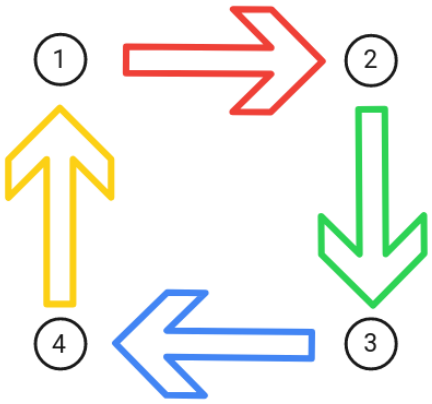
Práctica 5. Localización y mapeado

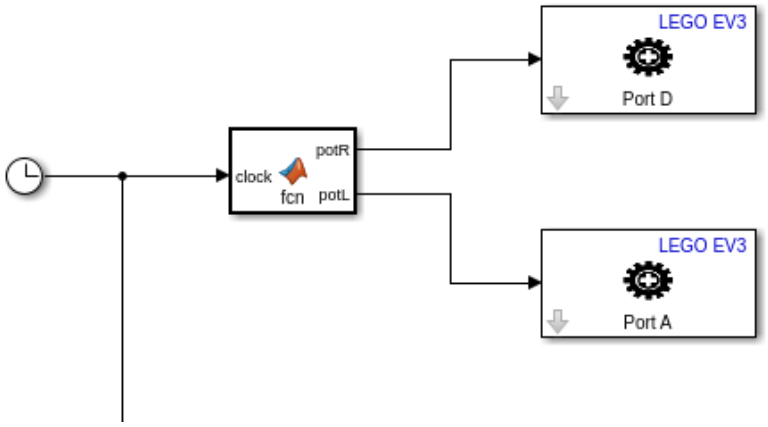
Hecho por Marcos Hidalgo Baños y Aquiles Fernández Gambero a día 09/05/2022

Planteamiento de la práctica.

En esta ocasión haremos que nuestro querido robotito LEGO EV3 recorra una loseta del suelo en sentido horario, tal y como se muestra en la imagen. Este proceso lo realizaremos únicamente mediante **estimaciones temporales**, que emplearemos para calcular la distancia recorrida en cada tramo de manera que sepamos cuando cambiamos entre el estado de línea recta (pasos 1, 3) y el estado de giro (pasos 2, 4).



Implementación en Simulink.



```
function [potR, potL] = fcn(clock)

    %Constantes generales
    P = 50; % potencia de las ruedas
    t_recta = 1.5 % tiempo en hacer una linea recta
    t_giro = 1 % tiempo en girar

    persistent ultimotiempo
    if (isempty(ultimotiempo))
        ultimotiempo = 0;
    end

    persistent estado
    if (isempty(estado))
        estado = false; % 0 --> Recto; 1 --> Giro
    end

    persistent total_time; % tiempo en segundos
    if (isempty(total_time))
        total_time = 0;
    end

    %Variables utiles
    Radio = 2.7 % cm
    Dist_Ruedas = 11.7 % cm
```

Como ya hemos comentado, nuestra función toma como input la medición del tiempo en ejecución para poder ajustar un valor determinado de potencia (50) en los motores del LEGO EV3.

Para ello, ajustaremos los tiempos de cada estado en **t_recta** y **t_giro** mediante ensayo y error.

Por último, tendremos que declarar dos variables persistentes: **últimoTiempo**, que nos permitirá saber el tiempo transcurrido desde la última iteración del programa, y **estado** que será la variable booleana que nos distinguirá entre cada segmento del circuito.

```
    % -------------------------
    % DESARROLLO DE LA PRACTICA
    % -------------------------

    deltat = clock-ultimotiempo;
    if (not(estado))
        % Mover en linea recta:
        potR = P;
        potL = P;
        total_time = total_time + deltat ;
        if (total_time > t_recta)
            total_time = 0;
            estado = true;
        end
    else
        % Girar a la derecha:
        potR = 0;
        potL = P;
        total_time = total_time + deltat ;
        if (total_time > t_giro)
            total_time = 0;
            estado = false;
        end
    end
ultimotiempo = clock;
```

Debido a la simplicidad del circuito, sabemos que una vez acabemos de realizar una línea recta siempre pasaremos al estado de giro y viceversa.

Esta transición la haremos solamente si el tiempo total de ejecución es mayor que el tiempo estimado de estado, lo cual implica que hemos alcanzado el nodo esquina correspondiente.

Como dato curioso, el robot permite realizar los giros hacia la derecha parando el motor de la rueda derecha a la vez que mantenemos la potencia de la izquierda. Esto funciona de manera inversa para girar hacia la izquierda, pero en esta ocasión (de nuevo por la simplicidad del recorrido) no necesitaremos trazar giros antihorarios.

Localización off-line.

Una vez conseguido que nuestro robot realice el recorrido deseado, procederemos a añadir nuevos componentes al circuito con el objetivo de mantener trackeada la posición relativa del robot a lo largo del camino. Para ello, deberemos hacerle saber a nuestro robotito cuáles son las posiciones que va a tomar y su correspondiente tiempo, que junto a la determinación del diámetro de las ruedas y la distancia que las separa nos permitirán graficar (en el último apartado del informe) los resultados de la odometría calculada por Matlab para el experimento y de la odometría calculada por el robot.

Contenido de la función 'logger'

Para la localización off-line, es necesario utilizar datos y variables que han sido guardadas en un fichero externo durante la primera etapa de la práctica. El archivo se trata de **muestras_p51.log** y tiene está formateado como muchas líneas de texto, todas de números enteros, que matlab interpretará como una matriz de seis columnas.

```
function [t0,rotl0,rotr0,x1,y1,theta1] = logger (stop)
persistent myFile;

if (isempty(myFile))
    myFile= fopen('C:\Users\alumnopr09\Downloads\','r');
end

if(myFile>=0)
    if(stop>0)
        fclose(myFile);
    else
        %Imprimimos los datos dentro del archivo
        a = fscanf(myFile,'%d');
        t0=a(0,1);
        rotl0=a(0,2);
        rotr0=a(0,3);
        x1=a(0,4);
        y1=a(0,5);
        theta1=a(0,6);
    end
end
```

La función logger se trata de un código bastante sencillo, sin ningún tipo de complicaciones. Con fopen abriremos el archivo antes mencionado y hasta que éste se acabe lo estaremos escaneando de principio a fin, asignando a cada variable de salida los datos que va leyendo uno a uno en cada línea en el siguiente orden:
T0, rotl0, rotr0, x1, y1, y theta1.

IMPORTANTE: Esta función ha sido modificada después de la redacción del informe, la versión final es la que aquí se muestra.

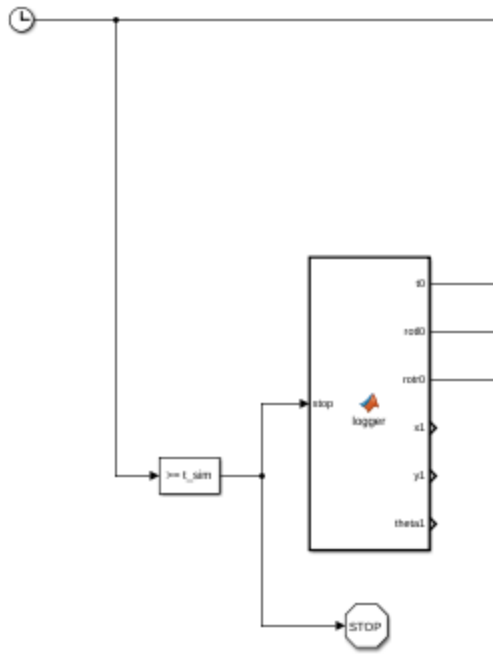
Contenido del fichero Odometry.m

Como su nombre bien explica, este fichero se encargará de implantar el modelo cinemático del robot en función de nueve variables: La última pose calculada **(x0, y0, theta0)** , el tiempo que marcaba el cronómetro cuando se tomó esa pose **(t0)** , lo que los encoders de las ruedas medían en aquel momento **(rotl0, rotr0)** y el tiempo actual que marca el cronómetro **(t1)** junto a los valores actuales que marcan los encoders **(rotl1, rotr1)**.

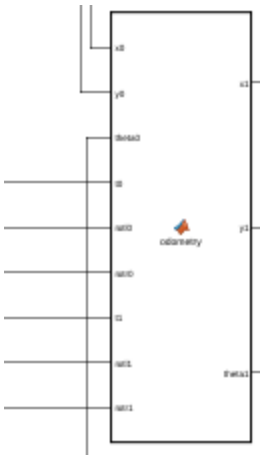
Los primeros seis valores en la primera ejecución del código serán 0, y se actualizarán conforme vaya avanzando el tiempo. Sin embargo, los valores x0, y0 y theta0 dependerán de las salidas futuras (x1, y1, theta1) mientras que rotl0, rotr0 y t0 serán los datos que leímos del archivo de texto externo.

$$\Delta x = \frac{\Delta\theta_l \cdot R + \Delta\theta_r \cdot R}{2} \cdot cos(\theta_0)$$
$$\Delta y = \frac{\Delta\theta_l \cdot R + \Delta\theta_r \cdot R}{2} \cdot sin(\theta_0)$$
$$\Delta\theta = \frac{\Delta\theta_r \cdot R - \Delta\theta_l \cdot R}{D}$$

Entradas y salidas de 'logger'



Entradas y salidas de 'odometry'



IMPORTANTE:

Este bloque ha sido modificado después de la redacción del informe, la versión final es la que aquí se muestra.

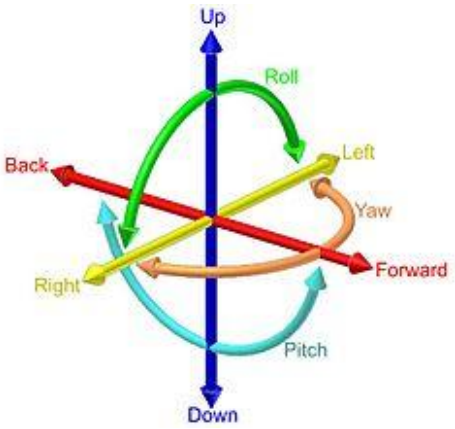
Implementación del modelo cinemático.

```
function [x1,y1,theta1]=odometry(x0,y0,theta0,t0,rotl0,rotr0,t1,rotl1,rotr1)
    D = 11.7;
    R = 2.7;
    deltaThetaL = rotl1 - rotl0;
    deltaThetaR = rotr1 - rotr0;
    deltaTiempo = t1 - t0;

    if deltaTiempo == 0
        x1 = x0;
        y1 = y0;
        theta1 = theta0;
    else
        x1 = (deltaThetaL * R) + (deltaThetaR * R) / 2 * cos(theta0);
        y1 = (deltaThetaL * R) + (deltaThetaR * R) / 2 * sin(theta0);
        theta1 = (deltaThetaR * R) - (deltaThetaL * R) / D;
    end

end
```

Es interesante también hacer notar que D es la distancia en centímetros entre cada rueda y que R es el radio de estas (en centímetros también), valores que nos serán necesarios para calcular x1, y1 y theta1. El resto del código trata de asegurarse de que cuando el tiempo sea 0 todas las variables lo sean también y de traducción de las expresiones matemáticas.

¿A qué modelo cinemático de los vistos en clase se corresponde?

Podemos descubrir qué clase de vehículo es nuestro robotito según sus **grados de libertad**. Recordemos que este concepto podía tratarse desde dos perspectivas distintas:



Definición A: Punto de vista Matemático, mínimo nº de valores independientes que definen la posición y orientación. En nuestro caso, **3: Eje X, Eje Y,** y la **orientación,** pero no Roll, pitch o yaw**.**

Definición B: Punto de vista Ingenieril, número de actuadores independientes. Nuestro lego cuenta con **2, uno por cada motor.**

Empecemos por descartar el modelo de la bicicleta, porque el LEGO no tiene ruedas traseras fijas ni ruedas delanteras, solo hay dos y están alineadas horizontalmente.

Siguiendo las definiciones descritas, que el grado de libertad A sea mayor que el B nos revela que se trata de un vehículo no holónomo, parecido a un coche. No siempre puede moverse de una pose a otra directamente y hay que maniobrar en ocasiones.

Esta información es esencial, pues una de las características generales del **modelo diferencial es** que es **un modelo no holónomo**, muy utilizado en robots con 2 ruedas.

Localización on-line.

En la localización online, el robot calcula a tiempo real la odometría en cada iteración usando el mismo modelo cinemático. El siguiente esquema trata de representar la idea general del modelo, ya que este debe considerar las salidas de iteraciones anteriores para calcular la pose actual del robot.

Como se puede observar, la retroalimentación realizada se realiza 'a lo bruto' mediante cableado en Simulink. Sin embargo, se debería realizar mediante el uso de variables persistentes pues Matlab no asimila de manera adecuada este tipo de arreglos.

Versión final del proyecto.