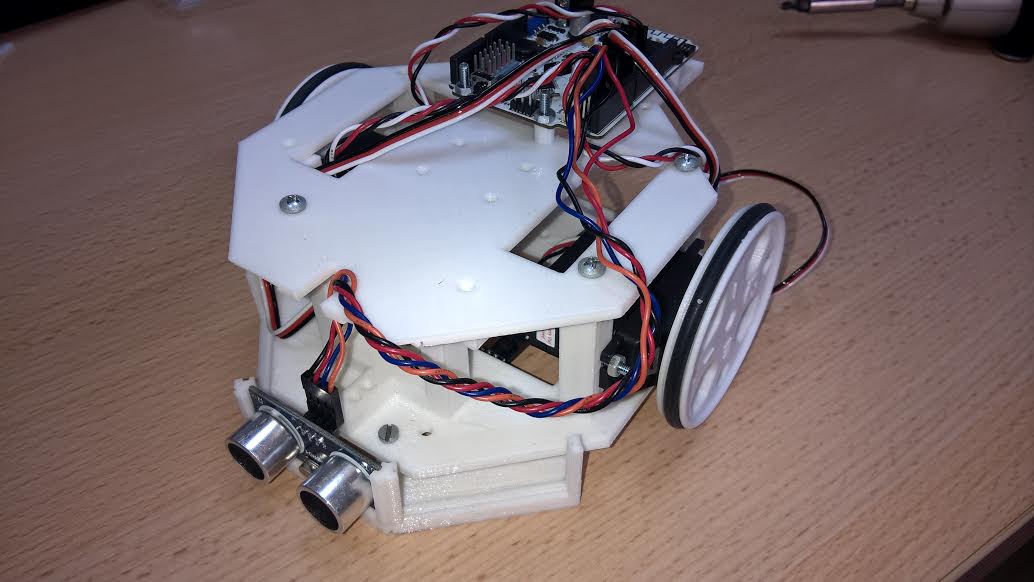
|  |
| --- |
| Software para Robots Jordán Pascual : pascualjordan@uniovi.es |

|  |  |
| --- | --- |
| 10 | Robots Móviles – Exploración de caminos y trayectorias |

# Introducción

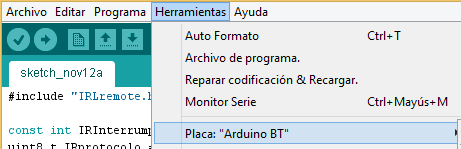
En esta práctica partimos del robot anterior, modificado para incluir un mayor número de sensores infrarrojos (4) que le permitan detectar las líneas sin realizar correcciones de trayectoria (barridos).



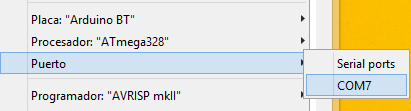
# Conceptos

Es una versión de la **Arduino BT, muy similar a la placa que utilizamos anteriormente.**

* Tiene una palanca de encendido/apago en un lateral (Roja y Azul).
* La conexión al PC se realiza por micro USB
* Debemos configurar el tipo de placa en Arduino sino no subirá los programas



* No suele indicar el nombre de la placa en el puerto, no obstante, sí que puede estar conectada igualmente.



* Suele tardar unos minutos en subir los programas, más que la placa Arduino Uno.
* Tiene Bluetooth integrado

**Esquema de conexiones:**

|  |  |
| --- | --- |
| Componente | Conexión (Pin digital) |
| Sensor 1: de lineal izquierdo | 10 |
| Sensor 2: de línea central-izquierdo | 2 |
| Sensor 3: de lineal central derecho | 3 |
| Sensor 4: de lineal derecho | 11 |
| Servo izquierdo | 8 |
| Servo derecho | 9 |
| Ultrasonidos Trig | 4 |
| Ultrasonidos Echo | 5 |

I

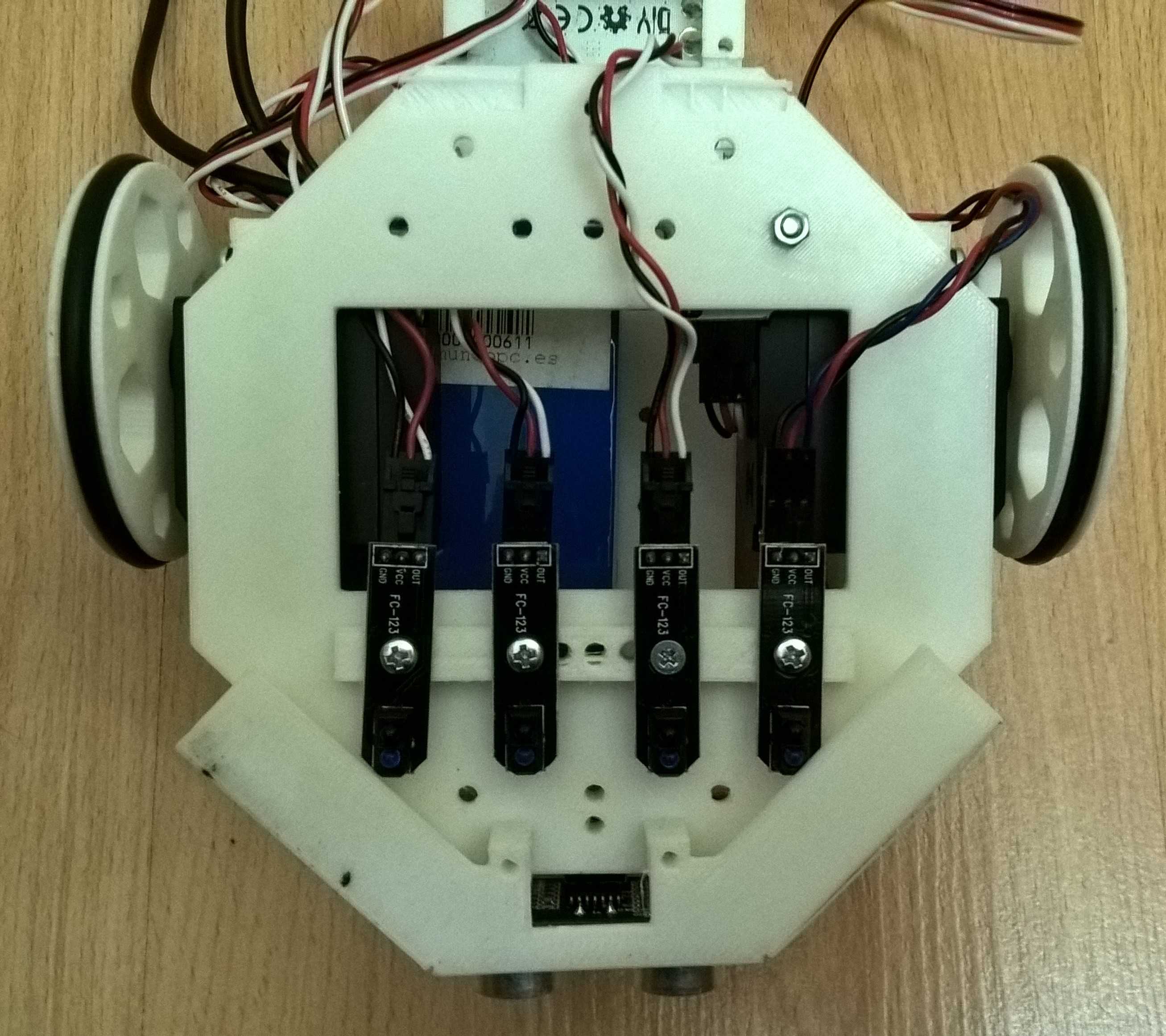
D

4

3

1

2



|  |
| --- |
| **Importante**. Vamos a respetar siempre las posiciones de los pines que se muestran en el siguiente esquema.  El objetivo es que todos los robots estén conectados de igual manera y el código pueda ser probado en cualquiera de ellos. |

## Introducción

Esta actividad corresponde con el conjunto de **actividades finales del bloque 3** que se desarrollará durante las sesiones de prácticas 10 y 11. Se desarrollará en los grupos de 3-5 que ya han sido formados anteriormente. Debe ser presentada obligatoriamente ante el profesor durante el transcurso las sesiones de prácticas: 10, 11 o inicio de la 12.

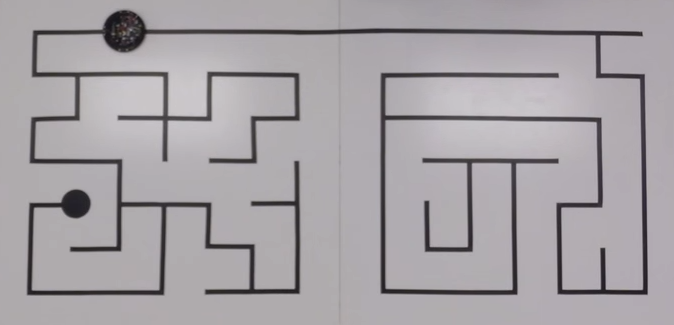
El conjunto de esta práctica tiene un peso máximo sobre la nota del bloque 3 de 7 puntos, cada actividad tiene una puntuación asociada en función de su complejidad.

|  |
| --- |
| ♣ B.3.1 – Sigue líneas (Parejas) – 3 puntos  ♣ B.3.2 – Caminos y trayectorias (Grupo) – 8 puntos  ♣ B.3.3 – Sumo Wars (Parejas) – 1 puntos  **Total: 11 Puntos.**  **El bloque 3 tiene un peso de 40% en el total de la asignatura.** |

Se deben seleccionar los apartados que se desean realizar, en caso de que la nota de los apartados sobrepase los 8 puntos será truncada. No obstante, será tenido en cuenta para la obtención de la matrícula.

# 10.1 Resolución de laberintos (6 puntos)

Se debe desarrollar un controlador capaz de encontrar un objetivo en un laberinto de caminos como el que se muestra a continuación:



META

INICIO

El robot puede comenzar en cualquier punto del laberinto y debe encontrar la meta.

Ejemplos:

<https://www.youtube.com/watch?v=xyplzYmZmP8>

<https://www.youtube.com/watch?v=mJV-KDqHgDQ>

## Orientaciones

### Pausa inicial Para que el robot no comience a moverse según ejecutamos la aplicación debemos incluir un delay(10000) en el método setup, será tiempo suficiente para colocarlo en la posición de inicio.

### Prueba de sensores

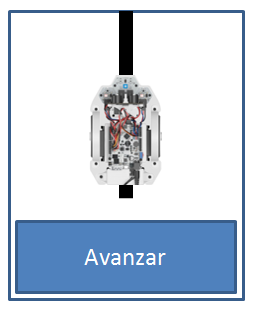
Inicialmente se debería probar que todos los sensores de línea funcionan de forma correcta.

### Algoritmo

Se puede aplicar cualquiera de los algoritmos de resolución mencionados en la clase de teoría 11: Left Hand Rule, Tremaux, Pledge, etc.

### Prueba individual de las situaciones

Se deben codificar cada una de las situaciones **de forma independiente y probar repetidas veces antes de pasar a la siguiente**, un enfoque a seguir podría ser ir resolviendo los siguientes tipos de situación:



* Avanzar recto por el camino (Se puede hacer usando solo los sensores centrales)



* Mantenerse recto en el camino con correcciones a derecha o izquierda (Se puede hacer usando solo los sensores centrales)



* Dar la vuelta en un camino sin salida



* Girar en una curva a la derecha



* Girar en una curva a la izquierda



* Detectar un cruce de caminos (izquierda - derecha) y tomar la decisión más apropiada



* Detectar un cruce de caminos (adelante - izquierda - derecha ) y tomar la decisión más apropiada.



* Detectar un cruce de caminos ( adelante - izquierda) y tomar la decisión más apropiada.



* Detectar un cruce de caminos (adelante - derecha) y tomar la decisión más apropiada.



* Detectar la meta y detener el robot durante 20 segundos, debe ser el tiempo suficiente para que podamos volver a colocarlo en la posición de inicio.

|  |
| --- |
| La parte más compleja de esta actividad consiste en programar correctamente los movimientos del robot y los sistemas de percepción. (Como recolectar los datos de los sensores de forma adecuada, como se mueven los motores para que gire y entre en el carril de forma correcta, que movimientos debe hacer para detectar diferentes tipos de cruces, cuando debe detenerse, etc.) |

### Uso de los sensores

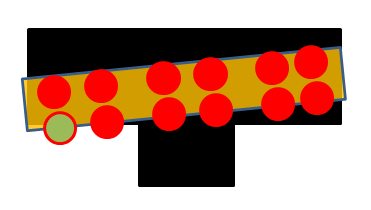
No podemos fiarnos de una única lectura de los sensores en un momento concreto nos dé un estado realista de la situación.

Los algoritmos suelen describir situaciones teóricas, donde todo funciona a la percepción, pero en la práctica con un robot y hardware real las cosas suelen ser bastante distintas.

La teoría nos dice que cuando el robot entra en una intersección los 4 sensores debe activarse a la vez, pero eso es bastante poco probable. Ya sea por retardos en las lecturas o por una trayectoria no optima probablemente los sensores no se activaran en la misma iteración del **loop()**.

|  |  |
| --- | --- |
|  |  |
|  |  |

Existen varias técnicas que podemos utilizar para solucionar este problema, una de ellas es no leer los sensores en un único punto, sino realizar N lecturas consecutivas en un corto periodo de tiempo y hacer una composición de la información.



|  |
| --- |
| void actualizarTodosLosSensores**(){**  **for(**int i**=**0**;** i **<** 4**;** i**++){**  sensor**[**i**]** **=** digitalRead**(**pinesSensores**[**i**]);**  **}**    // 4 lecturas adicionales  // Se guardan las que detectan línea en cualquiera de los 4 barridos  **for(**int i**=**0**;** i **<** 4**;** i**++){**  delay**(**20**);** // Delay con el movimiento actual    **for(**int i**=**0**;** i **<** 4**;** i**++){**  **if** **(**digitalRead**(**pinesSensores**[**i**])** **==** LINEA **)**  sensor**[**i**]** **=** LINEA**;**  **}**  **}**  **}** |

### Avanzar ligeramente para detectar el tipo de cruce

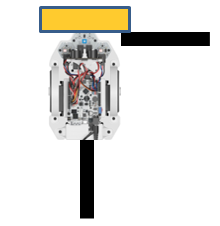
Para detectar ciertos tipos de camino hay que avanzar ligeramente hacia delante, para avanzar hacia delante podemos colocar los motores en posición de avance y avanzar durante unos pocos milisegundos (200-300), este tiempo debe ser muy corto porque durante el delay el robot no recibe percepciones y podría llegar incluso a salirse del carril si no lo afronta con el Angulo correcto.

|  |
| --- |
| **if** **(**Sensor1 **==** NO\_LINEA **&&** Sensor2 **==** LINEA **&&** Sensor3 **==** LINEA **&&** Sensor4 **==** LINEA**)** **{**  // Curva de derechas || Curva a derechas y frente  // \_ X X X    // Avanzar un poco  servoD**.**write**(**180**);**  servoI**.**write**(**0**);**  delay**(**300**);**    // Parar  servoD**.**write**(**90**);**  servoI**.**write**(**90**);**  delay**(**4000**);**    // Actualizar sensores  actualizarTodosLosSensores**();**    **if(** Sensor1 **==** LINEA **||** Sensor2 **==** LINEA **||** Sensor3 **==** LINEA **||** Sensor4 **==** LINEA **){**  // Algo de linea delante, es una curva a derechas y frente  // \_ X X X  avanzar**();**    **}** **else** **{**  // Solo Curva a derechas  girarDerecha**();**  **}**    **}** |

### Giros con poco delay

En los giros es muy probable que tengamos que sacar al robot de su trayectoria actual, para ello aplicamos un pequeño giro con un delay.

No conviene aplicar delays muy altos cuando se están realizando giros, cuando el código está ejecutando un delay deja de ejecutarse toda lógica y los motores mantiene su sentido de movimiento, en ese momento el robot no percibe nada.



Por ejemplo, para rotar 90 grados si queremos dar una curva:

* **NO** colocamos la posición de rotación en los motores y realizamos un delay durante 600 ms (Tiempo estimado para girar 90º), corremos el riesgo de que el robot se pase la línea.
* **SI,** colocamos la posición de rotación, y ejecutamos un while que vaya evaluando la medición de los sensores centrales, cuando uno entre en contacto con la línea detenemos el giro.

|  |
| --- |
| void girarDerecha**(){**  // girar hacia la derecha  servoD**.**write**(**0**);**  servoI**.**write**(**0**);**  delay**(**400**);** // Pequeño delay para que le dé tiempo a salir de su carril actual  // Si no puede que este ya en un carril y detectaría línea    boolean carrilEncontrado **=** false**;**  **while(!**carrilEncontrado**)** **{**  Sensor2 **=** digitalRead**(**pinSensor2**);**  Sensor3 **=** digitalRead**(**pinSensor3**);**    **if(** Sensor2 **==** LINEA **||** Sensor3 **==** LINEA **){**  carrilEncontrado **=** true**;**  **}**  **}**    // Parar  servoD**.**write**(**90**);**  servoI**.**write**(**90**);**  **}** |

### Giros y colocación previa

\*Antes de realizar algunos giros puede ser necesario mover ligeramente el robot hacia delante o hacia atrás, para que pueda colocarse de forma correcta en el carril.

# 10.2 Recordar el camino (2 Punto)

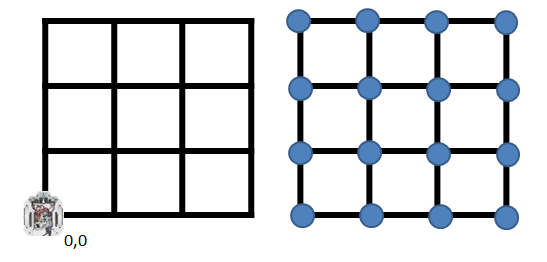
Uno de los cuatro puntos de la actividad se corresponde con que una vez encontrado el objetivo el robot se pueda volver a colocar en su posición inicial y repita la búsqueda del objetivo tomando únicamente caminos correctos.

Ver algoritmo explicado en teoría.

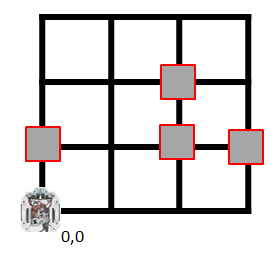
# 10.3 Mapeo y trayectoria esquivando obstáculos (6 Puntos)

Se colocará al robot en una matriz de 4 x 4 delimitada por líneas negras.

Debe utilizar los sensores de línea para moverse por la matriz.  
  
El robot de la coordenada 0,0 y debe realizar un mapeao completo que detecta que puntos de la matriz tienen un obstáculo.

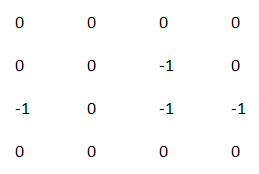


El robot conoce las dimensiones del mapa, 4 x 4 , pero no sabe que en ciertos puntos habrá obstáculos (que tiene que detectar con el sensor de ultrasonidos).



Debe tratar de recorrer todas las posiciones del mapa y determinar si en cada una de las 4 x 4 celdas hay obstáculos o no.

Internamente guardará la información del mapa en una matriz.



**La matriz debe mostrarse por la pantalla LiquidCrystal\_I2C (Ver práctica 8).**

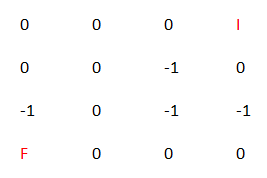


# 10.4 Volver a la coordenada 0,0 (2 Puntos)

#### Volver a la coordenada 0, 0

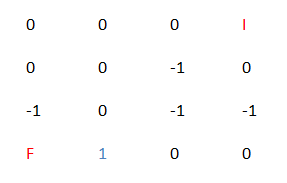
Una vez ha acabado de explorar todo el mapa tiene que volver a la coordenada 0 , 0.

Para calcular la trayectoria libre de obstáculos que debe seguir tiene que utilizar el **Wavefront** Ver teoría de Localización y trayectorias.

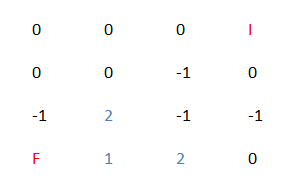


Aplicamos el algoritmo:

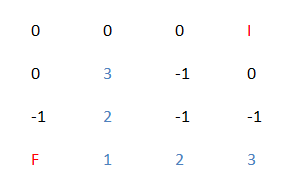
Paso 1:



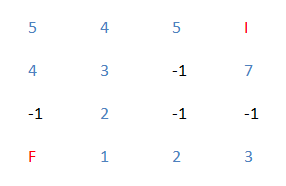
Paso 2:



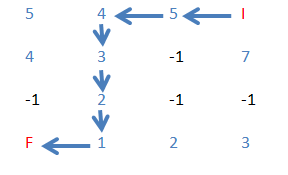
Paso 3:



Estado final:



Obtención de trayectoria:



También deberá poder ir a cualquier otra coordenada que se especifique, utilizando el mando de IR, las nuevas coordenadas se introducirán con el mando de IR únicamente cuando el robot este quieto.