DOCUMENTO Nº 2

Memoria de cálculo

Título del proyecto:

Diseño e Implementación de un Sistema de Control para un dispositivo de transporte equilibrado en dos ruedas teledirigido

Autor:

Francisco Javier Antón Díaz

Índice

Índice de ilustraciones

Índice de tablas

2.1.- Introducción al sistema físico

Las características físicas, como la altura, la masa, el centro gravedad, etc... própias de esta configuración del péndulo y las propias de los otros elementos, como la potencia y el par de los motores, serán considerados en la construcción y simulación del modelo matemático incorporándolas a las ecuaciones para reflejar así un comportamiento lo más ajustado posible al modelo real del robot.

El objetivo de la fase de modelado, es encontrar una expresión matemática que represente el comportamiento físico del sistema. Para modelar el sistema existen dos estrategias. La primera es tratar al sistema como un “caja negra" y realizar sobre el un conjunto de acciones (señales de entrada) observando cómo se comporta (estudiar las salidas) deduciendo un modelo matemático para este. Un ejemplo sería la técnica de Ziegler-Nichols. La segunda consiste en estudiar los procesos físicos que tienen lugar en el sistema para deducir su ley de comportamiento. El resultado que se pretende es la identificación del sistema a través de su función de transferencia.

El péndulo invertido es un problema muy habitual dentro del estudio de la ingeniería de control, con la diferencia de que el problema clásico consiste en un carro móvil sobre el que se encuentra un sólido que se pretende mantener en equilibrio. En nuestro caso partimos de un único cuerpo que incluye ruedas, motores, sensores y demás dispositivos.

Originalmente se puede considerar el problema del péndulo invertido como un sistema no lineal, pero la dificultad que implica el modelado de estos sistemas hace que sea necesaria una linealización del mismo. Todo sistema no lineal se puede linealizar dentro de un rango en el cuál se comporte se forma muy parecida a un sistema lineal. En el caso del péndulo invertido ese rango lo establecemos en torno al punto de equilibrio, es decir, el punto en que el robot se encuentra a cero grados con la vertical. Con ello, podemos simplificar enormemente la obtención de un modelo matemático que nos es suficiente para la próxima implementación de los controles en la planta real.

El modelo matemático sobre el que vamos a realizar las simulaciones y vamos a elaborar un controlador va a estar basado en seis variables: los ángulos correspondientes

al cuerpo del robot y de las ruedas, así como de las velocidades y aceleraciones angulares asociados a ellos. $\emptyset$ corresponde al ángulo del cuerpo que se puede obtener mediante la integración de la velocidad angular que devuelve el giróscopo $\dot{\emptyset}$, x es el avance de las rueda, y su valor se puede leer directamente de los encoders que los motores tienen instalados. Derivando este valor obtenemos la velocidad de avance $\dot{x}$. Las segundas derivadas que aparecen en la tabla se corresponden con las aceleraciones que sufren las ruedas y el cuerpo del robot, respectivamente.

2.2.- Análisis de las fuerzas y sistema de ecuaciones

La mayor parte del éxito a la hora de diseñar un buen regulador pasa por tener un modelo del sistema correcto. Hallarlo es una tarea complicada, y por ello, a menudo es necesario recurrir a la sencillez. En el caso del péndulo, se consigue con el análisis por separado de cada uno de los cuerpos.

Ilustración 1.- Cuerpo 0

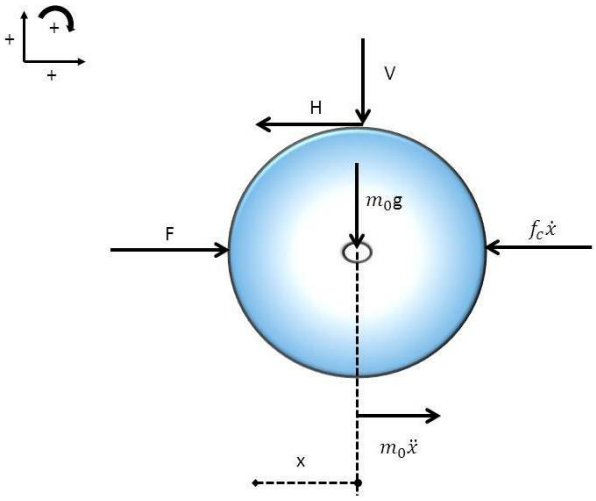
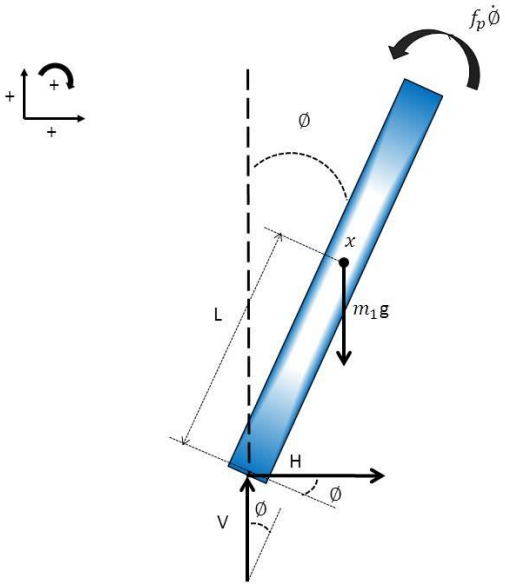


Ilustración 2.- Cuerpo 1



- m_0: Masa del cuerpo 0
- m_1: Masa del cuerpo 1
- f_c: Coeficiente de fricción de las ruedas del carro
- f_p: Coeficiente de fricción rotacional del péndulo
- L: Longitud desde centro de gravedad al extremo del péndulo

- J: Momento de inercia del péndulo en el centroide
- g: Gravedad
- F: Fuerza aplicada al carro
- x: Coordenadas de posición del carro respecto de la posición inicial
- ∅: Angulo del péndulo
- V: Fuerza de reacción vertical
- H: Fuerza de reacción horizontal

El péndulo invertido se puede concebir básicamente como un cuerpo rígido cuyo movimiento se limita a dos dimensiones. Las ecuaciones fundamentales de movimiento plano de un cuerpo rígido son:

Sumatorio de fuerzas en eje vertical igual a masa por aceleración en eje vertical, sumatorio de fuerzas en eje horizontal igual a masa por aceleración en eje horizontal y sumatorio de momentos en un punto igual al momento de inercia por aceleración angular.

$$\sum \boldsymbol{F_i = ma_i}$$

$$\sum \boldsymbol{F_j = ma_j}$$

$$\sum \boldsymbol{F_g = J\ddot{\emptyset}}$$

Sumando las fuerzas en el diagrama de *Cuerpo 1* en la dirección horizontal, se obtiene la siguiente ecuación del movimiento:

$$\mathbf{F - H - f_c\dot{x} = m_0\ddot{x}}$$

También se podrían sumar las fuerzas en la dirección vertical, pero no se ganará ninguna información útil. Por otro lado, teniendo en cuenta lo que se ha desplazado el centroide del *Cuerpo 2* en la dirección horizontal:

$$\boldsymbol{H = m_1 \frac{d^2(x + L \sin \emptyset)}{dt^2}}$$

Haciendo lo mismo pero en el eje vertical:

$$\boldsymbol{V = m_1 g + m_1 \frac{d^2(L - L \cos \emptyset)}{dt^2}}$$

Por último, calculando el sumatorio de momentos en el centroide del péndulo:

$$\boldsymbol{VL \sin \emptyset - HL \cos \emptyset - f_p \dot{\emptyset} = J \ddot{\emptyset}}$$

Aplicando un poco de álgebra, despejando y sustituyendo se obtienen las ecuaciones del movimiento de este sistema:

$$(m_0 + m_1)\ddot{x} + f_c \dot{x} + m_1 L \ddot{\theta} \cos\theta - m_1 L \dot{\theta}^2 \sin\theta = F$$

$$m_1 g L \sin\theta - (J + m_1 L^2)\ddot{\theta} - f_p \dot{\theta} - m_1 L \ddot{x} \cos\theta + m_1 L^2 \sin 2\theta \, \dot{\theta}^2 = 0$$

2.3.- Cálculo del torque generado a partir del valor de potencia dada al motor.

La ecuación de salida lo que busca es equilibrar la fuerza aplicada en la entrada para lo cual habrá que aplicar la fuerza contraria en el extremo opuesto. Para aplicar esa fuerza solo contamos con un valor de potencia que se le pasa a los motores. Este valor va desde -100 a 100. Este es un valor que se da a una variable que controla la velocidad

y sentido de los motores. Realmente el signo solo es el sentido de marcha de los motores, el valor que nos interesa es de 0 a 100.

Con lo cual, si queremos aplicar x Newton al extremo, ¿Qué valor de potencia habrá que darle a los motores?

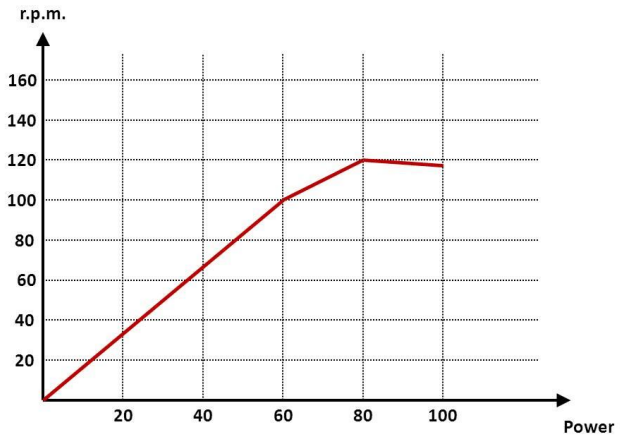
Para averiguar esto se han hecho diversos ensayos y se han obtenido diferentes curvas de funcionamiento. Al combinar todas ellas se puede hallar la relación entre la potencia aplicada a los motores y el par generado y por consiguiente la fuerza F necesaria para paliar a la fuerza F de entrada. Los ensayos se han hecho con la maqueta soportando una carga de 11.5 N*cm.

2.3.1.- Curva r.p.m./ Potencia

De esta curva se obtiene la relación entre la potencia que se le pasa al motor y las revoluciones por minuto que genera con una carga normal.

$$r.p.m. = 1.466\ Power$$

Ilustración 3 – Gráfica r.p.m./potencia

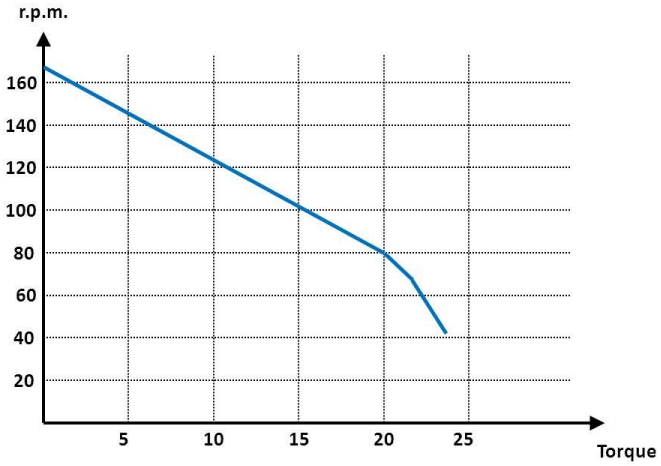


2.3.2.- Curva r.p.m./ Torque

De esta curva se obtiene la relación entre las revoluciones por minuto de las ruedas y el torque generado en el extremo del robot.

$$r.p.m. = -4.25\ Torque + 165$$

Ilustración 4 – Gráfica r.p.m./torque



2.3.3.- Resultado

Con estos datos podemos obtener una relación directa entre la fuerza necesaria para paliar la falta de equilibrio y el valor de potencia que necesitan los motores. Teniendo en cuenta que:

$$\text{Torque} = F*L$$

Se puede ver fácilmente:

$$\text{Power} = \frac{-4.25\ F\ L + 165}{1.466}$$

Donde L es la distancia desde el extremo inferior hasta el centro de gravedad del péndulo.

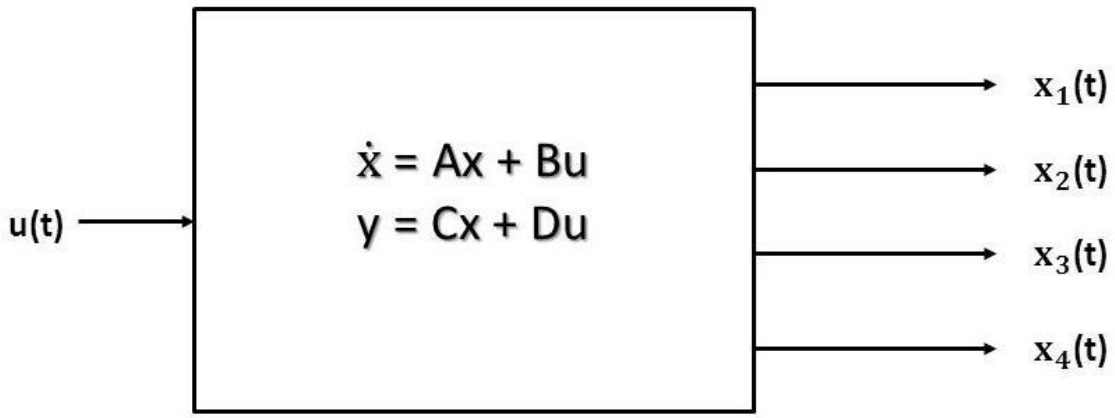
2.4.- Ecuación de estado no lineal

El espacio de estados es un método que permite modelar un sistema físico. Se representa por un conjunto de entradas, salidas y variables de estado relacionadas por ecuaciones diferenciales de primer orden que se combinan en una ecuación diferencial matricial de primer orden. A esta representación se le llama ecuación de estado. Una forma general de expresar la dinámica de un sistema lineal es:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

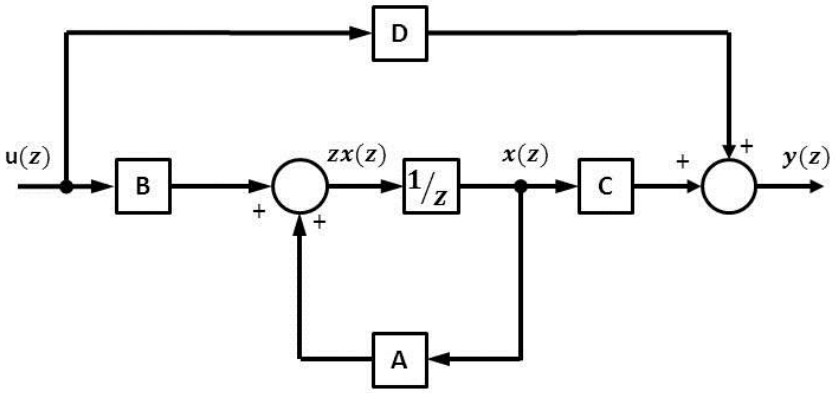
Ilustración 5 - Diagrama de estados



- **x** es el vector de estado del sistema. Contiene *n* elementos
- **u** es el vector de entrada. Contiene *m* elementos
- **y** es el vector de salida. Contiene *p* elementos
- **A** es la matriz de estado de dimensión *n x n*
- **B** es la matriz de entrada de dimensión *n x m*

- **C** es la matriz de salida de dimensión *p x n*
- **D** es la matriz de transmisión directa de dimensión *p x m*

Ilustración 6 - Diagrama de flujo de un sistema de estados



Este tipo de representación tiene la ventaja de que permite conocer el comportamiento interno del sistema, además de que se puede trabajar con sistemas cuyas condiciones iniciales sean diferentes de cero. Otra ventaja es que se facilita el diseño asistido por computadora, ya que los paquetes de software normalmente dependen de esta representación.

- El vector $\boldsymbol{x}$ que determina el estado del sistema contendrá cuatro elementos (posición del carro, primera derivada, posición del ángulo y su derivada).
- El vector $\boldsymbol{y}$ es la salida del sistema. Valoraremos dos salidas, la posición del carro y el ángulo del péndulo.
- El vector $\boldsymbol{u}$ es la entrada del sistema, tiene un único elemento que es la fuerza aplicada al carro, F.

$$x = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\emptyset} \\ \boldsymbol{\dot{x}} \\ \boldsymbol{\dot{\emptyset}} \end{bmatrix}$$

$$y = \begin{bmatrix} \boldsymbol{y_1} \\ \boldsymbol{y_2} \\ \boldsymbol{0} \\ \boldsymbol{0} \end{bmatrix}$$

$$u = F$$

Una vez conocida esta información se pueden calcular las matrices A, B, C, D de cada espacio.

2.5.- Cálculo de las matrices de estado

Para calcular las matrices primero es necesario linealizar el sistema en torno a un punto de equilibrio. Se podría recurrir a la descomposición en series de Taylor pero, por sencillez y dado que el ángulo de inclinación del robot siempre será muy pequeño, se harán unas simplificaciones básicas.

Para linealizar las ecuaciones 10 y 11 alrededor del punto de operación $x_0 = [0\ 0\ 0\ 0]$ se tomarán las siguientes aproximaciones:

$\cos \emptyset \cong 1$, $\sin \emptyset \cong \emptyset$, $\dot{\emptyset}^2 \cong 0$, dando como resultado:

$$(\boldsymbol{m_0} + \boldsymbol{m_1})\ddot{\boldsymbol{x}} + \boldsymbol{f_c}\dot{\boldsymbol{x}} + \boldsymbol{m_1 L}\ddot{\boldsymbol{\theta}} = \mathbf{F}$$

$$\boldsymbol{m_1 g L}\emptyset - (\mathbf{J} + \boldsymbol{m_1 L^2})\ddot{\boldsymbol{\theta}} - \boldsymbol{f_p}\dot{\emptyset} - \boldsymbol{m_1 L}\ddot{\boldsymbol{x}} = \mathbf{0}$$

Sustituyendo las variables de estado, $x_1 = x, x_2 = \emptyset, x_3 = \dot{x_1}, x_4 = \dot{x_2}$

$$(\mathbf{m_0} + \mathbf{m_1})\dot{\mathbf{x_3}} + \mathbf{f_c x_3} + \mathbf{m_1} L\dot{\mathbf{x_4}} = F$$

$$\mathbf{m_1} g L\mathbf{x_2} - (J + \mathbf{m_1 L^2})\dot{\mathbf{x_4}} - \mathbf{f_p x_4} - \mathbf{m_1 L}\dot{\mathbf{x_3}} = 0$$

Calculando $\dot{\boldsymbol{x_4}}$ y $\dot{\boldsymbol{x_3}}$:

$$\dot{x_3} = -\frac{m_1 g L}{A}x_2 - \frac{(J + m_1 L^2)f_c}{A m_1 L}x_3 + \frac{f_p}{A}x_4 + \frac{(J + m_1 L^2)}{A m_1 L}u$$

Donde $A = \frac{(J+m_1 L^2)(m_0+m_1)}{m_1 L} - m_1 L$

$$\dot{x_4} = -\frac{m_1 g L}{B}x_2 - \frac{f_c m_1 L}{(m_0 + m_1)B}x_3 + \frac{f_p}{B}x_4 + \frac{m_1 L}{(m_0 + m_1)B}u$$

Donde $B = \frac{(m_1 L)^2}{m_0+m_1} - (J + m_1 L^2)$

Las matrices de estado quedan asi:

$$\dot{x}(t) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & a_{42} & a_{43} & a_{44} \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ b_3 \\ b_4 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} x(t)$$

donde:

$$a_{32} = -\frac{m_1 g L}{A}$$

$$a_{33} = -\frac{(J + m_1 L^2) f_c}{A m_1 L}$$

$$a_{34} = \frac{f_p}{A}$$

$$a_{42} = -\frac{m_1 g L}{B}$$

$$a_{43} = -\frac{f_c m_1 L}{(m_0 + m_1) B}$$

$$a_{44} = \frac{f_p}{B}$$

$$b_3 = \frac{(J + m_1 L^2)}{A m_1 L}$$

$$b_4 = \frac{m_1 L}{(m_0 + m_1) B}$$

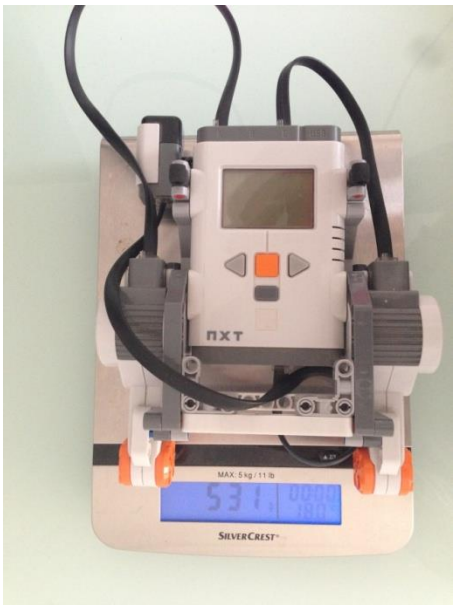
A continuación sustituyendo los valores de nuestro problema

- m_0: Masa del carro = Masa de las ruedas = 0.033 Kg

- m_1: Masa del péndulo = Masa del robot sin ruedas = 0.531 Kg
- f_c: Fricción de las ruedas del carro = 0.05 N*s/m
- f_p: Fricción rotacional del péndulo = 0.0726 $^{Kg * m^2}/_{rad * s}$
- L: Longitud desde centro de gravedad al extremo del péndulo = Medida desde el eje de ruedas hasta el centro de gravedad = 0.085 m
- J: Momento de inercia del péndulo = 0.0011 Kg*m^2
- g: Gravedad = 9.8 $^{m}/_{s^2}$
- F: Fuerza aplicada al carro
- x: Coordenadas de posición del carro
- ∅: Angulo del péndulo

Ilustración 7 - Peso de las ruedas

Ilustración 8 - Peso del robot sin ruedas



Los coeficientes de las matrices quedan así:

$$a_{32} = -26.7257$$

$$a_{33} = -0.3304$$

$$a_{34} = 4.3865$$

$$a_{42} = 333.9611$$

$$a_{43} = 3.0210$$

$$a_{44} = -54.8141$$

$$b_3 = 6.6083$$

$$b_4 = -60.4213$$

y la representación en espacio de estados queda:

$$\dot{x}(t) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -26.7257 & -0.3304 & 4.3865 \\ 0 & 333.9611 & 3.0210 & -54.8141 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ 6.6083 \\ -60.4213 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} x(t)$$

2.6.- Respuesta en lazo abierto

Una vez obtenidos los modelos matemáticos y antes de comenzar con el diseño de un regulador es necesario comprobar si el sistema es estable a lazo abierto. Para ello, se hará una simulación con Matlab. La entrada será un impulso unitario, lo que sería un "empujoncito" al robot.

Cálculos en Matlab:

```
>> A = [0 0 1 0; 0 0 0 1; 0 -26.7257 -0.3304 4.3865; 0 333.9611 3.0210
-54.8141]
B =[0; 0; 6.6083; -60.4213]
C = [0 1 0 0; 0 0 1 0]
D = [0; 0]
[num,den] = ss2tf(A,B,C,D)
t=0:0.01:5;
impulse(num,den,t)
axis([0 3 -60 60])
```

Ilustración 9 - Respuesta en lazo abierto de la salida Ø

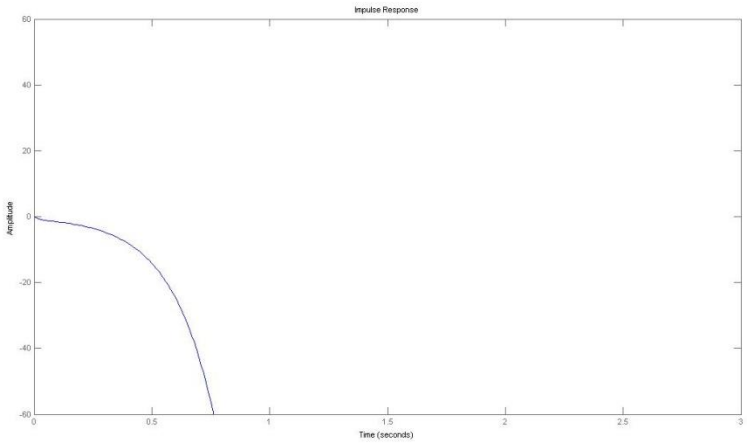
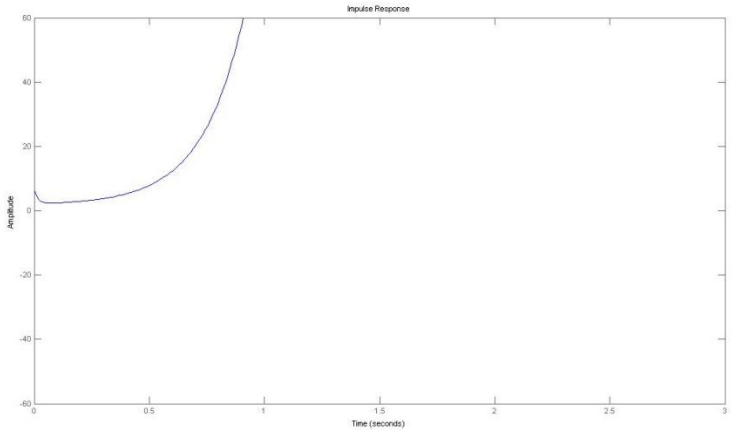


Ilustración 10 - Respuesta en lazo abierto de la salida $\dot{X}$



La salida (1) es el ángulo de inclinación del robot y la salida (2) es la velocidad del robot. Como se puede ver en los gráficos, la respuesta es insatisfactoria. El sistema es inestable. Es obvio que para mejorar la dinámica del sistema alguna especie de control tendrá que ser diseñada. El resultado es una inestabilidad en poco tiempo. Es un resultado muy malo. Necesitamos que alcance el equilibrio y con estabilidad.

Una vez conocido el comportamiento del sistema en ausencia de ningún tipo de controlador, debemos establecer el método que nos permita mantener al robot en equilibrio. Existen diversos métodos de control para resolver el problema de estabilidad del robot, entre ellos, modelos lineales, no lineales, predictivos, etc.

2.7.- Diseño del controlador

2.7.1.- Controlador LQR

Para el modelo del sistema expresado en el espacio de estado, la técnica LQR calcula la matriz de control [K(t)] tal que, realizando un control con realimentación de estado expresada mediante la ecuación $[u] = -[K(t)]\cdot[x]$, se minimiza la función de coste *J* en el intervalo de tiempo considerado (intervalo de optimización).

$$J = \int_0^T (x^T Q x + u^T R u)dt + x(T)^T M x(T)$$

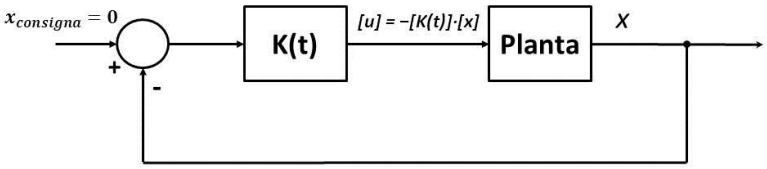
Los términos que componen la función de coste se interpretan de la siguiente manera:

- $x^T Q x$ es una medida de la desviación de los estados respecto los estados deseados
- $u^T R u$ es una medida del esfuerzo del control
- $x(T)^T M x(T)$ es una medida de la desviación de los estados respectolos estados deseados en el instante final del intervalo de optimización

Considerando que los estados [*x*] de la ecuación en el espacio de estado representan estados incrementales, el objetivo del control LQR consiste en llevar a los estados [*x*] lo más cerca posible de los estados de consigna o referencia $x_{consigna} = 0$.

El diseñador del control LQR determina las matrices [Q], [R] y [M], cuyos elementos deben ser positivos o cero.

Ilustración 11 - Diagrama de bloques LQR



En otros términos, la técnica LQR ofrece un regulador tal que se controla el sistema de forma óptima a partir de la minimización de la función de coste J previamente definida. En consecuencia, el control es óptimo para la función de coste definida, depende de la habilidad del diseñador establecer una función de coste J que se ajuste lo más fielmente posible a las necesidades de control del sistema. En general, ello supone que es necesario tener, en mayor o menor medida, un conocimiento de la planta.

Si se considera que el intervalo de optimización es infinito, el tiempo T es infinito. Bajo este entorno, la matriz resultante de la ecuación de Riccati es constante y, por tanto, la matriz de control $[K]$ también es constante. El controlador óptimo es invariante frente al tiempo en régimen permanente.

$$[u(t)] = -[K]\cdot[x(t)]$$

La solución de la ecuación de Riccati en régimen permanente es independiente de la matriz [*M*]. En consecuencia, si se desea calcular la matriz [*K*] de control para régimen permanente, se puede reducir la expresión de la función de coste J a la ecuación (5.8), al poder prescindir del término $x(T)^T Mx(T)$. En este caso, sólo es necesario definir las matrices [*Q*] y [*R*].

$$J = \int_0^T (x^T Qx + u^T Ru)dt$$

El cálculo de la matriz de control [K], a partir del modelo en el espacio de estados y una vez definidas las matrices [Q] y [R], es muy simple si se emplean herramientas informáticas, como por ejemplo, MATLAB®. Los pasos son los siguientes:

- **Ver si se cumplen las condiciones para la existencia de solución en régimen permanente:**

Si el sistema en el espacio es controlable y observable, existe una única matriz de control óptimo [*K*] tal que el sistema en lazo cerrado con el control $[u] = -[K]\cdot[x]$ es asintóticamente estable. En otros términos, es condición suficiente que el sistema sea controlable y observable para garantizar la solución de la ecuación de Riccati en régimen permanente y la existencia de una matriz de control óptimo.

Para ver si es controlable hay que comprobar que el rango de la matriz O es igual al rango del sistema.

$$O = \begin{bmatrix} C \\ CA \\ CA^2 \\ . \\ . \\ . \\ CA^{n-1} \end{bmatrix}$$

En nuestro caso la matriz O resulta:

$$O = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -3.6171 & -0.0117 & -0.0569 \\ 0 & -190.4057 & -1.7224 & -2.9963 \\ 0 & 10.8764 & 0.0981 & -3.4459 \\ 0 & 576.7427 & 5.1810 & -181.3299 \end{bmatrix}$$

El rango de la matriz se determina por Gauss y es igual a 4, tal y como el rango del sistema. Por lo tanto el *sistema es controlable*.

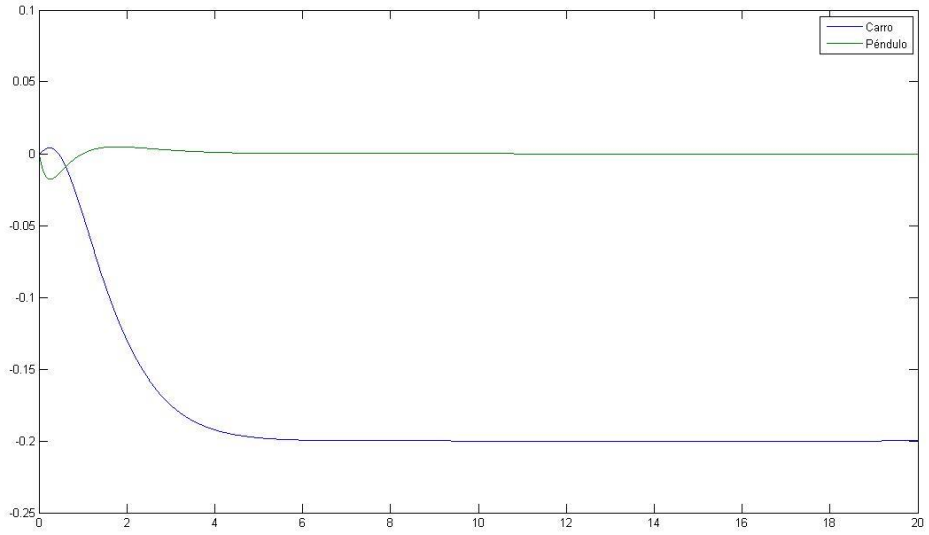
Por otra parte, todas las variables de estado son fácilmente medibles y se dispone de la información sobre todas ellas. En consecuencia, el sistema es observable y no es necesario emplear ningún criterio para verificar la observabilidad del sistema.

- **A continuación elegir unos valores iniciales para x e y. Por ejemplo comenzar con x=1 e y=1**

```
>> x=1;
     y=1;
     Q=[x 0 0 0; 0 0 0 0; 0 0 y 0; 0 0 0 0];
     R = 1;
     K = lqr(A,B,Q,R)
     Ac = [(A-B*K)];
     Bc = [B];
     Cc = [C];
     Dc = [D];
     T=0:0.01:20;
     U=0.2*ones(size(T));
     [Y,X]=lsim(Ac,Bc,Cc,Dc,U,T);
     plot(T,Y)
     legend('Carro','Péndulo')
```

En la figura generada debería verse lo siguiente:

Ilustración 12 - Respuesta con primer LQR



Con el siguiente valor de K:

```
>> K =  -1.0000  -17.0950   -1.8839   -0.4366
```

La línea azul corresponde al desplazamiento del carro y la línea verde al ángulo del cuerpo del robot.

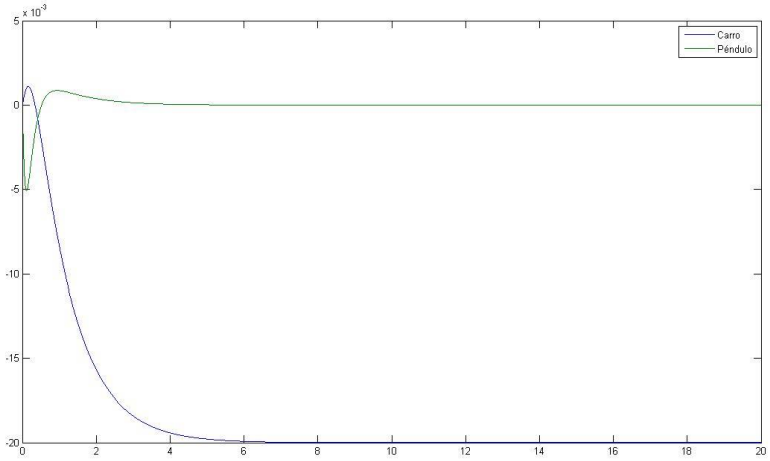
- **A continuación fijarse en la gráfica obtenida para ver el tiempo de subida y de establecimiento así como la sobreoscilación. Comparar con los límites preestablecidos. Si no es satisfactorio probar nuevos valores.**

El tiempo de establecimiento no es el pedido. Probar a aumentar x:

```
>> x=100;
     y=100;
     Q=[x 0 0 0; 0 0 0 0; 0 0 y 0; 0 0 0 0];
```

```
R = 1;
K = lqr(A,B,Q,R)
Ac = [(A-B*K)];
Bc = [B];
Cc = [C];
Dc = [D];
T=0:0.01:20;
U=0.2*ones(size(T));
[Y,X]=lsim(Ac,Bc,Cc,Dc,U,T);
plot(T,Y)
legend('Carro','Péndulo')
```

Ilustración 13 - Respuesta con segundo LQR



El tiempo de establecimiento ha mejorado mucho y la sobreoscilación se ha mantenido e incluso mejorado.

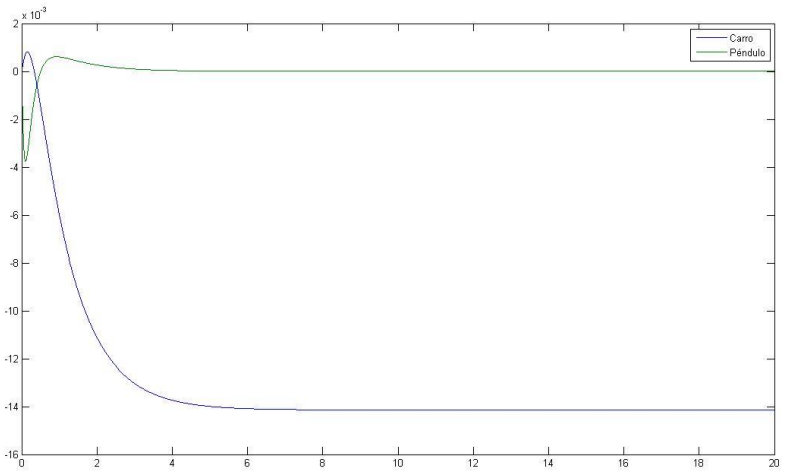
- **Si los límites no se cumplen entonces ir probando valores de x e y hasta conseguir los objetivos de diseño.**

Vamos a seguir probando valores de x e y. x=200, y=200

```
>>x=200;
```

```
y=200;
Q=[x 0 0 0; 0 0 0 0; 0 0 y 0; 0 0 0 0];
R = 1;
K = lqr(A,B,Q,R)
Ac = [(A-B*K)];
Bc = [B];
Cc = [C];
Dc = [D];
T=0:0.01:20;
U=0.2*ones(size(T));
[Y,X]=lsim(Ac,Bc,Cc,Dc,U,T);
plot(T,Y)
legend('Carro','Péndulo')
```

Ilustración 14 - Respuesta con tercer LQR



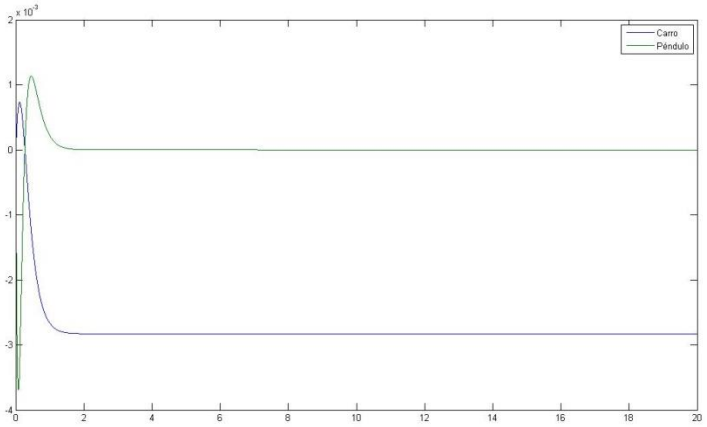
Se ha mejorado más aún el funcionamiento pero todavía no cumplimos el valor deseado de tiempo de establecimiento. Así que seguiremos probando valores.

- **Probar nuevos valores. X=5000, y=100:**

```
>>x=5000;
y=100;
Q=[x 0 0 0; 0 0 0 0; 0 0 y 0; 0 0 0 0];
```

```
R = 1;
K = lqr(A,B,Q,R)
Ac = [(A-B*K)];
Bc = [B];
Cc = [C];
Dc = [D];
T=0:0.01:20;
U=0.2*ones(size(T));
[Y,X]=lsim(Ac,Bc,Cc,Dc,U,T);
plot(T,Y)
legend('Carro','Péndulo')
```

Ilustración 15 - Respuesta con cuarto LQR



En este caso sí se cumplen las especificaciones dadas. Por lo tanto el valor de K obtenido es:

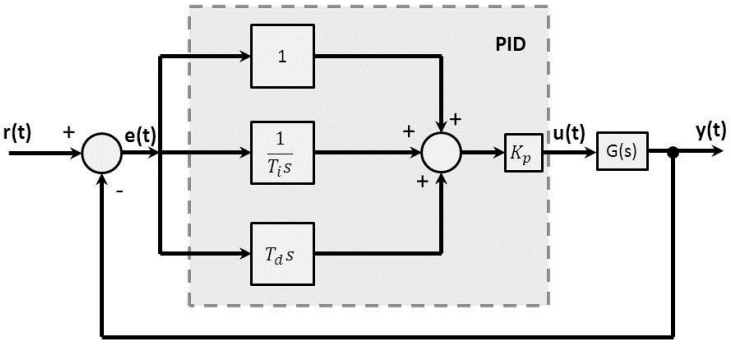
```
>> K = -70.7107  -122.0560  -42.2596  -5.5969
```

No obstante, por su sencillez y facilidad de integrar en el sistema construido usaremos un controlador PID.

2.7.2.- Controlador PID:

Debido al carácter práctico de nuestro proyecto de laboratorio hemos optado, dada su sencillez, por el modelo PID considerando el sistema lineal, para un rango acotado de funcionamiento, dentro del que nos ocuparemos de mantener al sistema funcionando. En la figura se muestra un ejemplo típico de controlador PID. La señal r(t) es el valor de referencia que deseamos mantener, en el caso del robot, es el valor del ángulo de equilibrio. y(t) es el valor del ángulo real obtenido a partir de la información proporcionada por el giróscopo. e(t) es la señal de error que actúa como entrada en el bloque PID. u(t) es el valor de la señal de control, en nuestro caso, la potencia aplicada a los motores para compensar las inclinaciones del propio sistema. Finalmente, G(s) es la función de transferencia del sistema que obtuvimos en la sección anterior (es muy sencillo obtener la función de transferencia a partir de las matrices de estado).

Ilustración 16 - esquema PID



Estableciendo adecuadamente el valor de las constantes que definen el PID, podemos obtener una respuesta del sistema estable aún en presencia de una excitación. El error acaba valiendo cero a lo largo del tiempo.

Ahora se está trabajando con sistemas en lazo cerrado, es decir, con realimentación. Donde restando el valor de la referencia a la salida anterior se obtiene el error que el PID irá tratando hasta que sea nulo. Los miembros de la familia de controladores PID, incluyen tres acciones: proporcional (P), integral (I) y derivativa (D). Estos controladores son los denominados P, I, PI, PD y PID.

- P: Acción de control proporcional da una salida del controlador que es proporcional al error. Un controlador proporcional puede controlar cualquier planta estable, pero posee desempeño limitado y error en régimen permanente.

$$u(t) = KP\ e(t).$$

- I: Acción de control integral da una salida del controlador que es proporcional al error acumulado, lo que implica que es un modo de controlar lento.

$$u(t) = K_i \int_0^t e(\tau)d\tau$$

- PI: Acción de control proporcional-integral se define mediante

$$\boldsymbol{u(t) = K_p e(t) + \frac{K_p}{T_i}\int_0^t e(\tau)d\tau}$$

donde Ti se denomina tiempo integral y es quien ajusta la acción integral. Con un control proporcional, es necesario que exista error para tener una acción de control distinta de cero. Con acción integral, un error pequeño positivo siempre nos dará una acción de control creciente, y si fuera negativo la señal de control será decreciente. Este razonamiento sencillo muestra que el error en régimen permanente será siempre cero.

- PD: Acción de control proporcional-derivativa se define mediante:

$$\boldsymbol{u(t) = K_p e(t) + K_p T_d \frac{de(t)}{dt}}$$

donde Td es una constante denominada tiempo derivativo. Esta acción tiene carácter de previsión, lo que hace más rápida la acción de control, aunque tiene la desventaja importante que amplifica las señales de ruido y puede provocar saturación en el actuador.

La acción de control derivativa nunca se utiliza por sí sola, debido a que sólo es eficaz durante períodos transitorios.

- PID: Acción de control proporcional-integral-derivativa. Esta acción combinada reune las ventajas de cada una de las tres acciones de control individuales. La ecuación de un controlador con esta acción combinada se obtiene mediante:

$$u(t) = K_p e(t) + \frac{K_p}{T_d}\int_0^t e(\tau)d\tau + K_p T_d \frac{de(t)}{dt}$$

La función del controlador es:

$$\mathbf{C_{PID}(s) = k_p(1 + \frac{1}{T_i s} + T_d s)}$$

2.7.2.- Diseño del controlador

El objetivo de este apartado es calcular la función de transferencia que represente el regulador óptimo.

La decisión de qué controlador utilizar es algo libre. Por supuesto, lo importante es usar uno que consiga los objetivos para un equilibrio óptimo y dentro de un rango de restricciones que le daremos. Estas restricciones no son mas que las adecuadas para que el robot tenga un rápido equilibrio y muy estable. Estas pueden ser:

- Tiempo de subida menor de 1 segundo
- 5% de Sobreoscilación

La elección de cual controlador elegir es algo subjetivo siempre partiendo de datos globales que tenemos de cada controlador.

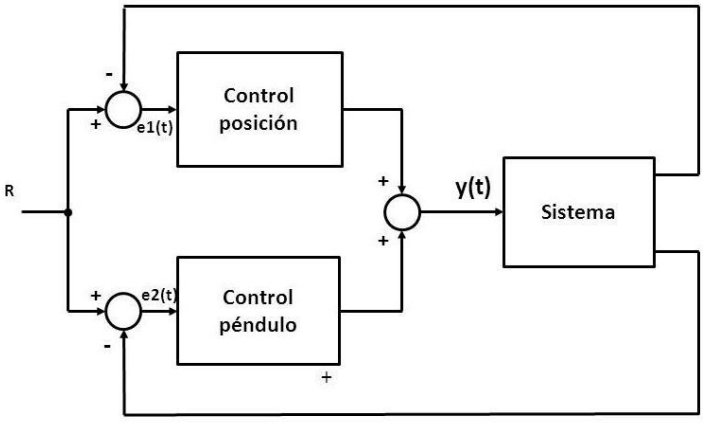
En la siguiente tabla se muestra la reacción de cada controlador respecto a cambios en los parámetros.

Tabla 1 - Variaciones PID

	Tiempo de subida	Saturación	Tiempo de estabilización	Error en estado estacionario	Estabilidad
Incrementa Kp	Decremento	Incremento	Pequeño incremento	Decremento	Reducción
Incrementa Ki o Ti	Pequeño decremento	Incremento	Incremento	Gran decremento	Reducción
Incrementa Kd o Td	Pequeño decremento	Decremento	Decremento	Cambios menores	Mejora

Según esta tabla y nuestro sistema hemos elegido usar un doble controlador PID de forma que el diagrama de bloques sería:

Ilustración 17 - Diagrama de bloques del sistema



La ecuación característica del bucle cerrado viene dada por:

$$y(t) = K_{p1}e_1(t) + k_{i1}\int_0^t e_1(\tau)d\tau + K_{p2}e_2(t) + k_{i2}\int_0^t e_2(\tau)d\tau$$

donde y(t) es la salida y u(t) es la entrada, en nuestro caso, el impulso N.

Para diseñar nuestros PID hay que respetar los límites que queremos conseguir.

- Tiempo de subida menor de 1 segundo
- 5% de Sobreoscilación

Para el diseño del primer PID recurriremos a la herramienta **sisotool** de MATLAB. Esta herramienta es SISO (Single Input Single Output), y nuestro sistema de estados tiene una entrada y dos salidas así que diseñaremos un PID para cada función de transferencia. Para separar las dos funciones de transferencia recurriremos a Matlab con la sentencia:

```
>> [num,den] = ss2tf(A,B,C,D)
```

El resultado es un denominador y dos numeradores distintos. Una función de transferencia es uno de los numeradores con el denominador y la otra función de transferencia es el otro numerador con el mismo denominador. Queda asi:

```
>> num =
         0         0  -60.4213    0.0005    0.0000
         0    6.6083   97.1900 -592.1136   -0.0000
>> den =
    1.0000   55.1445 -329.1021  -29.6024         0
```

2.7.2.1.- Diseño PID para salida x

Se define el sistema en el espacio de trabajo de MATLAB®

```
A = [0 0 1 0; 0 0 0 1; 0 -26.7257 -0.3304 4.3865; 0 333.9611 3.0210 -
54.8141];
B =[0; 0; 6.6083; -60.4213];
```

```
C = [1 0 0 0];
D = [0];
[num,den] = ss2tf(A,B,C,D);
G=tf(num,den);
```

Invocamos a la aplicación sisotool

```
sisotool(G)
```

Diseñamos un controlador mediante la introducción de un integrador y un cero real para anular el efecto del polo ubicado en el semiplano real derecho.

El controlador obtenido es:

$$Q = \frac{17632 \cdot (s^2 - 9.877 \cdot s + 24.39)}{s}$$

$$= 430044.48 \left[\frac{1-0.4049s+0.0410s^2}{s}\right]$$

$$= 430044.48 \left[-0.4049 + 0.041s + \frac{1}{s}\right]$$

$$= -174125.01 \left[1 - 0.1012s - \frac{1}{0.4049s}\right]$$

Donde:

$$K_c = -174125.01$$

$$T_d = -0.1012$$

$$T_i = -0.4049$$

Para nuestra programación nos interesa dejarlo de la forma:

$$C(s) = K_p error + K_i \sum error_{anterior} + K_d(error - error_{anterior})$$

$$K_p = -174125.01$$

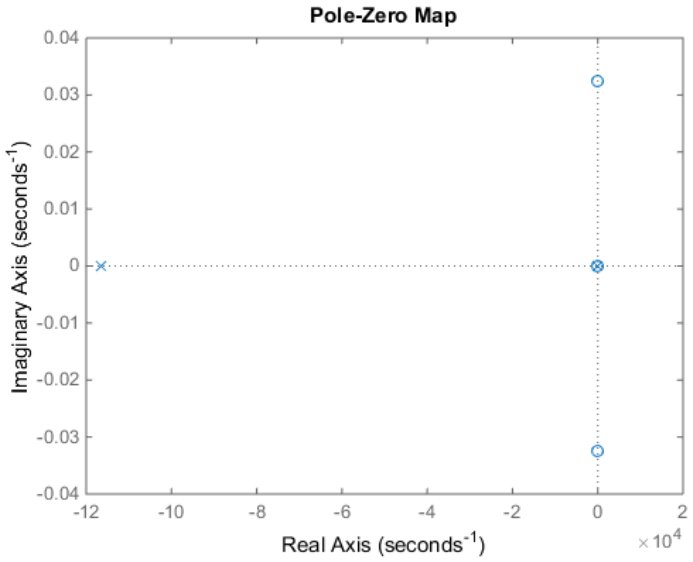
$$K_d = 17621.45$$

$$K_i = 430044.70$$

Al realimentar el sistema, no se observan polos en el semiplano real positivo:

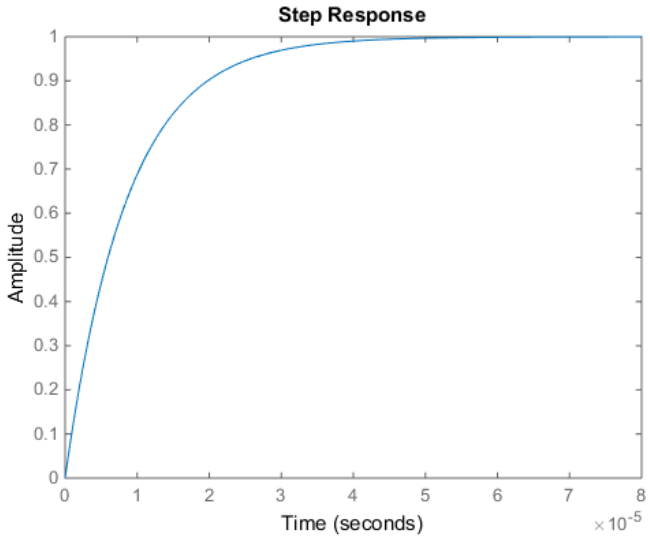
```
Gc=feedback(Q*G,1);
pzmap(Gc)
```

Ilustración 18 - Lugar raíces Q1



La salida ante una entrada escalón es la siguiente:

Ilustración 19 - Respuesta escalón Q1



Como se puede observar la salida es estable.

El valor final de la salida ante una entrada escalón se puede calcular con la función siguiente:

```
dcgain(Gc)
```

Que da como resultado 1.

También se podría haber obtenido con el uso del teorema del valor final:

$$\lim_{t\to\infty} y(t) = \lim_{s\to 0} s \cdot Y(s) = \lim_{s\to 0} s \cdot G_c \cdot \frac{1}{s} = 1$$

2.7.2.2.- Diseño PID para salida Ø

Se define el sistema en el espacio de trabajo de MATLAB para la función de transferencia de entrada Ø

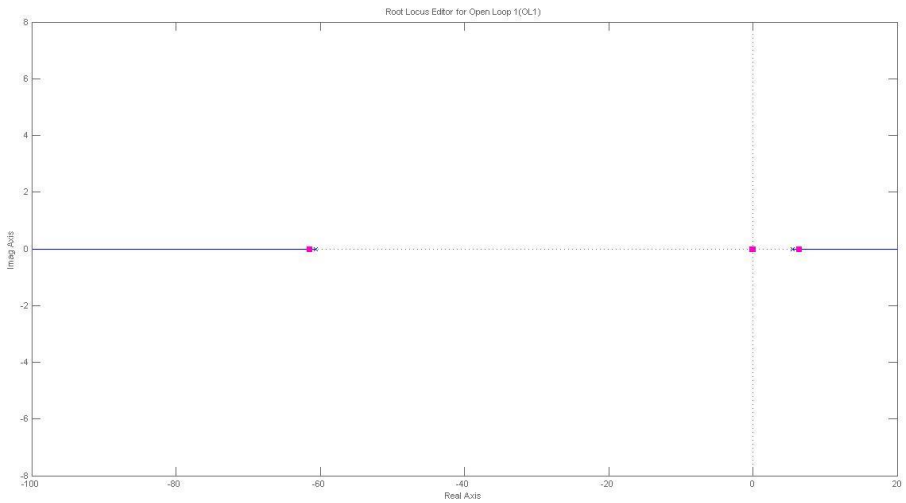
```
A = [0 0 1 0; 0 0 0 1; 0 -26.7257 -0.3304 4.3865; 0 333.9611 3.0210 -
54.8141];
B =[0; 0; 6.6083; -60.4213];
C = [0 1 0 0];
D = [0];
[num,den] = ss2tf(A,B,C,D);
G=tf(num,den);
```

Invocamos a la aplicación sisotool

```
sisotool(G)
```

El lugar de las raíces desvela un polo en el semiplano positivo, por lo tanto, inestabilidad en el sistema.

Ilustración 20 - Lugar raíces G2



Diseñamos un controlador mediante la introducción de un par de ceros reales para anular el efecto del polo ubicado en el semiplano real derecho.

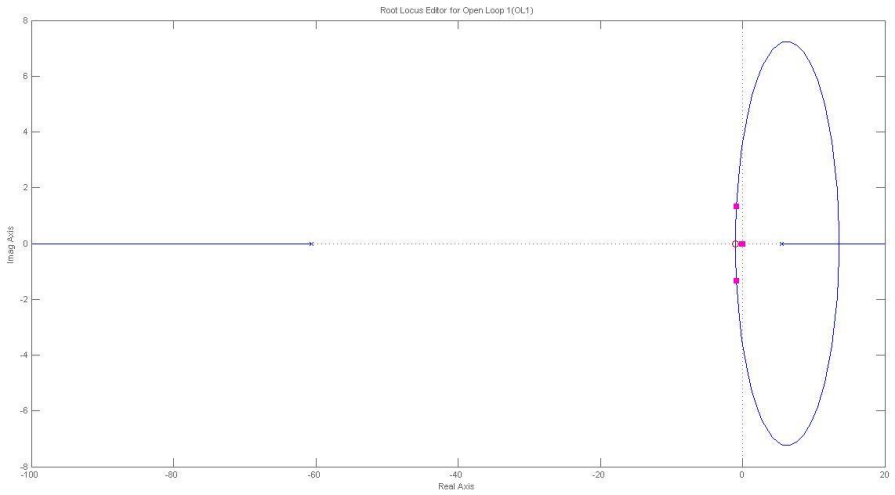
El controlador obtenido es

$$Q = 3.5148 \cdot (1 + s)(1 + s)$$

Al realimentar el sistema, no se observan polos en el semiplano real positivo:

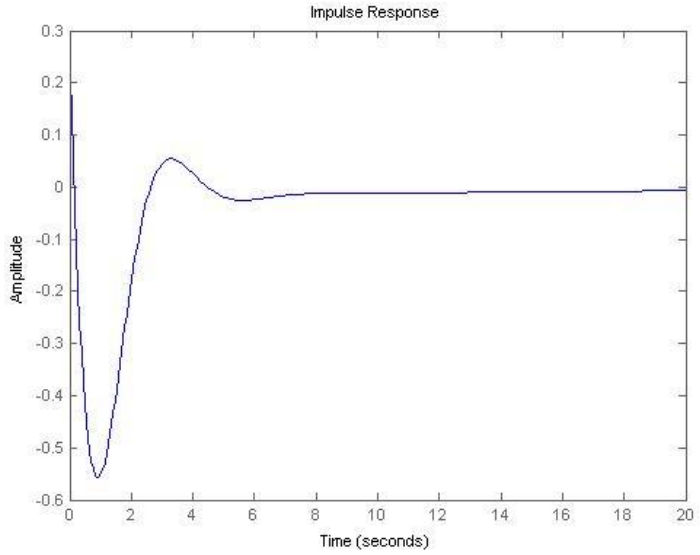
```
Gc=feedback(Q*G,1);
Pzmap(Gc)
```

Ilustración 21 - Lugar raíces Q2



La salida ante una entrada escalón es la siguiente:

Ilustración 22 - Respuesta impulso G2



Como se puede observar la salida es estable.

No obstante, intentaremos cumplir las condiciones de diseño, que son:

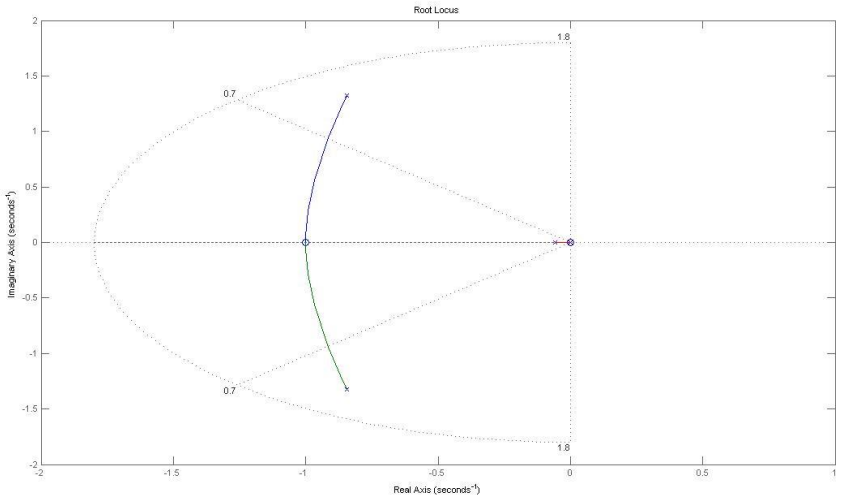
- Tiempo de subida menor de 1 segundo
- 5% de Sobreoscilación

Sobreoscilación menor que 5% significa una razón de amortiguación **Zeta** mayor que 0.7 y un tiempo de subida de 1 segundo significa una frecuencia natural **Wn** mayor que 1.8. Introducimos estos datos en Matlab para que cree unos límites en nuestro lugar de las raíces. Si logramos colocar los polos dentro de estos límites podremos cumplir las condiciones de diseño.

```
zeta=0.7;
Wn=1.8;
sgrid(zeta, Wn)
```

El resultado es el siguiente:

Ilustración 23 - Restricción lugar raíces



En la figura se ve que es imposible colocar los polos dentro de las líneas puntuadas diagonales y a la vez fuera del círculo punteado. Esto significa que las condiciones de diseño no se pueden cumplir con este controlador aunque el sistema sea estable. La solución es conformarse con un tiempo de subida de 3 segundos y una sobreoscilación de 6%.

2.8.- Código en nxc

El lenguaje que elegimos para la codificación del programa es NXC. Es muy parecido a C y a continuación se explican las distintas funciones en pseudocódigo.

2.8.1.- Función toma de offset del giróscopo

El offset de un aparato es la diferencia entre el valor que mide y el que debería haber tomado. Para verlo basta con dejar el giróscopo en reposo y el valor que debe devolver es cero ya que la velocidad es cero.

Esta función calcula el offset del giróscopo. Toma 100 muestras en medio segundo y hace una media aritmética. A continuación comprueba el máximo y el mínimo de las muestras. Si supera ciertos valores entonces empieza otra vez y toma otras muestras ya que entiende que las muestras no son válidas por no haber estabilizado bien el robot.

Los valores de Lectura_Maxima y Lectura_Minima van acotando cada vez mas la desviación a la hora de tomar muestras. Toma OFFSET_SAMPLES muestras en total. Una cada 5 milisegundos.

```
void Offset_giroscopio()
{
float Suma_Lecturas;
int  i, Lectura_Minima, Lectura_Maxima, Lectura;
ClearScreen();
TextOut(14, 56, "PROYECTO J&J");
TextOut(8, 16, "--Calibrando--");
Off(Ambos_Motores);
PlayTone(420,1000);
do
    {
    Suma_Lecturas = 0.0;
    Lectura_Minima = 1000;
    Lectura_Maxima = -1000;
    for (i=0; i<OFFSET_SAMPLES; i++)
        {
        Lectura = SensorHTGyro(Puerto_Giroscopio);
        if (Lectura > Lectura_Maxima)
           {
           Lectura_Maxima = Lectura;
           }
```

```
        if (Lectura < Lectura_Minima)
           {
           Lectura_Minima = Lectura;
           }
        Suma_Lecturas += Lectura;
        Wait(5);
        }
    }
    while ((Lectura_Maxima - Lectura_Minima) > 1);
    Offset_Giroscopio = Suma_Lecturas / OFFSET_SAMPLES +1;
    PlaySound(SOUND_UP);
    Wait(1000);
    NumOut(0,0,Offset_Giroscopio);
}
```

2.8.2.- Función Lee_giroscopio

La inercia giroscópica es la cualidad por la cual un giróscopo (o todo cuerpo de revolución rotando a alta velocidad) mantiene en todo momento fija su posición en el espacio. Esta es la cualidad que se usa para giros y horizontes, ya que lo que se busca es que al moverse el robot, el instrumento no se mueva y de esta forma se pueda conocer la situación del robot respecto al horizonte.

La deriva de giróscopo o deriva giroscópica es la variación de esa posición fija en el espacio de la cual estamos hablando, y se debe a dos problemas principalmente. Primero, que la posición del giróscopo es fija en el espacio absoluto, y para el horizonte artificial se necesita que sea fija pero siempre paralela a la tierra. Por ejemplo, un giróscopo paralelo a la tierra en el polo, al llegar al ecuador se vería totalmente acostado.

El otro problema de deriva es mecánico, ya que si el cuerpo de revolución no es perfecto (que nunca lo es) pueden aparecer roces, desbalanceos pequeños, etc, que a lo largo del tiempo, en uso irán haciendo que el giróscopo empiece a marcar de forma

incorrecta.

Para paliar un poco este efecto Deriva se recurre a una aproximación. Se toma el 99,95% del valor obtenido del offset en reposo y un 0,05% del valor que se obtenga del giróscopo esta vez en movimiento. Mientras mas movimiento detecte el giróscopo mayor será la influencia del efecto Deriva.

Esta función lee el giroscopio y le corrige el offset obtenido anteriormente introduciendo una corrección extra para el efecto Deriva.

El 99.95% del offset calculado antes + el 0.05% del valor de desviación del cero. Es lo que se llama la media móvil exponencial. Corrige el valor leído del giróscopo más rápidamente mientras más alejado del equilibrio esté el robot.

```
inline void Lee_giroscopio(float &Velocidad_Giroscopio, float &Angulo_Giroscopio)
{
float Lectura_Giroscopio;
Lectura_Giroscopio = SensorHTGyro(Puerto_Giroscopio);
Offset_Giroscopio = Deriva * Lectura_Giroscopio + (1-Deriva) * Offset_Giroscopio;
Velocidad_Giroscopio = Lectura_Giroscopio - Offset_Giroscopio;
Angulo_Giroscopio += Velocidad_Giroscopio*Intervalo;
}
```

2.8.3.- Función Lectura_Motores

El objetivo de Lectura_Motores es contar las vueltas que dan los motores. El objetivo de la función es actualizar la variable "Velocidad_Motores" que es la suma de la velocidad de ambos motores. Suma los datos de los encoders de los dos motores durante las 4 últimas llamadas a la función y lo divide entre cuatro. O lo que es lo mismo, hace la media de las 4 últimas tomas de datos. Al dividirlo por el tiempo del intervalo se obtiene la velocidad media de los motores de los 4 últimos ciclos.

Lo interesante es conocer la velocidad de los motores y el avance de los motores ya que estos son los dos valores que nos valdrán a la hora de usar un controlador.

```
inline void Lectura_Motores(float &Velocidad_Motores, float &Avance_Motores)
{
long Cuenta_Vueltas_Motor_Izquierdo, Cuenta_Vueltas_Motor_Derecho,
Cuenta_Vueltas_Motores_D;

Cuenta_Vueltas_Motor_Izquierdo = MotorRotationCount(Motor_Izquierdo);
Cuenta_Vueltas_Motor_Derecho = MotorRotationCount(Motor_Derecho);
Cuenta_Vueltas_Motores_Suma_Previa = Cuenta_Vueltas_Motores_Suma;
Cuenta_Vueltas_Motores_Suma = Cuenta_Vueltas_Motor_Izquierdo +
Cuenta_Vueltas_Motor_Derecho;
Cuenta_Vueltas_Motores_Resta = Cuenta_Vueltas_Motor_Izquierdo -
Cuenta_Vueltas_Motor_Derecho;
Cuenta_Vueltas_Motores_D = Cuenta_Vueltas_Motores_Suma -
Cuenta_Vueltas_Motores_Suma_Previa;
Avance_Motores += Cuenta_Vueltas_Motores_D;
Velocidad_Motores = Cuenta_Vueltas_Motores_D + Cuenta_Vueltas_Motores_DP1 +
Cuenta_Vueltas_Motores_DP2 + Cuenta_Vueltas_Motores_DP3)/(4*Intervalo);
Cuenta_Vueltas_Motores_DP3 = Cuenta_Vueltas_Motores_DP2;
Cuenta_Vueltas_Motores_DP2 = Cuenta_Vueltas_Motores_DP1;
Cuenta_Vueltas_Motores_DP1 = Cuenta_Vueltas_Motores_D;
}
```

2.8.4.- Función ControlDireccion

Esta función determina el valor que se le pasará a los motores basado en el equilibrio y el control de dirección. Asimismo limita esos valores ya que a los motores el valor absoluto máximo que se les puede pasar es 100.

```
inline void ControlDireccion(int Fuerza, int &Fuerza_Izq, int &Fuerza_Der)
{
int Fuerza_Girar;
 motorDiffTarget += motorControlSteer * Intervalo;
```

```
Fuerza_Girar = K_Direccion * (motorDiffTarget-Cuenta_Vueltas_Motores_Resta);
Fuerza_Izq = Fuerza + Fuerza_Girar;
Fuerza_Der = Fuerza - Fuerza_Girar;
if (Fuerza_Izq > 100)    Fuerza_Izq = 100;
if (Fuerza_Izq < -100)   Fuerza_Izq = -100;
if (Fuerza_Der > 100)   Fuerza_Der = 100;
if (Fuerza_Der < -100) Fuerza_Der = -100;
}
```

2.8.5.- Función Calculo_Intervalo

Calcula el tiempo de cada iteración del loop. Es necesario conocer el tiempo que se tarda en ejecutar una vuelta del código. Esto es útil para conocer la variable tiempo ya que para integrar o derivar necesitamos multiplicar o dividir por ella.

```
inline void Calculo_Intervalo(long Numero_Loops)
{
if (Numero_Loops == 0)
    {
    Intervalo = 0.0055;
    Comienzo = CurrentTick();
    }
else
    {
    Intervalo = (CurrentTick() - Comienzo)/(Numero_Loops*1000.0);
    }
}
```

2.8.6.- Función equilibrio

Es la función principal del equilibrio, la responsable de evitar la caída del robot. Las variables principales son:

- *Angulo_Giroscopio:* Es el ángulo de inclinación del robot. Se expresa en grados.
- *Velocidad_Giroscopio:* Valor del giróscopo después de restarle el offset y el efecto Deriva. Se expresa en grados/segundo.
- *Avance_Motores:* Es la posición de los motores. Se expresa en grados. Es la suma de los dos motores.
- *Velocidad_Motores:* Velocidad de las ruedas según los encoders de los motores. Se expresa en grados/segundo.

```
task equilibrio()
{
Follows(main);
float Velocidad_Giroscopio, Angulo_Giroscopio;
float Velocidad_Motores;
int Fuerza, Fuerza_Izq, Fuerza_Der;
long PosicionCorrecta;
long Numero_Loops = 0;

PosicionCorrecta = CurrentTick();
ResetRotationCount(Motor_Izquierdo);
ResetRotationCount(Motor_Derecho);
while(1)
        {
        Calculo_Intervalo(Numero_Loops++);
        Lee_giroscopio(Velocidad_Giroscopio, Angulo_Giroscopio);
        Lectura_Motores(Velocidad_Motores, Avance_Motores);
        Avance_Motores -= motorControlDrive * Intervalo;
        Fuerza= Ki_Motores*Avance_Motores + Kp_Motores*Velocidad_Motores +
Kp_Giro*Velocidad_Giroscopio + Ki_Giro*Angulo_Giroscopio + KDRIVE*motorControlDrive;

        if (abs(Fuerza) < 100)
             {
             PosicionCorrecta = CurrentTick();
             }
        ControlDireccion(Fuerza, Fuerza_Izq, Fuerza_Der);
        OnFwd(Motor_Izquierdo, Fuerza_Izq);
        OnFwd(Motor_Derecho, Fuerza_Der);
        if((CurrentTick()-PosicionCorrecta)>LimiteTiempoCaida)
             {
             break;
             }
        Wait(8);
string angulo = NumToStr(Angulo_Giroscopio);
string motor = NumToStr(Fuerza);
string frase = StrCat(angulo,",",motor);
WriteLnString(handle,frase,bytes_escritos);
        }
```

```
PlaySound(SOUND_DOWN);
Off(Ambos_Motores);
Wait(1000);
}
```

2.8.7.- Función taskControl

Esta función recibe los datos por BLUETOOTH® y los introduce en los valores que se le pasan a los motores para poder avanzar, retroceder o girar.

```
task taskControl()
{
Follows(main);
string msg0;
string msg1;
float valor0;
float valor1;
while(true)
   {
    ClearScreen();
         ReceiveRemoteString(0,false,msg0);
         ReceiveRemoteString(1,false,msg1);
    valor0=atof(msg0)-100;
    valor1=atof(msg1)-100;
    NumOut(10,LCD_LINE7,valor0);
    NumOut(10,LCD_LINE8,valor1);
    if (valor0 == -100) valor0 = 0;
    if (valor1 == -100) valor1 = 0;
    motorControlDrive = (valor0 + valor1) * 600 / 200.0;
    motorControlSteer = (valor0 - valor1) * 600 / 200.0;
    Wait(25);
   }
//Wait(10000);
}
```

2.9.- Control externo

Para darle movilidad al "robot" usaremos algún controlador externo e inalámbrico. Se han estudiado varias opciones disponibles en el mercado.

2.9.1.- Control por mando Lego

Lego incorpora en sus periféricos para los kits de montaje robótico unos mandos infrarrojos que conectan con nuestro kit de forma sencilla a través de un nuevo sensor receptor de infrarrojos. Los datos recibidos por el sensor son monitorizados por el microcontrolador e interpretados por nuestro código para conseguir avanzar, retroceder o girar. Esta primera solución es realmente fácil de implementar pero tiene el inconveniente de que solo nos serviría para esta maqueta de pruebas. Es algo exclusivo de Lego. Realmente, con lo que estamos trabajando es una maqueta y sería mas interesante una solución mas "universal".

Ilustración 24 – Mando infrarrojos Lego

Ilustración 25 – Receptor infrarrojos Lego

2.9.2.- Control con mando PSP

Otra posible solución para controlar el robot sería usar un mando de Sony Playstation®. Es un mando que se comunica por radio y por consiguiente necesitamos de un receptor para mandos de Playstation®. La programación del código para recibir las señales es muy sencilla. El lenguaje NXC ya trae implementada una función para uso exclusivo de este mando. Se trata de:

PSP_ReadButtonState(SensorPort, ADDR, currState)

El resultado nuevamente es muy bueno y aunque el mando Playstation® es algo mas genérico que el Lego todavía no es totalmente universal. Estamos obligados a comprar una marca en concreto.

Ilustración 26 - Mando Playstation®

2.9.3.- Control por PC via BLUETOOTH®

La última opción es sin duda la mas interesante. Es totalmente universal y de buen manejo. El único inconveniente que puede tener es "crear" un mando virtual. Para esto se puede recurrir al programa Visual Basic que es capaz de crear un entorno aceptable. La programación no es muy complicada pero requiere de conocimientos en programación en Visual Basic.

2.10.- Código Visual Basic

Para poder manejar el dispositivo a distancia nos hemos basado en la tecnología BLUETOOTH®. El programa necesario para generar el interfaz se puede escribir en código Visual Basic. El manejo de este código es muy sencillo e intuitivo. Además del dispositivo principal, éste interfaz también está creado a modo de muestra de lo que se podría conseguir. Realmente, lo más útil e inmediato es poder caminar hacia delante, caminar hacia atrás y ser capaz de girar. Si llegara el caso necesitamos ejecutar más acciones sería cuestión de ampliar el código para conseguir nuevos objetivos. Por ejemplo, algo que se podría hacer es mandar un mensaje de audio al dispositivo o que el dispositivo interprete un código de colores y actuar en consecuencia.

Nosotros nos vamos a limitar a movimiento avance, retroceso y giro.

Para conseguir manejar el dispositivo se han recurrido a las siguientes funciones VB.

2.10.1.- Vincular dispositivo

Para vincular el dispositivo con el PC previamente habrá que tener conectado el brick Lego con el PC vía BLUETOOTH®. A continuación, el interfaz debería poder vincular ambos dispositivos. El código que implementa esta acción es el siguiente:

```
Private Sub Button1_Click() Handles Button1.Click
        If ComboBox1.SelectedIndex = -1 Then Exit Sub
        'MsgBox("Conectando a : " & ComboBox1.Items(ComboBox1.SelectedIndex))
        Try
            With SerialPort1
                .PortName = ComboBox1.Items(ComboBox1.SelectedIndex)
                .BaudRate = 9600
                .Parity = IO.Ports.Parity.None
                .DataBits = 8
                .StopBits = IO.Ports.StopBits.One
                .ReadTimeout = 300 '300ms
                .WriteTimeout = 300 '300ms
            End With
            SerialPort1.Open()
            Label1.Text = "Conectado"
            Button1.Enabled = False
            Timer1.Enabled = True
        Catch ex As Exception
            Label1.Text = "Error. No Conectado "
            SerialPort1.Close()
            Button1.Enabled = True
            Timer1.Enabled = False
        End Try
    End Sub
```

Si pulsamos el botón “Desconectar” liberaremos el dispositivo del PC.

```
Private Sub Button7_Click(sender As System.Object, e As System.EventArgs) Handles
Button7.Click
        SerialPort1.Close()
        Label1.Text = "Desconectado"
        Button1.Enabled = True
        Timer1.Enabled = False
    End Sub
```

Ya con el robot conectado podemos empezar a usar las barras deslizantes para darle movimiento. Un temporizador mandará el valor de cada barra deslizante mediante 2 mensajes diferentes al PC y allí serán interpretados para aumentar o decrementar la velocidad de los motores. Los mensajes se lanzan cada 10 mS.

```
Private Sub Timer1_Tick(sender As System.Object, e As System.EventArgs) Handles
Timer1.Tick
        'track bar 1
        Dim byteOut(64) As Byte, Cadena As String
        Cadena = Str(TrackBar1.Value)
        lbl_tb1.Text = TrackBar1.Value - 100
        lbl_tb2.Text = TrackBar2.Value - 100
        Dim i As Integer
        Try
            byteOut(0) = Len(Cadena) + 5 'number bytes in output message
            byteOut(1) = &H0 'should be 0 for NXT
            byteOut(2) = &H80 '&H0 = reply expected &H80 = no reply expected
            byteOut(3) = &H9 'Send Bluetooth
            byteOut(4) = &H1 'Box Number - 1
            byteOut(5) = Len(Cadena) + 1 'message size with null terminator
            For i = 1 To Len(Cadena) 'copy bytes into output array
                byteOut(i + 5) = Asc(Mid(Cadena, i, 1))
            Next
            byteOut(Len(Cadena) + 6) = &H0 'add null terminator
            SerialPort1.Write(byteOut, 0, Len(Cadena) + 7) 'send message
        Catch ex As Exception
```

```
            MsgBox(ex.ToString)
        End Try
        If CheckBox1.Checked = True Then
            TrackBar2.Enabled = True
            TrackBar2.Value = TrackBar1.Value
            TrackBar2.Enabled = False
        End If

        'track bar 2
        Dim byteOut1(64) As Byte
        Cadena = Str(TrackBar2.Value)
        Try
            byteOut1(0) = Len(Cadena) + 5 'number bytes in output message
            byteOut1(1) = &H0 'should be 0 for NXT
            byteOut1(2) = &H80 '&H0 = reply expected &H80 = no reply expected
            byteOut1(3) = &H9 'Send Bluetooth
            byteOut1(4) = &H0 'Box Number - 1
            byteOut1(5) = Len(Cadena) + 1 'message size with null terminator
            For i = 1 To Len(Cadena) 'copy bytes into output array
                byteOut1(i + 5) = Asc(Mid(Cadena, i, 1))
            Next
            byteOut1(Len(Cadena) + 6) = &H0 'add null terminator
            SerialPort1.Write(byteOut1, 0, Len(Cadena) + 7) 'send message
        Catch ex As Exception
            MsgBox(ex.ToString)
        End Try

    End Sub
```

El botón “Igualar” anula la barra deslizante izquierda y hace que con solo la barra derecha actuemos sobre los dos motores por igual.

```
Private Sub CheckBox1_CheckedChanged(sender As System.Object, e As System.EventArgs)
Handles CheckBox1.CheckedChanged
        If CheckBox1.Checked = True Then
            TrackBar2.Enabled = True
            TrackBar2.Value = TrackBar1.Value
```

```
            TrackBar2.Enabled = False
        Else
            TrackBar2.Enabled = True
        End If
    End Sub
```

El botón “Reset” coloca las dos barras deslizantes en el centro, mandando un valor cero a cada motor, lo que equivale a dejar el robot quieto.

```
Private Sub Button8_Click(sender As System.Object, e As System.EventArgs) Handles
Button8.Click
        Dim byteOut(64) As Byte
        TrackBar1.Value = 100
        TrackBar2.Value = 100
    End Sub
```