UNIVERSIDAD DE SALAMANCA

Departamento de Informática y Automática

Facultad de Ciencias

Técnicas de inteligencia artificial en la construcción de robots móviles autónomos

José Andrés Vicente Lober

Julio 2003

D. **Vidal Moreno Rodilla**, Prof. Titular de Ingeniería de Sistemas y Automática de la Universidad de Salamanca y Dña. **Belén Curto Diego**, Pfa. Titular de Ingeniería de Sistemas y Automática de la Universidad de Salamanca

CERTIFICAN:

Que el trabajo de grado que se recoge en la presente memoria, titulada **Técnicas de inteligencia artificial en la construcción de robots móviles autónomos** y presentada por Don **José Andrés Vicente Lober** para optar al título de Grado por la Universidad de Salamanca, ha sido realizado bajo su dirección en el Area de Ingeniería de Sistemas y Automática del Departamento de Informática y Automática de la Universidad de Salamanca.

Salamanca, 4 de Julio de 2003

D. **Vidal Moreno Rodilla**
Prof. Titular de Universidad
Área de Ingeniería de Sistemas y Automática
Universidad de Salamanca

Dña. **Belén Curto Diego**
Profa. Titular de Universidad
Área de Ingeniería de Sistemas y Automática
Universidad de Salamanca

A Cresen

Agradecimientos

Quisiera agradecer a mis tutores Doña Belén Curto Diego y Don Vidal Moreno Rodilla su paciencia y colaboración. Tambíen quiero agradecer al resto de personas del Departamento que han colaborado y me han ayudado a tener tiempo suficiente para terminar el trabajo.

Tambíen quiero agradecer a mi família su inestimable ayuda y a Cresen su apoyo psicológico.

La inteligencia busca, pero quien encuentra es el corazón.

George Sand

Índice general

Índice de figuras

1

Introducción

En este capítulo se van a revisar conceptos clave relacionados con el objetivo de este trabajo, como son la robótica móvil y sus aplicaciones reales. A continuación se realizará una descripción de la Inteligencia Artificial (IA) como solución aplicada a la robótica, incluyendo el concepto de lógica borrosa como tecnología clave. Después veremos los agentes inteligentes, que se plantean como una metodología integradora entre el concepto de la Inteligencia Artificial clásica con sus diferentes tecnologías, y los distintos componentes y comportamientos de un robot autónomo.

La robótica móvil siempre se plantea en torno a unas metas, las cuales se consiguen en un ambiente determinado con el cual el robot interactúa por medio de percepciones y acciones. El grado de consecución de las metas está directamente relacionado con la autonomía de un robot. La metodología de agentes plantea de forma directa el concepto de autonomía en la medida de la consecu-

ción de metas, basándose en el tratamiento de las percepciones y acciones sobre un determinado ambiente. Esta metodología será la empleada en la realización de los robots autónomos de este trabajo.

Podemos representar la autonomía como los cuatro pilares de un edificio. Estos deben tener sus dimensiones en la proporción justa para que el tejado se sustente correctamente. Cuanto más altura tengan, el grado de consecución de objetivos es mayor y proporcional a su autonomía (ver figura 1.1).

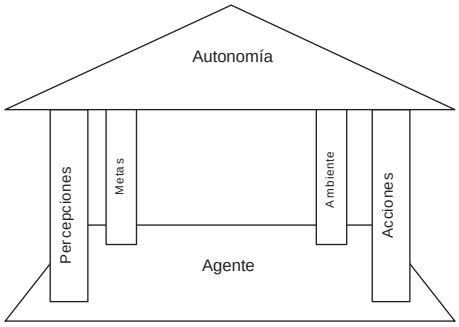


Figura 1.1: Representación de los pilares elementales de un agente

1.1. Robótica móvil

La robótica móvil comenzó en los años 70 como banco de pruebas para estudiar técnicas de Inteligencia Artificial. En los años 80 se tiene el despliegue definitivo debido al abaratamiento y mejores prestaciones de los ordenadores [Oll01].

La robótica móvil cubre muchos campos de aplicación, como son transporte de materiales, labores de limpieza, vigilancia y prospección, guiado de personas, y por desgracia aplicaciones militares. Los robots autónomos son sistemas mecánicos que pueden interactuar con un ambiente, y tienen un comportamiento individual aunque éste sea socializado. Las dificultades que se encuentran los vehículos autónomos son los entornos impredecibles y dinámicamente cambiantes, así como su limitada autonomía y sobre todo su posicionamiento en un determinado ambiente. Para evitar estos problemas se suelen utilizar robots guiados, por lo que se establecerá una comunicación con un ordenador central

que dirigirá sus movimientos e incluso coordinará varios robots colaboradores. Este guiado podrá ser por múltiples medios como railes, laser, balizas, etc. El principal problema de los robots guiados es que se restringen a caminos preestablecidos y siempre requieren algún tipo de adaptación en cada reestructuración del entorno.

Sin embargo los robots móviles autónomos no se limitan a caminos preestablecidos, ya que disponen de una capacidad de percepción. Son capaces de percibir el entorno e incluso de realizar una representación de él. También se les puede dotar de la capacidad de planificar sus movimientos evitando obstáculos, así como de detectar marcas o lugares característicos para conseguir una meta. Los robot móviles deben tener características de maniobrabilidad, controlabilidad, capacidad de tracción, estabilidad, eficiencia y consideraciones de navegación como la odometría. Pueden tener multitud de sistemas de tracción como son la configuración diferencial, en triciclo, Ackerman, síncrona, ruedas omnidireccionales, cadenas, patas, reptantes, etc, según el medio en el que se vayan a mover.

Una de sus principales dificultades es determinar su posición dentro de una zona determinada. Se pueden realizar estimaciones según una posición inicial por métodos odométricos, o bien utilizar sistemas de medidas absolutas respecto a una referencia externa, como balizas, satélites, etc.

El mecanismo de posicionamiento se basa en el sistema de percepción de que disponga. Existen multitud de sistemas sensores como sonar, laser, infrarrojos, radares, etc, aunque no es el único fin de estos, ya que la percepción del robot le ha de servir también para interaccionar con los elementos del ambiente.

Todo robot móvil se debe plantear las dos preguntas basicas: ¿dónde estoy? ¿cómo llego al objetivo?. Sin duda para resolver estas cuestiones se debe recurrir a la planificación de movimientos por medio de los actuadores del sistema y a técnicas de planificación, manteniendo quizá una representación del ambiente por el que se mueve.

1.2. Configuraciones cinemáticas en robótica móvil

Existen múltiples sistemas para proporcionar movilidad a un robot, siendo las características de maniobrabilidad y controlabilidad las más importantes. Sin embargo, existen otros aspectos básicos como la eficiencia o estabilidad que no deben ser olvidados. En robots de pequeño tamaño, como el caso que nos ocupa, se suelen utilizar sistemas basados en ruedas. Se van a revisar a continuación los distintos sistemas rodados de locomoción en robótica.

Configuración diferencial Este sistema se caracteriza por tener dos motores con reductora acoplados a sendas ruedas. En la figura 1.2 se puede observar un ejemplo, así como sus parámetros característicos.

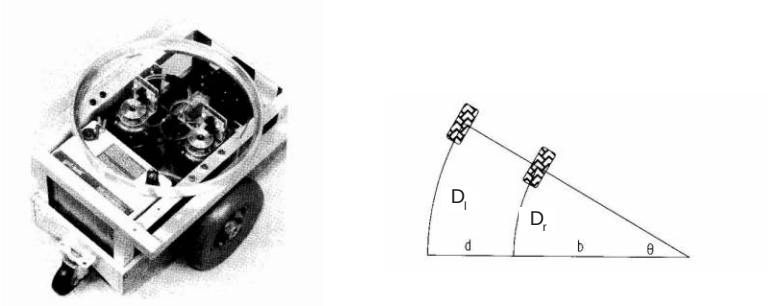


Figura 1.2: Principales parámetros de la configuración diferencial

Si consideramos que D_l y D_r son las distancias recorridas por la rueda izquierda y derecha y d es la distancia entre los dos puntos de contacto de las ruedas en el suelo, entonces el desplazamiento lineal D del punto central del robot se calcula mediante la siguiente expresion:

$$D = \frac{D_l + D_r}{2}$$

El cambio θ de orientación del robot vendrá dado por:

$$\theta = \frac{D_l - D_r}{d}$$

Esta arquitectura tiene las ventajas de ser sencilla de construir y fácil de controlar. Como principal inconveniente tiene la dificultad para trazar una línea

recta. Cualquier pequeña desigualdad en las ruedas o en el terreno hace que pierda la línea que estaba trazando.

Un diseño alternativo al diferencial es el basado en cadenas tipo tanque. Estos sistemas se utilizan cuando se necesita mucha capacidad de tracción. Mantiene la ventaja de la sencillez de manejo, pero tiene los inconvenientes de tener muy mala odometría y se pierde mucha energía en los giros debido a la fricción.

Configuración síncrona Se trata de un montaje en el que todas las ruedas son direccionales y tienen tracción, como se puede ver en la figura 1.3. Este sis-

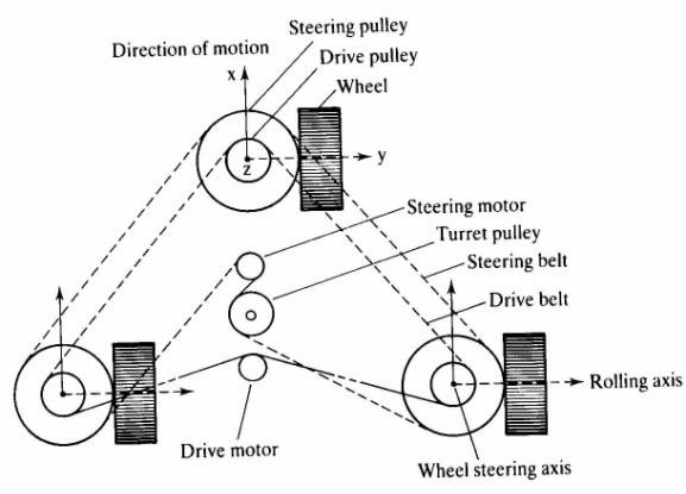


Figura 1.3: Sistema *Sinchro drive*

tema tiene la ventaja de tener un control sencillo al poder separar la tracción de la rotación. Además el seguimiento de una línea recta lo asegura la propia configuración mecánica. Sin embargo su principal inconveniente es la complejidad de diseño e implementación.

Ruedas omnidireccionales Este sistema permite realizar movimientos complejos debido a la disposición de las ruedas (figura 1.4). Sin embargo mecánicamente tiene dificultades para seguir líneas rectas y su implementación es muy complicada.

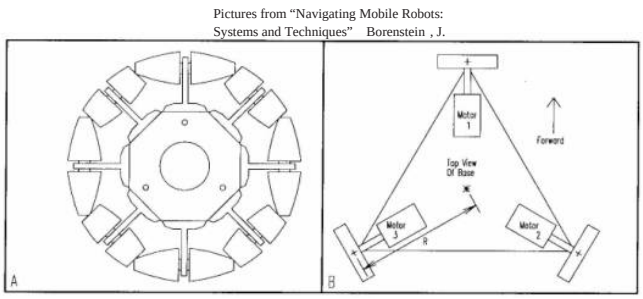


Figura 1.4: Sistema de tracción omnidireccional

Triciclo Este sistema se basa en tres ruedas, siendo una de ellas la directriz (ver figura 1.5). La principal ventaja es que no hay deslizamientos de las ruedas como ocurría en la configuración de ruedas omnidireccionales. La desventaja es que tiene una cinemática complicada. Si se tiene la tracción en las ruedas traseras, plantea problemas al girar en curvas, ya que la rueda interior gira a la misma velocidad que la exterior.

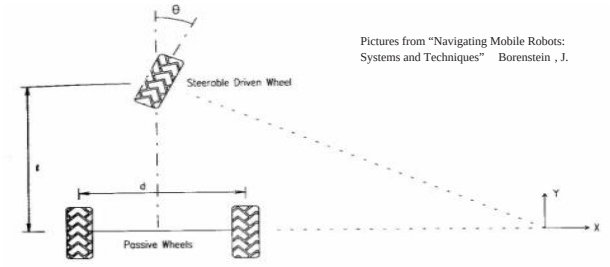


Figura 1.5: Configuración de un triciclo

Sistema Ackerman Es el sistema que normalmente utilizan los automóviles, se basa en cuatro ruedas, siendo dos de ellas directrices (ver figura 1.6). Su ventaja es que es fácil de implementar, pero tiene una cinemática muy complicada.

Sistema articulado En este sistema los dos ejes tienen capacidad directiva y de tracción. Es simple de implementar, pero es muy complicado de manejar (ver figura 1.7).

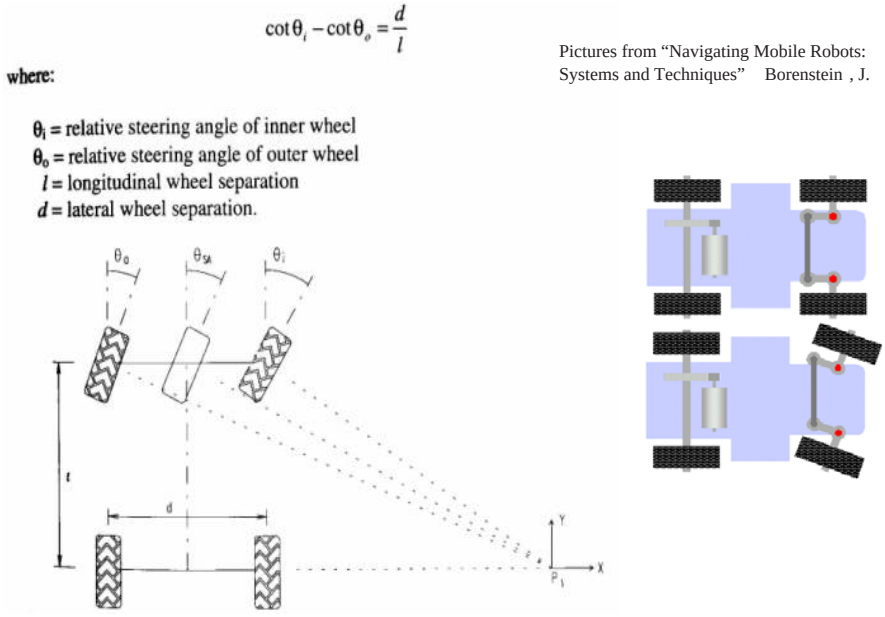


Figura 1.6: Configuración Ackerman

Figura 1.7: Sistema con dirección articulada

Framewalker Este sistema permite realizar movimientos de traslación y rotación por separado debido al deslizamiento horizontal del cuerpo del robot sobre las patas [BW99] [WT96]. El movimiento sobre una línea recta está asegurado por la mecánica. Las desventajas son la complejidad de diseño e implementación, y que la rotación y traslación no son simultáneas.(ver figura 1.8).

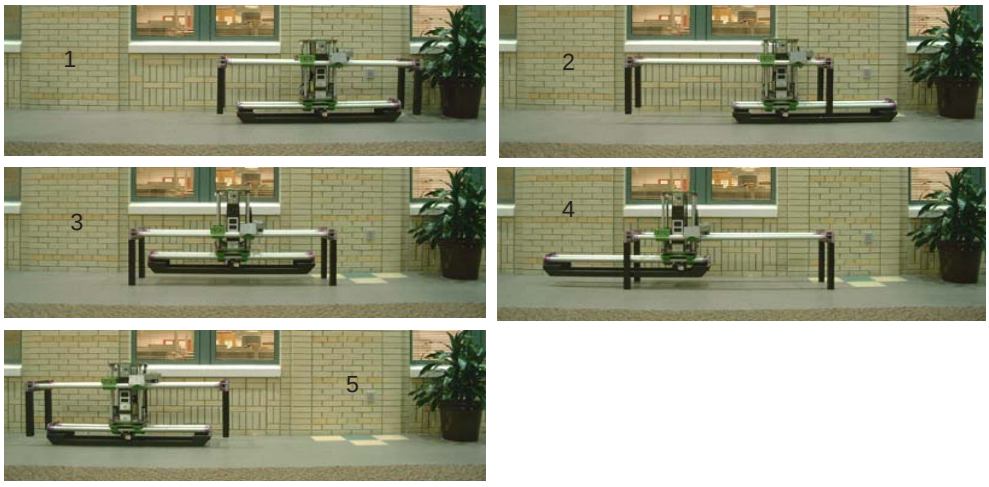


Figura 1.8: Imágenes de Framewalker en movimiento

Robot serpiente Estos robot se caracterizan porque tienen el aspecto y el movimiento en zig-zag característico de una serpiente [Yim01] [BRC95] [Nil97]. Tienen la ventaja de tener muchas aplicaciones con un sistema de tracción muy redundante. Sin embargo su control y planificación es muy complejo (figura 1.9).

1.3. Aplicaciones de la robótica móvil

Existe multitud de documentación referente a robots móviles. Podemos realizar una revisión de acuerdo a los fines para los que han sido diseñados. Sin embargo es posible afirmar que los sistemas robóticos aplicados a algún campo se basan en una parte que proporciona la movilidad, junto con otra parte que proporciona la aplicación específica. La mayor o menor integración de las

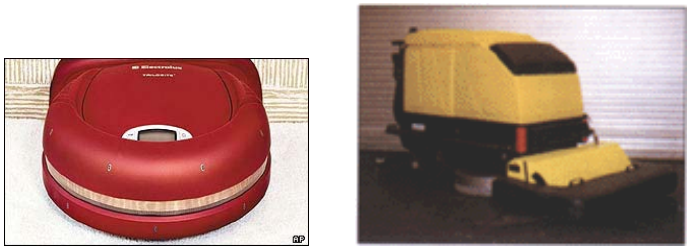
Figura 1.9: Robot tipo serpiente

dos partes proporciona la posibilidad de reutilización o de especialización de los robots.

Transporte Existen multitud de soluciones comerciales, funcionando ya desde hace mucho tiempo. Su sistema de guiado varía mucho, pero uno de los más utilizados es el de seguimiento de caminos fijos realizados con marcas, ya sean ópticas, inductivas, de luz, etc. En este caso se muestra en la figura 1.10 una solución de Flexitrack [FLE]. Se trata de carretillas de carga que realizan el seguimiento de una banda en el suelo. Se le pueden poner marcas en el camino para indicar puntos de parada y realizar carga o descarga. Dispone además de un radar para evitar colisiones. Existen varios modelos con distintas capacidades y tipos de carga.

Figura 1.10: Soluciones de transporte de Flexitrack

Robot de limpieza Los robots de limpieza están empezando a popularizarse en usos particulares. Existen soluciones que permiten aspirar una habitación de una forma bastante correcta. Tenemos un ejemplo claro con la aspiradora de Electrolux (figura 1.11(a)). Esta aspiradora tiene un sistema de percepción por ultrasonidos, de forma que le permite evitar obstáculos y elegir la ruta más adecuada. Otra solución bastante conocida (figura 1.11(b)), es la adoptada



(a) (b)

Figura 1.11: (a) Aspiradora autónoma de Electrolux (b) Robot limpiador del metro de París

por el metro de París. Se trata de un sistema autónomo para la limpieza de los andenes del metro. Su sistema de guiado se realiza por balizas magnéticas colocadas en el suelo.

Vigilancia y prospección Este tipo de robots se utilizan en los casos que resultan peligrosos para las personas, como zonas radioactivas, lugares con alta temperatura o alto riesgo, manipulaciones de materiales delicadas, lugares con alta presión, tuberías, minas, etc. También se utilizan robots en investigación aeroespacial, como el malogrado March Pathfinder.

(a) (b)

Figura 1.12: (a) March Pathfinder(NASA) (b) Groundhog, explorador de minas de la CMU

También se empiezan a ver robots de vigilancia como es el caso del mostrado en la figura 1.13. Tiene integrada una cámara de vídeo y un teléfono celular para realizar avisos en caso de detectar alguna intrusión.

Figura 1.13: Robot vigilante

Ayuda En este apartado se incluyen robots de guía, que se utilizan en centros comerciales, o como guías turísticos, etc. También existen robots de ayuda a discapacitados, robots lazarillos, silla de ruedas inteligente, etc.

En la figura 1.14 se observa un robot guía diseñado por el CARTIF. Este robot es capaz de realizar expresiones con sus rasgos faciales. Dispone de un sintetizador de voz y de un sistema de navegación. También dispone de una pantalla táctil para que los visitantes puedan interactuar con él. El sistema de guiado se basa en sonar y su movilidad está proporcionada por dos motores en disposición diferencial [DC02].

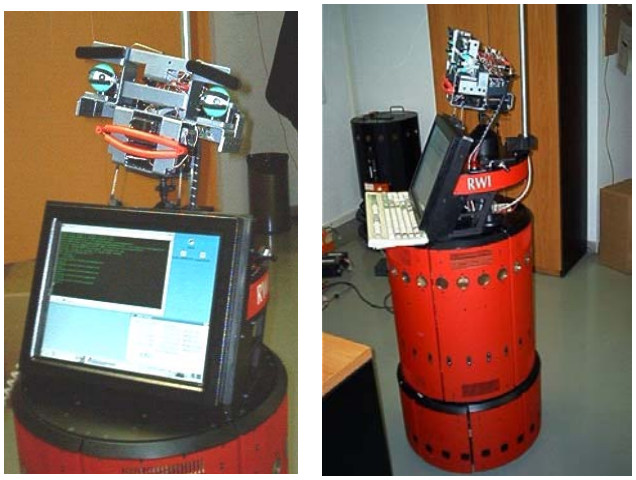
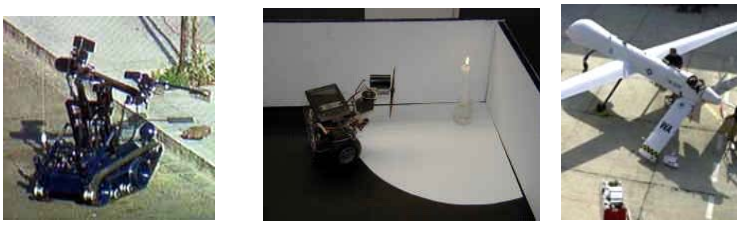


Figura 1.14: Robot guía del museo de Valladolid creado por el CARTIF

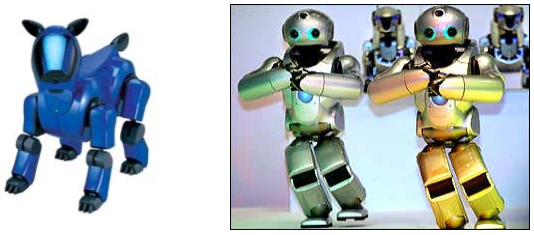
Aplicaciones especiales En este área se incluyen varios tipos de robots. Los robots de uso militar se suponen los más avanzados. Es el caso del avión espía de la figura 1.15(c). Sin embargo existen algunos falsos mitos sobre robots como los utilizados en la desactivación de explosivos, ya que la inteligencia principalmente es aportada por un operador remoto, y no incluyen prácticamente comportamientos autónomos inteligentes. También se han popularizan los llamados robot bomberos, que deben localizar y apagar un pequeño incendio. Este tipo de robots autónomos están todavía en investigación.



(a) (b) (c)

Figura 1.15: (a) Desactivador de explosivos, (b) robot bombero y (c) avión espía

Uso lúdico Se incluyen robots dedicados al entretenimiento o acompañamiento. Se han popularizado enormentente desde la inclusión en el mercado del robot Aibo [AIB] y SDR-4X [SDR] de Sony (figura 1.16(a)). Estos robots son capaces de adaptarse al entorno y aprender de él. Dispone de múltiples sensores como cámaras de vídeo, micrófonos, infrarrojos, etc. Reconoce órdenes de voz y el entorno en tiempo real, memoriza caras, etc.



(a) (b)

Figura 1.16: (a) Aibo y (b) SDR-4X de SONY

Concursos y competiciones Al igual que ocurre con las competiciones de automovilismo o motociclismo, los concursos de robótica pretenden ser el punto de evaluación y comparación de las distintas tecnologías aplicadas a la construcción de robots autónomos.

Existen multitud de competiciones de este tipo. Quizá la más representativa a nivel internacional sea RoboCup [ROB] (figura 1.17(a)). En esta competición se trata de jugar un partido de futbol con robots autónomos, los cuales son capaces de interactuar con otros robots y con el medio. Tambíen existen pruebas específicas con otros objetivos, muchas veces auspiciadas por casas comerciales.

Sin duda la competición más importante a nivel nacional es HISPABOT [HIS] (figura 1.17(b)). En este concurso existen las pruebas de rastreadores, sumo, laberinto y velocista, en las que se valoran muchos de los aspectos básicos en robótica, como la controlabilidad, interacción con el ambiente, técnicas de planificación, etc.

(a) (b)

Figura 1.17: (a) Imagen del concurso Robocup e (b) HISPABOT

1.4. Inteligencia Artificial

La Inteligencia Artificial (IA) es un campo científico en el que se trata de las que máquinas realicen tareas que podría realizar el hombre aplicando cualquier tipo de razonamiento. Para lograr este objetivo se necesitan programas informáticos.

En cierta medida cualquier programa informático puede ser considerado *inteligente*. El problema es diferenciar entre qué es lo que se considera un programa inteligente y qué no lo es.

Un programa inteligente es aquel que muestra un comportamiento similar al humano cuando se encuentra con un problema idéntico. No es necesario que el problema sea resuelto exactamente de la misma manera que lo haría una persona. Desde el punto de vista del comportamiento humano, la IA estudia la naturaleza de la inteligencia humana, como reproducirla e implementarla en un programa informático.

Sin embargo hay que diferenciar entre la inteligencia humana, y lo que se conoce como inteligencia ideal. En el primer caso se trata de la limitada inteligencia de las personas, que no coincide con la inteligencia ideal o racionalidad. Se considera que un sistema es racional si siempre hace lo correcto. El enfoque centrado en el comportamiento humano constituye una ciencia empírica, que entraña el empleo de hipótesis y confirmación mediante experimentos. El enfoque racionalista combina matemáticas e ingeniería. Ambos enfoques son las bases de sendos grupos de investigación que muchas veces critican el trabajo del otro grupo. Sin embargo ambas orientaciones han hecho valiosas aportaciones.

1.4.1. Lógica borrosa

La lógica borrosa es un tipo de solución práctica para implementar inteligencia de forma computacional. Fue introducida por la universidad de Berkeley a mediados de los años 60. Sin embargo las bases se establecieron gracias a Max Black [Bla37], a Karl Menger [Men42], y los artículos de los años 50 sobre relaciones borrosas de indistinguibilidad. Bajo el concepto de borroso, reside la idea de que las personas no pensamos con números, sino con etiquetas linguísticas. Estas etiquetas permiten que los objetos pasen de estar en una clase u otra de forma progresiva. La lógica borrosa se puede inscribir dentro del contexto de la lógica multivaluada.

En el lenguaje natural se describen objetos o situaciones en términos imprecisos como grande, joven, tímido, etc. El razonamiento basado en estos términos no puede ser exacto ya que normalmente representa impresiones subjetivas, quizá probables pero no exactas. Por ello la teoría borrosa reprsenta una forma muy adecuada de trabajar con el razonamiento humano. El razonamiento humano trabaja con unas reglas determinadas sobre nuestro sistema de representación borroso o subjetivo. De igual forma la lógica borrosa emula el razonamiento

humano aplicando reglas sobre nuestra representación borrosa del mundo. El objeto del razonamiento con reglas sobre conjuntos y etiquetas borrosas permite obtener un resultado también borroso, pero que podremos cuantificar para aplicar al control. Por lo tanto la lógica borrosa consigue representar el conocimiento humano en un área concreta y hacerlo computable.

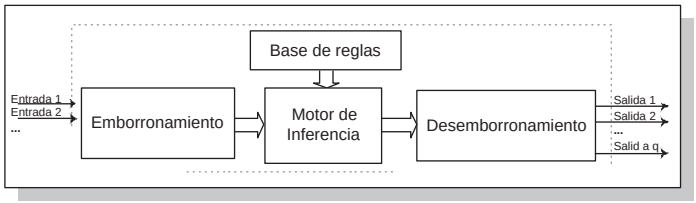


Figura 1.18: Etapas del procesamiento borroso

1.5. Agentes inteligentes

Los Agentes Inteligentes tienen una filosofía integradora de las distintas tecnologías utilizadas en Inteligencia Artificial. Se trata de una especificación formal que auna los distintos elementos hardware y software para el desarrollo de agentes inteligentes [CM02] [RN96].

Un Agente Inteligente es todo sistema que percibe su ambiente mediante sensores y que responde o actua de forma inteligente para conseguir un determinado objetivo. Los agentes humanos tienen ojos, oidos y otros órganos que les sirven de sensores, así como manos, piernas, boca y otras partes de su cuerpo que le sirven de efectores.

En el caso de los agentes robóticos, los sensores son cámaras, sensores de infrarrojos, etc. Los efectores son reemplazados mediante motores. En el caso de un agente software, sus percepciones y acciones vienen a ser las cadenas de bits codificados. Los agentes inteligentes se basan en la capacidad de percibir el ambiente que les rodea y tomar una decisión basándose no solamente en su sistema sensorial, sino también en su estado interno. De esta forma el agente será capaz de conseguir una meta para el que ha sido diseñado. Un agente inteligente es capaz de tener una memoria que representa el ambiente con el que interactua.

Esta memoria podría actualizarse para generar o ampliar su conocimiento del ambiente.

Un agente racional es aquel que hace lo correcto. Obviamente, esto es preferible a que haga algo incorrecto, pero ¿qué significa? Como un primer intento de aproximación se afirmará que lo correcto es aquello que permite obtener las mejores prestaciones.

El término medición de prestaciones trata de definir qué éxito ha tenido un agente al realizar una tarea. Desde luego no existe una medida fija que se pueda aplicar a todos los agentes. Sin embargo se deben disponer de sistemas eficientes de evaluación de prestaciones, de forma que permitan tomar uno u otro criterio posterior en función de este resultado.

Hay que dejar claro que existe una diferencia entre racionalidad y *omnisciencia*. Un agente omnisciente es aquel que sabe el resultado real que producirán sus acciones y su conducta es congruente con ello. Sin embargo en la realidad no existe la omnisciencia. No se puede culpar a un agente por no haber tomado en cuenta algo que no podía percibir, o por no emprender una acción de la que es incapaz. Sin embargo, la tolerancia en relación con la exigencia de perfección no es algo que tenga que ver con una actitud justa en favor de los agentes. Lo importante es que si se especifica que un agente inteligente siempre debe hacer lo que realmente es lo correcto, será imposible diseñar un agente para que satisfaga esta especificación. En resumen, el caracter de racionalidad de lo que se hace en un momento dado dependerá de cuatro factores:

- La medida con la que se evalúa el grado de éxito logrado
- Todo lo que hasta ese momento haya percibido el agente. A esta historia perceptual completa se le denomina la *secuencia de percepciones*
- Conocimiento que posea el agente acerca del medio
- Acciones que el agente puede emprender

1.6. Objetivos del trabajo

A continuación se van a presentar los principales objetivos de investigación y desarrollo que se han afrontado durante la realización de este trabajo de Grado.

Como objetivo fundamental, se plantea el diseño, construcción y programación de robots móviles para acudir a las reuniones que, a nivel nacional, se llevan a cabo en la actualidad, donde se ha pretendido la participación en diferentes categorías. Estos concursos plantean diferentes desafíos en los que la autonomía de los robots es el elemento básico y se proclama vencedor aquel que proporciona un mayor nivel de prestaciones.

Bajo esta idea, se plantean los principales enfoques desde los que se procede a la construcción de un robot: un diseño electromecánico que permita una mayor calidad y robustez de actuación o la incorporación de soluciones del campo de la Inteligencia Artificial que permitan optimizar las actuaciones de dicha construcción electromecánica. En este trabajo, sin dejar de lado el primer aspecto, se opta por diferentes razones por la realización de un mayor esfuerzo en el segundo aspecto intentando mostrar como la utilización de soluciones de la IA permiten alcanzar unos buenos resultados que pueden llegar incluso a la superación de las limitaciones que pudiera producir diseños electromecánicos no optimizados. En este contexto se plantean el resto de los objetivos de este trabajo, que se enumeran a continuación:

- Utilizar la arquitectura de agente inteligente en la concepción y diseño de un robot móvil. En este sentido se pretende mostrar las posibilidades que este concepto ofrece a la hora de implementar las diferentes capacidades de percepción, razonamiento y actuación. Se planteará el diseño desde este punto de vista, utilizando las arquitecturas de agente propuestas en la bibliografía.

- Asimismo, se analizarán las configuraciones mecánicas que cumplan con las especificaciones en cada caso y permitan tener un buen grado de maniobrabilidad. Con ello se pretende facilitar su control lo que puede proporcionar una mayor fiabilidad de los procedimientos que se desarrollen en este sentido. Se realizará un estudio al mayor detalle posible de las especificaciones de cada uno de los elementos electrónicos y mecánicos que constituirán los robots.

- En el diseño de los robots se han de utilizar elementos de coste reducido. Este objetivo se justifica desde una doble vertiente: primero en las limita-

ciones económicas de un centro universitario. Segundo, el perfil docente de la institución exige poner al alcance del mayor grupo de alumnos la posibilidad de construir sus propios robots con los que experimentar soluciones del campo de la Inteligencia Artificial.

- Finalmente, dado que los elementos computacionales estarán restringidos, se plantea la utilización de técnicas que proporcionen inteligencia a los robots que no requieran de grandes recursos cumpliendo con las especificaciones planteadas en cada caso. En este sentido, se plantea la lógica borrosa como una solución adecuada justificándose por el hecho de que en otros ámbitos y en el de la robótica ha proporcionado prometedores resultados.

1.7. Organización de la memoria

Esta memoria comienza con un pequeño repaso de los aspectos clave utilizado en las distintas etapas de este trabajo. A continuación se describe la arquitectura del sistema como agente, desde el punto de vista de los objetivos que se desean conseguir. En el siguiente capítulo se incluye la descripción del hardware de los robot realizados, finalizando con la descripción de las soluciones e implementación del software. Por último se encuentran los resultados y conclusiones obtenidas.

2

Arquitectura del sistema

En este capítulo se van a exponer las soluciones deseadas e implementadas con modelos de agentes, así como el posible planteamiento de prototipos de robots para las diversas pruebas con la que nos podemos encontrar en la mayor parte de competiciones nacionales.

2.1. Agentes

En el capítulo previo, se introdujo la idea básica de un agente mediante la descripción de su conducta. Estas son las acciones que se producen después de una determinada secuencia de percepciones. Vamos a estudiar como funcionan las cosas dentro de un agente.

Uno de los cometidos de la IA es el diseño de un *programa* de agente: una función que permita implantar la idea del agente para pasar de percepciones a acciones. Se supone que este programa se ejecutará en algún tipo de dispositivo

de cómputo, al que se le denominará *arquitectura*. Desde luego, el programa elegido debe ser aquel que la arquitectura acepte y pueda ejecutar dentro de sus limitaciones de potencia. Esta arquitectura podría tambien incluir un software que ofrezca cierto grado de aislamiento entre la computadora y el programa de agente, lo que permitiría la programación a un nivel superior. En general la arquitectura pone al alcance del programa las percepciones obtenidas mediante los sensores, lo ejecuta y alimenta al efector con las acciones elegidas por el programa conforme éstas se van generando. La relación entre agentes, arquitectura y programas podría resumirse de la siguiente manera:

agente=arquitectura + programa

Antes de proceder al diseño de un programa de agente, es necesario contar con una idea bastante precisa de las posibles percepciones y acciones que intervendrán, qué metas o medidas de prestaciones se supone lleve a cabo el agente, así como el tipo de ambiente en que tal agente operará. Estos datos reunidos se denominan matriz PAMA (Percepciones, acciones, metas y ambiente) [RN96].

Puede sorprender el hecho de que haya agentes que funcionen en un ambiente totalmente artificial, como por ejemplo un programa de traducción o de búsqueda. Lo que verdaderamente importa no es la diferencia entre ambientes reales y artificiales, sino la complejidad de la relación que existe entre la conducta del agente, la secuencia de percepciones que produce el ambiente y las metas que se espera alcance el agente. Algunos ambientes considerados como reales, de hecho son muy sencillos. Por ejemplo un robot diseñado para inspeccionar diversas piezas que pasan ante él en una cinta transportadora. Este puede suponer que la iluminación será siempre la misma, que lo único que está sobre la cinta son piezas conocidas y que solo se ejecutan dos acciones: aceptar y rechazar.

En contraste, en algunos muy ricos e ilimitados ámbitos, se utilizan agentes de software, tambien conocidos como robots software o *softbots*. Por ejemplo un agente diseñado para pilotar el simulador de vuelo de un 747. El simulador constituye un ambiente muy complejo y detallado. En él el agente de software debe elegir entre una amplia gama de acciones en tiempo real.

El más famoso de los ambientes artificiales es la prueba de Turing [Tur50], caracterizado porque tanto agentes reales como artificiales se encuentran en igualdad de condiciones, pero el ambiente plantea tantos desafíos que resulta

muy difícil para un agente de software desempeñarse tan bien como un humano. En la figura 2.1 se puede observar el esqueleto de un agente. Como se puede

```
Función ESQUELETO-AGENTE(percepción) return acción.
        Static: memoria

memoria ← ACTUALIZACIÓN-MEMORIA(memoria,percepción)
acción ← ESCOGER LA MEJOR ACCIÓN(memoria)
memoria ← ACTUALIZACIÓN-MEMORIA(memoria,acción)

return acción
```

Figura 2.1: Esqueleto básico de un agente

ver, cada vez que se solicite se actualiza la memoria para que refleje la nueva percepción, se escoge la mejor acción y también se refleja en la memoria la acción seleccionada. Obviamente la memoria es persistente de una solicitud a otra.

2.1.1. Tipos de agentes

Se va a dar una descripción básica de los distintos tipos de agentes [RN96].

Agentes reflejo simple En un agente de reflejo simple (figura 2.2) las reglas condición-acción permiten al agente establecer la conexión entre percepciones y acciones. Si observamos la figura 2.2, tendremos que los rectángulos se usan para indicar el estado interno en un momento dado del proceso de decisión del agente, y los óvalos representan la información de base utilizada en el proceso. Debe existir por lo tanto una función de interpretación de la entrada que genere una descripción abstracta del estado prevaleciente de la percepción. De la misma forma deberá disponer de una función que aporte la regla a aplicar en función de la entrada.

Agente reflejo de estado interno El agente reflejo simple explicado anteriormente funcionará sólo si se toma la decisión adecuada con base en la percepción en un momento dado. El problema surge debido a que los sensores no informan acerca del estado total del mundo. En estos casos el agente necesita almacenar algo de información en un estado interno que le permita discernir entre las percepciones del entorno que aún siendo iguales provocan una actuación dis-

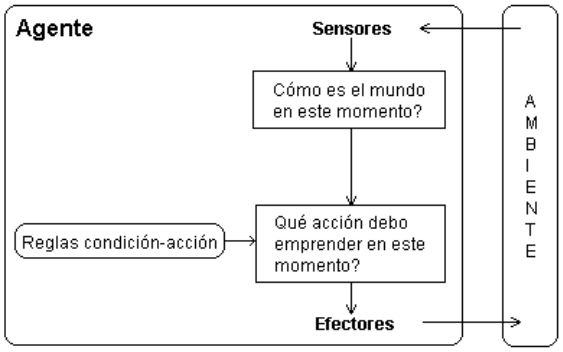


Figura 2.2: Representación de un agente reflejo simple

tinta. La actualización de esta información sobre el estado interno conforme va pasando el tiempo, exige la codificación de dos tipos de conocimiento en el programa del agente. En primer lugar se necesita cierta información de cómo está evolucionando el mundo, y en segundo lugar se necesita información sobre cómo las acciones del mismo agente afectan al mundo. En la figura 2.3 se puede observar la estructura del agente reflejo y también cómo se combinan las percepciones prevalecientes con el estado interno anterior para generar la descripción actualizada del estado siguiente.

Este tipo de agente debe incluir una función *actualizar-estado*, que es la responsable de crear la nueva descripción del estado interno para poder actuar consecuentemente a él.

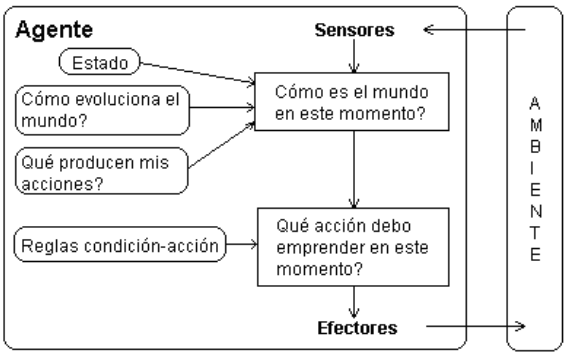


Figura 2.3: Representación de un agente reflejo de estado interno

Agente basado en metas Para decidir qué hay que hacer, no siempre basta con tener información acerca del estado que se encuentra en el ambiente, a veces se requiere cierto tipo de información sobre la meta. De esta forma el agente puede combinar la meta y los estados del ambiente para elegir aquellas acciones que permitan de mejor forma alcanzar la meta. En ocasiones esto es muy sencillo, cuando alcanzar una meta depende de responder con una sola acción; otras veces es más complicado, cuando el agente tenga que considerar, largas secuencias de actuaciones hasta que logre encontrar el *camino* que le lleve a la meta.

Por lo tanto en este tipo de agentes se utiliza el concepto de planificación. Estos son subcampos de IA que se ocupan de encontrar secuencias de acciones que permiten alcanzar las metas de un agente.

En la figura 2.4 se muestra la estructura del agente basado en metas. Se puede observar como se añade la función de evaluación de efectos de las acciones, previamente a tomar una actuación definiva.

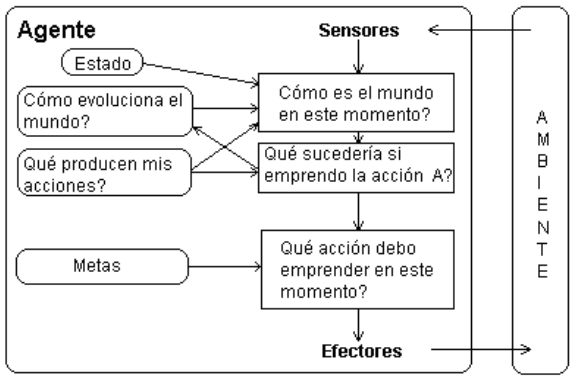


Figura 2.4: Representación de un agente basado en metas

Agentes basados en utilidad Las metas no bastan por sí mismas para generar una conducta de alta calidad. Por ejemplo, son muchas las secuencias de acciones que permitirían a un agente alcanzar su meta, pero de todas ellas, algunas son mejores en algún aspecto de especial interés, como la rapidez, seguridad, económica, etc. Las metas permiten establecer una tajante distinción entre estados *felices* e *infelices*, en tanto que mediante una medida las prestaciones más general sería posible establecer una comparación entre los diversos

estados del mundo (o secuencias de estados) de acuerdo a cómo exactamente harían feliz al agente en caso de lograrlos. Puesto que el término *feliz* no suena muy científico, la terminología que se acostumbra a utilizar es afirmar que si se prefiere un estado del mundo a otro, entonces ese estado ofrece mayor utilidad al agente.

Por lo tanto la utilidad es una función que correlaciona un estado y un número real mediante el cual se caracteriza el correspondiente grado de satisfacción, La completa especificación de la función de utilidad permite la toma de decisiones racionales en dos tipos de casos en los que las metas se encuentran con problemas.

- Cuando el logro de algunas metas implica un conflicto, y sólo algunas de ellas se pueden obtener, la función de utilidad definirá cual es el compromiso adecuado por el que optar

- Cuando son varias las metas que el agente podría obtener, pero no existe la certeza de poder lograr ninguna de ellas, la utilidad es una vía para ponderar la posibilidad de tener éxito considerando la importancia de las diferentes metas.

En la figura 2.5 podemos ver a un agente basado en la utilidad, y cómo se introduce su función de evaluación.

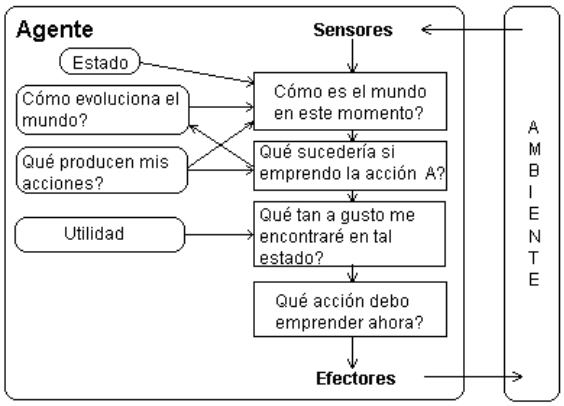


Figura 2.5: Representación de un agente basado en utilidad

2.2. Ambientes

En este apartado se va a estudiar el modo de acoplar un agente a un ambiente. La relación que existe de agentes con ambientes es siempre la misma: es el agente quien ejerce acciones sobre el ambiente que a su vez, aporta percepciones al agente. A continuación se verán los diferentes tipos de ambientes y cómo condicionan el diseño de los agentes.

Accesibles y no accesibles Si el aparato sensorial de un agente le permite tener acceso al estado de un ambiente, se dice que éste es accesible a tal agente.

Deterministas y no deterministas Si el estado siguiente de un ambiente se determina completamente mediante el estado actual y las acciones escogidas por los agentes, se dice que el ambiente es determinista.

Episódicos y no episódicos En un ambiente episódico, la experiencia del agente se divide en *episodios*. Cada episodio consta de un agente que percibe y actúa. La calidad de su actuación dependerá del episodio mismo, dado que los episodios siguientes no dependerán de las acciones producidas en episodios anteriores.

Estáticos y dinámicos Si existe la posiblidad de que el ambiente sufra modificaciones mientras el agente se encuentra deliberando, se dice que tal ambiente se comporta en forma dinámica en relación con el agente, de lo contrario se dice que es estático.

Discretos y contínuos Si existe una cantidad limitada de percepciones y acciones distintas y claramente discernibles, se dice que el ambiente es discreto.

2.3. Modelo PAMA

El objetivo de la IA es el diseño de un programa de agente. Para ello es necesario contar con una idea bastante precisa del ambiente, las percepciones y acciones, y por último de las metas. A este conjunto de datos se le conoce com matriz PAMA (Percepción, Acción, Meta y Ambiente).

Tipo de agente	Percepciones	Acciones	Metas	Ambiente
Conductor de taxi	• Cámaras, velocímetro, sistema GPS, sonar, micrófono	• Manejo del volante, acelerar, frenar, hablar con el pasajero	• Un viaje seguro, rápido, sin infracciones, cómodo, obtención de máxima ganancia	• Carretera, tráfico, peatones, cliente

Figura 2.6: Ejemplo de un agente conductor de taxi

En la figura 2.6 se puede ver la matriz PAMA para un agente conductor de taxi.

Sus percepciones se justifican por la necesidad de saber dónde se encuentra, quién más circula por su vía y a qué velocidad está circulando. Estos datos se obtienen de su sistema de percepción. Tambien es interesante tener un acelerómetro para medir la dureza de las curvas, así como conocer el estado del vehículo, por lo que necesitaremos varios sensores más.

Las acciones que puede producir este conductor de taxi son más o menos las mismas que las de un humano: control del acelerador, freno y del volante. Además necesitará un sintetizador de voz que le permita contestar al pasajero o a otros conductores.

Entre los objetivos a cumplir, principalmente deben ser: llegar de forma segura, con un consumo mínimo, tiempo reducido y respetar las normas de circulación. Desde luego, la consecución de alguno de estos objetivos está en conflicto con la consecución de otros, por lo que será necesario conceder ciertos márgenes. Por último sería necesario definir el ambiente de condición. ¿Irá por la ciudad o por el extrarradio? ¿Habrá problemas externos como nieve, lluvia, etc? ¿Se conducirá por el lado derecha o por la izquierda? obviamente vemos que mientras menos restringido sea el ambiente, menos complicado será el problema de diseño.

La cuestión ahora es, una vez reunidos los datos PAMA, seleccionar el modelo más adecuado de agente, según los tipos descritos en la sección 2.1. Normalmente el modelo más adecuado sería el de agente basado en utilidad, ya que permite una evaluación de las prestaciones, necesaria en nuestro ejemplo.

2.4. Modelo PAMA aplicado a la construcción de robots

Para mostrar las posibilidades que el concepto de agente ofrece en la implementación de capacidades de percepcion, actuación y consecución de metas en un determinado ambiente, en los siguientes apartados se va a aplicar el modelo PAMA para la construcción de μrobots. Concretamente se tendrán en cuenta las normas del concurso nacional HISPABOT [HIS], las cuales serán descritas, y a continuación se realizará una revisión de cada uno de los elementos PAMA aplicado a cada una de esas pruebas, concluyendo finalmente con el tipo de agente más adecuado que se puede utilizar en cada una de las categorías.

2.4.1. Rastreador

En esta prueba se valora la capacidad de un robot de realizar un recorrido por una pista con diversas rutas alternativas. Se trata de explotar la capacidad de seguimiento de caminos complejos, y de la capacidad de detección inequívoca de marcas para indicar rutas.

La pista será una línea negra sobre fondo blanco, y medirá entre 1.5 y 2.5 cm. Pueden tener tantas curvas y ángulos como la organización considere oportuno. Para tomar un camino correcto ante una bifurcación, habrá una marca paralela a la pista y separada entre 1 y 2 cm, midiendo entre 4 y 6 cm de largo. Estas marcas estarán a una distancia de entre 10 y 15 cm de una bifurcación. Por último indicar que los robots pueden tener un tamaño máximo de 20×30 cm.

La valoración final se hace por puntos, y en caso de empate por tiempo. Los puntos se acumularán en caso de penalización por haber tomado una bifurcación incorrecta o por tardar más de un tiempo determinado por la organización.

A continuación se presentará la especificación PAMA del agente.

2.4.1.1. Percepciones

El agente rastreador tiene que disponer de una percepción suficientemente precisa como para poder realizar un seguimiento de la línea con una cierta precisión y sin oscilaciones. Debido a los requisitos que se plantean en esta prueba, debe ser capaz de detectar las marcas de bifurcación correctamente,

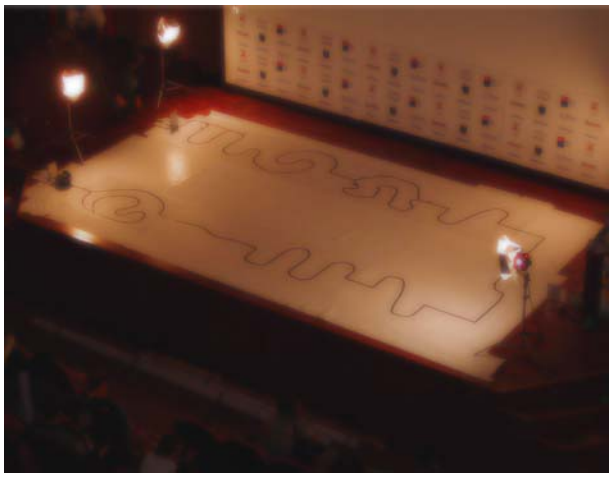


Figura 2.7: Imagen de una pista real de rastreadores

para poder tomar el camino correcto. Por lo tanto su sistema de percepción deberá aceptar las tolerancias propias del ancho de la pista, luminosidad, etc. Debido a la especial característica de la pista de ser negro sobre blanco, resultan adecuados los sensores infrarrojos reflectivos, de forma que si el haz infrarrojo rebota sobre negro, dará un voltaje alto, y si rebota en blanco será pequeño.

2.4.1.2. Acciones

El sistema efector de este agente debe ser capaz de realizar las maniobras indicadas en sus metas, de la forma más fiel posible. En esta prueba no prima la velocidad tanto como la capacidad de tomar el camino correcto, por lo que unos motores con suficiente par, que respondan inmediatamente serían adecuados. En este tipo de agente se suelen utilizar servos, ya que aunque no son demasiado veloces, disponen de un par alto y son soluciones cerradas con una fiabilidad grande. Las acciones serán la de moverse por la pista de acuerdo a las percepciones, de una forma lo más parecida posible al movimiento deseado.

2.4.1.3. Metas

Como la pista puede tener cualquier tamaño, se deben replantear las metas contínuamente y no llegar nunca a un estado final de parada. Una de sus metas será el seguimiento de la pista de la forma más fiel posible. Otra de sus metas

importante es la de tomar el camino correcto. Este objetivo necesitará una memoria para indicar a nuestro agente cual es el camino que debe tomar.

2.4.1.4. Ambiente

El ambiente, como ya se dijo, es una pista negra sobre un fondo blanco. Existe suficiente diferencia de color como para no confundirlos en condiciones normales. Sin embargo debemos ser capaces de filtrar las situaciones no ideales como pista sucia o pequeñas anomalías de los sensores.

2.4.1.5. Tipo de agente

Tipo de agente	Percepciones	Acciones	Metas	Ambiente
Rastreador	• Píxel blancos o negros	• Aplicar potencia a los motores	• Avanzar sobre la línea • Tomar bifurcaciones correctamente	• Pista negra sobre blanco

Figura 2.8: Matriz PAMA para la prueba de rastreadores

De acuerdo a la matriz PAMA de la figura 2.8, en esta prueba parece bastante adecuado plantear el agente reflejo con estado interno. No sería posible utilizar un agente de reflejo simple debido a que se necesita memorizar temporalmente hacia qué lado se debe tomar una bifurcación. Tampoco se pueden utilizar los agentes basados en metas o en utilidad, ya que la meta no conduce a un estado final o de parada, sino a un estado intermedio contínuo.

2.4.2. Velocista

Esta prueba valora la velocidad de un robot sobre una pista. Entra en juego sobre todo la velocidad máxima que un robot puede alcanzar siguiendo una pista, y cómo se resuelven los problemas de controlabilidad y autonomía. En este caso se trata de una pista con cuatro líneas negras paralelas durante todo el recorrido, con un ancho entre 1.5 y 2.5 cm. Estas calles tendrán un ancho entre 10 y 20 cm. El robot podrá guiarse con cualquiera de estas líneas, pero no podrá pisar nunca las exteriores ya que sería descalificado.

La organización medirá el tiempo que el robot tarda en dar un número determinado de vueltas. Hay una garantía de que las curvas serán de un radio

de 40 cm como mínimo. En la pista puede haber rampas con una pendiente máxima del 20 %. Por último señalar que las dimensiones máximas del robot son de 20×30 cm, con una altura máxima de 15 cm.

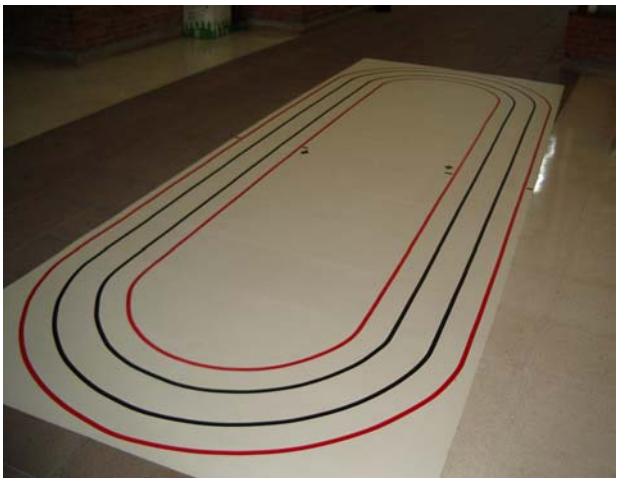


Figura 2.9: Pista para la prueba de velocistas

2.4.2.1. Percepciones

Al igual que el agente rastreador, este agente debe disponer de una percepción similar. Sin embargo debido a que prima la velocidad, todo sistema sensorial que aporte precisión en la lectura, redundará en un beneficio importante para realizar un seguimiento fiel de la pista. Esto se conseguirá en función del número de sensores utilizados, de su disposición y su posición en el robot.

2.4.2.2. Acciones

Es deseable que su sistema de actuación sea mucho más rápido que en el agente rastreador, ya que si bien no se necesita un par tan grande por ser curvas mucho más abiertas, se requiere una velocidad mucho mayor para conseguir la meta. Las acciones que se deben tomar son las de realizar movimientos sobre la pista, de forma que sean fiel reflejo de los deseos de movimiento.

2.4.2.3. Metas

Su objetivo o meta simplemente es seguir la línea de la forma más fiel y rápida posible. Este agente al igual que el rastreador no es capaz de saber si ha conseguido su meta, ya que el tamaño de la pista es desconocido y no debe existir una meta de terminación.

2.4.2.4. Ambiente

El ambiente es similar a la pista de rastreadores: una pista negra sobre fondo blanco, con la suficiente diferencia de color como para no plantear problemas de detección. Sin embargo hay que filtrar pequeños errores de los sensores, que pueden hacer ver todo negro en un momento dado, así como manchas o suciedad de la pista.

2.4.2.5. Tipo de agente

Tipo de agente	Percepciones	Acciones	Metas	Ambiente
Velocista	• Píxel blancos o negros	• Aplicar potencia a los motores	• Seguir la pista de forma rápida	• Pista negra sobre blanco

Figura 2.10: Matriz PAMA para la prueba de velocistas

La figura 2.10 recoge las conclusiones del estudio previo. Se puede deducir que el tipo de agente más adecuado es el de estado interno, ya que ante una condición se ejecuta una acción pero en la situación de perder la línea requiere actuar de otro modo, que podría ser torcer hacia el lado que perdió la pista. Por lo tanto resulta necesario un estado interno, pero no necesita evaluar la consecución de las metas, por lo que se justifica directamente la elección tomada.

2.4.3. Laberinto

El objetivo de esta prueba es conseguir salir de un laberinto. Éste se compone de 81 casillas de 40×40 cm, en disposición de 9×9 casillas. Las paredes del laberinto miden 25 cm de alto y serán blancas. Dispondrá de dos salidas. Los participantes podrán conocer el laberinto unos días antes del concurso, por lo que se puede aprovechar esta característica para el algoritmo utilizado. En caso

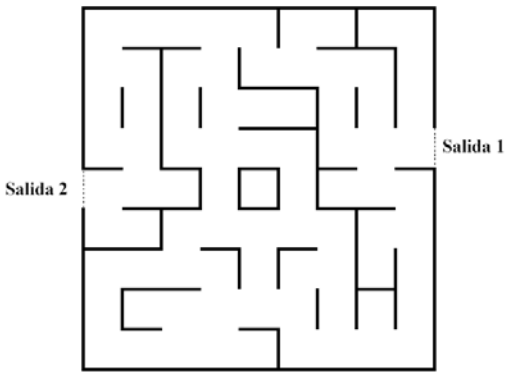


Figura 2.11: Plano de un laberinto

de salir del laberinto se contabilizará el tiempo empleado. Si no se salió del laberinto se valorará lo cerca que se estuvo de salir. No se conocerá la posición ni el sentido de inicio del robot. Los robots podrán tener unas dimensiones máximas de 20×30 cm de planta y 25 cm de altura como máximo.

2.4.3.1. Percepciones

Un agente de laberinto debe ser capaz de localizarse dentro del laberinto y salir de él. Desde el punto de visa de percepción, debe ser capaz de percibir las distancias a las paredes que tiene enfrente, atrás y a los lados. De esta forma, y al cabo de realizar algunos movimientos debería conseguir la primera meta que es saber dónde está. La solución del laberinto debería estar precalculada, por lo que la meta ahora sería tomar el camino correcto para salir. También resulta muy recomendable una capa intermedia en la percepción. Esta capa debería dar una estimación de la casilla en la que se encuentra el agente. Con estas dos consideraciones el agente tendrá una visión de alto nivel del laberinto, aislándolo de la complejidad para poder centrarse simplemente en las metas.

2.4.3.2. Acciones

Las acciones a realizar pueden ser el movimiento entre las casillas, y su ambiente, como ya se dijo, serán los pasillos del laberinto. Sin embargo este tipo de agentes debe solicionar un problema básico, que es el de poder realizar los movimientos básicos entre casillas. Por lo tanto resultaría recomendable una

capa de actuación que aisle al agente de la dificultad de realizar movimientos básicos de avanzar, girar, etc de forma que siempre se encuentre en el centro de la casilla.

Debido a las reducidas dimensiones de las casillas, el robot debería ser capaz de realizar giros sobre sí mismo, por lo que se aconseja disponer de motores en disposición diferencial.

2.4.3.3. Metas

Claramente en esta prueba la meta es la localización de la salida del laberinto en el menor tiempo posible. Debe ser capaz de calcular cual es la salida más cercana para dirigirse a ella.

2.4.3.4. Ambiente

El ambiente del laberinto serán las casillas. Cada una de estas casillas puede tener entre una y tres paredes. La unión de estas casillas constituirán los pasillos por los que se moverá el robot.

2.4.3.5. Tipo de agente

Tipo de agente	Percepciones	Acciones	Metas	Ambiente
Laberinto	• Distancia a las cuatro paredes	• Aplicar potencia a los motores	• Salir del laberinto rápidamente	• Laberinto con dos salidas

Figura 2.12: Matriz PAMA para la prueba de laberinto

Si observamos la figura 2.12, claramente en este tipo de agentes debemos utilizar un agente basado en metas, ya que se puede conocer de antemano el laberinto así como la forma de salir desde cualquier casilla.

2.4.4. Sumo

En esta prueba se compite contra otro robot. El objetivo es expulsar al adversario del tatami. Ésto no es más que un tablero circular que mide 175 cm de diámetro de color negro. Existe una línea blanca de 5 cm de ancho para delimitar el borde del tatami. Los robots no podrán pesar más de 3 kg y medir

20×20 cm como máximo en su situación de reposo. Es posible que posteriormente al arranque desplieguen palas o planchas para facilitar la expulsión del adversario (ver figura 2.13). La altura de los robot será libre. Al igual que las

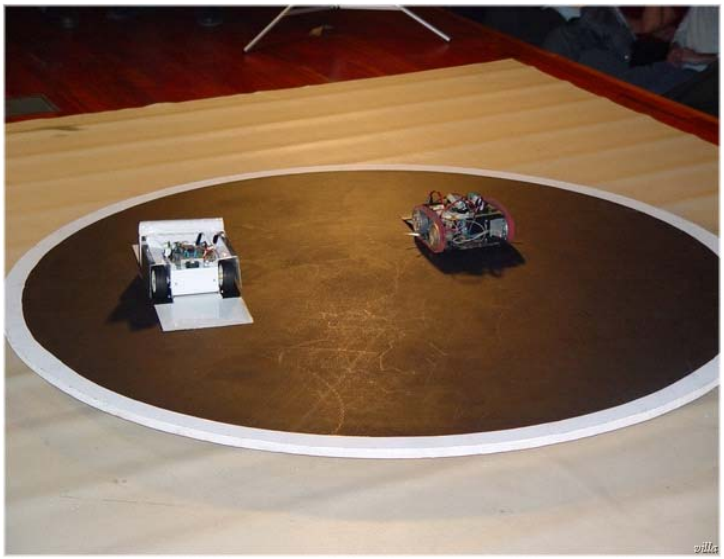


Figura 2.13: Pista de sumo con dos robot dispuestos a enfrentarse

competiciones de sumo humanas, no se podrá golpear ni provocar desperfectos deliberadamente, existiendo un sistema de penalización para ello.

2.4.4.1. Percepciones

El sistema de percepción de estos agentes se basa en la detección del final de la zona de juego, y en la detección del adversario. La zona de juego se marca con verde, y su límite en blanco, por lo que al igual que en los agentes rastreadores y velocistas, resulta fácil utilizar sensores de reflexión de infrarrojos para detectar límites.

Resulta más complicada la percepción del adversario, ya que no existe un sensor idóneo que sea capaz de detectar en todas las situaciones al adversario. El intento de medir la distancia a la que está el oponente suele resultar infructuoso, ya que no conocemos su trayectoria o ángulo de ataque. Empíricamente si observamos a los mejores robots de este tipo, suele ser suficiente detectar solamente el contacto físico con el adversario. También resultaría sumamente interesante

percibir si en nuestro intento de empujarlo, estamos consiguiendolo o no. Por ello sería muy importante detectar esta condición. Si decidimos percibir si las ruedas giran o si el consumo es normal, no sabremos si el robot está patinando, por lo que en este caso se aconsejan sistemas con un funcionamiento similar a un ratón óptico que nos indique si efectivamente el robot se está desplazando o está patinando, y sobre qué eje lo hace.

2.4.4.2. Acciones

Los actuadores de un agente de sumo deben ser motores acoplados a una reductora, de forma que el par que desarrollen sea muy grande, resultando muy importante transmitir esa potencia al suelo para conseguir empujar al adversario. Las acciones directas pueden ser simplemente las de moverse por el tatami y no salirse de él.

En la práctica, algunos robots de sumo utilizan unas planchas metálicas que se abren en el arranque y tienen el objetivo de subir encima al adversario para expulsarlo del tatami.

2.4.4.3. Metas

Las metas o comportamientos serán las de ser atacante o defenderse. Si se ataca, debemos percibir que es el robot el que empuja, ya que en caso contrario debería tomar una actitud de defensa. Efectivamente, un error muy común en robots de este tipo consiste en que los dos esten empujándose, sin lograr moverse del sitio aunque las ruedas de ambos giren, consiguiendo agotar la energía y resentir las mecánicas.

2.4.4.4. Ambiente

En cuanto al ambiente para este tipo de agentes, no es del todo correcto decir que es solamente el tatami, ya que es posible que el adversario suba encima de sus planchas o que una parte de nuestro agente se encuentre fuera del tatami pero no haya caido totalmente. Se debe distinguir estos tipos especiales del ambiente y tratar de percibir en cual de ellos está el robot para tratar de realizar el mejor comportamiento sin equivocaciones.

2.4.4.5. Tipo de agente

Tipo de agente	Percepciones	Acciones	Metas	Ambiente
Sumo	• Píxel blancos o negros • Contacto con oponente	• Aplicar potencia a los motores	• Expulsar al adversario	• Tatami circular

Figura 2.14: Matriz PAMA para la prueba de sumo

Si observamos la figura 2.14, concluiremos que sería posible implementar este agente sobre cualquiera de los dos modelos siguientes:

- Agente reflejo de estado interno. Se guarda un estado interno sobre la situación actual, para tomar el comportamiento adecuado de continuar un ataque o de realizar una evasión.

- Agentes basados en utilidad. Este tipo de agente podría ser adaptativo al nivel de dificultad del enemigo, ya que puede evaluar el grado de acierto en los ataques, probando diversas técnicas de ataque hasta encontrar el que haga cumplir la meta de mejor forma. Sin embargo el inconveniente es que añade la necesidad de implementar una función que evalue el grado de acierto, no siendo siempre sencillo.

2.5. Conclusión

En la figura 2.15 se verá un resumen completo con todas las pruebas del concurso HISPABOT, así como el agente seleccionado en su implementación

Sin duda, en el concurso HISPABOT se pone a prueba la capacidad de autonomía y el nivel de inteligencia de los robot. Resulta curioso observar como despiertan un interés especial los robot de sumo, implementados con el tipo más complejo de agente. Estos son los que más interaccionan con el médio dando verdadera sensación de inteligencia.

Categoría	Agente elegido	Percepciones	Acciones	Metas	Ambiente
Rastreador	Reflejo con estado interno	• Píxel blancos o negros	• Seguir la línea	• Avanzar sobre la línea • Tomar bifurcaciones correctamente	• Pista negra sobre blanco
Velocista	Reflejo con estado interno	• Píxel blancos o negros	• Seguir la línea	• Avanzar sobre la línea	• Pista negra sobre blanco
Laberinto	Basado en metas	• Distancia a las cuatro paredes	• Posicionarse en el laberinto • Avanzar por el camino correcto	• Salir del laberinto	• Laberinto con dos salidas
Sumo	Basado en utilidad	• Píxel blancos o negros • Contacto con oponente	• Empujar al adversario • Evitar salirse	• Expulsar al adversario	• Tatami circular

Figura 2.15: Matriz PAMA para todas las pruebas de HISPABOT

3

Descripción física del agente

En el capítulo previo se obtenía la matriz PAMA para los distintos agentes pertenecientes a las diversas categorías del concurso. Se han llevado a la práctica dos robots: Diego para la prueba de rastreadores, y D2 para las pruebas de rastreador y velocista.

En este capítulo se va a hacer una descripción de los dos robots implementados, atendiendo a sus características mecánicas y electrónicas. No pretende ser un manual de cómo construir un robot, pero sí una base de conocimiento de la que partir para realizar prototipos.

3.1. Sistemas físicos del agente

Como ya se ha comentado, un agente es una pareja hardware y software. Como ya se justificó en el capítulo anterior, se ha elegido el agente reflejo con estado interno para las pruebas de rastreadores y velocistas. En este apartado

se va a analizar cómo partiendo del concepto de este agente, podemos llegar a concluir el hardware utilizado en la construcción de los robots Diego y D2. En

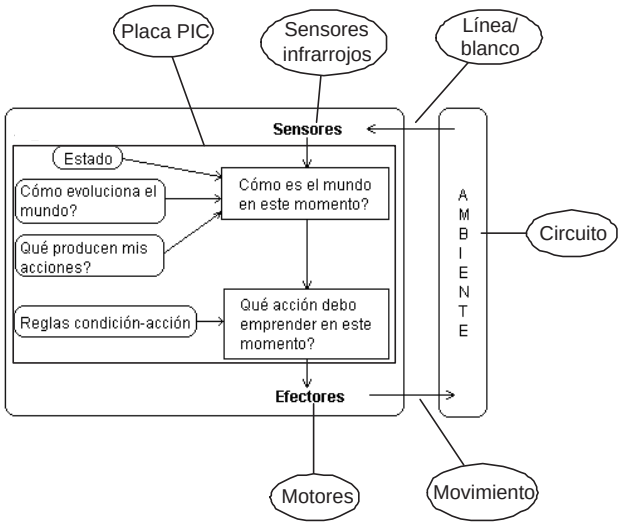


Figura 3.1: Agente con estado interno desde el punto de vista del hardware

la figura 3.1 se puede ver el agente y los elementos hardware que lo componen. Podemos ver como los sensores, propocionan la percepción del ambiente al robot. La percepción es el mecanismo de introducir de forma electrónica las lecturas de la línea dentro del sistema. Estos sensores deben ser capaces de distinguir de forma inequívoca los dos colores posibles de la pista para las pruebas de rastreadores y velocistas: blanco y negro. Sin embargo su tolerancia debe ser adecuada para no confundir los casos imperfectos. Los sensores más comunes son los de infrarrojos, ya que permiten modificar las tolerancias en la forma que necesitemos.

Los actuadores proporcionan una interacción con el ambiente. En este caso se encargan de realizar el movimiento siguiendo la pista. Los motores deben intentar reproducir fielmente los movimientos que resulten de los cálculos. Normalmente estos motores tendrán cajas reductoras para proporcionar la velocidad y/o potencia adecuadas. También debemos considerar las ruedas como una parte muy importante del robot. Deben tener unas dimensiones adecuadas para mantener una relación de velocidad/potencias adecuada, así como un agarre óptimo.

Dentro del sistema de actuación, y en los sistemas que se estime oportuno, se incluirán los mecanismos hardware para el control de velocidad que garanticen la velocidad correcta.

Los sistemas de actuación representan el 95 % del consumo total de energía de los robots. Este consumo será desconocido, dado que su entorno también lo es. No se conoce de antemano el esfuerzo que tendrán que realizar sus motores.

El resto de elementos del agente son partes del software que se ejecutará normalmente sobre una arquitectura sencilla, robusta y económica. Típicamente son sistemas microprocesador empotrados o microcontroladores.

Bajo el concepto de agente, una vez distinguidos los distintos elementos que se necesitan para su realización, se detallarán las soluciones hardware adoptadas en los dos robots implementados: un μrobot rastreador (Diego), y otro velocista/rastreador (D2).

3.2. μrobot rastreador Diego

Diego es un robot que se ha construido con materiales muy sencillos y de bajo coste. Sin embargo su simpleza no ha sido sencilla de obtener pues se han utilizado distintas configuraciones físicas hasta llegar al resultado final. Este robot ha sido el primero de su especie realizado en el Departamento de Informática y Automática de la Universidad de Salamanca. Los conocimientos adquiridos servirán de base para posteriores desarrollos.

3.2.1. Plataforma mecánica

Diego está realizado en una plataforma de plástico de 2 mm de grosor (figura 3.2). Este soporte contiene en su parte superior el microcontrolador y las pilas. En su parte inferior alberga los sensores y los actuadores. En la figura 3.3 se pueden ver las dimensiones de Diego.

De todos los sistemas de tracción que se han revistado, se ha elegido el sistema diferencial, ya que es una solución muy sencilla de implementar, disponiendo de forma implícita de capacidad de tracción y dirección. Esto se ha conseguido gracias a dos motores en configuración opuesta acoplados a sendas ruedas.

Los motores utilizados son servos modificados, los cuales proporcionan mu-

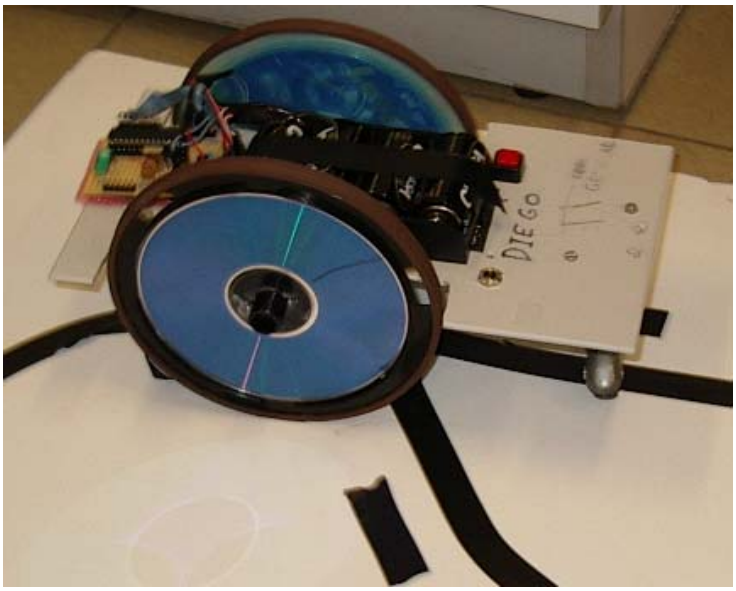


Figura 3.2: Vista general de Diego

cha potencia y poca velocidad: exactamente 1 rpm. Por lo tanto las ruedas que se han elegido son de un gran diámetro: 145 mm. Se han utilizado tapaderas de plástico, y para darle consistencia se le ha pegado un CDROM. Para evitar que derrape, se le ha colocado burlete de caucho en su banda de rodadura.

Dispone de una rueda loca en su parte trasera que sirve de apoyo a la estructura sin entorpecer la dirección tomada.

Debido a su disposición, este robot es capaz de realizar giros con un radio mínimo de 55 mm. Esto sucede en los giros totalmente diferenciales, cuando las ruedas giran a la misma velocidad pero en sentidos opuestos.

Sus sensores están colocados en forma de V invertida, de forma que coinciden con la trazada del giro diferencial. Esta configuración también es útil para eliminar situaciones ambiguas en las lecturas.

3.2.1.1. Actuadores

El sistema de actuadores de Diego se compone de dos servos modificados para giro libre [UYM01]. Estos están acoplados a unas ruedas de plástico de 14 cm de diámetro. Un servo es una solución cerrada que se compone de un motor,

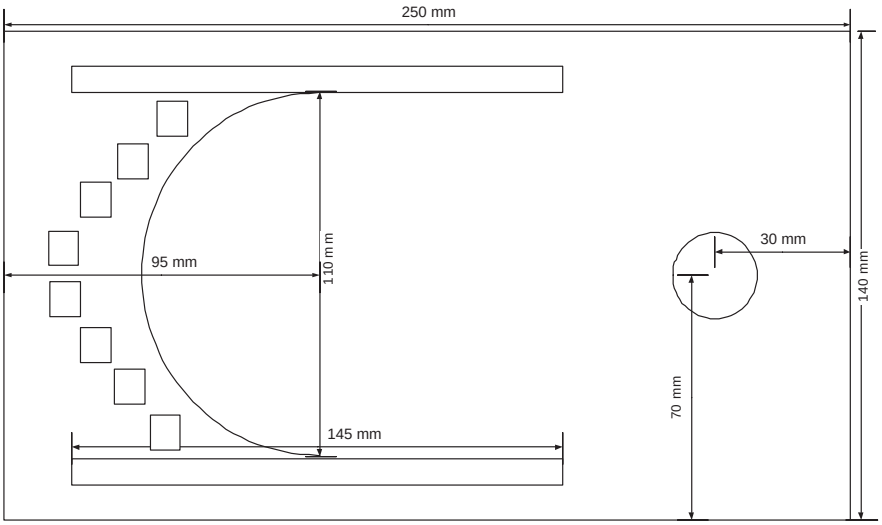


Figura 3.3: Dimensiones de Diego

una reductora y un driver de potencia que permite controlar la velocidad de giro del motor con entradas TTL, típicamente la salida del microcontrolador.

Sin embargo los servos tienen una característica que no es adecuada para nuestro robot. Estos disponen de unos topes físicos que hacen imposible que puedan girar más de unos 180 grados. También originalmente tiene un potenciómetro que gira a la vez que el eje final y sirve como realimentación de la electrónica integrada en el servo para saber cual es la posición en la que se encuentra. Estos dos elementos son eliminados (figura 3.4): se elimina el tope físico y se deja constante el potenciómetro. De esta forma se consigue el giro libre del eje. Los servos tienen la gran ventaja de ser soluciones cerradas, con

Figura 3.4: Etapas para el trucaje de un servo

un desarrollo final adecuado para aplicaciones como las que nos ocupan. Tienen un par de giro muy grande y son muy duraderos.

3.2.2. Diseño electrónico

Se ha utilizado el PIC 16F876 de MICROCHIP [MIC] como procesador del sistema. Su esquema es muy simple como se aprecia en la figura 3.5:

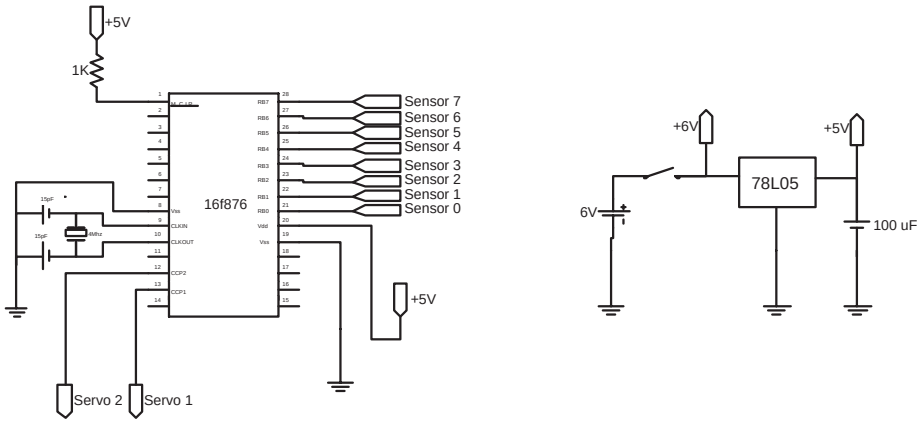


Figura 3.5: Conexiones del PIC y su fuente de alimentación

Este circuito es una solución integrada que contiene todos los elementos básicos para su funcionamiento inmediato sin una gran necesidad de componentes externos.

Dispone de 8 KB de memoria FLASH para código y 368 bytes de RAM. Se trata de un procesador con tecnología RISC, capaz de ejecutar una instrucción en 4 ciclos de reloj. Existen versiones de 4 y de 20 MHz. En nuestro caso hemos utilizado la versión de 4 MHz por resultar suficiente para el fin que va a estar dedicado. No se han utilizado muchas de sus funciones avanzadas como temporizadores, puertos A/D, UART, comunicación I2C, etc, haciendo uso básicamente de uno de sus puertos de 8 bits.

Como se puede observar, dispone de un cristal para conseguir una frecuencia de trabajo de 4 MHz. El pin 1 del PIC se utiliza para hacer un reset (con cero lógico). En nuestro caso siempre está a 1 gracias a la resistencia de 1K conectada a la alimentación. Esto es debido a que es suficiente con encenderlo y apagarlo para que retorne a un estado inicial.

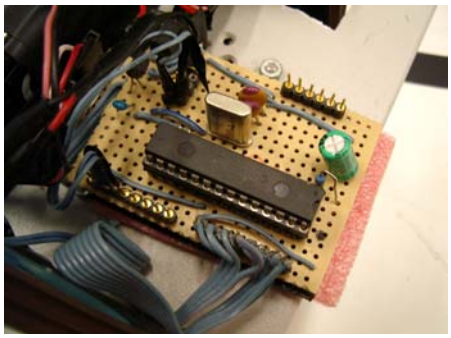


Figura 3.6: Aspecto de la placa de Diego

Las salidas PWM, pines 12 y 13, se conectan directamente a las entradas de los servos. Estos dispositivos se describen en el apartado de actuadores.

Los 8 sensores de infrarrojos, se conectan a los pines como se puede ver en el esquema. Estos pines se corresponden con los del puerto B de 8 bits del PIC. De esta forma con una sola lectura a este puerto tendremos directamente la entrada, representando cada bit a un sensor.

La alimentación se obtiene por medio de un regulador, capaz de proporcionar 5 V y una intensidad máxima de 100 mA, más que suficientes para el consumo típico de nuestro PIC. La salida de 6 V de esta fuente, conectada directamente a las pilas, se utiliza para alimentar a los servos y a los sensores, que representan la mayor parte del consumo del sistema (ver figura 3.5). De esta forma también se consigue aprovechar todo el potencial de las pilas consiguiendo así una mayor velocidad final de nuestro robot.

Por último, cabe comentar que el montaje de la fuente de alimentación y del PIC se ha realizado en una misma placa perforada, cuya imagen se muestra en la figura 3.6.

El control de posición original de los servos se realiza con PWM (modulación por ancho de pulso). Normalmente el servo se mantienen en la posición central si el pulso está en alto 1.5 ms, cada 10 ms. Si hacemos que el tiempo en alto esté por encima o debajo de 1.5 ms, girará en una u otra dirección más o menos rápido según la diferencia de tiempo de la posición central, hasta conseguir igualarse a su realimentación: el potenciómetro

Sin embargo al hacer la modificación se cambia un poco el sistema de control,

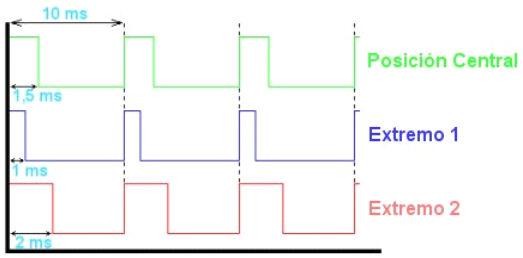


Figura 3.7: Trenes de pulsos para el control de servos

ya que el servo no logra igualarse con su referencia por lo que girará contínuamente.

3.2.2.1. Sensores

El sistema sensorial del robot Diego se compone de 8 sensores CNY-70 [VIS] en disposición de V invertida al sentido de la marcha. Cada uno de ellos se compone de diodo emisor de infrarrojos y de un fototransistor como puede verse en la figura 3.8.

Los sensores CNY-70 tienen una salida analógica. Dan valores entre 0.2 V y 4 V si reflejan el blanco o negro respectivamente. Sin embargo también proporcionan valores intermedios de voltaje en otros niveles de grises.

El montaje completo de cada uno de estos sensores, se compone de un CNY70, y dos resistencias, ambas utilizadas para limitar la intensidad máxima que circula por el LED y por el fototransistor. El esquema final de cada sensor se muestra en la figura 3.8. Cada una de las salidas de este esquema se conecta a una entrada del PIC (pines 28 a 21). En la figura 3.9 se puede ver el montaje final y la disposición de los sensores.

La salida de los sensores se conecta directamente a la entrada digital del PIC, de forma que siempre se leerá 0 ó 1 lógico. Estas entradas disponen una puerta *Schmitt trigger*, que hacen posible que no haya histéresis en la entrada.

Este tipo de puertas hacen posible que cuando reciben una señal de entrada menor de 1.7 v, pone su salida a 0 lógico. Si es mayor a 1.7 v la ponen a 1 lógico (5 v). Sin embargo si ya se encontraba con la salida a 1, no la pone a cero hasta

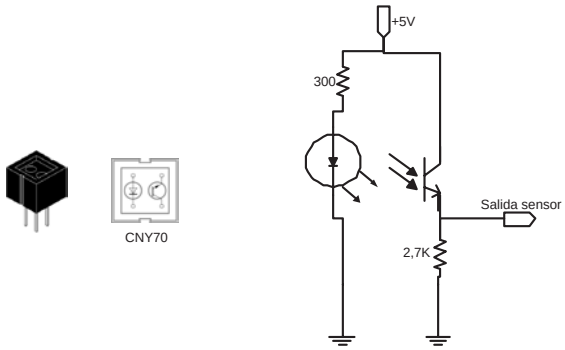


Figura 3.8: Aspecto de un CNY-70 y esquema de un sensor

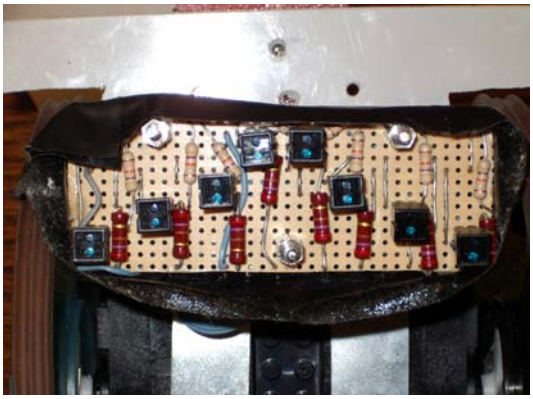


Figura 3.9: Placa de sensores de Diego

que su entrada no esté por debajo de 0.9 v. En la figura 3.10 se puede ver un ejemplo de su modo de operación.

3.3. μrobot rastreador y velocista D2

El nombre D2 proviene de Diego Versión 2. Para el desarrollo de D2, se pretende utilizar todo el conocimiento adquirido en la versión anterior. Se trata de obtener un robot más rápido que permita competir en las pruebas de rastreadores y de velocistas. Para ello se ha dotado de una serie de elementos que permiten llevar a cabo estas tareas.

En primer lugar, D2 ha sido dotado de un sistema de actuadores más potente que Diego. Sus motores son más grandes, así como toda la circuitería de control

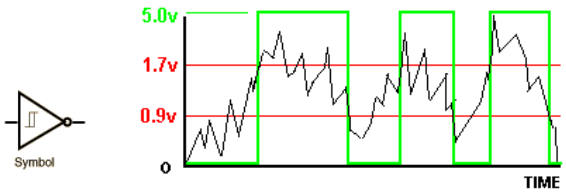


Figura 3.10: Ejemplo de funcionamiento de una puerta Schmitt trigger

de estos motores que ha tenido que dimensionarse adecuadamente para soportar correctamente el consumo total del robot. También se le ha dotado de unas ruedas más anchas que permiten un gran agarre en curva. Únicamente dispone de dos ruedas situadas en la parte trasera del robot. En la parte delantera se ha colocado un punto de apoyo para lograr un menor rozamiento.

Por último, su sistema sensorial se ha separado del eje de tracción lo suficiente como para asegurar una anticipación adecuada en las curvas o irregularidades de la pista.

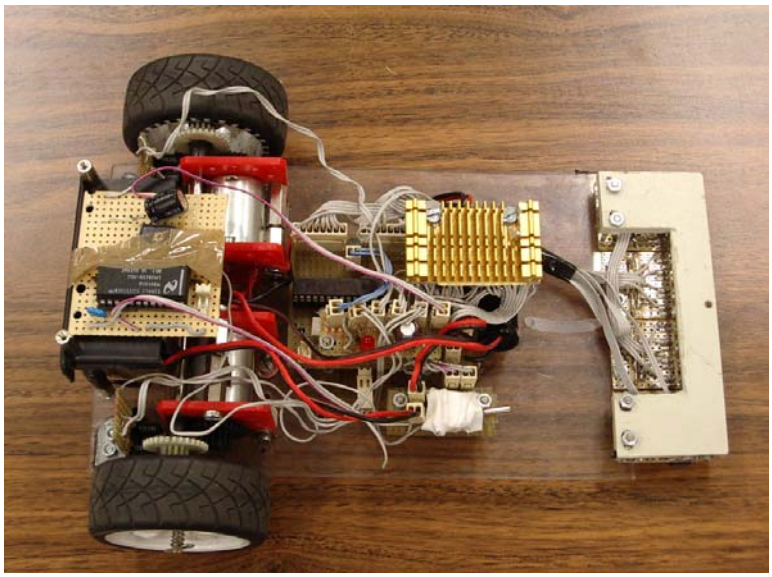


Figura 3.11: Vista general de D2

3.3.1. Evolución mecánica

Inicialmente los sensores de D2 estaban muy cerca del eje de las ruedas. Esto permitía realizar giros muy cerrados pero a costa de tener poca anticipación en la detección de las curvas. Con este sistema no se conseguía tener mucha velocidad y el resultado no era satisfactorio en la prueba de velocistas. Se trataba de una configuración similar a Diego.

Posteriormente se colocaron los sensores unos 15 cm por delante del eje de tracción, obteniendo un resultado bastante bueno. Este montaje de los sensores ha sido el que finalmente se ha mantenido.

En cuanto al sistema de tracción, inicialmente se disponía del sistema con dos motores sin control de velocidad, por lo que hacía que el robot alcanzase una gran velocidad en las rectas, pero sin embargo al llegar a las curvas se desestabilizaba y disminuía su rendimiento.

El último paso fue dotar a D2 de un tren de tracción más potente con un control de velocidad de tipo proporcional, obteniendose un resultado muy satisfactorio. Sin embargo también en este paso hizo falta dotar a D2 de unas etapas amplificadoras para el control de los motores mucho más potente, ya que con las originales no eran suficientemente.

3.3.2. Plataforma mecánica

La versión final de D2 se ha realizado sobre una plataforma de metacrilato de 13×25 cm. Este tipo de material tiene una gran resistencia y un peso razonable. Los taladros se realizan de forma muy segura y rápida.

De forma experimental, se ha concluido que el centro de masas debía estar próximo al eje de las ruedas. No es la mejor solución tenerlo en el propio eje de tracción ya que los giros adquieren una inercia mayor y hace que tenga un tiempo de respuesta mayor, aunque como contrapartida alcanza algo más de velocidad.

En la figura 3.12 se pueden ver las dimensiones de D2 así como la disposición de los sensores y actuadores.

También se han probado varios sistemas de apoyo de la parte delantera del robot, ya que es el tercer apoyo necesario por disponer solo de dos ruedas traseras. Los problemas que encontramos fueron:

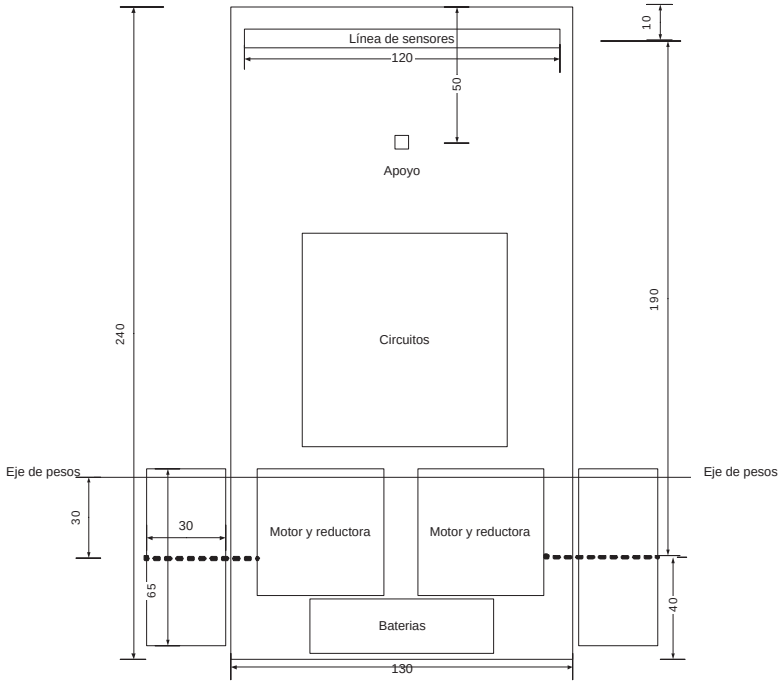


Figura 3.12: Dimensiones de D2

- Las ruedas locas entorpecen los giros ya que tienden a hacer que el robot tarde más en girar.
- Las bolas o patines hacen que la parte delantera se salga en las curvas debido a que no ofrecen resistencia a la fuerza centrífuga. Estos sistemas hacen que caiga toda la responsabilidad del giro en los motores, realizando un esfuerzo mayor.
- El apoyo de teflón finalmente ha sido el elegido, ya que ofrece un equilibrio bastante razonable entre la resistencia al avance y el agarre para desplazamientos laterales.

3.3.3. Actuadores

Inicialmente se utilizaban motores del tipo RE-140 [MAB], como el que se puede ver en la figura 3.13.

Estos motores son los típicos de juguetería con un tamaño bastante pequeño. Tienen la característica de que son muy baratos, aunque sus escobillas son

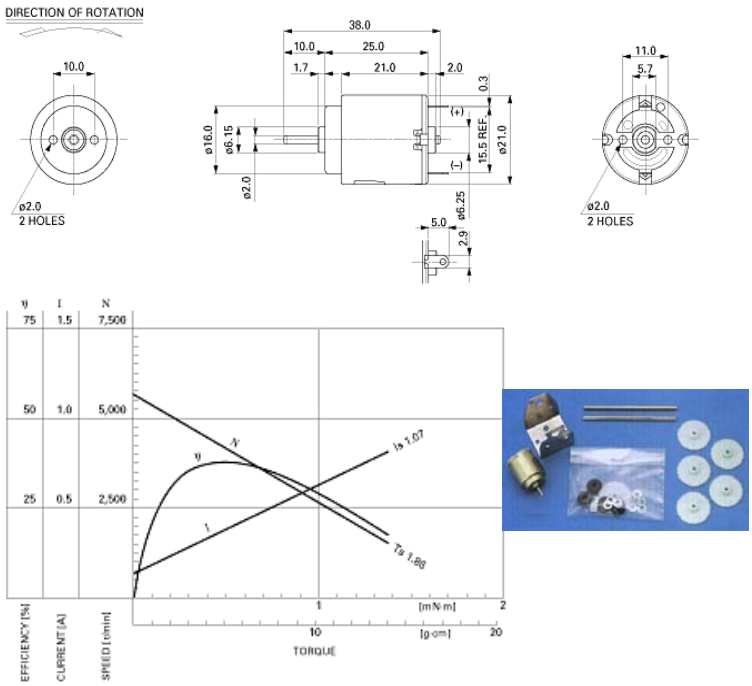


Figura 3.13: Características del motor RE-140 y su kit de engranajes

de chapa y se desgastan fácilmente. Estos motores tenían un comportamiento aceptable, sin embargo se empezaba a notar que aunque conseguía ir rápido, no tenían suficiente potencia para la prueba de velocistas. Puede verse en sus especificaciones como las revoluciones tienden a cero cuando existe un par resistente sobre el motor de unos 20 g.cm (gramos por centímetro). Esta situación ocurría en rampas, le costaba mucho trabajo subirlas si ya iba en marcha y apenas era capaz de arrancar si se encontraba parado encima de ellas. Esto demostraba que en la rampa los motores soportaban esfuerzos superiores a 20 g.cm.

El siguiente escalón en cuanto a motores lo representan los RE-280 [MAB]. La gráfica característica de estos motores se puede ver en la figura 3.14. Su eficiencia es máxima alrededor del punto de par 20 g.cm, en el que el RE-140 se paraba. Aqui, el nuevo motor obtiene unas 7500 rpm a 6 V. Parece bastante adecuado para la nueva versión de D2. Sin embargo debemos estar prevenidos, ya que tendrá más peso y deberá soportar un poco más de esfuerzo. Para cubrir

suficientemente este incremento de esfuerzo, se ha estimado que el esfuerzo ahora será de unos 30 g.cm en la rampa.

También se plantea el problema de la alimentación. Según la gráfica como su consumo en un momento dado puede aumentar hasta 4.7 A. No se trata de situaciones normales, pues su esfuerzo estará entorno a los valores antes comentados y su consumo normal será sobre 1 A, pero debemos tener en cuenta que la pista no es perfecta y que habrá picos de esfuerzo y consumo muy altos.

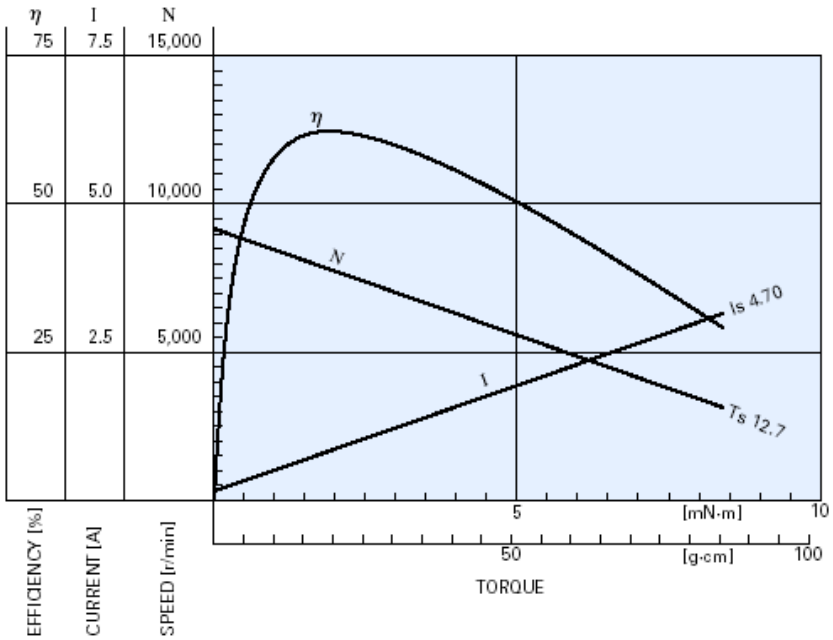


Figura 3.14: Características del motor RE280

La velocidad punta alcanzada con los motores RE-140 era de 1.5 m/s aproximadamente, y también giraba a unas 8000 rpm a 6 V. Por lo tanto los nuevos motores tenían unas prestaciones similares excepto su par que era notablemente mayor.

Las ruedas proceden de coches de modelismo de escala 1/10. Estas ruedas tienen un diámetro de 65 mm que equivale a un perímetro aproximado de 204.2 mm. Para que puedan ir a 1.5 m/s deben girar a $\frac{1500}{204,2} = 7{,}3$ vueltas por segundo.

Como ya se dijo antes, la velocidad óptima del motor RE-280 en carga son

7500 rpm, equivalentes a 125 rps (revoluciones por segundo). Eso equivale que la relación entre la rueda y el motor debe ser de $\frac{125}{7,345} = 17,1$.

Sin embargo este motor se compra con un kit de engranajes de doble corona con 30 y 12 dientes (ver figura 3.15). La relacción más parecida a la calculada anteriormente la conseguimos con tres engranajes: $(\frac{30}{12})^3 = 15,625$

La zona nominal de trabajo estimada, estará entre 5 y 30 g.cm. Si observamos la gráfica de la figura 3.14, podemos observar como el motor a 6 v, y con esta carga, girará entre 9000 rpm y 7000 rpm aproximadamente. Para la relacción que disponemos, esto equivale aproximadamente a velocidades entre 2 m/s y 1.55 m/s.

Para mantener la velocidad máxima en 1.5 m/s, y dada la relación disponible, el motor irá a menos revoluciones de las inicialmente calculadas, aproximadamente a unas 6900 rpm (115 rps). Aún así, si observamos su característica vemos como está en una zona con un rendimiento muy alto (60 % aproximadamente) que hace perfectamente posible la utilización de este motor.

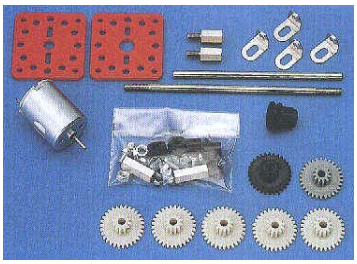


Figura 3.15: Motor RE280 y su kit de engranajes

Estos motores, al igual que los RE-140, tienen la desventaja de que sus escobillas son de chapa y se desgastaban rápidamente. La solución final fue modificar estos motores introduciendole escobillas de grafito, de forma que su duración es mucho mayor. También aumenta sus dimensiones bastante como se puede ver en la figura 3.16. Esto nos lleva a incrementar el ancho final del robot y a tener un peso mayor.

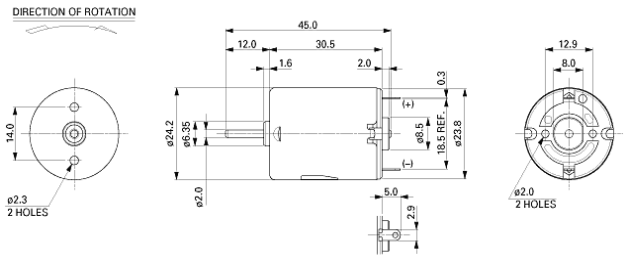


Figura 3.16: Dimensiones del motor RE-280

3.3.4. Diseño electrónico

D2 tiene un circuito similar a Diego. Se trata del PIC 16F876 con un cristal a 4 MHz. Sin embargo, D2 tiene unos motores muy distintos a Diego. Se trata de

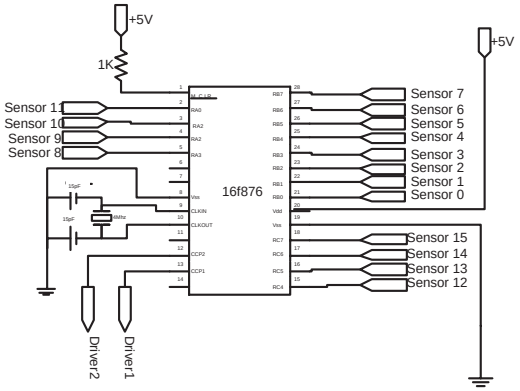
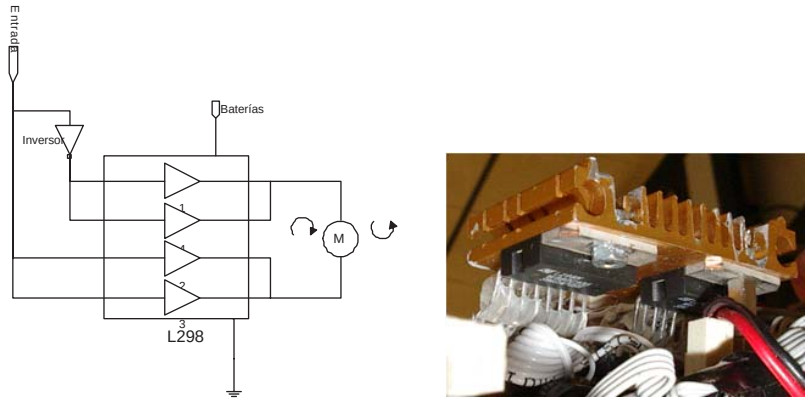


Figura 3.17: Esquema principal de D2

motores clásicos de corriente continua con reductora, como se vió en 3.3.3. Esto implica que se debe introducir obligatoriamente una etapa de potencia adecuada para manejarlos. Para ello se han elegido los *driver* de potencia L298 [ST] que proporcionan 2 A por cada uno de sus cuatro canales, permitiendo picos de hasta 3 A (figura 3.18(a)). En principio se calculó que para el control de un motor era suficiente con dos canales, por lo que con un solo L298 debería ser suficiente. Sin embargo en la práctica, se observa que cada motor tiene un pico de consumo que puede alcanzar los 4.7 A (figura 3.14), por lo que finalmente se ha utilizado un L298 por cada motor, empleando los cuatro canales disponibles



(a) (b)

Figura 3.18: Esquema e imagen del montaje de los driver de potencia en D2

de cada uno. De esta forma se pueden manejar consumos de hasta 4 A continuos y 6 A de pico por motor y driver, entrando dentro de los límites de consumo explicados.

Otro aspecto importante en D2 ha sido la alimentación. Se están utilizando baterías de NiMh de tamaño AA, con una capacidad de 1800 mA. Si medimos el consumo en vacío de D2 da valores en torno a 2 A. En carga crece de forma muy importante, llegando a veces a medirse casi 5 A, límite máximo de descarga para este tipo de baterías. Sin duda alguna, unas baterías con mayor límite de descarga, potenciarían bastante a D2, ya que actualmente el cuello de botella se encuentra en este punto. Posiblemente las más adecuadas sean las de modelismo tipo Sub-C, con una capacidad de descarga muchísimo mayor que las de tamaño AA que nosotros utilizamos. Con este cambio las baterías podrían dar toda la potencia demandada por los motores (hasta 4,7 + 4,7 = 9,4 A), y no estar limitado a los 5 A de las actuales.

En la práctica la duración de las baterías es de unos 10-12 minutos, dependiendo del consumo medio del motor que está relacionado con los esfuerzos que deben realizar debido a las curvas del circuito.

El control de los motores se hace con una señal PWM generada por el PIC. Para cada motor el driver dispone de dos entradas con niveles TTL. Valores iguales en estas entradas hacen que el motor se pare. Valores distintos, hacen que el motor gire en un sentido o en otro. En la práctica lo que se hace es meter

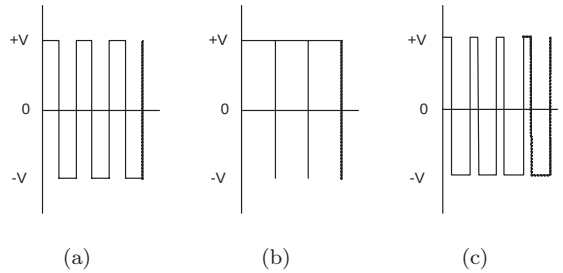


Figura 3.19: Ejemplos de ciclos PWM para conseguir un motor parado(a), adelante(b) y atras(c)

la señal PWM en una entrada y la negación de esta señal PWM en la otra. Así con un ciclo PWM del 50 % el motor está parado, con porcentajes mayores va en un sentido, y con porcentajes menores en el otro.

Este sistema tiene la ventaja de que incluso en situaciones de motor parado (con un ciclo PWM de 50 %) el motor se opone al movimiento, haciendo de freno. Sin embargo esto hace que haya un consumo incluso con el motor parado, ya que en este caso llegan valores de voltaje opuestos al motor. La figura 3.19(a), podemos observar los voltajes que le llegan al motor para que esté parado. En la figura 3.19(b) se ve un ejemplo de ciclo PWM para que el motor avance. En la figura 3.19(c) vemos otro ejemplo de ciclo PWM para el funcionamiento del motor en el otro sentido.

Para obtener la medida de la velocidad de los motores, se ha utilizado por cada caja reductora un sensor de corte H21A1. Estos sensores emiten un haz de luz infrarroja dirigida a un fototransistor, obteniendose a su salida 0 o 1 lógicos si la luz es interrumpida o no. En nuestro caso el elemento que interrumpe el haz son los dientes del engranaje reductor de la primera relacción.

Por lo tanto como el motor a 1.5 m/s va a 115 rps, y como cada revolución del motor tiene 12 dientes, tenemos que en un segundo pasarán delante de los sensores de corte $115 \times 12 = 1380$ dientes. Eso equivale exactamente a un diente cada 724 μs para la velocidad máxima controlada. Este periodo será mayor cuanto menor sea la velocidad y viceversa.

Debido a que la precisión máxima conseguida en la medición del periodo entre dientes ha sido aproximadamente de 30 μs, no se pueden medir los 724 μs que corresponden a 1.5 m/s, por lo que se redondea a por debajo a 720 μs para

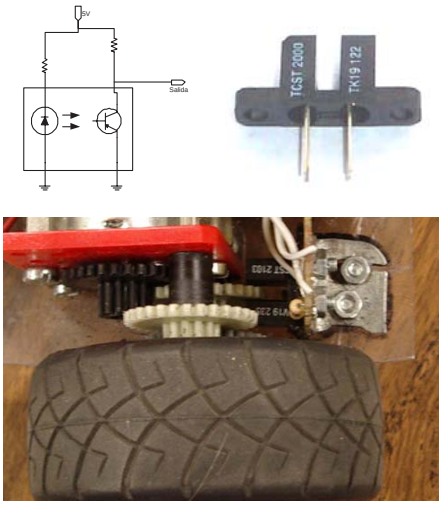


Figura 3.20: Esquema y aspecto de un sensor de corte y detalle de un motor con su sensor de velocidad

la velocidad máxima. Esto equivale a 24 bucles de 30 μs como se dijo. No se ha redondeado por arriba a 750 μs debido a que equivale a bajar la velocidad de 1.5 m/s.

Nuestra velocidad máxima controlada será de 1.5 m/s. Sin embargo, como se explicó en la sección 3.3.3, estos motores podrían proporcionar una velocidad de hasta 2 m/s. Esto equivale a medir un periodo de 543 μs (18 ciclos de 30 μs). Debe también observarse como a estas velocidades el error de cuantificación es muy grande, pues entre 2 m/s y 1.5 m/s, mediremos 18 o 24 ciclos de 30 μs respectivamente, por lo que la precisión en la medida de la velocidad no es excesivamente buena.

Si utilizasemos el PIC a 20 MHz, nuestra precisión mejoraría a 6 μs, por lo que nos permitiría controlar velocidades mucho mayores.

3.3.4.1. Sensores

El sistema de sensores de D2, aunque utiliza los mismos sensores de infrarrojos CNY70, se ideó de forma que no fuese tan crítico como Diego en cuanto a sus necesidades de detección de blanco y negro. De esta forma es posible hacer las pistas directamente en el suelo, sin que sea el fondo necesariamente blanco.

Esto favorece la creación de grandes pistas en pasillos o salas sin necesidad de utilizar tableros o cartulinas, solamente con cinta aislante negra.

Los sensores se han agrupado en placas de cuatro. Este diseño ha permitido probar diversas configuraciones. La disposición final ha sido en línea (ver figura

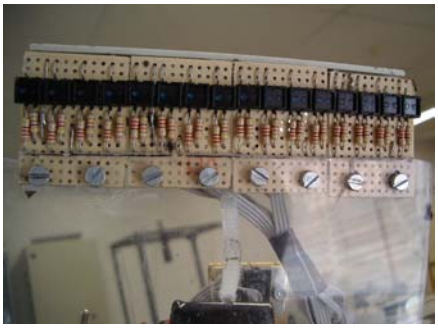


Figura 3.21: Imagen de los sensores en línea de D2

3.21), posiblemente una de las más adecuadas para la prueba de velocistas, aunque puede que no sea la mejor para rastreadores. Dejamos por lo tanto, como no podía ser de otra forma, una labor mayor al software para corregir este pequeño problema.

3.3.4.2. El microcontrolador PIC

Al igual que en Diego, en este montaje se ha utilizado el PIC 16F876 de MICROCHIP. En el caso de Diego se disponía del tiempo de CPU necesario para realizar el control del robot.

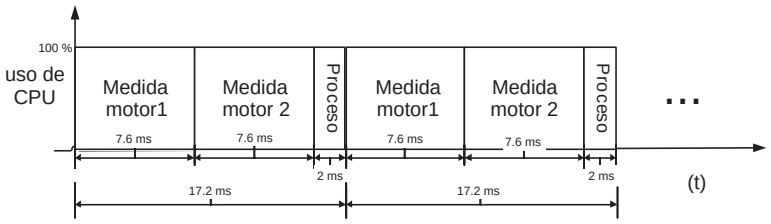


Figura 3.22: Diagrama de tiempos para el peor caso en un ciclo de control

Sin embargo con D2 el control de velocidad supone un consumo de tiempo de CPU mayor, ya que las lecturas de los sensores de corte se realizan mediante

una espera ocupada. Hay una gran cantidad de tiempo muerto esperando a que haya una transición 1-0-1 o 0-1-0 en los sensores de corte para saber la velocidad de desplazamiento. Como ya se dijo, para medir la velocidad, se mide realmente el periodo de tiempo transcurrido en pasar dos dientes de los engranajes. A velocidad máxima esto era 720 μs por cada motor, que sumados al tiempo de proceso de la entrada, daba un resultado de un control cada 3.5 ms, suficiente para evitar oscilaciones y reacciones extrañas.

Sin embargo en velocidades lentas, los dientes tardarán mucho más en pasar, por lo que el periodo de la medida será mayor. El periodo máximo medible es de 7.6 ms. Esto es así debido a que disponemos de un contador tipo byte, y cada unidad representa una precisión de 30 μs.

No es posible medir más de 7.6 ms de periodo, por lo que a velocidades mayores se fija siempre este periodo. En la figura 3.22 se pueden ver los tiempos para un ciclo de control en el peor de los casos. En total cada control tarda 17.2 ms en el peor de los casos.

Con D2 se han exprimido al máximo el MIPS (millón de instrucciones por segundo) que el PIC es capaz de procesar a 4 MHz. Además se ha hecho uso de 16 pines para sensores, 2 para el control por PWM de los motores y 2 más para el control de velocidad, quedando solamente dos pines libres de los 22 que tiene disponibles.

En la figura 3.23 se muestra el nivel de integración de la circuitería de D2.

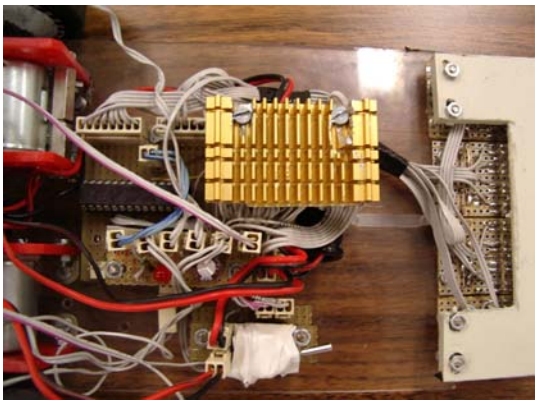


Figura 3.23: Imagen de la circuitería de D2

3.4. Comparación entre Diego y D2

Diego ha sido diseñado para competir en la prueba de rastreadores. Tiene una capacidad muy grande de giro aunque no tiene una velocidad demasiado alta. D2 se ha diseñado especialmente para la prueba de velocistas, aunque sin descartar en ningún momento la de rastreadores. Tiene una velocidad bastante más alta que Diego, aunque su capacidad de giro es menor. Esta limitación se ha solucionado por software de forma bastante satisfactoria como se comentará posteriormente en esta memoria.

Diego y D2 disponen de un sistema de tracción con dos motores opuestos. En Diego se trata de servomotores, los cuales son soluciones de motor, engranajes y *driver* de potencia integradas. En D2 se trata de un motor de corriente contínua acoplado a un conjunto de engranajes. Por lo tanto es una solución más flexible, pero menos robusta. Los *driver* de potencia son exteriores, y su gran ventaja es que admiten un amplio rango de voltajes de alimentación, permitiendo variar la velocidad final de forma sencilla.

D2 dispone de control de velocidad, al contrario que Diego. El control de velocidad es útil para mantener velocidades independientemente de la situación del robot en curvas, rectas e incluso subiendo y bajando rampas. Se vió la necesidad de añadir un control de velocidad debido a que D2 se aceleraba mucho en las rectas, por lo que al entrar en las curvas se llegaba a desestabilizar bastante. Diego no tiene control de velocidad. Sin embargo debido a la gran fuerza de sus servos, podemos estar seguros que si el robot debe ir a una velocidad los servos la alcanzan y mantienen con facilidad, con un tiempo de respuesta muy pequeño al contrario del que tenía D2 antes de instalarle el control de velocidad.

El sistema sensorial se basa en el mismo tipo de sensor, el CNY70, aunque con valores distintos de resistencias, lo cual hace posible que D2 pueda utilizar el suelo, sin ser blanco totalmente, al contrario que Diego, que necesita que el fondo sea muy blanco. Esto es muy útil para poder hacer las pistas directamente en el suelo con cinta aislante negra, de esta forma no hay dependencia de los tableros o cartulinas blancas para hacer el fondo de pista.

Como ya se comentó, Diego tiene 8 sensores y D2 16. Un mayor número de sensores permite una suavidad y precisión mayor mucho mayor.

La electrónica de los dos robots es similar en cuanto al PIC. Sin embargo en

D2 al tener los drivers de control de motores externos y control de velocidad la placa es bastante más grande.

Diego está alimentado con 4 pilas tamaño C, con lo que funciona a 6 V. Sin embargo D2 funciona con 6 baterías recargables tamaño AA, por lo que funciona a 7,2 V. D2 en principio no plantea problemas a la hora de añadir más baterías. Sin embargo con Diego puede haber problemas en el control de marcas de bifurcación ya que las temporizaciones se hacen según su velocidad, dependiente lógicamente del voltaje. Para corregir este problema habría que reprogramar el PIC a la nueva velocidad.

Como conclusión final cabe resaltar que por desgracia el tiempo dedicado al desarrollo del hardware es siempre mayor al del software, al contrario de lo que se podría pensar en robots de estas características.

4

Descripción de comportamiento inteligente

En este capítulo se va a realizar una revisión de todos los aspectos de comportamiento inteligente que han sido implementados. Se detallarán las distintas etapas por las que se ha pasado, desde su exposición y planteamiento, hasta llegar a las soluciones finalmente adoptadas.

4.1. Aspectos software del agente

En esta sección se va proponer una aproximación a la solución, basándonos en un agente reflejo de estado interno. También se tendrán en cuenta los conocimientos explicados en la sección 4.4.1.

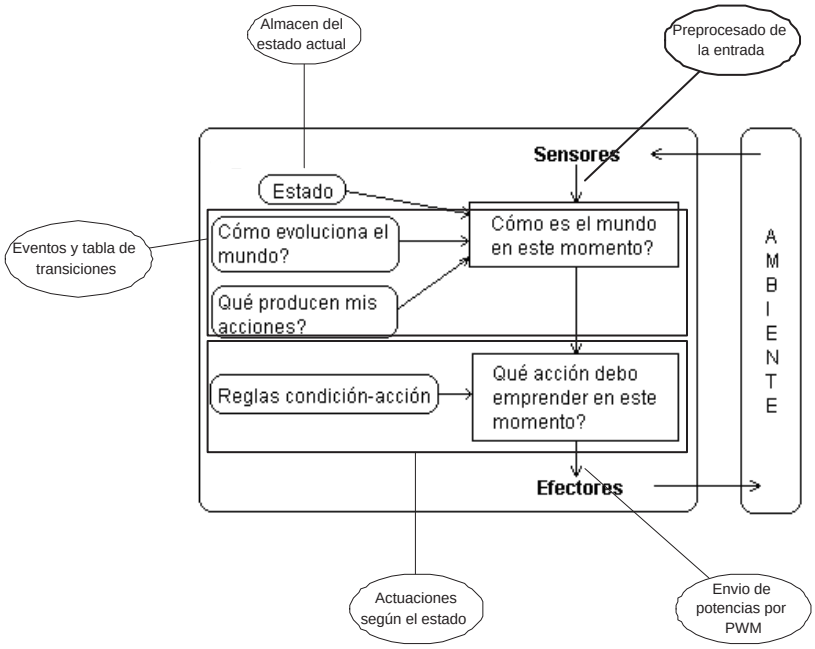


Figura 4.1: Aspectos software del agente con estado interno

En la figura 4.1, se pueden observar la equivalencia entre las partes del agente y la orientación a una solución práctica.

Según el agente propuesto se plantea la cuestión ¿cómo es el mundo en ese momento? Esta cuestión es alimentada directamente con la entrada. Sin embargo la entrada al agente se realiza en bruto. La entrada debe ser procesada para trabajar a un nivel superior. Por lo tanto el programa debe ser capaz de separar de forma correcta e inequívoca las distintas situaciones: línea, hueco o blanco.

Las cuestiones ¿cómo evoluciona el mundo? y ¿qué producen mis acciones?, se pueden reflejar como una serie de estados en los que el robot toma uno u otro comportamiento. Tambien se deben reflejar las transiciones entre los distintos estados, es decir, las condiciones que llevan a pasar de un estado a otro. De forma implícita deducimos que también debe existir un almacén del estado actual de funcionamiento.

La representación interna del ambiente debe provocar una reacción de nuestro robot-agente. Para conseguir esto, deberá realizar unas u otras reacciones

mecánicas, dependiendo del estado actual de funcionamiento. Normalmente a cada estado individual se le asociará una serie de reglas, que en función de la entrada provocarán una actuación del robot. Estas reglas se materializarán con el uso de controladores borrosos. Estos controladores permiten una abstracción de la entrada consiguiendo una suavidad importantísima para el correcto seguimiento de la pista.

En las siguientes secciones se van a describir individualmente las tres áreas de la implementación software de los robots por medio de agentes: Tratamiento de la entrada, la máquina de estados, y la lógica fuzzy.

4.2. Tratamiento de la entrada

A partir de las lecturas de los sensores, el tratamiento de la entrada debe permitir conocer la dirección de avance del robot independientemente del ancho de la línea. También debe permitir identificar si el robot se encuentra o no ante una marca de bifurcación y el filtrado de activaciones esporádicas de los sensores por imperfecciones de la superficie (ver sección 5.3) .

Es deseable que este sistema sea continuo, es decir, que para posiciones reales próximas, tengamos una valor de lectura ya tratada también próximo.

Hay que tener en cuenta que a priori no se conoce cuántos sensores van a estar activados, viendo la pista, y cuántos no, viendo el fondo. Por lo tanto hay que lograr un aislamiento de esta característica. En la figura 4.2 se muestra como puede variar la percepción según el ancho de la pista. En los tres casos mostrados el robot debería girar aproximadamente igual para desplazarse en la misma dirección de avance.

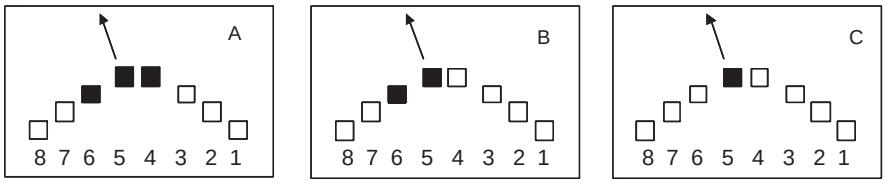


Figura 4.2: Ejemplos de percepciones para distintos anchos de pista

Una solución para obtener la dirección de avance sería hacer la media arit-

mética de los sensores activos. Por ejemplo, en el la figura 4.2(A), están activos los sensores 6, 5 y 4, la media aritmética será $\frac{6+5+4}{3} = 5$. En el caso B será $\frac{6+5}{2} = 5{,}5$ (redondeamos a 5), y en el caso C será 5. Esta solución nos permite tener una lectura ya procesada similar para los tres casos planteados. Esta solución también será válida para los casos que haya anchos mucho mayores con más sensores activos.

Conviene trabajar de forma similar en todas las lecturas. En el ejemplo vimos como se debía dividir entre el número de sensores activos (3 en el caso A, 2 en el B y 1 en el C). Para evitar este problema se propone manejar solo los bits activos de los extremos, por lo que en cada lectura trabajaremos con el primer sensor puesto a uno por la derecha y el primero puesto a uno por la izquierda. Siguiendo con el ejemplo de la figura 4.2, tendremos que el caso A serán el 6 y el 4, en el B el 6 y 5 y en el caso C el 5 solamente, el cual podemos considerar que es el mismo por la derecha que por la izquierda. De esta forma si sumamos estos dos bits en cada caso tendremos un valor indicativo del lugar donde tenemos la marca. Se trabaja siempre con un número de elementos constante (dos), se evita dividir y se trabaja siempre con valores enteros, evitando descartar decimales que pueden ser significativos.

Con esta solución, con un sistema de 8 sensores numerados del 1 al 8, tendremos lecturas una vez tratadas con valores entre 2 y 16. Con un sistema de 16 sensores tendríamos lecturas entre 2 y 32. Esta cuantificación de entrada tratada es mucho menor que intentar trabajar con todas las combinaciones posibles en 8 o 16 sensores (256 o 65535 casos), lo cual implicaría un uso de memoria mucho mayor y un código mucho más complejas. El tratamiento propuesto da una información de la posición con evolución lineal. En el caso de rastreadores,

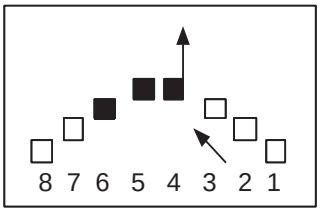


Figura 4.3: Ejemplo de tratamiento de la entrada para el control lateral

cuando se detecta una bifurcación, únicamente interesa procesar la entrada de los sensores correspondientes al camino correcto. En esta situación se obtiene el primer bit puesto a 1 por la izquierda o derecha, según corresponda. Para mantener una cuantización similar como se explicado previamente, se multiplica el valor de la posición del bit por 2. Se ignora la lectura por el lado opuesto a la marca. En la figura (figura 4.3) se puede ver un ejemplo de la salida para esta situación: el resultado sería tomar dos veces el valor del primer bit a uno por la derecha $4 + 4 = 8$.

Este sistema permite tratar solamente el camino correcto e ignorar el camino que no interesa. En la figura 4.4 se puede ver como funcionará el sistema para superar una bifurcación.

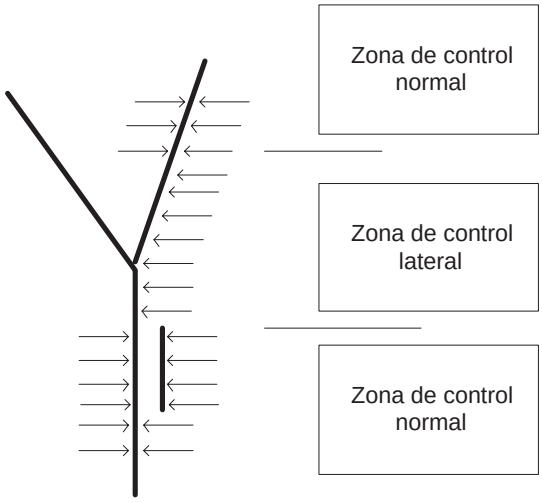


Figura 4.4: Ejemplo de utiilzación del control lateral

Como conclusión indicar que la solución que proponemos proporciona al algoritmo una medida de la posición del robot respecto a la línea, con una cantidad de valores relativamente pequeño y con independencia de la anchura de la línea.

En la prueba de rastreadores existe otro problema que hay que solucionar: Se trata de detectar si el robot se encuentra o no sobre una marca de bifurcación, ya que nuestro planteamiento ignora qué sensores están activos entre el primer sensor puesto a uno por la izquierda y el primero por la derecha. Para ello

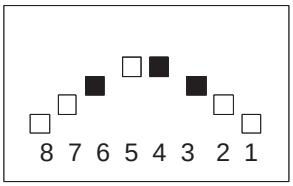


Figura 4.5: Ejemplo de detección de una marca de bifurcación

proponemos consultar si en las posiciones intermedias existe algún sensor puesto a cero. Si se cumple esta condición durante un tiempo mínimo se puede asegurar que el robot está sobre una marca (figura 4.5).

Otro aspecto muy importante es el filtrado de ruidos producidos por lecturas esporádicas ó erróneas de los sensores debido a suciedad, mal estado de la pista o reflejos. Este tipo de ruidos puede producir, que el robot gire de forma brusca y pierda la pista. Esta situación se presenta cuando el robot oscila y sus sensores se encuentran sobre la superficie blanca del fondo de la pista, la cual tiene imperfecciones que pueden llevar a un giro brusco y a perder definitivamente la línea. Los detalles de este filtrado se pueden estudiar en la sección 5.3.

Como conclusión cabe destacar que es muy importante tener una vista abstracta de alto nivel de la entrada y no tratar de trabajar con la percepción en bruto, ya que habría un número de casos muy grande y se trabajaría con mucha dificultad.

4.3. Máquina de estados

Como ya se vió en la sección 4.1, se utilizará un agente reflejo de estado interno. Este tipo de agentes disponen de una memoria que almacena el estado actual. También dispone implícitamente de la lista de condiciones que se deben cumplir para que el agente pase de un estado a otro y modifique su comportamiento.

Una máquina de estados precisamente es un formalismo para representar estados, transiciones, eventos y acciones. Es una solución fácilmente implementable que justamente cumple con las necesidades de utilización de los estados del agente.

La máquina de estados representa los distintos tipos de comportamientos que tiene el robot, según los eventos ocurridos en su mundo (la pista) para seguir un camino de forma correcta. En general una máquina de estados se compone de estados, eventos y acciones.

- Un estado representa una situación
- Un evento es un suceso que plantea el cambio de un estado a otro
- Una acción es un tipo de comportamiento

Para ser una máquina correcta debe cumplir las condiciones siguientes:

- Ser Finito. Debe de tener un número de estados representable
- Determinista. Debe cumplir las características de no tener estados ambíguos, y ser completo, es decir, que represente todos los estados o situaciones posibles

Una máquina de estados tiene bien definidos los eventos que reconoce y los que descarta, de forma que se construye basándose en un alfabeto. También debe incluir estado inicial o de arranque y algún estado final. Estos pueden coincidir. Si cumple todas estas características se conoce como Autómata Finito Determinista (AFD). Si no se definen estados finales se conocen como semiautómatas finitos. Formalmente un AFD es una estructura de la forma $AFD = (Q, Ent, T_R, q_i, F)$, donde:

- Q Conjunto de estados
- Ent Alfabeto de entrada
- T_R Función de transición del tipo $Q \times Ent \rightarrow Q$
- $q_i \in Q$ Estado inicial
- $F \subset Q$ Estados finales

En el caso que nos ocupa, se han implementado varias máquinas de estados según las pruebas y el robot al que van dirigidas.

4.4. Implementación del conocimiento con lógica fuzzy

La lógica fuzzy es una solución muy adecuada para este tipo de sistemas, ya que se consigue una suavidad de funcionamiento. Además se han realizado pruebas previas y simulaciones, de forma que se concluyó que resultaba la solución más interesante.

La forma de tratamiento de la entrada reduce considerablemente el número de casos a tratar. Así se plantea, como una solución idónea, tener precalculados todos los valores de salida en función de la entrada, sin tener la necesidad de trabajar con lógica borrosa a nivel de PIC. De esta forma evitamos realizar para cada ciclo de control las etapas de *fuzzificacion* y *defuzzificacion*, que por otro lado serían bastante costosas para el μprocesador utilizado.

4.4.1. Base de conocimiento

Diego y D2 están diseñados para pruebas distintas, como son rastreadores y velocistas. En ambos casos se plantea la necesidad de realizar un seguimiento de una línea negra sobre un fondo blanco, siendo deseable rapidez, suavidad y seguridad. Para ello como es obvio contamos con un número de sensores que nos indican si debajo de ellos está la línea o está el fondo de la pista.

De forma experimental, se ha constatado que uno de los sistemas móviles más sencillos de controlar son los sistemas con giros diferenciales. Estos sistemas parecen ser muy adecuados para utilizarse con un sistema sensorial como el planteado. Además la cinemática de ambos robots es muy sencilla haciendo reaccionar al robot de una forma directa sobre el eje deseado. Estos aspectos se vieron en detalle en el capítulo 3.

Para realizar un seguimiento adecuado de una línea, hay que tratar que el robot esté siempre en el centro. Esto significa que sus sensores estén siempre centrados respecto a la línea. Cuanto más se desvie el robot de esa posición ideal central, más debe girar para corregir la situación. Se pueden deducir rápidamente reglas para cada situación partiendo de esta premisa. Básicamente pensaremos que si la línea está hacia la izquierda del centro del robot, debe girar hacia ese

lado para intentar centrarlo y viceversa (figura 4.6). Además el par aplicado a los motores debe ser mayor cuanto más ladeado se encuentre.

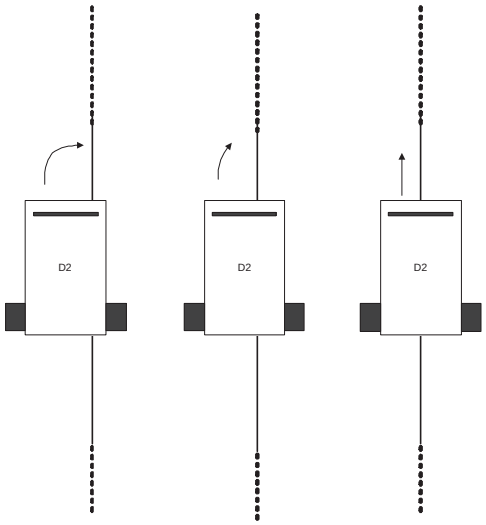


Figura 4.6: Ejemplos de reacciones según la posición respecto a la línea

Esta es una de las soluciones sencillas para resolver problemas de este tipo. Sin embargo no tenemos en cuenta la evolución respecto al tiempo de la desviación o del efecto de la correción. Esto no ha sido necesario ya que podemos considerar que la situación perfecta no existe y que el robot tendrá siempre una pequeña oscilación que consideramos despreciable o al menos asumible. No se trata de realizar un seguimiento de precisión, sino de encontrar una solución satisfactoria. Una vez planteado el problema del seguimiento de la línea, igual para rastreadores y velocistas, surge la necesidad de detectar bifurcaciones. Esto solamente se aplica al caso de los robot rastreadores.

En este caso existe una marca que indicará la condición excepcional que hará decidir para tomar la bifurcación correctamente. Para una persona resulta sumamente sencillo realizar esta tarea, detectar la marca y saber que giro debe realizar, al igual que si fuesemos conduciendo un coche. Lo primero que haremos será observar que se trata de una marca, y no de una mancha en la pista, por ejemplo. A continuación veremos en que lado de la pista está, derecha o izquierda. Por último tomaremos el camino correcto. Observamos como sin

darnos cuenta hemos diferenciado por etapas el reconocimiento y la decisión de tomar una bifurcación.

4.4.2. μrobot Diego

En esta sección se describirán las bases del diseño software de Diego, así como los detalles de su implementación con lógica fuzzy y su máquina de estados.

4.4.2.1. Planteamiento

El conocimiento utilizado para finalmente llegar a la implementación, se ha realizado partiendo del lenguaje natural. Se ha planteado la cuestión de cómo un humano realizaría el control del robot. Así se han descrito las reglas que se utilizarán para controlar a Diego.

En general partimos de plantear si el robot no está centrado respecto a la pista, deberá modificar su dirección sin dejar de avanzar, de forma que se centre. Además si la pista es muy sinuosa es posible que haya que torcer pero frenando.

De este comportamiento previo obtenemos que si el robot es capaz de reconocer que no está centradao respecto a la pista y el nivel de dificultad de la pista se podrá actuar en consecuencia.

Si extendemos el aparentemente simple conocimiento anterior, se tendrá que el robot puede estar ladeado hacia la derecha o a la izquierda, o bien estar centrado. Para la dificultad definiremos tres niveles fácil, medio y difícil.

Por lo tanto, ordenando el conocimiento podemos incorporar todas las reglas del tipo:

- *Si el robot está ladeado hacia la izquierda de la pista y el entorno es fácil entornces girar un pequeño ángulo a izquierda.*
- *Si el robot está centrado y el entorno es fácil entonces avanzar en la misma dirección.*
- *Etc.*

En cuanto a las actuaciones diferenciamos entre avanzar, girar y frenar. Esto equivale a cuatro niveles de actuación: girar un poco, girar mucho, ir recto y frenar. Como los dos motores están en disposición diferencial, del concepto

girar extraemos las potencias a aplicar a los motores, clasificándolas en parado, adelante poco, adelante a tope, atrás poco y atrás a tope. De esta forma combinándolas obtendremos diferentes tipos de giros y velocidades.

4.4.2.2. Etiquetas y conjuntos borrosos

Como ya se analizó en la sección 4.2, para la variable de lectura de dirección son posibles valores entre 2 y 16. En el caso de la dificultad, se han barajado valores posibles entre 0 y 1. La idea era disponer de un mecanismo que evaluase la dificultad de la pista cada cierto tiempo y devolviese un valor entre 0 y 1 adecuado como variable de entrada. Por lo tanto en esta prueba se utilizan dos variables de entrada:

- La lectura de los sensores ya procesada.
- Una valoración de la dificultad de la pista

Existen dos variables de salida, una para cada motor. Los valores posibles en la variable de salida están comprendidos entre 85 y 101, siendo el 92 parado. El aspecto y forma de las distintas etiquetas linguísticas de los conjuntos borrosos de Diego, se muestra en la figura 4.7. El conjunto borroso con las etiquetas

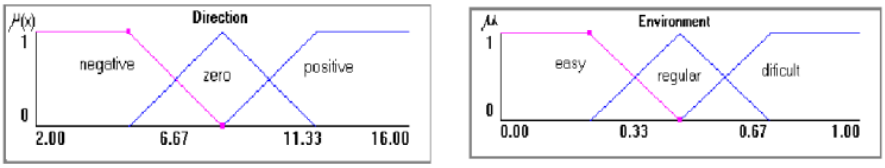


Figura 4.7: Conjuntos borrosos para las dos variables de entrada de Diego

linguísticas correspondientes para la salida de ambos motores se puede ver el la figura 4.8

4.4.2.3. Reglas borrosas

En la implementación definitiva se han incluido las reglas borrosas (ver fig 4.9) solamente para el tipo de pista difícil, ya que para implementar todos los niveles de dificultad se obtendría una superfície de potencias bastante grande y difícilmente representable y manejable. Sin embargo se han probado y simulado

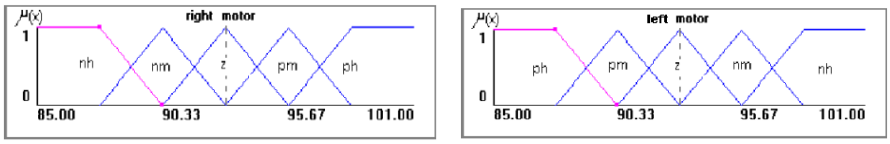


Figura 4.8: Conjuntos borrosos y etiquetas para las variables de salida de Diego

individualmente todas las posibilidades. Como se puede observar, las reglas son

motor izquierdo.

Entorno	Dirección		
	Negativa	Cero	Positiva
Fácil	ph	ph	z
Regular	ph	ph	nm
Difícil	ph	ph	nh

motor derecho

Entorno	Dirección		
	Negativa	Cero	Positiva
Fácil	z	ph	ph
Regular	nm	ph	ph
Difícil	nh	ph	ph

ph=positivo alto, pm=positivo medio, z=cero, nm=negativo medio, nh=negativo alto

Figura 4.9: Reglas de Diego

sencillas e intuitivas, se suavizan cuanto más fácil se considera el entorno. Como es lógico, las reglas son simétricas para ambos motores, ya que se trata de dos motores en disposición opuesta y hay que realizar inversión de signo para que el robot avance. Tambien se puede observar como siempre hay un motor que está girando a toda velocidad, con un valor *ph* (positivo alto). Esto significa que el robot va a su máxima velocidad mecánica en todo momento. Es una situación límite que ha probado que incluso en estos casos la lógica borrosa proporciona buenos resultados. Podemos afirmar que las reglas aplicadas explotan prácticamente todo el rendimiento mecánico de Diego.

4.4.2.4. Tablas de potencias

Como ya se comentó, en la implementeación del software que se ejecuta en el PIC no se codifican las reglas borrosas, sino una tabla con valores precalculados. Estas tablas definen cuáles son las potencias a aplicar a cada motor en función de la entrada. Se trata de un sistema totalmente reactivo, aunque para su obtención, como ya se vió en 4.4.2.2, se ha utilizado lógica borrosa.

El PIC tiene la capacidad de generar una señal PWM. Desde el punto de vista de la programación esta señal no es más que la escritura en un puerto de

un byte (valores entre 0 y 255), donde el 0 representa un ciclo PWM del 0 % y el 255 representa el 100 % del ciclo. Se definen dos tablas de potencia, una para cada motor. Estas tablas contienen los valores que se aplican directamente a la salida PWM del PIC para obtener distintas velocidades. En el caso de Diego, la potencia deseada es bastante parecida a la obtenida, sin necesidad de tener que medir la velocidad a la que giran los motores. Esto es debido a que los servos tienen un par de fuerza suficiente como para que en un tiempo despreciable, los motores alcancen la velocidad indicada.

Recordando lo explicado en 3.3.3, vemos que los servos aceptan una señal PWM de entrada. La modificación de un servo conlleva dejar fija su referencia de posición, por lo que los valores de control PWM iguales a esa referencia hacen que el motor esté parado; valores cercanos hacen que el motor vaya despacio y valores lejanos hacen que vaya más rápido.

Después de modificar los servos, se encontró experimentalmente que el valor de parado para ambos motores era el 92 sobre 255. También se encontró que la velocidad máxima en un sentido estaba en el 85 sobre 255 y en 101 sobre 255 en el otro sentido. Por lo tanto nuestros servos permitían 8 niveles de velocidad adelante y 8 niveles de velocidad atrás, más que suficiente para recorrer la pista de rastreadores sin problemas.

Como los motores tienen una disposición opuesta para permitir el giro diferencial, hay que hacer una corrección de signo al aplicar las potencias. De esta forma para que el robot vaya adelante a la máxima velocidad se debe aplicar en el registro PWM del motor 1 el valor 85 y 101 en el registro PWM para el motor 2.

En la figura 4.10 se observan las tres tablas barajadas, para los casos de dificultad de pista fácil, medio y difícil. La última de ellas, adecuada para el entorno difícil, incluye los valores finales a aplicar a los registros PWM de cada motor en función de las entradas. Estos valores son los que finalmente se incluirán en una matriz con elementos de 1 byte, para posteriormente introducirla en el código. Cabe resaltar que en todas las tablas siempre un motor se encuentra funcionando a la máxima velocidad, por lo que el rendimiento final es muy bueno. Sin embargo, como se puede comprobar, estas tres gráficas no son más que

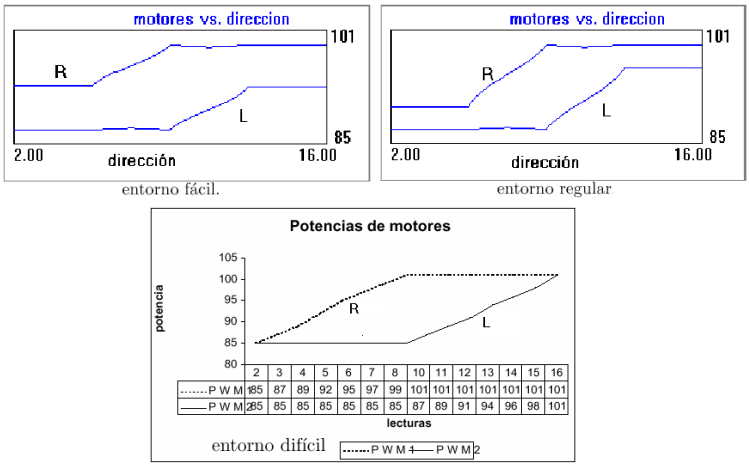


Figura 4.10: Tablas de potencias estudiadas para Diego

cortes de la superficie de potencias para todos los casos la lectura de la pista, con valores constantes para la dificultad (0, 0.5 y 1 respectivamente).

4.4.2.5. Máquina de estados

Diego está diseñado para la prueba de rastreadores. En esta prueba existen varias dificultades. La primera y más evidente es ser capaz de seguir una línea. Otra dificultad es la de ser capaz de detectar una bifurcación y tomar el camino correcto. Tambíen se ha contemplado las salidas de la pista. Por lo tanto hemos planteado cuatro estados posibles en Diego:

- 1. Seguir línea
- 2. Detectar marca
- 3. Tomar bifurcación
- 4. Fuera de pista

El estado inicial o de arranque será el de seguir línea (1). La percepción que Diego tiene del mundo es a través de 8 sensores de infrarrojos que finalmente se traduce en 8 valores binarios. Por lo tanto su mundo solamente tiene 256 valores posibles. Lógicamente no podemos trabajar a este nivel tan bajo, sino que debemos conceptualizar lo que está viendo. De esta forma definimos funciones

que con la lectura de esos 8 bits indiquen si se encuentra sobre una línea o estamos sobre una marca. La forma de detectar estas situaciones se explicó en la sección 4.2.

En el momento que se ha decidido el lado de la bifurcación a seguir, se inicia un temporizador con un tiempo suficiente para asegurar que ya ha pasado (ver figura 4.11). Según la velocidad de Diego se ha calculado este tiempo en 1 s; Para D2 ha sido de 0.5 s. En este tiempo recorrerán 0.4 m, suficiente para sobrepasar la bifurcación.

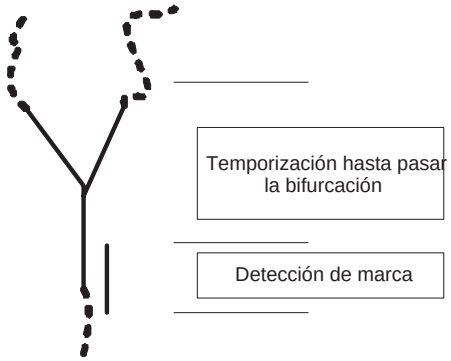


Figura 4.11: Detección de marca y temporización

Si resumimos los eventos comentados tendremos:

- A Línea normal
- B Hueco esporádico
- C Hueco seguro
- D Fin de temporizador de bifurcación
- E Viendo blanco (salida de pista)

Por último debemos obtener las acciones a realizar en cada estado. En este caso se van a utilizar los tipos de control descritos en la sección 4.2, y que son:

- En el estado 1: Control normal
- En el estado 2: Control normal
- En el estado 3: Control lateral

- En el estado 4: Girar hacia el lado por el que salió

Finalmente queda pendiente definir las transiciones de estados en función de los eventos. En la tabla de la figura 4.12 se observan los nuevos estados en función del estado actual y del evento.

Evento\Estado	1	2	3	4
A	1	1	3	1
B	2	2	3	1
C	-	3	-	1
D	-	-	1	-
E	4	4	4	4

Figura 4.12: Tabla de transiciones y estados de Diego

En la figura 4.13 se muestra gráficamente la máquina de estados, donde se pueden ver perfectamente los estados y los eventos que deben transcurrir para cambiar a otro estado.

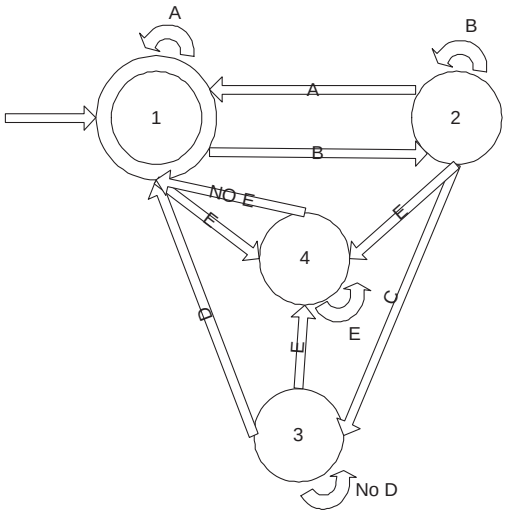


Figura 4.13: Máquina de estados de Diego

El estado 4 sucede cuando pierde la línea. En este caso el robot seguirá torciendo hacia el lado por el que salió hasta que la vuelva a detectar la pista. A este estado se entra desde cualquiera de los otros tres

Por último comentar que la máquina de estados se ha obtenido experimentalmente, a base de realizar multitud de pruebas. Se ha comprobado que esta

máquina de estados es un modelo muy robusto para la prueba de rastreadores, que sin duda sienta un precedente en D2 así como en futuros prototipos.

4.4.3. μrobot D2

A continuación se verán todos los detalles de implementación del conocimiento del robot D2, mostrando su planteamiento de alto nivel, los detalles del control borroso, y finalmente la tabla de potencias con los resultados.

4.4.3.1. Planteamiento de D2

Como se ha comentado previamente, Diego ha sentado un precedente en el sistema de control con lógica fuzzy para un comportamiento de rastreador. La ampliación de estas reglas para D2 ha consistido en diferenciar más niveles de posición respecto a la pista, ya que dispone de más sensores. Mientras en Diego se diferenciaba entre estar a la izquierda, a la derecha o centrado solamente, en D2 se han incluido etiquetas intermedias de forma que tendremos reglas del siguiente tipo:

- *Si estamos un poco a la derecha sobre la pista y el entorno es fácil, entonces giramos un poco a la derecha*
- *Si estamos muy a la derecha sobre la pista y el entorno es difícil, entonces giramos mucho a la derecha*
- etc.

Debido a las características eléctricas y mecánicas de D2, sus sistemas de actuación permiten muchos más valores de velocidades, se han distinguido más niveles de actuación.

4.4.3.2. Etiquetas y conjuntos borrosos

En la prueba de rastreadores para Diego y D2, resulta muy difícil en la práctica tener una evaluación continua de la dificultad de la pista, por lo que se han realizado pruebas con reglas similares a Diego para los tres tipos de entornos considerados: fácil, medio y difícil. El resultado es que en los entornos difíciles y medios el robot oscilaba significativamente, debido a que la velocidad

era bastante alta. En los entornos fáciles era bastante estable pero presentaba problemas en las curvas relacionados tambien con la velocidad. La conclusión inequívoca que se obtuvo es que se debía bajar la velocidad máxima del robot por medio de las reglas, ya que las de Diego no son válidas en D2.

Las etiquetas linguísticas se han ampliado para permitir mayor detalle y disponer de mayor capacidad de acelerar y frenar. De este modo, en la variable de entrada "lectura de sensores"se han incluido dos etiquetas más (figura 4.14). El conjunto "dificultad de la pista"se ha conservado exactamente igual que en Diego.

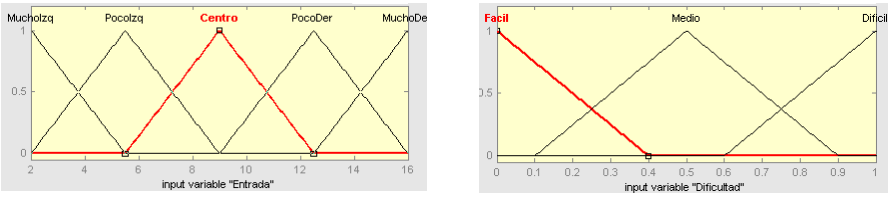


Figura 4.14: Conjuntos de entrada para D2

En las dos variables de salida, una por cada motor, también se han ampliado las etiquetas linguísticas respecto a Diego para permitir mayor detalle en la salida, como se puede observar en la figura 4.15.

En la prueba de velocistas con D2, las curvas no son tan cerradas como en rastreadores, por lo que después de probar varios niveles de dificultad, se optó igualmente por utilizar el nivel de dificultad fácil, pero la mayor velocidad posible. De este modo tendremos un robot muy estable siguiendo la línea y a una velocidad considerable.

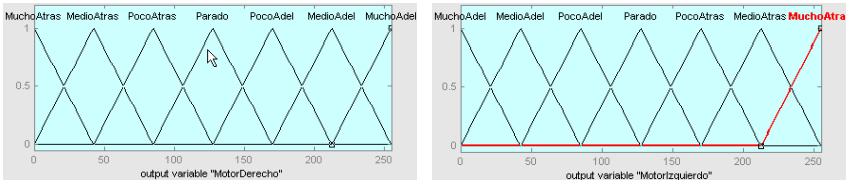


Figura 4.15: Conjuntos de salida para D2

4.4.3.3. Reglas borrosas

Como ya se comentó anteriormente, se han realizado reglas para velocistas y rastreadores por separado. Si bien el nivel de dificultad se ha mantenido en ambos casos a fácil, la velocidad máxima no ha podido ser la misma en ambas situaciones. Como se explicó antes una velocidad alta en rastreadores plantea problemas en las curvas que suelen ser muy cerradas. Puede observarse en la

Motor Derecho

Dificultad\Curva	Mucho Izq	Poco Izq	Centro	Poco Derecha	Muy Derecha
Fácil	PM	PM	PM	PM	Parado
Medio	PM	PM	PM	PP	NM
Dificil	PM	PM	PM	Parado	NG

Motor Izquierdo

Dificultad\Curva	Mucho Izq	Poco Izq	Centro	Poco Derecha	Muy Derecha
Fácil	Parado	PM	PM	PM	PM
Medio	NM	PP	PM	PM	PM
Dificil	NG	Parado	PM	PM	PM

PG=positivo grande, PM=positivo medio, PP=positivo pequeño
NG=negativo grande, NM=negativo medio, PP=negativo pequeño

Figura 4.16: Tabla de reglas para D2 en rastreadores

figura 4.16 como siempre hay al menos un motor que tiene una velocidad PM (positiva media), de forma que se establece una velocidad adecuada para este tipo de prueba.

Motor Derecho

Dificultad\Curva	Mucho Izq	Poco Izq	Centro	Poco Derecha	Muy Derecha
Fácil	PG	PG	PG	PM	Parado
Medio	PG	PG	PG	PP	NM
Dificil	PG	PG	PG	Parado	NG

Motor Izquierdo

Dificultad\Curva	Mucho Izq	Poco Izq	Centro	Poco Derecha	Muy Derecha
Fácil	Parado	PM	PG	PG	PG
Medio	NM	PP	PG	PG	PG
Dificil	NG	Parado	PG	PG	PG

PG=positivo grande, PM=positivo medio, PP=positivo pequeño
NG=negativo grande, NM=negativo medio, PP=negativo pequeño

Figura 4.17: Tabla de reglas para D2 en velocistas

En cambio en velocistas (figura 4.17) siempre existe al menos un motor con

una potencia PG (positiva alta). Esto es posible debido a que en esta prueba las curvas tienen un radio de giro mínimo relativamente cómodo para D2.

4.4.3.4. Tablas de potencias

La evolución de D2 ha pasado por dos etapas.

- En la primera etapa no había control de velocidad, por lo que la potencia a aplicar dependía de la lectura actual y de la anterior. El resultado era una superficie de potencias que aparece en la imagen 4.18.
- En la segunda etapa de D2, se utilizó el control de velocidad, por lo que la salida era función solamente de las lecturas de la línea, simplificando enormemente los requisitos de memoria.

La potencia en D2 se maneja de forma distinta que con Diego. En este caso, se utilizan *drivers* de potencia externos que posibilitan la capacidad de realizar un control más fino en cuanto a la potencia aplicada a cada uno de los motores. Concretamente se tiene que el valor 128 en el registro PWM provoca que el motor esté parado, un valor 0 hace que vaya adelante a tope en un sentido y 255 en el otro. Por lo tanto hay disponibles 127 niveles de potencia teóricos adelante y 127 atrás. Diego sólo disponía de 8 niveles en cada sentido.

El trabajo principal de D2 se centró en la prueba de velocistas. En la versión anterior no había control de velocidad, por lo que en las rectas el robot se aceleraba poco a poco, llegando a desestabilizarse al llegar a las curvas. En un intento de tratar de detectar esta situación de entrada en curva, se implementó un sistema para evaluar la salida en función de la lectura actual y una lectura anterior. El resultado, realizado con Matlab, daba superficies de potencia como la que se muestra en la figura 4.18. Puede verse como en esta superficie existen las variables de entrada *presente* y *pasado*. La variable *pasado* era la que se produjo en un lapso de tiempo anterior a la lectura actual.

Este tipo de superficies trataba de solucionar los problemas de la entrada en curva y las oscilaciones, consiguiendo de forma aceptable corregir estas situaciones aunque sacrificando bastante la velocidad. El resultado experimental era que el robot se frenaba bastante en la entrada a la curva. Por este motivo se planteó finalmente el uso de sistemas que permitiesen en todo momento medir

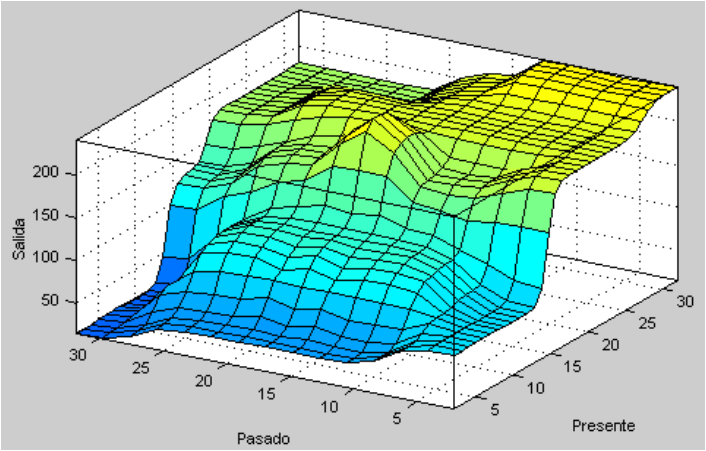


Figura 4.18: Tabla de potencias de la primera versión de D2

y controlar la velocidad real de giro de las ruedas. Este sistema permitió evitar las grandes tablas bidimensionales del método anterior, y obtener un resultado final mucho más rápido y seguro. Los mejores resultados se obtuvieron con reglas similares a las de Diego pero en un entorno fácil, resultando una tabla de potencias como la que aparece en la figura 4.19. Estas tablas son solo de un motor, ya que la del otro es simétrica respecto al eje x, y han sido obtenidas partiendo de la figura 4.20 tomando como constante (cero) el valor de la dificultad. Sin embargo, aunque en la figura 4.19 se indican directamente las potencias que finalmente han sido implementadas, podemos ver en 4.20 el aspecto de la superficie de potencias que ha sido generada para velocistas.

4.4.3.5. Máquina de estados

D2 es un robot pensado para participar en las pruebas de velocistas y de rastreadores. Las máquinas de estados son distintas. La de velocistas es muy sencilla, con solo dos estados (figuras 4.21 y 4.22).

- 1. Es su estado normal de fucionamiento, que es el de seguimiento de línea.
- 2. Pérdida de línea, en el que se trata de volver a encontrarla, torciendo hacia el lado por el que se perdió.

Los eventos son los siguientes:

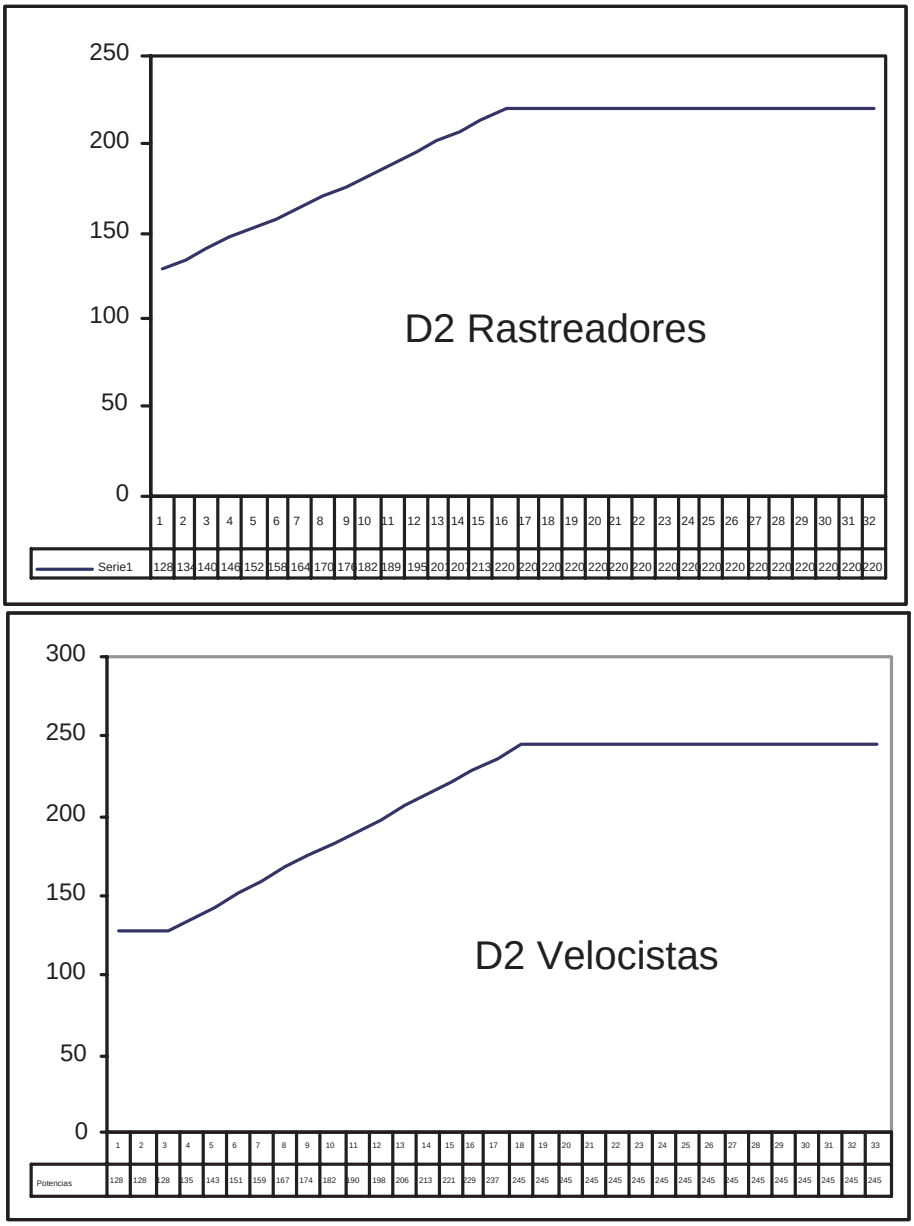


Figura 4.19: Tabla de potencias de un solo motor para rastreadores y velocistas

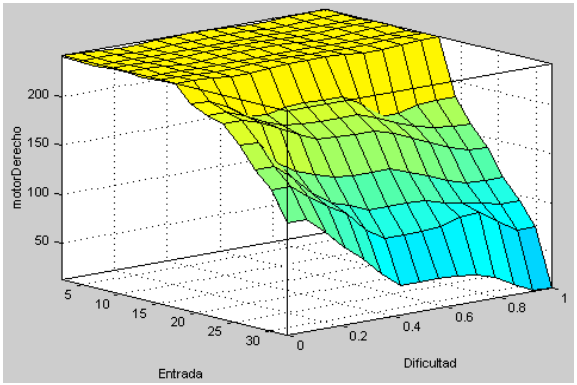


Figura 4.20: Superficie de potencias para D2 en velocistas

Evento\Estado	1	2
A	1	1
B	2	2

Figura 4.21: Tabla de transiciones y estados de D2 en velocistas

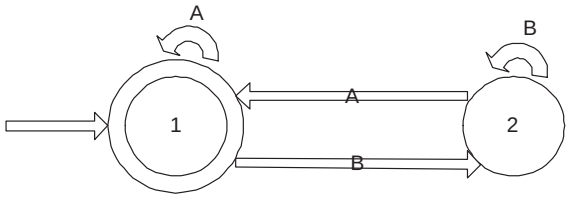


Figura 4.22: Máquina de estados para D2 en velocistas

- A. Viendo la línea
- B. Línea perdida

y por último sus acciones son:

- Seguir línea
- Torcer hacia donde se perdió la línea

Para la prueba de rastreadores inicialmente se utilizó la máquina de estados de Diego adaptada a las peculiaridades de D2. El resultado fue muy bueno ya desde el principio. Sin embargo, debido a que la disposición de los sensores de D2 es en línea, tiene más dificultades para tomar bifurcaciones raras y en ángulo recto, como las planteadas en la última edición del concurso. Lo normal era que se pasara de largo sin tomar el camino correcto. Estas bifurcaciones consisten en caminos en ángulos rectos como se observa en la figura 4.23. Sin embargo Diego no tenía ese problema debido a la disposición de los sensores en forma de V invertida, que hacía que el camino perpendicular fuera visto durante mucho más tiempo y se tomase correctamente. Para solucionar este problema

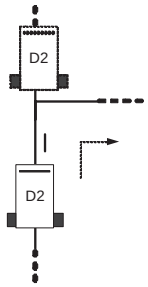


Figura 4.23: Bifurcaciones problemáticas para D2 con la máquina de estados de Diego

se optó por añadir un estado más que detectase esta situación, de forma que D2 finalmente fuese por el camino correcto. El nuevo estado compensa la situación de los sensores, torciendo rápidamente hacia el lado correcto, recuperando la visibilidad de la línea.

Sin embargo en esta versión de D2 no se han previsto situaciones como giros menores de 90 grados. Actualmente un ángulo menor de 90 grados podría ser

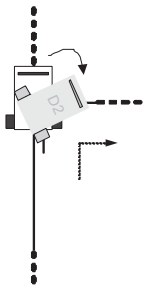


Figura 4.24: Efecto de añadir un nuevo estado a D2

confundido con una marca de bifurcacion. Esto se verá solucionado en una futura revisión del software de D2.

Todo esto añade más complejidad al software, ya que hay que manejar un nuevo evento que situe a D2 en el nuevo estado. Desde luego es complicado de detectar, ya que debe ser muy fiable para detectar estas situaciones. Este evento sucede cuando después de una bifurcación, se percibe un incremento muy grande en el ancho de la lectura, y posteriormente pequeño de nuevo. Esta situación se ve en la imagen 4.25, donde se aprecian las tres etapas de detección del evento. Resumiendo, los estados son los siguientes:

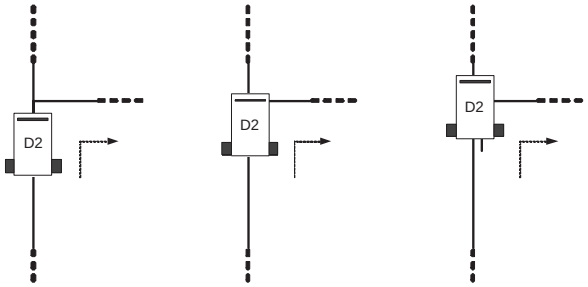


Figura 4.25: Etapas para detectar un ángulo recto después de una bifurcación

- 1 Seguimiento de línea
- 2 Detección de bifurcacion
- 3 Tomar bifurcación
- 4 Ángulo recto

- 5 Viendo blanco (pista perdida)

Las acciones que se realizan en cada uno de estos estados son:

- En el estado 1: Seguir línea con control normal
- En el estado 2: Seguir línea con control normal
- En el estado 3: Seguir línea con control lateral
- En el estado 4: Girar hasta encontrar línea perpendicular
- En el estado 5: Torcer hacia el lado por el que se salió

Los eventos quedan como se expone a continuación:

- A Línea normal
- B Hueco esporádico
- C Hueco seguro
- D Fin de temporizador de bifurcación
- E Detección de ángulo recto
- F Detección de blanco

La máquina de estados y su tabla de transiciones se muestran en la figura 4.26. Como se puede observar, la transición al estado 4 solo es posible desde el

Evento\Estado	1	2	3	4	5
A	1	1	3	1	1
B	2	2	3	-	1
C	-	3	-	-	1
D	-	-	1	-	1
E	-	-	4	-	-
F	5	5	5	5	5

Figura 4.26: Tabla de transiciones y estados para rastreadores con D2

estado 3. Sin embargo, en una futura revisión, se tendrá en cuenta la posibilidad de pasar al estado 4 desde el 1 debido a que este tipo de ángulos puede aparecer en cualquier parte del recorrido y no solamente después de una bifurcación. Un estado muy importante es el de perder la pista (5). A él se llegará en caso de

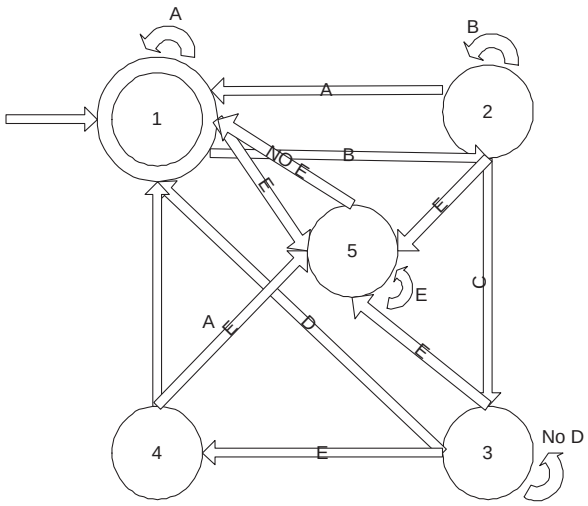


Figura 4.27: Máquina de estados de D2 para rastreadores

perder la pista desde cualquiera de los otros cuatro estados. Si esto ocurre en cualquier estado, seguiremos torciendo como lo estaba haciendo, de forma que en algún momento vuelva a encontrar la línea.

5

Programación del agente

5.1. Introducción

A continuación se van a describir los detalles técnicos más sobresalientes utilizados para la programación de los dos agentes estudiados, atendiendo al preprocesamiento de la entrada, máquina de estados y lógica fuzzy.

5.2. Restricciones de programación impuestas por el PIC

En la programación en C para estos PIC, hay que tener especial cuidado en cumplir una serie de reglas que a continuación se enumeran:

- Se disponen de 8 kbytes de memoria para código.

- Se dispone de 384 bytes de memoria RAM para uso de variables, aunque hay que tener en cuenta que el compilador asigna variables temporales, por lo que pueden ser menos.
- El tipo de variables utilizadas debe ser de tamaño byte solamente, con valores entre 0 y 255.
- Los parámetros de entrada y salida a funciones serán variables de tipo byte.
- Las matrices solamente pueden ser unidimensionales, con un tamaño máximo de 256 elementos.
- En todas las operaciones aritméticas se debe prestar especial atención en los desbordamientos. Se deben realizar funciones para realizar operaciones seguras. No se deben utilizar divisiones, ya que no están directamente soportadas por el PIC sino emulada por el compilador con múltiples instrucciones en ensamblador.
- Se deben probar partes pequeñas de código, debido a que no se puede depurar. Es importante ir creando bibliotecas de funciones utilizables ya probadas.

5.3. Filtrado de lecturas esporádicas

El filtrado de lecturas esporádicas se hace necesario sobre todo en el caso de que los sensores del robot no se encuentren sobre la línea negra, y capte un valor esporádico de negro, producido por suciedad o brillos en la pista. En la figura 5.2 se muestra un ejemplo de como un ruido, en el momento de encontrarse fuera de la pista, podría hacer que el robot se saliese.

Para solucionar esto, antes de realizar otra acción distinta a la que estabamos haciendo nos aseguramos de que ya estamos viendo efectivamente la pista y no es una lectura esporádica. En la figura 5.1 se puede ver la representación temporal en la percepción de una lectura esporádica. El tiempo *te* para considerar que se trata de una lectura esporádica y no de la pista se ha establecido en 4 ms para Diego y en 2 ms para D2. En este tiempo, y según su velocidad, ambos

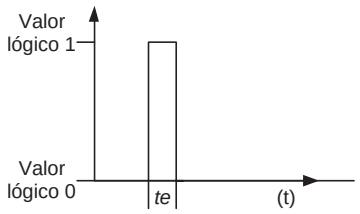


Figura 5.1: Representación temporal de una percepción esporádica

recorrerán 1.6 mm. En la figura 5.2 aparece el algortmo a seguir para evitar lecturas esporádicas.

5.4. Estructura del software para rastreadores

En la figura 5.3 se muestra el pseudocódigo del programa de la prueba de rastreadores:

El control de Diego se basa en un bucle principal en el que se realiza una lectura de la entrada, se preprocesa, se calcula el nuevo estado en función de la entrada y del estado actual, y por último se actúa.

La función *leeSensores* retorna la lectura del puerto que contiene los sensores de infrarrojos.

La función *procesaLectura* extrae el parámetro de la dirección con la solución que se explicó en 4.2. Para ello se han definido las siguientes funciones:

- char bitD(char n): Devuelve el primer bit puesto a uno por la derecha del parámetro de entrada n
- char bitI(char n): Devuelve el primer bit puesto a uno por la izquierda del parámetro de entrada n
- char ladoMarca(char D, char I, char antD,char antI): Retorna el lado por el que se ha localizado la marca. Esta información está en función de la lectura actual y la anterior.

La función *evento* retorna el código del evento que se ha producido en función de la entrada. En esta función se comprueban los eventos siguientes: sobre la línea, fuera de la línea, sobre un hueco y temporización terminada.

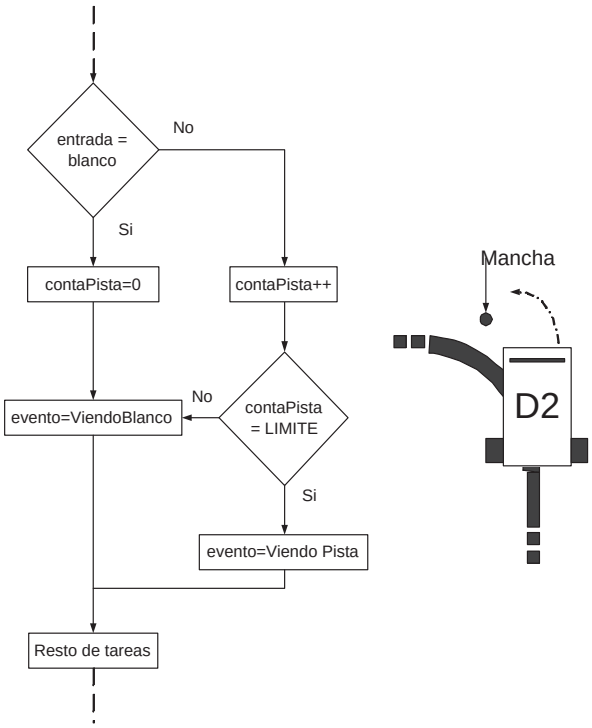


Figura 5.2: Modo de filtrar lecturas esporádicas y caso de ejemplo

La función *calculaEstado* devuelve el nuevo estado a partir del estado actual y del evento producido realizando una consulta a la tabla de transiciones y estados. Esta tabla es bidimensional, aunque para adaptarla al PIC se hizo unidimensional (figuras 4.12, 4.26 y 4.21).

La función *actuar*, es la que implementa el comportamiento del robot en función del estado actual y de la entrada. Como ya se dijo, las actuaciones posibles son control normal y control lateral. En el caso de D2, esta función incluye el sistema de control de velocidad.

5.5. Estructura del software de velocistas

El software para velocistas en principio es más sencillo que el de rastreadores. Como ya se explicó anteriormente se trata de un agente reflejo con estado interno, el cual para una determinada entrada y un estado, realiza una actuación según sus tablas.

```
MIENTRAS (1==1)
{
lectura=leeSensores();
lecturaTratada=procesaLectura(lectura);
estado=calculaEstado(estado,evento(lecturaTratada));
actuar(estado,lecturaTratada);
}
```

Figura 5.3: Pseudocódigo del programa de rastreadores

Su principal complejidad se basa en el sistema de control de velocidad. En cada ciclo de control, se lee el periodo de un paso entre dos dientes de la reductora de cada motor.

Para medir la velocidad se utiliza un sensor de infrarrojos de corte colocadondo entre sus dientes una rueda dentada. La señal para el control de velocidad, no es más que un bit cero o uno dependiendo si delante del emisor del sensor hay un diente del engranaje o no como se puede ver en el esquema de la figura 5.4. Este bit se lee de un puerto del PIC. Hay dos sensores, uno por cada rueda

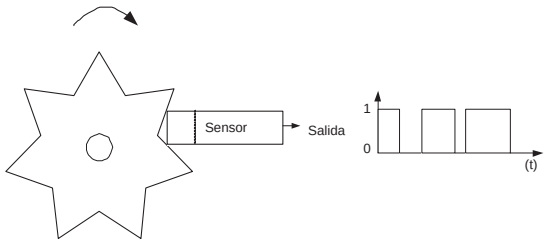


Figura 5.4: Esquema del sensor de velocidad sobre un engranaje

para permitir controlar la velocidad individualmente.

Según la relacción de los engranajes, para una velocidad final del D2 de 1.5 m/s hay un paso de un diente delante del sensor cada 700 μs. Sin embargo, a velocidades mayores el tiempo también será menor.El tiempo medido entre dientes es inversamente proporcional a la velocidad.

La única forma que tenemos en el PIC de medir este tiempo será por sondeo, contando el número de iteraciones de un bucle que tarda en suceder una

transición 0-1-0, o bien 1-0-1 por cada rueda. El código fuente en lenguaje C para medir la velocidad se puede ver en la figura 5.5:

```
byte contador=0;
byte lecInicio=SENSOR1; //Si está a uno mediremos 0-1,
                        //si está a 0 mediremos 1-0

//Esperamos al principio de la transición
while ((contador++<255)\&\&(SENSOR1==lecInicio));

//Estamos no estamos parados entonces medimos
if (contador<255) contador=0;

//Esperamos a que pase el 0 o el 1
while ((contador++<255)\&\&(SENSOR1!=lecInicio));

//Esperamos a que pase el 1 o el 0
while ((contador++<255)\&\&(SENSOR1==lecInicio));
```

Figura 5.5: Código del sistema para medir el periodo de paso entre dientes

Se denomina T_1 al periodo entre dientes para la rueda 1 y T_2 para la rueda 2. T_1, al igual que todas las variables, se recoge en un byte con 256 valores posibles, representando cada unidad un intervalo de 30 μs. En el caso que vaya muy lento siempre se leerá 255. En el caso que vaya a la velocidad máxima se leerá un valor de 23. Para conseguir una velocidad determinada se ha utilizado el control proporcional siendo k_p el parámetro proporcional, de forma que la potencia a aplicar al motor es un término BIAS (v_{BIAS}) mas un error (u) proporcional a la diferencia de la velocidad deseada (v_{ref}) y la medida(v_m).

- $V_{Motor} = V_{BIAS} + u$
- $u = k_p(v_{ref} - v_m)$
- $v_m = n(\frac{1}{T_1})$
- $T_1 = it \times cb$
- $k_p = 1 + \frac{V_m}{v_p}$

V_{Motor} es la velocidad a aplicar al motor
V_{BIAS} es el valor por defecto
u es el factor de error calculado
T_1 es el periodo medido
it es el número de iteraciones del bucle
cb es la constante de tiempo de cada iteración ($\approx 30\mu$s)

v_m es la velocidad medida
n es la constante de la relacción de los engranajes
v_p es la constante divisora del factor proporcional (16)

Debido a las limitaciones del PIC, solo es posible trabajar con variables tipo byte, por lo que las velocidades tendrán valores entre 0 y 255. El 255 representa a 1.5 m/s, el 128 equivale a parado, y el 0 equivale a -1.5 m/s.

Este controlador de velocidad se aplica en cada ciclo de control para los dos motores, por lo que su resultado final es bastante bueno a velocidades medias y altas. Sin embargo a velocidades bajas, nuestro contador del periodo entre dientes siempre estará a 255. Esto no es importante ya que D2 no tiene necesidad nunca de ir a velocidades bajas tanto en la prueba de rastreadores como en velocistas. La velocidad es la inversa del periodo, por lo que se necesita hacer una división. Sin embargo debido a que tan solo hay 256 valores posibles de periodo, se han precalculado los valores para todos los casos y se han puesto en una matriz, evitando de esta forma una división en cada cálculo.

El pseudocódigo para la prueva de velocistas se puede observar en la figura 5.6:

```
MIENTRAS (1==1)
{
lectura=leeSensores();
lecturaTratada=procesaLectura(lectura);
estado=calculaEstado(estado,evento(lecturaTratada));
actuar(estado,lecturaTratada);
}
```

Figura 5.6: Pseudocódigo para velocistas

Al igual que con el software de Diego, la función *leeSensores* proporciona una lectura de los puertos donde están conectados los sensores de infrarrojos. Esta lectura se procesa con la función *procesaLectura*. En este caso, el procesado solamente sirve para obtener el valor de dirección con el primer bit derecho e izquierdo puesto a 1. Para ello se utilizan las mismas funciones que en Diego:

- char bitD(char n): Devuelve el primer bit puesto a uno por la derecha del parámetro de entrada n
- char bitI(char n): Devuelve el primer bit puesto a uno por la izquierda del parámetro de entrada n

La función *evento* en este caso sólo debe detectar el evento de línea o blanco, filtrando previamente lecturas esporádicas, como se vió en 5.3.

La función *calculaEstado*, retorna el nuevo estado en función del estado actual y del evento. Los estados se almacenan conceptualmente en un array bidimiensional.

Por último la función *actuar*, realiza el comportamiento adecuado según el estado actual. En esta función se incluye el sistema de control de la velocidad explicado anteriormente.

5.6. Herramientas de programación

El código se ha realizado en C debido a la íntima relacción entre éste y el código ensamblador. Con la compilación se obtiene un código que resulta bastante optimizado, por lo que no se requiere su modificación antes de su ensamblado. La potencia proporcionada por el PIC a 4MHz, ha sido más que suficiente y no se ha necesitado afinar el código modificando el código ensamblador. Sin embargo se ha prestado atención en la optimización del código C de forma que las funciones repetitivas fuesen lo más rápidas posibles.

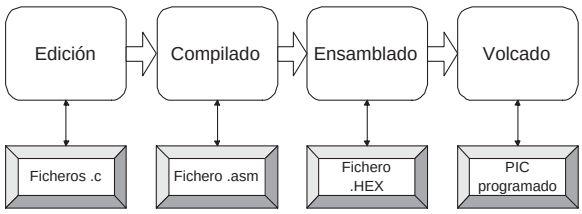


Figura 5.7: Etapas en la programación del PIC

En la figura 5.7 se muestran las etapas necesarias en la programación del PIC de cada robot. En la etapa de edición, se obtendrá el fichero en código C. En la compilación se obtiene un fichero con extensión .asm, el cual contiene

los mnemónicos en ensamblador para el PIC. Este fichero se podría editar en este punto si fuese necesario. A continuación se ensambla el fichero .asm y se obtendrá un fichero con extensión .HEX. Este fichero contiene el código máquina que se va a volcar en la última etapa al PIC.

Se ha utilizado el entorno integrado C2C++ [C2C], el cual permite manejar fácilmente todas las herramientas correspondientes a las etapas del desarrollo, aunque el ensamblador y el programa de volcado no pertenecen a este entorno. El ensamblador es el programa MPASM. Este programa es gratuito y lo proporciona directamente MICROCHIP el fabricante del PIC [MIC]. El programa del volcado del código máquina se realiza con el programa IC-PROG [ICP], tambien gratuito. La conexión del PIC al PC para el volcado se realiza con el programador SMT2.

5.7. Comparación entre Diego y D2

Se finalizó la programación de Diego en abril de 2002, y D2 en marzo de 2003. En ambos casos no se han realizado mejoras importantes desde la fecha debido a que se plantean como soluciones que explotarán al máximo las características de ambos robots. Sin embargo, D2 resulta un robot más versátil sobre el que sin duda se realizarán nuevas versiones de software.

D2 está claramente diseñado para velocistas y Diego para rastreadores, por lo que en rastreadores con D2 el software debe solucionar muchos problemas que Diego no tiene. Debido a la disposición de los sensores y actuadores de D2, se utiliza más el estado de salirse de la pista y seguir torciendo hacia el lado por el que la abandonó. Normalmente, en curvas cerradas D2 no podrá mantener sus sensores encima de la pista, ya que su cinemática se lo impide. El estudio de este problema ha sido muy importante en D2 como rastreador, ya que debe evitar ver lecturas esporádicas mientras no vea la pista para evitar salirse totalmente (ver figura 5.9). Sin embargo, será muy difícil ver a Diego sin los sensores sobre la pista, ya que su disposición se adapta a su radio de giro mínimo.

Hay bastantes diferencias en el tratamiento de la entrada debido al mayor número de sensores. También hay más niveles de potencia de los motores, 16 de Diego frente a 256 de D2. Estas diferencias hacen que se deba prestar mucha

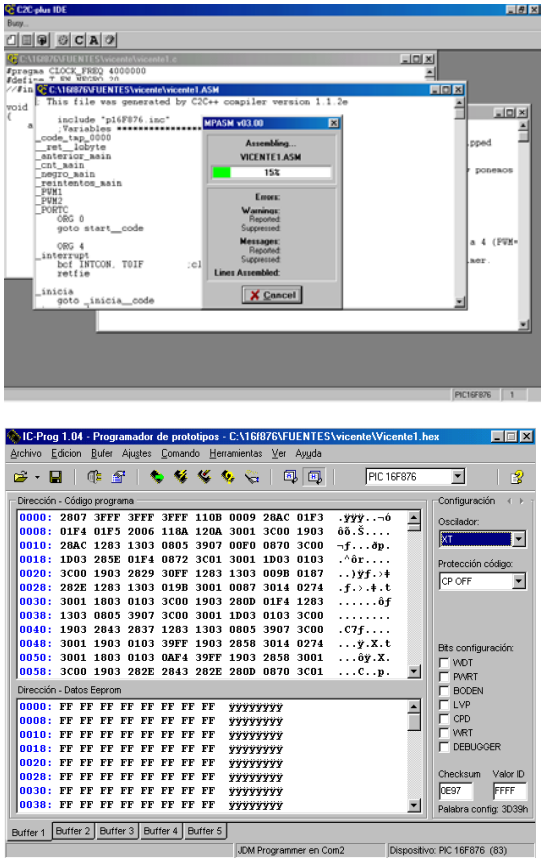


Figura 5.8: Entorno integrado C2C++, IC-PRog y tarjeta programadora SMT2

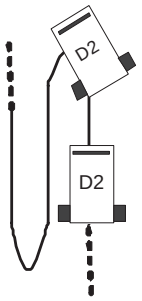


Figura 5.9: D2 en rastreadores en una curva cerrada

atención a la programación del PIC, ya que solo dispone de variables tipo byte. La naturaleza de D2 al tener los sensores en línea, hizo necesario añadir un nuevo estado de detección de ángulos rectos. En Diego no sucedía debido a que su disposición en V invertida favorece ver por más tiempo el ángulo.

La conclusión de este apartado es que se puede corregir por software la mayoría de las deficiencias del hardware. La lógica borrosa ayuda en ambos robots a evitar problemas como que los motores no vayan igual de rápido para la misma potencia aplicada, admitiendo tolerancias importantes.

6

Resultados y conclusiones

En este trabajo, se ha planteado la concepción de robots móviles destinados a la competición en diversas categorías. En este sentido, en las diferentes fases de su diseño y construcción se ha llegado a diversos resultados que, a continuación, se detallan:

- Se ha utilizado el concepto de agente inteligente en la definición de las especificaciones y componentes que constituyen el robot. Se ha constatado que, para aquellos que están destinados a la participación de concursos de robótica, la autonomía se ha convertido en el problema central de diseño. En este sentido, se ha realizado la descripción PAMA de cada uno de los robots, lo que ha conducido a la propuesta de la arquitectura de agente más adecuada en cada caso o competición. Finalmente, se han implementado dos robots: Diego para la especialidad de rastreadores y D2 para la de velocistas.

- A partir de la arquitectura propuesta en cada caso, se ha realizado un análisis de los sistemas de tracción más adecuados. En este sentido la configuración diferencial ha sido considerada como la más adecuada atendiendo fundamentalmente a un balance de los costes y la maniobrabilidad. Asimismo, se han considerado diferentes posibilidades motrices como es la utilización de servos o de motores de corriente continua con puente de potencia.

- Respecto a los sistemas sensores, la opción elegida ha sido la utilización de fotodetectores. Se trata de elementos de bajo coste cuyo procesamiento además requiere escasos recursos computacionales. Se ha desarrollado un procedimiento que permite disponer de una representación de la posición del robot lineal que es inmune al grosor de la línea.

- La arquitectura elegida ha sido la de agente reflejo con modelo interno. Ha sido la inaccesibilidad del ambiente la que ha justificado esta decisión. Ha sido necesario, en cada caso, desarrollar un autómata de estados finitos lo que ha requerido de un análisis detallado para determine los estados que describen al robot en cada instante, las transiciones así como las acciones en cada estado.

- El núcleo central del comportamiento inteligente del agente, se ha desarrollado utilizando la lógica borrosa. Esta elección se justifica por las ventajas que proporciona con suaves reacciones que permiten circular a velocidades altas sin oscilaciones que saquen al robot de la pista. Además, su aplicación se puede llevar a cabo mediante superficies o curvas de acción por lo que se reducen los requerimientos hardware y software. Así, el microcontrolador PIC 16F876 dispone de la potencia necesaria para la implementación final del control inteligente de los robots desarrollados.

- Los robots han acudido a diversos concursos que se han saldado con unos buenos resultados lo que muestra la validez de los prototipos diseñados y por tanto de las tecnologías aplicadas. Se ha comprobado como un diseño correcto de los comportamientos inteligentes ha permitido alcanzar unos buenos resultados incluso sobre plataformas electro-mecánicas sencillas y de bajo coste.

Además de estas conclusiones donde se han mostrado los principales avances técnico-científicos, nos ha parecido necesario incluir una serie de comentarios de índole práctico que pueden ser incluso más importantes que las anteriores. Éstos se sustentan en la experiencia personal de un constructor y desarrollador de robots y serán útiles para aquellos que afronten esta aventura.

- La realización de estos dos robots ha permitido obtener una base de conocimiento muy importante para la futura realización de otros prototipos. En este momento, con los conocimientos adquiridos, resulta relativamente sencillo llegar a una implementación funcional partiendo de los objetivos iniciales sin detenerse solamente en la simulación sin darnos cuenta que sobre el papel funciona todo.

- La realización de un robot se convierte en una tarea por etapas que permite ir realizando pruebas durante el desarrollo. Lo más recomendable es primero realizar la placa principal del sistema, comprobando que funciona correctamente. De esta manera es posible probar los sensores y actuadores por separado con pequeños programas. En la parte de software es recomendable hacer lo mismo, ir probando pequeñas partes de código. En efecto, en sistemas de este tipo hay funciones que se llaman contínuamente, por lo que se debe prestar mucha atención a la optimización del código.

- También han sido muy importantes las restricciones del PIC utilizado, ya que solo se pueden utilizar variables de tipo byte o char, y con una limitación de 384 bytes. Siempre se debe prestar atención a los límites de bucles, desbordamientos por sumas y restas, y la existencia otras limitaciones como el uso de arrays de una dimensión solamente, que hacen que se complique un poco la tarea. Sin embargo siempre es mucho mejor que programar en ensamblador, ya que su uso resulta desproporcionado. Los compiladores suelen generar un código ensamblador bastante bueno, ya que el lenguaje C se presta a ello. El uso de ensamblador directamente, solo se ve justificado cuando haya partes críticas en el tiempo, cosa que no ocurre en nuestro caso.

- Otro aspecto importante ha sido el coste del desarrollo. Económicamente no ha sido un coste muy grande pero si lo ha sido en tiempo utilizado.

> Resulta evidente que se debe mantener la proporcionalidad del coste del desarrollo, y tratar de evitar gastar grandes sumas de dinero. Efectivamente existen en el mercado sensores láser, motores sin escobillas, y una multitud de sutilezas que podrían haber sido una solución magnífica para el desarrollo de un robot, pero que hubiesen multiplicado el coste por diez y deslucirían el resultado. La dedicación en horas se ve justificada por la ausencia de material de base, que poco a poco se ha ido solucionando. La solución de compromiso sería la colaboración con otros departamentos especializados en mecánica y electrónica para la realización del robot. Sin embargo esto es muy bonito en teoría, pero en la práctica se debe pensar siempre en la máxima que dice: -si quieres algo hazlo tu mismo-.

Experimentalmente se ha concluido que cuando se crea un robot, una parte importante del tiempo es dedicada a solucionar problemas con los que no se contaba. Esto se puede solucionar en gran medida reutilizando diseños suficientemente probados. En nuestro caso, de forma evolutiva se ha llegado a la conclusión que la disposición de los distintos elementos del robot se asemejan bastante a los robots campeones de concursos. Como corolario cabe pensar que en adelante debemos aprovecharnos de diseños ganadores y no tratar de reinventar la rueda.

Por último, como revisión de las experiencias vividas entorno a Diego y D2, se destacan las más importantes:

- El software es una de las partes más importantes del robot
- Hay que tratar de tener un coste razonable
- El dia del concurso no se deben modificar los robots

La afirmación del primer punto se entiende rápidamente en los concursos o demostraciones de robótica. Existen muchísimos robot, con un hardware excelente, pero que no hacen nada bien. Es como tener un coche y no saber conducir.

El segundo punto, también se entiende en el mismo contexto. Se suelen invertir grandes cantidades de tiempo y dinero para desarrollar robots, pero con

resultados como los que demuestran Diego[1] y D2[2] se empieza a entender que no es tan necesario.

Como cierre, señalar que en los concursos, los robots no se deben cambiar ni tratar de realizar modificaciones de última hora, ya que pueden echan por tierra el trabajo de mucho tiempo.

Figura 6.1: Diego y D2

[1]Diego es Campeón de España 2002 (ALCABOT-02), Subcampeón de España 2003 (HISPABOT-03) y Subcampeón de Castilla-León 2003 (ROBOLID-03) en la prueba de rastreadores.

[2]D2 es Campeón de Castilla-León 2003 (ROBOLID-03) de rastreadores, Subcampeón de Castilla-León 2003 (ROBOLID-03) y cuarto en España 2003 (HISPABOT-03) en velocistas.

Bibliografía

[AIB] Aibo. http://www.aibo-europe.com/.

[Bla37] Max Black. Vagueness, an exercise in logical analysis. *Philosophy of Science*, 4(4):427–455, 1937.

[BRC95] J.W. Burdick, J. Radford, y G.S. Chirikjian. Sidewinding locomotion gait for hyper-redundant robots. *Advanced Robotics*, 9(3):195–21, 1995.

[BW99] J. Bares y D. Wettergreen. Dante II: Technical description, results and lessons learned. En *International Journal of Robotics Research*, tomo 18, páginas 621–649. Julio 1999.

[C2C] C2c++ c++ compiler. http://www.picant.com/c2cpp/cpp.html.

[CM02] J. M. Corchado y J. M. Molina. *Introducción a la Teoría de Agentes y Sistemas Multiagente*. Publicaciones Científicas, 2002.

[DC02] Salvador Dominguez y Eduardo Zalama Casanova. A mobile robot with enhanced gestual abilities. En *9th Mechatronics Congress*. 2002.

[FLE] Flexitrack. http://www.flexitrack.com.

[HIS] Hispabot. http://www.hispabot.org/.

[ICP] Ic-prog prototype programmer. http://www.ic-prog.com/.

[MAB] Mabuchi motor. http://www.mabuchi-motor.co.jp/english/.

[Men42] Karl Menger. Statistical metrics. En *National Academy of Science, U.S.A*, tomo 28, páginas 535–537. 1942.

[MIC] Microchip. http://www.microchip.com.

[Nil97] M.Ñilsson. Snake robot free climbing. En *International Conference on Robotics and Automation*, páginas 3415–3420. 1997.

[Oll01] A. Ollero. *Robótica. Manipuladores y robots móviles*. Marcombo Boixareu Editores, 2001. ISBN 84 267 1313 0.

[RN96] Stuart Russell y Peter Norvig. *Inteligencia Artificial. Un enfoque moderno*. Prentice Hall, 1996.

[ROB] Robocup. http://www.robocup.org/.

[SDR] Sdr-4x. http://www.tokyodv.com/news/SonySDR-4XRobot.html.

[ST] Stmicroelectronics. http://www.st.com.

[Tur50] A.M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[UYM01] Jose M. Angulo Usategui, Susana Romero Yesa, y Ignacio Angulo Martinez. *Microbótica*. Paraninfo, segunda edición, 2001.

[VIS] Vishay. http://www.vishay.com.

[WT96] D. Wettergreen y C. Thorpe. Developing planning and reactive control for a hexapod robot. En *International Conference on Robotics and Automation*, tomo 3, páginas 2718 – 2723. Abril 1996.

[Yim01] Mark Yim. Climbing with snake-like robots. En *IFAC Workshop on Mobile Robot Technology*. Mayo 2001.