eman ta zabal zazu

Informatika Fakultatea, EHU
Konputagailuen Arkitektura eta Teknologia Saila

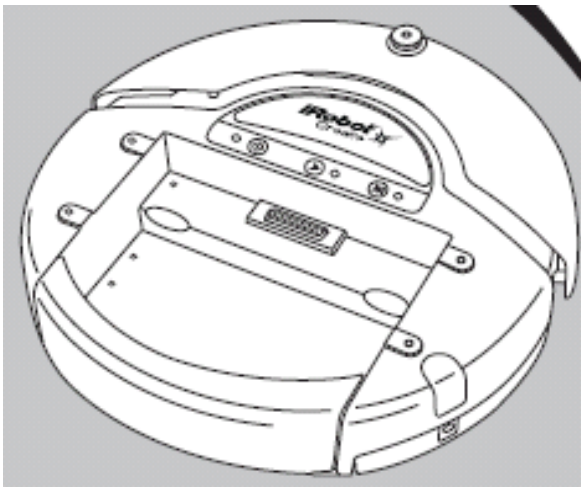
LABORATORIO DE ROBÓTICA

ROB 19-20

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 Practica5.zip con el código C/C++ y el código de la representación del laberinto 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. Navegación por el laberinto

- ***Mapping***: para empezar se coloca el robot en un nodo con una determinada orientación. El programa pedirá por pantalla la posición y orientación del robot para hacer el mapeo inicial. El mapa representa los **cruces** del laberinto como **nodos** y los **caminos** entre nodos como **vértices**.
- ***Driving***: El robot recorre el camino previsto navegando de nodo a nodo. Para ello sigue la línea negra entre nodos. Se pueden usar los sensores de barranco *Cliff Front Left* y *Cliff Front Right* para seguir la línea. El robot asume que ha llegado a un nodo cuando encuentra una línea perpendicular, o cruce, con los sensores de barranco laterales *Cliff left* o *Cliff Right*. Llegado a un nodo, si tiene que virar a derecha o izquierda, girará 90º o -90º hasta encontrar a línea negra perpendicular. El giro resulta más fácil si el eje vertical del robot coincide con el punto de cruce de las líneas.

Antes de empezar la práctica es necesario programar módulos que permitan navegar de

nodo a nodo en línea recta (detectando cruces) y girar a a derecha e izquierda.

2. Puesta en marcha

a. Importar el proyecto contenido en "Practica5.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:

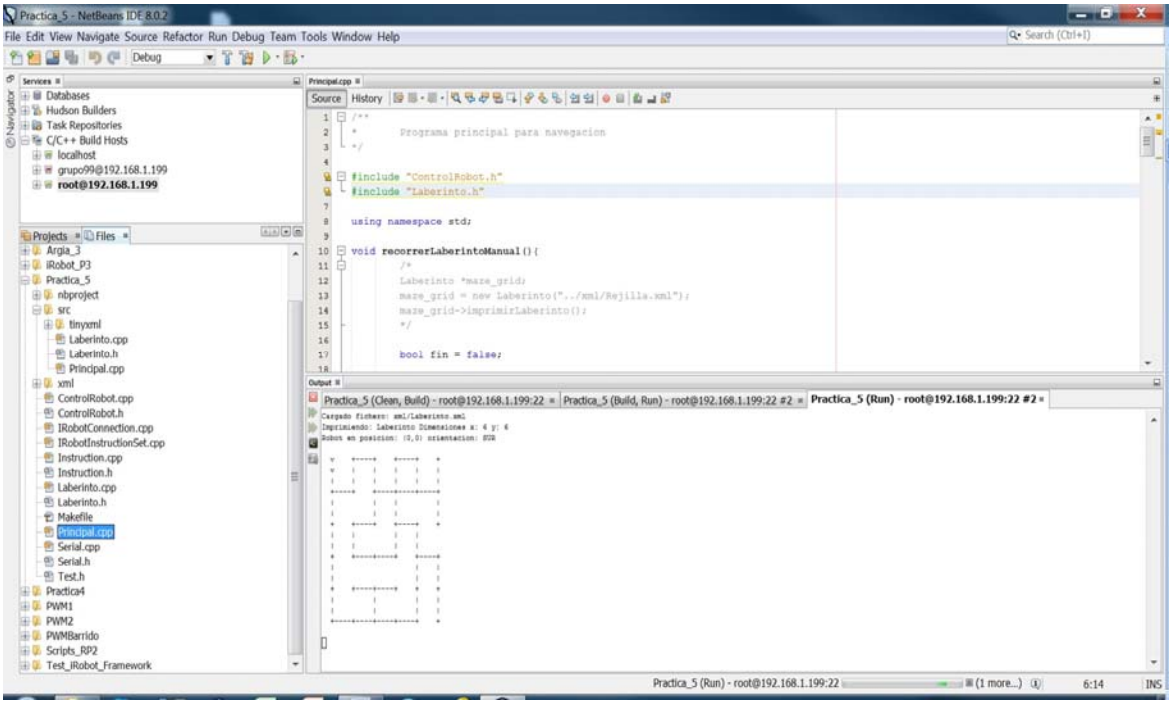


Figura 1. Aspecto del proyecto en NetBeans

c. Compilar y ejecutar el programa.

3. Programa de ejemplo

a. El fichero "Practica5.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

4. Práctica

a. **Simulación y control remoto simultáneos**
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. ***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.

b.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.

b.2 ***Driving***
Partiendo del programa anterior, haced que el robot recorra el laberinto real de un nodo A a otro B.

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 **retroceder al anterior nodo y replanificar el recorrido**.

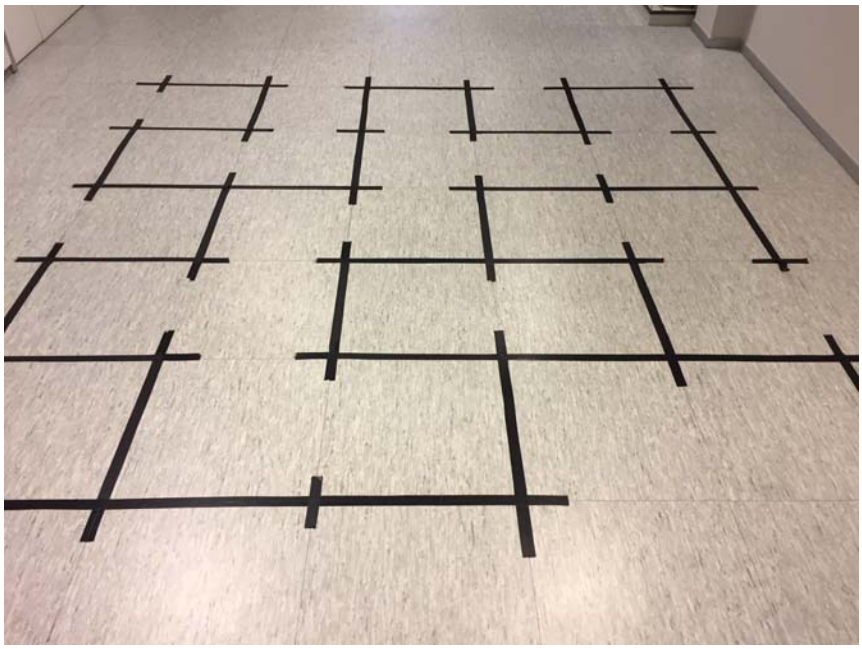


Figura 1. Laberinto de la práctica 5

5. Programación en C

Adaptar un algoritmo de pathfinding para calcular los recorridos en el grafo.
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. Preguntas:
   - ¿qué algoritmo de búsqueda habéis empleado?
   - ¿Por qué?

Evaluación de la Práctica 5	
Memoria	
• Claridad, concisión, corrección en la explicación del desarrollo de la práctica	2
Navegación A-> B	
• Leer Origen A y Destino B del teclado. Al apretar el botón central va del origen a destino sin salirse de las líneas negras	3
Navegación A-> B con evitación de obstáculos	
• Leer Origen A y Destino B del teclado y navegar del origen al destino. Cada vez que encuentra un obstáculo se vuelve al nodo anterior y re-planifica el trayecto. Debe evitar obstáculos sucesivamente hasta que se queda sin camino alternativo	3
Tiempo de navegación de X a Y con un obstáculo entre Z y T **(Los nodos X,Y,Z y T se asignarán en el momento del examen y serán los mismos para todos los grupos)**	
• Leer Origen X y Destino Y del teclado. Al apretar el botón central empezará a contar el tiempo que tarda del origen al destino sin salirse de las líneas negras, evitando un obstáculo. • Nota: 0 si no llega. Si llega: 1 al robot más lento de clase (T_{max}); 2 al más rápido (T_{min}); el resto: nota proporcional a su tiempo $\mathbf{T_x}$: $N_x = N_{max} - (\mathbf{T_x} - T_{min}) * (N_{max} - N_{min})/(T_{min} - T_{max})$	0 - 2
Nota	**10**