Práctica 6. Arquitecturas robóticas

Hecho por Marcos Hidalgo Baños y Aquiles Fernández Gambero a día 20/05/2022

Planteamiento de la práctica.

La última práctica de la asignatura consiste en la implementación de una arquitectura basada en comportamientos que permita al LEGO EV3 *recolectar los alimentos* (foraging).

El robot **navegará** por el entorno controlado, delimitado por un contorno que deberá evitar, correspondiente al recuadro negro de la imagen. En su interior se colocarán objetos que deberá poder identificar y opcionalmente obstáculos que limitarán su zona de trabajo.

Aunque el objetivo es recoger los objetos, no estableceremos ningún tipo de ruta como aparece plasmado en la imagen, por lo que nuestro robot recorrerá sin aparente patrón el entorno hasta toparse con los objetos. Esto dará lugar a un recorrido algo **caótico**.

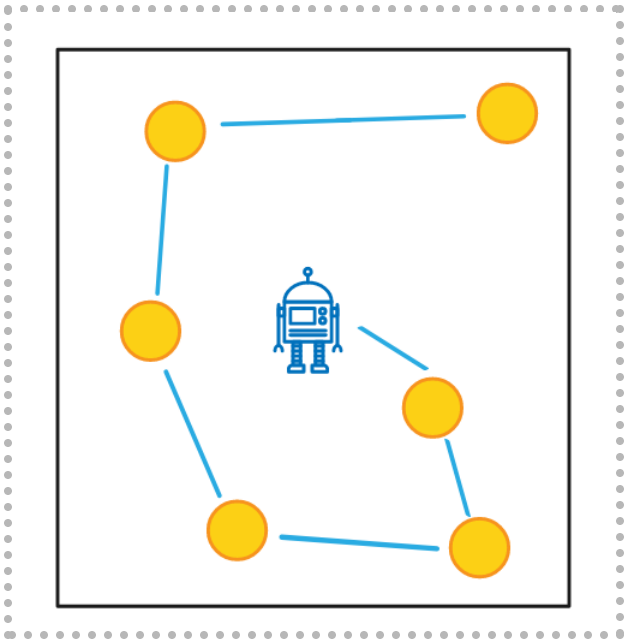


Imagen 1. Ruta idealizada a trazar por el robot.

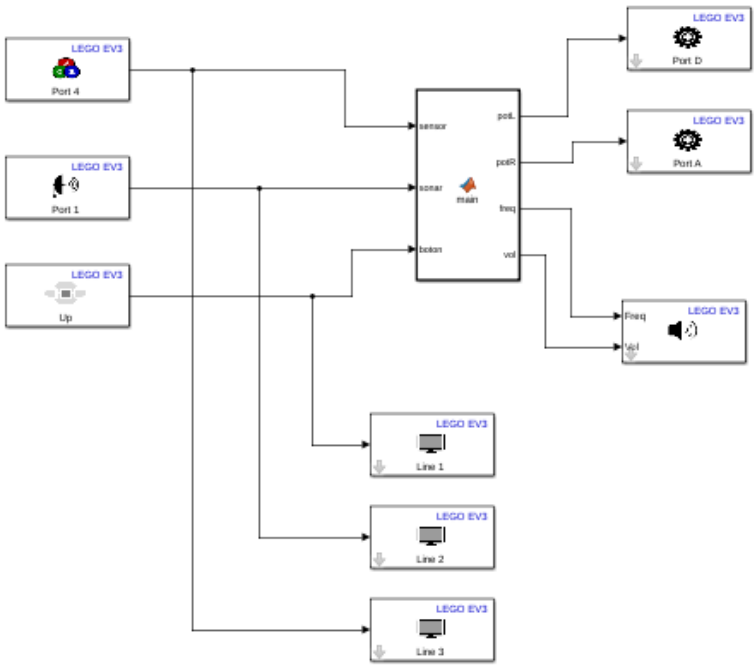
Desarrollo de la práctica.

La estructuración del código estará delimitada por cuatro principales componentes:

- El **bloque de control** principal, encargado del movimiento del robot.
- Un total de tres subrutinas exclusivas entre sí que permitirán al robot realizar unas acciones determinadas en función de la situación actual en la que se encuentre.
    - **Foraging**: Encargada del control de los estados del robot, nos ayudará a decidir qué comportamiento sigue nuestro LEGO, ya sea buscar su alimento o detenerse y de emitir el sonido cuando finalmente lo encuentra.
    - **ObstacleAvoidance**: Permitirá a nuestra máquina sortear y esquivar los obstáculos. Cuando el sónar encuentre uno enfrente, intentará dar marcha atrás y corregir su alineación.
    - **Wandering**: Es la subrutina de movimiento aleatorio, para que el robot vaya deambulando sin rumbo hasta que encuentre su alimento.

Con esta distinción, se facilitará la implementación del código todo lo posible al haber dividido el código en tres problemas independientes que pueden ser programados por separado. De esta manera, habremos conseguido implementar una arquitectura robótica fácilmente **exportable**, **manipulable** y **legible**.

Circuito implementado en Simulink.



Cuerpo del bloque principal de código.

Tal y como se muestra en la imagen anterior, toda actividad detectada por el robot mediante su *sensor*, *sonar* y *botón central* será transmitida al consiguiente bloque de código Matlab, el cual será el encargado de enviar los valores de *potencia* a cada rueda y el *volumen* de la *frecuencia* del pitido al altavoz.

```
function [potL, potR, freq, vol] = main(sensor, sonar, boton)
p = 50; % potencia motores
freq=0; % in Hz
vol=0; % from 0 to 100
potL=0; % pot motor izq
potR=0; % pot motor der

% estado = true --> buscar
% estado = false --> parado
persistent estado
if (isempty(estado))
    estado = true;
end
```

En primer lugar, comentar brevemente las variables más importantes usadas a lo largo del código.

La más representativa de todas es **estado**, ya que nos permitirá distinguir entre el momento de la búsqueda de objetivos y el bloqueo al detectarlos.

Durante ciertos momentos de la ejecución, se realizarán las llamadas a las subrutinas que dirigirán el flujo de ejecución, las cuales permitirán distinguir entre estados:

```
if (estado) % Estoy buscando
    % Compruebo si veo un papel
    detected = Foraging(sensor);
    if (detected) % Detecto algo
        estado = false;
    else % Sigo sin ver nada
        % Compruebo si tengo obstaculo
        [ignore, pl, pr] = ObstacleAvoidance(sonar,p);
        if (ignore) % Deambulamos
            % Movimientos aleatorios
            [pl, pr] = Wandering(p, randi([1,100],[1,1]));
        end
        potL = pl;
        potR = pr;
        estado = true; % Redundante

    end
else % he encontrado algo
    freq = 528; % in Hz
    vol=20; % from 0 to 100
    % boton = 0 --> Not pressed
    % boton = 1 --> Pressed
    if (boton)
        estado = true;
    end
end
```

Foraging. Devuelve **detected = true** si el sensor encuentra un papel.

ObstacleAvidance. Determina si podemos seguir deambulando o debemos retroceder. También nos devuelve la potencia de motores necesaria para ello.

Wandering. En el caso de no haber encontrado un obstáculo, podremos seguir proporcionando de manera aleatoria valores de potencia a los motores. Toma como parámetro un número aleatorio para determinar qué movimiento de los tres posibles (ir recto, girar hacia la derecha o girar hacia la izquierda) tomaremos en esta iteración concreta.

```
%------------------------------
% Estado del robot y del altavoz
%------------------------------
function detected = Foraging(sensor)
umbral_luz = 50;

if (sensor > umbral_luz) % ha encontrado un objeto
    detected = true;

else % no he encontrado nada
    detected = false;
end
```

El principal cometido de esta subrutina es comprobar que la intensidad obtenida por el sensor no supere un cierto valor de umbral (obtenido de manera empírica) pues esto supondría que hemos detectado un papel. Si este es el caso, se devuelve true a través de la variable **detected**.

```
%-----------------------------------
% Deteccion obstaculos y pot motores
%-----------------------------------
function [ignore, pl, pr] = ObstacleAvoidance(sonar,p)
umbral_dist = 35; %cm

if (sonar < umbral_dist)
   pl = -p;
   pr = -20;
   ignore = false;
else
   ignore = true;
   pl=0;
   pr=0;
end
```

De manera muy similar, para la detección y reajuste de potencias tras la detección de un obstáculo tendremos un valor umbral (también obtenido empíricamente) que permita distinguir cuándo el sónar nos muestra un valor lo suficientemente alto como para hacer que el robot se detenga.

```
%---------------------------------
% Movimientos aleatorios del robot
%---------------------------------
function [pl, pr] = Wandering(p, n_iteraciones)
pl_backup = p;
pr_backup = p;
if (n_iteraciones < 20) %Giro der
   pl_backup = randi([-p,p],[1,1]);
   pr_backup = -pl_backup;
elseif (n_iteraciones < 40) %Giro izq
   pr_backup = randi([-p,p],[1,1]);
   pl_backup = -pr_backup;
end
pl = pl_backup;
pr = pr_backup;
```

La implementación de esta subrutina es la más compleja de comprender de todas, porque por razones de diseño hemos decidido que el parámetro principal **n_iteraciones** sea un número aleatorio entre 1 y 100. Dependiendo de en qué segmento se encuentre, realizaremos una determinada asignación a los valores de potencia. La distribución de estos valores está completamente intencionada.

Conclusiones generales.

Tras haber dedicado tiempo y esfuerzo en la elaboración del código Matlab, nos ha resultado bastante llamativo la **facilidad** con la que se puede ordenar al robot para que realice las acciones descritas anteriormente en el informe. Sin embargo, la mayor dificultad encontrada ha sido la **compleja** red de estados excluyentes y la correspondiente sintaxis anidada de condicionales (sentencias if, llamadas a subrutinas, etc) que han hecho la elaboración del script un auténtico 'trabalenguas'.

Otro aspecto que nos llamó la atención fue la **torpeza** con la que el robot va a navegar por el entorno en busca de los papelillos, algo que es de esperar viendo la implementación de la subrutina Wandering. Tal y como aparece descrito en la imagen del comienzo del informe, un comportamiento algo más inteligente por parte del robot sería que detectase desde lejos aquellos objetivos que tuviera a su alrededor y trazase una ruta.

Evidentemente, con los **componentes** que posee nuestro LEGO EV3 esta iniciativa pierde sentido al no poseer una cámara integrada o una 'vista de pájaro' que le permita al robot desplazarse por la ruta óptima. Tal vez si dispusieramos de una camarita pudiéramos hacer algún truco de Visión por Computador que permita distinguir los papeles blancos del suelo.