe usar algún método de auto-localización. En este caso, este robot usa las imágenes que obtenía del techo del museo y su odometría para auto-localizarse.

Un último ejemplo es uno de los entornos claves en esta tesis: la liga de 4 patas de la RoboCup. En esta aplicación, 4 robots móviles autónomos juegan al fútbol contra otros 4. Para poder realizar algún tipo de estrategia y para tomar las decisiones adecuadas, es necesario que cada robot conozca su posición dentro del campo de juego. Para ello, el entorno se ha equipado con elementos fácilmente reconocibles visualmente, cuya posición es conocida de antemano, que el robot debe detectar.

Existen varias técnicas que se pueden usar en un robot móvil para obtener su posición en un entorno. Hay técnicas que son capaces de calcular la posición del robot a partir de la información en crudo de los sensores: GPS, odómetros, etc... Este

es, por ejemplo, el caso de sistema GPS, ya presentado anteriormente. Si un robot móvil está equipado con un dispositivo GPS, es capaz de obtener su posición en un eje de referencias global con una un error inferior a 10 metros. Este método de auto-localización solo es válido para robots que operan en entornos de exteriores.

También se puede incluir en esta técnica los robots móviles que usan sensores odométricos como único modo de calcular su posición. Conocida la posición inicial, ésta se actualiza exclusivamente con la información odométrica. Sin embargo,esta forma de calcular la posición del robot se ve afectada por la acumulación de errores o imprecisiones en la información odométrica. Esta técnica, denominada *dead reckoning*, no suele usarse de manera aislada, sino combinada con otras técnicas que corrigen los posibles errores con información sensorial adicional.

La triangulación mediante balizas activas consiste en detectar la posición relativa al robot de balizas presentes en el entorno. Estas balizas se colocan en en posiciones conocidas por el robot y son fácilmente detectables por el robot. Una vez obtenida la posición de dos o más balizas, se usan técnicas de triangulación para obtener la posición del robot. Se pueden usar balizas activas (RFID, Ultrasónicas, etc.) o pasivas (balizas coloreadas detectables mediante visión, carteles, etc.).

Existe un gran conjunto de técnicas, denominadas *probabilísticas*, que permiten modelar la creencia de posición del robot como una distribución de probabilidad. Asimismo, la información sensorial y los movimientos se modelan mediante funciones de probabilidad que modifican la creencia de la posición del robot.

Este grupo de técnicas es la más usada actualmente en el campo de la Robótica Móvil, y es donde vamos a realizar las aportaciones desarrolladas al hilo de esta tesis. Una de las principales razones para usar este enfoque es que el mundo real, donde se han de desenvolver los robots móviles reales, está sujeto a una gran incertidumbre. Las medidas sensoriales son ruidosas debido a la complejidad del entorno que rodea al robot y a cualquier posible evento que pueda afectar a la medida. El movimiento del robot también está sujeto a situaciones tales como colisiones, rozamiento con el suelo, posibles deslizamientos, etc. Los enfoques probabilísticos pretenden modelar toda esta incertidumbre y tenerla en cuenta en el proceso de auto-localización.

## 1.5. Objetivo de esta tesis

Una vez descrito el contexto de esta tesis, que es el campo de la auto-localización de un robot móvil, hemos de fijar los objetivos concretos de esta tesis.

La presente tesis pretende presentar el análisis, diseño, implementación y pruebas de mecanismos que resuelvan el problema de la auto-localización en robots móviles autónomos en interiores que usan patas como medio de locomoción y una cámara como sensor principal. Los mecanismos propuestos son el fruto del análisis de las necesidades de auto-localización de un robot móvil concreto y las implementación de soluciones adecuadas teniendo en cuenta los factores de fiabilidad y eficiencia.

Estos objetivos tienen varias implicaciones que es necesario comentar:

- El robot usa **patas** para desplazarse. Esto implica que el sistema de locomoción del robot es mucho más complejo que el de un robot que usa ruedas. La información odométrica en un robot con patas no es tan precisa y fiable como con uno que usa ruedas.

  Además, el movimiento de un robot con ruedas es mucho más estable que en uno con patas. Si se coloca una cámara en un robot con ruedas, la imagen se verá mucho menos afectada por el movimiento del robot que en un robot con patas, como es nuestro caso. En un robot con patas, la cámara puede sufrir grandes oscilaciones que afectan a la calidad de las imágenes.

- Los entornos donde se va a desarrollar esta tesis son **interiores**. En estos entornos las condiciones de luz estarán de algún modo controladas. Además, la superficie donde se desplaza el robot es homogénea y horizontal. El entorno podrá tener cierta estructura y, en algún caso puede estar ”ingenierizado”, esto es, diseñado para facilitar la percepción o la actuación del robot. El hecho de ser entornos de interiores elimina la posibilidad de usar sistemas como GPS.

- Se usará un **cámara** como sensor principal del robot. La visión uno de los sensores más ricos (y complejos) que hay, en donde se están produciendo innumerables avances en los últimos años. Esto es debido sobretodo al aumento de la potencia de cálculo de los controladores. A diferencia de otros sensores

como los basados en láser, este sensor puede estar sujeto a enorme ruido debido al movimiento de la cámara y a la dificultad del proceso de extracción de información.

Aunque el aspecto del procesamiento de la imagen para obtener la información del entorno es un factor con gran impacto en la eficacia de un algoritmo de localización, queda fuera de este trabajo. Se presentarán únicamente los problemas existentes como un factor más que será asumido en el desarrollo de las soluciones propuestas.

La plataforma experimental que usaremos es el robot AIBO, que se describe en profundidad en la sección 1.5.1. Esta plataforma es una de las más avanzadas de las disponibles comercialmente y fue desarrollada por SONY[7] con el objetivo de ser un robot mascota. Tiene forma de perro, por lo que dispone de 4 patas que usa para desplazarse. Está equipado con gran variedad de sensores, entre los que destaca una cámara situada en el morro. Esta plataforma robótica es muy atractiva debido a su robustez, versatilidad y bajo coste.

Los entornos en los que se llevarán a cabo las tareas de auto-localización son dos: entornos de oficina y el campo de la RoboCup. Ambos entornos son interiores y presentan condiciones diferentes. Aunque los dos son estructurados, en el caso del entorno de oficinas el robot se ha de adaptar al entorno y usar los elementos ”naturales” presentes en él para localizarse. Además, debe compartir su espacio con seres humanos. En el entorno de la RoboCup el entorno está *ingenierizado*. Todos los elementos presentes en él están diseñados para que el robot sea capaz de realizar mas fácilmente un conjunto de tareas similares a un partido de fútbol. Es más dinámico que el entorno de oficinas, como se describirá posteriormente y las condiciones ambientales suelen estar más controladas.

El primer objetivo será aplicar un método clásico auto-localización en un entorno de oficinas, descrito en la sección 1.5.2, pero teniendo en cuenta las características de un robot con patas cuyo sensor principal es la cámara. Se evalúa si las técnicas clásicas son apropiadas para solucionar el problema de la auto-localización teniendo en cuenta las características del entorno donde se desenvuelve el robot.

[7] http://support.sony-europe.com/aibo/

Posteriormente se evalúa la aplicación del métodos usados en el entorno de oficinas al entorno de la RoboCup, descrito en la sección 1.5.3, para comprobar si la aplicación de una técnica clásica puede ser tan general como para adaptarse correctamente a un entorno diferente con requisitos diferentes.

La aportación más relevante de esta tesis es la de desarrollar métodos particulares que resuelvan de una manera óptima el problema de la auto-localización en el entorno de la RoboCup. Estos métodos se han de integrar en el software del equipo TeamChaos, descrito en el apéndice A, y deben cumplir con unas restricciones de tiempo de procesamiento muy estrictas.

### 1.5.1. Plataforma experimental: el robot AIBO

La plataforma en la que se realizan los experimentos es el robot AIBO. Este robot con forma de perro tiene un avanzado mecanismo de locomoción con patas, y numerosos sensores. Es una plataforma robótica relativamente asequible ($\sim 2000$ €) que ofrece muchas posibilidades.

En concreto, utilizaremos el robot AIBO ERS7, mostrado en la figura 1.11. Se trata de un robot completamente autónomo equipado con un procesador MIPS a 576MHz y 64 MB de memoria principal. Recoge información de su entorno a través de varios componentes hardware, entre los que se incluyen: una cámara a color de 350K pixeles, tres sensores de infrarrojos para detectar distancias, un sensor de aceleración, un sensor de vibración, un micrófono estéreo, sensores de contacto en las patas para detectar cuándo están en contacto con el suelo, y sensores táctiles en la cabeza, en la barbilla y en el lomo.

El robot se mueve e interactúa con su entorno a través de varios actuadores incluyendo: un altavoz, varios *LED* de coloressituados en la espalda y en la cabeza, y 20 motores para mover cada una de las 3 articulaciones de sus 4 patas, 3 articulaciones situadas en el cuello, su boca, sus orejas y las dos articulaciones de su cola. AIBO también incorpora una tarjeta de red inalámbrica 802.11b.

Los sensores infrarrojos que incorpora el AIBO aportan una información del entorno que no es fiable, al estar sometidos a los vaivenes propios del movimiento con patas del robot y a su naturaleza eminentemente ruidosa. Este robot carece de

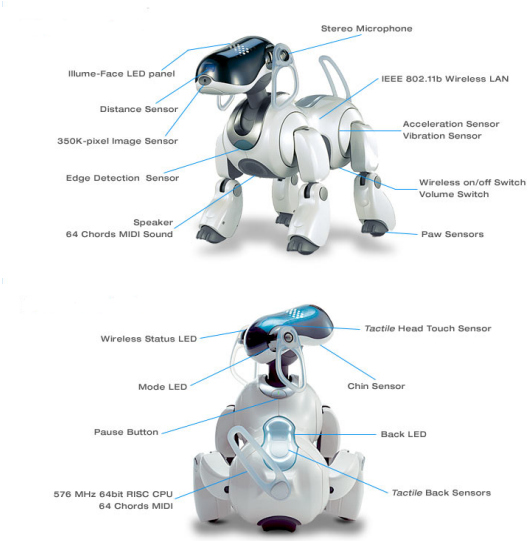


Figura 1.11: Sony AIBO ERS7 (Foto de Sony)

sensores precisos de distancia, como podría ser un láser. Por ello, el principal sensor que se usa es la cámara.

Las aplicaciones que se ejecutan en el robot se copian en una tarjeta de memoria, que se introduce en el robot. El entorno de programación es OPEN-R[8]. SONY ofrece este entorno de programación, introducido en el apéndice A y ampliamente descrito en [MGCRJ04].

### 1.5.2. Entornos de oficina

Los entornos de oficina han sido los elegidos para probar algoritmos de navegación y localización desde los primeros trabajos en robótica móvil. En este entorno existen gran cantidad de aplicaciones en los que un robot móvil puede ser útil: vigilancia, asistencia, reparto de correo, etc.

Actualmente, estos entornos se consideran sencillos, puesto que presentan una estructura regular donde existen una serie de elementos característicos, como son la presencia de puertas, despachos, luces o pasillos (figura 1.12). En el caso de la

[8]http://www.openr.org/

Figura 1.12: Entorno de oficina percibido por el robot AIBO donde existen elementos comunes como puertas y paredes

iluminación, está normalmente controlada, aunque existe cierta variabilidad debido, entre otros motivos, a la luz que entra por las ventanas.

En estos entornos existe también una actividad humana (o robótica) que hace que el entorno no sea completamente estático. La presencia humana en un entorno de oficinas alrededor del robot puede hacer que obtenga lecturas de los sensores que no se correspondan con las esperadas en esa posición.

Los entornos de interiores están compuestos de pasillos y despachos en la mayor parte de los casos. Las tareas que se desea que haga un robot en este entorno son principalmente de navegación.

### 1.5.3. Fútbol robótico: RoboCup

La competición RoboCup es una iniciativa internacional para promover el avance científicio en el campo de la Inteligencia Artificial y la Robótica. En esta competición se presenta un problema complejo: el fútbol. La razón de elegir este problema es que presenta un escenario desafiante a los investigadores en el campo de la Inteligencia Artificial y la Robótica, donde sus investigaciones pueden ser aplicadas, evaluadas y contrastadas.

Al igual que en el caso de la Inteligencia Artificial, donde se planteó como meta que una computadora venciera al campeón del mundo de ajedrez, en este caso la meta es que un equipo de robots humanoides derroten al equipo campeón del mundo humano en el año 2050. Ahora esta meta puede parecer irreal, al igual que lo pareció la meta del ajedrez cuando se planteó.

La RoboCup está formada por varias categorías que se diferencian principalmente por el tipo de robots que compiten. Hay varias ligas, entre las que destacan:

***Small Size*** En esta liga los robots tienen un tamaño inferior a 180 mm y se desplazan mediante ruedas. No son autónomos, sino que están controlados por un ordenador al que se conectan por una conexión inalámbrica. Este ordenador detecta la posición de cada robot y la pelota por medio de una cámara cenital situada sobre el campo de juego. Una vez detectados todos los elementos relevantes en el juego, este ordenador genera las acciones que deben realizar cada uno de los robots.

***Medium Size*** Los robots que participan en esta liga tienen un tamaño de 50 cm. de diámetro y 80 cm. de altura, como máximo. Son completamente autónomos. Se desplazan mediante ruedas y usan como sensor principal una cámara omnidireccional.

***Four-legged*** En esta liga compiten equipos de 4 contra 4 robots AIBO completamente autónomos. Todos los participantes usan el mismo modelo de robot y no está permitido modificarlo. De esta manera, esta liga tiene un enfoque más ”software” que el resto de las categorías, en las que cada equipo puede desarrollar o adquirir el robot que desee, siempre que cumpla las normas de su categoría. Esta categoría ha evolucionado a la *Standar League*, donde el robot es un humanoide.

En esta última categoría se ha desarrollado parte del trabajo de esta tesis. A diferencia de los entornos de oficina, el entorno de esta categoría de la RoboCup está diseñado específicamente para los robots AIBO, estando altamente *ingenierizado*. Como se puede observar en la figura 1.13, tiene marcas visuales con colores para que sean identificadas a través de la cámara del robot, las líneas son blancas y de anchura preestablecida, y existen focos que iluminan el campo. El campo de juego tiene $6x4$ metros. En el centro de ambos lados cortos están las porterías, y en los lados largos hay una baliza situada en el medio campo. Todas estas marcas visuales están coloreadas para ser detectadas mediante filtrado sencillo de las imágenes obtenidas por el robot.

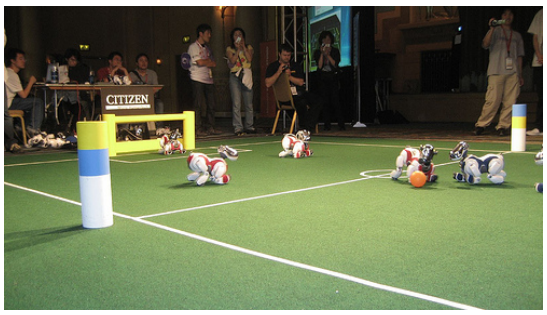


Figura 1.13: Campo de la juego de la RoboCup.

Las necesidades de localización son radicalmente distintas, por tanto, a las de los entornos de oficina. Las decisiones que el robot toma dependen enormemente de su posición exacta en el campo de juego. Algunas de estas decisiones se deben a las reglas y otras a las estrategias, por ejemplo:

- Al inicio del partido, al principio de la segunda parte y después de cada gol, cada robot ha de colocarse en una posición predeterminada del campo de juego diferente además si pertenece al equipo que saca o no. En la figura 1.14 se observan las posiciones de inicio, siendo el equipo rojo el que tiene derecho al saque.

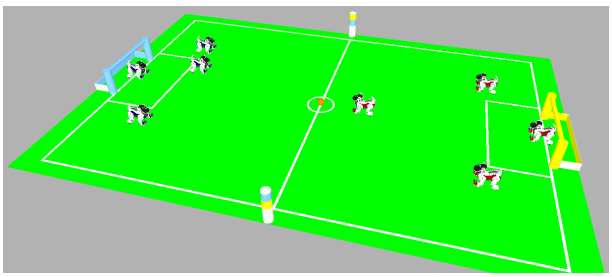

Figura 1.14: Disposición de los robots para el saque.

- Ningún robot puede abandonar los límites del campo ni entrar en su propia área. Si esto sucede, el robot es retirado del campo de juego durante 30 segundos.

- El robot que realiza el rol de portero, no debería abandonar el área. Además, es deseable que esté colocado entre la portería y la pelota con el fin de evitar que

le anoten un gol. Esto requiere un conocimiento muy preciso de su posición actual.

- Para realizar estrategias avanzadas, los robot han de ir a posiciones determinadas del campo dependiendo de su posición, la posición de la pelota y el rol que esté ejecutando en cada instante. En este caso, la posición real del robot ha de ser conocida de manera precisa para poder ir a la posición ideal.

- El golpeo de la pelota puede depender de su posición global y relativa a los objetos del campo.

Aunque el entorno de la RoboCup está altamente estructurado, el robot debe desarrollar sus tareas junto con otros siete robot. Esto hace que haya una gran incertidumbre en el movimiento que realiza el robot, ya que éste puede ser empujado, obstruido e incluso derribado por otro robot.

Las imágenes constituyen la fuente de información que usan los robots para percibir los elementos del entorno. Los algoritmos que se desarrollan para la auto-localización deben tener en cuenta que tanto la percepción como la locomoción son muy ruidosas y erróneas. Los robots colisionan entre ellos (por ejemplo, los robots marcados en la parte izquierda de la figura figura 1.15), haciendo que la información de odometría enviada al módulo de auto-localización sea errónea. De hecho, es usual que se formen "peleas" al obstruirse varios robots entre ellos; incluso pueden terminar boca arriba, como en la imagen derecha de la figura 1.15.

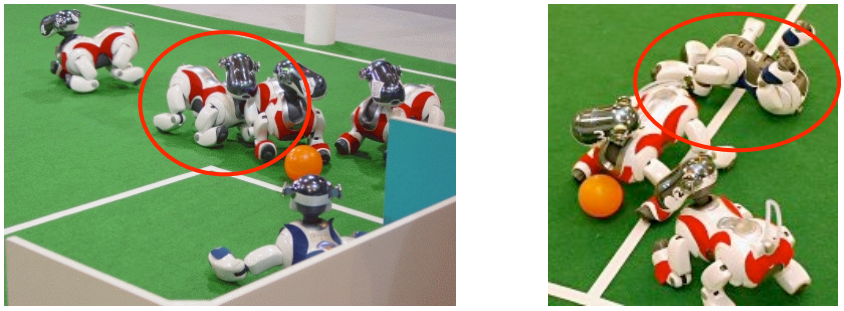

Figura 1.15: Situaciones en la cuales la odometría es errónea.

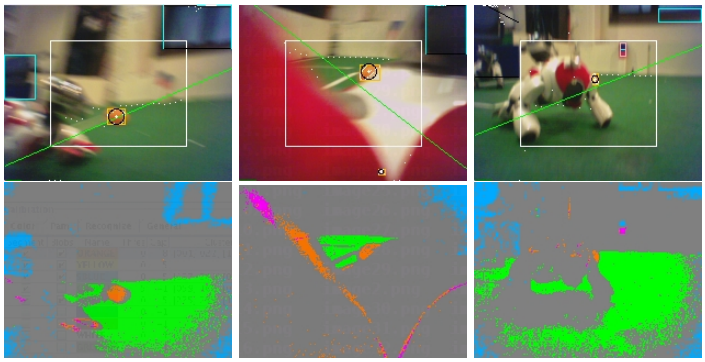

Figura 1.16: Imágenes percibidas por el robot y la segmentación en el espacio de color que realiza

Igualmente, la percepción en este entorno no está exenta de problemas. Por ejemplo, la figura 1.16 muestra en su parte inferior las imágenes filtradas por color capturadas por el robot. Encima se muestran las imágenes reales tomadas por la cámara con los objetos detectados recuadrados. Esta imagen ilustra hasta qué punto las imágenes presentan ruido y oclusiones debido a la colisión y las interacciones con robots. En las imágenes de la izquierda el robot se mueve tan deprisa que la baliza que debería detectar en el centro de la imagen queda totalmente difuminada e irreconocible; en las imágenes centrales, la visión del robot está ocluida por otro robot; y en las imágenes de la derecha la ventana se detecta como una portería debido a su color y a errores al calcular el horizonte visual por la presencia de otro robot.

Figura 1.17: La "niña baliza"(Roboludens 2006, Eindhoven (Holanda)).

Aunque el campo es un entorno controlado, no lo son sus alrededores, como ocurre en la situación ilustrada por la figura 1.17. Esta imagen se tomó en la edición de la RoboCup 2006, en la que apareció una niña vestida con los mismo colores que una baliza, y en el mismo orden. Esto hacía que los robots la percibieran como una baliza e inducía a errores en su auto-localización. Otro ejemplo se encuentra en la figura 1.18, donde un enorme globo naranja hacía que los robots lo percibieran como una pelota.

Figura 1.18: Entorno de juego en la RoboCup de Osaka. Nótese el globo naranja arriba a la derecha.

Otro problema añadido, aparte de las colisiones y los errores en la percepción, es la posibilidad durante un partido de que el robot sea sancionado. En ese caso, el robot es sacado manualmente del campo de juego por uno de los árbitros, que lo devolverán al centro transcurridos 30 segundos. Este problema de "secuestro" debe ser tenido en cuenta en el desarrollo de los algoritmos de auto-localización.

## 1.6. Estructura de esta tesis

En este capítulo se ha presentado el contexto donde se enmarca esta tesis, y se han presentado los objetivos y la propuesta de tesis que los cumple.

En el *capítulo 2* se hará una revisión bibliográfica de los trabajos más relevantes en el campo de la auto-localización de robots móviles. Una vez presentados estos trabajos, en el *capítulo 3* se estudiará emplear uno de estos métodos clásicos en

diversos entornos. El *capítulo 4* se centrará en dar una solución apropiada a la auto-localización de un robot móvil en el campo de la RoboCup con los requisitos que presenta el software donde se integra el sistema de auto-localización. Los resultados de este capítulo se analizarán en el *capítulo 5*, y posteriormente, en el *capítulo 6*, se presentarán las conclusiones que se desprenden de esta tesis.

# CAPÍTULO 2

# Revisión bibliográfica

En este capítulo revisamos los diferentes trabajos que han tratado de aportar soluciones al problema de auto-localización de un robot móvil. Durante esta revisión pretendemos describir los trabajos más representativos de este campo, así como mostrar el estado del arte actual. Este análisis no pretende ser exhaustivo, sino presentar las distintas alternativas que han surgido en la resolución de este problema así como sus características y limitaciones.

No todos los trabajos han seguido el mismo enfoque para resolver este problema, aunque el más popular es el enfoque probabilístico. Este enfoque sea ha mostrado eficaz al modelar la incertidumbre presente en el mundo real, que es siempre ruidoso y sujeto a situaciones inesperadas. El enfoque probabilístico plantea la auto-localización como un problema de estimación probabilística. La posición del robot se modela como una distribución de probabilidad y los modelos sensoriales como funciones de probabilidad. La práctica totalidad de los estudios en Robótica Móvil abordan este problema usando técnicas probabilísticas. De hecho, las aportaciones presentadas en esta tesis usan técnicas probabilísticas para estimar la posición del robot. Por esta razón, este capítulo se centra en presentar los principales trabajos de auto-localización que usan este conjunto de técnicas.

Dentro del enfoque probabilístico, es complicado realizar una clasificación de los métodos que se han empleado, ya que muchos trabajos combinan técnicas muy diferentes para conseguir solucionar este problema. Existen estudios donde se plantean clasificaciones de los trabajos de localización que se han desarrollado, como es el caso de [JBL96], [GBFK98] o [GF02]. En esta tesis proponemos una clasificación atendiendo a las técnicas más representativas de este campo: basados en el Filtro Extendido de Kalman (EKF), Markov y Monte Carlo. La mayoría de los trabajos realizados en el campo de la Robótica Móvil sobre auto-localización usan técnicas que se pueden encasillar en una de estas tres divisiones.

El orden en que presentamos cada una de estas técnicas atiende principalmente a razones cronológicas y de importancia desde el punto de vista de los métodos empleados en esta tesis. En primer lugar, en la sección 2.1 se describe uno de los métodos gaussianos más aplicados en la localización, el EKF. Este método ha sido aplicado con éxito en multitud de trabajos, y en esta sección se muestran los más representativos. A continuación, en la sección 2.2 se describen los fundamentos de los algoritmos markovianos y se presentan los principales trabajos que usan estas técnicas para la estimación de la posición de un robot móvil. Por último, en la sección 2.3 se describe una de las técnicas que más atención y que más éxito tiene actualmente, Monte Carlo.

## 2.1. Filtro Extendido de Kalman

El filtro de Kalman [Kal60] es un estimador recursivo óptimo usado para estimar procesos estocásticos, sobre todo en el campo del tratamiento de la señal. El filtro de Kalman presenta un conjunto de ecuaciones matemáticas que proporcionan una manera eficiente y recursiva de estimar el estado de un proceso, basado en minimizar la media del error cuadrático. Este filtro es muy potente atendiendo a varios aspectos: soporta estimación de estados pasados, presentes e incluso futuros y lo puede hacer cuando la precisión del sistema a medir es desconocida.

Resumidamente, el Filtro de Kalman tiene como objetivo la estimación del estado $s \in \Re^n$, y su incertidumbre asociada, de un sistema discreto en el tiempo gobernado por una ecuación lineal estocástica diferencial.

$$s_t = As_{t-1} + Bu_{t-1} + w_{t-1} \tag{2.1}$$

$$z_t = Hs_t + v_t \tag{2.2}$$

Donde, en la ecuación 2.1, $u_{t-1} \in \Re^l$ es la entrada de control que se efectúa sobre el sistema. La matriz $A$ representa como evoluciona el sistema de $s_{t-1}$ a $s_t$ en ausencia del componente de control o de ruido. La matriz $B$ indica como se modifica $s_t$ con la entrada de control $u_{t-1}$. La variable aleatoria $w_{t-1}$ representa el ruido que se incorpora al sistema, que corresponde a la distribución de probabilidad $p(w) \sim N(0, Q)$, donde $Q$ es la covarianza del error esperado.

El Filtro de Kalman se aplica a sistemas lineales. Al ser no lineales la mayor parte de los sistemas que modelamos, se aplica en su lugar el Filtro Extendido de Kalman, que linealiza el sistema en torno a $s_t$.

Dentro de esta sección clasificamos los trabajos realizados con EKFs en tres grupos. En primer lugar presentamos los trabajos que tienen en común el hecho de usar esta técnica para obtener la posición del robot de manera *local*, es decir, realizar un *tracking* del robot. En segundo lugar presentamos los trabajos que combinan esta técnica con alguna otra para conseguir un método de auto-localización *global*. Por último, mostraremos los trabajos más relevantes que usan esta técnica en el entorno dónde se desarrolla la parte principal de esta tesis: el entorno de la RoboCup.

### 2.1.1. Auto-localización local usando EKF

Uno de los primeros trabajos en localización que usa EKFs fue llevado a cabo por James L. Crowly en 1989 [Cro89]. Este trabajo es la extensión del que él mismo presentó en 1985 [Cro85]. El enfoque usado en en la propuesta inicial no era en absoluto probabilístico, sino basado en técnicas básicas de *scan matching*. En el trabajo posterior incorporaba un EKF para estimar la posición del robot.

En la propuesta inicial trataba de resolver la navegación de un robot móvil, equipado con un sensor de ultrasonidos rotatorio, en un entorno estructurado. Para realizar tareas de navegación global, se usaba un modelo global que representaba el entorno

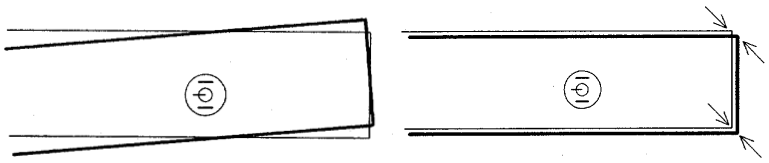

Figura 2.1: Corrección de la posición del robot con el modelo compuesto local.

como un conjunto de posiciones interconectadas. Usando este conjunto de posiciones, el robot era capaz de planificar una ruta desde la posición actual del robot hasta el destino. Para llevar a cabo las tareas de navegación se usaba un sistema de navegación local que lleva a cabo los pasos marcados por el plan global.

El sistema de navegación local mantenía la posición del robot respecto a un modelo global del entorno, y planificaba rutas locales para esquivar obstáculos. El sistema de navegación local utilizaba un modelo de su entorno más inmediato. Este modelo se codificaba en una estructura llamada "modelo compuesto local" (*composite local model*).

El modelo compuesto local integraba la información sensorial procedente de todos los sensores del robot. La información del modelo global también se integraba en esta estructura. Esta estructura tenía dos funciones principales. Por una parte, integraba la información potencialmente conflictiva de los sensores con la información sensorial percibida recientemente y con el modelo global del entorno. Por otra parte, servía como base de información para los procesos que se encargaban de la planificación local y global de las rutas, y otras tareas de alto nivel.

El modelo sensorial se basaba en la representación de los obstáculos percibidos por medio de segmentos. Una vez percibido un segmento, éste se comparaba con la información de los segmentos que tenía almacenados el modelo compuesto local y se corregía la posición del robot (respecto al modelo compuesto local) realizando pequeñas rotaciones y translaciones, como se muestra en la figura 2.1.

La localización del robot se calculaba realizando un *scan matching* entre el modelo compuesto local y el modelo global, como se muestra en la figura 2.2.

Los segmentos percibidos $S_o$ se representaban, teniendo en cuenta la incertidumbre que se podía presentar en la medida, mediante la tupla
$\{x_o, y_o, c_o, h_o, \theta_o\}$ con una incertidumbre asociada $\{\sigma_{c_o}, \sigma_{\theta_o}\}$, donde $(x_o, x_o)$ es la po-

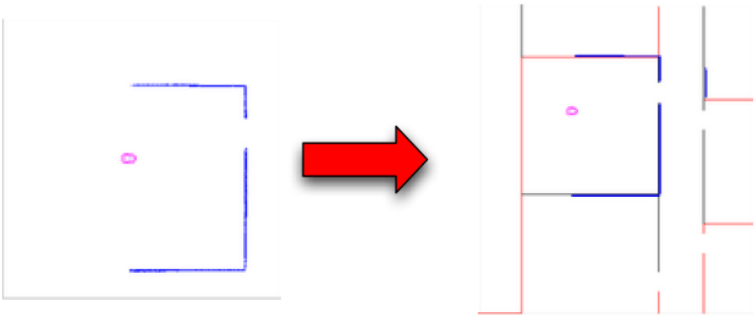

Figura 2.2: Comparación entre el modelo compuesto local y el modelo global

sición del centro del segmento en coordenadas cartesianas con respecto al robot, $c_o$ es la longitud de la recta normal al segmento percibido que parte del centro del robot, $h_o$ es la mitad de la longitud del segmento y $\theta_o$ es la inclinación del segmento.

Cuando se percibía un segmento $S_o$, éste debía incorporarse al modelo compuesto local. Para incorporarse, se realizaban una serie de tests hasta encontrar el segmento que concordaba con el percibido.

Dado un segmento $S_o$ y un segmento $S_m$ existente en el modelo compuesto local, el primer test consistía en comprobar si ambos tenían la misma orientación,

$$(\theta_m - \theta_o)^2 \leq \sigma_{\theta_o}^2 + \sigma_{\theta_m}^2 \tag{2.3}$$

Si no se cumplía, se avanzaba hasta el siguiente segmento del modelo compuesto local. Si se cumplía, se realizaba un segundo test para comprobar si estaban alineados,

$$(c_m - c_o)^2 \leq \sigma_{c_o}^2 + \sigma_{c_m}^2 \tag{2.4}$$

Si se cumple esta condición, se comprobaba si concordaban,

$$(x_o - x_m)^2 + (y_o - y_m)^2 \leq h_o + h_m \tag{2.5}$$

El segmento con mayor longitud que cumplía estas tres condiciones, se usaba para corregir la posición estimada del robot y actualizaba el modelo compuesto local.

En la propuesta posterior, cada uno de los segmentos que concordaban con el modelo compuesto local se usaba para corregir la posición del robot mediante un

*EKF*. En este trabajo se actualizaba por un lado la posición $s_{xy}$ del robot, con su incertidumbre asociada $P_{xy}$, y por otro lado la orientación del robot.

La corrección de la posición del robot se realizaba por la diferencia en posición perpendicular $\triangle c = c_m - c_o$. Esta corrección solo se aplicaba en la dirección perpendicular al segmento, definido por el vector $M = [\ a \quad b\ ]^T$ desde el segmento observado. La ganancia de Kalman era

$$K_s = P_{xy} M \frac{1}{\sigma_{c_o}^2 + \sigma_{c_m}^2} \tag{2.6}$$

Finalmente, la corrección se aplicaba a la posición estimada actual de la siguiente manera,

$$s = s - K_s \triangle c \tag{2.7}$$

Sin embargo, el controlador del sistema de navegación local del robot estaba diseñado para aceptar un vector de corrección $\triangle s = [\ \triangle x \quad \triangle y\ ]$, y una ganancia $K_{xy}$. Esto se calculaba de la siguiente forma,

$$K_{xy} = K_p M^T \tag{2.8}$$

El vector de corrección era $\triangle P = M \triangle c$, de tal manera que la posición y la su covarianza se calculaban como sigue

$$s = s - K_{xy} \triangle P \tag{2.9}$$

$$P_{xy} = P_{xy} - K_{xy} P_{xy} \tag{2.10}$$

La orientación del robot se actualizaba calculando su ganancia de Kalman y aplicándola al ángulo anterior y a su covarianza asociada,

$$K_\alpha = \frac{\sigma_\alpha^2}{(\sigma_{\alpha_m}^2 + \sigma_{\alpha_o}^2)} \tag{2.11}$$

$$\alpha = \alpha - K_\alpha(\alpha_o - \alpha_m) \tag{2.12}$$

$$P_\alpha = P_\alpha - K_\alpha P_\alpha \tag{2.13}$$

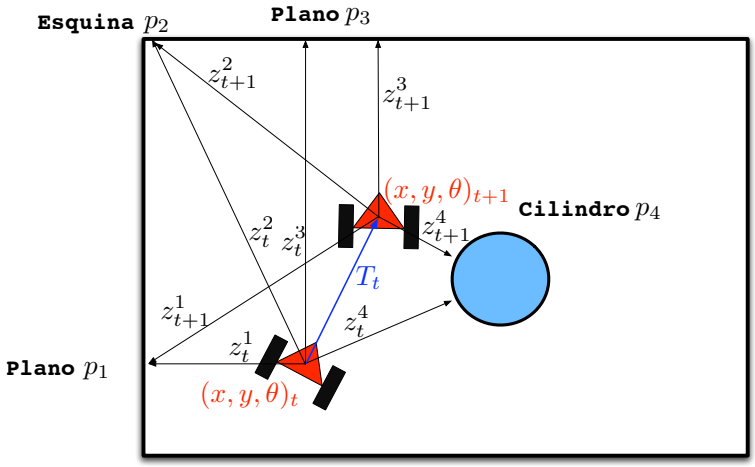


Figura 2.3: Conjunto de balizas percibidas por el robot.

Los resultados de este trabajo mostraban que el robot era capaz de seguir rutas en el entorno conociendo de manera precisa su posición.

Otra aplicación de los EKF a la localización es el trabajo descrito en [LDW91], donde se trataba de solucionar el problema de la auto-localización de un robot móvil mediante el seguimiento de elementos geométricos usando un *EKF*. Un elemento geométrico es aquel del entorno que puede ser observado con éxito después de realizar sucesivos desplazamientos y puede ser descrito mediante una parametrización geométrica. En la figura 2.3 se representa la detección de elementos geométricos tales como dos planos, una esquina y un cilindro. La detección y formación de estos elementos se detalla en [DW87].

La fase de predicción del estado $s$ y su incertidumbre $P$ se realizaba a partir de la entrada de control $u$ y el ruido en el proceso $w$ en el instante $t$:

$$s^-_{t+1} = f(s_t, u_t) + w_t \tag{2.14}$$

$$P^-_{t+1} = A_{t+1} P_t A^T_{t+1} + W_{t+1} Q_t W^T_{t+1} \tag{2.15}$$

El modelo de observación expresaba una observación sensorial en términos de la predicción de la posición del robot y de la posición del elemento que está siendo

observado. Después de haberse realizado la fase de predicción, se usaba la posición calculada del robot para predecir la observación de cada uno de los elementos $p_j$,

$$z_{t+1}^{i,-} = h^i(p_j, s_{t+1}^-) \tag{2.16}$$

En la fase de corrección se tomaban medidas reales $z_{t+1}^j$ de los elementos geométricos y se comparaba con las observaciones que se habían predicho $z_{t+1}^{i,-}$. La diferencia entre ambas observaciones se denominaba *innovación*, y se calcula de la siguiente manera

$$v_{t+1}^{ij} = [z_{t+1}^j - z_{t+1}^{i,-}] = [z_{t+1}^j - h^i(p_j, s_{t+1}^-)] \tag{2.17}$$

La covarianza de la innovación en la predicción era

$$S_{t+1}^{ij} \equiv E[v_{t+1}^{ij}(v_{t+1}^{ij})^T] = \bigtriangledown h^i P_{t+1}^- \bigtriangledown (h^i)^T + R_{t+1}^j \tag{2.18}$$

donde el jacobiano $\bigtriangledown h^i$ se evalúa en $s_{t+1}^-$ y $p^i$, y $R_{t+1}^j$ correspondían a la incertidumbre en la observación.

La ganancia de Kalman se calculaba entonces para estimar el estado $s_{t+1}$ y su covarianza asociada $P_{t+1}$,

$$K_{t+1} = P_{t+1}^- \bigtriangledown h^i S_{t+1}^{-1} \tag{2.19}$$

$$s_{t+1} = s_{t+1}^- + K_{t+1} v_{t+1} \tag{2.20}$$

$$P_{t+1} = P_{t+1}^- - K_{t+1} S_{t+1}^{-1} K_{t+1}^T \tag{2.21}$$

Los dos trabajos que acabamos de presentar usan sensores de distancia para detectar e interpretar los elementos geométricos del entorno. En el caso del trabajo presentado en primer lugar [Cro89] se extraen los segmentos que conforman el entorno más cercano al robot y se usan para corregir la predicción de la posición del robot. En el segundo trabajo [LDW91] el modelo de observación es más elaborado y se distinguen elementos geométricos más complejos que el robot usa para localizarse. En el trabajo que presentamos a continuación, [DIBB00], se usaba un EKF para estimar la

posición de un AGV (*Automatic Vehicle Guide*) que desarrollaba su actividad en un entorno industrial. El objetivo era obtener su posición de manera precisa, pues así lo requería la aplicación que debía realizar, consistente en transportar palés (armazones de madera empleados en el movimiento de carga). En este trabajo se proponía procesar las medidas procedentes de un sensor láser como una señal unidimensional, en lugar de extraer información de estas medidas.

El proceso de estimación de la posición constaba de varios pasos. En el primero de ellos se actualizaba la posición del robot con la información odométrica obtenida a partir del movimiento de las ruedas del vehículo.

A continuación, se filtraba la señal procedente del sensor láser. Esta señal se veía afectada por tres fuentes de ruido perceptivo: posibles reflexiones del láser, el error inherente al sensor (que era despreciable) y el que procedía de posibles objetos entre las paredes y el láser. Para eliminar el primer tipo de ruido se usaba un filtro de mediana que eliminaba las medidas erróneas (figura 2.4). Para eliminar las medidas correspondientes a objetos diferentes a las paredes del recinto se usaba el conocimiento existente sobre la posición de las paredes y la posición estimada del robot.

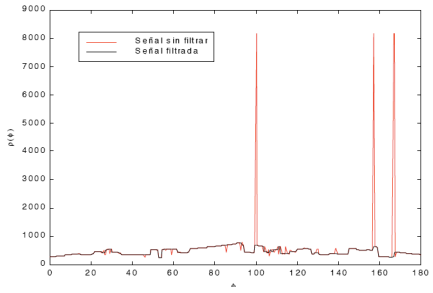


Figura 2.4: Filtrado de la señal láser para eliminar posibles reflexiones del haz del sensor.

Una vez filtrada la señal del sensor, se calculaba la orientación del vehículo. Para ello, se comparaba la medida obtenida con la medida teórica a partir de la posición estimada después de incorporar la odometría. De esta comparación se obtenía una corrección de la orientación de manera que se maximizaba la correspondencia entre ambas medidas. La posición final del vehículo se calculaba usando la estimación de su posición, tras incorporar la corrección de la orientación, y la diferencia entre las medidas reales y la teóricas.

Con esta aproximación se obtenía gran precisión en la estimación de la posición y un coste computacional menor que en el caso de la extracción de elementos del entorno a partir de la señal láser.

### 2.1.2. Auto-localización global usando EKF

Los trabajos descritos en la sección anterior trataban de realizar una auto-localización local, o *tracking* del robot. Existen otros trabajos que, aparte de realizar un *tracking* del robot, abordan el problema de la auto-localización a partir de un completa desconocimiento de la posición inicial del robot, además de mantener varias hipótesis sobre la posición del robot para el caso de secuestros o recuperación ante estimación erróneas. Este es el caso de trabajo presentado en [RB00], donde se pretendía obtener una localización global multihipótesis combinando un filtro de Kalman y una distribución de probabilidad. Las diferentes posibles posiciones del robot se representaban por medio de una distribución de probabilidad $f(s)$ sobre el estado del robot, donde $s = (x, y, \theta)$. Inicialmente

$$f_0(s) = \frac{1}{2\pi S} \tag{2.22}$$

La actualización de la distribución de probabilidad se realizaba de la siguiente manera: En el instante $t$ se realizaba una observación del entorno $z_t$ que contenía información sobre un elemento del entorno, por ejemplo supongamos que se trata de una puerta $A_i$. Podían existir varias puertas en el entorno, por lo que el módulo de extracción de características a partir de la información de los sensores exteroceptivos asociaba una probabilidad $P(z_t = A_i), i = 1 \dots N$ diferente a las posibles puerta concordante en el entorno. La posición de cada puerta $A_i$ se suponía conocida y se denotaba por $s_{A_i}$. La percepción de la puerta se modelaba como una distribución gausiana $f(s|z_t = A_i)$ con media $s_{A_i}$ y covarianza $P_{A_i} = E[(s - s_{A_i})(s - s_{A_i})^T]$.

A partir de cada observación se obtenía un distribución de probabilidad $f(s|z_t = A_i)$ correspondiente a una hipótesis $H_i, i = 1 \dots N$ sobre la posición del robot, que además tenía una probabilidad asociada $P(H_i)$. La distribución de probabilidad $f(s|H_i)$ se correspondía con $f(s|(z_t = A_i))$. La distribución de probabilidad $f(s)$

estaba formada por la combinación de las distribuciones de probabilidad de cada una de las hipótesis

$$f_t(s) = \sum_i P(H_i) f(s|H_i) \tag{2.23}$$

A lo largo de tiempo, las hipótesis con baja probabilidad eran desechadas.

En este trabajo se usaba un *EKF* para estimar el desplazamiento que había realizado el robot, en lugar de usarlo para estimar las hipótesis. Esto permitía que, en lugar de tener un *EKF* para cada hipótesis, se tuviera uno para todas las hipótesis. El *EKF* estimaba el desplazamiento producido $\widehat{\triangle s} = \begin{bmatrix} \widehat{\triangle x} & \widehat{\triangle y} & \widehat{\triangle \theta} \end{bmatrix}^T$ a partir de los sensores propioceptivos. La incertidumbre de esta estimación era $P_{\widehat{\triangle s}} = E[\widehat{\triangle s}\widehat{\triangle s}^T]$.

Para incorporar el desplazamiento realizado, se actualizaba cada una de las hipótesis,

$$\hat{s}_{H_i} = \hat{s}_{H_i} + \widehat{\triangle s}, i = 1 \dots N \tag{2.24}$$

$$P_{A_i} = P_{A_i} + P_{\widehat{\triangle s}}, i = 1 \dots N \tag{2.25}$$

De igual forma, en $t+1$, se obtenía una percepción de otro elemento $z_{t+1} = B_j, j \dots M$, y entonces las distribuciones de probabilidad que surgían de cada una de las posibles concordancias en el entorno se combinaban con la distribución de probabilidad que existía en $t$.

Otra aproximación al problema de la auto-localización global fue [Pia95]. En este caso realmente se usaba la estimación calculada por un EKF como la posición real del robot. La solución consistía en dividir el sistema de auto-localización del robot en dos subsitemas: El subsistema de localización local y el subsistema de localización global.

El subsistema de localización local se componía básicamente de un EKF que mantenía una estimación de la posición del robot mediante el seguimiento de características geométricas del entorno. Estas características geométricas y la forma en que corregían la posición del robot eran similares a las que se usaron en [Cro89], anteriormente descrito.

La aportación más importante de este trabajo fue el desarrollo del subsistema de localización global, que tenía como objetivos la monitorización, verificación y corrección de la estimación que calculaba el subsistema de localización local. Este sistema permitía aportar la posición inicial del robot en caso de no ser conocida y posiciones dónde era más probable que se encontrara el robot. Este subsistema mantenía un conjunto de hipótesis sobre las posibles posiciones del robot.

En el problema que se quería resolver había varios elementos del entorno con similares características, de modo que en el instante $t$ una lectura sensorial $z_t$ podía corresponder a varios posibles objetos $o^i, i = 1, 2, \ldots, m$. El objetivo era determinar qué objeto era el que se había observado en $t$. Para resolver a esta cuestión las hipótesis formaban una estructura en árbol, donde cada nodo de este árbol era la hipótesis $h_t^i$ relativa a los posibles orígenes $o^i$ de la observación $z_t$. Cada posible camino en este árbol representaba distintos escenarios $H_j$:

$$H_j = (h_0^{i_0}, h_0^{i_0}, \ldots, h_n^{i_n}) \tag{2.26}$$

, donde $i_0, \ldots, i_n \in 1, 2, \ldots, m$ que podían darse durante un periodo de tiempo $T = (t_0, t_1, \ldots, t_n)$.La evaluación de cada uno de los posibles escenarios $E(h_j)$ se calculaba de la siguiente manera,

$$E_t(h_j) = \frac{1}{n} \sum_{t-1}^{n} [Eval(h_t^i)]^2 \tag{2.27}$$

,donde $Eval(h_t^i)$ es un valor que indica la similitud entre la predicción de la observación a partir de la estimación de la posición del robot en $t$ y la hipótesis $h_t^i$. El escenario $H_j$ más probable se comparaba con la estimación actual de la posición actual con el fin de avaluarla y poder aportar una posición mejor del robot.

Cada hipótesis $h_t^i$ también se podía usar para determinar las regiones aproximadas donde se encontraba el robot. Dada la observación de un objeto $o^i$, se podía calcular la región $M^i$ como el conjunto de posibles posición del robot desde donde se podía percibir $o^i$. Esta región aportaba información sobre la posible posición inicial del robot, o usarse para mejorar el proceso de localización. Esta mejora se realizaba generando una observación heurística adicional $z_t^{i(+)}$, que era la distancia entre la

estimación de la posición del robot y la línea normal $p^i$ a $o^i$ que pasa por el punto central de $M^i$ (figura 2.5).

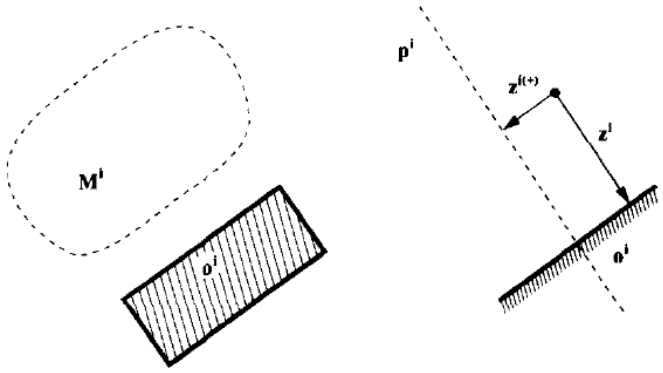


Figura 2.5: Generación de $z_t^{i(+)}$ a partir de la observación de un objeto $o^i$

Este trabajo mostraba una posible solución al problema de auto-localización global, aunque era complicado aplicarse por completo en entornos reales, ya que el árbol del posibles trayectorias podía llegar a tener un número muy elevado de nodos con el consiguiente inabordable coste computacional.

Estos trabajos trataban de abordar el problema de auto-localización global en entornos de oficina con múltiples e indistinguibles posibles observaciones. En el entorno de la RoboCup se simplifica este problema, ya que las observaciones son distinguibles entre ellas, y su número es mucho menor que en los trabajos anteriores. En la siguiente sección se presentan los trabajos que se han desarrollado en este entorno.

### 2.1.3. Aplicación de EKF al entorno de la RoboCup

Uno de los trabajos donde se desarrolló con éxito un método de auto-localización para fútbol robótico fue [GWN01]. Este trabajo se desarrollaba en el entorno de la liga media de la RoboCup y se utilizaba un EKF para estimar el estado del robot, que se representaba mediante una posición en el eje cartesiano y una orientación: $s = (x, y, \theta)$. En este caso, los robots se deplazaban mediante ruedas y estaban equipados con sensores de distancia y una cámara, como se puede observar en la figura 2.6.

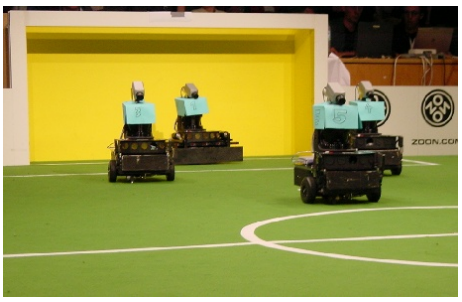

Figura 2.6: El equipo de la liga media *cs-freiburg*.

En este trabajo se desarrollaba una técnica de *map-matching*. En esta técnica se contrastaba el contorno del campo, codificado en un mapa (construído *a priori*), con las lecturas de un sensor láser. Tras aplicar esta técnica, se obtenían un conjunto de hipótesis sobre la posición del robot. Estas hipótesis se incorporaban al *EKF* en la fase de corrección.

En la liga de robots cuadrúpedos de esta misma competición hay pocos trabajos que usen *EKF* para localizar a un robot. Uno de los primeros fue [Gut02]. En este trabajo se desarrollaba un método de localización que combinaba técnicas Markovianas y *EKF* en robots con patas que usaban la cámara para percibir una serie de marcas visuales fijas en el entorno.

Por un lado, se mantenía una rejilla 2D de probabilidad de $120 \times 80$ celdas. Esta rejilla se actualizaba incorporando las acciones y las percepciones usando técnicas markovianas comunes. Asimismo, el *EKF* estimaba el estado $s = (x, y, \theta)$ incorporando las acciones según el modelo de actuación mostrado en la figura 2.7. Este modelo no es el habitual, ya que no modela el desplazamiento realizado $d$ como una gausiana $N(d, D)$, donde $D$ es la incertidumbre en la información odométrica. El modelo tenía en cuenta que podía colisionar con un robot y que podía ser manualmente desplazado a cualquier posición del entorno.

Cuando se percibía una marca visual $o$ , se comprobaba si era posible percibir tal marca atendiendo a información de la rejilla de probabilidad. Se evaluaba entonces $p(o|\tilde{s})$, siendo $\tilde{s}$ la celda con la probabilidad más alta de la rejilla. Si superaba un umbral $t_{obs}$, la observación se incorporaba al EKF. Si no, se descartaba. Posteriormente

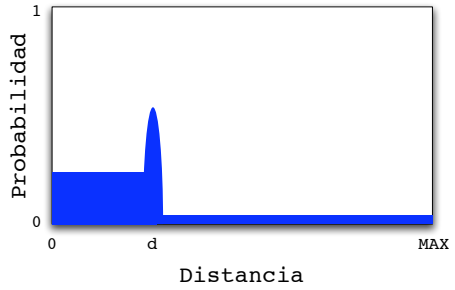


Figura 2.7: Modelo de movimiento de [Gut02].

se incorporaba $o$ a la rejilla de probabilidad y se comparaba la rejilla de probabilidad con el *EKF*. Si no hay concordancia, se iniciaba el *EKF* a $\tilde{s}$.

En la liga de las 4 patas, el entorno donde principalmente se ha desarrollado esta tesis, también han habido trabajos que usaban EKF para estimar la posición de un robot. Este es el caso de [LVRdS05], donde se describe una solución al problema de la auto-localización para el robot AIBO en el entorno de la liga de 4 patas de la RoboCup. Usa un modelo de actuación que calcula la estimación del estado del robot $s = (x, y, \theta)$ usando la odometría $u$ generada a partir del movimiento de las patas del robot,

$$x_t = x_{t-1} + (u^x_{t-1} + w^x_{t-1})cos\theta_{t-1} - (u^y_{t-1} + w^y_{t-1})sen\theta_{t-1} \tag{2.28}$$

$$y_t = y_{t-1} + (u^x_{t-1} + w^x_{t-1})sen\theta_{t-1} + (u^y_{t-1} + w^y_{t-1})cos\theta_{t-1} \tag{2.29}$$

$$\theta_t = \theta_{t-1} + u^\theta_{t-1} + w^\theta_{t-1} \tag{2.30}$$

aunque creemos que la incertidumbre en la actuación $R$ no se modela adecuadamente, lo que puede tener un impacto en la eficacia de este método,

$$R = \begin{pmatrix} 0{,}1|u^x| & 0 & 0 \\ 0 & 0{,}1|u^y| & 0 \\ 0 & 0 & 0{,}1|u^\theta + 0{,}001\sqrt{(u^x)^2 + (u^y)^2} \end{pmatrix} \tag{2.31}$$

En este trabajo se combinaban tres métodos: Monte Carlo, *Single Landmark* y EKF. El método de Monte Carlo, descrito en la sección 2.3, incorporaba varias hipóte-

sis procedentes de una estimación realizada por EKF y del método *Single Landmark*. El EKF implementado realizaba una estimación de la posición del robot a partir de la información odométrica y de la distancia y orientación a cada una de las marcas visuales presentes en el campo. El método *Single Landmark* consistía en determinar la posición del robot a partir de la percepción de una marca visual. Cuando se percibe un elemento visual a cierta distancia $d$, hay un área en forma de semicírculo con radio $d$ en la que es probable que se encuentre el robot. La actualización de la estimación del robot se realizaba calculando el punto medio entre la estimación anterior y este área, en dirección a la marca visual percibida.

Este método ha mostrado poder determinar correctamente la posición del robot, aunque la importancia de EKF en este trabajo es limitada. No hay apenas trabajos que usen EKF en este entorno para localizar al robot, aunque sí para determinar la posición de las marcas visuales y la pelota. En la siguiente sección presentaremos otro de los conjuntos de técnicas en que hemos clasificado los trabajos presentados en este capítulo.

## 2.2. Métodos Markovianos

La denominación de ”localización Markoviana” se debe a a los trabajos de Simmons y Koening [SK95]. En este trabajo se aplicaba el mecanismo denominado procesos de decisión de Markov en entornos parcialmente observables (*Partially Observable Markov Decision Processes*, POMDP) para la navegación de un robot móvil en entornos de oficinas.

Un POMDP es una generalización de los procesos de decisión de Markov (*Markov Decision Process, MDP*). Los MDP proporcionan un marco matemático para modelar la toma de decisiones en situaciones donde los resultados son parcialmente aleatorios bajo el control de un controlador. Un MDP es un proceso de control estocástico caracterizado por un conjunto de estados en los que hay varias acciones entre las que un controlador debe seleccionar. Realmente, los MDPs son cadenas de Markov, pero incluyendo varias opciones de actuación en cada estado y recompensas.

Un POMDP modela un controlador de un sistema determinado por un MDP, dado que el controlador no puede observar directamente el estado. En lugar de esto,

debe inferir una distribución de probabilidad sobre el estado de un agente basándose en un modelo del mundo y en observaciones locales.

La solución a un POMDP se basa en hallar la acción óptima, dado que un agente se encuentra en un estado y en un instante, teniendo en cuenta la creencia sobre un conjunto de estados. La acción óptima maximiza la recompensa esperada en un horizonte posiblemente infinito. La secuencia óptima de acciones se conoce como la política óptima del agente para interaccionar con su entorno.

Un POMPD modela la relación entre un agente y su entorno. Se define con una tupla $(S, A, O, P, R)$, donde $S$ es el conjunto finito de estados. $A$ es el conjunto finito de acciones. $O$ es el conjunto finito de observaciones. $P$ es la política de selección de acciones. $R$ es una función de recompensa que se define como

$$R : A \times S \rightarrow \Re \tag{2.32}$$

En cada periodo de tiempo, el entorno se encuentra en algún estado $s \in S$ y el agente realiza una acción $a \in A$. Al realizar la acción $a$, el entorno transita del estado $s$ al estado $s' \in S$ con una probabilidad $p(s'|s, a)$. Entonces el agente recibe una recompensa cuyo valor esperado es $r(s, a, s')$.

$Bel(s)$ es la creencia de que en el agente se encuentre en el estado $s$. Dado $Bel(s)$, después de realizar la acción $a$ y tomando la observación $o$, la creencia resultante $Bel'$ es

$$Bel'(s') = \eta P(o|s', a) \sum_{s \in S} P(s'|s, a) Bel(s) \tag{2.33}$$

donde $\eta = P(o|Bel, a)$ es una constante de normalización tal que,

$$P(o|Bel, a) = \sum_{s' \in S} P(o'|s, a) \sum_{s \in S} P(s'|s, a) Bel(s) \tag{2.34}$$

La posición del robot se establece como el estado con mayor probabilidad.

Los trabajos presentados que usan técnicas markovianas los vamos a clasificar dependiendo del tipo de sensores que usen, de tal manera que separaremos aquellos que usan mayoritariamente información *densa* procedente de sensores de distancia (láser, ultrasonidos, etc.) y aquellos que usan otro tipo de sensores. Al final, descri-

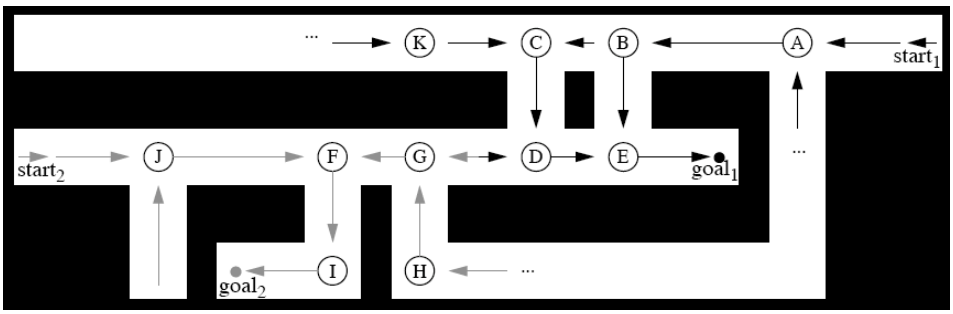


Figura 2.8: Entorno de oficinas dividido en nodos (círculos) y arcos (flechas).

biremos la implementación de un algoritmo markoviano que usa técnicas de lógica difusa como un caso particular y relevante desde el punto de vista de esta tesis, por desarrollarse en el entorno de las 4 patas de la RoboCup y por ser la base de los métodos desarrollados en esta tesis.

### 2.2.1. Trabajos que usan información sensorial *densa*

El trabajo de Simmons y Koening, publicado en 1995 ([SK95]), fue una de las primeras aproximaciones que usaba POMDPs para la navegación de un robot móvil autónomo en un entorno de oficinas. El entorno de oficinas se representaba por medio de un mapa topológico formado por nodos y arcos (figura 2.8). Cada nodo representaba un cruce de un pasillo con otro pasillo, con una puerta o un vestíbulo. Los arcos unían estos nodos y contenían la distancia real que había entre los nodos que unían.

En el trabajo presentado en [SK95] no se usaba toda la funcionalidad que aportan los POMDP. La navegación se planificaba usando un algoritmo $A^*$. Posteriormente, en 1997, presentaron un trabajo, [KS98], donde se estudiaban las diversas políticas para generar las trayectorias de navegación usando todos los recursos que tiene el mecanismo POMDP. Los experimentos del trabajo de Simmons y Koening se realizaron tanto en un simulador como en el robot móvil Xavier, descrito en la sección 1.2 de esta tesis.

Sin embargo, este trabajo se apoya en otro previo realizado alrededor del robot Dervish [NPB95] (figura 2.9). En este trabajo el robot debía navegar en un entorno de oficinas. Debía partir de una habitación, cruzar un pasillo y entrar en otra habitación.

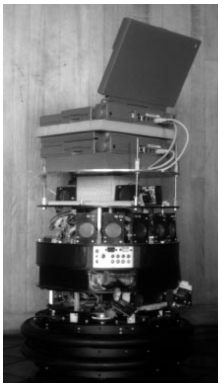

Figura 2.9: Dervish (1995).

Esta es la prueba planteada en el evento de Office Delivery[1], en la Competición y Exhibición de Robots de la AAAI en 1994, en la que Dervish participó, llevando a cabo el recorrido completo.

El robot Dervish era un NOMAD 100 equipado con 16 sensores de ultrasonidos, que originalmente estaban dispuestos en forma de anillo. Esta disposición se modificó en Dervish para poder detectar obstáculos a diferentes alturas (mesas, papeleras, etc.) y poder detectar el ángulo con respecto a las paredes. Estaba equipado con dos ordenadores. El primero de ellos permitía comunicación inalámbrica y se usaba para las tareas de navegación y localización. El otro ordenador se encargaba de generar sonidos y voz.

Este robot usaba ordenadores a bordo cuando sus robots contemporáneos solían estar conectados a una computadora fija. Las razones que argumentaban para esto es su independencia a pérdidas de comunicación debidas a interferencias, y al aumento de la autonomía del robot, capaz de cubrir áreas más amplias.

Este robot representaba el entorno mediante un mapa topológico, como el mostrado en la figura 2.10. Los autores argumentaban que modelar el entorno con estados de grano más fino está condenado al fracaso debido a la enorme complejidad computacional que se generaba.

[1] http://www1.elsevier.com/homepage/sac/robot/rcc/index.htt?mode=abs

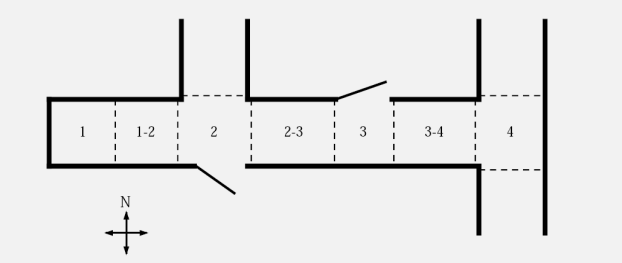


Figura 2.10: División topológica del entorno de oficina.

El robot mantenía la creencia sobre su posición como un conjunto de estados, en el que cada estado tenía asociado una probabilidad que indicaba la posibilidad de que el robot se encontrara en él. El robot asumía que su posición era el estado con mayor probabilidad. A partir de esta posición, el robot generaba un plan de acciones para llegar a su destino. Mientras realizaba este plan, el robot usaba sus sensores para colocarse paralelo a las paredes a una distancia prudencial y esquivar obstáculos.

La probabilidad asociada los estados evolucionaba incorporando la información sensorial proporcionada por los ultrasonidos. Esta información podía ser errónea y por esta razón se usaban *matrices de certeza*. Estas matrices codificaban la probabilidad de detectar los elementos del entorno (puertas, paredes y pasillos) dependiendo del estado en que se encontrara el robot.

En este trabajo la división del entorno se suponía realizada manualmente. Debido a esto, es costoso implantar este sistema en otro entorno diferente. En [CKK96], en cambio, dado un mapa métrico del entorno, se creaba una representación interna del entorno formada por una rejilla regular de nodos de 1 metro cuadrado. Un ejemplo de esta división regular en estados se puede apreciar en la figura 2.11.

Cada estado, además de la probabilidad de que se encuentre el robot en él, tenía asociado un tipo. Este tipo es *habitación* ó *pasillo*. Este tipado ayudaba a definir las observaciones ideales en cada estado. Por ejemplo, si dos estados adyacentes tenían diferente tipo, se suponía que en esta posición se debía detectar una puerta.

En este caso el sistema se implementó sobre un robot móvil B21, fabricado por *Real World Interface, Inc.*. Era un robot que se desplazaba mediante 4 ruedas y estaba equipado con 24 sensores de ultrasonidos y 24 sensores de infrarrojos.

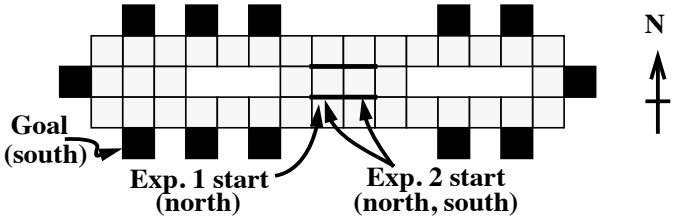


Figura 2.11: División regular del entorno en [CKK96].

El objetivo principal era encontrar estrategias de control óptimas para tareas de navegación. Aunque el objetivo principal no era la auto-localización del robot, en este trabajo se realizaba una aportación significativa con respecto al trabajo anterior, al modelar el error en las acciones, y no solo en las observaciones.

La arquitectura de este sistema presentaba dos niveles: *navegador* y *piloto*. El navegador recibía las observaciones, mantenía una distribución de probabilidad sobre el conjunto de estados en que el robot puede estar y elegía la siguiente acción. El piloto se ocupaba de llevar a cabo las acciones de alto nivel y procesar las observaciones para el navegador.

El navegador recibía un mapa del entorno que dividía automáticamente en una rejilla de $1m^2$, con cuatro estados por celda, correspondientes a las orientaciones Norte, Sur, Este y Oeste.

La creencia sobre la posición del robot se representaba como una distribución de probabilidad sobre el conjunto total de estados (4 $\times$ número de metros cuadrados del mapa). La actualización de esta distribución se realizaba incorporando *acciones abstractas* y *observaciones abstractas*. Las acciones abstractas eran acciones de alto nivel que llevaba a cabo el nivel inferior (el piloto). Eran un conjunto reducido de acciones: *desplazarse adelante* (1 metro)*, girar a la derecha, girar a la izquierda, no hacer nada* y *declarar un objetivo conseguido*. En este trabajo se asumía que las acciones podían fallar. Por ejemplo, era posible que se le comandase al robot avanzar un metro, lo que debía suponer que avanzaba al siguiente estado. El error de esta acción se codificaba asociando a la posibilidad de pasar correctamente cierta probabilidad, a la de no alcanzar el siguiente estado y a la de avanzar dos estados otra probabilidad.

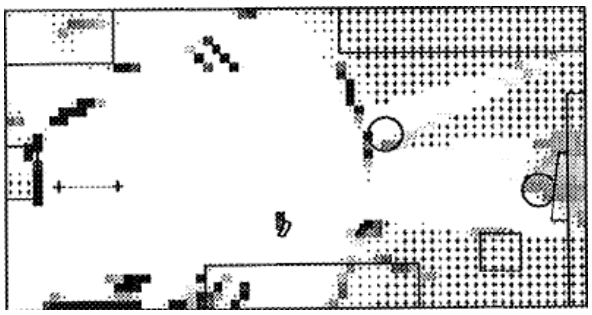

Figura 2.12: rejilla de ocupación donde se fusiona la información sensorial.

Las observaciones abstractas, también generadas en el nivel inferior, estaban limitadas a los elementos: *pared, puerta espacio abierto* o *indeterminación* que el robot se podía ancontrar a su derecha, a su izquierda o delante de él.

El piloto llevaba a cabo las acciones abstractas y generaba las observaciones abstractas a partir de sus sensores. El piloto mantenía una *rejilla de ocupación*, similar al planteado Moravec en [Mor88]. En esta rejilla, mostrada en la figura 2.12 se fusionaba toda la información sensorial de la que dispone el robot. La rejilla no mostraba todo el entorno, sino que era una representación del robot egocéntrica donde se representaba el entorno más cercano. Las celdas donde era más probable que hubiera un obstáculo están mostradas en esta figura en color más oscuro. A partir del análisis de la rejilla de ocupación se generaban las percepciones abstractas. Además, se utilizaba para calcular las trayectorias de evitación de los obstáculos que se encontraba el robot mientras llevaba a cabo las acciones.

En este trabajo se usaban POMDPs para planificar las acciones de navegación. Así, se utilizaba una *función de recompensa* $S \times A \rightarrow \Re$, donde $S$ es el espacio de estados y $A$ el conjunto de acciones. Se planteaban varias estrategias de control para conseguir maximizar la recompensa y de esta manera llegar al objetivo.

Hasta ahora hemos descrito trabajos que fundamentalmente se desarrollaron en entornos de oficina representados como un conjunto de estados discreto (tanto representaciones topológicas como de rejilla). En el trabajo presentado en [BCF+98] se desarrollaba una aplicación en la que un robot realizaba tareas de guía en el "*Deutsches*


Figura 2.13: El robot RHINO.

*Museum*" de Bonn, Alemania. La tarea principal de este robot era realizar recorridos interactivos para los visitantes del museo. Además, mediante un interfaz web, se proporcionaba "presencia virtual" a personas de todo el mundo.

Este robot guía se llamaba RHINO (figura 2.13). Se trataba de un robot móvil B21, como el usado en el trabajo anteriormente comentado. La única diferencia era que este robot estaba equipo con una cámara montada encima de un cuello mecánico.

En este trabajo cada estado $s$ representaba la posición del robot en el espacio $(x, y, \theta)$. Inicialmente, la distribución de probabilidad sobre el conjunto de estados $Bel(s)$ era uniforme.

Cuando el robot realizaba la acción $a$, la probabilidad se actualizaba de acuerdo al teorema de la probabilidad total:

$$Bel'(s') = \int P(s'|s, a)Bel(s)ds \tag{2.35}$$

y cuando se incorporaba una observación, se actualiza de acuerdo con la regla de Bayes,

$$P(s|o) = \alpha P(o|s)P(s) \tag{2.36}$$

, siendo $\alpha$ un factor de normalización.

Desafortunadamente, es difícil aplicar técnicas markovianas con éxito en entornos tan densamente poblados como este (observe la imagen de la derecha de la

Figura 2.14: El robot Minerva.

imagen 2.13). Por esta razón, en este trabajo se filtraban las lecturas sensoriales que proceden de los sensores de distancia. Cada una de las 24 lecturas de distancia se clasificaban en *corruptas* o *no corruptas*. Las corruptas corresponden a aquellas que eran debido a la presencia de personas alrededor del robot, y la no corruptas eran aquellas que correspondían a objetos estáticos y permanentes del entorno. Esta discriminación se realizaba mediante un ”filtro de entropía”,

$$\triangle H(s,o) = -\int P(s)logP(s)ds + \int P(s|o)logP(s|o)ds \qquad (2.37)$$

Si la lectura sensorial hacía aumentar la certeza sobre la posición del robot ($\triangle H(s,o) > 0$), se consideraba auténtica. El problema de incorporar únicamente las lecturas que pasan este filtro era que en las situaciones en las que el robot se perdía, no se aceptaban lecturas sensoriales. Esto producía que el robot nunca se llegara a recuperar. Para resolver este problema se incorporaban todas las lecturas sensoriales que pasaban el filtro, y varias de las que no lo pasaban, elegidas al azar.

El trabajo realizado en el museo ”*Deutsches Museum*” con el robot RHINO propició que se abordara el desarrollo del robot Minerva [TBB+99] (figura 2.14) en el "*Museo nacional de Historia Americana Smithsonian*”. Las tareas que debía realizar en este caso eran similares a las de RHINO, pero en un entorno una unidad de magnitud más grande y con mayor presencia aún de público.

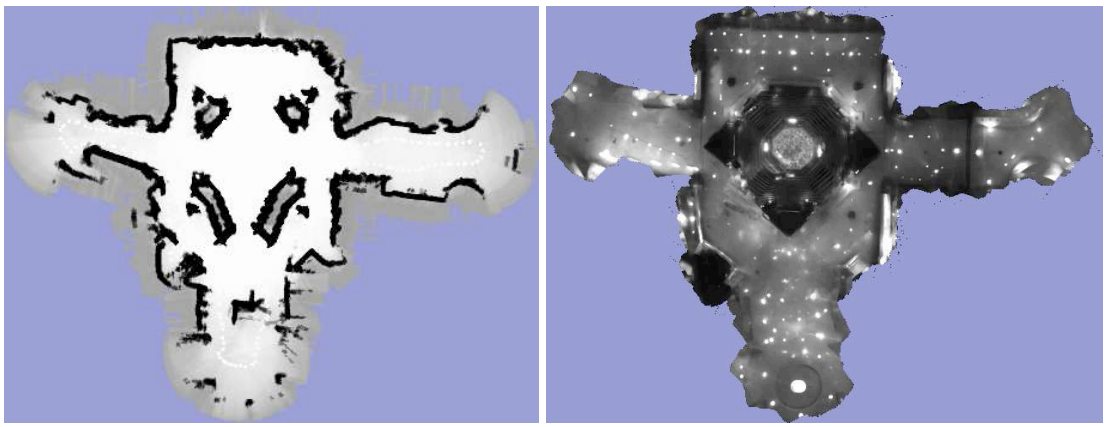

Figura 2.15: Mapa de ocupación (izquierda) y mapa de textura del techo del museo (derecha).

Este robot aportaba dos diferencias principales con respecto a su antecesor. La primera de ellas era que Minerva *aprendía* su mapa. La segunda era que utilizaba las imágenes del techo del museo como una observación más que le ayudaba a localizarse.

Minerva mantenía dos mapas: un mapa de ocupación, que le indicaba la estructura y los obstáculos fijos del entorno (imagen izquierda de la figura 2.15), y un mapa de textura del techo del museo (imagen derecha de la figura 2.15). Estos mapas los construía el robot mientras era teleoperado mediante un joystick por un operador. Durante su recorrido teleoperado, iba incluyendo en los mapas las percepciones sensoriales (láser e imagen de la cámara orientada al techo). La incorporación de observaciones y de movimientos se realiza de idéntica forma a la de su antecesor, mediante las ecuaciones 2.35 y 2.36.

Al operar en entornos muy poblados, en este caso también se clasificaban durante su operación las medidas en corruptas y no corruptas, pero usando una técnica distintas denominado ”filtro de novedad”. El robot calculaba las lecturas sensoriales esperadas,

$$E_{P(o|s,m)}[o] = \int dist(s) Bel(s) ds \tag{2.38}$$

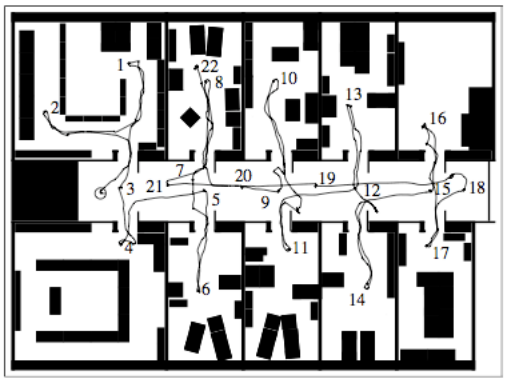


Figura 2.16: Aplicación de navegación en entornos de oficina, donde se muestra la ruta que sigue el robot..

, donde $m$ es el mapa del entorno que se aplicaba y $dist(s)$ era la medida exacta que se podía esperar si el robot estaba en la posición $s$, calculada usando *ray-tracing*. Las medidas se consideraban corruptas si eran más cortas que el valor esperado $E_{P(o|s,m)}[o]$.

Los trabajos realizados sobre Minerva y RHINO se resumen en [FBT99], donde se aplicaban estas técnicas en entornos de oficina (figura 2.16). En estos entornos se realizaban tareas de navegación, mostrando ser un método robusto y preciso.

La mayor parte de los trabajos anteriores usaban métodos donde la información sensorial correspondía a la distancia percibida por parte de cada uno de los sensores del robot. Algunos autores se han referido a los métodos que usaban este tipo de información sensorial como *métodos densos*.

Así, Konolige afirma en [KC99] que, aunque los *métodos densos* han demostrado ser efectivos, la potencia computacional que se requiere para incorporar esta información aplicando un marco markoviano es muy elevada. Él, en cambio, proponía usar una nueva técnica que se basaba en la *correlación* de las lecturas sensoriales de distancia con un mapa. Según indicaba, el tiempo de cómputo al usar esta técnica se reducía en dos órdenes de magnitud.

Esta técnica, muy parecida a técnicas de correlación propias del procesamiento de imágenes, formaba una plantilla sensorial a partir de la información de la distancia

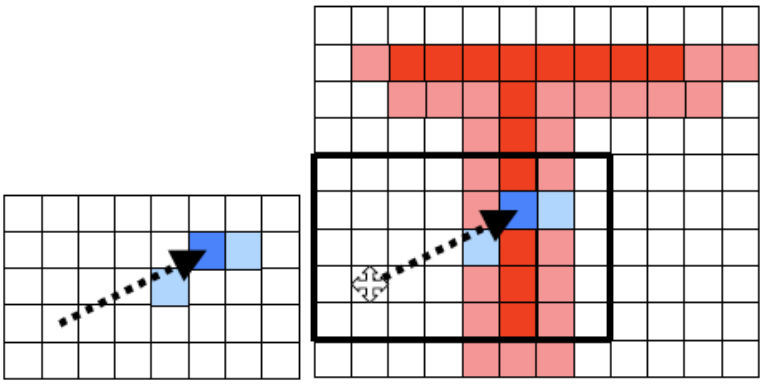

Figura 2.17: Plantilla sensorial (izquierda) y cómo se superpone a un mapa de rejilla (derecha).

a los obstáculos. Un valor alto de correlación indicaba alta correspondencia entre las lecturas sensoriales y el mapa. La operación de correlación de la plantilla sensorial se realizaba sobre todo el mapa. En la figura 2.17 se representa a la izquierda la plantilla sensorial. Esta plantilla representa la detección de un obstáculo cerca del robot. En la imagen situada a la derecha en la misma figura se observa una buena correspondencia con el mapa. La misma buena correspondencia se podría producir desplazando el robot verticalmente. De este modo se puede tener varias hipótesis de la posición del robot.

Dado una plantilla sensorial $P$ y un mapa $M$, para cada celda $i$ en $P$, se calculaba la probabilidad de obtener una lectura sensorial $o$ estando el robot en la posición $s$ y habiendo un objeto del mapa en la celda $i$,

$$c_i = p'(o|s, m_i) \tag{2.39}$$

La correlación se computaba entonces como

$$Corr(o, s) = \sum_i c_i p(m_i) \tag{2.40}$$

Para cada uno de los estados $s \in S$, su probabilidad se actualiza con la información sensorial

$$p(s|o) = kCorr(o, s)p(s) \tag{2.41}$$

### 2.2.2. Trabajos que usan información sensorial *no densa*

Desde hace algunos años el abaratamiento de las cámaras y el aumento de la capacidad de cómputo de las computadoras que forman parte de los robot han permitido que surjan varios trabajos que usan la cámara como sensor principal. Por ejemplo, el trabajo presentado en [GTC01] perseguía localizar a un robot en entornos dinámicos de exteriores. Su principal aportación era asociar la información extraída mediante el método de análisis de componentes principales (*Principal Component Analysis, PCA*) a partir de imágenes a posiciones del entorno. Esta información se usaba en su modelo de observación.

El método PCA es una transformación de un vector a otro conjunto de datos de menor dimensión usado a menudo para reducir conjuntos de datos multidimensionales . Dependiendo del campo de aplicación, a este método también se le conoce como la transformación discreta *Karhunen-Loève* la tranformación *Hotelling* o descomposición de propiedades ortogonales y se ha usado ampliamente en campos como la Robótica y el reconocimiento facial.

En [GTC01] se propuso dividir topológicamente el entorno en un conjunto de posiciones con varias orientaciones posibles por cada una. Cada posición con una orientación distinta era lo que se consideraba un estado. En una fase previa se tomaban varias imágenes en cada uno de los estados. Usando el método PCA, la información visual en cada estado se reducía a un vector de dimensiones mucho menores a las de una imagen. En este proceso, se definía una matriz de transformación que se usaba posteriormente para realizar la misma transformación a las imágenes tomadas por el robot. Cuando el robot tomaba una imagen la transformaba a este vector. Una vez obtenido el vector que representaba esta imagen $v$, se podía obtener la distancia al vector asociado a cualquier estado $s$, $d(v, s)$. Cuanto menor era esta distancia, mayor era la similitud entre los dos vectores.

En este trabajo se empleaba un POMDP para navegar. La plataforma utilizada disponía de dos cámaras $c_1$ y $c_2$ situadas perpendicularmente para obtener mayor

información sensorial (figura 2.18). La siguiente ecuación define el modelo de observación que se empleaba en el POMDP para actualizar la posición del robot.

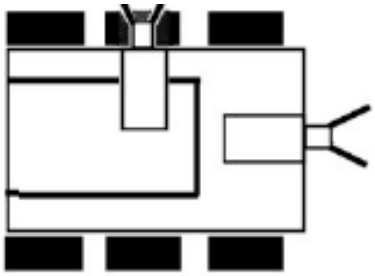

Figura 2.18: Configuración de las cámara del robot

$$p(o|s) = \frac{1}{d(v_{c_1} v_{c_2}, s)} \tag{2.42}$$

siendo $d(v_{c_1} v_{c_2})$ un valor entre 0 y 1 que representaba la similitud entre los vectores extraídos de las imágenes tomadas con las cámaras y los vectores asociados al estado $s$.

En este trabajo se usaba la imagen completa, aunque resumida por media del método PCA, para formar el modelo de observación del robot. En el caso de [Kl04], el modelo de observación se fundamentaba, como se muestra en la figura 2.19, en dos características distintas extraídas de la imagen que percibía el robot: el histograma de la imagen y las componentes invariantes a la escala (*Scale-Invariant Features (SIFT)*).

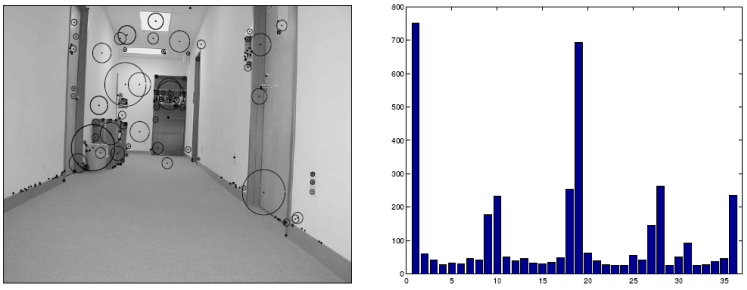


Figura 2.19: Características extraídas de la imagen: SIFTs (derecha) e histograma (izquierda)

- El histograma representa el gradiente de orientación de la imagen. Cada pixel contribuía a este histograma $h(x, y)$ a partir de su valor $I$ , de la siguiente manera:

$$\begin{pmatrix} I_x \\ I_y \end{pmatrix} = \begin{pmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{pmatrix} \tag{2.43}$$

$$h(x, y) = \frac{atan2(I_x, I_y)}{\sqrt{I_x^2 + I_y^2}} \tag{2.44}$$

  Para mantener algo de información espacial, el histograma no se realizaba de la imagen completa, sino de 5 porciones (los cuatro cuadrantes y la zona central) de la imagen original.

- La segunda característica utilizada son las SIFTs, propuestas por D. Lowe en [Low03]. Estas características son porciones de la imagen altamente distinguibles, que pueden ser detectadas eficientemente y se muestran estables ante grandes variaciones del punto de vista desde donde se toma la imagen y la escala.

Cada uno de los estados en que topológicamente se representaba el entorno tenía asociado un histograma y un conjunto de SIFTs. Esta asociación se llevaba a cabo durante la fase de creación del mapa, antes de la operación con el robot. La distribución de probabilidad sobre el conjunto de estados se actualizaba dinámicamente incorporando la información sensorial tanto del histograma como de los SIFTs.

Otra línea de trabajo es el uso como información sensorial la señal que emiten las antenas de radio frecuencia. En el caso del WiFI, es cada día más común que gran cantidad de entornos estén dotados con antenas para ofrecer servicio de red. Estas antenas suelen estar colocadas en puntos fijos y permanentes del entorno. La señal que emiten pueden medirse por las tarjetas de red de bajo coste que son cada día más habituales en robots y dispositivos móviles. La potencia de la señal percibida depende principalmente de la distancia a la antena, por lo que se usa como una sensor de distancia. La ventaja de los métodos que usan este tipo de información sensorial es que pueden usarse tanto en robots móviles como en cualquier tipo dispositivo móvil.

La parte negativa de estos métodos es la presencia de errores en la medida debido a factores tales como: refracciones, absorciones, reflexiones, etc.) que se deben a la naturaleza del tipo de ondas que hacen posible esta comunicación

Un trabajo que usaba este tipo de información sensorial es [LBR+02]. En este trabajo se presentaba un método bayesiano para la estimación de la posición de un dispositivo móvil inalámbrico (como el que pueda llevar incorporado un robot) en un entorno de oficinas.

El estado del dispositivo se representaba como una tupla $(x, y, \theta)$ que indicaba su posición métrica en el entorno. El vector de observación consiste en $k$ medidas de potencia de antenas. Cada antena podría aparecer en la medida hasta $k$ veces. El vector de observación era

$$o =< k, f1, ..., f_N, (b_1, \lambda_1), ..., (b_k, \lambda_k) >$$

donde $k$ era el número total de medidas de la potencia de la antena, $N$ era el número total de antenas, $f_i$ era el contador de apariciones de la antena $i$, $b_j$ representaba la antena en la medida $j$ y, $\lambda_j$ era la potencia de la medida para la antena $b_j$. La probabilidad de obtener una medida $o$ en un estado $s_i$ se calculaba de la siguiente manera:

$$p(o|s_i) = (\prod_{j=1}^{N} p(f_j|s_i)) \cdot (\prod_{j=1}^{k} p(\lambda_j|b_j, s_i)) \tag{2.45}$$

El modelo se definía como un conjunto de estados
$S = s_1, s_2, ..., s_n$, un conjunto de observaciones $O = o_1, o_2, ..., o_m$, una función de probabilidad $\lambda : S \times O \rightarrow [0, 1]$, y una matriz A que indicaba como evolucionaba la distribución de probabilidad. La matriz $A$ codificaba cómo se desplazaba el dispositivo móvil por el entorno. Esta matriz se usaba para calcular como evolucionaba la creencia sobre $S$ a lo largo del tiempo,

$$Bel(S_t) = ABel(S_{t-1}) \tag{2.46}$$

A se define en este trabajo heurísticamente asumiendo que el dispositivo no se desplaza a gran velocidad por el entorno y no se teleporta.

Por último, es necesario describir el trabajo presentado en [LBM03] por su relación directa con los métodos descritos en el capítulo 3 de esta tesis. En este trabajo, donde se implementa un POMDP en un entorno de oficinas, tenía como objetivo la asistencia personal. Se usó un robot con ruedas que usaba fundamentalmente una cámara para percibir su entorno y localizarse. Representaba su entorno topológicamente, como se puede observar en la figura 2.20. Cada una de las divisiones correspondían a zonas con similares percepciones y donde se podían realizar similares acciones. Básicamente identificaba zonas de pasillo y zonas donde había puertas. En cada una de estas zonas identificaba 4 estados que se correspondían con cada una de las orientaciones Norte, Sur, Este y Oeste.

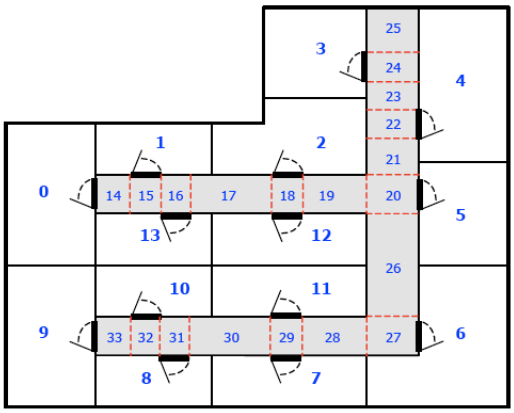


Figura 2.20: Representación topológica de un entorno de oficinas.

Mantenía una distribución de probabilidad $Bel$ sobre cada uno de los estados $s \in S$ del conjunto de estados total. Cuando el robot realizaba una acción $a_t$, actualizaba la probabilidad de cada estado como se muestra en la ecuación 2.47.

$$Bel(S_t = s) = \int P(s|a_t, s')Bel(S_{t-1} = s')ds' \tag{2.47}$$

El modelo de movimiento definía $P(s|a_t, s')$ (la probabilidad de estar en el estado $s$ partiendo de $s'$ y habiendo realizado la acción $a_t$) a partir de las limitaciones que estaban implícitas en la representación del entorno y de la probabilidad de tener éxito en las acciones. El conjunto de acciones se limitaba a giros en ambas direcciones y

avanzar hasta el siguiente estado, que era identificado mediante una detección visual de los marcos de las puertas del pasillo (figura 2.21).

Figura 2.21: Detección de los marcos de las puertas para detección de cambios de estado.

El modelo de observación se basaba en los elementos percibidos alrededor del robot y la estimación de la cantidad de techo detectado (figura 2.22). Cuando más techo se detectara, más lejos se encontraba del final del pasillo. La distribución de probabilidad se realizaba a partir de estas observaciones y la probabilidad de obtenerlas desde cada uno de los estados.


Figura 2.22: Detección de la superficie del techo.

### 2.2.3. Localización Difusa Markoviana (FMK)

Otro enfoque distinto, que combina la robustez de los modelos basados en Markov y el bajo coste computacional de los modelos difusos, podemos encontrarlo en [Saf97], y aplicado al dominio de la liga de 4 patas de la RoboCup en [BSZ00] y [HPMBS05]. Este método es especialmente relevante para esta tesis, ya que se ha implementado dentro del software del equipo *TeamChaos* y es la base de los métodos combinados propuestos en el capítulo 4.

Este trabajo se desarrolla utilizando el robot AIBO, presentado anteriormente en la sección 1.5.1. El entorno donde se desarrolla el trabajo, el campo de juego de la RoboCup, descrito en la la sección 1.5.3.

En los trabajos aplicados a la RoboCup, el campo de juego se divide en una cuadrícula $G_t$ tal que $G_t(x, y)$ representa la posibilidad, en $[0, 1]$, de que el robot se encuentre en la posición $(x, y)$. Cada una de las posiciones de esta cuadrícula es una celda difusa de dimensión configurable denominada *fcell*, representada por medio de un trapezoide difuso (figura 2.23). Cada *fcell* contiene contiene información sobre la posibilidad de que el robot se encuentre en esa celda, e información sobre cual es el rango de orientaciones más probables para el robot, es decir, se trata realmente de una cuadrícula de $2\frac{1}{2}$D.

Cada *fcell* se representa por medio de un trapezoide difuso. En la figura 2.23 podemos ver este trapezoide, definido por la tupla

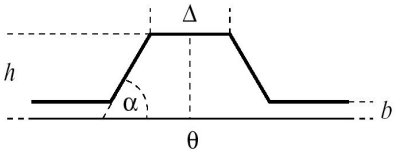


Figura 2.23: *fcell* representando el ángulo $\theta$

$$< \theta, \Delta, \alpha, h, b >$$

donde $theta$ es el centro del trapecio, definido en el rango $[-\pi, \pi]$, y codifica la orientación del robot dentro de la celda con una incertidumbre de $\Delta$. La componente $\alpha$ es la inclinación de los lados de este trapecio. La componente $h$ es la altura del trapezoide, definido en el rango $[0, 1]$. Esta componente codifica la posibilidad de que el robot se encuentre en esta celda. Por último, la componente $b$, definida en el rango $[0, 1]$, define la posibilidad mínima de de la celda.

Intuitivamente, si $h$ es bajo, la probabilidad de estar en esta celda es baja. Si $h$ es alto, es muy probable que el robot esté en esta posición. Si el trapezoide es ancho, existe gran incertidumbre sobre la orientación del robot. Si el trapezoide es estrecho, o tiene incluso forma de triángulo (porque $\Delta$ es prácticamente nulo), la incertidumbre de orientación es tan baja que podemos afirmar que es $\theta$.

Al igual que en el resto de los métodos markovianos, el proceso de localización es iterativo, teniendo cada ciclo un paso de predicción y otro de actualización. La fase de predicción se realiza cada vez que se realiza una acción, difuminando la posibilidad en la dirección del movimiento. En la fase de actualización se incorpora la información visual. Cada observación de una marca visual se compone de una distancia y un ángulo. La incorporación de estas dos componentes de la observación se realiza por separado.

Para calcular la posibilidad de cada una de las celdas de la rejilla tras una observación de una marca visual en el instante $t$, construimos la distribución de probabilidad $S_t(\cdot|r)$, tal que $S_t(x, y|r)$ es la posibilidad de que el robot se encuentre en la posición $(x, y)$, siendo $r$ el módulo del vector $\overrightarrow{r}$, que contiene la información de distancia y orientación a la marca visual percibida. Para cada observación, actualizamos el mapa $G_t$ de la siguiente manera,

$$G_t(x, y) = G_t(x, y) \times S_t(x, y|r) \tag{2.48}$$

sonde $\times$ denota el operador de intersección difusa. Esta operación puede llevarse a cabo de mucha formas, y en este caso se usa el operador de multiplicación, ya que las observaciones son independientes entre sí.

Para incorporar la orientación a la marca visual percibida se realiza una intersección entre cada una de las celdas de la distribución $S_t(x, y, \theta|\overrightarrow{r})$ y $G_t(x, y, \theta)$ como se muestra en la figura 2.24. Si ambos trapecios se solapan, como se muestra en la parte superior de esta figura, es porque la observación es coherente con la orientación anterior de la celda y la incertidumbre disminuye. En el caso de no solapar porque la observación no sea coherente con la orientación de la celda, se aumenta la incertidumbre, como se muestra en la parte inferior de esta figura.

La incorporación de la acción en la fase de predicción se realiza difuminando la posibilidad entre las celdas cercanas. Para esto se ha usado un operación de dilatación similar al usado en técnicas de visión por computador, donde la posibilidad de cada celda se combina con las que están a su alrededor.

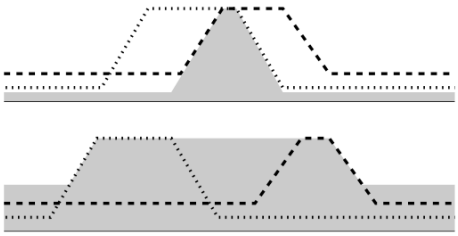

Figura 2.24: Intersección de dos trapecios difusos.

En cada momento, la posición del robot se calcula por el centro de la región de la rejilla con mayor posibilidad. La calidad de esta información se define a mediante la extensión de esta región.

Un ejemplo de la aplicación secuencial de estas dos operaciones se puede observar en la figura 2.25. El robot parte de un estado de total incertidumbre, y mediante la información odométrica y la información de la posición relativa de la portería y de la baliza superior derecha, termina localizándose correctamente.

## 2.3. Métodos basados en Monte Carlo

Los métodos basados en técnicas markovianas tienen como principal inconveniente la gran cantidad de cálculo necesaria para actualizar todos los estados en que el robot se puede encontrar. Esto es especialmente crítico en el caso de entornos extensos o en el caso de necesitar gran precisión, debido a que el número de estados puede ser muy grande. Por esta razón se han buscado alternativas que solucionen este problema.

A final de la década de los noventa se comenzó a aplicar una técnica denominada localización de Monte Carlo (*Monte Carlo Localization*, MCL) para la localización de robots móviles. Esta técnica ya era conocida con anterioridad, y se aplicaba habitualmente en áreas como la visión computacional, las redes dinámicas probabilísticas, estadística, el seguimiento de objetos, etc.

En MCL, en lugar de usar una función de probabilidad para representar la incertidumbre sobre la posición del robot, usa un conjunto de muestras aleatorias extraídas de esta función de probabilidad. Cada vez que el robot se mueve o toma una observa-

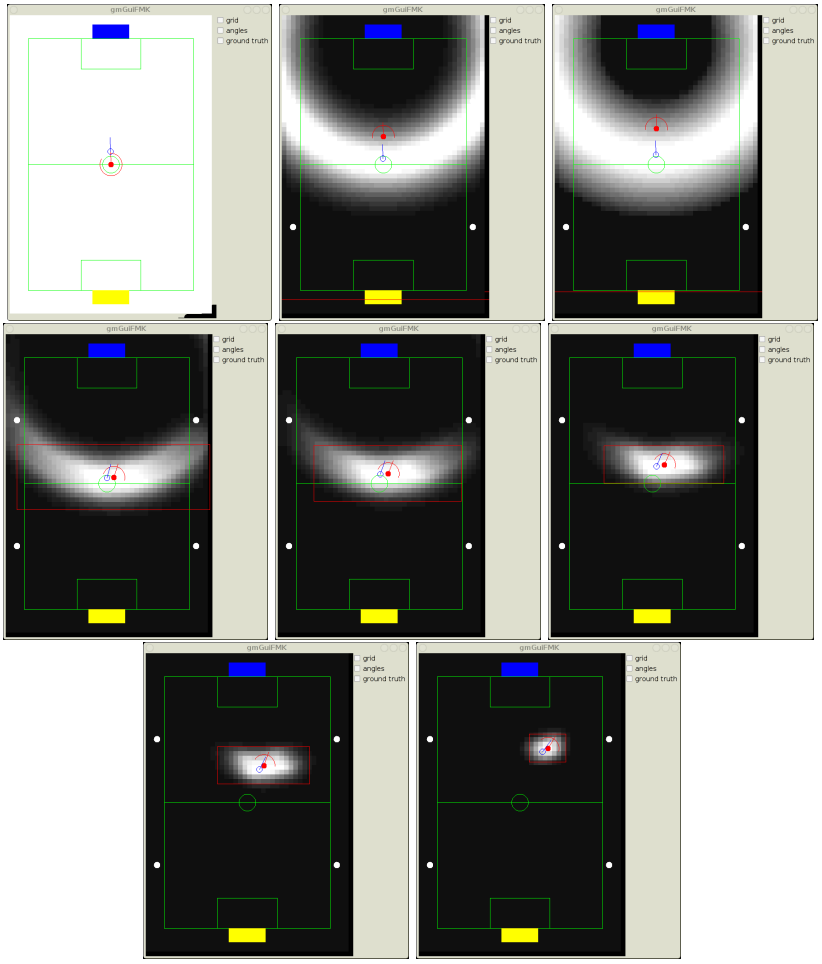

Figura 2.25: Proceso de localización del robot mediante grid borroso. Se parte de una situación de total ignorancia (arriba izquierda). Con la información sensorial de la portería y de la baliza izquierda el robot consigue afinar su posición para conseguir una localización fiable (derecha abajo) .

ción, se modifica el conjunto de muestras. Esta modificación puede consistir en una variación de la información de cada muestra o añadiendo o eliminando muestras.

Con esta técnica se consiguen importantes ventajas con respecto a las técnicas anteriormente descritas:

- Con respecto a los Filtros de Kalman, esta técnica permite representar distribuciones multimodales.
- Con respecto a las aproximaciones Markovianas basadas en rejilla, reduce considerablemente la cantidad de memoria requerida y el tiempo de computación.
- Además, es más preciso que cualquier aproximación basada en rejilla, ya que el estado representado por las muestras no está discretizado.
- Es más sencillo de implementar.

Hay que señalar que este método de auto-localización se describe en este capítulo por completitud. A pesar de las ventajas que presenta respecto al los algoritmos descritos anteriormente en este capítulo, este algoritmo no se ha tenido en cuenta en el desarrollo de los métodos de auto-localización aportados en esta tesis.

En el año 1999 se publicaron varios trabajos ([DFBT99] y [FBDT99]), desarrollados en el robot RHINO y Minerva, que supusieron la consagración de la aplicación de esta técnica en robot móviles autónomos.

La idea principal de esta técnica era representar la creencia $Bel(s)$ sobre la posición del robot como un conjunto de muestras o *partículas* (de aquí la razón de por qué se denomina también a este método "filtro de partículas") $S = s_i, i = 1 \dots N$. Este conjunto de partículas suponía una aproximación discreta a una distribución de probabilidad. Cada partícula contenía la siguiente información,

$$< s, p >=< (x, y, \theta), p >$$

donde $s = (x, y, \theta)$ era la posición del robot, y $p \geq 0$ era un peso asociado a la partícula, análogo a una probabilidad discreta. Por consistencia, $\sum_{n=1}^{N} p_n = 1$.

Cuando el robot se movía, MCL generaba $N$ nuevas muestras para aproximar la posición del robot tras realizar el movimiento. Cada una de las nuevas muestras se

generaba aleatoriamente a partir de las muestras que existían antes del movimiento, con una probabilidad asociada al valor $p$ de cada una. Si $s'$ era la posición de una de las partículas en el conjunto de muestras anterior, la posición de la nueva partícula se calculaba cogiendo una sola muestra de $p(s|s', u)$, siendo $u$ la acción realizada. El valor $p$ de la nueva partícula era $N^{-1}$.

Cuando el robot obtenía una observación $o$, variaba los pesos del conjunto de partículas del mismo modo que se hacía en las aproximaciones markovianas,

$$p \leftarrow \alpha P(o|s) \tag{2.49}$$

donde $\alpha$ era el factor de normalización que forzaba a que $\sum_{n=1}^{N} p_n = 1$. Esta normalización se realizaba en dos pasos, primero se calculaban todos los valores $p$, y luego se normalizaban.

La cantidad de partículas $N$ no era constante. MCL ajustaba esta cantidad dinámicamente. Antes de normalizar los valores $p$ de las partículas, se tomaban las partículas resultantes de la incorporación de la observación, ordenadas de mayor a menor por su valor $p$. Se comenzaban a sumar $p$, y cuando excedían de un umbral $\eta$ se eliminaban las partículas que no se habían contabilizado aún. Esto hacía que cuando el conjunto de muestras representaban con alta certeza la posición del robot el número de partículas diminuían, ya que no era necesario mantener tantas partículas para localizar al robot, y $N$ era pequeño. Si, por el contrario, no se tenía certeza sobre la posición del robot (gráfica de la izquierda en la figura 2.26), $\sum_{n=1}^{N} p_n$ no llegaba a superar $\eta$ y el conjunto de muestras era elevado. En la figura 2.26 se muestra la distribución de partículas en un entorno de oficina. En la figura situada a la izquierda de observa lo que acabamos de comentar. Si las partículas se concentraban en un solo punto, eran necesarias menos partículas que en el caso en que aún no se conocía la posición del robot, y éstas debían distribuirse por todo el entorno, como se muestra en la figura situada a la derecha.

Además, tras cada ciclo movimiento-observación, se añadían nuevas partículas en posiciones aleatorias del entorno. Esto permitía que se pudiera relocalizar un robot al inicio de su operación, cuando se perdía o cuando era manualmente desplazado de un lugar a otro del entorno (problema del secuestro).


Figura 2.26: Representación de la posición de las partículas en un mapa.

A partir de este trabajo han surgido variantes que optimizaban varios aspectos del método MCL. En [TFBD00] se proponía una mejora del método MCL denominada *Mixture-MCL*, basada en invertir el proceso de muestreo de MCL. Mientras MCL calculaba la nueva posición del robot incorporando la acción y después usaba la observación para modificar el peso de cada partícula, *Mixture-MCL* creaba un conjunto de partículas nuevo a partir de las observaciones y luego usaba la odometría para verificar su correspondencia con el conjunto de partículas anterior.

Este método se recuperaba más rápido de situaciones de secuestro o de total desconocimiento. Además, era computacionalmente menos costoso que el MCL básico. La desventaja era que se requería un modelo de observación que permitiera crear rápidamente un conjunto de partículas a partir de los datos sensoriales. Esto era posible en multitud de casos, sobre todo en los que se usaban balizas en el modelo de observación. En otros casos era más complicado, y se usaban técnicas tales como *árboles*, para generar el conjunto de partículas a partir de las observaciones.

Otra mejora al MCL era la propuesta en [LV00] y denominada *Sensor Resetting Localization*, SRL. La motivación de este trabajo era usar menos partículas (lo que implica menor coste computacional), soportar mayores errores en los modelos de movimiento y observación y tener en cuenta movimientos que no se podían modelar.

El método SRL añadía a MCL un paso más tras la incorporación del movimiento y de las observaciones. Denominamos $L$ a la posición del robot estimada a partir del conjunto de partículas $S$, y $o$ a la observación del entorno. Si $P(L|o)$ era inferior a

un umbral $\eta$, se reemplazaban algunas partículas con otras generadas a partir de la distribución de probabilidad dada por la información sensorial $P(s|o)$.

El número de partículas que se mantenían de la población anterior era proporcional a la probabilidad media de las partículas e inversamente proporcional a la probabilidad esperada $\eta$. La formula que indica cuántas partículas han de ser sustituidas, $ns$, era

$$ns = \frac{1 - \mu_{p_i}}{\eta} * N, \forall i \in S_i \tag{2.50}$$

Este trabajo se implementó con éxito en un robot AIBO en la liga de las 4 patas de la RoboCup. Era sencillo generar nuevas partículas a partir de $P(s|o)$, ya que la posición de las balizas era conocida de antemano (figura 2.27). Un detalle de este trabajo era que el peso de las partículas no se reiniciaba después de cada ciclo de actualización.

En este mismo escenario han surgido varios trabajos que aplican SRL. Así, en [SKS05] se describen mejoras prácticas realizadas sobre SRL para conseguir que:

- Que el robot, cuando realiza tareas de navegación, se estabilice cuando alcance al punto destino.
- Que sea robusto a colisiones.
- Que se adapte rápidamente a situaciones de secuestro.

Para conseguirlo su modelo de observación sólo tenía en cuenta el ángulo con respecto a las marcas visuales del campo. La actualización del peso de cada una las partículas se realizaba calculando para cada observación de una marca visual $l \in L$ el valor de la función $s$ para cada partícula. Esta función comparaba el ángulo esperado $\theta^l_{esp}$ a partir de la posición que contenía la partícula, y el ángulo medido $\theta^l_{med}$ :

$$s(\theta^l_{esp}, \theta^l_{med}) = \begin{cases} e^{-50\omega_l^2}, & \text{si } \omega_l < 1 \\ e^{-50(2-\omega_l)^2}, & \text{en otro caso} \end{cases} \tag{2.51}$$

$$\omega_l = \frac{|\theta^l_{med} - \theta^l_{esp}|}{\pi} \tag{2.52}$$

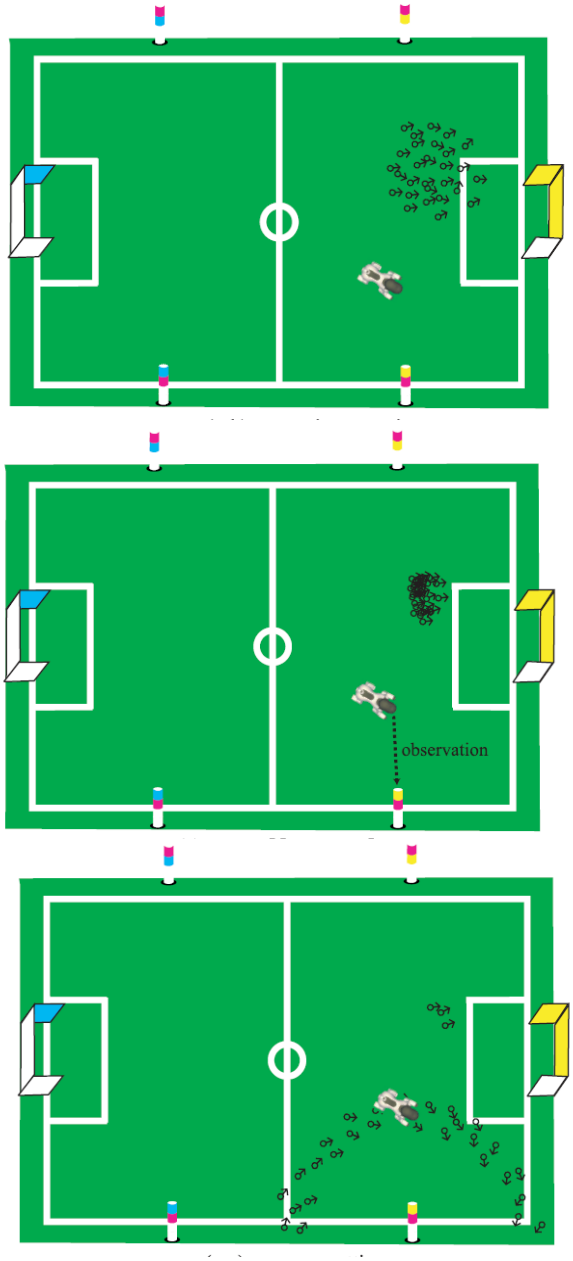


Figura 2.27: Proceso de generación de nuevas partículas con SRL.

El valor provisional del peso $p$ de una partícula se calculaba como el producto de $s$, aplicado a todas las marcas visuales observadas, ya que estas observaciones eran independientes,

$$p = \prod_{l \in L} s(\theta^l_{esp}, \theta^l_{med}) \tag{2.53}$$

El peso actualizado $p_t$ de una partícula se calcula filtrando el valor anterior de $p$. Esto se hace para ser robusto a errores en la percepción.

$$p_t = \begin{cases} p_{t-1} + 0{,}1 & \text{si } p > p_{t-1} + 0{,}1 \\ p_{t-1} - 0{,}05 & \text{si } p < p_{t-1} - 0{,}05 \\ p & \text{en otro caso} \end{cases} \tag{2.54}$$

Además, en este trabajo se mantenía un historial de las marcas visuales detectadas. La información de cada marca visual se actualizaba con el movimiento del robot y se mantenía hasta que ha pasado una cantidad de tiempo significativa o el robot se ha desplazado significativamente. Esto mejoraba la generación de partículas nuevas a partir de la triangulación de sus posiciones.

## 2.4. Discusión

A lo largo de este capítulo hemos repasado los trabajos más representativos en el campo de localización probabilística. Hemos visto tres grandes familias de algoritmos: basados EKFs, en Markov y en Monte Carlo. Existen otros enfoques no probabilísticos, pero son trabajos muy aislados que se basan principalmente en técnicas de scan-mathcing (que posteriormente evolucionan a métodos probabilísticos) y triangulación.

Los trabajos basados en métodos que usan EKF han mostrado poder corregir la estimación de la posición de un robot que se calcula a partir de su modelo de actuación mediante la incorporación de observaciones obtenidas por sus sensores extereoceptivos. El EKF es un marco matemático que proporciona una solución óptima al problema de la estimación del estado de un robot, pues permite incorporar toda la información que influye en él, y modelar de manera adecuada las imprecisiones y el ruido propios del mundo real. El coste computacional de EKF es muy bajo, lo que permite incorporarlo en aplicaciones que se desarrollan en tiempo real.

A lo largo de la primera parte de este capítulo hemos presentado los trabajos relevantes, desde el punto de vista de esta tesis, que usan esta técnica. Los trabajos que usan EKF como único método de auto-localización son capaces únicamente de realizar una auto-localiación local o *tracking* del robot, ya que modelan el estado del robot como una distribución de probabilidad monomodal gausiana. Esto plantea varios problemas en su aplicación a escenarios reales, dónde la posición del robot no es conocida. Además, para evaluar este algoritmo en caso de estimaciones erróneas o secuestro sólo se puede analizar su incertidumbre, lo que no es adecuado en la mayor parte de los casos, sobre todo en escenarios donde no se producen percepciones frecuentemente y es normal que la incertidumbre sea elevada. Es necesario combinar el método EKF con otros métodos [RB00][Pia95] para obtener la capacidad de evaluación de la estimación del estado y la posición inicial del robot.

Los métodos markovianos mantienen una distribución de probabilidad sobre el conjunto de estados en los que el robot puede encontrarse en cada momento. Tienen la ventaja, sobre los métodos basados en EKF, de que son métodos de auto-localización global. La desventaja de estos métodos es el gran coste computacional que supone actualizar la probabilidad de cada uno de los estados del espacio de estados en que el robot puede encontrarse, lo que hace que no se puedan aplicar cuando el número de estados es elevado y las actualizaciones de la estimación se producen con frecuencia, como es el caso del entorno de la RoboCup.

La aproximación basada en Monte Carlo soluciona el problema del tiempo de computación al muestrear el espacio de estados donde el robot se puede encontrar. De hecho, este método es el principalmente usado por los equipos que participan en la liga de 4 patas de la RoboCup. Hemos descrito los trabajos realizados con esta aproximación debido a la importancia que tiene en el campo de la auto-localización, aunque no se ha contemplado en la realización de esta tesis.

Además, hay que tener en cuenta que no todos los métodos de auto-localización descritos en este capítulo se pueden aplicar satisfactoriamente en esta tesis, ya que muchos de ellos están diseñados apara robots con ruedas dotados con sensores precisos de distancia. La locomoción mediante ruedas permite obtener información odométrica más precisa que en el caso de las patas. De hecho, en la mayor parte de los trabajos de auto-localización se usan en robots con ruedas, contando con información

odométrica precisa donde la incertidumbre en la actuación es fácil de modelar. Los sensores de distancia usados en estos trabajos tienen una precisión difícil de conseguir por los métodos basados en visión, que son los que se usan en esta tesis.

Como acabamos de comentar, no hay muchos trabajos que aborden el problema de la auto-localización usando un robot con patas. De hecho, fuera del ámbito de la RoboCup, los trabajos sobre este tipo de robots se limitan mayoritariamente a resolver problemas de control, no abordando aún problemas de alto nivel, como el de la auto-localización, quizás debido a que hasta el momento no han habido plataformas robóticas comerciales fuera del entorno de la RoboCup.

Los métodos de auto-localización desarrollados como principal aportación de en esta tesis se apoyan en varios de los conceptos mostrados en esta revisión bibliográfica. En primer lugar, el modelo de actuación del EKF diseñado al hilo de esta tesis usa un enfoque similar al usado en [LVRdS05], aunque aportando un estudio sobre la incertidumbre en el modelo de actuación. El modelo de observación se basa en predecir la posición de las marcas visuales que se encuentran en el terreno de juego, y compararlas con las observaciones reales, usando los mismos conceptos que se describen en[LDW91].

Como hemos comentado anteriormente, EKF por si solo no es capaz de aportar una auto-localización global, con lo que hemos optado por combinarlo con el método descrito en la sección 2.2.3, que trata de resolver el problema del tiempo de cómputo mediante la fusión de técnicas de lógica borrosa y markovianas. Este método, por sí solo es capaz de aportar la posición del robot con unos tiempo de computación razonables.

A lo largo de los siguientes capítulos nos centraremos en la resolución de los problemas planteados como objetivos en esta tesis. De hecho, en el siguiente capítulo se abordará la implementación de un método de auto-localización en el entorno de oficinas que, como hemos visto en este capítulo, es dónde se han realizado hasta ahora la mayoría de los trabajos de auto-localización.

# CAPÍTULO 3

# Aplicación de métodos markovianos

En este capítulo se presenta el diseño y la implementación de un método markoviano en dos entornos distintos: el entorno de oficinas y el de la RoboCup, presentados en la sección 1.5.2 y 1.5.3, respectivamene. El trabajo desarrollado en este capítulo sirve como punto de partida para los métodos desarrollados en el capítulo 4, orientados principalmente a este segundo entorno.

El objetivo del trabajo realizado en este capítulo es evaluar un método ”clásico” desarrollado para robots con ruedas y comprobar si también es adecuado para la auto-localización de un robot que se desplaza con patas y que usa como sensor principal una cámara en el entorno de la RoboCup. Este robot ha sido presentado previamente en la sección 1.5.1.

En la siguiente sección se analizan las características que rodean al problema que se desea solucionar y se barajan las opciones más adecuadas.

## 3.1. Introducción

En el capítulo anterior se realizó un estudio sobre los trabajos realizados para desarrollar métodos de auto-localización. En este estudio se mostró que muchos de los

trabajos considerados ”clásicos” han demostrado poder aportar con éxito soluciones al problema de la auto-localización.

El entorno donde se han desarrollado los principales trabajos ”clásicos” de localización es el de oficinas. Esto es debido a ser propicio para un gran número de aplicaciones de robótica móvil y presentar una estructura fácilmente codificable en un robot. Es un entorno muy regular donde existen elementos comunes como son puertas, pasillos y habitaciones.

Gran parte de estos trabajos han sido evaluados con robots equipado con ruedas, sensores de distancia y, en algunos casos, una o varias cámaras. Estas características ofrecen una serie de ventajas. Al desplazarse mediante ruedas, el sistema de locomoción se simplifica. El desplazamiento se produce modificando la velocidad de giro de cada rueda de manera adecuada. El cálculo de la información odométrica es simple y se realiza a partir del giro de las ruedas. A pesar de ello, pueden existir errores en el cálculo de la odometría debido a deslizamientos, aunque la incertidumbre de la información odométrica es fácilmente caracterizable. Asimismo, los sensores de distancia, comunes en la mayoría de los robots con ruedas, aportan información precisa de los obstáculos alrededor del robot.

Otra ventaja de los robots con ruedas es su estabilidad. Normalmente no sufren balanceos continuos. Esto es importante para la calidad de la información sensorial. Por ejemplo, si el robot está equipado con una cámara, se puede obtener una secuencia de imágenes que no esté afectada por un balanceo excesivo de la misma.

Actualmente se tiende a que los robots se desplacen con patas en lugar de con ruedas. Esta evolución resulta natural para un ser humano, en primer lugar, por su interés en producir robots más parecidos a los seres vivos. En segundo lugar, los robots con patas pueden acceder a sitios a los que a los robots con ruedas les es imposible. Un robot con ruedas no puede subir escaleras, por ejemplo, y esto limita la posibilidad que poder desplazarse en un edificio de varias plantas que carezca de elevadores. También es complicado que se desplacen por terrenos excesivamente irregulares.

El uso de robots con patas también presenta una serie de problemas a resolver. La forma en que se desplazan es más compleja que en los robots con ruedas. En el caso de los robots con patas, el desplazamiento se produce como resultado de complejos cálculos de las posiciones de cada articulación. Además, dependiendo de la forma del

robot, existen factores que con los robots con ruedas no se contemplan, como es el caso de que el robot ha de mantenerse en equilibrio mientras mueve sus patas. En el caso de robots cuadrúpedos o bípedos el problema es más acentuado que en los robots que disponen de mayor cantidad de patas.

La forma en la que se obtiene la información odométrica no es tan directa como en el caso de los robots con ruedas, en que la odometría se generaba a partir del giro de las ruedas, sus dimensiones y su relación entre ellas. En el caso de robots con patas, la odometría se calcula a partir del movimiento de las patas. Eso hace que la información odométrica sea más sensible a posibles irregularidades de la superficie o a posibles deslizamientos. La fiabilidad de esta información es muy inferior al caso del robot con ruedas.

Los robots evolucionan, pero no hay muchos trabajos que evalúen si los métodos que se usaron sobre robots con ruedas y sensores de distancia son igualmente válidos sobre robots modernos, equipados con cámaras y que se desplazan con patas.

En este trabajo se evalúa esta cuestión en dos fases. En la primera de ellas, desarrollada en la sección 3.2, el estudio se efectuará en un entorno de oficinas. En este tipo de entornos la presencia de personas alrededor del robot puede hacer que la información sensorial se vea alterada. Durante la operación los experimentos se comprueba que esta presencia no tiene gran impacto. No es habitual que las personas choquen con el robot ni le desplacen frecuentemente, ni que se produzcan oclusiones de manera sistemática. Los métodos ”clásicos” han demostrado ser adecuados en estos entornos y se evaluará si sucede los mismo si el robot tiene características diferentes.

En la segunda fase, desarrollada en la sección 3.3, el entorno será mucho más dinámico que en el entorno de oficinas. En el entorno que se propone en esta segunda fase, el fútbol robótico, sí que se produce gran cantidad de ruido sensorial que procede principalmente de los otros robot. Es un entorno muy dinámico donde los choques, desplazamientos, secuestros, oclusiones y falsos positivos son habituales. Tampoco es una ventaja que el entorno sea más reducido que en el caso de los entorno de oficina. Además, la precisión que se requiere es mucho mayor en este tipo de entornos.

Este estudio será útil porque se analizarán las dificultades que presenta el entorno, las capacidades del robot, las características del problema que se ha de resolver y

la viabilidad de los métodos clásicos. Por esta razón, los experimentos realizados en este capítulo no pretenden ser exhaustivos, sino que ofrezcan unos resultados cualitativos que muestren la viabilidad o no de los métodos usados.

## 3.2. Aplicación de métodos ”clásicos” en entorno de oficina

El objetivo de esta sección es detallar el proceso de diseño, implementación y experimentación de un método de auto-localización visual de un robot móvil con patas en un entorno de oficina usando una de las técnica descritas en el capítulo 2.

En la sección 3.2.1 se mostrarán las alternativas tenidas en cuenta para la elección de un método de localización representativo y adecuado al problema planteado en el entorno de oficinas. En la siguiente sección se detallará el estudio de los elementos necesarios para la implementación del método elegido. Por último, en la sección 3.2.4 se presentarán los resultados obtenidos con la experimentación.

### 3.2.1. Características del problema

El robot ha de realizar tareas de navegación deliberativa. Debe poder satisfacer situaciones del tipo ”navega hasta el despacho número 1”. Como se puede observar, las precisión con la que se especifican las posiciones es muy baja.

Cuando el robot se desplaza a lo largo del entorno, se puede encontrar con cambios de iluminación que pueden afectar a la percepción del entorno. También puede que el robot perciba a personas que pasan junto a él y que erróneamente las confunda con paredes o con puertas. El sistema ha de ser lo suficiente robusto como para tener en cuenta estos factores.

Al iniciarse el robot, no tiene ninguna información de su posición. Ha de ser capaz de calcular su posición partiendo de una situación de total incertidumbre. Pueden existir situaciones en la que el robot esté perfectamente localizado y sea desplazado manualmente de un lugar a otro del entorno. En este caso, ha de ser capaz de recuperarse de esta situación y volver a calcular su posición correcta.

### 3.2.2. Selección del método adecuado

Una vez presentado el problema, se debe seleccionar un método ”clasico” adecuado a los requisitos expuestos anteriormente. Entre los métodos más generales que han demostrado haber solucionado problemas de auto-localización en este tipo de entornos se han mostrado en el capítulo 2. Existen muchos otros métodos en la literatura, pero éstos son los más comunes en los trabajos que solucionan problemas generales.

El Filtro Extendido de Kalman es un método óptimo de estimación del estado de un sistema, como se ha presentado en el capítulo 2. Ofrece una precisión elevada y si el sistema se modela correctamente ofrece una solución óptima a la estimación de la posición del robot. A pesar de estas ventajas, este método no es un método global, sino de ”tracking”. Aunque este método fuera capaz de localizar al robot partiendo de una gran incertidumbre, sería incapaz de recuperarse correctamente de situaciones de desplazamiento manual del robot de una posición a otra del entorno.

Los métodos de Monte Carlo ofrecen también una precisión elevada. Además, son métodos globales capaces de localizar al robot desde absoluta incertidumbre y recuperarse de situaciones de ”secuestro”. Los métodos Markovianos tiene características similares, sólo que en este caso se mantiene una distribución de probabilidad para el conjunto total de estados y en el caso de Monte Carlo, es un método muestreado.

Atendiendo únicamente a las características intrínsecas de cada método, es complicado establecer el método más adecuado. Para decidirnos entre una opción u otra nos guiamos por el tipo de representación del entorno que vamos a usar. La representación métrica ofrece un precisión elevada, pero pensamos que no es necesaria tanta precisión para realizar la navegación global entre despacho. Basta con representar el entorno en estados que se corresponden con porciones del entorno, como puede ser ”despacho 4”, ”intersección del pasillo Sur y Este” o ”porción de pasillo situada delante de la puerta del despacho 3”. Esta división topológica permite que el número de estados sea reducido y que las localizaciones tengan un significado semántico único.

Una vez decididos a usar una representación topológica, la decisión sobre el método a usar está más clara. Al igual que en el trabajo descrito en [LBM03], del

que hemos usado como base para esta implementación, no decantamos por un método markoviano. En la siguiente sección se detallan los elementos del sistema de localización que han de ser definidos.

### 3.2.3. Adaptación al problema concreto

El fundamento del método es mantener una distribución de probabilidad $Bel$ sobre un conjunto de estados $S = \{s_1, s_2, ..., s_n\}$. $Bel(S_t = s)$ es la probabilidad de estar en el estado $s$ en el instante $t$.

Como este método se aplica sobre un conjunto de estados, lo primero es identificar los estados que representan las distintas posiciones del robot en el entorno. La resolución que queremos es la mínima posible para poder llevar a cabo las tareas de navegación. Esto simplifica el sistema y hace que el tiempo de cómputo sea menor, aunque no existan grandes restricciones en este sentido.

A continuación, se definen a partir de un mapa métrico tradicional (por ejemplo el realizado por el arquitecto al diseñar el edificio) los nodos que indican las diferentes posiciones del robot que son relevantes para la aplicación. Estas posiciones se definen por las alternativas de navegación que el robot tenga en cada una. De esta manera, como se muestra en la figura 3.1, el entorno se representa como un conjunto de nodos interconectados. Para definir cada uno se tiene en cuenta las intersecciones de pasillos y las porciones de pasillo situadas enfrente de una o varias puertas, ya que en estas posiciones el robot tiene la posibilidad de entrar en una habitación o seguir por el pasillo.

Se definen dos tipos de nodos: pasillo y despacho. La razón de no realizar divisiones en estados dentro de los despachos es que las tareas de navegación que el robot lleva a cabo son entre despachos, no dentro de los mismos. Para navegar dentro del despacho se prevé que el robot use algún otro algoritmo de navegación concreto para esta tarea.

Una vez representado el entorno en un conjunto de nodos, hay que tener en cuenta que en cada uno de ellos hay varias orientaciones posibles. Estas orientaciones en los nodos constituyen zonas del entorno con valor semántico desde el punto de vista de la navegación, y que forman el conjunto de estados en los que el robot se puede

encontrar. Por ello, se toman como estados diferentes de cada uno de los nodos pasillo, y cada una de las cuatro orientaciones Norte, Sur, Este u Oeste, como se muestra en la figura 3.2. No es necesaria mayor resolución angular, ya que es suficiente para poder navegar de un despacho a cualquier otro en un entorno de pasillos con ángulos rectos como el que nos ocupa. El proceso completo se ilustra en la figura 3.1, donde se representa el entorno de dos pasillos y 4 despachos con 9 nodos que dan lugar a 28 estados.

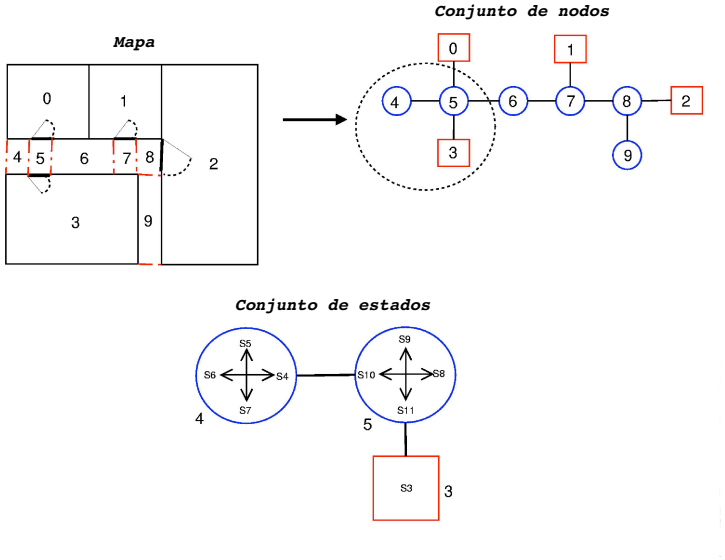


Figura 3.1: Formación de los estados a partir de la descomposición de los nodos del entorno

Esta representación tiene como ventaja añadida la característica de *vencidad* de estados. Esta característica hace, por ejemplo, que los estados destino tras un desplazamiento del robot queden reflejados implícitamente en la representación. Además, un mapa topológico almacena de forma sencilla y compacta esta información.

$Bel$ se inicia al comienzo de la operación del robot como una distribución de probabilidad uniforme. El método Markoviano se basa en estimar recursivamente la distribución de probabilidad $Bel(S_t)$. El conjunto de primitivas de movimiento, denominadas acciones, $a_t \in A$, que el robot puede realizar va a ser muy reducido, como se comentó anteriormente:

- `Avanzar` hasta el siguiente estado.

- `Girar hacia la derecha 90°`(en sentido horario).
- `Girar hacia la izquierda 90°` (en sentido antihorario).

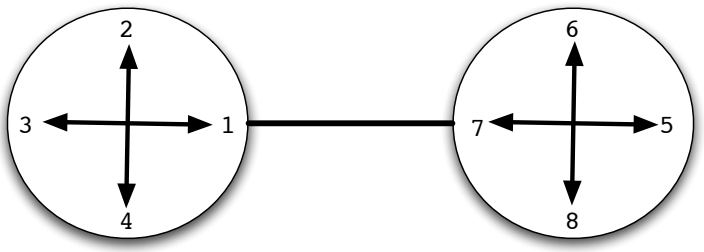


Figura 3.2: Ejemplo de dos nodos conectados

La acción de `Avanzar` hace que el robot se desplace en línea recta hacia adelante hasta detectar un cambio de estado. El robot observa cada poco tiempo los elementos (puertas, paredes, etc.) que tiene a su derecha o izquierda. Si alguno de estos elementos cambia, se considera que ha cambiado de estado. La figura 3.2 es útil para dar una idea intuitiva del efecto que tiene esta acción. Si el robot se encuentra en el estado 1 y realiza la acción de avanzar, el estado final será el 5. Esto es, en el nodo contiguo con la misma orientación. Si el robot realiza la acción de `girar hacia la derecha 90°` en sentido horario estando en el estado 1, el estado final sería el 4.

La acción de `avanzar` incorpora también un sistema para centrar y alinear el robot en el pasillo a la vez que realiza el avance. Esto permite corregir posibles deslizamientos o situaciones en las que las acciones de giro no han sido precisas. El robot tiene situado en su cabeza, debajo de la cámara, un sensor de infrarrojos que es capaz de medir distancias, aunque con una fiabilidad baja. Cada vez que el robot se dispone a comenzar un movimiento, obtiene 3 medidas de distancia a cada lado de él, como se muestra en la figura 3.3. A partir de estas medidas se puede calcular si el robot está centrado y alineado con el pasillo, y corregir así su posición y orientación.

Se ha diseñado este conjunto de operadores para que cualquier ruta pueda ser codificada combinándolos. Se ha buscado también reducir el conjunto de operadores, ya que simplifica enormemente el modelo de movimiento del robot.

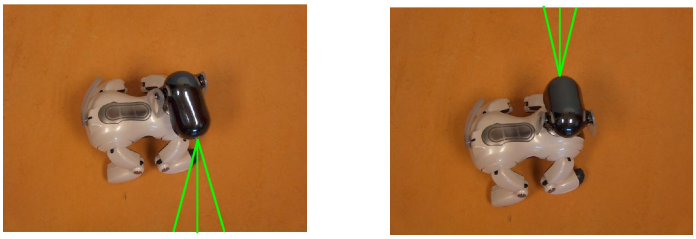

Figura 3.3: Obtención de ángulo con respecto la pared para centrarse y alinearse en el pasillo.

| $s$ \ $s'$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|
| 4 | 0.15 | 0.7 | 0.15 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0.15 | 0.7 | 0.15 | 0 | 0 | 0 | 0 |
| 6 | 0.15 | 0 | 0.15 | 0.7 | 0 | 0 | 0 | 0 |
| 7 | 0.7 | 0.15 | 0 | 0.15 | 0 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0.15 | 0.7 | 0.15 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0.15 | 0.7 | 0.15 |
| 10 | 0 | 0 | 0 | 0 | 0.15 | 0 | 0.15 | 0.7 |
| 11 | 0 | 0 | 0 | 0 | 0.7 | 0.15 | 0 | 0.15 |

Tabla 3.1: Porción del modelo de movimiento para la acción `Girar hacia la izquierda 90°`.

Cada vez que se realiza una acción se actualiza la probabilidad del conjunto de estados. Cada uno de los estados actualiza su probabilidad teniendo en cuenta la probabilidad anterior, la acción realizada y la influencia de los otros estados:

$$Bel(S_t = s) = \int P(s|a_t, s')Bel(S_{t-1} = s')ds' \tag{3.1}$$

Intuitivamente, $P(s|s', a_t)$ se entiende como la probabilidad de, partiendo del estado $s'$, alcanzar el estado $s$ habiendo realizado la acción $a_t$. $P(s|s', a_t)$ se define mediante el modelo de movimiento del robot.

El modelo de movimiento se define en la fase de diseño del algoritmo y se codifica mediante un conjunto de matrices, una para acción. Una porción de la matriz correspondiente a la acción `Girar hacia la izquierda 90°` para los estados representados en la figura 3.2 se puede observar en la tabla 3.1

| | | | | |
|---|---|---|---|---|
| GIzq | N: 0.15 | G: 0.70 | GG: 0.15 | GGG: 0.0 |
| GDer | N: 0.15 | G: 0.70 | GG: 0.15 | GGG: 0.0 |
| Avan | N: 0.20 | A: 0.6 | AA: 0.15 | AAA: 0.05 |

Tabla 3.2: Incertidumbre en la ejecución de la acción.

Los valores mostrados en la tabla 3.1 se han calculado experimentalmente tras analizar los efectos de una acción en los estados cercanos. La tabla 3.2 representa estos efectos. Para el caso de los giros a ambos lados, se ha calculado la probabilidad de cada una de las acciones mediante una sencilla experimentación en la que el robot realiza cada acción alrededor 20 veces y se comprueba en cuantos casos lleva a cabo la acción correctamente. De esta manera, la probabilidad de que el robot no realice el giro correcto y permanezca en el mismo estado (N) es 0.15. La probabilidad de que gire 90º correctamente (G) es 0.70, y la probabilidad de que gire hasta el estado situado a 180º (GG) es 0.15, y la probabilidad de que gire hasta el estado situado a 270º (GGG) es 0. De igual manera, para la acción de avanzar hasta el siguiente estado, la probabilidad de quedarse en el mismo estado (N) es 0.20, de realizar correctamente la acción (A) es 0.6, y de avanzar más allá del estado al que se quiere llegar (AA y AAA) es 0.15 y 0.05 respectivamente.

Para verificar la detección de los cambios de estado, se realizó un experimento en que el robot debía avanzar en línea recta a lo largo del pasillo y detenerse momentáneamente cuando detectara los cambios de estado. De esta manera se comprobó de forma sencilla la detección de los cambios de estado. En la figura 3.4 se muestra el resultado. Las detecciones correctas de cambios de estado se muestran como círculos azules y las detecciones incorrectas se muestran como cuadrados rojos. En este experimento se obtuvo un 82 % de éxito en las detecciones de cambios de estado. Al no disponer de ninguna herramienta de verificación, se comprobó manualmente la detección correcta del cambio de estado. Esta verificación se puede realizar fácilmente debido a las paradas que realiza el robot.

Al incorporar la observacion $o_t \in O$, el robot actualiza $Bel$ de manera que los estados varían su probabilidad según sea más probable obtener $o_t$ desde tal o cual estado. Intuitivamente, los estados desde los que es más probable obtener una

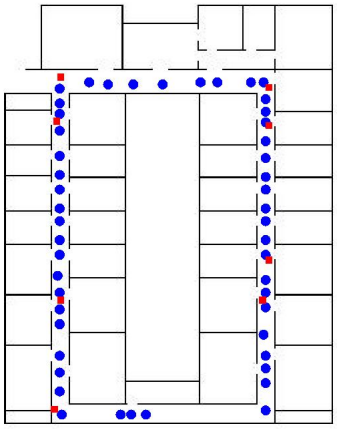

Figura 3.4: Detecciones correctas de cambios de estado (círculos azules) y detecciones incorrectas de cambios de estado (cuadrados rojos).

observación, aumentan más su probabilidad, mientras que los estados desde los que es improbable obtener una observación, ven reducida su probabilidad.

$$Bel(S_t = s) = \alpha_t P(o_t|s) Bel(S_{t-1} = s) \tag{3.2}$$

$\alpha_t$ es un coeficiente de normalización para que $\sum Bel(S) = 1$. Los elementos significativos que se ha decidido usar son los comunes en los entornos de oficina y fácilmente reconocibles. En concreto, se ha decidido usar la presencia de las paredes, las puertas y las luces del techo:

- **La observación del número de puertas** tiene como objetivo detectar las puertas usando las imágenes procedentes de la cámara. Las puertas son fácilmente detectables mediante un filtrado de color, ya son el único elemento rojo del entorno con grandes dimensiones. Un ejemplo de la extracción de esta información se puede observar en las figuras 3.6 y 3.7, donde las puertas aparecen recuadradas en azul.

- **La observación de los obstáculos entorno del robot** ofrece una información significativa y útil para estimar la posición del robot. En la figura 3.5 se representan las posibles observaciones que puede obtener el robot. El robot obtiene, mediante el filtrado de color de las imágenes obtenidas con la cámara, los ele-

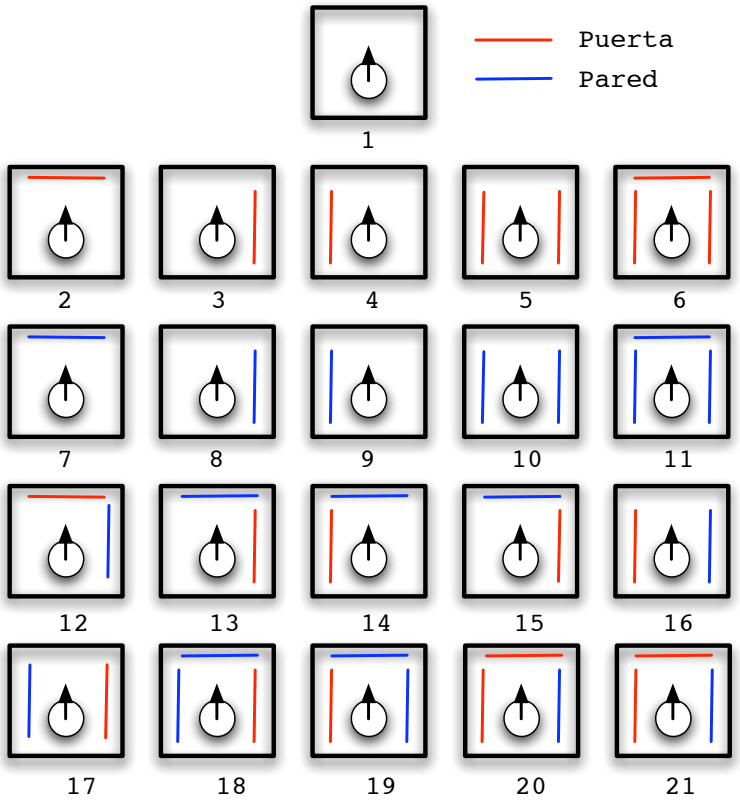


Figura 3.5: Modelo de observación para la detección de objetos en el entorno del robot.

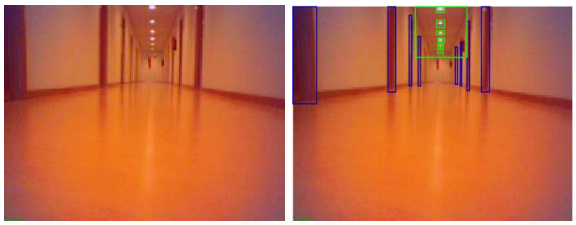

Figura 3.6: Detección de 6 luces y 8 puertas en un estado del pasillo alejada del final del mismo.

mentos que hay alrededor de él. Si hay gran cantidad de píxeles rojos, es una puerta. Si la mayor parte de los píxeles son blancos, es una pared. Si no predominan ninguno de los otros dos colores, nos encontramos ante un espacio abierto.

- **La observación del número de luces** tiene como objetivo estimar la distancia al fondo del pasillo. Un ejemplo de la extracción de esta información se puede observar en las figuras 3.6 y 3.7, donde las luces se recuadran en verde. Si se evalúa un estado orientado hacia el fondo del pasillo y se detecta una elevada cantidad de luces, indica que el final del pasillo está lejos, mientras que si se detectan pocas significa que el robot se encuentra cerca del final del pasillo.

  La detección de las luces se realiza mediante un filtrado de color, detectando las zonas más brillantes. Para limitar el área de búsqueda, se busca únicamente en la zona central superior. Esta zona de búsqueda se representa en las figuras 3.6 y 3.7 como un gran recuadro verde.

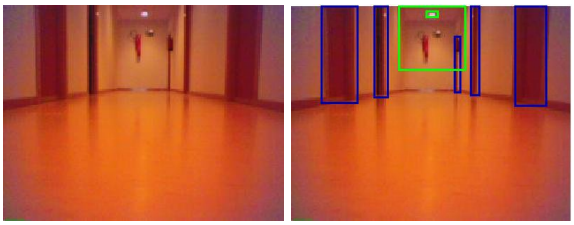

Figura 3.7: Detección de 1 luz y 5 puertas en un estado del pasillo cercana al fondo.

En el cálculo de la probabilidad $p(o_t|s)$ se tienen en cuenta los posibles errores en la extracción de información de las imágenes. Por ejemplo, si un robot en una posición debería detectar 6 luces, la probabilidad se concentra en la percepción de 6 luces. Aún así, para ser robusto a errores, la probabilidad no debería agruparse sólo en 6 luces (como la distribución degenerada mostrada en la figura 3.8) , sino que se distribuye alrededor de las 6 luces (columnas rojas en la figura 3.8).

### 3.2.4. Resultados experimentales

Para verificar el correcto funcionamiento del método implementado se llevaron a cabo varios experimentos. Estos experimentos se realizaron en nuestro entorno de trabajo en un día de actividad normal, con lo que es habitual la presencia de personas caminando en los pasillos, puertas abiertas, etc. Estas situaciones producen información sensorial errónea que pone a prueba la robustez del sistema.

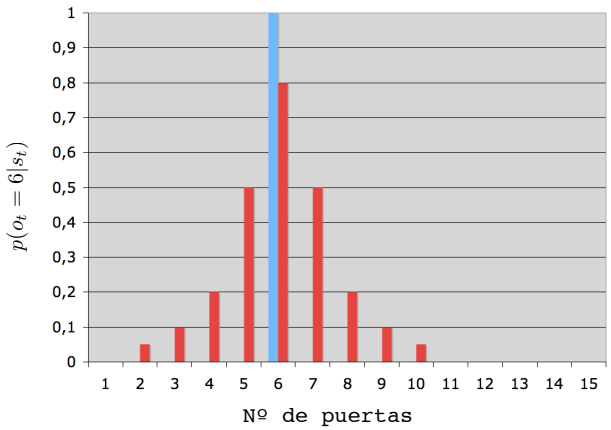


Figura 3.8: En azul $p(o_t|s)$ sin tener en cuenta errores en la observación si a priori es conocido que $o_t = 6$. En rojo $p(o_t|s)$ teniendo en cuenta errores en la observación.

Para analizar el error en la estimación que presenta el sistema de localización, se diseñaron dos experimentos: uno con la posición inicial conocida y otro con la posición inicial desconocida. En cada experimento se mide el error en la estimación de la posición. Existen varias formas de calcular este valor. En este caso decidimos contar el número de nodos de distancia que separan la posición real del robot de la estimación que calcula el sistema de auto-localización. Esta estimación es el estado con más probabilidad calculado por el algoritmo de localización, es decir, la moda de la distribución $Bel$.

En el primer experimento el robot debe desplazarse desde el punto 1 al 2 mostrado la figura 3.9, siendo la posición inicial del robot desconocida. En la figura 3.10 se muestra el error en nodos de distancia. Al comienzo, la probabilidad de cada uno de los estados es la misma, pero a partir del instante 2 la probabilidad tiende a concentrarse en unos pocos estados. Durante los primeros 11 ciclos, hay dos estados, separados entre sí, que alternativamente son los que tienen la probabilidad mayor debido a la simetría del entorno. Por esta razón se observan picos en la gráfica del error. Alrededor del ciclo 11, la estimación se estabiliza, concentrándose la probabilidad en un estado, siendo este la posición estimada del robot. A partir de este momento el robot queda localizado, aunque errores en la percepción y en la detección de los cam-

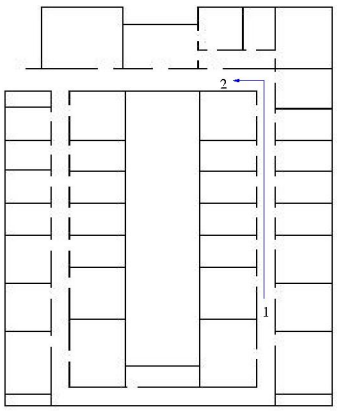


Figura 3.9: El robot se mueve de la posición 1 a la posición 2, desconociendo su posición inicial.

bios de estado producen ligeras variaciones, pero el robot se recupera rápidamente de esta situación.

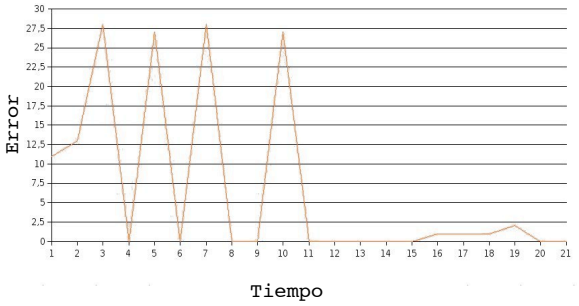


Figura 3.10: Error entre la posición real del robot y la estimación, siendo ésta última la moda de $Bel$.

En el segundo experimento el robot se desplaza del punto 1 al 2 de la figura 3.11. La posición inicial del robot es conocida de antemano, por tanto, el error inicial es 0, como se puede observar en la figura 3.12, que representa la evolución del error. En el ciclo 24, errores en la percepción causan que el robot estime erróneamente su posición por unos momentos, pero al desaparecer estos error, el robot se recupera y vuelve a estimar correctamente su posición.

Estos experimentos nos permiten extraer interesantes conclusiones:

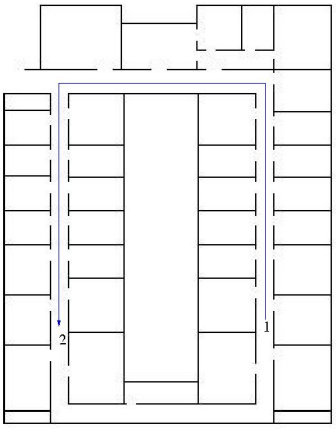


Figura 3.11: El robot se mueve de la posición 1 a la posición 2, conociendo su posición inicial.

- Esta aproximación es válida en la tarea de localizar el robot. A los largo de las ejecuciones que hemos realizado, el robot ha sido capaz de conocer su posición desde una posición desconocida.

- El robot es capaz de mantenerse auto-localizado a pesar de errores en la medida y en la realización de las acciones, recuperándose en pocos ciclos.

- Esta aproximación no requiere grandes requisitos de computación. Un entorno relativamente grande se representa con una cantidad de estados mucho menor que en las aproximaciones de rejilla comunes. A pesar del grano grueso de esta representación, el robot es capaz de realizar tareas de navegación entre distintas zonas de manera adecuada.

Después de comprobar la viabilidad de este método, se decidió tratar de adaptarlo al problema principal de esta tesis, que es la localización de robots en el entorno de la RoboCup. En la siguiente sección se muestra cómo se ha llevado a cabo esta adaptación.

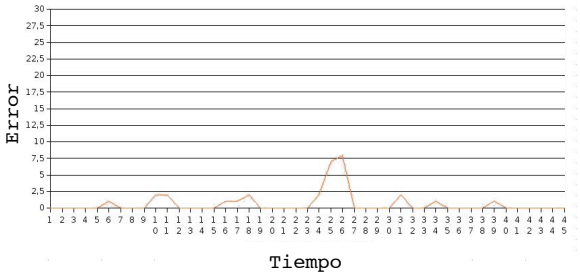


Figura 3.12: Error entre la posición real del robot y la estimación, siendo ésta última la moda de $Bel$.

## 3.3. Entorno de la RoboCup

En esta sección del estudio se desea verificar si el método empleado en la sección anterior es suficientemente general y robusto para aplicarse en un entorno mucho más dinámico.

Como ya se ha comentado, el entorno de la RoboCup es muy diferente a los entornos de oficina descritos anteriormente. Es un entorno más reducido y dinámico, donde se producen multitud de colisiones, oclusiones, falsos positivos y desplazamientos manuales. El comportamiento de los robots también es diferente: Debe mostrar un comportamiento vivaz, buscar los elementos relevantes del juego (p.e. la pelota), y realizar distintas tareas (desplazarse a una posición global del campo de juego, ir a por la pelota, colocarse entre la portería y la pelota, etc.).

Un requisito adicional al de la precisión es el consumo de CPU, ya que el robot realiza varias tareas en paralelo que no podrían llevarse a cabo si los tiempos de computación son demasiado elevados.

El primer paso es determinar cómo se presenta el entorno en estados. En la aplicación de los entorno de oficina, cada estado tenía en común el conjunto de acciones de navegación posibles, y similares observaciones que desde ella se podían obtener. Es complicado aplicar esta aproximación en el caso de la RoboCup, donde los modelos perceptivos y de movimiento son más complejos. En este caso, la división de entorno del robot en estados se puede hacer de múltiples maneras. Se puede dividir en una rejilla regular donde todos las celdas tienen las mismas dimensiones y los mismos estados para representar la orientación del robot (figura 3.13). Esto satisface

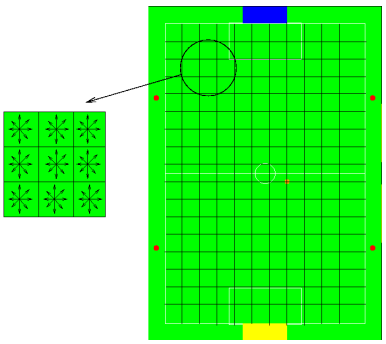

Figura 3.13: Campo dividido en nodos de la misma dimensión. Cada nodo tiene 8 estados separados de forma uniforme

la necesidad de estimar métricamente la posición del robot, si fuese necesario. La estimación métrica se obtiene directamente partir de la moda de $Bel$, y el número de estados responde a un compromiso entre la precisión necesitada y los recursos computacionales disponibles.

Otra posibilidad es configurar el espacio de estados dependiendo de las acciones que el robot tenga que desempeñar en cada zona del campo (figura 3.14) o la importancia de esa zona. Por ejemplo, un robot en tareas defensivas en la mitad de su propio campo no necesita saber nada más que si está en la derecha o izquierda del campo, y una orientación aproximada para lanzar la pelota hacia el campo contrario. Esta decisión necesita menos precisión de localización que posiciones cerca de la portería contraria por parte de los delanteros porque la localización se orienta hacia el uso que de ella se va a hacer.

El modelo de movimiento no puede ser similar al que se ha usado en los entornos de oficina. En este caso no se pueden detectar los cambios de estado en base a las percepciones del entorno del robot. En este caso sólo se puede usar la odometría que calcula el módulo de locomoción. Como comentamos anteriormente, la odometría de un robot con patas presenta una elevada incertidumbre.

Es difícil generalizar un método que sólo utilizando odometría sea capaz de detectar los cambios de estado con este tipo de división. Debido a esto se decidió que la división regular del entorno fuese una rejilla de celdas de iguales dimensiones. Únicamente ha de configurarse el tamaño de cada celda. Los cambios de estado se pueden

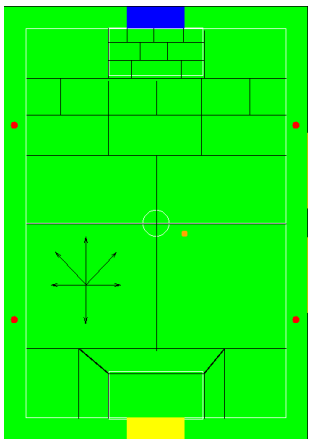

Figura 3.14: Campo dividido en nodos con distintas dimensiones. Cada tiene un número diferente de estados, que corresponden a orientaciones arbitrariamente diseñadas para cada nodo

realizar de manera genérica con odometría, y la incertidumbre se puede modelar más facilmente. Esta división permite que la información de localización sea fácilmente traducible a una posición métrica en el campo de juego.

**El modelo de movimiento** está basado en la odometría, como se comentó anteriormente. Los cambios de estado se calculan acumulando la información odométrica. Al igual que en los entornos de oficina, las acciones modifican la probabilidad de los estados. El modelo de movimiento, al igual que en el caso de interiores, se basa en modificar la probabilidad no sólo del estado teóricamente destino, sino también los estados próximos para reflejar posibles errores en el movimiento del robot, tales como los que el robot no se mueva (en el caso de colisiones), se desvíe de su trayectoria o que el desplazamiento no se corresponde con lo indicado por la información de odometría.

En la figura 3.15 se puede observar un ejemplo ilustrativo que muestra la evolución de la rejilla de probabilidad atendiendo a la información odométrica. La probabilidad se difumina a los estados cercanos, para tener tener en cuenta la incertidumbre asociada al movimiento.

**El modelo de observación** que el robot usa para localizarse es su posición relativa a las marcas visuales que existen en el campo. El sistema ha de ser robusto a

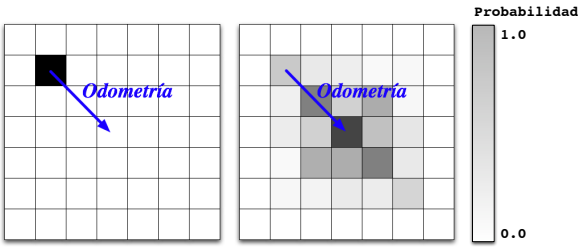


Figura 3.15: Modelo de movimiento basado en odometría en una configuración de rejilla regular. Se observa la influencia de los estados más probables en sus vecinos.

errores en estas observaciones, ya que el robot está realizando un movimiento continuo, y las medidas pueden no ser exactas.

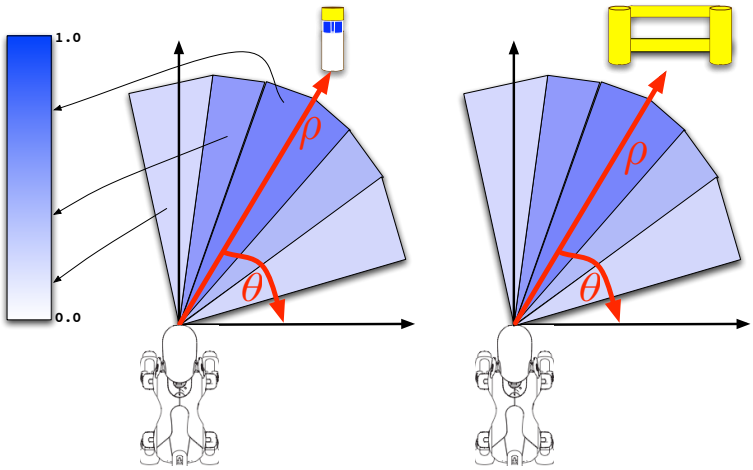


Figura 3.16: Modelo de observación $p(o|s)$ robusto a errores en la estimación del ángulo de una observación en el eje de referencia del robot.

El robot almacena principalmente dos valores para cada marca visual $(\rho, \theta)$, donde $\rho$ es la distancia a la marca, $\theta$ es el ángulo respecto al eje del robot

El modelo de observación $p(o|s)$ se define experimentalmente. Los modelos que se aplican en el caso de la portería y la baliza son similares. Por ejemplo, en la figura 3.16 se muestra una observación de una baliza y de una portería en cierto estado. En ese estado, las probabilidades de la observación de la baliza con un ángulo $\theta$, están dibujados en azul. Si la diferencia entre la observación teórica estando en $s$ y $\theta$ es menor de $10^o$ (zona en azul oscuro), la probabilidad de obtener como observación la

baliza es 0.4, si es menor de 30º (está en la zona azul claro) $p(o|s)$ es 0.25. Si está en el azul más claro, es decir, que es difícil haber podido observar la baliza con una medida $o$ desde $s$, $p(o|s)$ es 0.1.

Los experimentos realizados, a pesar de los múltiples ajustes, no consiguieron que este método obtuviese localizaciones correctas. Tanto es así, que no ha sido posible extraer algún resultado numérico mínimamente fiable, incluso sin otros jugadores. Además, surgieron varios problemas:

- El sistema no fue capaz de detectar de manera fiable los cambios de estado a partir de una odometría poco exacta.
- La enorme cantidad de falsos positivos que se presentaban en el entorno, hacía que el sistema no convergiera a una única estimación, y que ésta no se mantuviera estable.
- Para poder localizar al robot con la mínima precisión (de posición y orientación) necesaria que requerían los comportamientos que hacían uso de esta información, era necesario representar el entorno mediante gran cantidad de estados. Esto suponía demasiado tiempo de computación.

Estos problemas, sobre todo éste último, hicieron que definitivamente se decidiera explorar otras alternativas, aunque consideramos positivo describir esta aproximación a la solución del problema como motivación de los algoritmos que se describirán en los siguientes capítulos.

## 3.4. Discusión

En este capítulo se ha mostrado la misma aproximación en los dos entornos a la auto-localización de robots con patas usando un método markoviano. Este método ha probado ser efectivo en entornos de oficina con robots equipados con ruedas y sensores de distancia, como la literatura acredita, pero no se ha demostrado que sea igualmente válido en el entorno de la RoboCup, cuando el robot usado tiene patas y no cuenta con estos sensores, sino con una cámara de la cual se extrae toda la información necesaria.

En la aplicación del método markoviano en el entorno de oficinas, se ha dividido el entorno el entorno en estados, tal y como se realiza en trabajos anteriores con robots con ruedas. Las diferencias con trabajos anteriores se centran en definir el modelo de movimiento y de observación.

Para el modelo de movimiento no se ha usado exclusivamente la odometría, al considerar que existían otros métodos, como se han descrito, más fiables para la detección de cambios de estado. Como alternativa se han detectado visualmente los cambios de estado. En los experimentos se muestra cómo esta detección es fiable con el método propuesto.

También es honesto comentar que la experimentación no ha sido extensa, por no ser este estudio el objetivo principal de esta tesis. El esfuerzo se ha concentrado en comprobar si el método y los modelos son suficientes para resolver el problema propuesto con el método markoviano que ha demostrado ser adecuado con un robot con patas.

Se ha aplicado esta aproximación al entorno de la RoboCup, que tiene unas características y restricciones radicalmente distintas. Desde el inicio de su implementación en el nuevo entorno se han presentado dificultades. La primera dificultad es la división del entorno en estados y la detección de transición entre estados. Al haber restricciones importantes de tiempo de cómputo, la precisión no ha podido ser suficiente para las necesidades del resto del sistema. La detección del cambio de estado ha sido insatisfactoria, al realizarlo con odometría y ser ésta poco fiable en entornos tan dinámicos donde los choques y bloqueos por parte de otros robots son frecuentes.

A la vista de estos resultados descartamos este método markoviano de localización para el entorno de la RoboCup y nos centramos en capítulos posteriores en el diseño de un método particular de auto-localización que sea capaz de satisfacer todos los requisitos de tiempo de cómputo, fiabilidad y precisión que se requiere en esta aplicación.

# CAPÍTULO 4

# Localización métrica en entornos dinámicos

En el capítulo anterior se describió una adaptación de un método markoviano de auto-localización cuyas principales características son que se ejecuta en un robot con patas y que usa fundamentalmente la visión como fuente de información. Los entornos donde se ha probado son los entornos de oficina y el entorno de la RoboCup. Los resultados mostraron que el método usado, si bien era adecuado en entornos de oficina, no lo era en el entorno de la RoboCup. Por lo tanto es necesario desarrollar nuevos métodos que aporten soluciones adecuadas para este entorno. En este capítulo se describe el diseño de estos métodos y cómo se integran en el software del equipo TeamChaos

La integración de estos métodos de auto-localización en la arquitectura software del equipo TeamChaos no se tuvo en cuenta en el capítulo anterior, al tratarse de un estudio previo que sólo buscaba examinar la viabilidad del método estudiado, pero sí se analizará en este capítulo.

Para comprender varios aspectos analizados en este capítulo es necesario describir brevemente la interacción entre los módulos que componen el software. En el apéndice A se encuentra una descripción detallada de la arquitectura software del

equipo. Este software está compuesto por un conjunto de módulos que se comunican entre sí, como se puede observar en la figura 4.1.

El módulo que lleva a cabo la auto-localización del robot se denomina *GM* (*Global Model*). En este módulo se han insertado todos los métodos de auto-localización descritos en este capítulo. Este módulo recibe la información generada por los módulos *PAM* (*Perception and Anchoring Module*) y *CMD* (*Commander*), encargados de las tareas de percepción y de locomoción, respectivamente.

La información que genera el módulo *GM* la usa el módulo *CTRL* (*Controller*). Este módulo toma las decisiones sobre las acciones que tiene que realizar el robot, como pueden ser aproximarse a la pelota, golpearla para que entre en la portería, etc. Estas decisiones se envían al módulo *CMD* en forma de comandos de movimiento.

Varias de las tareas que realiza el robot dependen de la información de auto-localización que genera este módulo. Además, el robot lleva a cabo decisiones estratégicas implementadas por el mecanismo *Switch!*, descrito brevemente en la sección 5.4 y en [CEA06]. Estas decisiones dependen fuertemente del sistema de localización, pues se basan en hacer que el robot se coloque en posiciones específicas dependiendo del rol (atacante, defensa, centro o portero) que ha decidido *Switch!*. Es especialmente crítico el caso del portero, cuyo comportamiento para colocarse entre la pelota y la portería es extremadamente sensible a la información de auto-localización.

La información que genera el módulo *PAM* es la percepción local de las marcas visuales existentes en el entorno. La percepción local se representa mediante un estructura denominada *lps*, que contiene la posición, relativa al eje de coordenadas del robot, de los elementos del entorno, así como información complementaria que indica su validez y antigüedad. La información que genera el módulo *CMD* se compone de una odometría que representa el desplazamiento aproximado que realiza el robot.

Cada vez que la cámara obtiene una imagen se inicia un nuevo ciclo de ejecución. La frecuencia con que la cámara obtiene las imágenes es de 30 por segundo(fps). El procesamiento que llevan a cabo todos los módulos ha de realizarse dentro de este ciclo de ejecución. Por lo tanto, el tiempo máximo de procesamiento que se puede realizar en un ciclo de ejecución corresponde al tiempo que hay entre la captación de dos imágenes consecutivas ($1/30 = 33$ ms).

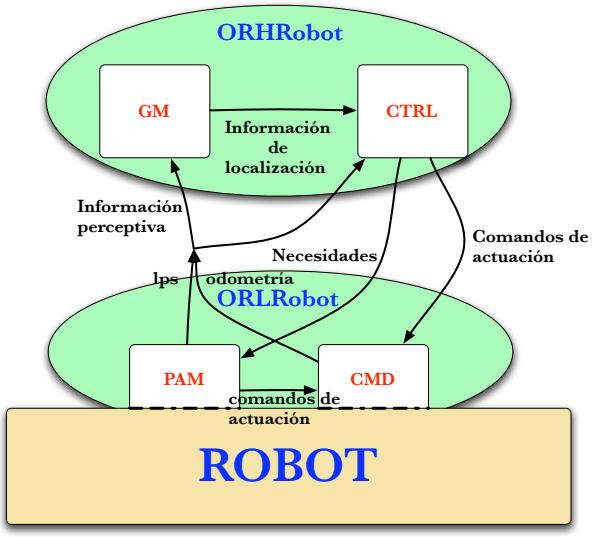


Figura 4.1: Parte de la arquitectura software de *TeamChaos*

En la figura 4.1 pueden verse varios módulos. Muchos de ellos realizan tareas que necesitan un cómputo intensivo, como pueden ser el filtrado de una imagen y el procesamiento de los comportamientos de bajo nivel. La sincronización de los módulos es vital para que el sistema funcione correctamente.

Todos los módulos realizan sus funciones dentro de un ciclo de ejecución. Para garantizar la correcta sincronización de los módulos, el tiempo total de ejecución del conjunto de módulos no puede exceder del tiempo máximo de un ciclo de ejecución. Para solucionar los problemas de sincronización que genera esta situación, se podrían descartar una o varias imágenes consecutivas, aumentando el tiempo máximo permitido. Esto va en contra de la capacidad de reacción del robot. Si se descartan demasiadas imágenes, el robot actuará con información del entorno demasiado desfasada como para poder realizar correctamente sus tareas.

Como ejemplo ilustrativo de un problema de sincronización, en determinada fase del desarrollo del software, surgió un comportamiento erróneo del sistema que hacía que se produjeran oscilaciones en el control. Se producía cuando el robot giraba para encarar una pelota que estaba muy cerca. El incremento del tiempo de cómputo, que crecía exponencialmente con el tamaño de la pelota debido al algoritmo de reconocimiento usado, hacía que el robot girara más allá de la posición de la pelota cada vez

que giraba para encararla, volviendo a tener que girar hacia al otro lado para volver a encararla, estando otra vez en la misma situación.

Atendiendo a esta arquitectura software y a los requisitos que presenta, es necesario tomar decisiones sobre los diferentes aspectos que componen el método de auto-localización que se desarrolla en este capítulo. Esta arquitectura imprime un requisito de vivacidad a las soluciones de auto-localización aceptables.

Las representaciones basadas en rejilla tienen el inconveniente de que el tiempo de cómputo necesario para actualizar todas las celdas es demasiado elevado. El método *FMK*, presentado en el capítulo 3, presenta este problema de tiempo de cómputo, aunque su diseño hace que la actualización de las celdas sea más rápida que los algoritmos markovianos clásicos que usan esta representación. Se puede minimizar este tiempo aumentando el tamaño de cada celda y haciendo que el número de celdas necesario sea menor, a costa de perder precisión. Las representaciones topológicas no parecen ser adecuadas, como se analizó en el capítulo 3.

Creemos más adecuada una representación continua del entorno. Con ello conseguimos, en primer lugar, que la precisión obtenida no se vea limitada por el tamaño de cualquier división regular o irregular en forma de rejilla que se pueda proponer, propia de aproximaciones topológicas o markovianas basadas en rejilla. En segundo lugar, que la interfaz con el resto del sistema se ve simplificada y más adecuada. Los comportamientos del robot están implementados suponiendo que la posición del robot está definida métricamente. Si el método de localización estima métricamente la posición del robot, esta información puede ser usada directamente por estos comportamientos.

El EKF y los algoritmos muestreados son los métodos continuos de auto-localización más habituales, presentados en el capítulo 2. Los algoritmos de muestreo son actualmente los más usados por los equipos que participan en la RoboCup. En este trabajo queremos abordar la opción del *EKF*, porque este método permite obtener una estimación métrica de la posición del robot con unas necesidades de tiempo de computación relativamente bajas.

Como ya hemos expuesto en el capítulo 2, usar un método local como EKF presenta varios problemas:

1. La estimación es *monomodal*. Con un *EKF* se puede mantener una única hipótesis de la posición del robot.

2. La *evaluación* es difícil. Cuando un *EKF* es erróneo o diverge la única información disponible es la incertidumbre de la estimación, y no siempre a partir de esta incertidumbre se puede evaluar correctamente la exactitud del método.

3. Problema de *inicialización*. En la sección 1.3 se dividió la habilidad de auto-localización en *local* o *global*. El *EKF* pertenece a esta primera división, ya que necesita conocer la posición inicial del robot. En el problema que se desea resolver no se cuenta con la posición inicial, con lo que es necesario algún mecanismo para calcularla. Asimismo, se debe tener en cuenta el caso de que la estimación de un *EKF* se considere errónea y deba ser reinicializado.

Para solventar estos problemas, en esta tesis se propone combinar uno o varios *EKFs* con un método global, en particular *FMK*. Lo que se pretende obtener con esta combinación en una precisión elevada, aportada por *EKF*, y la capacidad de recuperación que aporta *FMK*. La decisión de usar *FMK* como método global se fundamenta en dos razones. La primera de ellas es que tiene unos tiempos de computación razonables para resoluciones bajas. Al usarse únicamente para resolver los problemas enumerados anteriormente, no se necesita una precisión elevada. La segunda razón es práctica. Este método ya está implementado e integrado en el software del equipo.

Además del desarrollo de los métodos de auto-localización propuestos en este capítulo, es necesario obtener el modelo de observación y el modelo de actuación que se usan en ellos, particularizados para el robot y el escenario concreto de la RoboCup. La información perceptiva que produce el módulo *PAM* no contiene la incertidumbre en la percepción de cada elemento. Asimismo, la información odométrica que calcula el módulo *CMD* no contiene información precisa sobre la incertidumbre en el desplazamiento. Además, tanto la información perceptiva como odométrica pueden presentar errores sistemáticos que han de ser conocidos y corregidos. Por esta razón, se ha realizado un estudio para obtener ambos modelos de una forma precisa.

La figura 4.2 muestra un esquema donde aparecen, en forma de triángulos, dónde se sitúan los componentes software que aplican los modelos de percepción y de

actuación en la arquitectura software del sistema. El modelo de percepción recibe la percepción local del robot, `lps`. El modelo de percepción corrige los posibles errores sistemáticos del módulo *PAM* y le añade información sobre la incertidumbre en la medida. Del mismo modo, el modelo de actuación corrige la información odométrica y la completa con la incertidumbre. El módulo *GM* recibe el resultado de aplicar ambos modelos y actualiza convenientemente la estimación de posición y su incertidumbre asociada.

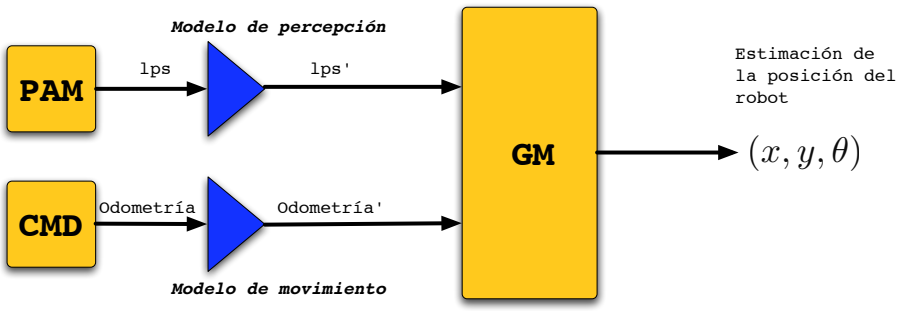


Figura 4.2: Corrección e incorporación de incertidumbre a las entradas del *GM*

Antes de describir con detalle los métodos desarrollados en este capítulo, se presentarán los modelos de percepción y actuación en las secciones 4.1 y 4.2, respectivamente. Para continuar con la descripción del sistema de auto-localización propuesto, en la sección 4.3 se describe el diseño concreto del *EKF* implementado. Finalmente, en la sección 4.4 se presenta nuestra propuesta para el desarrollo de métodos de auto-localización que combinan varios *EKF* con el método *FMK*, siendo esto nuestro aporte principal al campo de la auto-localización de un robot móvil. Los experimentos llevados a cabo para validar estas aproximaciones se describen en el capítulo 5.

## 4.1. Modelo de percepción

El robot basa todo su conocimiento del entorno que le rodea en la información que extrae de las imágenes que toma con su cámara. El entorno está ingenierizado para facilitar la percepción del robot usando este sensor. Los elementos fijos del entorno que debe percibir son las balizas y las porterías, cuya posición es conocida *a priori*.

Estos elementos tienen un aspecto y unos colores preestablecidos, como se puede observar en las imágenes tomadas por el robot que se muestran en la figura 4.3.

Figura 4.3: Imágenes tomadas por la cámara del robot.

El módulo *PAM* realiza un procesamiento de las imágenes crudas, construyendo una observación elaborada, llamada $z$, que resume la información de los elementos relevantes observados (puertas, balizas y pelota) de cada imagen.
Cada observación $z$ contiene la información relativa a un elemento del entorno. La principal información que se desea mantener en cada observación es un ángulo $\theta$ con respecto a este elemento y una distancia $\rho$, como se puede observar en la figura 4.4.

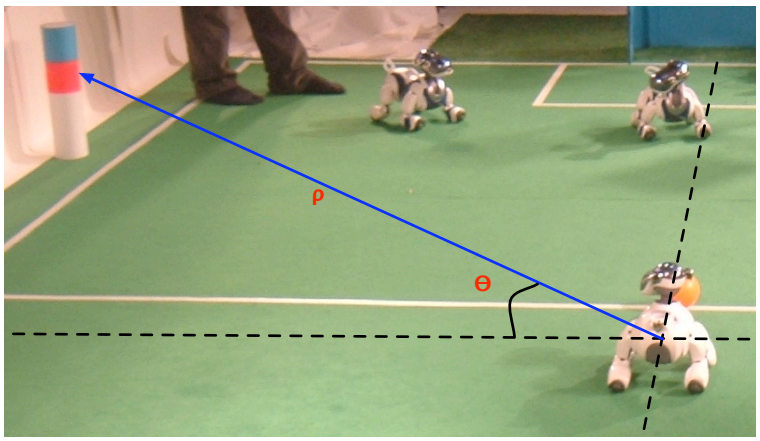


Figura 4.4: percepción de un elemento.

En esta figura 4.5 se muestra el proceso de detección de un elemento del entorno, en este caso, la pelota, aunque es similar para el resto de los elementos del entorno. De hecho, no usaremos la pelota como elemento relevante para la auto-localización, por ser muy dinámico. La pelota tiene un color naranja fácilmente identificable en el campo de juego. El proceso comienza con la toma de una imagen. La imagen se transforma al espacio de color HSV, al ser mas robusto frente a variaciones de iluminación.

Hay un reducido número de colores que pueden estar presentes en los elementos del entorno (verde, azul, amarillo y naranja). Estos colores se especifican mediante rangos en el espacio de color HSV usando la herramienta de calibración que incorpora la herramienta ChaosManager, descrita en la sección A.2. Una vez segmentada la imagen, los píxeles que pertenecen al mismo color se agrupan en rectángulos denominados *blobs*. Además, a partir del color verde de la moqueta, se calcula la línea del horizonte. En el siguiente paso se eliminan los *blobs* que, por sus dimensiones, densidad o distancia al horizonte, no cumplen con las características propias de ningún tipo de elemento. La baliza está formada por dos colores situados uno encima del otro. Por esta razón, al detectar este elemento también se tiene en cuenta la relación entre *blobs* y su posición relativa.

Tras filtrar la imagen y obtener un *blob* que cumple las características de una pelota, el robot manda los comandos necesarios al módulo *CMD* para mantenerla centrada en la imagen. Esto simplifica la extracción de la información de este elemento, ya que si está centrado, la orientación relativa del robot a la pelota es el ángulo de la articulación del cuello del robot respecto de su eje de coordenadas. La distancia a la pelota se obtiene a partir de las dimensiones del *blob*, o la estimación del radio de la pelota si ésta se percibe parcialmente en la imagen.

Dejando aparte los posibles falsos positivos que surgen a raíz de no haber podido eliminar los *blobs* de la imagen que no pertenecen a ningún elemento, la información $(\rho, \theta)$ extraída de la imagen puede presentar errores sistemáticos e imprecisiones. Los errores sistemáticos pueden provenir de problemas físicos en el robot, o de errores en el algoritmo que extrae el ángulo o la distancia a un elemento.

El modelo de percepción caracteriza el error y la precisión que presenta la información generada por el módulo *PAM* para poder usarla en el módulo *GM*. Es necesario realizar esta caracterización para corregir posibles errores sistemáticos en la información que genera este módulo. Asimismo, es necesario analizar la precisión de la información generada y el error máximo que podemos esperar en esta información.

La caracterización del modelo perceptivo para cada una de las dos marcas visuales utilizadas (portería y baliza) es diferente por varios motivos:

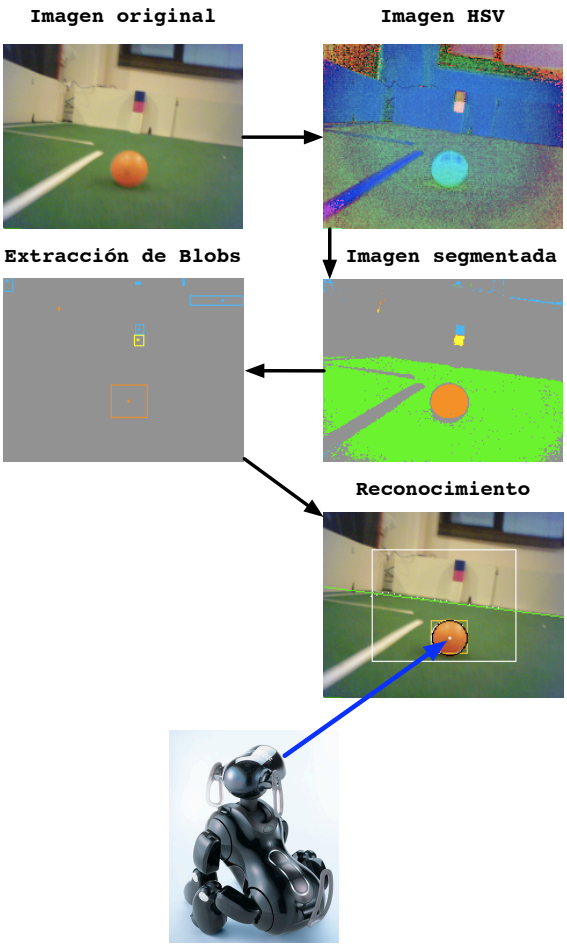


Figura 4.5: Proceso de extracción de información de una imagen en el *PAM*.

- Las dimensiones de ambos elementos no son similares. Como se puede observar en la figura 4.3, la portería es el elemento de mayor tamaño, lo que permite que se pueda observar desde cualquier punto del campo. En cambio, la baliza tiene un tamaño menor y no es detectable a gran distancia.

- La forma de la portería depende del lugar desde el que se percibe. No es lo mismo percibir la portería de frente que desde un lateral. La baliza, por el contrario, no varía su aspecto, presentando siempre la misma forma independientemente del punto desde el que se observe.

- La portería está formada por un solo color, mientras que la baliza por dos colores alineados uno debajo del otro. Debido a esto, la portería se confunde fácilmente con cualquier otro objeto presente en el entorno que tenga un color semejante. Para la baliza, en cambio, es más difícil que el sistema de percepción genere falsos positivos producidos por otros elementos del entorno (aunque se puede producir, como se mostró en la figura 1.17).

- Debido a la implementación del *PAM* y a la forma de la portería, ésta puede ser detectada parcialmente cuando el robot se encuentra cerca, haciendo que la información de distancia sea errónea.

Como comentamos anteriormente, el modelo de percepción caracteriza el error que presenta la información generada por el módulo *PAM*. El error es la diferencia entre la información perceptiva que calcula el módulo *PAM* y la real, calculada por un sistema externo. Este error se caracteriza con la media $\mu$ y la desviación típica $\sigma$.

Si la media $\mu$ es diferente de 0 el sistema tiene un error sistemático que debe ser corregido. Por ejemplo, a una distancia de 1300 mm. el robot podría detectar usualmente la portería 200 mm. más lejos de lo que está en realidad. Esto se podría deber a una calibración errónea del módulo encargado de la estimación de la distancia, o un error de implementación. Si se comprueba que esto se produce, se podría corregir la medida restándole 200 mm. a la medida obtenida a esa distancia.

El error máximo que podemos esperar en la información perceptiva se puede averiguar tras analizar la desviación típica $\sigma$ del error en el cálculo de la percepción. Si $\sigma$ es pequeña, el sensor se considera preciso, ya que su error máximo esperado es bajo. Siguiendo con el ejemplo anterior, se podría comprobar que cuando el robot se encuentra a 1300 mm de la portería, la desviación típica del error en la estimación de la distancia, $\sigma_\rho$, es 500 mm. Esto indica que es posible que se obtengan valores de distancia entre 800 mm. y 1800 mm. Si $\sigma_\rho$ fuese de 100 mm, los valores de distancia estarían en un rango de 1200 y 1400, siendo esta información mucho más fiable y precisa.

Los modelos perceptivos se caracterizan teniendo en cuenta la distancia a la que se encuentra el robot del objeto que se está midiendo. No es lo mismo el error que

puede existir al percibir un elemento que está a 1 metro que el que se produce al percibir un elemento que está a 5 metros.

Para ambos elementos se analiza la información perceptiva en diferentes condiciones con el objetivo de que el modelo sea lo más completo y exacto posible. El robot se coloca a distintas distancias con diferentes orientaciones, y se realiza tanto con el robot parado como con el robot moviendo sus patas en el sitio sin desplazarse. De este modo se puede tener en cuenta cómo afecta el movimiento de la cámara situada en la cabeza del robot a la percepción. En cada posición, el robot busca todos los elementos del entorno. Cuando encuentra uno lo centra en la imagen durante menos de un segundo, lo procesa (obtiene $z$) y vuelve a buscar otro elemento.

### 4.1.1. Modelo perceptivo de la portería

La obtención del modelo perceptivo de la portería se obtiene a partir de percepciones reales de este elemento. El robot se coloca manualmente en posiciones determinadas del campo de juego, y almacena las medidas $(\rho, \theta)$ que obtiene de la portería durante 40 segundos. El conjunto de posiciones ha de ser suficientemente variado para que el modelo sea representativo y fiable. En la figura 4.6 se muestra el conjunto de posiciones y orientaciones en la que se ha colocado al robot en este análisis. Las posiciones seleccionadas corresponden a aquellas situadas enfrente de la portería, partiendo de un distancia de 500 mm. hasta llegar a 4000 mm. con incrementos de 500 mm. En cada posición, se coloca al robot 0, 45 y 90 con respecto

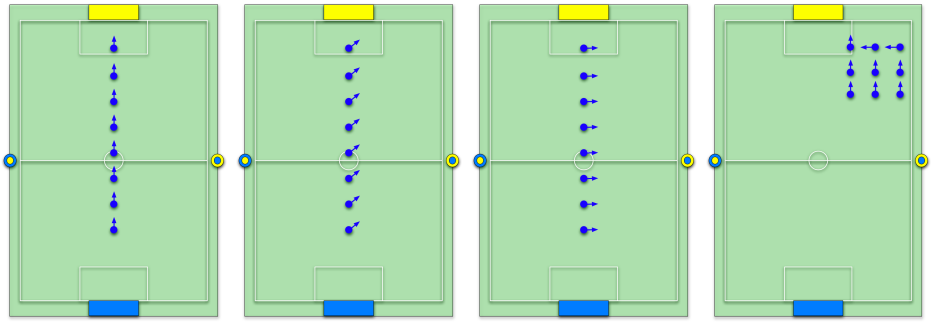

Figura 4.6: Posiciones y orientaciones del robot para obtener el modelo de percepción de la portería.

a la portería, por si esto pudiera afectar a la percepción. Además, se ha situado un segundo conjunto de puntos en posiciones laterales (gráfico situado a la derecha en la figura 4.6). El objetivo de este segundo conjunto es cubrir los casos en que la portería presenta un aspecto diferente del que tiene cuando se percibe frontalmente. Creemos que el conjunto total de puntos seleccionados permite obtener un modelo perceptivo razonablemente completo de este elemento.

Usando las herramientas que se han desarrollado para almacenar las observaciones del robot, descritas en el apéndice B, se puede automatizar el proceso de toma de observaciones. De esta manera se puede conseguir un gran número de ellas desde cada posición.

Tras comparar las medidas perceptivas obtenidas con la distancia y la orientación reales del robot al elemento medido, se puede obtener el error y la precisión del sistema, y así construir el modelo perceptivo ajustado.

Los resultados del análisis se muestran en la figura 4.7, donde se representa en rojo cómo evoluciona la media de la diferencia entre la medida calculada por el módulo *PAM* y la real según se aleja al robot de la portería, $\mu_\rho$. Las líneas azules gruesas muestran la desviación típica $\sigma_\rho$ y las azules finas los valores máximos y mínimos.

Los valores numéricos representados en esta gráfica se muestran en las tablas 4.1 y 4.2. En esta tabla, la primera columna indica el rango de distancia en la que los valores de la fila son válidos. La segunda columna muestra el número de muestras que se han tomado en el análisis del error en este rango. Las demás columnas muestran $\mu_\rho$, $\sigma_\rho$, el valor mínimo y máximo del error observados en las muestras.

Un análisis de estos datos muestra que la distancia que calcula el módulo de visión es bastante fiable, aumentando conforme el robot se va alejando de la portería. Por ejemplo, cuando el robot se encuentra entre 1000 y 1500 mm., $\mu_\rho$ es $-50{,}45$ mm., lo que es bastante poco para estar a tanta distancia. Además, el error máximo que se puede esperar en este dato es menor a 16 centímetros.

En el caso del ángulo, se observa que sucede al revés, siendo $\mu_\theta$ mayor error en distancias cortas. Esto se debe a que en dichas posiciones el robot no ve la portería completa. No se debe olvidar la información que se obtiene al percibir la portería es el centro de la misma.

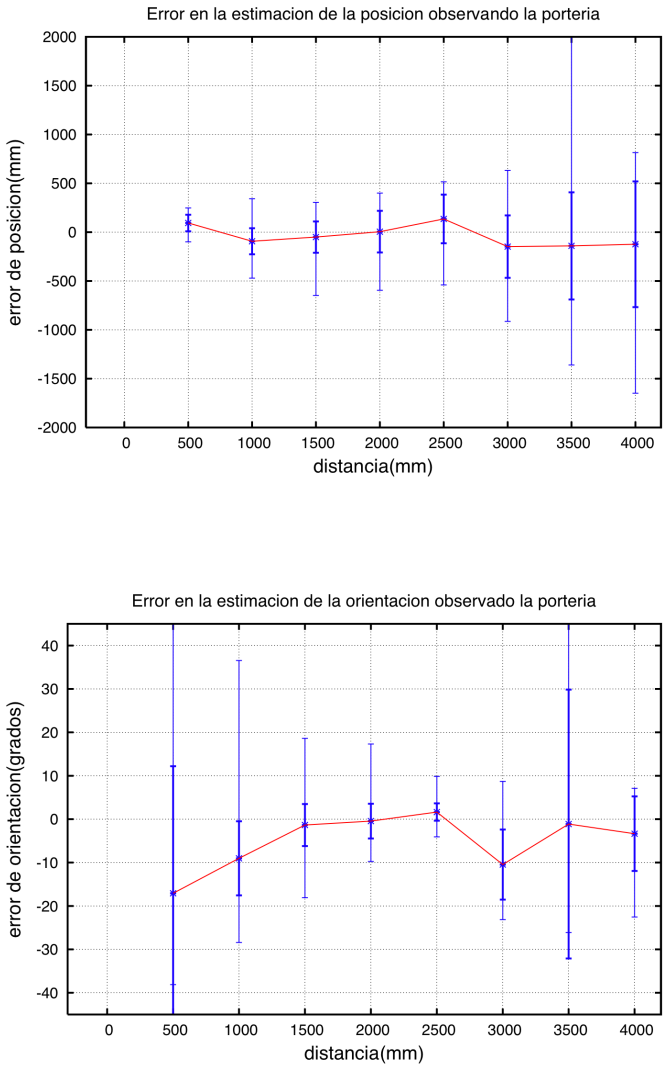


Figura 4.7: Resultados del error en estimación de la distancia (izquierda) y orientación (derecha) a la portería.

### 4.1.2. Modelo perceptivo de la baliza

Para obtener el modelo de percepción de la baliza se realiza un experimento similar al caso de la portería. El robot se coloca en las posiciones y orientaciones mostradas en la figura 4.8. No es necesario colocar al robot por todo el campo ya

| rango (mm) | muestras | $\mu_\rho$ (mm) | $\sigma_\rho$ (mm) | min (mm) | max (mm) |
|---|---|---|---|---|---|
| 0-500 | 220 | 93.76 | 85.076 | -99.10 | 248.0 |
| 500-1000 | 218 | -93.49 | 133.45 | -470.21 | 341.966 |
| 1000-1500 | 611 | -50.45 | 160.05 | -647.77 | 305.0 |
| 1500-2000 | 632 | 5.04 | 213.29 | -595.32 | 400.0 |
| 2000-2500 | 372 | 135.31 | 249.40 | -540.0 | 516.0 |
| 2500-3000 | 166 | -148.18 | 319.58 | -914.0 | 632.0 |
| 3000-3500 | 148 | -140.53 | 548.58 | -1360.0 | 2000.0 |
| 3500-4000 | 125 | -123.74 | 643.44 | -1649.0 | 815.0 |

Tabla 4.1: Resumen estadístico del error en la estimación de la distancia a la portería.

| rango (mm) | muestras | $\mu_\theta$ (grados) | $\sigma_\theta$ (grados) | min (grados) | max (grados) |
|---|---|---|---|---|---|
| 0-500 | 220 | -17.08 | 29.28 | -38.12 | 45.53 |
| 500-1000 | 218 | -9.024 | 8.54 | -28.41 | 36.54 |
| 1000-1500 | 611 | -1.34 | 4.83 | -18.08 | 18.6 |
| 1500-2000 | 632 | -0.44 | 3.99 | -9.74 | 17.32 |
| 2000-2500 | 372 | 1.63 | 2.00 | -4.08 | 9.86 |
| 2500-3000 | 166 | -10.45 | 8.06 | -23.14 | 8.68 |
| 3000-3500 | 148 | -1.124 | 30.96 | -26.12 | 197.2 |
| 3500-4000 | 125 | -3.34 | 8.59 | -22.54 | 7.09 |

Tabla 4.2: Resumen estadístico del error en la estimación de la orientación a la portería.

que, al contrario que en el caso de la portería, la baliza presenta el mismo aspecto desde cualquier posición. Asimismo, sólo se realiza el análisis con una de las balizas azul-amarilla, al tener similares características a la otra baliza. Al igual que en el caso de la portería, en cada posición se toman observaciones durante 40 segundos y se comparan con las medidas reales.

Los resultados demuestran que la información de distancia para las balizas es similar que en el caso de la portería en distancias cortas. Esta fiabilidad aumenta considerablemente conforme el robot se va alejando de la baliza, como se muestra en la gráfica superior de la figura 4.9. La gráfica inferior de esta misma figura y la tabla 4.3 muestran el caso del ángulo a la baliza. Se observa que esta información es bastante precisa, al ser un elemento pequeño percibido entero en la mayor parte de los casos.

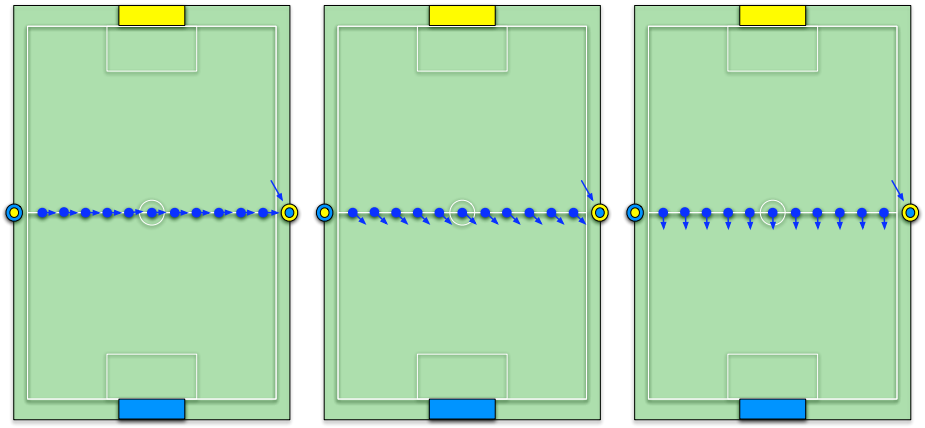

Figura 4.8: Posiciones y orientaciones del robot para obtener el modelo de percepción de la baliza.

La figura 4.9 muestra los resultados obtenidos al estimar tanto la distancia como la orientación a la baliza. Dado que este error varía dependiendo de la distancia a la baliza, la figura muestra cómo evoluciona el error según se aumenta la distancia real entre el robot y la baliza. En rojo muestra la $\mu_\rho$, y en azul oscuro se muestra $\sigma_\rho$ y los valores máximos y mínimos. Los datos en formato numérico representados en la figura se presenta en las tablas 4.3 y 4.4.

| rango (mm) | muestras | $\mu_\rho$ (mm) | $\sigma_\rho$ (mm) | min (mm) | max (mm) |
|---|---|---|---|---|---|
| 0-500 | 522 | 112.05 | 30.77 | 50.0 | 150.0 |
| 500-1000 | 873 | 98.6 | 74.44 | -174.0 | 220.0 |
| 1000-1500 | 1183 | 71.84 | 167.29 | -529.0 | 358.0 |
| 1500-2000 | 1181 | 119.43 | 229.67 | -826.0 | 409.0 |
| 2000-2500 | 899 | 342.06 | 407.77 | -1060.0 | 864.0 |
| 2500-3000 | 1225 | 244.78 | 343.4 | -1135.0 | 1223.0 |
| 3000-3500 | 732 | 507.45 | 328.5 | -631.0 | 2050.0 |

Tabla 4.3: Resumen estadístico del error en la estimación de la distancia a la baliza.

La información obtenida en la caracterización del error a cada elemento se almacena en el robot. Antes de incorporar la información visual al sistema de autolocalización se modifican aplicando el modelo perceptivo, como se representa en la figura 4.2, anteriormente descrito.

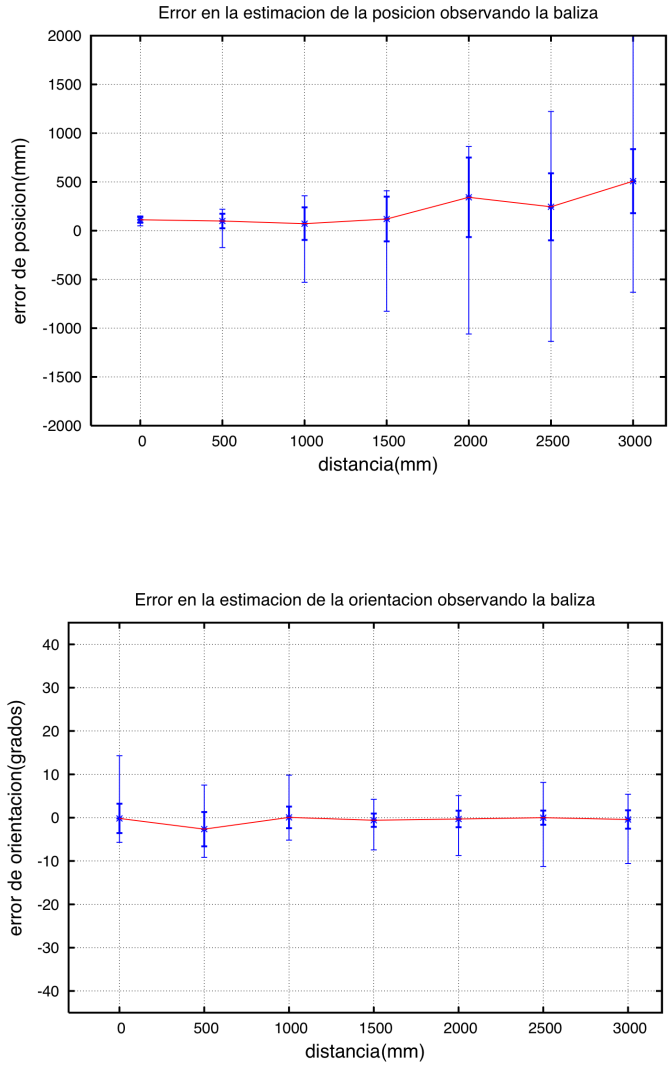


Figura 4.9: Resultados del error en estimación de la distancia (arriba) y orientación (abajo) a la baliza.

| rango (mm) | muestras | $\mu_\theta$ (grados) | $\sigma_\theta$ (grados) | min (grados) | max (grados) |
|---|---|---|---|---|---|
| 0-500 | 522 | -0.17 | 3.38 | -5.68 | 14.3 |
| 500-1000 | 873 | -2.65 | 3.95 | -9.15 | 7.52 |
| 1000-1500 | 1183 | 0.06 | 2.49 | -5.17 | 9.82 |
| 1500-2000 | 1181 | -0.59 | 1.53 | -7.43 | 4.23 |
| 2000-2500 | 899 | -0.3 | 1.9 | -8.74 | 5.09 |
| 2500-3000 | 1225 | -0.003 | 1.63 | -11.27 | 8.13 |
| 3000-3500 | 732 | -0.42 | 2.12 | -10.58 | 5.41 |

Tabla 4.4: Resumen estadístico del error en la estimación de la orientación a la baliza.

El proceso de corrección de la información perceptiva usando este modelo es el siguiente: En el instante $t$ el módulo *PAM* genera una percepción $z_t$ contenida en la estructura `lps`. Al aplicarle el modelo de percepción, se calcula una percepción corregida $z'_t$ que, además, contiene la incertidumbre $R_t$ en la percepción. Esta percepción $z'_t$ forma parte de una nueva estructura `lps'` que recibe el módulo *GM*.

La corrección realizada sobre $z_t$ se hace en función del tipo de elemento y la distancia a la que $z_t$ indica que se ha percibido el elemento. Por ejemplo, supongamos que el robot obtiene una observación $z_t$ de una baliza que indica que se encuentra situada a $2213$ mm. con una orientación de $34$ grados respecto del robot. Antes de que el módulo *GM* reciba esta información, se modifica aplicándole el modelo de percepción, como se mostró en la figura 4.2. Como la distancia a la baliza es $2213$ mm, se obtiene de las tablas 4.3 y 4.4 $\mu$ y $\sigma$ en el rango $2000 - 2500$ mm correspondiente al tipo baliza. Con estos valores se modifican los datos de entradas como se muestra en la 4.1.

$$z' = z - correccion = \begin{pmatrix} 2213 \\ 34 \end{pmatrix} - \begin{pmatrix} 342{,}06 \\ -0{,}3 \end{pmatrix} = \begin{pmatrix} 1866{,}94 \\ 34{,}3 \end{pmatrix} \tag{4.1}$$

$$R_t = \begin{pmatrix} 407{,}77^2 & 0 \\ 0 & 1{,}9^2 \end{pmatrix} \tag{4.2}$$

## 4.2. Modelo de actuación

El robot se desplaza por el campo de juego haciendo uso de sus patas. Puede desplazarse hacia adelante o retroceder, lateralmente a ambos lados y girar en ambos sentidos respecto de un eje vertical que pasa por su centro. Normalmente el movimiento del robot corresponde a una combinación de estos 2 tipos de desplazamientos y una componente de giro. Además, cada uno de estos movimientos puede realizarse a diferentes velocidades.

Las velocidades a las que el robot realiza cada movimiento las establece el módulo *CTRL*, como resultado de la ejecución de los comportamientos básicos que guían el comportamiento general del robot. Este módulo genera comandos de movimiento que se componen de velocidad lineal, lateral y de rotación. Los comandos de movimiento se envían periódicamente (uno en cada ciclo de control) al módulo *CMD*, encargado del movimiento del robot.

Cada vez que el módulo *CMD* recibe un comando de movimiento, ajusta la coordinación entre las patas del robot para satisfacer este comando. El *CMD* calcula constantemente la posición de cada una de las 12 articulaciones de las patas. Cada 8 ms. el robot puede adoptar una combinación de posiciones distinta.

Al final de cada ciclo de control, el módulo *CMD* genera un conjunto de valores que representan el desplazamiento que ha realizado el robot en este ciclo de control. Estos valores son el desplazamiento lineal, lateral y de rotación que se ha llevado a cabo en este ciclo de control, y se calculan en función de la velocidad realizada y el tiempo transcurrido.
Esta información de odometría la usan los métodos de auto-localización desarrollados para actualizar el estado del robot. Es vital que esta información sea lo más precisa posible y que, además, se pueda saber el error máximo esperado en esta información.

El modelo de actuación del robot AIBO que vamos a describir en esta sección tiene en cuenta los comandos enviados al robot y la odometría calculada por el módulo *CMD*. Describiremos el error de esta odometría en función de los comandos.

El análisis de las medidas de odometría en desplazamientos se realiza en función de la velocidad, ya que pueden existir errores que dependan de ella. El procedimiento es establecer una serie de comandos de velocidad y comprobar cuál es la relación

entre la odometría que calcula el módulo de locomoción y el desplazamiento real que realiza el robot. Esta relación se representa como un factor de corrección que se usa para corregir la odometría calculada en función del comando de movimiento que se ha llevado a cabo. Además, se obtiene también un factor de incertidumbre que indica la precisión de esta odometría.

Este análisis se ha realizado en primer lugar para movimientos lineales y posteriormente para movimientos de rotación. El modelo de actuación para movimientos laterales no se ha caracterizado, ya que los comportamientos básicos existentes en el momento del análisis no hacían uso de ellos. De todas maneras, el sistema contempla la posibilidad de incorporar odometría resultante de movimientos laterales, para lo que habrá que modelar el error de este tipo de movimientos.

### 4.2.1. Análisis de movimientos lineales

El análisis de las medidas de odometría en desplazamientos lineales se ha realizado haciendo que el robot se desplace la misma distancia a diferentes velocidades. La velocidad mínima ha sido 50 mm/seg y la máxima 400 mm/seg. Hemos realizado el análisis cada 25 mm/seg.

Cada uno de los comandos de velocidad ha sido repetido 10 veces para evitar que posibles errores en el desplazamiento del robot, como pueden ser posibles deslizamientos, afecten a este análisis y los resultados no sean fiables. En cada una de las repeticiones el robot avanza $2{,}5$ metros.

En el análisis de las medidas de odometría en desplazamientos lineales se ha tenido en cuenta también cómo afecta la velocidad lineal a la rotación del robot. De esta manera se pueden obtener resultados que indiquen si el robot tiende a girar en algún sentido sistemáticamente al avanzar. Esto se puede producir por imprecisiones en la generación del movimiento o por asimetrías en la construcción del robot. La batería del robot, que es un elemento con un peso no despreciable, no se encuentra en el centro del robot, sino en su lado izquierdo. Esto desplaza el centro de masas del robot y podría hacer que el robot tendiera a girar hacia un lado.

Tras este análisis, para cada una de las velocidades lineales comandadas se obtiene la siguiente información:

| Velocidad | $f_l$ | $f_{pl}$ | $f_r$ | $f_{pr}$ |
|---|---|---|---|---|
| 50.0 | 0.850 | 0.197 | 0.000188 | 0.001 |
| 75.0 | 0.748 | 0.154 | 0.000231 | 0.000364 |
| 100.0 | 0.838 | 0.320 | -0.001 | 0.009 |
| 125.0 | 0.914 | 0.587 | 0.0000883 | 0.000286 |
| 150.0 | 0.868 | 0.162 | 0.001 | 0.009 |
| 175.0 | 1.012 | 0.765 | -0.00004001 | 0.000671 |
| 200.0 | 0.896 | 0.248 | 0.0000433 | 0.000564 |
| 225.0 | 0.884 | 0.150 | 0.0000369 | 0.000245 |
| 250.0 | 0.874 | 0.148 | 0.000132 | 0.000249 |
| 275.0 | 1.045 | 1.160 | 0.000158 | 0.000267 |
| 300.0 | 0.920 | 0.406 | -0.0000127 | 0.000533 |
| 325.0 | 0.873 | 0.266 | -0.0000403 | 0.000294 |
| 350.0 | 0.993 | 0.760 | -0.000338 | 0.001 |
| 375.0 | 0.968 | 0.536 | -0.000333 | 0.000384 |
| 400.0 | 0.919 | 0.308 | -0.000485 | 0.000474 |

Tabla 4.5: Resultados del análisis de la velocidad lineal.

- Un factor de corrección lineal $f_l$. Este factor es el cociente entre el desplazamiento real y la odometría lineal calculada por el módulo de locomoción. En cada una de las 10 repeticiones del análisis para cada velocidad se obtiene un factor parcial de corrección lineal. El valor de $f_l$ definitivo será la media de estos factores parciales.

- Un factor de incertidumbre lineal $f_{pl}$. Este factor indica la precisión de la odometría al aplicarle el factor de corrección $f_l$. Esta precisión se aproxima como la desviación típica de los 10 factores parciales de corrección lineal.

- Un factor de corrección rotacional $f_r$. La orientación del robot puede verse afectada por diversos factores, como hemos comentado anteriormente. Este factor representa cómo afecta la velocidad lineal a la rotación del robot. Se calcula de una manera similar a $f_l$, pero esta vez siendo el cociente entre la rotación real del robot y la odometría lineal obtenida del módulo de locomoción.

- Un factor de incertidumbre rotacional $f_{pr}$. Este factor indica la precisión de la odometría rotacional calculada después de aplicarle el factor de corrección $f_r$.

Esta precisión se calcula como la desviación típica de los 10 factores parciales de corrección rotacional.

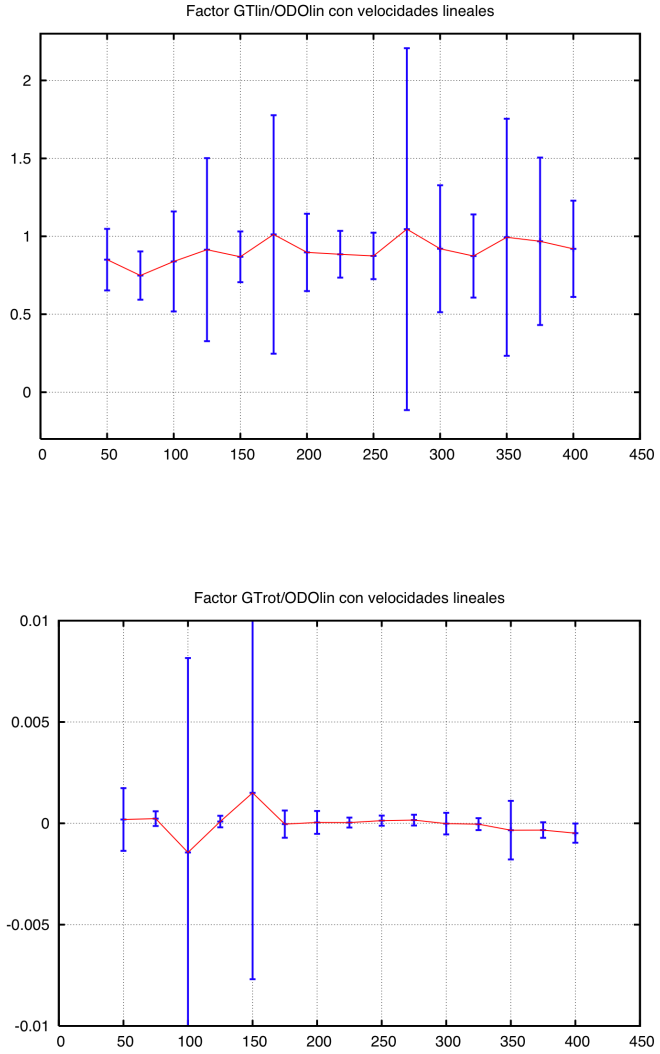


Figura 4.10: Factor de corrección de la velocidad lineal.

Tras realizar el análisis completo con el robot real, los resultados obtenidos se muestran en la tabla 4.5. Estos resultados permiten aportar varias conclusiones. La

primera de ellas es que el desplazamiento lineal real es, para la mayor parte de las velocidades, ligeramente menor que la odometría calculada por el módulo de locomoción, ya que $f_l$ es menor que 1. Además, la odometría presenta una elevada incertidumbre para algunas velocidades. La figura 4.10 muestra la representación gráfica de esta tabla. La línea roja es el valor de $f_l$ y las azules representan $f_{pl}$. El eje de abscisas representa las distintas velocidades lineales y el de ordenadas el factor de corrección. Cuanto más cortas sean las líneas azules, más precisa es la odometría corregida. Hay velocidades, sobre todo la de 275 mm/seg que presentan una elevada incertidumbre.

En cuanto a cómo afecta la velocidad lineal a la de rotación, se observa que esto se produce sobre todo en velocidades entre 100 y 200 mm/seg. Siendo mínimo el impacto en las otras velocidades.

### 4.2.2. Análisis de rotaciones

El análisis de las medidas de odometría en giros se ha realizado haciendo que el robot gire sobre sí mismo estableciendo diferentes velocidades de rotación, en un sentido y en otro (velocidades positivas para giros hacia la izquierda y negativas hacia la derecha). La velocidad mínima es de 15 grados/seg y la máxima es de 100 grados/seg. Hemos realizado el análisis cada 5 grados/seg.

Cada uno de los comandos de velocidad rotacional ha sido repetido 10 veces y en cada una de las repeticiones el robot gira durante 30 segundos.

En el análisis de las medidas de odometría en rotaciones no se ha tenido en cuenta el desplazamiento lineal, ya que resulta extremadamente complejo medirlo y, tras observar el movimiento de robot, podemos decir que es inapreciable.

Tras este análisis, para cada una de las velocidades de rotación comandadas se obtiene la siguiente información:

- Un factor de corrección rotacional $e$. Este factor es el cociente entre el giro real y la odometría rotacional calculada por el módulo de locomoción. En cada una de las 10 repeticiones del análisis para cada velocidad se obtiene un factor parcial de corrección rotacional. El valor de $e$ definitivo será la media de estos factores parciales.

- Un factor de incertidumbre rotacional $e_p$. Este factor indica la precisión de la odometría después de aplicarle el factor de corrección $e$. Esta precisión se calcula como la desviación típica de los 10 factores parciales de corrección rotacional.

Tras realizar el análisis completo con el robot real, los resultados obtenidos se muestran en la tabla 4.6. La figura 4.11 muestra la representación gráfica de esta tabla. La línea roja es el valor de $e$ y las azules representan $e_p$. El eje de abscisas representa las distintas velocidades de rotación y el de ordenadas el factor de corrección. Cuanto más cortas sean las líneas azules, más precisa es la odometría corregida.

La principal conclusión es que, para los giros, la odometría calculada por el módulo de locomoción es muy similar a la obtenida en la experimentación con el robot real. Asimismo, salvo en el caso aislado del comando de $-50$ grados/seg, tiene una incertidumbre baja.

| Velocidad | $e$ | $e_p$ | Velocidad | $e$ | $e_p$ |
|---|---|---|---|---|---|
| 15.0 | 0.996 | 0.117 | -15.0 | 1.015 | 0.186 |
| 20.0 | 0.998 | 0.123 | -20.0 | 1.022 | 0.123 |
| 25.0 | 1.00 | 0.105 | -25.0 | 1.004 | 0.137 |
| 30.0 | 1.014 | 0.129 | -30.0 | 1.023 | 0.148 |
| 35.0 | 1.001 | 0.164 | -35.0 | 0.998 | 0.167 |
| 40.0 | 0.994 | 0.141 | -40.0 | 1.007 | 0.151 |
| 45.0 | 0.998 | 0.1580 | -45.0 | 0.998 | 0.169 |
| 50.0 | 1.007 | 0.1825 | -50.0 | 1.043 | 0.605 |
| 55.0 | 1.000 | 0.1866 | -55.0 | 0.977 | 0.180 |
| 60.0 | 1.005 | 0.296 | -60.0 | 0.963 | 0.174 |
| 65.0 | 1.009 | 0.180 | -65.0 | 0.984 | 0.167 |
| 70.0 | 0.993 | 0.196 | -70.0 | 0.971 | 0.176 |
| 75.0 | 0.996 | 0.219 | -75.0 | 0.983 | 0.220 |
| 80.0 | 1.004 | 0.321 | -80.0 | 0.997 | 0.261 |
| 85.0 | 1.005 | 0.254 | -85.0 | 1.003 | 0.222 |
| 90.0 | 0.984 | 0.199 | -90.0 | 0.999 | 0.211 |
| 95.0 | 0.978 | 0.187 | -95.0 | 1.014 | 0.224 |
| 100.0 | 0.981 | 0.210 | -100.0 | 1.008 | 0.176 |

Tabla 4.6: Resumen estadístico del cálculo del factor de corrección de la velocidad rotacional.

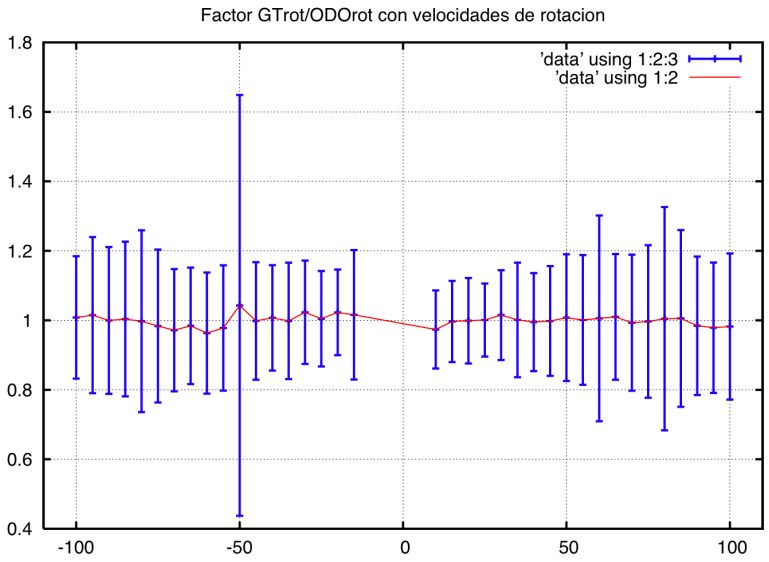


Figura 4.11: Factor de corrección de la velocidad rotacional.

Veamos un ejemplo de cómo se utiliza este modelo de actuación para corregir y completar la odometría que procede del módulo *CMD* y va hacia el módulo *GM*, como se mostró en la figura 4.2 . En un determinado momento el módulo *CTRL* manda al módulo *CMD* un comando de movimiento que especifica que su velocidad lineal es $100$ mm/seg y su velocidad de rotación es $50$ grados/segundo. La coordinación de las patas se ajusta para conseguir esta velocidad. En el siguiente ciclo de ejecución, que dura $33$ ms. la odometría $u_t$ que genera el módulo *CMD* indica que el robot se ha desplazado $3{,}3$ mm, y ha rotado $2{,}5$ grados.

$$u_t = \begin{pmatrix} u_{lin} \\ u_{lat} \\ u_{rot} \end{pmatrix} = \begin{pmatrix} 3{,}3 \\ 0 \\ 1{,}65 \end{pmatrix}$$

El modelo de actuación corrige estas medidas, obteniendo unos valores de odometría $u'_t$ y añadiendo a esta información la incertidumbre $Q_t$ que existe en este desplazamiento. Debido a que los valores de corrección e incertidumbre son distintos

dependiendo de la velocidad, se obtienen de las tablas 4.5 y 4.6 los valores correspondientes a la velocidad lineal de $100$ mm/seg y a la velocidad de rotación de $50$ grados/segundo.

$$u'_t = \begin{pmatrix} u_{lin,t} \cdot f_l^{100} \\ 0 \\ u_{lin,t} \cdot f_r^{100} + u_{rot,t} \cdot e^{50} \end{pmatrix} = \begin{pmatrix} 3{,}3 \cdot 0{,}838 \\ 0 \\ 3{,}3 \cdot -0{,}001 + 1{,}65 \cdot 1{,}007 \end{pmatrix} =$$

$$= \begin{pmatrix} 2{,}7654 \\ 0 \\ 1{,}65825 \end{pmatrix}$$

$$Q_t = \begin{pmatrix} (3{,}3 \cdot 0{,}320)^2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & (3{,}3 \cdot 0{,}009 + 1{,}65 \cdot 0{,}1825)^2 \end{pmatrix} =$$

$$= \begin{pmatrix} (1{,}056)^2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & (0{,}330825)^2 \end{pmatrix}$$

## 4.3. Filtro extendido de Kalman

Una vez caracterizados los modelos de percepción y de actuación, en esta sección se muestra cómo se ha diseñado el *EKF* usado en este trabajo. Este EKF integra la información perceptiva y de actuación en una estimación de posición y orientación, con su incertidumbre asociada.

Los algoritmos de auto-localización de un robot móvil que usan *EKF* son bien conocidos, como mostramos en la sección 2.1. En el presente capítulo se definen los elementos de un *EKF* que son específicos de este problema.

En primer lugar se define qué entendemos por estado del robot. Supondremos que el robot del que se desea conocer su posición se mueve en un plano y puede tener cualquier orientación. Por lo tanto, la posición del robot se define como el vector de estado $s \in \Re^3$:

$$\mathbf{s} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{4.3}$$

Asimismo, este vector de estado tiene asociada una matriz de covarianza $P \in \Re^{3\times 3}$ que contiene la incertidumbre asociada al estado $s$ que se define como:

$$\mathbf{P} = \begin{pmatrix} \sigma_x^2 & \sigma_x\sigma_y & \sigma_x\sigma_\theta \\ \sigma_y\sigma_x & \sigma_y^2 & \sigma_y\sigma_\theta \\ \sigma_\theta\sigma_x & \sigma_\theta\sigma_y & \sigma_\theta^2 \end{pmatrix} \tag{4.4}$$

La estimación de la posición del robot utilizando *EKF* es un proceso recursivo que se divide en dos fases: predicción y corrección. En la primera fase, el movimiento que realiza el robot modifica el vector de estado $s$. En la segunda fase, el robot usa su cámara para obtener la posición de los elementos visuales del entorno, y así corregir la estimación de la posición realizada en la fase anterior.

En $t = 0$, el estado $s_0$ y su incertidumbre $P_0$ toman diferentes valores según el conocimiento que se tiene inicialmente de la posición del robot. Si la posición inicial es conocida, $s_0$ se inicializa en dicha posición. Si se conoce la incertidumbre de la posición $s_0$, se inicializa $P_0$ a dicho valor o a uno arbitrario que codifique la fiabilidad de esta información de posición inicial.

Si la posición inicial es desconocida, se inicializa $s_0$ con un valor arbitrario y $P_0$ a una incertidumbre tal que permita que $s$ converja a cualquier posición del campo de juego una vez obtenida una medida del entorno. En este caso es crítico que la primera medida del entorno que se incorpore al *EKF*, que reduce la incertidumbre, sea correcta. En caso de una medida incorrecta o un falso positivo, la estimación será errónea y difícilmente convergerá a la posición correcta. Esta es la razón de por qué un *EKF* no debe iniciarse a una posición desconocida, sobre todo en escenarios con mucho ruido sensorial.

En la fase de predicción se calcula la posición del robot $s_t^-$ tras haber realizado un desplazamiento $u_{t-1}$ desde la posición $s_{t-1}$. En la figura 4.12 se representa gráficamente la dinámica del robot. En esta figura los vectores unitarios $\hat{X}_a$ y $\hat{Y}_a$ definen el eje de referencia global. Inicialmente el robot se encuentra en la posición

$s_{t-1} = (X_{t-1}, Y_{t-1}, \theta_{t-1})$, y su eje de referencia local al robot se define por los vectores unitarios $\hat{X}_{t-1}$ y $\hat{Y}_{t-1}$. Tras incorporar la odometría representada por la entrada de control $u_{t-1}$, la posición del robot $s_t^- = (X_t, Y_t, \theta_t)$, siendo su eje del referencias local el formado por los vectores unitarios $\hat{X}_t$ y $\hat{Y}_t$.

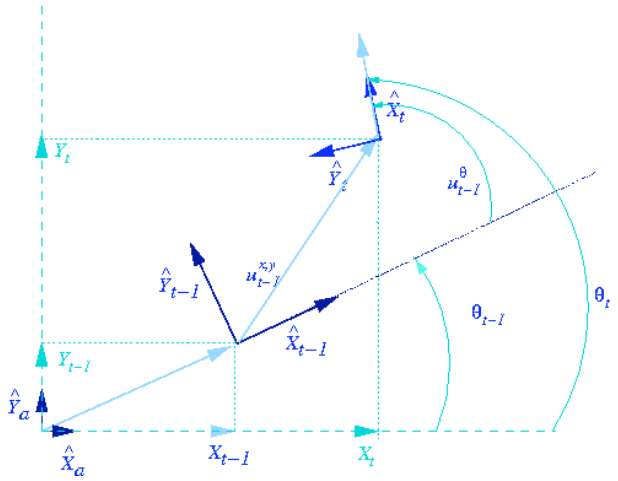


Figura 4.12: Dinámica del robot.

En la fase de predicción, al tratarse de un sistema no lineal, el cálculo del estado del robot $s_t^-$ se realiza mediante la función no lineal $f$ que predice el estado *a priori* $s_t^-$ a partir del estado anterior $s_{t-1}$, la odometría $u_{t-1}$ y el ruido $w_{t-1}$ que proviene de la incorporación de esta odometría. Cada componente de $s_t$ se calcula independientemente usando la descomposición de $f$ mostrada en las ecuaciones 4.6-4.8.

$$s_t^- = f(s_{t-1}, u_{t-1}, w_{t-1}) \tag{4.5}$$

$$x_t = x_{t-1} + (u_{t-1}^x + w_{t-1}^x)cos\theta_{t-1} - (u_{t-1}^y + w_{t-1}^y)sen\theta_{t-1} \tag{4.6}$$

$$y_t = y_{t-1} + (u_{t-1}^x + w_{t-1}^x)sen\theta_{t-1} + (u_{t-1}^y + w_{t-1}^y)cos\theta_{t-1} \tag{4.7}$$

$$\theta_t = \theta_{t-1} + u_{t-1}^\theta + w_{t-1}^\theta \tag{4.8}$$

Para aplicar el Filtro Extendido de Kalman en la predicción de $s_t^-$, en primer lugar se aplica la función $f$ sin el ruido $w_{t-1}$, como muestra la ecuación 4.9.

$$s_t^- = \begin{pmatrix} x_t^- \\ y_t^- \\ \theta_t^- \end{pmatrix} = f(s_{t-1}, u_{t-1}, 0) =$$

$$\begin{pmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{pmatrix} + \begin{pmatrix} u_{t-1}^x cos\theta_{t-1} - u_{t-1}^y sen\theta_{t-1} \\ u_{t-1}^x sen\theta_{t-1} + u_{t-1}^y cos\theta_{t-1} \\ u_{t-1}^\theta \end{pmatrix} \tag{4.9}$$

El ruido $w_{t-1}$ se usa en el cálculo de la covarianza del error $P_t^-$, que representa la incertidumbre sobre la posición del robot, como muestra la ecuación 4.10. Esta ecuación está formada por dos componentes, representadas gráficamente en la figura 4.13. La primera componente, $A_t P_{t-1} A_t^T$, corresponde a la incorporación a $P_t^-$ de la incertidumbre almacenada en la matriz $P_{t-1}$ antes de realizar el movimiento. A esta incertidumbre se le aplica la matriz de transformación $A_t$ desde el eje de referencia global al eje de referencia local del robot. El motivo de multiplicar $P_{t-1}$ por $A_t$ y después por su traspuesta es el de combinar $P_{t-1}$, que está en el eje de referencia global, con la odometría $u_{t-1}$, que se encuentra en el eje de referencia del robot en $t-1$.

La segunda componente es $W_t Q_{t-1} W_t^T$. Esta componente corresponde a la incertidumbre generada en el desplazamiento, y que incrementa la ya existente en $P_{t-1}$. La incertidumbre generada en el desplazamiento es $Q_{t-1} = E[w_{t-1} w_{t-1}^T]$. Esta incertidumbre depende de la velocidad, y el modelo de actuación que define su valor en cada iteración se ha presentado en la sección 4.2. Las matriz de transformación $W_t$ permite trasladar la incertidumbre generada en el eje del robot al eje de referencia global.

$$P_t^- = A_t P_{t-1} A_t^T + W_t Q_{t-1} W_t^T \tag{4.10}$$

$$A_t = \frac{\partial f}{\partial s} = \begin{pmatrix} 1 & 0 & -u_{t-1}^y cos\theta_{t-1} - u_{t-1}^x sen\theta_{t-1} \\ 0 & 1 & u_{t-1}^x cos\theta_{t-1} - u_{t-1}^y sen\theta_{t-1} \\ 0 & 0 & 1 \end{pmatrix} \tag{4.11}$$

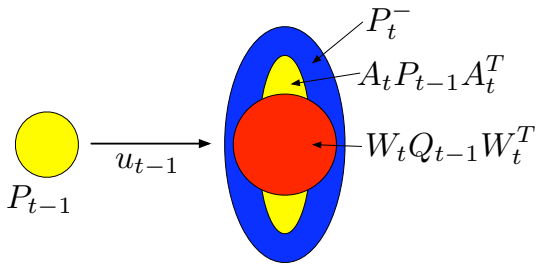


Figura 4.13: Evolución de $P_{t-1}$ a $P_t^-$

$$W_t = \frac{\partial f}{\partial w} = \begin{pmatrix} cos\theta_{t-1} & -sen\theta_{t-1} & 0 \\ sen\theta_{t-1} & cos\theta_{t-1} & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.12}$$

En la figura 4.14 se muestra cómo evoluciona el estado del robot y su incertidumbre al incorporar varias lecturas odométricas.

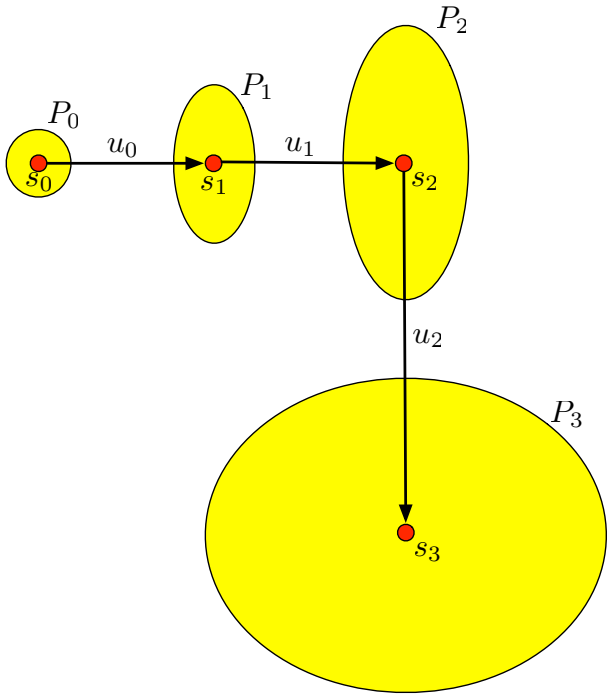


Figura 4.14: Evolución del estado y de su incertidumbre al incorporar la odometría.

En la fase de corrección se toman medidas reales de los elementos geométricos $m_{1\cdots n}$ existentes en el entorno, correspondientes a las balizas y a las porterías, para corregir la predicción $s_t^-$. Para cada observación $\hat{z}_t^i$ del elemento $m_i$ percibido se compara con la predicción de la observación $h_t^i$ para obtener la innovación $v_t^i$. La predicción de la observación $h_t^i$ se realiza en función de la predicción del estado $s_t^-$ y la posición conocida de $m_i$.

$$z_t^i = h_t^i(m_i, s_t^-) = \begin{pmatrix} distancia(m_i, s_t^-) \\ angulo(m_i, s_t^-) \end{pmatrix} = \\ = \begin{pmatrix} \sqrt{(m_t^x - s_t^{x,-})^2 + (m_t^y - s_t^{y,-})} \\ atan2(m_t^x - s_t^{x,-}, m_t^y - s_t^{y,-}) - s_t^{\theta,-} \end{pmatrix} \tag{4.13}$$

$$v_t^i = [\hat{z}_t^i - z_t^i] = [\hat{z}_t^i - h_t^i(m_i, s_t^-)] \tag{4.14}$$

No todas las observaciones $\hat{z}_t^i$ tomadas se incorporan. Durante un encuentro en la RoboCup hay multitud de casos en los que se producen falsos positivos en la percepción. Una forma de ser robusto a estas situaciones es aplicar un *filtro de incoherencias* para rechazar las observaciones que no son coherentes con la posición estimada del robot. Para esto, antes de incorporar una observación, se calcula un valor $\delta_t^i$ (ecuación 4.15). Este valor representa el grado de coherencia de una observación $\hat{z}_t^i$ con respecto al estado $s_t^-$. Este valor es útil para rechazar los casos de correspondencias equivocadas. Así pues, si este valor supera un *umbral de rechazo*, se rechaza la medida y no se incorpora al *EKF*.

$$\delta_t^i = (z_t^i - \hat{z}_t^i)^T (S_t^i)^{-1} (z_t^i - \hat{z}_t^i) \tag{4.15}$$

El *umbral de rechazo* se ha configurado heurísticamente como un valor dinámico entre $5$ y $100$. Esto se hace así debido a que cuando la posición estimada por *EKF* y la posición real del robot son diferentes y $P$ comienza a ser elevado, las percepciones correctas dejan de ser coherentes y $\delta_t^i$ es elevado. No hay muchas posibilidades de recuperación si el *umbral de rechazo* fuese estático y bajo, ya que no se incorporarían estas percepciones correctas. Por el contrario, cuando la posición del robot se

estima correctamente y $P$ se reduce, el valor de $\delta_t^i$ para las percepciones correctas es muy bajo. En ese momento, el *umbral de rechazo* debe ser bajo para dejar pasar las percepciones con menos error y rechazar las demás.

El *umbral de rechazo* se inicia en 20, y disminuye a la mitad con cada percepción aceptada por este mecanismo. Con cada percepción rechazada, el *umbral de rechazo* se incrementa en una vez y media su valor.

La *ganancia de Kalman* $K_t^i$, especifica el grado en que la medida $m_i$ se incorpora a la nueva estimación del estado. Este valor se obtiene de la siguiente forma :

$$K_t^i = P_{t-1}(H_t^i)^T(S_t^i)^{-1} \tag{4.16}$$

donde,

$$S_t^i = H_t^i P_{t-1}(H_t^i)^T + R_t^i \tag{4.17}$$

$$H_t^i = \frac{\partial h_t^i(m_i, s_t^-)}{\partial s_t} = \begin{pmatrix} -\frac{m_x^i - s_{t,x}^-}{\sqrt{q}} & -\frac{m_y^i - s_{t,y}^-}{\sqrt{q}} & 0 \\ \frac{m_y^i - s_{t,y}^-}{q} & -\frac{m_x^i - s_{t,y}^-}{q} & -1 \\ 0 & 0 & 0 \end{pmatrix} \tag{4.18}$$

$$q = (m_{t,x}^i - s_{t-1,x})^2 + (m_{t,y}^i - s_{t-1,y})^2$$

Con cada percepción $\hat{z}_t^i$, se calcula el nuevo estado del sistema $s_t$ la nueva covarianza del error en el sistema $P_t$ La covarianza del error en el sistema, $P_t$, se calcula usando la ecuación:

$$P_t = (I - K_t^i H_t^i)P_{t-1} \tag{4.19}$$

$$s_t = s_t^- + K_t^i v_t^i \tag{4.20}$$

Los elementos que forman parte del método de auto-localización basado en *EKF* ya se han definido en esta sección. En el capítulo 5 se analizarán sus resultados. En la siguiente sección se mostrará cómo se combinan uno o varios *EKFs* con un método global.

## 4.4. Combinaciones de *EKF* con un método global

Al inicio de este capítulo se expusieron los problemas que se presentaban al usar exclusivamente un único *EKF* para la auto-localización de un robot en el entorno de la RoboCup. Básicamente, estos problemas están relacionados con la incapacidad del *EKF* para localizar al robot partiendo de una posición desconocida o cuando el robot es desplazado manualmente de un lugar a otro del entorno, y con la evaluación de los *EKFs*.

Además, se persigue solucionar el problema del alto consumo computacional que necesitaba la implementación original de *FMK*. Como comentamos al inicio de este capítulo, la sincronización de los diferentes módulos (percepción, generación de movimiento, localización, coordinación, etc.) que se ejecutan en el robot es crítica. La suma de los tiempos de cada uno no debe exceder del tiempo máximo de ejecución, que corresponde al tiempo entre que el robot obtiene dos imágenes consecutivas (en media cada $33,810$ ms.). Para complicarlo más, el tiempo de cómputo necesario para cada módulo no es fijo, pudiendo variar entre un ciclo y otro, con lo que es conveniente no apurar completamente el tiempo de ciclo y dejar un pequeño margen. En la figura 4.15 se observa el porcentaje de consumo de tiempo de CPU de cada módulo cuando se usa *FMK* en el módulo *GM*.

Uno de los principales requisitos de este trabajo es el de reducir el tiempo de computación del módulo *GM*, que como se aprecia en la figura 4.15 es de los más elevados. El tiempo de cómputo de este módulo depende sobre todo del número de celdas que conforman la rejilla que divide al campo, que a su vez es función del tamaño de cada una. En este gráfico está definida a 100 mm, que es la configuración original.

En esta sección se presenta la propuesta de esta tesis: algoritmos híbridos, que se componen del método global *FMK* y uno o varios *EKFs*, para satisfacer las necesidades de auto-localización del robot cumpliendo con los requisitos y restricciones que se han planteado anteriormente.

Se plantean dos estrategias de combinación: combinar un *EKF* con *FMK* (al que denominaremos abreviadamente *KFMK*) o varios *EKFs* con *FMK* (que también abreviaremos como *NKFMK*). Estas estrategias se detallan en las siguientes secciones.

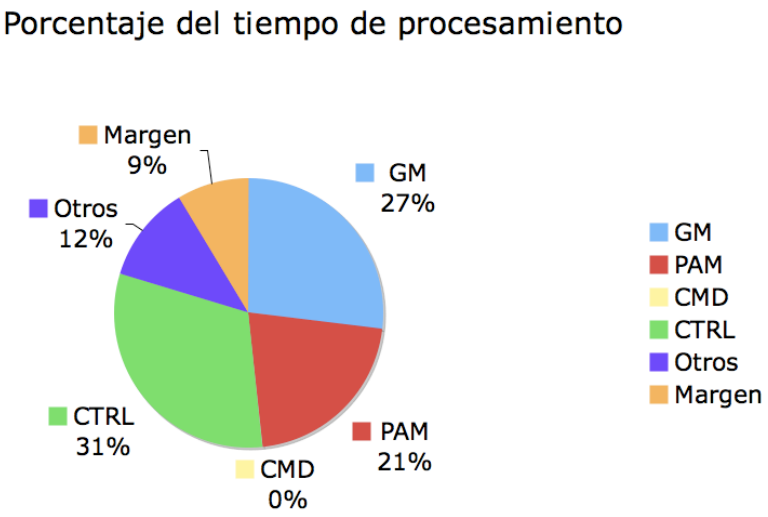


Figura 4.15: Porcentaje de tiempo que usa cada módulo en un ciclo.

### 4.4.1. *KFMK*

La primera idea que intentamos para solventar los problemas del EKF fue usar un único *EKF* y el método *FMK*. Como se puede observar en la figura 4.16, ambos métodos se ejecutan de manera independiente, teniendo las mismas entradas. Estas entradas son la información odométrica y la perceptiva, tal y como se han definido en la sección 4.1. Este método tiene como salida la posición del robot. Esta posición es directamente la estimación que calcula el *EKF*.

Es importante averiguar cuál es el tamaño óptimo de la celda en la rejilla de *FMK*. Con este objetivo se ha realizado un análisis que ha consistido en realizar varias ejecuciones del método presentado en esta sección con distintas configuraciones de tamaño de celda. Se ha medido el tiempo de cómputo y el error en la estimación. La configuración óptima es la que combina un tamaño menor de la celda, para tener una precisión alta, con unos tiempos de computación razonables.

En la figura 4.17 se muestran los resultados de las ejecuciones del método *KFMK* para tamaños de celda de 100 a 1000 mm., que se representan en el eje de abscisas. En la gráfica situada a la izquierda de la misma figura se muestra el tiempo de cómputo en función del tamaño de la celda. Se observa cómo la diferencia entre 100 mm y

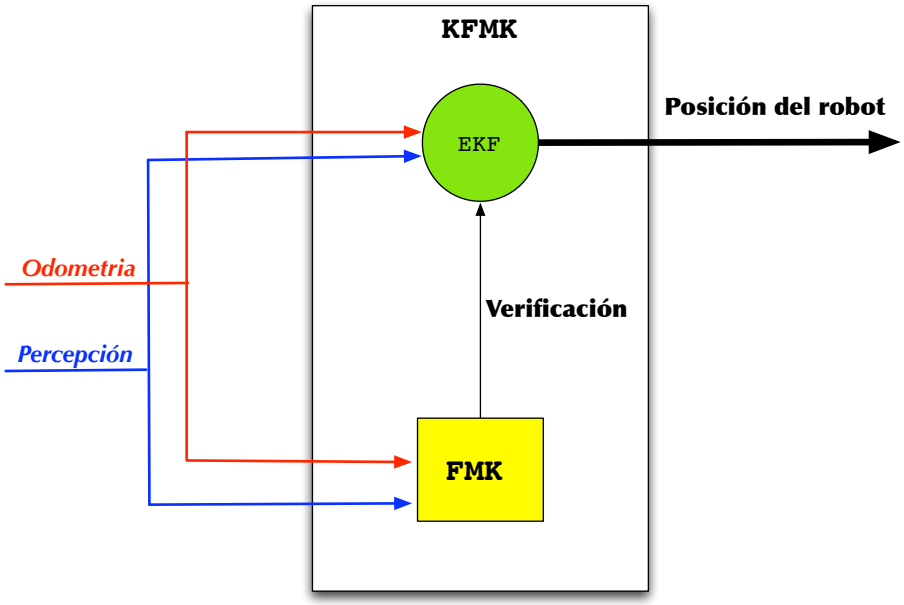


Figura 4.16: Esquema de combinación de un *EKF* y *FMK*.

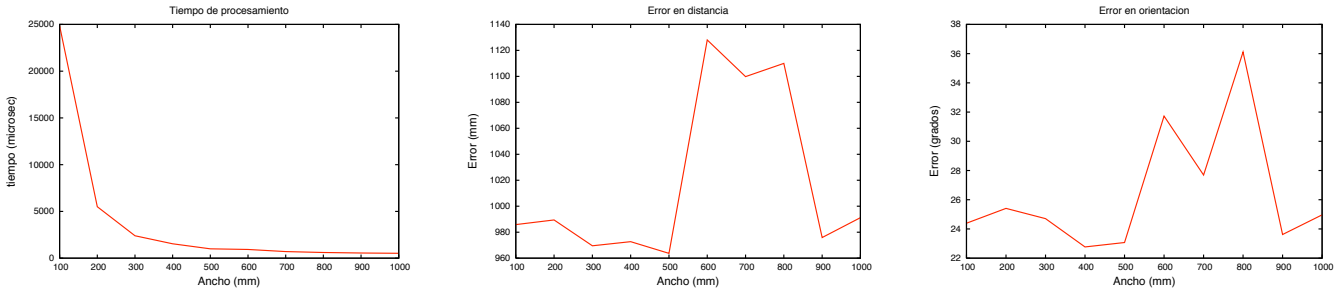


Figura 4.17: Variación del tiempo de computación (izquierda), error en la estimación de $x, y$ (centro) y error en estimación $\theta$ dependiendo del tamaño de la celda (eje de abscisas en todos).

tamaños de 300 mm a 400 mm es considerable, no obteniendo mejoras sensibles a partir de 500 mm. La gráfica central muestra el error en distancia en función del tamaño de la celda. El mínimo se sitúa en 500 mm y tamaños mayores no son capaces de localizar al robot. El mismo resultado se observa en el caso de la gráfica situada a la derecha, que corresponde al error en la orientación.

A la vista de estos resultados, parece adecuado usar un valor de $500$ mm de ancho de celda, ya que parece ofrecer una buena relación entre el tiempo de cómputo y la precisión. No se barajan tamaños mayores ya que no tendría sentido evaluar *EKFs* en celdas de casi un metro y no se obtienen mejoras significativas en cuanto a tiempo de computación.

Una vez definido el tamaño óptimo de la celda, vamos a describir el proceso de localización usando el método KFMK con esa resolución. Tanto el *FMK* como el *EKF* lo hacen igual que cuando se ejecutan de manera individual. Las únicas fases en las que interactúan es cuando se consulta la información de *FMK* para la inicialización del *EKF* y para su evaluación:

- La información de localización que almacena *FMK* se puede observar en la figura 4.18. Este método mantiene una rejilla, que en este caso hemos configurado para que el ancho de cada celda sea mayor que en caso original, en el que el ancho estaba ajustado a 100 mm. Cada una de las celdas de esta rejilla representa la posibilidad de que el robot se encuentre en el área que corresponde a la celda. La posición de la posición del robot se calcula a partir de las celdas donde sea más posible que se encuentre. En la figura se puede observar que hay 9 celdas cuya probabilidad es mayor que el resto. La posición del robot se calcula como el centro de estas 9 celdas. La incertidumbre sobre esta estimación se establece midiendo el ancho y el alto del conjunto de celdas probables. $dx$ es el ancho de este conjunto y $dy$ el alto. La calidad de esta estimación es inversamente proporcional a la incertidumbre.

- Al iniciar el proceso, *EKF* se inicia en la posición que *FMK* indica, estableciendo como covarianza $P$ del *EKF* la incertidumbre que tenga *FMK*. En las ecuaciones 4.21 y 4.22 se refleja cómo el estado $s$ y la covarianza $P$ en esta estimación se inician con la información de *FMK*.

  La posición inicial del robot y la incertidumbre que se usan para inicializarlo son las de *FMK*.

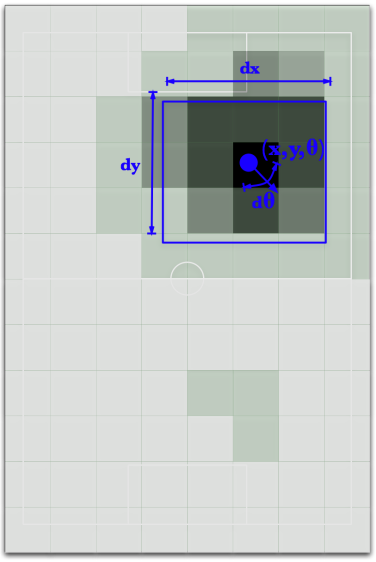


Figura 4.18: Posición indicada por FMK y su incertidumbre asociada. Las celdas oscuras indican mayor posibilidad.

$$s_{EKF} = \begin{pmatrix} x_{EKF} \\ y_{EKF} \\ \theta_{EKF} \end{pmatrix} = \begin{pmatrix} x_{FMK} \\ y_{FMK} \\ \theta_{FMK} \end{pmatrix} \tag{4.21}$$

$$P_{EKF} = \begin{pmatrix} (\frac{dx_{FMK}}{2})^2 & 0 & 0 \\ 0 & (\frac{dy_{FMK}}{2})^2 & 0 \\ 0 & 0 & (\frac{d\theta_{FMK}}{2})^2 \end{pmatrix} \tag{4.22}$$

- Cuando la incertidumbre del *EKF* comienza a crecer puede ser porque la estimación de la posición del robot es errónea. Este crecimiento se debe a que las observaciones que el robot realiza de las marcas visuales no tienen que ver con las que deberían percibirse, y son rechazadas por medio del análisis del valor calculado en la ecuación 4.15 (*umbral de rechazo*). La calidad del *EKF* es inversamente proporcional a su incertidumbre. Si esta es inferior a un umbral, y la información que proporciona *FMK* tiene una calidad alta e indica que la estimación del *EKF* es errónea, se inicia el *EKF* usando la posición estimada

por *FMK* y su incertidumbre. Todo el proceso de localización queda reflejado en el algoritmo 1.

```
Initialize pos_fmk using full uncertainty;
Initialize pos_EKF using full uncertainty;
while true do
    Predict pos_fmk using odometry;
    Predict pos_EKF using odometry;
    Correct pos_fmk using landmark information;
    Correct pos_EKF using landmark information;
    if Prob(cell_fmk(pos_EKF,x, pos_EKF,y)) < threshold then
        if (quality(pos_fmk) is high) and (distance(pos_fmk, pos_EKF) is large) then
            Initilize pos_EKF to pos_fmk
        end
    end
    robot position ← pos_EKF;
end
```

**Algoritmo 1**: Estrategia de combinación de $FMK + EKF$.

### 4.4.2. Combinación de varios *EKF* con FMK

El método descrito en la sección anterior combina un único *EKF* con un *FMK*. Otra opción que hemos diseñado es la de combinar varios *EKFs*, y no sólo uno, con un *FMK*. Mantener un *EKF* no es demasiado costoso computacionalmente, como mostraremos a continuación. Veremos qué ventajas puede aportar tener varios *EKFs*.

El método anterior tiene varios problemas. El primero de ellos es que no puede manejar varias hipótesis a la vez. Únicamente es capaz de mantener una hipótesis y reiniciarse si la información de *FMK* lo cree conveniente. Esta reinicialización es poco flexible. Si la evaluación del *EKF* es incorrecta se pierde la estimación que existía hasta ese momento. Esto se puede producir cuando se percibe un falso positivo durante varios segundos y erróneamente se avalúa negativamente el *EKF* actual y se reinicia a una posición equivocada.

Esta segunda estrategia de combinación usa varios *EKFs* que se ejecutan en paralelo con un método *FMK*. El esquema mostrado en la figura 4.19 es parecido a la estrategia de combinación descrita anteriormente, pero con varios *EKFs*.

El número de *EKFs* varía durante el proceso de localización para describir las múltiples hipótesis sobre la posición del robot. Al inicio se crea un único *EKF*. Si aparece una nueva hipótesis sobre una posición, se crea un nuevo *EKF* en esa posición

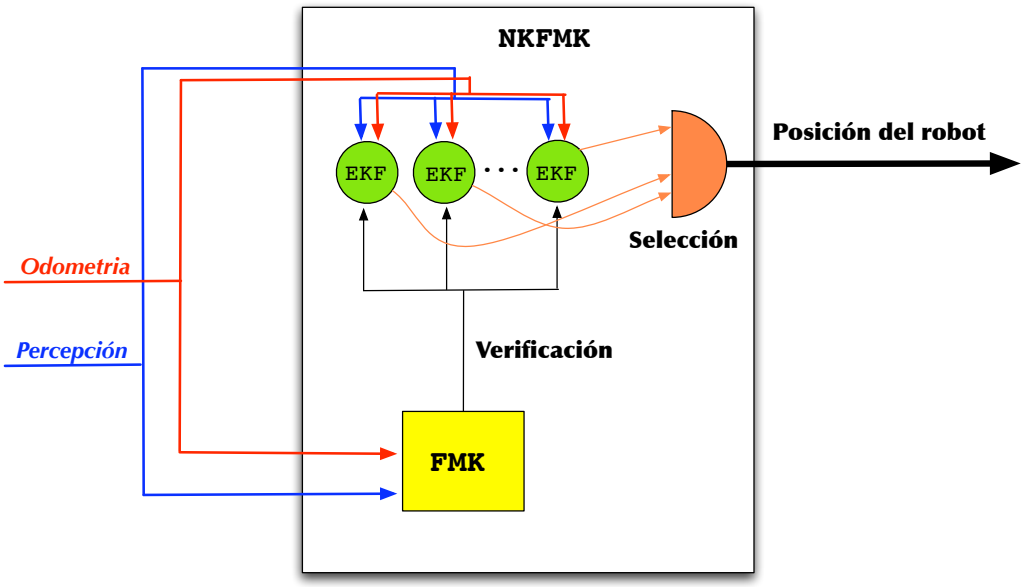


Figura 4.19: Esquema de combinación de varios *EKFs* y *FMK*.

que evoluciona en paralelo con los ya existentes. Si una hipótesis deja de ser válida, o hay varios *EKFs* para una misma hipótesis, se elimina un *EKF*. De esta manera el número de *EKFs* es dinámico.

La información que aporta *FMK* es doble. Por un lado, se encarga de aportar nuevas hipótesis sobre la posición del robot. Si *FMK* indica un posición con poca incertidumbre y no hay ningún *EKF* que contemple esta hipótesis, se inicia un *EKF* en esta posición. Por otro lado, si *FMK* indica que la estimación de uno de los *EKFs* es muy poco probable, ese *EKF* se elimina si coincide que su incertidumbre es elevada.

Esta estrategia soluciona varios de los problemas que se presentan en la estrategia descrita anteriormente, en la que se usa un único *EKF*. Las condiciones para tener en cuenta la hipótesis que aporta *FMK* ya no son tan restrictivas como en el caso de un único *EKF*. Todas las hipótesis con poca incertidumbre son aceptadas y se inicia un *EKF* para contemplar esta hipótesis. Además, aunque se inicie un *EKF* nuevo y éste llegue a ser el que tenga menor incertidumbre, los anteriores *EKFs* no se eliminan directamente, permitiendo volver a la estimación anterior si las observaciones nuevas lo aconsejan.

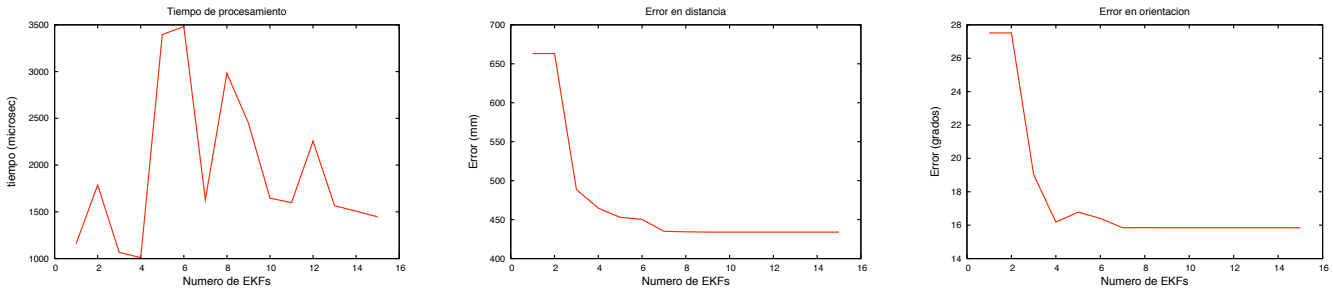


Figura 4.20: Variación del tiempo de computación (izquierda), error en la estimación de $x, y$ (centro) y error en estimación $\theta$ dependiendo del número máximo de *EKFs*.

Aunque haya varios *EKFs*, la salida del módulo *GM* debe ser una única posición. La hipótesis que se toma como salida del módulo *GM* es la que estima el *EKF* con menor incertidumbre. El algoritmo 2 detalla el pseudocódigo del ciclo de ejecución de este método.

La principal cuestión de diseño en este caso es el número de *EKFs* máximos adecuado para estimar la posición del robot. Mantener un *EKF* no es gratis, ya que consume una serie de recursos. Si se permite que haya un número elevado de *EKFs* el tiempo de cómputo podría dispararse. Para averiguar el número máximos de *EKFs*, se realizó un análisis que consistió en ejecutar este método varias veces variando el número máximo de *EKFs* que se conviene crear.

El resultado se muestra en la figura 4.20, donde se representa el coste computacional (gráfica de la izquierda) y el error (gráfica central y derecha). El coste computacional es bajo en el caso de 1, 3 y 4 *EKFs* como máximo. Esta gráfica no es lineal ya que no representa el número fijo de *EKFs*, sino el máximo número de ellos. Además, el tiempo de cómputo que requiere un *EKF* también depende de la incertidumbre que tenga, siendo más costoso de actualizar si la incertidumbre es elevada. Este tiempo de cómputo según aumenta la incertidumbre se debe exclusivamente a cuestiones de implementación y de cómo las bibliotecas matemáticas realizan el cálculo de la inversa de una matriz.

El error no parece mejorar a partir de 7 *EKF* como máximo, siendo la mejora más notable entre 2 y 4 *EKFs*. A la vista de estos resultados la opción elegida es limitar el número máximo de *EKFs* a 4.

```
Initialize pos_fmk using full uncertainty;
while true do
    Predict pos_fmk using using odometry;
    foreach active EKF_i do
        Predict pos_EKF_i using odometry;
    end
    Correct pos_fmk using landmark information;
    foreach active EKF_i do
        Correct pos_EKF_i using landmark information;
    end
    foreach active EKF_i do
        foreach active EKF_j do
            if distance(pos_EKF_i, pos_EKF_j) < threshold_near) then
                if uncertainty(EKF_i) > uncertainty(EKF_j) then
                    remove EKF_i;
                else
                    remove EKF_j;
                end
            end
        end
        if Prob(cell_fmk(pos_EKF_i,x, pos_EKF_i,y)) < threshold then
            if (quality(pos_fmk) is high) and (distance(pos_fmk, pos_EKF) is large) and (uncertainty(EKF) is
            high) then
                remove EKF_i;
            end
        end
    end
    if (quality(pos_fmk) is high) and (distance(pos_EKF_j, pos_fmk) is large) ∀i, j then
        Initialize EKF ← fmk;
    end
    if there is any active filter then
        robot position ← pos_EKF_i where i is the filter with lower uncertainty;
    else
        robot position ← pos_fmk;
    end
end
```

**Algoritmo 2**: *FMK* with multiple *EKF* combination strategy.

En el capítulo siguiente se realizará una extensa experimentación de los métodos propuestos en este capítulo.

# CAPÍTULO 5

# Experimentación

Este capítulo está dedicado a presentar los resultados experimentales conseguidos con los métodos de auto-localización propuestos en el capítulo 4 de esta tesis. Los experimentos presentados en este capítulo validan y caracterizan los métodos propuestos para solucionar este problema en el entorno de la RoboCup.

Los métodos de auto-localización que se comparan en la experimentación son el método Markoviano Difuso (*FMK*), el Filtro Extendido de Kalman (*EKF*), el método que combina un *EKF* y el método *FMK* (*FMK+EKF*) y el método que combina varios *EKFs* y el método *FMK* (*FMK+nEKF*).

Para realizar la experimentación se han implementado los métodos anteriormente enumerados tanto para el software de reproducción (sección B.3) como para el software que se ejecuta en el robot cuadrúpedo AIBO.

La implementación en el software de reproducción permite ejecutar los métodos de auto-localización en un ordenador. Este software usa el contenido de un fichero como entrada de datos. Este fichero contiene la información perceptiva y odométrica que recibe el módulo encargado de la auto-localización en el software que se ejecuta en el robot durante los experimentos con el robot real. Esto permite que se puedan probar simultáneamente uno o varios métodos de auto-localización distintos sobre los mismos datos y así realizar una comparación exhaustiva. También es posible ejecutar

el mismo método con diferentes configuraciones, lo que permite afinar los parámetros de cada método (tamaño de celda de *FMK*, número máximo de *EKFs* en el método *FMK+nEKF*, condiciones de reinicio de *EKFs*, etc.). Con este software es posible obtener resultados experimentales de varios métodos cuando sus entradas son siempre las mismas y de esta manera evaluarlos en igualdad de condiciones.

La implementación de los algoritmos de auto-localización para el robot AIBO se integra en el software del equipo TeamChaos, que se describe en el apéndice A. Un objetivo de esta implementación es evaluar cualitativamente los métodos propuestos y su impacto en el comportamiento real del robot. Ésta es la única manera de comprobar cómo afectan los métodos de auto-localización al comportamiento real de los robots durante el juego.

Existen varias situaciones en las que el robot debe usar el conocimiento sobre su posición para desplazarse a una posición inicial especificada antes de iniciar el encuentro. Estas situaciones son la fase inicial antes de comenzar el juego y la fase que sigue a cada gol anotado al equipo contrario.

Además, durante un encuentro normal existen diferentes roles. Cada jugador toma un rol distinto según su situación en el campo y su distancia a la pelota. En el caso de nuestro equipo, la asignación dinámica de roles se realiza mediante el protocolo *Switch!* [CEA06]. En este protocolo se identifican 3 roles:

- ***Kicker***. Es el comportamiento encargado de chutar la bola hacia la portería y se activa cuando el robot es el más cercano a la pelota. Sólo necesita la información de su posición cuando no percibe ninguna de las porterías. En este caso, la información de localización se usa sobre todo para que el robot dirija la pelota hacia la portería contraria y evita que la envíe hacia su propia portería. Cuando este rol está activo el sistema de visión activa busca únicamente la pelota y esporádicamente una portería, lo que hace que la cantidad de información perceptiva disponible sea escasa. Durante este rol, el robot no realiza ningún tipo de navegación global.

- ***Supporter***. Es el comportamiento encargado de colocar al robot en una posición de apoyo al *Kicker*. Esta posición de apoyo trata de aprovechar el espacio libre que hay cerca de la portería adversaria. El sistema de visión activa busca

por igual tanto la pelota como las porterías y las balizas. Si este rol está activo el robot debe estar atento cuando la pelota se acerca a él, desplazándose a puntos stratégicos determinados del campo usando la información sobre su posición. El éxito de este rol depende en gran medida de la información de auto-localización.

- ***Defender***. Es el comportamiento encargado de colocar al robot entre la pelota y su propia portería. El robot trata de mantenerse a la misma distancia de ambos elementos. Al igual que en el caso del *Supporter*, el sistema de visión activa busca por igual tanto la pelota como las balizas y las porterías. Al tener que colocarse entre la pelota y su portería, no es suficiente tener percepción local, sino que debe disponer de información de auto-localización muy precisa.

Tras analizar las posibles situaciones de juego, se observa que la información de auto-localización que se requiere depende de la tarea que esté realizando el robot. Cuando el robot ha de ir a una posición inicial, ya sea antes del partido o después de cada punto anotado, usa la información de auto-localización para ir a una posición global del campo de juego. De la misma manera, cuando está activo el rol *Supporter* o *Defender*, el robot también ha de navegar hacia posiciones globales del campo de juego.

Además, durante el rol *Kicker*, el robot usa la información de localización en algunas ocasiones para decidir hacia dónde ha de golpear la pelota. Esto hace que el robot necesite consultar esporádicamente la información de localización aunque no la esté usando para navegar.

Es complejo evaluar el éxito de los métodos de auto-localización para llevar a cabo estas acciones durante un partido. El comportamiento del robot se ve afectado por gran cantidad de factores y sus tareas varían dinámicamente. Por esta razón se ha decidido realizar varios experimentos previos para evaluar de manera unitaria las capacidades auto-localización:

- **Experimentos de localización pasiva**. Durante el juego el robot puede hacer uso de la información sobre su posición para determinar su próxima acción. Por esta razón, en este experimento se evalúa la capacidad del robot para mantenerse localizado mientras realiza una tarea diferente a la de navegación.

- **Experimentos de recuperación frente a secuestros**. Durante un partido de la RoboCup es habitual que un robot sea manualmente desplazado de un sitio a otro del campo de juego. También es usual que un robot sea sancionado y sacado del campo de juego. Tras unos segundos, el robot es devuelto al campo en una posición arbitraria. Como es difícil reproducir las situaciones de juego, se realizarán experimentos en que el robot se ve sometido a situaciones de secuestro. Esto evaluará si los métodos son capaces de recuperarse de esta situación y volver a estimar correctamente la posición del robot.

- **Experimentos de navegación por puntos de control**. Durante el juego el robot debe desplazarse de un punto a otro del campo. Para comprobar si los métodos propuestos son adecuados para llevar a cabo esta tarea se plantean varios experimentos. Se evalúa la capacidad del robot de usar la información de auto-localización para navegar a través de una serie de puntos de control.

Para el análisis del error de los métodos de auto-localización es necesario disponer de la posición real del robot. Esta posición se obtiene mediante el sistema de ”verdad absoluta” descrito en la sección B.2. Este sistema detecta mediante un sistema de cámaras cenitales situadas sobre el campo un patrón visual situado encima del robot, recogiendo en cada momento su posición y su orientación. La información obtenida se usa de dos formas. Por un lado, esta posición se envía directamente al robot que la puede usar como un sistema de localización externo. Por otro lado, se almacena en un fichero de ”log” junto con la información odométrica y preceptiva para usarla posteriormente como entrada de datos al reproductor. La cantidad de entradas de este fichero es considerable, siendo alrededor de 4500 por minuto.

En las siguientes secciones se describen los experimentos llevados a cabo. En cada uno de ellos se aportan resultados tanto cualitativos como cuantitativos. El primer grupo de experimentos se realiza con un único robot (secciones 5.1, 5.2 y 5.3). En algunos de ellos el robot usa el sistema de localización externo para aportar la posición real y en otros casos el robot usa su propia estimación sobre su posición. En la segunda parte de los experimentos se analiza el error en la estimación de la posición en una situación de juego con la presencia de varios robots (sección 5.4). Aparte de los resultados cuantitativos, se observará si el robot es capaz de jugar correctamente

(golpear la pelota y colocarse en la posición adecuada) usando la estimación de cada uno de los métodos de auto-localización. En el último experimento se presenta una comparativa de tiempos de ejecución de cada uno de los métodos.

## 5.1. Auto-localización pasiva

El objetivo de este experimento es evaluar la calidad de la información que aportan los métodos de auto-localización en las situaciones en las que no se realiza una búsqueda activa de los elementos del entorno que se usan para localizarse (balizas y porterías). En este experimento se configura la atención del robot para que principalmente observe la pelota, que se sitúa cerca del centro del campo. Al igual que en el caso del comportamiento del *Kicker*, esporádicamente entran en el campo visual del robot balizas y porterías que le ayudan a auto-localizarse. Así pues, se evalúan los métodos de auto-localización en modo pasivo, esto es, cuando el objetivo no es buscar los elementos del campo de juego para localizarse ni navegar. Es importante puntualizar que en estas pruebas no hay más robots en el terreno de juego, como ocurre en una situación de juego real.

Para sistematizar el proceso de experimentación se ha hecho que el robot siga una ruta entre varios puntos usando el sistema de localización externo. La ruta definida en este experimento se muestra en la figura 5.1.

Este experimento se compone de una treintena de repeticiones en condiciones y trayectorias similares. De esta manera los resultados finales no se ven afectados por posibles errores perceptivos puntuales. La duración de cada una de las repeticiones varía ligeramente de una a otra, siendo la media de 55 segundos.

La cuestión que se plantea en este experimento se resuelve analizando los siguientes aspectos:

- ***Posición estimada y posición real***. En el plano cartesiano del suelo se muestran las coordenadas $(x, y)$ correspondientes a la estimación de la posición del robot y la posición real del robot captada por el sistema de verdad absoluta. Esta representación permite comprobar visualmente la diferencia entre ambas. Gracias a esta representación, se pueden explicar diferentes situaciones que se

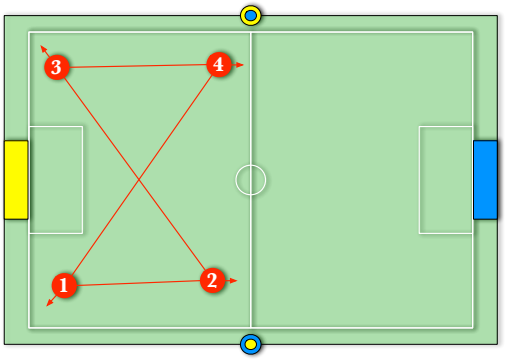


Figura 5.1: Recorrido del experimento usando un sistema de localización externo.

dan durante el experimento en base a los elementos fijos del campo de juego que percibe.

- ***Evolución del error en la estimación durante el recorrido***. Se analiza cómo evoluciona el error cuando varía la cantidad y calidad de las observaciones. Se presentará en forma de gráficas que reflejan la evolución del error a lo largo del tiempo en el experimento.

- ***Resumen numérico del error en la estimación***. Se presenta la media del error, su desviación típica y una serie de estadísticos que describen la fiabilidad y precisión de cada uno de los métodos numéricamente.

- ***Intervalos de confianza del error***. Un intervalo de confianza en estadística es un intervalo de valores en el cual, con una probabilidad o nivel de confianza determinado, se sitúa el parámetro a analizar, en este caso el error en la estimación. Un intervalo de confianza indica la probabilidad de que cierto valor se encuentre en un determinado intervalo de valores. En este caso, la probabilidad se calcula experimentalmente como la proporción de valores de error que se encuentran en cada intervalo.

El análisis de los intervalos de confianza es útil en esta aplicación, ya que representa el porcentaje de tiempo que el robot se encuentra mejor o peor localizado. Creemos que para la aplicación en la que se usarán los métodos de auto-localización, a menudo es preferible que el robot esté razonablemente localizado mucho tiempo aunque sea con poca precisión a que tenga mucha precisión normalmente pero se pierda completamente con cierta frecuencia.

En primer lugar analizamos dos ejecuciones diferentes, representativas de situaciones habituales, por la cantidad y calidad de las observaciones. Esto nos permite analizar el comportamiento y robustez de los algoritmos frente a percepciones ruidosas

Las gráficas de la primera ejecución a analizar para cada uno de los métodos de auto-localización se muestran en la figura 5.2. Las 4 figuras superiores muestran en el plano cartesiano en rojo la posición real del robot y en azul la estimación de cada uno de los métodos de auto-localización. Las dos gráficas de la fila inferior muestran la evolución del error en la estimación a lo largo del experimento. El eje de abscisas es el tiempo real en la ejecución del experimento. En la gráfica situada a la izquierda, el eje de ordenadas representa la distancia euclídea entre la posición estimada por los métodos de auto-localización y la posición real del robot. En la gráfica situada a la derecha, el eje de ordenadas representa la diferencia en grados entre la orientación estimada y la real.

Durante esta ejecución del experimento se producen de manera reiterada varios errores en la percepción:

**[1]** Cuando el robot llega al punto de control etiquetado como 2 en la figura 5.1, el robot percibe la portería amarilla un metro más lejos de lo que debería por fallos en el sistema perceptivo.

**[2]** En la trayectoria que discurre desde el punto de control 2 al 3, el robot percibe momentáneamente dos balizas en la posición real de la baliza situada enfrente de él hacia la derecha. Esta situación ocurre durante un instante, aunque a continuación el robot no recibe ninguna percepción hasta que casi ha llegado al punto de control 3.

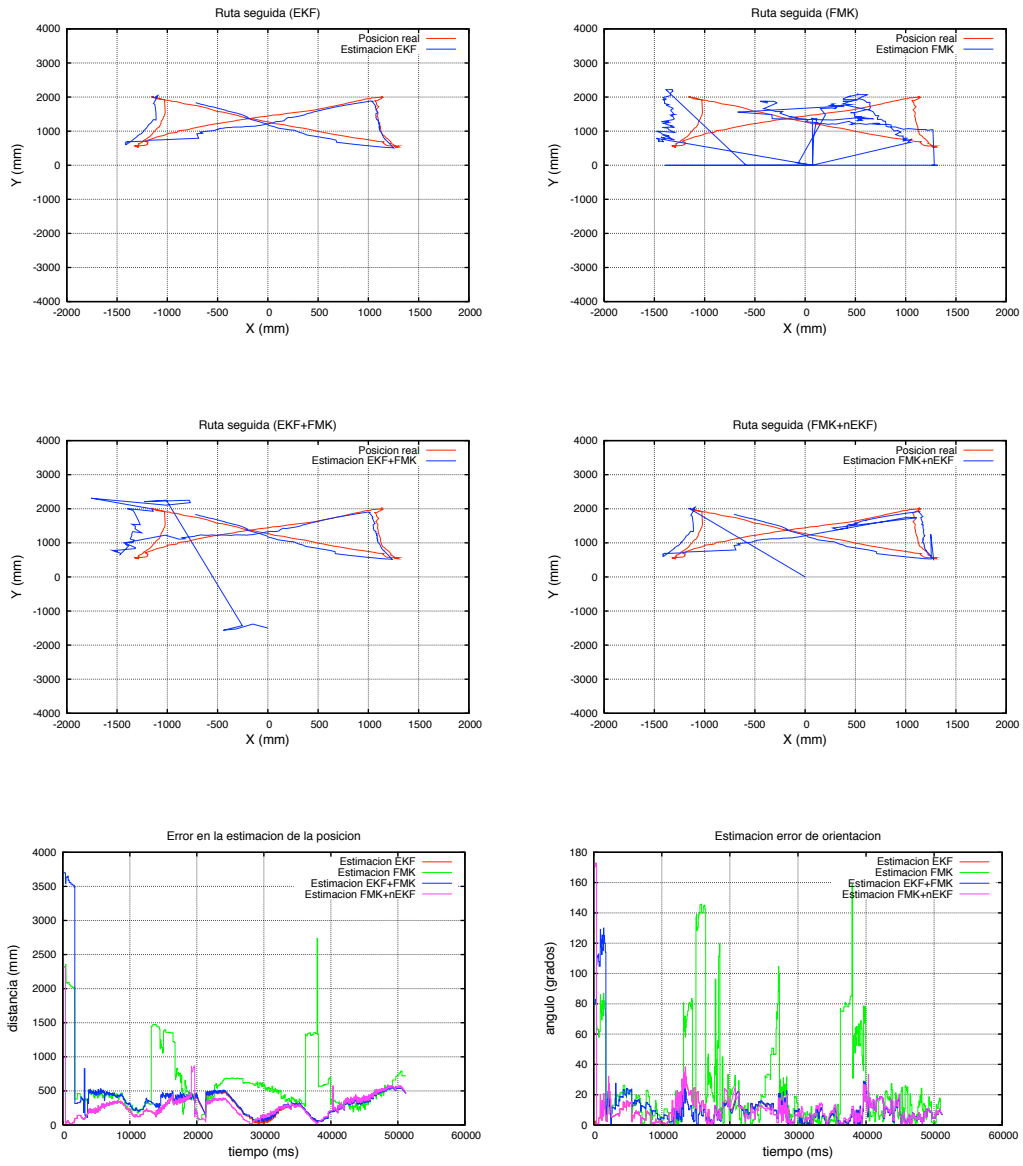


Figura 5.2: Primera ejecución del experimento.

**[3]** A la mitad de la trayectoria que discurre desde el punto de control 3 al 4, el robot vuelve a percibir las dos balizas juntas en el mismo sitio de antes. Se recupera de esta situación cuando llega al punto de control 4 y encara al primer punto de control.

**[4]** Cuando el robot se encuentra cerca de la portería amarilla (al llegar al punto de control 1 y al finalizar el experimento), ésta se percibe parcialmente y el módulo de visión calcula erróneamente el ángulo a ella.

Los distintos métodos de auto-localización se comportan de manera diferente cuando suceden los errores anteriormente enumerados. La odometría es ruidosa, pero se mantiene la mayor parte del tiempo en los valores de error caracterizados en la sección 4.2.

Durante esta ejecución, el método *EKF* no se ve afectado prácticamente por los errores en la percepción enumerados anteriormente. Tras iniciarse, su incertidumbre disminuye y las observaciones erróneas son rechazadas gracias al *filtro de incoherencias* que se presentó al final de la sección 4.3. En la gráfica superior izquierda se observa cómo la estimación que realiza este método siempre está cerca de la posición real del robot.

Hay que puntualizar que el método *EKF*, al ser un método de auto-localización local o *tracking*, se inicia en cada ejecución en la posición real del robot obtenida mediante el sistema de ”verdad absoluta”. De otra manera sería imposible realizar un análisis de sus resultados ya que es complicado que converja a la posición del robot al inicio de su operación.

El método *FMK* es el más afectado por los errores en la percepción. Aunque no le afecta demasiado el error perceptivo **[1]**, en el transcurso del punto de control 2 al 3 el error perceptivo **[3]** hace que la probabilidad se reparta prácticamente por igual entre todas las celdas de la rejilla de probabilidad. Esto hace que la estimación de la posición del robot sea el centro del campo y también que la incertidumbre sea máxima. Esta situación se aprecia en las gráficas de la fila inferior como un aumento significativo del error. Esta misma situación se vuelve a producir al llegar al punto de control 4, debido al error perceptivo **[3]**.

El método *FMK+EKF* se inicia cuando el método *FMK* aporta una hipótesis que considera fiable, es decir, cuando su incertidumbre es baja. Se puede observar en las gráficas inferiores que el error en distancia del método *FMK+EKF* disminuye tras un periodo inicial de convergencia. A continuación, este método se comporta de manera semejante al método *EKF*, ya que la incertidumbre del *EKF* que mantiene internamente este método, que está estimando la posición del robot, se mantiene baja durante el resto del trayecto. Al mantenerse baja la incertidumbre, la estimación no se evalúa como errónea y no da lugar a que se inicie el *EKF* en alguna hipótesis nueva indicada por *FMK*.

El método *FMK+nEKF* comienza su ejecución de manera semejante al método *FMK+EKF*. La diferencia principal es que las hipótesis que aporta *FMK* no se descartan, sino que se inician nuevos *EKFs* en ellas.

Durante esta ejecución, el *EKF*, que se inicia en la posición correcta, consigue estimar la posición del robot durante todo el recorrido. Aún así, hay dos instantes en los que se inician varios *EKFs* erróneos y que momentáneamente tienen menos incertidumbre que el *EKF* correcto. Esto se produce la primera vez en el transcurso del punto de control 2 al 3. Al no existir percepciones y rechazar el falso positivo de la baliza que comentamos anteriormente, la incertidumbre del *EKF* aumenta. Aunque este *EKF* rechazó el falso positivo, *FMK* falló e inició un nuevo *EKF* en la posición errónea. Al seguir percibiendo durante varios ciclos el falso positivo, la incertidumbre del *EKF* correcto supera a la del erróneo y durante un instante la estimación de este método corresponde a un *EKF* equivocado y es errónea. Cuando se vuelven a tener percepciones correctas, la incertidumbre del *EKF* correcto disminuye y el método recupera los niveles bajos de error. En las gráficas de error inferiores se observan estas situaciones como un aumento momentáneo del error. En el resto de la ejecución el error es bajo.

Los errores perceptivos enumerados anteriormente se vuelven a producir en mayor o menor medida a los largo de las distintas ejecuciones que se realizan en este experimento. En la ejecución que acabamos de comentar el *EKF* no se ve muy afectado por estos errores perceptivos. La razón es que la mayor parte son rechazados y el espacio de tiempo entre percepciones correctas no es muy grande. Tanto el méto-

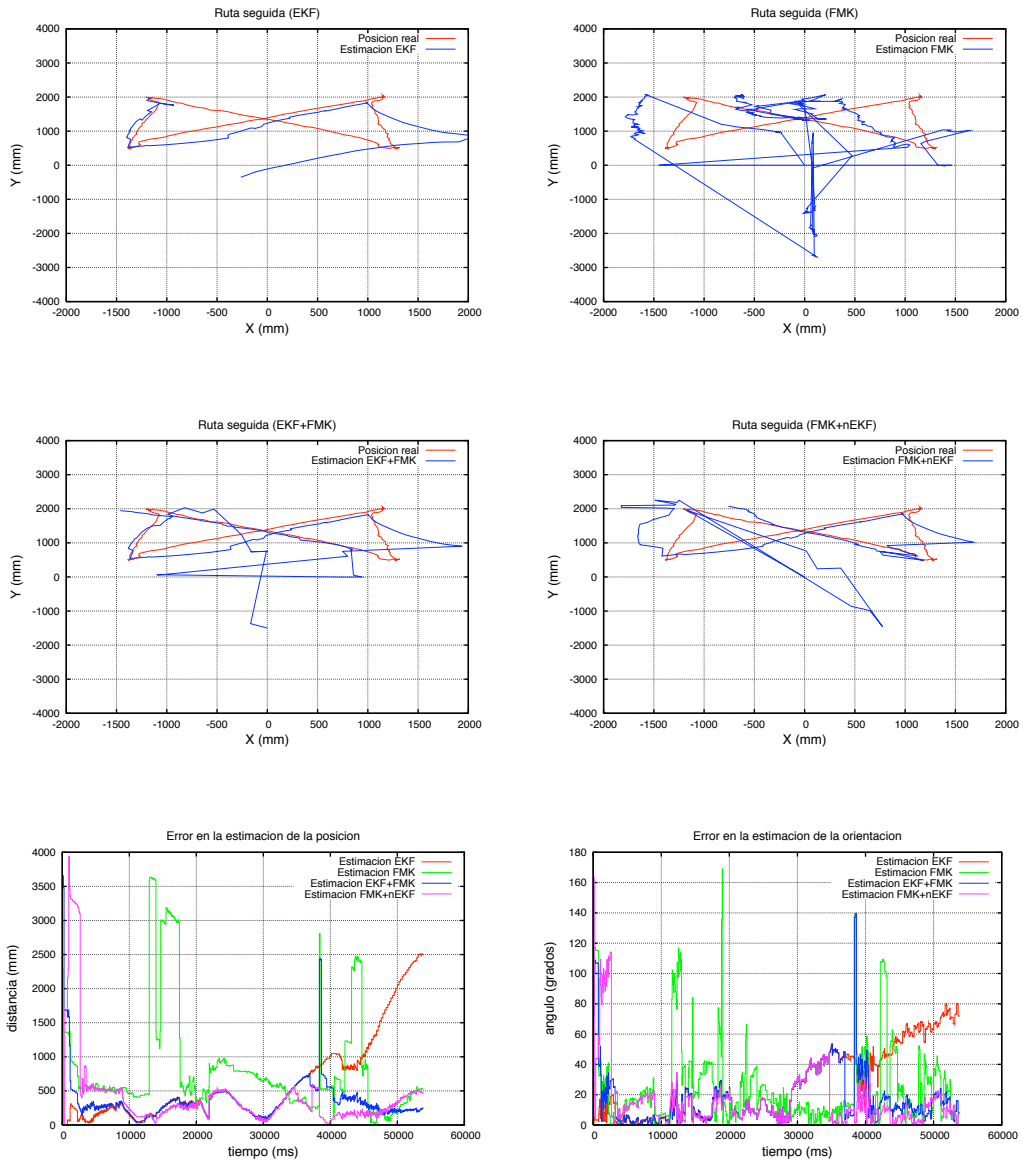


Figura 5.3: Segunda ejecución del experimento.

do *FMK+nEKF* como *FMK+EKF* mantienen internamente desde el comienzo de la ejecución hasta el final el mismo *EKF* para obtener la posición del robot.

La segunda ejecución que pasamos a comentar está representada en la figura 5.3 de una forma similar a la anterior. En ésta las percepciones presentan mayor error y en determinados momentos la información odométrica presenta mayor ruido que en el caso anterior. En el trayecto desde el punto de control 2 al 3, la portería amarilla se percibe mucho más lejos de lo que en realidad está. Asimismo, en el trayecto desde el punto de control 3 al 4, las percepción de las balizas presentan imprecisiones en la distancia a las mismas. En el último trayecto, tras el punto de control 4, la portería amarilla vuelve a percibirse más lejos de los que en realidad está.

El método *EKF* se ve afectado por dos tipos de errores tras llegar al punto de control 3. El primero de ellos se debe al cálculo de la odometría en el giro en este punto. A continuación, los errores al percibir las balizas hacen que estas percepciones sean rechazadas, no pudiendo corregir la estimación y terminando por divergir completamente de la posición real del robot.

El método *FMK* se ve afectado sobre todo por la percepción de la portería amarilla. En las gráficas que corresponden a su error, éste es elevado en el momento en que percibe esta portería. La estimación se desplaza incluso hasta cerca de la portería azul, como se muestra en la gráfica que representa la posición real y la estimación de este método.

En este caso, el método *FMK+EKF* debe reiniciar su *EKF* cuando se producen los errores tras el giro en el punto de control 3 (instante 35000 en la gráfica inferior de la figura 5.3). Cuando el *EKF* que mantenía la estimación de la posición del robot ha divergido, comienza a aumentar su incertidumbre al no incorporar percepciones. Cuando el *EKF* se encuentra en la posición correspondiente a una celda de *FMK* que se considera improbable, y a una distancia elevada de la hipótesis que mantiene *FMK* sobre la posición del robot, el *EKF* se inicia con la nueva hipótesis del *FMK*. En la gráfica del error se observa cómo el error aumenta entonces hasta más de dos metros. Esto se produce porque el *EKF* se inicia en un primer momento a una posición incorrecta. Al iniciarse con una incertidumbre elevada, el filtro se vuelve a reiniciar a la posición correcta cuando se comienzan a percibir correctamente las marcas visuales del campo y *FMK* aporta la hipótesis correcta.

El método *FMK+nEKF* afronta de diferente forma los errores en las percepciones. Cuando éstos se producen, comienzan a iniciarse nuevos filtros en las posiciones que indica *FMK*. Cuando el *EKF* que mantenía la estimación anterior de la posición del robot diverge, hay otros *EKF* en posiciones más cercanas a la real del robot que comienzan a ver disminuir su incertidumbre al incorporar percepciones correctas. Este método se recupera de esta manera más rápidamente de lo que lo hace el método *FMK+EKF*, manteniéndose estable y preciso a partir de este momento.

El análisis que acabamos de realizar sobre estas dos ejecuciones representativas muestra cómo se comportan los métodos de auto-localización evaluados. Este comportamiento se mantiene a lo largo de la treintena de ejecuciones que hemos realizado en este experimento. Los errores perceptivos que hemos comentado se producen en mayor o menor medida en estas ejecuciones. Existen errores esporádicos, pero hay otros que aparecen a lo largo de todo el experimento sistemáticamente. La percepción errónea de la distancia a la portería amarilla cuando el robot está cerca del punto de control 2 es frecuente en situaciones reales de juego, por lo que este experimento nos parece muy interesante. También se percibe erróneamente el ángulo a esta portería cuando el robot está muy cerca. Asimismo, en este experimento existen habitualmente errores en la percepción de balizas cerca del punto de control 4. Estos errores se deben principalmente a problemas de iluminación y a situaciones en que las marcas visuales no son percibidas completamente, por lo que no se corrigen correctamente por el modelo perceptivo analizado en la sección 4.1.

A continuación vamos a analizar el error medio. Como comentamos anteriormente, todas las ejecuciones del experimento tienen una duración similar y el recorrido es aproximadamente el mismo. Esto implica que podremos suponer que en un instante, en todas las ejecuciones del experimento, el robot se encontrará en una posición que podemos considerar igual y por tanto donde obtendrá similares percepciones. Con esta consideración, podemos analizar el error medio en el tiempo, tanto en distancia como en orientación.

En la figura 5.4 representamos la media (línea azul) y la desviación típica del error (espacio entre las líneas rojas) en la estimación tanto en distancia (columna de la izquierda) como en orientación (columna de la derecha). La primera fila corresponde a la media de error y la varianza en la estimación obtenidas con el método

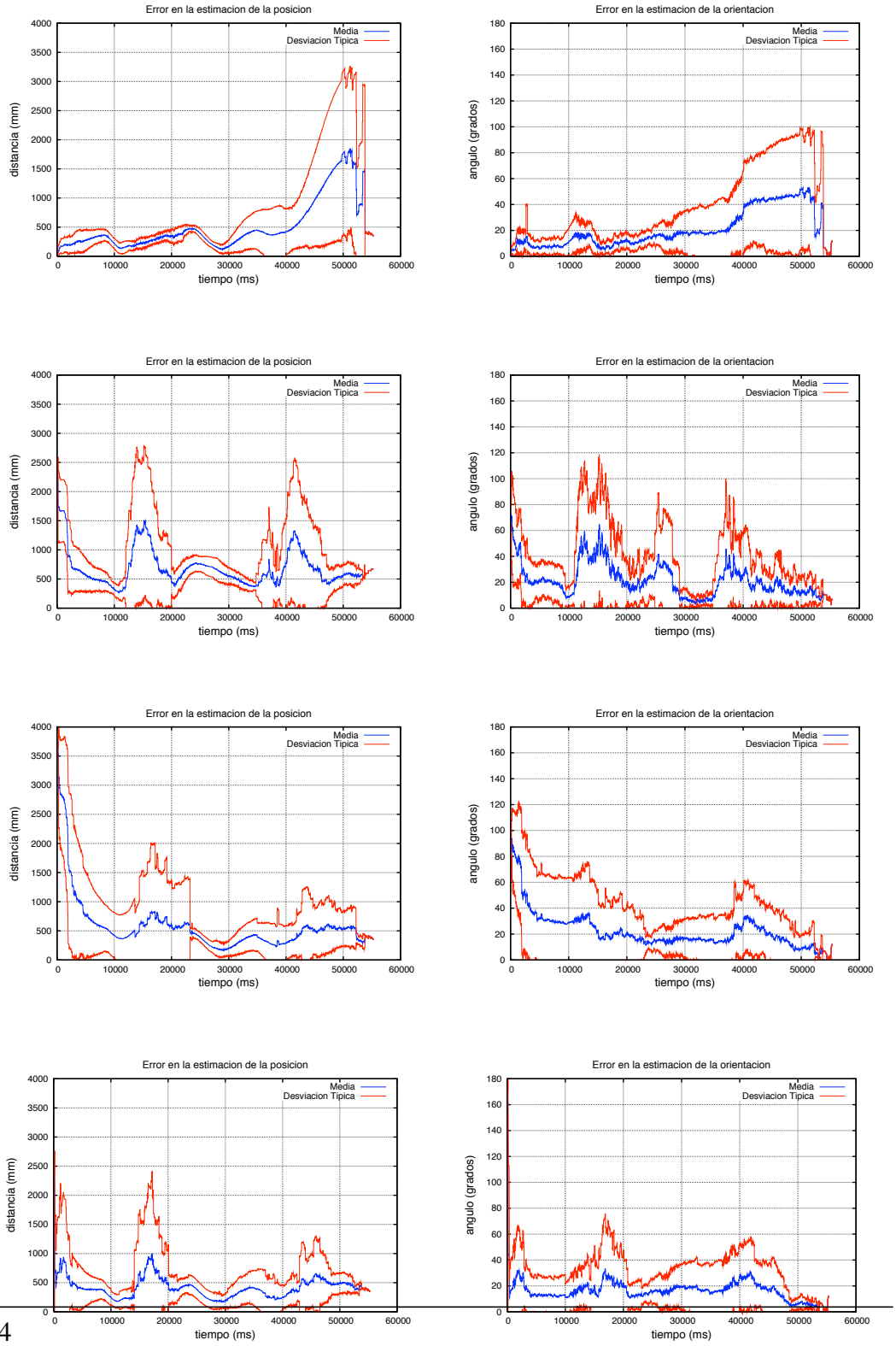




Figura 5.4: Media de error de en distancia (columna de la izquierda)y orientación (columna de la derecha) entre la estimación en la posición y la posición real del robot.

*EKF*. Como se puede observar en las gráficas, el error medio y la desviación típica se mantienen bajos al inicio, durante los primeros 30 segundos, más o menos. A partir de este instante estos valores comienzan a crecer. Esto indica que este método estima correctamente la posición del robot hasta que se acerca al punto de control 3 de la figura 5.1. En ese instante el robot se ve afectado por errores perceptivos y de actuación que hacen que diverja en la mayor parte de las ejecuciones del experimento.

La segunda fila corresponde al método *FMK*. En esta gráfica se muestra cómo el error medio tiene máximos en los momentos en los que se producen los giros, que corresponden con situaciones de falsos positivos, errores elevados en la percepción de la portería amarilla o errores en la actuación. En el resto de la ejecución las percepciones son correctas y la estimación de la posición de este método es aceptable. Estos resultados ponen de manifiesto lo sensible que es este algoritmo a los errores perceptivos que se producen frecuentemente durante el juego.

El error del método *FMK+EKF* se representa en la tercera fila de esta figura. Lo más destacado es el elevado error medio y la elevada desviación típica al inicio de las ejecuciones. Esto es debido a que este método tarda en converger hasta que estima correctamente la posición real del robot. A partir de los primeros 23 segundos, en los que también se producen errores perceptivos que afectan a la estimación, en la mayor parte de las ejecuciones se alcanza la posición correcta. La baja desviación típica indica que la práctica totalidad de los métodos analizados alcanza en este momento un error inferior a 500 mm.

El método *FMK+nEKF* converge rápidamente y estima correctamente la posición del robot. Su error se mantiene bajo, aunque aumenta ligeramente cuando se producen los errores perceptivos comentados anteriormente.

Por último, en la figura 5.5 se presentan de forma agrupada toda la información de medias descrita anteriormente con el objetivo de poder compararlas fácilmente. Se puede observar que el método *FMK+nEKF* mantiene la estimación más precisa durante todo el recorrido.

Una vez analizadas las gráficas correspondientes a los experimentos, es necesario analizar el error numéricamente. El estadístico que resume de manera más descriptiva los resultados es la media del error. Además de la media, los cuartiles dan idea de la cantidad de ciclos en los que el error es alto.

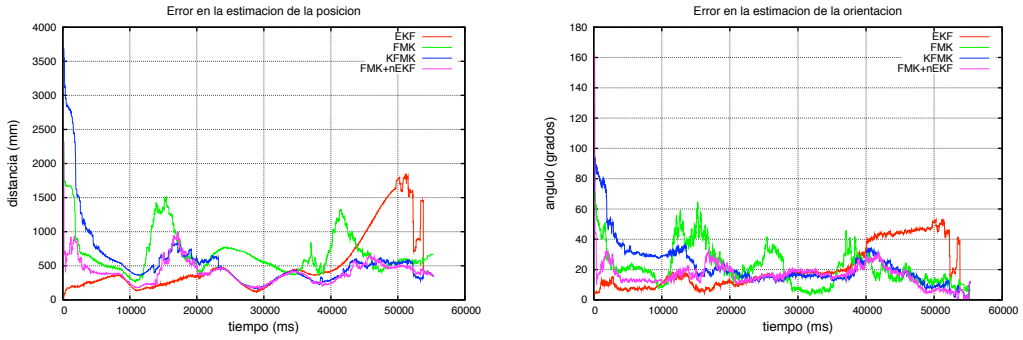


Figura 5.5: Media de error en distancia y orientación entre la estimación en la posición y la posición real del robot para todos los algoritmos.

| Estadístico | *EKF* | *FMK* | *FMK+EKF* | *FMK+nEKF* |
|---|---|---|---|---|
| Primer Cuartil | 175.03 | 331.21 | 211.25 | 183.37 |
| Mediana | 285.43 | 509.81 | 334.09 | 291.25 |
| Tercer Cuartil | 448.45 | 759.03 | 535.4 | 451.54 |
| **Media** | **472.12** | **687.72** | **578.39** | **415.71** |
| Desviación típica | 625.26 | 638.8 | 740.31 | 484.9 |

Tabla 5.1: Análisis estadístico del error en distancia de la estimación de la posición del robot (mm).

Como se observa en las tablas 5.1 y 5.2, el error máximo corresponde al algoritmo *FMK*, siendo el más preciso *FMK+nEKF*. Como se ha comentado anteriormente, es engañoso que el segundo mejor método sea *EKF*. Funciona correctamente en este experimento porque, al haberlo inicializado a la posición correcta, más que localización, se trata de un ejemplo de *tracking*. Para comprobarlo basta con observar los resultados del experimento del secuestro (sección 5.2), donde discutiremos este asunto con más detalle.

Como comentamos al inicio de esta sección, otro aspecto analizado es el de los intervalos de confianza. En este experimentos hemos analizado los intervalos de confianza en el caso del error en la estimación de la posición y el error en la estimación de la distancia.

| Estadístico | *EKF* | *FMK* | *FMK+EKF* | *FMK+nEKF* |
|---|---|---|---|---|
| Primer Cuartil | 4.6835 | 5.8107 | 5.4687 | 4.6768 |
| Mediana | 10.458 | 13.033 | 11.989 | 9.989 |
| Tercer Cuartil | 23.146 | 26 | 27.954 | 20.593 |
| **Media** | **20.567** | **22.299** | **23.23** | **17.327** |
| Desviación típica | 26.362 | 27.914 | 28.262 | 21.759 |

Tabla 5.2: Análisis estadístico del error en orientación de la estimación de la posición del robot (grados).

En el primer caso, el del error en la estimación de la posición, hemos definido 5 intervalos de confianza: $[0, 300]$, $[300, 500]$, $[500, 1000]$, $[1000, 1500]$ y $[1500, 3000]$, todos ellos refiriéndose a la distancia euclídea entre la posición estimada y la real, en milímetros. Cuando el error se encuentra en los dos primeros intervalos, la información de localización es suficientemente precisa para que el robot pueda cumplir satisfactoriamente cualquier acción dependiente de la información de auto-localización. Cuando el error se encuentra en el intervalo $[500, 1000]$, es complicado que el robot pueda cumplir acciones que necesiten precisión en la información de auto-localización pero, aun así, esta información es útil ya que permite saber en qué zona del campo se encuentra el robot. Los dos últimos intervalos representan una estimación de la posición equivocada que representa el tiempo en que el robot se encuentra perdido.

En la figura 5.6 se han representado los resultados en el análisis del error en la estimación de la posición usando intervalos de confianza. La gráfica superior muestra los resultados por cada intervalo y la inferior el acumulado. En el eje de abscisas se representan cada uno de los intervalos de confianza anteriormente descritos. En el eje de ordenadas se representa el porcentaje de tiempo que el error en la estimación de la posición se encuentra en cada intervalo.

Como se puede apreciar en esta figura, los métodos cuyo error se mantiene más del $50\,\%$ del tiempo en el intervalo de error más preciso son *EKF* y *FMK+nEKF*. En este caso hay que volver a tener en cuenta que el método *EKF* se inicia a la posición real del robot y todo lo que se puede decir entonces es que este método realiza el $50\,\%$ se su tiempo un *tracking* casi perfecto. En este intervalo se observa que el porcentaje de tiempo menor es para *FMK*, que sólo estima su posición con gran precisión el $20\,\%$

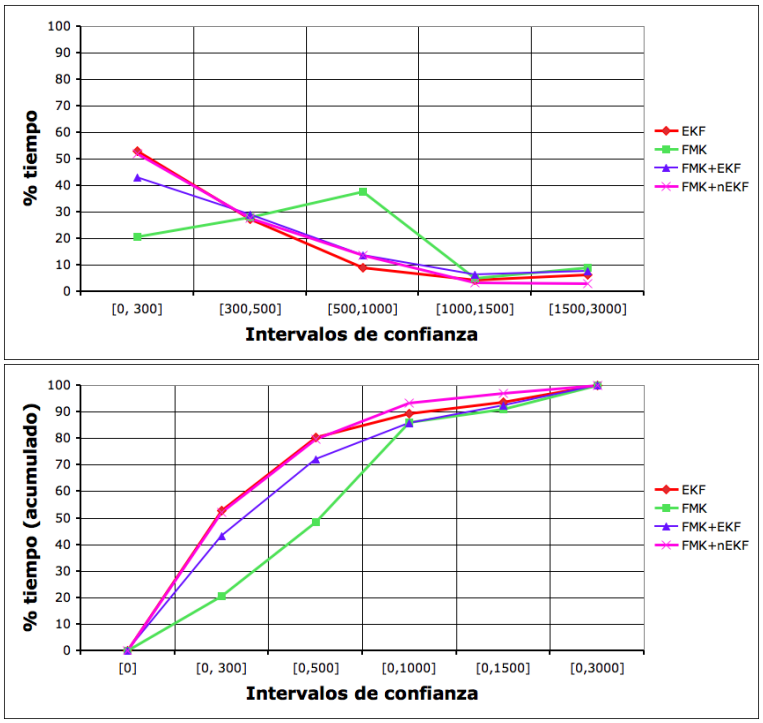


Figura 5.6: Intervalos de confianza del error en el caso de la estimación de la distancia.

del tiempo. En el siguiente intervalo, $[300, 500]$, los métodos coinciden en situar el porcentaje de tiempo que se encuentran en este intervalo alrededor del $27\,\%$. Durante este tiempo se pueden llevar a cabo perfectamente las acciones dependientes de la información de auto-localización.

En el siguiente intervalo, $[500, 1000]$, se observa un dato significativo. En este intervalo de error es en el que se concentra el mayor tiempo de ejecución el método *FMK*. Este intervalo lo interpretamos anteriormente como aquel en el que la estimación sobre la posición del robot permite determinar en qué zona del terreno de juego se encuentra, que es realmente para lo que usamos *FMK* en los métodos combinados, pero no es útil para realizar tareas que requieren gran precisión en la información de auto-localización.

En el caso del error en la estimación de la orientación, hemos definido también 5 intervalos de confianza: $[0, 10]$, $[10, 20]$, $[20, 45]$, $[45, 90]$ y $[90, 180]$, todos ellos refiriéndose a diferencia entre la orientación estimada y la real, en grados. La información

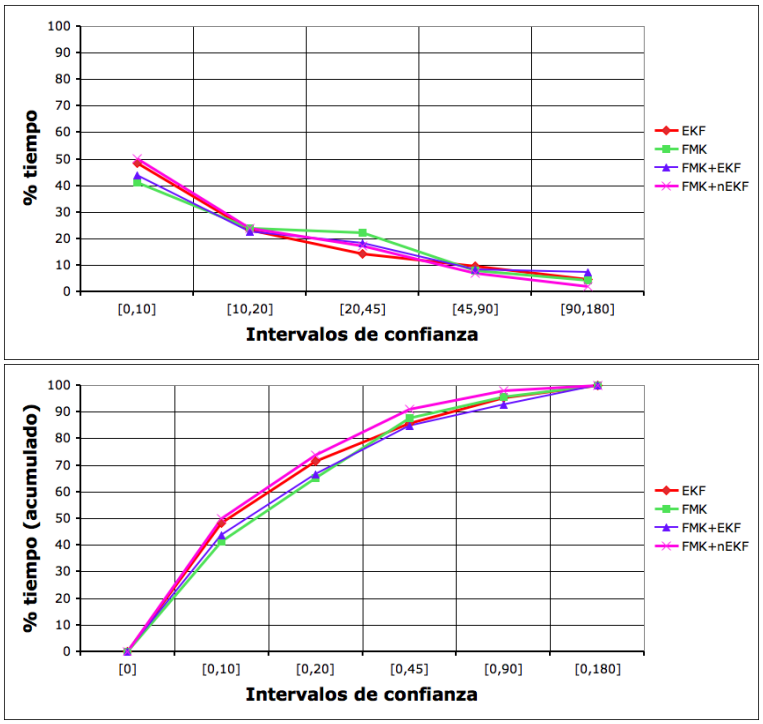


Figura 5.7: Intervalos de confianza del error en el caso de la estimación de la orientación.

de orientación es vital para realizar las acciones dependientes de la información de auto-localización, aunque se pueden realizar correctamente hasta que el error supera los 45 grados. A partir de este umbral, aunque el error en la estimación de la posición sea bajo, el robot tiende a perderse al avanzar debido a que la posición real y la estimada se separan.

La figura 5.7 corresponde al análisis de los intervalos de confianza para el error en la orientación. Tiene un formato similar al de la figura 5.6, sólo que el eje de abscisas corresponde a los intervalos de confianza relativos a la orientación. Esta figura muestra resultados coherentes con la de distancia excepto en el caso de *FMK*, donde su error es similar al de los otros métodos.

## 5.2. Situaciones de secuestro

La capacidad de recuperarse ante situaciones de desplazamiento manual del robot se emplea en los casos en que el robot es sancionado durante el juego. En este caso, el robot se retira manualmente del campo por un árbitro. Tras 30 segundos, el robot es devuelto al campo de juego.

Este experimento se ha diseñado para medir la capacidad de recuperación de cada método de auto-localización ante esta situación de secuestro. Se mide el tiempo que tarda el robot en volver a estimar la posición correcta tras el secuestro y la evolución del error después de haberse recuperado de esta situación.

En la figura 5.8 se muestra el recorrido que se ha diseñado en este experimento. En color rojo se muestra el recorrido que realiza el robot sobre el campo de juego. Después del punto de control número 2, se produce un desplazamiento manual del robot desde el punto ”so” (*secuestro origen*) al punto ”sd” (*secuestro destino*). Este experimento se ha repetido en 28 ocasiones, siendo la duración del mismo de unos 35 a unos 40 segundos.

Al igual que en el experimento anterior, el robot se guía por medio del sistema de localización externo para llevar a cabo el recorrido. Del mismo modo, el robot guarda la información perceptiva y odométrica para posteriormente poder repetir, sin usar el robot real, el experimento con cada uno de los métodos de localización.

En primer lugar, se analiza el error medio de cada uno de los métodos a lo largo de las repeticiones de las que se compone el experimento. Se muestra si los métodos se recuperan del secuestro y cómo evoluciona su error a continuación.

Tras el secuestro, el método *EKF* comienza a rechazar las observaciones que recibe, por no ser coherentes con la estimación que mantiene. Debido a que no incorpora observaciones, únicamente modifica la estimación mediante su modelo de actuación. Esto se debe a que la incertidumbre de la estimación comienza a crecer, debido a que únicamente se realiza su fase de predicción. Cuanto mayor es la incertidumbre, mayor es la probabilidad de que una observación sea considerada coherente y el *EKF* pueda llegar a recuperarse.

La figura 5.9 muestra el error medio en todas repeticiones de la trayectoria en la estimación de la posición (izquierda) y de la orientación (derecha) de las ejecucio-

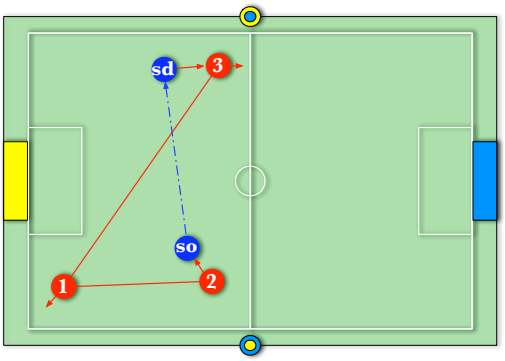


Figura 5.8: Recorrido del experimento con secuestro.

nes de este experimento. En azul se muestra la media del error y las líneas en rojo representan la desviación típica. El secuestro se produce en el instante señalado por la etiqueta ”so” en azul.

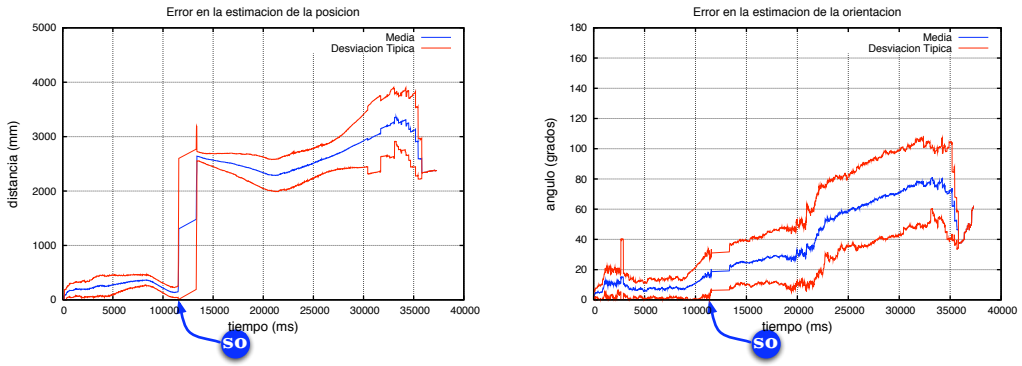


Figura 5.9: Media de error en distancia y orientación entre la estimación en la posición y la posición real del robot para el método *EKF*.

En las gráficas se observa que este método es incapaz de recuperarse ante esta situación de secuestro. Justo antes de producirse el secuestro la incertidumbre en la estimación de la posición del robot suele ser muy baja debido a la continua incor-

poración de percepciones coherentes. También hay que tener en cuenta que al ser la incertidumbre tan baja, el *filtro de incoherencias* es más estricto que en el caso de incertidumbres elevadas. Cuando se produce el secuestro, este *filtro de incoherencias* hace que no se vuelvan a incorporar observaciones hasta que la incertidumbre aumenta. También hay que tener en cuenta que según aumenta la incertidumbre, el *filtro de incoherencias* es menos eficaz y más permisivo, y existe la posibilidad de que acepte observaciones erróneas o falsos positivos. Esto hace que disminuya la incertidumbre en una posición errónea, volviendo a tener el mismo problema que al inicio del secuestro.

El método *FMK* se recupera rápidamente de las situaciones de secuestro. Al incorporar un par de observaciones correctas ya se puede obtener la posición del robot correctamente. Esta es una característica positiva de este método. La parte negativa es que si a continuación se producen varias percepciones erróneas falla en la estimación de la posición del robot. Esta inestabilidad es una de las características que hemos constatado a lo largo de esta experimentación.

La figura 5.10 muestra la evolución del error medio en la estimación de la posición y la orientación del robot en el mismo formato que las anteriores cuando se usa el algoritmo *FMK*. Se observa que tras el secuestro este método es capaz de recuperarse. A lo largo de las repeticiones del experimento hemos comprobado que la percepción de la distancia a las porterías es errónea debido a ciertas condiciones de iluminación. Esto hace que el método *FMK* muestre cierta inestabilidad posterior al secuestro, llegando la media del error a superar el metro de diferencia entre la posición real y la estimada.

El método *FMK+EKF* tiene un comportamiento que trata de sacar partido a las características positivas de los dos métodos anteriormente descritos. El *EKF* comienza a rechazar las observaciones que recibe y su incertidumbre comienza a crecer. Mientras tanto, el *FMK* comienza a aportar hipótesis sobre la posición del robot. Durante los instantes posteriores al secuestro *FMK* indica correctamente la posición del robot y evalúa que la estimación del *EKF* como poco probable, ya que la celda donde se encuentra la estimación del *EKF* tiene una probabilidad baja. Se cumplen entonces dos de las condiciones para reiniciar el *EKF*: la posición que indica *FMK* está a una distancia suficientemente alejada de la estimación de *EKF* y que *FMK* considera

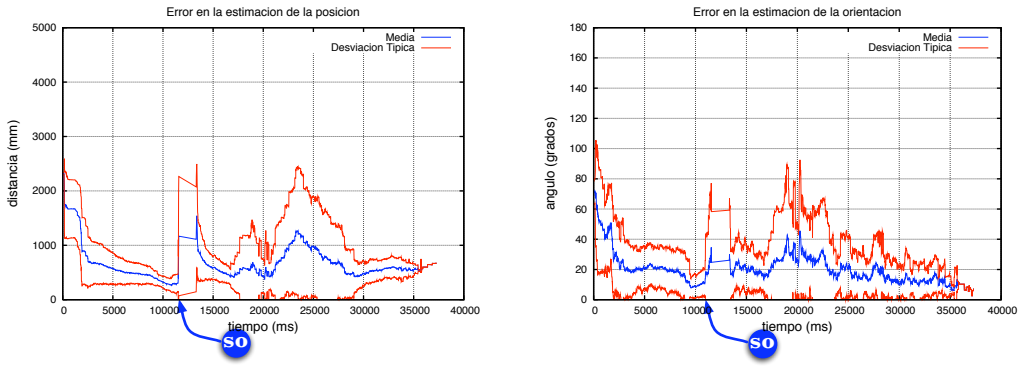


Figura 5.10: Media de error en distancia y orientación entre la estimación en la posición y la posición real del robot para el método *FMK*.

improbable la estimación de *EKF*. Lo único que falta es que la incertidumbre de *EKF* sea suficientemente elevada. En el instante que lo sea y se sigan cumpliendo las otras dos condiciones, el *EKF* se inicia a la posición indicada por *FMK*.

Tras el reinicio, el *EKF* disminuye su incertidumbre con observaciones correctas. Si no consigue disminuir la incertidumbre, bien porque se ha reiniciado erróneamente o porque se producen falsos positivos, puede ocurrir que vuelva a ser reiniciado a otra posición distinta.

Al igual que en la descripción de los métodos anteriores, la gráfica 5.11 muestra el error en la estimación de la posición y la orientación del robot. Se observa claramente lo que acabamos de comentar. La estimación de la posición del robot acaba siendo correcta, pero emplea mucho tiempo en recuperarse, como analizaremos a continuación. Esto es debido a que la incertidumbre del *EKF* tarda tiempo en crecer y, como acabamos de comentar, este es uno de los requisitos para reiniciarlo.

En el caso del método *FMK+nEKF* lo que sucede cuando se produce un secuestro es que el *FMK* aporta rápidamente una hipótesis correcta sobre la posición del robot, donde se inicia un nuevo *EKF*. A partir de ese instante comienzan a competir el *EKF* que mantiene la estimación anterior al secuestro y el nuevo *EKF* que se ha iniciado en la posición correcta. El *EKF* que muestre menor incertidumbre será el que aporte la posición del robot como salida del módulo *GM*. El anterior *EKF* deja de incorporar observaciones y comienza a crecer su incertidumbre. El nuevo *EKF*

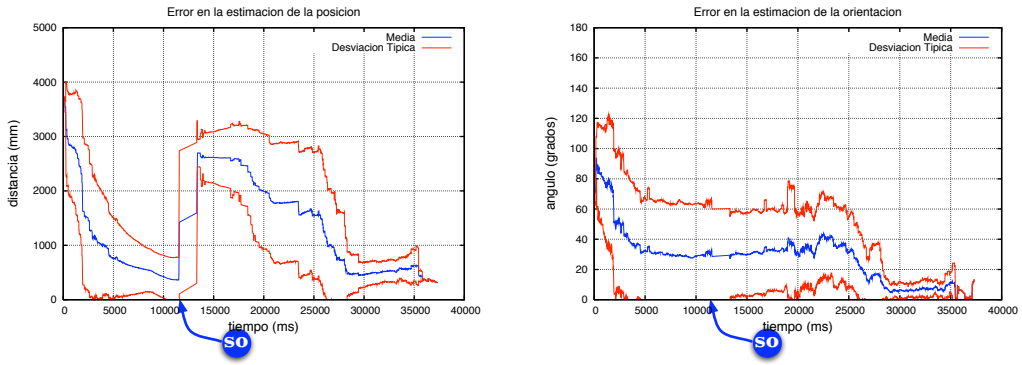


Figura 5.11: Media de error en distancia y orientación entre la estimación en la posición y la posición real del robot para el método *FMK+EKF*.

comienza a incorporar observaciones y en pocos ciclos su incertidumbre será menor que la del anterior *EKF*, recuperándose este método del secuestro.

La figura 5.12 muestra el error medio en las ejecuciones de este experimento con el método *FMK+nEKF*. La recuperación es muy rápida, casi tanto como el método *FMK*. Este tiempo es el que tarda el nuevo *EKF* en disminuir su incertidumbre. A diferencia de *FMK*, el error se mantiene bajo hasta el final del experimento y no se ve tan afectado por los errores en la percepción de las porterías comentados anteriormente. En este caso estos falsos positivos hacen que se reincien nuevos *EKFs* en hipótesis erróneas de *FMK*, pero estos errores perceptivos no son suficientemente persistentes para que estos *EKFs* puedan llegar a resultar ganadores en la competición por el *EKF* con menos incertidumbre.

En la figura 5.13 se representan 4 ejecuciones representativas de este experimento. Estas gráficas muestran el error en la estimación de la posición y orientación del robot a los largo del tiempo. Cada color es un método distinto de localización, como muestra la leyenda de la figura. En esta figura se observan varias cuestiones interesantes. Todos los métodos, salvo *EKF*, terminan estimando correctamente la posición del robot. Otra cuestión que se observa es que el método que más rápidamente se recupera es *FMK*, pero los errores posteriores en la percepción de las porterías hacen que vuelva a perderse por unos instantes. En la figura superior izquierda se observa cómo el error es muy elevado durante gran parte del experimento tras la recuperación.

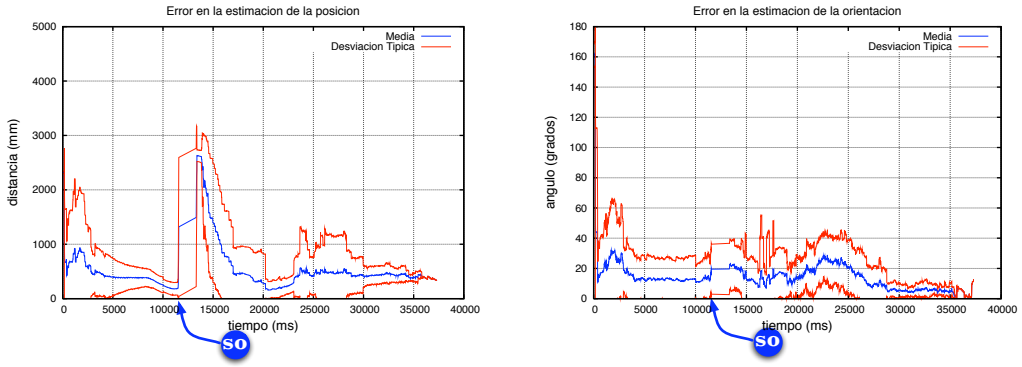


Figura 5.12: Media de error en distancia y orientación entre la estimación en la posición y la posición real del robot para el método *FMK+nEKF*.

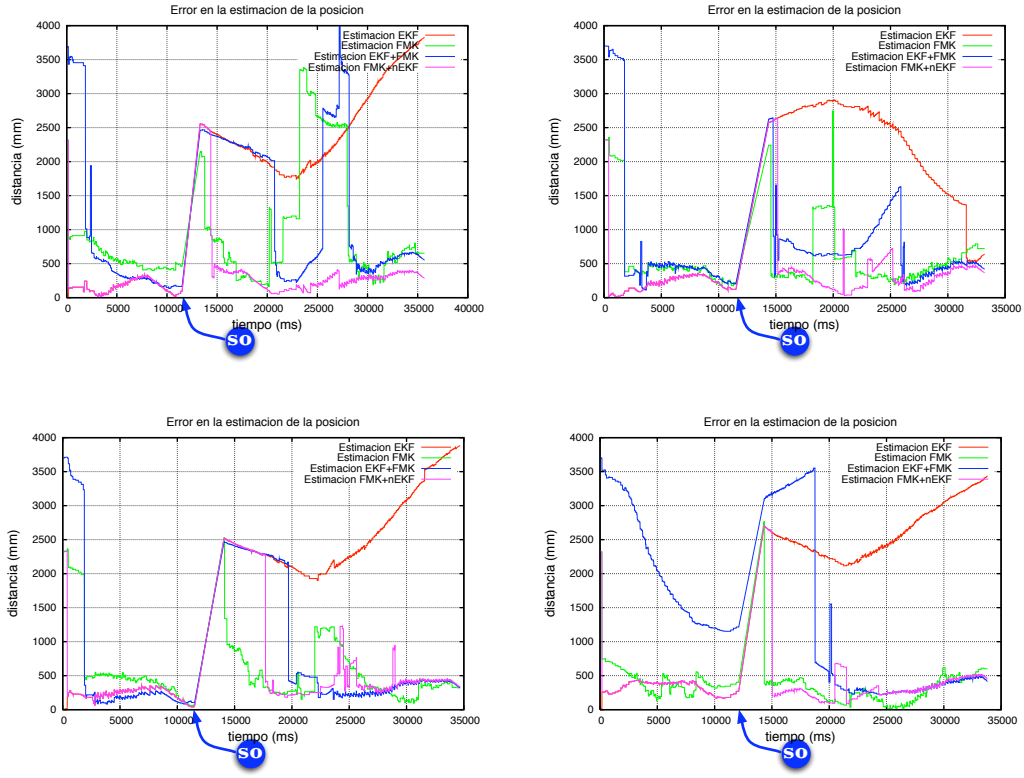


Figura 5.13: Error en distancia entre la estimación en la posición y la posición real del robot.

Cuando el error del método *FMK* no es tan elevado, el método *FMK+EKF* funciona correctamente y de manera estable, incluso mejor que *FMK+nEKF* (gráfica inferior izquierda). Las ejecuciones del método *FMK+nEKF* muestran en todas las gráficas una recuperación rápida y un error posterior estable y bajo. Aún así, los errores en la percepción afectan a la estimación, como se observa en la gráfica inferior izquierda. Esto es debido a que hay instantes en que *EKFs* erróneos momentáneamente tienen una incertidumbre menor que el *EKF* correcto. Al no ser descartado inmediantamente este *EKF* correcto, vuelve a disminuir su incertidumbre, recuperándose de nuevo. Aun así, los *EKFs* erróneos no están a gran distancia de la posición real del robot y ésta es la razón por la que pequeños errores de estimación de distancias a las porterías puedan hacer disminuir más la incertidumbre en *EKFs* incorrectos que en el correcto.

Otro de los aspectos que es interesante medir es el tiempo de recuperación tras el desplazamiento. Los tiempos de recuperación se han medido usando el error en la estimación de la posición. Cuando el robot se secuestra manualmente el error aumenta hasta los dos metros, que es la distancia en la que se deposita de nuevo el robot. El tiempo entre que es depositado en el suelo hasta que el error de la estimación vuelve a ser bajo (lo consideramos por debajo de 500 mm) es lo que llamamos *tiempo de recuperación*.

En la tabla 5.3 se pueden ver los tiempos de recuperación de cada uno de los métodos. En esta tabla también se refleja el hecho de que en los experimentos con el método *EKF* nunca llega a recuperarse del secuestro.

Los tiempos de recuperación son similares en *FMK* y *FMK+nEKF*, siendo un segundo y medio y dos segundos, respectivamente. El método *FMK+EKF* tiene una media de recuperación de casi nueve segundos, lo que es una cifra demasiado elevada. La razón de este tiempo de recuperación tan elevado es que tardan en darse las condiciones adecuadas para el reinicio del *EKF*.

Como comentamos al inicio de la sección, también queremos evaluar la estabilidad de los métodos de localización tras haberse recuperado del secuestro. Esto es interesante porque a veces sucede que la recuperación no es permanente y puede volver a una estimación errónea. En el caso del método *FMK+nEKF* podría ser perfectamente posible que el *EKF* anterior siguiera siendo la hipótesis con menor incertidumbre, aunque esto no suele suceder, como se ha observado anteriormente al

| Estadístico | *EKF* | *FMK* | *FMK+EKF* | *FMK+nEKF* |
|---|---|---|---|---|
| **Media** | - | **1685** | **8711** | **2184** |
| Stdev | - | 1763 | 7483 | 1659 |
| Mínimo | - | 1 | 496 | 532 |
| Máximo | - | 7040 | 15680 | 6943 |
| % de recuperación | 0 % | 100 % | 100 % | 100 % |

Tabla 5.3: Tiempos de recuperación (milisegundos)

| Estadístico | *FMK* | *FMK+EKF* | *FMK+nEKF* |
|---|---|---|---|
| Primer Cuartil | 272.0901 | 256.1640 | 193.1269 |
| Mediana | 446.1491 | 453.5119 | 318.7507 |
| Tercer Cuartil | 647.2442 | 693.3578 | 451.4300 |
| **Media** | **638.0704** | **692.5487** | **382.2962** |
| Desviación típica | 636.3381 | 725.1610 | 372.0771 |

Tabla 5.4: Análisis estadístico del error en distancia de la estimación de la posición del robot (mm).

presentar los resultados. Aún así, es importante comprobar numéricamente qué método tiene un error menor tras la recuperación. Con este fin se presentan las tablas 5.4 y 5.5 para el error en la estimación de la posición y el ángulo, respectivamente. En estas tablas se observa que el error de *FMK* y *FMK+EKF* es similar y el error de *FMK+nEKF* es casi la mitad que en los otros casos. Estos valores concuerdan con las gráficas mostradas anteriormente.

| Estadístico | *FMK* | *FMK+EKF* | *FMK+nEKF* |
|---|---|---|---|
| Primer Cuartil | 6.135 | 5.809 | 3.557 |
| Mediana | 13.13 | 14.39 | 8.003 |
| Tercer Cuartil | 25.44 | 36.37 | 18.99 |
| **Media** | **19.8** | **24.5** | **13.93** |
| Desviación típica | 22.23 | 25.36 | 15.43 |

Tabla 5.5: Análisis estadístico del error en orientación de la estimación de la posición del robot (grados).

## 5.3. Recorrido por una secuencia de puntos de control

En esta sección queremos evaluar el rendimiento de los diferentes métodos en las tareas de navegación global. Esta capacidad se emplea cuando el robot tiene activos los roles de *supporter* o *defender* para desplazarse a la posición calculada como óptima dependiendo de la posición de la pelota y de los roles que los otros robots tienen activos.

| Punto de control | $x$ | $y$ |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 0 | 1500 |
| 3 | 0 | -1500 |
| 4 | 800 | 400 |
| 5 | -800 | 400 |
| 6 | 800 | -400 |
| 7 | 800 | 400 |
| 8 | 1500 | 2000 |
| 9 | -1500 | -2000 |
| 10 | 0 | -2000 |
| 11 | 0 | 2000 |
| 12 | 0 | 0 |

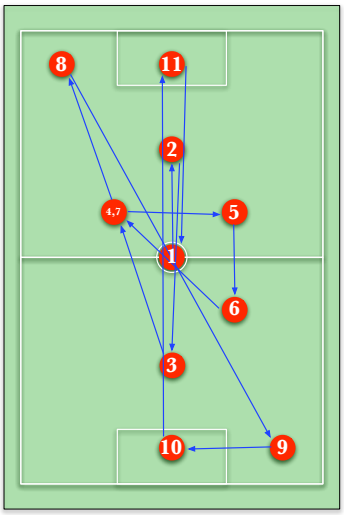


Figura 5.14: Puntos de control

Los experimentos anteriores se han llevado a cabo usando un sistema de localización externo. En este experimento no se usa el sistema de localización externo para guiar al robot a lo largo de los puntos de control, sino que utiliza las percepciones de los elementos del entorno para localizarse. De esta manera medimos si la estimación que aporta cada método de localización permite llevar a cabo la navegación. Aún así, usamos el sistema de ”verdad absoluta” para medir los errores en la localización.

En este experimento el robot sigue el recorrido mostrado en la figura 5.14. El robot realiza el recorrido siguiendo los puntos de control etiquetados incrementalmente. Comienza en el punto de control 1, luego va al 2,3,4... y así hasta volver al 1. El robot considera que ha alcanzado un punto de control cuando su distancia estimada al

mismo es inferior a 300 mm. Cuando esto sucede el robot realiza un corto baile para indicar que ha llegado a un punto y continúa con el siguiente.

Se ha repetido el recorrido 14 veces, con el objetivo de minizar la influencia de errores perceptivos puntuales, para que no afecten a las conclusiones que se puedan extraer del experimento.

La duración de los experimentos varía de uno a otro. Se considera finalizado el experimento cuando ha llegado al último punto de control. Durante el recorrido el robot puede perderse y volver a recuperarse varias veces. Debido a esto la duración de cada repetición del experimento no es la misma, siendo en todos los casos superior a 3 minutos e inferior a 5 minutos.

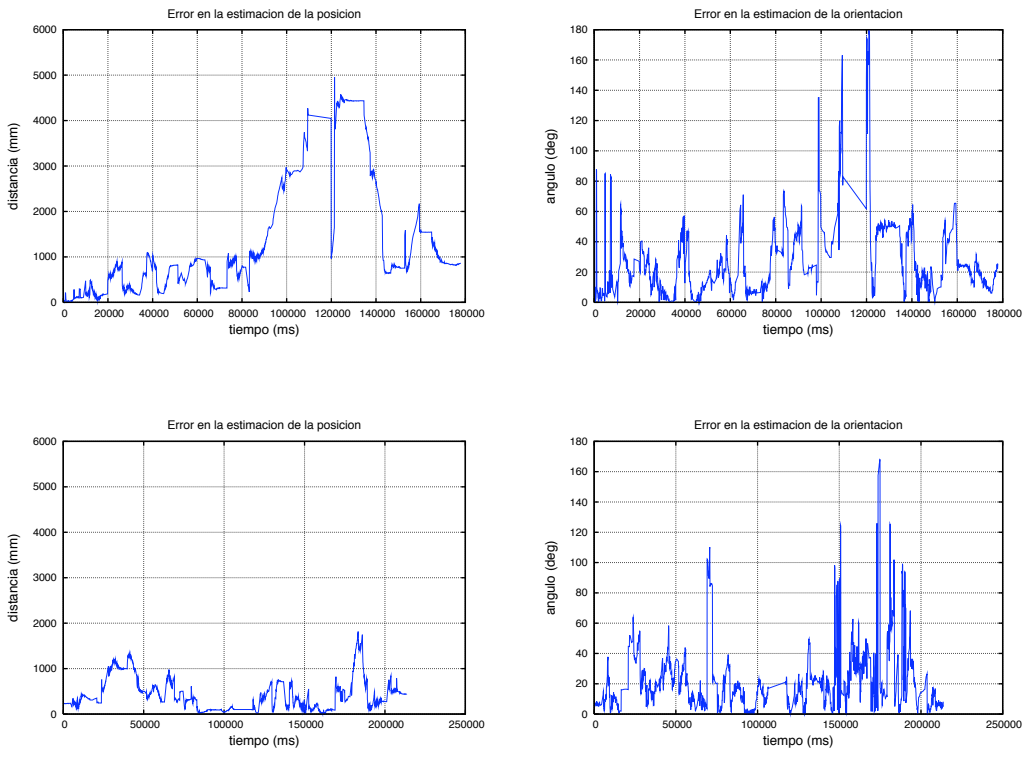


Figura 5.15: Error en distancia y en orientación del método *EKF*.

En la figura 5.15 se muestra la evolución del error de la estimación en el tiempo de dos ejecuciones mientras se sigue la ruta usando el método *EKF* para auto-

localizarse. La fila superior corresponde a un experimento y la inferior a otro. La columna de la izquierda es el error al estimar la posición del robot y la de la derecha es el error al estimar la orientación.

En la gráfica superior se observa que el error en la estimación de la posición es bajo hasta el segundo 85 aproximadamente. Hasta este instante el robot sigue la ruta perfectamente. A partir de este instante la percepción de una portería en un lugar equivocado hace que la estimación se desvíe de la posición real del robot. Esto hace que el robot no incorpore percepciones correctas y actualice su estimación únicamente con la odometría. A consecuencia de esto, se detectan los puntos de control en posiciones equivocadas y termina su recorrido por los puntos de control sin haber realizado correctamente la trayectoria.

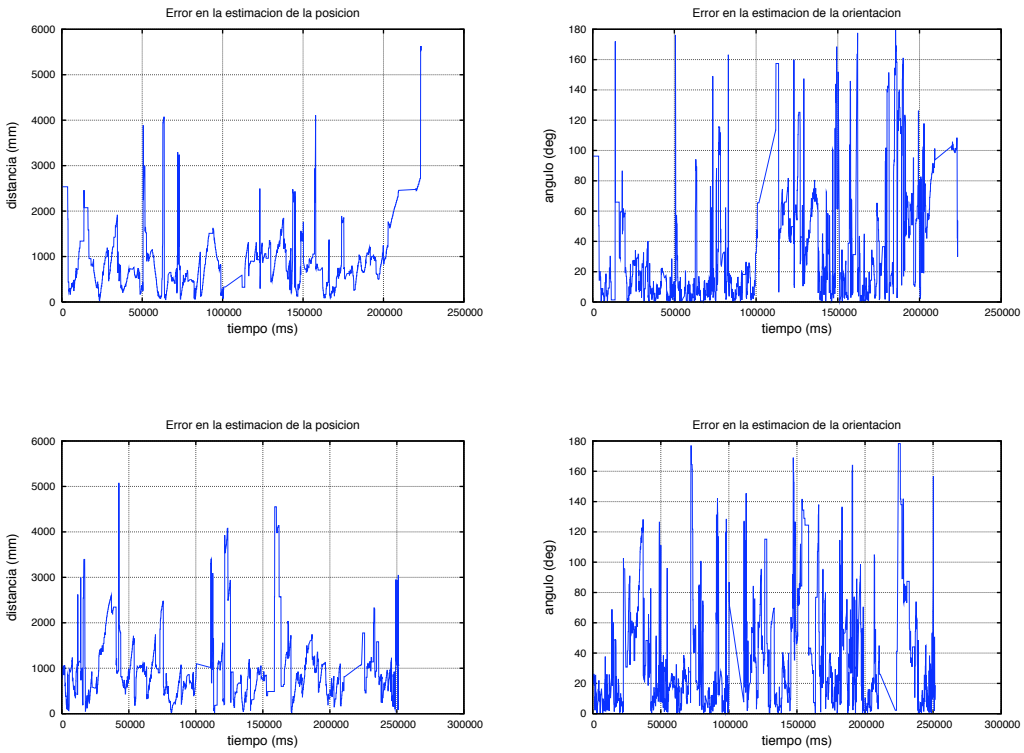


Figura 5.16: Error en distancia y en orientación del método *FMK*.

En la fila inferior se muestra otra ejecución en la que el error en la estimación de la posición es bajo. El robot consigue hacer toda la ruta de manera correcta. Durante esta ejecución no hay errores perceptivos que el sistema no pueda rechazar y que hagan divergir la estimación. Esta ejecución demuestra que si se rechazan correctamente las percepciones erróneas y el robot no se ve desplazado, ya sea manualmente o por otro robot, este método puede hacer que el robot realice tareas de navegación de manera precisa.

En la figura 5.16 se muestran las gráficas de evolución del error en dos repeticiones cuando se usa el método *FMK*. Al igual que en el análisis anterior, la fila superior corresponde a un experimento y la inferior a otro. La columna de la izquierda es el error al estimar la posición del robot y la de la derecha es el error al estimar la orientación.

Cuando el robot usa este método se observa que el robot se pierde y se recupera contínuamente. Se debe a que incorpora todas las percepciones, sean correctas o no. Esto hace que las celdas tengan probabilidades similares y que la estimación a menuda caiga en el centro del campo. Cuando esto sucede el robot detiene su marcha y gira para encarar la dirección que el método de auto-localización le marca como correcta para seguir la ruta. Al girarse, el robot suele volver a obtener percepciones correctas y se recupera otra vez. Además, hay varias ocasiones en las que el robot percibe erróneamente la llegada a varios de los puntos de control.

Las gráficas muestran que el error aumenta y disminuye continuamente, teniendo máximos en los instantes en que se producen falsos positivos, ya que este método no establece ningún criterio para rechazarlos.

El método *FMK+EKF* se ha mostrado en estos experimentos adecuado para tareas de navegación. El robot sigue la ruta correctamente la mayor parte del tiempo. Hay ocasiones en las que que el error en la estimación aumenta debido a que se ve afectado por errores en la actuación e imprecisiones en la percepción. En este momento el robot se desvía de la ruta. Si el *EKF* no recibe percepciones correctas para corregir la estimación, en unos segundos se reinicia y vuelve a retomar la ruta.

Las gráficas del error en dos de las ejecuciones se muestran en la figura 5.17. La diferencia principal con respecto al método *FMK* es que es mucho más estable. El error en posición se mantiene la mayor parte del tiempo por debajo de un metro.

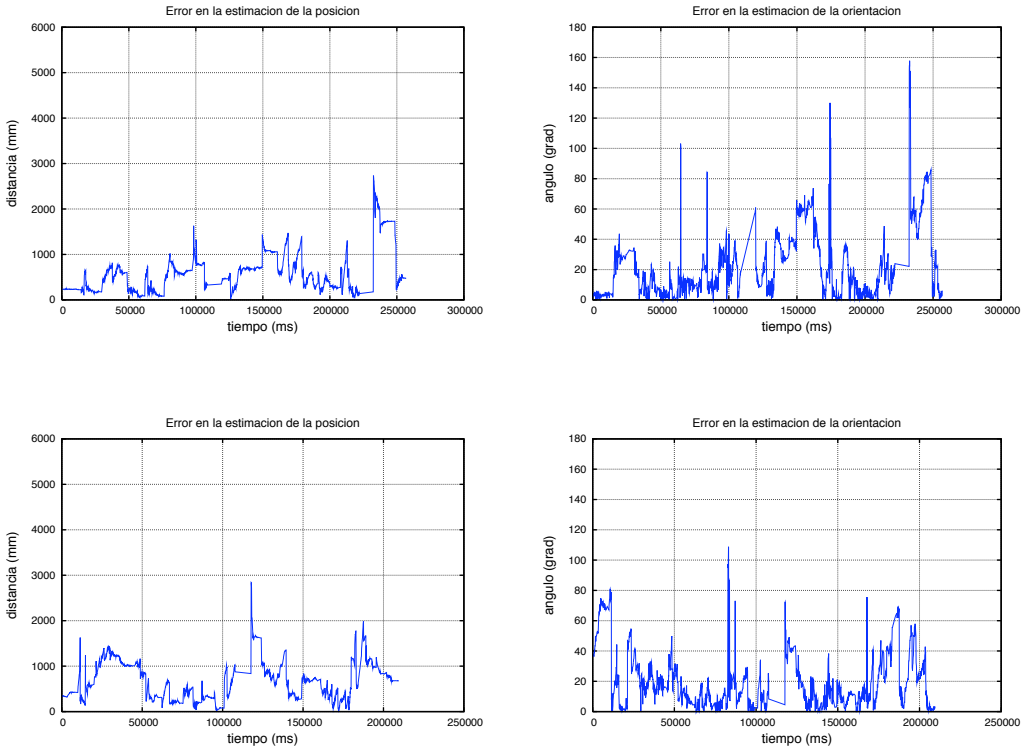


Figura 5.17: Error en distancia y en orientación del método *FMK+EKF*.

Para seguir la ruta correctamente es importante que el error en la estimación de la orientación sea bajo, como lo es en la mayor parte del experimento.

Cuando el robot usa método *FMK+nEKF* para seguir la ruta se obtienen los mejores resultados. El robot sigue la ruta con mayor precisión que los métodos mostrados anteriormente y se puede observar que el número de ocasiones que visiblemente pierde la posición correcta del robot es muy bajo. Debido a esto, es el método que menos tarda en completar el recorrido, no superando los tres minutos y medio en prácticamente ninguna ejecución.

Las gráficas correspondientes al error en dos de las ejecuciones de este experimento se muestran en la figura 5.18. El error tanto en posición como en orientación es menor que en las ejecuciones con los métodos anteriores.

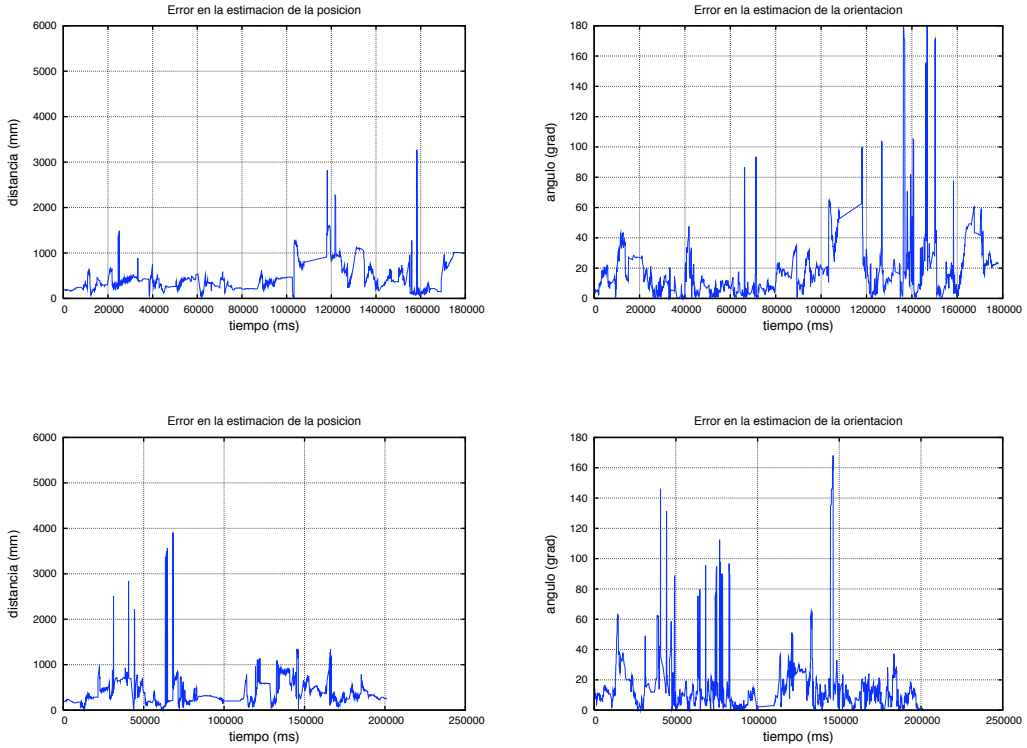


Figura 5.18: Error en distancia y en orientación del método *FMK+nEKF*.

| Estadístico | *EKF* | *FMK* | *FMK+EKF* | *FMK+nEKF* |
|---|---|---|---|---|
| **Media** | **208.6** | **253.3** | **246.6** | **205.9** |
| Desviación típica | 19.6 | 21.4 | 23.4 | 20.3 |
| Mínimo | 177.9 | 223.4 | 209.5 | 178.1 |
| Máximo | 232.3 | 281.2 | 272.7 | 234.5 |

Tabla 5.6: Análisis estadístico del tiempo empleado en completar el recorrido (segundos).

| Estadístico | *EKF* | *FMK* | *FMK+EKF* | *FMK+nEKF* |
|---|---|---|---|---|
| Primer Cuartil | 263.23 | 476.2 | 352.65 | 218.44 |
| Mediana | 487.4 | 749.1 | 588.91 | 352.7 |
| Tercer Cuartil | 844.56 | 1131.3 | 1012.2 | 554.06 |
| **Media** | **785.13** | **1002.9** | **742.22** | **484.72** |
| Desviación típica | 874.56 | 924.83 | 548.5 | 462.92 |

Tabla 5.7: Análisis estadístico del error en distancia de la estimación de la posición del robot (mm).

Puesto que el comportamiento básico de navegación que ejecuta el robot es el mismo a lo largo de todas las ejecuciones de este experimento, merece la pena mostrar un análisis del tiempo que el robot emplea en completar el recorrido. Emplear menos tiempo en completar el recorrido significa que el robot está menos tiempo perdido, aunque hay que tener en cuenta que en algún caso el robot se ha saltado puntos de control.

En la tabla 5.6 se muestra un resumen el tiempo, en segundos, que emplea el robot usando cada uno de los métodos analizados. Se observa que el método que me-

| Estadístico | *EKF* | *FMK* | *FMK+EKF* | *FMK+nEKF* |
|---|---|---|---|---|
| Primer Cuartil | 6.3989 | 8.3738 | 7.543 | 6.2133 |
| Mediana | 15.197 | 21.189 | 19.044 | 12.883 |
| Tercer Cuartil | 29.902 | 55.366 | 40.1 | 25.521 |
| **Media** | **22.778** | **38.335** | **30.473** | **22.37** |
| Desviación típica | 23.982 | 42.032 | 33.417 | 27.723 |

Tabla 5.8: Análisis estadístico del error en orientación de la estimación de la posición del robot (grados).

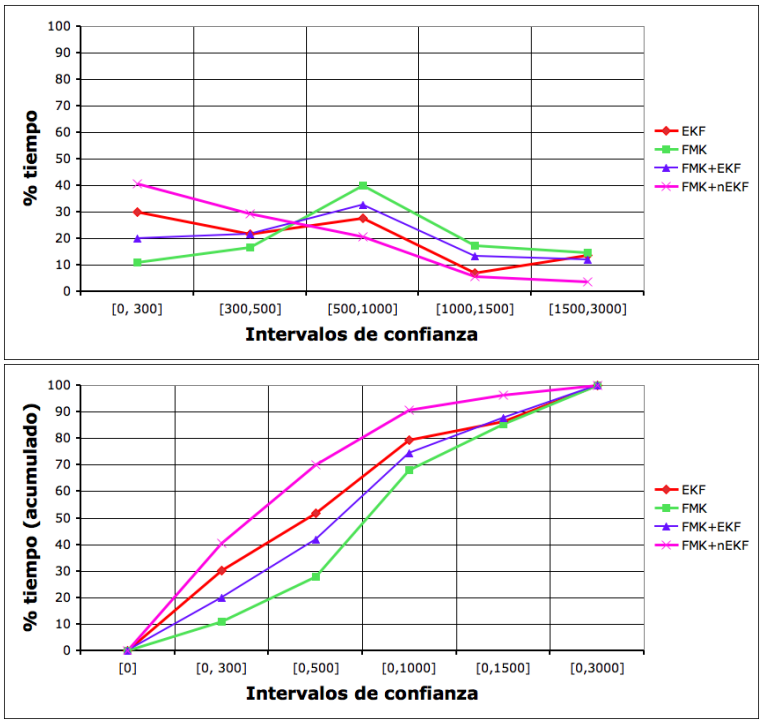


Figura 5.19: Intervalos de confianza del error en el caso de la estimación de la distancia.

nos tiempo emplea, en media, es el *FMK+nEKF*. En los experimentos se comprueba visualmente que esto es así, como se ha explicado anteriormente. Con el método que el robot emplea más tiempo en completar el recorrido es *FMK*, llegando a emplear hasta 281 segundos en completarlo en el peor de los casos.

Durante los recorridos realizados en este experimento se ha realizado un análisis del error que se muestran en las tablas 5.7, para el caso del error en la estimación de la posición, y 5.8, para el caso del error en la estimación de la orientación. Se observa, por ejemplo, cómo la media del error en el método *FMK* es de un metro. En cambio, el error en los métodos combinados en sensiblemente menor, llegando a ser de menos de medio metro en el mejor de los casos, que corresponde al método *FMK+nEKF*.

En la figura 5.19 se representan los resultados del análisis del error en la estimación de la posición usando intervalos de confianza. La gráfica superior muestra los resultados por cada intervalo. En el eje de abscisas se representan los 5 intervalos de confianza en que hemos los posibles valores de error: $[0, 300]$, $[300, 500]$,

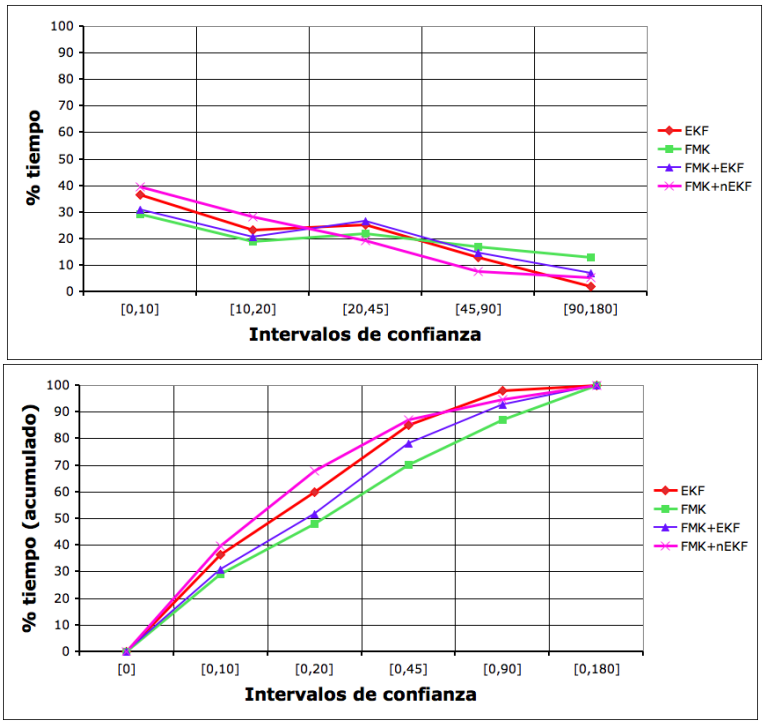


Figura 5.20: Intervalos de confianza del error en el caso de la estimación de la orientación.

$[500, 1000]$, $[1000, 1500]$ y $[1500, 3000]$, todos ellos refiriéndose a distancia euclídea entre la posición estimada y la real, en milímetros. En la figura inferior se muestran los porcentajes acumulados, de modo que se representan los intervalos $[0, 0]$, $[0, 300]$, $[0, 500]$, $[0, 1000]$, $[0, 1500]$ y $[0, 3000]$.

En estas gráficas se observa una distribución del error diferente a la del experimento descrito en la sección 5.1. Una de las diferencias principales es que en este caso la duración es mayor al haber más puntos de control, y esto tiene un impacto negativo sobre todo en el método *EKF*. Esto se debe a que hay más situaciones donde el robot se puede perder y es complicada su recuperación.

Como se observa en esta figura, el valor de *EKF* para el intervalo $[0, 300]$ es menor en este experimento debido a que los instantes de menor error corresponden a los momentos iniciales del experimento. Aún así, hasta casi la mitad del tiempo del experimento el robot consigue estimar correctamente la posición del robot, antes

de perderse. A partir de este momento, como hemos comentado anteriormente, no es capaz de recuperarse.

Sin duda el método que mejor se ha comportado en el experimento es *FMK+nEKF*, siendo el error menor a 500 mm el $70\,\%$ del tiempo. El método *FMK* no presenta tan buenos resultados. Casi el $70\,\%$ del tiempo estima correctamente en qué zona del campo está (su error en inferior a un 1000 mm.) y, como se puede observa en la figura 5.20, acierta en la estimación de su orientación, pero sólo estima con cierta precisión (error inferior a 500 mm.) su posición menos del $30\,\%$ del tiempo.

## 5.4. Análisis de situaciones de juego

Tras realizar experimentos unitarios sobre situaciones de navegación y secuestro, en esta sección se resumen la experimentación realizada durante situaciones reales de juego. En este experimento el robot se encuentra en una fase normal de juego, tomando distintos roles según el mecanismo de intercambio de roles descrito anteriormente. Esta es la situación real donde los métodos de auto-localización se van a aplicar.

La utilidad de este experimento es ver cómo evoluciona el error en la estimación dependiendo del rol activo en cada momento y observar cómo se comporta cada método en situaciones de juego reales en las que el robot se ve afectado por la existencia de varios robots alrededor. Se espera que los resultados varíen según el rol, ya que los roles especifican los elementos a los que se debe prestar atención visual. Cuando el robot tiene el rol *kicker* observará con más atención la pelota y menos el resto de las marcas visuales que con los otros roles.

Además, se comprobará qué métodos son más robustos a choques, oclusiones y falsos positivos producidos por otros robots. Ésta es una diferencia fundamental con respecto a los experimentos anteriores, donde no estaban presentes estos factores habituales durante un encuentro de la RoboCup.

En este experimento no se ha usado el software de reproducción para ejecutar los distintos métodos de auto-localización, ya que en él no se ha implementado el sistema de roles. Además, el rol que se adopta en cada momento depende del rol del resto de los robot y de su posición relativa a la pelota. Sí hemos utilizado este software para comprobar qué eventos perceptivos han sucedido en cada momento y

así poder explicar el comportamiento de los métodos de auto-localización durante el experimento. Esto lo hemos llevado a cabo registrando en cada momento la posición real del robot, con el sistema de ”verdad absoluta” que hemos desarrollado, y los elementos que percibe.

Se realizarán 5 fases de juego de alrededor de 5 minutos por cada método de auto-localización. Hay que tener en cuenta que es complicado que situaciones que se han producido en una ejecución (colisiones, oclusiones, etc.) se repitan en otra ejecución. Para minimizar el efecto que puedan tener situaciones esporádicas y difícilmente repetibles, los tiempos que se han ejecutado cada uno de los métodos suman alrededor de veinte minutos.

### 5.4.1. Situaciones reales de juego empleando el método *EKF*

En primer lugar analizaremos las estimaciones que realiza el método *EKF* en situaciones de juego real. En la figura 5.21 están representados tres periodos de juego de 3-5 minutos de duración. En las gráficas se representa el error en la estimación de la posición (columna de la izquierda) y de orientación (columna de la derecha). Para diferenciar el error en la estimación de los métodos de auto-localización cuando el robot tiene activo un rol u otro, se han coloreado de diferente manera dependiendo del rol. Cuando el rol activo es *kicker*, la porción de gráfica es de color azul. Cuando el robot activa el rol de *defender* es rojo y el rol de *supporter* verde.

En las gráficas se puede observar que hay porciones que no tienen información. Esto se debe a que el robot almacena en memoria la información perceptiva y odométrica y cada 100 segundos la almacena en el fichero de ”log”. Este proceso de almacenamiento dura alrededor de 10 segundos, en la que el robot se queda completamente inmóvil.

El análisis de las gráficas para el método *EKF* nos ofrece algunas conclusiones. La primera es que el método puede mantener una estimación de la posición del robot basándose principalmente la actuación si ésta ha sido caracterizada correctamente, como se ha hecho aquí. Las percepciones permiten corregir las posibles desviaciones que se puedan producir. Además, las percepciones correctas reducen la incertidumbre de la estimación. Además, el *filtro de incoherencias* hace que el método sea menos

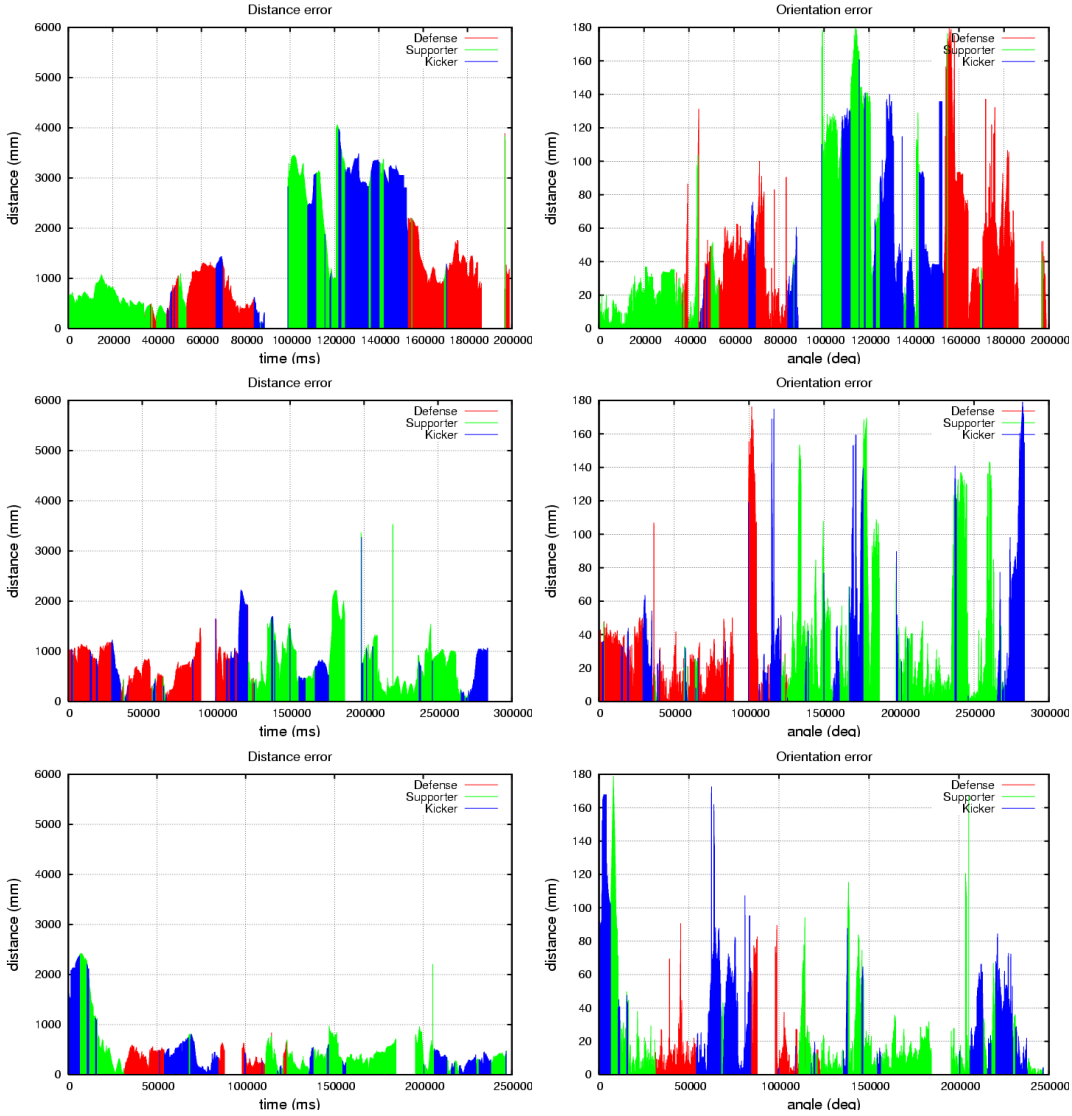


Figura 5.21: Error en distancia y en orientación del método *EKF*.

sensible a situaciones de falsos positivos. Esto implica que este método no sea muy sensible al rol que está activo en cada momento.

La situación que hace que el método falle se produce cuando la orientación se ve modificada por agentes externos, por ejemplo, al chocar con otro robot disputando la bola. A partir de este instante, si el robot no recibe percepciones inmediatamente para corregir esta situación, la estimación comienza a divergir de la posición correcta.

Cuando el error en distancia es ya muy elevado, comienza a rechazar incluso las percepciones correctas que podrían hacer que la estimación se recuperase.

En la porción de juego de la gráfica superior ha sucedido esta situación. El robot se encuentra razonablemente localizado, siendo su error inferior a un metro. La tarea de *supporter* la realiza correctamente, yendo a posición de apoyo al *kicker*. A continuación tiene una fase, de la marca de tiempo 50.000 a la 85.000, en la que el robot realiza tareas defensivas y llega a despejar la pelota. Analizando los datos se puede observar que durante la ejecución de esta tarea el robot choca ligeramente con otro robot y se produce también un falso positivo, al confundir una portería azul con la parte azul de una de las balizas. Aún así, percepciones correctas posteriores hacen que la estimación se corrija. Hasta ese instante la estimación de la posición del robot es aceptable. En el instante 85.000 el robot activa el rol *kicker* y avanza hacia la pelota. En este avance choca con un robot del equipo contrario, que le impide el avance y le hace girar. A partir del choque la estimación no se puede recuperar y el error se mantiene elevado el resto de esta fase de juego.

En el resto de las fases de juego mostradas en este figura, el error se muestra bajo y no parece que la activación de un rol u otro influya significativamente en el aumento del error en la estimación a raíz de la diferente atención que se presta a los elementos del juego (pelota en el caso de *kicker* y todos los elementos en el caso de los demás roles). La única consideración es la de que es más probable que cuando vaya a por la pelota choque con un robot del otro equipo que también quiera hacerse con el control de la pelota. Aún así, durante estas fases, aunque existen colisiones, éstas no son muy persistentes y el robot se recupera inmediatamente con percepciones correctas.

En la secuencia de juego representada por la segunda fila se producen dos situaciones en las que la posición estimada se separa de la posición real del robot. La primera es en el instante 120.000, donde el robot choca con un robot para ir a por la pelota estando en el rol *kicker*. Esto hace que la estimación diverja hasta que se activa el rol defensa. Inmediatamente vuelve a prestar atención a las marcas visuales y consigue corregir su posición. La segunda situación se produce cuando el robot tiene activo el rol de *supporter* y se encuentra cerca de la portería contraria. Desde esta posición, el robot no percibe las balizas del medio campo durante un periodo de tiempo significativo, lo que hace que su incertidumbre crezca. Las percepciones de

la portería que tiene a su lado se rechazan porque ésta es percibida parcialmente y la distancia que se calcula a ella es mayor de lo que realmente es. Llega un instante que la incertidumbre crece tanto debido al movimiento que la percepción errónea de la portería se acepta como válida y la estimación comienza a divergir de la posición real del robot.

En resumen, usando *EKF* en situaciones de juego real se observa varios resultados:

- El robot estima correctamente su posición y la estimación no se deteriora significativamente aunque no se encuentre observando marcas visuales, como es el caso de cuando tiene activo el rol *kicker*.

- Cuando el robot ha de desplazarse a una posición del campo de juego suele hacerlo rápidamente, ya que la estimación se mantiene estable durante el desplazamiento.

- El sistema es capaz de descartar las percepciones erróneas la mayoría de las ocasiones en que se producen si a su vez obtiene percepciones correctas.

- El método es muy sensible a situaciones de colisión con otros robots que hacen cambiar su orientación. Si no obtiene percepciones posteriores que corrijan la estimación, aumenta el error en la estimación y termina perdiéndose.

- Cuando el robot estima erróneamente su posición y el error supera los dos metros, es complicado que vuelva a recuperarse. Cuando esto sucede es frecuente ver cómo el robot abandona el campo o choca con el fondo de las porterías o con las balizas.

- Por último, comprobamos que este método este no es adecuado para situaciones reales de juego debido a su escasa capacidad de recuperación. Durante un partido de la RoboCup no es aceptable que el robot cuando se pierda, debido a errores perceptivos o a oclusiones, no se vuelva a recuperar.

### 5.4.2. Situaciones reales de juego empleando el método *FMK*

En la gráfica 5.22 se muestra la evolución del error en tres situaciones de juego cuando el robot usa el método *FMK*. Durante las situaciones de juego se observa que la estimación es muy inestable. Cualquier falso positivo hace que todas las celdas de la rejilla tomen casi la misma probabilidad y entonces la estimación de la posición del robot se calcule como el centro del campo. Lo positivo es que cuando deja de recibir percepciones erróneas y recibe varias correctas se vuelve a recuperar, estimando correctamente la posición del robot con bastante precisión, es decir, no se producen situaciones de pérdida absoluta (abandono del campo p.e.).

En la gráfica superior se muestra la primera situación de juego a analizar. Dura aproximadamente tres minutos y durante la misma el robot se pierde y se recupera varias veces. Nada más comenzar, estando el robot en su campo cerca de una baliza, orientado hacia la portería contraria y con el rol *defender* activo, el robot percibe momentáneamente una baliza y, por error, también la portería contraria. Al alejarse de la baliza para ir otra vez a la posición defensiva, deja de observar este falso positivo y se recupera inmediatamente.

Posteriormente, activa el rol de *kicker* y vuelve a aumentar el error. Esto se produce porque deja de prestar atención a las marcas visuales, salvo esporádicamente a la portería, y la odometría hace que la probabilidad se reparta rápidamente entre las celdas. Existen situaciones en las que, además, el robot se encuentra muy cerca de la portería y no la observa completamente. Esto hace que la estimación de la distancia a la portería sea errónea. Es complicado que el robot recupere la estimación correcta de su posición hasta que active otro rol y preste atención a otros elementos.

En resumen, durante las ejecuciones de este experimento se observa lo siguiente:

- Cuando el robot percibe varios falsos positivos, o tiene activo el rol de *kicker*, el método suele fallar al estimar la posición del robot.
- Se recupera rápidamente cuando recibe dos o más percepciones correctas.
- Salvo cuando tiene activo el rol *kicker* durante un extenso periodo de tiempo las situaciones en que el error es alto son momentáneas. No se producen situaciones en que el robot sale del campo de juego de manera permanente.

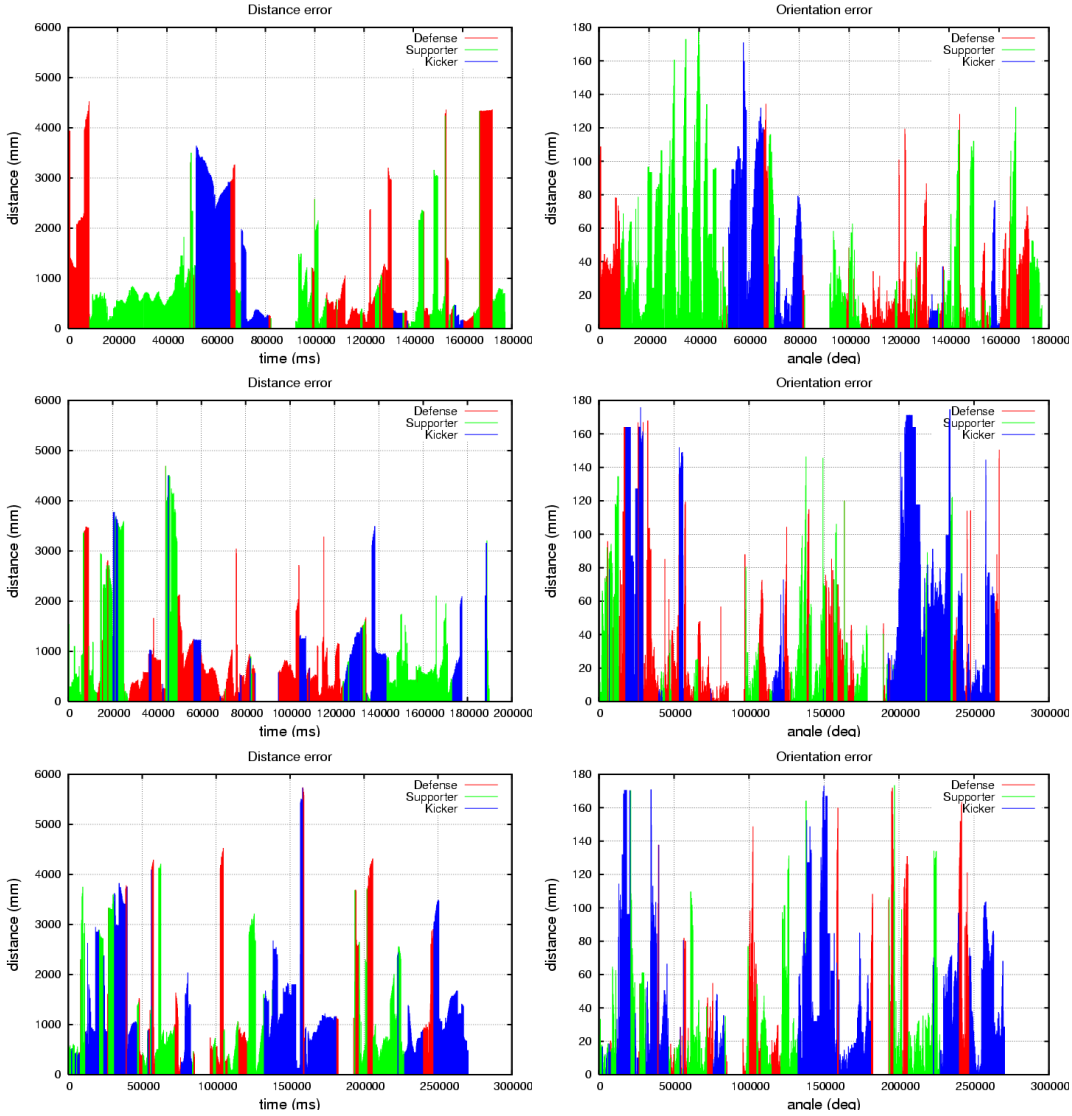


Figura 5.22: Error en distancia y en orientación del método *FMK*.

- El robot encuentra dificultades para alcanzar la posición idónea desarrollando los roles *supporter* o *defender*. Suele ir poco a poco y se observa cómo gira constantemente, ya que no se mantiene estable su estimación. Esto se produce debido a la inestabilidad del método ante errores perceptivos que son relativamente frecuentes.

- Las situaciones de colisión no suponen la causa de errores prolongados en la estimación de la posición.

### 5.4.3. Situaciones reales de juego empleando el método *FMK+EKF*

La gráfica 5.23 muestra varias situaciones de juego cuando el robot usa el método *FMK+EKF*.

La gráfica superior muestra cómo la estimación es correcta y estable durante la práctica totalidad del primer minuto, activando los roles de *kicker* y *supporter* alternativamente. Durante este periodo, la estimación de la posición del robot la calculaba un mismo *EKF* y no es necesario usar la información de *FMK* para corregir, salvo al comienzo de la ejecución para iniciar por primera vez el *EKF*. Durante este espacio de tiempo se producen errores perceptivos que son rechazados adecuadamente y, aunque existen colisiones, éstas no hacen divergir al *EKF*, al recibir percepciones correctas inmediatamente.

Alrededor del final del primer minuto, el robot activa el rol *defender* e inicia un desplazamiento a su propio campo. Durante el trayecto el robot choca repetidamente con otro robot y comienza a divergir la posición estimada de la real al no recibir percepciones correctas. Posteriormente comienza otra vez a percibir las dos balizas y la portería, lo que hace que el *FMK* se recupere inmediatamente e indique que la posición estimada por *EKF* es errónea y que existe otra hipótesis mas fiable en otra posición. En ese instante se descarta el actual *EKF* y se inicia otro en la posición correcta. Otra situación similar ocurre varios segundos después a raíz de que *FMK* estima erróneamente otra posición del campo de juego debido a un falso positivo y hace que el *EKF* se inicie, por error, en una posición incorrecta. Tras varios segundos la situación se corrige de nuevo.

En las otras dos ejecuciones representadas se producen situaciones similares. Es poco común que el *FMK* haga que el *EKF* se inicie erróneamente, ya que es necesario que se produzcan varias percepciones incorrectas para que la calidad asociada a la estimación de *FMK* sea alta aunque la posición calculada sea incorrecta. Recordemos que sólo cuando esta calidad es alta se puede producir un reinicio del *EKF* que, a su vez, debe tener una incertidumbre alta asociada a su estimación.

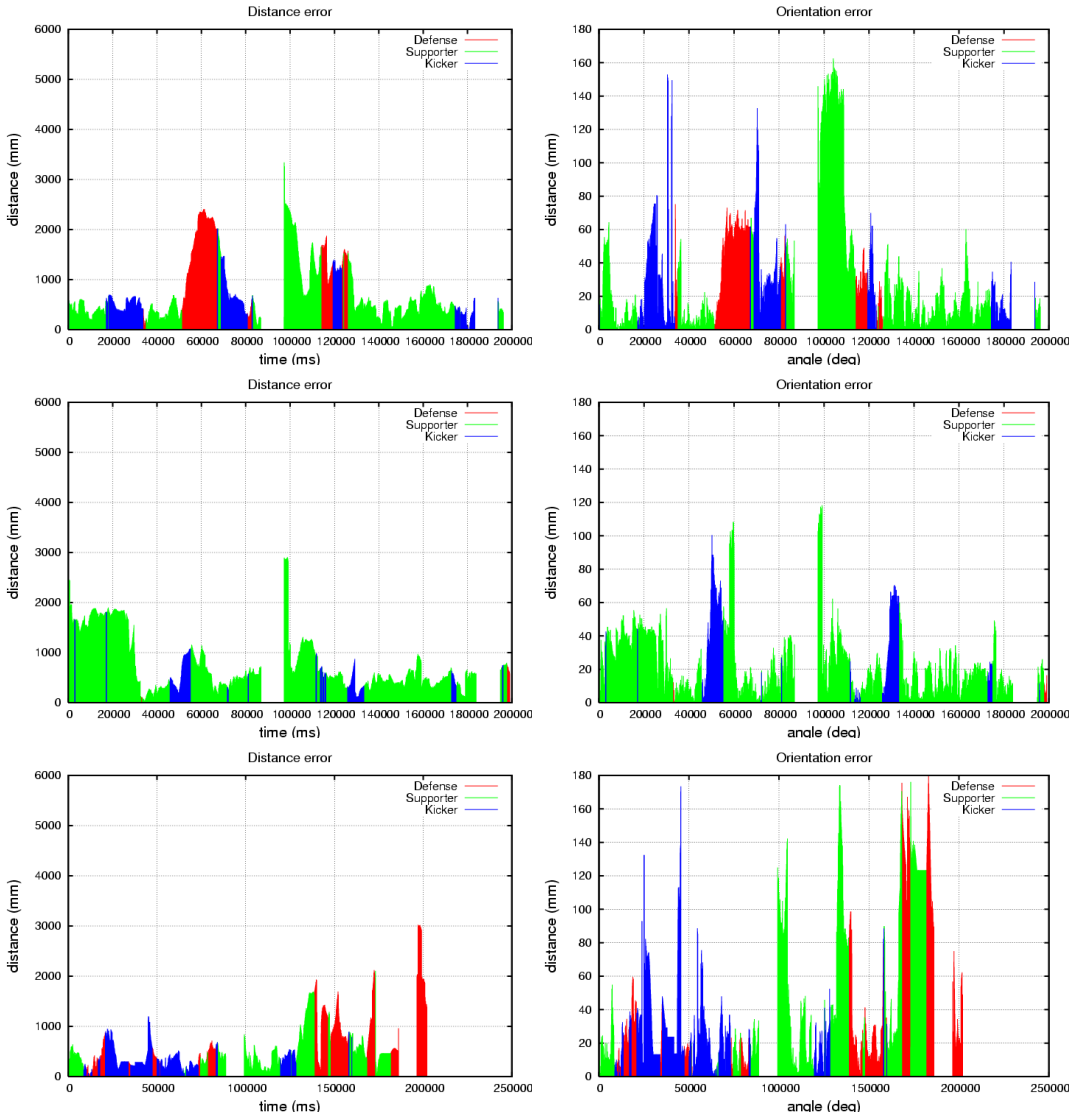


Figura 5.23: Error en distancia y en orientación del método *FMK+EKF*

En la ejecución representada en la gráfica inferior se observa que el error es bajo durante más de un minuto y medio, independientemente del rol que se encuentre activo. En esta fase no se producen inicios del *EKF* y éste mantiene correctamente su estimación.

Creemos que los resultados del experimento son francamente buenos y muestran las mejores cualidades de los dos métodos que combinan. De los resultados obtenidos durante las ejecuciones de este experimento se pueden sacar las siguientes conclusiones:

- La precisión es mucho mayor que en los dos métodos anteriores.
- Mantiene el error e la estimación estable y con valores bajos.
- Se recupera en todos los casos en que el error indica que el robot se pierde, aunque esta recuperación no es inmediata, sino tras varios segundos.
- Al igual que el método *EKF*, no parece afectarle el hecho de activar el rol *Kicker*.

### 5.4.4. Situaciones reales de juego empleando el método *FMK+nEKF*

La evolución del error cuando se usa en juego el método *FMK+nEKF* se muestran en la figura 5.24. En ella se aprecia que, aunque las estimaciones son precisas cuando la estimación es correcta, el método no es tan estable como en el caso anterior.

En la ejecución representada en la gráfica superior, el método *FMK* aporta múltiples hipótesis. Se inician y se eliminan constantemente *EKFs*. Normalmente los *EKFs* iniciados en situaciones erróneas no llegan a disminuir tanto su incertidumbre como para ser seleccionados como el *EKF* que estima la posición de este método. Durante esta fase de juego hay instantes en que el *EKF* no incorpora percepciones que hagan disminuir la incertidumbre. En esos instantes, durante pocos ciclos, la estimación del método se corresponde con la de otra hipótesis incorrecta que ha conseguido disminuir su incertidumbre hasta ser menor que la del *EKF* correcto.

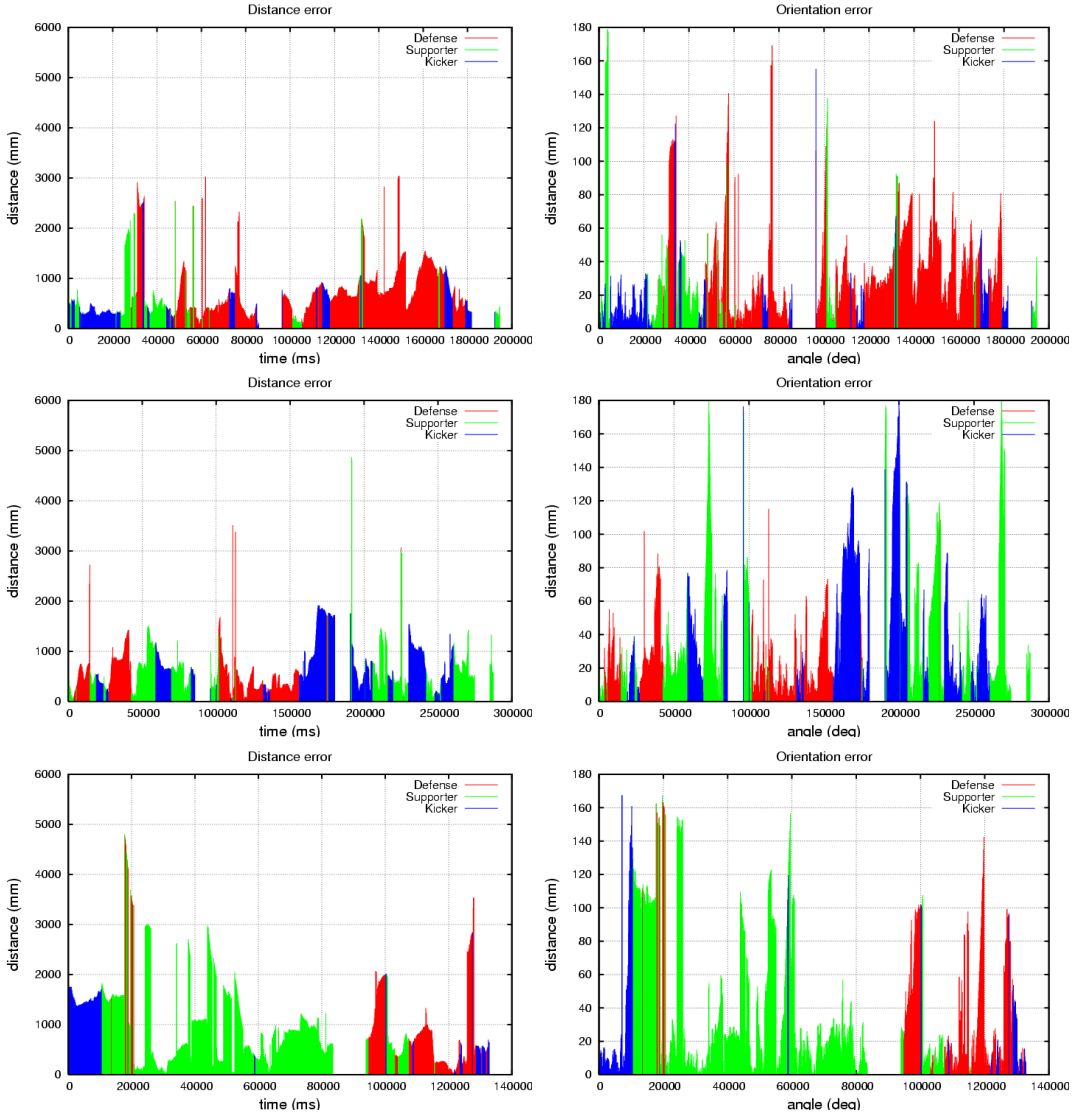


Figura 5.24: Error en distancia y en orientación del método *FMK+nEKF*

En la ejecución representada por las gráficas de la segunda fila, se observa que el error se mantiene inferior a un metro la práctica totalidad del tiempo hasta el instante 160.000. En este instante el robot activa el rol *kicker* y va hacia la pelota sin percibir ningún elemento fijo del entorno. El robot colisiona con otro robot que está cerca de la pelota y queda atrapado a escasos centímetros de la pelota. El *EKF* sigue incorporando el modelo de actuación, por lo que estima que el robot avanza, aunque esté

atrapado. Posteriormente, por un instante, el robot activa el rol *supporter* y percibe una portería y una baliza. El *FMK* estima correctamente la posición del robot y se crea un *EKF* en esta hipótesis, asociándole una incertidumbre baja. Al poco tiempo, la incertidumbre del *EKF* erróneo supera a la del que se acaba de crear, haciendo que el robot se recupere de esta pérdida.

Las primeras conclusiones obtenidas tras este experimento indican que:

- La precisión es similar al método *FMK+EKF*
- Se muestra estable, independientemente del rol.
- Se recupera en todos los casos de pérdida con mayor rapidez que en el caso de *FMK+EKF*.

### 5.4.5. Análisis de la cantidad error durante el experimento

El análisis anterior muestra cómo se comporta cada uno de los métodos de auto-localización frente a las situaciones habituales que se producen durante el juego. Estos comportamientos permiten observar varias de las características que cada método: cómo se comporta dependiendo del rol activo, cómo le afectan los falsos positivos y la influencia del resto de los robots, sobre todo en forma de colisiones durante el desplazamiento.

A continuación se presenta el análisis de error que resume la precisión de los métodos durante todas las ejecuciones de las que se compone este experimento. Decidimos dividirlo según el rol que se encuentre activo para poder analizar si esto tiene influencia en el comportamiento de los métodos.

En primer lugar, en la tabla 5.9 se resume el error en la estimación de la posición del robot por cada uno de los métodos de auto-localización. La tabla se divide en tres zonas representadas con diferentes colores. Cada zona resume la información del error cuando el robot tiene activo determinado rol. Así, en azul se representa la información referente al rol *kicker*, en verde *supporter* y en rojo *defender*, al igual que se representa en las gráficas anteriores.

La primera de las zonas es la del rol *kicker*. Se observa que el valor medio de error menor corresponde al método *FMK+EKF*. Este resultado es el esperado después

| Estadístico | *EKF* | *FMK* | *FMK+EKF* | *FMK+nEKF* |
|---|---|---|---|---|
| Primer Cuartil | 285.58 | 544.61 | 291.885 | 356.35 |
| Mediana | 546.07 | 1069.6 | 496.64 | 543.6 |
| Tercer Cuartil | 1110.7 | 1751.3 | 1052.1 | 1076.5 |
| **Media** | **949.53** | **1414** | **746.57** | **874.54** |
| Desviación típica | 933.91 | 1203.3 | 624.54 | 806.71 |
| Primer Cuartil | 282.73 | 434.01 | 341.25 | 294.95 |
| Mediana | 437.92 | 660.5 | 537.43 | 556.9 |
| Tercer Cuartil | 831.76 | 1160.6 | 1100.6 | 920.26 |
| **Media** | **688.88** | **1020.6** | **824.92** | **746.58** |
| Desviación típica | 705.73 | 954.71 | 740.01 | 674.71 |
| Primer Cuartil | 479.3 | 431.37 | 352.88 | 333.29 |
| Mediana | 812.79 | 759.17 | 727.18 | 577.38 |
| Tercer Cuartil | 1203.8 | 1386.7 | 1852 | 857.11 |
| **Media** | **987.1** | **1205.7** | **1152.5** | **749.39** |
| Desviación típica | 689.11 | 1189.5 | 997.39 | 721 |

Tabla 5.9: Análisis estadístico del error en posición.

de observar gráficas de las ejecuciones de este método. El segundo método con menor valor es *FMK+nEKF*, seguido de *EKF*. El error del método *FMK* es casi el doble del *FMK+EKF*. Este elevado error en el método *FMK*, que ya se ha explicado en parte anteriormente, se debe a que necesita distintas percepciones válidas para estimar correctamente la posición del robot, y cuando este rol está activo no se presta atención nada más que a la pelota y a la portería. En las otras zonas, correspondientes a los otros roles, se observa que el error es menor.

En esta primera zona, y en la correspondiente al rol *supporter* se observa cómo el valor menor del primer cuartil corresponde al método *EKF*. Esto refuerza la idea de que este método tiende a ser muy preciso cuando la estimación es correcta, pero que cuando diverge, los valores de error son muy elevados.

En la zona coloreada en verde, correspondiente al rol *supporter*, se observa que el error es significativamente menor. De hecho, el método *FMK* mejora sensiblemente, aunque sigue sin ser el que tiene el error más bajo. La disminución del error en este método se debe a que cuando el robot tiene este rol activo, percibe más cantidad

de marcas visuales. Aún así, sigue estando afectado por falsos positivos que hacen que el error se mantenga aún alto.

El resto de los métodos experimentan también una disminución del error cuando el robot desempeña este rol. Está claro que el aumento de percepciones visuales distintas mejora los resultados de localización.

| Estadístico | *EKF* | *FMK* | *FMK+EKF* | *FMK+nEKF* |
|---|---|---|---|---|
| Primer Cuartil | 11.142 | 17.547 | 12.182 | 11.897 |
| Mediana | 29.401 | 38.787 | 26.539 | 28.061 |
| Tercer Cuartil | 52.263 | 79.772 | 56.935 | 61.943 |
| **Media** | **40.17** | **54.905** | **41.438** | **43.447** |
| Desviación típica | 38.166 | 47.062 | 41.559 | 42.177 |
| Primer Cuartil | 6.3622 | 8.8689 | 7.7368 | 7.4752 |
| Mediana | 14.753 | 23.473 | 18.621 | 16.653 |
| Tercer Cuartil | 35.779 | 55.411 | 39.23 | 34.064 |
| **Media** | **31.261** | **37.852** | **30.363** | **31.189** |
| Desviación típica | 39.153 | 38.941 | 34.343 | 37.798 |
| Primer Cuartil | 12.003 | 9.1986 | 11.007 | 10.565 |
| Mediana | 27.002 | 24.219 | 22.936 | 22.929 |
| Tercer Cuartil | 50.474 | 54.82 | 60.165 | 42.975 |
| **Media** | **37.675** | **38.535** | **40.67** | **32.795** |
| Desviación típica | 35.786 | 39.513 | 42.678 | 32.848 |

Tabla 5.10: Análisis estadístico del error en orientación de la estimación de la posición del robot (grados) en situaciones de juego en el rol *Kicker*.

Asimismo, cuando desempeña el rol *defender* también se aprecian resultados parecidos a los anteriores, salvo que existe un aumento del error en la estimación del método *FMK+EKF*. Para este aumento no hemos encontrado una explicación lógica. Tras revisar en profundidad las distintas ejecuciones en el reproductor, lo único destacado que pudiera explicar este aumento es que durante las 5 ejecuciones de este experimento, con duraciones de 3-5 minutos, coincide que este rol está activo en pocas ocasiones. De hecho, en la figura 5.23, que muestra la mitad de las ejecuciones realizadas, en la gráfica central se observa que no llega a activarse prácticamente en ninguna ocasión. Además, en la gráfica superior se activa dos veces y en una de ellas sucede una situación de error debida a una obstrucción, de la que el método tarda al-

rededor de 20 segundos en recuperarse. Ésta puede ser la única explicación de porqué este valor es tan elevado con respecto a cuando tiene activo el rol *supporter*, en el que tiene similares percepciones y tareas.

La tabla 5.10 se muestra el error en la estimación de la orientación del robot por cada uno de los métodos de auto-localización. El error en orientación está íntimamente ligado al de posición, ya que cuando el robot estima su posición incorrectamente, lo hace tanto para su posición como orientación. La mayor parte de las consideraciones que se aportaron en la tabla anterior pueden aplicarse en esta.

### 5.4.6. Análisis del error mediante intervalos de confianza

Otro factor muy importante a la hora de evaluar los diferentes métodos es el relativo al porcentaje del tiempo de ejecución que el error se encuentra en determinados intervalos de confianza. Podemos considerar una localización como muy precisa cuando el error es inferior a 300 mm, pues es el tamaño aproximado del robot. Cuando el error es inferior a 500 mm, el robot se considera aceptablemente localizado y consideramos que puede realizar tareas de navegación a puntos globales ya que, como mucho se van a quedar a medio metro del objetivo, lo que consideramos aceptable. Cuando el error se encuentra entre 500 mm. y 1000 mm. entendemos que el robot no puede navegar con precisión, pero puede aún llevar a cabo las tareas propias de los roles *supporter* y *defender*, salvo cuando está cerca de su propia portería. Si el error es inferior a 1.500 mm y mayor que un 1000 mm., la localización sólo puede usarse, como mucho, para determinar si está en su campo de ataque o de defensa.

El análisis de los intervalos de confianza lo realizaremos únicamente del error en la estimación de la posición del robot, por ser ésta información la más relevante para comparar los resultados. Los intervalos de confianza del error en la estimación de la orientación muestran resultados similares de los que no se pueden extraer conclusiones fiables. En el caso del error en distancia, hemos representado la información de los intervalos de confianza de dos formas distintas. En primer lugar, en la figura 5.25 hemos representado, para cada método de auto-localización, los intervalos de confianza del error dependiendo del rol que se encuentra activo en cada momento. Esta representación nos permite observar fácilmente si rol que se encuentra activo influye

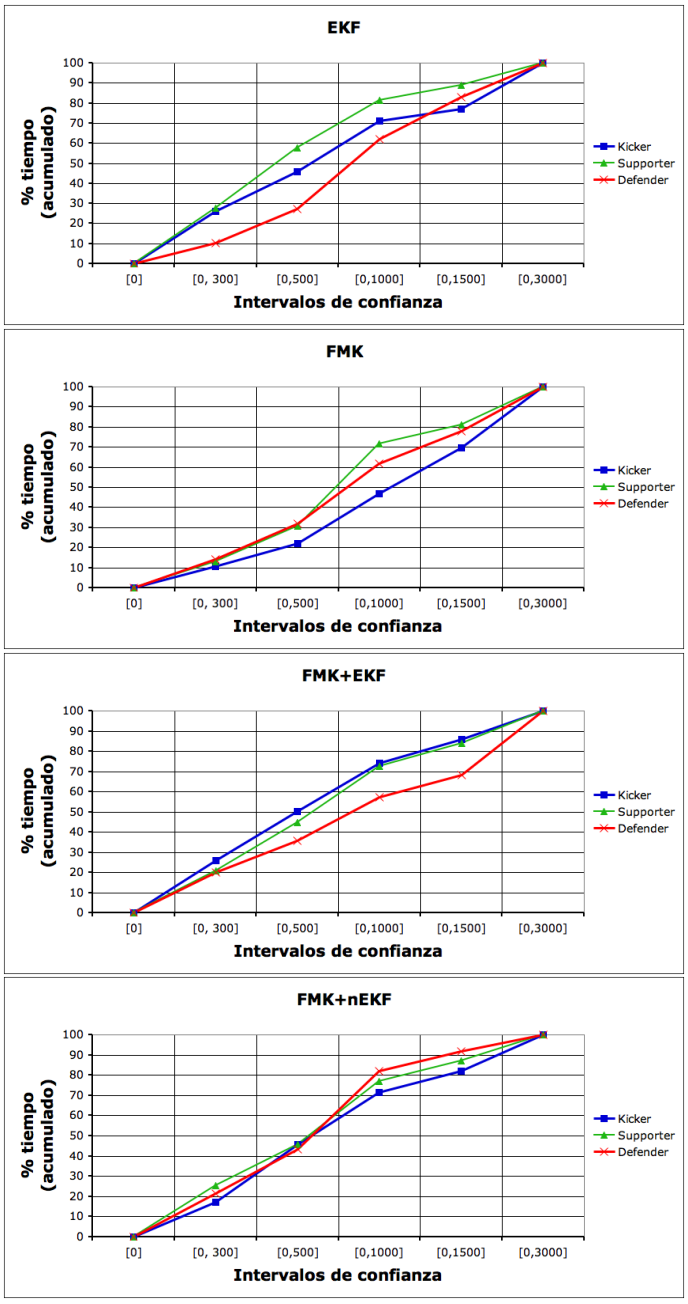


Figura 5.25: Intervalos de confianza del error de cada método de auto-localización dependiendo del rol que se encuentra activo.

en el error en la estimación de la posición. En segundo lugar, en la figura 5.26 hemos representado, para cada rol que el robot pueda activar, los intervalos de confianza del error dependiendo de cada método de auto-localización. Esta representación permite observar qué método estima mejor la posición del robot cuando el robot activa determinado rol.

En la figura 5.25 se pueden observar varios resultados de cada uno de los métodos de auto-localización. En primer lugar, el error en la estimación de la posición en el método *EKF* no es similar en los tres roles. Se observa gran diferencia entre los resultados cuándo se encuentra activo el rol *defender* y cuándo se encuentran activos los otros dos roles. Cuando el robot activa el rol de *defender* durante una situación de juego real, trata de ir a una posición global del terreno de juego. Si el error en su estimación aumenta y el robot se pierde, es habitual observar cómo se marcha del campo sin ser capaz, en la mayoría de las ocasiones, de recuperarse de este error. En estos casos es complicado que observe la pelota cerca ya que, en el mejor de los casos, se encuentra fuera de los límites del campo orientado hacia ella, con lo que el rol que suele estar activo cuando está perdido es el de *defender*. Esto no suele suceder con el rol *supporter* ya que, debido que se intenta situar cerca de la pelota, no abandona el campo habitualmente e incorpora observaciones frecuentemente. De hecho, cuando el robot activa el rol *supporter* es cuándo mejores resultados obtiene. Cuando el rol *kicker* se activa, deja de incorporar observaciones. Esto no afecta demasiado a la estimación de *EKF*, ya que usa su modelo de actuación para actualizar la estimación. La diferencia a favor del rol *supporter* se debe principalmente a que errores en la actuación o los choques frecuente por alcanzar la pelota hacen acumular error en la estimación y no son corregidos inmediatamente hasta que cambia de rol e incorpora observaciones.

En el caso de método *FMK* se observa cómo el error es mayor en el caso de tener activo el rol *kicker*. Esto se produce al no incorporar observaciones mientras intenta ir a la pelota y jugarla. Como ya hemos comentado varias veces a lo largo de este capítulo, el modelo de actuación difumina rápidamente la posibilidad de cada celda a sus vecinas. Esto produce que en poco tiempo el valor de cada una de las celdas sea similar y la posición del robot se calcule como el centro del campo. Cuando el resto

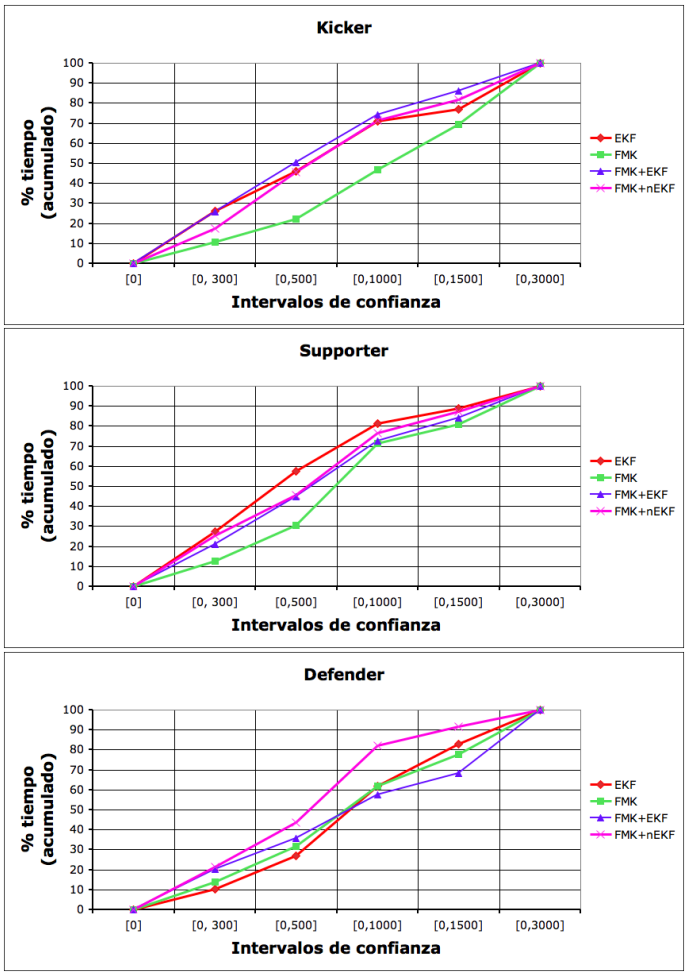


Figura 5.26: Intervalos de confianza del error de cada método de auto-localización dependiendo del rol que se encuentra activo.

de los roles están activos se observa que los resultados son mejores, al incorporar frecuentemente percepciones.

El método *FMK+EKF* muestra resultados similares activando los roles *kicker* y *supporter*, pero no en el rol *defender*, que es sensiblemente peor que los otros dos. Esto tiene que ver con que el funcionamiento, hasta que se reinicia el *EKF*, es similar al método *EKF* descrito anteriormente. Al ser relativamente lento en recuperarse del error, acumula mucho error cuando activa este rol. En cambio, en intervalos de con-

fianza inferiores a 1000 mm., los resultados son mejores a *EKF* debido a los reinicios que se producen a la posición del robot correcta.

El método *FMK+nEKF* muestra resultados similares en los tres roles. Esto se debe a que este método tiene una mecanismo más eficaz para recuperarse frente a pérdidas, con lo que no se producen las situaciones que acabamos de comentar con el método *FMK+EKF*, cuya causa es este excesivo tiempo de recuperación.

En la figura 5.26 se pueden comparar los diferentes métodos de auto-localización en cada uno de los roles. En el rol *Kicker* y *Supporter* se observan similares resultados, a excepción de *FMK*, que es sensible al hecho de no incorporar observaciones cuando el robot tiene activo este rol. Cuando el robot activa el rol *Defender* se observa que el método *FMK+nEKF* obtiene los mejores resultados. Esto es debido principalmente a que este método siempre elige el *EKF* con menor incertidumbre. Al incorporar con mayor frecuencia observaciones se potencia que disminuya la incertidumbre del los *EKF* que se encuentran en posiciones del robot correctas, seleccionado normalmente el *EKF* adecuado.

## 5.5. Análisis del tiempo de procesamiento

El tiempo de procesamiento de cada uno de los métodos de localización es un factor importante a la hora de evaluar los diferentes métodos de auto-localización. Ya se ha insistido en que una de las motivaciones principales para abordar esta investigación fue mejorar un método de auto-localización que a menudo sobrepasaba unos límites de tiempo aceptables. Esto hacía que surgieran problemas de sincronización entre los distintos módulos del software en el AIBO cuando competía en la RoboCup.

En los métodos analizados en este capítulo, este tiempo de computación puede variar de un ciclo a otro dependiendo de factores diversos:

- El tiempo de procesamiento del método *FMK* depende fundamentalmente del número de celdas del que se compone su rejilla. Este valor es fijo, pero el tiempo de procesamiento también se ve influido por el número de balizas percibidas. Cada incorporación de una marca visual obliga a operaciones extras en cada una de las celdas que conforman la rejilla.

- El método *EKF* realiza el ciclo de corrección tantas veces como percepciones válidas tenga.

- El método *FMK+EKF*, al ser una combinación de los dos métodos anteriores, los factores que influyen en el tiempo de procesamiento son los propios de cada uno. El tiempo de procesamiento del código que los combina es prácticamente despreciable.

- El método *FMK+nEKF* se ve sometido a los factores que influyen en el tiempo de procesamiento del método *FMK* y a la suma de cada uno de los *EKF* que mantiene. El tiempo de procesamiento de este método es muy variable debido a que el número de *EKFs* activos varía dinámicamente según las necesidades de localización. Además, el código que los combina es más complejo que en el caso de un solo *EKF*, teniendo que destruir e iniciar a menudo un nuevo *EKF*, con el consiguiente aumento de tiempo al liberar y reservar memoria.

A todos estos factores, además, hay que contar con posibles tiempos que se dedican a que el sistema operativo del robot atienda eventos que puedan ocurrir (mensajes de red, recepción de imágenes de la cámara, etc.).

En este experimento se mide el tiempo total de procesamiento que consume cada uno de los métodos de auto-localización. Para realizar el análisis se anota en ficheros todo el tiempo que el control pasa por rutinas relacionadas con el cálculo de la posición del robot. Este tiempo incluye la incorporación del modelo de actuación, el procesamiento de las percepciones visuales y el cálculo de la estimación de la posición del robot.

Se han realizado varias ejecuciones en situaciones normales de juego. En cada ejecución se ha usado un método distinto de auto-localización. No es posible medir el tiempo de procesamiento que utiliza cada método exactamente ante los mismos datos de entrada, ya que los tiempos que se muestran en esta sección se han obtenido de ejecuciones reales en el robot, donde sólo puede haber un método activo a la vez. Una alternativa hubiera sido realizar esta prueba en el reproductor. Esto no se ha hecho así por dos razones. La primera es que entonces se mostrarían tiempos de computación de un ordenador de sobremesa, donde hay muchos procesos ejecutándose a la vez. La

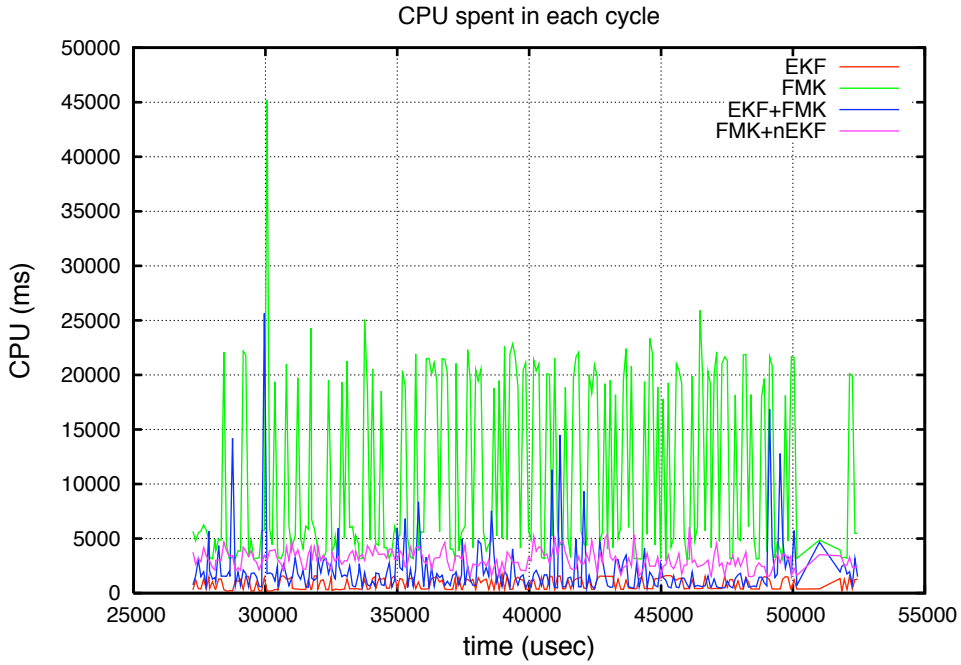


Figura 5.27: Tiempo de procesamiento por ciclo de cada método de localización

segunda es que queremos obtener datos reales de cuanto tardan *estos* cálculos en *este* robot. Podría ocurrir que el robot o el ordenador de sobremesa dispusiera de hardware que acelerara alguno de los cómputos que realiza alguno de los métodos, alterando los resultados.

En la figura 5.27 se superponen en la misma gráfica las distintas ejecuciones donde se muestran los tiempos de procesamiento. El objetivo de esta representación es tener una idea general de los tiempos de cada método y poder compararlos visualmente. Como acabamos de comentar, no tiene sentido valorar que un método aumente el tiempo de ejecución y otro no en el mismo instante.

Se puede observar cómo el método que menos consume es el *EKF*, que está representado por una línea roja. Además, como se puede observar en el resumen estadístico mostrado en la tabla 5.11, este método no presenta grandes variaciones sobre

| Estadístico | *EKF* | *FMK* | *FMK+EKF* | *FMK+nEKF* |
|---|---|---|---|---|
| Minimo | 166 | 2693 | 354 | 1009 |
| Primer Cuartil | 326 | 4117 | 624 | 3207 |
| Mediana | 384 | 5282 | 1377 | 22.929 |
| Tercer Cuartil | 1221 | 18157 | 1861 | 4009 |
| Maximo | 2327 | 57913 | 26354 | 10488 |
| Media | 662.05 | 9070.9 | 1722.3 | 3395.3 |
| Desviación típica | 491.68 | 7921.5 | 1929.5 | 1271.4 |

Tabla 5.11: Análisis estadístico de tiempo de procesamiento de cada método de localización ($\mu$s).

su media. El valor máximo que ha llegado a alcanzar a lo largo de los experimentos son 2327 $\mu$s.

El método que más recursos computacionales consume es *FMK* (con una configuración de 100 mm de ancho de celda), que tiene una media de 9070.9 $\mu$s. No es sólo que consume 13 veces más recursos computacionales que *EKF*, sino que su tiempo de procesamiento varia significativamente de un ciclo a otro, pudiendo llegar a tardar 57913 $\mu$s.

Los métodos combinados parten del tiempo de computación del método *FMK* y del tiempo de computación uno o varios *EKF*, según qué método. En este caso, el ancho de celda se establece a 500 mm, lo que hace que no consuma tanto como el método *FMK* que se ejecuta en solitario y que tiene la celda en 100 mm, lo que reduce el tiempo de ejecución considerablemente.

El método combinado que más consume es, obviamente el método *FMK+nEKF*. Aún así, este método consume tres veces menos que el método *FMK*. Esto pone de manifiesto que el objetivo de disminuir significativamente el tiempo de procesamiento, una de las motivaciones de este trabajo, se ha cumplido ampliamente.

# CAPÍTULO 6

# Conclusiones

El objetivo de esta tesis ha sido desarrollar soluciones al problema de la auto-localización de robots móviles que cuentan con patas para desplazarse y con una cámara como su principal sensor.

El problema de la auto-localización no es sencillo. En el capítulo 2 se ha presentado multitud de técnicas de auto-localización que proponen diferentes soluciones a partes del problema. La dificultad al abordar un problema de auto-localización depende en gran medida de los sensores y actuadores, además de la capacidad de cómputo, con que cuenta el robot. Así, los robots con ruedas cuentan con una odometría mucho más precisa de lo que puede ser la que calcula un robot con patas. Asimismo, es más sencillo usar sensores de distancia (p.e. sensor láser) que aporta una información fiable y precisa, que la imagen capturada por una cámara, que es muy sensible a las condiciones de iluminación y que necesita mucho procesamiento.

En el caso concreto de eta tesis, todos los trabajos se han realizado en el robot AIBO. Los entornos en los que se pretende conseguir que el robot se autolocalice son los de oficina y, principalmente el entorno de la RoboCup 4LL[1]. En este último entorno es donde se han se han realizado las principales aportaciones de esta tesis. El entorno y la plataforma robótica usada presentan una serie de dificultades que es ne-

[1]Liga de las 4 patas (Four Legged League)

cesario comentar. En primer lugar, el robot dispone de cuatro patas para desplazarse, por lo que la odometría no es tan precisa como en los robots con ruedas, como acabamos de comentar. Además, en el entorno de la RoboCup el robot comparte el entorno con otros siete robots, y son frecuentes los choques y empujones que producen desplazamientos impredecibles. En segundo lugar, el robot usa una cámara situada en su cabeza como principal fuente de información. Además de la dificultad en el procesamiento de la imagen y en los problemas de iluminación que pueda existir, la cámara sufre de un balanceo continuo debido al movimiento del robot que introduce también numerosos errores difíciles de modelar. También hay que tener en cuenta que el entorno de la RoboCup es un entorno dinámico con múltiples agenes que pueden producir falsos positivos y oclusiones de las marcas visuales del campo.

En este capítulo se resumen, en primer lugar, los trabajos que se han realizado en esta tesis. A continuación se presentarán las conclusiones que han obtenido analizando los pros y contras de cada método y haciendo hincapié en la adecuación al problema que se pretende resolver y en los métodos seguidos. Asimismo, se mostrarán las publicaciones nacionales e internaciones realizadas al hilo de esta tesis. Dos de ellas se han publicado en revistas que se encuentran entre las diez primeras del índice de impacto JCR, lo que avala la calidad de los trabajos desarrollados en esta tesis. Por último, se marcarán cuales son las principales lineas futuras para continuar el trabajo presentado en esta tesis.

## 6.1. Aportaciones realizadas

Las aportaciones realizadas en esta tesis se pueden dividir en tres grupos. En primer lugar, las *aportaciones principales*, que son el desarrollo de métodos de auto-localización en entornos de oficina y el desarrollo de dos métodos de auto-localización, *FMK+EKF* y *FMK+nEKF*, que se integran en el software del equipo TeamChaos. El desarrollo de estos últimos métodos se motiva en que la necesidad de disponer de técnicas más adecuadas que la ya existente en el software del TeamChaos, *FMK*, que es en el que se basan los métodos desarrollados.

A continuación se describen las *aportaciones secundarias* realizadas en esta tesis. Estas aportaciones tienen que ver con la metodología llevada a cabo en el desarrollo

de los métodos descritos, con el análisis de los modelos de actuación y percepción, y con el desarrollo de un *EKF* adaptado al problema particular del entorno de la RoboCup. Este método, junto con *FMK*, son los dos componentes principales de los métodos *FMK+EKF* y *FMK+nEKF*.

Por último y como tercer tipo de aportaciones, describiremos el conjunto de herramientas desarrolladas al hilo de esta tesis.

### 6.1.1. Aportaciones principales

En esta sección se presenta las aportaciones más significativas llevadas a cabo en esta tesis. En primer lugar, el desarrollo del un método de auto-localización topológica markoviana, que ha permitido analizar las características de la plataforma y la viabilidad del método en el entorno de la RoboCup. En segundo lugar, se han desarrollado dos métodos de auto-localización cuyo objetivo concreto es el entorno de la RoboCup, integrándose dentro del software del TeamChaos.

#### Auto-localización topológica Markoviana

La aportanción consiste en la implementación de un método de auto-localización para el robot Aibo en un entorno de oficinas. El método desarrollado se basa en representar el entorno topológicamente como un conjunto de estados y transiciones entre ellos. Usando técnicas markovianas, se mantiene una distribución de probabilidad sobre cada uno de los estados. A esta distribución de probabilidad se incorpora el movimiento del robot e información perceptiva de origen visual.

Este trabajo ha sido verificado experimentalmente, como se describe en la sección 3.2.4, demostrando la viabilidad de este método en robots con patas y y que utilizan la visión como sensor principal.

Se ha estudiado también la viabilidad de este método en el entorno de la RoboCup. Para ello se ha representado el campo como un conjunto de estados, optando en este caso por una de rejilla. Pero los resultados no fueron satisfactorios lo que ha servido de catalizador para el resto de desarrollos.

**Métodos de auto-localización en el entorno de la RoboCup**

La aportación más relevante de esta tesis ha sido el desarrollo de varios métodos de auto-localización en el entorno de la RoboCup, integrados dentro del software del equipo TeamChaos.

Para la implementación de los métodos de auto-localización desarrollados en esta tesis, se ha partido de la implementación del algoritmo *FMK*, descrito en la sección 2.2, que se había usado tradicionalmente en el software del equipo TeamChaos. Este es un método global basado en una rejilla de celdas difusas de 100 mm. de lado. Debido a esta representación y al aumento de las dimensiones del campo, los tiempos de computación se incrementaron hasta alcanzar los límites de lo que se consideraba aceptable, como se muestra en las figuras 4.15 y 5.22, y en el análisis que las acompaña. Además, se había observado que este método es demasiado sensible a los errores en la percepción, indicando en esos casos que la posición del robot es el centro del campo y la incertidumbre es máxima, como se ha mostrado en la experimentación realizada.

La aportanción ha consistido en combinar *FMK* con uno y varios *EKFs* que se ejecutan en paralelo usando la misma información perceptiva y odométrica. En el primero de los métodos, denominado *FMK+EKF*, se usa la posición calculada por *FMK* para iniciar un *EKF*, que es el que se usa para proporcionar la información de localización. Asimismo, se usa la información de las celdas de la rejilla de *FMK* para verificar la estimación de *EKF*. La verificación se realiza comprobando la incertidumbre asociada a la estimación de *EKF* y el valor de la posibilidad de la celda en la rejilla de *FMK* correspondiente a la posición que estima *EKF*. Si la verificación indica que la estimación de *EKF* es incorrecta, se inicia de nuevo en la posición calculada por *FMK*.

Este método ha demostrado cumplir con los objetivos marcados. En primer lugar, el consumo de CPU se ve reducido en un 82 % con respecto al método *FMK* original. En segundo lugar, hemos probado que este método es capaz de mantener una estimación fiable y precisa de la posición del robot, como se ha demostrado con la experimentación mostrada en el capítulo 5. En tercer lugar, hemos probado que este método es capaz de mantener la estimación correcta sobre la posición del robot

en situaciones en las que robot no presta atención a las marcas visuales y se dirige a la pelota, mejorando notablemente el método *FMK*, que no es capaz de mantener la estimación tras varios ciclos sin incorporar percepciones. Por último, también hemos demostrado que este método es capaz de recuperarse de situaciones de secuestro, aunque los tiempos de recuperación han sido muy elevados, como se muestra en la tabla 5.3, por ejemplo. Otro aspecto negativo es que el reinicio del método tras una situación de pérdida atraviesa a veces situaciones de inestabilidad. Este comportamiento se debe a la excesiva dependencia de la información de *FMK*, que es a menudo inestable.

El segundo método, denominado *FMK+nEKF* soluciona los problemas apreciados en el método anterior. Este método mantiene una población de varios *EKFs* que varía dinámicamente. En este caso, se usa *FMK* para verificar cada uno de los *EKFs* y para aportar hipótesis sobre la posición del robot cuando la incertidumbre de su estimación es alta. Cada vez que *FMK* aporta una hipótesis, se inicia un nuevo *EKF* en ella. Los *EKF* que aumentan su incertidumbre y cuya estimación corresponde a celdas de *FMK* poco probables se eliminan. La posición del robot se determina como la estimación del *EKF* con menor incertidumbre.

Este método tiene unos tiempos de cómputo algo superiores a *FMK+EKF*, aunque en media el consumo de CPU se ve reducido en un 63 % con respecto al método *FMK* que era nuestra referencia y punto de partida. El método es similar en estabilidad a *FMK+EKF*, aunque muestra tiempos de recuperación frente a secuestro menores, de alrededor de 2 segundos, como se puede observar, de nuevo, en la tabla 5.3.

La reducción del tiempo de cómputo se debe fundamentalmente a que se ha reducido el tamaño de las celdas que componen la rejilla en la implementación original hasta los 500 mm. Esto supone una disminución dramática del número de estados y, por consiguiente, del tiempo de computación. Esta modificación hace que *FMK* no sea capaz de determinar con precisión la posición del robot, pero permite que se use para evaluar, cuando su incertidumbre es baja, si un robot se encuentra en una posición determinada del campo. Además, permite obtener una estimación poco precisa, pero fiable, de la posición del robot.

Nos gustaría resaltar que se ha realizado una extensa experimentación con el robot real. Los experimentos analizan el comportamiento de los métodos de auto-localización en situaciones de auto-localización pasiva (sección 5.1), secuestro (sección 5.2), navegación global (sección 5.3) y situaciones reales de juego (sección 5.4).

Como resumen a estas conclusiones, en la figura 6.1 se muestra una valoración personal que hemos realizado de cada uno de los métodos que ha sido objeto de experimentación. Hemos valorado varias características que creemos que deben reunir los métodos de auto-localización: Precisión en la estimación de la posición, estabilidad a lo largo del tiempo de la estimación, robustez frente a errores perceptivos y de actuación, recuperación frente a secuestros y al inicio de la operación del robot, y necesidades de tiempo de cómputo. Cada una de las características las hemos valorado de 0 a 10 en función de la experiencia con cada uno de los métodos, siendo 10 la valoración que mejor satisface cada una de las necesidades de cada característica. En esta gráfica se resumen las conclusiones que acabamos de presentar.

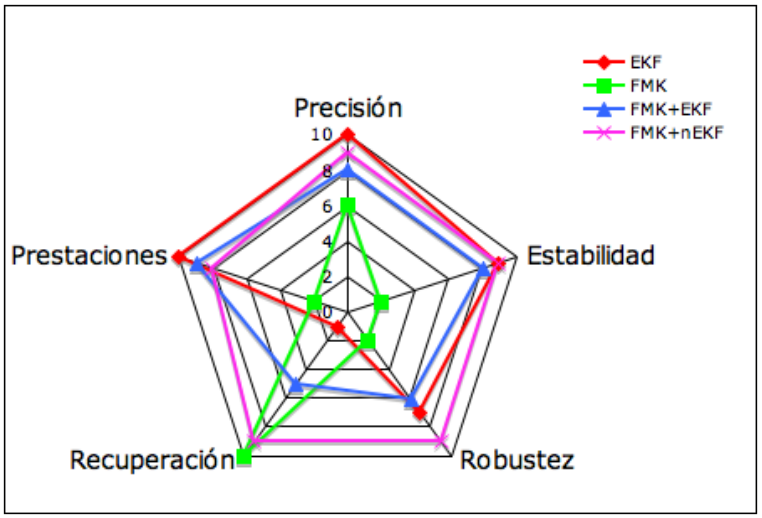


Figura 6.1: Resumen de las características de los métodos analizados.

Por último, indicar que estos dos métodos se han probado no sólo en el laboratorio, sino en campeonatos reales, contra equipos internacionales. En particular, la validación fundamental se realizó en la edición del German Open celebrado en

Hannover en el año 2007. Los resultados de la aplicación de estos métodos en este campeonato fueron subjetivamente muy satisfactorios.

### 6.1.2. Aportaciones secundarias

En primer lugar, se ha realizado un análisis del modelo de movimiento (sección 4.2) y de percepción (sección 4.1) en el entorno de la RoboCup para corregir y modelar el error que se produce tanto en la actuación como en laspercepción de las marcas visuales del campo de juego. Este estudio es fundamental para el éxito de los métodos de auto-localización. Además, puede ser útil para aplicar otros métodos, o para quien trabaje con este tipo de robots.

También se ha diseñado un *EKF* para estimar la posición del robot usando los modelos de movimiento y percepción que acabamos de citar. Este método es capaz de realizar una localización local o *tracking* del robot, aunque no es capaz de recuperarse de situaciones de completo desconocimiento (necesita conocer la posición inicial), secuestro o estimaciones erróneas, como se ha mostrado en el capítulo de experimentación.

Hay que mencionar el desarrollo de un mecanismo para rechazar las percepciones consideradas incoherentes con la estimación actual, que denominamos *filtro de incoherencias*. Este mecanismo puede aplicarse en otras aproximaciones que usen EKF en similares circunstancias.

### 6.1.3. Herramientas

Además de las contribuciones teóricas (métodos FMK+EKF y nFMK+EKF) y de los trabajos para su puesta en funcionamiento en un robot real (calibración, modelos de percepción, movimiento, etc.), se han desarrollado una serie de herramientas que también consideramos que son aportaciones interesantes. Así, se ha usado el simulador descrito en la sección B.1 y que ha sido desarrollado a partir de una versión inicial existente en el software del equipo TeamChaos, incorporándole notables mejoras. El uso de este simulador ha permitido que los métodos puedan probarse con percepciones sintéticas y así detectar tempranamente posibles errores de diseño.

Se ha desarrollado también una herramienta de reproducción, descrita en la sección B.3, que permite ejecutar en un ordenador los métodos de auto-localización desarrollados en esta tesis con datos reales de percepción, odometría y posición real del robot. Esta herramienta permite contrastar los distintos métodos de auto-localización con los mismos datos de entrada, y así analizar el comportamiento de métodos diferentes en las mismas condiciones, lo que no es posible hacer directamente con el robot. Esta herramienta también ha permitido ejecutar los métodos de auto-localización con diferentes configuraciones y observar cómo se comporta ante las mismas entradas.

Se ha implementado, a partir de una implementación básica realizada por los miembros del equipo TeamChaos, un sistema que se ejecuta en el robot real y recoge estos datos en ficheros para obtener los datos reales de percepción y odometría, como se describe en la sección B.3.1. Asimismo, se ha combinado este sistema de recogida de información con un sistema de ”verdad absoluta” (descrito en la sección B.2) que, a partir de un sistema de cámaras situadas en el techo del laboratorio, detecta al robot y calcula su posición real. Los ficheros generados con estas herramienta contienen la información perceptiva y de odometría, y están sincronizados con la del sistema de ”verdad absoluta” con menos de de 25 ms. de desviación en el tiempo. Con estos ficheros se puede alimentar el reproductor anteriormente descrito. El sistema de ”verdad absoluta” ha sido verificado para garantizar que el error que presenta al detectar la posición del robot es inferior a 7 cms. siendo de media 4 cms.

## 6.2. Publicaciones producidas dentro de esta tesis

Es relevante mostrar las publicaciones que se han desarrollado al hilo de esta tesis, ya que avalan la calidad del trabajo realizado y su relevancia en el campo de la auto-localización de robots móviles. Varias de estas publicaciones se han presentado en congresos nacionales e internacionales, siendo las dos últimas que vamos a enumerar publicadas en revistas con alto índice de impacto:

- Martín, F., González-Careaga, R., Cañas, J.M., Matellán, V. *Programming model based on concurrent objects for the AIBO robot*, Actas de las XII Jornadas

de Concurrencia y Sistemas Distribuídos (http://dalila.sip.ucm.es/jjcc04), Universidad Rey Juan Carlos y Universidad Complutense de Madrid, junio 2004.

- Martín, F., Matellán, V., Cañas, J.M., Barrera, P. *Visual Based Localization for a Legged Robot*, Lecture Notes in Computer Science, Volume 4020/2006, RoboCup 2005: Robot Soccer World Cup IX, ISSN 0302-9743, pág. 708-715.

- Martín, F., Matellán, V., Barrera, P., Cañas, J.M. *Localización basada en lógica difusa y filtros de Kalman para robots con patas*, Actas del Campus Multidisciplinar en Percepción e Inteligencia, págs. 310-321, Vol-I, ISBN: 84-689-9560-6. Albacete Julio 2006.

- Martín, F., Matellán, V., Cañas, J.M., Barrera, P. *Localización topológica basada en visión para robots móviles*, Actas de las XXVII Jornadas de Automática de Almería, págs. 1081-1088. ISBN 84-689-9417-0. DL. AL-240-2006. Almería Septiembre 2006.

- Martín, F., Matellán, V., Barrera, P., Cañas, J.M. *Localization of legged robots combining a fuzzy-Markov method and a population of extended Kalman filters*, Robotics and Autonomous Systems, , págs. 870-880, Vol. 55, 12ª edición, ISSN: 0921-8890. Diciembre 2007.

- Samperio, R., Huosheng H., Martin, F., Matellán, V. *A hybrid approach to fast and accurate localisation for legged robots*, Robótica, Cambridge University Press (Aceptada y pendiente de publicación).

## 6.3. Trabajos futuros

En cuanto a las líneas por las que se va a continuar la investigación realizada en esta tesis hemos identificado varias:

- Adaptación de los métodos desarrollados en esta tesis al robot NAO. El robot NAO, mostrado en la figura 1.7, es el robot que sustituye al AIBO en la RoboCup. El hecho de que la evolución sea hacia robots humanoides dotados con

cámara prueba que ésta es la tendencia actual en robótica móvil y no refuerzan en la que creencia de que los supuestos de esta tesis son correctos.

- Queremos estudiar también la posibilidad de mantener estimaciones de las marcas visuales del campo usando varios *EKFs* como modelo del mundo. La idea es mantener, por cada elemento visual del entorno (portería, centro del campo, esquina, etc.) una lista dinámica de *EKFs* que estimen su posición. Cada vez que se perciba un elemento se comprobará si hay algún *EKF* en la lista que ya esté estimando su posición. Si es así, se actualizará este *EKF*. Si no es así, se iniciará otro *EKF* para mantener la estimación de esta hipótesis. Si la incertidumbre asociada a la estimación de un *EKF* se incrementa demasiado, por no incorporar observaciones de este elemento, se descarta este *EKF*.

  Esta aproximación creemos que aportará varias ventajas. En el enfoque actual, si el robot percibe un elemento y en el instante siguiente percibe un falso positivo del mismo elemento, se descarta la anterior percepción y es posible que durante un periodo de tiempo se mantenga el falso positivo. En el enfoque que queremos abordar, si esto sucede no se descarta la anterior observación, sino que se inicia un *EKF* para atender a esta hipótesis con una incertidumbre mayor que la de la percepción anterior. Si es un falso positivo, que sólo se percibe por un instante, se terminará descartando ese *EKF*. Si es correcta la percepción, en pocos ciclos su incertidumbre disminuirá y se podrá usar como la percepción más verosímil.

  Otra ventaja es lo natural que resulta combinar este modelo de observación con los métodos desarrollados en esta tesis. Se podrán incorporar de manera más natural las estimaciones de cada percepción en la fase de corrección de los *EKFs* que mantienen la posición del robot.

- Estimamos interesante también estudiar la combinación de percepciones entre varios robots para mejorar la estimación de la posición del robot. Como hemos explicado a lo largo de esta tesis, es posible que un robot cuando se dirige a la pelota deje de observar las marcas visuales del campo. Esto produce que el robot deje de incorporar observaciones. Creemos que combinando las observa-

ciones de varios robots tomando la pelota como referencia se puede mejorar la auto-localización del robot en estos casos.

# APÉNDICE A

# Software del equipo TeamChaos

En este apéndice se describe la plataforma software que se ha usado en esta tesis. Esta plataforma ha sido desarrollada por los integrantes del equipo TeamChaos con el objetivo fundamental de participar en la liga de las 4 patas de la Robocup ((hoy conocida como liga de la plataforma estándar). Para la realización de esta tesis se han desarrollado nuevas herramientas y módulos que han sido integrados en esta plataforma. Entre el software desarrollado para esta tesis destacan los métodos de auto-localización que se analizan en el capítulo 4 y las herramientas que se describen en el apéndice B.

TeamChaos es un equipo, formado por varias universidades, que compitió en la liga de 4 patas de la Robocup desde 1999 hasta 2007. Originalmente se denominó *Team Sweden*, ya que inicialmente estaba formada por un consorcio de varias universidades de Suecia y la universidad de Murcia.

Desde el año 2004, en el que la universidad Rey Juan Carlos se incorporó al equipo junto con la universidad de Alicante, el equipo TeamChaos ha participado en varios eventos nacionales e internacionales, entre los que destacan la competición Robocup en Osaka (Japón, 2005) y Bremen (Alemania, 2006); competiciones europeas en Paderborn (Alemania, 2005), Eindhoven (Holanda, 2006) y Hannover (Alemania, 2007); y el campeonado Latinoamericano en Santiago de Chile (Chile, 2007). De

entre todas ellas, la edición de campeonato europeo en Hannover en el 2007 es especialmente importante por su relación con esta tesis. En esta competición se probaron con éxito por primera vez los trabajos desarrollados en ella. Nuestro trabajo de localización ha sido especialmente útil como base para la coordinación entre los robots mediante el mecanismo *Switch!* [CEA06], también probado en esta competición.

Actualmente el equipo TeamChaos está formado por las universidades de Murcia, Rey Juan Carlos, Carlos III y Rovira i Virgili.

## A.1. El software del robot: TeamChaos

*TeamChaos* es el software ejecutado por el robot AIBO. Se trata de una evolución de la arquitectura *ThinkingCap* [MS01], que se puede observar en la figura A.1. Como comentamos en la sección 1.5.1, el robot únicamente puede ser programado en C++ sobre el entorno *OPEN-R*. Este entorno estructura las aplicaciones en objetos *OPEN-R*, que pueden ser vistos como procesos independientes que se ejecutan en paralelo. Es importante remarcar que estos objetos *OPEN-R* son las unidades de ejecución en las que se estructuran las aplicaciones desarrolladas en este robot, y no guardan relación con los objetos C++ que son instancias de las clases implementadas dentro de este software, ni con los módulos GM, CTRL, etc. que vamos a describir, que son las unidades funcionales en las que dividimos nuestra aplicación. De esta manera, cada objeto *OPEN-R* contiene uno o varios módulos, y cada módulo sigue un paradigma de programación orientado a objetos y está compuesto de varias clases.

Pues bien, la aplicación *TeamChaos* se compone de tres de estos objetos *OPEN-R*: ORHRobot, ORLRobot y ORTCM.

- **ORHRobot**. Este objeto *OPEN-R* se encarga de las tareas de alto nivel que no dependen del hardware del robot. Esta división permite una mejor sincronización y la posibilidad de ejecutar este software en un ordenador convencional y así depurarlo fácilmente. Contiene dos módulos: GM y CTRL.
  - **GM**. El módulo GM (*Global Model*) se encarga de mantener un mapa global del entorno y la posición del robot en el mismo. Se puede ver como una caja negra (figura A.3) que recibe la información perceptiva de los

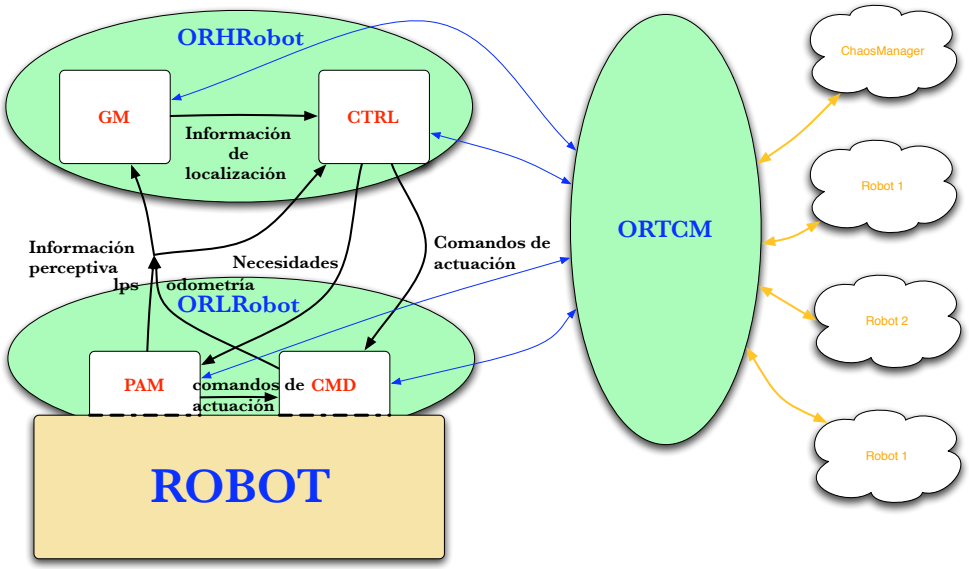


Figura A.1: Arquitectura del TeamChaos

módulos del objeto ORLRobot. Esta información consta de información del entorno local e información odométrica. La información del entorno local está compuesta por la posición, relativa al robot, de cada elemento que percibe el robot (portería, pelota, balizas...) en su entorno. Esta información se procesa, por ejemplo, para calcular la posición del robot o en otros módulos.

En este módulo, por tanto, es dónde se desarrollan las tareas de localización usando la información perceptiva que proviene del objeto ORLRobot. Este es el módulo más importante, desde el punto de vista de este trabajo. En la versión anterior a este trabajo, el módulo GM únicamente implementaba el algoritmo de localización FMK (descrito en la sección 2.2), como se muestra en el figura A.2. Uno de los trabajos desarrollados al hilo de esta tesis fue modificar el GM para poder usar diferentes algoritmos de localización de una manera sencilla y transparente.

- **CTRL**. El módulo CTRL (*Controller*) envía los comandos de actuación (velocidad lineal, lateral, de rotación y tipo de golpeo de pelota) al módulo encargado de la locomoción, CMD, que se describirá a continuación.

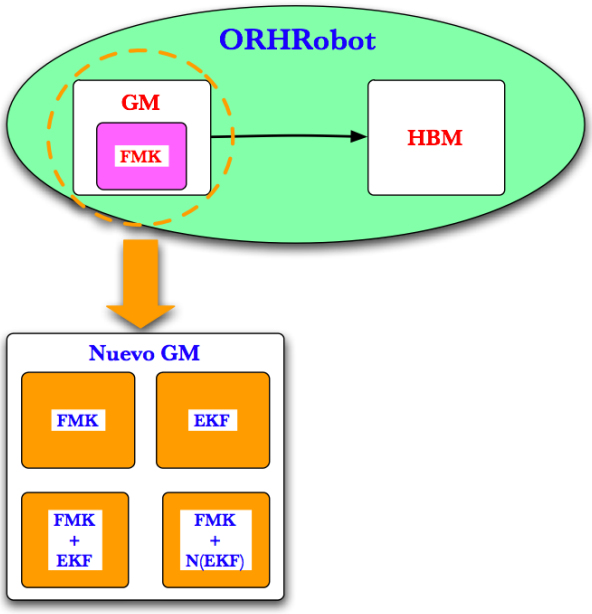


Figura A.2: Nueva implementación del módulo GM


Figura A.3: Entradas y salidas del módulo GM

Para calcular esos comandos usa la información sensorial y la posición estimada del robot.

Este módulo está dividido en dos niveles. El nivel inferior, denominado *LuaBeh* está formado por comportamiento simples desarrollados en el lenguaje LUA. Estos comportamientos básicos son, por ejemplo, *"ir a la pelota"*, *"ir a una posición global"* o *"busca la pelota"*. El nivel superior se denomina *Hierarchical Finite State Machine, HFSM*, y define el comportamiento de alto nivel del robot. Se trata de una máquina de estados formada por estados y transiciones condicionales ente ellos, donde el estado activo selecciona y ejecuta un comportamiento simple del nivel inferior.

- **ORLRobot**. Este objeto se encarga de las tareas de bajo nivel que se relacionan con el hardware del robot. Está íntimamente ligado al hardware del robot, y sus módulos realizan directamente operaciones sobre él. Está estructurado en dos módulos: CMD y PAM.

  - **PAM**. El módulo PAM (*Perception and Anchoring Module*) implementa un mecanismo de atención visual basado en una serie de ”necesidades” de percepción. Estas ”necesidades” indican los objetos del mundo real a los que se debe prestar atención. Este módulo genera las ordenes para establecer la orientación adecuada de la cámara para percibir los elementos que el módulo CTRL haya establecido como relevantes.
  - **CMD**. El módulo CMD (*CoManDer*) traduce los comandos de velocidad que recibe a una secuencia de posiciones de cada una de las articulaciones del robot. Realiza las tareas de coordinación motriz y equilibrio. También ejecuta secuencias de movimientos predeterminados llamados *kicks*, que se usan para golpear la pelota. Asimismo, recibe también del PAM la orientación que debe tener el cuello del robot para realizar la percepción activa de los elementos del entorno.

- **ORTCM**. Es objeto *OPEN-R* contiene el módulo TCM (*Team Comunication Module*), que canaliza las comunicaciones entre los módulos del robot con otros robots o con el software *ChaosManager*.

Todos estos módulos forman parte del software que se ejecuta en el robot. En la siguiente sección se presenta la aplicación *ChaosManager*, cuyas funciones están íntimamente ligadas al software que acabamos de describir.

## A.2. La aplicación *ChaosManager*

La aplicación ChaosManager contiene varias herramientas indispensables para las tareas de gestión, calibración y depuración del software ejecutado en el robot. Se puede usar para preparar el software que se ejecuta en el robot o conectarse a él para monitorizarlo y realizar cambios en su configuración.

Esta aplicación se ejecuta en un ordenador. Se ha desarrollado por completo en Java para que pueda ser usada sobre múltiples sistemas operativos sin ningún cambio.

La aplicación ChaosManager está compuesta por varias interfaces gráficas que se encargan de funciones diferentes. Se puede acceder a cada una de ellas seleccionando la pestaña adecuada. En la figura A.4 se muestra la organización de esta aplicación. La primera división se realiza en herramientas del jugador (*player tools*) y del equipo (*team tools*). La herramientas del jugador gestionan engloban aquellas relacionadas con un jugador en individual (calibración de visión, del modo de caminar, golpeos de pelota, comportamientos de alto y bajo nivel, y monitor de estado). Las herramientas del equipo engloban aquellas que sirven para gestionar el equipo completo de robots.

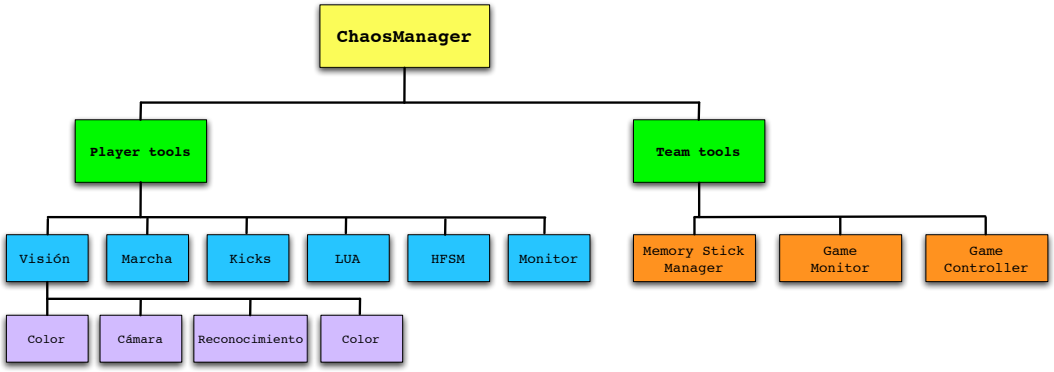


Figura A.4: Estructura de la aplicación *ChaosManager*.

# APÉNDICE B

# Herramientas de simulación y evaluación de algoritmos de localización

En este capítulo se describen las herramientas auxiliaresy aplicaciones desarrolladas o ampliadas al hilo de esta tesis para facilitar proceso de diseño y verificación de los algoritmos al trabajar con robots.

En este trabajo se han desarrollado varias herramientas útiles en todo el proceso de desarrollo de los algoritmos de auto-localización:

- Se han implementado los algoritmos de auto-localización descritos en esta tesis, además de en el software que se ejecuta en el robot, dentro de la herramienta ChaosManager.

- Se ha implementado un *simulador* que permite probar los algoritmos de manera más sencilla.

- Se ha desarrollado una infraestructura hardware/software que, haciendo uso de cámaras cenitales, permite obtener la ”verdad abosoluta” sobre la posición de robot en el campo de juego.

- Se ha diseñado un sistema para recoger información relevante de la auto-localización en el robot y combinarla con la obtenida por el sistema de ”verdad absoluta” para poder analizar la precisión.
- Se ha implementado un *reproductor* que permite ejecutar los algoritmos desarrollados en el ChaosManager sobre datos reales reales.

A continuación procedemos a la descripción de cada uno de estos elementos.

## B.1. Simulador

El simulador, mostrado en la figura B.1, se ha desarrollado dentro de la aplicación ChaosManager, haciendo uso de sus recursos. Se ha partido de una versión ya existente a la que se le ha añadido la mayor parte de su funcionalidad actual.

Este software simula un robot en el campo de juego de la RoboCup que puede realizar movimientos circulares o trayectorias a través de puntos de control.

A partir de la posición del robot genera las percepciones sintéticas de los elementos del entorno. Igualmente, a partir de estos movimientos simulados del robot se genera la información odométrica. Tanto la información perceptiva como la odometría se modifica aplicándole un nivel de ruido configurable desde la aplicación.

Cada uno de los métodos de auto-localización implementados se pueden ejecutar en el simulador por el simulador usando al información sintética que genera el simulador. Ya se mostró en la figura A.3 que cada uno de los métodos puede ser visto como una caja negra que recibe unas entradas perceptivas y devuelve una estimación de localización. El simulador puede ejecutar uno o varios métodos de auto-localización, introduciéndole exactamente los mismo datos perceptivos generados sintéticamente y recibiendo las estimaciónes de cada uno de ellos, que son analizadas. No sólo puede ejecutar métodos distintos, también puede ejecutar el mismo método con distintas configuraciones para analizar la influencia de los distintos parámetros de su ejecución.

Esta herramienta de simulación es también útil para comprobar el funcionamiento de un método de auto-localización en ausencia de ruido con percepciones simuladas y ver cómo varían los resultados según se aumenta éste. La herramienta es de gran

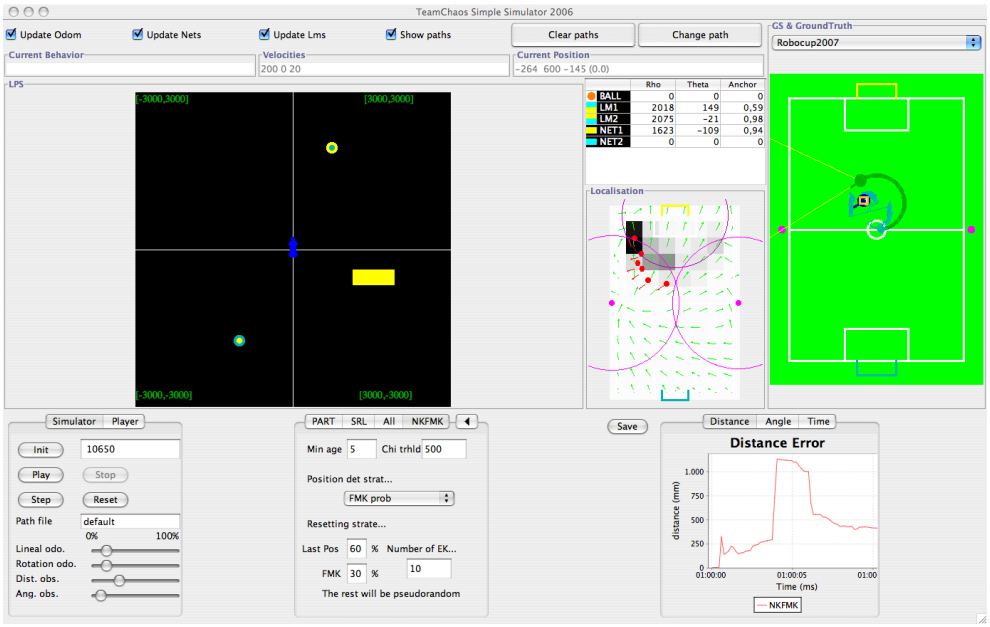


Figura B.1: Simulador

utilidad en el proceso de depuración de los métodos. Asimismo, también permite que se puedan comparar algoritmos con similares entradas perceptivas y de actuación. Los resultados que produce son muy variados: tiempos de cómputo, errores, posición real, estimada, etc...

## B.2. Sistema de ”verdad absoluta”

En el análisis del comportamiento de los distintos algoritmos hemos mostrado el error con la posición real del robot. Para poder calcular ese error necesitamos un sistema que obtenga de forma autónoma la posición real del robot, a esto se denomina un sistema de verdad absoluta o "ground truth".

El sistema de ”verdad absoluta” proporciona la posición real del robot mediante la detección de un patrón visual (figura B.2) colocado encima del robot mediante un sistema de cámaras. Permite aportar la información necesaria para poder comparar la estimación de la posición del robot, calculada por el módulo de auto-localización, con su posición real.

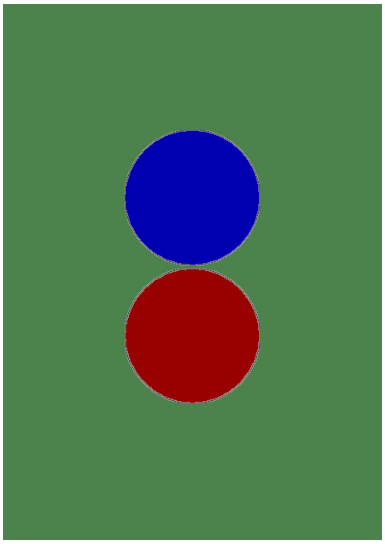

Figura B.2: Patrón situado encima del robot.

El sistema que hemos construido utiliza como base el paquete *mezzanine*[1], modificado para adaptarse a las necesidades del problema a resolver. Este software analiza las imágenes procedentes de una cámara cenital y calcula la posición del robot dentro del campo de juego.

La aplicación *mezzanine* (figura B.5) se compone de un servidor que filtra las imágenes que recibe de la cámara y de una utilidad de calibración. Con la utilidad de calibración se puede definir la parte de la imagen que se debe filtrar, los colores del patrón y establecer una serie de puntos que sirven para corregir la aberración de la lente (figura B.6).

El software *mezzanine* se ejecuta como un servicio que puede proporcionar la información de la posición real de varios robots simultánemente. Se pueden desarrollar aplicaciones que se comunican con este servicio a través de un mecanismo de memoria compartida para recibir esta información.

La información generada la hemos usado de dos maneras distintas:

- **Como sistema de localización**. La información se envía directamente al robot, como se muestra en el esquema de la figura B.3. El robot obtiene la información de su posición y la usa en lugar de la estimación generada por el módulo GM.

[1] http://playerstage.sourceforge.net/mezzanine/mezzanine.html

Este modo es especialmente útil para el desarrollo y depuración de comportamientos que necesitan usar información de localización. Si se depura, por ejemplo, el comportamiento simple de ”ir a una posición global”, es difícil distinguir cuándo los errores que se producen son debidos al algoritmo usado para generar las acciones o cuando son debidos a que la información de localización no es correcta. El sistema de verdad absoluta permite desarrollar comportamientos simples independientemente de la bondad del sistema de localización.

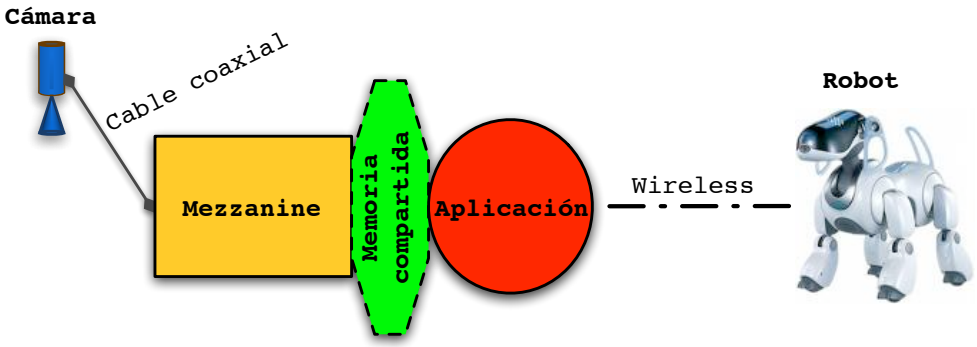


Figura B.3: Comunicación con *mezzanine* por memoria compartida.

- **Almacenamiento en ficheros de ”log”**. La información se puede almacenar en disco, junto con una marca de tiempo, y se puede usar por el sistema de reproducción que se describirá en la sección B.3.

### B.2.1. Instalación hardware

El esquema del sistema de ”verdad absoluta” desarrollado para evaluar los algoritmos propuestos en esta tesis puede verse en la figura B.4. Consta de dos computadoras a las que se conectan sendas cámaras cenitales. Cada una de estas computadoras ejecutan el software *Mezzanine*. Un proceso envía a una tercera computadora la posición del robot calculada por cada cámara. El motivo de usar dos cámaras es que cada una de las cámaras es capaz de cubrir solo la mitad del campo de juego, por restricciones de altura y de resolución.

Para que el software *Mezzanine* sea capaz de detectar al robot en una imagen, éste lleva colocado en su parte de arriba un patrón fácilmente detectable mediante

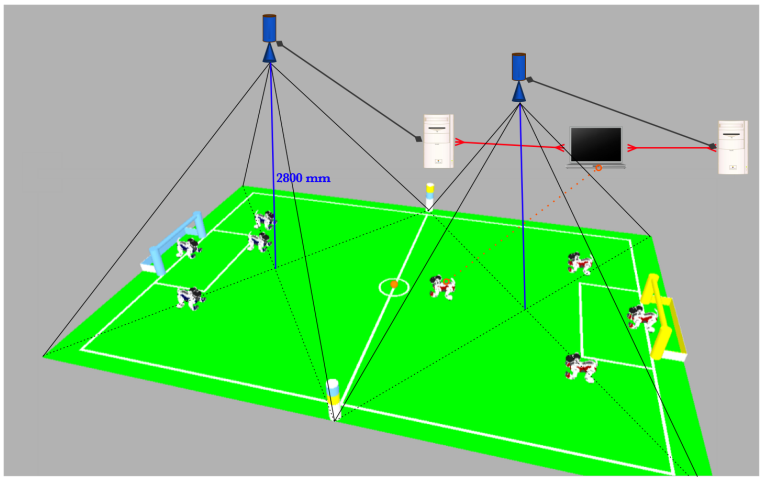


Figura B.4: Sistema de ”verdad absoluta”

el filtrado de color de la imagen. Los círculos de este patrón son de unos 3 cm de diámetro. La razón de necesitar dos puntos es para calcular la orientación del robot.

### B.2.2. Corrección y verificación del sistema

La generación de la información sobre la posición del robot se realiza en dos fases. En la primera fase, el software *Mezzanine* calcula, a partir de la detección de la marca visual, la posición de robot. Esta información no está corregida adecuadamente para tener en cuenta la aberración de la lente. Por esta razón realizamos una segunda fase, en la que se corrige esta información.

En la fase de corrección se parte de la posición $(x_m, y_m)$ que genera el software *Mezzanine*. Para obtener la posición corregida $(x_c, y_c)$ se aplica las siguientes ecuaciones:

$$x_c = ax_m^2 + by_m^2 + cx_m^2 + dy_m^2 + e \tag{B.1}$$

$$y_c = fx_m^2 + gy_m^2 + hx_m^2 + iy_m^2 + j \tag{B.2}$$

Los factores de corrección $a, b, c, d, e, f, g, h, i$ y $j$ se obtuvieron manualmente mediante un proceso de calibración. En este proceso se marcaron 210 puntos en diferentes zonas del campo sobre el campo (figura B.7) cuya posición real es cono-

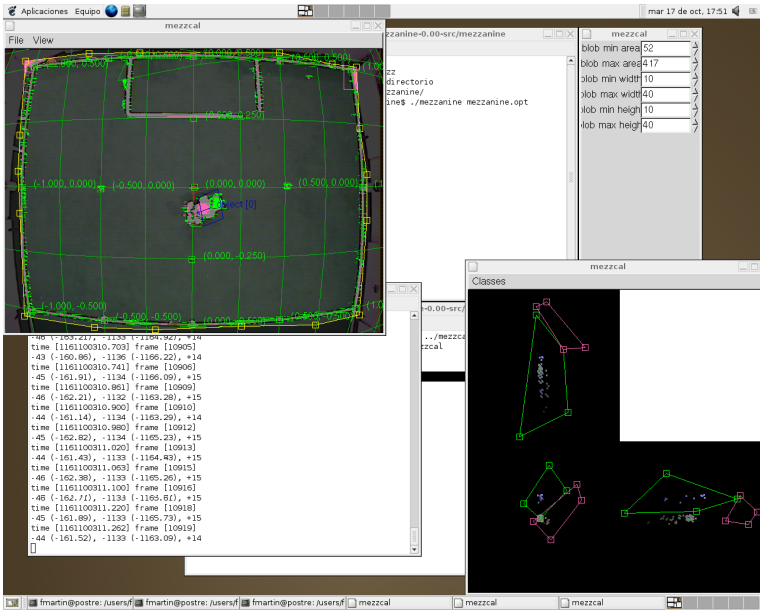


Figura B.5: Aplicación *mezzanine*.

cida. Los factores de corrección se obtienen resolviendo los sistemas planteados en las ecuaciones B.3 y B.4 por un método de mínimos cuadrados, ya que son sistemas sobredimensionados. En estos sistemas los puntos $(x_{r,i}, y_{r,i})$ corresponde a la posición real del punto sobre el campo de juego y $(x_{m,i}, y_{m,i})$ es la posición que genera *Mezanine*.

$$\begin{pmatrix} x_{r,1} \\ x_{r,2} \\ \vdots \\ x_{r,210} \end{pmatrix} = \begin{pmatrix} x_{m,1}^2 & y_{m,1}^2 & x_{m,1} & y_{m,1} & 1 \\ x_{m,2}^2 & y_{m,2}^2 & x_{m,2} & y_{m,2} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{m,210}^2 & y_{m,210}^2 & x_{m,210} & y_{m,210} & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} \tag{B.3}$$

$$\begin{pmatrix} y_{r,1} \\ y_{r,2} \\ \vdots \\ y_{r,210} \end{pmatrix} = \begin{pmatrix} x_{m,1}^2 & y_{m,1}^2 & x_{m,1} & y_{m,1} & 1 \\ x_{m,2}^2 & y_{m,2}^2 & x_{m,2} & y_{m,2} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{m,210}^2 & y_{m,210}^2 & x_{m,210} & y_{m,210} & 1 \end{pmatrix} \begin{pmatrix} f \\ g \\ h \\ i \\ j \end{pmatrix} \tag{B.4}$$

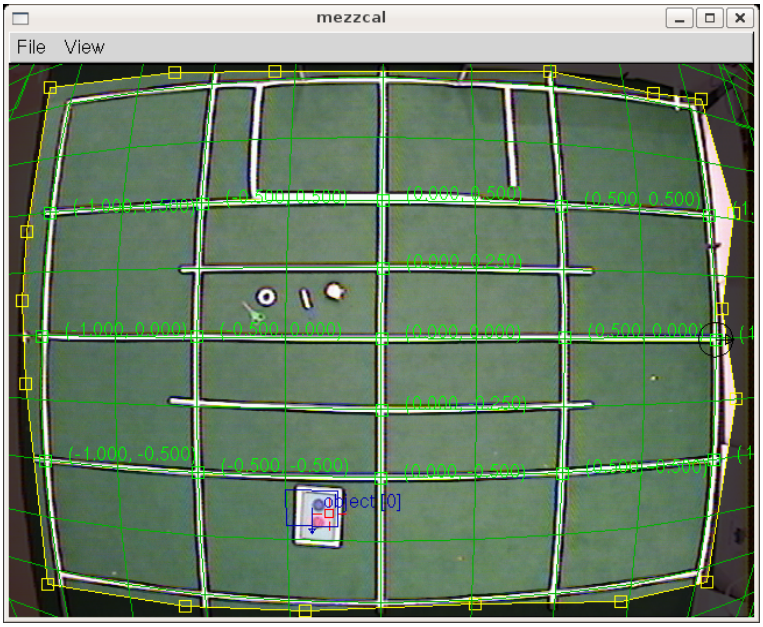


Figura B.6: Establecimiento de los puntos de control para la corrección de la aberración de la lente.

Una vez obtenidos los factores de corrección, se realizó un test para validarlos. Para esta validación situamos 20 puntos sobre el campo, cuya posición real es conocida (figura B.8), y comprobamos el valor obtenido tras la doble corrección. Los resultados se muestran en la tabla B.1. Las posiciones reales son $x_r$ e $y_r$. Las que indica el mezzanine son $x_m$ e $y_m$. $distancia$ es la distancia euclídea entre la posición real y la obtenida por el sistema de ”verdad absoluta” y nos indica el error que se comete en la detección.

Como se puede apreciar en la tabla, el error medio es de $39,24$ mm., y el error máximo es de 64,66 milímetros. Estos errores son suficientemente reducidos para considerar que el sistema de ”verdad absoluta” es válido para medir la precisión de los algoritmos de auto-localización desarrollados en esta tesis.

## B.3. Reproductor

Anteriormente se presentó un simulador capaz de generar las entradas necesarias al módulo de localización. Pero aún así, probar un algoritmo de localización

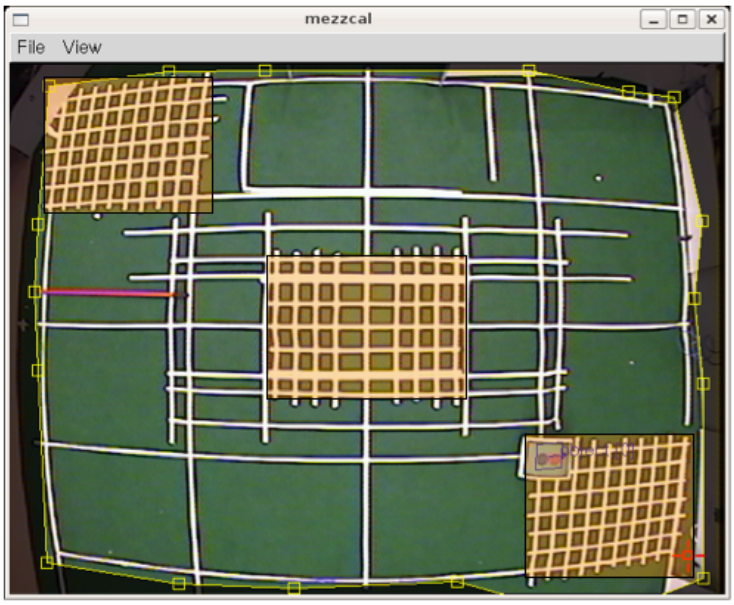


Figura B.7: Establecimiento de los puntos de control adiciones para la segunda fase de la corrección de la aberración de la lente.

| nº | $x_r$ | $y_r$ | $x_m$ | $y_m$ | $distancia$ |
|---|---|---|---|---|---|
| 1 | 1350 | 2362 | 1350 | 2340 | 22 |
| 2 | 1575 | 2531 | 1550 | 2490 | 48,02 |
| 3 | 1575 | 2193 | 1540 | 2190 | 35,128 |
| 4 | 1125 | 2193 | 1170 | 2210 | 48,1 |
| 5 | 1125 | 2531 | 1170 | 2510 | 49,658 |
| 6 | -450 | 333 | -490 | 340 | 40,6 |
| 7 | -225 | 500 | -250 | 500 | 25 |
| 8 | -225 | 166 | -240 | 190 | 28,3 |
| 9 | -675 | 166 | -730 | 200 | 64,66 |
| 10 | -675 | 500 | -720 | 500 | 45 |
| 11 | -1350 | 1008 | -1350 | 1030 | 22 |
| 12 | -1125 | 1179 | -1140 | 1190 | 18,6 |
| 13 | -1125 | 837 | -1140 | 850 | 19,85 |
| 14 | -1575 | 837 | -1540 | 860 | 41,88 |
| 15 | -1575 | 1179 | -1530 | 1200 | 49,66 |
| 16 | 450 | 1687 | 480 | 1700 | 32,7 |
| 17 | 675 | 1856 | 720 | 1880 | 51 |
| 18 | 675 | 1518 | 720 | 1540 | 50 |
| 19 | 225 | 1518 | 270 | 1540 | 50 |
| 20 | 225 | 1856 | 260 | 1880 | 42,44 |
| | | | | Media | 39,24 |
| | | | | Varianza | 13 |

Tabla B.1: Tabla de comparación entre las medidas (en mm) del mezzanine y las reales.

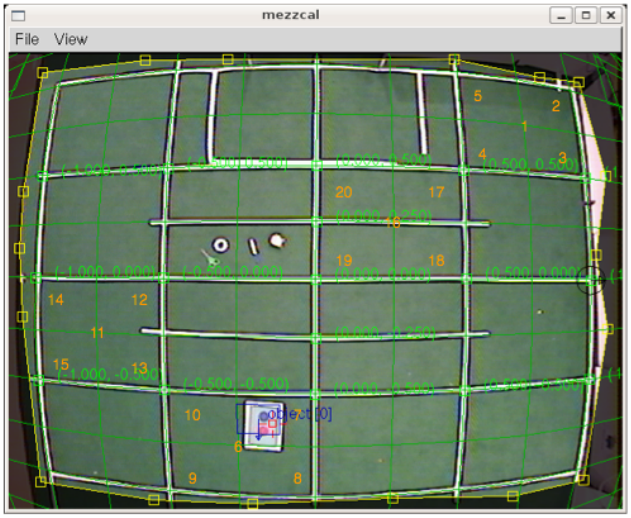


Figura B.8: En naranja los índices de los puntos usados para el test.

de un robot real en un entorno simulado no garantiza que funcione en un entorno real. El mundo real tiene una incertidumbre difícil de modelar. Por esta razón a este software de simulación se le ha añadido la capacidad de ejecutar algoritmos de auto-localización usando datos reales. A esto le hemos denominado ”reproducción”.

Con las mismas herramientas de monitorización y generación de resultados, ahora se le añade la posibilidad de que la información perceptiva provenga de la información captada por el propio robot t en ejecuciones reales. Además se puede incluir la información de la posición real del robot que acabamos de comentar.

### B.3.1. Información real de localización

El Reproductor necesita la información sensorial que se obtiene en el robot y la posición real del robot que aporta el sistema de verdad absoluta. Esta información está contenida en ficheros de ”log”. Estos ficheros de ”log” contienen toda la información necesaria para poder reproducir la ejecución de los algoritmos de auto-localización descritos en esta tesis y evaluar la información producida. Un ejemplo de un fichero de ”log” utilizado es el siguiente:

```
17581 GTRUTH -384 1105 0.417107
```

```
17585 ODOM-LPS 27.926 0.000 -0.005 37.57616 0.00000 0.00230  4063 -2.2780 0.01 129541 0.0000
        0.0000  2504 -0.8665 0.99 136557 117928.2734 0.0008  2541 -0.8900 0.99 136 549 107915.0547
        0.0014  1915 0.8500 0.97 136517 25616.4238 0.0071  3412 -1.8896 0.87 136353 102133.4922
        0.0198
17594 GTRUTH -384 1105 0.41710717608 GTRUTH -384 1105 0.417107
17682 ODOM-LPS 27.926 0.000 -0.005 37.57616 0.00000 0.00230  4084 -2.2848 0.01 129541 0.0000
        0.0000  2480 -0.8767 0.98 136557 117928.2734 0.0008  2517 -0.8994 0.98 136549 107915.0547
        0.0014  1858 0.9605 1.00 136585 25616.4238 0.0071  3427 -1.8994 0.86 136353 102133.4922
        0.0198
17691 GTRUTH -384 1105 0.41710717705 GTRUTH -384 1105 0.417107
```

Cada línea comienza con una marca del instante de tiempo en que fue generada. A continuación se encuentra una etiqueta que muestra que información hay a continuación. En el caso de estar etiquetada como `GTRUTH`, a continuación se encuentra la posición $(x, y, \theta)$ de robot. En el caso de estar etiquetada como `ODOM-LPS`, la información que hay a continuación es la odometría y la información perceptiva de los 5 elementos relevantes (dos porterías, dos balizas y la pelota).

El esquema mostrado en la figura B.9 muestra el proceso de creación de los ficheros de ”log”. Estos son, en detalle, los pasos que se llevan a cabo:

1. Una aplicación que se ejecuta en el ordenador de ”control” manda un mensaje a los ordenadores en los que se ejecutan las aplicaciones que reciben la información del servidor Mezzanine y al robot. En este mensaje se indica el nombre del fichero en que debe almacenarse la información.

2. A partir de ese instante, los sistemas que reciben el mensaje comienzan a guardar la información. El robot guarda la información de las líneas `ODOM-LPS` y el sistema de ”verdad absoluta” las líneas `GTRUTH` del fichero de ”log”.

3. Es importante que las marcas de tiempo del fichero de ”log” que se almacena en los distintos sistemas tengan una relación temporal. Si la marca de tiempo de una información se corresponde con otra información con una marca de tiempo superior a 1 segundo la información no sería útil, ya que el robot puede haberse desplazado considerablemente en ese tiempo. Para verificar esta propiedad de sincronía, en el momento en que cada sistema recibe el mensaje de inicio, se inicia contador de tiempo y se contesta con un mensaje de asentimiento.

Se ha comprobado que todos los mensajes de asentimiento llegan en menos de 25ms. De esta manera se garantiza que la información está sincronizada para nuestro uso.

4. Para finalizar el proceso de recogida de información, el ordenador de control envía un mensaje a todos los sistemas para que finalicen sus procesos de almacenamiento de información.

5. El fichero de cada sistema se combina en uno único.

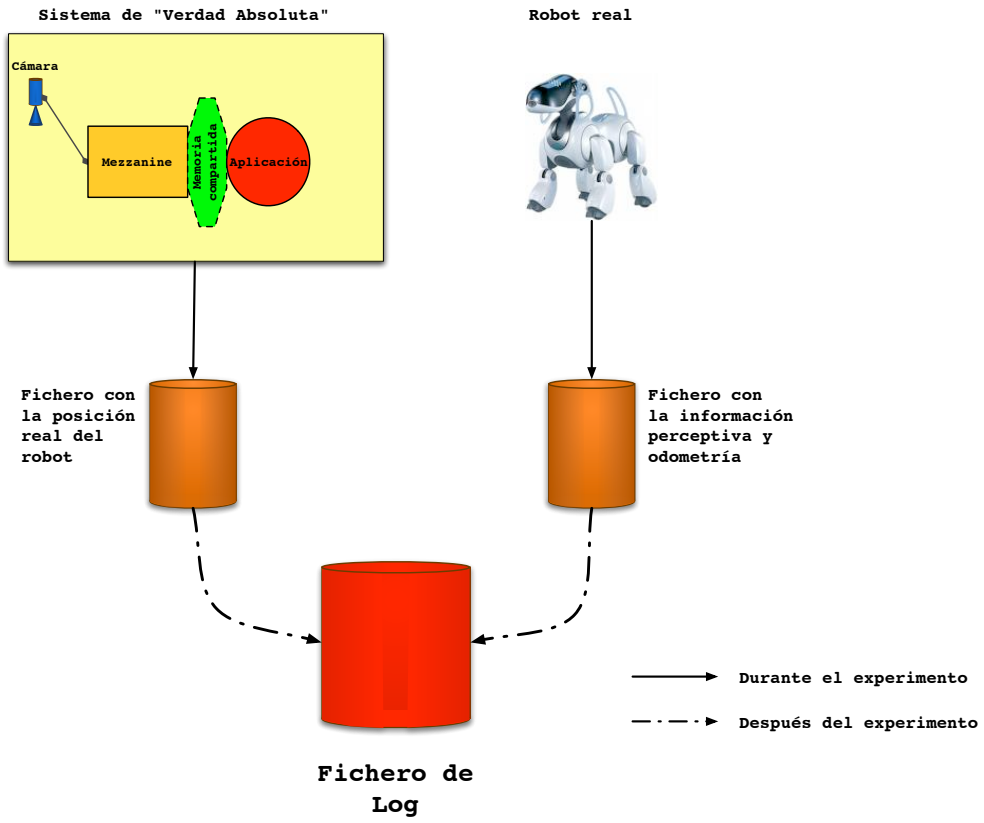


Figura B.9: Esquema de la creación de los ficheros de ”log”.

# BIBLIOGRAFÍA


[Ark98] R. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, MA., 1998.

[BCF+98] W. Burgard, A.B. Cremers, Dieter Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and Sebastian Thrun. The interactive museum tour-guide robot. In *Proc. of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

[BSZ00] P. Buschka, A. Saffiotti, and Z.Wasik. Fuzzy landmark-based localization for a legged robot. In *Proceedings of the International Conference on Intelligent Robots and Systems 2000*, páginas 1205–1210, Takamatsu, Japan, Octubre 2000.

[CEA06] José M. Cañas y Víctor Gómez Carlos E. Agüero, Vicente Matellán. Switch! dynamic roles exchange among cooperative robots. In *Proceedings of the II International Workshop on multi-agent robotic systems, MARS 2006*, Setúbal, Portugal., 2006.

[CKK96] Anthony R. Cassandra, Leslie Pack Kaelbling, and James A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile robot navigation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, páginas 963–972, 1996.

[Cro85] James L. Crowley. Navigation for an intelligent mobile robot. *IEEE Journal on Robotics and Automation*, 1(1), páginas 31– 41, Marzo 1985.

[Cro89] J. L. Crowley. World modeling and position estimation for a mobile robot using ultrasonic ranging. In *Proc. of the 1989 IEEE International Conference on Robotics and Automation (Vol. 2)*, páginas 674–680, Scottsdale, AZ, 1989.

[DFBT99] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA99)*, páginas 1322–1328, Mayo 1999.

[DIBB00] Darío Luis Delgado, Miguel Zamora Izquierdo, Luis M. Tomás Balibrea, and Humberto Martínez Barberá. An algorithm for mobile robot localization using a 2d laser range finder. In *RAAD:2000: Robotics in Alpe-Adria-Danube Region*, Maribor, Eslovenia, 2000.

[DW87] H. Durrant-Whyte. Uncertain geometry in robotics. *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, 4, páginas 851–856, Marzo 1987.

[FBDT99] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99).*, páginas 343–349, Julio 1999.

[FBT99] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11, páginas 391–427, 1999.

[GBFK98] J. Gutmann, W. Burgard, D. Fox, and K. Konolige. An experimental comparison of localization methods. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'98)*, volumen 1, 27-31, páginas 992 – 997, Octubre 1998.

[GF02] J.-S. Gutmann and D. Fox. An experimental comparison of localization methods continued. *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, 1, páginas 454–459, 2002.

[GTC01] Franck Gechter, Vincent Thomas, and François Charpillet. Robot localization by stochastic vision based device. In *The 5th World Multi-Conference on Systemics, Cybernetics and Informatics - SCI 2001 ? The 7th International Conference on Information Systems Analysis and Synthesis - ISAS 2001, Orlando, FL, USA*, Julio 2001.

[Gut02] J.-S. Gutmann. Markov-kalman localization for mobile robots. *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, 2, páginas 601–604 vol.2, 2002.

[GWN01] J.-S. Gutmann, T. Weigel, and B. Nebel. A fast, accurate, and robust method for self-localization in polygonal environments using laser-range-finders. *Advanced Robotics*, 14(8), páginas 651–668, 2001.

[HPMBS05] D. Herrero-Pérez, H. Martínez-Barberá, and A. Saffiotti. Fuzzy self-localization using natural features in the four-legged league. *Lecture Notes in Computer Science. Robocup 2004*, 3276, páginas 110 – 121, 2005.

[JBL96] J.Borenstein, B.Everett, and L.Feng. *Navigating mobile robots: Systems and techniques*. Ltd. Wesley, MA, 1996.

[Kal60] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME-Journal of Basic Engineering*, 82(Series D), páginas 35–45, 1960.

[KC99] Kurt Konolige and Ken Chou. Markov localization using correlation. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, páginas 1154–1159, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[Kl04] Jana Kosecká and Fayin li. Vision based topological markov localization. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, volumen 2, páginas 1481,1486, Barcelona (Spain), Abril 2004.

[KS98] Sven Koenig and Reid Simmons. Xavier: A robot navigation architecture based on partially observable markov decision process models. In R. Bonasso D. Kortenkamp and R. Murphy, editors, *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, páginas 91 – 122. MIT Press, 1998.

[LBM03] María E. López, Luis Miguel Bergasa, and M.S.Escudero. Visually augmented POMDP for indoor robot navigation. *Applied Informatics*, páginas 183–187, 2003.

[LBR+02] Andrew M. Ladd, Kostas E. Bekris, Algis Rudys, Lydia E. Kavraki, Dan S. Wallach, and Guillaume Marceau. Robotics-based location sensing using wireless ethernet. In *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*, páginas 227–238, New York, NY, USA, 2002. ACM Press.

[LDW91] J.J. Leonard and H.F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *Robotics and Automation, IEEE Transactions on*, 7(3), páginas 376–382, June 1991.

[Low03] D. Lowe. Distinctive image features from scale-invariant keypoints. In *International Journal of Computer Vision*, volumen 20, páginas 91–110, 2003.

[LV00] Scott Lenser and Manuela M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation, ICRA 2000*, volumen 2, páginas 1225–1232, San Francisco, CA, USA, 2000.

[LVRdS05] R. Lastra, P. Vallejos, , and J Ruiz-del Solar. Self-localization and ball tracking for the robocup 4-legged league. In *Proceeding of the 2nd*

*IEEE Latin American Robotics Symposium LARS 2005*, Sao Luis (Brazil), Sept 2005.

[MGCRJ04] F. Martín, González-Careaga, Cañas R., and V. J.M., Matellán. Programming model based on concurrent objects for the aibo robot. In *Actas de las XII Jornadas de Concurrencia y Sistemas Distribuídos, JCSD 2004*, páginas 367–380, Navas del Marqués, Ávila (España), 2004.

[Mor88] Hans Moravec. Sensor fusion in certainty grids for mobile robots. *AI Mag.*, 9(2), páginas 61–74, 1988.

[MS01] H. Martínez and A. Saffiotti. Thinkingcap-II architecture, 2001. en línea en http://ants.dif.um.es/~humberto/robots/tc2/.

[Nil84] Nils J. Nilsson. Shakey the robot. Technical Report 323, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Abril 1984.

[NPB95] Illah Nourbakhsh, Rob Powers, and Stan Birchfield. Dervish: An office-navigating robot. *AI Magazine*, 16(2), 1995.

[Pia95] Marek Piasecki. Global localization for mobile robots by multiple hypothesis tracking. *Robotics and Autonomous Systems*, 16(1), páginas 93–104, 1995.

[RB00] Stergios I. Roumeliotis and George A. Bekey. Bayesian estimation and kalman filtering: A unified framework for mobile robot localization. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation, ICRA 2000*, páginas 2985–2992, San Francisco, CA, USA, 2000.

[Saf97] A. Saffiotti. The uses of fuzzy logic in autonomous robot navigation. *Soft Computing*, 1(4), páginas 180–197, 1997. Online at http://www.aass.oru.se/~asaffio/.

[SGH+97] Reid Simmons, R. Goodwin, K. Haigh, S. Koenig, Joseph O'Sullivan, and Manuela Veloso. Xavier: Experience with a layered robot architecture. In *Agents '97*, volumen 8, páginas 22–33, 1997.

[SK95] Reid Simmons and Sven Koening. Probabilistic navigation in partially observable environments. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*, páginas 1080–1087, Montreal (Canada), Julio 1995.

[SKS05] Mohan Sridharan, Gregory Kuhlmann, and Peter Stone. Practical vision-based monte carlo localization on a legged robot. In *IEEE International Conference on Robotics and Automation*, páginas 3366– 3371, Abril 2005.

[SRK93] Alessandro Saffiotti, Enrique Ruspini, and Kurt Konolige. A fuzzy controller for flakey, an autonomous mobile robot. Technical Report 529, AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, Abril 1993.

[TBB+99] Sebastian Thrun, Maren Bennewitz, Wolfram Burgard, Armin B. Cremers, Frank Dellaert, Dieter Fox, Dirk Hahnel, Charles R. Rosenberg, Nicholas Roy, Jamieson Schulte, and Dirk Schulz. MINERVA: A tour-guide robot that learns. In *KI - Kunstliche Intelligenz*, páginas 14–26, 1999.

[TFBD00] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2), páginas 99–141, 2000.