



Estructuras de Datos y Algoritmos I

Trabajo Práctico Final

Agustín López

Carrera: Licenciatura en Ciencias de la Computación

Facultad de Ciencias Exactas Ingeniería y Agrimensura

1 Actividad 1

En esta actividad se presenta el desarrollo e implementación de un algoritmo de navegación para un robot en un entorno desconocido, donde el robot se mueve a ciegas y solo puede detectar los obstáculos al “chocarse” con ellos. El objetivo es que el robot llegue a la posición objetivo de manera eficiente, evitando obstáculos y explorando nuevas áreas de forma sistemática.

A continuación, se detallan los pasos del algoritmo y las estructuras relevantes para su implementación.

1.1 Descripción del Algoritmo

Problem Algoritmo

El algoritmo para la navegación del robot sigue los siguientes pasos:

1. **Paso 1:** El robot comienza moviéndose en dirección vertical buscando alinearse con la posición objetivo. Cuando el robot logra llegar a la altura del destino o se encuentra con algún obstáculo, intenta moverse en dirección horizontal siguiendo el mismo principio. Este proceso se repite en bucle hasta que el robot no pueda acercarse más al objetivo utilizando este método.
2. **Paso 2:** Cuando el robot llega a una posición en la que no puede avanzar más, desde su posición actual se mueve hacia una celda vecina que no haya sido visitada previamente. Este paso asegura que el robot explore nuevas áreas del mapa y evita el movimiento redundante en regiones ya descubiertas. Después, el robot vuelve al paso 1.
3. **Paso 3:** Si no hay celdas vecinas no visitadas, el robot retrocede una posición por la ruta previamente recorrida y regresa al paso 1.

Este enfoque permite al robot explorar el entorno sin quedar atascado en un bucle infinito y llegar a la posición objetivo de manera eficiente, maximizando la cobertura del área e intentando minimizar el tiempo de búsqueda.

1.2 Estructuras de Datos Utilizadas

Estructura Robot: Para la implementación del algoritmo, una de las estructuras más importantes es Robot, que representa la información con la que cuenta el robot y que luego utiliza para seguir el algoritmo descripto.

```

1  typedef struct {
2      int i;
3      int j;
4  } _Punto;
5  typedef _Punto* Punto;
6
7  typedef enum {
8      LEFT,
9      RIGHT,
10     UP,
11     DOWN
12 } Direccion;
13
14 typedef struct {
15     Punto pos;
16     Punto dest;
17     Pila movimientos;
18     TablaHash visitados;
19 } _Robot;
20 typedef _Robot *Robot;
```

Descripción de los campos:

- **Punto pos y dest:** Representan la posición actual y la posición destino del robot, respectivamente.
- **Pila movimientos:** Una pila implementada con una lista general simplemente enlazada con un puntero al primer elemento que almacena elementos de tipo `Direccion*`. Cuando el robot realiza un movimiento en alguna dirección que no sea un retroceso (por ejemplo, hacia la izquierda), se agrega el movimiento a la pila. De este modo, cuando el robot tenga que realizar un retroceso, el movimiento que se ejecute será en la dirección opuesta a la almacenada en el tope de la pila.
- **TablaHash visitados:** Una tabla hash que almacena las posiciones visitadas por el robot. Esta tabla se utiliza para evitar que el robot visite una celda más de una vez (o dos veces si contamos el retroceso) y para determinar si hay celdas sin visitar en el mapa para posteriormente moverse hacia ellas. La elección de la estructura `TablaHash` se basa en su eficiencia en la búsqueda en tiempo constante, teniendo en cuenta que la búsqueda es algo que ocurre muchas veces mientras se ejecuta el algoritmo. Además, durante la ejecución no se necesita eliminar elementos de la tabla.

2 Actividad 2

En esta actividad, el objetivo sigue siendo el mismo que en la actividad anterior, pero en este caso el robot cuenta con un sensor que le permite reconocer si hay o no obstáculos en cierta cantidad de celdas hacia las cuatro direcciones cardinales (arriba, abajo, izquierda, derecha). El robot se desplaza en un mapa representado por una matriz de dimensiones $N \times M$ que almacena elementos de tipo `char` y debe llegar a un destino específico desde una posición inicial, evitando obstáculos y utilizando el sensor para obtener información sobre su entorno e intentando optimizar el uso del mismo, dado que se considera costoso.

2.1 Descripción del Algoritmo

Problem Algoritmo

El algoritmo para la navegación del robot con sensor sigue los siguientes pasos:

1. **Paso 0:** En este momento el robot no cuenta con nada de información, por lo que usa el sensor para detectar obstáculos y/o casillas libres a su alrededor.
2. **Paso 1:** El robot siempre conoce su posición inicial y la posición destino, entonces utilizando una implementación basada en el algoritmo A* calcula la ruta más corta hacia el destino asumiendo que las casillas desconocidas están libres pero tienen sumado un costo fijo que representa el costo de usar el sensor.
3. **Paso 2:** Una vez que se construye el camino óptimo, el robot se mueve a lo largo de él mientras las casillas sean conocidas, si la siguiente casilla del camino es desconocida utiliza el sensor para asegurarse de que no hay un obstáculo y vuelve al paso 1 recalculando la ruta con la nueva información.

Este enfoque permite al robot explorar el entorno de manera más eficiente y segura, utilizando la información del sensor para evitar obstáculos y encontrar un camino que quizás no sea el más corto pero intentará usar la menor cantidad de veces posible el sensor.

2.2 Estructuras de Datos Utilizadas

```

1  typedef struct {
2      Punto *pos;
3      Punto *dest;
4      Mapa *mapa;
5      DGList camino;
6  } _Robot;
7  typedef _Robot *Robot;
8
9  typedef struct {
10     Punto padre;
11     int costo;
12 } CeldaInfo;

```

Estructuras Robot y CeldaInfo: Para la implementación del algoritmo, estas son las estructuras centrales que representan información clave al momento de calcular la ruta óptima y que el robot se mueva a lo largo de ella.

2.2.1 Robot

Descripción de los campos:

- **Punto *pos y *dest:** Representan la posición actual y la posición destino del robot, respectivamente al igual que en la actividad 1.
- **Mapa *mapa:** Representa la información que tiene el robot sobre el mapa actual. Es una matriz de dimensiones $N \times M$ que almacena elementos de tipo **char** donde cada celda puede tener uno de los siguientes valores:
 1. ‘?’: Celda desconocida.
 2. ‘.’: Celda libre.
 3. ‘#’: Celda con obstáculo.

Esta matriz se irá actualizando a medida que el robot utilice el sensor y será clave al momento de calcular los costos.

- **DGList camino:** Una lista general simplemente enlazada con un puntero al inicio y otro al final que almacena elementos de tipo **char*** que son los caracteres que se imprimirán al final del programa para que el sensor interprete si el robot llegó o no a su destino. El puntero al inicio es para poder recorrer la lista al momento de imprimir la ruta y el puntero al final es para poder agregar elementos al final de la lista en tiempo constante cada vez que el robot realiza un movimiento.

2.2.2 CeldaInfo

Descripción de los campos:

- **Punto padre:** Un punto que representa el nodo padre de una celda en la matriz. Esto permite seguir el camino de regreso desde el destino hasta la posición inicial una vez que el camino más corto ha sido calculado.
- **int costo:** Representa el costo acumulado de llegar a esta celda desde la posición inicial del robot durante el cálculo de la ruta. Este campo es clave para aplicar el algoritmo, donde se minimiza este costo para encontrar el camino más corto.

2.3 Algunas detalles relevantes para el cálculo y seguimiento de la ruta óptima

2.3.1 Cálculo de Costos

El cálculo de costos se basa en una matriz $N \times M$ de `CeldaInfo`, que se inicializa con los valores de costo establecidos en `INT_MAX` y con el puntero a la celda padre configurado como el punto `(-1, -1)`, indicando que la celda no tiene un parente asignado.

El proceso comienza añadiendo la posición inicial del robot a una cola de prioridad con un costo de 0. La matriz de costos se actualiza con este valor inicial.

A continuación, se extraen los elementos de la cola de prioridad y se calculan los costos de las celdas adyacentes utilizando el siguiente criterio:

- Para una celda que no es un obstáculo, el costo total $f(i, j)$ se calcula como la suma del costo acumulado $g(i, j)$ de llegar a esa celda desde la posición inicial, más una heurística $h(i, j)$ que estima el costo restante para llegar desde $c_{i,j}$ hasta el destino. Formalmente, esto se expresa como:

$$f(i, j) = g(i, j) + h(i, j)$$

- Para una celda desconocida, el costo total $f(i, j)$ se ajusta añadiendo un valor fijo C_{sensor} que representa el costo asociado al uso del sensor. Así, el costo se calcula como:

$$f(i, j) = g(i, j) + h(i, j) + C_{\text{sensor}}$$

- Para una celda que es un obstáculo, se asigna un costo extremadamente alto:

$$\text{INT_MAX},$$

para garantizar que esta celda no se considere en el cálculo de la ruta.

Si el costo calculado para una celda es menor que el costo previamente almacenado en la matriz, se actualiza el valor y se modifica el puntero al parente de la celda. Además, se inserta un nodo en la cola de prioridad con el nuevo costo calculado. Este proceso continúa hasta que la cola de prioridad se vacíe o se alcance la celda de destino. Dado que se asume que siempre hay un camino válido desde la posición inicial del robot hasta el destino, la cola de prioridad no debería vaciarse antes de llegar a la celda de destino.

2.3.2 Seguimiento de la ruta:

Una vez que se calcula la ruta óptima, el robot simplemente reconstruye la ruta del destino siguiendo al nodo parente de la matriz de `CeldaInfo` hasta llegar a la posición inicial. De esta forma, se obtiene la ruta óptima que el robot debe seguir para llegar a su destino.

Cuando el robot comienza a moverse a lo largo de este camino. Si el robot encuentra una celda desconocida, utiliza el sensor para obtener información y luego recalcula el camino desde su posición actual hasta el destino con la nueva información disponible.