

## Robótica Móvil

### Práctica 4: Navegación con iRobot Create



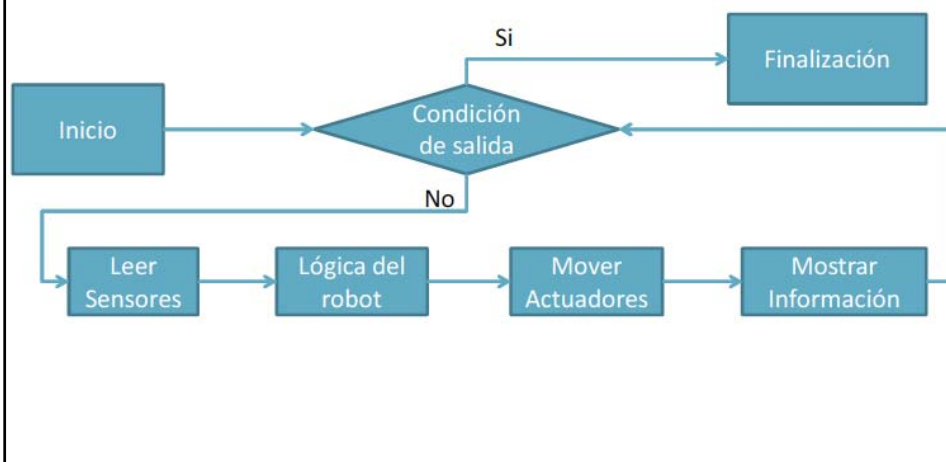
## Descripción de la práctica

- **Objetivos**
  1. Programar un algoritmo de navegación mediante el seguimiento de marcas (en este caso una línea marcada en el suelo) para el iRobot Create.
  2. Programar un algoritmo de evitación de obstáculos.
- **Procedimiento (detallado en la descripción de la práctica):**
  1. Puesta en marcha (proyecto que hay que copiar y ajuste de los parámetros)
  2. Programa de ejemplo (como funciona un seguidor de líneas básico)
  3. Programa en C (qué hay que hacer para programar las nuevas funciones del robot)
  4. Realización de la práctica (funcionalidad deseada)
  5. Informe a presentar

## Programa ejemplo: **ControlRobot.cpp**

- Importar el proyecto **Framework\_Practica4.zip**
- Este fichero contiene
  - **Fuentes:**
    - **Sigue\_linea.cpp**, el programa principal
    - **ControlRobot.cpp**: un ejemplo de seguimiento de línea mediante un algoritmo básico que hace uso del sensor de acantilado frontal izquierdo para seguir una línea marcada con cinta aislante negra en el suelo.
    - Otros fuentes necesarios: IrobotConnection.cpp, IrobotInstructionSet.cpp, Instruction.cpp, Serial.cpp
  - **Headings:**
    - **ControlRobot.h** que encapsula varias funciones usadas por siguelinea.
    - Otros headings necesarios: IrobotConnection.h, IrobotInstructionSet.h, Instruction.h, Serial.h

## Sistema de control



## Sigue\_linea.cpp

```
/**
 * Algoritmo genérico para iCreate
 */

#include "ControlRobot.h"

using namespace std;

int main(int argc, char * argv[])
{
    ControlRobot robot;

    robot.inicializacion();

    while(!robot.condicionSalida()){
        robot.leerSensores();
        robot.logicaEstados();
        robot.moverActuadores();
        robot.imprimirInfo();
    }

    robot.finalizacion();
    return 0;
}
```

## ControlRobot.h

```
class ControlRobot {
private:
    IRobotConnection *robot;

    struct Sensores_iCreate {
        // variables para almacenar información del
        // sensor de acantilado frontal izquierdo
        unsigned int front_left;
        bool fl;
    };
    struct Sensores_iCreate sensores;

    struct Actuadores_iCreate {
        // variables para el comando Drive direct
        int vel_der;
        int vel_izq;
        // variables para el comando Drive
        int velocidad;
        int giro;
    };
    struct Actuadores_iCreate actuadores;

    char estado_actual;
    char estado_anterior;
    char motores;

public:
    ControlRobot(void);
    ~ControlRobot(void);

    void inicializacion();
    bool condicionSalida();
    void leerSensores();
    void logicaEstados();
    void moverActuadores();
    void imprimirInfo();
    void finalizacion();
};
```

# Estructura del programa

- **Inicialización**
  - Inicializar las variables y estructuras
  - Gestión de la conexión con el robot
- Mientras no se cumpla una condición de salida:
  - Leer sensores
  - Aplicar lógica (Máquina de estados)
  - Activar actuadores
  - Mostrar información por pantalla
- Si se cumple la condición de salida:
  - Gestión de la desconexión del robot
  - Finalización de estructuras y variables

```
/**
 * Prepara la conexión IRobotConnection e inicializa todas
 * las variables que necesitemos en el programa
 */
void ControlRobot::inicializacion(void)
{
    int COM_port;
    char puerto[30];

    // Solicitamos el puerto COM por entrada estándar
    cout << "Puerto COM: ";
    cin >> COM_port;
    sprintf(puerto, "COM%d", COM_port);

    robot = new IRobotConnection(puerto);

    // Iniciamos la conexión
    cout << "Connecting... ";
    robot->connect();
    cout << "Done!!\n" << endl;

    // Comando 128 start
    robot->start();
    Sleep(500); // Esperamos medio segundo a que cambie de modo

    // Comando 132 modo full
    robot->full();
    Sleep(500); // Esperamos medio segundo a que cambie de modo

    estado_actual = INICIAL;
    estado_anterior = INICIAL;
    motores = PARADO;
}
```

## Estructura del programa

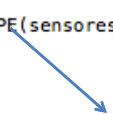
- Inicialización
  - Inicializar las variables y estructuras
  - Gestión de la conexión con el robot
- **Mientras no se cumpla una condición de salida:**
  - **Leer sensores**
  - Aplicar lógica (Máquina de estados)
  - Mover actuadores
  - Mostrar información por pantalla
- Si se cumple la condición de salida:
  - Gestión de la desconexión del robot
  - Finalización de estructuras y variables

```
/**
 * Calcula si se han dado las condiciones necesarias para terminar el programa
 * return bool:
 *     - true para terminar
 *     - false para continuar al menos un ciclo m.s
 */
bool ControlRobot::condicionSalida()
{
    return false; // nos mantenemos siempre dentro del bucle
}

/**
 * Obtiene y trata la información de los sensores relevantes al programa,
 * para ello usa la struct Sensores_iCreate sensores;
 */
void ControlRobot::leerSensores()
{
    sensores.front_left = robot->updateSensor(iRobotSensors::CLIFFFRONTLEFTSIGNAL );
    // Ajustamos el valor al máximo permitido por la especificación 0I
    if(sensores.front_left>4095) sensores.front_left=4095;

    sensores.fl = DUCT_TAPE(sensores.front_left);
}

#define DUCT_TAPE(value) (value < 500)
```

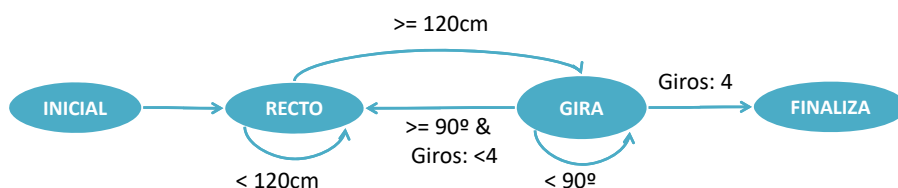


## Estructura del programa

- Inicialización
  - Inicializar las variables y estructuras
  - Gestión de la conexión con el robot
- Mientras no se cumpla una condición de salida:
  - Leer sensores
  - **Aplicar lógica (Máquina de estados)**
  - Mover actuadores
  - Mostrar información por pantalla
- Si se cumple la condición de salida:
  - Gestión de la desconexión del robot
  - Finalización de estructuras y variables

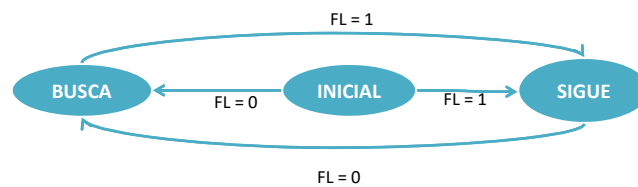
## Máquina de Estados: Ejemplo *Dead Reckoning* Cuadrado

Sensor Distancia	Sensor Giro	Número Giros	ESTADO ANTERIOR	NUEVO ESTADO
-	-	-	INICIAL	RECTO : {Avanzar}
< 120cm	-	-	RECTO	RECTO
>=120cm	-	-	RECTO	GIRA : {Girar derecha}
-	< 90º	-	GIRA	GIRA
-	>= 90º	< 4	GIRA	RECTO
-	>= 90º	4	GIRA	FINALIZA :{Parar}



## Máquina de Estados: Sigue\_Línea Básico (1 sensor)

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



```

/**
 * Contiene la lógica del programa
 */
void ControlRobot::logicaEstados()
{
    // Actualizamos los estados:
    estado_anterior = estado_actual;

    if(sensores.fl){
        estado_actual = SIGUE;
    }else{
        estado_actual = BUSCA;
    }

    // Se decide que hacer con los parametros del robot
    switch(estado_actual){
        case SIGUE:
            motores = RECTO;
            break;
        case BUSCA:
            motores = GIRAR_DERECHA;
            break;
        default:
            break;
    }
}

```

// Estados de la lógica

```

#define INICIAL 5
#define SIGUE 1
#define BUSCA 0

```

// Estados para los motores

```

#define GIRAR_DERECHA 0
#define GIRAR_IZQUIERDA 1
#define RECTO 2
#define PARADO 3

```

## Estructura del programa

- Inicialización
  - Inicializar las variables y estructuras
  - Gestión de la conexión con el robot
- Mientras no se cumpla una condición de salida:
  - Leer sensores
  - Aplicar lógica (Máquina de estados)
  - **Activar actuadores**
  - Mostrar información por pantalla
- Si se cumple la condición de salida:
  - Gestión de la desconexión del robot
  - Finalización de estructuras y variables

```
/**
 * Activa los actuadores correspondientes en función de lo decidido
 * en la lógica del programa
 */
void ControlRobot::moverActuadores()
{
    switch(motores){
        case PARADO:
            actuadores.vel_der = 0;
            actuadores.vel_izq = 0;
            break;
        case RECTO:
            actuadores.vel_der = 170;
            actuadores.vel_izq = 170;
            robot->leds( 3, 90,200 );
            break;
        case GIRAR_DERECHA:
            actuadores.vel_der = -50;
            actuadores.vel_izq = 50;
            robot->leds( 3, 0,0 );
            break;
        default:
            break;
    }

    robot->driveDirect(actuadores.vel_der, actuadores.vel_izq);
}
```

```
// Estados para los motores
#define GIRAR_DERECHA 0
#define GIRAR_IZQUIERDA 1
#define RECTO 2
#define PARADO 3
```



## Estructura del programa

- Inicialización
  - Inicializar las variables y estructuras
  - Gestión de la conexión con el robot
- Mientras no se cumpla una condición de salida:
  - Leer sensores
  - Aplicar lógica (Máquina de estados)
  - Mover actuadores
  - **Mostrar información por pantalla**
- Si se cumple la condición de salida:
  - **Gestión de la desconexión del robot**
  - **Finalización de estructuras y variables**

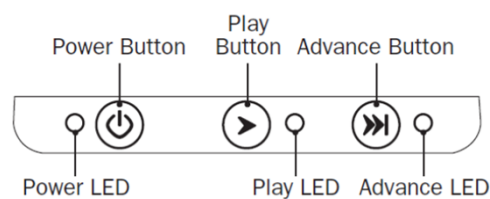
```
/*
 * Muestra información relevante al usuario
 */
void ControlRobot::imprimirInfo(void)
{
    char estado[20];
    switch(estado_actual){
        case INICIAL : sprintf(estado,"INICIAL"); break;
        case SIGUE :   sprintf(estado,"SIGUE"); break;
        case BUSCA :   sprintf(estado,"BUSCA"); break;
    }

    printf("Estado: %s ",estado);
    printf(": %s ", sensores.fl ? "true" : "false");
    printf(" Valor: %d", sensores.front_left);
    printf("\n");
}

/**
 * Cierra conexiones abiertas
 */
void ControlRobot::finalizacion(void)
{
    robot->disconnect();
    delete robot;
}
```

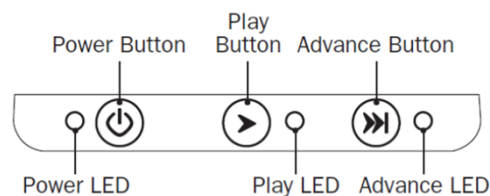
## Modificación de la finalización del programa

- Usualmente los programas empotrados no acaban nunca.
- En este caso necesitamos poder pararlo por razones prácticas.
- Lo haremos mediante el botón de avance (doble flecha)
  - Actualizar la estructura de sensores para que lea el sensor botón
  - Modificar la condición de salida de la máquina de estados
  - Modificar la finalización del robot para desactivar los actuadores



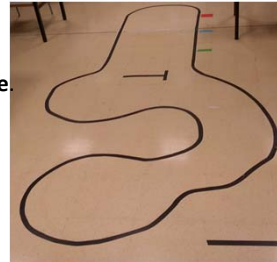
## Inicio de la ejecución del programa

- 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).



## Seguimiento de la línea

- Procedimiento:
  - Usar la estructura del programa en C++ sigue línea con 1 sensor y completarlo con más sensores del robot
- Objetivo:
  - Completar el circuito en el menor tiempo posible **sin salirse**
- Requisitos:
  - Utilizar dos, tres o cuatro sensores de barranco.
- Sugerencias
  - Sacar por pantalla solamente los datos que os ayuden a verificar cómo funciona el algoritmo
  - Usar los LEDs del robot para recibir feedback de la actividad de los sensores,
- Prueba:
  - Seguimiento de un circuito en ambos sentidos de las dos direcciones (puntuar la precisión –no salirse del circuito- y la velocidad).



## Evitación de obstáculos

- Cuando el robot detecte mediante los *bumpers* un obstáculo (ladrillo) bloqueando la línea, deberá rodearlo. Después de superar el obstáculo el robot debe seguir recorriendo el circuito.
  - Por *dead reckoning*
  - Usando el sensor de pared (*wall*)
- Explicar en el informe cuáles son las condiciones para poder usar una u otra técnica.



## Resumen de la programación necesaria

- Modificar los ficheros **ControlRobot.h** y **ControlRobot.cpp** para cambiar el comportamiento del robot
- 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 necesitan ser llamados en la inicialización para funcionar correctamente (ejemplo, sensor de distancia y ángulo).
- 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**
- 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
- Cambios en la función de imprimir por pantalla:
  - Modificar el método **void ControlRobot::imprimirInfo** en **ControlRobot.h**
- 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**

## 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 para el seguimiento de la línea, incluyendo una explicación de la máquina de estados utilizada, el número de sensores de barranco utilizado y por qué habéis seleccionado ese número.
3. Descripción detallada de la solución diseñada para la evitación de obstáculos, incluyendo la técnica que habéis utilizado y por qué la habéis seleccionado.