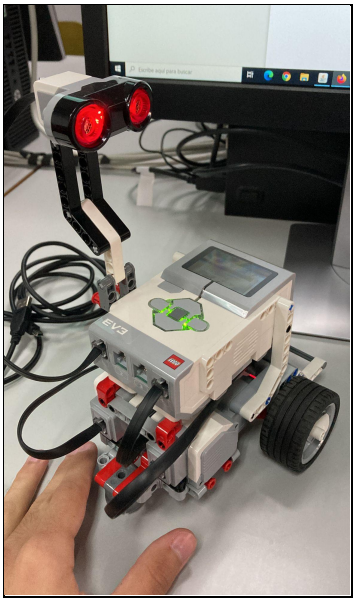
Práctica 2. Microbots en BA y BC

Hecho por Marcos Hidalgo Baños y Aquiles Fernández Gambero a día 21/03/2022

Ejercicio 1. Control en bucle abierto



El propósito de los primeros dos ejercicios es hacer el mismo programa en bucle abierto y cerrado, que consiste en hacer al **Robot Lego EV3** girar las ruedas 860 grados cada una y parar el programa. En éste primer ejercicio, lo haremos con el bucle abierto, así que intentaremos calibrar y encontrar un valor de Potencia adecuado para las ruedas de forma que gire los 860 grados que buscamos, teniendo en cuenta la franja de error de 10 grados arriba y abajo.

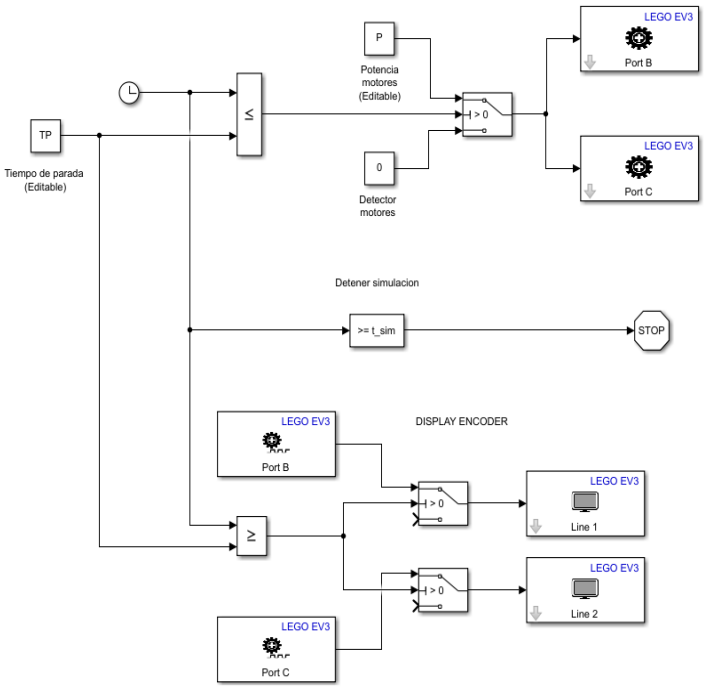
Recapitulando un poco, el control en bucle abierto se traduce en que la referencia, y por tanto, la acción de control es independiente de la salida del sistema. Por ello, muchas veces los sistemas de *Open loop* son sensibles a perturbaciones, que deben de ser corregidas con calibraciones.

Para este ejercicio tendremos que definir una serie de variables directamente en Matlab:

- **TP**: Tiempo de parada medido en segundos, en cuanto el reloj supere éste valor las ruedas se detendrán. Por ejemplo: 2 segundos.

- **t_sim**: Tiempo total de simulación en segundos, hemos puesto 10.

- **P**: Potencia de los motores. Ésta es la variable que deberemos de tantear y calibrar. Hemos escogido un valor de 80.



El circuito está diseñado para que una vez el reloj supere el tiempo estipulado por t_sim, la simulación se detenga. En la parte norte del circuito encontramos el bloque Constante P (que definiremos en matlab) conectado a un bloque switch. Una vez termine el tiempo de espera TP, la potencia de los motores pasará de P a 0, deteniendo los motores.

El hemisferio sur del diagrama muestra el codificador de cada rueda conectado al bloque *Display*, lo que mostrará por pantalla en grados lo que cada rueda recorre.

Es interesante hacer notar que, como se trata de un sistema de bucle abierto, es sensible a las perturbaciones. Por lo tanto, las medidas tomadas cuando el robot se encuentra apoyado en el suelo (Imagen 1) serán distintas de las tomadas cuando sujetamos al robot en nuestra mano, que serán mayores (Imagen 2).

Imagen 1. El peso del robot afecta a las mediciones

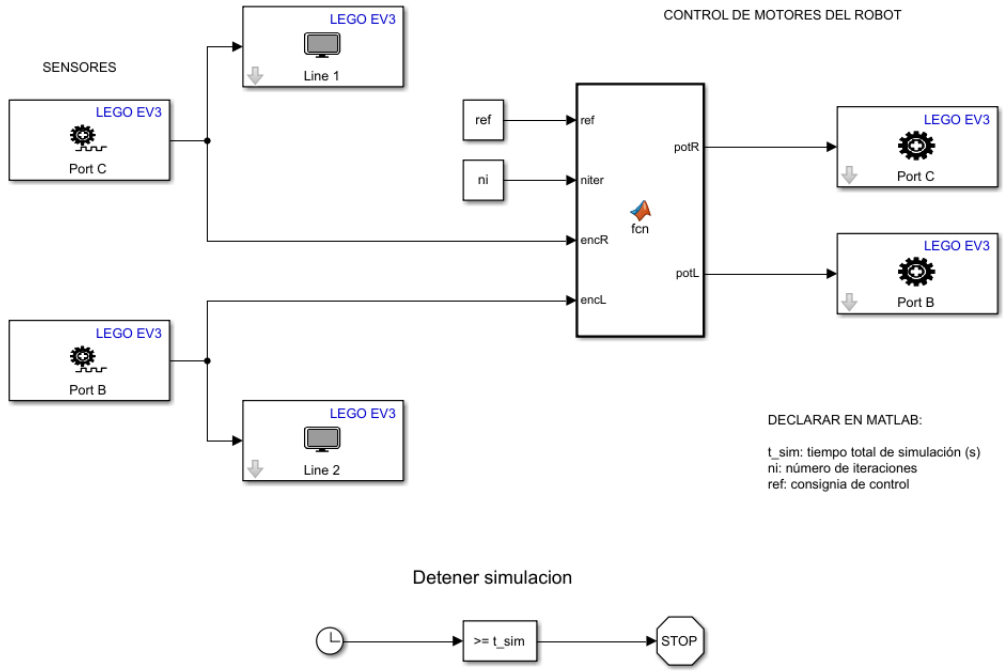
Imagen 2. Contrarrestamos el peso al sostenerlo

Ejercicio 2. Control en bucle cerrado

En los sistemas de bucle cerrado la referencia varía con la salida gracias a la nuevamente introducida retroalimentación. Normalmente consiste en configurar acordemente el ajuste de control a la salida. El controlador en éste ejercicio funciona mediante un todo o nada, (*on* u *off*) lo que quiere decir que las ruedas o se moverán con la Potencia designada o estarán paradas.

Otro aspecto a destacar es que no tendremos que ajustar ningún parámetro a mano en el controlador del bucle cerrado (que veremos a continuación). Esto es debido a que la retroalimentación implementada permite al controlador ajustar la acción de control según la salida del sistema en la iteración anterior.

El circuito es algo distinto al del bucle abierto. Los motores y la función no dependen del tiempo de la simulación, sino del número de iteraciones. En éste circuito deberemos definir en matlab un número de iteraciones $\mathbf{n_i}$ y una consigna de control **ref**.




```
function [potR,potL]    = fcn(ref,niter,encR,encL)
    persistent iter;

    if (isempty(iter))
        iter = 0;
    end

    iter = iter +1;

    if(iter <= niter)
        potR = int32(ref-encR);
        potL = int32(ref-encL);

        if(potR>100)
            potR=int32(100);
        elseif(potR<-100)
            potR = int32(-100);
        end

        if(potL>100)
            potL=int32(100);
        elseif(potL<-100)
            potL = int32(-100);
        end

    else
        potR =int32(0);
        potL =int32(0);
    end
```

El bloque función es interesante de analizar, ya que requiere programación en Matlab:

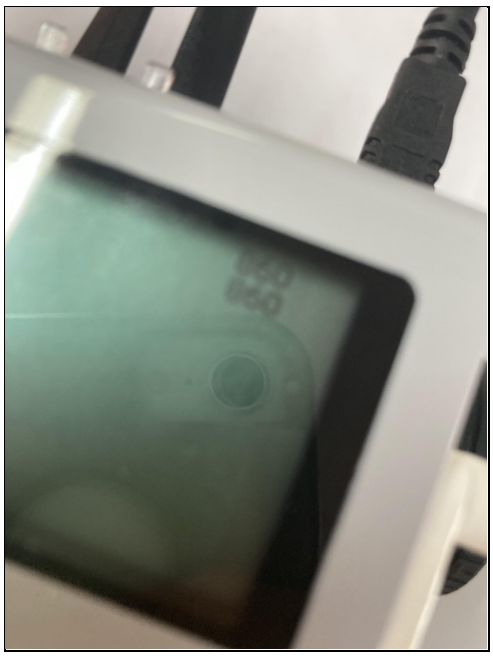
La función tendrá como entrada la consigna de control y los bloques codificadores de cada rueda, **ésta será la retroalimentación**.

Tras declarar las variables persistentes y actualizar la variable iter, entramos en el bucle. En éste designamos la potencia de cada rueda según la retroalimentación obtenida.

Comenzamos por hacer el casting a int32 a la resta de la consigna y las variables de entrada, y llegamos a unos pequeños bloques if, que se encargarán de acotar y ajustar la potencia. Si las señales son mayores que 100, las dejarán acotadas a 100. Por otro lado, si es -100, se quedarán acotadas también a -100.

Cuando acaben las n_i iteraciones, finalmente la potencia que se transmitirá a los motores será de 0 y se detendrán.

Comparación y efectos ante perturbaciones.

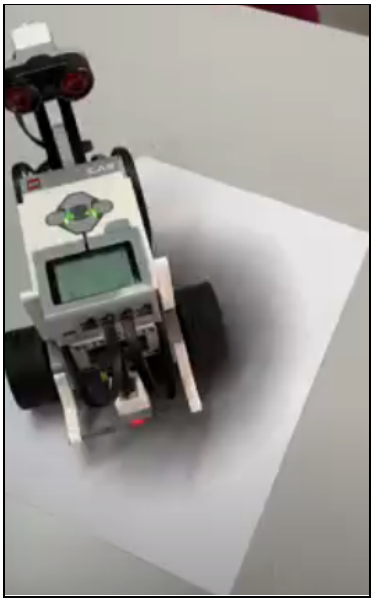


Los valores en grados de cada rueda en éste caso son muchísimos más **precisos** y **exactos** que los del bucle cerrado. En ambos casos, tanto agarrando el robot como dejándolo en libertad en el suelo, las ruedas giran un total de 860 grados exactos.

Como hemos comprobado, el *open loop* es más sensible a perturbaciones. Ésta es la naturaleza del bucle abierto, debe de ser calibrado y pre–programado en concreto para tener en cuenta éstos casos.

El bucle cerrado, por otro lado, es más versátil y resistente a perturbaciones debido a la retroalimentación, que modifica después la referencia y la señal de entrada acorde a los ajustes que sean necesarios implementar.

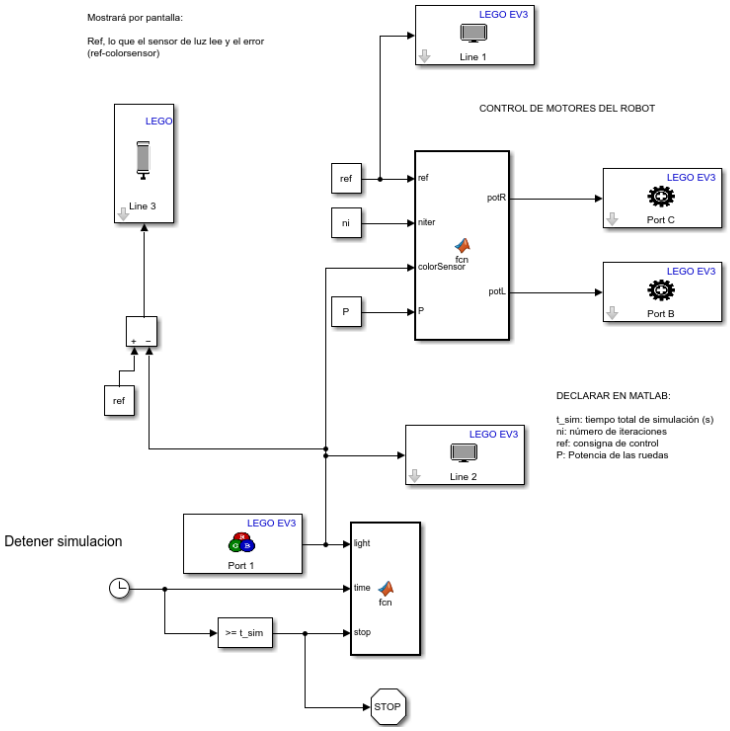
Ejercicio 3. Seguimiento de líneas



El control de bucle cerrado tiene como ventaja también que el algoritmo de control puede ser implementado de distintas formas a parte del ya conocido todo o nada, como el método ***bang-bang***, en el que se tiene en cuenta una pequeña banda de histéresis a la hora de tomar acción. (Definida como **U=5**)

Éstos algoritmos más sofisticados se pueden traducir en interesantes implementaciones, como el seguimiento de líneas de un robot. En éste ejercicio se nos entregó una hoja de papel con distintos valores de grises, y el propósito del ejercicio era designar un tono de gris en concreto para que el robot lo siga en todo momento. Al ser una elipse y no una línea recta, el robot encontraría perturbaciones constantemente, habiendo de corregirlas pertinentemente cambiando la alineación del robot.

El Circuito de simulink en éste ejercicio es uno sencillo también ésta vez. Quitando los bloques para mostrar por pantalla los valores, éste se quedará a la mitad de bloques que los otros circuitos anteriores, aunque a la hora de hacer ajustes en el código y de comprobar el funcionamiento del robot éstos bloques nos ayudaron enormemente.



Como en el Bucle Cerrado On-Off, el circuito depende de una **referencia,** un **número de iteraciones n_i**, y una **potencia P,** que deberán ser definidas en matlab. (Además del tiempo de simulación deseado).

El número de iteraciones debe ser simplemente un número grande (2000) para comprobar que el robot funcione y dé una vuelta al circuito, por lo que no requiere de un valor preciso.

Por otro lado, el valor de referencia se tratará del tono de gris que el robot deberá de seguir, medido por el sensor de luz en una escala del 0 (negro) al 100 (blanco). Éste valor hubo de ser tomado en el laboratorio el día de la práctica, puesto que como el sensor trabaja con luz reflejada puede haber condiciones externas que afecten a las lecturas.

Nosotros, tras colocar el robot en la elipse del papel y medir con el sensor de luz el tono en concreto reflejado, anotamos 22 y usamos ese valor.

```
function [potR,potL]    = fcn(ref,niter,colorSensor,P)
    persistent iter;

    if (isempty(iter))
        iter = 0;
    end

    iter = iter +1;
    colorSensor = int32(colorSensor);

    if(iter <= niter)
    %Calculo del error

         err = ref-colorSensor;

        %Capa la potencia de cada rueda. error debería de acercarse lo
        %máximo posible a 0, que es cuando la potencia será P. En caso de
        %ser mayor que 5 o menor que -5 se parará para corregir la
        %alineación
        if(err>5)
            potR = int32(-P);
            potL = int32(P);
        elseif (err<=-5)
            potR = int32(P);
            potL = int32(-P);
        else
            potR = int32(P);
            potL = int32(P);

        end
    else
        potR =int32(0);
        potL =int32(0);
    end
```

En éste caso también es interesante analizar el código del bloque función, que tomará como entrada la referencia, el número de iteraciones, la lectura del sensor y la potencia estipulada.

Tras inicializar las variables persistentes y actualizar el contador, es **importante hacerle un *casting* de 32 bits a la lectura del sensor de color.** De otro modo, la sentencia if no funcionará correctamente.

Entramos en el bucle, y lo primero que encontramos es el cálculo del error como la diferencia entre la referencia que tomamos (22 en éste caso) y el valor que en realidad lee el sensor.

En un escenario perfecto, éste error será de 0, o por lo menos, no mayor que 5 o menor que -5, lo que se traduce en que no habría que corregir la potencia y ésta podría seguir siendo positiva, desplazando el robot hacia delante.

Si la medida fuese más clara de lo esperado, el error calculado sería menor que menos cinco **(se saldría de la banda de histéresis)** y significaría que nuestro *Lego* se está saliendo de la elipse. Por lo tanto, para corregir la alineación le suministraremos una potencia positiva a la rueda derecha y una negativa a la rueda izquierda, que hará que ésta rueda vaya hacia atrás y el robot gire sobre sí mismo hasta volver a entrar en la elipse.

Del mismo modo, si el error calculado fuese mayor que 5, la alineación del robot sería corregida dotándole a la rueda izquierda de una potencia positiva y a la rueda derecha de una negativa.

Finalmente, tras acabar las 2000 iteraciones el robot se detiene y termina el programa.