Inteligencia Artificial
Pedro Amador Díaz
09/10/2011

UOC
Universitat Oberta de Catalunya
www.uoc.edu

Índice

Introducción

Cada año se celebran concursos (tanto en el ámbito nacional como internacional) en los que instituciones y aficionados a la robótica compiten en innumerables categorías (sumo, laberinto, fútbol, rescate, bípedos…). Una de estas categorías es la de robots velocistas, estos deben autopilotarse en un circuito cerrado alcanzando la mayor velocidad posible.

Figura 1. Carteles y logotipos de algunos de los concursos que se celebran cada año.

El robot velocista puede navegar sobre las dos líneas negras que conforman el circuito, siempre que no pise la línea roja (límite exterior e interior de la pista). En las primeras rondas eliminatorias, no se establece velocidad mínima, el robot debe realizar un total de 3 vueltas cronometrándose el tempo empleado para ello. A las siguientes rondas solo pasan los robots más veloces, los cuales competerán en parejas. Los dos robots comienzan en extremos opuestos del circuito, ganando el que alcance a su oponente.

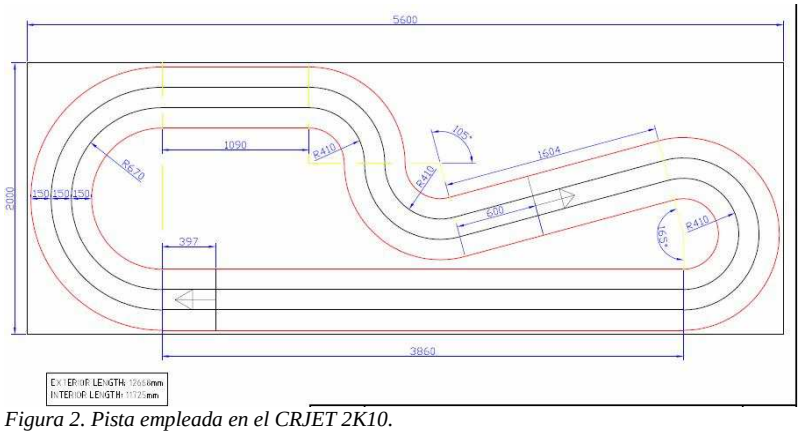


Figura 2. Pista empleada en el CRJET 2K10.

Una técnica de pilotaje es seguir el trazado de una de las dos líneas negras, navegando sobre la misma (esta es la técnica que emplearé es este proyecto). Otras técnicas de pilotaje consisten en ir alternando el trazado de las dos líneas negras o emplear cámaras y visión artificial para percibir los limites de la pista y su trazado.

Para solucionar este problema emplearé varias técnicas de IA:

- Control reactivo basado en el conocimiento. Necesario para seguir la pista.
- Sistemas multiagente e inteligencia emergente. Dotará al robot de dirección y tracción integral a las cuatro ruedas.
- Algoritmo genético. Permitirá la adaptación automática del robot a la pista.

Objetivos

El objetivo último es conseguir una plataforma robótica, que permita experimentar con conceptos aprendidos durante la carrera y que a su vez, cumpla con los requisitos mínimos para poder participar en un concurso de robótica.

Objetivos principales

1- Desarrollar una plataforma electromecánica que permita experimentar con el control multiagente y la inteligencia emergente.

2- Un sistema de control en tiempo real y determinista, que permita dar respuestas rápidas a las condiciones cambiantes del entorno.

3- Comunicación serial para monitorizar el comportamiento del sistema.

4- Crear un entorno de pruebas.

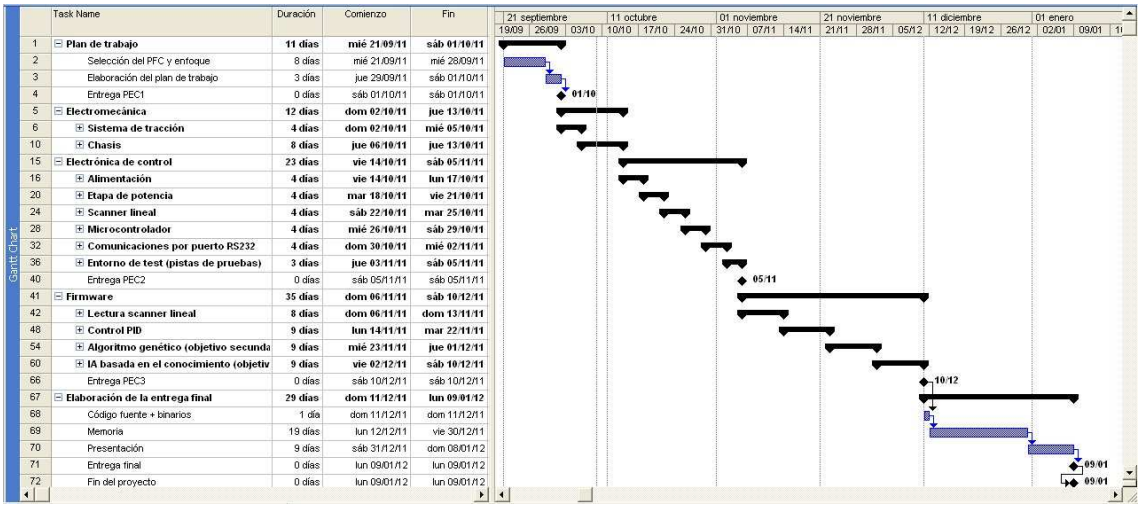
5- Implementar un control sobre IA reactiva PID.

Objetivos secundarios

6- Sintonizar el control PID mediante algoritmos genéticos.

7- Implementar un sistema de IA basado en el conocimiento, que permita al robot ajustar su velocidad al radio de la curva.

Planificación



	Task Name	Duración	Comienzo	Fin
1	**Plan de trabajo**	**11 días**	**mié 21/09/11**	**sáb 01/10/11**
2	Selección del PFC y enfoque	8 días	mié 21/09/11	mié 28/09/11
3	Elaboración del plan de trabajo	3 días	jue 29/09/11	sáb 01/10/11
4	Entrega PEC1	0 días	sáb 01/10/11	sáb 01/10/11
5	**Electromecánica**	**12 días**	**dom 02/10/11**	**jue 13/10/11**
6	**Sistema de tracción**	**4 días**	**dom 02/10/11**	**mié 05/10/11**
10	**Chasis**	**8 días**	**jue 06/10/11**	**jue 13/10/11**
15	**Electrónica de control**	**23 días**	**vie 14/10/11**	**sáb 05/11/11**
16	**Alimentación**	**4 días**	**vie 14/10/11**	**lun 17/10/11**
20	**Etapa de potencia**	**4 días**	**mar 18/10/11**	**vie 21/10/11**
24	**Scanner lineal**	**4 días**	**sáb 22/10/11**	**mar 25/10/11**
28	**Microcontrolador**	**4 días**	**mié 26/10/11**	**sáb 29/10/11**
32	**Comunicaciones por puerto RS232**	**4 días**	**dom 30/10/11**	**mié 02/11/11**
36	**Entorno de test (pistas de pruebas)**	**3 días**	**jue 03/11/11**	**sáb 05/11/11**
40	Entrega PEC2	0 días	sáb 05/11/11	sáb 05/11/11
41	**Firmware**	**35 días**	**dom 06/11/11**	**sáb 10/12/11**
42	**Lectura scanner lineal**	**8 días**	**dom 06/11/11**	**dom 13/11/11**
48	**Control PID**	**9 días**	**lun 14/11/11**	**mar 22/11/11**
54	**Algoritmo genético (objetivo secunda**	**9 días**	**mié 23/11/11**	**jue 01/12/11**
60	**IA basada en el conocimiento (objetiv**	**9 días**	**vie 02/12/11**	**sáb 10/12/11**
66	Entrega PEC3	0 días	sáb 10/12/11	sáb 10/12/11
67	**Elaboración de la entrega final**	**29 días**	**dom 11/12/11**	**lun 09/01/12**
68	Código fuente + binarios	1 día	dom 11/12/11	dom 11/12/11
69	Memoria	19 días	lun 12/12/11	vie 30/12/11
70	Presentación	9 días	sáb 31/12/11	dom 08/01/12
71	Entrega final	0 días	lun 09/01/12	lun 09/01/12
72	Fin del proyecto	0 días	lun 09/01/12	lun 09/01/12

Figura 3. Captura de pantalla de la planificación del proyecto (MSProject).

Estimo una dedicación de 3 horas diarias.

Destacar que aunque hay un total de 7 días festivos, he marcado estos como días laborables, pues será precisamente en estos días y los fines de semana cuando podré corregir posibles desviaciones temporales en el plan de proyecto. Este es un aspecto muy importante, pues los plazos de tiempo son ajustados y al haber una única persona dedicada al proyecto, todas sus tareas son secuenciales y el camino critico pasa por cada una de ellas.

Días festivos anteriormente mencionados:

12/10/2011, 01/11/2011, 06/12/2011, 08/12/2011, 25/12/2011, 26/12/2011 y 01/01/2012.

Sistema electromecánico

Esta constituido por unas estructuras rígidas unidas por articulaciones, a la que llamaremos chasis. Esta estructura se mueve gracias a los actuadores (motores), elementos mecánicos que transmiten el movimiento a las articulaciones del robot, en nuestro caso ruedas.

Tracción

El sistema de tracción es el conjunto de elementos que aplican una fuerza a la pista para que esta a su vez realice otra fuerza sobre el robot, de igual magnitud pero en sentido contrario, que hará que el este se desplace (tercera ley de Newton).

El AIIV esta dotado de un sistema 4x4, de forma que las cuatro ruedas tienen la capacidad de traccionar. Esto es una ventaja sobre la mayoría de robots velocistas, los cuales siguen una configuración de triciclo con solo dos ruedas tractoras y una tercera sin capacidad de empujar, y un coeficiente de fricción cercano a cero. La fuerza normal generada por el peso que recae sobre esta tercera rueda es desperdiciada para tareas tan como la aceleración, el frenado o la oposición a la fuerza centrifuga en el paso por curva.

Figura 4. Robot velocista configurado en forma de triciclo.

Motores

Un motor es una máquina que transforma la energía eléctrica en movimiento. Hay muchos tipos diferentes, pero el más empleado en la robótica de bajo coste es el motor de corriente continua con escobillas, este es el que emplearé en el proyecto AIIV.

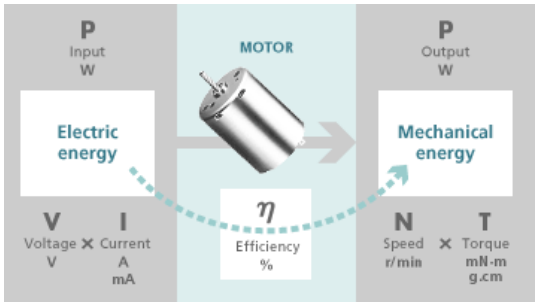


Figura 5. Transformación de la energía eléctrica en mecánica.

Un motor de corriente continua (DC) con escobillas está formado por un imán permanente en el exterior (estator) y unas bobinas electromagnéticas montadas en el eje del motor (armadura o rotor). Las escobillas se deslizan por unas piezas metálicas colocadas en el eje del motor, al girar estas conmutan la potencia de una bobina a otra, de forma que los campos magnéticos generados por las bobinas estén continuamente atraídos por el imán y el motor gire siempre en la misma dirección.

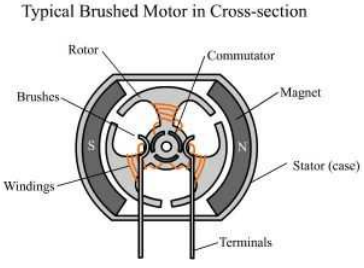


Figura 6. Motor DC con escobillas.

Los parámetros principales que describen las características de un motor son la velocidad, medida en rpm, y el torque, medido en mN·m o g·cm. La unidad del torque nos muestra la dependencia que hay entre la fuerza y la distancia a la que se aplica. Por ejemplo, un motor con 2 mN·m de torque puede producir una fuerza de 2 mN con una palanca de 1 metro conectada a su eje, 1 mN con una palanca de 2 metros, 0,5 mN con una palanca de 4 metros, y así sucesivamente.

Todo motor tiene una velocidad máxima (cuando no se aplica fuerza sobre su eje) y un máximo torque (cuando la fuerza aplicada en su eje para el motor). Estos parámetros se llaman free-running speed y stall torque (Ts). El motor consumo menos corriente cuando no se aplica fuerza sobre su eje, y la corriente aumenta a medida que la fuerza aplicada sobre su eje aumenta, hasta llegar al punto de paro (TS). Esta relación se puede ver en la figura 7, donde se relaciona la incendiad consumida por el motor (recta I), con el torque, las rpm y la eficiencia del motor. Esta gráfica es proporcionada por el fabricante y será de ayuda para calcular la caja reductora, así como la etapa de potencia que alimenta los motores.

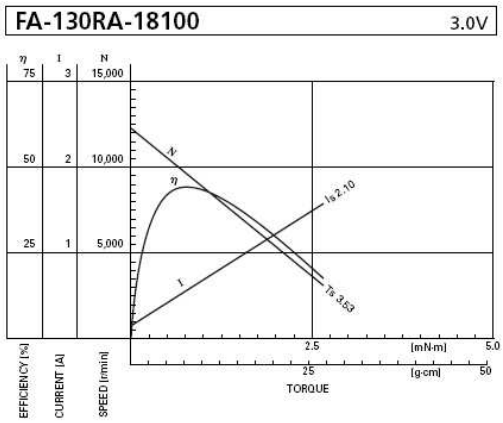


Figura 7.Grafica de las características del motor.

He optado por emplear 4 motores FA-130RA, fabricados por Mabuchi Motor.

La elección la he basado en las ventajas que este modelo aporta y que expongo a continuación:

- Torque suficiente para mover el robot.
- RPM suficientes para superar el 1m/s de velocidad máxima.
- Tensión nominal adecuada para un robot de baja tensión (3v).
- Se trata de un motor de bajo coste (1,99 $).
- Tiene una vida útil de unas 18 horas y es fácil encontrar repuesto.
- Ampliamente utilizado en la industria del pequeño juguete y la robótica de baja tensión.
- Otro fabricante (Tamiya) aporta una solución en la que integra dos motores FA-130RA con dos cajas reductoras.

Cajas reductoras

Normalmente la free-running speed (velocidad sin carga) de un motor es más alta que la velocidad que queremos alcanzar en la rueda. La caja reductora es un sistema de engranajes que convierten la alta velocidad y bajo torque entregados por del motor en una velocidad inferior y un torque mayor con el que poder mover las ruedas del robot. Si no aumentamos el torque, este no será suficiente para vencer la resistencia al avance y el robot no se moverá.

La resistencia al avance es la fuerza que se opone al movimiento del robot y depende de los siguientes factores:

- El coeficiente aerodinámico del robot. Este factor es despreciable a las velocidades que se desplaza el AIIV.
- La resistencia a la rodadura.
- La fricción de los ejes y engranajes de la caja reductora.
- La pendiente de la pista. Este factor es despreciable pues se supone que la pista no tiene desniveles.

En el caso del AIIV, esta fuerza es de 0.16N. Es un valor tan baja que esta fuerza opositora al avance puede ser despreciada. Esto conlleva que el robot podría arrancar sin reductoras, pero las aceleraciones y deceleraciones serian muy pobres, siendo inviable un robot velocista con esas prestaciones.

La fuerza opositora al avance se ha calculado empleando un plano inclinado, un clásico de la dinámica de partículas. El experimento consiste en colocar el robot en el plano inclinado, con la alimentación desconectada y los motores desengranados. Acto seguido se va aumentando el ángulo 'a' hasta que el robot empieza a deslizarse por el plano inclinado. En ese momento la fuerza opositora al avance ha sido superada por Px. Para calcular Px solo nos hace falta saber el peso del robot, que en nuestro caso es de 0.55 Kg.

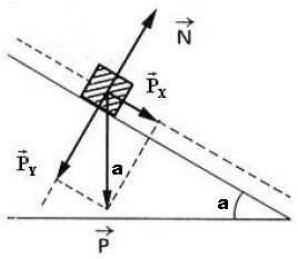


Figura 8. Plano inclinado.

Calculo de la fuerza opositora al avance:

$$angulo = 1.83^{o}$$

$$gravedad = 9.81 m/s^2$$

$$masa = 0.55 Kg$$

$$P_x = (gravedad * masa) * \sin(angulo) = (9.81 m/s^2 * 0.55 Kg) * \sin(1.83^{o}) = 0.16 N$$

Volvamos a la elección de la caja reductora. He optado por emplear 2 cajas reductoras 70168 Double gearbox kit fabricadas por Tamiya. Este kit puede ser configurado de cuatro formas distintas proporcionando diferentes relaciones reductoras. Esta es una característica interesante cuando se trabaja con prototipos, ya que nos permite experimentar con diferentes relaciones de velocidad-torque.

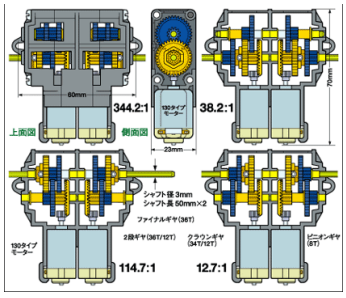


Figura 8. Doble gearbox kit de Tamiya.

La elección la he basado en las ventajas que este modelo aporta y que expongo a continuación:

- Cada kit integra dos motors FA-130RA y dos cajas reductoras, formando un conjunto muy fiable.
- Bajo coste. Cada kit tiene un coste de 9.25$.
- Fácilmente reconfiguradle. Configuraciones reductoras disponible: 344.2:1, 114.7:1, 38.2:1, 12.7.

En el caso del proyecto AIIV, el factor decisivo a la hora de escoger la relación reductora ha sido la velocidad punta que queremos que alcance el robot. Según los objetivos marcados por el proyecto, será sufriente con alcanzar una velocidad de 0.94m/s, por lo que optaré por una reductora de 38.2:1. Si en el futuro se quiere refinar el diseño y alcanzar mayor velocidad, se podrá reconfigurar las cajas reductoras con un factor 12.7, aunque para ello se tendrá que rebajar el peso del robot, pues con su masa actual las aceleraciones son tan pobres que se hace imposible controlar el movimiento del mismo.

Teniendo en cuenta que la velocidad máxima que puede alcanzar el motor es de 12300 rpm (ver figura 7) y las relaciones reductoras proporcionadas por el Double gearbox kit, obtendríamos las siguientes velocidades.

Relación reductora	Revoluciones por segundo en le eje de la rueda	Distancia recorrida por el robot en un segundo
344.2:1	(12300/60)/344.2=0.06rps	0.056m*3,1416*0.06rps=0.01m
114.7:1	(12300/60)/114.7=1.79rps	0.056m*3,1416*1.79rps=0.31m
38.2:1	(12300/60)/38.2=5.37rps	0.056m*3,1416*5.37rps=0,94m
12.7	(12300/60)/12.7=16.14rps	0.056m*3,1416*16.14rps=2,84m

Ruedas

He optado por emplear 2 pares de ruedas 70111 Sport tire set fabricadas por Tamiya.

Figura 9. Ruedas Sport Tire de Tamiya.

La elección la he basado en las ventajas que este modelo aporta y que expongo a continuación:

- Fácilmente acoplable a las reductoras Double gearbox kit.
- Bajo coste. Cada parar tiene un coste de 6.5 $.
- Elevado coeficiente de fricción sobre una superficie de papel (0.80).

El coeficiente de fricción no ha sido facilitado por el fabricante, con lo que he tenido que calcularlo con la ayuda de un plano inclinado y las leyes de la mecánica clásica de Newton.

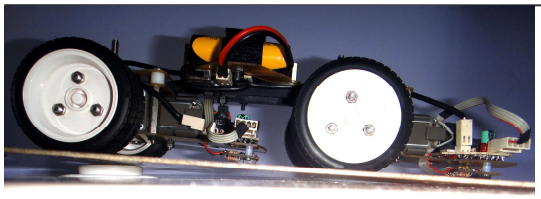


Figura 10. Robot AIIV sobre el plano inclinado.

El experimento consiste en colocar el robot en el plano inclinado, con las ruedas bloqueadas. Acto seguido se va aumentando el ángulo que forma el plano inclinado con el suelo hasta que el robot empieza a resbalar.

En ese momento la componente tangencial del peso, ha superado la fuerza de rozamiento. Sabiendo el ángulo del plano inclinado en ese instante, se puede calcular el coeficiente de fricción estático como la tangente del ángulo del plano.

El coeficiente de fricción es un parámetro fundamental, ya que limitará la velocidad máxima en el paso por curva.

Conjunto Motores-reductoras-ruedas

Como resultado de que el fabricante Tamiya haya diseñado las ruedas y reductoras para que puedan ser esambladas junto con los motores Mabuchi FA-130, despues del montaje optenemos un sistema tractor 4x4, fiable, flexible (las cajas reductoras pueden ser reconfiguradas) y a un precio economico.

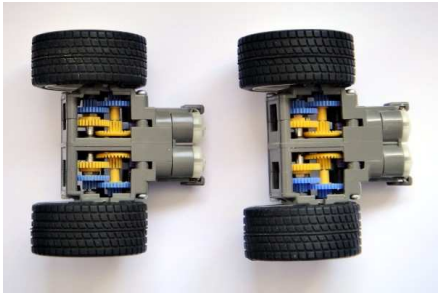


Figura 11. Sistemas de tracción montados.

Coste del sistema de tracción 4x4

Articulo	Unidades	Precio por unidad
Tamiya 70111 Sport tire set	2	6.5 $
Tamiya 70168 Double gearbox kit	2	18,5 $
	Total	31,5 $

Chasis

Da forma y rigidez al robot. El chasis es el soporte de las baterías, la electrónica de control, los sensores y los actuadotes (motores) del robot.

Diseño

Este aspecto es innovador en este tipo de robot. Hasta el momento, ningún robot velocista había estado formado por dos agentes independientes.

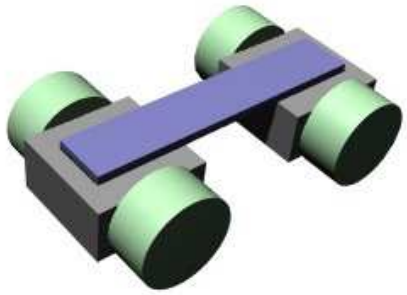


Figura 12. Configuración multiagente.

Esta configuración forma una granja de robots. Esta comunidad o granja está formada por dos robots, los cuales realizan una tarea en un mismo entorno de forma simultánea e independiente. Cada uno actúa sin saber que el otro también está ahí, pero el conjunto de sus necesidades aporta mayores prestaciones que la conjunción de todas ellas en un solo agente (inteligencia emergente). Como estos robots no están coordinados, es la estructura del chasis del AIIV la que determina que los objetivos de los agentes no sean contrarios. Los dos agentes tienen el objetivo de seguir la línea lo más rápidamente posible, avanzando siempre hacia delante. Estos están unidos mecánicamente por una pieza central que dispone de dos ejes de un grado de libertad cada uno, de forma que el agente A (delantero) tirará del agente B (trasero) o en el caso contrario el agente A será empujado por el agente B. Solo se pueden dar estos dos escenarios en los que el resultado final es siempre el mismo, el robot avanza siguiendo la línea a la mayor velocidad posible.

Esta configuración no solo aporta una tracción 4x4 sino que también conlleva una dirección a las cuatro ruedas, también llamada 4RD.

Construcción

El chasis está elaborado, en su mayoría, en fibra de vidrio debido a su ligereza y resistencia. Para el eje central que une los dos agentes, he empleado metacrilato, ya que ofrece mayor rigidez que la fibra de vidrio. Se han ensamblado todas las piezas empleando separadores, tornillos y tuercas de acero. Por ultimo he empleado una tira de velcro para asegurar la fijación de la batería, así como su rápida sustitución.

Figura 13. Despiece del chasis.

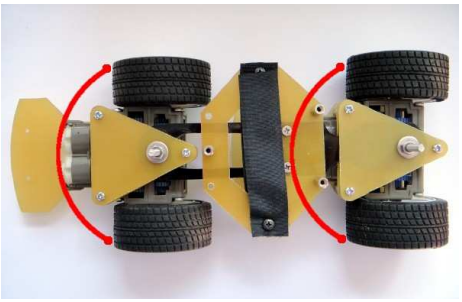


Figura 14. En rojo, el grado de libertad de cada agente.

En la figura 14 puede verse como cada uno de los dos agentes puede girar sobre si mismo (líneas rojas), esto posibilita la dirección 4RD, lo que unido a la tracción 4x4, mejora la dinámica del vehiculo respecto a los robot velocistas en forma de triciclo.

Coste del chasis

Articulo	Unidades	Precio por unidad
Placa de fibra de vidrio de 30x30cm y 1mm de grosor	1	10 $
Placa de metacrilato de 25x25cm y 6mm de grosor	1	5.5 $
Separador de 2.5cm (paquete de 4)	2	1.29 $
Tornillos de 3mm (paquete de 25)	1	0.69 $
Espaciadores de nylon (paquete de 50)	1	1.79 $
Tuerca de 3mm (paquete de 25)	1	0.59 $
	Total	22,15 $

Electrónica de control

Sistema electrónico que tiene como función, controlar el robot, dotándolo de inteligencia. Sus datos de entrada están constituidos por la información proporcionada por los sensores. La salida de este sistema tiene como destino los actuadores del robot.

Alimentación

Por motivos de simplicidad y para no elevar el coste del robot, he optado por una única alimentación tanto para los actuadores (motores) como para los sensores y la electrónica de control.

Antes de estudiar la batería empleada, hay dos conceptos que deben de entenderse con claridad:

1- La tensión es la fuerza con la que los electrones son empujados, y se mide en voltios (V). Las baterías tienen una tensión nominal de 1.2V.

2- La intensidad o corriente eléctrica, es la cantidad de electrones que la batería puede empujar por segundo, y se medida en amperios (A). Una corriente de 1A corresponde a cerca de 6 × 1018 electrones que fluyen hacia un lado y al otro cada segundo.

Batería

Para cualquier batería, si se intenta extraer más y más intensidad, la tensión producida por la batería se reducirá, bajando hasta llegar a cero en la corriente de corto circuito. A este efecto es hay que tenerlo en cuenta cuando calculemos los controladores de los motores, pues si estos no limitan la intensidad, la tensión entregada por la batería podría situarse por debajo de los 4V, tensión a la que el microcontrolador dejaría de funcionar, dejando al robot sin control.

He optado por baterías del tipo NiCd por las siguientes razones:

- Pueden entregar elevadas intensidades.
- Son resistentes a las descargas rápidas y totales.

Las desventajas de este tipo de baterías son:

- Poca capacidad (800mAh).
- Efecto memoria.
- Son contaminantes.
- Un peso elevado.

Para contrarrestar la baja capacidad he adquirido tres paquetes de baterías y he dotado al VIIA de un sistema que permite cambiarlos en poco tiempo.

Coste de las baterías

Articulo	Unidades	Precio por unidad
Paquete de 5 células de NiCd (800mAh)	3	10 $
	Total	30 $

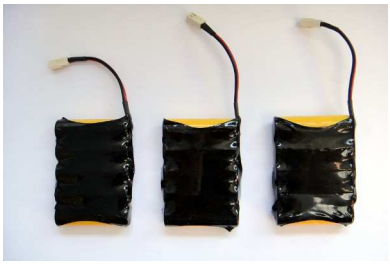


Figura 15. Paquetes de baterías NiCd.

Regulador

La electrónica de control y sensores deben ser alimentados con una tensión estable de 5 voltios. Como hemos visto, las baterías varían su tensión en función del grado de descarga y de la intensidad entregada. Una batería formada por 5 células en serie puede tener una variación de su tensión entre 7V y 5V. Con una tensión de 7V podríamos llegar

Figura 16. Regulador

a dañar la electrónica de control. Para evitar estas oscilaciones he empleado el regulador de 5V LM2940.

Coste reguladores de tensión

Articulo	Unidades	Precio por unidad
Regulador LM2940 5V	2	1.19 €
	Total	2.38 €

Etapa de potencia

El primer problema a considerar es la forma de alimentar el motor, ya que la corriente máxima que puede proporcionar cualquier línea de salida del un microcontrolador esta limitada a 25 mA como máximo. En esta sesión veremos como mediante las líneas de salida del microcontrolador podemos gobernar los motores para que cambien su sentido de giro o varíen su velocidad.

Puente en H

Existen varias formas de lograr que los motores inviertan su sentido de giro, En el AIIV utilizamos una fuente común con un conmutador doble es decir uno de 6 contactos, en todos los casos es bueno conectar también un capacitor en paralelo entre los bornes del motor, para amortiguar la inducción que generan las bobinas internas del motor.

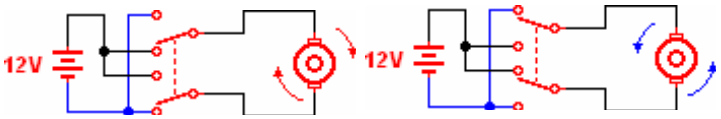


Figura 17. Cambio del sentido de giro de un motor empleando conmutadores o relés configurados formando un puente H.

Aunque esta última opción es una de las más prácticas, tiene sus inconvenientes ya que los relés suelen presentar problemas mecánicos y de desgaste, lo ideal sería disponer de un circuito un poco más sólido, quitando los relés y haciendo uso de transistores, estos últimos conectados en modo corte y saturación. A esta configuración se le llama puente en H.

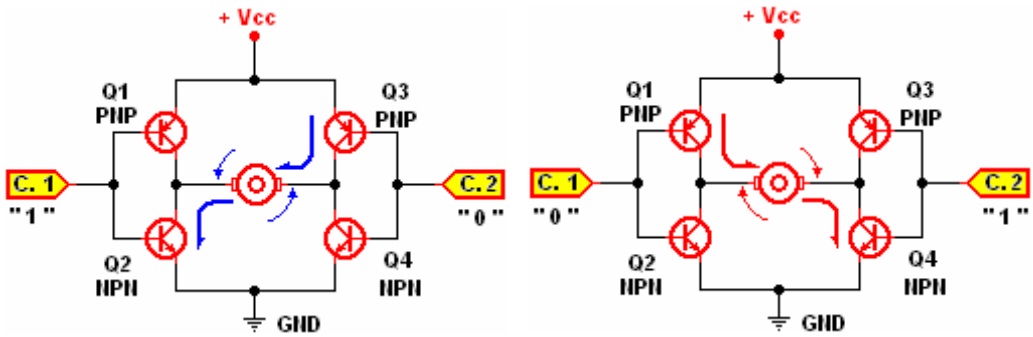


Figura 18. Cambio del sentido de giro de un motor empleando transistores configurados formando un puente H.

El puente en H está formado por cuatro transistores que trabajan en conmutación y se comportan como interruptores controlados por la señal que les lega a las entradas C.1 y C.2. Su funcionamiento es el siguiente:

- Cuando se activa la entrada C.1 a nivel alto y la entrada C.2 a nivel bajo los transistores Q3 y Q2 (NPN y PNP) entran en saturación simultáneamente, mientras que Q1 y Q4 están en corte por ser signo contrario (PNP y NPN respectivamente). En estas condiciones el motor gira en un sentido, por ejemplo en el contrario a las agujas del reloj (figura 18 izquierda).

- Cuando se invierte las señales de entrada, es decir C.1 a nivel bajo y C.2 a nivel alto. Los transistores que se saturan son Q1 y Q4, mientras que los que entran en estado de corte son Q2 y Q3. Esto hace que el motor gire en sentido contrario (figura 18 derecha).

Driver L293D

El L293D es un driver de 4 canales capaz de proporcionar una corriente de salida de hasta 0.6A por canal. Cada canal es controlado por señales de entrada compatibles TTL y cada pareja de canales dispone de una señal de habilitación que desconecta las salidas de los mismos. Este driver también integra ocho diodos que lo protegen frente a los picos de fuerza contraelectromotriz producidos por las cargas inductivas de las bobinas, de los motores en el momento de la conmutación. La figura 19 describe como conectar el driver L293D con el resto de componentes que forman el robot.

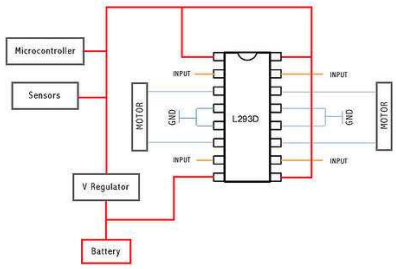


Figura 19. Driver L293D en el conjunto del sistema.

He optado por el driver L293D por las siguientes razones:

- Bajo coste.
- Simplicidad de montaje al incorporar los diodos de protección.
- Protección térmica frente a sobrecarga de los motores.
- Limitación de la intensidad entregada a los motores (0.6A) que impide que la tensión entregada por la batería se situé por debajo de los 4V, situación que comprometería la alimentación del microcontrolador.
- La caída de tensión provocada por el driver es de 2.6V. Con lo que aplicando a la entrada una tensión de 6V, tendremos 3.4V a la salida. Esta es una tensión muy cercana a los 3V de tensión nominal necesarios para alimentar los motores FA-130RA.

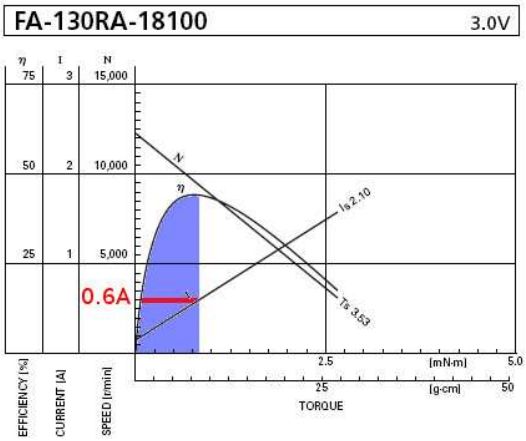


Figura 20. Rendimiento del motor con corriente limitada.

Acto seguido analizaré como influye la limitación de corriente entregada por los drivers en el rendimiento del motor.

Como se puede ver en la gráfica de la figura 20, con el límite de corriente consumida estaremos forzando al motor a trabajar hasta la zona de máxima eficiencia (zona azul). Otra repercusión es que el torque, no superará los 0,88 mN*m, esto provocará que las ruedas no derrapen. En contrapartida, se perderá un poco en aceleraciones.

¿Cómo podemos calcular si una rueda perderá tracción?

Lo primero es calcular la fuerza normal que ejercerá la pista sobre cada rueda:

$gravedad = 9.81 m/s^2$

$Peso = 0.55 Kg$

$Fnormal = 9.81 m/s^2 * 0.55 Kg = 5.39 N$

$FnormalEnRueda = 5.39 N / 4 ruedas = 1.35 N$

Acto seguido calculamos que cantidad de la fuerza normal, puede ser trasmitida, multiplicándola por el coeficiente de fricción de la rueda.
1.35N*0.8=1.08N
La rueda patinará cuando la fuerza aplicada por la rueda, superé los 1.08N.

Ahora calcularemos si el motor superará esta fuerza. Del análisis de la gráfica de la figura 20, podemos extraer que el torque no superará los 0.875mN*m.

Sabiendo que el factor de reducción de la caja reductora es de 12.7:1 y que el diámetro de la rueda es de 56mm, podemos calcular la fuerza aplicada por cada rueda a la pista.

$FuerzaAplicadaPista = 0.875 mN \cdot m / ((56 mm / 2) / 12.7) = 0.4 N$

Como la fuerza aplicada por la rueda sobre la pista, no supera a la fuerza normal de la pista sobre la rueda (0.4N<1.08N) la rueda no patinará

Control de velocidad PWM

La velocidad de un motor de corriente continua depende del valor medio de la tensión en sus extremos.

El sistema más utilizado para controlar la velocidad de un motor DC de pequeña potencia es mediante la modulación por ancho de pulso PWM (*Pulse Width Modulation*) de una señal cuadrada TTL, como muestra la figura 21. Bajo el control PWM, el motor gira a una velocidad determinada por la media del nivel de la señal cuadrada.

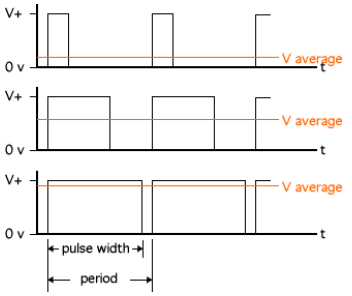


Figura 21. Control de velocidad PWM.

La tensión continua media entregada al motor se controla manteniendo la frecuencia constante, y variando el tiempo que la señal permanece en alto, es decir, variando el ciclo de trabajo (*duty cycle*). Así, si el ciclo de trabajo es del 50% se suministra al motor una tensión media del 50%, con un ciclo de trabajo del 20% sólo una quinta parte de la tensión máxima es suministrada a la carga.

Coste de los drivers

Articulo	Unidades	Precio por unidad
Driver L293D	2	2.89 €
	Total	5.78 €

Scanner lineal

El scanner lineal es un sensor mediante el cual el robot puede conocer su posición con respecto a las líneas negras que están marcadas en la pista. Este sensor constituye la entrada de datos al microcontrolador.

En el caso del AIIV, cada uno de los dos scanner están formados por tres sensores CNY70. El CNY70 es un sensor óptico reflexivo con salida a transistor, fabricado por Vishay Telefunked Semiconductors. Tiene una construcción compacta donde el emisor de luz y el receptor se colocan en la misma dirección para detectar la presencia de un objeto por medio del empleo de la reflexión del haz de luz infrarroja IR sobre el objeto, como puede verse en el afigura 22. La longitud de onda de trabajo es 950nm. El emisor es un diodo LED infrarrojo y el detector consiste en un fototransistor. La distancia del objeto reflejado debe estar entre los 5 y 10mm.

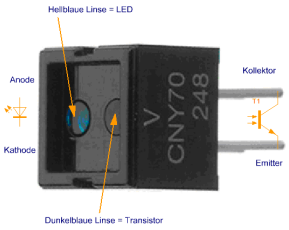


Figura 22. Sensor reflexivo CNY70.

Para polarizar correctamente estos sensores, es necesario una resistencia de 220 ohmios y otra de 4K7 ohmios, conectados como se muestra en la figura 23. Este esquema eléctrico representa la polarización de uno de los tres sensores que forman cada uno de los dos scanner que equipa el AIIV. En el punto rotulado con la palabra salida, se encuentra la tensión entregada por el sensor, esta es proporcional a la radiación infrarroja reflejada por la superficie de la pista.

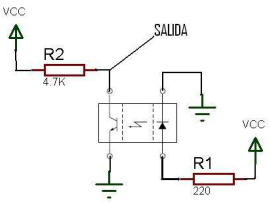


Figura 23. Polarización CNY70.

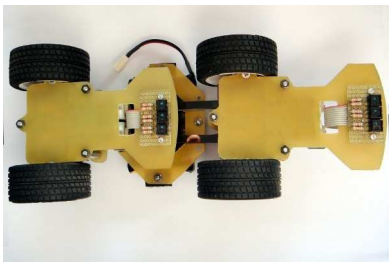


Figura 24. Ensamblado del scanner.

He optado por la configuración de tres sensores CNY70 colocados en línea, pues son el número suficiente para cubrir toda la línea negra que se debe seguir. Colocar más sensores solo habría encarecido y complicado el robot innecesariamente. En la figura 24 se muestra la ubicación de los dos scanners. La posición que ocupan los sensores está inspirada en el sistema de guiado de misiles y cohetes, que a su vez siguen el principio del péndulo invertido.

Las salidas de los fototransistores de los sensores CNY70 se acoplan a tres entradas analógicas del microcontrolador. Una función en el firmware del microcontrolador, será la encargada de descodificar estas señales y convertirlas en un valor de aproximación, que indique al AIIV el grado de aproximación a la línea negra.

Coste scanner lineal

Articulo	Unidades	Precio por unidad
CNY70	6	1.07 €
	Total	6.42 €

Microcontrolador

Recibe también el nombre de Unidad Central de Proceso (UCP ó CPU) y consta de la unidad central y el camino de datos. La primera es la sección encargada de interpretar las instrucciones del programa y generar las señales de control para su ejecución. El camino de datos está basado en una ALU y tiene la misión de realizar las operaciones lógicas y aritméticas que implican las instrucciones.

La UCP a su vez esta conectada con la memoria. En el caso de los microcontroladores no se sigue la arquitectura Von Neuman, sino que se sigue la arquitectura Harvard, en la que se dispone de mapas y buses independientes para las instrucciones y para los datos. Con esta arquitectura se permite el acceso en paralelo a los datos y a las instrucciones, así como la adecuación de las características de cada memoria a los requisitos de su contenido (RAM, EPROM, FLASH).

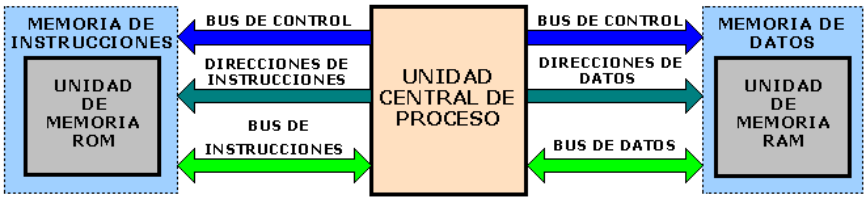


Figura 24. Arquitectura Harvard.

Los procesadores de los modernos microcontroladores responden a la arquitectura RISC (Computadores de Juego de Instrucciones Reducido), que se caracteriza por tener un juego de instrucciones muy sencillas y poco numeroso, que se ejecutan en la mayoría de los casos en un solo ciclo.

El PIC 16F876

Para implementar el AIIV he optado por emplear dos microcontroladores PIC 16F876 fabricados por microchip.

Figura 25. PIC 16F876.

Seguidamente paso a enumerar las características han hecho decidirme por este modelo:

- Tensión mínima de alimentación de 4V, cuando su reloj funciona con un cristal de 4Mhz. Esta tensión es adecuada para trabajar con un robot alimentado a baterías.
- Memorias RAM de datos de 368 Bytes y una EEPROM de 256 Bytes, suficientes para almacenar los datos generados por los algoritmos genéricos o los modelos de la inteligencia basada en el conocimiento.
- 5 entradas conversoras A/D, de los cuales 3 son necesarios para las lecturas del scanner lineal.
- Un puerto serie USART que será empleado para monitorear el correcto funcionamiento del firmware, así como para cargar nuevas versiones del mismo mediante un bootloader.
- Dos generadores de señales PWM (CCP1 y CCP2), que serán necesarias para controlar la velocidad de los motores.

- Tres puertos con salidas digitales, las cuales será necesarias para gobernar el sentido de giro de los motores y los LEDs del display.
- Se trata de un microcontrolador de 8 bits de bajo coste.

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F873	4K	192 Bytes	128 Bytes
PIC16F876	8K	368 Bytes	256 Bytes

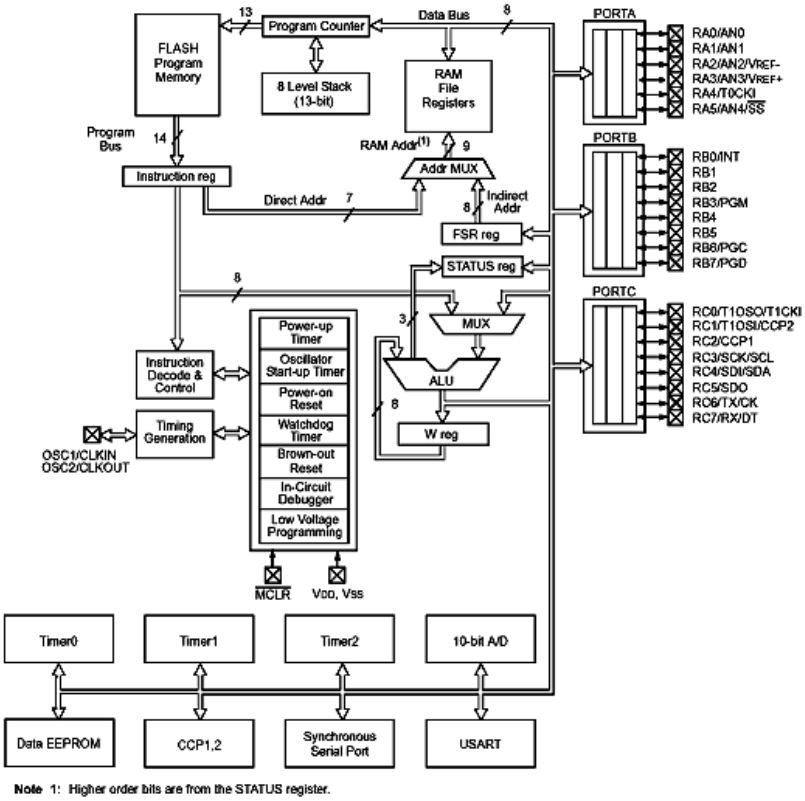


Figura 26. Diagrama de bloques del PIC 16F876.

Coste microcontroladores

Articulo	Unidades	Precio por unidad
PIC 16F876	2	7.78 €
Cristal de cuarzo 4Mhz	2	0.94 €
	Total	17.44 €

Lenguaje de programación

Los microcontroladores PIC se pueden programar en el ensamblador MPASM, pero he preferido emplear un lenguaje de más alto nivel como el C. Este nos permite un código más legible y rápido de codificar.

La versión de C específica para programar microcontroladores PIC se llama CSS C.
He empleado el IDE CSS y he compilado el código con el PCWH. Una vez compilado el código, se genera un archivo con extensión HEX que puede ser cargado en el microcontrolador.

Para grabar el código compilado en la memoria flash he empleado un programador TE-20, basado en el JDM. Este es uno de los más baratos (tienen un coste inferior a los 20 euros). Con él, se pueden programar los PIC 16X, 18X y EEPROMS 93CXX. Se comunica con el ordenador a través del puerto serie y no requiere de fuente de alimentación externa, ya que utiliza los +5v que llegan por el puerto.

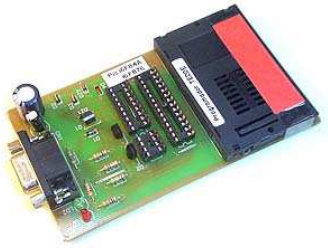


Figura 27. Programador TE-20

Como software de grabación he utilizado el IC-Prog, que puede bajarse libremente de la Web www.ic-prog.com .

Bootloader

¿Que es un Bootloader? Es un programa muy pequeño (256 bytes en este caso) que permite descargar programas al PIC usando únicamente el puerto serie, sin ningún hardware adicional.

¿Qué ventajas aporta? Solo es necesario utilizar un grabador de PIC una vez, para grabar el programa Bootloader. Una vez cargado el Bootloader en el PIC ya podemos descargar en él nuestros programas vía serie, sin necesidad de grabador, todas las veces que queramos. Como el hardware del AIIV incorpora puerto serie, no es necesario quitar el PIC de su zócalo: usando el puerto serie de nuestro proyecto hacemos la descarga. Este punto es extremadamente útil, pues debido al diseño del AIIV, es complejo acceder al microcontrolador para proceder a su extracción.

¿Como funciona? El Bootloader se carga en el final de la flash de programa del PIC y coloca el vector de interrupción de arranque apuntándolo. Arranca cuando alimentamos el procesador y espera un comando por el puerto serie. Si no lo recibe continua con la ejecución normal de nuestro programa. Si lo recibe comienza a recibir un programa por el puerto serie y a grabarlo en la flash de programa del PIC.

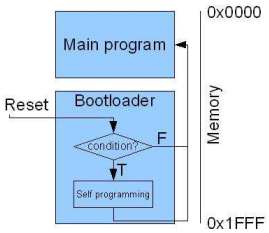


Figura 28. Funcionamiento del Bootloader

¿Que necesita? Es necesario tener el Bootloader configurado para nuestro hardware y cargado en el PIC, una conexión serie con el PC y el programa descargador (por ejemplo PICbootPlus).

¿Tiene algún inconveniente? Si. Es necesario modificar nuestros programas para que puedan trabajar con el Bootloader, ya que no podemos usar los últimos 256 bytes de memoria (en un PIC16F876 de 8K de memoria es el 3,1 % inutilizable). Se ha de añadir la siguiente línea al comienzo del código fuente:

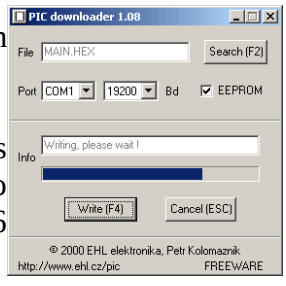


Figura 29. Funcionamiento del Bootloader

```
#org 0x1F00, 0x1FFF void loader16F876(void) {} //protect bootloader code for the 8k
```

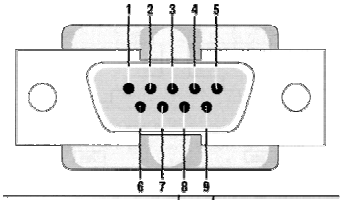
Comunicaciones por puerto RS-232

Para este proyecto, será necesaria la recogida de datos en un ordenador personal para su posterior estudio, por lo que será necesario dotar al sistema de la capacidad de intercambiar información con un ordenador.

RS-232

Los puertos serie son accesibles mediante conectores. La norma RS-232 establece el conector DB-9, así como la forma de conectar un ordenador con un MODEM, por lo que aparecen en los conectores, más patillas de las estrictamente necesarias.

Cada una de las patillas del conector RS-232 tiene una función especificada por la norma. Hay unos terminales por los que se transmiten y reciben datos y otros que controlan el establecimiento, flujo y cierre de la comunicación.



Pin	Signal	Pin	Signal
1	Data Carrier Detect	6	Data Set Ready
2	Received Data	7	Request to Send
3	Transmitted Data	8	Clear to Send
4	Data Terminal Ready	9	Ring Indicator
5	Signal Ground		

Figura 30. Patillaje del conector DB-9.

Pero para comunicar un microcontrolador con un ordenador es suficiente con tres líneas:

- Línea de transmisión (TxD), pin 3.
- Línea de recepción (RxD), pin 2.
- Pin de masa (SG), pin 5.

La velocidad a la que pueden trabajar los puertos COM de un ordenador es normalmente 2400, 4800, 9600, 19200, 38400, 57600, 115200 Baudios. Para el proyecto AIIV es suficiente con 19200 baudios. Esta es la velocidad máxima se puede alcanza con el PIC16F876 con un reloj a 4Mhz.

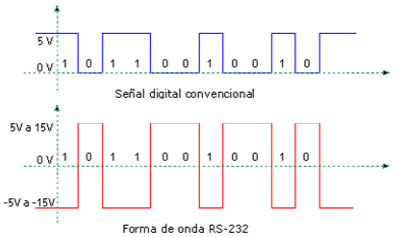


Figura 31. Niveles lógicos de tensión RS-232.

La figura 31 ilustra los niveles lógicos que debe cumplir una transmisión serie según la norma RS-232.

Estos valores de tensión proporcionan una amplia inmunidad al ruido, de gran utilidad cuando los cables deben pasar por zonas cercanas a elementos que generan interferencias eléctricas: motores, transformadores, equipos de comunicaciones, etc. Estos elementos, unidos a la longitud del cable, pueden hacer disminuir la señal hasta en varios voltios, sin que afecte adversamente al nivel lógico de la señal.

MAX232

El microcontrolador trabaja con niveles de tensión TTL y lógica positiva, a diferencia del puerto RS-232, que trabaja con lógica negativa y unos niveles de tensión más altos.

En el mercado hay muchos circuitos integrados que permiten la conversión entre niveles TTL y niveles RS-232. Entre ellos destaca el transceptor MAX232, fabricado por Dallas Semiconductor-MAXIM.

El MAX232 convierte los niveles RS-232 (cerca de +12 y -12V) a voltajes TTL (0 a 5V) y viceversa sin requerir nada más que una fuente de +5V. Basándome en el circuito eléctrico de la figura 32 he desarrollado un inteface que permite hacer la conversión RS232-TTL sin necesidad de una alimentación externa de +5V. Esta tensión le he obtenido del mismo puerto RS-232 del ordenador, más concretamente del pin 4 (DTR).

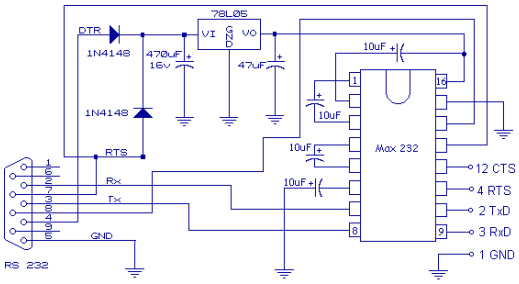


Figura 32. Esquema eléctrico del conversor RS232-TTL autoalimentado.

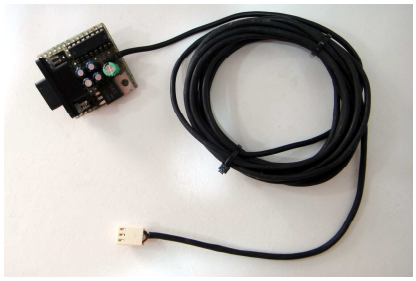


Figura 33. Interface RS232-TTL.

Coste interface RS232-TTL

Articulo	Unidades	Precio por unidad
MAX232	1	0.78 €
Regulador LM2940 5V	1	1.19 €
Conector hembra RS-232	1	0.80 €
	Total	2.77 €

Esquema eléctrico

Como resultado final del diseño del sistema de control, obtenemos un esquema eléctrico (figura 34). En este se han substituido los sensores ópticos CNY70 por potenciómetros, la razón de este cambio es debida a que este esquema puede simular el funcionamiento del sistema mediante la aplicación Proteus. Los potenciómetros adecuadamente configurados, simulan diferentes lecturas procedentes del scanner lineal.

El esquema eléctrico también presenta dos instrumentos de medida virtuales, proporcionados por la aplicación Proteus:

- Un Terminal virtual conectado a en los pines 17 y 18 (TX y RX respectivamente), del microcontrolador, en él podremos visualizar la salida los mensajes que salen por el puerto serie del micro.
- Un osciloscopio virtual conectado a los pines 12 y 13 del microcontrolador (CCP2 y CCP1 respectivamente), en el que podremos visualizar la anchura de los pulsos PWM.

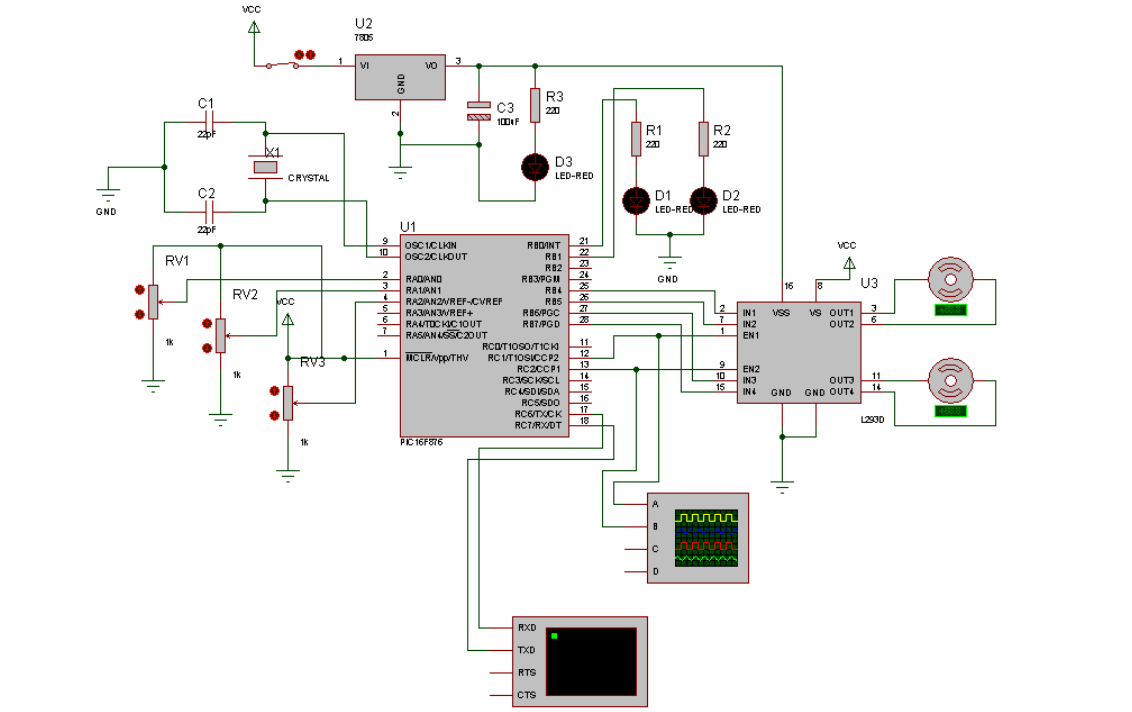


Figura 34. Esquema eléctrico del sistema de control.

Este circuito se monta por duplicado en sendas placas de prototipado. Cada una forma el sistema de control de cada uno de los dos agentes que componen el robot AIIV (agente A y agente B). El las figuras 35 y 36 puede verse la placa de control de uno de los agentes finalizada, así como su emplazamiento final en el conjunto del robot.

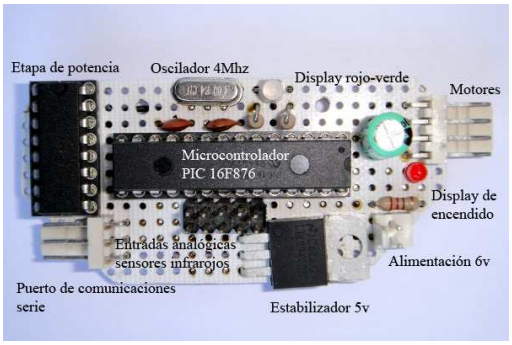


Figura 35. Circuito de control montado sobre una placa de prototipado.

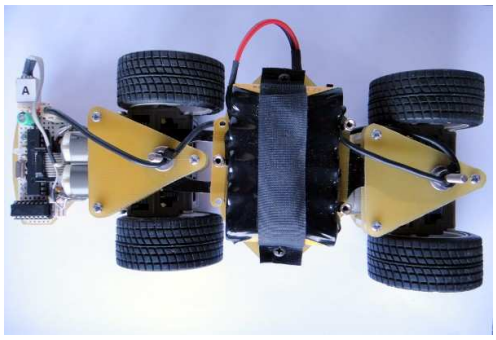


Figura 36. Robot AIIV completamente montado, donde puede verse el circuito de control del agente A. El agente B queda oculto por el paquete de baterías.

En este momento ya tenemos el prototipo totalmente ensamblado, con un peso total de 550 gramos.

Test del sistema electromecánico

Para comprobar el perfecto funcionamiento de todo el sistema electromecánico, he desarrollado un pequeño software. Su función es acelerar el vehiculo durante medio segundo, al 100% de potencia. Durante esta primera frase se hace una lectura de los

sensores de infrarrojos que componen el scanner lineal. Acto seguido el vehiculo se frena invistiendo el sentido de giro de los motores durante un cuarto de segundo, a una potencia del 25%. Todas las fases por las que pasa el programa son monitoreadas mediante mensajes mandados por el puerto RS-232.

Primero se he ejecutado el software de test hardware en el simulador Proteus.
El resultado ha sido satisfactorio, en la captura de pantalla de la figura 37, puede verse los mensajes mandados por el puerto serie, así como las señales PWM generadas por los CCP1 y CCP2.

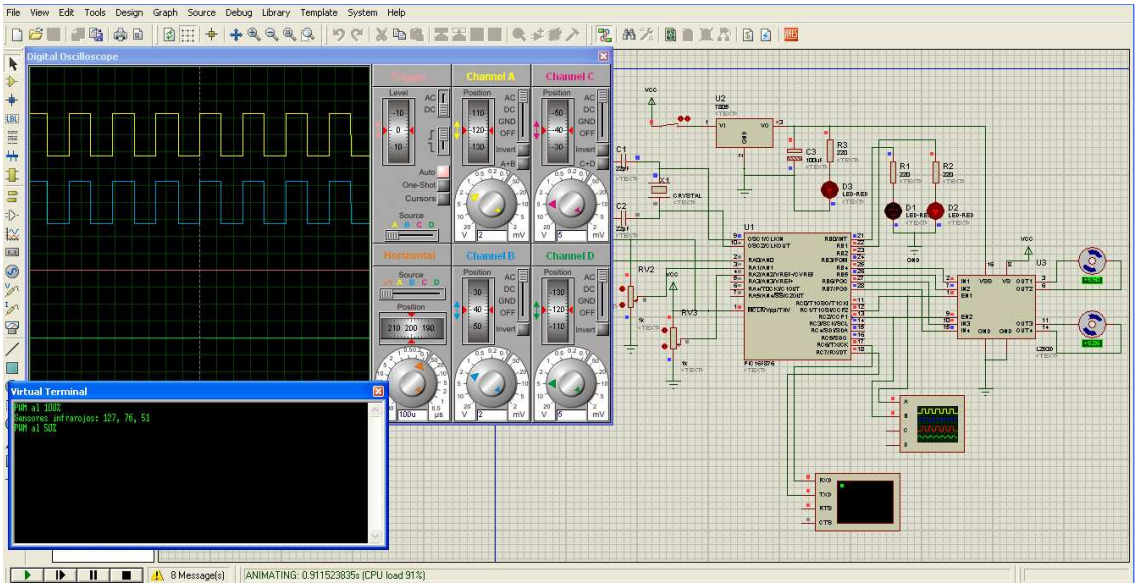


Figura 37. Sistema de control simulado mediante la aplicación Proteus. En este caso se trata de una variante del software de test hardware que configura los PWM al 50%.

Por ultimo podemos ver el código fuente del software de test hardware. No hay que preocuparse si alguna parte de este código no se entiende, pues en los siguientes capítulos, a medida que desarrolle el software que gobierne el AIIV, iré desgranando cada una de las sentencias que lo componen.

```
/************************************************************************/
/*                                                                      */
/* Proyecto final de carrera: AIIV                                      */
/* Área:                      Inteligencia Artificial                   */
/* Universiadad:              UOC                                       */
/* Auto:                      Pedro Amador Díaz                         */
/* Fecha:                     24/10/2011                                */
/*                                                                      */
/************************************************************************/

#include <16F876.h>
#FUSES XT,NOWDT /* Oscilador de cristal de cuarzo, no perro guardián */
#use delay(clock=4000000) /* Frecuencia del oscilador (cristal de cuarzo) */
#use rs232(baud=19200, xmit=pin_c6, rcv=pin_c7) /* Configuración RS-232 */
#byte puerto_a = 05 /* Dirección del puerto A */
#byte puerto_b = 06 /* Dirección del puerto B */
#byte puerto_c = 07 /* Dirección del puerto c */

/* Variables globales */
int array[3];
int sensor;

void main()
{
   set_tris_a(0xFF); /* Puerto A como entreada */
   set_tris_b(0x00); /* Puerto B como salida */
   set_tris_c(0x00); /* Puerto C como salida */
```

```
   /* Inicialización de los PWM */
   setup_timer_2(T2_DIV_BY_1,255,1);
   setup_ccp1(ccp_pwm);
   setup_ccp2(ccp_pwm);

   /* Motores izquierdo y derecho activados */
   bit_set(puerto_b,4);
   bit_clear(puerto_b,5);
   bit_set(puerto_b,6);
   bit_clear(puerto_b,7);

   /* Puertos A0 A1 A2 A3 A4 entradas analógicas */
   setup_adc( ADC_CLOCK_INTERNAL );
   setup_adc_ports( ALL_ANALOG );

   /* PWM motores izquierdo y derecho al 100% */
   set_pwm1_duty(255);
   set_pwm2_duty(255);

   /* Mensaje por terminal RS-232 y display led */
   printf ("PWM al 100%%\n\r");
   bit_set(puerto_b,0);
   bit_clear(puerto_b,1);

   /* Lectura de los tres sensores infrarrojos */
   sensor = 0;
   while (sensor<3)
   {
      set_adc_channel(sensor);
      delay_cycles(255);
      array[sensor]=read_adc();
      sensor++;
   }

   /* Mensaje por terminal RS-232 indicando los valores leidos por los sensores
      infrarrojos*/
   printf ("Sensores infrarojos: %u, %u, %u\n\r",array[0], array[1], array[2]);

   /* Espera de medio segundo durante el cual el robot se desplaza a máxima velocidad */
   delay_ms(500);

   /* Motores izquierdo y derecho en inversión de giro */
   bit_clear(puerto_b,4);
   bit_set(puerto_b,5);
   bit_clear(puerto_b,6);
   bit_set(puerto_b,7);

   /* PWM motores izquierdo y derecho al 50% */
   set_pwm1_duty(127);
   set_pwm2_duty(127);

   /* Mensaje por terminal RS-232 y display led */
   printf ("Freno motor\n\rPWM al 50%%\n\r");
   bit_set(puerto_b,1);
   bit_clear(puerto_b,0);

   /* Espera 1/4 de segundo */
   delay_ms(250);

   /* PWM motores izquierdo y derecho al 0% */
   set_pwm1_duty(0);
   set_pwm2_duty(0);

   /* Mensaje por terminal RS-232 y display led */
   printf ("PWM al 0%%\n\rFin del test\n\r");
   bit_clear(puerto_b,0);
   bit_clear(puerto_b,0);

}
```

Los efectos que provoca la ejecución del software de test sobre el robot AIIV pueden verse en el video Test1. En el puede verse al robot acelerar y acto seguido detenerse.

Figura 38. Video test1.

Diseño de un entorno de test

Para el diseño del circuito de test se ha tenido en cuenta la normativa del concurso Cosmobot, la cual suele ser la misma para todos los concursos celebrados en el territorio español. De esta normativa extraemos que el radio menor se una pista nunca será inferior a 40cm. Por otro lado disponemos de un ejemplo de pista de competición (figura 2). A partir de estos datos he elaborado dos pistas de test:

- Una pista circular, con un radio de 41cms
- Una pista que combina rectas y curvas de diferentes radios (no inferiores a 41 cms).

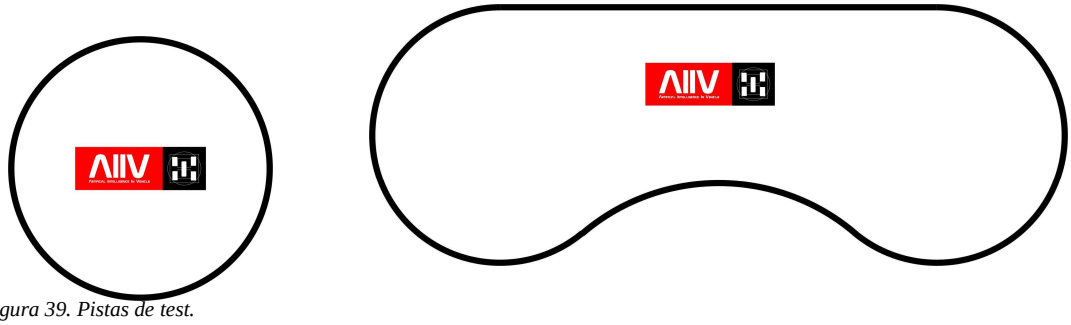


Figura 39. Pistas de test.

Al elaborar estas pistas de test he tenido especial cuidado en que el resultado sea de reducidas dimensiones, para que pueda ser empleada en espacios de reducidos, y a su vez no sea compleja de desplegar y almacenar.

La pista de mayores dimensiones nos permite poner a prueba algunas características del robot que la pista circular no. Estas son:

- Velocidad punta en recta.
- Curvas en ambos sentidos.
- Cambios buscos de dirección (chicane).

Firmware

Acto seguido describiré el software que será gravado en la memoria flash del microcontrolador y formará la lógica de control del robot.

Directivas del precompilador

La primera sección que compone el software del AIIV, son las directivas del precompilador CCS. A continuación paso a describirlas:

#include <16F876.h>
Permite incluir en nuestro programa uno o mas archivos (conocidos como header file) que posean extensión .h. Estos archivos contienen información sobre funciones, sus argumentos, el nombre de los pines de un modelo determinado de PIC o cualquier otra cosa que usemos habitualmente en nuestros programas. Esto permite no tener que escribir un montón de cosas cada vez que comenzamos un programa nuevo: basta con incluir el .h correspondiente.

#device ADC=8 /* Resolución del modulo A/D de 8 bits
Esta directiva informa al compilador que arquitectura de hardware utilizaremos, para que pueda generar código apropiado para la cantidad de RAM, ROM y juego de instrucciones disponibles. En este caso estamos informando al compilador que la función read_adc() retornara valores de 8 bits.

#fuses XT,NOWDT,PUT,NOPROTECT,BROWNOUT
Permite modificar el valor de los fuses del microcontrolador que estamos empleando. Los valores posibles dependen de cada microcontrolador en particular.

- Tipo de oscilador, cristal de cuarzo: XT.
- Wach Dog Timer desactivado: NOWDT.
- Protección de código desactivada: NOPROTECT.
  - Se protege el código para que no pueda ser leído mediante un grabador de microcontroladores.
- Brown Out Reset activado: BROWNOUT.
  - Se reinicia el microcontrolador si hay una bajada de tensión.
- Power Up Timer activado: PUT
  - Durante los primeros instantes, tras la puesta en marcha del microcontrolador, se espera los milisegundos necesarios para que la tensión de la alimentación y el oscilador se estabilicen.

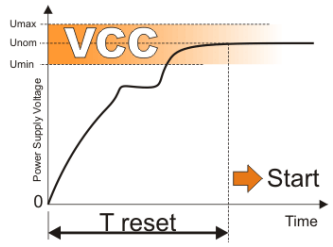


Figura 40. Power Up Timert.

#use delay(clock=4000000)
Esta directiva indica al compilador la frecuencia del procesador, en ciclos por segundo, a la vez que habilita el uso de las funciones DELAY_MS() y DELAY_US()

#use rs232(baud=19200, xmit=pin_c6, rcv=pin_c7)
Esta directiva le dice al compilador la velocidad en baudios y los pines utilizados para la I/O serie. Esta directiva tiene efecto hasta que se encuentra otra directiva RS232. Esta directiva habilita el uso de funciones tales como GETCH, PUTCHAR y PRINTF.

#byte puerto_a = 05
#byte puerto_b = 06
#byte puerto_c = 07
Permite crear una nueva variable de un Byte de tamaño, que es colocada en la memoria del PIC en la posición del byte x. Esta es una herramienta muy útil para acceder de una manera sencilla a los registros.

#define PWM 80
Define la etiquete PWM que será substituida en tiempo de compilación por el valor 80.

Scanner

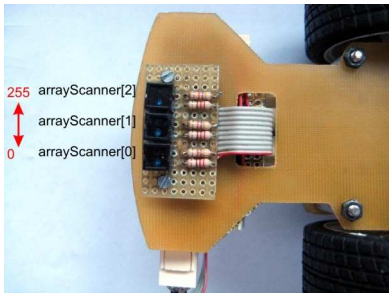


Figura 41. Scanner.

La primera operación a llevar a cabo es configurar el conversor A/D. Como ya hemos visto en el apartado dedicado al microcontrolador, este posee cinco entradas analógicas. Estas son seleccionadas mediante un circuito multiplexor y diseccionadas hacia el conversor A/D.

La función setup_adc_ports(ALL_ANALOG) configura las cinco patillas como entradas analógicas, estas son: pin2, pin3, pin4, pin5, pin7.

La función setup_adc(ADC_CLOCK_DIV_8) configura la frecuencia de trabajo del conversor A/D, que en nuestro caso sera la frecuencia del oscilador del microcontrolador dividida por 8.

Para ir seleccionando cada uno de los tres puertos analógicos que utilizamos por cada agente, se emplea la sunción set_adc_channel() a la que se le pasa como parámetro el número del puerto deseado. Hay que tener en cuenta realizar una pausa de cómo mínimo 10µs antes de proceder a tomar un valor valido (lectura). Este tiempo es necesario para que el multiplexor realice el cambio de puerto (canal).

Finalmente mediante la función read_adc() procedemos a tomar la lectura digital del conversor A/D. Las lecturas de los tres sensores de infrarrojos se guardan en la matriz arrayScanner[]. En la figura 40 puede verse los elementos de la matriz arrayScanner[] asociados a la lectura procedente de cada uno de los tres sensores, que componen cada scanner. Como el conversor A/D es de 8 bits, los valores devueltos por este están en el rango de 0-255. Debido a la polarización empleada en los sensores infrarrojos, el 0 corresponde al negro y 255 al blanco. Pero para realizar los algoritmos del AIIV es más adecuado es caso contrario, por este motivo se han invertido los valores devueltos por el conversor mediante la expresión 255-read_adc().

Calibrado

Los valores almacenados en arrayScanner[] deben ser calibrados. Esto es debido a la tolerancia de los componentes electrónicos que forman el scanner. Para tal cometido se ha creado la función sensorsCalibration(), la cual tiene como parámetro de entrada un puntero a la matriz arrayScanner[]. Al ser un puntero, la función tiene la capacidad de modificar los valores de la matriz de entrada. Para el ajuste de este proceso he empleado una carta compuesta de dos zonas, una blanca y otra negra. El robot se coloca en cada una de las dos zonas y se toman las lecturas de los scanners de los dos agentes en cada una de ellas. Estas son las lecturas obtenidas:

Zona blanca:
Agente A: arrayScanner[0]=7, arrayScanner[1]=7, arrayScanner[2]=8
Agente B: arrayScanner[0]=7, arrayScanner[1]=7, arrayScanner[2]=9

Zona negra:
Agente A: arrayScanner[0]=228, arrayScanner[1]=213, arrayScanner[2]=228
Agente B: arrayScanner[0]=232, arrayScanner[1]=232, arrayScanner[2]=242

Cada agente tendría una calibración diferente de su scanner y eso conllevaría tener que cargar una versión diferente del software para cada uno. Para evitarlo he realizado la media de las dos lecturas y he conseguido unos parámetros de calibración, que si bien no son los óptimos para cada agente si son suficientes y evitan tener que tener dos versiones diferentes del software.

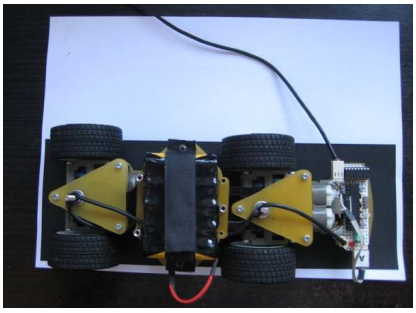


Figura 42. Calibrado del scanner.

Parámetros de calibración unificados:

Zona blanca:
arrayScanner[0]=7, arrayScanner[1]=7, arrayScanner[2]=9

Zona negra:
arrayScanner[0]=230, arrayScanner[1]=222, arrayScanner[2]=235

Expresión de calibrado:

```
arrayScanner[sensor]=((float)(arrayScanner[sensor]-maxMinScanner[sensor][0])/
    (maxMinScanner[sensor][1]-maxMinScanner[sensor][0]))*255
```

Calculo de la posición

Antes de realizar el cálculo de la posición, se determina si las lecturas han superado cierto umbral. En nuestro caso el umbral se ha fijado en el 50%. Se ha puesto esta comprobación para descartar cálculos de posición con lecturas en las que la señal sea tan baja que el ruido de estas pueda influir negativamente.

Para calcular la posición del robot respecto a la línea negra, he creado la función linePosition(). Esta tiene como parámetro de entrada un puntero a la matriz arrayScanner[] con sus valores ya calibrados. Su salida es un valor en el rango 0-255, siendo 128 centrado. Estos valores corresponden con las posiciones indicadas en la figura 40 (valores en rojo).

Expresión de cálculo de la posición:

```
(((0*(int32)arrayScanner[0])+(255*(int32)arrayScanner[1])+(510*(int32)arrayScanner[2]))/
        ((int32)arrayScanner[0]+(int32)arrayScanner[1]+(int32)arrayScanner[2]))/2;
```

Esta no es más que una media ponderada.

Actuadores (motores)

Para el control de los actuadores he empleado una función para cada uno de los dos motores. Estas son motor1() y motor2(). Como parámetro de entrada se emplea un único valor que determina la potencia aplicada y el sentido de giro. La potencia viene determinada por el valor absoluto del parámetro de entrada, y el sentido del giro por su signo.

De esta se puede controlar cada uno de los actuadotes con un byte.
127: Máxima potencia, giro sentido horario.
64: 50% potencia, giro horario.
0: Motor parado.
-64: 50% potencia, giro antihorario.
-127 Máxima potencia, giro sentido antihorario.

En la tabla anterior no he puesto todos los valores posibles, pero se sobreentiende que todo valor entre -127 y 127 es posible.

El sentido de giro se controla mediante las salidas digitales que a su vez atacan a las entradas de los puentes H (chip L293D), mientras que la potencia aplicada se controla mediante los generadores de pulsos PWM que también se aplican a los puentes en H.

Motor1 giro sentido horario:
Puerto b_4 = 0
Puerto b_5 = 1

Motor1 giro sentido antihorario:
Puerto b_4 = 1
Puerto b_5 = 0

Motor2 giro sentido horario:
Puerto b_6 = 1
Puerto b_7 = 0

Motor2 giro sentido antihorario:
Puerto b_6 = 0
Puerto b_7 = 1

Ejemplos de control de la salida digital:

Salida 0 lógico en el puerto digital b_6:
bit_clear(puerto_b, 6)

Salida 1 lógico en el puerto digital b_6:
bit_set(puerto_b, 6)

Ejemplos de control del generador de pulsos PWM:

PWM del motor1 al 25%:
set_pwm1_duty(64)

PWM del motor2 al 50%:
set_pwm2_duty(127)

Llegados a este punto el lector talvez se puede haber dado cuenta de un detalle. El valor absoluto del parámetro de entrada de la función de control de los actuadotes solo puede alcanzar un valor máximo de 127 que correspondería a una potencia del 50%, cuando en las especificaciones de la sunción este valor debería corresponder con una potencia aplicada del actuador del 100%. La forma de corregir esta desviación es sencillamente, multiplicar por dos el valor absoluto de el parámetro de entrada a la función de control de actuadores.

Comportamiento reactivo

Llegados a este punto ya podemos implementar un comportamiento reactivo o inteligencia reactiva. Este tipo de comportamiento se adapta mejor que el deliberativo, al tipo de problema que tenemos que resolver (adaptación al entorno en tiempo real), ya que las respuestas del sistema son mucho más rápidas.

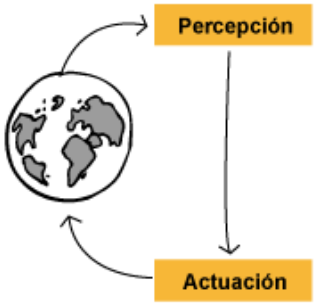


Figura 43. Comportamiento reactivo.

Algunas de las propiedades del comportamiento reactivo son:

- El robot está inmerso en el mundo real y no requiere operar con representaciones abstractas de éste, sino con el mundo mismo. A esta propiedad se le denomina localización.

- Un robot presenta características físicas que se deben considerar en sus acciones en el mundo. Esta es la propiedad llamada corporeidad.

- Comportamiento emergente: la "inteligencia" surge de la interacción con el entrono (mundo). No es una propiedad del robot o del entorno por separado.

- Propiedad "puesta a tierra": la información que el robot utiliza se recoge directamente del mundo y no a través de símbolos como en la IA original.

- Propiedad de "dinámica ecológica": un agente físico no reside en el vacío sino que está inmerso en un entorno que varía continuamente en el espacio y en el tiempo. Estas variaciones dinámicas excepto para entornos altamente estructurados, son difíciles o incluso imposibles de caracterizar.

- Estos sistemas actúan de modo más rápido porque requieren menos niveles de procesamiento de la información y se ejecutan de forma y no sincronizada, considerando módulos de comportamiento que requieren mucho menos gasto de computación.

- Son capaces de tratar situaciones no previstas porque confían muchos más en el entorno como fuente de información y de determinación de sus acciones.

- Tienen menos problemas de representación de los estímulos sensoriales porque no intentan mantener representaciones objetivas del entorno.

- Finalmente, no tienen problemas de explosión exponencial combinatoria porque no emplea métodos de búsqueda tradicionales.

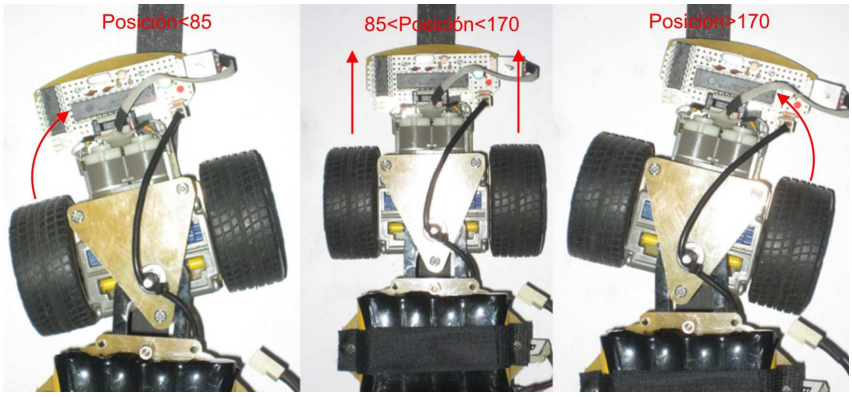


Figura 44. Tres reglas que conforman el comportamiento reactivo de cada uno de los dos agentes del AIIV.

El comportamiento reactivo se puede obtener a partir de tres reglas básicas:

- Si posición inferior a 85, frenar rueda derecha.
- Si posición mayor de 85 y menor de 175, las dos ruedas a máxima potencia.
- Si posición mayor a 175, frenar rueda izquierda.

Las reglas deben ser evaluadas en el orden que aparecen en la lista anterior.

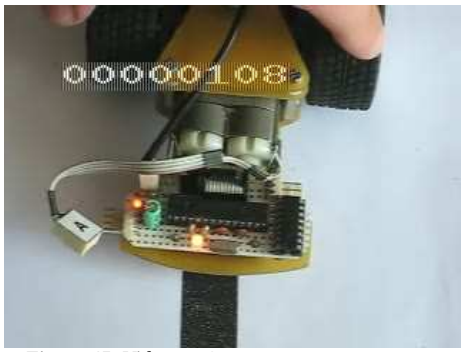


Figura 45. Video test2.

En el video test2 puede verse como el agente A pasa por los tres estados posibles:

- Giro a la derecha
- Seguir recto
- Giro a la izquierda

Estos estados son indicados por el LED (display) que cambia de color.

Figura 46. Video test3.

En el video test3 ya podemos ver el robot evolucionando sobre la pista de test circular. Este test demuestra como la tesis que defendía al principio del proyecto, era correcta. Los dos agentes siguen un comportamiento reactivo y realizan su tarea de forma independiente, pero a su vez surge la inteligencia emergente. Esto es debido a que la morfología del robot forma una granja de robots. Este fenómeno es el mismo que puede apreciarse en comunidades de hormigas o en redes neuronales, en la que la funcionalidad del conjunto es superior a la suma de las funcionalidades de los elementos que forman parte.

En este video también puede apreciarse como la reacción de los agentes A y B es brusca. Este comportamiento pasará a ser más suave, con la incorporación del algoritmo de control PID, el cual modulará la potencia que se aplica a los motores.

Control PID

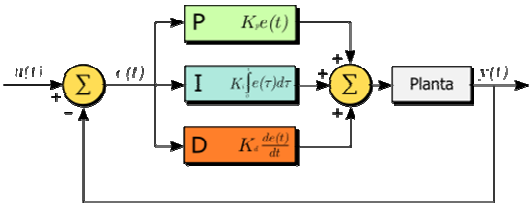


Figura 47. Diagrama de en bloques de un controlador PID.

Un PID (Proporcional Integral Derivativo) es un mecanismo de control por realimentación que se utiliza en sistemas de control industriales. Un controlador PID corrige el error entre un valor medido y el valor que se quiere obtener calculándolo y luego sacando una acción correctora que puede ajustar al proceso acorde. El algoritmo de cálculo del control PID se da en tres parámetros distintos: el proporcional, el integral, y el derivativo. El valor Proporcional determina la reacción del error actual. El Integral genera una corrección proporcional a la integral del error, esto asegura que aplicando un esfuerzo de control suficiente, el error de seguimiento se reduce a cero. El Derivativo determina la reacción del tiempo en el que el error se produce. La suma de estas tres acciones es usada para ajustar al proceso vía un actuador, como puede ser los motores que mueven las ruedas del robot.

Para el correcto funcionamiento de un controlador PID que regule un proceso o sistema se necesita, al menos:

1. Un sensor, que determine el estado del sistema, en nuestro caso el scanner lineal.

2. Un controlador, que genere la señal que gobierna al actuador.

3. Un actuador, que modifique al sistema de manera controlada, en nuestro caso los motores que gobiernan las ruedas motrices.

El sensor proporciona una señal digital al controlador, la cual representa el punto actual en el que se encuentra el proceso o sistema. En nuestro caso esta señal procede del scanner lineal y es un valor de 0 a 255, siendo 128 el estado robot centrado en la línea negra.

El controlador lee una señal externa que representa el valor que se desea alcanzar. Esta señal recibe el nombre de punto de consigna (o punto de referencia), la cual es de la misma naturaleza y tiene el mismo rango de valores que la señal que proporciona el sensor. En nuestro caso esta señal consigna es el valor 128 que representa el estado robot centrado en la línea negra.

El controlador resta la señal de punto actual a la señal de punto de consigna, obteniendo así la señal de error, que determina en cada instante la diferencia que hay entre el valor deseado (consigna) y el valor medido. La señal de error es utilizada por cada uno de los 3 componentes del controlador PID. Las 3 señales sumadas, componen la señal de salida que el controlador va a utilizar para gobernar al actuador. La señal resultante de la suma de estas tres se llama variable manipulada.

Las tres componentes de un controlador PID son: parte Proporcional, acción Integral y acción Derivativa. El peso de la influencia que cada una de estas partes tiene en la suma final, viene dado por la constante proporcional, la constante integral y la constante

derivativa, respectivamente. Se pretenderá lograr que el bucle de control corrija eficazmente y en el mínimo tiempo posible los efectos de las perturbaciones.

Implementación en código c:

```
proportional = (signed int)position-128; /* Calculo de la componente proporcional */
integral = integral + proporcional; /* Calculo de la componente integral */
derivative = proportional - last_proportional; /* Calculo de la componente derivada */


power = PWM - abs(((float)proportional*1.0) + ((float)integral*1.0) + ((float)derivative*1.0));
last_proportional = proportional;
```

Las pruebas realizadas tras incorporar el control PID, no han sufrido una mejora significativa. Esto es debido a que aun falta el ajuste (sintonía) de los pesos de las componentes proporcional, integral y derivativa.

Ajuste controlador PID

La práctica ha demostrado que con dimensiones reducidas de scanner, el robot se vuelve inestable fácilmente. Esto se traduce en que muchos de los individuos de una población generada por un sistema evolutivo serian inviables. Estos individuos inviables harían salir al robot de la pista, con la consiguiente perdida de tiempo al restablecer el experimento y el riesgo de estrellar el robot con algún elemento externo a la pista.

Simulación

La solución ha pasado por emplear un modelo de simulación para buscar los limites para los cuales el controlador PID se vuelve inestable y evitar que el algoritmo genético cree individuos con alguno de estos genes prohibitivos.

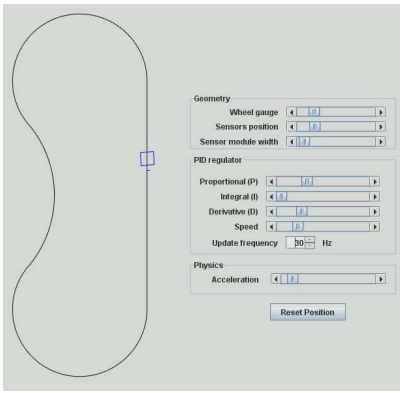


Figura 48. Simulador de robots sigue líneas.

El código del simulador se puede bajar de la dirección http://www.ostan.cz/LineFollowerSimulator.

Al parecer este código no posee ninguna licencia restrictiva, con lo que he realizado un par de cambios en el mismo. Por un lado he cambiado el método RobotSandBox() de la plase RobotSandbox, para que la pista sea idéntica a la que he diseñado para las pruebas (escala 1:4). Por otro lado he cambiado la clase SettingsJPanel, para que por defecto los parámetros de la simulación se ajusten a las dimensiones y configuración del robot AIIV. El código es fácilmente importable en el IDE NetBeans y ejecutándolo se puede ver la importancia de las dimensiones del scanner.

Los pesos óptimos obtenidos mediante la simulación son:

- Proporcional = 0.7
- Integral = 0.0
- Derivativa = 0.02

Como la componente integral es nula, he preferido eliminar su cálculo en el algoritmo del PID, ya que este se realiza en como flotante, y para el microcontrolador es un gasto innecesario de tiempo de procesado. Recordaré que el microcontrolador PIC16F876 no posé coprocesador matemático, y las operaciones en coma flotante son especialmente costosas en tiempo de procesador.

Aplicando estos pesos al control PID, el robot mejora su comportamiento considerablemente.

En el video test4 se puede ver como se ha corregido las oscilaciones del robot, ganando estabilidad. A su vez se ha podido aumentar su velocidad hasta 0.7 m/s.

Figura 49. Video test4 (test final).

Algoritmo genético

El cometido del AG es el de buscar una combinación óptima de los coeficientes Kp, Ki y Kd (proporcional, integral y derivativo) del control PID.

Codificación cromosoma

Por las pruebas realizadas con el simulador y las pruebas empíricas en pista, he establecido los siguientes límites en los coeficientes.

Coeficiente	Límite inferior	Límite superior
Kp	0.2	0.4
Ki	0.2	0.4
Kd	0.00	0.02

Cada coeficiente corresponde a un gen y estos se codifican de la siguiente forma:

Genotipo	Resolución	Obset	Pendiente
Kp	0 - 200	200	0.001
Ki	0 - 200	200	0.001
Kd	0 - 200	0	0.0001

Con esta codificación es suficiente con un byte para almacenar cada gen.

Función de evaluación

Acto seguido pasaré a explicar como se ha calculado el valor de calidad.

$$calidad = \frac{\sum_{ciclosdevida}^{0} |error|}{ciclosdevida}$$

Como se puede ver en la formula, el valor calidad corresponde a la media de los valores absolutos del error del sistema de control PID, durante la vida del individuo. Este valor tiene un rango de 0 a 127 (siendo 0 la mejor calidad y 127 la peor).

La calidad nos indica el grado de adaptación del individuo al medio y servirá, en la etapa de selección, para escoger los individuos más aptos. Por este motivo, una vez se ha finalizado la etapa de evaluación, se guarda junto al cromosoma evaluado, con lo que cada individuo necesita 4 bytes al ser almacenado. Como queremos tener un histórico del funcionamiento del AG, los cromosomas se guardan en una memoria eeprom, que en nuestro caso tiene una capacidad de 256 bytes y que nos limita a un máximo de 64 individuos.

Vida de cada individuo

Respecto al tiempo de vida de cada individuo, lo he fijado en 2 segundos por la limitación de tiempo que hay que aplicar a cada experimento evolutivo. Con este límite de vida, se evalúan 55 individuos en unos 2 minutos. En la práctica este tiempo se alarga un poco más, debido al tiempo extra que generan las diferentes etapas del AG.

De lo visto hasta ahora tengamos en cuenta una consideración muy importante. Si quisiéramos evaluar todas las combinaciones de genes posibles, tendríamos que explorar una superficie de 200*200*200= **8000000 individuos**, que a 2 segundos de vida cada uno, haría necesario un experimento de 185 días. Es aquí donde el AG nos demuestra todo su potencial, pues con mínimo de evaluaciones, consigue encontrar el individuo optimo.

Máquina de estados del AG

El diagrama de flujo de la figura 50 ilustra muy bien el funcionamiento de un algoritmo genético.

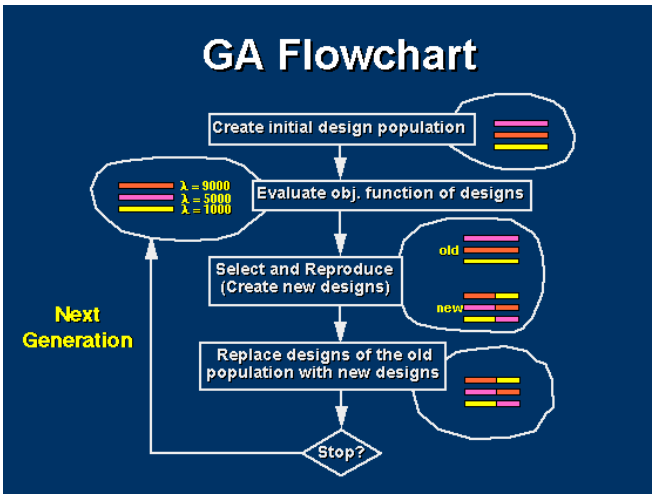


Figura 50. Diagrama de flujo del algoritmo genético.

En una primera etapa se genera una población inicial que en el caso que nos ocupa es de 5 individuos, estos surgen de sucesivas mutaciones de un cromosoma semilla.

En la primera iteración del bucle, no se pasa por la etapa de valuación ni selección y se llega directamente a la de reproducción.

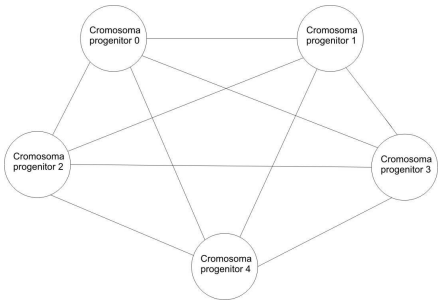


Figura 51. Cruce de cromosomas.

La reproducción consisten en cruzar los 5 individuos mejor adaptados entre si, de forma que obtenemos 10 individuos más, como indica la figura 51.

Una vez tenemos 15 individuos en la población, se evalúa cada uno durante 2 segundos y se les otorga una calidad que denota con que bondad se han adaptado al medio.

Más tarde, en la etapa de selección, se marcan los 5 individuos mejor adaptados al medio, estos son los que tendrán derecho a reproducirse en el ciclo siguiente. Tengamos en cuenta que la selección no se realiza solo sobre los descendientes, sino que también entran en ella los 5 progenitores de estos (otro método es hacer la selección solo sobre la población descendiente).

Este ciclo se reitera 5 veces, desarrollándose 5 generaciones.

Resultados del AG sobre del prototipo AIIV

El AG se ha testeado sobre el circuito circular, y el comportamiento del robot puede verse en el video llamado test5. En dicho video se apreciarse como algunos individuos, que se adaptan mal al medio, hacen oscilar al robot durante los dos segundos de su vida (ejemplos en los fotogramas 2300 y 4120).

Figura 52. Test5, AG.

Pero los datos más interesantes son los que se extraen del volcado de la memoria eeprom. Este proceso se puede llevar a cabo durante los primeros 10 segundos del siguiente experimento.

Agente A

Cromosoma 1= 154, 100, 100, 0
Cromosoma 2= 154, 96, 100, 0
Cromosoma 3= 154, 96, 107, 0
Cromosoma 4= 154, 96, 70, 0
Cromosoma 5= 154, 140, 70, 0
Cromosoma 6= 83, 98, 100, 33
Cromosoma 7= 154, 120, 103, 27
Cromosoma 8= 154, 77, 85, 19
Cromosoma 9= 196, 120, 85, 12
Cromosoma 10= 154, 96, 187, 10
Cromosoma 11= 154, 96, 94, 33
Cromosoma 12= 154, 118, 85, 19
Cromosoma 13= 66, 96, 88, 8
Cromosoma 14= 154, 118, 143, 14
Cromosoma 15= 154, 113, 70, 14
Cromosoma 16= 175, 108, 114, 16
Cromosoma 17= 154, 104, 13, 22
Cromosoma 18= 125, 96, 137, 18
Cromosoma 19= 154, 144, 165, 16
Cromosoma 20= 17, 116, 77, 11
Cromosoma 21= 131, 108, 63, 18
Cromosoma 22= 175, 119, 189, 12
Cromosoma 23= 110, 16, 79, 27
Cromosoma 24= 154, 115, 79, 6
Cromosoma 25= 129, 107, 115, 14
Cromosoma 26= 175, 186, 136, 55
Cromosoma 27= 154, 105, 83, 32
Cromosoma 28= 110, 96, 9, 10
Cromosoma 29= 85, 2, 132, 12
Cromosoma 30= 175, 154, 82, 94
Cromosoma 31= 131, 156, 86, 100
Cromosoma 32= 106, 1, 81, 19
Cromosoma 33= 26, 105, 83, 51
Cromosoma 34= 179, 115, 78, 46
Cromosoma 35= 41, 106, 113, 27
Cromosoma 36= 132, 32, 98, 70
Cromosoma 37= 154, 45, 133, 14
Cromosoma 38= 110, 138, 137, 8
Cromosoma 39= 178, 106, 132, 6
Cromosoma 40= 46, 105, 44, 14
Cromosoma 41= 88, 96, 19, 20
Cromosoma 42= 113, 106, 43, 30
Cromosoma 43= 110, 105, 86, 21
Cromosoma 44= 85, 198, 78, 29
Cromosoma 45= 41, 126, 82, 17
Cromosoma 46= 82, 101, 159, 14
Cromosoma 47= 154, 112, 133, 13
Cromosoma 48= 110, 96, 190, 7
Cromosoma 49= 132, 117, 130, 8
Cromosoma 50= 166, 80, 105, 6
Cromosoma 51= 182, 101, 110, 7
Cromosoma 52= 144, 122, 66, 15
Cromosoma 53= 25, 105, 83, 14
Cromosoma 54= 132, 126, 44, 9
Cromosoma 55= 88, 117, 55, 7

Agente B

Cromosoma 1= 154, 100, 100, 0
Cromosoma 2= 154, 96, 100, 0
Cromosoma 3= 154, 96, 107, 0
Cromosoma 4= 154, 96, 70, 0
Cromosoma 5= 154, 140, 70, 0
Cromosoma 6= 83, 98, 100, 19
Cromosoma 7= 154, 120, 103, 15
Cromosoma 8= 154, 77, 85, 16
Cromosoma 9= 196, 120, 85, 7
Cromosoma 10= 154, 96, 187, 5
Cromosoma 11= 154, 96, 94, 22
Cromosoma 12= 154, 118, 85, 6
Cromosoma 13= 66, 96, 88, 5
Cromosoma 14= 154, 118, 143,
Cromosoma 15= 154, 113, 70, 8
Cromosoma 16= 175, 108, 114, 11
Cromosoma 17= 110, 96, 13, 12
Cromosoma 18= 125, 107, 165, 9
Cromosoma 19= 154, 144, 136, 9
Cromosoma 20= 17, 108, 86, 9
Cromosoma 21= 175, 119, 63, 10
Cromosoma 22= 175, 119, 189, 6
Cromosoma 23= 110, 16, 115, 13
Cromosoma 24= 110, 107, 79, 6
Cromosoma 25= 129, 118, 114, 7
Cromosoma 26= 164, 186, 188, 28
Cromosoma 27= 110, 96, 83, 11
Cromosoma 28= 100, 96, 135, 8
Cromosoma 29= 132, 2, 133, 9
Cromosoma 30= 120, 154, 138, 45
Cromosoma 31= 164, 156, 166, 78
Cromosoma 32= 142, 1, 134, 61
Cromosoma 33= 26, 107, 115, 109
Cromosoma 34= 179, 101, 83, 62
Cromosoma 35= 132, 112, 113, 13
Cromosoma 36= 164, 32, 188, 40
Cromosoma 37= 110, 45, 137, 8
Cromosoma 38= 154, 138, 165, 5
Cromosoma 39= 178, 101, 98, 5
Cromosoma 40= 46, 107, 138, 8
Cromosoma 41= 164, 118, 19, 6
Cromosoma 42= 113, 113, 99, 10
Cromosoma 43= 110, 107, 86, 13
Cromosoma 44= 110, 198, 48, 17
Cromosoma 45= 154, 126, 76, 8
Cromosoma 46= 82, 98, 142, 9
Cromosoma 47= 110, 112, 137, 6
Cromosoma 48= 154, 117, 190, 4
Cromosoma 49= 154, 101, 130, 5
Cromosoma 50= 122, 80, 93, 3
Cromosoma 51= 182, 119, 131, 4
Cromosoma 52= 166, 104, 66, 5
Cromosoma 53= 25, 117, 126, 8
Cromosoma 54= 110, 101, 44, 7
Cromosoma 55= 154, 122, 55, 6

En color amarillo he señalado los cromosomas que mejor adaptación han tenido al medio. Por último he realizado una prueba final sobre la pista grande, en la que los agentes A y B han sido programados con los cromosomas que presentan una mejor adaptación (cromosomas 50). El resultado ha sido óptimo, el vehiculo robot se desplaza sin oscilaciones a una velocidad de 0.75m/s.

Figura 53. Test6, Final.

Conclusiones

El AIIV ha constituido una buena plataforma para demostrar, como una pequeña granja de dos robots de comportamiento reactivo, puede cooperar y hacer surgir la inteligencia emergente. A su vez he podido experimentar con los algoritmos genéticos y comprobar como pueden adaptar nuestros programas al medio en el que se desenvuelven. También he podido implementar el algoritmo de control de procesos más extendidos en la industria, el PID. Por último comentar que todo esto se ha implementado sobre microcontroladores, verdaderos sistemas informáticos concentrados en un microchip, los cuales nos obligan a optimizar el código fuente (recordemos que el PIC16F876 cuenta con tan solo 8KBytes de memoria flash, 256Bytes de RAM y 256Bytes de EEPROM).

A pesar de que el tiempo disponible no me ha permitido completar el último objetivo secundario (inteligencia basada en el conocimiento), el comportamiento del robot en el test final ha sido satisfactorio.

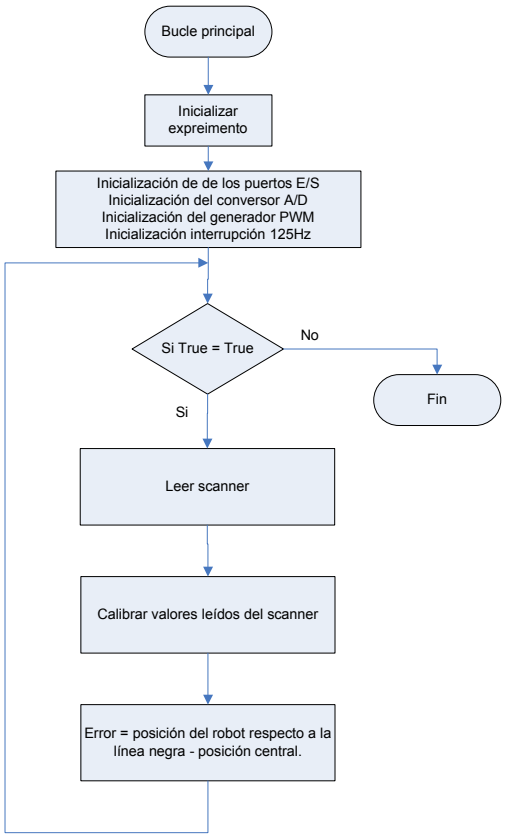
Posibles mejoras:

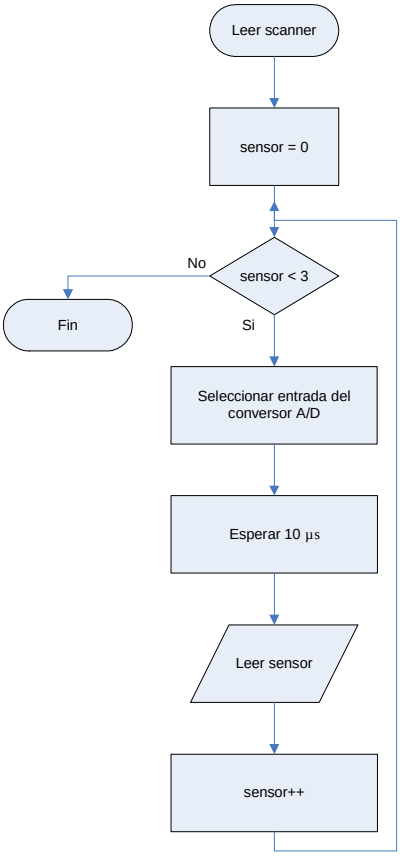
- Aumentar las dimensiones del scanner, con un mínimo de cinco elementos sensores de infrarrojos.
- Disminuir el peso del robot para mejorar su aceleración. Actualmente pesa 550 gamos.
- Cambiar el driver L293D por uno que proporcione mayor potencia a los actuadores.

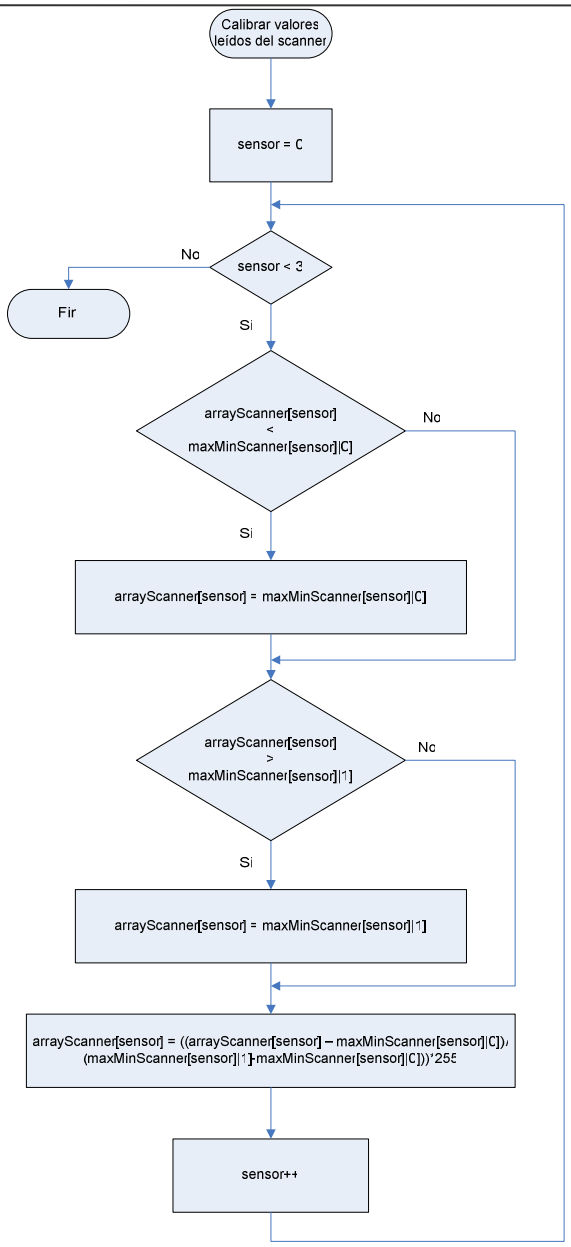
Espero que más estudiantes se animen a realizar su trabajo final de carrera apoyándose en la robótica. Esta es un medio excepcional para poder experimentar con tecnologías relacionadas con la informática, las cuales son empleadas en los departamentos de I+D del presente y el futuro (IA, microcontroladores, controles de procesos PID, sensores…).

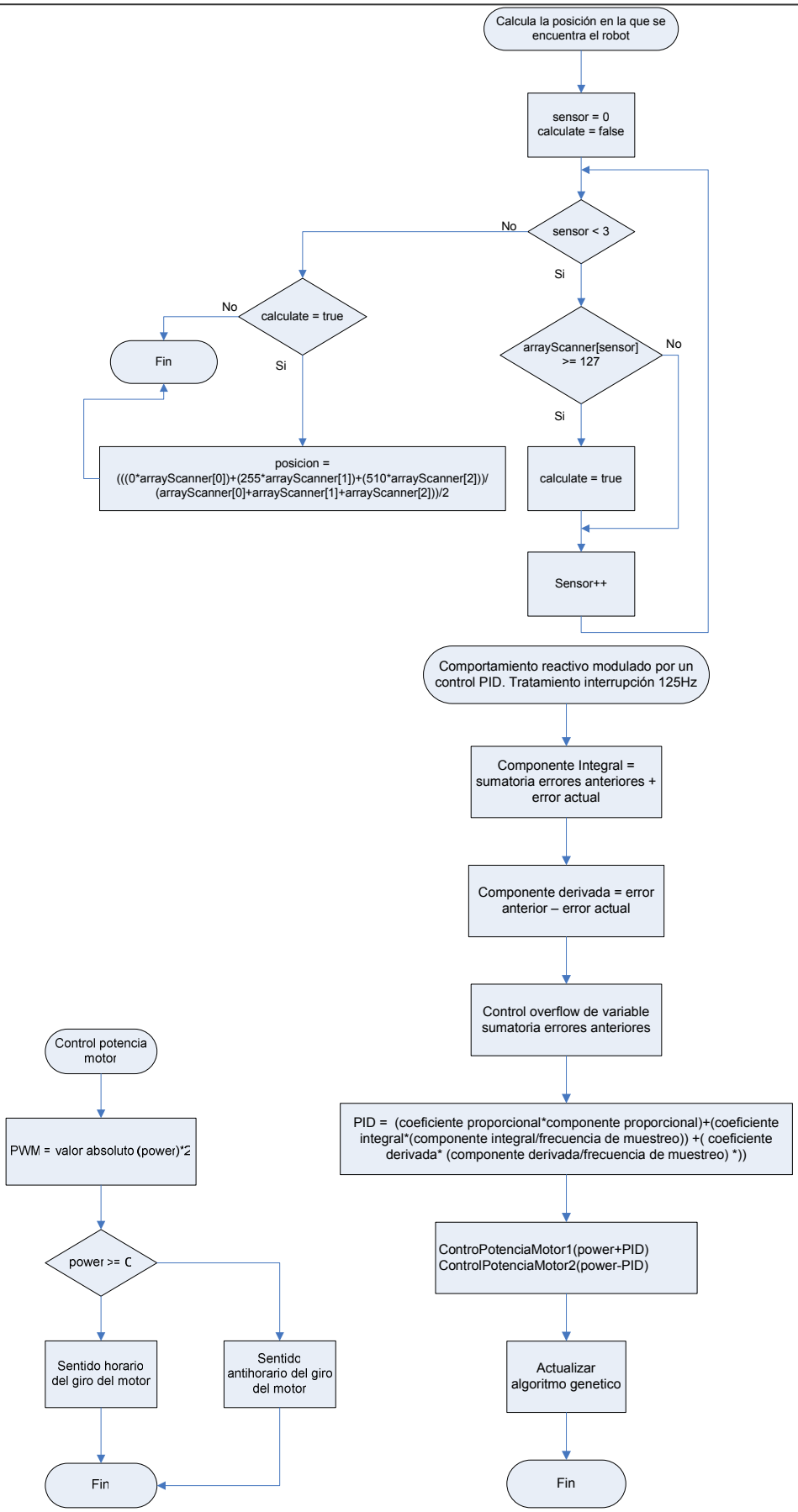
Apéndices

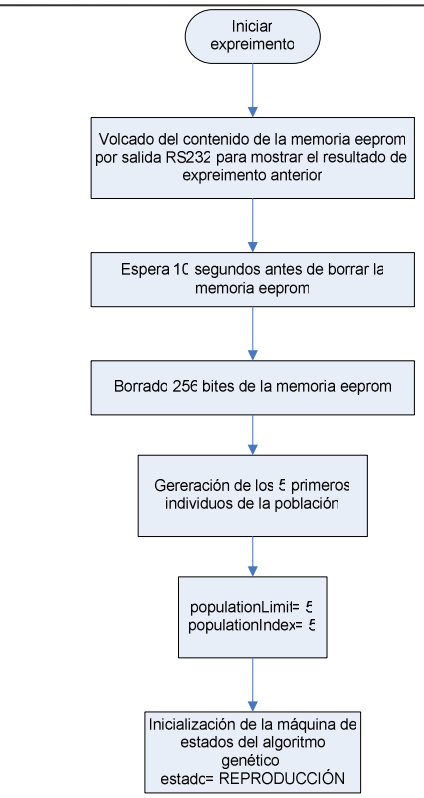
A. Diagramas de flujo

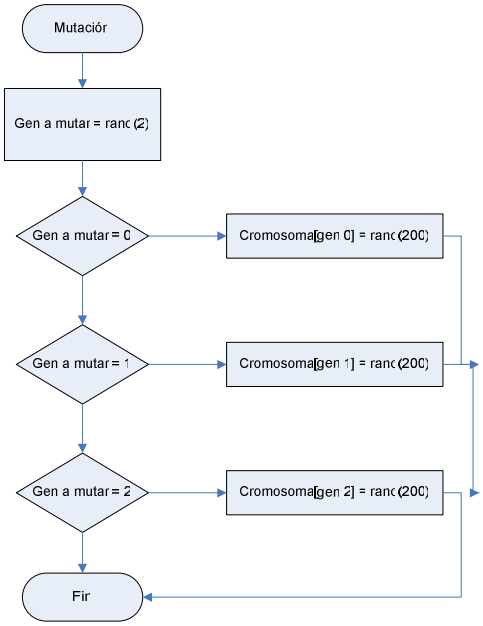


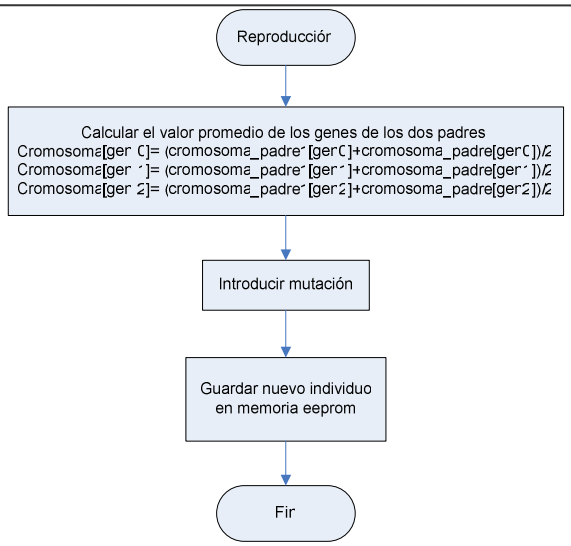


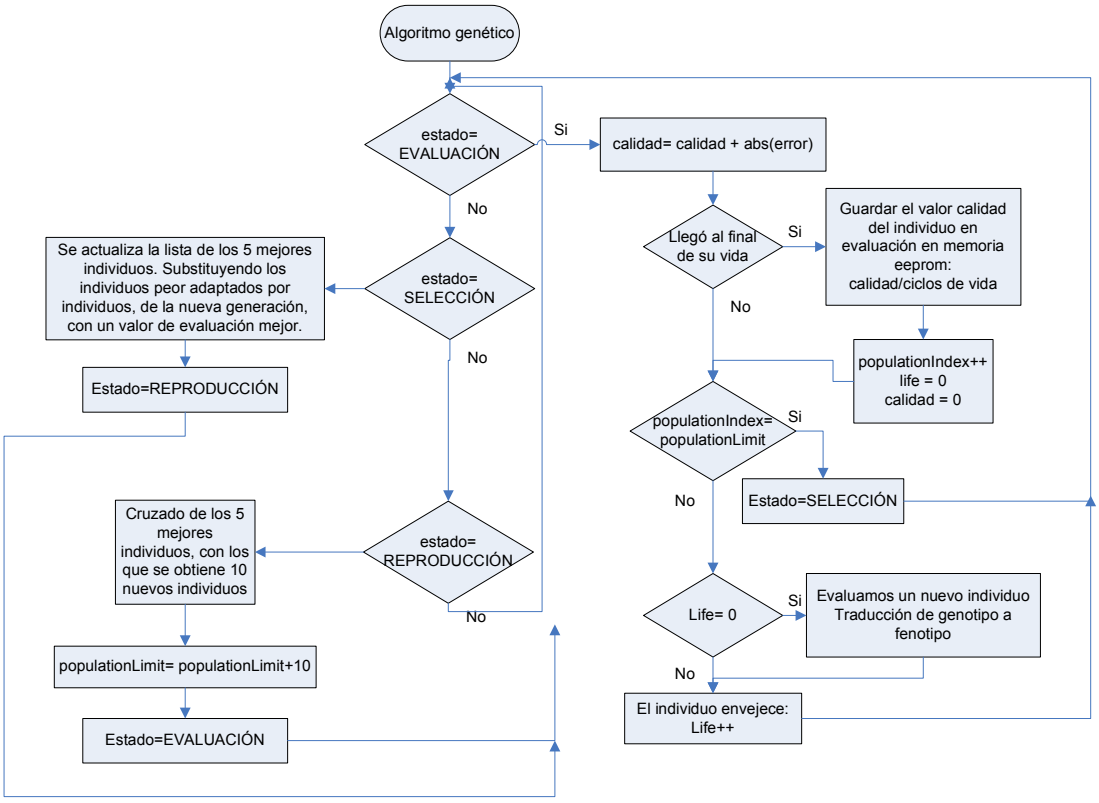

Calibrar valores leídos del scanner
sensor = 0
No
sensor < 3
Fir
Si
arrayScanner[sensor] < maxMinScanner[sensor][0]
No
Si
arrayScanner[sensor] = maxMinScanner[sensor][0]
arrayScanner[sensor] > maxMinScanner[sensor][1]
No
Si
arrayScanner[sensor] = maxMinScanner[sensor][1]
arrayScanner[sensor] = ((arrayScanner[sensor] – maxMinScanner[sensor][0])
(maxMinScanner[sensor][1]-maxMinScanner[sensor][0]))*255
sensor++


Calcula la posición en la que se encuentra el robot
sensor = 0
calculate = false
sensor < 3
No
Si
calculate = true
No
Fin
Si
arrayScanner[sensor] >= 127
No
Si
posicion = (((0*arrayScanner[0])+(255*arrayScanner[1])+(510*arrayScanner[2]))/ (arrayScanner[0]+arrayScanner[1]+arrayScanner[2]))/2
calculate = true
Sensor++
Comportamiento reactivo modulado por un control PID. Tratamiento interrupción 125Hz
Componente Integral = sumatoria errores anteriores + error actual
Componente derivada = error anterior – error actual
Control overflow de variable sumatoria errores anteriores
PID = (coeficiente proporcional*componente proporcional)+(coeficiente integral*(componente integral/frecuencia de muestreo)) +(coeficiente derivada* (componente derivada/frecuencia de muestreo) *))
ControPotenciaMotor1(power+PID)
ControlPotenciaMotor2(power-PID)
Actualizar algoritmo genetico
Fin
Control potencia motor
PWM = valor absoluto (power)*2
power >= 0
Sentido horario del giro del motor
Sentido antihorario del giro del motor
Fin


Iniciar expreimento
Volcado del contenido de la memoria eeprom por salida RS232 para mostrar el resultado de expreimento anterior
Espera 10 segundos antes de borrar la memoria eeprom
Borrado 256 bites de la memoria eeprom
Gereración de los 5 primeros individuos de la población
populationLimit= 5
populationIndex= 5
Inicialización de la máquina de estados del algoritmo genético
estado= REPRODUCCIÓN


Mutación
Gen a mutar = rand(2)
Gen a mutar = 0
Cromosoma[gen 0] = rand(200)
Gen a mutar = 1
Cromosoma[gen 1] = rand(200)
Gen a mutar = 2
Cromosoma[gen 2] = rand(200)
Fin


Reproducciór
Calcular el valor promedio de los genes de los dos padres
Cromosoma[ger C]= (cromosoma_padre[gerC]+cromosoma_padre[gerC])/2
Cromosoma[ger 1]= (cromosoma_padre[ger1]+cromosoma_padre[ger1])/2
Cromosoma[ger 2]= (cromosoma_padre[ger2]+cromosoma_padre[ger2])/2
Introducir mutación
Guardar nuevo individuo en memoria eeprom
Fir


Algoritmo genético
estado= EVALUACIÓN
Si
calidad= calidad + abs(error)
No
Llegó al final de su vida
Si
Guardar el valor calidad del individuo en evaluación en memoria eeprom: calidad/ciclos de vida
No
populationIndex++
life = 0
calidad = 0
populationIndex= populationLimit
Si
Estado=SELECCIÓN
No
Life= 0
Si
Evaluamos un nuevo individuo
Traducción de genotipo a fenotipo
No
El individuo envejece: Life++
estado= SELECCIÓN
Se actualiza la lista de los 5 mejores individuos. Substituyendo los individuos peor adaptados por individuos, de la nueva generación, con un valor de evaluación mejor.
No
Estado=REPRODUCCIÓN
estado= REPRODUCCIÓN
Cruzado de los 5 mejores individuos, con los que se obtiene 10 nuevos individuos
No
populationLimit= populationLimit+10
Estado=EVALUACIÓN

B. Código fuente

```
/*************************************************************************/
/*                                                                       */
/* Proyecto final de carrera: AIIV                                       */
/* Área:                      Inteligencia Artificial                    */
/* Universiadad:              UOC                                        */
/* Autor:                     Pedro Amador Díaz                          */
/* Fecha:                     24/10/2011                                 */
/*                                                                       */
/*************************************************************************/

#include <16F876.h>

#device ADC=8 /* Resolución del modulo A/D de 8 bits */
#fuses XT,NOWDT,PUT,NOPROTECT,NOLVP,BROWNOUT /* Configuración microcontrolador */
#use delay(clock=4000000) /* Frecuencia del oscilador (cristal de cuarzo) */
#use rs232(baud=19200, xmit=pin_c6, rcv=pin_c7) /* Configuración RS-232 */
#byte puerto_a = 05 /* Dirección del puerto A */
#byte puerto_b = 06 /* Dirección del puerto B */
#byte puerto_c = 07 /* Dirección del puerto c */

#include <stdlib.h>

/* Variables globales */
int maxMinScanner[3][2]= {{7,230},{7,222},{9,235}}; /* Valores máximos y mínimos */
int arrayScanner[3]= {0, 0, 0}; /* Valores leídos por el scanner */
unsigned char selected[5][5]= {{0,255}, {1,255}, {2,255}, {3,255}, {4,255}}; /* Cromosomas
seleccionados (índice y calidad) */
unsigned char PWM= 50; /* Potencia máxima aplicable a los actuadores 0-127 */
unsigned char populationIndex= 0; /* Índice de los individuos de la población */
unsigned char chromosome[3]= {100, 100, 100}; /* Combinación de genes inicial */
unsigned char populationLimit = 0; /* Puntero al último individuo de la población  */
unsigned char life= 0; /* Vida del individuo actual */
signed long error= 0, quality= 0;
float Kp= 0.3, Ki= 0.3, Kd= 0.01;
enum {REPRODUCTION, EVALUATION, SELECTION} state;


/*************************************************************************/
/*                                                                       */
/* motor1                                                                */
/* Descripción:      Control de la potencia y sentido de giro del motor 1*/
/* Entrada:          De -127 a 127                                       */
/* Salida:                                                               */
/* Auto:             Pedro Amador Díaz                                   */
/* Fecha:            24/10/2011                                          */
/*                                                                       */
/*************************************************************************/

void motor1 (signed long power)
{
   /* Tratamiento del desbordamiento del parámetro de entrada */
   if (power>=127)
      power = 127;
   else if (power<=-127)
      power = -127;

   /* Control de la potencia mediante PWM */
   set_pwm1_duty(abs((signed int)power)*2);

   if (power>=0)
   {
      bit_clear(puerto_b,4);
      bit_set(puerto_b,5);
   }
   else /* Para valores negativos, inversión de giro del motor */
   {
      bit_set(puerto_b,4);
      bit_clear(puerto_b,5);
   }
}


/*************************************************************************/
/*                                                                       */
/* motor2                                                                */
/* Descripción:      Control de la potencia y sentido de giro del motor 2*/
/* Entrada:          De -127 a 127                                       */
/* Salida:                                                               */
/* Auto:             Pedro Amador Díaz                                   */
/* Fecha:            24/10/2011                                          */
```

```
/*                                                                            */
/******************************************************************************/

void motor2 (signed long power)
{
   /* Tratamiento del desbordamiento del parámetro de entrada */
   if (power>=127)
      power = 127;
   else if (power<=-127)
      power = -127;

   set_pwm2_duty(abs((signed int)power)*2);   /* Control de la potencia mediante PWM */

   if (power>=0)
   {
      bit_clear(puerto_b,6);
      bit_set(puerto_b,7);
   }
   else /* Para valores negativos, inversión de giro del motor */
   {
      bit_set(puerto_b,6);
      bit_clear(puerto_b,7);
   }
}

/******************************************************************************/
/*                                                                            */
/* mutation                                                                   */
/* Descripción:      Muta los genes del cromosoma                             */
/* Entrada:                                                                   */
/* Salida:                                                                    */
/* Auto:             Pedro Amador Díaz                                        */
/* Fecha:            01/01/2012                                               */
/*                                                                            */
/******************************************************************************/

void mutation()
{
   switch (rand()/10923)
   {
      case 0:
         chromosome[0]= rand()/164;
         break;
      case 1:
         chromosome[1]= rand()/164;
         break;
      case 2:
         chromosome[2]= rand()/164;
         break;
   }
}

/******************************************************************************/
/*                                                                            */
/* chromosomeReproduction                                                     */
/* Descripción:      Reproduce un nuevo cromosoma a partir de dos padres      */
/*                   e introduce una mutación en uno de sus genes             */
/* Entrada:          Puntero a los cromosomas padres y posición donde se      */
/*                   almacenará el cromosoma del nuevo individuo              */
/* Salida:                                                                    */
/* Auto:             Pedro Amador Díaz                                        */
/* Fecha:            01/01/2012                                               */
/*                                                                            */
/******************************************************************************/

void chromosomeReproduction(unsigned char parent1, unsigned char parent2, unsigned char
reproducedChromosome)
{
         chromosome[0]= (unsigned char)(((unsigned long)read_eeprom((parent1*4)+0)+(unsigned
long)read_eeprom((parent2*4)+0))/(unsigned long)2);
         chromosome[1]= (unsigned char)(((unsigned long)read_eeprom((parent1*4)+1)+(unsigned
long)read_eeprom((parent2*4)+1))/(unsigned long)2);
         chromosome[2]= (unsigned char)(((unsigned long)read_eeprom((parent1*4)+2)+(unsigned
long)read_eeprom((parent2*4)+2))/(unsigned long)2);

         mutation();

         write_eeprom(((reproducedChromosome)*4)+0,chromosome[0]);
         write_eeprom(((reproducedChromosome)*4)+1,chromosome[1]);
         write_eeprom(((reproducedChromosome)*4)+2,chromosome[2]);
         write_eeprom(((reproducedChromosome)*4)+3,0);
}

/******************************************************************************/
/*                                                                            */
/* geneticAlgorithm                                                           */
/* Descripción:      Máquina de estados que forma el algoritmo genético       */
/* Entrada:                                                                   */
```

```
/* Salida:                                                                  */
/* Auto:            Pedro Amador Díaz                                       */
/* Fecha:           01/01/2012                                              */
/*                                                                          */
/****************************************************************************/

void geneticAlgorithm()
{
   unsigned char index1=0, index2= 0;
   struct {
      unsigned char index;
      unsigned char value;
   } minQuality;

   switch (state)
   {
      case EVALUATION:

      /* Calculo del valor calidad, con el que se evaluará el éxito del individuo */
      quality = quality + abs(error);

      /* Terminó la vida del actual individuo */
      if (life>=250)
      {
         /* Guardamos el valor de calidad del individuo evaluado */
         write_eeprom((populationIndex*4)+3,(int)(quality/250));

      /* Siguiente cromosoma para un nuevo individuo */
      populationIndex++;
      life = 0;
      quality = 0;
      }

      if (populationIndex==populationLimit)
      {
         state= SELECTION;
         break;
      }

      /* Ha nacido un nuevo individuo */
      if (life==0)
      {
         /* A partir de los genes formamos el fenotipo del nuevo individuo */
         Kp= ((unsigned long)read_eeprom((populationIndex*4)+0)+200)/1000.0;
         Ki= ((unsigned long)read_eeprom((populationIndex*4)+1)+200)/1000.0;
         Kd= (read_eeprom((populationIndex*4)+2))/10000.0;
         printf ("Kp= %1.3f, Ki= %1.3f, Kd= %1.3f, Quality= %d\n", Kp, Ki, Kd, (int)(quality/250));
      }

      /* El individuo en evaluación envejece */
      ++life;

         break;

      case SELECTION:

            /* Actualizamos el array de los mejores 5 individuos de la población */
            index1= populationIndex-10;
            for (index1; index1<populationLimit; index1++)
            {
               /* Buscamos el individuo más débil de la generación anterior */
               index2= 0;
               minQuality.value= 0;
               minQuality.index= 0;
               for (index2; index2<5; index2++)
               {
                  printf("%u %u %u %u\n\r", minQuality.index, minQuality.value, index2,
selected[index2][1]);
                  if (selected[index2][1]>=minQuality.value)
                  {
                     minQuality.value= selected[index2][1];
                     minQuality.index= index2;
                  }
               }

               if (minQuality.value>=read_eeprom((index1*4)+3))
               {
                  selected[minQuality.index][0]= index1;
                  selected[minQuality.index][1]= read_eeprom((index1*4)+3);
               }
            }

         state= REPRODUCTION;
         break;

      case REPRODUCTION:

         /* Cruzamos a los 5 individuos de la población */
```

```
         chromosomeReproduction(selected[0][0], selected[1][0], populationIndex+0);
         chromosomeReproduction(selected[0][0], selected[2][0], populationIndex+1);
         chromosomeReproduction(selected[0][0], selected[3][0], populationIndex+2);
         chromosomeReproduction(selected[0][0], selected[4][0], populationIndex+3);
         chromosomeReproduction(selected[1][0], selected[2][0], populationIndex+4);
         chromosomeReproduction(selected[1][0], selected[3][0], populationIndex+5);
         chromosomeReproduction(selected[1][0], selected[4][0], populationIndex+6);
         chromosomeReproduction(selected[2][0], selected[3][0], populationIndex+7);
         chromosomeReproduction(selected[2][0], selected[4][0], populationIndex+8);
         chromosomeReproduction(selected[3][0], selected[4][0], populationIndex+9);

         populationLimit = populationLimit + 10;

         state= EVALUATION;
         break;
   }

   if (populationIndex==55) /* Fin del experimento evolutivo*/
   {
      motor1(0);
      motor2(0);
      bit_clear(puerto_b,0);
      bit_set(puerto_b,1);
   }
   else /* La evolución continua */
   {
   /* Frecuencia de muestreo del bucle PID de 125Hz */
   set_timer1(64536);
   }
}

/*****************************************************************************/
/*                                                                           */
/* experimentRestart                                                         */
/* Descripción:      Lectura de la memoria EEPROM para mostrar el resultado  */
/*                   del experimento anterior. Pasados 10 segundos, preparamos*/
/*                   uno nuevo                                               */
/* Entrada:                                                                  */
/* Salida:                                                                   */
/* Auto:             Pedro Amador Díaz                                       */
/* Fecha:            30/12/2011                                              */
/*                                                                           */
/*****************************************************************************/

void experimentRestart()
{
   long index,seconds;

   /* Lectura EEPROM */
   for (index= 0; index<256; index=index+4)
   {
      printf("Cromosoma %ld= %u, %u, %u, %u \n\r "(index/4)+1, read_eeprom(index), \
      read_eeprom(index+1), read_eeprom(index+2), read_eeprom(index+3));
   }

   /* Espera 10 segundos */
   for (seconds=0; seconds<11; seconds++)
   {
      delay_ms(1000);
   }

   /* Borrado EEPROM */
   for (index= 0; index<256; index++)
   {
      write_eeprom(index,0);
   }

   /* Generación inicial de 5 individuos*/
   for (index= 0; index<5; index++)
   {
      mutation();

      write_eeprom((index*4)+0,chromosome[0]);
      write_eeprom((index*4)+1,chromosome[1]);
      write_eeprom((index*4)+2,chromosome[2]);
      write_eeprom((index*4)+3,0);

      ++populationLimit;
   }

   populationIndex= populationLimit;
   state = REPRODUCTION;

}


/*****************************************************************************/
/*                                                                           */
```

```
/* scannerReader                                                          */
/* Descripción:      Lectura de los tres sensores infrarrojos             */
/* Entrada:          Puntero al array que almacenará los valores leídos   */
/*                   por los sensores infrarrojos                         */
/* Salida:                                                                */
/* Auto:             Pedro Amador Díaz                                    */
/* Fecha:            24/10/2011                                           */
/*                                                                        */
/**************************************************************************/

void scannerReader(int *arrayScanner)
{
   int sensor;

   for (sensor=0; sensor<3; sensor++)
   {
      set_adc_channel(sensor); /* Cambio de canal A/D */
      delay_us(10); /* Espera tras cambiar de canal */
      arrayScanner[sensor]=255-read_adc();  /* Lectura valor analógico */
   }
}


/**************************************************************************/
/*                                                                        */
/* sensorsCalibration                                                     */
/* Descripción:      Calibrado de las lecturas de los sensores infrarrojos*/
/* Entrada:          Puntero al array que almacena los valores leídos     */
/*                   por los sensores infrarrojos                         */
/* Salida:                                                                */
/* Auto:             Pedro Amador Díaz                                    */
/* Fecha:            24/10/2011                                           */
/*                                                                        */
/**************************************************************************/

void sensorsCalibration(int *arrayScanner)
{
   int sensor;

   for (sensor=0; sensor<3; sensor++)
   {
      /* Lectura inferior al mínimo posible */
      if (arrayScanner[sensor]<maxMinScanner[sensor][0])
         arrayScanner[sensor]=maxMinScanner[sensor][0];

      /* Lectura superior al máximo posible */
      if (arrayScanner[sensor]>maxMinScanner[sensor][1])
         arrayScanner[sensor]=maxMinScanner[sensor][1];

      /* Normalizado de las lecturas de los sensores infrarrojos */
      arrayScanner[sensor]=((float)(arrayScanner[sensor]-maxMinScanner[sensor][0])/
      (maxMinScanner[sensor][1]-maxMinScanner[sensor][0]))*255;
   }
}


/**************************************************************************/
/*                                                                        */
/* linePosition                                                           */
/* Descripción:      Calculo de la posición en la que se encuentra el robot*/
/*                   respecto a la línea                                  */
/* Entrada:          Puntero al array que almacena los valores leídos     */
/*                   por los sensores infrarrojos                         */
/* Salida:           0 al 255, siendo 128 centrado                        */
/* Auto:             Pedro Amador Díaz                                    */
/* Fecha:            24/10/2011                                           */
/*                                                                        */
/**************************************************************************/

int linePosition(int *arrayScanner)
{
  static int position=128;
  boolean calculate=false;
  int sensor;

  for (sensor=0; sensor<3; sensor++)
  {
      /* Solo se calcula si la lectura de la línea tiene suficiente contraste */
      if (arrayScanner[sensor]>=127)
         calculate=true;
  }

  if (calculate==true)

    /* Cálculo ponderado */
    position=(((0*(int32)arrayScanner[0])+(255*(int32)arrayScanner[1])+(510*(int32)arrayScanner[2]))/
           ((int32)arrayScanner[0]+(int32)arrayScanner[1]+(int32)arrayScanner[2]))/2;
```

```
 return position;
}

/*****************************************************************************/
/*                                                                           */
/* reactiveControlPID                                                        */
/* Descripción:      Inteligencia reactiva modulada mediante control PID     */
/* Entrada:          Posición del robot respecto a la línea. Del 0 al 255,   */
/*                   siendo 128 centrado                                     */
/* Salida:                                                                   */
/* Auto:             Pedro Amador Díaz                                       */
/* Fecha:            30/12/2011                                              */
/*                                                                           */
/*****************************************************************************/

#INT_TIMER1
void reactiveControlPID()
{
   signed long PID= 0, errorDerivative= 0;
   static signed long errorSum, errorLast;

   /* Calculo de la componente integral */
   errorSum = errorSum + error;

   /* Tratamiento del desbordamiento de la variable errorSum */
   if (errorSum>=32000)
      errorSum = 32000;
   else if (errorSum<=-32000)
      errorSum = -32000;

   /* Cálculo de la componente derivada */
   errorDerivative = error - errorLast;
   errorLast = error;


   PID = (signed long)((Kp*error)+(Ki*(errorSum/125))+(Kd*(errorDerivative*125)));

   /* Tracción con dirección diferencial */
   motor1(PWM+PID);
   motor2(PWM-PID);

   /* Actualización de la máquina de estados que conforma el algoritmo genético */
   geneticAlgorithm();
}

/*****************************************************************************/
/*                                                                           */
/* reactiveControl                                                           */
/* Descripción:      Inteligencia reactiva simple                            */
/* Entrada:          Posición del robot respecto a la línea. Del 0 al 255,   */
/*                   siendo 128 centrado                                     */
/* Salida:                                                                   */
/* Auto:             Pedro Amador Díaz                                       */
/* Fecha:            24/10/2011                                              */
/*                                                                           */
/*****************************************************************************/

void reactiveControl (int position)
{
   if (position < 85) /* Gira a la derecha */
   {
      /* Led de color verde */
      bit_set(puerto_b,0);
      bit_clear(puerto_b,1);

      /* Motor derecho frenado */
      motor1(-10);
      motor2(75);
   }
   else if (position < 170) /* Sigue recto */
   {
      /* Led de color amarillo */
      bit_set(puerto_b,0);
      bit_set(puerto_b,1);

      /* PWM motores izquierdo y derecho a la misma velocidad */
      motor1(75);
      motor2(75);
      }
   else /* Gira a la izquierda */
   {
      /* Led de color rojo */
      bit_clear(puerto_b,0);
      bit_set(puerto_b,1);

      /* Motor izquierdo frenado */
      motor1(75);
      motor2(-10);
```

```
   }
}


/*************************************************************************/
/*                                                                       */
/* main                                                                  */
/* Descripción:      Función que inicializa el sistema y contiene el bucle    */
/*                   principal (motor del sistema reactivo sin control PID)   */
/* Entrada:                                                              */
/* Salida:                                                               */
/* Auto:             Pedro Amador Díaz                                   */
/* Fecha:            24/10/2011                                          */
/*                                                                       */
/*************************************************************************/

void main()
{
   /* Leemos el resultado del experimento anterior
      y pasados 10 segundos iniciamos uno nuevo */
   experimentRestart();

   /* Inicialización de los timers */
   setup_timer_1(T1_INTERNAL | T1_DIV_BY_8);
   enable_interrupts(INT_TIMER1);
   enable_interrupts(GLOBAL);

   set_tris_a(0xFF); /* Puerto A como entreada */
   set_tris_b(0x00); /* Puerto B como salida */
   set_tris_c(0x00); /* Puerto C como salida */

   /* Puertos A0 A1 A2 A3 A4 entradas analógicas */
   setup_adc( ADC_CLOCK_DIV_8 );
   setup_adc_ports( ALL_ANALOG );

   /* Inicialización de los PWM */
   setup_timer_2(T2_DIV_BY_1,255,1);
   setup_ccp1(ccp_pwm);
   setup_ccp2(ccp_pwm);

   /* Inicialización del display */
   bit_set(puerto_b,0);
   bit_clear(puerto_b,1);

      set_timer1(64536);   set_timer1(64536);set_timer1(64536);

   while (true)
   {
      /* Lectura de los valores recogidos por los sensores de infrarrojos */
      scannerReader(arrayScanner);

      /* Calibrado de los valores recogido por los sensores */
      sensorsCalibration(arrayScanner);

      /* Cálculo del error */
      error = (signed int)linePosition(arrayScanner)-128;

      /* Para un control reactivo sin control PID, comentar las líneas de código
         referentes al timer1, calculo de error y descomentar las siguientes líneas*/
      //reactiveControl (linePosition(arrayScanner))
   }
}
```

Bibliografía

Pajares Martinsanz, Gonzalo; Santos Peñas, Matilde (2005). *Inteligencia artificial e ingeniería del conocimiento*. Madrid: RA-MA Editorial.

Pajares Martinsanz, Gonzalo; de la Cruz García, Jesús Manuel (2010). *Aprendizaje automático. Un enfoque práctico*. Madrid: RA-MA Editorial.

Santos Reyes, José; Richard, J. Duro (2004). *Evaluación artificial y robótica autónoma*. Madrid: RA-MA Editorial.

Palacios Municio, Enrique; Remiro Domínguez, Fernando; J. López Pérez, Lucas (2004). *Microcontrolador PIC16F84. Desarrollo de proyectos*. Madrid: RA-MA Editorial.

Angulo Usategui, José Mª; Romero Yesa, Susana; Angulo Martínez, Ignacio (1999). *Microbótica. Tecnología, aplicaciones y montaje práctico*. Madrid: Thomsom Editores.

Angulo Usategui, José Mª; Romero Yesa, Susana; Angulo Martínez, Ignacio (2000). *Microcontroladores PIC. Diseño práctico de aplicaciones. Segunda parte: PIC 16F87X*. Madrid: McGRAW-HILL.

Tipler, Paul A (1994). *Física*. Editorial Reverté.

Charais, John; Lourens, Ruan (2004). *Software PID Control o fan Inverted Pendulum Using the PIC16F684*. Microchip Technology Inc.

MABUCHI MOTOR.
<http://www.mabuchi-motor.co.jp/>

Pololu. Robotics and electronics.
< http://www.pololu.com/>

Ostan.
< http://www.ostan.cz/>