Capítulo 8

Análisis de los algoritmos de solución de laberintos

La solución de laberintos ha sido estudiada por muchos años en el campo de las matemáticas, por lo que existen numerosos algoritmos que pueden ser aplicados.

Un algoritmo es una serie de instrucciones que instruyen paso a paso a una máquina para la solución de un problema específico, la solución en nuestro caso es la ruta que le toma al robot navegar desde el inicio hasta la meta, para lo que requiere conocer información sobre la presencia de las paredes [32].

Aunque existen una gran variedad de algoritmos de solución de laberintos que pueden aplicarse a los robots micromouse, se analizarán tres que resultan especialmente aptos para el robot objeto de ésta tesis, estos son:

Algoritmo de azar o *random algorithm*.

Algoritmo de seguidor de paredes.

Algoritmo de inundación o fill flood algorithm.

8.1 Algoritmo de azar

Este algoritmo consiste en que el robot toma arbitrariamente la dirección en cada uno de los nodos que corresponden a las celdas que tienen más de una posible dirección [36], de esta manera el robot tienen altas posibilidades de llegar a la meta entre otras ventajas como las que se numeran a continuación:

- No se requiere de mucha capacidad de procesamiento por parte del controlador.
- Se simplifica el diseño del sistema electrónico.
- Al requerir de menos elementos de control se reduce el peso y se pueden colocar sistemas de alimentación de más duración.

Por otro la implementación de este algoritmo tiene las siguientes desventajas:

- Ausencia total de inteligencia.
- No se puede saber en ningún momento la posición u orientación dentro del laberinto.
- Así tampoco el robot sabe en qué momento ha llegado a la meta.
- No se puede estimar el tiempo que le tomará al robot completar la tarea de encontrar la solución del laberinto por lo que puede llevarse más de la cantidad de tiempo límite que se tiene respecto al reglamento de las competencias.

Por las razones anteriores el algoritmo de azar representa una elección poco adecuada.

8.2 Algoritmo de seguidor de paredes

Este algoritmo se basa en la regla de que siguiendo siempre por la derecha o por la izquierda se encontrará invariablemente la salida de una laberinto.

El algoritmo se aplica de la siguiente manera [33]:

Basados en el seguidor de paredes por la izquierda se aplican los pasos en la secuencia que se enlista enseguida:

Paso 1: Checa la presencia de la pared izquierda.

Paso 2: Si la pared izquierda está presente entonces se hace la bandera flagl = 1, si no está presente, entonces se hace que el valor de la bandera tenga valor 0, esto es flagl=0.

Paso 3: Si flagl = 1 entonces se pasa al paso 4, caso contrario, flagl = 0 hacer un giro izquierdo de 90 grados.

Paso 4: Comprobar la presencia de la pared frontal.

Paso 5: Si la pared frontal está presente se hace que la bandera flagf = 1, si no, flagf = 0.

Paso 6: Si la bandera flagf = 0, el robot se moverá hacia adelante en línea recta, si la bandera tiene el valor complementario entonces se mueve hacia delante.

Ventajas de este algoritmo

No se necesitan controladores complejos como los procesadores, microcontroladores, DSP, etc.

Pueden utilizarse solo algunos sensores y estos pueden ser tan simples como los detectores mecánicos de contacto.

Se puede simplificar el sistema de control electrónico lo que lo hace menos propensos a fallas.

Suele ser el primer algoritmo de solución de laberintos en la etapa de pruebas ya que permite la solución efectiva de determinados laberintos y permite la depuración de sensores y otros elementos electrónicos para pasar a algoritmos más sofisticados con mejores prestaciones en cuanto a velocidad e inteligencia.

Por el lado de las desventajas tenemos que:

No representa una técnica de inteligencia artificial.

El robot solo resuelve el laberinto, no tiene la habilidad de aprender o reconocer la ruta más corta o que lo lleve lo más rápido posible desde el inicio hasta la meta.

8.3 Algoritmo de inundación

Se basa en la comparación física de que se dejamos fluir agua desde el inicio hasta la meta esta lo hará por la ruta más eficiente que generalmente es el camino de menor distancia desde el punto de inicio hasta la meta.

Ventajas

Este algoritmo puede valerse de la representación matricial de las celdas del laberinto, lo que es factible a su vez representarse eficientemente en la memoria de un sistema digital como una computadora o un microcontrolador. De esta manera el algoritmo de control representará características como la posición y la orientación mediante vectores de números [34].

Desventajas

El algoritmo debe depurarse hasta ser contenido espacio limitado que suelen tener los controladores para sistemas embebidos además debe tenerse en cuenta la cantidad de datos que puede procesar un sistema embebido para permitir la operación en tiempo real que se requiere para tener posibilidades de hacer una puntuación que permita ganar.

El robot utiliza los valores de las distancias para moverse a través del laberinto. El valor de las distancias, que representan que tan lejos se encuentra la celda de destino, se enumera en orden descendente hasta que el robot alcanza el final.

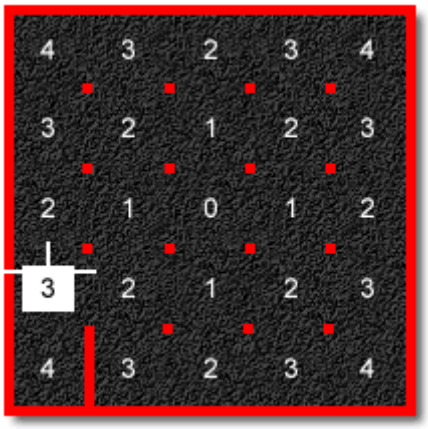

Figura 8.1 Laberinto ejemplo

Mientras el robot encuentra nuevas paredes durante su exploración, los valores de las distancias necesitas ser actualizados. En vez de llenar el laberinto entero con valores, solo se cambia los valores que necesitan ser actualizados. Por ejemplo si el robot se encuentra en la posición que se muestra en la siguiente figura 8.2 y se mueve hacia adelante una celda y descubre una pared.

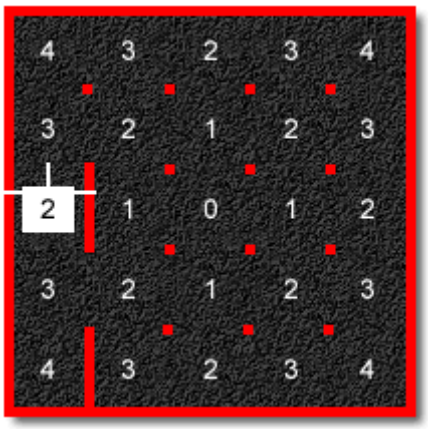

Figura 8.2 Avance del robot en el laberinto

El robot no puede ir a su derecha ni a su izquierda, pero ir hacia a adelante o hacia atrás significa aumentar los valores de las distancias, lo que no nos interesa hacer, así que los valores necesitan ser actualizados, cuando nos encontramos con éste escenario, deberemos de seguir la regla:

Si la celda actual no es la celda de destino, este valor deberá actualizarse de la forma: uno más el mínimo valor de sus vecinos abiertos.

En el ejemplo que nos ocupa, el mínimo valor de las celdas vecinas abiertas es 3. Agregando 1 a este valor resulta en 3 + 1 = 4. El laberinto tiene entonces el siguiente aspecto de la figura 8.4.

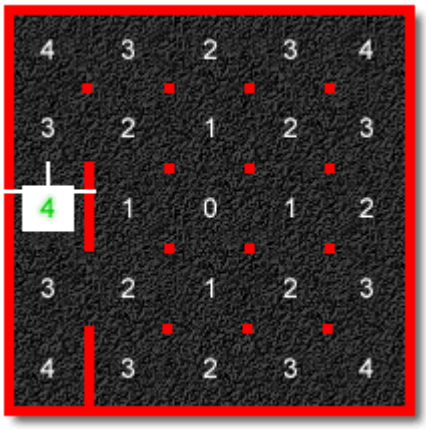

Figura 8.3 Ajuste del valor de las celdas

Hay veces cuando actualizando el valor de las celdas causará que el valor de las celdas vecinas no sigan la regla "1 + el mínimo valor", así que deberán ser actualizadas también. Podemos ver en la figura 8.4 que la celda de arriba y la de abajo tienen celdas vecinas en las cuales el mínimo valor es 2. Agregando 1 es este valor obtenemos el valor correspondiente así que las celdas de arriba y abajo no violan la regla y la actualización de valores se ha realizado de la manera correcta y sin problemas [35].

Ahora que el valor de las celdas ha sido actualizado, el robot puede una vez más seguir por las celdas en orden descendente.

El procedimiento para este algoritmo es como sigue

Actualización del valor de las celdas (si es necesario)

Asegurarse que el stack está vacío

Colocar el valor de la celda actual en el stack

Repetir el siguiente set de instrucciones hasta que el stack este vacío:

{

Leer el valor de la celda en el stack

¿Es el valor de ésta celda 1 + el valor mínimo del valor de sus vecinos abiertos?

No- Cambie el valor de la celda a 1 + el mínimo valor de sus vecinos abiertos y coloque en el stack todos las celdas vecinas abiertas en el stack para ser analizadas

Si- No hacer nada

}

La mayor parte del tiempo el robot el robot es relativamente rápido porque solo actualiza los valores que requieren ser actualizados y así el robot puede hacer su próximo movimiento de forma más rápida.

Cada vez que el robot llega a una celda, el robot debe hacer lo siguiente

1) Actualizar el mapa de las paredes

2) Actualizar el valor de las distancias (solo en caso de ser necesario)

3) Decidir cuál de las celdas tienen el menor valor de distancia

4) Moverse hacia la celda vecina que tenga el menor valor de distancia

8.4 Algoritmo de navegación y solución

El laberinto de la competencia consiste en múltiplos de un cuadrado de 18 cm por 18 cm, pero como las paredes tienen un espesor de 1.2 cm las dimensiones de cada una de las celdas son 16.8 x 16.8, la meta está colocada en uno de los extremos del laberinto al igual que el inicio del laberinto.

La odometría puede realizarse a través del hardware adecuado, por medio de un análisis de las posibilidades que se encuentran disponibles en el mercado local se ha decidido por sistemas adaptados de sistemas encoder, pero estos sistemas tienen la desventaja se no ser muy precisos y de desajustarse frecuentemente [36].

Otra opción sería de optar por motores con sistemas reductores de velocidad que permitan adaptar el torque y con sistemas de medición de la velocidad o mejor aún que puedan cuantificar la distancia de manera confiable.

Debido a la imposibilidad de conseguir motores con las características necesarias se colocaron sensores para poder medir aproximadamente las revoluciones de las ruedas de manera que se pueda tener la medición de la distancia que se ha recorrido. La utilidad de estos elementos se evaluará en las pruebas reales.

Por otro lado la adición de nuevos sensores requiere agregar más elementos electrónicos que puedan adaptar las señales que generan estos, de manera que puedan ser conducidas al controlador que requiere de señales con características específicas de voltaje especialmente.

Se ha seleccionado el algoritmo de la lógica de seguimiento de paredes pues se pudo participar en el Torneo Mexicano de Robótica que se realizó en el ITAM y el reglamento de la competencia micromouse se adaptó para que se utilizara esta metodología de solución.