Análisis e implementación de algoritmos para la solución de laberintos de estructura conocida

Omar Rodríguez, Erik Hernández

Facultad de Informática, Universidad Autónoma de Querétaro, México

Resumen. El presente capítulo explica el análisis e implementación de algoritmos para lograr que un robot móvil sea capaz de navegar de forma autónoma dentro de un laberinto, de 3.6x3.6 metros, conocido con anterioridad y que consta de paredes de 25 cm de altura con ángulos rectos y calles de 40 cm de ancho. La principal dificultad, radica en que no se conoce con anterioridad la posición y orientación inicial del robot. En el capítulo se analizan varios algoritmos clásicos aplicables a la solución de laberintos y se escoge uno para desarrollarlo completamente sobre un robot específico con un microcontrolador PIC16F877.

1 Introducción

Los concursos de robótica van cobrando cada vez mayor fuerza, no solo a nivel académico, sino a nivel empresarial, puesto que representan una base muy importante para la creación de proyectos aplicables directamente en los procesos de desarrollo e investigación de la industria.

Entre los objetivos particulares que se persiguen, están analizar los problemas de codificación del algoritmo en un microcontrolador específico, programar las etapas de adquisición de datos y control de motores en el microcontrolador.

La categoría consiste en la navegación autónoma a través de un laberinto de 3.6x3.6 metros desde un punto de partida no conocido hasta encontrar la salida. El laberinto estará constituido por paredes de 25 cm de altura, diseñadas en ángulos rectos y con calles de 40 cm de anchura. Todas las medidas son aproximadas, el robot deberá ser capaz de soportar una incertidumbre de medida del 10%.

2 Desarrollo

2.1 Descripción del robot

Por las características de la categoría, es necesario que el robot tenga la posibilidad de girar sobre su propio eje, y realice giros y paros con rapidez. Las dimensiones máximas del robot están limitadas a 20 cm de ancho, 30 cm de largo y 25 cm de alto, sin restricción de peso.

Por los requerimientos de movilidad mencionados anteriormente, se colocaron dos servomotores en el eje transversal, lo que permite realizar movimientos controlados,

A. Gelbukh, G. Sidorov, W. Olán, J. Vera (Eds.) Recientes avances en la ciencia de la computación en México, pp.117-127.

como se puede ver en la figura 1, sin perder en momento alguno la posición actual. Para equilibrar el robot este se apoya en una tercera rueda que gira libremente sin representar fricción significativa al sistema, obteniendo un robot que es capaz de alcanzar altas velocidades y máxima maniobrabilidad.

La interacción con el medio se realiza mediante sensores infrarrojos, que detectan la existencia o no de paredes cercanas, estos sensores trabajan con modulación estándar de 40 KHz. En cuanto a los servomotores, estos ya incluyen sus respectivas etapas de amplificación de potencia y se les envía la dirección y velocidad de giro a través de un pulso cuadrado de ancho proporcional a la velocidad deseada. La posición de cada una de las llantas del robot se mide con dos sensores infrarrojos ubicados en cada una de ellas, de tal forma que proporcionan el numero de vueltas que se realiza, por lo que se pueden dar las vueltas de 90°, 180° o 360° con toda precisión, a su vez esto sirve para conocer cuanto ha avanzado el robot.

Como plataforma de programación y control el sistema cuenta con un microcontrolador PIC16F877 [1], que además de controlar sensores y motores, debe contener el programa principal, almacenar la estructura del laberinto en memoria y desarrollar todas las estrategias de control y tareas complementarias que se requieren para el funcionamiento del robot. Para soportar todo esto, se escogió este microcontrolador ya que posee una arquitectura RISC de alto rendimiento que cuenta con 368 bytes de memoria RAM, 256 bytes de EEPROM para datos y 8192 localidades de memoria FLASH de 14 bits por localidad para almacenamiento del programa y datos permanentes, además cuenta con periféricos tales como un ADC de 10 bits con multiplexor de 8 canales, 2 salidas PWM, 3 contadores-temporizadores, comunicación SPI, I^2C, USART, 33 pines de entrada y salida digital de uso general y 13 canales de interrupción. Todos los programas se desarrollan en lenguaje C utilizando el compilador PICC de CCS, el cual tiene como principal limitación, sobre todo para esta tarea, que no pueden manejarse apuntadores a la memoria FLASH.

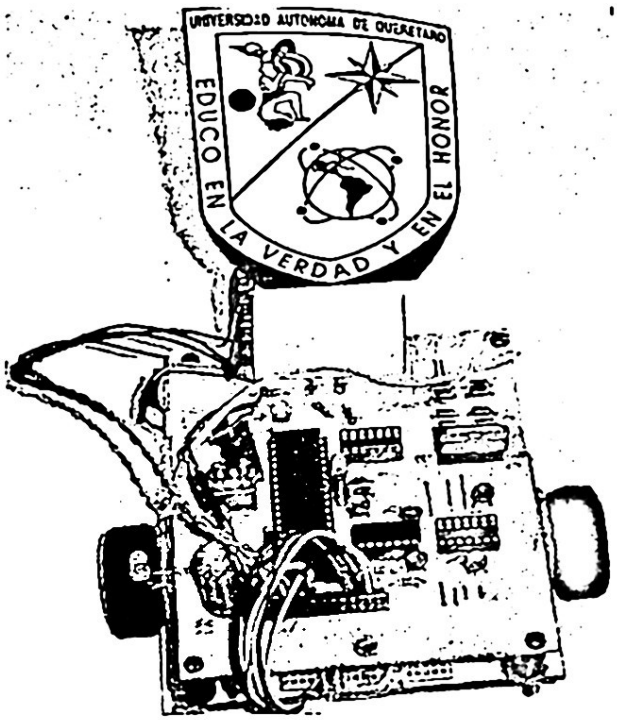

Fig. 1. Imagen del robot

Para equilibrar el robot este se apoya en una tercera rueda que gira libremente sin representar fricción significativa al sistema, obteniendo un robot que es capaz de alcanzar altas velocidades y máxima maniobrabilidad.

La interacción con el medio se realiza mediante sensores infrarrojos, que detectan la existencia o no de paredes cercanas, estos sensores trabajan con modulación estándar de 40 KHz. En cuanto a los servomotores, estos ya incluyen sus respectivas etapas de amplificación de potencia y se les envía la dirección y velocidad de giro a través de un pulso cuadrado de ancho proporcional a la velocidad deseada. La posición de cada una de las llantas del robot se mide con dos sensores infrarrojos ubicados en cada una de ellas, de tal forma que proporcionan el numero de vueltas que se realiza, por lo que se pueden dar las vueltas de 90°, 180° o 360° con toda precisión, a su vez esto sirve para conocer cuanto ha avanzado el robot.

Como plataforma de programación y control el sistema cuenta con un microcontrolador PIC16F877 [1], que además de controlar sensores y motores, debe contener el programa principal, almacenar la estructura del laberinto en memoria y desarrollar todas las estrategias de control y tareas complementarias que se requieren para el funcionamiento del robot. Para soportar todo esto, se escogió este microcontrolador ya que posee una arquitectura RISC de alto rendimiento que cuenta con 368 bytes de memoria RAM, 256 bytes de EEPROM para datos y 8192 localidades de memoria FLASH de 14 bits por localidad para almacenamiento del programa y datos permanentes, además cuenta con periféricos tales como un ADC de 10 bits con multiplexor de 8 canales, 2 salidas PWM, 3 contadores-temporizadores, comunicación SPI, I^2C, USART, 33 pines de entrada y salida digital de uso general y 13 canales de interrupción. Todos los programas se desarrollan en lenguaje C utilizando el compilador PICC de CCS, el cual tiene como principal limitación, sobre todo para esta tarea, que no pueden manejarse apuntadores a la memoria FLASH.

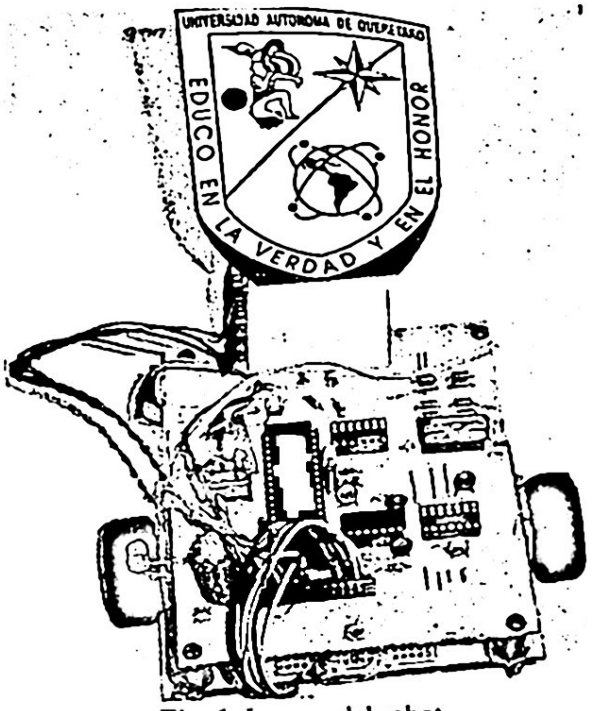

Fig. 1. Imagen del robot

2.2 Descripción y análisis de algoritmos

2.2.1 Algoritmos de la mano derecha y de la mano izquierda

Los primeros algoritmos que tenemos que discutir por su popularidad y sencillez, son los algoritmos de la mano derecha y de la mano izquierda, estos algoritmos se basan en seguir una pared del laberinto (izquierda o derecha según el caso) y resultan efectivos en casi todos los casos, se conozca o no el laberinto. Sus principales desventajas, radican en que no resuelven laberintos no conectados (con islas internas) si no se empieza desde una de las paredes exteriores del laberinto y por otro lado la trayectoria que siguen para encontrar la salida dista mucho de ser la óptima, ya que recorre todas las opciones hasta encontrar la salida.

Dados que el problema que se plantea consiste en un laberinto no conectado y el punto de inicio es desconocido, estos algoritmos no pueden ser aplicados, Aunque modificaciones de este algoritmo, como la planteada por Lucas en 1882 [2] que consiste en ir pintando una línea por las trayectorias recorridas logran resolver satisfactoriamente aún este tipo de laberintos, si utilizamos este tipo de algoritmos perderíamos la ventaja que se tiene por conocer la estructura del laberinto con anterioridad.

2.2.2 Algoritmo de recorrido recursivo

Este algoritmo se basa en el recorrido de un camino, regresando fallo cuando encontramos una pared, y éxito si encontramos una salida, en cualquier otro caso, intentamos recorrer de manera recursiva en cualquiera de las 4 direcciones posibles, cuando intentamos una nueva dirección trazamos una línea, y borramos una línea cuando encontramos fallo. Este método encontrará siempre una salida, si es que existe, pero en la mayoría de los casos, no será el camino mas corto, y requiere recorrer una gran parte del laberinto para salir de él. El hecho de que la competencia sea por tiempo, y la arquitectura que estamos manejando, impide manejar la recursividad de manera sencilla, además del hecho que de las 2 posibles salidas existentes en el laberinto, solo una de ellas es válida y no existe manera de descartar la otra utilizando este algoritmo.

2.2.3 Algoritmo de Tremaux [3]

Esta es la versión del algoritmo anterior pero de manera tal que pueda ser ejecutada por un humano. Se recorre progresivamente el grafo del laberinto. Este asegura que todos los vértices dentro de cierta distancia sean visitados. Es fácil alcanzar los vértices a una distancia 1, y regresar al punto de partida. Cada camino de estos es marcado para llegar de X0 a X1, como el camino de entrada. Para llegar a un vértice a una distancia 2, desde el origen, se selecciona el camino anterior y se vuelve a marcar. Cuando todos los caminos posibles de X1 son visitados, se marca el camino como cerrado, y se toma el siguiente camino desde el origen, hasta que no queden caminos disponibles desde el origen.

2.2.4 Algoritmo de Tarry [3]

Este método publicado por Tarry en 1895, crea un camino cíclico en el laberinto, pasando por cada cruce una y solo una vez en cada dirección. Iniciando en una

posición A, selecciona un camino cualquiera y lo marca de manera especial, cuando llega al siguiente cruce, puede haber sido visitado o no, y lo marca acorde con ello. El algoritmo termina en el lugar de inicio, habiendo recorrido cada cruce, por lo menos una vez en cada dirección.

Una mejora a los dos algoritmos anteriores fue propuesta por Fraenkel [4], [5], la cual asegura que cada cruce será transitado, a lo más, una vez en cada dirección.

Estos tres algoritmos anteriores, fueron descartados porque, al igual que los de mano izquierda y derecha, suponen que el laberinto no se conoce previamente, por lo que requieren recorrer una gran parte del laberinto para encontrar la salida.

2.2.5 Algoritmo de Pledge

Otro algoritmo de muy sencilla implementación, es el de Pledge [6] que consiste en mantener la dirección en que se esta apuntando en cada momento, a pesar de su simplicidad, al igual que los algoritmos de la mano izquierda y derecha, no es aplicable a todo tipo de laberintos, puesto que en ciertos casos, como los encontrados en esta categoría, el robot queda ciclado en un solo camino.

2.2.6 Algoritmo de Lee

El algoritmo planteado por Lee[7], consta de dos partes, primero la creación del mapa del laberinto en memoria, por medio de recorridos sucesivos. Durante estos recorridos, se anotan las paredes que tiene cada celda conforme el robot las visita., así, si la celda solo tuviera pared al norte, se le asigna un 8, pero si además tiene también pared al este, se le asigna un 10, 8 de la pared norte y 2 de la pared este. Esta parte, permite ver que requeriremos una cantidad de memoria, de tamaño suficiente para, por lo menos, almacenar un arreglo con la información de cada celda del laberinto.

Una vez que se tiene un mapa del laberinto, se asignan un número a cada celda, de la siguiente manera:

Iniciamos colocando un 0 en cada una de las celdas del laberinto que se consideren como meta. Deberemos recorrer el laberinto columna por columna en repetidas ocasiones. Para cada celda en la que esté el robot, se busca en nuestro mapa si dicha celda aún no está numerada y si las celdas a las que puedo llegar a partir de la actual ya se encuentran numeradas, si no es así, en esta visita no haremos nada. En caso de que por lo menos alguna de las celdas vecinas ya se encuentre numerada, y la actual no, suponiendo que el número de Lee más pequeño de las celdas vecinas sea n, a la celda actual le asignamos n+1.

Este proceso se repite hasta que ya no pueda realizar cambios en el mapa, lo que supondrá que ya todas las celdas se encuentran numeradas.

Lo anterior supone que ya se tiene en memoria el mapa del laberinto. De no ser así, se puede modificar la generación de números, suponiendo que el laberinto solo tiene paredes en los extremos y calculando los números. Al cambiar de una celda a la siguiente, verificamos si en realidad no existen más paredes, en caso contrario, actualizamos el mapa y recalculamos los números.

Una vez generados los números es fácil encontrar el camino dentro del laberinto, y además se asegura que será siempre el camino más corto, puesto que estando en

cualquier celda del laberinto basta con moverse a la celda vecina que contenga un número menor.

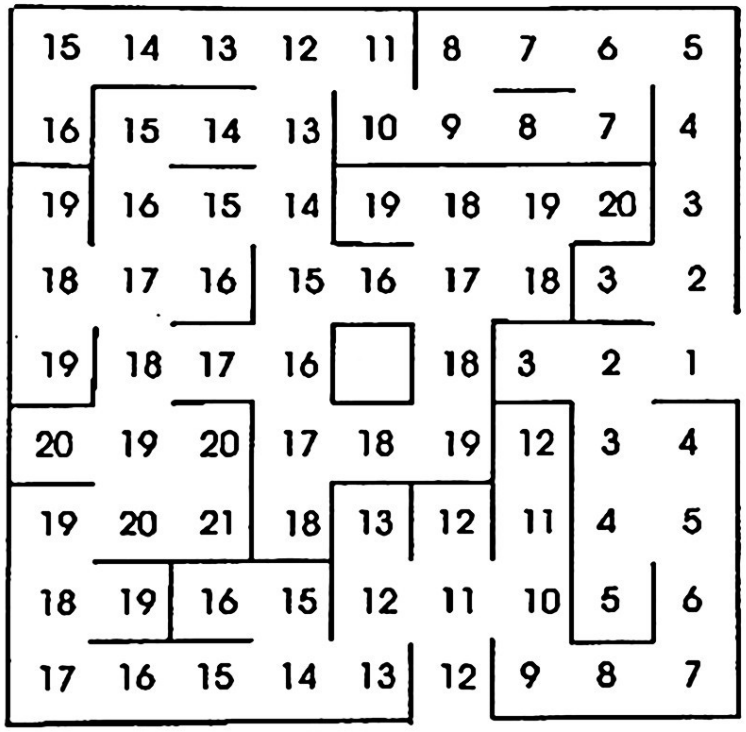

Fig. 2. Estructura del laberinto, con valores de distancia a la salida

Este algoritmo fue escogido como base para la solución de nuestro problema, porque además de resolver todo tipo de laberintos y garantizar la ruta óptima, plantea la solución del problema en dos etapas, primero conocer el laberinto y después recorrerlo, esta característica lo hace muy fácil de adecuar a nuestras necesidades, ya que conocemos de antemano el laberinto y podemos realizar el análisis y la numeración del mismo con antelación.

2.3 Desarrollo e implementación del algoritmo

Debido a las características del problema y a la plataforma sobre la que se va a programar, se escogió desarrollar un algoritmo de bajos requerimientos basado principalmente en el algoritmo de Lee, pero aprovechando el conocimiento previo del laberinto, por lo que el proceso de asignación de números se puede realizar incluso antes de programar el robot. Esto junto con el desarrollo de un método que nos permite identificar en el menor número de movimientos posibles (generalmente entre 3 y 6 movimientos), la ubicación y orientación actual del robot, proporciona la información suficiente para después seguir la segunda parte del algoritmo de Lee hasta llegar a la salida, lo cual se realiza por la ruta óptima.

Primeramente estudiamos el laberinto de prueba de la figura 2 y numeramos cada casilla con la distancia en casillas que se requiere recorrer para llegar desde ahí a la salida. Por lo que para salir de él, sólo es necesario moverse siempre hacia una casilla de menor valor.

Para almacenar el conocimiento previo que se tiene del laberinto, se utilizan 4 arreglos de 9x9 en memoria flash, cada uno de estos arreglos guarda toda la información del laberinto visto desde una de las direcciones posibles (norte, sur, este y oeste) en cada localidad del arreglo (14 bits), se almacena el número de casilla en los 6 bits más significativos y las paredes que tiene esa casilla en los 4 bits menos significativos, quedando como se ilustra en la figura 3.

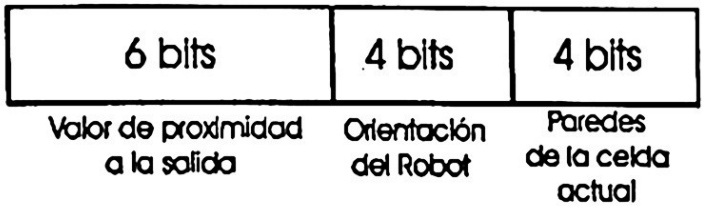

Fig. 3. Forma en que se almacena la información de cada casilla

Además de tener los cuatro arreglos de 9x9 que almacenan el laberinto visto con diferentes orientaciones, se tiene un arreglo más de 9x9 que va generando el laberinto según se mueve el robot, dicho arreglo se almacena en RAM. Un ejemplo de cómo se almacena este laberinto visto desde la orientación de la figura 2, se muestra en la figura 4, en esta, el primer número sigue indicando los 6 bits superiores que corresponden a la distancia a la salida y el segundo número identifica las paredes de cada casilla.

0F/9	0E/C	0D/C	0C/8	08/A	08/9	07/C	06/8	05/A
10/7	0F/9	0E/C	0D/2	0A/5	09/4	08/C	07/6	04/3
13/B	10/1	0E/8	0D/2	13/D	12/8	13/8	14/E	03/3
12/1	11/0	10/6	0F/1	10/C	11/0	12/6	03/D	02/1
13/7	12/1	11/C	10/2		12/3	03/D	02/8	01/4
14/D	13/0	14/A	11/2	12/C	13/6	0C/B	03/1	04/A
13/9	14/4	15/6	12/7	0D/B	0C/B	08/3	04/1	05/2
12/1	13/E	01/D	0F/A	0C/1	08/0	0A/2	05/7	06/3
11/5	10/C	0F/C	0E/4	0D/6	0C/3	09/5	08/C	07/6

Fig. 4. Volcado de memoria de una orientación del laberinto, los números están en hexadecimal

Cuando el recorrido empieza, el robot identifica las paredes que tiene a su alrededor y obtiene un número de identificación de la casilla en cuestión, este número lo va comparando, uno por uno, con la información de las paredes de cada una de las localidades del primer arreglo y supone que se encuentra en la primer casilla compatible del primer arreglo. Después de esto se mueve a la siguiente casilla en

dirección a la salida, si ésta también es compatible con la posición en la que supone que está, continua de la misma forma, si no es compatible, descarta la posición original supuesta y busca la siguiente que sea compatible. Este procedimiento se sigue hasta que la determinación de la posición real del robot sea inequívoca. Dentro de este proceso, si el primer arreglo es agotado sin encontrar las casillas que correspondan, significa que la orientación inicial del robot no era la supuesta y se sigue buscando en los siguientes arreglos.

2.4 Principales problemas de codificación

Como ya se había mencionado, uno de los principales problemas que se tienen es que el compilador usado no maneja apuntadores a la memoria de programa (FLASH), por lo que para poder manejar los arreglos que almacenan la información del laberinto hubo que programar funciones para lectura de la memoria FLASH, estas funciones se desarrollaron de forma tal que ingresan como parámetro las coordenadas de la casilla que se quiere leer y regresan el valor de la misma, una función lee la parte baja que contiene la información de las paredes y otra lee el valor de distancia a la salida.

Otro problema importante es que además de estar ejecutando el algoritmo descrito, el microcontrolador debe estar tomando las lecturas permanentes de los sensores de presencia, lecturas de las posiciones de los motores y mandando de forma continua las señales de control a los motores. Para poder hacer todo esto al mismo tiempo, se utilizó un esquema de programación orientada a eventos, aprovechando muchos de los múltiples canales de interrupción del PIC16F877.

Para el control de los motores se tiene que enviar una señal cuadrada de 40 ms de periodo en donde el tiempo en alto es proporcional a la velocidad del motor, si el tiempo en alto es de 1.5 ms el motor permanecerá en reposo, mientras que se moverá a la derecha con tiempos de encendido mayores y a la izquierda con tiempos menores, la velocidad del motor será igual a la diferencia entre el tiempo de encendido y 1.5 ms.

Para generar dos señales de este tipo para los dos motores, se utilizó el timer 1, con el se hace el calculo del tiempo deseado se configura y posteriormente el timer generará una interrupción cuando el tiempo haya concluido.

Los sensores ópticos (GP2D02 [8]), requirieron ser colocados de manera particular, puesto que, como se puede apreciar en la figura 5, la distancia mínima confiable de medición es de 8 cm, por ello fueron montados en el centro del robot, por debajo de la placa de circuitos, asegurando de esta manera que sea el mismo cuerpo del robot el que garantice esa distancia mínima entre el sensor y las paredes.

Los sensores se leen a través de una interfaz serie síncrona, generando un tren de pulsos como el de la figura 6.

El inicio del ciclo de lectura del sensor, inicia al cambiar la señal de Vin del sensor de alto a bajo, y se activa el canal de interrupción externa. Una vez que el sensor haya realizado la medición y esté listo para enviar el dato, la señal de datos cambia de estado, activando la interrupción externa del microcontrolador. Durante esta interrupción, se habilita la comunicación serie y se habilita la interrupción de la misma. Al activarse esta interrupción, el dato ya se encuentra disponible en el buffer de datos, leemos el dato y habilitamos la interrupción del timer 2, ajustamos el

periodo del mismo al mínimo de 1.5 ms necesarios para que el sensor este listo para realizar una nueva medición. Una vez que se activa la interrupción del timer 2, podemos iniciar nuevamente el ciclo, enviando la señal de inicio de medición, desactivando la interrupción del timer 2 y activando la interrupción externa.

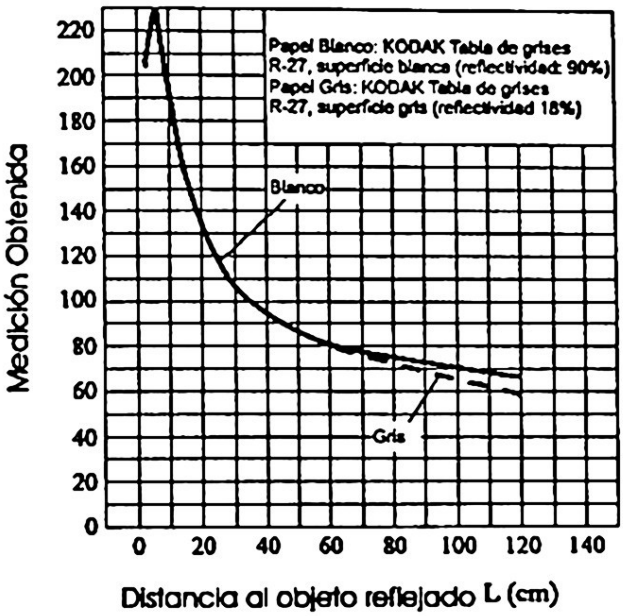

Fig. 5. Gráfica de medición del sensor infrarrojo, el eje vertical representa el número digital que retorna el sensor

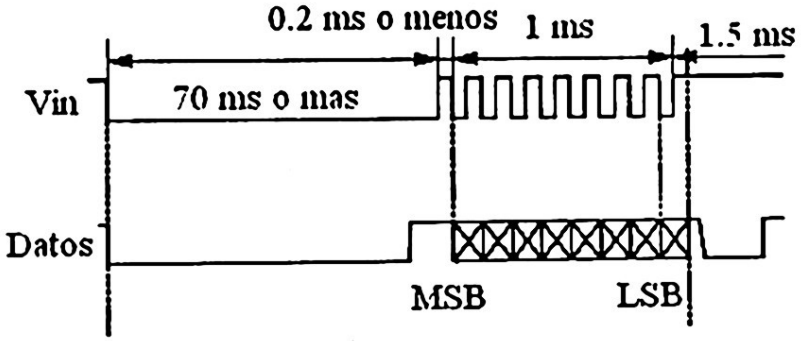

Fig. 6. Gráfica de tiempos del sensor infrarrojo.

Realizar diferentes tareas dentro de un mismo microcontrolador genera ciertos problemas en el momento en que estas se tienen que realizar al mismo tiempo, esto es, si se deseara recorrer una distancia para posicionarse en la casilla siguiente, esto se podría realizar mediante una rutina que llevará cierto tiempo en producirse y con la cual se realizara la tarea; si en el tiempo en que se encuentra trabajando la rutina tienen que realizarse otras tareas para leer los sensores, medir distancia recorrida, etc., esta información se perdería debido a que para ser realizados se tiene que haber terminado la rutina en curso, en cambio con los eventos no importa que ocurrieran

varios eventos dentro de la realización de la tarea porque en el momento que estos se producen el microcontrolador los realiza y regresa a terminar la tarea en curso.

3 Resultados

Las pruebas realizadas, nos permitieron observar que en la mayoría de los casos el robot logra identificar su posición en alrededor de 5 o 6 movimientos, aunque en general podríamos decir que esto depende en gran medida de cual sea su orientación original, ya que cuando se orienta inicialmente al norte determina su posición en 2 o 3 movimientos y cuando su orientación es al oeste llegan a ser hasta 7 u 8 movimientos.

Cabe destacar que la mayor parte de las dificultades que se tuvieron en le implementación del algoritmo, no se dieron en cuanto al manejo de las decisiones, sino en la etapa de control, donde el manejo y calibración de sensores, la medición de distancias, tolerancia de incertidumbres etc. requirieron un trabajo de puesta a punto muy considerable.

4 Conclusiones

Consideramos que la estrategia de programar todas las etapas de control del robot a través de eventos, fue uno de los grandes aciertos en este trabajo, logrando separar este problema, del algoritmo modificado de Lee que se planteó para la elección de la ruta. ya que de lo contrario habría resultado casi imposible controlar cada una de las partes además de hacerlas interactuar con el resto.

Otra posible solución que se pudo haber seguido es la utilización de dos microcontroladores para dividir las tareas, uno de ellos programado para controlar únicamente el movimiento de los motores para no afectar con otras cosas la eficiencia de un recorrido exacto y el otro con la lectura de los sensores y la ejecución del algoritmo.

En cuanto al algoritmo utilizado para la elección de la ruta a seguir, podríamos concluir que es óptimo, dadas las condiciones y la información con que cuenta el robot. Sin embargo, si pudiésemos dotar al robot de mas información, se podrían plantear nuevos algoritmos que reduzcan drásticamente la distancia entre el mejor y peor caso, además de reducir también el número de movimientos necesarios para identificar la posición del robot, logrando que esta identificación sea casi inmediata. En este sentido, la información más relevante y accesible seria la orientación del robot, para lo cual lo tendríamos que incorporarle una brújula electrónica; además sería útil conocer, no solo la existencia o no de cada pared, sino también la distancia en casillas que existe en cada sentido hasta la siguiente pared.

Cabe mencionar, que aunque para el laberinto planteado en este trabajo, el algoritmo presenta un alto grado de eficiencia, si no se modificase y se utilizara en un laberinto de mayores dimensiones, se reduciría considerablemente la eficiencia del peor de los casos, por lo que se plantea como trabajo a futuro la incorporación al robot de los medios necesarios para recabar los datos de orientación y distancia a la pared

mas cercana en cada dirección, lo que nos llevará también a las respectivas modificaciones del algoritmo.

Referencias

[1] Microchip_Technology, *PIC16F87X Data Sheet*: Microchip Technology Inc., 2001.
[2] E. Lucas, *Recréations mathématiques*, vol. 1, 1882.
[3] O. Ore, "Theory of Graphs," *American Math. Soc. Colloquium Publications*, vol. XXXVII, 1962.
[4] A. S. Fraenkel, "Economic traversal of labyrinths," in *Mathematics Magazine*, vol. 43, 1970, pp. 125-130.
[5] A. S. Fraenkel, "Economic traversal of labyrinths," in *Mathematics Magazine*, vol. 44, 1971, pp. 12.
[6] A. d. H. Abelson, *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*: MIT Press, 1980.
[7] C. Y. Lee, "An algorithm for path connections and its applications," *IRE transactions on electronic computers*, vol. 10, pp. 346-365, 1961.
[8] SHARP_CORPORATION, *Databook 1999/2000*, 2000.