Proyecto Fin de Máster
Máster en Ingeniería Industrial

Control avanzado de un robot móvil auto-equilibrado

Autor: Cecilia González González

Tutor: Ignacio Alvarado Aldea

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017/2018

Departamento de Teoría de la Señal y Comunicaciones

Proyecto Fin de Máster
Máster en Ingeniería Industrial

Control avanzado de un robot móvil autoequilibrado

Autor:
Cecilia González González

Tutor:
Ignacio Alvarado Aldea
Ignacio Alvarado Aldea

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017/2018

Proyecto Fin de Máster: Control avanzado de un robot móvil autoequilibrado

Autor: Cecilia González González
Tutor: Ignacio Alvarado Aldea

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Dedico este trabajo a mi familia. A mis pacientes padres y a mi hermana que siempre sabe escucharme y ayudarme. También quiero dedicar este proyecto a Pablo Krupa, que ha estado siempre resolviendo mis dudas. A mi tutor Ignacio, le agradezco el reto que suponía este trabajo. A mis amigos, que son los que mejor saben lo difícil que puede ser sacar adelante el trabajo de fin de máster compaginándolo con el trabajo.

Tengo la suerte de haber podido contar con todos ellos, que conmigo han conseguido que este proyecto salga. Gracias.

Cecilia González González

Sevilla, 2018

Resumen

Este proyecto consiste en la descripción del Control Predictivo basado en Modelo y su implementación en un sistema real. En este caso, el sistema será un vehículo autoequilibrado.

La implementación del MPC en el sistema radica en el desarrollo de este tipo de control en Arduino. Para lo cual es necesario el desarrollo inicial en Maltab, donde se realizan las simulaciones para comprobar el buen funcionamiento del bucle de control.

Abstract

This project consists in the description of a Model predictive Control and its implementation in a real system. In this case, the system will be a self-balancing vehicle.

The implementation of the MPC in the system lies in the development of this type of control in Arduino. For which it is necessary the initial development in Maltab, where the simulations are carried out to verify the good functioning of the control loop.

Índice Abreviado

Índice

Notación

$\mathbb{R}$	Cuerpo de los números reales
$\mathbb{Z}$	Cuerpo de los números complejos
$\|\|\mathbf{v}\|\|$	Norma del vector $\mathbf{v}$
$s.a$	Sujeto a
sin	Función seno
cos	Función coseno
$\frac{\partial y}{\partial x}$	Derivada parcial de y respecto a x
$\leqslant$	Menor o igual
$\geqslant$	Mayor o igual
$\simeq$	Aproximado
$\equiv$	Equivalente a
Δ	Incremento
$\in$	Pertenece a
$\subset$	Contenido en
A^T	Matriz traspuesta
$\dot{x}$	Primera derivada de x
$\ddot{x}$	Segunda derivada de x
$\frac{d}{dt}$	Derivada en el tiempo
x_{eq}	x en el punto de equilibrio
x_{ek}	x en el instante k
$\hat{x}$	x estimado
x^*	x óptimo
rad	Radianes
s	Segundos
kg	Kilogramo
m	Metro
arg	Argumento
mín	Minimización
$\sum_{i=0}^{N-1}$	Sumatorio desde $i = 0$ hasta $N - 1$
MPC	Control Predictivo por Modelo
FISTA	Fast Iterative Shrinkage Thresholding Algorithm
SSTO	Steady-State Target Optimization
OBS	Estimador de estados
QP	Programación cuadrática

1 Introducción

Derrotados son los que dejan de luchar

José Mujica

Muchos problemas de control y automatización de procesos industriales son resueltos con control predictivo. Esta estrategia es usada en sistemas que constan de varias variables, con un comportamiento dinámico complejo o inestable. Este tipo de control pretende predecir el comportamiento futuro de la planta a través del conocimiento del modelo matemático del proceso que se desea controlar.

El control predictivo se compone de diversas disciplinas, entre ellas, el control predictivo basado en modelo (MPC). El MPC se caracteriza por buscar una actuación que responda a la optimización de una función de coste. Esta está relacionada con el comportamiento futuro del sistema. Para poder calcular la acción del control hace falta resolver un problema de optimización (cuadrática). Existen muchos algoritmos que resuelven este problema. Sin embargo, en este proyecto el algoritmo que se va a desarrollar es el algoritmo FISTA (Fast Iterative Shrinkage Thresholding Algorithm). Se quiere implementar este algoritmo ya que el sistema que se usa es de bajos recursos.

Entre las ventajas del control predictivo, este permite formular en el dominio del tiempo, siendo una técnica más intuitiva. Por otra parte, puede tratar con sistemas multivariables y no lineales. Además permite que la ley de control garantice criterios de optimización. Por último, admite la integración de restricciones.

Como inconvenientes cabe destacar que requiere un coste computacional alto, debido a la gran cantidad de datos necesarios para los cálculos. Esto complica la aplicación de este tipo de control en sistemas que requieren una rápida actuación como es este caso. Por otra parte, el cumplimiento de las restricciones hace compleja la búsqueda de la solución. Además estas restricciones aparecen tanto en la acción de control como en las variables de estado.

Será de gran importancia tener en cuenta la restricción de tiempo del sistema. Esto es así ya que se está implementando el control en un sistema real que requiere una respuesta en un tiempo determinado que no puede sobrepasarse.

2 Descripción del sistema y modelo

2.1 Descripción del sistema

El Segway es un sistema a controlar muy estudiado, es el péndulo invertido. Este sistema es interesante por ser lineal, multivariable y contener un cero de fase no mínima.

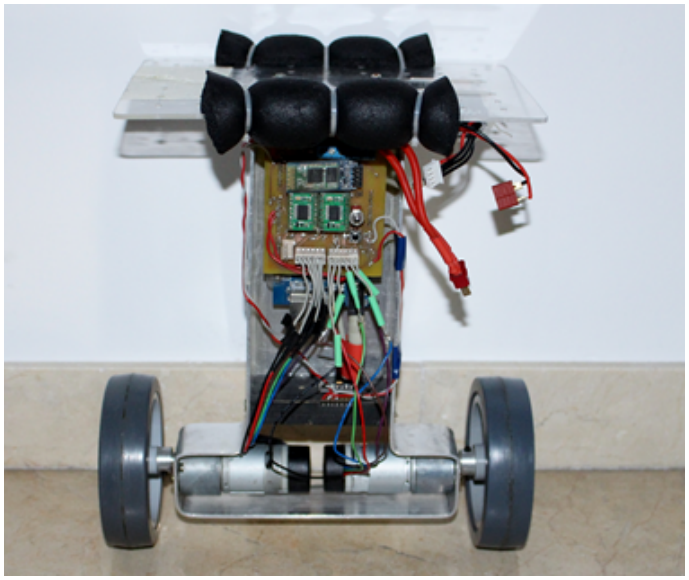


Figura 2.1 Vehículo autoequilibrado.

Se sabe que este tipo de sistema contiene dos posiciones de equilibrio, una de ellas estable correspondiente al punto de mínima altura del péndulo y la inestable, que se encuentra en el punto de máxima altura. Ya que se habla de péndulo invertido, la posición que se desea controlar es la de equilibrio inestable. Este equilibrio se conseguirá en momentos puntuales. Porque el sistema tiende a no permanecer en él debido a perturbaciones o ruidos en las señales de entrada. Por otra parte, el sistema contiene perturbaciones, lo que hace más sencillo la pérdida de equilibrio. Particularmente, este Segway tiene una perturbación provocada por la diferencia entre el punto geométrico del mismo y el centro de gravedad. Esto provoca que el sistema tienda a inclinarse hacia el centro de gravedad como se puede observar en la Figura 2.2.

Como consecuencia, es indispensable un sistema de control que garantice la permanencia del vehículo autoequilibrado en su punto de equilibrio o en su defecto en la posición más cercana al punto de equilibrio.

2.2 Modelado

En este apartado se pretende establecer un modelo físico-matemático del sistema. Por lo tanto se realiza una explicación del vehículo autoequilibrado con el uso de ecuaciones de su comportamiento dinámico. Este modelo será el usado en las simulaciones del sistema.

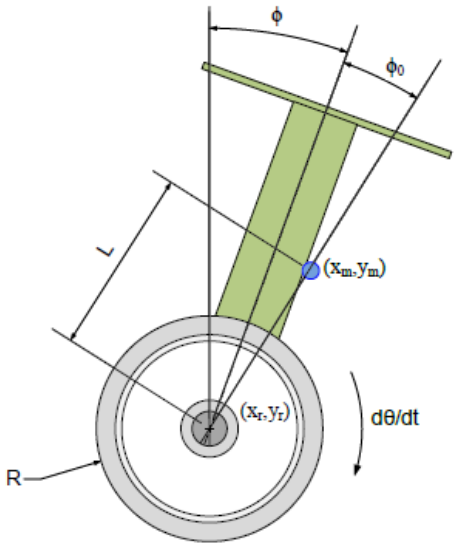


Figura 2.2 Esquema del Segway.

2.2.1 Modelo no lineal

El modelo no lineal es el modelo real del vehículo autoequilibrado. Para el desarrollo del mismo se hará uso de los parámetros que aparecen en la tabla 2.1. Para poder observar algunos de estos parámetros, se muestran representados en la Figura 2.2.

Tabla 2.1 Parámetros físicos del sistema.

Parámetro	Descripción	Valor
m_r	Masa de una rueda	0.14 kg
R	Radio de las ruedas	0.05 m
M	Masa del vehículo sin ruedas	0.82 kg
L	Distancia del eje de las ruedas al CG	0.098 m
g	Aceleración gravitatoria	9.81 m/s^2

También será necesaria la definición de los momentos de inercia del sistema, cuyas fórmulas y definición se muestra a continuación:

$$J_r = \frac{1}{2} m_r R^2$$
$$J_\phi = ML^2$$

En el desarrollo del modelo no se usarán estos parámetros sino los definidos en la tabla 2.1.

La descripción dinámica del sistema estará basada en las ecuaciones de Lagrange. Se define la Lagrangiana

Tabla 2.2 Momentos de inercia.

Parámetro	Descripción	Valor
J_r	Momento de inercia de una rueda	$1.75 \cdot 10^{-4}\ kg \cdot m^2$
J_ϕ	Momento de inercia del vehículo sin las ruedas	$7.875 \cdot 10^{-3}\ kg \cdot m^2$

como la energía cinética del sistema menos la energía potencial, siendo la energía cinética: la traslacional más la rotacional.

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = Q_i$$
$$L = T_{tras} + T_{rot} - V$$

Las coordenadas generalizadas elegidas son:

Tabla 2.3 Coordenadas generalizadas.

Coordenada	Descripción
ϕ	Ángulo de inclinación del segway
θ	Ángulo que se da con el giro de la rueda con respecto al plano vertical

Primero se realizará la definición de cada una de las energías para el cálculo de la Lagrangiana. Se desarrolla inicialmente la fórmula de la energía cinética de traslación.

$$T_{tras} = 2 \cdot \frac{1}{2} m_r (\dot{x}_r^2 + \dot{y}_r^2) + \frac{1}{2} M (\dot{x}_m^2 + \dot{y}_m^2) \tag{2.1}$$

donde las velocidades linales del centro de gravedad y de las ruedas se definen como:

$$x_r = R\theta \qquad \dot{x}_r = R\dot{\theta} \tag{2.2}$$
$$y_r = R \qquad \dot{y}_r = 0 \tag{2.3}$$
$$x_m = R\theta + L\sin(\phi + \phi_0) \qquad \dot{x}_m = R\dot{\theta} + L\dot{\phi}\cos(\phi + \phi_0) \tag{2.4}$$
$$y_m = R + L\cos(\phi + \phi_0) \qquad \dot{y}_m = -L\dot{\phi}\sin(\phi + \phi_0) \tag{2.5}$$

Sustituyendo estas expresiones en (2.1), desarrollando y agrupando términos se obtiene:

$$T_{tras} = (m_r + \frac{1}{2}M)R^2\dot{\theta}^2 + \frac{1}{2}ML^2\dot{\phi}^2 + RLM\dot{\theta}\dot{\phi}\cos(\phi + \phi_0) \tag{2.6}$$

Seguidamente se determina la energía cinética rotacional.

$$T_{rot} = 2 \cdot \frac{1}{2} J_r \dot{\theta}^2 + \frac{1}{2} J_\phi \dot{\phi}^2 \tag{2.7}$$

Sustituyendo J_r y J_ϕ, referidos a los momentos de inercia de la rueda y del vehículo respectivamente, por los parámetros de la tabla 2.1, la energía rotacional quedará expresada como:

$$T_{rot} = \frac{1}{2} m_r R^2 \dot{\theta}^2 + \frac{1}{2} M L^2 \dot{\phi}^2 \tag{2.8}$$

Por último, se precisa la ecuación de la energía potencial.

$$V = MgL\cos(\phi + \phi_0) \tag{2.9}$$

Teniendo estas tres energías formuladas, se puede definir la Lagrangiana.

$$L = (m_r + \frac{1}{2}M)R^2\dot{\theta}^2 + \frac{1}{2}m_r R^2\dot{\theta}^2 + ML^2\dot{\phi}^2 + RLM\dot{\theta}\dot{\phi}\cos(\phi+\phi_0) - MgL\cos(\phi+\phi_0) \tag{2.10}$$

Se definirá el par motor externo aplicado a partir de las ecuaciones de Lagrange de cada una de las coordenadas lagrangianas, es decir ϕ y θ. El signo del par motor dependerá de la coordenada lagrangiana. Es decir, el signo depende de la influencia del par respecto a la coordenada lagrangiana. En el caso del ángulo de giro de la rueda, θ, se pondrá como positivo. Esto es así porque θ aumenta en el sentido del movimiento. Sin embargo, el ángulo de inclinación tendrá un par motor externo negativo, ya que se busca que el ángulo de inclinación disminuya. Lo que hace que el Segway busque el punto de mayor altura.

$$-T = \frac{d}{dt}\frac{\partial L}{\partial \dot{\phi}} - \frac{\partial L}{\partial \phi} \tag{2.11}$$

$$T = \frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} - \frac{\partial L}{\partial \theta} \tag{2.12}$$

Se expresa el resultado de cada uno de los términos de las ecuaciones de Lagrange para la coordenada ϕ.

$$\frac{\partial L}{\partial \dot{\phi}} = 2ML^2\dot{\phi} + MRL\dot{\theta}\cos(\phi+\phi_0) = 2b\dot{\phi} + c\dot{\theta}\cos(\phi+\phi_0) \tag{2.13}$$

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\phi}} = 2ML^2\ddot{\phi} + MRL\ddot{\theta}\cos(\phi+\phi_0) - MRL\dot{\theta}\dot{\phi}\sin(\phi+\phi_0) = 2b\ddot{\phi} + c\ddot{\theta}\cos(\phi+\phi_0) - c\dot{\theta}\dot{\phi}\sin(\phi+\phi_0) \tag{2.14}$$

$$\frac{\partial L}{\partial \phi} = -MRL\dot{\theta}\dot{\phi}\sin(\phi+\phi_0) + MgL\sin(\phi+\phi_0) = -c\dot{\theta}\dot{\phi}^2\sin(\phi+\phi_0) + d\dot{\phi}\sin(\phi+\phi_0) \tag{2.15}$$

Se declaran los términos para la coordenada θ de las ecuaciones de Lagrange.

$$\frac{\partial L}{\partial \dot{\theta}} = 2(m_r + \frac{1}{2}M)R^2\dot{\theta} + m_r R^2\dot{\theta} + MRL\dot{\phi}\cos(\phi+\phi_0) = 2a\dot{\theta} + c\dot{\phi}\cos(\phi+\phi_0) \tag{2.16}$$

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} = 2(m_r + \frac{1}{2}M)R^2\ddot{\theta} + m_r R^2\ddot{\theta} + MRL\ddot{\phi}\cos(\phi+\phi_0) - MRL\dot{\phi}^2\sin(\phi+\phi_0) = 2a\ddot{\theta} + c\ddot{\phi}\cos(\phi+\phi_0) - c\dot{\phi}^2\sin(\phi+\phi_0) \tag{2.17}$$

$$\frac{\partial L}{\partial \theta} = 0 \tag{2.18}$$

donde

$$a = (m_r + \frac{1}{2}M)R^2 + \frac{1}{2}m_r R^2 \qquad b = ML^2 \tag{2.19}$$

$$c = MRL \qquad d = -MgL \tag{2.20}$$

Finalmente, se puede definir el par motor aplicado sustituyendo las derivadas parciales por los términos antes calculados.

$$-T = 2b\ddot{\phi} + c\cos(\phi+\phi_0)\ddot{\theta} - d\sin(\phi+\phi_0)\dot{\phi} \tag{2.21}$$

$$T = 2a\ddot{\theta} + c\cos(\phi+\phi_0)\ddot{\phi} - c\sin(\phi+\phi_0)\dot{\phi}^2 \tag{2.22}$$

Por lo tanto, igualando las ecuaciones se obtiene la ecuación que expresa el comportamiento dinámico del vehículo autoequilibrado.

$$(2a+c\cos(\phi+\phi_0))\ddot{\theta} + (2b+c\cos(\phi+\phi_0))\ddot{\phi} - c\dot{\phi}^2\sin(\phi+\phi_0) - d\sin(\phi+\phi_0) = 0 \tag{2.23}$$

2.2.2 Modelo lineal

El controlador busca llevar el sistema a la posiciíon de equilibrio. Para el diseño del controlador se necesitará linealizar el modelo, es decir linealizar la ecuación que define el comportamiento dinámico del segway.

Para realizar esta linealización, se supone que el ángulo de inclinación del vehículo autoequilibrado es muy pequeño por lo que se puede utilizar la siguiente aproximación: $\phi + \phi_0 << 1$.

Ya que en la ecuación aparecen las funciones de sin y cos, considerando que el ángulo $\phi + \phi_0 << 1$, se puede afirmar:

$$\sin(\phi+\phi_0) \simeq \phi+\phi_0 \qquad\qquad \cos(\phi+\phi_0) \simeq 1 \tag{2.24}$$

Se define la función F que será linealizada en torno al punto de equilibrio.

$$F \equiv (2a+c)\ddot{\theta} + (2b+c)\ddot{\phi} - c\dot{\phi}^2(\phi+\phi_0) - d(\phi+\phi_0) = 0 \tag{2.25}$$

El punto de equilibrio de este sistema teniendo en cuenta la perturbación provocada por la diferencia entre el punto del centro geométrico y el del centro de masa, junto con sus derivadas, será:

$$\phi_{eq} = -\phi_0 \qquad\qquad \dot{\phi}_{eq} = 0 \tag{2.26}$$

$$\ddot{\phi}_{eq} = 0 \qquad\qquad \ddot{\theta}_{eq} = u_{eq} = 0 \tag{2.27}$$

Para linealiar F se aplicará una serie de Taylor alrededor de un punto de operación, en este caso el punto de equilibrio. Ya que la variación entre las variables y su punto de equilibrio es muy pequeña, se desprecian los términos de orden superior. El resultado final será una ecuación diferencial lineal. Es importante destacar que el rango de validez del modelo linealizado estará limitado por el punto de linealización.

$$\frac{\partial F}{\partial u}\Big|_{eq}(u-u_{eq}) + \frac{\partial F}{\partial \ddot{\phi}}\Big|_{eq}(\ddot{\phi}-\ddot{\phi}_{eq}) + \frac{\partial F}{\partial \dot{\phi}}\Big|_{eq}(\dot{\phi}-\dot{\phi}_{eq}) + \frac{\partial F}{\partial \phi}\Big|_{eq}(\phi-\phi_{eq}) = 0 \tag{2.28}$$

Se muestra lo obtenido en cada una de las derivadas parciales de la función F.

$$\frac{\partial F}{\partial u}\Big|_{eq} = 2a+c \qquad\qquad \frac{\partial F}{\partial \ddot{\phi}}\Big|_{eq} = c+2b \tag{2.29}$$

$$\frac{\partial F}{\partial \dot{\phi}}\Big|_{eq} = 0 \qquad\qquad \frac{\partial F}{\partial \phi}\Big|_{eq} = -d \tag{2.30}$$

Se sustituyen los resultados de las derivadas parciales en la función de linealización y se iguala a cero.

$$(2a+c)u+(c+2b)\ddot{\phi}-d(\phi+\phi_0)=0 \tag{2.31}$$

Para simplificar la ecuación será definida como:

$$eu+f\ddot{\phi}-d(\phi+\phi_0)=0 \tag{2.32}$$

donde

$$e=2a+c \qquad\qquad f=c+2b \tag{2.33}$$

Por lo tanto, el sistema en espacio de estados quedará:

$$\begin{bmatrix}\dot{\phi}\\ \ddot{\phi}\\ \ddot{\theta}\end{bmatrix}=\begin{bmatrix}0 & 1 & 0\\ \frac{-d}{f} & 0 & 0\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\phi\\ \dot{\phi}\\ \dot{\theta}\end{bmatrix}+\begin{bmatrix}0\\ \frac{-e}{f}\\ 1\end{bmatrix}\ddot{\theta}+\begin{bmatrix}0\\ \frac{-d}{f}\\ 0\end{bmatrix}\phi_0 \tag{2.34}$$

es decir, el sistema de control utilizará la siguiente ecuación matricial:

$$\dot{x}(t)=Ax(t)+Bu(t)+E_pd(t) \tag{2.35}$$

donde se controlará la aceleración de la rueda del vehículo autoequilibrado. En cuanto a las variables de salida del sistema, la única variable que queda libre es la velocidad de la rueda y a la cual se le puede exigir el seguimiento de una velocidad de referencia. Las otras variables se encuentran fijadas en el punto de equilibrio como se puede observar en las ecuaciones 1.20 y 1.21.

$$\begin{bmatrix}\dot{\theta}\end{bmatrix}=C\begin{bmatrix}\phi\\ \dot{\phi}\\ \dot{\theta}\end{bmatrix}=\begin{bmatrix}0 & 0 & 1\end{bmatrix}\begin{bmatrix}\phi\\ \dot{\phi}\\ \dot{\theta}\end{bmatrix} \tag{2.36}$$

Cálculo de las matrices del modelo

Para el cálculo de las matrices del sistema en tiempo discreto se usará la herramienta Matlab. Se deben tener dos aspectos en cuenta:

1. Tiempo discreto
Se debe realizar el cambio de sistema en tiempo continuo a sistema en tiempo discreto.

$$\dot{x}(t)=Ax(t)+Bu(t)+E_pd(t)\longrightarrow x_{k+1}=A_dx_k+B_du_k+E_pd_k \tag{2.37}$$

$$t=T_m\cdot k \tag{2.38}$$

donde se define el tiempo de muestreo $T_m=0.04s$.

2. Perturbación
Se sabe que Matlab no es capaz de tener en cuenta en el espacio de estado una perturbación. Por lo tanto para el cálculo de la matriz de la pertubación, es decir E_p, tendrá que ser definida en B_d. De esta forma en la acción de control se añadirá la constante ϕ_0, que es el ángulo de inclinación.

$$\dot{x}(t)=Ax(t)+B^{'}u(t)\longrightarrow x_{k+1}=A_dx_k+B_d^{'}u_k \tag{2.39}$$

Por lo tanto el sistema definido en (1.40) se transformará para el cálculo de las matrices en Matlab y quedará:

$$\begin{bmatrix} \dot{\phi} \\ \ddot{\phi} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ \frac{-d}{f} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \phi \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ \frac{-e}{f} & \frac{-d}{f} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \ddot{\theta} \\ \phi_0 \end{bmatrix} \tag{2.40}$$

Se muestra el código utilizado para el cálculo de A_d, B_d, C_d y D_d, donde B_d será $B_d^{'}$ la cual contiene E_p.

Código 2.1 Cálculo del sistema en discreto.

```
%Parámetros del modelo Segway
g=9.81;
mr=0.140; % Masa de cada rueda
R=0.05; % Radiod e la rueda
M=0.82; % Masa del cuerpo del pendulo
L=0.098; % Distancia del centro de gravedad al eje de las ruedas
% Momentos de inercia.
Jr=0.5*mr*R^2;
J=M*L^2;
% Constantes
a=(mr+M/2)*R^2+Jr;
b=M*L^2/2+J/2;
c=M*R*L;
d=-M*g*L;
e=2*a+c;
f=2*b+c;
A=[0 1 0; -d/f 0 0; 0 0 0];
B=[0 0; -e/f -d/f; 1 0];
C=[0 0 1];
D=[0 0];
Tm=0.04;
sys=ss(A,B,C,D);
sysd=c2d(sys,Tm);
[Ad,Bd,Cd,Dd]=ssdata(sysd)
```

A la izquierda se encuentran las matrices del sistema continuo donde se ha añadido E_p en B convirtiéndose en $B^{'}$. A la derecha este mismo sistema ha sido modificado a tiempo discreto.

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 39.9 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \longrightarrow A_d = \begin{bmatrix} 1.0321 & 0.0404 & 0 \\ 1.6121 & 1.0321 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$B^{'} = \begin{bmatrix} 0 & 0 \\ -0.36 & 39.9 \\ 1 & 0 \end{bmatrix} \longrightarrow B_d^{'} = \begin{bmatrix} -2.8959 \cdot 10^{-4} & 0.0321 \\ -0.0146 & 1.6121 \\ 0.04 & 0 \end{bmatrix}$$

Finalmente, separando la matriz $B_d^{'}$ en B_d y E_p, estas quedarán definidas como:

$$A_d = \begin{bmatrix} 1.0321 & 0.0404 & 0 \\ 1.6121 & 1.0321 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad B_d = \begin{bmatrix} -2.8959 \cdot 10^{-4} \\ -0.0146 \\ 0.04 \end{bmatrix} \quad E_p = \begin{bmatrix} 0.0321 \\ 1.6121 \\ 0 \end{bmatrix} \tag{2.41}$$

No se han expuesto las matrices C_d y D_d ya que no hay ningún cambio en ellas al discretizar el sistema.

Límites del sistema

Los valores de los límites se tomarán diferentes a los valores físicos que debe tener el sistema. En el cálculo del algoritmo de control, será necesaria la definición de restricciones tanto para el estado como para la acción de control.

Los límites del estado tendrán la misma restricción para ambas inclinaciones del segway. Tanto la positiva como la negativa. Para el ángulo de inclinación se conoce que su límite máximo físico es de $\pm 90\circ$. Sin embargo, este límite es demasiado amplio por lo que se restringe la inclinación a $\pm 30\circ$. Como máxima variación de la inclinación se podrá alcanzar hasta $\pm 5 rad/s$. Y como velocidad de la rueda hasta $\pm 12 rad/s$.

$$\begin{bmatrix} -30 \cdot \frac{\pi}{180} \\ -5 \\ -12 \end{bmatrix} \leq \begin{bmatrix} \phi \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} \leq \begin{bmatrix} 30 \cdot \frac{\pi}{180} \\ 5 \\ 12 \end{bmatrix} \begin{pmatrix} rad \\ rad/s \\ rad/s \end{pmatrix} \tag{2.42}$$

Se utlizará el valor experimental obtenido de la aceleración máxima que pueden dar los motores de las ruedas que es $\pm 150 rad/s^2$, siendo esto el límite de la acción de control.

$$-150 \leq u \leq 150 \left(\frac{rad}{s^2} \right) \tag{2.43}$$

3 Control Predictivo con Algoritmo FISTA

3.1 Control Predictivo basado en Modelo (MPC)

3.1.1 Modelo general

El Control Predictivo basado en Modelo es una estrategia de control que minimiza una función de costes. Esta técnica de control puede considerar tanto restricciones en las entradas del sistema como en la acción de control. Esta estrategia de control requiere solucionar un problema de optimización en cada periodo de muestreo. El controlador deberá predecir cómo evolucionará el sistema y calculará la acción de control.

Sea el siguiente modelo en tiempo discreto que será usado por los controladores predictivos para la predicción de la evolución del sistema:

$$x_{k+1} = Ax_k + Bu_k \tag{3.1}$$
$$y_k = Cx_k + Du_k \tag{3.2}$$
$$x(0) = x_0 \tag{3.3}$$

Donde $x_k \in \mathbb{R}^n$ es el vector de n componentes del estado actual del sistema, $u_k \in \mathbb{R}^m$ es el vector de m componentes de la acción de control del sistema, $x_{k+1} \in \mathbb{R}^n$ es el vector de n componentes del estado del sistema en el siguiente periodo de muestreo, $y_k \in \mathbb{R}^p$ es el vector de p componentes de la salida actual del sistema y $A \in \mathbb{R}^{nxn}$, $B \in \mathbb{R}^{nxm}$, $C \in \mathbb{R}^{pxn}$, $D \in \mathbb{R}^{pxm}$son matrices que describen el sistema en el espacio de estados.

A continuación se definen las restricciones en el estado y en la acción de control:

$$LB_x \leq x_k \leq UB_x \tag{3.4}$$
$$LB_u \leq u_k \leq UB_u \tag{3.5}$$

Donde $LB_x \in \mathbb{R}^n$, $LB_u \in \mathbb{R}^m$ son vectores con el valor del límite inferior de las variables de estado y acción de control respectivamente, y $UB_x \in \mathbb{R}^n$, $UB_u \in \mathbb{R}^m$ vectores con los valores de los límites superiores.

Para el MPC será necesario definir la función de coste que se quiere optimizar minimizándola:

$$J(x,u,x_r,u_r) = \sum_{i=0}^{N-1} \|x(i) - x_r\|_Q^2 + \|u(i) - u_r\|_R^2 \tag{3.6}$$

Donde se define N como el horizonte de predicción, x_r es el estado de referencia y u_r es la acción de control de referencia.

El problema de optimización que se plantea quedará de la siguiente forma:

$$J^* = \min_u J(x,u,x_r,u_r) \tag{3.7}$$
$$s.a. \quad x_{k+1} = Ax_k + Bu_k \tag{3.8}$$
$$x_r = Ax_r + Bu_r \tag{3.9}$$
$$x(0) = x_0 \tag{3.10}$$
$$x(N) = x_r \tag{3.11}$$
$$LB_x \leq x_k \leq UB_x \tag{3.12}$$
$$LB_u \leq u_k \leq UB_u \tag{3.13}$$
$$siendo \quad k = 0,...,N-1 \tag{3.14}$$

Donde x_0 es el estado actual y Q y R son matrices de ponderación que penalizan la diferencia entre los valores de x y x_r, y entre u y u_r.

Finalmente para poder utilizar el algoritmo FISTA se reformula el problema y se escribe como un problema QP (programación cuadrática) que quedará de la siguiente forma:

$$Z^* = \arg\min_Z \frac{1}{2} Z^T HZ + f^T Z \tag{3.15}$$
$$s.a. \quad LB \leq Z \leq UB \tag{3.16}$$
$$EZ = b \tag{3.17}$$

Donde H es una matriz diagonal y definida positiva. La función objetivo a minimizar estará relacionada con la función de coste. Las restricciones harán referencia al modelo donde E es una matriz que contiene las matrices A y B, ya que la expresión $EZ = b$ representa la ecuación $x_{k+1} = Ax_k + Bu_k$.

3.1.2 Modelo con perturbación

El modelo en tiempo discreto del Segway tendrá una perturbación como consecuencia de la diferencia entre el centro de gravedad y el centro geométrico del sistema real. Por lo tanto se definirá el siguiente modelo que será usado para la evolución del vehículo:

$$x_{k+1} = Ax_k + Bu_k + E_p d \tag{3.18}$$
$$y_k = Cx_k + Du_k \tag{3.19}$$
$$x(0) = x_0 \tag{3.20}$$

Donde d es el ángulo contrario al que se provoca por la diferencia entre el centro de gravedad y el centro geométrico. Es el ángulo en el punto de equilibrio del sistema. Y E_p es el vector de n componentes de la perturbación.

En consecuencia, el problema de optimización del sistema será:

$$J^* = \min_u J(x,u,x_r,u_r) \tag{3.21}$$
$$s.a. \quad x_{k+1} = Ax_k + Bu_k + E_p d \tag{3.22}$$
$$x_r = Ax_r + Bu_r + E_p d \tag{3.23}$$
$$x(0) = x_0 \tag{3.24}$$
$$x(N) = x_r \tag{3.25}$$
$$LB_x \leq x_k \leq UB_x \tag{3.26}$$
$$LB_u \leq u_k \leq UB_u \tag{3.27}$$
$$siendo \quad k = 0,...,N-1 \tag{3.28}$$

En este caso el problema QP se formulará de la misma forma que en el apartado anterior:

$$Z^* = \arg\min_Z \frac{1}{2} Z^T H Z + f^T Z \tag{3.29}$$
$$s.a. \quad LB \leq Z \leq UB \tag{3.30}$$
$$EZ = b \tag{3.31}$$

La diferencia estará en la expresión $EZ = b$, que en lugar de representar la ecuación $x_{k+1} = Ax_k + Bu_k$, se le añadirá la perturbación quedando la ecuación como $x_{k+1} = Ax_k + Bu_k + E_p d$, lo que provoca que el vector b cambie añadiéndose en él esa perturbación.

3.2 Algoritmo FISTA

En esta sección se explica el algoritmo FISTA para solucionar el problema QP del apartado anterior. Inicialmente se supondrá el problema sin las restricciones de límites en los valores inferiores y superiores de las variables de estado y acción de control. Por lo tanto el problema será:

$$Z^* = \arg\min_Z \frac{1}{2} Z^T H Z + f^T Z \tag{3.32}$$
$$s.a. \quad LB \leq Z \leq UB \tag{3.33}$$
$$EZ = b \tag{3.34}$$

Donde H será definida positiva y diagonal.

3.2.1 Problema sin restricciones de igualdad

Sea el mismo problema pero sin las restricciones de igualdad:

$$Z^* = \arg\min_Z \frac{1}{2} Z^T H Z + f^T Z \tag{3.35}$$
$$s.a. \quad LB \leq Z \leq UB \tag{3.36}$$

En este caso la solución es directa:

$$Z^* = -fH^{-1} \tag{3.37}$$
$$s.a. \quad Z^* \in \mathbb{Z} \tag{3.38}$$

Se sabe que la matriz H es diagonal y definida positiva, por lo tanto su inversa será otra matriz diagonal y positiva cuyos elementos de dicha diagonal serán los inversos de la original. Por otra parte se sabe que cada i-ésimo elemento del vector solución Z_i^* deberá cumplir los límites LB y UB. Si la solución no cumple con estos límites porque el mínimo no esté contenido en el intervalo donde esta se necesita, entonces la solución al problema no será la óptima sino uno de los valores de los límites y se tomará aquel que provoque el mínimo.

Ejemplo 1

En este ejemplo el mínimo óptimo de la función se encuentra entre los límites que se muestran con las gráficas verde (límite inferior) y roja (límite superior).

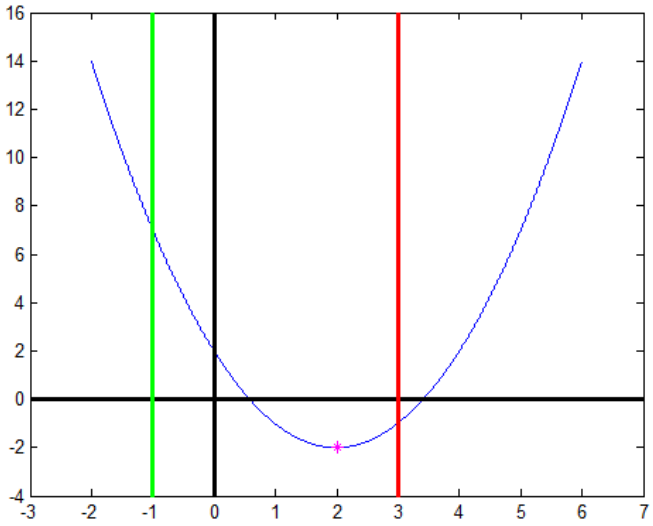


Figura 3.1 Ejemplo 1.

Ejemplo 2

En este ejemplo el mínimo óptimo de la función no se encuentra entre los límites que se muestran con las gráficas verde y roja, por lo que el mínimo como solución posible será el mínimo entre los valores límite. En este caso el mínimo se da en el límite superior.

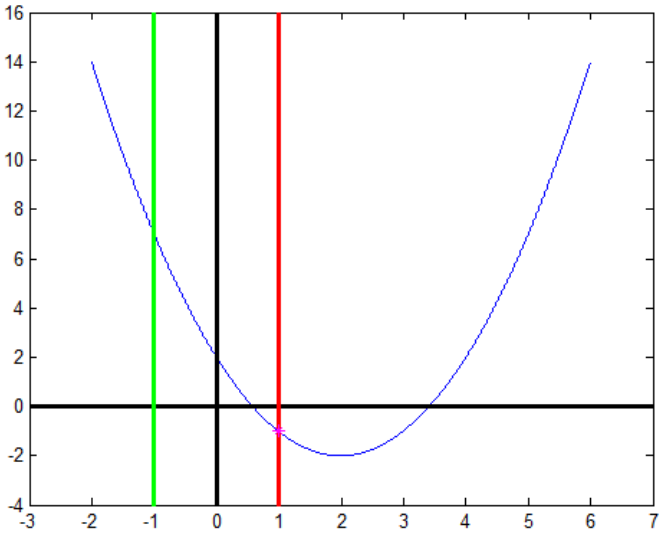


Figura 3.2 Ejemplo 2.

Ejemplo 3

En este ejemplo la función tiene un máximo óptimo y el mínimo se encontrará en el valor mínimo de uno de los límites. En este caso el mínimo se da en el límite inferior.

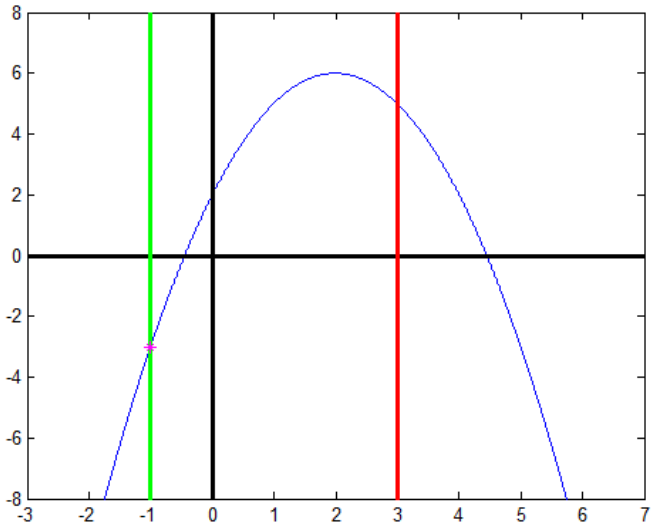


Figura 3.3 Ejemplo 3.

3.2.2 Problema con restricciones de igualdad

Se parte del problema inicial:

$$J^* = \arg\min_Z \frac{1}{2} Z^T H Z + f^T Z \tag{3.39}$$

$$s.a. \quad LB \leq Z \leq UB \tag{3.40}$$

$$EZ = b \tag{3.41}$$

Como se ha demostrado, este problema sin las restricciones de igualdad se resuelve con un cálculo directo, sin embargo al añadir la restricción de igualdad se complica la resolución del mismo. Las restricciones de tipo igualdad no establecen fronteras al conjunto de las soluciones factibles del problema, sin embargo reducen las dimensiones del espacio donde el problema estará definido, dificultando así su cumplimiento. Sería más sencillo poder encontrar la solución si se tuvieran solo desigualdades, ya que habría un mayor rango posible de resultados como se muestra en el ejemplo anterior.

Extremos condicionados: multiplicadores de Lagrange

Para poder calcular los extremos condicionados de la función objetivo, es decir el máximo o mínimo de la misma, se define la función Lagrangiana con la cual se facilita el cálculo, ya que esta consigue transformar el extremo condicionado en un extremo libre, es decir un problema sin restricciones de igualdad. Por lo que para encontrar el mínimo en lugar de usar la función objetivo se utilizará la Lagrangiana. Se define:

$$J(Z) = \frac{1}{2} Z^T H Z + f^T Z \tag{3.42}$$

$$L(Z,x) = J(Z) - x^T (EZ - b) \tag{3.43}$$

Se define $J(Z)$ como la función objetivo que se quiere minimizar y $L(Z,x)$ como la función Lagrangiana la cual contiene los multiplicadores de Lagrange. Este método contiene un procedimiento para encontrar máximos y mínimos de funciones de múltiples variables sujetas a restricciones. Consiste por tanto en pasar de un problema restringido con n variables a uno sin restricciones con $n+k$ variables, donde k es la cantidad de restricciones. Las k nuevas variables son llamadas multiplicadores de Lagrange.

En consecuencia, se definirán tantos multiplicadores de Lagrange como ecuaciones de ligadura o lo que es

lo mismo restricciones de igualdad. En este caso al tener una única restricción se definirá solo un multiplicador. Es decir se añade solo una variable más al problema eliminando así la restricción de igualdad. Esta nueva variable es x. El tamaño del multiplicador x, dependerá del tamaño de la variable de la ecuación de ligadura, siendo esa variable el vector Z, cuyo tamaño es $N*(n+m)$. Por lo tanto, el multiplicador de Lagrange x será definido con las mismas dimensiones que el vector Z.

Se sabe que se pretende calcular el Z para el que la función Lagrangiana es mínima, por esta razón se define $Z(x)$.

$$Z(x) = \arg\min_{Z} L(Z,x) \tag{3.44}$$

Finalmente usando el teorema de la función implícita [1], el sistema será resoluble. Este teorema permite que en una o más ecuaciones de varias variables se pueda definir una o varias de ellas como función de las demás. Esto lograría que la Langragiana solo dependiera de la variable x haciendo que la variable Z depende de x.

$$f(x) = L(Z(x),x) \tag{3.45}$$

Es conocido que el coste óptimo J^* se obtiene cuando $EZ = b$, es decir cuando se cumple la condición de igualdad, quedando $L(Z^*,x) = J^*$. En una cota no óptima, $L(Z,x)$ será menor que $J(Z)$ al ser $x^T(EZ^* - b)$ mayor a cero y estar restándose en la expresión $L(Z,x)$. También se nombrará $L(Z,x)$ como $f(x)$ que será menor que $J(Z)$, por lo tanto se cumple que:

$$J^* = L(Z^*,x) \geq f(x) = L(Z(x),x) \tag{3.46}$$

Es claro entonces que se pretende encontrar la cota inferior más alta, es decir que para alcanzar el óptimo, J^*, se necesita encontrar el x para el que $f(x)$ sea máximo, se deberá resolver:

$$x^* = \arg\max_{x} f(x) \tag{3.47}$$

Como conclusión se quiere calcular el Z para el cual la Lagrangiana, $L(Z,x)$, es mínima. Si se quiere expresar con la variable x, entonces se desea obtener x para la cual $f(x)$ que solo depende de esta variable es máxima.

Finalmente se calculará el resultado final de la función objetivo que se usará en el algoritmo FISTA

[1] Teorema de la función implícita
Sea $F : A \subset \mathbb{R}^n \times \mathbb{R} \mapsto \mathbb{R}$ definida en un entorno del punto $(x_0,y_0) \in A$, A abierto de $\mathbb{R}^n \times \mathbb{R}$. Supongamos que:

1. $F(x,y) := F(x_1,x_2,\ldots,x_n,y) \in C^{(p)}(A), p \geq 1$,
2. $F(x_0,y_0) := F(x_{01},x_{02},\ldots,x_{n0},y_0) = 0$,
3. $F_y^{'}(x_0,y_0) = \frac{\partial F(x_{01},x_{02},\ldots,x_{n0},y_0)}{\partial y} \neq 0$.

Entonces existe un abierto $I = I_x \times I_y = (x_0 - h,x_0 + h) \times (y_0 - k,y_0 + k)$ alrededor del punto $(x_0,y_0), I \subset A$, y una función $f : I_x \subset \mathbb{R}^n \mapsto I_y \subset \mathbb{R}$ tal que:

1. $F(x,y) = 0$ en I si y solo si $f(x) = y$,
2. $f(x) \in C^{(p)}(I_x)$,
3. Para todo $x \in I_x$, las derivadas parciales de $f(x)$ se calculan por la fórmula
$\frac{\partial f(x)}{\partial x_i} := \frac{\partial f(x_1,\ldots,x_n)}{\partial x_i} = -[F_y^{'}(x,f(x))]^{-1} \cdot [F_{x_i}^{'}(x,f(x))], i = 1,2,\ldots,n$,
donde por $F_{x_i}^{'}$ denotamos la derivada parcial $\frac{\partial F}{\partial x_i}$.

teniendo en cuenta los multiplicadores de Lagrange y añadiendo por tanto la nueva variable relacionada con la restricción de igualdad. Desarrollando la función $Z(x)$ se obtendrá:

$$Z(x) = \arg\min_{Z} \frac{1}{2} Z^T HZ + f^T Z - x^T (EZ - b) =$$
$$= \arg\min_{Z} \frac{1}{2} Z^T HZ + f^T Z - x^T EZ + x^T b$$

Ya que se busca el Z para el que la función se minimice, se puede afirmar que el término $x^T b$ es una constante. Si se define la misma función pero sin la parte constante quedaría:

$$g(z,x) = \arg\min_{Z} \frac{1}{2} Z^T HZ + (f^T - x^T E)Z$$

Por tanto, la función $Z(x)$ como $g(x)$ alcanzan el mínimo en el mismo valor de Z, se puede afirmar:

$$\arg\min_{Z} \frac{1}{2} Z^T HZ + f^T Z - x^T EZ + x^T b =$$
$$= \arg\min_{Z} \frac{1}{2} Z^T HZ + (f^T - x^T E)Z$$

Es decir que para encontrar el Z que haga mínima la función, tanto $Z(x)$ como $g(z,x)$ ofrecen el mismo resultado. Por último usando las propiedades de la traspuesta la función $g(z,x)$ quedará:

$$\arg\min_{Z} \frac{1}{2} Z^T HZ + (f - E^T x)^T Z$$

La función obtenida, $g(z,x)$, será la que se minimice utilizando el algoritmo FISTA. Se tendrán en cuenta las restricciones del vector Z en este cálculo.

Teorema local de Taylor

El algoritmo FISTA necesitará hacer una serie de iteraciones antes de encontrar la solución. El algoritmo consiste en encontrar el Z óptimo, para el que se minimiza $g(z,x)$. Considerando que x será constante. La forma más rápida de encontrar la solución es usando el teorema local de Taylor, que propone derivar una aproximación local. Este teorema proporciona la forma de calcular lo que debe variar el valor del vector Z para conseguir encontrar un extremo, es decir un máximo o un mínimo de la función.

Una vez encontrado el Z que minimiza $g(x,z)$, se comprueba si se cumple que la norma de $EZ - b$ es menor o igual al valor de TOL, siendo TOL un valor que se aproxima a cero. Se sabe que si el valor de la norma de un vector tiende a cero, quiere decir que cada uno de los elementos del vector tenderá a cero. Por lo tanto se estarían cumpliendo las restricciones de igualdad. Y se habría encontrado la solución. Si esto no sucede se volverá a actualizar el valor de x, en el que además se le añaden una serie de operaciones posteriores a este cálculo, que agilizan la búsqueda de la solución.

Estos cálculos consiguen aproximar el valor de Z a la solución óptima. Por lo tanto se logrará hallar el Z^* en menor tiempo, ya que se necesitarán menos iteraciones para resolver el problema. Esto es al añadir el paso III y IV en c) en el algoritmo.

A continuación se define el teorema local de Taylor:

$$f(x+\Delta x) - f(x) \geq -\Delta x^T (EZ(x) - b) - \frac{1}{2}\Delta x^T W \Delta x \tag{3.48}$$

siendo $W = EH^{-1}E^T$.

Se busca que esta desigualdad se convierta en una igualdad, por lo tanto se debe maximizar $h = -\Delta x^T (EZ(x) - b) - \frac{1}{2}\Delta x^T W \Delta x$, ya que lo que se busca es Δx para el cual h es máxima, quedando la siguiente función:

$$\arg\max_{\Delta x} -\Delta x^T (EZ(x) - b) - \frac{1}{2}\Delta x^T W \Delta x = \arg\min_{\Delta x} \Delta x^T (EZ(x) - b) + \frac{1}{2}\Delta x^T W \Delta x = \tag{3.49}$$

$$= \arg \Delta x^T \min_{\Delta x}((EZ(x) - b) + \frac{1}{2} W \Delta x) = \arg \Delta x^T ((EZ(x) - b) + \min_{\Delta x} \frac{1}{2} W \Delta x) \tag{3.50}$$

Una función o la mitad de una función alcanza su valor mínimo en el mismo punto. Por lo tanto se cumple que $\min_{\Delta x} \frac{1}{2} W \Delta x = \min_{\Delta x} W \Delta x$. Finalmente queda como:

$$\arg \Delta x^T ((EZ(x) - b) + \min_{\Delta x} W \Delta x) = \arg \Delta x^T \min_{\Delta x}((EZ(x) - b) + W \Delta x) \tag{3.51}$$

En este problema de optimización se alcanzará el óptimo en el punto en el que se anule el gradiente, ya que si el gradiente es nulo la pendiente también lo es y por tanto será un punto con un máximo o un mínimo.

$$\frac{f(x+\Delta x) - f(x)}{\Delta x} = (EZ(x) - b) + W \Delta x = 0 \tag{3.52}$$

Para terminar se obtiene el valor de Δx para conseguir que el gradiente sea nulo y por tanto se encuentra así un extremo.

$$\Delta x = -W^{-1}(EZ(x) - b) \tag{3.53}$$

Como conclusión, se busca variar la x para encontrar el punto donde la pendiente es nula obteniendo así un extremo.

Implementación del algoritmo FISTA

La implementación del algoritmo FISTA quedaría:

1. $k \leftarrow 1, x_0 \leftarrow 0, y_1 \leftarrow 0, t_1 \leftarrow 1, FIN \leftarrow NO, Q \leftarrow W^{-1}$

2. Repetir hasta $FIN = SI$

a) Calcular $Z(y_k)$ tal que

$$\min_Z \frac{1}{2} Z^T H Z + (f - E^T y)^T Z$$
$$s.a. LB \leq Z \leq UB$$

b) Si $\|EZ(y_k) - b\| \leq TOL$, entonces $FIN \leftarrow SI$

c) Si no

I. Calcular $\Delta y \leftarrow -Q(EZ(y_k)-b)$

II. Hacer $x_k \leftarrow y_k + \Delta y$

III. Hacer $t_{k+1} \leftarrow \frac{1}{2}(1+\sqrt{1+4t_k^2})$

IV. Hacer $y_{k+1} \leftarrow x_k + \frac{t_k-1}{t_{k+1}}(x_k - x_{k-1})$

V. Hacer $k \leftarrow k+1$

3. $Z^* \leftarrow Z_k$

Habría que tener en cuenta que la y que se describe en el algoritmo es la x que se ha descrito en el teorema de Taylor.

En ocasiones puede resultar demasiado complejo encontrar la solución óptima, por esta razón se debe añadir dentro de la programación un número máximo de iteraciones. Si se alcanzara esta cantidad de iteraciones se devolvería como resultado la última Z calculada y finalizaría el programa. Se sabe que este valor no será el óptimo, ya que el cálculo se ha parado antes de encontrarlo, pero posiblemente sí sea una buena aproximación del mismo.

A continuación se muestran las definiciones de las matrices: El vector Z que contiene hasta N vectores de estado y N vectores con la acción de control, por lo que su tamaño será de $(N(m+n))$:

$$Z = \begin{pmatrix} x_n(0) \\ u_m(0) \\ x_n(1) \\ u_m(1) \\ \vdots \\ x_n(N-1) \\ u_m(N-1) \end{pmatrix}$$

donde el vector x estará definido como $x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ y el vector u estará definido como $u = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{pmatrix}$.

La matriz H tendrá unas dimensiones de $(N(m+n)xN(m+n))$, se define como:

$$H = \begin{pmatrix} Q & 0 & 0 & 0 & \cdots & 0 \\ 0 & R & 0 & 0 & \cdots & 0 \\ 0 & 0 & Q & 0 & \cdots & 0 \\ 0 & 0 & 0 & R & \cdots & 0 \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & Q & 0 \\ 0 & 0 & 0 & \cdots & 0 & R \end{pmatrix}$$

El vector f será de dimensión $(N(m+n))$:

$$f = \begin{pmatrix} -Qx_r \\ -Ru_r \\ -Qx_r \\ -Ru_r \\ \vdots \\ -Qx_r \\ -Ru_r \end{pmatrix}$$

La matriz E tendrá dimensiones de $((N+1)nxN(m+n))$ y se define:

$$E = \begin{pmatrix} I & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ A & B & -I & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & A & B & -I & \cdots & 0 & 0 \\ \vdots & \vdots & & & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & A & B \end{pmatrix}$$

El vector b en el modelo general tendrá una dimensión de $((N+1)n)$ y será definido como:

$$b = \begin{pmatrix} x \\ 0 \\ 0 \\ \vdots \\ x_r \end{pmatrix}$$

donde el vector $x = \begin{pmatrix} x_1(0) \\ x_2(0) \\ \vdots \\ x_N(0) \end{pmatrix}$ y el vector $x_r = \begin{pmatrix} x_{r,1} \\ x_{r,2} \\ \vdots \\ x_{r,N} \end{pmatrix}$.

Sin embargo, ya que el sistema tiene la perturbación, el vector b se definirá como:

$$b = \begin{pmatrix} x \\ -E_p d \\ -E_p d \\ \vdots \\ x_r - E_p d \end{pmatrix}$$

Los vectores de los límites inferior y superior tendrán las mismas dimensiones que Z y vienen definidos como:

$$LB = \begin{pmatrix} LBx \\ LBu \\ LBx \\ LBu \\ \vdots \\ LBx \\ LBu \end{pmatrix} \qquad UB = \begin{pmatrix} UBx \\ UBu \\ UBx \\ UBu \\ \vdots \\ UBx \\ UBu \end{pmatrix}$$

El código implementado en Matlab que describe el algoritmo FISTA se muestra a continuación:

Código 3.1 Cálculo del MPC.

```
x=zeros((N+1)*n,1);x_ant=zeros((N+1)*n,1);y=zeros((N+1)*n,1);
t=1;FIN=0;cont1=0;
Qfista=inv(E/H*E');
%matrices f1 y b
f1=zeros(N*(n+m),1);
for i=1:n+m:N*(n+m)
f1(i:i+n-1,1)=-Q*xr;
f1(i+n:i+n,1)=-R*ur;
end
dd=Ep*d_filtrada
b=zeros((N+1)*n,1);
b(1:n,1)=x_filtrada;
for i=1:3:N*n
b(n+i:n+i+2)=-dd;
end
b((N+1)*n-(n-1):(N+1)*n,1)=xr-dd;

while FIN==0
z=zeros(n,1);
f=f1-E'*y;
for i=1:N*(n+m)
z1=-f(i,1)/H(i,i);
z(i)=max(Llim(i,1),min(z1,Ulim(i,1)));
end
if norm(E*z-b)<=1e-6
FIN=1;
else
incry=-Qfista*(E*z-b);
x=y+incry;
t_sig=0.5*(1+sqrt(1+4*t^2));
y=x+(((t-1)/t_sig)*(x-x_ant));
cont1=cont1+1;
x_ant=x;
t=t_sig;
end
if cont1>2000
FIN=1;
disp('Se supera el número de iteraciones')
end
u=z(4);
end
end
```

3.3 Bucle del MPC y componentes

En este apartado se explicará el bucle de control completo. Como entrada se encuentra el parámetro r, que representa la referencia. En este caso la referencia será la velocidad de la rueda.

A su vez en cada bloque existirán una serie de valores ajustables, calculados previamente.Estos valores aparecen en cada uno de lo bloques del controlador y entre ellos están: N, Q y R, parámetros descritos anteriormente en la explicación del Algoritmo FISTA.

Los bloques a programar del control son: Filtro, SSTO, MPC y OBS. En cuanto al sistema, este será el cálculado en el Capítulo 1, en la sección del Modelo no Lineal, comprobando así que el control se realiza correctamente y es aplicado al sistema que se ha descrito.

Primero se muestra un diagrama de bloques como resumen esquemático del lazo de control en la Figura 3.4.

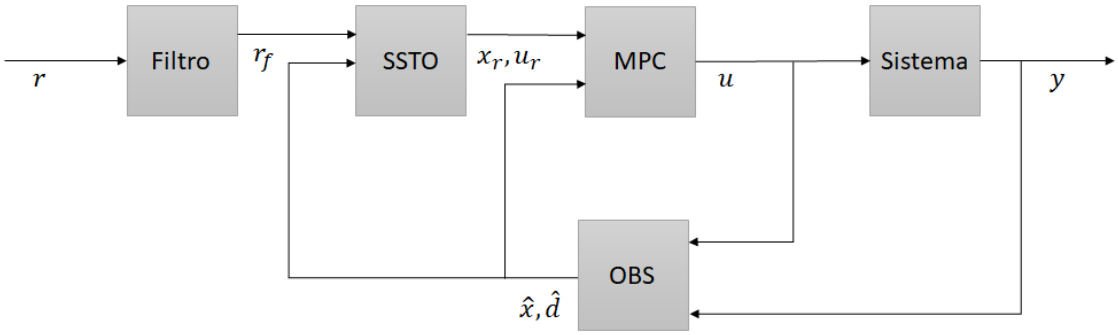


Figura 3.4 Diagrama de bloques.

Entre los componentes que aparecen en el esquema se hace un pequeño resumen de cada uno de ellos:

1. SSTO o Steady-State Target Optimization: calcula el punto de equilibrio asociado a la referencia que tiene de entrada, es decir r_f. En resumen calcula puntos de equilibrio alcanzables por el sistema, x_r y u_r. En este caso el bloque SSTO también usará el Algoritmo FISTA, ya que este calculo requiere que se realice una optimización asociada a unas restricciones.

2. OBS o estimador de estados: se utilizará el Filtro de Kalman que proporcionará el estado actual estimado, $\hat{x}$ y la perturbación estimada $\hat{d}$. Toma como entradas la acción de control u y la salida de la planta y.

3. Filtro de referencia: consigue que los cambios de referencia se realicen de forma exponencial en lugar de usar un escalón. Tiene como entrada r y como salida el suavizado o filtrado de esa referencia r_f.

4. MPC o Control Predictivo basado en Modelo: este bloque es el explicado en el apartado del Algoritmo FISTA. Se encarga de resolver la minización de la función objetivo sujeta a una serie de restricciones. Por lo tanto este bloque calcula la actuación del sistema, que será la entrada al bloque de la Planta. Para el cómputo de esto, se tendrán como entradas los puntos de equilibrio calculados por el SSTO, es decir x_r y u_r, y el estado actual estimado $\hat{x}$.

3.3.1 SSTO

Este bloque se usa para la obtención óptima de puntos de equilibrio. Puede existir el caso de referencias no alcanzables. Esto es resuelto ya que el SSTO soluciona el problema encontrando puntos óptimos para esas referencias no alcanzables. Su finalidad es conseguir que la salida del sistema converja hacia la referencia, es decir que la salida esté lo más cerca posible del valor de referencia exigido:

$$\lim_{k\to\infty} \|y_k - rk\| = 0$$

Se quiere por tanto encontrar el valor de un punto de equilibrio, es decir x_r, u_r y h_r. Siendo h_r la holgura, que facilita la búsqueda de la solución. El problema planteado en este componente del bucle se define como:

$$(x_r^*, u_r^*, h_r^*) = \arg\min_{x_r, u_r, h_r} \|x_r\|_{Q_r}^2 + \|u_r\|_{R_r}^2 + \|h\|_T^2 \tag{3.54}$$

$$s.a. \quad x_r = Ax_r + Bu_r + E_p d \tag{3.55}$$

$$r = Cx_r + h \tag{3.56}$$

$$LB_x \leq x_r \leq UB_x \tag{3.57}$$

$$LB_u \leq u_r \leq UB_u \tag{3.58}$$

Al ser un problema de minimización se puede utilizar el algoritmo FISTA al que se ha dado uso anteriormente para la minimización de la diferencia entre el estado de equilibrio y el del modelo lineal en tiempo discreto. En este caso se busca la minimización de la diferencia entre la referencia de la salida del sistema. Ya que se va a dar uso del algoritmo FISTA, este se reformula como un problema QP.

$$Z^* = \arg\min_Z \frac{1}{2} Z^T H Z + f^T Z \tag{3.59}$$

$$s.a. \quad LB \leq Z \leq UB \tag{3.60}$$

$$EZ = b \tag{3.61}$$

Donde la matriz H tendrá unas dimensiones de $(m+n+p)x(m+n+p)$, se define como:

$$H = \begin{pmatrix} Q_r & 0 & 0 \\ 0 & R_r & 0 \\ 0 & 0 & T \end{pmatrix}$$

El vector f será de dimensión $m+n+p$:

$$f = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

La matriz E tendrá dimensiones de $(n+m+p)x(n+p))$ y se define:

$$E = \begin{pmatrix} A-I & B & 0 \\ C & 0 & I \end{pmatrix}$$

El vector b tendrá una dimensión de $n+p$ y será definido como:

$$b = \begin{pmatrix} -E_p d \\ r \end{pmatrix}$$

El vector Z en este caso solo contiene el estado de equilibrio alcanzable x_r, el vector de la acción de control de equilibrio alcanzable y una tolerancia h, por lo que su tamaño será de $m+n+p$:

$$Z = \begin{pmatrix} x_r \\ u_r \\ h \end{pmatrix}$$

Los vectores de los límites inferior y superior tendrán las mismas dimensiones que Z y vienen definidos como:

$$LB = \begin{pmatrix} LBx \\ LBu \\ -\infty \end{pmatrix} \qquad UB = \begin{pmatrix} UBx \\ UBu \\ \infty \end{pmatrix}$$

Código 3.2 Cálculo del SSTO.

```
function [xr,ur]=SSTO(H1,E1,yr,dd,Ulim,Llim,n,m,p)
%Algoritmo FISTA
%inicialización
x=zeros(n+p,1);x_ant=zeros(n+p,1);y=zeros(n+p,1);
t=1;FIN=0;cont1=0;
Qfista=inv(E1/H1*E1');
f1=zeros(n+m+p,1);
b=zeros(n+p,1);
b(1:n,1)=dd;
b(n+1:n+p,1)=yr;
while FIN==0
z=zeros(n,1);
f=f1-E1'*y;
for i=1:n+m+p
z1=-f(i,1)/H1(i,i);
z(i)=max(Llim(i,1),min(z1,Ulim(i,1)));
end
if norm(E1*z-b)<=1e-6
FIN=1;
else
incry=-Qfista*(E1*z-b);
x=y+incry;
t_sig=0.5*(1+sqrt(1+4*t^2));
y=x+(((t-1)/t_sig)*(x-x_ant));
cont1=cont1+1;
x_ant=x;
t=t_sig;
end
if cont1>2000
FIN=1;
disp('Se supera el número de iteraciones')
end
end
xr=[z(1);z(2);z(3)];
ur=z(4);
end
```

Este código se puede comprobar antes de su implementación su correcto funcionamiento en el sistema. Para ellos basta con verificar que se cumple la condición:

$$Ax_r + Bu_r + E_p d = x_r \tag{3.62}$$

3.3.2 OBS

El OBS o estimador de estados calcula el estado estimado en cada instante. La utilización de este bloque hace necesario que el sistema sea observable. Esto quiere decir que teniendo cualquier secuencia de vectores de estado y de control, es posible obtener una estimación del estado actual. Y además esta será obtenida en un tiempo finito usando únicamente las salidas del sistema.

Observabilidad

Considérese el sistema lineal

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{3.63}$$
$$y(t) = Cx(t) + Du(t) \tag{3.64}$$

La matriz de observabilidad será definida como

$$O = \begin{pmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{pmatrix}$$

Se sabe que el sistema es observable siempre y cuando la matriz de observabilidad tenga un rango igual a n. Se comprueba en Matlab que este sistema es observable ya que el rango de la matriz O es 3 y su determinante no es nulo.

Filtro de Kalman

Entre las distintas formas de resolver este problema existen diferentes soluciones posibles. Se puede estimar el estado usando el observador de Luenberger o un filtro de Kalman. Se va a utilizar el filtro de Kalman, ya que este consigue elegir de forma óptima la ganancia del error. El filtro de Kalman es un algoritmo recursivo, lo que quiere decir que puede usar en tiempo real. Necesita como datos el estado calculado, la salida del sistema y la matriz de incertidumbre.

El sistema representado en el espacio de estados quedará:

$$x_{k+1} = Ax_k + Bu_k + E_p d + w_k \tag{3.65}$$
$$y_k = Cx_k + v_k \tag{3.66}$$
$$d_{k+1} = d_k \tag{3.67}$$

donde w_k es ruido blanco de valor promedio nulo y v_k es ruido blanco de valor promedio nulo. Siendo conocida la acción de control u_k, la estimación del estado anterior x_k, la salida del sistema y_k, $Q_{0,k}$, que es la matriz de covarianza del modelo y $R_{0,k}$, que es la matriz de covarianza del ruido en las mediciones, es decir el de los sensores, se podrá estimar el estado siguiente.

Para la resolución, el algoritmo del filtro de Kalman contiene dos etapas: predicción y estimación.

1. Predicción

En esta etapa se realiza primero una predicción del estado a priori:

$$x_k^- = A\hat{x}_{k-1} + Bu_{k-1} + E_p\hat{d}_{k-1} \tag{3.68}$$

A continuación se calcula la matriz de covarianza del error asociada a la predicción del estado a priori:

$$P_k^- = AP_{k-1}A^{-1} + Q_{0,k} \tag{3.69}$$

2. Estimación

En la segunda etapa se calcula la ganancia de Kalman K con la siguiente ecuación:

$$K_k = P_k^- C^T (C P_k^- C^T + R_{0,k})^{-1} \tag{3.70}$$

Seguidamente se realiza una estimación a posteriori:

$$\hat{x}_k = x_k^- + K_k(y_k - C x_k^-) \tag{3.71}$$

Por último, se calcula la covarianza del error asociada a la estimación a posteriori:

$$P_k = (I - K_k C) P_k^- \tag{3.72}$$

También se dará uso del filtro de Kalman para la estimación de la perturbación. La ecuación final que se obtiene para su cálculo será:

$$\hat{d}_{k+1} = \hat{d}_k + L_d(y_k - C\hat{x}_k) \tag{3.73}$$

Código 3.3 Cálculo del OBS.

```
Ro=I;
Qo=diag(1);
xpr=A*x_filtrada+B*u+Ep*d_filtrada;
Ppr=Qo+A*Pest*A';
Ko=Ppr*C'/(C*Ppr*C'+ Ro);
x_filtrada=xpr+Ko*(yk-C*xpr);
Pest=(I-Ko*C)*Ppred;
d_filtrada=d_filtrada+[-1 0 0]*(C*xk-C*x_filtrada);
```

3.3.3 Filtro de referencias

Con este bloque se desea suavizar el cambio de referencia. Si no se añadiera este filtro se tendría un escalón. Sin embargo, usando un filtro de primer orden, el cambio de referencia se producirá de forma exponencial hacia la nueva referencia. Se puede definir un filtro de primer orden como un simple retardo de la señal. En su forma diferencial, el filtro se representa como:

$$\tau_f \frac{dr}{dt} + r_f(t) = r(t) \tag{3.74}$$

Si esta ecuación se pasa a tiempo discreto queda expresada de la siguiente manera:

$$\tau_f \frac{r_f(t) - r_f(t-1)}{\triangle t} + r_f(t) = r(t) \tag{3.75}$$

Para la definición de este filtro se usa un parámetro llamado α cuyos valores están comprendidos entre 0 y 1 y cuya fórmula será

$$\alpha = \frac{1}{\frac{\tau_f}{\triangle t} + 1} \tag{3.76}$$

Por lo tanto la ecuación final obtenida tendrá la siguiente formulación

$$r_{f,k+1} = \alpha r + (1 - \alpha) r_{f,k} \tag{3.77}$$

Para poder observar cómo afecta el cambio del parámetro α si existe un cambio en la referencia. Se realiza

un pequeño programa en Matlab que simule el comportamiento exponencial que tendrá ahora este cambio de referencia.

Se comprueba que a medida que el valor de α disminuye, el suavizado de la gráfica es mayor, haciendo que la dinámica también sea más lenta ya que tardará más tiempo en alcanzar la referencia exigida inicialmente a la entrada. Este parámetro será ajustado inicialmente.

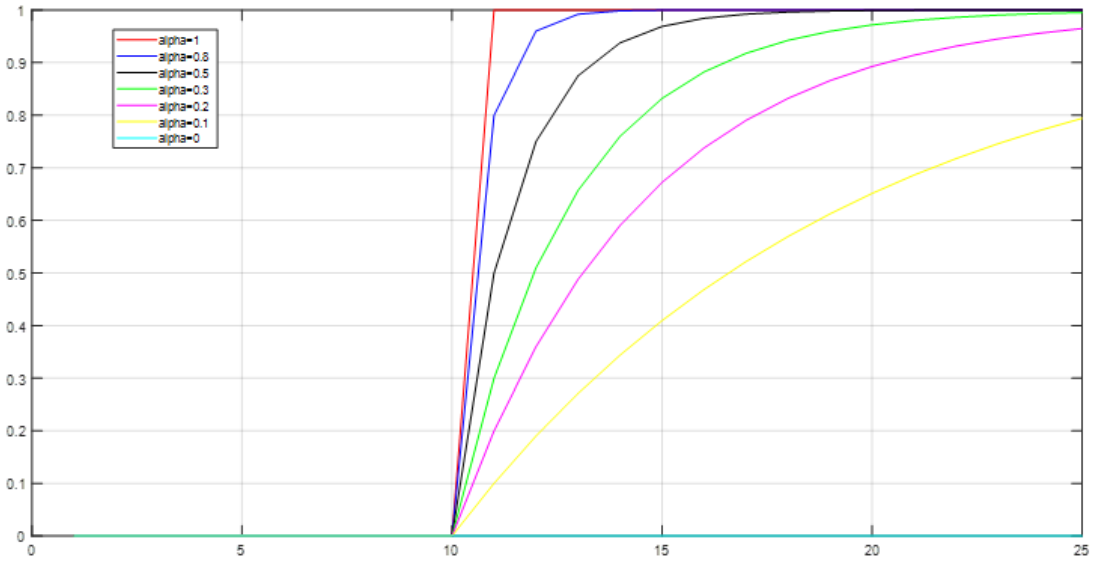


Figura 3.5 Gráfica del filtro con distinta α.

A continuación se muestra parte del código utilizado para generar las gráficas. En particular se observa el usado para un valor de α igual a la unidad. Este tipo de filtro es muy utilizado en los controladores.

Código 3.4 Cálculo de las gráficas del filtro.

```
alfa=1;
r1=0;
r2=[];
p=0;
for r=0:1
for i=1+p:10+3/2*p
r2=[r2; alfa*r+(1-alfa)*r1];
r1=r2(i);
end
p=10;
end
plot(1:25,r2');
```

4 Simulación en Matlab

Una de las virtudes del ingeniero es la eficiencia.

Guang Tse

En este Capítulo se explicará cómo afectan los valores de los parámetros al sistema. Se dará un valor ajustado a cada uno y se podrán observar los resultados obtenidos a través de las gráficas simuladas en Matlab.

Las variables que se muestran en las gráficas son las del vector z, que son ϕ, $\dot{\phi}$, $\dot{\theta}$ y u que es $\ddot{\theta}$. Es decir, se representa el ángulo de inclinación, la variación del ángulo de inclinación, la velocidad de la rueda y la aceleración de la misma.

4.1 Ajuste de los parámetros del sistema

Primero se dará una explicación de los valores elegidos para los parámetros del sistema en cada uno de los puntos de este apartado. Entre los bloques donde se deben ajustar los parámetros está tanto el MPC como el OBS y el SSTO, ya que todos tienen matrices de ponderación.

4.1.1 Horizonte de predicción N

Entre los parámetros que se pueden ajustar en este sistema está el horizonte de predicción, N. Este puede afectar al coste de ejecución del algoritmo, ya que a medida que aumenta su valor también aumenta el coste de ejecución. Esto se debe a que si se incrementa N, las dimensiones de las matrices del problema se agrandan considerablemente.

A su vez, si N es demasiado pequeño, el sistema podría no estabilizarse o que la actuación llegue al límite en algún instante. Si el sistema no se estabiliza esto hace que el coste de ejecución aumente también de forma notable. El algoritmo FISTA en este caso al no encontrar la solución llega al máximo número de iteraciones. Por esta razón se ve incrementado el tiempo que necesita para el cálculo de la solución que además no será óptima.

Por lo tanto, el horizonte de predicción debe tener el mínimo valor posible para disminuir el coste de ejecución. Pero dentro de ese valor mínimo, el sistema tiene que alcanzar el equilibrio sin que la acción de control llega a saturarse. Teniendo esto en cuenta, el valor final de N es 20.

4.1.2 Matrices Q y R del MPC

Entre las variables del sistema que interesa más su control se tiene ϕ y $\dot{\theta}$, es decir el ángulo de inclinación del vehículo autoequilibrado y la velocidad de giro de la rueda. Es por esta razón que se debe dar más peso en la matriz de ponderación Q a estas variables y no tanto a la variable $\dot{\phi}$.

La relación entre Q y R debe ser tal que la respuesta transitoria sea lo menor posible. Además el esfuerzo de control no puede sobrepasar, ni tener un valor demasiado cercano al límite del mismo. Por lo tanto se busca el equilibrio entre rapidez del sistema en cuanto a alcanzar el régimen permanente y no acercarse demasiado al máximo valor impuesto en la restricción del esfuerzo de control.

Finalmente, los valores de las matrices de ponderación Q y R serán:

$$Q = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10 \end{bmatrix} \qquad R = 0.1$$

4.1.3 Matrices Q_0 y R_0 del OBS

Se sabe que Q_0 es la matriz de covarianza del modelo y que R_0 es la matriz de covarianza del ruido en las mediciones, es decir el ruido de los sensores. Teniendo en cuenta la relación R_0/Q_0, se puede buscar una mayor influencia de la medida de los sensores o una mayor influencia del modelo.

Ya que el modelo es bueno, la influencia de los sensores deberá ser superior, por esta razón la relación R_0/Q_0 deberá ser mayor o al menos igual a la unidad. Siendo los valores tomados:

$$Q_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad R_0 = 10$$

4.1.4 Matrices Q_r y R_r del SSTO

En este caso las matrices Q_r y R_r tienen el mismo significado que en el MPC. Esto es así ya que el bloque SSTO utiliza el mismo modelo para los cálculos. Por lo tanto, Q_r es la matriz que ponderar el error en el estado y R_r la matriz que pondera el error en la señal de control.

La diferencia entre el SSTO y el MPC es que el SSTO busca minimizar la diferencia entre la salida de la planta y la referencia que se desea alcanzar. En cambio el MPC pretende minimizar la diferencia entre el estado de referencia y el estado estimado actual.

Los valores que se le han asignado a estas matrices son:

$$Q_r = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad R_r = 0.1$$

4.1.5 Matriz T del SSTO

En el bloque SSTO se ha añadido una holgura para facilitar la búsqueda de la solución. Esta holgura se define usando la variable h. Esto provoca que el vector z esté modificado al tener que añadirse dicha variable que aparece en la ecuación de minimización. Por lo tanto esta variable tendrá una matriz por la que será multiplicadda siendo esta T.

Para que no surjan problemas y la holgura pueda ser grande, se define un valor que tienda a infinito en esta matriz.

$$T = \begin{bmatrix} 10^8 & 0 \\ 0 & 10^8 \end{bmatrix}$$

4.1.6 Límites del SSTO

Por otra parte, también habría que tener en cuenta que al cambiar la dimensión de z también cambia la dimensión del vector de los límites de las variables. Se añadirá en consecuencia un valor límite a la variable h.

Se disminuyen los valores límites tanto del estado como de la actuación. Sin embargo, se usará un valor límite de la holgura que tienda a infinito.

$$LB = \begin{pmatrix} -20 \cdot \frac{\pi}{180} \\ -2 \\ -9 \\ -100 \\ -10^8 \end{pmatrix} \qquad UB = \begin{pmatrix} 20 \cdot \frac{\pi}{180} \\ 2 \\ 9 \\ 100 \\ 10^8 \end{pmatrix}$$

4.2 Gráficas

En este apartado se pueden ver los resultados en Matlab. Una vez ajustados los parámetros se comprobará inicialmente que se sigue la referencia y que los cambios en la misma se realizan corectamente.

En cada simulación o experimento se muestra cada una de las curvas de las variables del vector z representadas en la misma gráfica y un conjunto de gráficas con las variables por separado para que puedan ser observadas ajustando su tamaño al límite entre el cual se encuentra la curva de cada una.

4.2.1 Saturación de la actuación y de la velocidad de la rueda

Una primera comprobación de que el algoritmo FISTA se cumple con respecto a las restricciones de la actuación se realiza con este experimento. Para poder verificar el cumplimiento de la actuación se utiliza un horizonte de predicción de 9. De esta forma en las gráficas se puede observar que la actuación satura en ciertos momentos aunque el sistema alcanza el equilibrio. Sin embargo, nunca debería llegar a tener un valor tan alto la acción de control.

También se puede observar que se alcanza el límite de la velocidad de la rueda. Se confirma así que las restricciones de velocidad también se cumplen con el algoritmo de control.

En este ejemplo se busca una velocidad de la rueda de 2 rad/s.

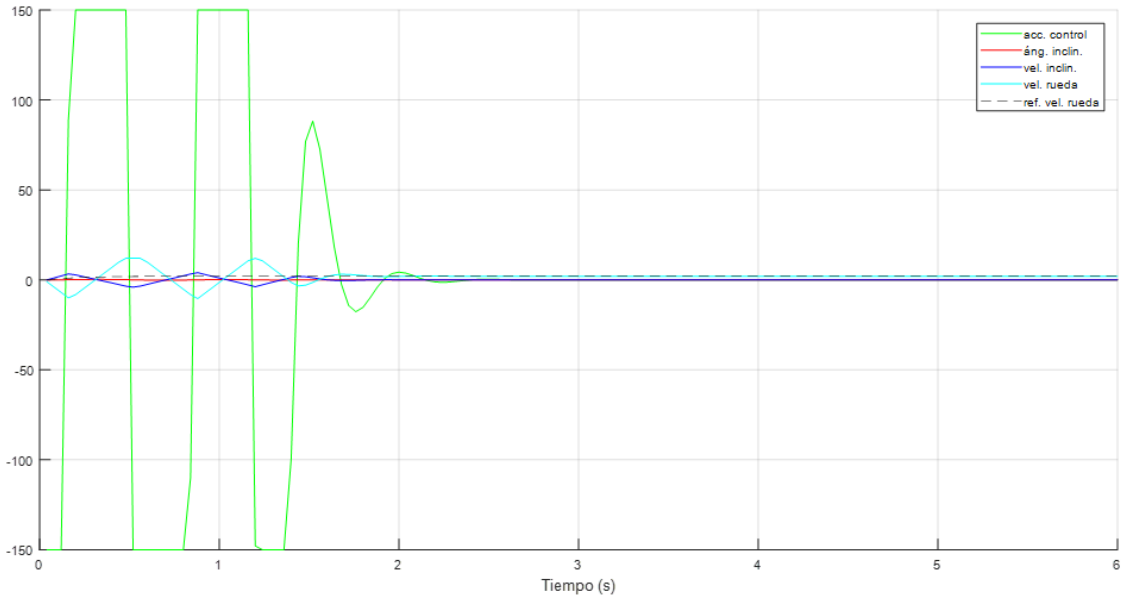
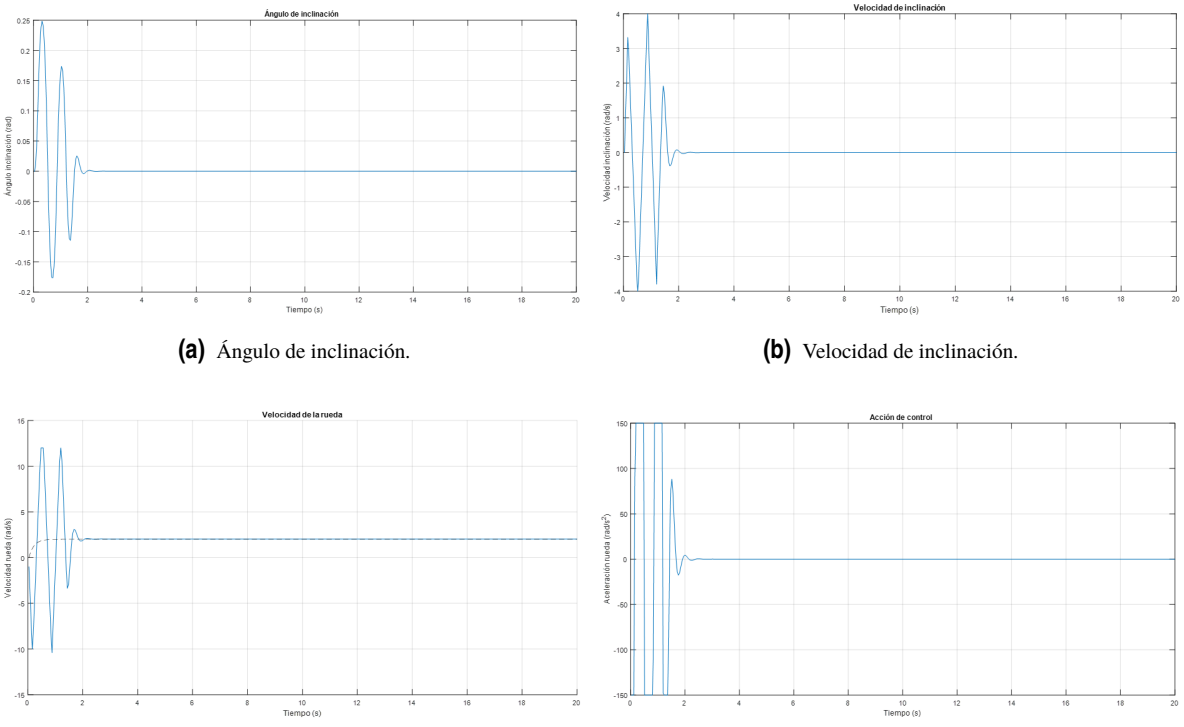


Figura 4.1 Saturación de la actuación.



(a) Ángulo de inclinación.

(b) Velocidad de inclinación.

(c) Velocidad de la rueda.

(d) Acción de control.

Figura 4.2 Saturación de la actuación con referencia de velocidad igual a 2 rad/s.

4.2.2 Gráficas con perturbación nula

Inicialmente se realizarán experimentos donde se compruebe el seguimiento de la referencia, es decir de la velocidad de la rueda.

Se van a hacer cuatro comprobaciones diferentes:

1. Vehículo autoequilibrado parado manteniendo su posición con velocidad de la rueda nula.
2. Vehículo autoequilibrado con velocidad de la rueda positiva.
3. Vehículo autoequilibrado con velocidad de la rueda negativa.
4. Varios cambios de referencia tanto positiva como negativa en la velocidad de la rueda.

Referencia de velocidad nula

En este caso se busca que el comportamiento del vehículo autoequilibrado sea permanecer parado, ya que la referencia de velocidad de la rueda es nula.

Como resultado se debe obtener este seguimiento de velocidad a cero, además de que se alcance el equilibrio inestable del sistema. Esto quiere decir que el ángulo de inclinación deberá de estabilizarse siendo su valor igual a cero.

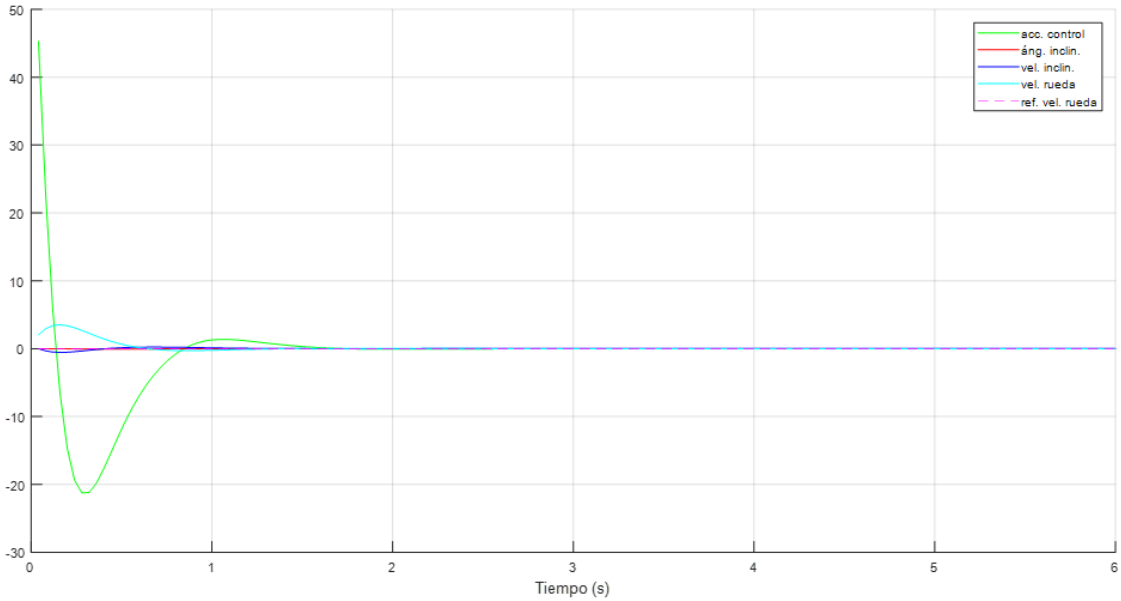
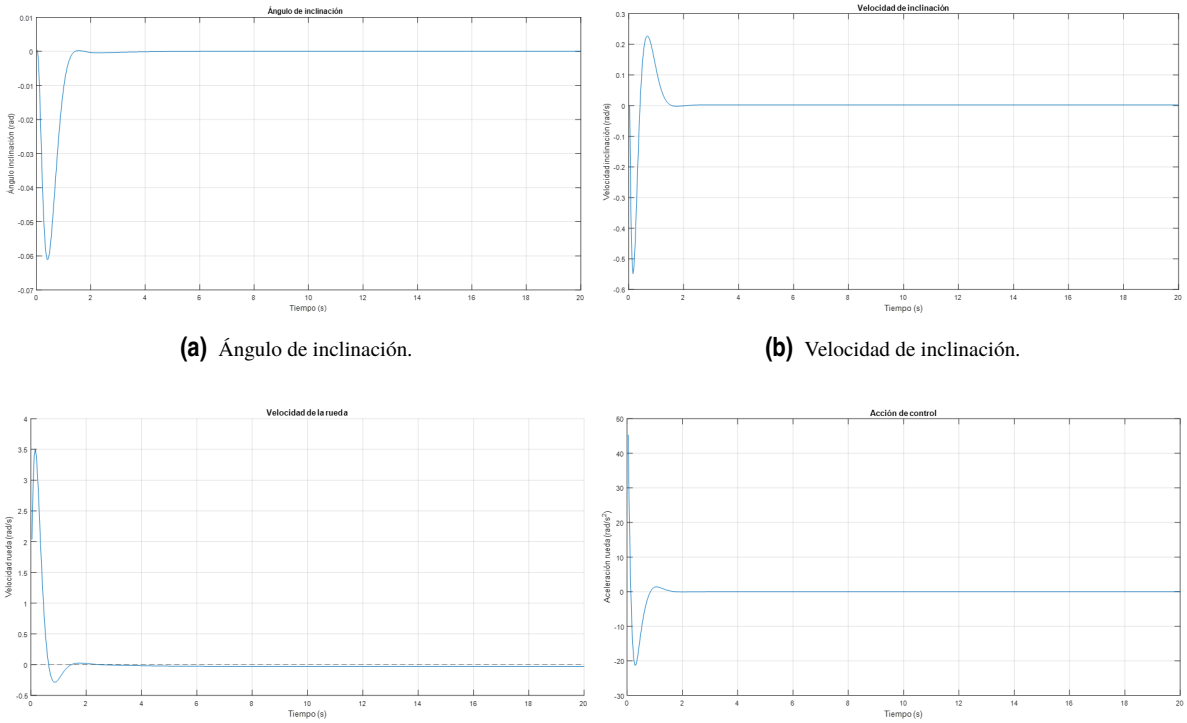


Figura 4.3 Referencia de velocidad nula y sin perturbación.



(a) Ángulo de inclinación.

(b) Velocidad de inclinación.

(c) Velocidad de la rueda.

(d) Acción de control.

Figura 4.4 Referencia nula y perturbación nula.

Referencia de velocidad positiva

En esta simulación se busca verificar que el sistema es capaz de mantener cierta velocidad positiva en la rueda en lugar de permanecer quieto. La referencia que se le impone es una velocidad de 2 rad/s.

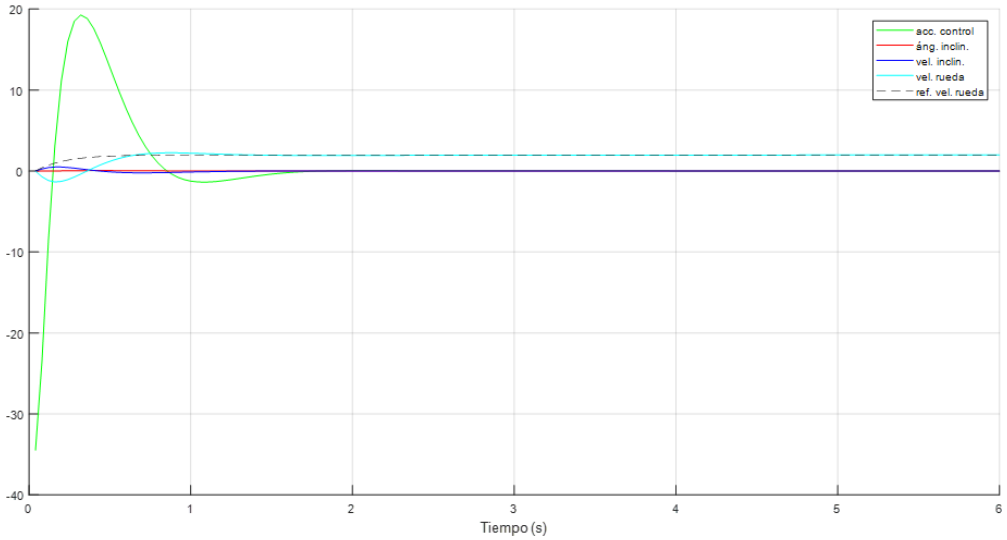
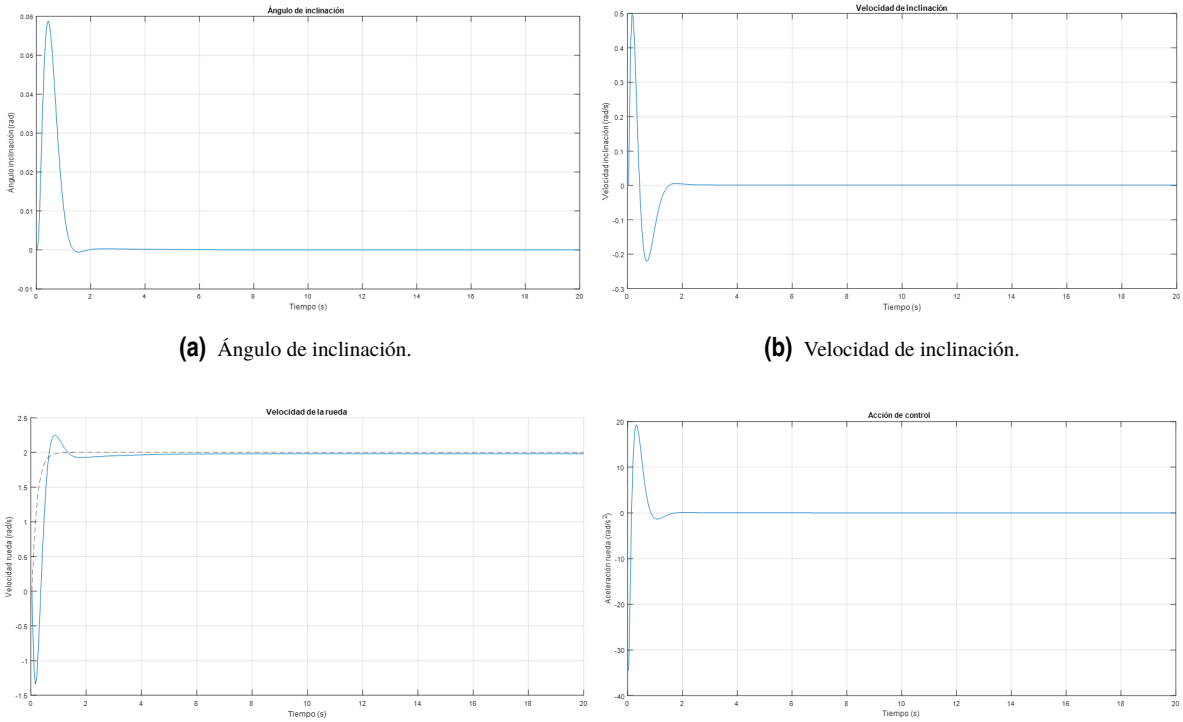


Figura 4.5 Referencia de velocidad igual a dos y sin perturbación.



(a) Ángulo de inclinación.

(b) Velocidad de inclinación.

(c) Velocidad de la rueda.

(d) Acción de control.

Figura 4.6 Referencia igual a dos y perturbación nula.

Referencia de velocidad negativa

Por último, se comprueba que el control también funcione para velocidades negativas. En este caso la simulación se ha realizado para una velocidad de la rueda de -1 rad/s.

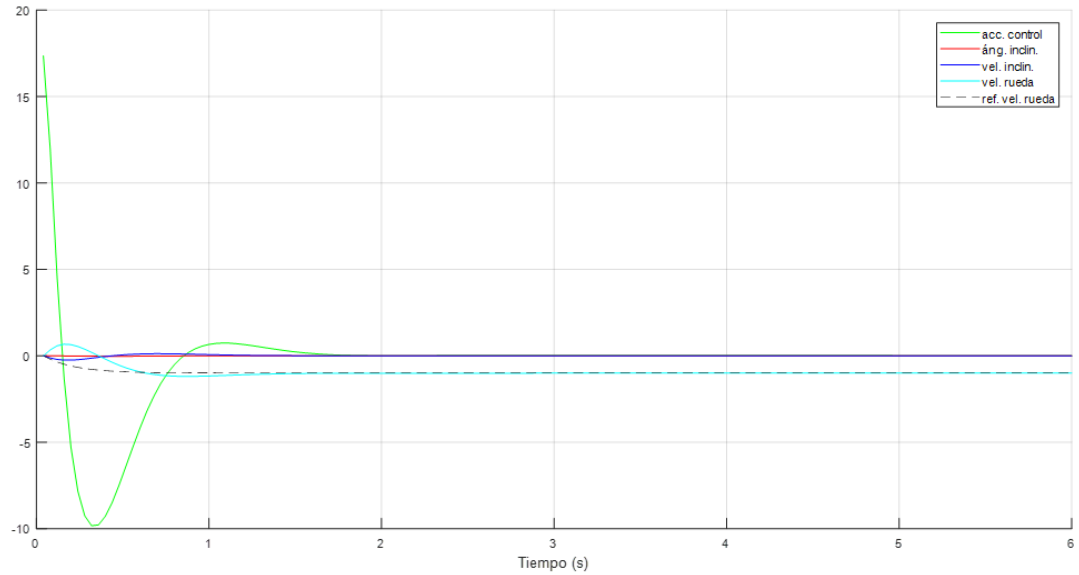
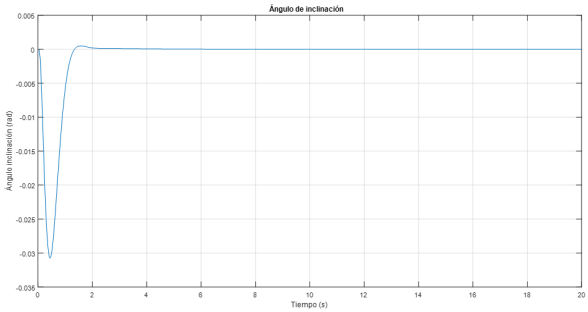
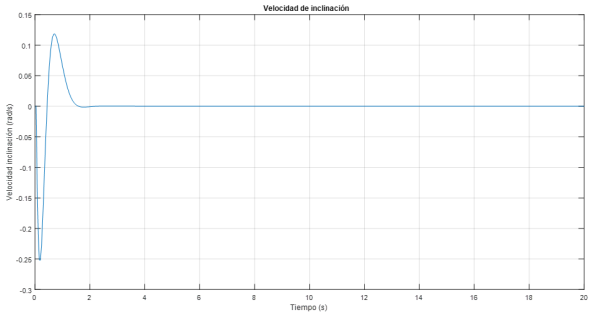


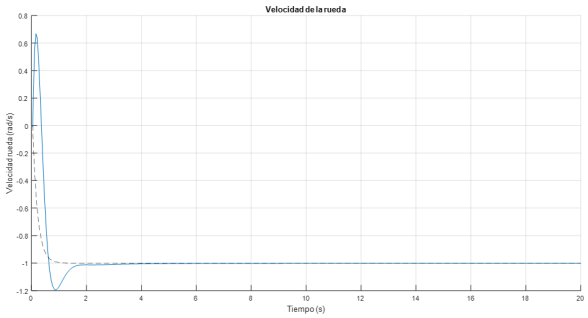
Figura 4.7 Referencia de velocidad igual a menos uno y sin perturbación.



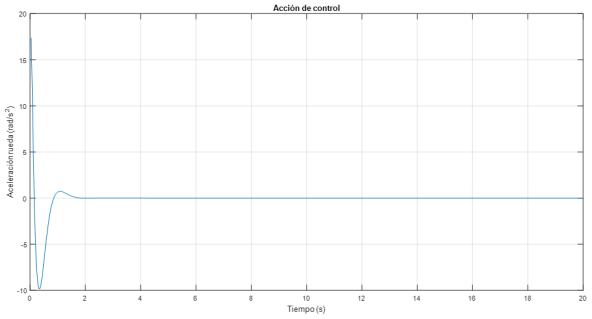
(a) Ángulo de inclinación.



(b) Velocidad de inclinación.



(c) Velocidad de la rueda.



(d) Acción de control.

Figura 4.8 Referencia igual a menos uno y perturbación nula.

Varios cambios de referencia

En este apartado se cambiará el valor de referencia dos veces. Primero se exige una referencia positiva de 1 rad/s y a continuación, pasados 8 s se exige una referencia de -1 rad/s.

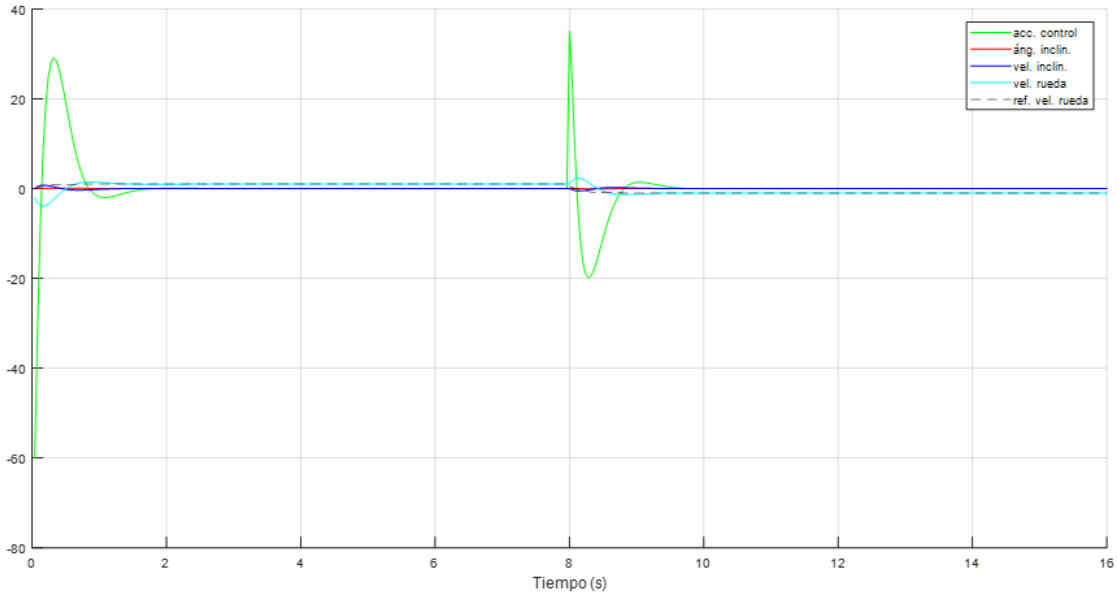
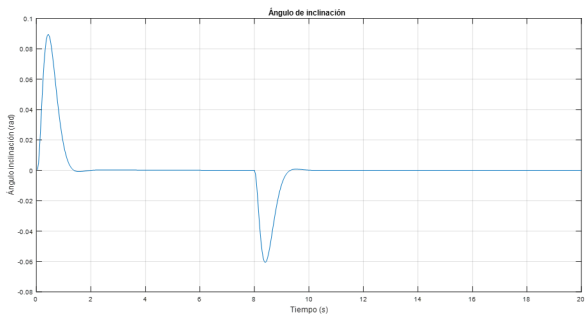
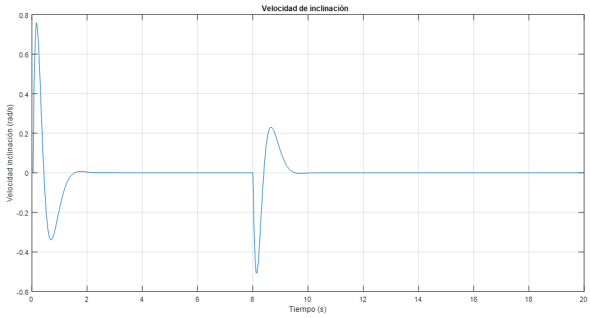


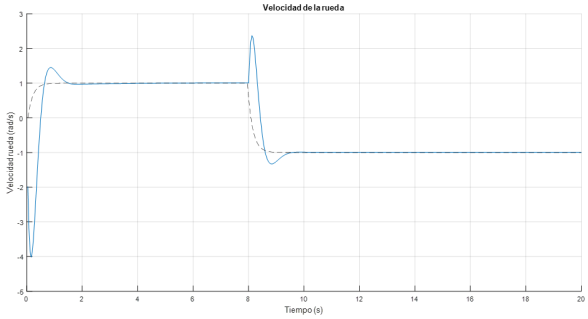
Figura 4.9 Referencia de velocidad de uno a menos uno y sin perturbación.



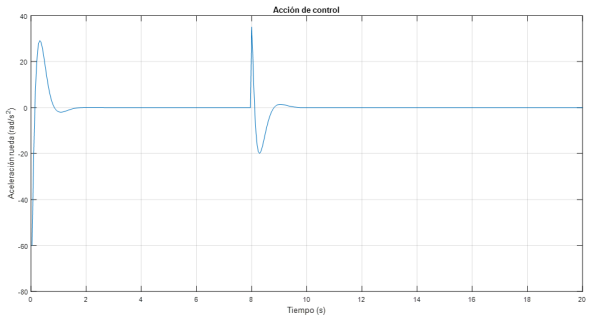
(a) Ángulo de inclinación.



(b) Velocidad de inclinación.



(c) Velocidad de la rueda.



(d) Acción de control.

Figura 4.10 Referencia de uno a menos uno y perturbación nula.

4.2.3 Gráficas con perturbación distinta a cero

En este punto se busca que el ángulo de inclinación sea contrario a la perturbación alcanzando así el equilibrio, ya que se ha modificado el punto donde se encuentra el centro de masa.

Si no se tratara esta perturbación, el segway no podría seguir la referencia de velocidad ya que este tendría que moverse para no perder el equilibrio. Esto es así porque el sistema buscaría un ángulo de inclinación nulo, tomando esto como el punto de equilibrio. Sin embargo, en estos casos el punto de equilibrio se encuentra en el ángulo de inclinación provocado por la perturbación y con signo contrario a la misma.

Referencia de velocidad nula

Se sabe que el sistema tiene una perturbación provocada por la diferencia entre el centro de gravedad y el centro geométrico. Se comprueba de forma experimental que el ángulo que provoca esto es de 0.03 *rad*. Por esta razón para que el vehículo autoequilibrado permanezca en equilibrio deberá estar inclinado -0.03 *rad*.

Por otra parte, se añade que también haga un seguimiento de velocidad nula, es decir que permanezca parado.

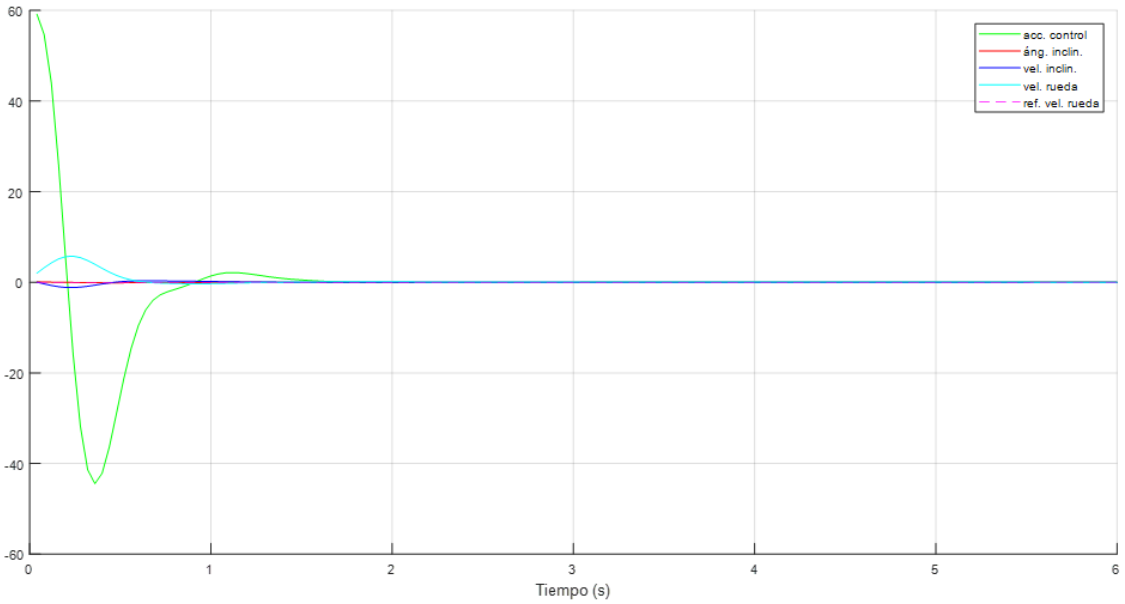
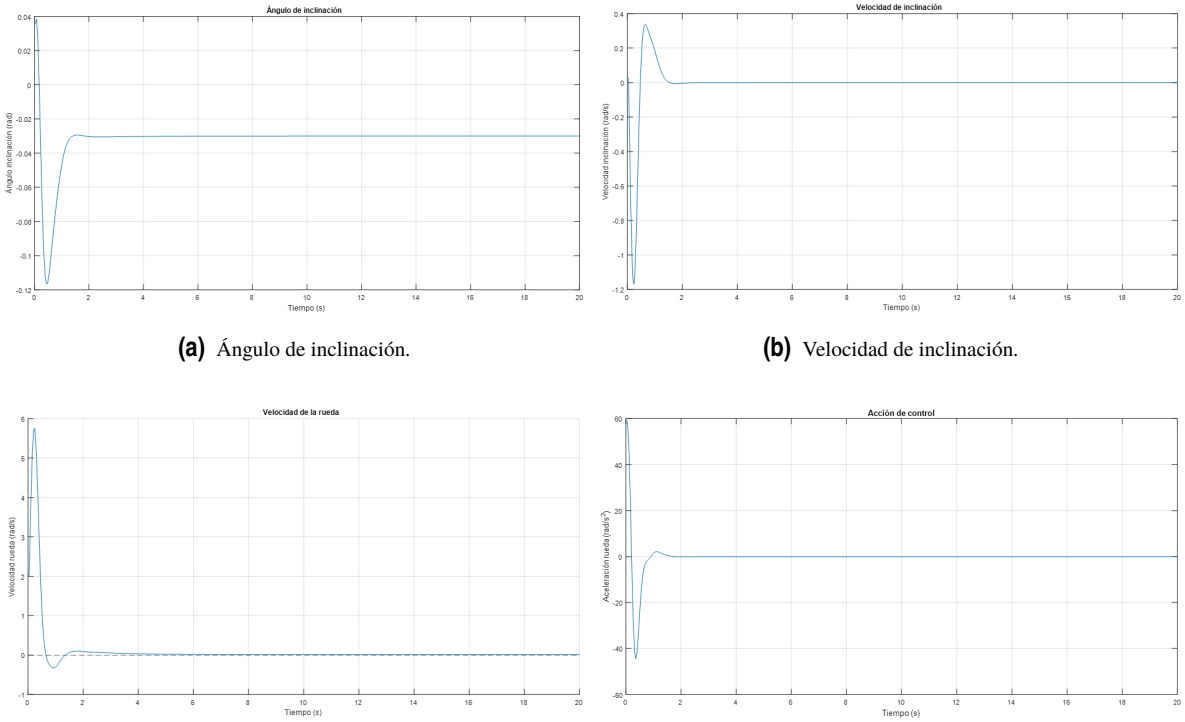


Figura 4.11 Referencia de velocidad nula y con perturbación.

Estos valores antes descritos se pueden observar mejor en las gráficas por separado de cada una de las variables. Se comprueba que el ángulo de inclinación permanece en un valor de -0.03 *rad* y que la velocidad de la rueda alcanza una velocidad nula.



(a) Ángulo de inclinación.

(b) Velocidad de inclinación.

(c) Velocidad de la rueda.

(d) Acción de control.

Figura 4.12 Referencia nula y perturbación igual a 0.03 *rad*.

5 Implementación en Arduino

El objeto final de este trabajo es la implementación del código generado en Matlab en una placa de Arduino. El vehículo autoequilibrado ya se encuentra montado y contiene una placa Arduino Mega 2560, por lo tanto se volcará el código en este dispositivo.

El microcontrolador es el que recibe los datos de los sensores y a partir de eso realiza las operaciones necesarias para el calculo de la señal de entrada a los actuadores.

5.1 Características de la placa

Entre los elementos más interesantes de la placa se puede destacar su microcontrolador, el tipo de comunicaciones que se pueden realizar y las características electrónicas y eléctricas del dispositivo.

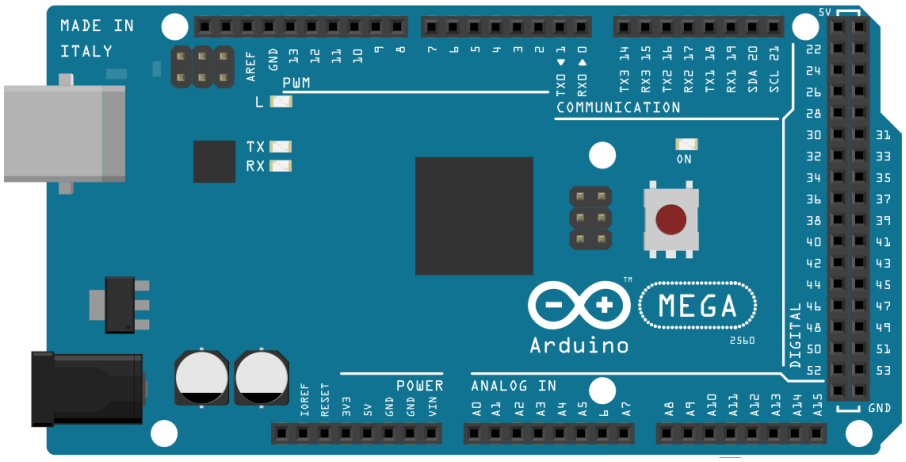


Figura 5.1 Placa Arduino Mega 2560.

Esta unidad controladora está basada en el microcontrolador Atmel ATmega2560. Este chip es de 8 bits y es capaz de trabajar con una SRAM, una EEPROM y una memoria flash. Esta placa está basada en arquitectura AVR. A continucación se muestran las características del microcontrolador.

En cuanto a las comunicaciones, la placa dispone de un conector USB. El código se cargará usando este tipo de comunicación desde el ordenador a la placa. En cuanto a la alimentación del chip, existen dos maneras. Se puede alimentar a través del conector jack, el cual utiliza el segway de forma normal o por el USB, los casos en los que se carga el código a la placa.

Tabla 5.1 Características del microcontrolador.

Característica	Medida (unidades)
Microcontrolador	ATmega2560
Memoria flash	$256KB$
SRAM	$8KB$
EEPROM	$4KB$
Frecuencia de reloj	$16MHz$
Temporizadores	6 Timers

Por otra parte, también contiene una serie de pines que realizan otras comunicaciones. Estas se pueden observar en la tabla 5.2.

Tabla 5.2 Características electrónicas.

Característica	Medida (unidades)
UARTs	4 pines
SPI	5 pines
TWI (I2C)	2 pines

Por último, faltan mencionar las características electrónicas del dispositivo que se puede apreciar en la tabla 5.3.

Tabla 5.3 Características electrónicas.

Característica	Medida (unidades)
Tensión nominal	$5V$
Tensión de entrada (uso normal)	$7-12V$
Tensión de entrada (límite)	$6-20V$
Pines de E/S digital	54 pines (15 para PWM)
Pines de E analógica	16 pines
Corriente continua por pin E/S digital	$40mA$
Corriente continua por pin 3V3	$50mA$

Pines de la placa utilizados

1. Pines PWM (utilizados por los motores): 7 y 9.
2. Pines de interrupciones externas (utilizados por los encoders): 2, 3, 18 y 19.
3. Pines de comunicación I2C: 20 y 21.
4. Pines de comunicación por puerto serie (utilizados por Bluetooth): 0 y 1.
5. Pines de salidas digitales (para el motor): 40, 41, 42 y 43.
6. Temporizadores: 0, 1, 2, 3 y 4.
7. Pines de alimentación auxiliar: 3V3, 5 V y GND.

5.2 Desarrollo del código

En este apartado se explica cómo se va a desarrollar el código en Arduino a partir del código en Matlab. Hay que tener en cuenta que Arduino tiene una serie de limitaciones hardware. Entre estas limitaciones hay que tener en cuenta el límite de la memoria de esta placa, lo que compromete la forma de desarrollar el código, que debe computar el mínimo número de operaciones o que estas operaciones sean realizadas de la forma más rápida posible.

En lo puntos que se muestran a continuación, se explica cómo se ha conseguido utilizar menos memoria y reducir la cantidad de cálculos.

5.2.1 Reducción del número de operaciones y datos

El Algoritmo FISTA contiene muchas operaciones de multiplicaciones de matrices que en Matlab no constituyen un problema en su programación, sin embargo en Arduino que es una programación más parecida a C, hay que definir cómo se ejecuta la operación multiplicación de matrices. Es decir, hay que establecer la multiplicación de las filas de la primera matriz con las columnas de la segunda matriz y cada elemento sumarlo en su fila y columna correspondiente.

Es fácil comprobar que una de las multiplicaciones con matrices grandes del algoritmo son con la matriz E. Analizando esta matriz, se puede observar la gran cantidad de ceros que encierra y también su extenso tamaño debido al valor del horizonte.

Para poder reducir el número de cálculos, lo lógico sería no tener que computar multiplicaciones por cero. Por esta razón se cogerá una parte de la matriz la cual contenga la parte no nula de la misma y que además esta se repita a lo largo de la matriz, es decir a medida que se avanza en la diagonal de la misma.

Si se examina, se puede notar que hay una zona de ella que se repite. Aparece constantemente la estructura $[AB-I]$ en cada fila. Por lo tanto, se va a modificar esta matriz para que se utilice únicamente esa región, lo que consigue que haya menos operaciones además de necesitar menos memoria ya que se disminuye considerablemente el tamaño de E. La matriz E pasaría de tener una dimensión de $((N+1)nxN(m+n))$ a $nx(2n+m)$.

Sin embargo, también es cierto que la matriz E contiene, en una de sus filas estructuras diferentes a la mencionada anteriormente. Entre ellas, $[I]$ y $[AB]$. Se sabe que la matriz E será usada para el cálculo de la norma para comprobar la tolerancia, $\|EZ(y_k)-b-d\| \leq TOL$.

En el programa se construirá la matriz $EZ(y_k)-b-d$, donde d está referido a la perturbación. Ya que $EZ(y_k)$ puede ser diferente, se definen 3 partes distintas de la matriz y se usarán según la componente del vetor Z que se esté utilizando. No obstante, la base de la que se dispondrá siempre será la estructura primera $[AB-I]$. Ya que si se está en la zona de la matriz identidad no será necesario definirla al contener unos. Y si por el contrario solo se necesita la estructura $[AB]$, esta se reutilizará de la $[AB-I]$.

Queda explicar cómo se obtiene esta región de la matriz. Ya que se busca hacer el menor número posible de cálculos, el cómputo de los valores de esta estructura no lo desarrollará Arduino. Se facilitarán los valores de las componentes fuera de línea, es decir fuera de Arduino. La placa tendrá directamente cada uno de los términos ya definidos.

$$E = \begin{pmatrix} I & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ A & B & -I & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & A & B & -I & \cdots & 0 & 0 \\ \vdots & \vdots & & & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & A & B \end{pmatrix} \qquad \longrightarrow \qquad E = \begin{pmatrix} A & B & -I \end{pmatrix}$$

$$\begin{pmatrix} A & B & -I \end{pmatrix} = \begin{bmatrix} 1.0321 & 0.0404 & 0 & -2.8959 \cdot 10^{-4} & -1 & 0 & 0 \\ 1.6121 & 1.0321 & 0 & -0.0146 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0.04 & 0 & 0 & -1 \end{bmatrix}$$

Para comprobar el uso de estas tres zonas de la matriz, se muestra el código comentado en el que se observa la construcción de la matriz $EZ(y_k) - b - d$.

Código 5.1 Cálculo de la matriz $EZ(y_k) - b - d$.

```
for(i=0;i<n;i++) {
//Primera parte de E sería la matriz identidad, I*zmpc-b-d, siendo b x_filtrada
Ez_b_d[i]=Datos->zmpc[i]-b[i]-d[i];
//El final de la matriz E sería [A B], [A B]*zmpc-b-d, siendo b xr
aux2[i]=0;
for(j=0;j<n+m;j++) {
aux2[i]+=Datos->E[i*(2*n+m) + j]*Datos->zmpc[(n+m)*(N-1)+j];
}
Ez_b_d[n*N+i]=aux2[i]-b[i+N*n]-d[i+N*n];
}
for(ii=n,jj=0;ii<n*N;ii+=n,jj+=n+m) {
for(i=0;i<n;i++) {
//El resto de la matriz E [A B -I]zmpc-b-d, siendo b cero
aux2[i]=0;
for(j=0;j<2*n+m;j++) {
aux2[i]+=Datos->E[i*(2*n+m) + j]*Datos->zmpc[jj+j];
}
Ez_b_d[ii+i]=aux2[i]-d[ii+i];
}
}
for(i=0;i<(N+1)*n;i++){
aux1+=Ez_b_d[i]*Ez_b_d[i];
}
```

Por otra parte, también se utiliza en el algoritmo FISTA la traspuesta de E. Este cálculo en lugar de ser realizado por el Arduino, se obtendrá fuera de línea. Es decir, Arduino no tendrá que hacer el cálculo de la traspuesta, consiguiendo así ahorrar el número de operaciones. Además al igual que la matriz E, para esta también se definirá dentro de la placa solo una pequeña parte de la misma. De esta forma la matriz E_t ya no tendría la dimensión de $(N(m+n)x(N+1)n)$ sino que su tamaño se transformaría en $2nx(n+m)$.

Se pueden observar dos tipos distintos de región $\begin{pmatrix} I & A_t \\ 0 & B_t \end{pmatrix}$ y $\begin{pmatrix} -I & A_t \\ 0 & B_t \end{pmatrix}$. Ya que no existe una gran diferencia entre ellas, la única región que se le da a Arduino es la primera, de la que se conoce que solo será usada una vez. Para el uso de la otra estructura, simplemente se multiplicará por -1 en la matriz identidad. Esto se puede observar en el código que aparece al final de este apartado.

Por lo tanto fuera de línea, se definirá la nueva matriz E_t como:

$$E_t = \begin{pmatrix} I & A_t & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & B_t & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & -I & A_t & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & B_t & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & & & & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & -I & A_t \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & B_t \end{pmatrix} \longrightarrow E_t = \begin{pmatrix} I & A_t \\ 0 & B_t \end{pmatrix}$$

$$\begin{pmatrix} I & A_t \\ 0 & B_t \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1.0321 & 1.6121 & 0 \\ 0 & 1 & 0 & 0.0404 & 1.0321 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & -2.8959 \cdot 10^{-4} & -0.0146 & 0.04 \end{bmatrix}$$

donde A_t y B_t son las traspuestas de A_d y B_d respectivamente.

Ya que el uso de la matriz E_t requiere conocer también otros vectores, se explica cómo utilizar menos memoria cambiando el tamaño de los mismos. El cálculo que se efectúa usando E_t es $f = f1 - E_t y$. Se necesita por tanto el vector $f1$, al cual, en la teoría, se le ha nombrado como f. Sin embargo, el vector f se definirá en el código de Arduino como $f1$ siendo en realidad f la siguiente expresión: $f = f1 - E_t y$. Para reducir la cantidad de cálculos se disminuye también el tamaño de este vector que en vez de ser de $(N(m+n))$, tiene una dimensión de $n+m$. Su transformación quedará:

$$f1 = \begin{pmatrix} -Qx_r \\ -Ru_r \\ -Qx_r \\ -Ru_r \\ \vdots \\ -Qx_r \\ -Ru_r \end{pmatrix} \longrightarrow f1 = \begin{pmatrix} -Qx_r \\ -Ru_r \end{pmatrix}$$

Habiendo realizado ya las transformaciones para la disminución del tamaño de matrices y vectores, el cálculo final de f será:

Código 5.2 Cálculo de f.

```
// Cálculo de f donde f=f1-E'*y
neg=0;
for(ii=0,jj=0;ii<N*(m+n);ii+=n+m,jj+=n) {
for(i=0;i<n+m;i++) {
aux[i]=0;
for(j=0;j<2*n;j++) {
if(j<n && neg==1){
//segunda estructura, parte de la matriz identidad negativa
aux[i]+=-Datos->Et[i*2*n + j]*y[jj+j];
}
//primera estructura entera, y parte positiva de la segunda estructura
else{
aux[i]+=Datos->Et[i*2*n + j]*y[jj+j];
}
}
```

```
f[ii+i]=f1[i]-aux[i];
}
neg=1;
}
```

También se disminuye el tamaño de H, donde solo se definirán una única vez las matrices Q y R.

$$H = \begin{pmatrix} Q & 0 & 0 & 0 & \cdots & 0 \\ 0 & R & 0 & 0 & \cdots & 0 \\ 0 & 0 & Q & 0 & \cdots & 0 \\ 0 & 0 & 0 & R & \cdots & 0 \\ \vdots & \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & Q & 0 \\ 0 & 0 & 0 & \cdots & 0 & R \end{pmatrix} \longrightarrow H = \begin{pmatrix} Q & 0 \\ 0 & R \end{pmatrix}$$

Finalmente, quedaría añadir que para disminuir la cantidad de datos que adquiere Arduino para realizar los cálculos, se guardan solo los datos de los límites superiores positivos de las restricciones del espacio de estados y de la acción de control. Esto es posible ya que los valores de los límites inferiores son iguales pero de signo contrario. La trasformación que se realiza es la que se muestra a continuación donde LB es el límite inferior de la restricción y UB el inferior.

$$LB = \begin{pmatrix} LBx \\ LBu \\ LBx \\ LBu \\ \vdots \\ LBx \\ LBu \end{pmatrix} \quad UB = \begin{pmatrix} UBx \\ UBu \\ UBx \\ UBu \\ \vdots \\ UBx \\ UBu \end{pmatrix} \longrightarrow lim = \begin{pmatrix} UBx \\ UBu \end{pmatrix}$$

5.2.2 Generación de matrices para Arduino

Como se ha mencionado antes, habrá matrices que no las calculará Arduino, sino que serán directamente definidos sus valores en la placa. Esto se ha podido comprobar en E, E_t y lim. Sin embargo, estas no son las matrices más grandes que contiene el algoritmo.

Existe otra matriz cuyas dimensiones hacen inviable su cálculo durante la resolución del control, Esta es W^{-1}. Se define W como $W = EH^{-1}E^T$. Esta matriz requiere el cálculo de una matriz inversa, una matriz traspuesta y la multiplicación de matrices de dimensiones considerables. Por otra parte, no es W lo que se necesita sino su inversa. Lo que significaría añadir otra operación compleja de programar y con la necesidad de un amplio tiempo de cálculo en Arduino.

La solución a este problema es realizar el cálculo de las componentes de esta matriz en Matlab y pasar a Arduino directamente sus valores. Ya que en Matlab en solo una línea de código se consigue el valor de W^{-1}.

Por otra parte, no se puede reducir su tamaño ya que a pesar de que E y H sean matrices con muchos ceros, el resultado final de las operaciones, no es una matriz que contenga muchos ceros ni un patrón visible. Por lo que resultaría tedioso copiar y pegar dicha matriz en el código de Arduino. Es por esta razón, por la que además se crea un código en Matlab que genera el código de Arduino donde está definida la matriz W^{-1}.

Matriz W^{-1}

La dimensión de dicha matriz depende en cuanto a su tamaño de N y en cuanto a su valor de Q, R, A_d y B_d. Esto es así ya que los cálculos son realizados con las matrices H y E que contienen Q, R, A_d y B_d respectivamente. Por lo tanto el tamaño de W^{-1} dependerá de estas matrices también siendo su tamaño final $((N+1)nx(N+1)n)$. Y si $N = 20$ y $n = 3$ sería una matriz cuadrada de 63x63.

A continuación se muestra el código de Matlab utilizado para generar el código en Arduino con la matriz W^{-1} ya calculada. Por lo tanto en Arduino solo entrarán los valores de esa matriz, ya que el cómputo de sus elementos lo realiza Matlab.

Código 5.3 Cálculo de la matriz W^{-1} (Qfista).

```
datos=fopen('datos.txt','w+');
H=zeros(N*(n+m));
E=zeros((N+1)*n,(n+m)*N);
I=eye(n);
aux=0;
for cont=1:N
for i=1:n
H(i+n*(cont-1)+aux,i+n*(cont-1)+aux)=Q(i,i);
end
for i=n+1:m+n
H(i+n*(cont-1)+aux,i+n*(cont-1)+aux)=R;
aux=aux+1;
end
end
aux=0;
for i=1:N
E(n+1+n*(i-1):2*n+n*(i-1),i+n*(i-1):n+n*(i-1)+aux)=Ad;
E(n+1+n*(i-1):2*n+n*(i-1),i+n*(i-1)+n:m+n+n*(i-1)+aux)=Bd;
if i<N
E(n+1+n*(i-1):2*n+n*(i-1),i+n*(i-1)+n+m:n+m+n+n*(i-1)+aux)=-I;
end
if i==1
E(i:n,i:n)=I;
end
aux=aux+1;
end

Qfis=[];
Qfis=inv(E/H*E')

for i=0:((N+1)*n-1)
for j=0:((N+1)*n-1)
fprintf(datos,'\t');
fprintf(datos,'(*Planta).Qfista[');
fprintf(datos,'%i]={',i*(N+1)*n+j);
fprintf(datos,'%.4f};\n',Qfis(i+1,j+1));
end
end

fclose(datos);
```

5.2.3 Disminución de la aproximación a cero

En el algortimo FISTA, existe una parte en la que se busca que la restricción de igualdad se cumpla. Es decir que $EZ = b + d$ o lo que es lo mismo que $\|EZ(y_k) - b - d\| = 0$. Ya que esto resultaría demasiado complejo, se busca que esta expresión se aproxime a cero por lo tanto se usa un valor llamado tolerancia. Esta tolerancia será un valor cercano a cero.

En el caso de Matlab, es fácil poder exigir que el valor de la tolerancia sea muy pequeño. Esto es así debido a que Matlab es capaz de trabajar con una cantidad muy grande de decimales, hasta 15 posiciones decimales. Esto hace que Matlab sea capaz de ofrecer un resultado mejor. Por lo tanto la convergencia del algoritmo hacia un valor próximo a cero es más sencilla. Es decir se consigue que se cumpla prácticamente la restricción de igualdad.

En Arduino esta convergencia es mucho más compleja. Esto es debido a que la capacidad máxima que tiene Arduino en cuanto al número de decimales exactos es 6. Se puede comprobar que esto es así tanto usando datos de tipo float como usando datos de tipo double. Ambos permiten únicamente 6 decimales exactos. Esto quiere decir que Arduino en sus cálculos usa menos de la mitad de decimales que Matlab. Por otra parte, es cierto que se puede pedir a Arduino que muestre más de 6 decimales, sin embargo se comprobará que no van a coincidir con los decimales que en realidad aparecen en Maltab bien calculados.

Por esta razón la tolerancia en Arduino no puede ser un valor tan pequeño como en Matlab. Si se exigiera un valor demasiado próximo a cero, el algoritmo no convergería. Esto provocaría que hiciera el máximo número de iteraciones y diera como resultado un solución subóptima. Sin embargo, la solución que ofrezca Arduino siempre será menos exacta que Maltab. Aunque la diferencia en cuanto a los valores de los dígitos de los decimales se puede encontrar entre el cuarto, esta solución será válida para el control del Segway. No es necesaria tanta precisión en el sistema real.

Como conclusión, en Maltab se puede exigir una tolerancia de $1 \cdot 10^{-6}$. En cambio, en Arduino, solo se exigirá una tolerancia de $1 \cdot 10^{-2}$. De esta forma se solventaría el problema de la no convergencia en Arduino por tener una restricción demasiado estricta.

5.2.4 Eliminar las raíces cuadradas

Dentro del algoritmo existe el uso de raíces cuadradas, algo que necesita más tiempo de cálculo en comparación a otras operaciones. Como se quiere conseguir poder realizar el control en el menor tiempo posible, es bueno considerar eliminar cálculos con mayor tiempo de cómputo.

El cálculo de la norma es un cálculo en el que al final se necesita hacer la raiz cuadrada, esto se encuentra en el punto del cálculo de la norma, $\|EZ(y_k) - b - d\| \leq TOL$. Es decir $\sqrt{EZ(y_k) - b - d} \leq TOL$. Para que Arduino no tenga que realizar el cálculo de la raiz. Lo que se deberá exigir será $EZ(y_k) - b - d \leq TOL^2$. De esta forma se elimina ese cómputo y en el valor de la tolerancia, se pondrá el valor de la tolerancia al cuadrado directamente calculado. Sin que Arduino tenga que hacer ninguna multiplicación.

Esto no supondrá un gran cambio en la disminución del tiempo de ejecución del algoritmo, pero si se está cercano a alcanzar el tiempo en el que se debe calcular el control, este cambio puede ser decisivo.

5.2.5 Problema

Como ya se ha mencionado en los anteriores apartados, el algoritmo FISTA debe ser operado en el menor tiempo y con los menores recursos posibles. Para ello, se han reducido el número de operaciones y datos, al igual que no se han dado restricciones tan estrictas en la convergencia del algoritmo. También se han eliminado operaciones que podían tardar más tiempo en su cómputo. Sin embargo, a pesar de todas estas implementaciones sigue existiendo el problema del tiempo.

Restricción de tiempo del sistema

Se debe considerar que el sistema real debe realizar el control cada 40 ms. Por esta razón es importante tener en cuenta que el bucle de control no puede sobrepasar esta restricción de tiempo. Si no se tuviera en cuenta esto, el vehículo autoequilibrado no podría conseguir el equilibrio en ningún caso ya que la acción de control no sería conocida en el tiempo requerido.

Esto quiere decir que se está trabajando en un sistema de tiempo real. La respuesta debe ser garantizada dentro de la restricción de tiempo. Por esta razón se realiza la comprobación de que el algoritmo FISTA es calculado en un tiempo menor o igual a 40 ms.

Sin embargo, debido a la gran dimensión de las matrices, Arduino necesita un tiempo bastante superior a 40 ms, siendo el mejor de los casos un cálculo que tarda alrededor del segundo.

5.3 Solución posible

Hasta ahora todas las modificaciones para el menor uso de recursos y el cómputo en menor tiempo han sido implementadas en el Software. Sin embargo, se comprueba que todas estas transformaciones no son suficientes para el cumplimiento de la restricción de tiempo. Esta restricción es imprescindible cumplirla en el sistema real. Si no se cumple el control nunca será calculado en el instante que es indispensable.

Existe otro tipo de solución, esta sería implementar en el vehículo autoequilibrado otro Hardware más adaptado a las necesidades del nuevo tipo de control. Entre las posibles soluciones en cuanto al cambio de Hardware una de ellas sería utilizar una placa de Arduino más adecuada a las exigencias del algortimo FISTA.

Una de las placas que parece ofrecer la solución que se precisa es la Arduino Due.

5.3.1 Comparación Arduino Mega vs Arduino Due

El Arduino Due es idóneo para proyectos que necesitan una alta capacidad de procesamiento, ya que tiene 32 bits, en lugar de 8 bits. Se podrán realizar operaciones con datos de 4 bytes en un solo ciclo de reloj, en vez de 1 byte en un solo ciclo. Esto quiere decir que los cálculos se realizarán a una mayor velocidad, que es justo lo que necesita este tipo de sistema. De esta forma se solventaría el hándicab de la restricción de tiempo del segway.

Sin embargo, es posible que con esta solución no se consiga que el control se realice cada 40 ms. Por lo tanto, también se debería seguir mejorando el software.

Tabla 5.4 Comparación.

Característica	**Mega**	**Due**
Microcontrolador	ATmega2560	SAM3X8E ARM Cortex-M3
Bits	$8bits$	$32bits$
Memoria flash	$256KB$	$512KB$
SRAM	$8KB$	$96KB$
EEPROM	$4KB$	-
Frecuencia de reloj	$16MHz$	$84MHz$
Temporizadores	6 Timers	9 Timers
UARTs	4 pines	4 pines
SPI	5 pines	5 pines
TWI (I2C)	2 pines	2 pines
Tensión nominal	$5V$	$3.3V$
Tensión de entrada (uso normal)	$7-12V$	$5-12V$
Tensión de entrada (límite)	$6-20V$	$6-16V$
Pines de E/S digital	54 pines (15 para PWM)	54 pines (12 para PWM)
Pines de E analógica	16 pines	12 pines
Pines de S analógica	-	2 pines
Corriente continua por pin E/S digital	$40mA$	$800mA$
Corriente continua por pin 3V3	$50mA$	$800mA$

6 Líneas futuras y conclusión

Como conclusión de este trabajo, existe una línea de mayor desarrollo del Software. Ya que el gran problema de Arduino es realizar los cálculos en un sistema real con una restricción de tiempo, hay más formas de conseguir decrementar el tiempo de cómputo. Una de ellas es el uso de números enteros en lugar de reales.

Se ha explicado en el capítulo anterior que la cantidad de decimales exactos que se usen es importante, por lo tanto no sería conveniente eliminarlos. Sin embargo, se podría normalizar el problema para después escalarlo de forma que se usen al menos unos 6 ó 7 decimales. Es decir, que los números con los que se trabajarán serán enteros donde al ser escalados sus cifras decimales pasan a la parte del entero. Por lo tanto, las matrices del algoritmo FISTA serían transformadas para que los límites de u y x se encuentren en el intervalo $[0,10^7]$. Estaría de esta forma el problema normalizado y escalado.

Una vez calculada la acción de control normalizada y multiplicada por 10^7, se realizaría la transformación contraria. De esta forma se daría al sistema el valor real de la actuación.

Además de poder hacer este desarrollo del Software, también sería conveniente utilizar una placa de Arduino más acorde a las exigencias del algoritmo de control que se quiere implementar. Esto requeriría reprogramar todo el código del sistema, ya que cambiará según sus especificaciones físicas.

Otro solución posible, sería añadir una placa de Arduino que contenga solo el bucle de control, sin la lectura de sensores y la realización de la actuación. Para esto habría que enviar la actuación calculada por la placa que haga el MPC a la placa que ya está instalada.

Apéndice A
Códigos de Matlab

Código A.1 Inicialización de variables globales.

```
global A B Ep C Cd D G n m p LBx UBx LBu UBu Q R b xk E H lim
% Parametros del modelo del segway.
N=20;
% Condiciones iniciales:
phi0=0*pi/180; % angulo inical de inclinación del pendulo
dphi0=0; % velodidad inicial de giro del pendulo
vr0=0; % velocidad de giro de las ruedas.
%declaracion variables
np=150;
Tm=0.04;
A=[1.032072427807300 0.040426724937844 0;1.612138909106717 1.032072427807300
    0;0 0 1.000000000000000];
B=[-0.000289588215330; -0.014556317106940;0.040000000000000];
Ep=[0.032072427807300; 1.612138909106717;0.040000000000000];
C=[1 0 0;0 1 0;0 0 1];
G=[0 0 1];
Cd=zeros(n,p);
D=[0;0;0];
LBx=-[30*pi/180 ; 5 ; 10]; % limite superior e inferior de X
UBx=[30*pi/180 ; 5 ; 10];
LBu=-150; % limite superior e inferior de U
UBu=150;
[fil,n]=size(A);
[filb,m]=size(B);
p=n;
xf=zeros(n,np);
%estado
y=zeros(p,np); %salida
u=zeros(m,np); %acc. cntrol
xf(1,1)=phi0;
xf(2,1)=dphi0;
xf(3,1)=vr0;
Q=[10 0 0;0 1 0;0 0 10];
R=0.1;
xk=xf(:,1);
lim=zeros((n+m)*N,1);
```

```
for i=1:n+m:(n+m)*N
lim(i:i+n-1)=UBx;
end
for i=n+1:n+m:(n+m)*N
lim(i:i+m-1)=UBu;
end
```

Código A.2 Definición de las variables del MPC.

```
%Variables para el MPC
H=zeros(N*(n+m));
E=zeros((N+1)*n,(n+m)*N);
I=eye(n);
aux=0;
for cont=1:N
for i=1:n
H(i+n*(cont-1)+aux,i+n*(cont-1)+aux)=Q(i,i);
end
for i=n+1:m+n
H(i+n*(cont-1)+aux,i+n*(cont-1)+aux)=R;
aux=aux+1;
end
end
aux=0;
for i=1:N
E(n+1+n*(i-1):2*n+n*(i-1),i+n*(i-1):n+n*(i-1)+aux)=Ad;
E(n+1+n*(i-1):2*n+n*(i-1),i+n*(i-1)+n:m+n+n*(i-1)+aux)=Bd;
if i<N
E(n+1+n*(i-1):2*n+n*(i-1),i+n*(i-1)+n+m:n+m+n+n*(i-1)+aux)=-I;
end
if i==1
E(i:n,i:n)=I;
end
aux=aux+1;
end
```

Código A.3 Definición de las variables del SSTO.

```
Qr=[10 0 0; 0 1 0; 0 0 1];
Rr=0.1;
T=[1e8 0; 0 1e8];
H1=zeros(n+m+2);
E1=zeros(n+2,n+m+2);
I=eye(n);
I1=eye(2);
for i=1:n
H1(i,i)=Qr(i,i);
end
H1(n+1:m+n,n+1:m+n)=Rr;
H1(n+m+1:n+m+2,n+m+1:n+m+2)=T;
E1(1:n,1:n)=A-I;
E1(1:n,n+1:n+m)=B;
E1(n+1:n+2,1:n)=[1 0 0;0 0 1];
E1(n+1:n+2,n+m+1:n+m+2)=I1;
Ulim1=zeros(n+m+2,1);
```

```
Llim1=zeros(n+m+2,1);
Ulim1(1:n,1)=[20*pi/180 ; 2 ; 10];
Ulim1(n+m,1)=100;
Ulim1(n+m+1:2+n+m,1)=10^8;
Llim1(1:n,1)=-[20*pi/180 ; 2 ; 10];
Llim1(n+m,1)=-100;
Llim1(n+m+1:2+n+m,1)=-10^8;
```

Código A.4 Bucle de Control con Algoritmo FISTA.

```
u=0;
b=zeros((N+1)*n,1);
dr_gorro=0;
dr=0;
alpha=0.2;
r_rampa=0;
np=500;
Cm=[1 0 0];
r=0;
Pest = zeros(3,3); % matriz de covarianza estimada
Ppr = zeros(3,3); % matriz de covarianza predicha
Ko = zeros(3,3); % matriz de ganancias de Kalman
tsim=14;
ruido=[0.0015*rand-0.0015*rand 0.0015*rand-0.0015*rand 0.0015*rand-0.0015*rand
    ]';
for k=1:np
h=0.005;
w=dr;
[t,y] = Rungekutta(@(t,y) Mdlsegwconpert(y,u,w),[(k-1)*Tm k*Tm],xk,h);
xk=y(end,:)';
yk=C*xk+ruido;
% OBSERVADOR: Filtro de Kalman
Ro=I;
Qo=diag(1);
xpr=A*x_filtrada+B*u+Ep*d_filtrada;
Ppr=Qo+A*Pest*A';
Ko=Ppr*C'/(C*Ppr*C'+ Ro);
x_filtrada=xpr+Ko*(yk-C*xpr);
Pest=(I-Ko*C)*Ppred;
d_filtrada=d_filtrada+[-1 0 0]*(C*xk-C*x_filtrada);
r_rampa=(1-alpha)*r_rampa+alpha*r';
[xr,ur]=SSTO(H1,E1,[0;r_rampa],Ep*d_filtrada,Ulim1,Llim1,n,m,2);
%FISTA
x=zeros((N+1)*n,1);x_ant=zeros((N+1)*n,1);y=zeros((N+1)*n,1);
t=1;FIN=0;cont1=0;
Qfista=inv(E/H*E');
%matrices f1 y b
f1=zeros(N*(n+m),1);
for i=1:n+m:N*(n+m)
f1(i:i+n-1,1)=-Q*xr;
f1(i+n:i+n,1)=-R*ur;
end
dd=Ep*d_filtrada;
b=zeros((N+1)*n,1);
b(1:n,1)=x_filtrada;
for i=1:3:N*n
```

```
b(n+i:n+i+2)=-dd;
end
b((N+1)*n-(n-1):(N+1)*n,1)=xr-dd;

while FIN==0
z=zeros(n,1);
f=f1-E'*y;
for i=1:N*(n+m)
z1=-f(i,1)/H(i,i);
z(i)=max(Llim(i,1),min(z1,Ulim(i,1)));
end
if norm(E*z-b)<=1e-6
FIN=1;
else
incry=-Qfista*(E*z-b);
x=y+incry;
t_sig=0.5*(1+sqrt(1+4*t^2));
y=x+(((t-1)/t_sig)*(x-x_ant));
cont1=cont1+1;
x_ant=x;
t=t_sig;
end
if cont1>2000
FIN=1;
disp('Se supera el número de iteraciones')
end
u=z(4);
end
end
```

Código A.5 Función del SSTO.

```
function [xr,ur]=SSTO1(H1,E1,yr,dd,Ulim,Llim,n,m,p)
%Algoritmo FISTA
%inicialización
x=zeros(n+p,1);x_ant=zeros(n+p,1);y=zeros(n+p,1);
t=1;FIN=0;cont1=0;
Qfista=inv(E1/H1*E1');
f1=zeros(n+m+p,1);
b=zeros(n+p,1);
b(1:n,1)=dd;
b(n+1:n+p,1)=yr;
while FIN==0
z=zeros(n,1);
f=f1-E1'*y;
for i=1:n+m+p
z1=-f(i,1)/H1(i,i);
z(i)=max(Llim(i,1),min(z1,Ulim(i,1)));
end
%norm(E1*z-b)
if norm(E1*z-b)<=1e-6
FIN=1;
else
incry=-Qfista*(E1*z-b);
x=y+incry;
t_sig=0.5*(1+sqrt(1+4*t^2));
```

```
y=x+(((t-1)/t_sig)*(x-x_ant));
cont1=cont1+1;
x_ant=x;
t=t_sig;
end
if cont1>2000
FIN=1;
disp('Se supera el número de iteraciones')
%norm(E1*z-b)
end
end
xr=[z(1);z(2);z(3)];
ur=z(4);
end
```

Código A.6 Modelo no lineal.

```
function [T,Y]=Rungekutta(f,intervalo,y0,h)
t0=intervalo(1);
tf=intervalo(2);
y0=y0(:);
T=(t0:h:tf)';
N=length(T)-1;
Y=ones(N+1,1)*y0';
for k=1:N
k1=f(T(k),y0);
k2=f(T(k)+h/2,y0+h*k1/2);
k3=f(T(k)+h/2,y0+h*k2/2);
k4=f(T(k)+h,y0+h*k3);
y1=y0+(h/6)*(k1+2*k2+2*k3+k4);
Y(k+1,:)=y1';
y0=y1;
end
end

function [ dx ] = Mdlsegwconpert(x,u,w)
g=9.81;
mr=0.140; % Masa de cada rueda
R=0.05; % Radiod e la rueda
M=0.82; % Masa del cuerpo del pendulo
L=0.098; % Distancia del centro de gravedad al eje de las ruedas
Jr=0.5*mr*R^2;
J=M*L^2;
a=(mr+M/2)*R^2+Jr;
b=M*L^2/2+J/2;
c=M*R*L;
d=-M*g*L;
dx=zeros(3,1);
dx(1)=x(2);
dx(2)=(((c*x(2)^2*sin(x(1)+w))-d*sin(x(1)+w)-u*(2*a+c*cos(x(1)+w)))/(2*b+c*cos(
    x(1)+w)));
dx(3)=u;;
end
```

Apéndice B

Código de Arduino

Código B.1 Definición de variables.

```
#define N 20
#define n 3
#define m 1
struct DatosMPC{
float A[9];
float B[3];
float Q[9];
float R[1];
float xr[3];
float ur[1];
float x_filtrada[3];
float Et[24];
float lim[4];
float zmpc[N*(n+m)];
float E[21];
float Qfista[((N+1)*n)*((N+1)*n)];
float Ep[3];
float pert;
};
```

Código B.2 Carga de datos en Arduino.

```
void setup() {
Serial.begin(9600);
DatosMPC Planta;
llenaPlanta(&Planta);
}
```

Código B.3 Cálculo de la actuación.

```
void loop() {
DatosMPC Planta;
llenaPlanta(&Planta);
float u[m]={0};
```

```
Calculo_U(&Planta,u);
}
```

Código B.4 Algoritmo FISTA.

```
void Calculo_U(struct DatosMPC *Datos, float u[m]){
float Qxr[n];
float Rur[m];
float f1[n+m];
int i,i0,i1,j,ii,jj;
float b[(N+1)*n];
float aux[n+m];
int neg;
float x[(N+1)*n];
float x_ant[(N+1)*n];
float y[(N+1)*n];
float deltay[(N+1)*n];
int FIN,k;
int cont;
float t;
float t_sig;
float f[(n+m)*N];

float z;
float signo;
float u1;
int horizonte;

float Ez_b_d[(N+1)*n];
float norma;
float aux1;
float aux2[n];

int valores=0;

//Inicialización
t=1;
FIN=0;
cont=1;
for (i=0; i<n*(N+1);i++){
x[i]=0.0;
x_ant[i]=0;
y[i]=0;
}

//formación del vector f1
for (j = 0; j < n; j++){
Qxr[j]=0;
for (i = 0; i < n; i++) {
Qxr[j]+= Datos->Q[j*n+i] * Datos->xr[i];
}
}
for (i = 0; i < m; i++) {
Rur[i] = Datos->R[i] * Datos->ur[i];
}
```

```
//f1={-Qxr0[0],-Qxr1[0],-Qxr2[0],-Rur0[0],-Qxr0[1],-Qxr1[1],-Qxr2[1],-Rur0
    [1],...,-Qxr0[N-1],-Qxr1[N-1],-Qxr2[N-1],-Rur0[N-1]}, esto debería ser pero
     se simplifica
//f1={-Qxr0,-Qxr1,-Qxr2,-Rur0};
for (i0 = 0; i0 < n; i0++) {
f1[i0] = -Qxr[i0];
}
f1[3] = -Rur[0];

//formación del vector b que formaba parte de la ecuación de igualdad
//b={x_filtrada0[0],x_filtrada1[0],x_filtrada2[0],0,0,...,0,xr0[N],xr1[N],xr2[N
    ]}
for (i=0; i<n*(N+1);i++){
b[i]=0;
}
for (i = 0; i < n; i++) {
b[i + N*n] = Datos->xr[i];
}
for (i = 0; i < n; i++) {
b[i] = Datos->x_filtrada[i]; //el valor de x inicialmente filtrado (Kalman)
}

//formación del vector d que formaba parte de la ecuación de igualdad y que
    contiene la perturbación
//d={0,0,0,EpD1[1],EpD2[1],EpD3[1],...,EpD1[N],EpD2[N],EpD3[N]}

for (i=0; i<N+1;i++){
for (j=0;j<n;j++){
d[i*n+j]=Datos->Ep[j]*Datos->pert;
}
}
for (i = 0; i < n; i++) {
d[i] = 0;
}

//algoritmo FISTA
while (!(FIN != 0))
{
//Cálculo de la función objetivo
// Cálculo de f
//Se necesitará realizar anteriormente el cálculo de A traspuesta y B
    traspuesta para poder calcular la matriz E traspuesta
//E traspuesta será una de las entradas a esta función
// donde f=f1-E'*y
neg=0;

for(ii=0,jj=0;ii<N*(m+n);ii+=n+m,jj+=n) {
for(i=0;i<n+m;i++) {
aux[i]=0;
for(j=0;j<2*n;j++) {
if(j<n && neg==1){
aux[i]+=-Datos->Et[i*2*n + j]*y[jj+j];
}
else{
aux[i]+=Datos->Et[i*2*n + j]*y[jj+j];
}
}
```

```
f[ii+i]=f1[i]-aux[i];
}
neg=1;
}

//Cálculo de la z óptima minimizando
for (horizonte=0;horizonte<N;horizonte++){
for (i = 0; i < (n+m); i++) {
if(i<n){
z = -f[i+(n+m)*horizonte] / Datos->Q[i*n+i];
}
else{
z = -f[i+(n+m)*horizonte] / Datos->R[i-n] ;
}
//cálculo del signo de z
signo = z;
if (z > 0.0) {
signo = 1.0;
}
else if (z < 0.0) {
signo = -1.0;
}
else {
if (z == 0.0) {
signo = 0.0;
}
}
//poner la z positiva
z = fabs(z);
//guardar el límite superior
u1 = Datos->lim[i];
//saber si supera el valor de z calculado al valor del límite superior
z = (z <= u1) ? z : u1;
Datos->zmpc[i+(n+m)*horizonte] = z * signo;
}
}

//Cálculo del valor absoluto para la tolerancia
for(i=0;i<n;i++) {
//Primera parte de E sería la matriz identidad, I*zmpc-b-d, siendo b x_filtrada
Ez_b_d[i]=Datos->zmpc[i]-b[i]-d[i];
//El final de la matriz E sería [A B], [A B]*zmpc-b-d, siendo b xr
aux2[i]=0;
for(j=0;j<n+m;j++) {
aux2[i]+=Datos->E[i*(2*n+m) + j]*Datos->zmpc[(n+m)*(N-1)+j];
}
Ez_b_d[n*N+i]=aux2[i]-b[i+N*n]-d[i+N*n];
}
for(ii=n,jj=0;ii<n*N;ii+=n,jj+=n+m) {
for(i=0;i<n;i++) {
//El resto de la matriz E [A B -I]zmpc-b-d, siendo b cero
aux2[i]=0;
for(j=0;j<2*n+m;j++) {
aux2[i]+=Datos->E[i*(2*n+m) + j]*Datos->zmpc[jj+j];
}
Ez_b_d[ii+i]=aux2[i]-d[ii+i];
}
```

```
}
for(i=0;i<(N+1)*n;i++){
aux1+=Ez_b_d[i]*Ez_b_d[i];
}
normaq=aux1;


//Taylor
if(normaq <= 1E-4)
{
FIN=1;
}
else
{
for(i=0;i<(N+1)*n;i++)
{
deltay[i]=0.0;
for(j=0;j<(N+1)*n;j++)
{
deltay[i]+=-Datos->Qfista[i*(N+1)*n + j]*Ez_b_d[j];
}
x[i]=y[i]+deltay[i];
}
t_sig=0.5*(1+sqrt(1+4*t*t));
for(i=0;i<(N+1)*n;i++)
{
y[i]=x[i]+((t-1)/t_sig)*(x[i]-x_ant[i]);
x_ant[i]=x[i];
}
cont++;
t=t_sig;
}
if (cont > 50)
{
FIN = 1;
// exceso de bucle
}
}
for (i = 0; i < m; i++)
{
u[i] = Datos->zmpc[i + n];
}
}
```

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] D. Muñoz La Peña C. González, I. Alvarado, *Low cost two-wheels self-balancing robot for control education*, The International Federation of Automatic Control, July 2017.

[2] Carlos Bordons Alba Daniel Rodríguez Ramírez, *Apuntes de ingeniería de control*, Departamento de Ingeniería de Sistemas y Automática, Universidad de Sevilla, 2005.

[3] Cecilia González González, *Mejora del software de un vehículo autoequilibrado de tipo péndulo invertido de bajo coste*, Universidad de Sevilla, 2016.

[4] Victor Manuel Villalar Lara, *Control predictivo para un vehículo tipo segway*, Universidad de Sevilla, 2017.

[5] Alfonso García Navarro, *Implementación de controladores predictivos en arduino*, Universidad de Sevilla, 2016.

[6] Antonio J. Croche Navarro, *Vehículo autoequilibrado de tipo péndulo invertido de bajo coste*, Universidad de Sevilla, 2013.

[7] Renato Álvarez Nodarse, *Apuntes de diferenciación de funciones de varias variables*, Departamento de Análisis Matemático Facultad de Matemáticas, Septiembre 2017.