



UNIVERSIDAD SIMÓN BOLÍVAR
INGENIERÍA DE LA COMPUTACIÓN

Planificación de Rutas y Control en Tiempo Real de un Robot Móvil

Por
KENNY RAFAEL VIVAS DÍAZ

Tutores:
Ivette Carolina Martínez
Jesús Miguel Ferrer

Proyecto de Grado
Presentado ante la Ilustre Universidad Simón Bolívar
como requerimiento parcial para optar al Título de
Ingeniero en Computación

Sartenejas, Julio de 2005

Planificación de Rutas Globales en Tiempo Real y Control de un Robot Móvil

por

Kenny Rafael Vivas Díaz

RESUMEN

En este trabajo se desarrolla una plataforma robusta para el control de un robot móvil, planificando su ruta en tiempo real para evitar colisiones contra obstáculos en un ambiente dinámico, mientras el robot se dirige a un objetivo específico. Para planificar la ruta se utiliza un modelo basado en un automata celular desarrollado en trabajos anteriores, optimizándolo para mejorar las rutas alcanzadas. Se desarrollaron los módulos de captura de video y reconocimiento de imágenes, planificación de rutas en tiempo real, comunicación inalámbrica con el robot y el prototipo usado para los experimentos.

Se logró desarrollar una arquitectura completa que permite a un robot holonómico desplazarse en un espacio controlado mientras seguía el rastro de una pelota, replanificando su ruta en tiempo real en espacios dinámicos y estáticos.

A Manuela...

...por tus eternos cuentos de Tío Tigre y Tío Conejo.

AGRADECIMIENTOS

Es difícil decidirse a la hora de dar gracias. Primeramente, y sin que suene a cliché, a Dios (se llame como se llame) por ponerme en el camino a tantas personas a quien ahora voy a agradecer. A Nilda y a Abisahí, mis papas, a Grey y Nani, mis hermanas, por su protección, su paciencia y su apoyo, en fin, por ponerme en este camino. A Juan Ramón Jiménez, por enseñarme la gran diferencia entre un profesor y un maestro. A Jesús Ravelo, por enseñarme el orgullo de ser quien se desea ser. A Jesus Miguel Ferrer, por ser más que un tutor un amigo y un guía durante tanto tiempo. A Miguel Castro, por poner en mis manos este proyecto, a Carolina Chang por abrirme la puerta al mundo de la robótica. A Luis Astorga, por enseñarme que hasta algo tan abstracto como un autómatas tiene un lado filosófico. A Carolina Martínez por ayudarme a llevar este barco a puerto.

A todos los que de alguna manera remaron conmigo, y a todo el que con un consejo o una idea contribuyó con este proyecto: Gelvis, Rosa, Tere, Nelson, Jonathan, César, Ciro, Paco, Julio, Fran, Diego, Gabriel, Pedro, Daniel, Jennie, Celso, David, David (Hongo), Ezequiel, Luis, Gabriela, Marynel, y los que faltan.

A Kare, por su apoyo, su dedicación, su esfuerzo, su paciencia, su ayuda, su tiempo, sus correcciones, sus consejos y silencios.

A todo el que creyó en este proyecto, por darme el apoyo necesario, y a todo el que no lo hizo, por darme aún más razones para culminarlo con éxito.

A todos, muchas gracias.

Índice General

Introducción.....	1
Capítulo 1 – Marco Teórico.....	5
1.1 Autómata Celular	5
1.1.1 Clasificación.....	6
1.1.2 Aplicaciones.....	7
1.2 Paradigmas de la Robótica.....	7
1.2.1 El Paradigma Jerárquico.....	9
1.2.2 El Paradigma Reactivo.....	11
1.2.3 El Paradigma Híbrido Deliberativo / Reactivo:.....	12
1.3 Navegación y Planificación de rutas.....	13
1.3.1 Planificación Topológica de Rutas.....	14
1.3.2 Planificación Métrica de Rutas.....	15
1.3.3 Algoritmos de planificación de rutas aplicados en RoboCup.....	16
Capítulo 2 – Arquitectura del Sistema.....	17
2.1 Módulo de Visión.....	19
2.2 Módulo de Planificación.....	20
2.3 Módulo de Acción.....	20
Capítulo 3 – Captura y procesamiento de imágenes.....	22
3.1 Calibración de imágenes.....	24
3.2 Procesamiento de la imagen:.....	24
3.3 Representación del Cspace.....	25
Capítulo 4 – Autómata Celular aplicado a planificación de rutas.....	28
4.1 Pasos Previos.....	28
4.1.1 Reconfiguración del Cspace.....	29
4.1.2 Posición del robot.....	29
4.2 Aplicación del autómata celular.....	30
4.2.1 Implementación.....	31
4.2.2 Escogencia de la ruta calculada.....	32
4.2.3 Ángulo de aproximación.....	34
4.3 Envío de las intrucciones al robot.....	35
Capítulo 5 – Módulo de a bordo.....	37
5.1 Robot Prototipo.....	38
5.2 Navegación del robot.....	38
Capítulo 6 – Hardware y Software utilizados	40
6.1 Robot.....	40
6.2 Cámara de Video	42

6.3 Computadores utilizados.....	43
6.4 Ambientes de desarrollo.....	43
Capítulo 7 – Experimentos y Resultados.....	46
7.1 Experimentos.....	48
7.1.1 Planificación sin obstáculos.....	48
7.1.2 Planificación con obstáculos.....	48
7.1.3 Planificación con objetivo móvil sin obstáculos.....	48
7.1.4 Planificación con objetivo estático y obstáculos móviles.....	49
7.1.5 Planificación con obstáculos y objetivo móvil.....	49
7.2 Resultados.....	49
7.2.1 Planificación sin obstáculos.....	49
7.2.2 Planificación con obstáculos.....	50
7.2.3 Planificación con objetivo móvil sin obstáculos.....	50
7.2.4 Planificación con objetivo estático y obstáculos móviles.....	52
7.2.5 Planificación con obstáculos y objetivo móvil.....	53
Capítulo 8 – Conclusiones y recomendaciones.....	55
8.1 Objetivos logrados.....	55
8.2 Aportes.....	56
8.3 Recomendaciones.....	57
Apéndice A – Manual de Usuario.....	59
Bibliografía.....	61

Índice de Tablas

Tabla 4.1	Estados de los autómatas finitos.....	30
Tabla 4.2	Significado de los comandos enviados al robot.....	35
Tabla 6.1	Ficha Técnica de la HandyBoard.....	42
Tabla 6.2	Ficha Técnica del prototipo.....	42
Tabla 7.1	Valores promedio de los intervalos de reconocimiento de colores en cancha.....	47
Tabla 7.2	Resultados del experimento 1.....	49
Tabla 7.3	Resultados del experimento 2.....	50
Tabla 7.4	Resultados del experimento 3.....	53
Tabla 7.5	Resultados del experimento 4.....	54

Índice de Figuras

Figura 2.1	Arquitectura del sistema.....	19
Figura 3.1	Representación del Cspace.....	26
Figura 3.2	Ángulo de orientación del robot.....	27
Figura 4.1	Retrabajo del Cspace.....	29
Figura 4.2	Ejemplo de configuración final del autómata celular.....	31
Figura 4.3	Ruta calculada por el Autómata	32
Figura 4.4	Escogencia de la próxima celda a girar.....	33
Figura 4.5	Ángulo de aproximación.....	34
Figura 6.1	Robot prototipo.....	41
Figura 6.2	Detalles del robot prototipo / HandyBoard.....	41
Figura 7.1	Ambiente del experimento.....	47

INTRODUCCIÓN

En el campo de la robótica móvil se conoce como planificación de rutas (en adelante PP, por las siglas en inglés de *path planning*) al problema de conseguir una trayectoria libre de colisiones a través de un conjunto de obstáculos desde un punto inicial a otro punto final [3]. Dicho problema es ampliamente estudiado en robótica y es aplicado como parte principal en la solución de tareas más complejas. Ya sea que se diseñe un robot de salvamento en labores de rescate, un explorador en la superficie de Marte o un jugador de fútbol robótico, se deben implementar sistemas que les permitan trasladarse en ambientes con obstáculos para cumplir sus tareas específicas.

Una técnica de PP depende principalmente del tipo de sensores que posea el robot (a bordo o fuera del cuerpo del mismo), ya que éstos son los encargados de recoger información del ambiente que lo rodea, y a partir de estos datos el robot es capaz de construir un modelo abstracto del espacio en el que se desenvuelve. No toda la información disponible es relevante para la planificación de rutas: Mientras más

datos introduzcamos a nuestro sistema más variables tendremos que controlar, siendo más difícil dirigir un robot al objetivo. Los sensores pueden ser desde detectores de tacto hasta avanzados sistemas de visión, GPS y radares que permiten tener un rango de alcance mucho mayor [19].

Otro punto a tomar en cuenta es si el algoritmo de PP puede o no ejecutarse en tiempo real. Si se va a planificar una ruta en un ambiente estático es preferible crear la ruta para luego seguirla hasta que se llegue al objetivo. En cambio, si se precisa de respuestas rápidas en un ambiente dinámico, se deben utilizar técnicas que permitan la reprogramación de una ruta en tiempo real.

El objetivo de este trabajo de investigación es el diseño e implementación de un sistema capaz de planificar rutas en tiempo real para un robot móvil, usando para ello la visión global proveniente de una cámara de video. Para ello se utiliza una técnica para PP en tiempo real basada en un autómata celular, que fue anteriormente publicada en [4], introduciendo mejoras en la detección de objetos por visión, y optimizando las heurísticas de selección de caminos posibles, además de implementar algoritmos de detección de objetos mediante video, sistemas y protocolos de comunicación por radio para el control del robot y la fabricación de un prototipo para pruebas.

“Un autómata celular es un sistema dinámico discreto en el espacio y en el tiempo, que opera de manera uniforme, compuesto por un arreglo de células que interactúan entre sí siguiendo ciertas reglas definidas” [9]. La aplicación de dichos sistemas es bastante amplia, desde la simulación fenómenos físicos hasta replicación

de procesos biológicos, entre muchos otros.

El comportamiento del autómata definido para este trabajo permite la respuesta en tiempo real frente a los cambios bruscos que pueden ocurrir en ambientes dinámicos, como por ejemplo en un partido de fútbol robótico. Dentro de una cancha se da lugar a la interacción de al menos 4 agentes robóticos persiguiendo una bola e intentando lograr el objetivo de introducirla en el arco contrario.

Los capítulos que componen este documento están organizados de la siguiente forma:

En el capítulo 1 se presenta un breve resumen sobre los conceptos principales, técnicas e implementaciones más conocidas de PP, una definición detallada sobre autómatas celulares y una explicación sobre los paradigmas de la robótica y navegación, conceptos estudiados para desarrollar el presente trabajo.

En el capítulo 2 se presenta una visión global de la arquitectura desarrollada, sus componentes y la relación entre ellos.

En el capítulo 3 se detalla el proceso de captura de imágenes, la identificación de objetos y la creación de un mapa de estados que constituirá la entrada del autómata en la próxima etapa.

El capítulo 4 describe detalladamente la técnica de PP usando un autómata celular.

El capítulo 5 se centra en la especificación del prototipo robótico, sus componentes y el programa de a bordo, así como en la interfaz de comunicación que permite controlarlo inalámbricamente.

En el capítulo 6 se explican los ambientes de desarrollo y los detalles técnicos del hardware utilizado en este trabajo de investigación.

El capítulo 7 expone los resultados de la aplicación de la técnica utilizada y la metodología de las pruebas implementadas.

Finalmente el capítulo 8 contiene las conclusiones y algunas recomendaciones pertinentes.

Capítulo 1

MARCO TEÓRICO

1.1 Autómata Celular

Introducido en 1940 por John von Neuman y Stanislaw Marcin Ulam, el concepto de Autómata Celular (en adelante, AC) define “una colección teóricamente infinita de autómatas finitos interconectados” [9]. Cada uno de estos autómatas finitos puede entenderse como una célula en un gran sistema interrelacionado. El comportamiento o evolución de un AC es discreto en el espacio y el tiempo.

Según la definición de Neuman [9], un AC tiene las siguientes características:

- Un arreglo n-dimensional dividido en celdas, llamado *Espacio Celular*.
- Un autómata finito con estados definidos en cada celda del *Espacio Celular*. En el modelo de Neuman cada célula contiene una copia del mismo autómata, aunque se pueden definir AC con distintos autómatas finitos en el mismo espacio celular.

La unión entre celda y autómeta es lo que se define como *célula*.

- Cada célula tiene asociado un conjunto de células llamado *vecindad*. La forma, definición y composición de dicho conjunto es definida al diseñar el autómeta y puede o no contener a la célula a la cual se asocia.
- El estado de cada célula en un tiempo $t + 1$ viene definido unívocamente por el estado de dicha célula y el de las que componen su vecindad en el tiempo t , según la función denominada función de transición o evolución.
- Un estado v_0 , definido como “estado silente” o inicial, para cada autómeta finito.
- Una configuración inicial para todo el Espacio Celular.

1.1.1 Clasificación

Según Stephen Wolfram [27] los AC se clasifican, de acuerdo a su evolución a partir de un estado “desordenado”¹, en las siguientes categorías:

- Homogéneos: El AC evoluciona hasta que casi todos sus autómetas finitos tienen el mismo estado.
- Regulares: El AC evoluciona hasta producir un conjunto de estructuras conocidas como *osciladores*. Los osciladores son configuraciones fijas de un subconjunto de celdas del espacio celular.
- Caóticos: La evolución resulta en patrones caóticos y/o impredecibles.
- Complejos: La evolución produce grupos de células que pueden permanecer activas (o “vivas”) por un tiempo infinito.

Para simplificar, en este trabajo se hablará de autómetas celulares de una

¹ Los estados de los autómetas finitos son asignados aleatoriamente.

dimensión, si su espacio celular se representa como un arreglo unidimensional de células, dos dimensiones, si el espacio celular es una matriz, etc.

1.1.2 Aplicaciones

El interés principal de Neuman al estudiar los AC fue su aplicación en la organización de sistemas biológicos (reproducción de células, etc.) y el problema de la autorreplicación de sistemas [7] (ej. Sistemas que producen otros sistemas). En la actualidad se encuentran estudios y aplicaciones basadas en AC para la investigación en campos tan disímiles como sistemas ecológicos, modelos de multiagentes, procesamiento de imágenes, sistemas de difusión de ondas. Ejemplos de las aplicaciones antes nombradas pueden encontrarse en [9].

En particular, en el presente trabajo se discute un AC para resolver el problema de planificación de rutas en tiempo real de un robot móvil, solución propuesta anteriormente por Bering, Bracho, Castro y Moreno en [4].

1.2 Paradigmas de la Robótica

La palabra robot deriva del checo *robota* que significa sirviente. Fue introducida por Karel Capek, dramaturgo checo, en su obra *R.U.R*, estrenada en enero de 1921[19]. Desde entonces se ha asociado al robot con una máquina humanoide capaz de desempeñar tareas humanas. Poco a poco, conforme la computación y la electrónica fueron avanzando más y más robots fueron integrados a la vida cotidiana. En los años 60 aparecen los primeros brazos robóticos en industrias ensambladoras fabricados por la empresa Unimation [18], realizando tareas repetitivas y bien definidas. En la actualidad la robótica puede incluso ayudarnos a alcanzar y explorar otros planetas

[23].

El primer modelo a seguir por los diseñadores de robots fue replicar el comportamiento y funciones del hombre. Hoy en día, lejos de parecerse a los humanos, los robots se parecen más a herramientas: brazos soldadores en una línea de producción, exploradores en otros planetas, naves no tripuladas de reconocimiento, ejemplos que se pueden encontrar en [18,19]. Todos estos aparatos son robots cuya apariencia no es ni remotamente humanoide. Al no poder replicar el comportamiento de seres más evolucionados como los primates superiores, se pensó entonces en seres vivos más simples, como los insectos [19], para así ganar destrezas en el camino de crear un robot cada vez más parecido al ser humano. Pero incluso allí la ciencia probó estar todavía bastante lejos de crear un robot con las mismas características de un ser vivo.

Según Murphy[19], en robótica existen tres paradigmas generales en el diseño de robots, todos ellos basados en las primitivas de **Sentir** (Obtener información del medio a través de los sensores), **Planificar** (Tomar decisiones basadas en la información que se posea sobre cierta acción a realizar) y **Actuar** (llevar a cabo la tarea). La manera como estas primitivas se priorizan en el diseño del robot viene definida por el paradigma en cuestión:

1.2.1 El Paradigma Jerárquico

En uso desde 1967 hasta la actualidad. Fue implementado por primera vez en Shakey de SRI [19]. Está definido por la secuencia Sentir – Planificar - Actuar. Cada uno de estos pasos es consecuencia directa del anterior. Se puede ejemplificar de la

siguiente forma: “Ver un objetivo, decidir la ruta a tomar, luego caminar por la ruta”. El robot luego de recibir de sus sensores la información del ambiente, planificará su próxima acción basado en un modelo creado con estos datos y luego la ejecutará. El crear un modelo del ambiente en base a la información de los sensores conlleva varios problemas importantes: qué tanta información necesita tener el robot para llevar a cabo con éxito su tarea (esto depende de los sensores que posea el robot y su capacidad de procesar la información obtenida), y qué tan parecido al mundo real debe ser el modelo generado.

1.2.2 El Paradigma Reactivo

El paradigma reactivo nació en la década de los 80, cuando Arkin, Brooks y Payton[19] publicaron sus trabajos sobre arquitecturas robóticas ligeras y orientadas a seres vivos menos complejos como los insectos. A diferencia del paradigma jerárquico, el reactivo no precisa de ninguna planificación para generar una acción determinada [19]. Su basamento biológico es simple: la mayoría de los seres vivos poseen sensores de algún tipo (antenas, papilas, ojos, tacto, oído, etc.) y sus conductas y reacciones están definidas por los estímulos que reciba del mundo exterior a través de dichos sensores. En pocas palabras las acciones son resultados directos de los estímulos que reciben los sensores. No se requiere planificación previa a la acción, ya que los sensores están directamente conectados a las acciones a tomar. Se sigue el patrón Sentir – Actuar.

Una aplicación interesante de este paradigma es la navegación por medio de Campos Potenciales [14], que permite al robot viajar por un ambiente evitando

obstáculos y dirigiéndose a uno o varios objetivos, moviéndose entre campos de atracción (objetivos) o de repulsión (obstáculos).

Aunque el paradigma reactivo ha generado resultados interesantes, es evidente que el no planificar ninguna acción es algo limitativo a la hora de ejecutar tareas específicas.

1.2.3 El Paradigma Híbrido Deliberativo / Reactivo

Propuesto por Arkin y Mackenzie en 1988 con AuRA[19], es una mezcla de los dos anteriores. En este paradigma, el robot toma la información de los sensores y la usa para decidir cómo descomponer una tarea en otras más pequeñas. Cada una de estas subtareas tiene una acción asociada que debe realizar el robot, y se van ejecutando siguiendo el paradigma reactivo.

Este esquema permite una reacción mucho más rápida en tiempo de ejecución que el paradigma jerárquico, a la vez que permite planificar las acciones de una manera lógica, de forma tal que el comportamiento del robot no es puramente reactivo.

Actualmente, como es de esperarse, el paradigma híbrido es el más estudiado [19]. Tiene cierta analogía con los sistemas nerviosos de seres vivos evolucionados, como el ser humano, que poseen sistemas distintos para las actividades conscientes y las acciones reflejo.

1.3 Navegación y Planificación de rutas

La navegación es un problema básico dentro de la robótica móvil: la habilidad de trazar una ruta desde el punto de partida hasta un punto destino es crítica para poder moverse de un punto a otro [10]. La mayor cualidad de un robot móvil es su capacidad

de trasladarse en su medio, a diferencia de un manipulador robótico que debe permanecer en un sitio fijo. La movilidad de un robot introduce nuevos retos de implementación y diseño, ya que puede encontrarse con obstáculos e incluso perderse en el camino.

Según Robin R. Murphy [19], existen 5 criterios importantes a la hora de calificar una técnica de planificación y navegación:

- **Complejidad:** Define la eficiencia del algoritmo y la capacidad que el robot tenga de ejecutarlo.
- **Representación del Terreno:** ¿Con cuánto detalle es necesario representar el ambiente que rodea al robot? Por ejemplo, si se tiene un robot móvil que se desenvuelve en un ambiente plano y controlado quizás no sea necesario captar y representar la orientación de un área en particular, pero si es un explorador en terreno irregular esta información es crucial para el éxito de la operación.
- **Representación de las restricciones físicas del robot:** Los robots no son parte de un algoritmo computacional, son máquinas. Y como tales tienen restricciones en cuanto forma, tamaño, mecanismos, etc. La característica más importante a tomar en cuenta en una planificación de rutas es si el robot es *holonómico* o no. Se define como holonómico a un robot capaz de girar mientras se mantiene sobre un mismo punto [19]. Modelos como el robot *Khepera* [13], que son de forma redonda y giran sobre su propio eje central, son un ejemplo de robot holonómico. Un robot no holonómico es un reto mayor a la hora de planificar una ruta: no se le puede hacer girar, sino describir curvas que son más complicadas de planificar y controlar.

- **Compatibilidad con el Paradigma Reactivo:** El planificar una ruta es una característica del paradigma jerárquico visto anteriormente, pero puede ser implementada mediante el paradigma híbrido, haciendo que el robot ejecute acciones de manera reactiva mientras que las acciones son planificadas paralelamente. Se desea acercarse al paradigma reactivo puesto que permite mayor rapidez de respuesta y simplicidad de implementación.
- **Soporta replanificación:** Crucial a la hora de moverse por ambientes dinámicos. Un robot que planifica su ruta puede encontrar que el ambiente cambió luego de elegido su camino, por lo cual la ruta que está siguiendo puede ser la equivocada. Es evidente que se prefiera un algoritmo de planificación capaz de reaccionar frente a esos cambios y cambiar la ruta a tiempo.

Existen dos acercamientos importantes en materia de planificación de rutas:

1.3.1 Planificación Topológica de Rutas:

No precisa de la fabricación de un mapa, sino de la ubicación de puntos de control, denominados *landmarks* [16] que el robot debe seguir para completar una trayectoria mayor hacia su objetivo final. Dichos *landmarks* pueden ser cualquier objeto real en el ambiente. De esta forma una ruta a seguir por un robot se compone de pequeñas subrutas que empiezan y terminan en un *landmark*.

Dentro del robot, el mundo se puede ver como un grafo no dirigido cuyos nodos son los *landmarks* y sus aristas caminos posibles entre ellos. Se puede entonces escoger el camino usando técnicas de búsqueda sobre dicho grafo. Esto es lo que se conoce como el Método Relacional [19].

Otro método importante, el asociativo [19], se basa en que el robot puede reconocer los *landmarks* y así construir su ruta. La analogía con el ser humano es bastante cercana: Generalmente se siguen direcciones como “Gira a la derecha, en la tercera entrada, ve al fondo del pasillo hasta las escaleras”. Claramente se necesita que los *landmarks* sean distinguibles y percibibles por el robot, quien sigue un conjunto de instrucciones que provienen del conocimiento previo de los distintos objetos del ambiente que lo rodea. Este método se usa generalmente en arquitecturas que no permiten al robot tener una visión global de su espacio, sino una parte reducida de su entorno, y cuando los landmarks son *distinguibles* por el sistema. Son arquitecturas complejas, con gran peso en el reconocimiento (ya sea por visión, señales de radio, etc.) de los objetos que marcarán el camino del robot.

1.3.2 Planificación Métrica de Rutas:

A diferencia de la anterior, la planificación métrica tiene como objetivo conseguir una ruta mínima siguiendo parámetros de medición, como tiempo, distancia, etc. [19]. Su concepto principal es la representación del ambiente que rodea al robot (que de ahora en adelante se denominará como *Cspace*). De manera similar a la planificación topológica, el planificador convierte la ruta en pequeñas trayectorias, pero no dependen de *landmarks*, sino de puntos definidos en el *Cspace* conocidos como *waypoints* [19]. Los *waypoints* no siempre tienen correspondencia con objetos en el mundo real, como los *landmarks*, y generalmente tienen coordenadas fijas en el *Cspace*. Una gran desventaja de esta técnica (que también se puede observar en la planificación topológica) es el hecho de que se corre el riesgo de seguir una ruta sobre

un modelo del mundo que ya ha cambiado, o sea, que el modelo de ambiente que percibió el robot antes de planificar su ruta puede haber modificado al momento de ejecutar las acciones, por ejemplo si los obstáculos o el objetivo cambian de posición. Esto presenta un reto a los desarrolladores, que deben crear algoritmos capaces de reaccionar en tiempo real a estos cambios.

Existen dos componentes principales dentro de un planificador métrico: La representación del *Cspace* y el algoritmo que decidirá la ruta a seguir. El *Cspace* es generalmente representado como un grafo o una matriz cuyas celdas definen la forma de los obstáculos y zonas libres dentro del ambiente. Existe un tipo especial de matriz para representación del *Cspace*, los *Quadrees* [15], o matrices cuyas celdas contienen matrices mas pequeñas, que permiten detallar al *Cspace*, de la misma forma que se aumenta la resolución en una imagen para verla más nítida. Específicamente, dentro de las matrices se maneja el concepto de “grados de libertad” que son las celdas a las cuales un robot puede moverse en un momento dado. Por ejemplo, en representaciones planas del *Cspace* (una matriz de 2 dimensiones) una celda puede tener un máximo de 8 grados de libertad, las 8 celdas que la rodean.

Otras representaciones, como los *MeadowMaps* [2], convierten el *Cspace* en un conjunto de polígonos, cuyos vértices sirven como *waypoints*, aunque los algoritmos para generar los polígonos son bastante costosos, lo que hace que esta técnica no sea adecuada para ambientes dinámicos. Similares a éstos son los Mapas de Voroni [8], con la diferencia de que éstos se basan en la creación de una línea equidistante a todos los puntos del espacio en el que se encuentre el robot, conocida como línea de

Voroni. La intersección de varias de estas líneas forma los Vértices de Voroni [8], y el robot sigue la línea trazada cambiando de dirección en los vértices si es necesario. Esta técnica permite replanificación y es comúnmente usada para la representación de *Cspace*, generando un grafo usando las líneas y los puntos de corte entre ellas como aristas y nodos, respectivamente [19].

Los algoritmos de planificación basados en grafos son bastante conocidos en teoría de recorrido de grafos y árboles. El algoritmo de A*[15,20] es uno de los más populares y eficientes a la hora de conseguir caminos mínimos en tiempo reducido.

Entre los basados en representaciones mediante matrices destacan los algoritmos de propagación de ondas [21], semejantes a Campos Potenciales. La diferencia entre estas dos técnicas radica en su uso: los algoritmos de propagación generalmente se implementan para crear una “onda” (generalmente un valor numérico en cada celda del *Cspace*) que inicia en el punto donde se encuentra el robot y alcanza a los demás objetos del espacio. Una analogía es el sistema de navegación de muchos vehículos y seres vivos: Emiten una onda (de radio, de sonido, etc.) que se propaga desde su ubicación hasta chocar con los diferentes objetos de su entorno. Los campos potenciales generalmente se crean alrededor de obstáculos y objetivos, no en el robot mismo, y son usados en implementaciones reactivas.

En [4] se puede observar la implementación de un autómata celular usado para planificar rutas globales. Esa técnica es también estudiada e implementada en este trabajo.

1.3.3 Algoritmos de planificación de rutas aplicados en RoboCup

Un buen número de técnicas de path planning son actualmente utilizadas y mejoradas año tras año durante las competencias de RoboCup [26]. Este evento da la oportunidad de enfrentar en la vida real diferentes soluciones de diseño en el campo de la robótica. Esto trae como consecuencia la renovación e invención de nuevas técnicas mucho más eficientes. El objetivo de este trabajo es presentar una solución de path planning que pueda utilizarse en un ambiente parecido al de las competencias de RoboCup.

La mayoría de las soluciones propuestas actualmente en Robocup poseen una capa de estrategias que permite el reconocimiento de movimientos y la anticipación de jugadas y trayectorias (esquema que va más allá de los límites del presente trabajo), e implementan paradigmas reactivos e híbridos para lograr que los robots cumplan su objetivo [5,11,12].

La solución presentada en este trabajo difiere de las técnicas actualmente utilizadas en RoboCup [11,12] en el hecho de que un autómeta celular no precisa de un tiempo de entrenamiento ni depende del reconocimiento de patrones, como es el caso de las técnicas de redes neuronales [6].

Capítulo 2

ARQUITECTURA DEL SISTEMA

Para la implementación del sistema de planificación de rutas por control de visión se desarrolló una estructura basada en módulos, cada uno de ellos encargado de la realización de un conjunto de tareas específicas. Dicha arquitectura fue creada en base al esquema de RoboCup[25], publicado por la IEEE[24] con motivo del próximo concurso de Robótica para Estudiantes, evento que se repite año tras año y que tiene como motivo, entre las categorías de RoboCup Soccer, *“el crear para el año 2050 un equipo de robots capaz de enfrentarse en igualdad de condiciones a un equipo de seres humanos en un partido de fútbol”* [25].

Las categorías manejadas por RoboCup Soccer este año son: Very Small Size, Small Size, Medium Size y Four Leged. En particular el presente trabajo esta enmarcado en las reglas de la categoría Small Size. Las características de la

infraestructura propuesta por la IEEE para la liga de pequeños robots [25] se pueden resumir como sigue:

- Una cámara colocada perpendicularmente sobre la cancha capta las imágenes del juego y envía los datos a un servidor de visión
- Los objetos de la cancha poseen un color asociado y distinguible. Los robots poseen un diseño de colores en la parte superior. Este diseño no sólo se usa para diferenciarlo de los demás objetos de la cancha, sino también para conocer su orientación. El “balón” es una pelota de golf de color naranja y el fondo de la cancha es de un color que va desde el negro al verde pizarra.
- El servidor de visión, luego de captar imágenes y preprocesarlas, envía esa información a otras estaciones encargadas de las estrategias del juego, la planificación de rutas, etc.
- Una vez decidida la estrategia y los movimientos del equipo, se debe comunicar las órdenes a los robots en cancha por un medio inalámbrico, ya sean emisores de radio o una pequeña red WIFI.

Para los efectos del presente trabajo se implementó una arquitectura basada siguiendo las reglas de RoboCup [25]. No se implementó un sistema de inteligencia artificial que permitiera la realización de jugadas ya que esto escapaba a los límites del trabajo y se centraron esfuerzos en la implementación de un planificador de rutas en tiempo real.

Se usaron tres módulos principales, cada uno ejecutándose en un computador diferente. En la figura 2.1 se puede observar un esquema del sistema propuesto.

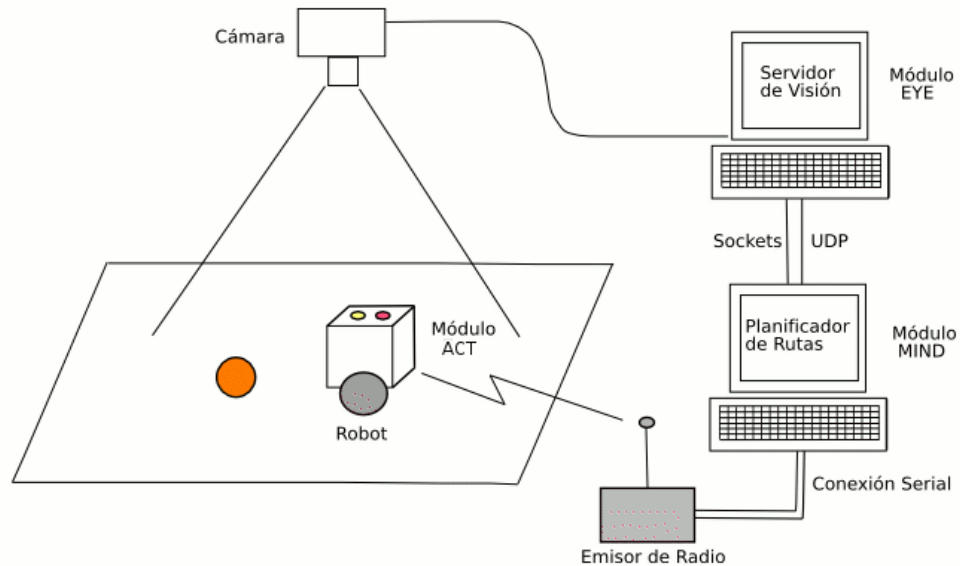


Fig. 2.1. Diagrama de la Arquitectura del Sistema.

2.1 Módulo de Visión

El módulo se denomina **EYE** y se encarga de capturar las imágenes provenientes de la cámara, clasificar los colores e identificar los elementos en la cancha (frente y retaguardia del robot, posición de la pelota y de los obstáculos). Para simplificar el sistema se consideró sólo un robot en cancha, siendo los demás objetos clasificados como obstáculos, exceptuando la pelota. Este módulo también es responsable de la creación del *Cspace*, que será luego la entrada para el planificador de rutas, y el cálculo de la orientación del robot.

El módulo será explicado en detalle en el capítulo 3.

2.2 Módulo de Planificación

El módulo de planificación se denominará **MIND**, e implementa un autómata

celular de dos dimensiones para generar una ruta en tiempo real desde el robot hacia la pelota. Para ello recibe por red el Cspace, la posición de pelota y obstáculos, y la posición y orientación del robot. El módulo MIND puede ejecutarse en una sola estación junto con el módulo EYE, colocando como dirección la misma estación en la que se ejecuta EYE (Esto se verá con mas detalle en el manual de usuario incluido como apéndice en este documento). Con estos datos inicializa la matriz de estados que será la configuración inicial del espacio celular del AC.

Un submódulo de comunicación es implementado dentro de MIND para enviar las acciones por un aparato de radio al robot.

El capítulo 4 discute ampliamente los detalles de implementación de este módulo.

2.3 Módulo de Acción

Denominado **ACT**, es el programa que se ejecuta dentro del robot y le permite la navegación y recepción de acciones del exterior. Posee métodos para girar, moverse en línea recta y detenerse, además del método para recibir las acciones en su receptor de radio. En el capítulo 5 se explica con más detalle la programación del código de a bordo.

Este esquema permite amplia escalabilidad, ya que puede ser implementado para que varios robots persigan la pelota simultáneamente y en tiempo real. Además de ello se puede aprovechar el procesamiento paralelo de sistemas separados e independientes para aumentar el rendimiento general del sistema. La separación por módulos también facilita el trabajo de investigación, al ser cada uno independiente del

otro se pueden desarrollar nuevas técnicas de reconocimiento de colores, planificación de rutas, diseños de robots e incluso sistemas de soporte a estrategias y control de multiagentes que permita a un equipo jugar coordinadamente contra otro.

Las estaciones en las que se ejecutan los módulos MIND y EYE están conectadas en una red local mediante un HUB, y su comunicación se implementó usando el protocolo UDP. En un ambiente como el descrito hay muy poca probabilidad de que los paquetes se pierdan dentro de la red, por lo cual no fue necesario implementar una comunicación mediante un protocolo más seguro como TCP/IP que provee un control de flujo. Además el protocolo UDP garantiza rapidez de recepción de paquetes en tiempo de ejecución.

Capítulo 3

CAPTURA Y PROCESAMIENTO DE IMÁGENES

Es crucial para cualquier planificador basado en el paradigma híbrido obtener un modelo del espacio en el cual se desenvuelve el robot, para proceder a planificar acciones sobre dicho modelo. Siguiendo las reglas de la IEEE para la liga de pequeños robots [25], se implementó un sistema de control por visión que permite observar y diferenciar los objetos que se encuentren dentro de la cancha, generando así un *Cspace*. Con esta motivación se implementa el módulo EYE de reconocimiento y ubicación de objetos por visión.

Existen 3 elementos principales a detectar dentro de la cancha:

- **La pelota:** Es de color naranja vivo y representa el objetivo al cual el robot debe llegar.
- **El robot:** Se debe diferenciar no solo su ubicación, sino la orientación que tiene

dentro de la cancha. Esto se logra al colocar dos parches de color distintos, uno en la parte frontal (amarillo) y otro en la trasera (rosa).

- **Los obstáculos:** Pueden ser barreras dentro de la cancha o incluso otros robots que intenten evitar que se llegue al objetivo. Todos son marcados con parches de color azul.

En líneas generales el algoritmo se ejecuta como sigue:

1. La cámara capta la imagen de la cancha, enviándola al módulo EYE a razón de 30 frames/segundo.
2. Se interpreta la imagen como un arreglo de $320 * 240$ píxeles y se recorre píxel por píxel. La identificación de objetos se hace por los valores RGB del píxel: se comparan con valores previamente obtenidos por una aplicación de calibración previa a la corrida del algoritmo. El uso del módulo de calibración se explica con detalle en el manual de usuario anexo a este documento.
3. Se almacena los valores de pelota, robot y obstáculos dentro de una matriz de $30 * 27$ celdas, que representa nuestro ambiente. Cada celda representa un espacio de 25 cms^2 , cubriendo un área de $150\text{cm} * 135\text{cm}$, aproximadamente la mitad de una mesa de ping pong.
4. La ubicación de cada objeto en la cancha es guardada dentro de la matriz y en el caso de la pelota y del robot, se almacenan aparte a fin de agilizar el algoritmo de planificación en el próximo módulo.
5. Los datos son almacenados en un paquete que luego será enviado por red a la siguiente estación.

3.1 Calibración de colores en la imagen

Previo a la corrida del módulo EYE se ejecuta una aplicación de calibración que permite reconocer los valores RGB de objetos en cancha manualmente. El programa consiste en una interfaz que permite, por medio de barras deslizantes, ajustar los valores mínimos y máximo de cada canal (R, G, B), mientras se resaltan las zonas de color que poseen esos rangos en una pantalla de video que presentaba a la cancha y los objetos en ella.

Al usar la aplicación de calibración se colocaron en la cancha cuadrados de cartulina con todos los colores a reconocer (rosa, amarillo, azul, naranja), distribuidos en grupos de a 4, en 9 lugares distintos al mismo tiempo. Se procedió a calibrar un color a la vez, hasta que se obtuviera el intervalo más ajustado para cada uno de los 4 colores. Esto se realizó para asegurar que el módulo EYE reconociera el color en cualquier lugar de la cancha.

Se encontró que los valores RGB son afectados por las condiciones de luz del ambiente, y se requirió de varias calibraciones para lograr valores óptimos que permitieran reconocer los objetos en toda la cancha. El programa de calibración se desarrolló usando *xcapttest* [24], con una interfaz grafica en Gtk+ [16].

3.2 Procesamiento de la imagen

La captura de la imagen se implementó usando como base la aplicación *xcapttest*. Dicha aplicación genera un arreglo de 320 * 240 píxeles que representa a la imagen captada por la cámara. Los píxeles se procesan separando sus canales R

(Rojo) G (Verde) y B (Azul), cuyos valores son comparados con valores previamente calculados con el calibrador para su reconocimiento.

La cámara utilizada por la arquitectura puede generar un máximo de 30 cuadros por segundo, y el algoritmo es capaz de procesar un promedio de 25 de ellas. Se enfocó la búsqueda del algoritmo en regiones específicas de la imagen para reconocer la cancha, usando sólo un subconjunto de píxeles del arreglo. Se evita así el ruido procedente de objetos colocados fuera de la cancha y del suelo que puedan ser captados por la cámara, obteniendo un área efectiva de búsqueda de 280 x 215 píxeles.

Para la implementación de este módulo se usó como base de diseño el implementado por el equipo CMUnited [5] campeón de RoboCup 98.

Sólo para la identificación de la ubicación del robot se implementó una funcionalidad que permite optimizar su búsqueda: Al reconocer la ubicación del robot en un frame F , se fija una ventana de búsqueda de 40 píxeles por lado en el frame $F+1$, centrada en la posición anterior. De esta manera no se pierden ciclos de procesador buscando el robot en toda la cancha. Si se diera el caso de que el robot no se encuentra en la ventana fijada, se recomienza la búsqueda por todo el arreglo de píxeles en la siguiente iteración. Se decidió no implementar esta optimización para la pelota, ya que sus movimientos pueden ser muy bruscos y no sería útil acotar su búsqueda.

3.3 Representación del Cspace

Al final de cada iteración, el algoritmo devuelve una representación del espacio

celular en una matriz de enteros. El valor de cada celda representaba su contenido:

- 999 Espacio vacío.
- 1 Posición de la pelota
- 2 Obstáculo.

La figura 3.1 da un ejemplo más claro de cómo se representa la imagen capturada por la cámara en una matriz de enteros. El color blanco indica un espacio vacío, el azul indica que estas casillas poseen el valor definido para obstáculo y así mismo para la posición de la pelota y el robot.

La posición del robot no se almacena en la matriz en este paso, sino es almacenada separadamente en el paquete a enviar, y su orientación es calculada usando la posición de sus dos parches de color, tomando como referencia un sistema de coordenadas centrado en la esquina superior izquierda de la cancha (Figura 3.2). Este dato también es enviado al planificador dentro del paquete.

Luego de inicializada, la matriz es colocada en el paquete de datos, el cual se envía por un socket al siguiente módulo del sistema.

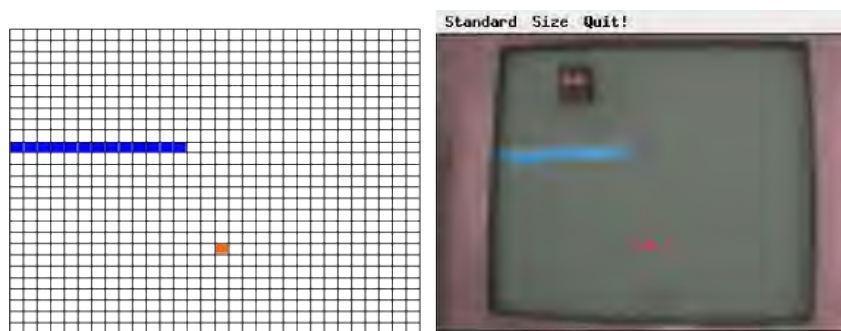


Fig. 3.1. Representación del Cspace dentro del algoritmo de visión. La imagen de la izquierda es la representación del Cspace percibido en la imagen de la derecha, captada por la cámara.



Fig. 3.2. Correspondencia de la orientación del robot con respecto a un sistema de coordenadas centrado en la esquina superior izquierda del robot.

Capítulo 4

AUTÓMATA CELULAR APLICADO A LA PLANIFICACIÓN DE RUTAS

El módulo MIND implementa la planificación de rutas utilizando un autómata celular de dos dimensiones. El algoritmo es suficientemente rápido para ejecutarse en tiempo real, siendo capaz de dirigir al robot en ambientes dinámicos, procesando un promedio de 20 frames/seg en la estación utilizada para el experimento (detalles de hardware serán explicados en el capítulo 6).

4.1 Pasos Previos

Luego de obtener las posiciones del objetivo, del robot y de los obstáculos, se procede a planificar la ruta necesaria. El módulo de visión EYE crea una matriz cuyas celdas poseen valores dependiendo de los objetos que representen: ya sean obstáculos, el robot, la pelota o espacios vacíos en la cancha. Sin embargo, el modelo

del espacio que se obtiene todavía no es el correcto, puesto que las celdas de la matriz son de 5 cm. de ancho, lo que puede generar caminos posibles cuyas dimensiones reales sean mucho menores al ancho real del robot (18 cm.), conduciendo a rutas erróneas. Es por eso que antes de planificar la ruta se debe preparar el Cspace para convertirlo en el Espacio Celular que será la entrada del autómeta:

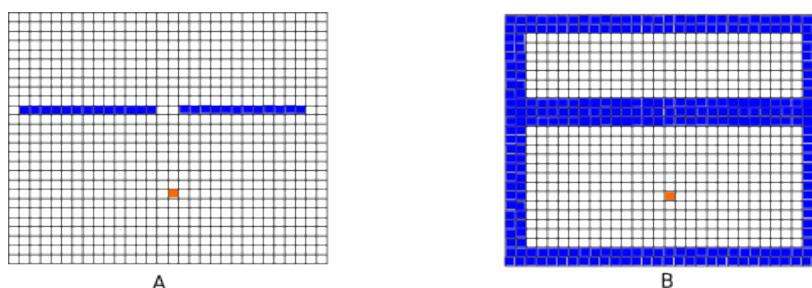


Fig. 4.1. En A se pueden ver posibles caminos a través de los obstáculos, pero el robot no puede pasar por un camino de menos de 20 cm. de ancho. El algoritmo de corrección (B) solventa esos errores

4.1.1 Reconfiguración del Cspace

Para corregir el error de percepción de las dimensiones reales del robot, se aumentan los bordes de los obstáculos en una casilla. Los bordes de la cancha también son aumentados para evitar que el robot se acerque demasiado a las barreras laterales. En la figura 4.1 se puede observar un ejemplo del retrabajo de la matriz. Este proceso es implementado usando un autómeta celular de dos dimensiones, definido en [4].

4.1.2 Posición del robot:

La posición del robot se extrae del paquete de datos obtenido del módulo EYE. La posición del mismo es marcada dentro de la matriz usando un código específico

que se explicara en el siguiente punto.

4.2 Aplicación del autómata celular

La representación del Cspace resultante luego del retrabajo de la matriz y el cálculo de las posiciones del robot sirve como configuración inicial del Espacio Celular de un autómata de dos dimensiones. Cada célula se define entonces como un autómata finito con los siguientes estados:

Estado	Significado
-1	Posición del Robot
999	Espacio vacío
-2	Obstáculo
1	Pelota
[2, 30]	Distancia Manhattan hacia la pelota

Tabla 4.1. Estados de los autómatas finitos

La vecindad de cada célula viene definida por las 8 celdas que la rodean. Una definición formal para este AC puede encontrarse en [4].

El objetivo del autómata es calcular el número de celdas que se deben recorrer desde la posición de la pelota y la casilla actual. El comportamiento del mismo se asemeja a una onda expansiva que parte desde la posición de la pelota hasta llegar a la vecindad del robot, si es posible alcanzarlo. La alcanzabilidad del objetivo se deduce al final de la evolución del autómata: Si la expansión incluye a la vecindad de la célula cuyo estado es -1 (O sea, la que contiene al robot), entonces existe un conjunto de células que definirán un camino válido entre el robot y la pelota. De lo contrario el objetivo es inalcanzable y no se envían instrucciones al próximo módulo.

El robot se representa como una casilla cuyo estado es el definido anteriormente, sin consideración alguna sobre la orientación del mismo.

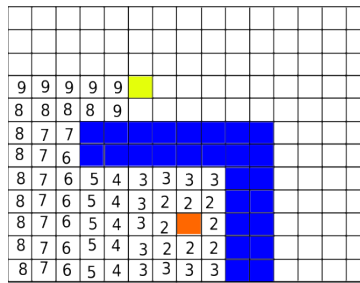


Fig. 4.2. Ejemplo de corrida del autómata. La casilla naranja es la pelota, la amarilla el robot y las azules son obstáculos.

4.2.1 Implementación

El algoritmo que implementa el comportamiento de este autómata es, en pseudo código:

```

Grid[ALTO] [ANCHO] se inicializa con el valor máximo en cada célula.
DISTANCIA = 2
VECINDAD_ROBOT[8] # Define un conjunto de células que rodean a la célula que posee al robot.
Mientras la VECINDAD de ROBOT sólo tenga valores VACÍO y OBSTÁCULO
{
    CAMBIOS = 0;
    Grid [ ALTO ] [ ANCHO ]
    For i -(0,ALTO)
    {
        For i -(0,ANCHO)
        {
            Si Grid [ i ][ j ] no es OBSTÁCULO ni ROBOT ni VACÍO
            {
                Todas las células de la vecindad de Grid[ i ][ j ] que
                tengan estado VACÍO, se cambian a estado
                DISTANCIA y se aumenta la variable CAMBIO en 1 por
                cada célula modificada
            }
            Si Grid[ i ][ j ] fue modificada y se encuentra en VECINDAD_ROBOT[8]
            {
                # Esto quiere decir que la evolución ha llegado a la vecindad del
                # robot, por lo cual no hace falta seguir expandiendo al
                autómata.
                Salir
            }
        }
    }
    Si CAMBIOS ==0 { Salir }
    # Si CAMBIOS == 0 luego de una iteración completa sobre la matriz de células significa
    # que la expansión se detuvo al ser contenida por los obstáculos en la cancha. Esto
    # termina el algoritmo porque no es posible llegar a la célula destino. Si la expansión
    # hubiese llegado antes a la célula que contiene al robot el algoritmo hubiese parado en
    # la guardia anterior a ésta.
}

```

```

    DISTANCIA = DISTANCIA +1
}

```

Una variable que detecta cambios dentro de la matriz se usa para detener el algoritmo si durante una iteración ninguna célula cambia su estado. Esto indica que la expansión se ha detenido antes de alcanzar a la célula que contiene el valor de robot, lo cual sucede cuando los obstáculos tienen bloqueado todo camino posible hacia el objetivo .

4.2.2 Escogencia de la ruta calculada

A partir del estado final del autómata (si la expansión alcanza a la vecindad de la célula que “contiene” al robot) se puede escoger una ruta si se toman en cuenta las células con menor valor , en forma descendente, desde el robot hacia la pelota como se puede observar en la figura 4.3. Usando esta técnica la ruta óptima no esta completamente definida, así que, para generalizar, se toma la ruta calculada como subóptima.

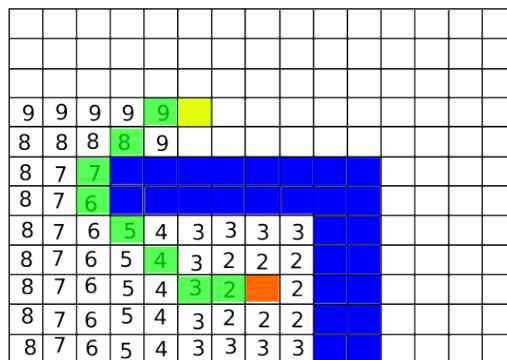


Fig. 4.3. Ruta calculada a partir del estado final del autómata celular..

Pero no se puede seguir toda la ruta calculada. Al planificar las rutas en tiempo real el algoritmo debe soportar replanificación, y una variación en el espacio celular en un tiempo T puede modificar drásticamente la ruta calculada en el tiempo T – 1. Es por

eso que sólo se escoge la célula inmediata hacia la cual el robot puede moverse, y se replanifica la ruta en la próxima iteración.

Para simplificar el modelo no se consideraron otros factores como la velocidad del robot ni la pelota, ni se implementaron estrategias de anticipación de movimientos.

La próxima célula a escoger se calcula de la siguiente manera:

1.- Se toman de la vecindad de R (célula que “contiene” al robot) aquellas células cuyos estados (de sus autómatas finitos) sean los mínimos, excluyendo las células que contengan “obstáculos”.

2.- A cada una de estas células se le asocia, según su posición respecto de R, un ángulo de giro, tomando como referencia un plano cartesiano cuyo punto de corte entre ejes es el centro de la célula donde se encuentra el robot. En la figura 4.4 se puede observar la disposición de las 8 células alrededor de la célula que contiene al robot, y los valores en grados asignados a cada una de ellas.

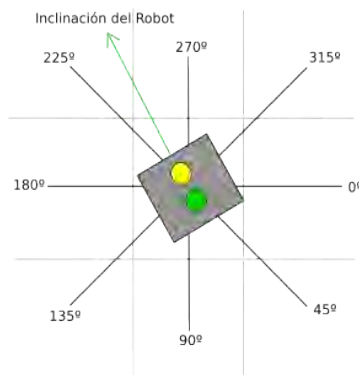


Fig. 4.4. Esquema de escogencia del ángulo a girar, cada casilla de la vecindad tiene un ángulo de giro asociado en concordancia con un sistema cartesiano centrado en el robot.

3.- Tomando en cuenta el ángulo de orientación del robot (calculado por el módulo EYE y proporcionado como parte del paquete de datos recibido) se calcula la

dirección y el ángulo de giro que el robot debe realizar para alcanzarlas.

4.- Se escoge entonces aquella célula hacia la cual se deba girar menos, sin importar la dirección.

4.2.3 Ángulo de aproximación

En trabajos anteriores sobre esta técnica [4] se implementaron heurísticas que favorecían el movimiento del robot en línea recta, eligiendo siempre horizontales y verticales, por lo cual el robot era incapaz de moverse en una línea diagonal, forzándolo a realizar trayectorias escalonadas. En este trabajo se implementó un “ángulo de aproximación”, esto es: si la casilla objetivo estaba dentro de un ángulo definido al frente del robot, seguía moviéndose hacia adelante hasta que la desviación de la ruta fuese mayor. En la figura 4.5 se puede observar un esquema que representa los dos casos posibles y la dirección que tomará el robot en cada uno de ellos. Con esta técnica se evita hacer giros muy pequeños e innecesarios, que aumentan el tiempo de trasladarse hacia el objetivo.

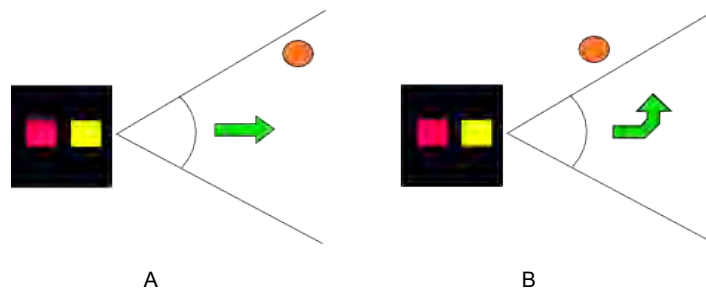


Fig. 4.5. Heurística de navegación: Si el objetivo se encuentra dentro del ángulo de aproximación del robot, se sigue la trayectoria hacia adelante (A). En cambio si el objetivo está fuera de ese ángulo (B), se le indica al robot que debe girar.

4.3 Envío de las instrucciones al robot

Un problema importante en esta etapa es la transmisión de acciones al robot en

la cancha. Se encontró que un “cordón umbilical” restringiría los movimientos y añadiría tonos de color que podían ser detectados erróneamente por el módulo EYE, y se optó por usar un sistema de comunicación inalámbrico. Un circuito RF usado en trabajos anteriores fue mejorado y utilizado en el presente trabajo, permitiendo comunicar acciones por radio al robot prototipo.

Se utilizó una aplicación de código libre basada en C++ que permite enviar información por el puerto serial de la computadora, al cual se conecta el emisor de radio. La comunicación por puerto serial se hizo usando palabras de 8 bits. Se usaron los siguientes comandos:

Comando	Acción
1	Moverse hacia adelante
Entre 100 y 130	Girar a la derecha, un número de pulsos igual al código - 100
Entre 140 y 170	Girar a la izquierda, un número de pulsos igual al código - 140
2	Detenerse

Tabla 4.2. Significado de los comandos enviados al robot

El número de pulsos (medidos por los sensores infrarrojos unidos a las ruedas) que el robot debe registrar para hacer media vuelta (girar 180 grados) es 30. El ángulo a girar se convierte a pulsos en el módulo MIND y se envía como parte del comando. El sistema de comandos fue implementado a manera experimental, y soporta varios robots simultáneamente, de la siguiente forma:

1.- Se envía un código que va de 200 a 224, que es la identificación del robot al cual vamos a transmitir. Esto hace que el robot pase a modo de acciones, y sólo sea él quien ejecute el comando siguiente. Así se pueden tener varios agentes robóticos

identificados cada uno con un código diferente.

2.- Se envía en una sola palabra la acción a realizar. Esto se debe a que la comunicación puede perderse si se implementa un protocolo que envíe la información por partes, como por ejemplo ID + Código acción + Parámetros de acción. Limitando el número de envíos se minimiza la posibilidad de errores

El protocolo usado es unidireccional, puesto que no se cuenta con un emisor desde el robot que confirme que los comandos enviados se reciban correctamente.

Capítulo 5

MÓDULO DE CONTROL

El paradigma híbrido utilizado para diseñar el comportamiento del robot se caracteriza por reunir algunos aspectos de los sistemas reactivos y jerárquicos. De esta forma el robot no se limita a planificar su entorno para luego actuar, como lo haría siguiendo el paradigma jerárquico, o reaccionaría a estímulos en sus sensores, como lo haría de manera reactiva. El paradigma híbrido permite planificar acciones para luego ejecutarlas por medio de acciones predefinidas desencadenadas por estímulos específicos.

El proceso de PLANIFICAR para luego (SENTIR – ACTUAR) propuesto por el paradigma híbrido se implementa en el sistema como sigue:

1. Primero se planifica la ruta posible en el módulo MIND desde el punto de partida hacia el objetivo, usando la imagen tomada de la cámara con el módulo EYE.

2. Acto seguido se envía al robot un estímulo en forma de código numérico, por medio del emisor de radio, que desencadenará una acción previamente definida (girar, detenerse o seguir adelante). La arquitectura libera al robot de los cálculos de la planificación, y este se limita a “sentir” por medio de su receptor de radio.

El módulo a bordo del robot, o ACT, es precisamente el encargado de recibir los códigos de acción del módulo MIND y llevarlos a acciones en la cancha. Sigue una arquitectura simple de Esclavo – Maestro, en la que el programa espera acciones por su receptor de radio, las traduce y ejecuta la acción asociada al estímulo. Es el módulo más pequeño de la arquitectura.

5.1 Robot Prototipo

El robot prototipo fue construido siguiendo el diseño del equipo campeón en la competencia RoboCup 1998[25]. Se trata de un robot holonómico construido con el kit Lego MindStorm, que usa como procesador un HandyBoard[27] en cuyo puerto serial se conecta un receptor de radio. Una especificación más detallada de sus componentes se puede ver en el siguiente capítulo.

Se instalaron sensores de pulso en las ruedas del robot, los cuales detectan el movimiento de un pequeño engranaje de 8 dientes unido a los ejes principales. Se detectan 16 pulsos (o ticks) por revolución, equivalentes a 1 tick por centímetro recorrido.

5.2 Navegación del robot

Con la información obtenida de los sensores de pulso, se pudieron desarrollar

rutinas que controlaban el movimiento en línea recta del robot, y se logró hacer rectas de más de 3 metros, a una velocidad de 30 cm/seg, con apenas unos 5 grados de desviación de la ruta original.

De esta forma se pueden crear trayectorias complicadas usando tan sólo las tres primitivas de movimiento programadas (avanzar, girar, detenerse), desencadenadas por los estímulos recibidos del planificador.

Capítulo 6

HARDWARE Y SOFTWARE UTILIZADOS

Para la implementación de los tres módulos principales el sistema, se contó con diferentes plataformas de hardware, así como ambientes de desarrollo capaces de soportar el proceso de creación del proyecto.

6.1 Robot

El prototipo de robot cazador fue diseñado y fabricado usando piezas del kit Lego MindStorm 2.0 [29] y Technnic [30]. Al no contar con el procesador RCX [29] incluido en los kits de Lego se optó por usar la tarjeta HandyBoard [27] como cerebro del robot. El robot cuenta con dos sensores de pulso para el control de giro de las ruedas laterales y un receptor de radio que interpreta las señales enviadas desde el módulo MIND.

Las acciones se reciben por el puerto serial, en el cual está conectado el

receptor de radio. Consta de una línea serial de 8 bits de datos, recibiendo a 9600 baudios por segundo. El contar con sólo un receptor de radio limita el sentido de la comunicación, por lo cual no se puede extraer información del robot mas allá de la que la cámara puede observar. Las acciones se envían siguiendo un protocolo diseñado para tal efecto.

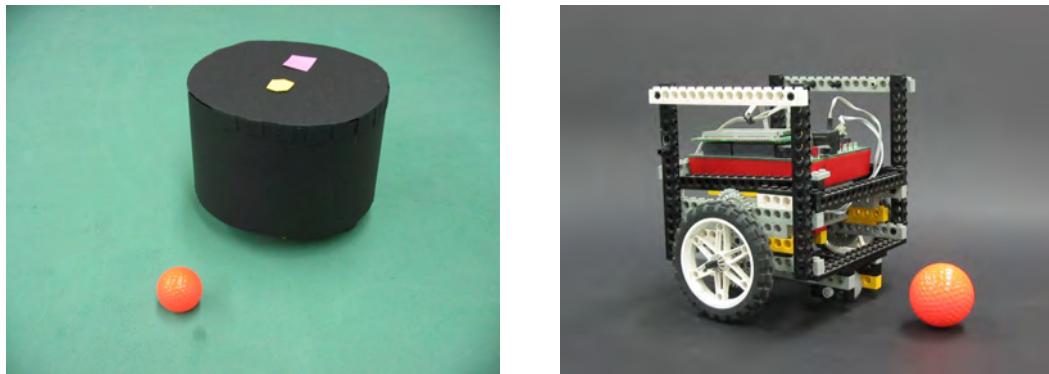


Fig. 6.1. (Izquierda) El robot prototipo en la cancha. Obsérvense los parches de color de la parte superior, usados para el posicionamiento global. (Derecha) Aspecto interno del robot.

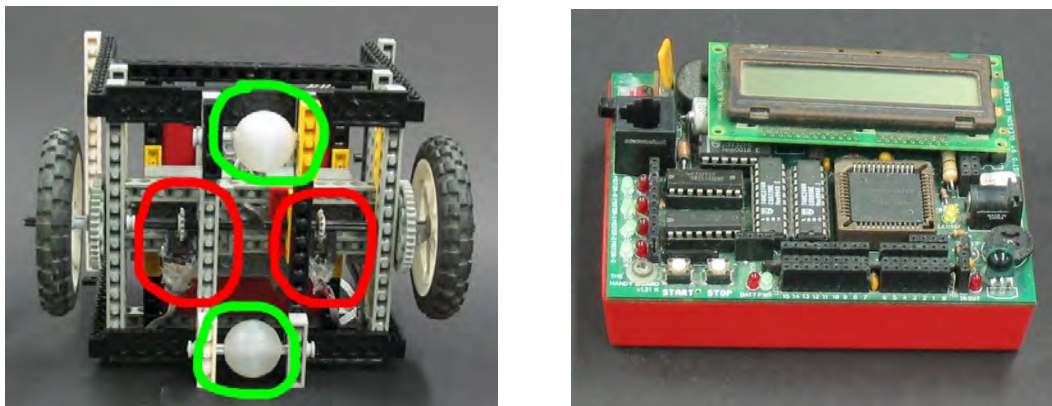


Fig. 6.2. Detalles del robot: (Izquierda) En rojo, sensores de pulso acoplados a los ejes de las ruedas, en verde, rodamientos auxiliares. (Derecha) Detalle de la HandyBoard.

Ficha Técnica de la HandyBoard	
Componente	Información Técnica
Procesador	Motorola MC68HC11 a 2 MHz.
RAM	32K.
Chips auxiliares	2 Chips L293D, que controlan hasta 4 motores DC.
Pantalla	Pantalla LDC de 16 x 2 caracteres.
Puertos	9 puertos digitales y 7 analógicos para sensores. 1 Puerto Serial estándar para comunicación.
Dimensiones	4.25 x 3.15 pulgadas.
Baterías	Baterías internas NiCad recargables, de 9.6v.

Tabla 6.1. Ficha técnica de la HandyBoard.

Ficha Técnica del Robot Prototipo	
Componente	Información Técnica
Tamaño	Largo 15 cm, ancho 21 cm, alto 19 cm.
Peso	Aproximadamente 200 gramos sin el procesador.
Velocidad	Max 70 cm/seg
Motores	2 motores DC LEGO, conectados a una transmisión independiente de relación 1/50.
Sensores	2 sensores de pulso, uno en cada rueda, calibrados a 30 pulsos por revolución.
Comunicación	Módulo RF Linx, modelo RXM418LCR, conectado a puerto serial estándar.
Autonomía	45 minutos con máxima carga

Tabla 6.2. Ficha Técnica del robot prototipo.

6.2 Cámara de Video

Para la arquitectura se usaron dos computadoras separadas. La encargada de la captura y procesamiento de imágenes (Módulo EYE) es una Pentium III ejecutándose a 550 MHz, con 256 Mb. de RAM. Esta posee una tarjeta capturadora de video Martos I, la cual se conecta a la cámara por medio de una entrada RCA. El controlador de video que se utiliza es Martos Meteor FrameGrabber. Esta estación

corre bajo Linux Gentoo, Kernell 2.4 [31].

Para la captura de las imágenes se utilizó una cámara color 3CCD, marca Mintron, equipada con un lente varifocal que permite ver la cancha en su totalidad. La conexión a la tarjeta capturadora se hizo por cable coaxial, siendo necesaria la conversión a un terminal RCA. La cámara está colocada en un soporte a 220 centímetros arriba de la cancha y genera 30 frames/seg.

6.3 Computadores utilizados

La encargada de ejecutar el módulo planificador es un AMD Athlon XP+ 2400, corriendo a 2.0 Ghz, con 512 Mb de RAM, bajo una plataforma Linux Gentoo Kernell 2.6. Esta estación se utilizó además para el desarrollo de ambos módulos. Ambas computadoras se comunican entre si por medio de paquetes UDP en una LAN a 100 Mbps.

El paquete que comunica al módulo EYE con el módulo MIND tiene la siguiente estructura:

- 1.- Una matriz de enteros de 50 filas por 27 columnas, contiene la representación de la imagen. Esta matriz sera luego el espacio celular usado por el autómata

- 2.- Dos enteros que contienen la posición del robot dentro de la matriz

- 3.- Un entero que se utiliza para el chequeo de errores. Contabiliza el numero minimo de elementos que fueron captados por la camara (el robot y la pelota). Si no fueron captados estos dos objetos el paquete se descarta.

- 4.- Un entero que contiene la orientación del robot en la cancha.

6.4 Ambientes de desarrollo

Los módulos MIND y EYE fueron implementados casi en su totalidad en ANSI C, a excepción de las librerías de comunicación serial, que están basadas en C++. Se escogió ANSI C por ser un lenguaje eficiente a la hora de ejecutar algoritmos que requieren mucho poder de cálculo comparado con otros lenguajes de programación como JAVA y C++, además de poseer librerías para el manejo de sockets y la comunicación por red entre módulos. Las aplicaciones usadas para apoyar a los módulos (xcapttest) están implementadas en ANSI C y esto fue otro factor determinante a la hora de escogerlo como lenguaje de programación y no otros más eficientes como ASSEMBLER [32]. La implementación del código en este último también requería que se estudiara a fondo un lenguaje específico de una arquitectura y esto aumentaba el tiempo en el cual se debía desarrollar el proyecto, desviándolo de sus objetivos primarios.

El módulo ACT cargado dentro de la HandyBoard está encargado de la interpretación de las señales de radio recibidas y la puesta en marcha de las acciones básicas como moverse hacia delante, detenerse y girar un número determinado de grados. Dicho módulo fue implementado usando InteractiveC 6.0 [28], un ambiente de programación y desarrollo de robots basado en lenguaje C, que permitió crear los procedimientos necesarios para la comunicación por el puerto serial y las primitivas de navegación.

Para desarrollar la herramienta auxiliar de calibración de colores se usó ANSI C con una interfaz desarrollada en GTK+ 1.2. Este paquete gráfico resulta adecuado

para el sistema, puesto que está basado en ANSI C, además de proveer todas la herramientas que se necesitaron a la hora de desarrollar la interfaz de la aplicación de calibración. Las librerías de X también fueron usadas a la hora de presentar el video tanto para el programa de calibración como para el módulo EYE.

Capítulo 7

EXPERIMENTOS Y RESULTADOS

El objetivo de los experimentos realizados es planificar en tiempo real una ruta global para un robot móvil, utilizando para ello una cámara de video colocada encima del ambiente en el cual se desarrolla la acción. El punto de llegada es una pelota de golf de color naranja, colocada en un sitio aleatorio de una cancha de color verde pizarra. La pelota puede estar moviéndose o no, al igual que los obstáculos colocados en la cancha, marcados con un color azul para el reconocimiento por visión. El robot es de color negro, con dos parches de color diferente que indican el frente y la parte trasera del mismo.

La metodología de los experimentos es la siguiente: se realizan pruebas en un ambiente sin obstáculos, con un objetivo estático alineado con la dirección del robot. Se va aumentando la dificultad de los mismos hasta incluir obstáculos estáticos y

móviles y un objetivo móvil, observando la trayectoria trazada por el robot hasta la pelota de golf. Se realizaron un total de 5 tipos de pruebas. Se registra la ruta trazada por el robot y el tiempo que tarda en conseguir su objetivo.

El ambiente en el que se desenvuelve el robot es una mesa rectangular de color verde pizarra, de dimensiones 153 cm x 137 cm, rodeada de paredes de 8 cms de alto, de color negro mate, como se puede observar en la figura 7.1.

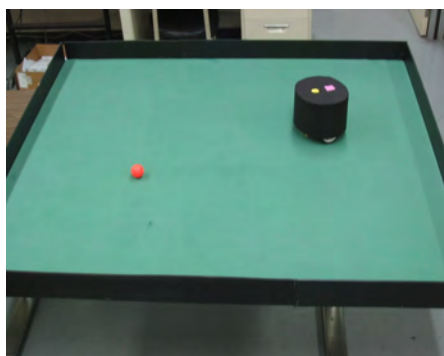


Fig. 7.1. Ambiente del experimento

Previo a la aplicación del planificador, se calibran los valores RGB de los objetos en cancha, uno por uno. Estos valores serán introducidos en el módulo EYE para su reconocimiento. En total se reconocen 4 colores: naranja (pelota), verde (parte trasera del robot), amarillo (parte delantera del robot) y azul (obstáculos). La tabla siguiente resume los intervalos promedios de RGB para los 4 colores buscados.

Objeto	R mín.	R máx.	G mín.	G máx.	B mín	B máx
Pelota (naranja)	175	255	74	117	74	117
Frente Robot (amarillo)	170	255	186	255	0	255
Detrás del robot (rosa)	197	255	0	159	159	255
Obstáculo(azul)	79	170	143	154	159	255

Tabla 7.1. Valores promedio de los intervalos de reconocimiento de los colores en cancha.

7.1 Experimentos

Es importante resaltar que para cada repetición del mismo experimento el robot parte desde un punto definido, apuntando siempre a la misma dirección según sea el caso. El punto inicial y la orientación varían dependiendo del experimento.

7.1.1 Planificación sin obstáculos

El primer experimento consistió en colocar al robot y su objetivo en la cancha a una distancia de 50 cms. No se colocaron obstáculos. Se ejecutaron 10 intentos y se midió el tiempo que el robot tardaba en llegar a su objetivo.

7.1.2 Planificación con obstáculos

Los obstáculos fueron rectángulos de aproximadamente 5 cms por lado de cartulina azul. Fueron colocados formando barreras entre el robot y su objetivo, de 4 formas distintas, como puede observarse en la tabla 7.3. Se ejecutaron 10 corridas por configuración de obstáculos.

7.1.3 Planificación con objetivo móvil sin obstáculos

Se fijó la pelota a una vara de color negro para poder manipularla fácilmente sin generar lecturas erróneas al clasificador de colores, y se movió frente al robot describiendo trayectorias rectas y cambiando de dirección aleatoriamente. Se hicieron 4 corridas de 5 minutos cada una. No se ejecutaron 10 corridas seguidas debido a que las baterías del robot no permitían una autonomía de mas allá de los 24 minutos usando el receptor de radio.

7.1.4 Planificación con objetivo estático y obstáculos móviles

Se fijó un conjunto de 3 x 1 parches de color azul a una vara de color negro, de la misma manera que se hizo para mover al objetivo en el experimento anterior, y se interpuso entre el robot y su objetivo, cambiando la posición de la barrera móvil según era necesario. Se hicieron 4 pruebas.

7.1.5 Planificación con obstáculos y objetivo móvil.

Se colocó un obstáculo móvil en el centro de la cancha y se movió el objetivo de la misma forma que en experimentos anteriores. Se ejecutaron 4 corridas de 5 minutos cada una.

7.2 Resultados

7.2.1 Planificación sin obstáculos

Los 10 intentos realizados fueron satisfactorios. El robot pudo fácilmente planificar su ruta en línea recta hasta el objetivo. Se registró el tiempo que tarda el robot en alcanzar su objetivo y el número de colisiones² que presentó y se resumieron en la tabla 7.2. Se observó que las trayectorias trazadas por el robot tienden a ser rectas, así se colocara el objetivo diagonalmente con respecto al robot. Esto es resultado del ángulo de aproximación implementado, con el cual se minimizan los desvíos en la ruta hacia el objetivo.

	T. Mínimo	T. Máximo	Tiempo Promedio	% de colisiones
Luego de 10 corridas	3,71 sg	8,75 sg	6,33 sg	0

Tabla 7.2. Tiempos y porcentaje de colisiones en el experimento 1.

² Choques del robot contra las barreras azules y las paredes de la cancha antes de tocar la pelota.

7.2.2 Planificación con obstáculos

Se probaron 4 diferentes configuraciones de obstáculos, entre ellas una que hacía que la pelota fuera totalmente inaccesible por el robot. El robot pudo sortear los obstáculos principalmente por su capacidad de moverse en líneas diagonales, funcionalidad implementada gracias al ángulo de aproximación, técnica explicada con anterioridad. Es importante destacar que las trayectorias dentro de una misma configuración de obstáculos son semejantes, con lo cual se descarta el hecho de que el robot llegara a su objetivo por azar. Se registró el tiempo que tardó el robot para alcanzar la pelota, la trayectoria trazada y el número de corridas que presentaron colisiones en cada una de las 10 corridas dentro de cada configuración de obstáculos. Vale destacar que en el caso de la configuración en la que la pelota es totalmente inalcanzable el robot mantuvo su posición sin moverse. Estos resultados fueron resumidos en la tabla 7.3.

7.2.3 Planificación con objetivo móvil sin obstáculos

El robot responde a los cambios bruscos de dirección de su objetivo con cierto retraso, debido a que su arquitectura no le permite ser un poco más rápido. Aún así se mantiene a la caza de su objetivo sin perder el rumbo en la mayoría de las oportunidades. Se hicieron 4 corridas en una cancha desprovista de obstáculos, con una duración de 5 minutos cada una. Se encontró que al robot se le hace difícil seguir cambios demasiado bruscos en tiempos muy cortos. Las baterías incluídas en el robot jugaron un papel determinante en la realización de estos experimentos, ya que

al encender el radio receptor se disminuía la autonomía del prototipo drásticamente y no era posible realizar varias pruebas de larga duración para observar en detalle el comportamiento del planificador. Esta es la única prueba en la cual no se obtienen resultados medibles (tiempo, colisiones, etc). Se implementó para asegurarse que el robot reaccionareplanificando su ruta según los cambios de dirección su objetivo.



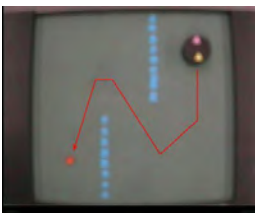

Configuración y trayectoria	T. Mínimo	T. Máximo	T. Promedio	% de colisiones
	17,17 sg	50,93 sg	26,535 sg	30,00%
	6,27 sg	14,09 sg	9,433 sg	0,00 %
	39,13 sg	66.5 sg	47,282 sg	30,00%
	N/A	N/A	N/A	0,00%

Tabla 7.3. Trayectorias trazadas, tiempos de recorrido y porcentaje de colisiones en cada una de las configuraciones de obstáculos.

7.2.4 Planificación con objetivo estático y obstáculos móviles

Se realizaron 4 corridas de 5 minutos cada una, colocando la barrera móvil en la trayectoria del robot y registrando las colisiones. La pelota se colocó en una de las mitades de la cancha y el robot en la otra. Se encontró que el planificador es capaz de reaccionar en un ambiente dinámico, pero aún así el robot colisionó contra la barrera. Esto se debe a dos causas principales:

1.- En primera instancia el robot no es lo suficientemente rápido procesando las señales, por lo cual se fue restringiendo el número número de acciones generadas por MIND hasta llegar 2 por segundo, valor con el cual el robot lograba moverse de forma fluida.

2.- En algunos casos ocurrió que el planificador no pudiera enviar la acción de detenerse o girar antes de que el robot colisionara. Esto sucedía en cambios muy bruscos, como por ejemplo colocar la barrera a menos de 1 centímetro del robot, con un movimiento muy rápido. Otro aspecto importante es que el robot genera sombras sobre la barrera, y esto causa que en ocasiones el módulo EYE no reconozca el color azul del obstáculo al cambiar su patrón RGB, llevando a la planificación de una ruta errónea.

Por cada intento de colocar la barrera se registró la posible colisión. En total se hicieron 50 intentos por corrida y se resumieron en la tabla 7.4.

En este experimento no se registró el tiempo en el cual el robot llegaba a su objetivo, pues sólo se intentaba medir la capacidad del mismo de esquivar los obstáculos de su ambiente replanificando su ruta en tiempo real. El tiempo en llegar a

el objetivo podía hacerse tan largo como se deseara simplemente bloqueando al robot.

Ejecuciones	# colisiones	% colisiones
1	4	8.00%
2	3	6.00%
3	2	4.00%
4	8	16.00%
Totales (En 200 intentos de bloqueo)	17	8.50%

Tabla 7.4. Número de colisiones registradas para el experimento

7.2.5 Planificación con obstáculos y objetivo móvil.

Se probó que el robot es capaz de planificar una ruta en un ambiente completamente dinámico, aunque aumentaron las colisiones con la barrera. De la misma forma que en el experimento anterior se realizaron 4 corridas de 5 minutos cada una, moviendo al objetivo aleatoriamente en la cancha con la vara y se hicieron 200 intentos de bloqueo con la barrera móvil, 50 en cada corrida.

Como en el experimento anterior, el tiempo en lograr el objetivo de llegar a la pelota no fue registrado y la pelota era movida rápidamente cuando el robot se encontraba cerca de llegar a ella. Se deseaba registrar la capacidad del mismo de esquivar los obstáculos y replanificar la ruta en tiempo real. Los resultados se resumen en la tabla 7.5.

Aún así, en este experimento y en el anterior, se observó que el robot se dirigía hacia la pelota y replanificaba su ruta en función de la posición de su objetivo, por lo cual se descarta la posibilidad de que se moviera de forma aleatoria dentro de la cancha.

Ejecuciones	# colisiones	% colisiones
1	14	28.00%
2	13	26.00%
3	16	32.00%
4	18	36.00%
Totales (En 200 intentos de bloqueo)	61	30.50%

Tabla 7.5. Número de colisiones registradas para el experimento

Capítulo 8

CONCLUSIONES Y RECOMENDACIONES

En el presente trabajo se desarrolló un sistema que permite la planificación en tiempo real de rutas globales usando un autómata celular con la información que provee un sistema de visión global, con lo que se pudieron lograr los objetivos pautados. El sistema consta de tres módulos de software implementados ejecutándose en estaciones separadas y precisó del diseño y fabricación de un robot prototipo para pruebas.

8.1 Objetivos logrados

Se implementó un algoritmo de planificación en tiempo real de rutas globales usando un autómata celular, que permite a un robot móvil perseguir un objetivo definido en un ambiente con obstáculos estáticos y dinámicos. La arquitectura soporta la planificación de rutas de varios robots en paralelo, basta con hacer pequeñas

modificaciones en el sistema de reconocimiento de colores e incluir una nueva estación ejecutando al planificador. Además pueden experimentarse con otros algoritmos de planificación, reemplazando el módulo MIND por un módulo distinto.

La respuesta en tiempo real del sistema está restringida por la arquitectura y componentes físicos del robot, más que por ineficiencia de los algoritmos implementados. Se tuvo que disminuir la cantidad de paquetes procesados por segundo en el planificador, de 20 a 2, ya que el robot era incapaz de reaccionar lo suficientemente rápido para ejecutarlas. Planificando a 2 paquetes por segundo el robot demostró ser capaz de conseguir su camino en ambientes complicados con pocas colisiones, replanificándola en tiempo real. Aún así, las limitaciones del prototipo causaron que la capacidad de respuesta frente a obstáculos móviles a alta velocidad no sea la óptima, llevando en muchos casos a colisiones con los mismos.

Se observó que las rutas planificadas por el autómata celular tienden a ser subóptimas, es decir, aunque siempre se consigue un camino entre el objetivo y el robot (cuando el primero es alcanzable) esta ruta no necesariamente será la más corta. Esto depende de la configuración de los obstáculos y de la posición del objetivo respecto al robot.

8.2 Aportes

Se implementó el algoritmo de visión optimizando el reconocimiento del robot dentro de la cancha, acotando el espacio en el que se hace la búsqueda en función de la posición del robot en el frame anterior. Esto aumenta la respuesta del algoritmo de visión.

Se logró mejorar la trayectoria descrita por el robot obtenida en trabajos anteriores [4] al incluir el concepto de ángulo de aproximación descrito en el capítulo 4. Esto evita que las trayectorias sean demasiado rígidas y aumenta la capacidad de maniobra del robot en la cancha, permitiéndole desenvolverse mejor en laberintos más complicados. Aún así se presentaron colisiones, mas que todo por que la capacidad del prototipo no le permitía realizar mas de 3 acciones por segundo, reduciendo su capacidad de reacción.

Más allá de simplemente implementar un planificador de rutas con alguna técnica específica, se logró diseñar e implementar una arquitectura completa, robusta y modular que puede usarse como plataforma para el desarrollo de aplicaciones más complejas.

8.3 Recomendaciones

El problema principal encontrado durante este trabajo gira en torno a los sistemas de hardware disponibles, que fueron una seria limitación para el rendimiento general del sistema.

En primer lugar, el sistema de comunicación serial por radio no es el más apropiado para la aplicación, aunque aún así probó ser efectivo. Se recomienda usar una pequeña red WIFI conectada al robot, para así implementar mejores y más robustos protocolos de control a distancia del robot. Más aún, durante las pruebas finales el módulo de radio presentó fallas de recepción, que influyeron en el resultado de algunos experimentos.

La arquitectura del prototipo también resultó una limitante para el sistema, y se

recomienda usar un robot mas robusto y rápido para pruebas, por ejemplo uno del tipo Khepera [13], ya que las prestaciones del robot prototipo no eran las óptimas para el experimento. Un sistema de locomoción diferencial de tres ruedas como las usadas actualmente por la mayoría de los equipos participantes en RoboCup [25] es el óptimo si se desea implementar un equipo de jugadores de fútbol robótico.

Un nivel más de control de multiagentes y estrategias puede implementarse y colocarse justo antes del planificador de rutas, como en [11,12,16], con lo cual se puede controlar a un equipo de robots cooperando para lograr una meta común, como por ejemplo un equipo de RoboCup.

Apéndice A

Manual del Usuario

Esta sección está destinada al manejo de las aplicaciones del sistema.

A.1 Reconocimiento de Imágenes

Antes de nada es crucial calibrar y ejecutar la aplicación del reconocedor de imágenes. Se deben usar condiciones de luz controladas, siendo la luz blanca de bombillos de neón la solución óptima. El experimento debe realizarse en un sitio cerrado, lejos de fuentes de luz naturales, ya que están pueden variar a lo largo del experimento y generar lecturas erróneas por parte del sistema de reconocimiento.

El primer paso es calibrar los colores que se usarán. Para ello se cuenta con una aplicación de calibración implementada al efecto. Debe ejecutarse en el computador que posea la capturadora de video, usando:

```
./calibracion -f NombreArchivo
```

Al ejecutarlo aparecerá una sencilla interfaz con 6 barras deslizantes y 5 botones. Para empezar la calibración se debe pulsar el botón “Calibrar”.

Se presentará una pequeña ventana de video. Al mover las barras deslizantes (dos para cada canal, con valores mínimo y máximo) se acotarán los canales RGB del calibrador, marcando con un color azul los colores que entren en estos intervalos. Si se desea por ejemplo calibrar el color de la pelota, se coloca en la cancha y se trabaja con las barras deslizantes hasta que en la pantalla de video se presente un color azul

sobre la pelota. Luego se presiona el botón PELOTA de la interfaz. Así con cada color de la cancha (Partes trasera y anterior del robot, obstáculos).

Los valores quedarán registrados en el archivo que fue pasado por parámetro.

Se procede a invocar la aplicación EYE con la siguiente línea de comandos:

```
./eye -d nombreMaquina -f ArchivoColores -p Puerto -v [Modo Gráfico]
```

-d Se especifica el nombre de la máquina que ejecutará el planificador.

-f Se especifica el nombre del archivo que contiene la calibración de los colores.

-v Con esta opción se presentan en la pantalla de video los objetos reconocidos marcados con una cruz de color.

-p El número de puerto

A.2 Planificador de Rutas

Se ejecuta el planificador con la siguiente línea de comandos:

```
./mind [número de puerto]
```

El número de puerto debe coincidir con el del servidor de visión.

A.3 Módulo a bordo del robot

El robot debe estar cubierto con su “casco de juego” orientado en la posición correcta según su frente y parte trasera. Se debe verificar que tanto el receptor como el transmisor de radio estén encendidos y bien conectados a sus respectivas estaciones.

Se recomienda no colocar las manos ni objetos extraños en la cancha, ya que podrían generar lecturas erróneas al reconocedor de imágenes.

BIBLIOGRAFÍA

- [1] Arkin R. C. "Navigational Path Planning for a Vision-based Mobile Robot". *Robotica* Vol 7. Pag 49-63. (1989)
- [2] Arkin Ronald C. "AuRA: Principles and Practice in Review". Georgia Institute of Technology. Atlanta. USA. (1989)
- [3] Baltes, Jacky y otros. "Adaptative path planner for Highly Dynamic Enviroments". Centre for Image Technology and Robotics, University of Auckland, Auckland New Zealand.
- [4] Bering, C. y otros. "An Algorithm for Robot Path Planning with Cellular Automata". (2000).
- [5] Bowling Michael y Manuela Veloso "Motion Control in CMUnited-98". Carnegie Mellos University. (1998)
- [6] Bugmann, Guido y otros. "Route Finding by Neural Nets". School University of Plymouth.
- [7] BURKE, A., Ed. "Essays on Cellular Automata". University of Illinois Press, Champaign, IL. (1970)
- [8] Choset, Howie y Joel Burdick. "Sensor Based Planning, Part I: The Generalizaed Voronoi Graph". California Institute of Technology, (1995)
- [9] Codd, E.F. "Cellular Automata". ACM Monograph Series. Academic Press, New York. (1968)
- [10] Dean, "Thomas L.. Planning and Control". Morgan Kaufmann Publishers Inc. San Francisco, CA, USA. (1991)
- [11] Han, Kun y Manuela Veloso. "Reactive Visual Control for Non-Holonomic Robotics Agents". Computer Science Departmen, Carnegie Mellon University. (1998)
- [12] Hetch, Bastian y otros. "A neural Coach for teaching robots using diagrams". Freie Universität. Berlin.
- [13] K-Team, "Khepera User Manual", Préverenges (Suiza) 1999.
- [14] Koren, Y. y J. Borenstein. "Potential Field Methods and their inherent limitations for

Mobile Robots Navigation”. University of Michigan. (1991)

[15] Kraetzschmar, Gerard y otros. “Probabilistic Quadrees for Variable-Resolution Mapping of Large Environments”. University of Ulm. Alemania.

[16] Kuipers, Benjamin y Yung-Tai Byun. “A Robot Exploration and Mapping Strategy Based on a Semantic Hierarchy of Spatial Representations”. University of Texas at Austin.

[17] Logan S. “Gtk+ Programming in C”. Prentice Hall PTR, (2001).

[18] McCloy, D. y M. Harris. “Robotics, an introduction” Open University Press (Gran Bretaña) 120-124 (1967).

[19] Murphy, Robin. “Introduction to AI Robotics”. The MIT Press. (2000)

[20] Ortega, Maruja y Oscar Meza. “Grafos y Algoritmos”, Equinoccio USB (1993)

[21] Su, Charles W. “Path Planning for the Dual Use Detachable Mobile Manipulator”. Carnegie Mellon University. (1994)

En Internet

[22] Graphics Applications of Cellular Automata
<http://madeira.cc.hokudai.ac.jp/RD/takai/automa.html>
25/05/2006.

[23] Sojourner Rover Home Page
<http://mpfwww.jpl.nasa.gov/rover/sojourner.html>, 18/07/2006.

[24] Xcptest Source Code
<http://www.hta-bi.bfh.ch/Resources/Computing/SunGroup/cgi/cvsweb.cgi/~checkout~/polyphem/src/v4l-demo/xcptest.c?rev=1.1&content-type=text/plain&sortby=rev>,
14/03/2005.

[25] IEEE Portal Site
<http://www.ieee.org/portal/site>.2006.
12/06/2006

[26] RoboCup Official Site <http://www.robocup.org/>. 14/07/2006

- [27] Universality and Complexity in Cellular Automata (1984)
<http://www.stephenwolfram.com/publications/articles/ca/84-universality/>.
15/05/2006.
- [27] The HandyBoard Home page. <http://handyboard.com/>. 16/03/2006.
- [28] Interactive C Home Page. <http://www.newtonlabs.com/ic/>. 11/07/2006
- [29] Lego Mindstorms Home Page - <http://mindstorms.lego.com/>. 11/07/2006
- [30] Lego Technic Home Page -
<http://www.lego.com/eng/create/technic/productpage.aspx>. 11/07/2006
- [31] Gentoo Home Page - <http://www.gentoo.org/>. 13/07/2006
- [32] ARM ASSEMBLER - <http://www.heyrick.co.uk/assembler/>. 12/07/2006
- [33] Meteor Frame Grabber on GlobalSpec
http://frame-grabbers.globalspec.com/Industrial-Directory/matrox_frame_grabber
12/07/2006