Simulación de un péndulo invertido

Proyecto fin de carrera

Titulación: Ingeniero Técnico en Informática de Sistemas

Autor: Jose Luis Beltrán Alonso

Director: Pascual Pérez Blasco

6 de Diciembre de 2010

Índice general

Índice de figuras

Capítulo 1

Introducción

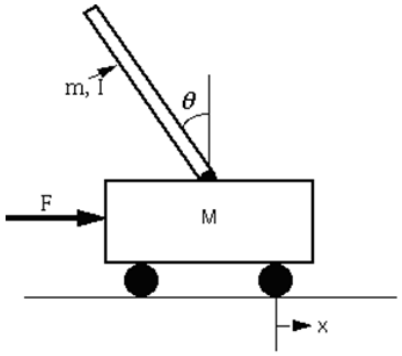
El péndulo invertido es conocido por ser uno de los problemas más importantes y clásicos de la teoría de control. Se trata de un control inestable y no lineal. A menudo, es utilizado como ejemplo académico, principalmente por ser un sistema de control más accesible, y por otro lado, permite mostrar las principales diferencias de control de bucle abierto y de su estabilización a bucle cerrado. Pese a existir diferentes técnicas a la hora de diseñar el regulador óptimo capaz de estabilizar el péndulo, no todas representan la mejor opción. En la presente memoria se analizarán alguno de los métodos más conocidos.

1.1. Descripción del péndulo invertido

El péndulo invertido es un servo mecanismo que consta de un carro en el cual está montado un péndulo que puede girar libremente. El carro está controlado por un servomotor y su principal función es la de aplicar fuerzas al péndulo. Como la finalidad de este proyecto es dar la posibilidad de ejecutar el algoritmo de control en un sistema real (como, por ejemplo, un Segway), implica que el carro puede desplazarse sin limitación alguna, es decir, que si estuviese montado sobre un riel, éste no tendría topes.

Si se considera al péndulo separado del carro, éste tiene dos puntos de equilibrio: uno estable, abajo; y otro inestable, arriba. El objetivo del control es cambiar la dinámica del sistema para que en la posición vertical, arriba, se tenga un punto de equilibrio estable. En otras palabras, la idea es encontrar la fuerza que ha de aplicarse al carro para que el péndulo no se caiga, incluso

si se le perturba con un empujón tipo escalera o impulso.



M	Masa del carro
m	Masa del péndulo
b	Fricción del carro
l	Longitud al centro de masa del péndulo
I	Inercia del péndulo
F	Fuerza aplicada al carro
x	Coordenadas de posición del carro
theta	Ángulo del péndulo respecto de la vertical

Figura 1.1: Esquema péndulo invertido

1.2. Objetivos

El proyecto que se presenta a continuación permite la simulación del comportamiento físico de un sistema dinámico que evoluciona con el tiempo. El objetivo es diseñar un sistema de control óptimo que permita estabilizar un péndulo invertido asi como el desarrollo de una aplicación capaz de simularlo. Los sistemas de control requieren del hardware adecuado para obtener los datos de forma precisa. Normalmente, se consigue con un microcontrolador o un DSP, pero en este caso, se hará mediante una simulación. La aplicación ha sido realizada en un Applet de Java lo que facilita su ejecución desde cualquier navegador web. Sin embargo, para el diseño del regulador se ha utilizado la herramienta Matlab, principalmente, por su fácil manejo en el cáculo de matrices.

Respecto al algoritmo de control, el objetivo es estabilizar el péndulo partiendo éste inicialmente en la parte superior, por lo que no se contempla el algoritmo encargado de levantar el péndulo y lograr estabilizarlo en su posición vertical.

Capítulo 2

Modelado Matemático del Péndulo

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 él un conjunto de acciones (señales de entrada) observando cómo se comporta (estudiar las salidas) deduciendo un modelo matemático para éste. 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**.

2.1. 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.

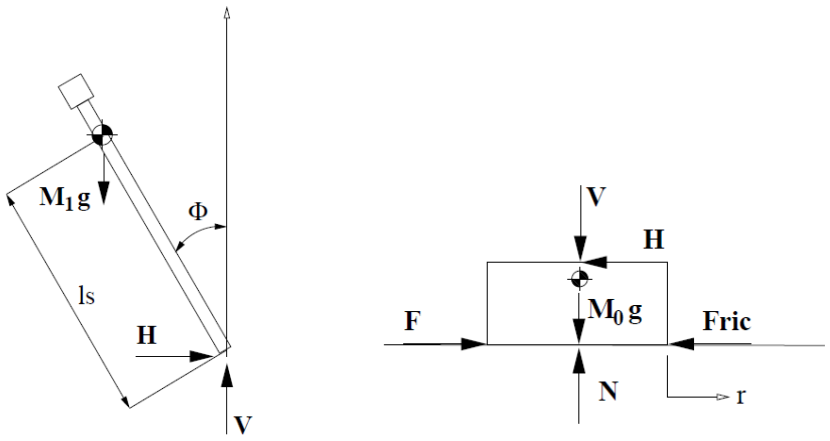


Figura 2.1: Diagramas de cuerpo libre del sistema.

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:

$$\sum F_i = ma_i \tag{2.1}$$

$$\sum F_j = ma_j \tag{2.2}$$

$$\sum F_G = I\alpha_g \tag{2.3}$$

Sumando las fuerzas en el diagrama de cuerpo libre del carro en la dirección horizontal, se obtiene la siguiente ecuación del movimiento:

$$M\ddot{x} + b\dot{x} + N = F \tag{2.4}$$

También se podría sumar las fuerzas en la dirección vertical, pero no se ganará ninguna información útil. Por otro lado, sumando las fuerzas en el diagrama de cuerpo libre del péndulo en la dirección horizontal, se puede obtener una ecuación para N:

$$N = m\ddot{x} + ml\ddot{\theta}cos\theta - ml\dot{\theta}^2 sin\theta \tag{2.5}$$

Si se sustituye esta ecuación en la (2.4), se obtiene la primera ecuación del movimiento de este sistema:

$$(M+m)\ddot{x} + b\dot{x} + ml\ddot{\theta}cos\theta - ml\dot{\theta}^2 sin\theta = F \tag{2.6}$$

Para obtener la segunda ecuación del movimiento, es necesario sumar las fuerzas perpendiculares al péndulo. Resolviendo el sistema a lo largo de este eje se simplifica el cálculo algebraico.

$$Psin\theta + Ncos\theta - mgsin\theta = ml\ddot{\theta} + m\ddot{x}cos\theta \tag{2.7}$$

Para librarse de los términos P y N en la ecuación (2.7), se han sumado los momentos sobre el centroide del péndulo para obtener la primera ecuación (2.8) mostrada a continuación. Finalmente, combinando dicha ecuación con la (2.6), se obtiene la segunda ecuación dinámica (2.9).

$$-Plsin\theta - Nlcos\theta = I\ddot{\theta} \tag{2.8}$$

$$(I + ml^2)\ddot{\theta} + mglsin\theta = -ml\ddot{x}cos\theta \tag{2.9}$$

Para facilitar la labor, se puede trabajar solo con funciones lineales. Para ello, se asume que theta = Pi + ø, donde ø representa un pequeño ángulo en la dirección vertical. Por lo tanto, las dos ecuaciones de movimiento serán:

$$(I + ml^2)\ddot{\phi} - mgl\phi = ml\ddot{x} \tag{2.10}$$

$$(M + m)\ddot{x} + b\dot{x} - ml\ddot{\phi} = u \tag{2.11}$$

2.2. Función de transferencia

Cualquier sistema físico (mecánico, eléctrico, etc.) se puede traducir a una serie de valores matemáticos a través de los cuales se conoce el comportamiento de estos sistemas frente a valores concretos. Esto es lo que permite la función de transferencia. Se trata de un modelo matemático que a través de un cociente relaciona la respuesta de un sistema (salida) a una señal de entrada o excitación. Por definición una función de transferencia se puede determinar según la expresión:

$$H(s) = \frac{Y(s)}{U(s)} \tag{2.12}$$

Donde H(s) es la función de transferencia (también denotada como G(s)). Y(s) es la transformada de Laplace de la respuesta y U(s) es la trasformada de Laplace de la señal de entrada.

Aplicándolo al péndulo invertido, para obtener analíticamente la función de transferencia de las ecuaciones del sistema linealizado, se ha de tomar primero la transformada de Laplace de las ecuaciones del sistema. Donde se obtienen las siguientes ecuaciones:

$$(I + ml^2)\Phi(s)s^2 - mgl\Phi(s) = mlX(s)s^2 \tag{2.13}$$

$$(M + m)X(s)s^2 + bX(s)s - ml\Phi(s)s^2 = U(s) \tag{2.14}$$

Dado que la salida de interés en este caso es el ángulo Phi, en primer caso se resuelve la primera ecuación para X(s), donde posteriormente se sustituirá en la segunda ecuación:

$$X(s) = \left[\frac{(I + ml^2)}{ml} - \frac{g}{s^2}\right]\Phi(s) \tag{2.15}$$

$$(M + m)\left[\frac{(I + ml^2)}{ml} + \frac{g}{s}\right]\Phi(s)s^2 + b\left[\frac{(I + ml^2)}{ml} + \frac{g}{s}\right]\Phi(s)s - ml\Phi(s)s^2 = U(s) \tag{2.16}$$

Reordenando la función de transferencia se obtiene:

$$\frac{\Phi(s)}{U(s)} = \frac{\frac{ml}{q}s^2}{s^4 + \frac{b(I+ml^2)}{q}s^3 - \frac{(M+m)mgl}{q}s^2 - \frac{bmgl}{q}s} \tag{2.17}$$

$$q = \left[(M + m)(I + ml^2) - (ml)^2\right]$$

De la función de transferencia de arriba puede verse que hay un polo y un cero en el origen. Estos puede ser cancelados y la función de transferencia será:

$$\frac{\Phi(s)}{U(s)} = \frac{\frac{ml}{q}s}{s^3 + \frac{b(I+ml^2)}{q}s^2 - \frac{(M+m)mgl}{q}s - \frac{bmgl}{q}} \tag{2.18}$$

$$q = \left[(M + m)(I + ml^2) - (ml)^2\right]$$

En este punto ya se tiene calculada la función de transferencia que representa al sistema físico del péndulo invertido. Mediante esta ecuación ya se puede realizar pruebas para comprobar si el sistema es estable a lazo abierto (véase 2.4 “Respuesta a lazo abierto”)

2.3. Ecuación de estado

El espacio de estados es otro 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:

$$x = Ax + Bu$$
$$y = Cx + Du$$

Donde el vector **x** es el estado del sistema y contiene n elementos para un sistema de orden n. **u** es el vector de entrada y contiene m elementos. **y** contiene p elementos y es el vector de salida. **A**, de dimensión n x n es la matriz del sistema. **B**, de dimensión n x m es la matriz de entrada. **C**, de dimensión p x n es la matriz de salida y **D** es una matriz de dimensión p x m.

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 **x** que determina el estado del sistema contendrá cuatro elementos (posición del carro, primera derivada, posición del ángulo y su derivada). En el caso del vector **y** se ha considerado que el péndulo consta de tres sensores, uno para la posición del carro r, otro para el ángulo ϕ y el último para la velocidad del carro $\dot{r}$. El vector **u** tiene un único elemento que es la fuerza aplicada al carro. Una vez conocida esta información se pueden determinar las matrices de **C** y **D** (donde D=0).

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} r \\ \Phi \\ \dot{r} \\ \dot{\Phi} \end{bmatrix} \qquad y = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad u = F$$

Para obtener A y B es necesario expresar las ecuaciones 2.1 y 2.6 en la forma:

$$\dot{x} = f(x, u) \tag{2.19}$$

de tal forma que:

$$\dot{x}_1 = f_1(x, u) = x_3 \tag{2.20}$$

$$\dot{x}_2 = f_2(x, u) = x_4 \tag{2.21}$$

de la ecuación 2.6 y haciendo $\alpha = M_1 l_s$

$$\dot{x}_3 = \frac{\alpha \dot{x}_4 cos x_2 - \alpha {x_4}^2 sen x_2 - F_r x_3 + u}{M} \tag{2.22}$$

$$\dot{x}_4 = \frac{\alpha \dot{x}_3 cos x_2 + \alpha g sen x_2 - C x_4}{\Theta} \tag{2.23}$$

Sustituyendo 2.22 en 2.19 se obtiene $\dot{x}_3 = f_3(x, u)$ pero la ecuación resultante no es lineal. Para poder representarla como ecuación de estado lineal debe tener la siguiente forma:

$$\dot{x}_i = \sum_{j=1}^{n} a_{ij} x_j + b_i u \tag{2.24}$$

Donde a_{ij} y b_i son constantes. Para linealizar la ecuación $\dot{x}_3 = f_3(x, u)$ se puede expresar $f_3(x, u)$ como una serie de Taylor y utilizar únicamente el primer término.

$$\dot{x}_3 \approx \sum_{i=1}^{4} \left[\frac{\partial f_3(x, u))}{\partial x_i} \bigg|_{x=0, u=0} \Delta x_i \right] + \frac{\partial f_3(x, u)}{\partial u} \bigg|_{x=0, u=0} \Delta u \tag{2.25}$$

Al calcular las derivadas parciales y hacer $\beta = \Theta M - \alpha^2$ se tiene:

$$a_{31} = \frac{\partial f_3(x,u)}{\partial x_1}\bigg|_{x=0,u=0} = 0 \tag{2.26}$$

$$a_{32} = \frac{\partial f_3(x,u)}{\partial x_2}\bigg|_{x=0,u=0} = \frac{\alpha^2 g}{\beta} \tag{2.27}$$

$$a_{33} = \frac{\partial f_3(x,u)}{\partial x_3}\bigg|_{x=0,u=0} = \frac{\Theta F_r}{\beta} \tag{2.28}$$

$$a_{34} = \frac{\partial f_3(x,u)}{\partial x_1}\bigg|_{x=0,u=0} = -\frac{\alpha C}{\beta} \tag{2.29}$$

$$b_3 = \frac{\partial f_3(x,u)}{\partial u}\bigg|_{x=0,u=0} = -\frac{\Theta}{\beta} \tag{2.30}$$

Para el caso de $\dot{x}_4$ hay que seguir el mismo procedimiento que el realizado para $\dot{x}_3$. A partir de este punto, ya se pueden determinar los coeficientes de **A** y **B**.

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & a_{42} & a_{43} & a_{44} \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ 0 \\ b_3 \\ b_4 \end{bmatrix}$$

Finalmente, ya puede expresarse en la forma de espacio de estados:

$$\dot{X}_n = A_n x_n + B_n u$$
$$Y_n = C x_n$$

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\Phi} \\ \ddot{\Phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(I+ml^2)b}{I(M+m)+Mml^2)} & \frac{m^2gl^2}{I(M+m)+Mml^2)} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlb}{I(M+m)+Mml^2)} & \frac{mgl(M+m)}{I(M+m)+Mml^2)} & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\Phi} \\ \ddot{\Phi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I+ml^2}{I(M+m)+Mml^2)} \\ 0 \\ \frac{ml}{I(M+m)+Mml^2)} \end{bmatrix} \tag{2.31}$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

2.4. Respuesta a 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 donde se asumirán los siguientes valores que se mantendrán durante todos los ejercicios prácticos:

M: Masa del carro = 0.5 Kg
m: Masa del péndulo = 0.5 Kg
b: Fricción del carro = 0.1 N/m/seg
l: Longitud péndulo = 0.3 m
I: Inercia del péndulo = 0.006 Kg*m^2
F: Fuerza aplicada al carro
x: Coordenadas de posición del carro
Θ: Ángulo del péndulo

Dado que se han obtenido dos modelos matemáticos diferentes por métodos distintos, el siguiente paso será hacer una simulación para cada uno de ellos. Cada simulación hará uso de unos requerimientos de diseño diferentes debido al tipo de variables con el que trabaja.

2.4.1. Solución: Mediante función de transferencia

En el caso de la función de transferencia solo se va a tratar con una sistema de única salida (el ángulo del péndulo), y por lo tanto, los requerimientos de diseño son:

- Tiempo de establecimiento menor a 5 segundos.
- Ángulo del péndulo siempre menor que 0.05 radianes de la vertical.

El numerador y denominador de la función de transferencia se almacenaran en vectores de tal forma que el código quedaría así:

```
M = 0.5;
m = 0.2;
b = 0.1;
i = 0.006;
```

```
5  g = 9.8;
6  l = 0.3;
7  q = (M+m)*(i+m*l^2)-(m*l)^2;
8  num = [m*l/q 0]
9  den = [1 b*(i+m*l^2)/q -(M+m)*m*g*l/q -b*m*g*l/q]
```

Donde su salida debería ser:

```
num =
    4.5455      0
den =
    1.0000      0.1818   -31.1818        -4.4545
```

Para observar la respuesta del sistema a una fuerza impulsiva de 1N aplicada al carro, se introducirá un impulso a la entrada mediante la función impulse. Para ello es necesario añadir el siguiente código:

```
11 t=0:0.01:5;
12 impulse(num,den,t)
13 axis([0 1 0 60])
```

La función axis se encarga de dibujar los ejes que aparecerán en la gráfica. En este caso, el eje X representa el tiempo en segundos y el vector Y la amplitud. Como resultado de la simulación se obtiene la gráfica donde puede observarse como la respuesta es totalmente insatisfactoria. Como era de esperar no es estable a lazo abierto y por lo tanto será necesario diseñar un regulador capaz de estabilizar el péndulo (ver capítulo 3).

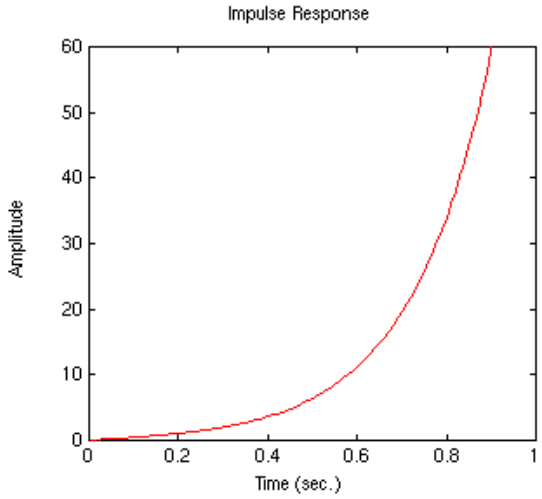


Figura 2.2: Respuesta F.D.T en lazo abierto

2.4.2. Solución: Mediante espacio de estados

En este caso los requerimientos de diseño son distintos pues se trata de un sistema multisalida y el objetivo ya no es solo controlar el ángulo del péndulo sino también la posición del carro. Por lo tanto, los requerimientos de diseño a tener en cuenta son:

- Tiempo de establecimiento de x y θ menor a 5 segundos.
- Tiempo de subida para x menor que 0.5 segundos.
- Sobrepico de θ menor que 20 grados.

En primer lugar hay que calcular las matrices A,B,C y D a partir de la expresión obtenida en 2.31

```
M = 0.5;
m = 0.2;
b = 0.1;
i = 0.006;
g = 9.8;
l = 0.3;

p = i*(M+m)+M*m*l^2; %denominador para las matrices A y B
```

```
A = [0       1                0            0;
     0 -(i+m*l^2)*b/p  (m^2*g*l^2)/p    0;
     0       0                0            1;
     0 -(m*l*b)/p        m*g*l*(M+m)/p   0]
B = [0;
    (i+m*l^2)/p;
     0;
     m*l/p]
C = [1 0 0 0;
     0 0 1 0]
D = [0; 0]
```

Como resultado se obtiene:

```
A =
   0  1.0000 0       0
   0 -0.1818 2.6727  0
   0  0      0       1.0000
   0 -0.4545 31.1818 0
B =
   0
   1.8182
   0
   4.5455
C =
   1 0 0 0
   0 0 1 0
D =
   0
   0
```

Al igual que en el caso anterior hay que introducir una entrada que en este caso será un escalón de 0,2 metros. Añadiendo el siguiente código:

```
T=0:0.05:10;
U=0.2*ones(size(T));
[Y,X]=lsim(A,B,C,D,U,T);
plot(T,Y)
axis([0 2 0 100])
```

Se observa la salida donde la línea azul representa la posición del carro y la línea verde el ángulo del péndulo. Al igual que en la figura anterior, para mejorar la dinámica del sistema es necesario incluir alguna clase de control. Este regulador se diseñará en el apartado 3.2.

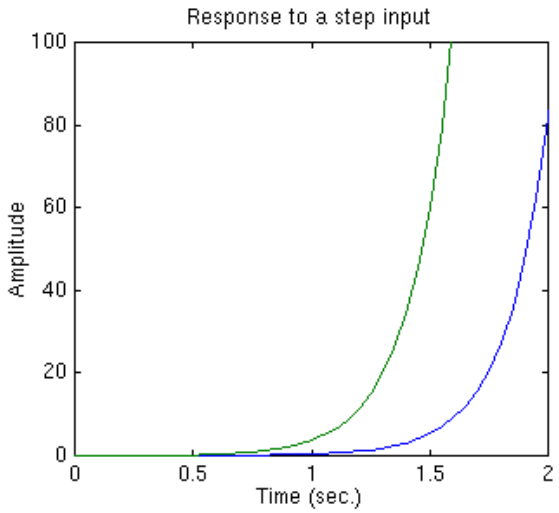


Figura 2.3: Respuesta mediante espacio de estados

Capítulo 3

Estabilización del péndulo

Llegados a este punto se ha comprobado que la dinámica del sistema por si sola no es capaz de mantener el péndulo estable. Aquí nace la necesidad de incluir un regulador capaz de "decidir" en cada momento cuál es la acción a realizar sobre el carro para que el péndulo no se desestabilice. Existen distintos métodos para diseñar un regulador. En este capítulo se abordará concretamente el diseño en espacio de estados mediante la creación de un regulador LQR. También se mostrará otras tres posibles soluciones al problema del péndulo invertido, mediante el método del Lugar de las Raíces, haciendo uso de un control PID, y con el método de respuesta en frecuencia.

3.1. Controlador PID

Los controladores PID han logrado colocarse en un sector importante en la industria pues han demostrado ser muy robusto para muchas de sus aplicaciones. El objetivo de este apartado es lograr diseñar un PID capaz de estabilizar el péndulo invertido y analizar posteriormente si el controlador diseñado será el más adecuado para el sistema.

3.1.1. Estructura

La estructura de un controlador PID es simple pero en ocasiones su sencillez puede ser también su debilidad. Considerando un lazo de control de una entrada y una salida (SISO) de un grado de libertado obtenemos el siguiente diagrama:

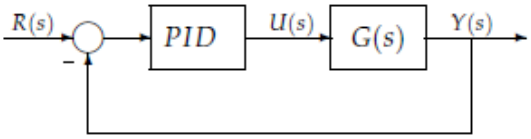


Figura 3.1: Diagrama en bloques

A diferencia del capítulo anterior, ahora se está trabajando con sistemas a 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, es decir: u(t) = KP e(t). Un controlador proporcional puede controlar cualquier planta estable, pero posee desempeño limitado y error en régimen permanente.

- **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.

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

  La señal de control u(t) tiene un valor diferente de cero cuando la señal de error e(t) es cero. Por lo que se concluye que dada una referencia constante, o perturbaciones, el error en régimen permanente es cero.

- **I: Acción de control proporcional-integral** se define mediante

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

  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.

- **I: Acción de control proporcional-derivativa** se define mediante:

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

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 reúne las ventajas de cada una de las tres acciones de control individuales. La ecuación de un controlador con esta acción combinada se obtiene mediante:

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

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

3.1.2. Diseño PID

El objetivo de este apartado es calcular manualmente la función de transferencia que represente el regulador óptimo. Normalmente, el procedimiento válido a la hora de diseñar un PID es comenzar utilizando sólo el control proporcional. De forma que el diagrama de bloques sería:

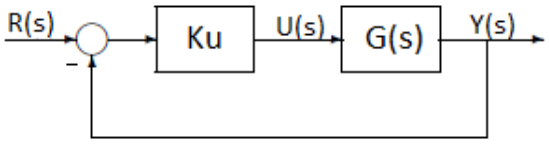


Figura 3.2: Regulador P

La ecuación característica del bucle cerrado viene dada por:

$$M(z) = \frac{K_u G(z)}{1 + K_u G(z)} \tag{3.1}$$

El denominador aparece sumado ya que la realimentación es negativa, en caso de ser positiva habría que sustituirlo por 1 - Ku G(z). Para entender mejor la teoría, a continuación, se detalla un ejemplo manual sencillo. En el caso del péndulo invertido se hará uso de la herramienta Matlab tal y como se detalla en el apartado 3.1.3.

La forma de diseñar este PID es haciendo uso del método de ajuste de Nichols-Ziegler. Básicamente consiste en lograr unos valores para la I (integral) y D (derivados) de modo que la ganancia sea cero. La función de transferencia del regulador tiene la siguiente estructura:

$$G_r(z) = \frac{a_0 z^2 + a_1 z + a_2}{z(z-1)} \tag{3.2}$$

$$a_o = k_c \left(1 + \frac{T_d}{T}\right); a_1 = K_c \left(-1 + \frac{T}{T_i} - \frac{2T_d}{T}\right); a_2 = k_c \frac{T_d}{T} \tag{3.3}$$

Para cada tipo de controlador se sigue los siguientes parámetros:

- Controlador P: K_c=$0.5K_u$
- Controlador PI: K_c=$0.45K_u T_i = T_u/1{,}2$
- Controlador PID: K_c=$0.6K_u T_i = T_u/2T_d = T_u/8$

Donde,

$$T_u = \frac{2\pi}{W_u} \quad W_u = \frac{1}{T} arctg\left(\frac{\beta}{\alpha}\right)$$

Para este ejemplo, se va a asumir que se tiene la siguiente función de transferencia:

$$G(z) = \frac{1}{z^2 - 0{,}87} \tag{3.4}$$

• Paso 1 Calcular Ku

Conociendo la ecuación característica en bucle cerrado se puede igualar a cero el denominador para despejar Ku:

$$1 + KuG(z) = 0 \tag{3.5}$$

$$1 + Ku\frac{1}{z^2 - 0,87} = 0 \tag{3.6}$$

El siguiente paso será resolver la ecuación de segundo grado manteniendo la incógnita Ku y forzando a que el módulo de los polos sea 1.

$$z = \frac{0,8 \pm \sqrt{0,8^2 - 4Ku}}{2} \tag{3.7}$$

$$|z| = 1 = \sqrt{\left(\frac{0,8}{2}\right)^2 + \left(\frac{\sqrt{4Ku - 0,8^2}}{2}\right)} \tag{3.8}$$

$$1 = 0,4^2 + \frac{4Ku - 0,8^2}{4} \tag{3.9}$$

Finalmente, despejando se obtiene que Ku=1

• Paso 2. Obtener polos para Ku

Una vez obtenida Ku ya se puede sustituir en la ecuación de segundo grado (3.7):

$$z = \frac{0,8 \pm \sqrt{0,8^2 - 4}}{2} = \frac{0,8 \pm \sqrt{0,64 - 4}}{2} = \frac{0,8 \pm 1,83j}{2} = 0,4 \pm 0,916j \tag{3.10}$$

• Paso 3. Calculo de Tu

Se supone que el valor de periodo es de 0.1. Es importante destacar que la división entre α y β se calcula en radianes.

$$W_u = \frac{1}{0{,}1} arctg\left(\frac{0,4}{0,916}\right) = 11,593$$

$$T_u = \frac{2\pi}{11,593} = 0,542 \tag{3.11}$$

• Paso 4. Obtener regulador discreto.

- $K_c = 0,6K_u = 0,6 * 1 = 0,6$
- $T_i = T_u/2 = 0,271$
- $T_d = T_u/8 = 0,06775$

• Paso 5. Obtener fdt del regulador discreto.

En este punto, sustituyendo los valores obtenidos en la función 3.3 se puede hallar los valores para a_0, a_1 y a_2.

$$G_r(z) = \frac{1,0065z^2 - 1,1916z + 0,4065}{z(z-1)} \tag{3.12}$$

• Paso 6. Obtener ecuación en diferencia para implementar en PC

Llegados a este punto ya se ha obtenido el regulador que se estaba buscando y no es necesario continuar. No obstante, se va a detallar como obtener la ecuación en diferencia, es decir, sacar la expresión capaz de ser interpretada por un ordenador.El procedimiento a seguir será obtener la antitransformada a partir de la f.d.t calculada en el anterior apartado.

$$G_r(z) = \frac{M(z)}{E(z)} = \frac{a_0 z^2 + a_1 z + a_2}{z^2 - z} \tag{3.13}$$

$$(a_0 z^2 + a_1 z + a_2)E(z) = (z^2 - z)U(z) \tag{3.14}$$

$$a_0 e(k) + a_1(l-1) + a_2 e(k-2) = u(k) - u(k-1) \tag{3.15}$$

$$u(k) = a_0 e(k) + a_1 e(k-1) + a_2(k-2) + u(k-1) \tag{3.16}$$

3.1.3. Ejemplo péndulo invertido.

En el caso del péndulo invertido, como ya se ha mencionado anteriormente, se va a hacer uso de la herramienta Matlab. Asi que partiendo del código ya utilizado en el apartado 2.4.1 y siguiendo con los requerimientos de diseño especificados en el mismo apartado ya se puede comenzar a realizar pruebas:

```
M = .5;
m = 0.2;
b = 0.1;
i = 0.006;
g = 9.8;
l = 0.3;

q = (M+m)*(i+m*l^2)-(m*l)^2;    %simplifica entrada

num = [m*l/q  0]
den = [1  b*(i+m*l^2)/q  -(M+m)*m*g*l/q  -b*m*g*l/q]
```

En ocasiones la mejor forma de obtener un regulador óptimo es a base de prueba y error, por ello, en principio, se asume que el valor proporcionar, integral y derivativo valdrá 1:

```
kd = 1;
k = 1;
ki = 1;
numPID = [kd k ki];
denPID = [1 0];
numc = conv(num,denPID)
denc = polyadd(conv(denPID,den),conv(numPID,num))
```

La función polyadd no existe en Matlab pero ha sido creada en un fichero 'm' distinto con el propósito de sumar dos polinomios aun cuando no tuviesen la misma longitud. A continuación, se muestra el código del fichero polyadd.m [1].

```
%Copyright 1996 Justin Shriver
%POLYADD(poly1,poly2) : suma dos polinomios posiblemente no apareados
if length(poly1)0
  poly=[zeros(1,mz),short]+long;
else
  poly=long+short;
end
```

Ahora ya se puede comenzar con la simulación real donde introduciendo un impulso a la entrada la salida es:

```
t=0:0.01:5;
impulse(numc,denc,t)
axis([0 1.5 0 40])
```

[1] No es necesario importar el fichero a Matlab si se encuentra en el mismo directorio que el proyecto con el que se está trabajando.

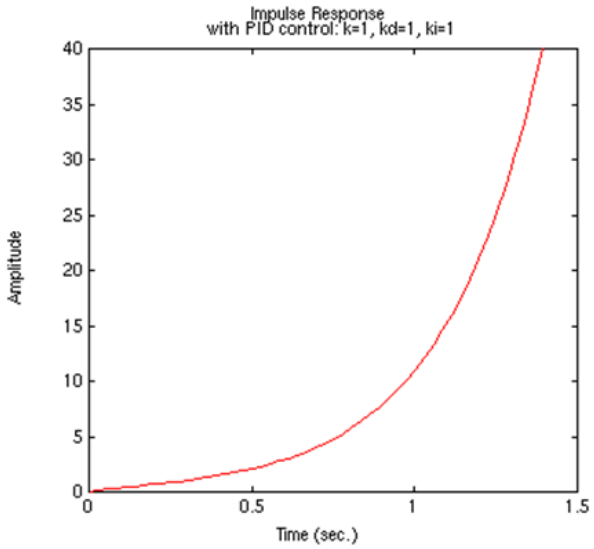


Figura 3.3: PID: Respuesta ante impulso de entrada

Como se observa la respuesta aún no es estable. Como primera prueba, se incrementa la proporcional dando un valor k=100. Y su salida es:

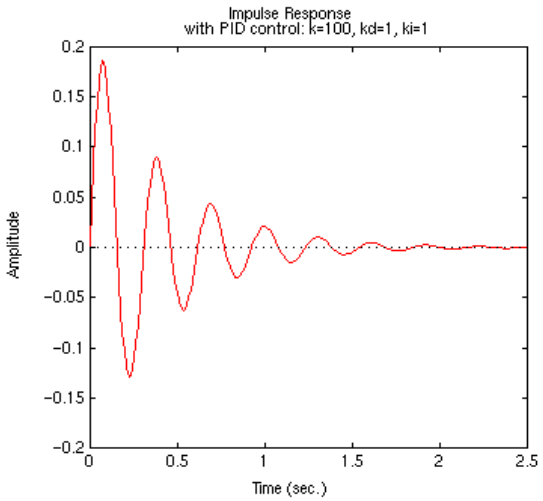


Figura 3.4: PID: Respuesta ante incremento proporcional

Se puede observar que el tiempo de establecimiento empieza a ser aceptable pues es cercano a los 2 segundos. Además, el error estacionario es cero, así que no es necesario ningún control integral.

El siguiente paso es intentar reducir el sobrepico, para ello, se prueba con un valor Kd de 20:

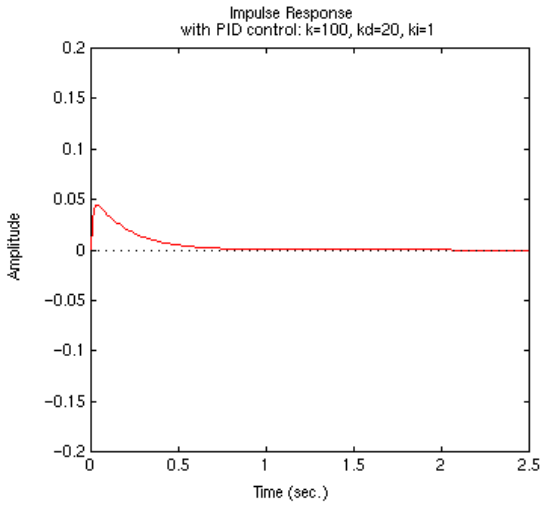


Figura 3.5: PID: Respuesta ante incremento derivativo

Como se puede ver, el sobrepico se redujo de modo que el péndulo no se mueve más que 0.05 radianes de la vertical. Se han satisfecho todos los criterios de diseño, así que ya no se necesita más iteraciones. Al parecer ya se ha obtenido un PID capaz de hacer estable el péndulo pero no se ha tenido en cuenta que ocurre con la posición del carro. Simulando el sistema con la función de transferencia en lazo cerrado que incluye la posición del carro, se obtiene:

```
M = 0.5;
m = 0.2;
b = 0.1;
i = 0.006;
g = 9.8;
l = 0.3;

q = (M+m)*(i+m*l^2)-(m*l)^2;
num1 = [m*l/q  0  0];
den1 = [1  b*(i+m*l^2)/q  -(M+m)*m*g*l/q  -b*m*g*l/q  0];
num2 = [(i+m*l^2)/q  0  -m*g*l/q];
den2 = den1
```

```
kd = 20;
k = 100;
ki = 1;
numPID = [kd k ki];
denPID = [1 0];

numc = conv(num2,denPID);
denc = polyadd(conv(denPID,den2),conv(numPID,num1));
t=0:0.01:5;
impulse(numc,denc,t)
```

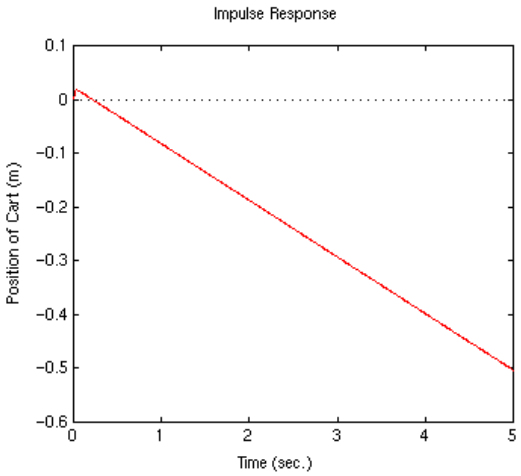


Figura 3.6: PID: Respuesta carrito ante impulso

Como se observa, el carro se desplaza a una velocidad constante y nunca se detiene. A pesar de que el controlador PID estabiliza el ángulo del péndulo, este diseño no es útil para el objetivo que se pretende alcanzar. Por ello, el siguiente paso es intentar un diseño en espacio de estado.

3.2. Regulador LQR

La ventaja de tener el sistema representado en el espacio de estados es que el diseño de un controlador LQR resulta más sencillo. La idea es establecer una retroalimentación completa mediante una combinación lineal de las variables de estado. A pesar de ser lo ideal, la realidad es que el estado que se mide no es completo, por lo que sería conveniente utilizar un observador de orden reducido para estimar parte del estado. En este caso, se asume que todos los estados son medibles. El resultado final del algoritmo de control sería el siguiente:

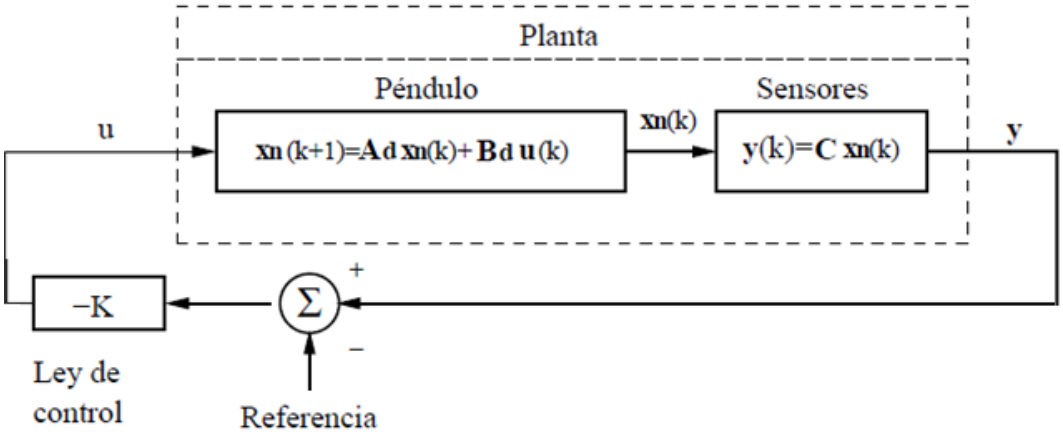


Figura 3.7: Regulador LQR

3.2.1. Base teórica

En la figura 3.7 se puede ver un esquema del algoritmo de control. K es la matriz que retroalimenta el estado para obtener la señal de control, es decir: u(k)= -Kx(k). El siguiente paso en el proceso de diseño es encontrar el vector K que determina la ley de control para la realimentación. Esto puede hacerse de varias formas. La primera opción es asignar arbitrariamente el patrón de polos que se desee que tenga el sistema en lazo cerrado. Y la otra opción es hacer uso de un control óptimo variable en el tiempo. En este caso la opción elegida es un control óptimo LQR. Se trata de minimizar la función:

$$J = \sum_{k=0}^{N} \left[X_n^T(k) Q X_n(k) + u^T(k) R u(k) \right] \tag{3.17}$$

J es una función de costo que generalmente está asociada con la energía del sistema. Se trata de un control óptimo en el sentido de que busca minimizar la energía. La expresión $X_n^T(k)QX_n(k)$ representa la energía que aporta cada estado, **Q** es una matriz no negativa definida (una posibilidad es una matriz diagonal, cuyos elementos sean positivos). A través de **Q** se puede elegir el peso que tiene cada estado en la función J. Se puede conceptualizar $X_n^T(k)QX_n(k)$, como una x^2 (la energía normalizada) en un sistema con una sola variable. Como en este caso u(k) contiene un sólo elemento, $R \geq 0$ es un elemento que indica el peso que se le quiere dar a la energía asociada con la señal de control.

La ecuación 3.13 está sujeta a la restricción impuesta por el sistema (ver funcion 2.31).

$$X_n(k+1) = A_d X_n(k) + B_d u(k) \tag{3.18}$$

Resolver estas ecuaciones requiere de la utilización de métodos numéricos. Dado que el resultado es difícil de obtener, solo mencionar que la solución resulta de una ley de control variable en tiempo:

$$u(k) = -K(k)X_n(k)$$

Para sistemas con coeficientes constantes, K(k) permanece fija durante un periodo de tiempo y luego decae a cero. Si se hace $N \to \infty$ en 3.13 entonces la ganancia del controlador permanece fija todo el tiempo:

$$K(k) = K$$

El valor exacto de J no es relevante, solo se pretende encontrar la K que asegure que sea mínimo. Lo importante es el valor relativo que tienen los elementos de Q y R entre sí. Dichos parámetros serán los encargados de balancear la importancia relativa de la entrada y los estados en la función de costo que se esta tratando de optimizar. El caso más simple es considerar R=1 y proponer Q a partir de ahí.

3.2.2. Diseño en espacio de estados

Una vez analizada la parte teórica ya se puede completar el regulador al que se hacía referencia en el apartado 2.4.2. En dicho apartado se tomaban los siguientes criterios de diseño:

- Tiempo de establecimiento de x y theta menor que 5 segundos.
- Tiempo de Subida para x menor que 0.5 segundos.
- Sobrepico de theta menor que 20 grados (0.35 radianes).

A los cuales además se incluye un cuarto criterio, un error estacionario entre el 2 %. Las características del péndulo siguen siendo las mismas:

```
M = 0.5;
m = 0.2;
b = 0.1;
i = 0.006;
g = 9.8;
l = 0.3;
```

Echando un vistazo a la simulación se determinaron que las ecuaciones de estado para este problema son:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\Phi} \\ \ddot{\Phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -0{,}1818 & 2{,}6727 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -0{,}4545 & 31{,}1818 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\Phi} \\ \ddot{\Phi} \end{bmatrix} + \begin{bmatrix} 0 \\ 1{,}8182 \\ 0 \\ 4{,}5455 \end{bmatrix} \tag{3.19}$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\Phi} \\ \ddot{\Phi} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u$$

Para la obtención del regulador se va a utilizar la función "lqr" que dará el controlador óptimo. Esta función permite elegir dos parámetros, R y Q, donde, como ya se dijo anteriormente, el caso más simple es considerar R=1, y Q, por ejemplo, valdrá Q=C'*C. El método LQR, básicamente permite el control de ambas salidas. Para obtener una respuesta deseable, el controlador puede sintonizarse cambiando los elementos no nulos en la matriz Q. Para obtener la matriz Q es suficiente con introducir el comando C'*C

```
Q =
          1     0     0     0
          0     0     0     0
          0     0     1     0
          0     0     0     0
```

El elemento en la posición 1,1 se usará para la posición del carro y el elemento en la posición 3,3 para el ángulo del péndulo. El valor de la entrada R permanecerá en 1. Ahora que se conoce la matriz Q ya se puede calcular el valor de K y mostrar gráficamente el resultado.

```
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=0:0.01:5;
U=0.2*ones(size(T));
[Y,X]=lsim(Ac,Bc,Cc,Dc,U,T);
plot(T,Y)
legend('Carrito','Péndulo')
```

```
K =
   -1.0000   -1.6567   18.6854    3.4594
```

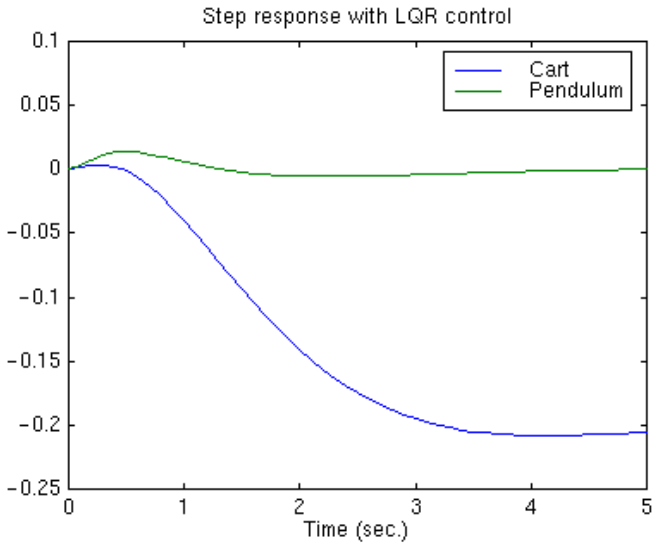


Figura 3.8: Respuesta controlador LQR

La curva en verde representa el ángulo del péndulo, en radianes y la curva azul representa la posición del carro en metros. Como se puede ver, este gráfico no es satisfactorio. Los sobrepicos del péndulo y del carro parecen estar bien, pero debe mejorarse sus tiempos de asentamiento, y debe bajarse el tiempo de subida del carro.

También puede observarse que el carrito no se halla cerca de la ubicación deseada sino que de hecho se ha movido en la dirección contraria. Este error será tratado en la sección siguiente, en este momento hay que resolver los tiempos de subida y de establecimiento.

Para ello hay que variar los valores de las variables x e y para modificar así la matriz Q. Por ejemplo, incrementando la variable x, se consigue que los tiempos de establecimiento disminuyan, por otro lado, incrementando 'y' se reduce el movimiento angular del péndulo.

Para valores x=5000; y=100 se obtiene la siguiente respuesta:

```
K =
   -70.7107   -37.8345   105.5298    20.9238
```

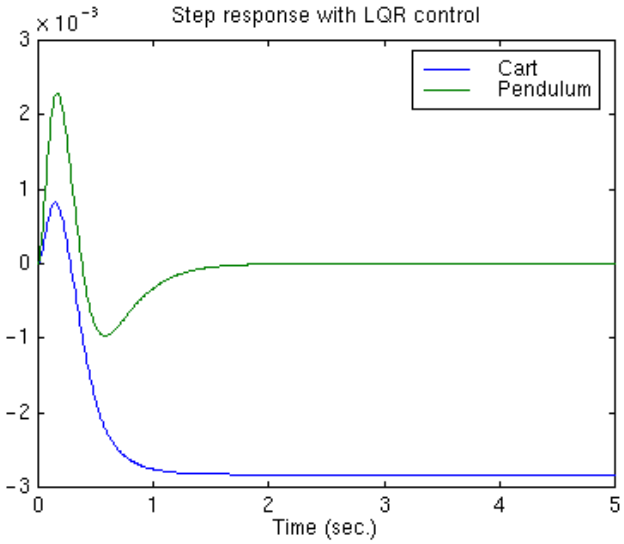


Figura 3.9: Respuesta controlador LQR ante incremento de x,y

Como se puede ver si se sigue incrementando x e y, se podría mejorar aun más la respuesta pero también serían mayores los esfuerzos de control usados. La respuesta del sistema tiene un tiempo de establecimiento por debajo de los 2 segundos. Llegados a este punto se ha cumplido el objetivo de diseñar un regulador capaz de estabilizar tanto el ángulo del péndulo como la posición del carro.

3.3. Observadores

En el control con realimentación de estados se asumía la disponibilidad de las variables de estado. En la práctica este caso no siempre es posible, ya sea porque ciertos estados no son medibles, bien porque es muy difícil o muy caro medirlos. Para implementar una realimentación completa de estados, entonces, es necesario añadir un dispositivo dinámico, llamado observador o estimador de estados, cuya salida sea una estimación del vector de estados. En esta sección se introduce observadores de orden completo, donde el observador tiene el mismo orden que la planta,es decir, se estima todo el vector de estados. Se considera entonces el sistema:

$$\dot{x}(t) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t)$$

donde A,B y C son conocidas, y la entrada u(t) y la salida y(t) son medibles, aunque no el estado x(t). El problema es estimar x(t) de esta información. Para ello hay dos soluciones posibles:

- Observador a lazo abierto.
- Observador de orden reducido.

3.3.1. Observador a lazo abierto

Conociendo A y B, es posible duplicar la ecuación de estados original construyendo el sistema:

$$\hat{\mathrm{x}}(\mathrm{t})=\mathrm{A}\hat{\mathrm{x}}(\mathrm{t})+\mathrm{Bu}(\mathrm{t})$$

Esta duplicación es un observador a lazo abierto, donde si se tuvieran las mismas condiciones inciales, entonces para toda entrada u(t) se tiene que $\hat{\mathrm{x}}(\mathrm{t})=\mathrm{x}(\mathrm{t})$, $\forall T \geq 0$. Básicamente si el sistema es observable se puede utilizar un observador en lazo abierto para estimar el vector de estados. Sin embargo, el observador en lazo abierto tiene la desventaja de tener que calcular el estado inicial cada vez que se utilice el estimador.

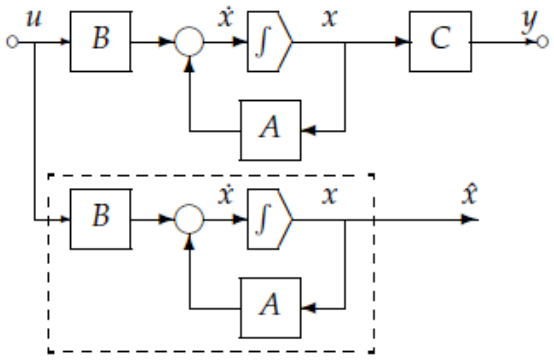


Figura 3.10: Observador a lazo abierto

3.3.2. Observador de orden reducido

Si el par (A,C) es observable, usando la matriz no singular:

$$\mathcal{O} = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

$$\bar{A} = \mathcal{O}A\mathcal{O}^{-1} = \begin{bmatrix} 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 0 & 1 \\ -\alpha_n & -\alpha_{n-1} & \dots & -\alpha_2 & -\alpha_1 \end{bmatrix}$$

$$\bar{C} = C\mathcal{O}^{-1} = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \end{bmatrix}.$$

En estas coordenadas la salida queda como el primer estado, y así, no es necesario construir un observador para estimar todo el estado, sino solamente los n-1 restantes. Este observador es de orden reducido.

Capítulo 4

Implementación

La aplicación que se presenta en este capítulo permite la simulación del comportamiento físico de un sistema dinámico, sistema que evoluciona con el tiempo. Dicha aplicación ha sido desarrollada en forma de *applet* Java utilizando la programación orientada a objetos. Para ello, se han utilizado tanto los entornos de desarollo integrado (IDE) de NetBeans, como Eclipse, ambos proyectos de código abierto fundado por Sun MicroSystems.

Como producto final, el *applet* tiene el objetivo de permitir estudiar el comportamiento del péndulo invertido de un modo visual, permitiendo al usuario interactuar sobre él en tiempo real. Gracias a la portabilidad que Java ofrece, es posible integrar dicho *applet* a una página Web y visualizarlo desde cualquier navegador compatible con java (Internet Explorer, Mozilla Firefox, Google Chrome, etc).

4.1. Péndulo invertido en Matlab

En los capítulos anteriores, se ha estudiado como obtener el modelo del péndulo invertido así como el controlador capaz de estabilizarlo. La herramienta que se utilizó entonces para ir realizando las pruebas era Matlab. El motivo de utilizar ésta herramienta y no otra, es básicamente, por las altas prestaciones para el cálculo numérico y la facilidad que Matlab ofrece a la hora de trabajar con matrices. Por este motivo, una vez terminado todo el proceso de diseño, el resultado obtenido se ha almacenado en un fichero 'm' al que se le ha llamado "pendulo.m" (véase Apéndice A).

Al ejecutar el fichero puede observarse como el péndulo va estabilizándose de forma gradual ante una inclinación inicial de once grados. El resultado final es el mostrado en la siguiente gráfica donde el eje x representa la distancia y el eje y la altura (ambos en metros).

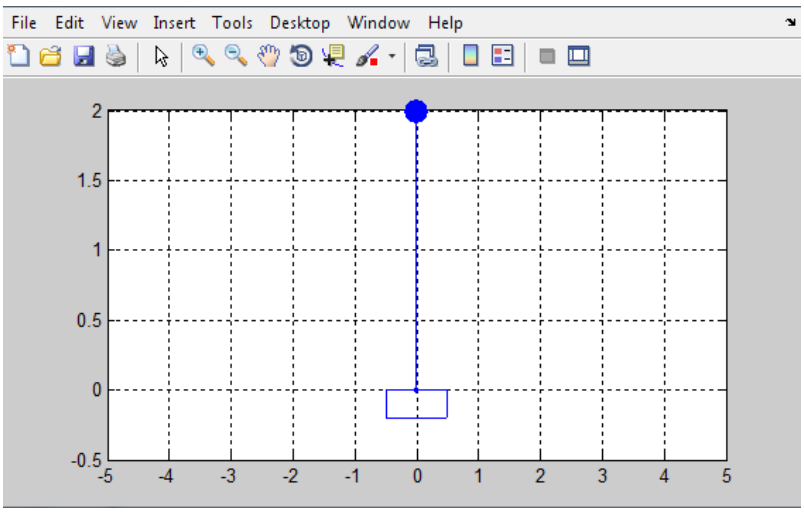


Figura 4.1: Representación péndulo invertido en Matlab

Naturalmente esta representación se puede mejorar, aunque no hay que olvidar que el objetivo es la comprobación del correcto funcionamiento del regulador. No obstante, en internet se pueden encontrar ejemplos de reguladores que muestran al usuario una simulación más completa, donde, no solo se puede comprobar el estado del péndulo sino también la fuerza aplicada al carro así como el ángulo y la posición que éste mantiene en cada momento (véase Apéndice B). Por usar un ejemplo, a continuación se muestra la simulación de un péndulo invertido que parte de inclinación de 15º:

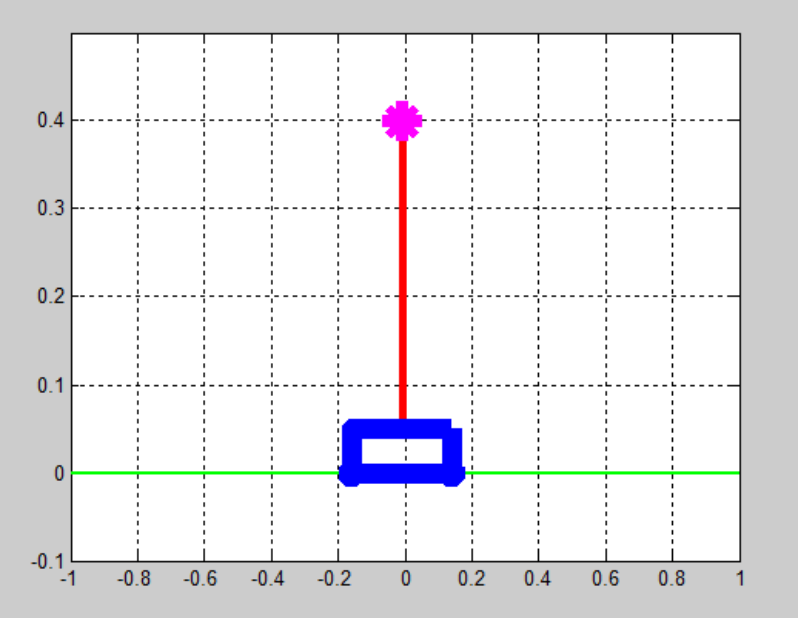


Figura 4.2: Representación alternativa en Matlab

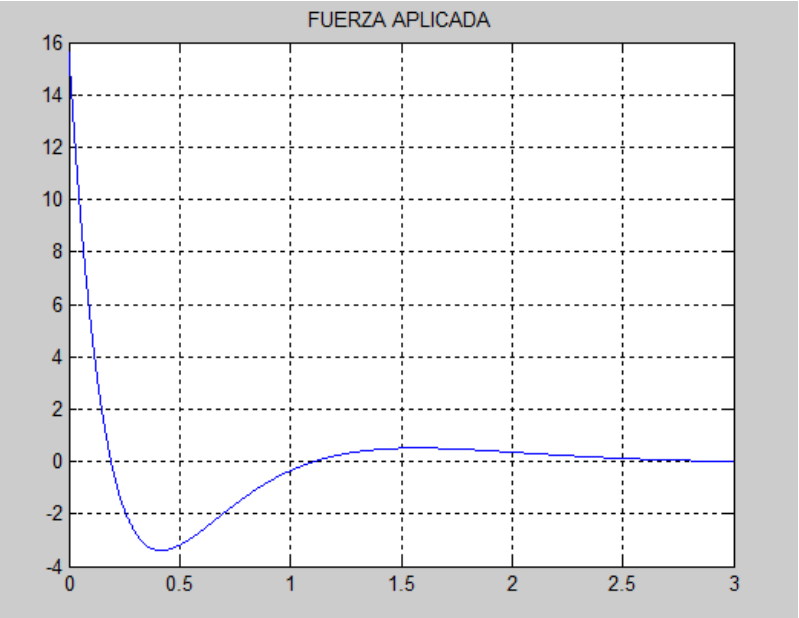


Figura 4.3: Fuerza aplicada al carro

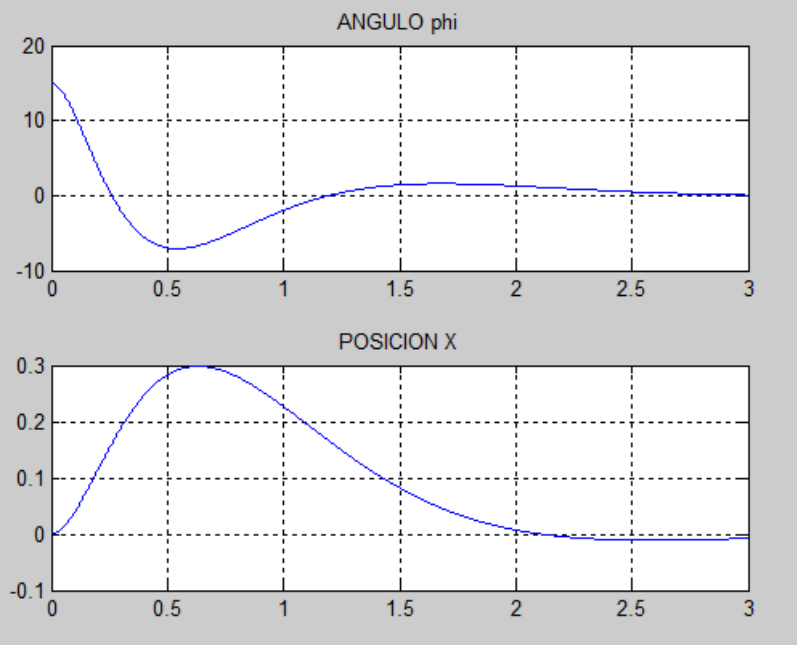


Figura 4.4: Ángulo y posición del carro

4.2. Conversión 'Matlab to Java'

Una vez obtenido el código en Matlab y comprobada su correcta funcionalidad, se ha decidido portarlo a Java. El motivo de esta decisión, es la intención de integrar la aplicación *applet* en un servidor Web. De este modo, se facilita su acceso desde cualquier equipo con acceso a internet y, por otro lado, evita la necesidad de tener un Matlab instalado en la máquina.

Los principales inconvenientes encontrados a la hora de pasar a Java el código con el que se trabajó en Matlab son:

- El tratamiento con matrices.
- Las funciones de control.

El problema para tratar con matrices (sobre todo a la hora de operar con ellas) viene dado porque Java no ofrece herramientras concretas para trabajar con matrices. Para resolver el problema, es necesario acudir a librerías creadas por terceros que implementan con éxito las operaciones básicas con las que se va a trabajar. En el "Capítulo 5. Librerías" se detalla cada una de las librerías utilizadas, así como que funcionalidad tiene cada una de ellas y cuál se ha

adaptado mejor a las necesidades del proyecto.

En lo que respecta a las funciones de control, al igual que el problema de las matrices, no existen en Java funciones ya creadas con las que se pueda trabajar. Esto crea la necesidad de empezar desde cero aquellas funciones que sean necesarias. Para el proyecto en cuestión ha sido necesario crear:

- Función c2d.
- Función expm.
- Función dlqr.

4.2.1. Función c2d

La función "continuous to discrete" (c2d) es necesaria para obtener la discretización del sistema contínuo en espacio de estados. El código fuente puede obtenerse del fichero c2d.m donde principalmente se encuentra:

$function[Phi, Gamma] = c2d(a, b, t)$
$\ldots$
$[m, n] = size(a);$
$[m, nb] = size(b);$
$s = expm([[ab] * t; zeros(nb, n + nb)]);$
$Phi = s(1 : n, 1 : n);$
$Gamma = s(1 : n, n + 1 : n + nb);$

La función recibe como argumentos las matrices A y B del espacio de estados y un tercer argumento t que es el tiempo de muestreo. Como argumentos de salida devuelve dos matrices *Phi* y *Gamma* que son las que se utilizarán posteriormente para el observador y el cálculo del regulador óptimo. La principal complicación de esta función es la obtención de la matriz s resultante de la llamada a *expm*.

4.2.2. Función expm

Es utilizada desde la función c2d. El código fuente puede obtenerse del fichero Matlab expm.m. Echando un vistazo al código, éste puede parecer un poco engorroso, no obstante, la idea principal para resolverlo reside en calcular lo siguiente:

```
EIG(X) and EXPM(X) = V*diag(exp(diag(D)))/V
```

X es la matriz que la función recibe como argumento. V corresponde a los eigenvectores de la matriz X y D es la matriz con los eigenvalues. Matlab cuenta con una función llamada *eig()* que devuelve las dos matrices (V,D) con el resultado. En el caso de Java, esta función viene incorporada en la librería *Jama* de modo que no ha sido necesario implementarla.

4.2.3. Función dlqr

Al igual que anteriores casos, ha sido necesario crearse una función partiendo desde cero. En este caso se trata de la función dlqr, la cuál, en Matlab se encarga de obtener el regulador *lqr* óptimo.

```
function [K,K1]=regulador(G,H,C,D,Peso)
 G1= [ G zeros(4,1); -C* G 1];
H1= [H; -C*H];
Q= [Peso(1) 0 0 0 0; 0 Peso(2) 0 0 0 ; 0 0 Peso(3) 0 0 ; 0 0 0 Peso(4) 0; 0 0 0 0 1]
R= [1];
Pact=diag(0,4);
var=1;
i=0;
while ((abs(var)¿0.0001) && (i¡6000 ))
Pant=Pact;
Pact=Q+G1'*Pant*G1-G1'*Pant*H1*inv(R+H1'*Pant*H1)*H1'*Pant*G1;
Pact-Pant;
var=sum(sum(Pact-Pant));
i=i+1;
end
```

```
P=Pact;
KK=inv(R+H1'*P*H1)*H1'*P*G1;
K=[KK(1) KK(2) KK(3) KK(4)];
K1=-KK(5);
```

4.3. Aplicación Java

4.3.1. Diseño

El *applet* a desarrollar es un pequeño programa hecho en Java que puede incrustarse en un documento HTML lo que permite crear programas que cualquier usuario puede ejecutar desde su navegador. En el diseño del applet, Java proporciona dos paquetes gráficos: AWT y Swing. En la aplicación se ha decidido utilizar Swing por tener mayor cantidad de controles y componentes prefabricados que facilitan el diseño a la vez que enriquecen la interfaz.

Panel parámetros actuales

La funcionalidad de este panel reside en mostrar al usuario los parámetros actuales con los que el péndulo está en ese momento. El valor de cada parámetro es editable lo que permite hacer una simulación bajo distintas situaciones.

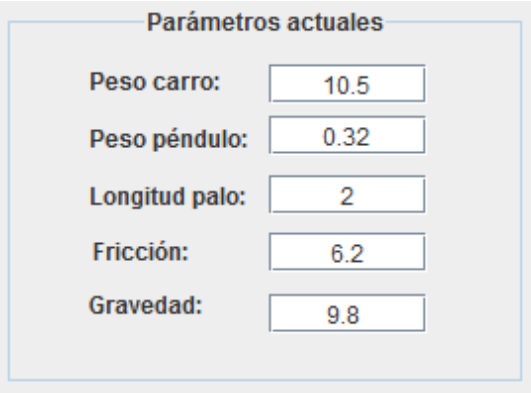


Figura 4.5: Parámetros péndulo

Panel Iniciar/Detener

Sirve para iniciar o detener la simulación del péndulo. Para introducir nuevos valores en el panel de "Parámetros actuales", es necesario detener la simulación e iniciarla de nuevo para que éstos tengan efecto.

Figura 4.6: Panel arranque péndulo

Panel control automático/manual

Formado por dos checkbox donde solo puede estar activo uno al mismo tiempo, su funcionalidad consiste en decidir si el péndulo se ejecuta en modo automático o manual. En modo automático, el péndulo se ejecuta con los parámetros del panel "Parámetros actuales", en modo manual, se controla mediante la barra de scroll. Inicialmente el péndulo aparece estabilizado formando un ángulo de 90º respecto al carro.

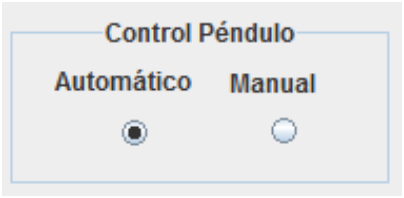


Figura 4.7: Panel control Automático/Manual

Estado péndulo

Consta de dos JTextField que muestran el valor de la posición y ángulo en cada instante.

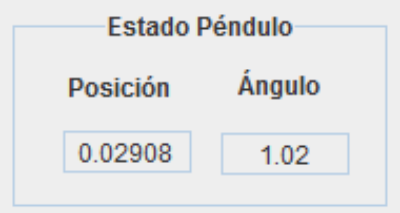


Figura 4.8: Posición y ángulo del péndulo

Panel de dibujo

Fijado en la parte superior del *applet* es la zona donde aparece representado el péndulo. Está compuesto por un JPanel de fondo blanco. La función "paint(Graphics g)", dibuja 3 figuras (cuadrado, línea y círculo) con la siguiente unión:

```
g.fillRect(xx, 250, 70, 30);
g.fillOval((375+Math.sin(radianes)*150),(250-Math.cos(radianes)*150), 20, 20);
g.drawLine((385+Math.sin(radianes)*150),100,xx+35,250);
```

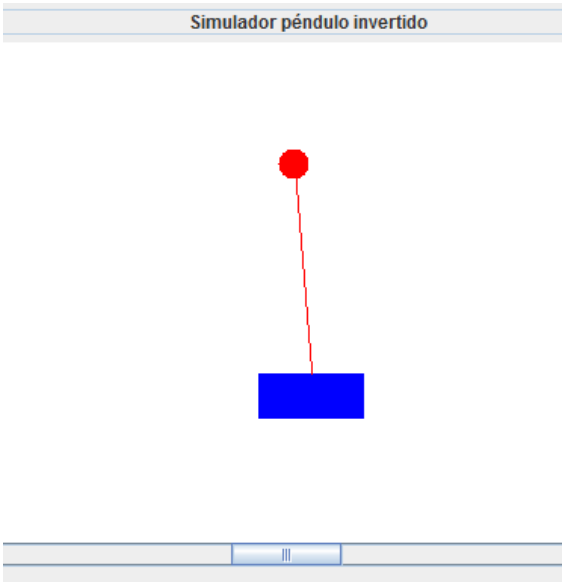


Figura 4.9: Panel de dibujo

Interaz del *applet*

La vista completa de la interfaz queda de la siguiente forma:

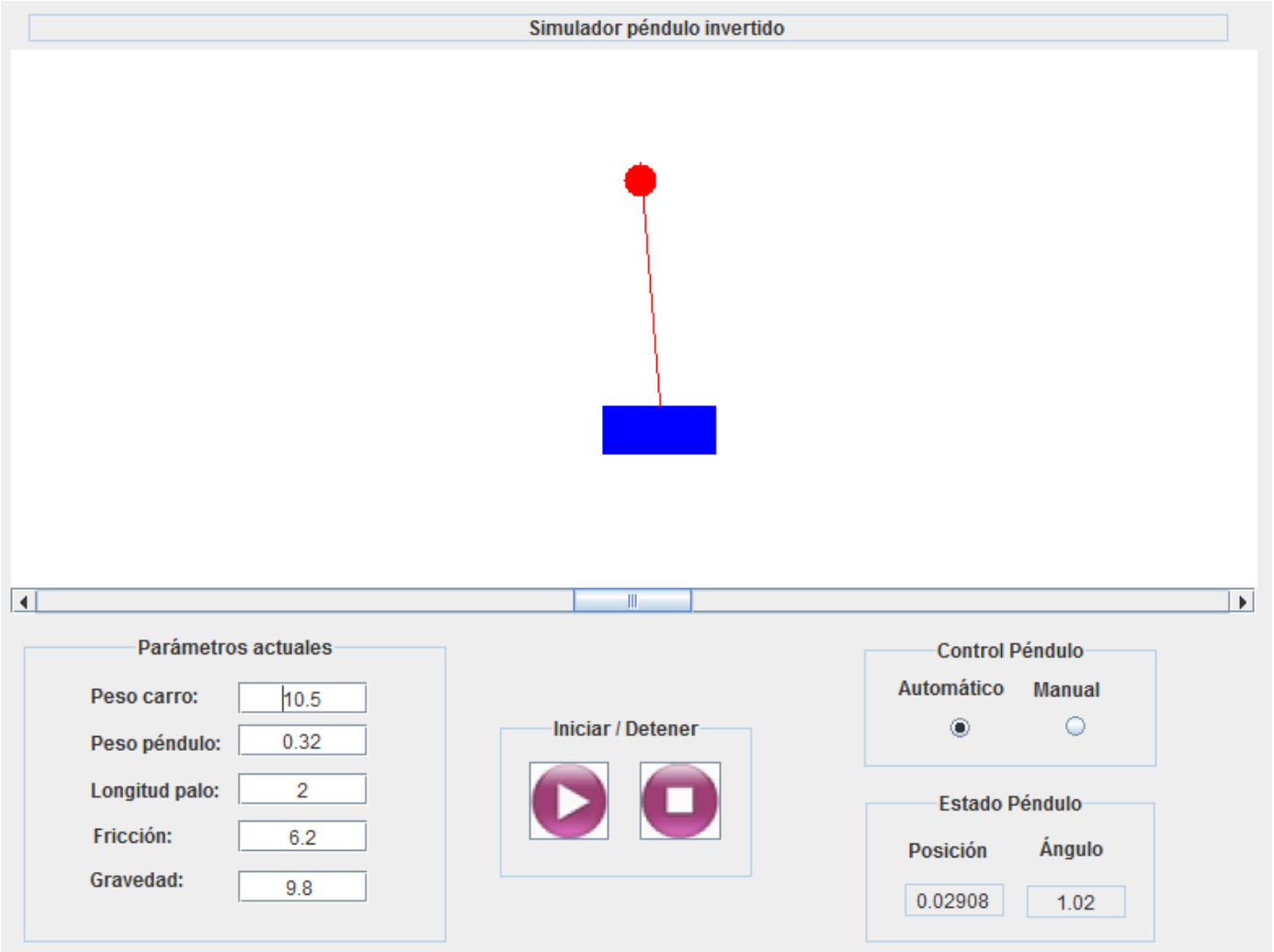


Figura 4.10: Interfaz *applet*

4.3.2. Código

El *applet* se ha creado en un fichero de tipo "Visual Class" llamado "simulador.java"que permite el diseño de la interfaz de forma visual, generando automáticamente el código de los objetos añadidos. La clase extiende de *JApplet* e implementa *Runnable* para el uso de hilos.

Como todo *applet*, éste cuenta con un método init() encargado de inicializar los componentes del mismo. Además, se utiliza el método para establecer un tamaño fijo de la ventana (800x600). Por otro lado, se crea un hilo independiente llamado "Timer" que se encargará de todo el cáculo numérico del péndulo mientras que el hilo principal será el encargado del refresco de imagen.

Con el inicio del hilo "Timer.start()" se lanza el método run() que contiene tanto la discretización del espacio de estados así como la obtención del regulador óptimo. Como última tarea, se ejecuta un bucle "*while*" encargado de obtener la posición y ángulo del péndulo en cada instante de tiempo. Tras cada iteración, el hilo se duerme durante un periodo de 2 milisegundos, tiempo suficiente para que el otro hilo pueda refrescar la imagen con los nuevos valores.

```
public void init() {
    timer = new Thread(this);
    timer.start();
    this.setSize(800, 600);
    this.setContentPane(getJContentPane());
}
@Override
public void paint(Graphics g) {

    super.paint(g);
    //Carrito
    g.setColor(Color.BLUE);
    g.fillRect(xx, 250, 70, 30);

    //Bola pendulo
    g.setColor(Color.RED);
    double radianes = ((Math.PI+ang)/180)*5;
    g.fillOval((int)(350+25+Math.sin(radianes)*150),(int)(250-Math.cos(radianes)*150), 20, 20)

    //Palo
    g.drawLine((int)(350+25+10+Math.sin(radianes)*150),100,xx+35,250);
}
```

```
    public void run()
    {    M=M1+M0;
         theta=thetas+(M1*Math.pow(ls, 2));
         alpha=M1*ls;
         beta=(theta*M)-Math.pow(alpha,2);
         a31=0;
         a32=Math.pow(alpha,2)*g;
         a33=-theta*Fr/beta;
         a34=-alpha*Cc/beta;
         b3=theta/beta;
         a41=0;
         a42=alpha*M*g/beta;
         a43=-alpha*Fr/beta;
         a44=-M*Cc/beta;
         b4=alpha/beta;

         cargaMatrices();
         c2d();
         regulador(phi,gamma,VectorPesoEstado);
         bucle();

         if (u>limitu) u=limitu;
         if (u<-limitu) u=-limitu;

         ang = x.get(1,0);
         repaint();
         try { Thread.sleep(3);
         } catch (InterruptedException ex) {
           System.out.println("Error sleep");
         }
         System.out.println("Posición: " + x.get(0, 0) + " angulo: "+ x.get(1, 0));
     }// while
}//run
```

Capítulo 5

Librerías

En este capítulo se presentan las distintas librerías utilizadas durante el proyecto. En capítulos anteriores, se mencionaba la necesidad de recurrir a éstas librerías para solventar las deficiencias de Java respecto al tratamiento con matrices. En secciones siguientes se describe brevemente en que consiste cada una de ellas. Aunque solo Jama ha sido implantada finalmente en el proyecto, las demás se incluyen porque en un principio se estudió la posibilidad de comunicar el *applet* con Matlab para que este último se encargara del cálculo numérico.

5.1. Librería Jama

Se trata de un paquete básico de álgebra lineal que ofrece clases de nivel de usuario para construir y manipular matrices reales. Tiene como objetivo servir como matriz de la clase estándar para Java. La librería ha sido desarrollada por *The MathWorks* y por el Instituto Nacional de Estándares y Tecnología *"National Institute of Standards and Technology", NIST*.
Jama se compone de seis clases Java:

- CholeskyDecomposition.
- EigenvalueDecomposition.
- LUDecomposition.
- Matrix.
- QRDecomposition.
- SingularValueDecomposition.

En este caso, solo han sido necesarias el uso de las clases “Matrix”, utilizada para el tratamiento de matrices, y la clase “EigenvalueDecomposition”, utilizada para el proceso de discretización del espacio de estados.

La clase Matrix proporciona las operaciones fundamentales del álgebra lineal numérica. Cuenta con varios constructores para crear matrices de doble precisión y coma flotante. Facilita el acceso a submatrices y elementos de la matriz y todas las operaciones aritméticas básicas incluyen la adición y la multiplicación de los elementos de la matriz. En resumen, Jama constituye una buena solución para el tratamiento de matrices.

5.2. Librería Ejml

EJML (*Efficient java Matrix*) es una librería de álgebra lineal para la manipulación de matrices. Sus objetivos de diseño son:

1) Ser computacionalmente lo más eficiente posible, tanto para matrices pequeñas como grandes.
2) Ser accesible tanto para novatos como expertos.

EJML es libre, escrito en Java 100 % y ha sido publicado bajo la licencia LGPL. Consta de tres caraterísticas notables que permiten al programador combinar simplicidad y eficiencia.

1) A través de una interfaz simplificada que facilita el desarrollo.
2) EJML selecciona el mejor algoritmo para cada uso.
3) Cuenta con optimizaciones adicionales para operaciones con matrices pequeñas.

5.3. Librería JMathLib

JMathLib tiene como objetivo ser un clon de Matlab pero completamente escrito en java. Cuenta con una biblioteca de funciones matemáticas diseñada para ser usadas en la evaluación de expresiones complejas y mostrar los resultados en forma gráfica. Se puede utilizar de manera interactiva a través de una ventana tipo terminal o para la interpretación archivos *m* y clases java.

Desde la web de la librería *http://www.jmathlib.de/* es posible utilizar un *applet* que simula de manera exitosa una consola equivalente a la consola de Matlab. La web también documentación detallada y de fácil entendimiento para el usuario sobre la utilización de sus funciones. Pese a ser una librería con gran abanico de operaciones y algoritmos optimizados, no cuenta con la parte más importante que se necesitaba para el proyecto, es decir, la implementación de las funciones de control.

No obstante, es un recurso muy recomendado si la intención es trabajar con imagenes y gráficos en 3D. Pese a que esta librería también trabaja con matrices, se optó por Jama ya que ésta se especializa únicamente en matrices.

Bibliografía

[1] Modelado péndulo invertido: http://www.ib.cnea.gov.ar/~control2/Links/Tutorial_Matlab_esp/invpen.html

[2] Wilson J. Rugh. Linear System Theory. Prentice Hall, 2nd edition, 1995.

[3] Katsuhiko Ogata. Sistemas de control en tiempo discreto, 4nd edición, 1996.

[4] Realimentación de estados y observadores: http://www.eng.newcastle.edu.au/~jhb519/teaching/caut2/clases/Cap8.pdf

[5] Java GUI Builders: http://www.fullspan.com/articles/java-gui-builders.html

[6] Aprenda Java como si estuviera en primero: http://www.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/Java/Java2.pdf

[7] Gráficos Java: http://sunsite.dcc.uchile.cl/java/docs/JavaTut/Cap5/grafico.html

[8] Programación en LaTeX: http://plagatux.es

Apéndices

Apéndice A

Pendulo.m

```
% CONTROL MANUAL Y AUTOMÁTICO DE PÉNDULO INVERTIDO
% Este script ejecuta la simulación de un péndulo mediante lazos de control PID. El control
% automático incorpora un doble lazo incluyendo un lazo con control del ángulo del péndulo
% (para que se sitúe en posición vertical) y otro lazo más lento para hacer que la posición
% de la base se sitúe en un punto dado en el estado de equilibrio.
%
% Fecha: 2006-11-02
% Autor: Ignacio Díaz
% Area de Ingeniería de Sistemas y Automática
% Universidad de Oviedo
%
% Modificado por: Pascual Pérez
% Instituto de Informática Industrial.
% Universidad de Politécnica de Valencia

clear;
close all;
clc;

% M masa del carro 0.5 kg
% m masa del péndulo 0.5 kg
```

```
% b fricción del carro 0.1 N/m/sec
% l longitud al centro de masa del péndulo 0.3 m
% I inercia del péndulo 0.006 kg*m2̂
% u fuerza escalón aplicada al carro
% x coordenadas de posición del carro
% phi ángulo del péndulo respecto de la vertical

M0=10.5;
M1=0.329;
ls=2.0;
thetas=0.008;
Fr=6.2;
C=0.009;
g=9.8;

 % El carro que va con ruedas no puede acelerar más de limitu
limitu=1e6;  % no hay limite

M=M1+M0
theta=thetas+M1*ls*ls
alpha=M1*ls
beta=theta*M-alpha*alpha;
Tm=1/100

 %linealizacion por taylor
a31=0;
a32=alpha*alpha*g;
a33=-theta*Fr/beta;
a34=-alpha*C/beta;
b3=theta/beta;

a41=0;
```

```
a42=alpha*M*g/beta;
a43=-alpha*Fr/beta;
a44=-M*C/beta;
b4=alpha/beta;

Am = [0 0 1 0 ; 0 0 0 1 ; a31 a32 a33 a34 ; a41 a42 a43 a44]
Bm = [ 0 ; 0 ; b3 ; b4]

% Cm nos da los valores del estado que habrá que multiplicarlos por la
% matriz de ganancia para aplicar la nueva u
% se pueden medir las cuatro variables de estado
 Cm = [1 0 0 0 ; 0 1 0 0 ; 0 0 1 0 ; 0 0 0 1];
Dm = [0; 0; 0; 0];   %espacio de estados discreto
 [Fm,Gm,Hm,Jm]=c2dm (Am,Bm,Cm,Dm,Tm,'zoh');

% estado inicial el pendulo un poco inclinado
 x=[0 ; pi/16 ; 0 ;0 ]
u=0;
 % especificar la dinámica que se quiere y calcular la matriz con el
% comando lqr para el sistema discreto
% es controlable y observable el sistema
 co = ctrb (Fm,Gm);
ob = obsv (Fm,Hm);

Controllability = rank (co)
Observability = rank (ob)
  ppos=5000;%factor de peso para la posición del carro
pphi=100;%factor de peso para el ángulo del péndulo

Qm=[ppos 0 0 0; 0 0 0 0; 0 0 pphi 0; 0 0 0 0];
Rm = 1;
Km = dlqr(Fm,Gm,Qm,Rm)
```

```
 % La matriz Cnm establece que factor de entrada se debe reeescalar
 Cnm=[1 0 0 0];
Nbar=rscale(Fm,Gm,Cnm,0,Km)
f = figure(1);
set(f,'pos',[200,600,1000 250]);
u=0;
k = 2;
while ((k<20*1/Tm) && (isempty(get(f,ćurrentchar)=[0])))
if ((k>0)&&(Tm*k<1)) Ref=0;
else Ref=0;
end
k = k + 1;

 % REPRESENTACIÓN GRÁFICA DE LA SIMULACIÓN
% si lo dejamos trabajar no funciona puesto que la linealización es
% para angulos pequeños de la inclinación del péndulo
% como funciona la linealización.
% Lo mejor seria obtener linealizar el sistema en varios puntos de
% trabajo y aplicar ganancias en distintos puntos.
% para verificar el control simular el sistema con las ecuaciones no
% lineales, calcular las matrices de ganancia para diferentes puntos de
% trabajo y aplicarlas al calculo en bucle cerrado.

% Asignamos la entrada
% Ecuación de estados no lineal
 x(1) = x(1) + Tm*x(3);
x(2) = x(2) + Tm*x(4);
sx2=sin(x(2));
cx2=cos(x(2));
aux=(theta*M-alpha*alpha*cx*cx);
x(3) = x(3) + Tm*(alpha*alpha*g*sx2*cx2-Fr*theta*x(3)-alpha*C*cx2*x(4)-alpha*sx2*x(4)*x(4)+theta*u)/aux;
x(4) = x(4) + Tm*(alpha*M*g*sx2-alpha*Fr*cx2*x(3)-M*C*x(4)-alpha*alpha*sx2*cx2*x(4)*x(4)+alpha*cx2*u)/
```

```
 % simular una inclinación del pendulo –¿no funciona –¿limitando la u si
% funciona OK!
% bueno si limitamos mucho la u el sistema no puede controlarse.
% x(2)=0.00000002;
% control del pendulo realimentando la matriz K
% La matriz K se ha calculado sobre el sistema linealizado
 u=Ref*Nbar-Km*x;
 % limitar la referencia por mucho que se quiera acelerar no se puede
 if (u>limitu) u=limitu;
end
if (u<-limitu) u=-limitu;
end
u;
 % Ecuación de salida
% x(1)=pos
% x(2)=theta
 x;
p1 = x(1);
plot(p1,0,'.');
hold on;
th = x(2);
p2 = (x(1)-ls*sin(th))+j*(ls*cos(th));

line([-0.5+p1,+0.5+p1],[-0.2,-0.2]);
line([-0.5+p1,+0.5+p1],[0.0,-0.0]);
line([-0.5+p1,-0.5+p1],[0.0,-0.2]);
line([0.5+p1,0.5+p1],[0.0,-0.2]);
line(real([p1,p2]),imag([p1,p2]));
plot(real(p2),imag(p2),'.','markersize',40);
hold off;

grid on;
```

```
axis([-5 5 -0.5 2]);
drawnow;
end
close all
clear all
```

Apéndice B

Simulación alternativa.m

```
% CONTROL OPTIMAL
% ANIMACION DE PENDULO INVERTIDO
%-------------------------
% Copyright (c) Henry Mendiburu
%-------------------------
 clear all
close all
clc
 % Constantes
 m = 0.05;
M = 1.5;
L = 0.35;
g = 9.8182;

a21 = (M+m)*g/(M*L);
a41 = -m*g/M;
b21 = -1/(M*L);
b41 = 1/M;
 %Espacio Estado
 A = [0 1 0 0; a21 0 0 0; 0 0 0 1; a41 0 0 0];
B = [0; b21; 0; b41];
```

```
 %Control Optimal
 q1 = 100;
q2 = 0;
q3 = 100;
q4 = 0;

Q = diag([q1 q2 q3 q4]);

R = [1];

P = are(A,B*inv(R)*B',Q);

K = inv(R)*B'*P;
 % SIMULACION
 ti = 0;
tf = 3;
dt = 0.001;
k=1;

phi0 = input('Angulo Inicial (grados) : ');

x(1,1) = phi0*pi/180;
x(2,1) = 0;
x(3,1) = 0;
x(4,1) = 0;

for t=ti:dt:tfº
phi = x(1,1);
phip = x(2,1);
F = -K*x;
 %Modelo del pendulo No Lineal
 numx2p = F - m*g*sin(phi)*cos(phi) + m*L*phip*phip*sin(phi);
```

```
denx2p = (M + m*sin(phi)*sin(phi));
x2p = numx2p/denx2p;
phi2p = (g*sin(phi)-x2p*cos(phi))/L;
 %Actualizando valores
 x(1,1) = x(1,1) + x(2,1)*dt;
x(2,1) = x(2,1) + phi2p*dt;
x(3,1) = x(3,1) + x(4,1)*dt;
x(4,1) = x(4,1) + x2p*dt;
 %Guardando Matrices
 ANG(k,1) = x(1,1)*180/pi;
ang(k,1) = x(1,1);
POS(k,1) = x(3,1);
Fuerza(k,1) = F;
T(k,1) = t;
k = k+1;
end

figure(1)
subplot(211)
plot(T,ANG)
title('ANGULO phi')
grid on
subplot(212)
plot(T,POS)
title('POSICION X')
grid on

figure(2)
plot(T,Fuerza)
title('FUERZA APLICADA')
grid on
 % ANIMACION
```

```
 kf = k-1;
a = 0.3;
b = 0.05;
plano = [-1 1 -0.1 0.5];

figure(3)
axis(plano)

for k=1:10:kf
A0X = POS(k,1);
A0Y = b;
A1X = POS(k,1)+a/2;
A1Y = b;
A2X = POS(k,1)-a/2;
A2Y = b;
A3X = POS(k,1)-a/2;
A3Y = 0;
A4X = POS(k,1)+a/2;
A4Y = 0; Xcarro = [A1X A
2X A3X A4X A1X];
Ycarro = [A1Y A2Y A3Y A4Y A1Y];
ApX = POS(k,1) + L*sin(ang(k,1));
ApY = b + L*cos(ang(k,1));
Xpend = [A0X ApX];
Ypend = [A0Y ApY];
As1x = -1;
As1y = 0;
As2x = 1;
As2y = 0;
Xsuelo = [As1x As2x];
Ysuelo = [As1y As2y];
 %GRAFICA
```

```
 clf;
plot(Xsuelo,Ysuelo,'g','linewidth',2)
hold on
plot(Xpend,Ypend,'r','linewidth',4)
hold on
plot(Xcarro,Ycarro,'b','linewidth',10)
hold on
plot(ApX,ApY,'*m','linewidth',20)
hold on
plot(A3X,A3Y,'*b','linewidth',13)
hold on
plot(A4X,A4Y,'*b','linewidth',13)
axis(plano)
grid on
pause(1/2000)

end

disp(' ')
disp('.....FIN.....')
```

Apéndice C

Código fuente *applet*

```
package principal;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.GridBagLayout;
import java.awt.Rectangle;
import javax.swing.BorderFactory;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JApplet;
import javax.swing.JScrollBar;
import javax.swing.JTabbedPane;
import javax.swing.JTextField;
import javax.swing.border.TitledBorder;
import Jama.Matrix;
import javax.swing.JButton;
import java.awt.GridBagConstraints;
import javax.swing.ImageIcon;
import java.awt.Dimension;
import javax.swing.JTextArea;
```

```
import java.awt.Point;
import javax.swing.JRadioButton;
public class principal extends JApplet implements Runnable
private static final long serialVersionUID = 1L;
static double M0 = 10.5; // Masa carro
static double M1 = 0.329; // Masa péndulo
static double ls = 2.0; // Longitud palo en metros
static double thetas = 0.008;
static double Fr = 6.2; // Fricción
static double Cc = 0.009;
static double g = 9.8; // Gravedad
static double limitu = 1000000; // Límite aceleración
static double Tm = 0.01; // Tiempo muestreo
static double M;
static double theta, alpha, beta;
static double a31,a32,a33,a34;
static double a41,a42,a43,a44;
static double b3,b4;
static Matrix A,B,C,D,eigD,eigV;
static Matrix phi, gamma, x;
static double ppos=5000; // factor de peso para la posición del carro
static double pphi=100; // factor de peso para el ángulo del péndulo
static double[] VectorPesoEstado = ppos,0,pphi,0;
static double[][] Km=-234.7,1267,-270.4,714.5;
static double K1=-0.9596;
static double u = 0;
static double[][] v=0;
static double k = 2; // Empezamos en k=2 para tener acceso a dos muestras anteriores
static double ref = 0.0; static int xx = 0;
static double ang;
Thread timer;
private JPanel jContentPane = null;
```

```
private JPanel jPanelPrincipal = null;
private JLabel labelPesoCarro = null;
private JLabel labelPesoPendulo = null;
private JLabel labelLongitudPalo = null;
private JLabel labelFriccion = null;
private JLabel labelGravedad = null;
private JTextField textPesoCarro = null;
private JTextField textPesoPendulo = null;
private JTextField textLongitudPalo = null;
private JTextField textFriccion = null;
private JTextField textGravedad = null;
private JPanel panelSimulador = null;
private JScrollBar scrollPendulo = null;
private JPanel panelIniciarDetener = null;
private JButton btnInicio = null;
private JButton btnParar = null;
private JPanel panelControl = null;
private JLabel labelAutomatico = null;
private JLabel labelManual = null;
private JRadioButton jRadioButtonAutomatico = null;
private JRadioButton jRadioButtonManual = null;
private JPanel panelInfo = null;
private JLabel labelPendulo = null;
private JLabel labelAngulo = null;
private JTextField textPosicion = null;
private JPanel panelTitulo = null;
private JLabel labelTitulo = null;
public principal()
super();
public void init()
timer = new Thread(this);
timer.start();
```

```
this.setSize(800, 600);
this.setContentPane(getJContentPane());
@Override
public void paint(Graphics g)
super.paint(g);
//Carrito
g.setColor(Color.BLUE);
g.fillRect(xx, 250, 70, 30);
//Bola pendulo
g.setColor(Color.RED);
double radianes = ((Math.PI+ang)/180)*5;
g.fillOval((int)(350+25+Math.sin(radianes)*150),(int)(250-Math.cos(radianes)*150), 20, 20);
//Palo
g.drawLine((int)(350+25+10+Math.sin(radianes)*150),100,xx+35,250);
private JPanel getJContentPane()
if (jContentPane == null)
jContentPane = new JPanel();
jContentPane.setLayout(null);
jContentPane.add(getJPanelPrincipal(), null);
jContentPane.add(getPanelSimulador(), null);
jContentPane.add(getScrollPendulo(), null);
jContentPane.add(getPanelIniciarDetener(), null);
jContentPane.add(getPanelControl(), null);
jContentPane.add(getPanelInfo(), null);
jContentPane.add(getPanelTitulo(), null);
return jContentPane;
public void run()
M=M1+M0;
theta=thetas+(M1*Math.pow(ls, 2));
alpha=M1*ls;
beta=(theta*M)-Math.pow(alpha,2);
a31=0;
```

```
a32=Math.pow(alpha,2)*g;
a33=-theta*Fr/beta;
a34=-alpha*Cc/beta;
b3=theta/beta;
a41=0;
a42=alpha*M*g/beta;
a43=-alpha*Fr/beta;
a44=-M*Cc/beta;
b4=alpha/beta;
cargaMatrices();
c2d();
// estado inicial el pendulo un poco inclinado
double[][] tempX = 0,Math.PI/16,0,0;
x = new Matrix(tempX);
regulador(phi,gamma,VectorPesoEstado);
while (k¡20*1/Tm)
k++;
/*
x(1) = x(1) + Tm*x(3);
x(2) = x(2) + Tm*x(4);
sx2=sin(x(2));
cx2=cos(x(2));
aux=(theta*M-alpha2*cx22);
x(3)=x(3)+Tm*(alpha2*g*sx2*cx2-Fr*theta*x(3)-alpha*C*cx2*x(4)-alpha*sx2*x(4)2+theta*u)/aux;
x(4)=x(4)+Tm*(alpha*M*g*sx2-alpha*Fr*cx2*x(3)-M*C*x(4)-alpha2*sx2*cx2*x(4)2+alpha*cx2*u)/aux;*
double[][] tmp3 = x.getArray();
tmp3[0][0] = tmp3[0][0] + Tm*tmp3[2][0];
tmp3[1][0] = tmp3[1][0] + Tm*tmp3[3][0];
double sx2 = Math.sin(tmp3[1][0]);
double cx2 = Math.cos(tmp3[1][0]);
double var1= theta*M-Math.pow(alpha,2)*Math.pow(cx2,2);
```

```
tmp3[2][0] = tmp3[2][0] + Tm*(Math.pow(alpha,2)*g*sx2*cx2-Fr*theta*tmp3[2][0]-alpha*Cc*cx2*tmp3[3][0]-
alpha*sx2*Math.pow(tmp3[3][0],2)+theta*u)/var1;
tmp3[3][0] = tmp3[3][0] + Tm*(alpha*M*g*sx2-alpha*Fr*cx2*tmp3[2][0]-M*Cc*tmp3[3][0]-
Math.pow(alpha,2)*sx2*cx2*Math.pow(tmp3[3][0],2)+alpha*cx2*u)/var1;
x = new Matrix(tmp3);
//y=Hm*x;
double[][] hm = 1,0,0,0;
Matrix Hm = new Matrix(hm);
double[][] y = multiplicarMatrices(Hm.getArray(),x.getArray());
//v=v+Ref-y;
v = restarMatrices(sumaMatrices(new Matrix(v),ref),y);
//u=-Km*x+K1*v;
//u = multiplicarMatrices(Km,-1);
if (u¿limitu) u=limitu;
if (u¡-limitu) u=-limitu;
// x(1) pos
// x(2) theta
xx = (int) (x.get(0, 0)*1000+350);
ang = x.get(1,0);
System.out.println("Valor xx: - xx);
repaint();
try
Thread.sleep(3);
catch (InterruptedException ex)
System.out.println(.Error sleep");
System.out.println("Posición: - x.get(0, 0) + .ángulo: - x.get(1, 0));
// while
//run
// Da valores a las matrices A,B,C y D
private static void cargaMatrices()
double[][] tempA = 0,0,1,0,0,0,0,1,a31,a32,a33,a34,a41,a42,a43,a44;
A = new Matrix(tempA);
```

```
double[][] tempB = 0,0,b3,b4;
B = new Matrix(tempB);
double[][] tempC = 1,0,0,0;
C = new Matrix(tempC);
double[][] tempD = 0;
D = new Matrix(tempD);
private static double[][] multiplicarMatrices(Matrix a, double T)
double[][] temp = a.getArray();
for(int i=0;i¡a.getRowDimension();i++)
for (int j=0;j¡a.getColumnDimension();j++)
temp[i][j]=temp[i][j]*T;
return temp;
private static double[][] multiplicarMatrices(double A[][], double B[][])
Matrix a = new Matrix(A);
Matrix b = new Matrix(B);
double[][] resul = new double[a.getRowDimension()][b.getColumnDimension()];
for(int i = 0; i ¡a.getRowDimension(); i++)
for(int j = 0; j ¡b.getColumnDimension(); j++)
for(int k = 0; k ¡a.getColumnDimension(); k++)
resul[i][j] += A[i][k] * B[k][j];
return resul;
private static double[][] sumaMatrices(Matrix a, double b)
double[][] resul = new double[a.getRowDimension()][a.getColumnDimension()];
for(int i=0;i¡a.getRowDimension();i++)
for (int j=0;j¡a.getColumnDimension();j++)
resul[i][j]=a.get(i,j)+b;
return resul;
private static double[][] restarMatrices(double[][] A, double[][] B)
Matrix a = new Matrix(A);
Matrix b = new Matrix(B);
double[][] resul = new double[a.getRowDimension()][a.getColumnDimension()];
for(int i=0;i¡a.getRowDimension();i++)
```

```
for (int j=0;j¡a.getColumnDimension();j++)
resul[i][j]=a.get(i,j)-b.get(i, j);
return resul;
private JPanel getJPanelPrincipal()
if (jPanelPrincipal == null)
labelGravedad = new JLabel();
labelGravedad.setBounds(new Rectangle(43, 143, 62, 16));
labelGravedad.setText("Gravedad:");
labelFriccion = new JLabel();
labelFriccion.setBounds(new Rectangle(45, 116, 54, 16));
labelFriccion.setText("Fricción:");
labelLongitudPalo = new JLabel();
labelLongitudPalo.setBounds(new Rectangle(43, 88, 83, 16));
labelLongitudPalo.setText("Longitud palo:");
labelPesoPendulo = new JLabel();
labelPesoPendulo.setBounds(new Rectangle(43, 59, 81, 16));
labelPesoPendulo.setText("Peso péndulo:");
labelPesoCarro = new JLabel();
labelPesoCarro.setBounds(new Rectangle(43, 30, 69, 16));
labelPesoCarro.setText("Peso carro: ");
jPanelPrincipal = new JPanel();
jPanelPrincipal.setLayout(null);
jPanelPrincipal.setBounds(new Rectangle(21, 390, 264, 192));
jPanelPrincipal.setBorder(BorderFactory.createTitledBorder(null, "Parámetros actuales", Ti-
tledBorder.CENTER, TitledBorder.DEFAULT_POSITION, newFont("Dialog", Font.BOLD, 12), newColor(51,
jPanelPrincipal.add(labelPesoCarro, null);
jPanelPrincipal.add(labelPesoPendulo, null);
jPanelPrincipal.add(labelLongitudPalo, null);
jPanelPrincipal.add(labelFriccion, null);
jPanelPrincipal.add(labelGravedad, null);
jPanelPrincipal.add(getTextPesoCarro(), null);
jPanelPrincipal.add(getTextPesoPendulo(), null);
```

```
jPanelPrincipal.add(getTextLongitudPalo(), null);
jPanelPrincipal.add(getTextFriccion(), null);
jPanelPrincipal.add(getTextGravedad(), null);
return jPanelPrincipal;
private JTextField getTextPesoCarro()
if (textPesoCarro == null)
textPesoCarro = new JTextField();
textPesoCarro.setBounds(new Rectangle(134, 30, 80, 20));
textPesoCarro.setText("10.5");
textPesoCarro.setHorizontalAlignment(JTextField.CENTER);
textPesoCarro.setEditable(true);
return textPesoCarro;
private JTextField getTextPesoPendulo()
if (textPesoPendulo == null)
textPesoPendulo = new JTextField();
textPesoPendulo.setBounds(new Rectangle(134, 56, 80, 20));
textPesoPendulo.setText("0.32");
textPesoPendulo.setHorizontalAlignment(JTextField.CENTER);
textPesoPendulo.setEditable(true);
return textPesoPendulo;
private JTextField getTextLongitudPalo()
if (textLongitudPalo == null)
textLongitudPalo = new JTextField();
textLongitudPalo.setBounds(new Rectangle(134, 86, 80, 20));
textLongitudPalo.setText("2");
textLongitudPalo.setHorizontalAlignment(JTextField.CENTER);
textLongitudPalo.setEditable(true);
return textLongitudPalo;
private JTextField getTextFriccion()
if (textFriccion == null)
textFriccion = new JTextField();
textFriccion.setBounds(new Rectangle(134, 115, 80, 20));
```

```
textFriccion.setText("6.2");
textFriccion.setHorizontalAlignment(JTextField.CENTER);
textFriccion.setEditable(true);
return textFriccion;
private JTextField getTextGravedad()
if (textGravedad == null)
textGravedad = new JTextField();
textGravedad.setBounds(new Rectangle(134, 146, 80, 20));
textGravedad.setText("9.8 ");
textGravedad.setHorizontalAlignment(JTextField.CENTER);
textGravedad.setEditable(true);
return textGravedad;
private JPanel getPanelSimulador()
if (panelSimulador == null)
panelSimulador = new JPanel();
panelSimulador.setLayout(new GridBagLayout());
panelSimulador.setBounds(new Rectangle(15, 31, 767, 331));
panelSimulador.setBackground(Color.white);
return panelSimulador;  private JScrollBar getScrollPendulo()
if (scrollPendulo == null)
scrollPendulo = new JScrollBar();
scrollPendulo.setOrientation(JScrollBar.HORIZONTAL);
scrollPendulo.setLocation(new Point(15, 362));
scrollPendulo.setValue(45);
scrollPendulo.setSize(new Dimension(766, 16));
return scrollPendulo;
private JPanel getPanelIniciarDetener()
if (panelIniciarDetener == null)
panelIniciarDetener = new JPanel();
panelIniciarDetener.setLayout(null);
panelIniciarDetener.setBounds(new Rectangle(314, 440, 159, 102));
```

panelIniciarDetener.setBorder(BorderFactory.createTitledBorder(null, Ïniciar / Detener", TitledBorder.CENTER, TitledBorder.DEFAULT$_POSITION, newFont("Dialog", Font.BOLD, 12), newCol

panelIniciarDetener.add(getBtnInicio(), null);

panelIniciarDetener.add(getBtnParar(), null);

return panelIniciarDetener;

private JButton getBtnInicio()

if (btnInicio == null)

btnInicio = new JButton();

btnInicio.setBounds(new Rectangle(20, 29, 49, 48));

btnInicio.setIcon(new ImageIcon("D:/home/jose/Desktop/eclipse/workspace/aplicacionPFC/img/inicia

return btnInicio;

private JButton getBtnParar()

if (btnParar == null)

btnParar = new JButton();

btnParar.setBounds(new Rectangle(88, 29, 50, 48));

btnParar.setIcon(new ImageIcon("D:/home/jose/Desktop/eclipse/workspace/aplicacionPFC/img/parar

btnParar.setPreferredSize(new Dimension(97, 73));

return btnParar;

private JPanel getPanelControl()

if (panelControl == null)

labelManual = new JLabel();

labelManual.setBounds(new Rectangle(106, 24, 56, 16));

labelManual.setText("Manual");

labelAutomatico = new JLabel();

labelAutomatico.setBounds(new Rectangle(23, 23, 70, 16));

labelAutomatico.setText(.Automático");

panelControl = new JPanel();

panelControl.setLayout(null);

panelControl.setBounds(new Rectangle(538, 392, 184, 82));

panelControl.setBorder(BorderFactory.createTitledBorder(null, Çontrol Péndulo", TitledBorder.CENTER, TitledBorder.DEFAULT$_POSITION, newFont("Dialog", Font.BOLD, 12), newColor(51, 5

panelControl.add(labelAutomatico, null);

panelControl.add(labelManual, null);

panelControl.add(getJRadioButtonAutomatico(), null);

panelControl.add(getJRadioButtonManual(), null);

return panelControl;

private JRadioButton getJRadioButtonAutomatico()

if (jRadioButtonAutomatico == null)

jRadioButtonAutomatico = new JRadioButton();

jRadioButtonAutomatico.setBounds(new Rectangle(51, 46, 21, 21));

jRadioButtonAutomatico.setSelected(true);

return jRadioButtonAutomatico;

private JRadioButton getJRadioButtonManual()

if (jRadioButtonManual == null)

jRadioButtonManual = new JRadioButton();

jRadioButtonManual.setBounds(new Rectangle(122, 45, 21, 21));

return jRadioButtonManual;

private JPanel getPanelInfo()

if (panelInfo == null)

labelAngulo = new JLabel();

labelAngulo.setBounds(new Rectangle(109, 28, 50, 16));

labelAngulo.setText("Ángulo");

labelPendulo = new JLabel();

labelPendulo.setBounds(new Rectangle(28, 29, 57, 16));

labelPendulo.setText("Posición");

panelInfo = new JPanel();

panelInfo.setLayout(null);

panelInfo.setBounds(new Rectangle(539, 486, 182, 96));

panelInfo.setBorder(BorderFactory.createTitledBorder(null, .Estado Péndulo", TitledBorder.CENTER, TitledBorder.DEFAULT$_{P}OSITION, newFont("Dialog", Font.BOLD, 12), newColor(51, 51, 51)$));

panelInfo.add(labelPendulo, null);

panelInfo.add(labelAngulo, null);

panelInfo.add(getTextPosicion(), null);

panelInfo.add(getTextAngulo(), null);

```
return panelInfo;
private JTextField getTextPosicion()
if (textPosicion == null)
textPosicion = new JTextField();
textPosicion.setBounds(new Rectangle(26, 58, 61, 20));
textPosicion.setText("0.02908");
textPosicion.setHorizontalAlignment(JTextField.CENTER);
textPosicion.setEditable(false);
return textPosicion;
private JTextField getTextAngulo()
if (textAngulo == null)
textAngulo = new JTextField();
textAngulo.setBounds(new Rectangle(101, 59, 61, 20));
textAngulo.setHorizontalAlignment(JTextField.CENTER);
textAngulo.setText("1.02");
textAngulo.setEditable(false);
return textAngulo;
private JPanel getPanelTitulo()
if (panelTitulo == null)
labelTitulo = new JLabel();
labelTitulo.setBounds(new Rectangle(307, 0, 174, 16));
labelTitulo.setText("Simulador péndulo invertido");
panelTitulo = new JPanel();
panelTitulo.setLayout(null);
panelTitulo.setBounds(new Rectangle(26, 9, 738, 17));
panelTitulo.setBorder(BorderFactory.createTitledBorder(null, , TitledBorder.DEFAULT_JUSTIFICAT
panelTitulo.add(labelTitulo, null);
return panelTitulo;
//class
```

Apéndice D

Regulador

```
function [K,K1]=regulador(G,H,Peso)
C= [ 1 0 0 0];
D=[0];
G1= [ G zeros(4,1); -C* G 1]
H1= [H; -C*H];
Q= [Peso(1) 0 0 0 0; 0 Peso(2) 0 0 0 ; 0 0 Peso(3) 0 0 ; 0 0 0 Peso(4) 0; 0 0 0 0 1];
R= [1];
Pact=diag(0,4);
var=1;
i=0;
while ((abs(var)¿0.0001) && (i¡6000 ))
Pant=Pact;
Pact=Q+G1'*Pant*G1-G1'*Pant*H1*inv(R+H1'*Pant*H1)*H1'*Pant*G1;
Pact-Pant;
var=sum(sum(Pact-Pant));
i=i+1;
end
P=Pact;
KK=inv(R+H1'*P*H1)*H1'*P*G1
K=[KK(1) KK(2) KK(3) KK(4)];
K1=-KK(5);
```