Memoria – RoboPacMan

Cátedra Bosch–UCM

Jesús Parra Torrijos

jespar01@ucm.es

La intención final de este proyecto es la creación del videojuego MsPacMan vs Ghosts en físico mediante unos robots que han sido proporcionados tanto por la UCM como por BOSCH para poder trabajar en ello.

Para ello, la idea del proyecto se ha dividido en dos fases: una primera fase en la cual los robots están programados para jugar al videojuego por sí mismos, de forma aleatoria, mediante un programa en el cuál son capaces de reconocer el circuito del PacMan y también si han sido atrapados por los demás robots; y una segunda fase en la cuál está integrado en los robots un sistema de comunicación con una inteligencia artificial en la cuál el comportamiento de los robots en el juego deja de ser aleatoria y se convierte en un juego de supervivencia por parte del robot que hace de MsPacMan en el cuál intenta escapar de los demás robots, los fantasmas, mediante una serie de instrucciones mandadas desde la IA encargada de reconocer dónde están los fantasmas y escoger las mejores rutas posibles del circuito para escapar de ellos y conseguir la victoria.

PRIMERA FASE: Funcionamiento aleatorio

Primeros pasos

Como todo proyecto, esto debe de empezar por la base. Se comenzó todo partiendo de los robots que la Cátedra Bosch–UCM proporcionó para la realización del proyecto. En concreto, son unos robots *mBot Mega*, fabricados por *MakeBlock*. Son unos robots que están compuestos por cuatro ruedas mecánicas, cada una conectada a un motor, donde están a su vez conectadas a su placa MegaPi, una placa basada en una placa Mega 2560 compatible con *Arduino* y *Raspberry Pi*, en la cuál se tiene en la placa un apartado para la comunicación con la *Raspberry Pi*, unos 10 pines para poder conectar sensores a nuestra disposición, un apartado para conectar módulos de transmisión inalámbrica, *Bluetooth* en nuestro caso, un conector de tipo USB–B, y los pines para conectar los cuatro motores de las cuatro ruedas.

Lo primero que se hizo con estos robots fue la construcción de los mismos, ya que estaban desmontados. Los robots fueron montados según el fabricante indica en el manual, con todos los sensores (dos de seguimiento de línea con IR, dos sensores de choque, tres sensores de detección de obstaculos con IR y dos módulos con luces LED

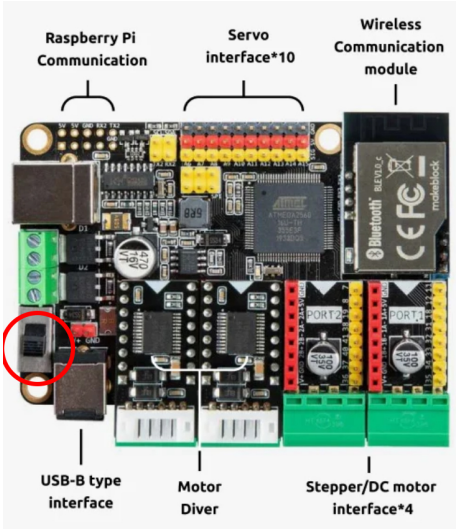

Figura 1: Placa MegaPi montada en los mBot Mega.

RGB). Cada uno de los sensores van conectados en la zona de "*Servo interface**10" que vemos en la anterior imagen, de manera predeterminada: es decir, las placas LED RGB van conectados en los pines A9 y A10, los sigue líneas van conectados en los pines A11 y A12, los sensores de detección de obstaculos del A13 al A15, los sensores de detección de barreras en los A16 y A17. Los motores conectados a las ruedas van conectados en la sección "*Stepper/DC motor interface**4". Así, cada robot es funcional, y además con la app del fabricante se puede manejar como un coche teledirigido, pudiéndosele cambiar la velocidad, que gire, cambiar la luz de las placas LED RGB y con los sensores de detección de obstaculos y de barrera se puede poner un programa que tiene para detectar la habitación y que funcione de forma independiente evitando obstaculos.

Estos fueron los primeros pasos para ver el funcionamiento de los robots que teníamos entre manos, pasos muy básicos con el material de base que nos daban los fabricantes. Para empezar a «cacharrear» con el robot, empezamos a trastear con el *software* predeterminado del fabricante. Éste es un lenguaje de programación basado en bloques, donde sólamente se pueden desarrollar unos programas sencillos, como los que podemos ver en el ejemplo siguiente.

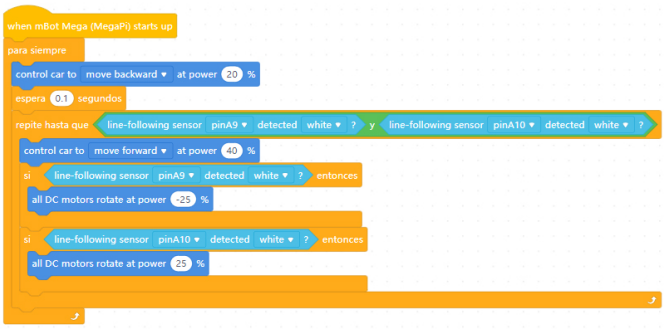

Figura 2: Programa básico para que el robot siga líneas.

Este programa es un ejemplo básico en el cuál lo único que hace el robot mBot Mega es que al encenderse, una vez se toca el interruptor, rodeado en rojo en la imagen, si los sensores que tiene conectados en la parte inferior del vehículo leen blanco, el coche va marcha atrás a una velocidad del 20 % de la velocidad total del robot, si ambos sensores leen negro, que el robot avance hacia delante al 40 % de la velocidad total posible, que únicamente el sensor colocado a la derecha lee negro y el de la izquierda lee blanco, que el robot gire a la derecha al 35 % de la velocidad total hasta que los dos sensores lean negro y pueda seguir la línea, y lo mismo ocurre si la situación es la inversa, que el de la derecha lea blanco y el de la izquierda negro, gire a la izquierda para reemprender la marcha.

Una parte positiva del *software* del fabricante es que este código por bloques lo traducen directamente a un código basado en C++ funcionable perfectamente en *Arduino*, que es una plataforma de creación de electrónica en código abierto, basada en hardware y software libre. Gracias a esto, se ha facilitado mucho la tarea de puesta a punto del robot, ya que no estaríamos empezando de cero sino de una base ya establecida. Una vez con esto sabido y probado, se empezó a buscar las soluciones de lo que sería el problema de este proyecto, la detección de las líneas del mapa del MsPacMan vs Ghosts.

El hecho de que el mBot Mega, nuestro robot, siguiese una simple línea era muy sencillo y lo teníamos dominado, incluso siendo una línea curvada. El problema llegaría en los giros de 90° y en las intersecciones.

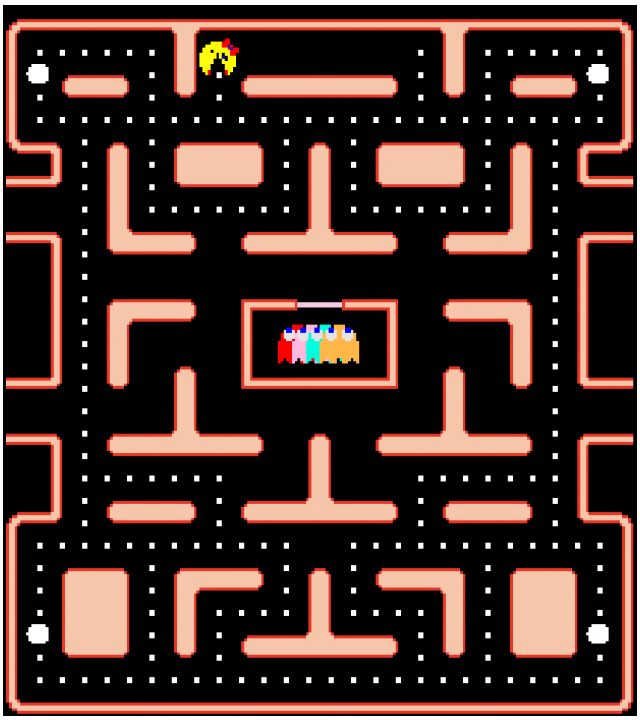

Figura 3: Mapa MsPacMan vs Ghosts original.

Reconocimiento del mapa

La figura anterior nos deja el mapa con el que en un principio estaríamos trabajando, un mapa lleno de giros e intersecciones para los cuales nuestro robot no estaba preparado por las condiciones que se tenían en un principio. Debíamos diseñar el mapa de una forma concreta para que el robot lo pudiera seguir, lo que se concluyó que, gracias a poder contar con materiales de proyectos anteriores, en este caso sensores IR, se optó por hacer el mapa con una línea negra de un grosor $2mm$ más gruesa que la distancia entre los dos sensores sigue-líneas, con giros e intersecciones a 90°. El programa del sigue líneas, para la línea recta era robusto y preciso, por lo que a ésto solamente tuvimos que añadirle otros dos sensores más, uno en el extremo de la derecha y otro en el extremo de la izquierda para poder detectar los giros e intersecciones.

Gracias al *software* proporcionado por el fabricante podíamos dominar el control del robot para que pudiese hacer giros en modo tanque, es decir, girando una serie de grados sobre su propio eje. Este tipo de giro es fundamental para este proyecto en concreto, ya que gracias a esto podemos lograr un giro de 90° que es lo deseado.

No tuvo ningún misterio y más allá que unas cuantas pruebas, tanto para derecha como para izquierda, para entender cómo giraría el robot a nuestro antojo, con los sensores nuevos colocados en los extremos ya tendríamos por fin el objetivo de seguir el nuevo mapa, únicamente sería integrarlo en el programa de sigue-líneas anterior.

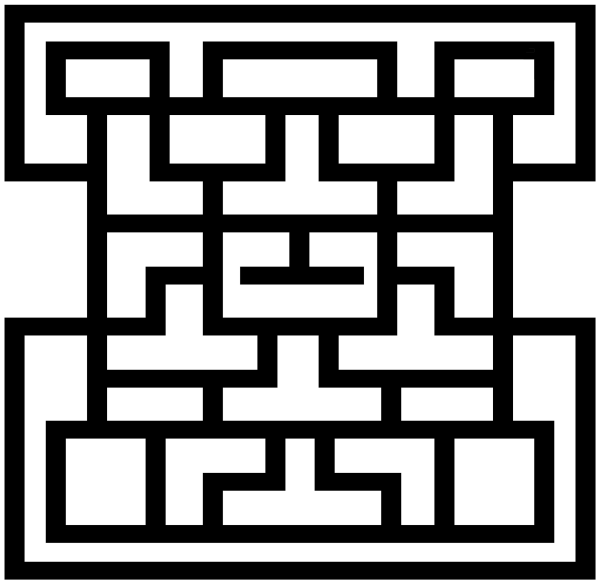

Figura 4: Mapa MsPacMan vs Ghosts propuesto para el proyecto.

Este tema podría parecer trivial, pero en realidad no lo era. Teníamos únicamente los giros a derecha y a izquierda, pero nos faltaban unos cuantos casos más a considerar, como a derecha y a izquierda a la vez; a derecha, a izquierda y de frente; a derecha y de frente; y a izquierda y de frente. Todo esto lo teníamos que considerar y añadir al programa. Se encontró algún que otro problema con la implementación de tantos casos, pero al final se logró ajustando el *hardware*, ya que la idea del programa era suficientemente robusta y eficaz como para que funcionase. Se optó por rehacer en una impresora 3D la placa que sujeta por debajo del robot a los sensores sigue líneas, poniendo los sensores sigue-líneas en el centro, más juntos entre sí, y los sensores que detectan si hay línea a derecha o a izquierda en la parte máss alejada de los sensires sigue-líneas del robot, para que no interfieran con el seguimiento de la línea. Además, la colocación de todos los sensores se tuvo en cuenta para que los sensores sigue-líneas y de los extremos pudieran detectar todas las combinaciones de casos posibles, ya que si algún sensor de los extremos medía si había alguna línea a derecha o izquierda, los sensores sigue-líneas también podían medir si había o no camino de frente, por lo que se simplificaba el tener que añadir más sensores de detección de intesecciones.

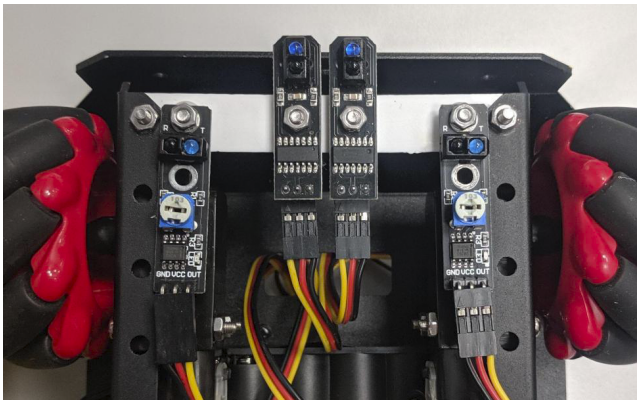

Figura 5: Colocación de sensores en la nueva placa impresa en 3D.

Con esto, ya tendríamos detectados todas los casos posibles de intersecciones y giros y podríamos hacer que nuestros robots funcionasen siguiendo las líneas. Ahora la idea era implementar estos casos y que los robots elijan las elección del camino de forma aleatoria. Esto es trivial en *Ar-*

duino, donde ya hemos pasado a trabajar por la facilidad de editar cualquier parte del robot. Ahí, llamamos a una función *random()* para que elija un número aleatorio para elegir entre las opciones posibles, sea girar a la derecha, a izquierda, que siga de frente o que se dé marcha atrás, dependiendo del tipo de intersección que se tenga.

Últimos preparativos de la primera fase

Así tendríamos casi completada esta parte, la del funcionamiento aleatorio, pero nos faltaría por implementar un sistema que detecte los choques, sea del fantasma a MsPacMan como viceversa. Esto se haría colocando varios de los sensores de detección de proximidad en los robots y programándolos de forma que cada caso sea distinto. Por ejemplo, si se encuentran los robots de frente, que cada uno se dé la vuelta y vuelvan a intentar perseguirse; si se encuentra uno detrás de otro, pues uno se daría la vuelta y el que se encuentra por detrás aceleraría la marcha para alejarse del que le intentaba pillar. Esta parte es trivial en el sentido de colocación de los sensores en el robot y en la programación, ya que haríamos que si el sensor, que es IR, detectase algún movimiento, parase todo lo que estaba haciendo y haga el movimiento de "cazado" por parte del otro robot. En teoría, esto era funcional, pero en la práctica, los sensores IR utilizados no son muy precisos y aunque se pueda regular su sensibilidad, no es del todo fiable que funcione ya que se puede deslumbrar por la luz natural o detectar el suelo, cuando no detecta otro robot, e interrumpir el programa que estaba funcionando y hacer el movimiento de "cazado"

Con esto, la primera fase, por el momento, estaría completada, dejando sólamente a la IA su actuación para que fuese completamente un juego completamente automatizado.

SEGUNDA FASE: Integración de la IA

Conexión por *Bluetooth*

Llega la parte clave en el proyecto, conectar el robot con la IA ya hecha. Este debería ser un paso sencillo, pero no lo ha sido. Parte del problema viene en parte por el desconocimiento del funcionamiento del robot, más concretamente de la placa MegaPi que es la placa que usan los robots.

Partimos del módulo de *bluetooth* de *Makeblock*, el fabricante, el cual trabaja a una frecuencia de 115200 baudios, información que ha tenido que ser buscada y rebuscada en páginas del propio fabricante ya que no podríamos saberlo de ninguna manera porque no se da ninguna información en el manual del robot. Este módulo está conectado a la placa MegaPi en 4 pines, uno de $5V$, otro de GND, y los pines RX3 y TX3. Esto será clave para entender cómo comunicarnos con el módulo. Este módulo se comunica desde el *firmware* de fábrica con la app de Makeblock, pero esta comunicación es demasiado compleja para poder adaptarla a nuestro programa.

Como hemos comentado anteriormente, gracias a que teníamos material del proyecto anterior se ha podido facilitar mucho la continuación del proyecto. Teníamos dos módulos *bluetooth* HC–05, que son maestro-esclavo, y una placa Arduino

UNO R3 extra. Con esto, podíamos sustituir el módulo de *bluetooth* de *Makeblock* que no se puede configurar, por un módulo que podemos configurar y saber toda los datos del propio módulo para poder comunicarnos con él.

La comunicación del proyecto anterior estaba basado en la placa Arduino UNO R3 haciendo de "servidor", comunicándose con el robot mediante los dos módulos de *bluetooth*, el que estaba conectado en el servidor sería el maestro, y el que está en el robot sería el esclavo. Con esto sería todo mucho más sencillo, ya que la conexión entre el PC y el módulo de *Makeblock* se tornaba imposible sin el *plug-in* de *bluetooth* de *Makeblock* que además estaba fuera de stock y no se podía comprar en ninguna tienda.

Teníamos entonces un módulo maestro HC–05 conectado a la placa Arduino UNO R3, y comunicándose entre sí con el HC–05 que actuaba como esclavo en la MegaPi. Este último se podía conectar a la placa en la misma posición que estaba conectado el módulo original. Se tuvo muchos problemas al hacer la comunicación entre los módulos ya que en un principio se quería hacer una comunicación por puerto serie simulado, conectando los pines RX y TX del módulo a unos pines en concreto de la placa MegaPi, ya que buscando información así se hace en las demás placas, y así funciona en el "servidor" de la placa Arduino UNO R3, pero en la MegaPi, al estar colocado en los pines RX3 y TX3 no hacía falta simular el puerto serie sino que podíamos programarlo con una librería de Arduino, como *SoftwareSerial*, ya que si estaba en RX3 y TX3, podíamos usar *Serial3* para la comunicación. El hallazgo de esto podría parecer trivial pero fue lo más costoso del proyecto ya que al no tener ninguna información del funcionamiento de la placa MegaPi fue un descubrumiento mucho más tedioso de lo esperado.

Conexión de la IA al robot

Una vez configurado el módulo esclavo HC–05 conectado al robot, y el módulo HC–05 maestro conectado al "servidor" de la placa Arduino UNO R3, y comunicándose entre sí, ya podíamos conectar los programas de la IA, que estaban en *Java*, con el robot. El robot se conecta al "servidor", mandándole un mensaje de confirmación de que están conectados,el servidor recibe el mensaje de que está preparado y listo para inicializar, y ya aquí el servidor se comunica con el programa en Java y se pone el programa en marcha.

Los programas necesarios para ambos Arduinos están adjuntados en los repositorios disponibles.

Una vez sincronizamos el robot con el servidor, el servidor nos indica que todo está conectado con el robot, y se inicializa la simulación, es momento de colocar el robot en el mapa y la IA se encarga de controlar el robot por todo el mapa como muestra el simulador.

Debido a la falta de tiempo del proyecto, el objetivo final de conectar al simulador los robots fantasmas ha resultado imposible, pero el objetivo de hacer funcionar el videojuego de MsPacMan con el simulador en el mundo físico con robots podemos resolverlo con éxito. En el mismo repositorio también se encuentran subidos los vídeos de las presentaciones finales, grabando tanto el simulador como la parte donde actúan los robots sobre el mapa.