Práctica 3. Microbots controlador PID

Hecho por Marcos Hidalgo Baños y Aquiles Fernández Gambero a día 25/03/2022

Prólogo: Objetivos de la práctica

Si bien ya hemos cubierto el funcionamiento de los circuitos de bucle abierto y de dos algoritmos de control del bucle cerrado, tales como On-Off y Bang-Bang, en ésta práctica cubriremos el funcionamiento mediante un **controlador PID**.

Para éste ejercicio, nos fue entregado el Lego Mindstorm y el contorno delineado de la provincia de Granada para el cual deberemos programar un código Matlab/Simulink para el seguimiento de líneas usando un controlador PID.

El controlador PID, al igual que el on-off o el Bang-Bang, se trata de un controlador de los algoritmos de bucles cerrados. Lo atractivo de éste es que nos permitirá ajustar la Potencia de los motores a través de un circuito de retroalimentación teniendo en cuenta la diferencia entre la variable real y la deseada.

El Algoritmo PID trabaja con tres parámetros diferentes:

- El **Proporcional**: Que dependerá del error actual.
  - Dada por: $u = K_p \cdot e$
- El **Integral**: Que dependerá de los errores pasados.
  - Dada por: $u = K_i \int e(t)\,dt$
- El **Derivativo**: Que es una predicción del error futuro.
  - Dada por: $u = K_d (d(e) / d(t))$

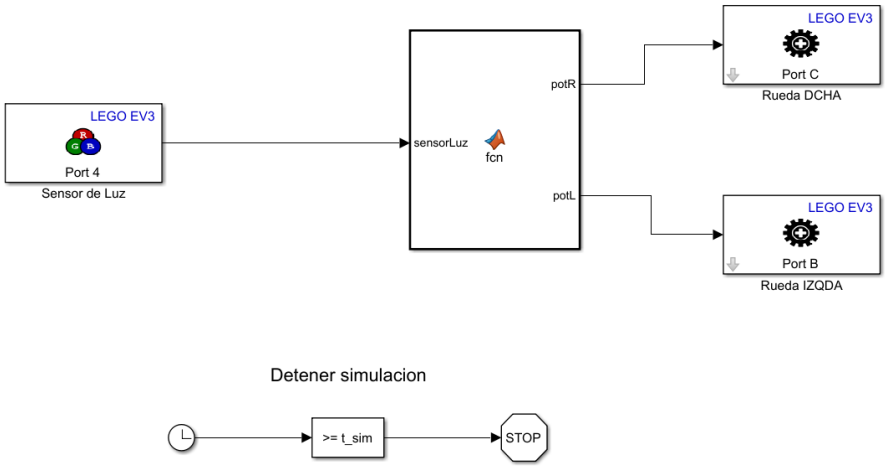
Estos valores serán posibles ser puestos en común mediante la suma de éstos tres, lo que nos permitirá ajustar el proceso mediante el control de la potencia suministrada a los motores. Así pues, la expresión final que sigue el controlador se trata de:

$$U = K_p \cdot e + K_i \int e(t)\,dt + K_d (d(e) / d(t))$$

Planteamiento general del programa.

Como ha sido establecido, el propósito de la práctica es programar un código de seguimiento de líneas. Sin embargo, es interesante hacer notar que en vez de seguir la línea, el robot **seguirá el contorno de ésta**, ya que es más sencillo detectar de ésta forma cuando el robot se sale de ésta y cuando permanece en ella.

Esta vez, el circuito en simulink será increíblemente sencillo, dependiendo por completo de un bloque función y de las lecturas del sensor de luz, al contrario que en la práctica 2 que debían ser introducidos consignas de control, valores de referencia o bloques *encoder* relacionados con los motores.



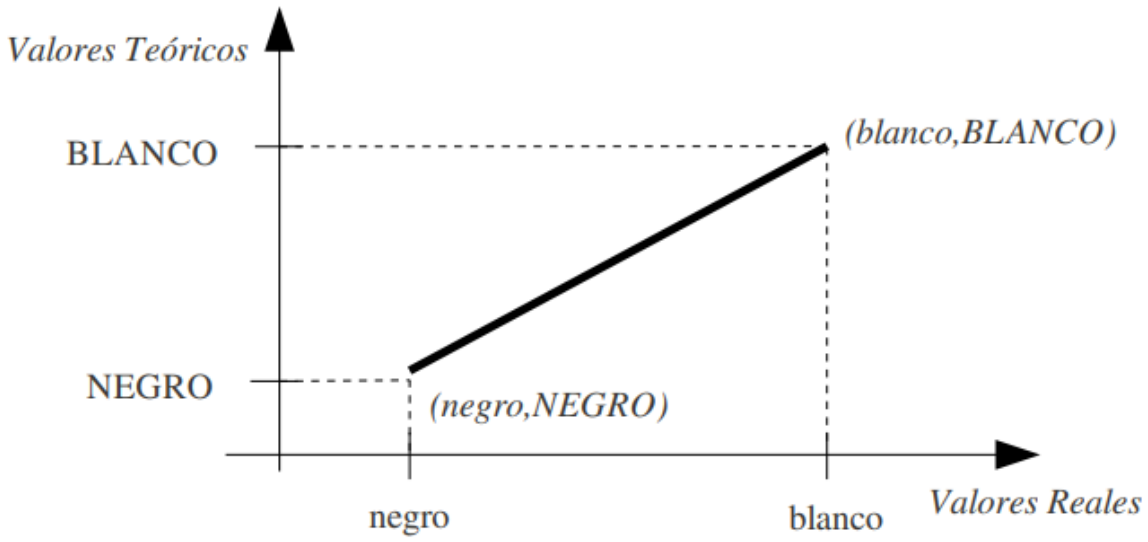
Por lo tanto, el grueso y la dificultad de la práctica se encontrará en el código del bloque función en sí, el cual seguirá el siguiente esquema:

1. Definición de Constantes
2. Método Main
    a. Declaración de variables
    b. Calibración del sensor de luz
    c. Algoritmo de control en bucle
        i. Lectura del sensor
        ii. Ajuste de la lectura
        iii. Actualización del controlador
        iv. Ajustación con Wind-Up.
        v. Ajuste de la Potencia con motores de Saturación

Calibración de la luz del sensor.

Para hacer la calibración es importante establecer qué valores consideraremos Blanco y cuales consideraremos Negro. El Controlador Wind-UP funciona con valores teóricos perfectos, que serán las constantes que necesitaremos para el código, que definiremos como **BLANCO** (valor 75) y **NEGRO** (valor 35).

Es importante hacer notar que las condiciones del entorno influyen en las lecturas y no serán las mismas siempre, por lo que es necesario normalizar/escalar los valores reales de la luz que nos devuelve el sensor. Así que antes de empezar situaremos el robot en la parte blanca del papel y luego en la negra, tomaremos los valores reales de ambos colores y crearemos una gráfica en la que colocaremos los valores reales en el eje de abscisas y los teóricos en el eje de ordenadas. Deberemos crear una recta que va desde el punto (negro,NEGRO) al (blanco,BLANCO), cuyos valores podremos utilizar en nuestro programa.



Análisis en profundidad del código.

```
function [potR, potL] = fcn(sensorLuz)
    % Constantes generales
    Kp = 0.5;
    Ki = 0.05;
    Kd = 0.1;
    BLANCO = 70;
    NEGRO = 35;
    P = 10;

    % Inicializacion de variables
    sat = 60;
    wup = 30;

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

    persistent lastError;
    if (isempty(lastError))
        lastError = 0;
    end
```

El bloque código recibe como entrada solo la señal del sensor detector de luz, que codifica la luz reflejada del entorno y del papel, y como en los bucles bang-bang o abiertos, la salida será la potencia de cada rueda por separado.

Siguiendo el esquema del código estipulado, comenzamos el algoritmo declarando las Constantes necesarias:

Kp, Ki, Kd, BLANCO, NEGRO y **P.**

También es importante declarar las variables sat (saturación) y wup (wind-up), que serán 60 y 30 respectivamente. Tras ello, inicializamos las variables persistentes: **integral** y **lastError** (necesarias para calcular la integral).

```
    % Calibracion de luz del sensor
    blanco = 69;
    negro = 6;

    % Lectura del sensor
    sensorLuz_d = double(sensorLuz);
    % Ajuste de lectura a los niveles del sensor
    lightValue = (sensorLuz_d*(BLANCO-NEGRO)/(blanco-negro))+((-negro*(BLANCO-NEGRO))+(NEGRO*(blanco-negro)))/(blanco-negro);
    error = lightValue - (BLANCO+NEGRO)/2;
    integral = integral + error;
    derivate = error - lastError;
    % Actualizacion de la actuacion del controlador
    turn = Kp*error + Ki*integral + Kd*derivate;
    % Ajuste de actuacion con WIND-UP
    if (turn > wup)
        turn =  Kp*error + Kd*derivate;
        integral = integral - error;
    end

    potR = P + turn;
    potL = P - turn;
    % Ajuste de la potencia de los motores con sat.
    if (potR > sat)
        potR = sat;
    end
    if (potL > sat)
        potL = -sat;
    end
    lastError = error;
```

El código continúa con la calibración del sensor, empezando con la declaración de los valores reales de luz y oscuridad que leímos en el mapa con el Lego Mindstorm y continuando con la lectura del sensor de luz, con un casting a double.

Como se ha comentado anteriormente, éstos valores serán necesarios para crear una función que normalizará el valor de la luz, para lo que será necesario usar la expresión abajo indicada, que se traducirá en la línea de código que le sigue en los que **y serán los valores deseados** y **x serán los valores del sensor de luz**. Así que es solo cuestión de despejar el término 'y' de la expresión.

$$\frac{(x-negro)}{(blanco-negro)}=\frac{(y-NEGRO)}{(BLANCO-NEGRO)}$$

```
lightValue=
(sensorLuz_d*(BLANCO-NEGRO)/(blanco-negro))+((negro*(BLANCO-NEGRO
)) +(NEGRO*(blanco-negro)))/(blanco-negro);
```

Se calcula el error como la diferencia entre la lectura escalada y normalizada y la media de los valores BLANCO y NEGRO. Esto queda reflejado en la línea del código:

```
error = lightValue + (BLANCO+NEGRO)/2;
```

Ahora llega la parte en la que se actualizan los errores pasados y en la que se intentan predecir los errores futuros. Debemos de modificar los parámetros integral y derivativos del algoritmo.

- Integral, al tratarse del parámetro basado en los errores pasados, será ajustado como sí mismo más el error de ésta iteración.
- Derivativo, que será declarada como la diferencia entre el error y el último error de la iteración más adelante en el código.

Llegamos a la parte en la que tiene lugar la **actuación del controlador PID**.

Como las variables integral, derivativa y error han sido calculadas anteriormente, es sólo cuestión de sumarlas todas y multiplicarlas por las constantes que le corresponden: K_p para el error, K_i para integral y K_d para derivativo.

Llegamos a la línea en la que se debe ajustar la actuación del wind-up. Es sabido que la acción de control integral puede generar el efecto wind up, que consiste en que el error no desaparece y la acción de control integral continúa aumentando, lo que puede derivar poco a poco en que por culpa del error en aumento se pierda el control del bucle o que comience a decrecer, pero no sea posible corregirlo tras tanto error acumulado.

Para evitar aquella situación desafortunada, implementamos el **valor umbral wind up**, para que si la acción de control total lo supera, ésta sea eliminada y se le reste al error integral el último valor de error.

```
% Ajuste de actuacion con WIND-UP
if (turn > wup)
    turn =  Kp*error + Kd*derivate;
    integral = integral - error;
end
```

La acción de control total se tratará de la suma de las tres acciones de control, la proporcional, derivativa e integral. Todo ésto se refleja en el código en una sentencia if, en la que si turn (el valor de la actuación del controlador) es mayor que wup (valor umbral Wind up) se re-evalúa turn sin tener en cuenta la acción de control integral, y a ésta se le resta el útlimo error.

Finalmente, la potencia de las ruedas derecha e izquierda serán:

PotR = P + turn; ***PotL = P - turn;***

Merece la pena remarcar que, al igual que en el bucle bang-bang, si el error fuese cero el robot continuaría recto, pero si éste fuese positivo tenderá a ir a la derecha y si fuera negativo, el robot tenderá a cambiar su rumbo hacia la izquierda.

Es interesante también fijarse en el ajuste de la potencia con el valor de la Saturación. Para evitar que el robot haga movimientos toscos o bruscos, si la potencia es mayor que el valor **sat** definido al principio del código, ésta será limitada a este valor.

```
potR = P + turn;
potL = P - turn;
% Ajuste de la potencia de los motores con sat.
if (potR > sat)
    potR = sat;
end
if (potL > sat)
    potL = -sat;
end
lastError = error;
```

Epílogo: Conclusiones

Aunque sea considerablemente más difícil de implementar que el control en bucle abierto o los de bucle cerrado en bang-bang o en on-off, el controlador PID es realmente **sofisticado** y **elegante**. Es prácticamente independiente de cualquier señal externa, resistente a las perturbaciones del medio y muy autosuficiente.

Comparemos los circuitos en simulink del controlador PID elaborado en esta misma práctica (Imagen 1) y del controlador Bang-Bang de la entrega anterior (Imagen 2).

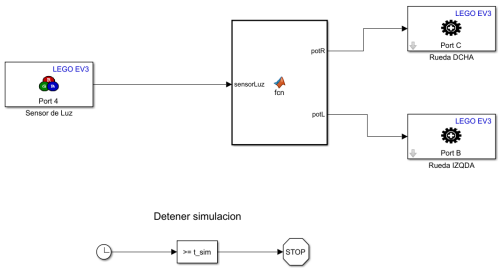


Imagen 1. Controlador PID

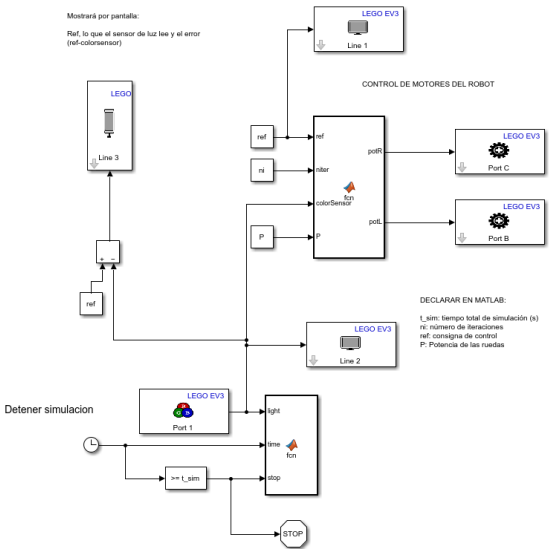


Imagen 2. Controlador Bang-Bang

Es fácil ver las diferencias entre ambos modelos, siendo el primero mucho más **sencillo** y **compacto** que su homólogo. Este aspecto se hace notorio en el número de componentes requeridos, el número de conexiones y la cantidad de variables que maneja el software (especialmente para las entradas de los bloques de función).

Es también pertinente mencionar que el código del bloque función del controlador PID es muchísimo más **robusto** que todos los demás, y mucho más **eficiente**. En las pruebas realizadas en el laboratorio pudimos percibir claramente cómo el robot se desplazaba de una manera mucho más fluida y dinámica que con los otros controladores.