

Robótica Móvil

Práctica 5: Pathfinding

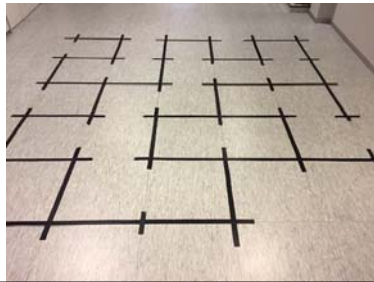
Robótica, Sensores y Actuadores
2019-2020

Representaciones de un laberinto

- **En forma de grafo:**
 - Los nodos son cruces en el laberinto
 - Las aristas son las rectas entre cruces
 - Un nodo puede tener como máximo 4 aristas adyacentes
 - Grafo con pesos: El peso puede representar la distancia al siguiente vértice, u otros parámetros.
- **En forma de Matriz (NxN):**
 - Cada nodo tiene una coordenada X e Y
 - Hay una relación entre las coordenadas de la matriz y la forma del grafo:

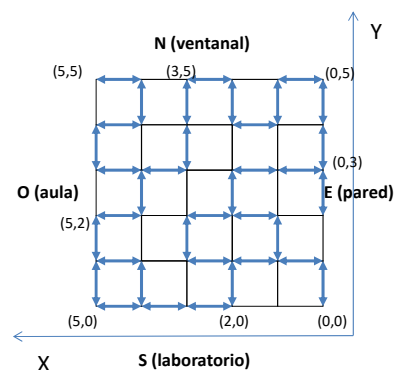
Características de nuestro laberinto

- Las aristas son líneas rectas entre dos vértices, siempre del mismo tamaño.
- En cada vértice podrán converger hasta cuatro aristas formando ángulos de 90° entre ellas
- Las orientaciones serán: NORTE, SUR, ESTE y OESTE
- Como referencia, la orientación OESTE es aquella en la que el robot se orienta hacia la Facultad de Derecho.



Estructura de laberinto.xml

```
<Laberinto nombre="Laberinto" dim_x="n"
  dim_y="m">
  // en nuestro caso n=6, m=6
  - <Nodo x="a" y="b">
    <Camino dir="norte" />
    <Camino dir="sur" />
    <Camino dir="este" />
    <Camino dir="oeste" />
  </Nodo>
  // Sólo deben aparecer los caminos
  existentes
  // El amino es bidireccional, hay que
  señalarlo en los nodos de ambos extremos.
  ...
</Laberinto>
```



Estructura de laberinto.xml

Por ejemplo:

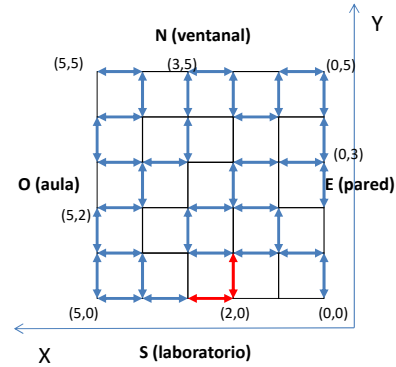
```
- <Nodo x="2" y="0">
  <Camino dir="norte" />
  <Camino dir="oeste" />
</Nodo>
```

Quiere decir que desde el nodo (2,0) se puede avanzar en las direcciones:

- norte, al nodo (2,1)
- oeste, al nodo (3,0)

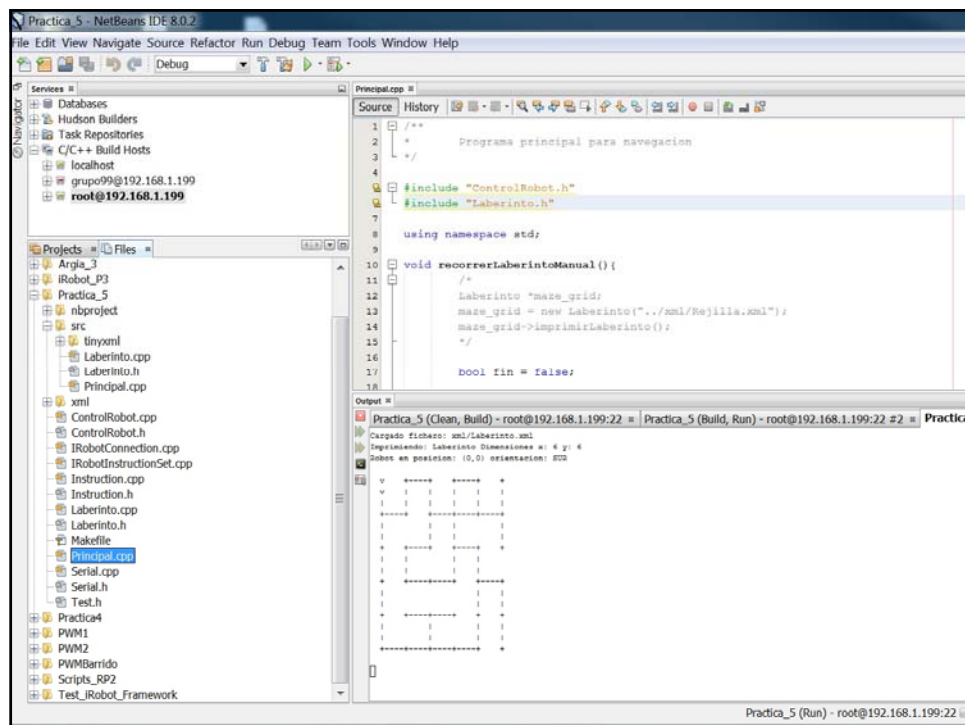
// Sólo aparecen los caminos existentes

// El camino es bidireccional: debe aparecer en los nodos de ambos extremos.



Laberinto.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
-<Laberinto nombre="Laberinto" dim_x="6"
  dim_y="6">
-<Nodo x="0" y="0">
  <Camino dir="abajo" />
</Nodo>
-<Nodo x="1" y="0">
  <Camino dir="abajo" />
  <Camino dir="derecha" />
</Nodo>
-<Nodo x="2" y="0">
  <Camino dir="abajo" />
  <Camino dir="derecha" />
  <Camino dir="izquierda" />
</Nodo>
-<Nodo x="3" y="0">
  <Camino dir="derecha" />
  <Camino dir="izquierda" />
</Nodo>
-<Nodo x="4" y="0">
  <Camino dir="abajo" />
  <Camino dir="derecha" />
  <Camino dir="izquierda" />
  <Camino dir="derecha" />
</Nodo>
...
-<Nodo x="1" y="5">
  <Camino dir="izquierda" />
  <Camino dir="arriba" />
</Nodo>
-<Nodo x="2" y="5">
  <Camino dir="arriba" />
  <Camino dir="derecha" />
</Nodo>
-<Nodo x="3" y="5">
  <Camino dir="izquierda" />
  <Camino dir="arriba" />
</Nodo>
-<Nodo x="4" y="5">
  <Camino dir="arriba" />
  <Camino dir="derecha" />
</Nodo>
-<Nodo x="5" y="5">
  <Camino dir="izquierda" />
  <Camino dir="derecha" />
</Nodo>
...
</Laberinto>
```



Programa de Ejemplo

- El fichero "src_mapping.zip" contiene un ejemplo de cómo funcionan las clases y métodos añadidos para representar mapas en un programa.
- El algoritmo principal del ejemplo se encuentra en el fichero **Principal.cpp** y usa varias funciones encapsuladas en la clase **Laberinto.h/cpp**.
- El programa muestra una representación del laberinto del laboratorio y una representación gráfica del robot se mueve a través de él: >>>.
- Se pueden usar los siguientes comandos:
 - **a (avanza)**: El robot se mueve al siguiente nodo al que esté orientado (si es posible).
 - **n (norte)**: El robot orienta su posición al norte
 - **e (este)**: El robot orienta su posición al este
 - **w (oeste)**: El robot orienta su posición al oeste
 - **s (sur)**: El robot orienta su posición al sur
 - **imprimir**: Se muestra por pantalla el camino recorrido por el robot
 - **salir**: Finaliza el programa

```

imprimiendo el camino:
[0,0] [0,1] [1,1] [1,0] [2,0] [2,1] [2,2]
Imprimiendo: Laberinto Dimensiones x: 6 y: 6
Robot en posición: <2,2> orientacion: OESTE

```

Realización de la práctica (parte a)

a. Simulación y control remoto

Completar el programa ejemplo de modo que además de simular el robot en la pantalla, lo haga avanzar por el laberinto con los mismos comandos (Norte, sur, este, oeste, avanzar, finalizar).

Realización de la práctica (parte b)

b. Path planning y driving

El programa lee las coordenadas de los vértices A (origen) y B (destino) antes de empezar la ejecución. Luego calcula un camino de A a B y, por último, lo recorre.

1. Path planning:

Leer las coordenadas de los vértices A (origen) y B (destino), calcular un camino de A a B (sugerencia: Adaptar el algoritmo "Deep-first search")

Resolver primero el problema en el ordenador, y ejecutarlo en pantalla sobre el laberinto simulado.

2. Driving:

Partiendo del programa anterior hacer que el robot recorra el laberinto real de un nodo A a otro B.

El robot tiene que recorrer el camino **siguiendo la línea negra** y girando 90º en la dirección prevista cuando encuentre una línea perpendicular de (sin salirse ni realizar bucles).

Sugerencia: Se pueden usar los sensores de barranco *Cliff Front Left* y *Cliff Front Right* para seguir la línea recta entre dos nodos. El robot detectará que ha llegado a un cruce cuando detecte una línea recta perpendicular con los sensores de barranco laterales *Cliff left* o *Cliff Right*.

Realización de la práctica (parte c)

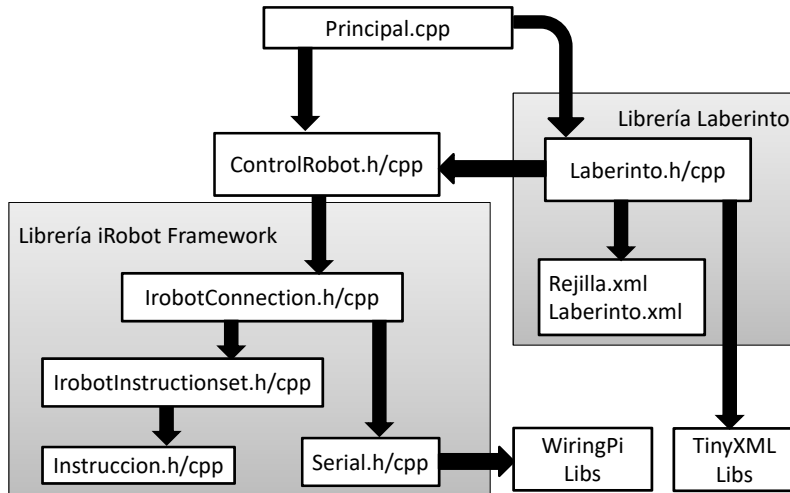
c. **Driving con evitación de obstáculos**

Hacer que el robot sea capaz de detectar obstáculos en el camino (un ladrillo sobre una arista entre dos nodos) mediante los bumpers. Cuando detecte un obstáculo tendrá que replanificar el recorrido.

Sugerencias

- Algoritmo “Deep-first search”:
 - Wikipedia: http://en.wikipedia.org/wiki/Depth-first_search
 - Youtube: <http://www.youtube.com/watch?v=iaBEKo5sM7w>
- Algoritmo de Dijkstra (con pesos):
 - Wikipedia:
http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
 - Youtube:
 - <http://www.youtube.com/watch?v=dS1Di2ZHI4k>
 - <https://www.youtube.com/watch?v=fgdCNuGPJnw>

Estructura del programa



Programación en C

Para cambiar el comportamiento del robot y ajustarlo al laberinto hay que modificar los ficheros:

- Laberinto.h y Laberinto.cpp,
- ControlRobot.h y ControlRobot.cpp
- Principal.cpp

Programación en C: Principal.cpp

- Modificar **Principal.cpp** para que resuelva los mapas.

```
int main(int argc, char * argv[])
{
    Laberinto *maze = new Laberinto("../xml/Laberinto.xml");

    // Generamos una solucion entre dos puntos origen [0,0] y destino [5,5]
    camino *solucion = maze->encontrarCamino(0,0,5,5);

    ControlRobot robot;

    robot.cargarMapa(maze);
    robot.recorrerCamino(solucion);

    robot.inicializacion();

    while(!robot.condicionSalida()){ // <-- Mientras no hayamos completado el "camino"
        robot.leerSensores();        // <-- Detectamos cruces
        robot.logicaEstados();        // <-- Decidimos que hacer en cada cruce
        robot.moverActuadores();      // <-- Nos orientamos en el Laberinto
        robot.imprimirInfo();
    }

    robot.finalizacion();

    return 0;
}
```

Programación en C: Laberinto.h/Laberinto.cpp

- En las clases **Laberinto.h/Laberinto.cpp** tenemos que completa el método:

```
camino* encontrarCamino (int x_orig, int y_orig, int x_dest, int  
                        y_dest);
```

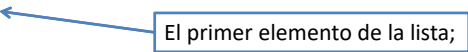
- Para ello podemos ayudarnos del método:

```
void modificarCamino(camino** cam, nodo **aux);
```


Programación en C: ControlRobot.h/ControlRobot.cpp

- En las clases **ControlRobot.h/ControlRobot.cpp** tenemos que crear los métodos:

```
void cargarMapa(Laberinto * lab);  
void recorrerCamino(camino * cam);
```

```
/**  
 * Método que obtiene el camino entre un nodo de entrada [x_orig, y_orig] y un nodo de  
 * salida [x_dest, y_dest].  
 * int x_orig: Posición origen del robot en la Matriz del Laberinto  
 * int y_orig: Posición origen del robot en la Matriz del Laberinto  
 * int x_dest: Posición destino del robot en la Matriz del Laberinto  
 * int y_dest: Posición destino del robot en la Matriz del Laberinto  
 * return camino*: Apuntador a una estructura tipo camino con la solución  
 */  
camino* Laberinto::encontrarCamino(int x_orig, int y_orig, int x_dest, int y_dest){  
    struct camino* solucion = (struct camino *) malloc (sizeof(camino));  
  
    // ... A COMPLETAR EN LA PRÁCTICA  
  
    return solucion;   
}  
  
/**  
 * Función que añade un nodo nuevo a un camino  
 * camino **cam: Puntero doble a la estructura camino que se va a modificar  
 * nodo **aux: Puntero doble al nodo que se quiere añadir en el camino  
 */  
void Laberinto::modificarCamino(camino** cam, nodo **aux){  
  
    struct camino * siguiente = (struct camino *) malloc (sizeof(camino));  
  
    siguiente->nodo_actual = *aux;  
    siguiente->anterior = *cam;  
    siguiente->siguiente = NULL;  
  
    (*cam)->siguiente = siguiente;  
    *cam = siguiente;  
}
```