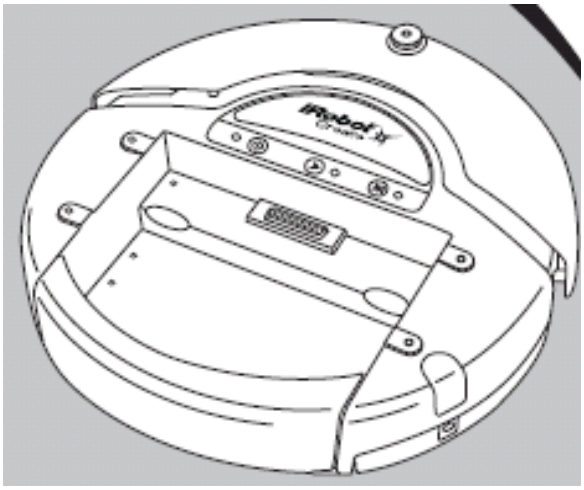
PRÁCTICA 5

Uso de mapas para *plannig* y *driving*

OBJETIVOS

Programar un algoritmo capaz de encontrar caminos entre nodos de un grafo y aplicarlo a iRobot. Aplicación a la planificación de desplazamientos por un laberinto y evitación de obstáculos.



MATERIAL NECESARIO

Hardware

- iRobot Create
- Raspberry Pi 2

Software

- NetBeans
- Archivo Workspace_iCreate.zip con el código C/C++
- Archivo src_mapping.zip con el código de la representación de los laberintos del laboratorio

Manuales (disponibles en eGela)

- Guía del usuario: iRobot Create Owner's Guide
- Manual del lenguaje de comandos: iRobot Create Open Interface
- Lista de funciones iRobotConnection

1. Puesta en marcha

a. Importar el proyecto contenido en "src_mapping.zip", como en las prácticas anteriores.
   El contenido debería ser el siguiente:
   1) Principal.cpp: Ejemplo que usa la clase Laberinto
   2) Laberinto.cpp: Clase principal para representar los laberintos del laboratorio
   3) Laberinto.h: Archivo con las definiciones de los métodos y variables.
   4) Carpeta de nombre **tinyxml** con la librería para leer los ficheros de los laberintos

5) Carpeta de nombre **xml** con los ficheros xml necesarios para describir los mapas de los laberintos que se usan en la práctiva.

b. Una vez importado el proyecto, deberíamos verlo cómo en la siguiente captura de pantalla:

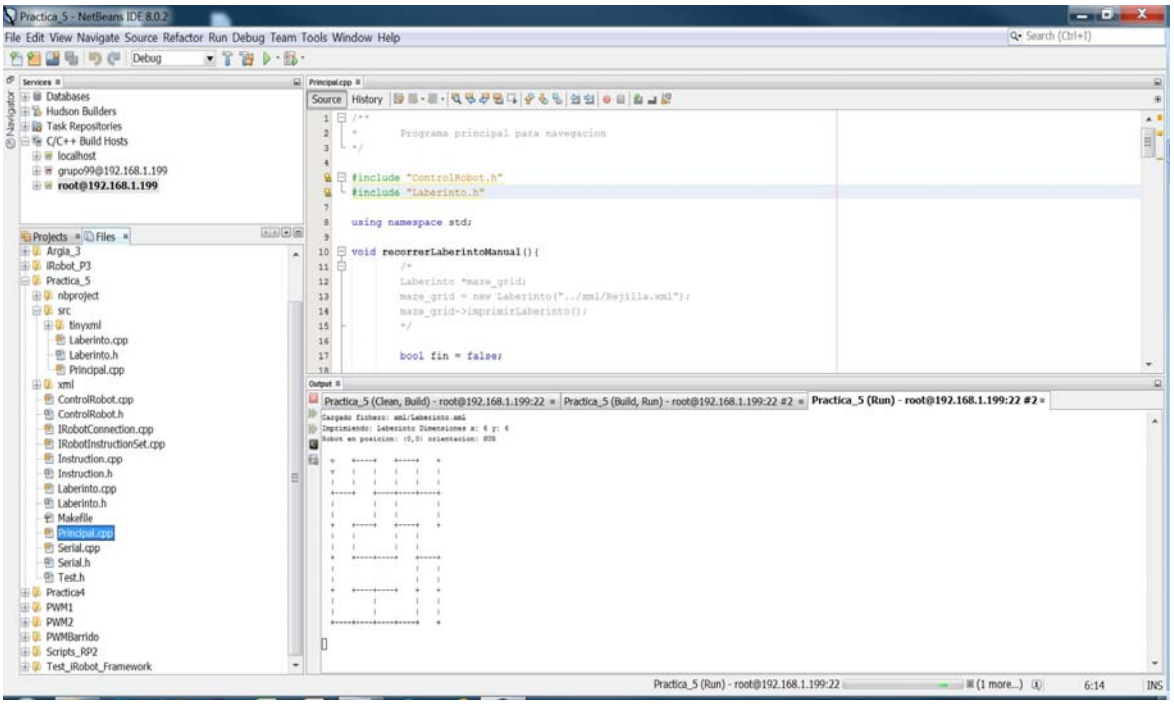


Figura 1. Aspecto del proyecto en NetBeans

c. Compilar y ejecutar el programa.

2. Programa de ejemplo

a. El fichero "src_mapping.zip" contiene un ejemplo de cómo funcionan las clases y métodos añadidos para representar mapas en un programa.

b. El algoritmo principal del ejemplo se encuentra en el fichero **Principal.cpp** y usa varias funciones encapsuladas en la clase **Laberinto.h/cpp**.

c. 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***: El robot se mueve al siguiente nodo al que esté orientado (si es posible).
- ***n***: El robot orienta su posición al norte
- ***e***: El robot orienta su posición al este
- w: El robot orienta su posición al oeste
- ***s***: El robot orienta su posición al sur
- ***imprimir***: Se muestra por pantalla el camino recorrido por el robot
- ***salir***: Finaliza el programa

Figura 2. Programa de ejemplo ejecutándose

3. Práctica

a. 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).

b. Recorrer el laberinto de un nodo A a otro B, leídos por el programa antes de empezar la ejecución.

b.1 Resolver primero el problema en el ordenador, y ejecutarlo en pantalla sobre el laberinto simulado.

b.2 Partiendo del programa anterior, haced que el robot recorra el laberinto real de un nodo A a otro B. Para el recorrido del laberinto, el robot debe avanzar siguiendo las líneas negras sin salirse y girar 90 grados en la dirección prevista cuando encuentre la línea perpendicular (a uno u otro lado).

Podéis 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*.

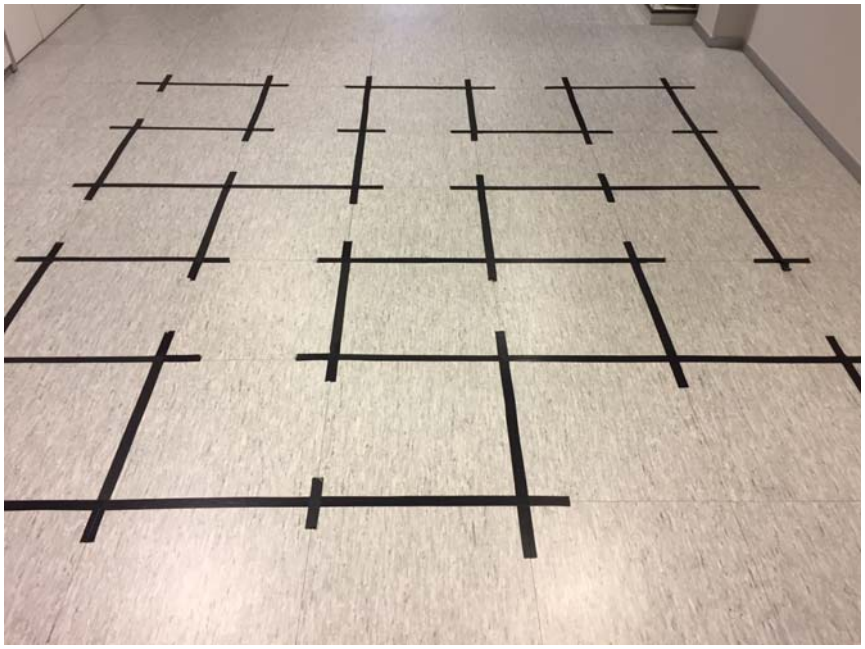


Figura 1. Laberinto de la práctica 5

c. Evitación de obstáculos:
   - Hacer que el robot sea capaz de detectar obstáculos en el camino (un ladrillo sobre una arista entre dos nodos). Cuando detecte un obstáculo tendrá que replanificar el recorrido.
     - Entrada Wikipedia: http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
     - Video Youtube: http://www.youtube.com/watch?v=dS1Di2ZHl4k

4. Sugerencias:

- Podéis encontrar varios algoritmos para resolver el problema en la siguiente entrada de la wikipedia: http://en.wikipedia.org/wiki/Pathfinding.
- Sugerencia: ***Adaptar*** el algoritmo pathfinding "Deep-first search" o "Búsqueda en profundidad" para calcular los recorridos en el grafo:
  - Entrada Wikipedia: http://en.wikipedia.org/wiki/Depth-first_search
  - Video Youtube: http://www.youtube.com/watch?v=iaBEKo5sM7w

5. Programación en C

Para cambiar el comportamiento del robot y ajustarlo al laberinto hay que modificar los ficheros **Laberinto.h/.cpp, ControlRobot.h/.cpp** y **Principal.cpp**:

- Hay que modificar **Principal.cpp** para que resuelva los problemas con 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;
}
```

Figura 5. Estructura el programa principal

- En las clases **Laberinto.h/.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);*

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

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

b. Informe a presentar

En las sesiones de evaluación de la última semana hay que hacer una demostración ante el profesor de todas las partes de la práctica.

El informe debe contener:

1. Código de los programas, exportando el proyecto en un fichero .zip
2. Descripción detallada de la solución diseñada.
3. Comentarios e incidencias.