



## LABORATORIO DE ROBÓTICA

ROB 2017-18

### PRÁCTICA 4

#### Navegación mediante marcas y evitación de obstáculos con iRobot Create

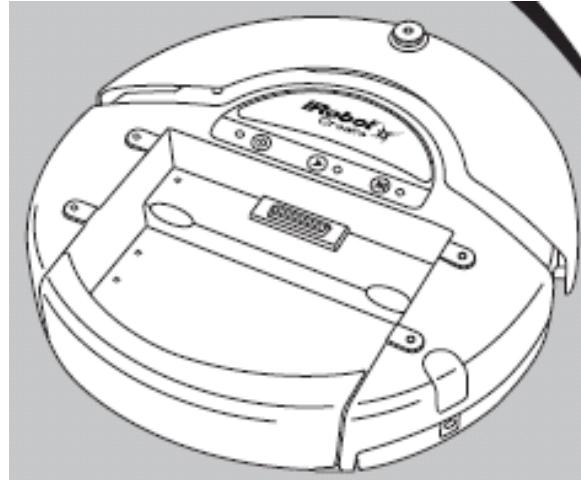
#### OBJETIVOS

Programar un algoritmo de navegación por marcas (mediante el seguimiento de una línea negra marcada en el suelo) y de evitación de obstáculos, para iRobot Create

#### MATERIAL NECESARIO

##### Hardware

- iRobot Create
- Raspberry Pi2



##### Software

- NetBeans
- Archivo Workspace\_iCreate.zip con el código C/C++
- Archivo src\_ejemplo.zip con el código de ejemplo del sigue\_linea.cpp

##### 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. Crear un nuevo proyecto a partir del archivo “Framework\_Practica4.zip”
- b. Seleccionar los estándares C11 y C++11, enlazar la librería wiringPi
- c. Seleccionar el host adecuado.

Además de los fuentes y headers de la práctica anterior, el contenido de proyecto debería ser el siguiente:

- 1) Sigue\_linea.cpp : Contiene el método main
- 2) RobotControl.cpp : Clase principal,
- 3) RobotControl.h : Archivo de definiciones para la clase RobotControl

## 2. Programa de Ejemplo

- Compilar y ejecutar el programa ejemplo **Sigue\_linea.cpp**.
- El ejemplo en **Sigue\_linea.cpp** contiene un algoritmo básico para iCreate que hace uso del sensor de acantilado frontal izquierdo para seguir una línea por el suelo del laboratorio.
- El programa principal se encuentra en el fichero **Sigue\_linea.cpp** y usa varias funciones encapsuladas en la clase **ControlRobot.h/cpp**.

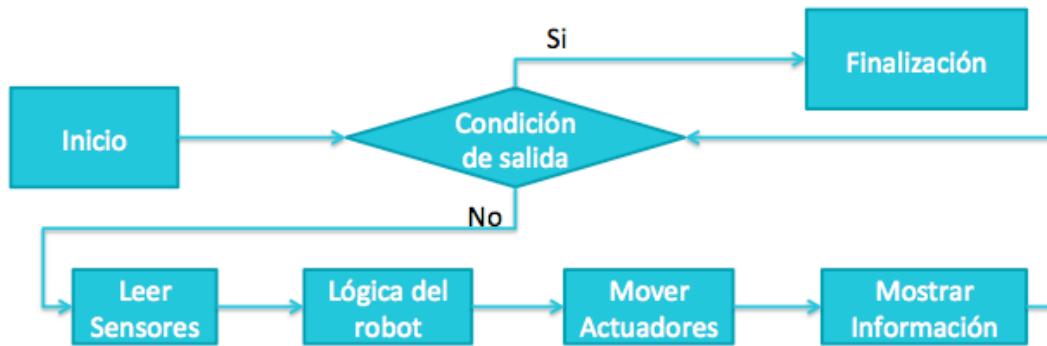


Figura 1. Algoritmo principal

- La máquina de estados que usa iRobot Create en el ejemplo es la siguiente:

L	FL	FR	R	W	ESTADO ANTERIOR	NUEVO ESTADO
-	0	-	-	-	INICIAL	<b>BUSCA: {Girar derecha}</b>
-	1	-	-	-	INICIAL	<b>SIGUE: {Avanzar}</b>
-	0	-	-	-	SIGUE	<b>BUSCA</b>
-	1	-	-	-	SIGUE	<b>SIGUE</b>
-	0	-	-	-	BUSCA	<b>BUSCA</b>
-	1	-	-	-	BUSCA	<b>SIGUE</b>

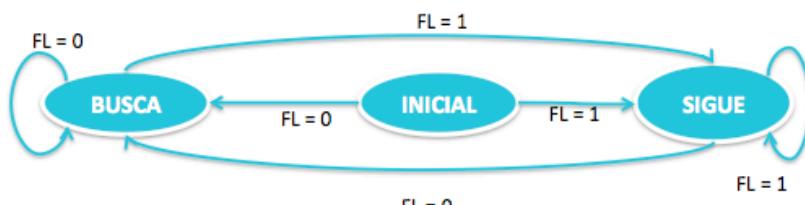


Figura 2. Máquina de estados

## 3. Programación en C

Para cambiar el comportamiento del robot hay que modificar los ficheros **ControlRobot.h** y **ControlRobot.cpp**. No es necesario cambiar **sigue\_linea.cpp**:

- Cambios en los sensores:

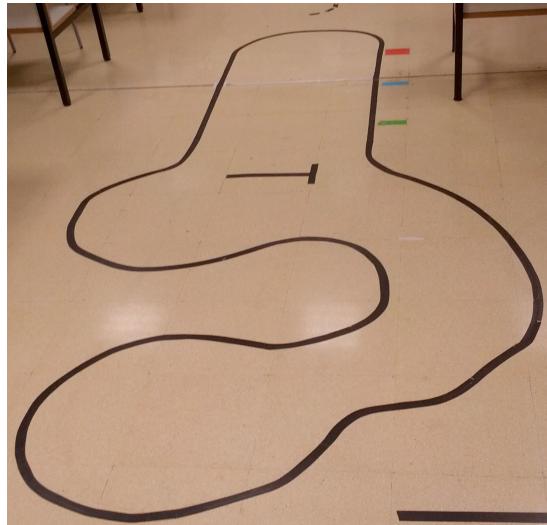
- Modificar la estructura **Sensores\_iCreate** en **ControlRobot.h**
- Modificar el método **void ControlRobot::leerSensores()** en **ControlRobot.cpp**.
- Algunos sensores de iRobot, por ejemplo el sensor de distancia o el de ángulo, requieren ser inicializados para funcionar correctamente. Para ello hay que leerlos en la inicialización (ignorando su respuesta).

- b. Cambios en la máquina de estados:
  - Incluir un `#define NUEVO_ESTADO` valor por cada nuevo estado.
  - Modificar el método `void ControlRobot::logicaEstados()` en ControlRobot.cpp
- c. Cambios en los actuadores:
  - Modificar la estructura `Actuadores_iCreate` en ControlRobot.h
  - Modificar el método `void ControlRobot::logicaEstados()` en ControlRobot.cpp
  - (opcional) Usar macros `#define` para los estados de los actuadores
- d. Cambios en la función de imprimir por pantalla:
  - Modificar el método `void ControlRobot::imprimirInfo` en ControlRobot.h
- e. Otras modificaciones en el comportamiento del robot pueden requerir:
  - Modificar el método `void ControlRobot::inicialización` en ControlRobot.h
  - Modificar el método `void ControlRobot::condiciónSalida` en ControlRobot.h
  - Modificar el método `void ControlRobot::finalización` en ControlRobot.h

## 4. Realización de la práctica

### a. Seguimiento de marcas

- Teniendo en cuenta la diferente sensibilidad de cada uno de los sensores de acantilado del iRobot (calibrada en la práctica anterior) hacer que el iRobot siga una raya marcada en el suelo con cinta aislante.
- Utilizar, como mínimo, dos sensores de barranco. Explicar en el informe cómo se ha decidido el número de sensores a utilizar.
- Al final del recorrido, presentar por pantalla:
  - distancia recorrida y
  - tiempo transcurrido.



Para calificar esta parte se realizará una prueba del seguimiento de un circuito de cinta aislante (que no tiene por qué ser el del laboratorio) en ambos sentidos. La puntuación de esta parte será inversamente proporcional al tiempo empleado en recorrer el circuito. Si un robot se sale del circuito el tiempo se considerará  $\infty$ .

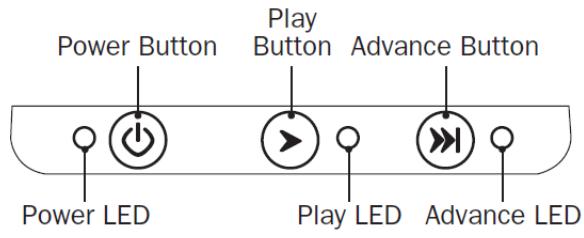
### b. Evitación de obstáculos

- Cuando el robot, mediante los *bumpers*, detecte un obstáculo (un ladrillo) bloqueando la línea, deberá rodearlo.
  - Haciendo un rodeo mediante *dead reckoning* y/o
  - Rodeando el obstáculo mediante el sensor de pared (*wall*).
  - (La segunda técnica puntúa el doble que la primera)
- Después de superar el obstáculo el robot debe seguir recorriendo el circuito.
- Para la evaluación de esta parte se realizará una prueba de evitación de un obstáculo (un ladrillo) mediante ambas técnicas.



### c. Especificaciones prácticas:

- Ya que el robot va a estar lejos del PC, utilizar el sensor del botón PLAY para que el robot se ponga en marcha (sin necesidad de hacerlo desde el PC).
- Aunque los programas en sistemas empotrados no finalizan nunca, en la fase de pruebas necesitamos detenerlo cuando queramos. Por ello, utilizar el sensor del botón de ADVANCE para detener el programa del robot, actuando sobre la condición de salida de la máquina de estados. Para ello, será necesario modificar el algoritmo para que a la finalización se desactiven los actuadores.



## 5. Informe a presentar

1. Código fuente de los programas en un fichero .zip obtenido mediante el comando exportar proyecto. [Puntúa la legibilidad de los programas]
2. Descripción detallada de la solución diseñada, incluyendo
  - Explicación de la máquina de estados utilizada
  - Cuantos sensores de barranco se han usado y cómo se ha decidido el número de sensores a utilizar
  - Cuáles son las condiciones para poder usar una u otra técnica en evitación de obstáculos
3. Comentarios e incidencias.