UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

DISEÑO DE UN SISTEMA DE CONTROL BASADO EN PLC PARA UNA PLATAFORMA MÓVIL (SEGWAY) VIRTUALIZADA EN RASPBERRY PI MEDIANTE LENGUAJE PYTHON

AUTOR: CARLOS GIMENO RUIZ

TUTOR: JOSÉ VICENTE SALCEDO ROMERO DE ÁVILA

Curso Académico: 2020-21

Agradecimientos

A mis padres y mi hermana, por estar siempre ahí apoyándome y no permitir que me relaje;

A mis amigos de verdad, por ayudarme a olvidar los malos momentos en las situaciones de mayor estrés;

Y a mi tutor, por darme la oportunidad de realizar este trabajo y comprender mi situación en todo momento durante estos meses.

Resumen

El presente Trabajo Final de Máster se encarga del diseño de un sistema de control que sea capaz de estabilizar la velocidad lineal y la inclinación del modelo no lineal de un Segway mediante diferentes técnicas, las cuales van desde las más simples, tales como una estructura de PIDs, hasta las más complejas, basadas en observadores de estado, control integral, control óptimo o control robusto.

En primer lugar, se da lugar a una fase de simulación consistente en la prueba y desarrollo de los controladores y realizada en el programa MATLAB. Una vez han sido validados, se inicia la fase experimental en la cual se certifica el correcto funcionamiento del sistema, y para ello se hace uso de una Raspberry Pi, encargada de virtualizar el comportamiento del Segway mediante lenguaje Python, y de un PLC, encargado de recrear los controladores diseñados mediante lenguaje basado en Texto Estructurado. De esta forma, estos dos dispositivos se comunican entre sí, dando lugar a la constante lectura y escritura de acciones de control y variables controladas.

Finalmente, se comparan los resultados de simulación y experimentación, apreciándose unas veces una gran similitud y otras veces cierta diferencia entre sí, lo cual es debido a problemas de cuantificación de las variables leídas y escritas desde los conversores A/D y D/A utilizados en el PLC y la Raspberry Pi, y a tiempos de computación que en algunos momentos pueden ser superiores al periodo de muestreo.

Palabras clave: diseño, sistema de control, Segway, MATLAB, Raspberry Pi, Python, PLC

Resum

El present Treball Final de Màster s'encarrega del disseny d'un sistema de control que puga estabilitzar la velocitat lineal i la inclinació del model no lineal d'un Segway mitjançant diverses tècniques, entre les quals es troben des de les més simples, com ara una estructura de PIDs, fins a les de major dificultat, basades en observadors d'estat, control integral, control òptim o control robust.

En primer lloc, es realitza una fase de simulació consistent en la prova i desenvolupament dels controladors i realitzada en el programa MATLAB. Una vegada han sigut validats, s'inicia la fase experimental, en la qual se certifica el correcte funcionament del sistema, i per dur a terme això s'utilitza una Raspberry Pi, encarregada de virtualitzar el comportament del Segway mitjançant llenguatge Python, i d'un PLC, encarregat de recrear els controladors dissenyats amb llenguatge basat en Text Estructurat. D'aquesta manera, aquests dos dispositius es comuniquen entre si, donant lloc a la constant lectura i escriptura d'accions de control i variables controlades.

Finalment, es comparen els resultats de simulació i experimentació, apreciant-se unes vegades una gran similitud i altres vegades una certa diferència entre si, la qual cosa és deguda a problemes de quantificació de les variables llegides i escrites des de els convertidors A/D y D/A utilitzats en el PLC i la Raspberry Pi, i a temps de computació que a vegades poden ser superiors al període de mostreig.

Paraules clau: disseny, sistema de control, Segway, MATLAB, Raspberry Pi, Python, PLC

Abstract

This Master's Final Thesis designs a control system that is able to stabilize the linear speed and inclinitation of the non-linear model of a Segway using different techniques, which go from the simplest, such as a PID structure, to the most difficult, like state observers, integral effect, optimal control or robust control.

First, a simulation phase consisting of the testing and development of the controllers is carried out in the MATLAB program. Once they have been validated, the experimental phase begins and the correct performance of the system must be certified. For this purpose, a Raspberry virtualizes the behaviour of the Segway using Python language, and a PLC recreates the designed controllers using Structured Text language. In this way, these two devices communicate between them, resulting in the continual reading and writing of control actions and controlled variables.

Finally, the results of simulation and experimentation are compared, and it can be appreciated that sometimes are similar and another times aren't, which is due to quantification problems while reading and writing variables with A/D and D/A converters in the communication between PLC and Raspberry Pi, and also to computation times that sometimes are higher than the sample time.

Keywords: design, control system, Segway, MATLAB, Raspberry Pi, Python, PLC

Índice general

Ilustraciones

Tablas

Scripts

Parte I

Memoria

Capítulo 1

Introducción

1.1. Antecedentes

El presente Trabajo Final de Máster (en adelante, TFM) se basa en una serie de herramientas software y hardware fundamentales para su correcta implementación final. Por ello, se hace necesario analizar brevemente el funcionamiento de cada una de ellas y sus principales virtudes.

1.1.1. Origen y funcionalidades del Segway

En la actualidad, un elevado porcentaje de la población requiere desplazarse constantemente y con rapidez durante su día a día para poder satisfacer sus necesidades laborales y cotidianas en un estilo de vida cada vez más acelerado, pero también se está promoviendo una concienciación de la sociedad respecto a la reducción de contaminación en el medio ambiente. De esta forma, los transportes eléctricos están cobrando una gran importancia entre la sociedad.

En concreto, el presente TFM se centra en la utilización y control del vehículo eléctrico denominado Segway [Ilustración 1], basado en un modelo físico muy similar a los patinetes eléctricos actuales, pero de un tamaño superior. Está formado por dos ruedas, cada una de ellas con un motor eléctrico independiente, y un mando direccional con el que el usuario orienta su movimiento.

Ilustración 1. Prototipo de Segway (Tomada de [1])

El Segway fue creado por el inventor estadounidense Dean Kamen en 2001 y dispone de un sistema de auto-equilibrio fácilmente manejable denominado estabilización dinámica. También está formado por giroscopios, sensores de inclinación, microprocesadores de alta velocidad y unos potentes motores eléctricos que, trabajando conjuntamente, perciben el centro de gravedad del usuario [2].

Por ello, el objetivo de este TFM es el de, utilizando un modelo del Segway basado en ecuaciones no lineales, estabilizarlo a partir de diferentes técnicas de control.

1.1.2. Origen y utilización de la Raspberry Pi

Como era presumible, el coste promedio del Segway no es asumible para la realización de pruebas. Asimismo, carece de sentido realizarlas sobre un dispositivo que incluye los sistemas de estabilización necesarios.

Por ello, para la fase experimental se ha optado por hacer uso de una Raspberry Pi [Ilustración 2] (en adelante, RPI), un dispositivo fabricado por primera vez en 2012 en Reino Unido y consistente en una placa de microordenador que utiliza el sistema operativo Raspbian (Linux) y que puede utilizarse como ordenador portable gracias a su variedad de funcionalidades [4].

Ilustración 2. Prototipo de Raspberry Pi (Tomada de [3])

En este TFM, la RPI se encarga de integrar las ecuaciones del modelo no lineal del Segway, programadas en lenguaje Python, simulando el comportamiento del mismo durante la fase experimental. Para encenderla, debe ser alimentada previamente con conexión eléctrica y Ethernet, y posteriormente se hace uso del programa VNC Viewer, el cual permite visualizar su contenido en otro ordenador ajeno simplemente introduciendo la dirección IP asignada y unas credenciales concretas. Así, su objetivo es comunicarse con un PLC, en el que se incluye el controlador encargado de estabilizar al modelo.

1.1.3. Origen y utilización del programa MATLAB

Previamente a la fase experimental, es necesaria una fase de simulación donde se compruebe la validez de los controladores, y para ello se hace uso de MATLAB [Ilustración 3], versión R2021a, un entorno de programación y cálculo numérico desarrollado por la compañía MathWorks y basado en el lenguaje M. Con todo ello, este programa es capaz de manipular todo tipo de vectores y matrices, realizar análisis iterativos de datos, desarrollar algoritmos y convertirlos a códigos basados en otros lenguajes, tales como C/C++ y HDL [5].

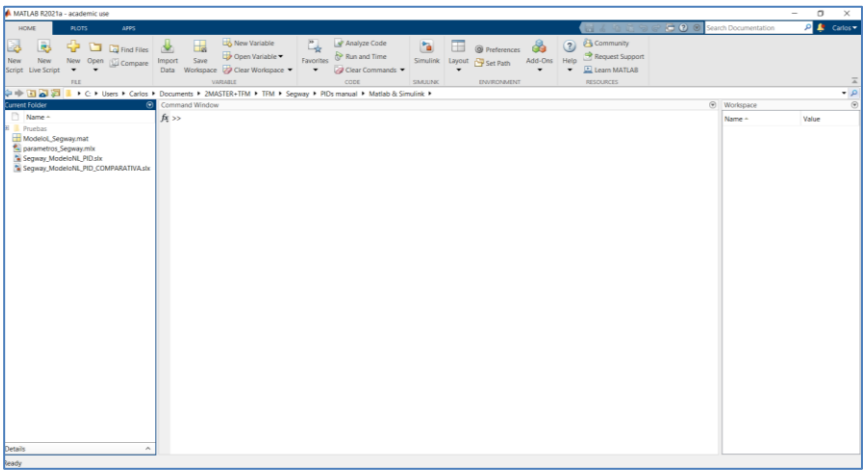

Ilustración 3. Ventana principal de MATLAB (Elaboración propia)

En MATLAB, además de *scripts*, también existe la posibilidad de programar funciones [Ilustración 4], las cuales requieren unos parámetros de entrada y devuelven unos parámetros de salida distintos. En ambos casos gozan de la extensión ".m" o ".mlx".

El programa incluye multitud de toolboxes, y en concreto se hace uso de Simulink, una app basada en el diseño mediante bloques y simulación de prototipos de sistemas y plantas físicas antes de ser llevados a la práctica [6]. A continuación, se muestra un ejemplo de fichero en Simulink [Ilustración 4] de extensión ".slx" donde se aprecia el uso de subsistemas que almacenan modelos de procesos, llamadas a funciones y gráficas, entre muchos otros. Por otro lado, gran parte de los bloques pueden ser convertidos a fragmentos de código expresados en lenguaje de Texto Estructurado (en adelante, ST) mediante la app PLC Coder, lo cual agiliza los tiempos en el paso a la fase experimental.

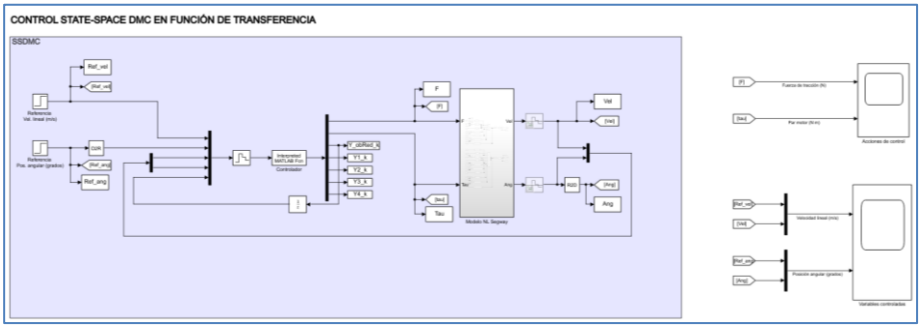

Ilustración 4. Ejemplo de fichero en Simulink (Elaboración propia)

1.1.4. Origen y utilización del programa Spyder

Antes de llevar a la fase experimental los controles desarrollados en simulación, se considera oportuno realizar unas pruebas intermedias basadas en lenguaje ST que corroboren que el código creado por PLC Coder es correcto, y para ello se hace uso del programa Spyder [Ilustración 5], un entorno de código abierto capaz de editar, analizar, depurar y crear datos a través del lenguaje Python [7]. Al igual que MATLAB, también se pueden crear *scripts* y funciones, en este caso de extensión ".py", importar librerías y almacenar todo tipo de variables.

Evidentemente, Python y ST no son idénticos entre sí, pero sí que es posible adaptar el código de un lenguaje al otro con relativa facilidad, por lo que este paso es de gran utilidad para saber con cierta seguridad si un control está bien desarrollado o no.

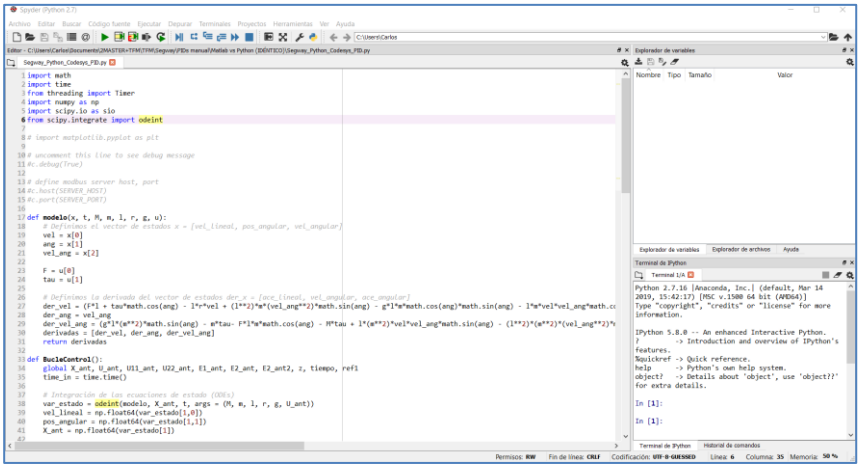

Ilustración 5. Ventana principal de Spyder (Elaboración propia)

1.1.5. Descripción del programa SoMachine

Tras comprobar el correcto funcionamiento de los controles en lenguaje Python/ST, se hace preciso pasar a la fase experimental, es decir, en la comunicación entre RPI y PLC, es decir, autómata programable que alberga el controlador del proceso. En este caso, se trabaja con el modelo Modicon M241 TM241CE40R [Ilustración 6] de la compañía Schneider Electric, que se encuentra instalado en un módulo junto a un interruptor que actúa como fuente de alimentación y que requiere de conexión Ethernet. Asimismo, dispone de una toma de tierra, dos entradas analógicas y dos salidas analógicas, las cuales se utilizan para la comunicación con la RPI.

Ilustración 6. PLC Modicon M241 TM241CE40R (Tomada de [8])

Este PLC requiere cargar sus programas mediante EcoStruxure Machine Expert (en adelante, SoMachine), un entorno único de software colaborativo para la programación intuitiva y puesta en marcha de máquinas, incluyendo lógica, control de movimiento, robótica/mecánica, simulación, diagnósticos, gestión inteligente de motores y cargas y funciones de automatización de red relacionadas [9]. En este TFM, los programas están basados en lenguaje ST, pero SoMachine también es capaz de trabajar con Diagrama de Contactos o GRAFCET.

Para programar en dicho lenguaje, se requiere el uso de, al menos, una Unidad de Organización del Programa (en adelante, POU) [Ilustración 7], que se compone de un directorio en el que declarar las variables y de un espacio donde diseñar el código. Mediante el uso de POUs, también es posible modificar el valor de las variables cuando el PLC se encuentra en línea.

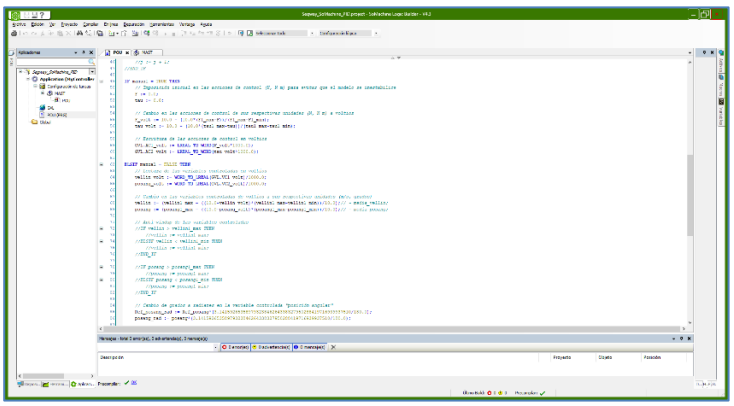

Ilustración 7. Ventana principal de SoMachine (Elaboración propia)

Del mismo modo, también existe la posibilidad de imponer el período de muestreo con el que se va a ejecutar el controlador, y se trata de una funcionalidad muy importante en el presente TFM debido al carácter cíclico del mismo.

1.2. Objetivos

Los objetivos principales del presente TFM son:

- **Diseñar un sistema de control basado en diferentes técnicas y capaz de estabilizar la velocidad lineal y la posición angular del Segway.**

A partir del conocimiento adquirido sobre técnicas de control avanzadas durante la especialidad de control de procesos, automatización y robótica, se desea desarrollar un potente sistema de control que englobe diferentes técnicas y sea capaz de estabilizar la velocidad lineal de las ruedas del Segway y la posición angular del mando direccional.

- **Utilizar un PLC durante la fase experimental del sistema de control del Segway**

El sistema de control se prueba inicialmente mediante simulaciones que verifiquen que es posible la estabilización del Segway, pero es en la fase experimental donde debe demostrarse que este sistema funciona correctamente, y para ello es vital el uso de un PLC que almacene y ejecute los controladores programados en lenguaje ST.

- **Utilizar una Raspberry Pi para simular el comportamiento del Segway**

Pese a su reducido tamaño y precio, la Raspberry Pi es un dispositivo que ofrece muchas funcionalidades. Por ello, se desea demostrar que la Raspberry Pi, integrando las ecuaciones oportunas del modelo, es capaz de simular el comportamiento del Segway con cierto realismo.

1.3. Motivación y justificación

Los orígenes de la ingeniería de control se remontan a la Antigua Grecia cuando, en el siglo III a.C., el inventor e ingeniero hidráulico Ktesibios diseña por primera vez una Clepsydra [Ilustración 8 (izq.)], un reloj de agua cuyo mecanismo consistía en un depósito de agua con un nivel que aumenta a velocidad constante gracias a un flotador que regula la entrada de agua a otro depósito auxiliar, de forma que su nivel y su caudal de salida al depósito principal fuesen constantes. Se utilizaban en juicios con el fin de asegurar que ambas partes dispusiesen del mismo tiempo de alegaciones finales [10].

Poco más tarde, Platón ideó una variación de estos relojes para dotarles de función automática y así conseguir que sus alumnos llegasen a tiempo a las clases. Ésta consistía en un flotador situado encima del vaso del reloj con unas bolas depositadas sobre él, las cuales se acumulaban durante la noche hasta alcanzar su máximo nivel al amanecer, momento en el que rebosaban y caían sobre un plato de cobre, haciendo que el ruido despertase a sus alumnos [Ilustración 8 (derecha)] [10].

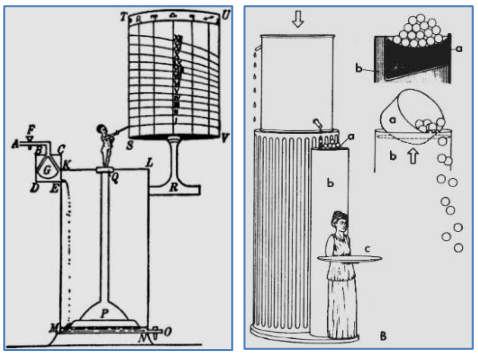

Ilustración 8. Reloj de agua de Ktesibios (izquierda) y de Platón (derecha) (Tomadas de [10])

Así, puede apreciarse que el control de procesos siempre ha sido requerido por la humanidad para su avance. De hecho, los actuales patinetes eléctricos no son más que una variación del Segway, y han

resultado exitosos en términos de velocidad y estabilización gracias a la ingeniería de control.

Por todo ello, la motivación del presente TFM es la de mostrar algunos de los diferentes tipos de control principales en que pueden haberse basado los ingenieros encargados de diseñar los Segways y patinetes eléctricos para su correcta estabilización.

1.4. Estructura del trabajo

El presente TFM está constituido principalmente por tres documentos: la memoria, en la que se definen detalladamente todos los controles desarrollados; la bibliografía, en la que se muestran todos los enlaces a páginas web, documentos y libros empleados para profundizar en ciertos conceptos, así como su fecha de consulta; y el presupuesto, en el que se exponen los costes económicos y recursos necesarios para la realización de dicho trabajo.

La memoria presenta, sin contar éste, 4 capítulos:

- **Segway**, donde se muestra la programación del modelo no lineal en MATLAB y en la RPI y se realiza la linealización del mismo para el posterior desarrollo de determinados controles.
- **Control del Segway mediante controladores PID**, donde se desarrolla el control del Segway basado en controladores PID, se muestra la programación utilizada y se compara la fase de simulación con la experimental.
- **Control del Segway mediante observadores de estado y efecto integral**, donde se muestra una variante de control basada en observadores de estado, realimentaciones del mismo y efecto integral, y también se comparan las diferentes fases.
- **Control del Segway mediante State-Space Model Predictive Control**, donde se desarrolla el control centrado en un tipo de control predictivo y se compara la fase de simulación con la experimental.
- **Control del Segway mediante control robusto**, donde se muestra el último método de estabilización del Segway, el control robusto, basado en técnicas creadas para ser implementadas sobre sistemas complejos.
- **Conclusiones**, donde se valoran los resultados obtenidos y las dificultades sufridas en el proceso.

Capítulo 2

Segway

En el presente capítulo, se va a presentar el conjunto de ecuaciones que conforman el modelo no lineal del Segway, así como la linealización del mismo, necesaria para la implementación de ciertos controles.

2.1. Modelo no lineal

Un Segway es una compleja estructura formada por sensores -concretamente giroscopios-, un sistema de control y un sistema motor que se complementan con relativa facilidad.

Sin embargo, el modelo de este dispositivo puede simplificarse como una plataforma móvil de cierta masa "M" sobre la que se dispone un péndulo invertido de cierta longitud "l" con una masa puntual "m" situada en el punto más alto del mismo [Ilustración 9]. Sobre la plataforma móvil se aplica una fuerza de tracción "F" con el objetivo de deslizarla y que adquiera cierta velocidad "$\dot{x}$", y sobre el eje de rotación del péndulo se aplica un par motor "τ" que debe mantenerlo con cierta inclinación "θ" sin inestabilizarse.

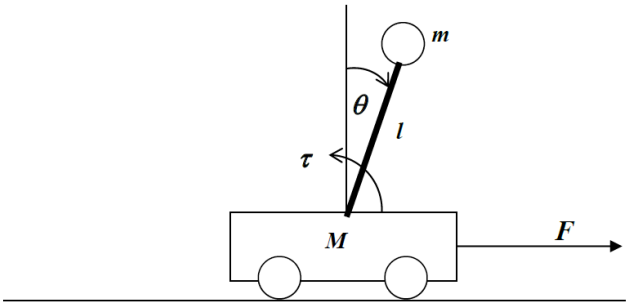

Ilustración 9. Modelo simplificado de un Segway (Tomada de [12])

Así, considerando también una fuerza de rozamiento "r" proporcional a la velocidad y la aceleración de la gravedad "g", puede definirse el modelo del sistema mediante dos ecuaciones obtenidas a partir de balance de fuerzas y momentos [12]:

$$F = (M + m) \cdot \ddot{x} + m \cdot l \cdot \cos(\theta) \cdot \ddot{\theta} - m \cdot l \cdot \sin(\theta) \cdot \left(\dot{\theta}\right)^2 + r \cdot \dot{x} \tag{2.1}$$

$$m \cdot g \cdot l \cdot \sin(\theta) - \tau = m \cdot l^2 \cdot \ddot{\theta} + m \cdot l \cdot \ddot{x} \cdot \cos(\theta) - m \cdot l \cdot \dot{x} \cdot \dot{\theta} \cdot \sin(\theta) \tag{2.2}$$

Este modelo de ecuaciones ODE también se sustenta en otros documentos e investigaciones, como [13] y [14], donde se encuentran modelos similares al mostrado, pero considerando una rotación diferente del péndulo y ciertas hipótesis que suprimen términos del modelo.

Los parámetros, entradas, salidas y estados mostrados en las ecuaciones (2.1) y (2.2) utilizan los siguientes valores y unidades [Tabla 1]:

Tabla 1. Valores y unidades de los parámetros del Segway (Elaboración propia)

VALORES Y UNIDADES DE LOS PARÁMETROS DEL MODELO			
Parámetro	**Definición**	**Valor**	**Unidad**
PARÁMETROS CONSTANTES			
$\boldsymbol{M}$	Masa de la plataforma móvil.	2.5	kg
$\boldsymbol{m}$	Masa del péndulo invertido.	0.6	kg
$\boldsymbol{l}$	Longitud del péndulo invertido.	1.2	m
$\boldsymbol{r}$	Fuerza de rozamiento proporcional a la velocidad de la plataforma móvil.	0.2	$\frac{N}{m/s}$
$\boldsymbol{g}$	Aceleración de la gravedad.	9.81	$\frac{m}{s^2}$
ACCIONES DE CONTROL			
$\boldsymbol{F}$	U_1: Fuerza de tracción aplicada sobre la plataforma móvil.	-	N
$\boldsymbol{\tau}$	U_2: Par motor aplicado sobre el eje de rotación del péndulo invertido.	-	$N\ m$
VARIABLES CONTROLADAS/VARIABLES DE ESTADO			
$\dot{x}$	Y_1-X_1: Velocidad lineal de la plataforma móvil.	-	$\frac{m}{s}$
$\ddot{x}$	Aceleración lineal de la plataforma móvil.	-	$\frac{m}{s^2}$
$\boldsymbol{\theta}$	Y_2-X_2: Posición angular del péndulo invertido.	-	rad
$\dot{\boldsymbol{\theta}}$	X_3: Velocidad angular del péndulo invertido.	-	$\frac{rad}{s}$
$\ddot{\boldsymbol{\theta}}$	Aceleración angular del péndulo invertido.	-	$\frac{rad}{s^2}$

Es decir, los vectores de acciones de control, variables controladas y variables de estado son:

$$U = \begin{bmatrix} F \\ \tau \end{bmatrix}, \qquad Y = \begin{bmatrix} \dot{x} \\ \theta \end{bmatrix}, \qquad X = \begin{bmatrix} \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} \tag{2.3}$$

No obstante, para implementar el modelo en MATLAB es necesario obtener una expresión de las incógnitas $\ddot{x}$ y $\ddot{\theta}$, es decir, la derivada primera de la velocidad lineal y la derivada segunda de la posición angular, de manera que al integrarlas sea posible obtener los valores de todas las salidas y estados.

Este procedimiento se resuelve rápidamente mediante el comando 'solve' simplemente introduciendo el modelo expresado en variables simbólicas, y almacenado en una matriz 2x1, y las variables a despejar. Así, se obtienen las expresiones:

(2.4) $$\frac{d^2x}{dt^2} = \ddot{x} = \frac{Fl + \tau\cos(\theta) - lr\dot{x} + l^2m\dot{\theta}^2\sin(\theta) - glm\cos(\theta)\sin(\theta) - lm\dot{x}\dot{\theta}\cos(\theta)\sin(\theta)}{l(-m\cos(\theta)^2 + M + m)}$$

(2.5) $$\frac{d^2\theta}{dt^2} = \ddot{\theta} = \frac{glm^2\sin(\theta) - m\tau - Flm\cos(\theta) - M\tau + lm^2\dot{x}\dot{\theta}\sin(\theta) - l^2m^2\dot{\theta}^2\cos(\theta)\sin(\theta) + \cdots}{l^2m(-m\cos(\theta)^2 + M + m)}$$

$$\frac{\cdots + Mglm\sin(\theta) + lmr\dot{x}\cos(\theta) + Mlm\dot{x}\theta\dot{\sin}(\theta)}{l^2m(-m\cos(\theta)^2 + M + m)}$$

En el caso de (2.4), sólo debe aplicarse una integración para obtener la velocidad lineal, mientras que en (2.5) la integración es doble, pues es necesario conocer tanto la velocidad como la posición angular. Por ello, se incluye una tercera expresión con la que el modelo queda completamente definido:

(2.6) $$\frac{d\theta}{dt} = \dot{\theta}$$

2.1.1. Implementación en MATLAB/Simulink

Se opta por implementar estas expresiones en MATLAB, y más concretamente en Simulink, donde se trabaja fácilmente con bloques, mucho más visuales. De esta forma, se ha creado el fichero "Segway_ModeloNL.slx" [Ilustración 10], donde se presenta un subsistema que desarrolla en su interior las expresiones (2.4) y (2.5), recibe las acciones de control y devuelve las variables controladas:

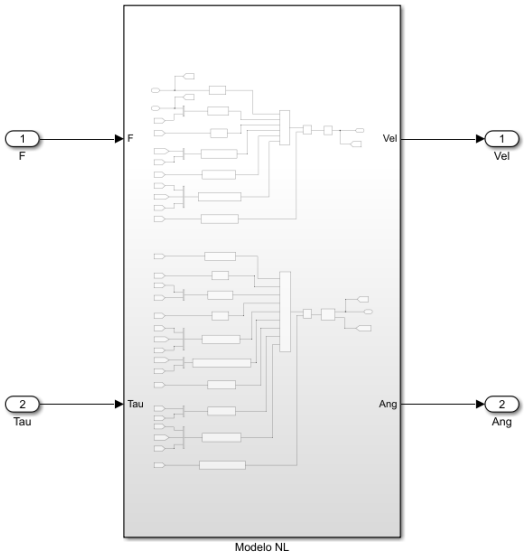

Ilustración 10. Fichero "Segway_ModeloNL.slx" (Elaboración propia)

Este subsistema se divide en un primer conjunto de bloques donde se desarrolla la expresión (2.4) [Ilustración 11] y un segundo conjunto donde se hace lo mismo con (2.5) [Ilustración 12], y debido a lo mencionado con anterioridad se cierran con un integrador simple y un integrador doble, respectivamente. Así, el modelo queda completamente definido para su uso durante la fase de simulación.

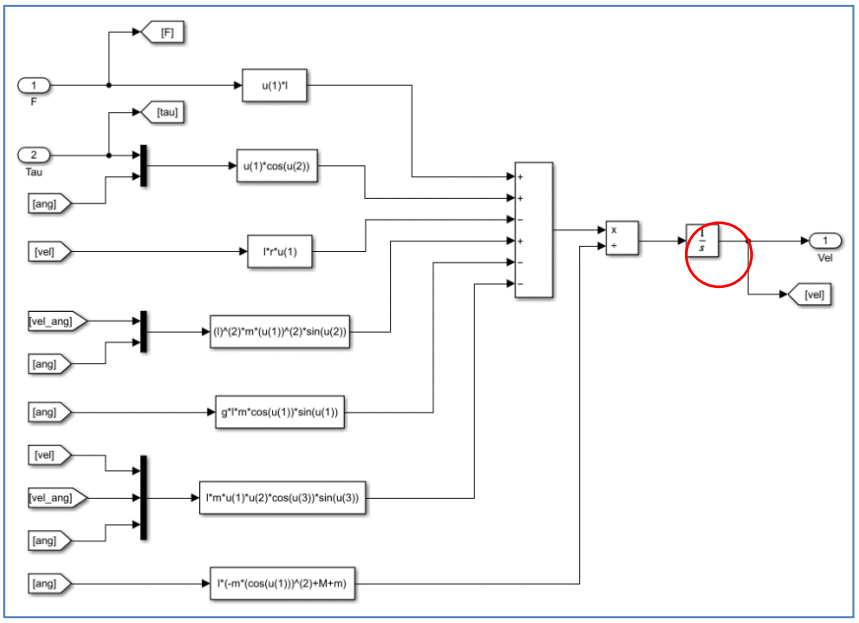

Ilustración 11. Fichero "Segway_ModeloNL.slx": ecuación de la aceleración lineal (Elaboración propia)

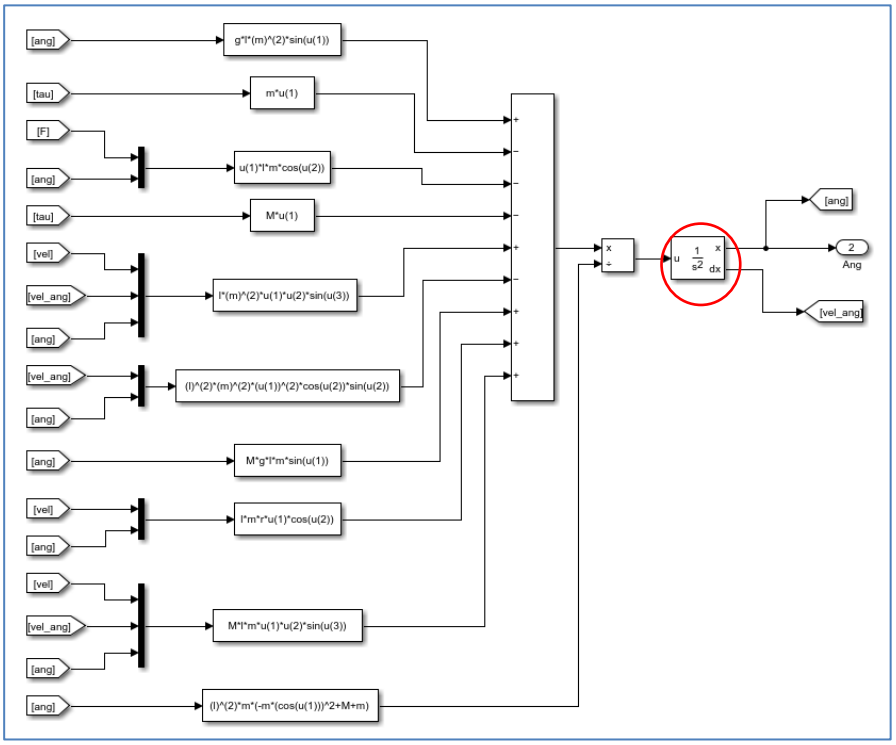

Ilustración 12. Fichero "Segway_ModeloNL.slx": ecuación de la aceleración angular (Elaboración propia)

2.1.2. Integración en la Raspberry Pi

Por otra parte, el modelo también debe ser desarrollado en la fase experimental, y para ello se programa en la RPI mediante el uso del script "segway_odeint.py" [Script 1], basado en lenguaje Python e integrado en la misma. Aquellas líneas con uno o dos símbolos "#" son comentarios que refuerzan el contenido del script o fragmentos de código que sólo se utilizan en ciertas condiciones, y el símbolo "…" se utiliza para abreviar en el caso de código repetitivo. La justificación de cada apartado es la siguiente:

2.1.2.1. Importación de librerías y declaración de variables

Se importan las librerías necesarias para integrar las ecuaciones, estipular el formato de almacenamiento de una variable y utilizar funciones matemáticas, respectivamente. También se declaran los parámetros fijos del modelo, el período de muestreo "Ts" de cada ciclo y el vector "X_ant" que inicializa a 0 las variables de estado.

También se declaran las variables "media_F" y "media_tau", que consisten en la media de cierta cantidad de valores de las acciones de control leídas cuando se están enviando valores nulos desde el PLC. Esto se realiza porque existe offset en la comunicación entre PLC y RPI, es decir, un problema de cuantificación que hace que en un dispositivo se lean valores distintos a los que se están enviando desde el otro y que puede provocar inestabilidad en el proceso. No obstante, no es una solución óptima, ya que el cálculo de las medias, omitido por carecer de importancia, incluye componentes aleatorios que hacen que cada vez se obtengan valores distintos.

Este problema aparece por partida doble, ya que también ocurre en el envío de valores de variables controladas desde la RPI al PLC.

2.1.2.2. Programación de funciones

Una vez declaradas las variables locales, se sigue con la programación de las funciones:

Función "modeloSegway"

Recibe como parámetros de entrada el vector de estados, el de acciones de control, el de tiempo y los parámetros fijos del modelo, y simplemente declara las ecuaciones de la derivada del vector de estados, de manera que sean posteriormente integradas por otra función.

Función "escala_u"

El envío de acciones de control del PLC a la RPI se realiza en forma de tensiones que oscilan entre 0 y 10 V. Por ello, esta función realiza la conversión de voltios a sus respectivas unidades, es decir, N y Nm, mediante interpolación numérica entre unos límites superiores e inferiores. Se procede como sigue:

$$\left.\begin{array}{r} 0\ V \leftrightarrow F_{min} \\ F_v \ \leftrightarrow ¿\,F? \\ 10\ V \leftrightarrow F_{max} \end{array}\right\} \rightarrow \frac{F_{max}-F}{F_{max}-F_{min}} = \frac{10-F_v}{10-0} \rightarrow F = F_{max} - \frac{(10-F_v)\cdot(F_{max}-F_{min})}{10} \tag{2.7}$$

Finalmente, se aplica un anti-windup sobre las acciones de control calculadas.

Función "escala_y"

Al igual que antes, las variables controladas también deben ser enviadas de la RPI al PLC en forma de tensiones que oscilan entre 0 y 10 V. Así, esta función realiza la conversión de sus respectivas unidades, es decir, m/s y grados sexagesimales, a voltios mediante interpolación numérica. La operación es igual que en la anterior función, pero realizada a la inversa, y se obtiene la siguiente ecuación:

$$v_v = 10 - \frac{10\cdot(v_{max}-v)}{(v_{max}-v_{min})} \tag{2.8}$$

Finalmente, se aplica un anti-windup sobre las variables controladas calculadas en forma de tensión.

Función "step"

Esta función llama a la función "modeloSegway" que almacena las ecuaciones ODE y las integra mediante el comando 'odeint', al cual le debe pasar como parámetros de entrada el vector de estados, el de tiempos, los parámetros fijos del modelo y el vector de acciones de control.

Dicho comando devuelve un vector con el valor de las variables de estado en m/s, rad y rad/s, respectivamente, por lo que se almacenan las variables controladas en "vel_lineal" y "pos_angular_grados", convirtiendo la segunda a grados sexagesimales por considerarse que el usuario se familiariza más con esta unidad. Finalmente, se actualiza el vector de estados y se devuelven los valores de las variables controladas.

Función "reset"

Se utiliza para realizar una puesta a cero de algunas variables importantes del proceso antes de iniciar un control, ya que éstas pueden almacenar valores diferentes de experimentos anteriores. También informa en pantalla al usuario de que el reseteo se ha realizado correctamente.

```
## IMPORTACION DE LIBRERIAS
from scipy.integrate import odeint
import numpy as np
import math
## VARIABLES LOCALES
M = 2.5
m = 0.6
l = 1.2
r = 0.2
g = 9.81
Ts = 0.05
X_ant = np.float64([0.0, 0.0, 0.0])
## PIDs MANUAL (50 60 70)
##media_F = -0.1128652
##media_tau = -0.0594064
…
## CONTROL HINFSTRUCT, H-INF, H-2, DK-ITERATION, MIXED-SENSITIVITY, MU-SYNTHESIS
media_F = -0.185626
media_tau = -0.100664
## PROGRAMACION DE FUNCIONES
# FUNCION modeloSegway
def modeloSegway(x, t, M, m, l, r, g, u):
    # Definimos el vector de estados x = [vel_lin, pos_ang, vel_ang]
    vel = x[0]
    ang = x[1]
    vel_ang = x[2]
    F = u[0]
    tau = u[1]
    # Definimos la derivada del vector de estados der_x = [ace_lin, vel_ang, ace_ang]
    der_vel = (F*l + tau*math.cos(ang) - l*r*vel + (l**2)*m*(vel_ang**2)*math.sin(ang) -
g*l*m*math.cos(ang)*math.sin(ang) - l*m*vel*vel_ang*math.cos(ang)*math.sin(ang))/(l*(-
m*((math.cos(ang))**2) + M + m))
    der_ang = vel_ang
    der_vel_ang = (g*l*(m**2)*math.sin(ang) - m*tau- F*l*m*math.cos(ang) - M*tau +
l*(m**2)*vel*vel_ang*math.sin(ang) - (l**2)*(m**2)*(vel_ang**2)*math.cos(ang)*math.sin(ang) +
M*g*l*m*math.sin(ang) + l*m*r*vel*math.cos(ang) + M*l*m*vel*vel_ang*math.sin(ang))/((l**2)*m*(-
m*((math.cos(ang))**2) + M + m))
    derivadas = [der_vel, der_ang, der_vel_ang]
    return derivadas
# FUNCION escala_u
def escala_u(F_v, tau_v, F_max, tau_max, F_min, tau_min):
    F = (F_max - ((10.0-F_v)*(F_max-F_min))/10.0)
```

```
    if (F > F_max):
        F = F_max
    if (F < F_min):
        F = F_min

    tau = (tau_max - ((10.0-tau_v)*(tau_max-tau_min))/10.0)
    if (tau > tau_max):
        tau = tau_max
    if (tau < tau_min):
        tau = tau_min
    return F, tau
# FUNCION escala_y
def escala_y(vellin, posang, vellin_max, posang_max, vellin_min, posang_min):
    vellin_v = 10.0 - (10.0*(vellin_max-vellin))/(vellin_max-vellin_min)
      if (vellin_v > 10.0):
         vellin_v = 10.0
      if (vellin_v < 0.0):
         vellin_v = 0.0

    posang_v = 10.0 - (10.0*(posang_max-posang))/(posang_max-posang_min)
      if (posang_v > 10.0):
         posang_v = 10.0
      if (posang_v < 0.0):
         posang_v = 0.0
    return vellin_v, posang_v
# FUNCION step
def step(F, tau):
    global X_ant, vel_lineal, pos_angular
    # PASO DE INTEGRACION
    U_k = [F, tau]
    t = np.linspace(0, Ts, 2)
    var_estado = odeint(modeloSegway, X_ant, t, args = (M, m, l, r, g, U_k))
    vel_lineal = np.float64(var_estado[1,0])
    pos_angular_grados = np.rad2deg(np.float64(var_estado[1,1]))
##    vel_lineal = np.float64(np.float64(var_estado[1,0])*0.58823529411764708)
##    pos_angular_grados = np.rad2deg(np.float64(np.float64(var_estado[1,1])*57.295779513082323))
    X_ant = np.float64(var_estado[1])
    return vel_lineal, pos_angular_grados
# FUNCION reset
def reset():
    global vel_lineal, pos_angular, X_ant
    vel_lineal = 0.0
    pos_angular_grados = 0.0
    X_ant = [0.0, 0.0, 0.0]
    print ("Segway reseted")
```

Script 1. Script "segway_odeint.py"

2.1.3. Comunicación entre Raspberry Pi y PLC

Para el control del proceso en fase experimental, es imprescindible que la RPI se comunique con el PLC donde se implementa el controlador, y para ello se deben seguir una serie de pasos:

2.1.3.1. Cableado entre Raspberry Pi y PLC

Los valores de acciones de control y variables controladas se envían en forma de variables analógicas, por lo que es necesario establecer un cableado entre PLC y RPI [Ilustración 13]. Por ello, se realizan las siguientes conexiones: tierra PLC – tierra RPI, entrada 1 PLC – salida 1 RPI, entrada 2 PLC -salida 2 RPI, salida 1 PLC – entrada 1 RPI y salida 2 PLC – entrada 2 RPI.

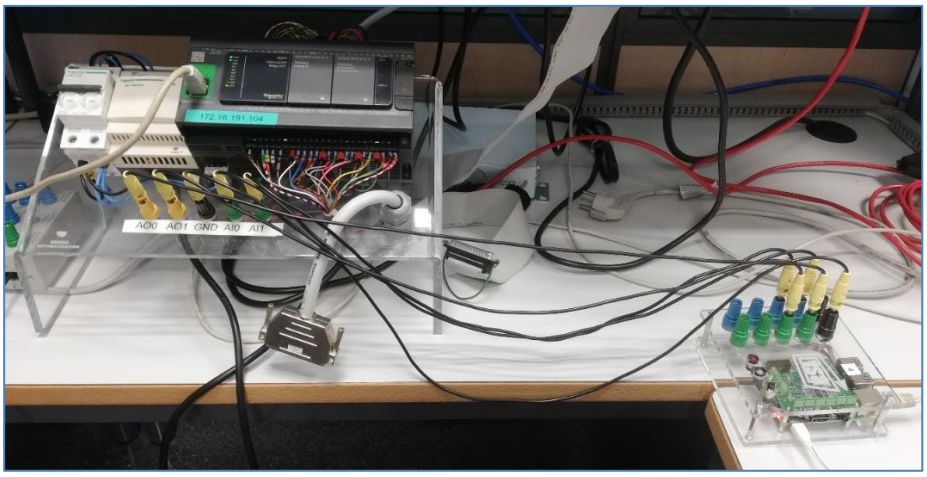

Ilustración 13. Cableado entre PLC y Raspberry Pi (Elaboración propia)

2.1.3.2. Encendido del PLC y arranque del controlador en SoMachine

En segundo lugar, se inicia la conexión entre el control programado en SoMachine y el PLC, y para ello debe conocerse previamente la dirección IP del autómata, cuyo formato suele ser "172.16.191.10X", donde X es un número del 0 al 9 según el puesto asignado en el laboratorio. Una vez iniciado, el controlador se encuentra enviando acciones de control nulas hasta arrancar el modelo de la RPI.

2.1.3.3. Arranque del modelo en Raspberry Pi e inicio del control

En tercer lugar, se arranca la RPI, a la que se le ha asignado la dirección IP estática "172.16.191.8", mediante el programa VNC Viewer. Una vez arrancada, se abre una ventana de símbolo del sistema en la que se debe cambiar el directorio de trabajo a la carpeta en la que se encuentran todos los *scripts* necesarios y ejecutar el *script* "main.py" [Script 2] mediante las líneas:

$\gg cd\ Programas\_python\ HIL\_webserver$
$\gg sudo\ python\ main.py$

Script "main.py"

La RPI almacena otros modelos de procesos además del Segway, el cual ha sido integrado por el autor de este TFM, y este *script* es el encargado de iniciar su control mediante la función "HILTask" en el caso de que el usuario lo solicite. También se encarga de establecer la configuración de luces LED de la RPI.

En el caso de seleccionar el proceso Segway, se reciben las acciones de control del PLC en voltios y se escalan mediante la función "escala_u", incluyéndose después en la llamada a la función "step" que obtiene las variables controladas. Éstas son almacenadas, junto a las acciones de control y el instante de tiempo, en ficheros que sirven posteriormente para visualizar resultados gráficamente, y convertidas mediante la función "escala_y" a tensiones que se envían al PLC.

El código completo de este script se encuentra en el anexo I [Script 33].

```
elif (globals.proceso == 'segway'):
    start_time = time.time()
    # LECTURA DE TENSIONES
    globals.Fv = adc.read_voltage(adc1)
    globals.tauv = adc.read_voltage(adc2)
    # FUNCION escala_u
    globals.F, globals.tau = segway_odeint.escala_u(globals.Fv, globals.tauv, globals.Fmax,
globals.taumax, globals.Fmin, globals.taumin)
    # APLICACIÓN DE ACCION DE CONTROL
    if globals.tiempo <= 10.0:
       globals.F = 0.0
       globals.tau = 0.0
    else:
       globals.F = globals.F - segway_odeint.media_F
       globals.tau = globals.tau - segway_odeint.media_tau
       print("Fuerza de traccion: "+str(globals.F)+" N"+"\n")
       print("Par motor: "+str(globals.tau)+" N m"+"\n")
    # FUNCION step
    globals.vellin, globals.posang = segway_odeint.step(globals.F, globals.tau)
    print("Velocidad lineal: "+str(globals.vellin)+" m/s"+"\n")
    print("Posicion angular: "+str(globals.posang)+" grados"+"\n")
    # ALMACENAMIENTO DE VARIABLES EN FICHEROS
##  with open("controlPID_50ms.txt",'a') as valores:
##  with open("controlPID_60ms.txt",'a') as valores:
##  …
##  if globals.tiempo <= 50.0:
##     valores.write(str(globals.tiempo)+' '+str(globals.F)+' '+str(globals.tau)+'
'+str(globals.vellin)+' '+str(globals.posang)+'\n')
    # FUNCION escala_y
    globals.vellinv, globals.posangv = segway_odeint.escala_y(globals.vellin, globals.posang,
globals.vellin_max, globals.posang_max, globals.vellin_min, globals.posang_min)
    # ENVIO DE TENSIONES
    dac.single_internal(dac1, globals.vellinv, False)
    dac.single_internal(dac2, globals.posangv, False)
    # ACTUALIZACION DEL TIEMPO
    Ts = segway_odeint.Ts
    globals.z = globals.z + 1.0
    globals.tiempo = Ts*globals.z
    print("Tiempo: "+str(globals.tiempo)+" seg"+"\n")
```

Script 2. Script “main.py”

Interfaz de usuario

El proceso anterior se inicia tras escribir “172.16.191.8:8000” en la barra de navegación de un navegador web y, una vez cargada la interfaz de usuario [Ilustración 14], establecer el Segway como proceso a simular, pulsar sobre el botón “Reset”, que tiene relación directa con la función “reset”, y pulsar sobre el botón “Run”, que ejecuta el bucle.

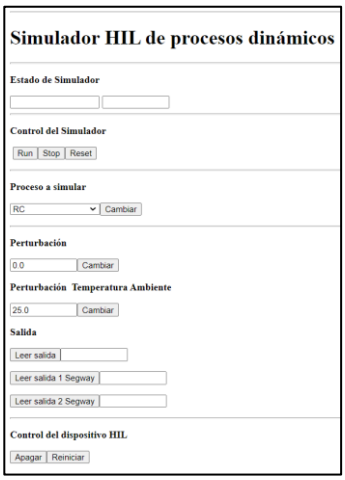

Ilustración 14. Interfaz de usuario: Simulador HIL de procesos dinámicos (Elaboración propia)

Entorno de trabajo

Ya con el controlador del PLC y el modelo de la RPI activados, se dispone de un entorno de trabajo [Ilustración 15] formado por una doble pantalla donde, a la izquierda, se opera con el controlador y, a la derecha, se visualizan en la RPI los constantes valores de acciones de control y variables controladas.

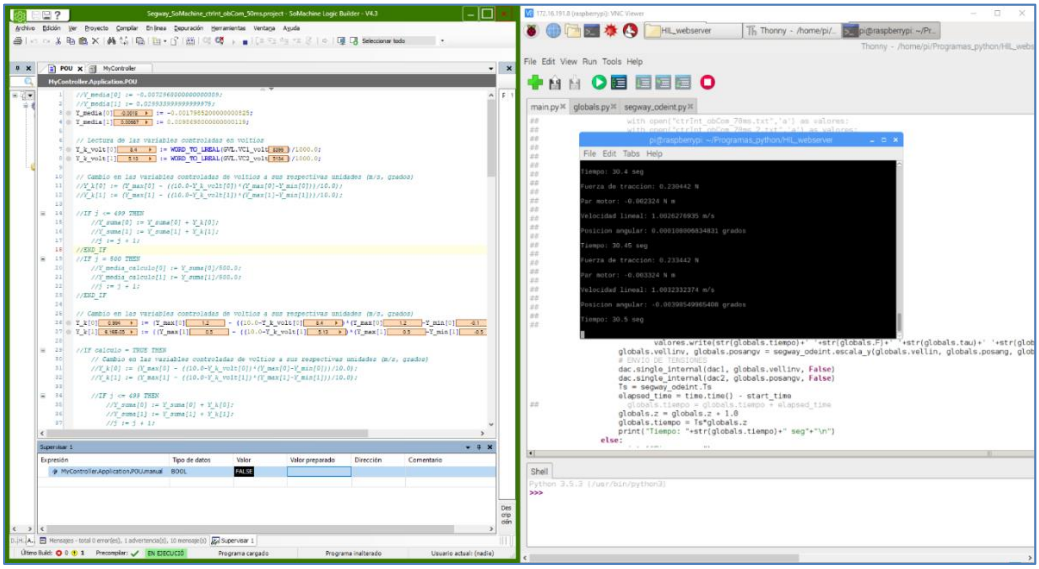

Ilustración 15. Entorno de trabajo disponible para el usuario (Elaboración propia)

2.2. Modelo lineal

El modelo no lineal del Segway es muy potente, pero sus no linealidades, tales como la dependencia de más de una variable en un mismo término, las que provocan dificultades a la hora de llevar a cabo ciertos controles. Por ello, se opta por desarrollar un modelo lineal del mismo que manifieste un comportamiento similar y que facilite dicho procedimiento.

2.2.1. Linealización en MATLAB

Para la linealización en espacio continuo de estados se ha programado en MATLAB el script "Segway_Linealizacion.mlx" [Script 3], el cual se divide en tres apartados:

2.2.1.1. Ecuaciones

Se declaran las variables simbólicas necesarias para definir el modelo basado en (2.1) y (2.2), y mediante el comando 'solve' se obtienen las ecuaciones (2.4) y (2.5). Asimismo, se define la ecuación del tercer estado y el vector de la derivada de estados "dX":

$$dX = \dot{X} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} Ace \\ Vel_{ang} \\ Ace_{ang} \end{bmatrix} \tag{2.9}$$

2.2.1.2. Linealización del modelo en espacio de estados

Se declaran los vectores de variables simbólicas de las acciones de control, de las variables de estado y de las variables controladas, tal y como se hizo en (2.3) y teniendo en cuenta que los dos primeros estados son coincidentes con las salidas.

Después, se define el punto de equilibrio utilizado para realizar la linealización, el cual es con la

plataforma móvil parada y el péndulo invertido formando 90° con la misma y sin rotación de su eje:

$$U_{eq} = \begin{bmatrix} F_{eq} \\ \tau_{eq} \end{bmatrix} = \begin{bmatrix} 0\ N \\ 0\ N\ m \end{bmatrix}, \qquad Y_{eq} = \begin{bmatrix} \dot{x}_{eq} \\ \theta_{eq} \end{bmatrix} = \begin{bmatrix} 0\ m/s \\ 0\ ° \end{bmatrix}, \qquad X_{eq} = \begin{bmatrix} \dot{x}_{eq} \\ \theta_{eq} \\ \dot{\theta}_{eq} \end{bmatrix} = \begin{bmatrix} 0\ m/s \\ 0\ ° \\ 0\ rad/s \end{bmatrix} \tag{2.10}$$

La representación interna del modelo no lineal posee la siguiente estructura:

$$\begin{cases} \dot{x} = f(x,u) \\ y = g(x,u) \end{cases} \tag{2.11}$$

Sin embargo, se busca un modelo linealizado alrededor del punto de equilibrio de la forma:

$$\begin{cases} \Delta\dot{x} = A \cdot \Delta x + B \cdot \Delta u \\ \Delta y = C \cdot \Delta x + D \cdot \Delta u \end{cases} \tag{2.12}$$

Donde el símbolo Δ se traduce en variables incrementales, es decir, en la diferencia entre el valor absoluto y el valor de equilibrio, pero al considerarse un punto de equilibrio nulo puede realizarse una simplificación y trabajar directamente en valores absolutos:

$$\begin{cases} \dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = A \cdot x + B \cdot u = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{32} \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \\ y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = C \cdot x + D \cdot u = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \end{cases} \tag{2.13}$$

De esta forma, se realiza el cálculo de las matrices A, B, C y D mediante derivación parcial de las funciones de la representación interna del modelo no lineal, aplicada con 'jacobian'. Posteriormente, se sustituyen las variables simbólicas de los estados y las entradas por sus valores de equilibrio mediante el comando 'subs':

$$A = \left(\frac{\partial f(x,u)}{\partial x}\right)_{eq} = \begin{bmatrix} \left(\frac{\partial f_1}{\partial x_1}\right)_{eq} & \left(\frac{\partial f_1}{\partial x_2}\right)_{eq} & \left(\frac{\partial f_1}{\partial x_3}\right)_{eq} \\ \left(\frac{\partial f_2}{\partial x_1}\right)_{eq} & \left(\frac{\partial f_2}{\partial x_2}\right)_{eq} & \left(\frac{\partial f_2}{\partial x_3}\right)_{eq} \\ \left(\frac{\partial f_3}{\partial x_1}\right)_{eq} & \left(\frac{\partial f_3}{\partial x_2}\right)_{eq} & \left(\frac{\partial f_3}{\partial x_3}\right)_{eq} \end{bmatrix} = \begin{bmatrix} -\frac{r}{m} & -\frac{gm}{M} & 0 \\ 0 & 0 & 1 \\ \frac{r}{Ml} & \frac{glm^2 + Mglm}{Ml^2m} & 0 \end{bmatrix} = \begin{bmatrix} -0.08 & -2.3544 & 0 \\ 0 & 0 & 1 \\ 0.0667 & 10.137 & 0 \end{bmatrix} \tag{2.14}$$

$$B = \left(\frac{\partial f(x,u)}{\partial u}\right)_{eq} = \begin{bmatrix} \left(\frac{\partial f_1}{\partial u_1}\right)_{eq} & \left(\frac{\partial f_1}{\partial u_2}\right)_{eq} \\ \left(\frac{\partial f_2}{\partial u_1}\right)_{eq} & \left(\frac{\partial f_2}{\partial u_2}\right)_{eq} \\ \left(\frac{\partial f_3}{\partial u_1}\right)_{eq} & \left(\frac{\partial f_3}{\partial u_2}\right)_{eq} \end{bmatrix} = \begin{bmatrix} \frac{1}{M} & \frac{1}{Ml} \\ 0 & 0 \\ -\frac{1}{Ml} & -\frac{M+m}{Ml^2m} \end{bmatrix} = \begin{bmatrix} 0.4 & 0.3333 \\ 0 & 0 \\ -0.3333 & -1.4352 \end{bmatrix} \tag{2.15}$$

$$C = \left(\frac{\partial g(x,u)}{\partial x}\right)_{eq} = \begin{bmatrix} \left(\frac{\partial g_1}{\partial x_1}\right)_{eq} & \left(\frac{\partial g_1}{\partial x_2}\right)_{eq} & \left(\frac{\partial g_1}{\partial x_3}\right)_{eq} \\ \left(\frac{\partial g_2}{\partial x_1}\right)_{eq} & \left(\frac{\partial g_2}{\partial x_2}\right)_{eq} & \left(\frac{\partial g_2}{\partial x_3}\right)_{eq} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.16}$$

$$D = \left(\frac{\partial g(x,u)}{\partial u}\right)_{eq} = \begin{bmatrix} \left(\frac{\partial g_1}{\partial u_1}\right)_{eq} & \left(\frac{\partial g_1}{\partial u_2}\right)_{eq} \\ \left(\frac{\partial g_2}{\partial u_1}\right)_{eq} & \left(\frac{\partial g_2}{\partial u_2}\right)_{eq} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{2.17}$$

Todo este procedimiento es una simple aplicación de la teoría de sistemas multivariables [15].

2.2.1.3. Sustitución de los parámetros y obtención del espacio de estados

Las matrices están expresadas en forma simbólica, por lo que es necesario declarar los parámetros fijos del modelo y sustituirlos en las mismas, obteniendo así fracciones que son detectadas como símbolos. Por ello, mediante el comando 'double' éstas ya son reconocidas como dígitos de doble precisión. Finalmente, se crea el espacio continuo de estados mediante 'ss'.

1. Ecuaciones

(1) F = (M+m)·Ace + m·l·cos(Ang)·Ace_ang - m·l·sin(Ang)·(Vel_ang)^2 + r·Vel

(2) m·g·l·sin(Ang) - tau = m·l^2·Ace_ang + m·l·Ace·cos(Ang) - m·l·Vel·Vel_ang·sin(Ang)

Despejamos las incógnitas Ace y Ace_ang:

```
clearvars;
syms M m l g r F tau ang vel_ang ace_ang vel ace;
param = [M m l r g];
Ec1 = F == (M+m)*ace + m*l*cos(ang)*ace_ang - m*l*sin(ang)*(vel_ang)^(2) + r*vel;
Ec2 = m*g*l*sin(ang) - tau == m*(l)^(2)*ace_ang + m*l*ace*cos(ang) - m*l*vel*vel_ang*sin(ang);
modelo = [Ec1; Ec2];
[Ace,Ace_ang] = solve(modelo,ace,ace_ang);
Vel_ang = vel_ang;
```

Se define el vector de la derivada de estados:

```
dX = [Ace; Vel_ang; Ace_ang];
```

2. Linealización del modelo en espacio de estados

```
U = [F tau];
X = [vel ang vel_ang];
```

Puesto que las dos salidas coinciden con las dos primeras variables de estado, el vector de salidas queda tal que así:

```
Y = [X(1); X(2)];

F_eq = 0; tau_eq = 0;
vel_eq = 0; ang_eq = 0; velang_eq = 0;
U_eq = [F_eq tau_eq];
Y_eq = [vel_eq ang_eq];
X_eq = [vel_eq ang_eq velang_eq];
A_sym = subs(jacobian(dX, X),[X U],[X_eq U_eq]);
B_sym = subs(jacobian(dX,U),[X U],[X_eq U_eq]);
C_sym = subs(jacobian(Y,X),[X U],[X_eq U_eq]);
D_sym = subs(jacobian(Y,U),[X U],[X_eq U_eq]);
```

3. Sustitución de los parámetros y obtención del espacio de estados

```
M_p = 2.5;
m_p = 0.6;
l_p = 1.2;
r_p = 0.2;
g_p = 9.81;
param_p = [M_p m_p l_p r_p g_p];
A = double(subs(A_sym,param,param_p));
B = double(subs(B_sym,param,param_p));
C = double(subs(C_sym,param,param_p));
D = double(subs(D_sym,param,param_p));
SYS = ss(A,B,C,D);
```

Script 3. Script "Segway_Linealizacion.mlx"

2.2.2. Comparativa entre el modelo no lineal y el modelo lineal

Para comprobar la similitud entre modelo no lineal y lineal, se ha desarrollado un fichero "Segway_ModeloNL_vs_ModeloL.slx" [Ilustración 16] en Simulink en el que ambos son sometidos a un simple control mediante PIDs discretizados. Para ello, se ha impuesto un cambio de referencia de la

velocidad lineal de 0 a 1 m/s en t = 5 segundos sin modificar la de la posición angular.

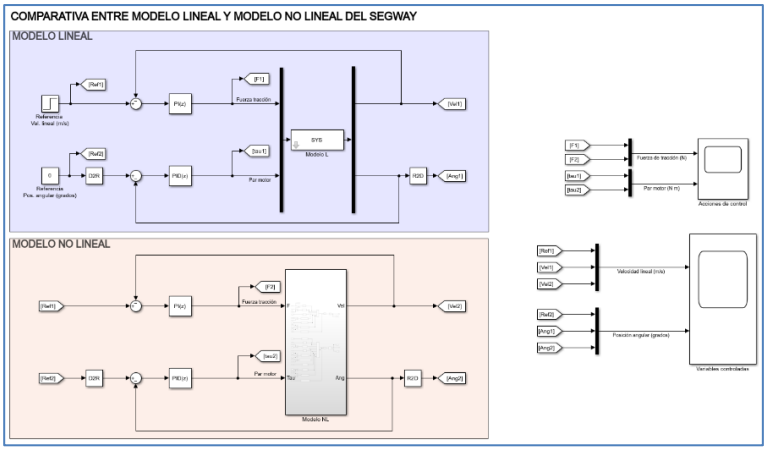

Ilustración 16. Fichero "Segway_ModeloNL_vs_ModeloL_slx" (Elaboración propia)

Los valores de las acciones de control [Ilustración 17] y de las variables controladas [Ilustración 18] son idénticos en ambos casos, ya que se están solapando las trayectorias, por lo que la linealización del modelo es correcta.

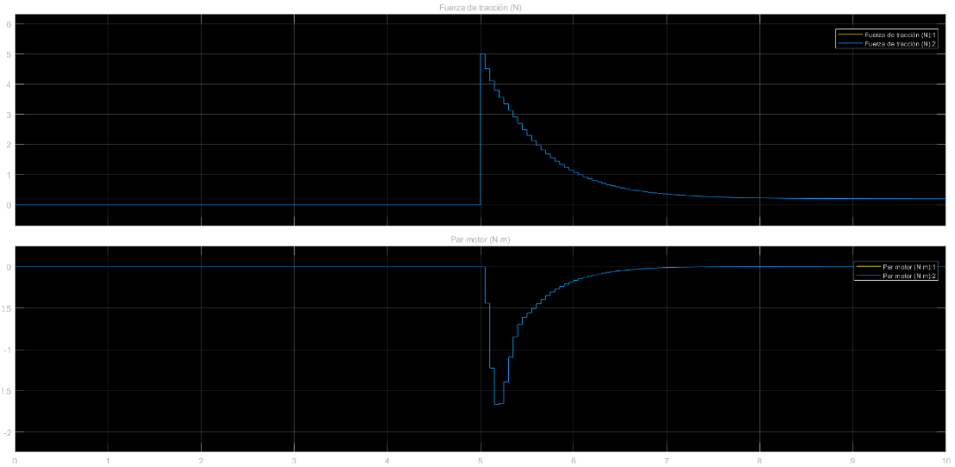

Ilustración 17. Fichero "Segway_ModeloNL_vs_ModeloL.slx": acciones de control (Elaboración propia)

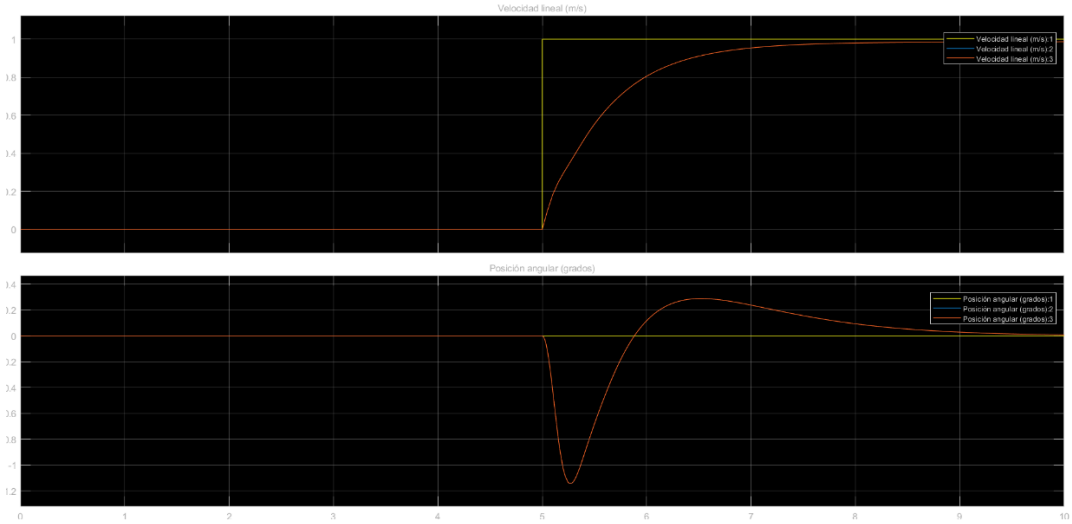

Ilustración 18. Fichero "Segway_ModeloNL_vs_ModeloL.slx": variables controladas (Elaboración propia)

Capítulo 3

Control del Segway mediante controladores PID

Una vez definido completamente el modelo no lineal y obtenido un modelo linealizado exitoso del mismo, siendo ambos muy inestables debido principalmente al balanceo del péndulo, es momento de aplicar diferentes técnicas de control capaces de estabilizarlo. En este capítulo, se implementa el control basado en controladores PID.

3.1. Antecedentes del controlador PID

El control basado en controladores con acción Proporcional – Integral – Derivativo (en adelante, PID) es uno de los más antiguos que se conoce, ya que en el siglo XVIII empezó a emplearse dos de estas tres acciones para controlar la velocidad de conducción de las máquinas de vapor, y a inicios del siglo XX el inventor Elmer Sperry lo desarrolló al completo con el fin de automatizar la dirección de los barcos de la Armada de los Estados Unidos [16].

Así, es uno de los controles más utilizados actualmente en la industria por su elevado rendimiento, y es por ello que se ha optado por implementarlo en primer lugar para el modelo de Segway.

3.2. Concepto teórico

Este controlador [Ilustración 19] recibe la señal de error, calculada como la diferencia entre la referencia y el valor real de la salida, y la aplica sobre los términos del controlador implementado, sea este P, PI, PD o PID, de manera que al sumarlos se obtiene la acción de control aplicada sobre el proceso.

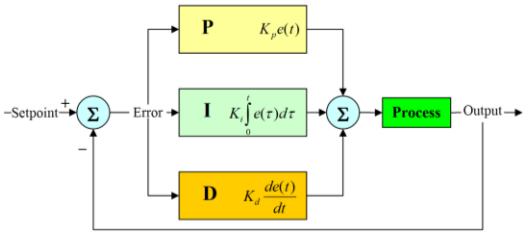

Ilustración 19. Estructura del control PID (Tomada de [17])

Cada término tiene unos objetivos distintos, y éstos se van a desgranar a partir de la información mostrada en [18]:

Acción Proporcional

Se trata de una acción de control que multiplica el error por una constante Kp con el objetivo de

reducirlo, es decir, si el error es grande la acción de control también lo será, tendiendo así a minimizarlo:

$$u_P(t) = K_P \cdot e(t) \tag{3.1}$$

De esta forma, al aumentar el valor de Kp aumenta la velocidad de respuesta del sistema y disminuye el error en régimen permanente, pero en el caso de excederse puede provocar la inestabilidad del sistema.

Acción Derivativa

Como se ha dicho antes, una acción proporcional desmedida puede provocar que el sistema se acelere demasiado y se produzcan oscilaciones en torno a la referencia fijada. Por ello, a veces es necesario que el controlador reconozca la velocidad a la que el sistema viaja hacia la mismo, siendo así capaz de frenarlo con antelación sin provocar sobrepulsos u oscilaciones, y es ahí donde aparece la acción derivativa, que es proporcional a la derivada del error y que se le podría llamar "velocidad" del error.

$$u_D(t) = K_D \cdot \frac{\partial e(t)}{\partial t} = K_P \cdot T_D \cdot \frac{\partial e(t)}{\partial t} \tag{3.2}$$

Esta acción está afectada por el uso de una constante Kd que, al aumentar su valor, aumenta también la estabilidad del sistema a costa de disminuir levemente la velocidad del mismo. Sin embargo, no tiene efecto alguno sobre el error en régimen permanente.

Acción Integral

Pese a que un controlador esté formado por una acción de control proporcional y una derivativa, es probable que siga existiendo cierto error en régimen permanente. Es por ello que se decide utilizar la acción integral, que es proporcional a la integral del error, es decir, a la suma o acumulación de error. Conforme pasa el tiempo, el error se acumula poco a poco y hace que la acción integral aumente, pero debe utilizarse con cuidado ya que adhiere cierta inercia al sistema que puede volverlo inestable.

$$u_I(t) = K_I \cdot \int e(t)\, d(t) = \frac{K_P}{T_I} \cdot \int e(t)\, d(t) \tag{3.3}$$

Así, al aumentar la constante Ki se disminuye el error en régimen permanente y aumenta ligeramente la velocidad del sistema, pero se puede producir un aumento de la inestabilidad del mismo.

Como conclusión, puede decirse que los tres tipos de acción de control son válidos y presentan diferentes ventajas e inconvenientes, por lo que previamente debe analizarse qué tipo de controlador requiere el proceso en el que se está trabajando y posteriormente encontrar un compromiso entre velocidad de respuesta, error en régimen permanente e inestabilidad. Aplicando la transformada de Laplace 's' se obtiene la siguiente expresión en el plano continuo:

$$u(t) = u_P(t) + u_D(t) + u_I(t) \rightarrow \frac{U(s)}{E(s)} = K_P + K_D \cdot s + \frac{K_I}{s} \tag{3.4}$$

3.3. Fase de simulación en MATLAB

3.3.1. Controlador en MATLAB

Pese a haber presentado el PID en el plano continuo, no es posible llevarlo a la práctica en el control del Segway en este estado, pues el lenguaje ST de SoMachine no permite el uso de la transformada de

Laplace. Por ello, es necesario discretizarlo a cierto período de muestreo mediante la transformada en 'z', de manera que pueda convertirse en ecuación en diferencias, la cual sí es programable en ST.

$$\frac{U(z)}{E(z)} = K_{P_z} + K_{D_z} \cdot \frac{1}{T_s} \cdot \frac{z-1}{z} + K_{I_z} \cdot T_s \cdot \frac{1}{z-1} \tag{3.5}$$

En esta expresión, el término de la acción derivativa se discretiza mediante el método del rectángulo posterior, $s = \frac{z-1}{T_s \cdot z}$, y el de la acción integral mediante el del rectángulo anterior, $s = \frac{z-1}{T_s}$.

Realizando las operaciones oportunas y considerando que el término "k" corresponde al instante actual, "k-1" al instante anterior y "k-2" a dos instantes antes, se obtiene la ecuación de un PID:

$$u(k) = u(k-1) + \left(K_{P_z} + \frac{K_{D_z}}{T_s}\right) \cdot e(k) + \left(K_{I_z} \cdot T_s - K_{P_z} - 2 \cdot \frac{K_{D_z}}{T_s}\right) \cdot e(k-1) + \frac{K_{D_z}}{T_s} \cdot e(k-2) \tag{3.6}$$

Con el objetivo de realizar simulaciones del control del Segway basado en controladores PID, se ha desarrollado un fichero "Segway_ModeloNL_PID.slx" [Ilustración 20] en Simulink, en el que se han realizado pruebas con los PID discretizados a períodos de muestreo muy reducidos, y esto es debido al carácter inestable del modelo, el cual hace que se requieran rápidas acciones de control.

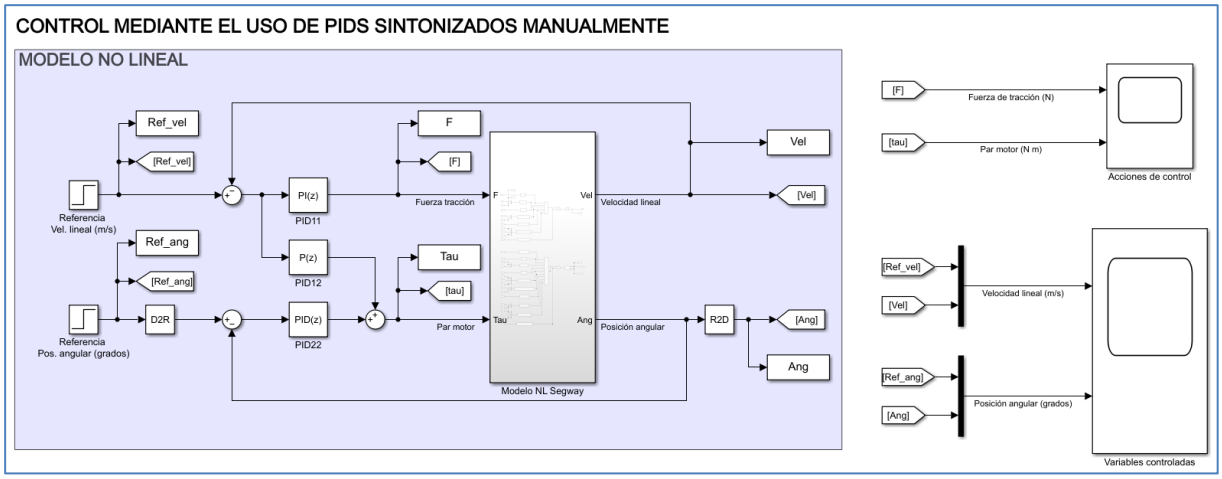

Ilustración 20. Fichero "Segway_ModeloNL_PID.slx" (Elaboración propia)

Asimismo, los PIDs se implementan según una serie de relaciones que pueden ser lógicas si se observa detenidamente el modelo del Segway [Ilustración 9]. De esta forma, se va a realizar un análisis de estas tres relaciones:

Asimismo, se establece una relación directa entre la fuerza de tracción ejercida sobre la plataforma móvil y la velocidad lineal de la misma, así como entre el par motor aplicado sobre el péndulo y la posición angular del mismo, tal y como es lógico si se realiza un pequeño análisis del modelo del Segway mostrado en la Ilustración 12. Asimismo, se ha comprobado que los resultados mejoran cuando el par motor del péndulo recibe información sobre la velocidad lineal de la plataforma. Por ello, se van a analizar estas tres relaciones:

Y1: Velocidad lineal (m/s) – U1: Fuerza de tracción (N)

Ciertamente, la plataforma móvil no puede obtener velocidad lineal sin que se le aplique cierta fuerza de tracción, por lo que la relación es obvia. En este caso, la mejor opción es un controlador PI, es decir, una acción proporcional que reduzca el tiempo de alcance de la referencia de velocidad lineal y una

acción integral que reduzca el error entre dicha referencia y la velocidad lineal real.

$$\frac{U_{11}}{E_{11}} = K_{P_{11}} + K_{I_{11}} \cdot T_s \cdot \frac{1}{z-1}$$

Puesto que se considera que la plataforma tiene un movimiento unidireccional, es muy difícil que oscile su comportamiento, por lo que la acción integral no afecta negativamente a su estabilidad. Por tanto, no es necesario el uso de una acción derivativa, ya que únicamente se estaría ralentizando el alcance de la referencia.

Y2: Posición angular (rad) – U2: Par motor (N m)

Uno de los objetivos es mantener una posición perpendicular del péndulo respecto a la plataforma, pero pequeños cambios en la velocidad lineal pueden producir balanceos bruscos que lo inestabilicen por completo sin poder recuperar su posición original. Por ello, se hace uso de un PID:

$$\frac{U_{22}}{E_{22}} = K_{P_{22}} + K_{D_{22}} \cdot \frac{1}{T_s} \cdot \frac{z-1}{z} + K_{I_{22}} \cdot T_s \cdot \frac{1}{z-1} \rightarrow$$

Así, la acción proporcional aumenta la rapidez de vuelta del péndulo a su referencia, la acción integral reduce el error de posición, es decir, evita que permanezca inclinado, y la acción derivativa reduce al mínimo la posibilidad de inestabilizarse.

Y1: Velocidad lineal (m/s) – U2: Par motor (N m)

Mediante simulación, se ha observado que el cálculo del par motor es más preciso tras recibir información de la velocidad lineal, ya que, considerando una acción proporcional y sumándola a u_{22}, se consigue anular el balanceo del péndulo casi en su totalidad. Por ello, se hace uso de un P:

$$\frac{U_{12}}{E_{12}} = K_{P_{12}}$$

En función del período de muestreo, se han empleado diferentes valores de las constantes de los PIDs [Tabla 2], los cuales fueron obtenidos de primera mano a través de la herramienta PID Tuner y posteriormente modificados por el usuario hasta dar con la configuración óptima:

Tabla 2. Constantes de los controladores PID (Elaboración propia)

CONSTANTES DE LOS CONTROLADORES PID			
	$T_s = 50\ ms$	$T_s = 60\ ms$	$T_s = 70\ ms$
$K_{P_{11}}$	5.625	5.625	5.625
$K_{I_{11}}$	0.3637	0.3637	0.3637
$K_{P_{22}}$	-56.93	-56.93	-45.432
$K_{I_{22}}$	-52.3083	-52.3083	-48.703
$K_{D_{22}}$	-9.7622	-9.7622	-9.2775
$K_{P_{12}}$	-1.303	-1.2	-1.2

3.3.2. Resultados

Para una simulación de 30 segundos de duración, con un cambio en la referencia de la velocidad lineal de 0 a 1 m/s en t = 10 segundos y manteniendo a 0° la de la posición angular, se aprecian resultados muy satisfactorios en las acciones de control [Ilustración 21] y variables controladas [Ilustración 22].

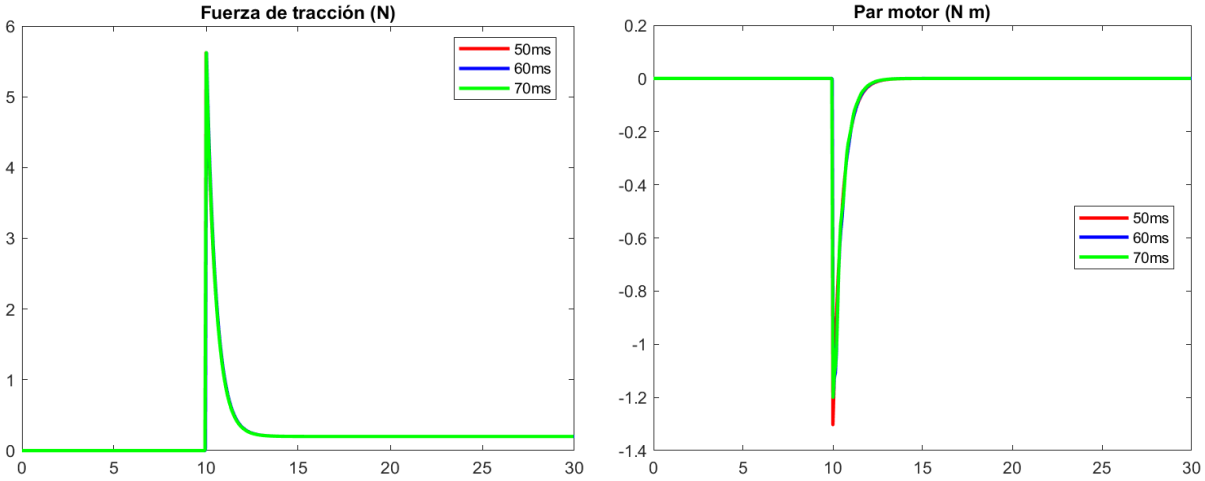

Ilustración 21. Control PID simulación – Acc. Control: F. tracción y par motor (Elaboración propia)

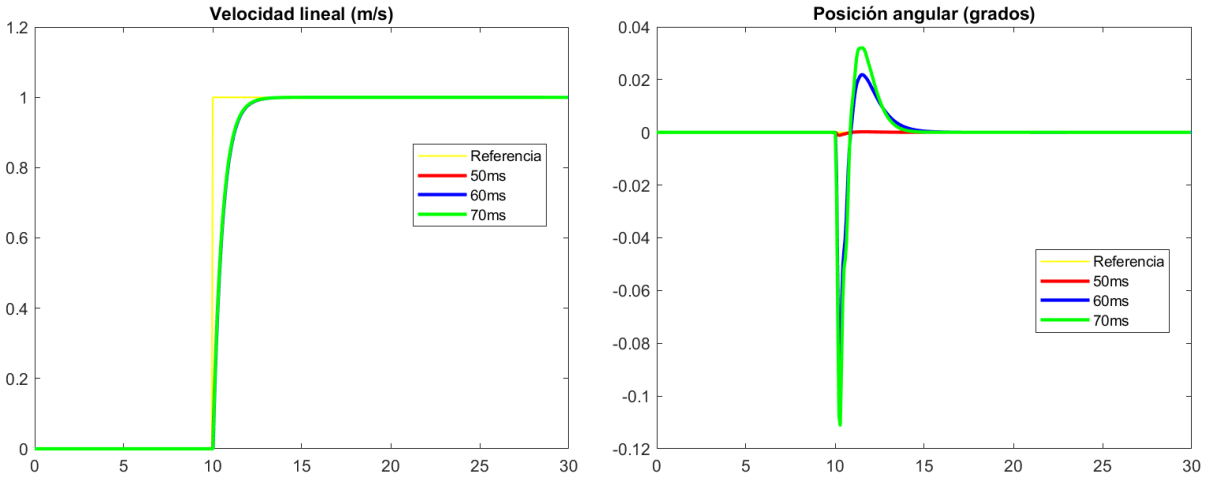

Ilustración 22. Control PID simulación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

La velocidad lineal alcanza la nueva referencia en aproximadamente 3.5 segundos con un error de posición prácticamente nulo, y la posición angular del péndulo apenas se ve afectada por dicho cambio, especialmente para Ts = 50 ms, ya que vuelve a los 0° en 5 segundos y el balanceo es casi inapreciable.

Las gráficas anteriores se han obtenido gracias al script "param_Segway_PID.mlx" [Script 4] desarrollado en MATLAB, del cual sólo va a mostrarse la estructura donde se eligen los parámetros y se obtiene la gráfica de la velocidad lineal, siendo idéntica la programación de las demás. Este script se ha diseñado para comodidad del usuario, debiendo responder "SI" a las preguntas que se le realiza en el caso de desear visualizar ciertos resultados, los cuales pueden ser únicamente de simulación para todos los períodos de muestreo o, si se escoge un período concreto, de la simulación, de las pruebas intermedias en Python y de la fase experimental en RPI.

El código completo de este script se encuentra en el Anexo I [Script 34], y la estructura del mismo respecto a la obtención de las gráficas se repite para todos los controles programados.

1. Parámetros y simulación en MATLAB

```
…
M = 2.5;
m = 0.6;
l = 1.2;
r = 0.2;
g = 9.81;
escalon = 10;
graf_MAT = input("¿Mostrar SÓLO los resultados de los diferentes Ts en MATLAB?: ",'s'); % Responder
SI en caso de desear mostrarlos
…
elseif graf_MAT == "SI"
    Ts_v = [0.05; 0.06; 0.07];
    for i = 1:1:3
        Ts = Ts_v(i);
        switch Ts
            case 0.05
                Kp11 = 5.625; Ki11 = 0.3637; Kp12 = -1.303; Kp22 = -56.9300; Ki22 = -52.3083; Kd22
= -9.7622; tiempo = 30.0;
            …
            otherwise
                disp("Período de muestreo no disponible");
        end
        sim("Segway_ModeloNL_PID.slx");
        if Ts == 0.05
            tiempo_MAT_50 = Vel.Time;
            Ref_vel_50 = Ref_vel.Data;
            Ref_ang_50 = Ref_ang.Data;
            Vel_MAT_50 = Vel.Data;
            Ang_MAT_50 = Ang.Data;
            F_MAT_50 = F.Data;
            Tau_MAT_50 = Tau.Data;
        end
    end
end
```

2. Resultados

```
…
elseif graf_MAT == "SI"
    Velocidad = figure(1);
    plot(tiempo_MAT_50,Ref_vel_50,'y','LineWidth',1); hold on;
    plot(tiempo_MAT_50,Vel_MAT_50,'r','LineWidth',2); hold on;
    plot(tiempo_MAT_60,Vel_MAT_60,'b','LineWidth',2); hold on;
    plot(tiempo_MAT_70,Vel_MAT_70,'g','LineWidth',2);
    title('Velocidad lineal (m/s)');
    legend('Referencia','50ms','60ms','70ms','Location','Best'); hold off;
    …
end
```

Script 4. Script "param_Segway_PID.mlx"

3.4. Fase experimental en Raspberry Pi y PLC

En simulación, el funcionamiento del control basado en PIDs es excelente, pero esto no tiene por qué ocurrir en la fase experimental, donde se utiliza una RPI y un PLC, ya que pueden existir problemas implícitos en el hardware utilizado o componentes internos e indetectables en el propio controlador que empeoren el comportamiento del proceso.

La herramienta PLC Coder de MATLAB/Simulink, mencionada anteriormente, se encarga de expresar en lenguaje ST los PIDs como la ecuación en diferencias (3.6). Para ello, simplemente deben integrarse estos controladores dentro de subsistemas, y de esta manera la herramienta puede generar el código, ya listo para ser almacenado en un proyecto de SoMachine.

3.4.1. Controlador en SoMachine

Siguiendo este procedimiento, se ha creado el proyecto "Segway_SoMachine_PID.project" [Script 5] en SoMachine, detallado a continuación. No obstante, sólo va a visualizarse la implementación del controlador, dejando así el código completo en el Anexo I [Script 35]. Aquellas líneas que comienzan con el símbolo "//" son comentarios que refuerzan el contenido del código o fragmentos de código que no están siendo utilizados.

Declaración de variables

Se declaran todas las variables utilizadas en el proyecto, teniendo casi todas ellas un significado intuitivo excepto las siguientes:

- "**manual**": variable booleana utilizada para cambiar el estado del controlador, es decir, para pasar del envío de acciones de control nulas al inicio del control.
- "**iter1**": variable booleana utilizada para no volver a entrar en la estructura de selección de los parámetros de los PID después de la primera iteración.
- "**Integrador_DSTATE11**", "**rtb_Tsamp22**", "**Integrador_DSTATE22**", "**UD_DSTATE22**": variables reales generadas por PLC Coder con el objetivo de desarrollar las ecuaciones de los PID.
- "**j**", "**Y_media**", "**Y_suma**", "**Y_media_calculo**": variables utilizadas para calcular el valor medio de las variables controladas cuando se están recibiendo valores nulos, es decir, el offset existente entre el valor que se envía desde la RPI y el que se lee en el PLC.

Elección de parámetros de los PID

Es una estructura IF en la que se seleccionan los parámetros de los PIDs según el período de muestreo escogido, y no se vuelve a acceder a ella tras la primera iteración.

Lectura de las variables controladas en voltios

Se almacenan las variables controladas recibidas en las variables globales "VC1_volt" y "VC2_volt". No obstante, tienen unidades de milivoltios y están siendo leídas como palabras, por lo que deben ser convertidas a variables numéricas reales y divididas por 1000 para almacenarlas en voltios.

Cálculo de las medias de las variables controladas

Esta sección se encuentra comentada porque sólo se utiliza inicialmente para calcular los valores del offset de las variables controladas. Para ello, se realiza su conversión de voltios a sus respectivas unidades (m/s y grados sexagesimales) y se acumulan en el vector "Y_suma" hasta llegar a los 500 valores. Finalmente, se obtiene la media de todos estos valores, calculada como el cociente entre los valores de "Y_suma" y 500. Estas medias ya se encuentran almacenadas en el vector "Y_media".

Conversión de las variables controladas de voltios a sus unidades

Se convierten las variables controladas de voltios a sus respectivas unidades (m/s y grados sexagesimales), pero ahora se les resta el offset calculado previamente, obteniendo así un valor mucho más cercano al que realmente se envía desde la RPI.

Arranque del controlador

Como se mencionó anteriormente, el controlador debe arrancarse antes que el modelo para que éste no inestabilice, y por este motivo se implementa una estructura IF en la que se envían acciones de

control nulas mientras el estado de “manual” sea TRUE, lo cual se da siempre al inicio y así es hasta que el usuario decide manipular su estado. Esta circunstancia se da cuando el proceso en la RPI se ha iniciado y han transcurrido aproximadamente 10 segundos, momento en el que “manual” cambia a FALSE y se accede a la segunda sección de la estructura.

En esta sección, el proceso ya intenta alcanzar la referencia fijada, y para ello inicia con un anti-windup de las variables controladas y la conversión de la segunda de grados sexagesimales a radianes. Posteriormente, calcula los errores entre referencia y variable controlada e inmediatamente las acciones de control, las cuales son filtradas por unos anti-windup, convertidas a voltios y enviadas a la RPI a través de variables globales.

```
// ARRANQUE DEL CONTROLADOR
IF manual = TRUE THEN
	// Imposición inicial en las acciones de control (N, N m) para evitar que el modelo se
inestabilice
	U_k[0] := 0.0;
	U_k[1] := 0.0;
	// Cambio en las acciones de control de sus respectivas unidades (N, N m) a voltios
	U_k_volt[0] := 10.0 - (10.0*(U_max[0]-U_k[0]))/(U_max[0]-U_min[0]);
	U_k_volt[1] := 10.0 - (10.0*(U_max[1]-U_k[1]))/(U_max[1]-U_min[1]);
	// Escritura de las acciones de control en voltios
	GVL.AC1_volt := LREAL_TO_WORD(U_k_volt[0]*1000.0);
	GVL.AC2_volt := LREAL_TO_WORD(U_k_volt[1]*1000.0);
ELSIF manual = FALSE THEN
	// Anti-windup de las variables controladas
	IF Y_k[0] > Y_max[0] THEN
		Y_k[0] := Y_max[0];
	ELSIF Y_k[0] < Y_min[0] THEN
		Y_k[0] := Y_min[0];
	END_IF
	IF Y_k[1] > Y_max[1] THEN
		Y_k[1] := Y_max[1];
	ELSIF Y_k[1] < Y_min[1] THEN
		Y_k[1] := Y_min[1];
	END_IF
	// Conversión de grados sexagesimales a radianes
	Y_k[1] := Y_k[1]*(3.14159265358979323846264338327950288419716939937510/180.0);
	// Cálculo del error entre referencia y salida
	E_k[0] := Ref[0] - Y_k[0];
	E_k[1] := Ref[1] - Y_k[1];
	// Cálculo de la acción de control "fuerza de tracción (F)"
	U11_k := Kp11*E_k[0] + Integrador_DSTATE11;
	Integrador_DSTATE11 := (Ki11*Ts*E_k[0]) + Integrador_DSTATE11;
	U12_k := Kp12*E_k[0];
	U_k[0] := U11_k;
	// Cálculo de la acción de control "par motor (tau)"
	U12_k := Kp12*E_k[0];
	rtb_Tsamp22 := (Kd22/Ts)*E_k[1];
	U22_k := (Kp22*E_k[1] + Integrador_DSTATE22) + (rtb_Tsamp22 - UD_DSTATE22);
	UD_DSTATE22 := rtb_Tsamp22;
	Integrador_DSTATE22 := (Ki22*Ts*E_k[1]) + Integrador_DSTATE22;
	U_k[1] := U12_k + U22_k;
	// Anti-windup de las acciones de control
	IF U_k[0] > U_max[0] THEN
		U_k[0] := U_max[0];
	ELSIF U_k[0] < U_min[0] THEN
		U_k[0] := U_min[0];
	END_IF
	IF U_k[1] > U_max[1] THEN
		U_k[1] := U_max[1];
	ELSIF U_k[1] < U_min[1] THEN
		U_k[1] := U_min[1];
	END_IF
```

```
        // Cambio en las acciones de control de sus respectivas unidades (N, N m) a voltios
        U_k_volt[0] := 10.0 - (10.0*(U_max[0]-U_k[0]))/(U_max[0]-U_min[0]);
        U_k_volt[1] := 10.0 - (10.0*(U_max[1]-U_k[1]))/(U_max[1]-U_min[1]);
        // Escritura de las acciones de control en voltios
        GVL.AC1_volt := LREAL_TO_WORD(U_k_volt[0]*1000.0);
        GVL.AC2_volt := LREAL_TO_WORD(U_k_volt[1]*1000.0);
END_IF
```

Script 5. Proyecto "Segway_SoMachine_PID.project"

3.4.2. Resultados

Pese a los diferentes inconvenientes de tiempos de computación, apreciados principalmente en control robusto, y de variabilidad en la cuantificación, la fase experimental del control basado en PIDs ha resultado exitosa, ya que tanto las acciones de control [Ilustración 23] como las variables controladas [Ilustración 24] presentan un comportamiento similar a las de simulación, aunque con un carácter mucho más oscilatorio en torno a sus referencias.

Para Ts = 50 ms, la oscilación de la posición angular ante el cambio de referencia en la velocidad lineal es mayor en la fase experimental que en la de simulación, para Ts = 60 ms muestra una gran similitud entre ambas fases y para Ts = 70 ms comienza a apreciarse cierto principio de inestabilidad en el par motor.

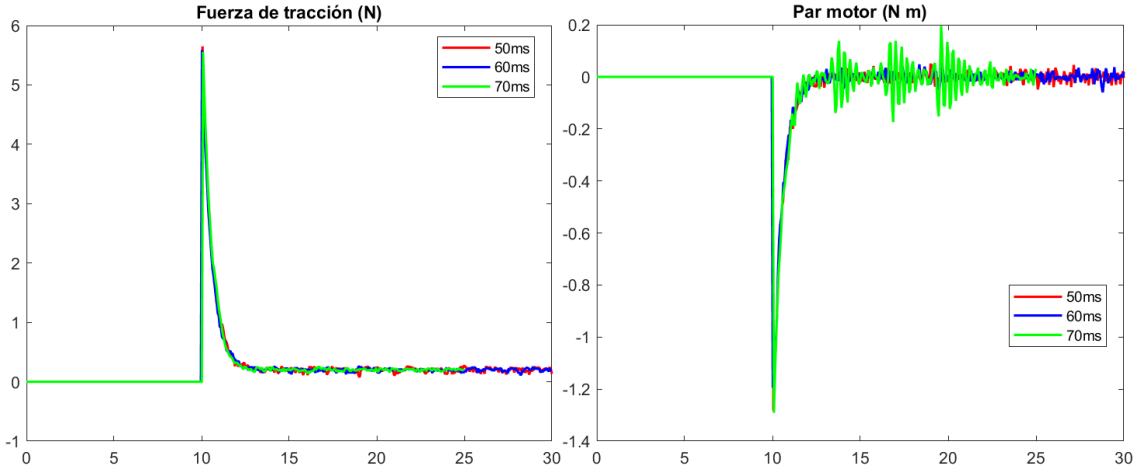

Ilustración 23. Control PID experimentación – Acc. Control: F. tracción y par motor (Elaboración propia)

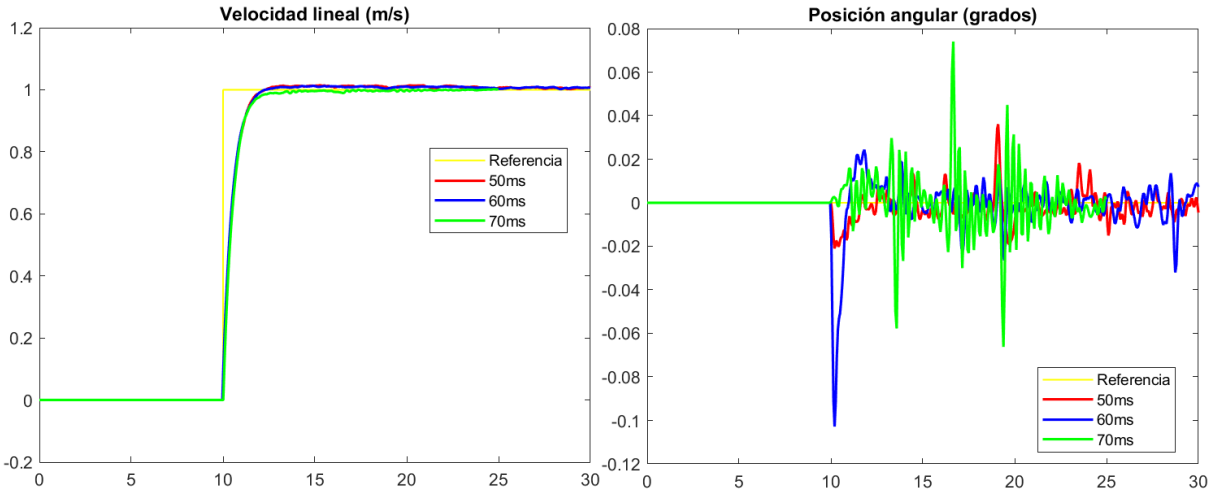

Ilustración 24. Control PID experimentación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

Las gráficas anteriores se han obtenido gracias al script “PID_SoMachine.mlx”, mostrado en el Anexo I [Script 36], desarrollado en MATLAB. Para cada tipo de control se ha implementado un *script* idéntico a éste, por lo que no serán mencionados en los posteriores capítulos.

En la comparativa entre la fase de simulación (línea roja, MATLAB), las pruebas intermedias (línea azul, PYTHON) y la fase experimental (línea verde, RPI) se aprecia una gran similitud en las acciones de control [Ilustración 25] y las variables controladas [Ilustración 26] para Ts = 60 ms, dándose quizás en menor medida en la posición angular debido a su habitual carácter inestable.

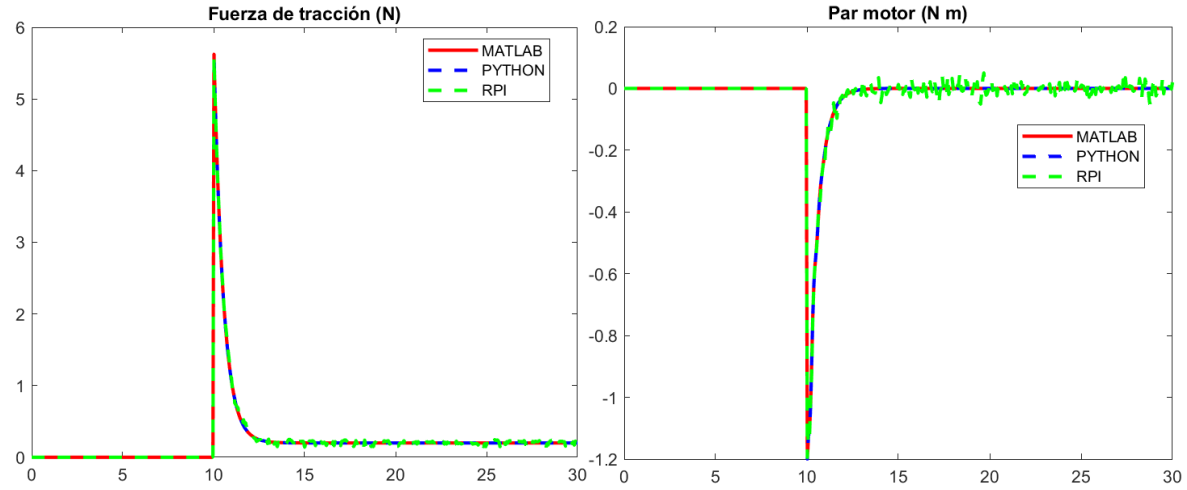

Ilustración 25. Control PID – comparativa acc. Control entre simulación y experimentación (Elaboración propia)

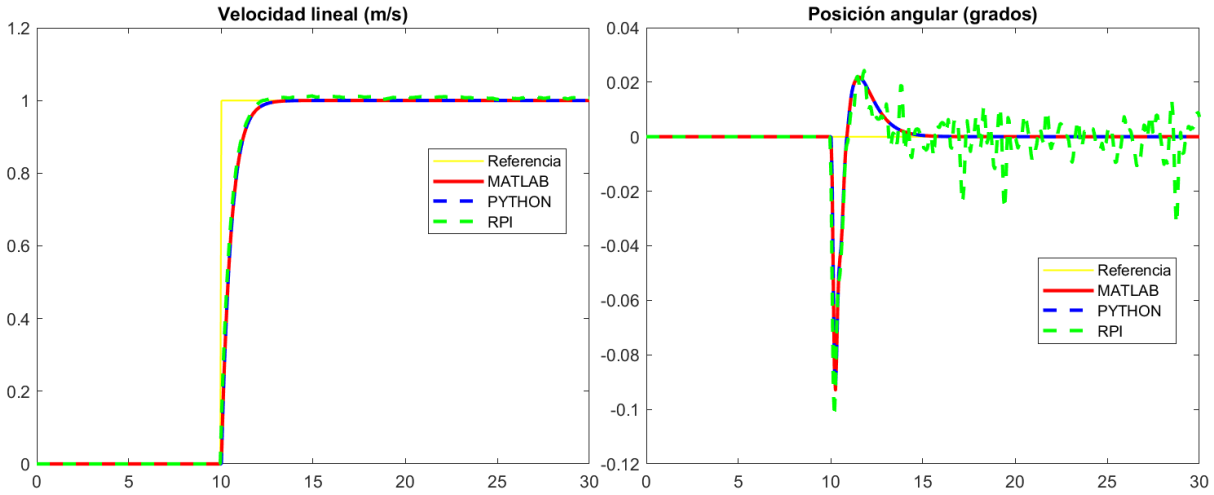

Ilustración 26. Control PID – comparativa var. Ctr entre simulación y experimentación (Elaboración propia)

En conclusión, la experimentación mediante controladores PID ha resultado exitosa, ya que la oscilación del péndulo respecto a los 0° marcados como referencia es mínima y la velocidad lineal requerida se alcanza en cuestión de pocos segundos.

Capítulo 4

Control del Segway mediante observadores de estado y efecto integral

El control basado en controladores PID ha sido capaz de alcanzar grandes prestaciones en el Segway con relativa facilidad. Sin embargo, puede que en la práctica no sea la solución óptima, por lo que se hace necesario conocer otras técnicas de control. Así, en este capítulo se van a visualizar técnicas basadas en realimentación del estado, observadores de estado y efecto integral.

Gran parte del contenido mostrado a continuación se ha basado en la teoría de observabilidad y controlabilidad [19], realimentación del estado [20], observadores [21], control óptimo [22] y filtro de Kalman [23] de la asignatura Ingeniería de Control.

4.1. Observador de orden reducido + Prealimentación de la referencia

4.1.1. Desarrollo teórico y aplicación sobre el Segway

Al contrario que el control basado en PIDs, en este tipo de técnicas no se calcula el error de posición utilizando el valor real de la salida del proceso, sino una estimación obtenida a partir del uso de observadores y realimentaciones. Por ello, es necesario profundizar en estos conceptos.

El Segway se compone de dos salidas que a su vez son las dos primeras variables de estado. Sin embargo, no es posible obtener directamente del modelo del proceso el valor de la tercera variable de estado, la velocidad angular, de manera que se conozca el vector de estados al completo y pueda realimentarse mediante la matriz de realimentación. Por este motivo, es necesario incluir un paso intermedio en el que un observador de orden reducido estime un valor del tercer estado.

Previamente al desarrollo de esta técnica y todas las de este capítulo, se ha comprobado que el modelo lineal es totalmente controlable, es decir, que, para todos los estados, partiendo de un estado inicial x_0 el sistema puede ser llevado a un estado final x_F en un tiempo determinado mediante una secuencia de acciones de control [19]. Para ello, se calcula su matriz de controlabilidad y se comprueba que su rango es igual al número de estados.

También se ha comprobado que es totalmente observable, es decir, que, para todos los estados, a partir de una serie de medidas de las salidas a lo largo del tiempo se puede determinar el estado del sistema en el instante inicial [19]. Para ello, se calcula su matriz de observabilidad y se comprueba que su rango es igual al número de estados.

Por otra parte, al igual que se hizo en el control PID, es necesario discretizar todos los espacios de estados, matrices y vectores utilizados con el fin de poder ser expresados en lenguaje ST.

4.1.1.1. Observador de estado de orden reducido

Un observador es un sistema dinámico, expresado en forma de espacio de estados, que proporciona una estimación de una o más variables de estado del sistema a partir de sus entradas y salidas, y uno de orden reducido estima únicamente aquellos estados que no forman parte de la salida. En el caso del Segway, sólo debe estimarse el valor del tercer estado, por lo que la subdivisión de su modelo lineal en espacio de estados discretizado es:

$$SYS_d = [A_d, B_d, C_d, D_d]$$

$$A_d = \left[\begin{array}{cc|c} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ \hline A_{31} & A_{32} & A_{33} \end{array}\right] = \begin{bmatrix} A_{11_d} & A_{12_d} \\ A_{21_d} & A_{22_d} \end{bmatrix}$$

$$B_d = \left[\begin{array}{cc} B_{11} & B_{12} \\ B_{21} & B_{22} \\ \hline B_{31} & B_{32} \end{array}\right] = \begin{bmatrix} B_{1_d} \\ B_{2_d} \end{bmatrix}$$

Por otra parte, es necesario fijar la dinámica del observador mediante asignación de polos estables de forma que sea más rápida que la del sistema observado, y para ello se crea la matriz L_{ObRed} mediante el comando 'place', que hace uso de las matrices $A12_d$ y $A22_d$ relacionadas con el estado estimado. Una vez creada, es posible diseñar las matrices del espacio de estados discreto del observador, el cual recibe como entradas las acciones de control y las variables controladas y devuelve como salida la estimación del estado:

$$L_{ObRed} = place(A_{22_d}^T, A_{12_d}^T, polos_{ObRed})$$

$$A_{ObRed} = A_{22_d} - L_{ObRed} \cdot A_{12_d}$$

$$B_{ObRed} = \begin{bmatrix} B_{2_d} - L_{ObRed} \cdot B_{1_d} & A_{21_d} - L_{ObRed} \cdot A_{11_d} + \left(A_{22_d} - L_{ObRed} \cdot A_{12_d}\right) \cdot L_{ObRed} \end{bmatrix}$$

$$C_{ObRed} = I_{numX-numY} = 1$$

$$D_{ObRed} = [0_{numX-numY,numU} \quad L_{ObRed}] = [0_{1,2} \quad L_{ObRed}]$$

$$\begin{cases} x_{k+1} = A_{ObRed} \cdot x_k + B_{ObRed} \cdot u_k \\ y_k = Y_{ObRed} = C_{ObRed} \cdot x_k + D_{ObRed} \cdot u\_k \end{cases}$$

4.1.1.2. Realimentación del estado observado

Tras obtener la salida del observador de orden reducido, se conoce al completo el vector de estados y es posible realizar un control por realimentación del estado que sea capaz de asignar todos los polos del sistema. Con ese fin, se utiliza el comando 'place' para el diseño de la matriz K_{Realim} mediante asignación de polos estables, los cuales deben ser preferiblemente más lentos que los del observador si se desea obtener un funcionamiento correcto, y se construye el vector Y_{Realim} realimentado al inicio del control para el cálculo de las acciones de control:

$$U_{Realim} = \begin{bmatrix} Y_1 \\ Y_2 \\ Y_{ObRed} \end{bmatrix}$$

$$K_{Realim} = place(A_d, B_d, polos_{Realim})$$

$$Y_{Realim} = K_{Realim} \cdot U_{Realim}$$

Así, el sistema global está formado por una dinámica del proceso fijada por K_{Realim} y una dinámica del

observador fijada por L_{ObRed}.

4.1.1.3. Prealimentación de la referencia

La realimentación del estado asegura la estabilidad de la salida, pero no el seguimiento de las referencias. Por ello, se aplica una prealimentación de la referencia que establece una ganancia unitaria del sistema formado por el modelo lineal del Segway y la matriz K_{Realim} en bucle cerrado, la cual se calcula como la matriz pseudoinversa de la matriz de ganancia estática de dicho sistema:

$$SYS_{bc_d} = [A_{bc}, B_{bc}, C_{bc}, D_{bc}] = [A_d - B_d \cdot K_{Realim}, B_d, C_d - D_d \cdot K_{Realim}, D_d]$$

$$G_{bc_d} = tf(SYS_{bc_d}) \rightarrow G_{est} = G_{bc_d}(z = 1)$$

$$F_{Prealim} = G_{est}^{\#} = (G_{est}^T \cdot G_{est})^{-1} \cdot G_{est}^T$$

De esta forma, ya pueden calcularse las acciones de control como la diferencia entre la referencia prealimentada y el vector Y_{Realim}.

4.1.2. Fase de simulación en MATLAB

4.1.2.1. Controlador en MATLAB

El modelo lineal del Segway presenta un polo inestable en s = 3.1763, un polo en s = -0.0645 que puede considerarse un integrador y un polo estable en s = -3.1918, el cual se traduce en un tiempo de establecimiento de 1.2532 segundos, calculado como $te = 4/\sigma$, por lo que se busca una dinámica del observador ligeramente más rápida. Por otro lado, la dinámica del proceso se ha escogido pensando en un comportamiento realista [Tabla 3]:

Tabla 3. Ob.Red+Prealim.Ref: Tiempos de establecimiento (Elaboración propia)

TIEMPOS DE ESTABLECIMIENTO DEL OBSERVADOR Y DE LA MATRIZ DE REALIMENTACIÓN				
	$T_s = 50\ ms$	$T_s = 60\ ms$	$T_s = 80\ ms$	$Ts = 100\ ms$
te_{ObRed}	0.5	1	1	1
te_{Realim}	2.5	2.5	2.5	2.5

El control en simulación se ha desarrollado en el fichero “Segway_ModeloNL_obRed.slx” [Ilustración 27] de Simulink donde se aplican todos los conceptos mostrados anteriormente, los cuales han sido programados previamente en el *script* “param_Segway_obRed.mlx” [Script 6] de MATLAB.

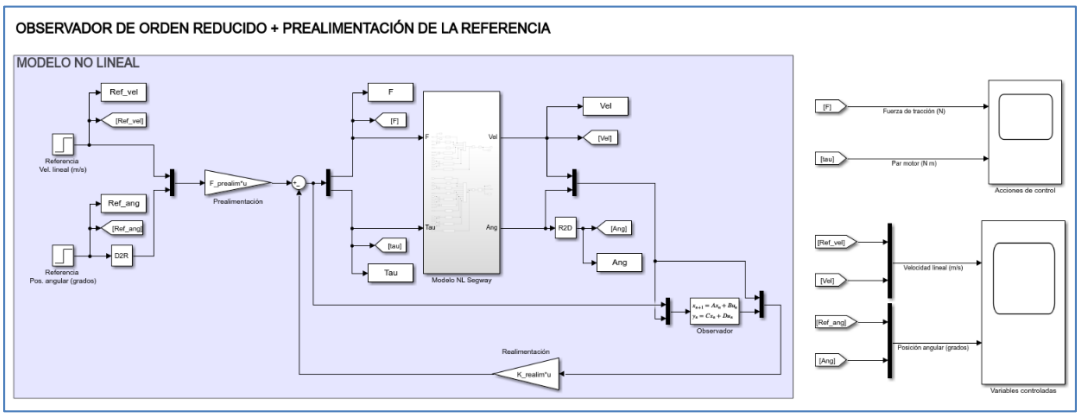

Ilustración 27. Fichero “Segway_ModeloNL_obRed.slx” (Elaboración propia)

1. Parámetros y simulación en MATLAB

```
…
load ModeloL_Segway.mat;
SYS = ModeloL_Segway;
pole(SYS); % polo en -3.1918 -> te = 1.2532 seg
rank(ctrb(SYS)); % = 3 = núm. estados -> Controlable
rank(obsv(SYS)); % = 3 = núm. estados -> Observable
…
if graf_MAT ~= "SI"
    Ts = input("Elija un período de muestreo: ");
    switch Ts
        case 0.05
            te_obred = 0.5; te_realim = 2.5;
        case 0.06
            te_obred = 1; te_realim = 2.5;
        case 0.08
            te_obred = 1; te_realim = 2.5;
        case 0.1
            te_obred = 1; te_realim = 2.5;
        otherwise
            disp("Período de muestreo no disponible");
    end

    SYSd = c2d(SYS,Ts,'zoh');
    [Adisc,Bdisc,Cdisc,Ddisc] = ssdata(SYSd);
    n = size(Adisc,1);
    m_ent = size(Bdisc,2);
    p = size(Cdisc,1);

    A11 = Adisc(1:2,1:2);
    A12 = Adisc(1:2,3);
    A21 = Adisc(3,1:2);
    A22 = Adisc(3,3);
    B1 = Bdisc(1:2,:);
    B2 = Bdisc(3,:);

    sigma_obred = 4/te_obred;
    p_obred = exp(-sigma_obred*Ts);
    L_obred = place(A22', A12', p_obred)';
    A_obred = A22-L_obred*A12;
    B_obred = [B2-L_obred*B1 A21-L_obred*A11+(A22-L_obred*A12)*L_obred];
    C_obred = eye(n-p,n-p);
    D_obred = [zeros(n-p,m_ent) L_obred];
    SYS_obred = ss(A_obred, B_obred, C_obred, D_obred, Ts);

    sigma_realim = 4/te_realim;
    p_realim = exp(-sigma_realim*Ts)*[1 1.01 1.001];
    K_realim = place(Adisc,Bdisc,p_realim);
    F_prealim = pinv(dcgain(ss(Adisc-Bdisc*K_realim,Bdisc,Cdisc-Ddisc*K_realim,Ddisc,Ts)));

    sim("Segway_ModeloNL_obRed.slx");
    …
end
```

2. Resultados

```
…
```

Script 6. Script “param_Segway_obRed.mlx”

4.1.2.2. Resultados

Este control ofrece resultados muy satisfactorios de acciones de control [Ilustración 28] y variables controladas [Ilustración 29] para las cuatro simulaciones realizadas, ya que se alcanza la velocidad lineal exigida en los 2.5 segundos impuestos en la realimentación y sin ningún tipo de sobreoscilación, y el balanceo del péndulo apenas sufre una ligera inclinación de 0.35°.

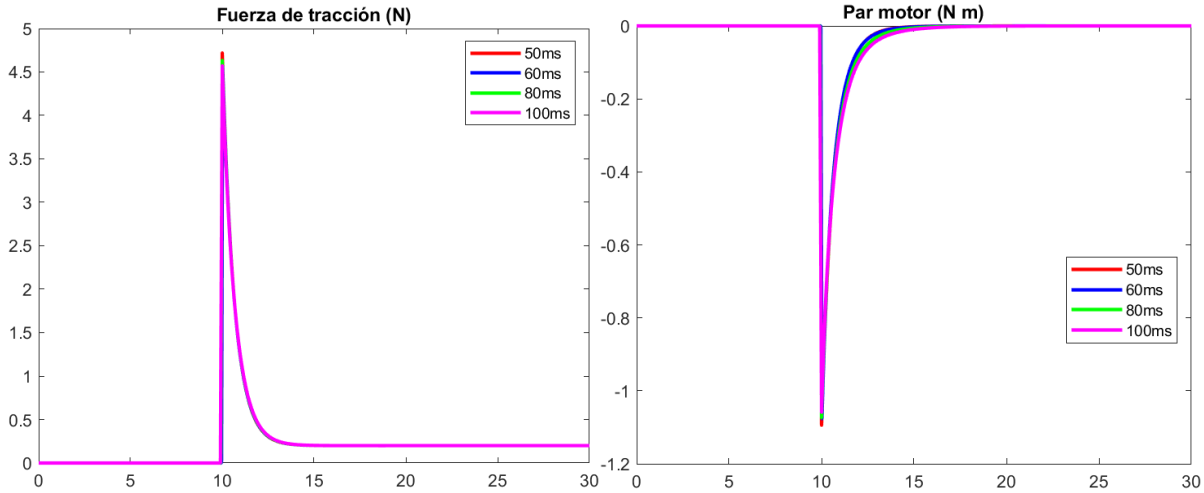

Ilustración 28. Ob.Red+Prealim.Ref simulación – Acc. Control: F. tracción y par motor (Elaboración propia)

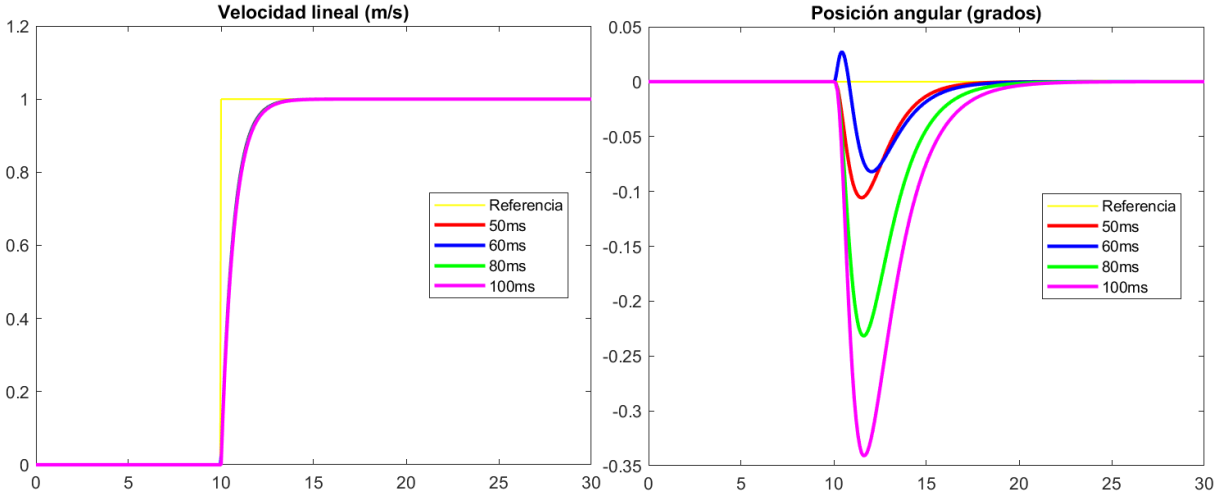

Ilustración 29. Ob.Red+Prealim.Ref simulación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

4.1.3. Fase experimental en Raspberry Pi y SoMachine

En la fase experimental de este control, y en la de todos los demás, existen los problemas de tiempos de computación y errores de cuantificación. Sin embargo, no evita que los resultados sean verdaderamente buenos.

Para la programación del controlador, se ha hecho uso de la herramienta PLC Coder de MATLAB/Simulink.

4.1.3.1. Controlador en SoMachine

Se ha desarrollado un proyecto "Segway_SoMachine_obRed_80ms.project" [Script 7] en SoMachine en el que se ha programado el control en lenguaje ST siguiendo el mismo procedimiento que en la fase de simulación. Como puede apreciarse, se ha omitido gran parte del contenido, y esto es debido a que es idéntico al del control PID. Por ello, únicamente se muestran aquellos métodos novedosos del controlador.

```
// DECLARACIÓN DE VARIABLES
PROGRAM POU
VAR
	…
	Ref_pre: ARRAY[0..1] OF LREAL;
	U_realim: ARRAY[0..2] OF LREAL;
	Y_realim: ARRAY[0..1] OF LREAL;
	Observador_DSTATE: LREAL := 0.0;
	rtb_Observador: LREAL := 0.0;
	Y_obRed_k: LREAL := 0.0;
	A_ob: LREAL := 0.72614903707369083;
	B_ob: ARRAY[0..3] OF LREAL := [-0.011625615191541681, -0.089213770680626261, 0.0984867846285391, -
0.39864877663281684];
	D_ob: ARRAY[0..3] OF LREAL := [0.0, 0.0, -0.35114596845916046, 3.7567276401137546];
	Krealim: ARRAY[0..5] OF LREAL := [4.43543970733464,-1.07380528693826,1.1038823035669809,-
8.76785075403287,1.7572984029142633,-2.6259234461674907];
	Fprealim: ARRAY[0..3] OF LREAL := [4.6354397073350562, 1.1038823032020464, -1.0738052869383341, -
1.7046507540658264];
END_VAR
…
// ARRANQUE DEL CONTROLADOR
	…
ELSIF manual = FALSE THEN
	…
	// Prealimentación de la referencia
	Ref_pre[0] := (Fprealim[0]*Ref[0]) + (Fprealim[1]*Ref[1]);
	Ref_pre[1] := (Fprealim[2]*Ref[0]) + (Fprealim[3]*Ref[1]);
	// Observador de orden reducido
	rtb_Observador := D_ob[2]*Y_k[0] + Observador_DSTATE;
	Observador_DSTATE := A_ob*Observador_DSTATE + B_ob[0]*U_k[0] + B_ob[1]*U_k[1] +
B_ob[2]*Y_k[0] + B_ob[3]*Y_k[1];
	Y_obRed_k := D_ob[3]*Y_k[1] + rtb_Observador;
	// Realimentación del estado y de las dos salidas
	U_realim[0] := Y_k[0];
	U_realim[1] := Y_k[1];
	U_realim[2] := Y_obRed_k;
	Y_realim[0] := (Krealim[4]*U_realim[2]) + (Krealim[2]*U_realim[1]) +
(Krealim[0]*U_realim[0]);
	Y_realim[1] := (Krealim[5]*U_realim[2]) + (Krealim[3]*U_realim[1]) +
(Krealim[1]*U_realim[0]);
	// Cálculo de las acciones de control
	U_k[0] := Ref_pre[0] - Y_realim[0];
	U_k[1] := Ref_pre[1] - Y_realim[1];
	…
END_IF
```

Script 7. Proyecto "Segway_SoMachine_obRed_80ms.project"

4.1.3.2. Resultados

Los resultados experimentales en acciones de control [Ilustración 30] y variables controladas [Ilustración 31] son favorables, siéndolo quizás menos en la posición angular debido a cierto error de posición existente respecto a su referencia, suceso que se da para todos los períodos de muestreo.

En la comparativa entre simulación, fase intermedia y experimentación para Ts = 80 ms [Ilustración 32] [Ilustración 33], escogido por ser el que muestra un menor error de posición en el ángulo del péndulo, la fuerza de tracción se muestra prácticamente igual pero no ocurre así con el par motor, y este es el motivo principal por el que la velocidad lineal es tan precisa y la inclinación del péndulo se ve ligeramente incrementada hasta 1.2°.

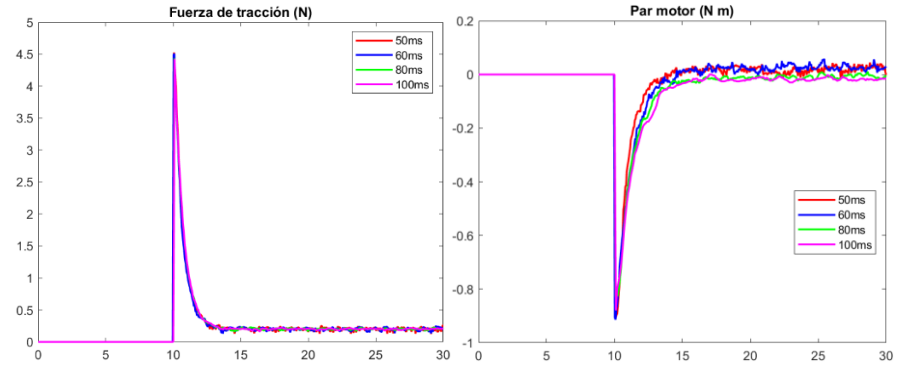

Ilustración 30. Ob.Red+Prealim.Ref experimentación – Acc. Control: F. tracción y par motor (Elaboración propia)

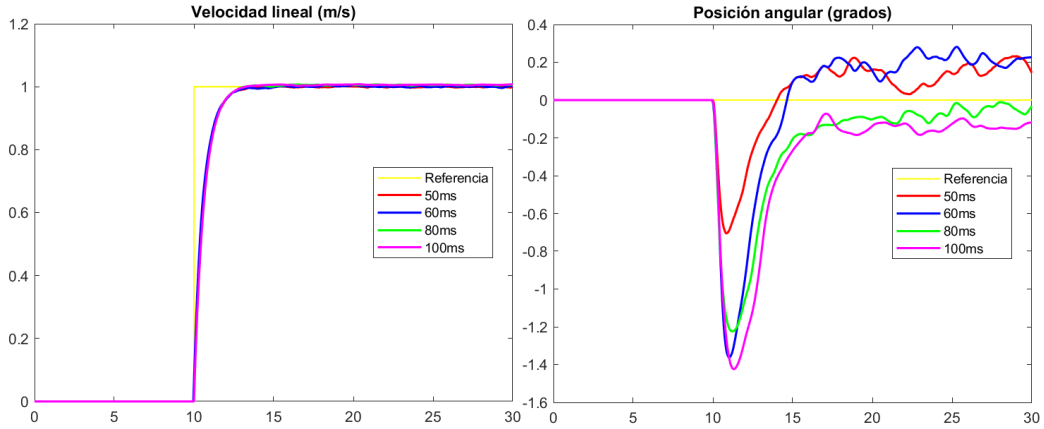

Ilustración 31. Ob.Red+Prealim.Ref experimentación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

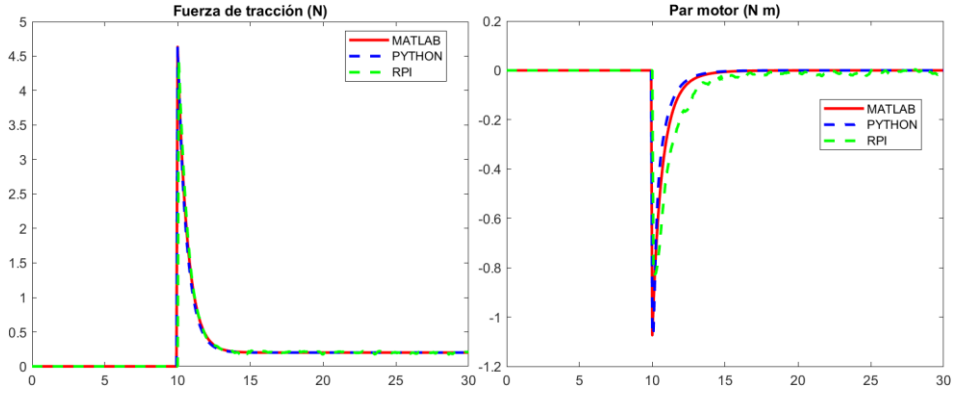

Ilustración 32. Ob.Red+Prealim.Ref - comparativa acc. Control entre simulación y experimentación (Elaboración propia)

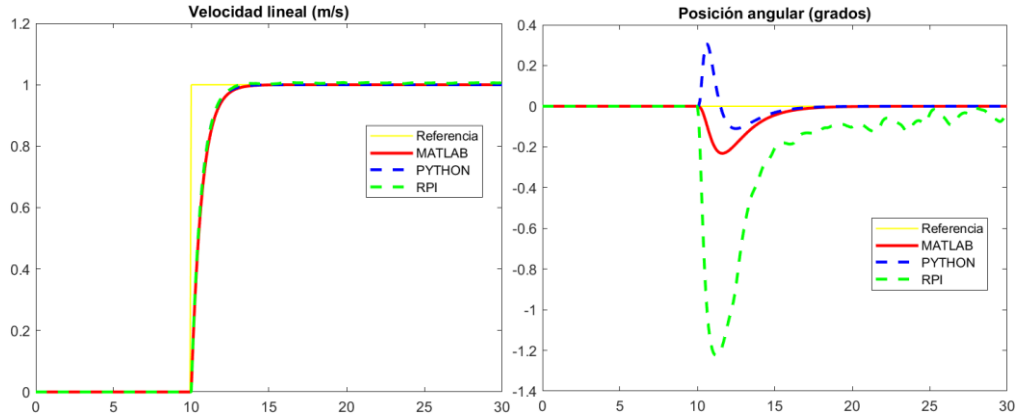

Ilustración 33. Ob.Red+Prealim.Ref - comparativa var. Ctr entre simulación y experimentación (Elaboración propia)

4.2. Observador de orden completo + Prealimentación de la referencia

4.2.1. Desarrollo teórico y aplicación sobre el Segway

Este control también se basa en los principios de realimentación del estado y de prealimentación de la referencia. Sin embargo, el vector U_{Realim} ya no se conforma utilizando las salidas del modelo y una estimación de la tercera variable de estado, sino una estimación del estado completo obtenida a partir de un observador de orden completo.

4.2.1.1. Observador de estado de orden completo

Para diseñar este observador, primero es necesario fijar su dinámica mediante asignación de polos estables, debiendo ser más rápida que la del sistema observado, y para ello se crea la matriz L_{ObCom} mediante el comando 'place' y las matrices A_d y C_d del modelo. Ya con esta matriz calculada, se diseña su espacio de estados discreto, el cual recibe como entradas las acciones de control y las variables controladas y devuelve como salida el vector de estados estimados:

$$L_{ObCom} = place(A_d^T, C_d^T, polos_{ObCom})$$

$$A_{ObCom} = A_d - L_{ObCom} \cdot C_d$$

$$B_{ObCom} = [B_d - L_{ObCom} \cdot D_d \quad L_{ObCom}]$$

$$C_{ObCom} = I_{numX} = I_{3x3}$$

$$D_{ObCom} = [0_{numX,numU} \quad 0_{numX,numY}] = [0_{3,2} \quad 0_{3,2}]$$

$$\begin{cases} x_{k+1} = A_{ObCom} \cdot x_k + B_{ObCom} \cdot u_k \\ y_k = Y_{ObCom} = C_{ObCom} \cdot x_k + D_{ObCom} \cdot u\_k \end{cases}$$

El cálculo de la prealimentación de la referencia y de la realimentación del estado es idéntico al anterior control, con la salvedad de que ahora U_{Realim} es:

$$U_{Realim} = \begin{bmatrix} Y_{ObCom_1} \\ Y_{ObCom_2} \\ Y_{ObCom_3} \end{bmatrix}$$

4.2.2. Fase de simulación en MATLAB

4.2.2.1. Controlador en MATLAB

Al igual que en el anterior control, se opta por tiempos de establecimiento [Tabla 4] del observador más pequeños que el del propio modelo con el fin de obtener su estabilización, y se escogen los de la realimentación según el comportamiento apreciado gráficamente en las salidas:

Tabla 4. Ob.Com+Prealim.Ref: Tiempos de establecimiento (Elaboración propia)

TIEMPOS DE ESTABLECIMIENTO DEL OBSERVADOR Y DE LA MATRIZ DE REALIMENTACIÓN			
	$T_s = 50\ ms$	$T_s = 60\ ms$	$T_s = 70\ ms$
$te_{Ob.Com}$	0.5	0.5	0.5
te_{Realim}	5.5	4	4

El control en simulación se ha desarrollado en el fichero "Segway_ModeloNL_obCom.slx" [Ilustración

34] de Simulink donde se aplican todos los conceptos mostrados anteriormente, los cuales han sido programados previamente en el script “param_Segway_obCom.mlx” [Script 8] de MATLAB.

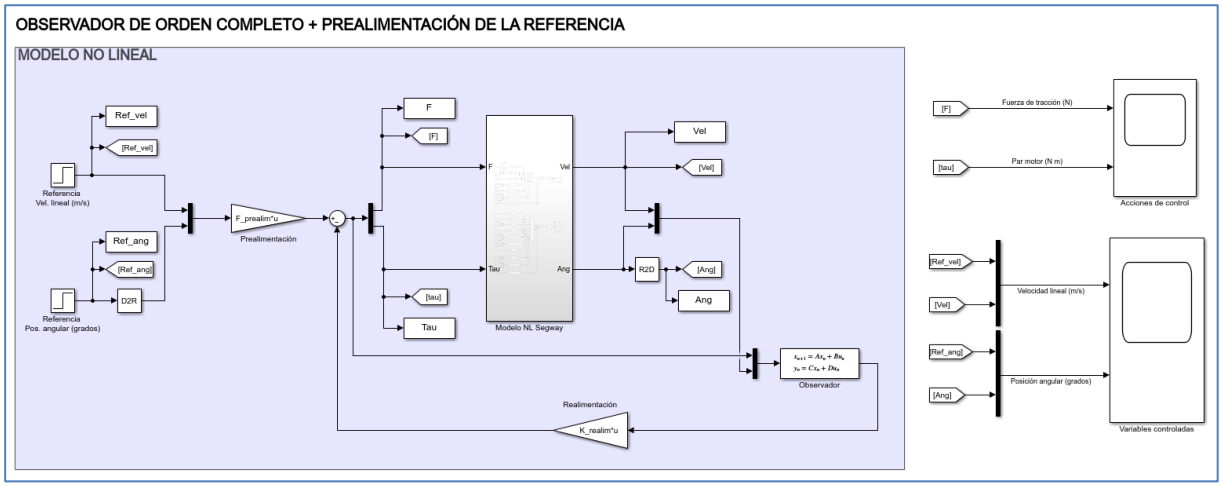

Ilustración 34. Fichero “Segway_ModeloNL_obCom.slx” (Elaboración propia)

1. Parámetros y simulación en MATLAB

```
…
if graf_MAT ~= "SI"
    Ts = input("Elija un período de muestreo: ");
    switch Ts
        case 0.05
            escalon = 10; tiempo = 30; te_obcom = 0.5; te_realim = 5.5;
        …
    end

    SYSd = c2d(SYS,Ts,'zoh');
    [Adisc,Bdisc,Cdisc,Ddisc] = ssdata(SYSd);
    n = size(Adisc,1);
    m_ent = size(Bdisc,2);
    p = size(Cdisc,1);

    sigma_obcom = 4/te_obcom;
    sigma_realim = 4/te_realim;

    p_obcom = exp(-sigma_obcom*Ts)*[1 1.01 1.001];
    p_realim = exp(-sigma_realim*Ts)*[1 1.01 1.001];

    L_obcom = place(Adisc', Cdisc', p_obcom)';
    A_obcom = Adisc-L_obcom*Cdisc;
    B_obcom = [Bdisc-L_obcom*Ddisc L_obcom];
    C_obcom = eye(n,n);
    D_obcom = [zeros(n,m_ent) zeros(n,p)];
    SYS_obcom = ss(A_obcom, B_obcom, C_obcom, D_obcom, Ts);

    K_realim = place(Adisc,Bdisc,p_realim);
    F_prealim = pinv(dcgain(ss(Adisc-Bdisc*K_realim,Bdisc,Cdisc-Ddisc*K_realim,Ddisc,Ts)));

    sim("Segway_ModeloNL_obCom.slx");
    …
end
```

2. Resultados

```
…
```

Script 8. Script “param_Segway_obCom.mlx”

4.2.2.2. Resultados

Este control ofrece resultados muy satisfactorios de acciones de control [Ilustración 35] y variables

controladas [Ilustración 36] para las tres simulaciones realizadas, ya que se alcanza la velocidad lineal exigida en los 5.5/4 segundos impuestos en la realimentación y sin ningún tipo de sobreoscilación, y el balanceo del péndulo apenas sufre una ligera inclinación máxima de 0.3°.

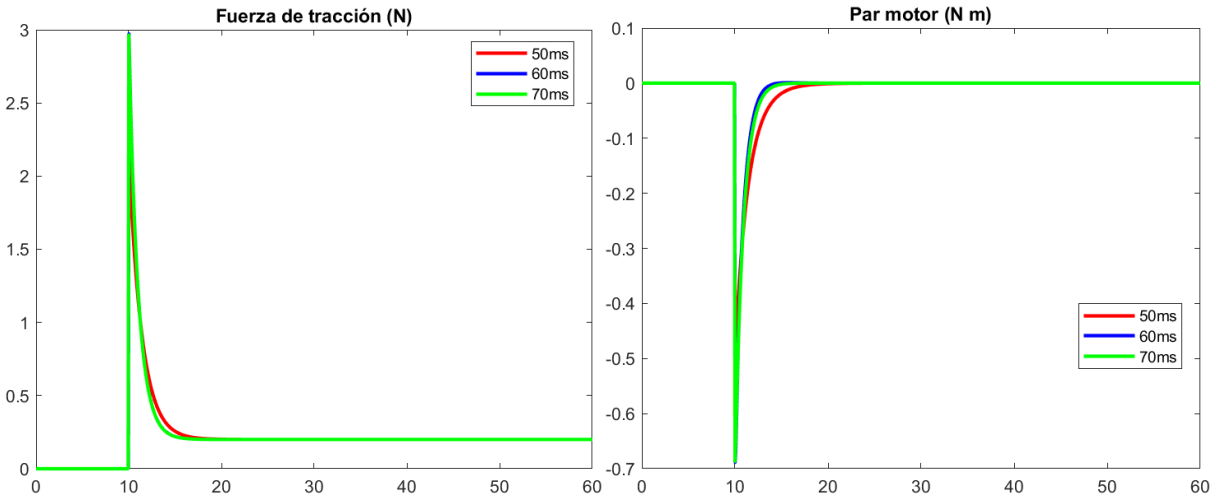

Ilustración 35. Ob.Com+Prealim.Ref simulación – Acc. Control: F. tracción y par motor (Elaboración propia)

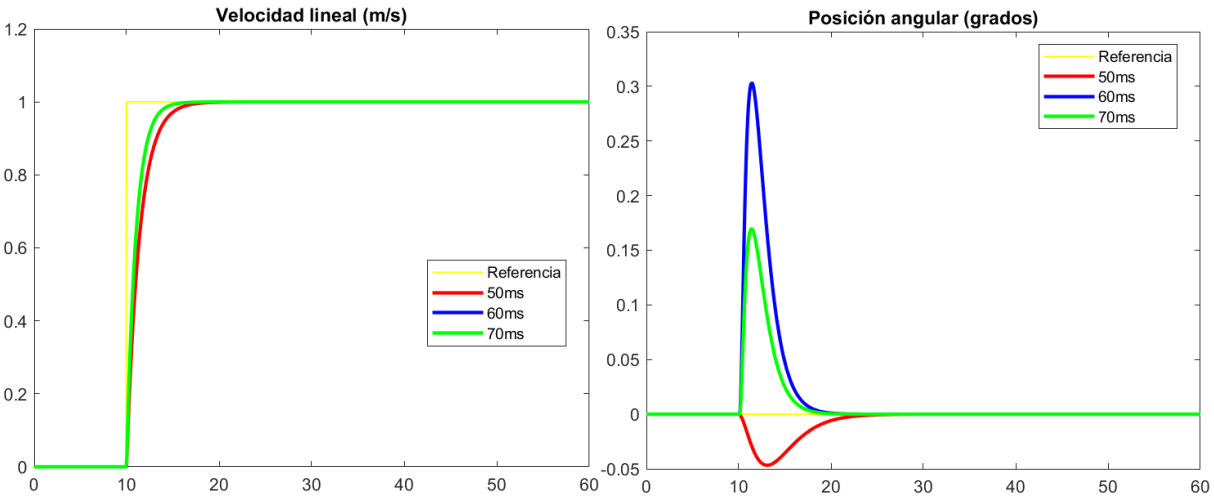

Ilustración 36. Ob.Com+Prealim.Ref simulación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

4.2.3. Fase experimental en Raspberry Pi y SoMachine

Para la programación del controlador, se ha hecho uso de la herramienta PLC Coder de MATLAB/Simulink.

4.2.3.1. Controlador en SoMachine

Se ha desarrollado un proyecto “Segway_SoMachine_obCom_70ms.project” [Script 9] en SoMachine en el que se ha programado el control en lenguaje ST siguiendo el mismo procedimiento que en la fase de simulación. Como puede apreciarse, se ha omitido gran parte del contenido por ser idéntico al del control PID, mostrando únicamente aquellos métodos novedosos del controlador.

```
// DECLARACIÓN DE VARIABLES
PROGRAM POU
VAR
	…
	A_ob: ARRAY[0..8] OF LREAL :=
[0.56470853063138937,0.0814757917792133,0.52169355550038743,0.036135099617365413,0.13027234360698525,-
2.8347153704630896,-0.0057814117392621447,0.070580782055544813,1.0249296170104332];
	B_ob: ARRAY[0..11] OF LREAL := [0.0279665964599455,-0.00081852584376027678,-
0.023461445284347411,0.023461445284337655,-0.0035295074485853766,-0.10124192437531822,0.42969815007661161,-
0.081312086610461159,-0.51700126644351563,-0.20184797993871992,0.89465727340344792,3.5498073306989379];
	C_ob: ARRAY[0..8] OF LREAL := [1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0];
	Krealim: ARRAY[0..5] OF LREAL := [2.76473317385882,-0.68676075133441894,0.300862878776913,-
7.690168862621058,0.97156379955094385,-1.6941439382164163];
	Fprealim: ARRAY[0..3] OF LREAL := [2.9647331738592366, 0.300862878411983, -0.68676075133449366, -
0.62696886265402174];END_VAR
…
// ARRANQUE DEL CONTROLADOR
	…
ELSIF manual = FALSE THEN
	…
	// Prealimentación de la referencia
	Ref_pre[0] := (Fprealim[0]*Ref[0]) + (Fprealim[1]*Ref[1]);
	Ref_pre[1] := (Fprealim[2]*Ref[0]) + (Fprealim[3]*Ref[1]);
	// Observador de orden completo
	rtb_Observador[0] := (C_ob[6]*Observador_DSTATE[2]) + ((C_ob[3]*Observador_DSTATE[1]) +
(C_ob[0]*Observador_DSTATE[0]));
	xnew[0] := (B_ob[9]*Y_k[1]) + ((B_ob[6]*Y_k[0]) + ((B_ob[3]*U_k[1]) + ((B_ob[0]*U_k[0]) +
((A_ob[6]*Observador_DSTATE[2]) + ((A_ob[3]*Observador_DSTATE[1]) + (A_ob[0]*Observador_DSTATE[0])))));
	rtb_Observador[1] := (C_ob[7]*Observador_DSTATE[2]) + ((C_ob[4]*Observador_DSTATE[1]) +
(C_ob[1]*Observador_DSTATE[0]));
	xnew[1] := (B_ob[10]*Y_k[1]) + ((B_ob[7]*Y_k[0]) + ((B_ob[4]*U_k[1]) + ((B_ob[1]*U_k[0]) +
((A_ob[7]*Observador_DSTATE[2]) + ((A_ob[4]*Observador_DSTATE[1]) + (A_ob[1]*Observador_DSTATE[0])))));
	rtb_Observador[2] := (C_ob[8]*Observador_DSTATE[2]) + ((C_ob[5]*Observador_DSTATE[1]) +
(C_ob[2]*Observador_DSTATE[0]));
	xnew[2] := (B_ob[11]*Y_k[1]) + ((B_ob[8]*Y_k[0]) + ((B_ob[5]*U_k[1]) + ((B_ob[2]*U_k[0]) +
((A_ob[8]*Observador_DSTATE[2]) + ((A_ob[5]*Observador_DSTATE[1]) + (A_ob[2]*Observador_DSTATE[0])))));
	Observador_DSTATE[0] := xnew[0];
	Observador_DSTATE[1] := xnew[1];
	Observador_DSTATE[2] := xnew[2];
	Y_obCom_k[0] := rtb_Observador[0];
	Y_obCom_k[1] := rtb_Observador[1];
	Y_obCom_k[2] := rtb_Observador[2];
	// Realimentación del estado y de las dos salidas
	U_realim[0] := Y_obCom_k[0];
	U_realim[1] := Y_obCom_k[1];
	U_realim[2] := Y_obCom_k[2];
	Y_realim[0] := Krealim[4]*U_realim[2] + Krealim[2]*U_realim[1] + Krealim[0]*U_realim[0];
	Y_realim[1] := Krealim[5]*U_realim[2] + Krealim[3]*U_realim[1] + Krealim[1]*U_realim[0];
	// Cálculo de las acciones de control
	U_k[0] := Ref_pre[0] - Y_realim[0];
	U_k[1] := Ref_pre[1] - Y_realim[1];
	…
END_IF
```

Script 9. Proyecto "Segway_SoMachine_obCom_70ms.project"

4.2.3.2. Resultados

Los resultados experimentales en acciones de control [Ilustración 37] y variables controladas [Ilustración 38] son favorables, siéndolo quizás menos en la posición angular debido a cierto error de posición y a la lentitud con la que alcanza la referencia, dándose para todos los períodos de muestreo.

En la comparativa entre simulación, fase intermedia y experimentación para Ts = 70 ms [Ilustración 39] [Ilustración 40], escogido por ser el que muestra un menor error de posición en el ángulo del péndulo, la fuerza de tracción se muestra prácticamente igual pero no ocurre así con el par motor, y este es el motivo principal por el que la velocidad lineal es tan precisa y la inclinación del péndulo se ve claramente incrementada hasta 3.5°.

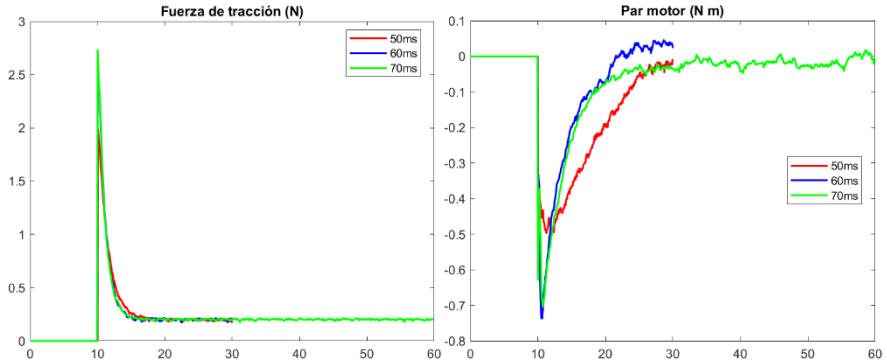

Ilustración 37. Ob.Com+Prealim.Ref experimentación – Acc. Control: F. tracción y par motor (Elaboración propia)

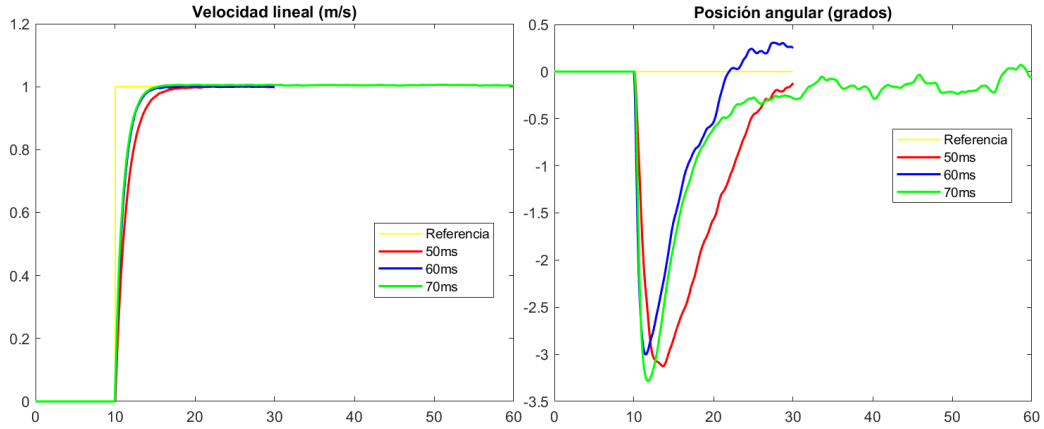

Ilustración 38. Ob.Com+Prealim.Ref experimentación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

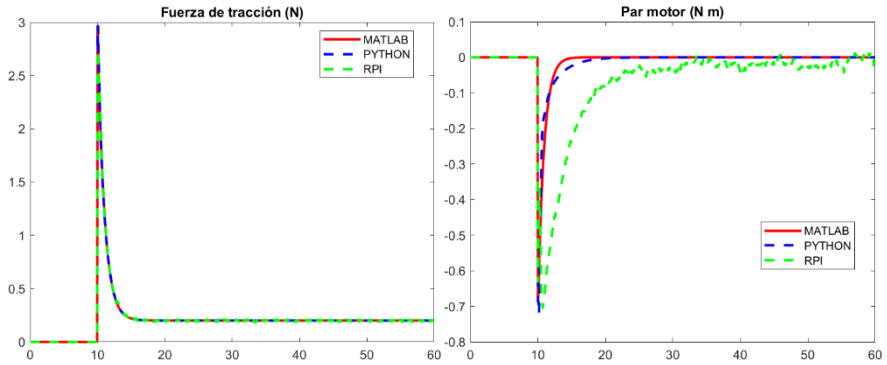

Ilustración 39. Ob.Com+Prealim.Ref – comparativa acc. Control entre simulación y experimentación (Elaboración propia)

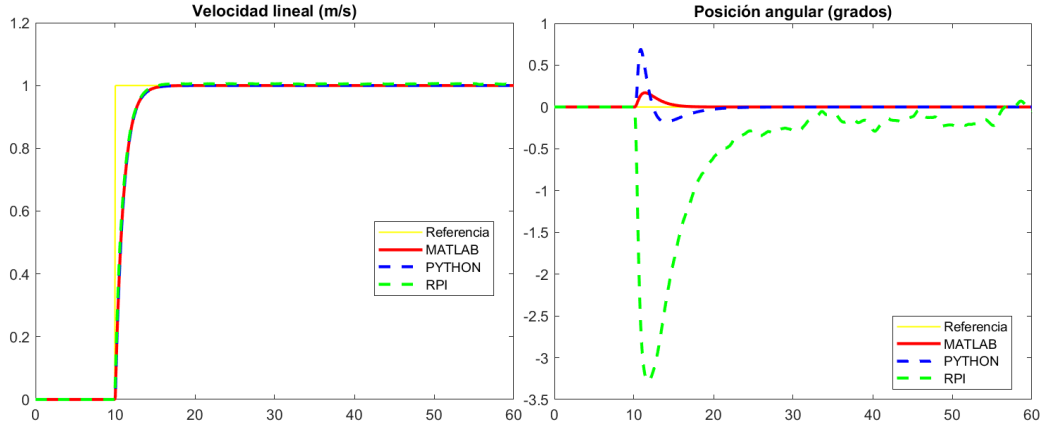

Ilustración 40. Ob.Com+Prealim.Ref – comparativa var. Ctr entre simulación y experimentación (Elaboración propia)

4.3. Observador de orden reducido + Control Integral

4.3.1. Desarrollo teórico y aplicación sobre el Segway

Este control también se basa en los principios de realimentación del estado y de estimación de la tercera variable de estado mediante un observador de orden reducido, tal y como se hace en el apartado 4.1. Sin embargo, ya no se aplica la prealimentación sobre las referencias, sino un control integral que hace necesaria una ampliación del vector U_{Realim} de entradas de la realimentación.

4.3.1.1. Control integral

Como se ha mencionado anteriormente, el control por realimentación del estado permite fijar la dinámica deseada al sistema, pero no asegura el seguimiento de referencias. Por ello, en este control se opta por utilizar el efecto integral sobre el error del sistema, es decir, la diferencia entre las referencias y las salidas. Este efecto hace que las salidas sigan a sus referencias en régimen permanente y compensa las perturbaciones constantes, en el caso de que existiesen. Para aplicar este efecto, sólo es necesario aplicar un integrador discretizado sobre el error mencionado, obteniéndose la salida:

$$E_k = Ref_k - Y_k \quad \rightarrow \quad Y_{CtrInt_k} = Y_{CtrInt_k-1} + E_{k-1}$$

Por otra parte, el vector de entradas de la realimentación U_{Realim} incluye las salidas del modelo del proceso, la estimación del estado sobrante y las dos salidas del control integral, y para calcular K_{Realim} debe realizarse una ampliación A_a y B_a de las matrices del modelo que considere estas salidas:

$$U_{Realim} = \begin{bmatrix} Y_1 \\ Y_2 \\ Y_{ObRed} \\ Y_{CtrInt_1} \\ Y_{CtrInt_2} \end{bmatrix}, \quad A_a = \begin{bmatrix} A_d & 0_{numX,numY} \\ -C_d & I_{numY} \end{bmatrix}, \qquad B_a = \begin{bmatrix} B_d \\ -D_d \end{bmatrix}$$

Las acciones de control se calculan como la realimentación negativa de $Y_{Realim}=K_{Realim}\cdot Y_{Realim}$.

4.3.2. Fase de simulación en MATLAB

4.3.2.1. Controlador en MATLAB

Al igual que en el anterior control, se opta por tiempos de establecimiento [Tabla 5] del observador más pequeños que el del propio modelo con el fin de obtener su estabilización, y se escogen los de la realimentación según el comportamiento apreciado gráficamente en las salidas:

Tabla 5. Ob.Red+Ctr.Int: Tiempos de establecimiento (Elaboración propia)

TIEMPOS DE ESTABLECIMIENTO DEL OBSERVADOR Y DE LA MATRIZ DE REALIMENTACIÓN				
	$T_s = 50\ ms$	$T_s = 60\ ms$	$T_s = 70\ ms$	$Ts = 80\ ms$
$te_{Ob.Red}$	1	1	1	1
te_{Realim}	2.5	2.5	2.5	2.5

El control en simulación se ha desarrollado en el fichero “Segway_ModeloNL_ctrInt_obRed.slx” [Ilustración 41] de Simulink donde se aplican todos los conceptos mostrados anteriormente, los cuales han sido programados previamente en el script “param_Segway_ctrInt_obRed.mlx” [Script 10] de MATLAB.

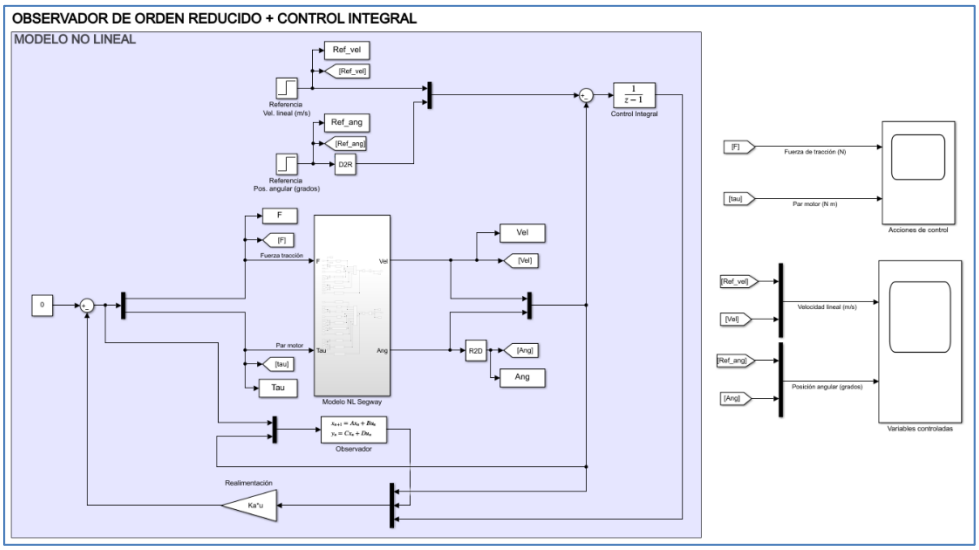

Ilustración 41. Fichero "Segway_ModeloNL_ctrInt_obRed.slx" (Elaboración propia)

1. Parámetros y simulación en MATLAB

```
…
if graf_MAT ~= "SI"
    Ts = input("Elige un período de muestreo: ");
    switch Ts
        case 0.05
            te_obred = 1; te_realim = 2.5;
        …
    end
    SYSd = c2d(SYS,Ts,'zoh');
    [Adisc,Bdisc,Cdisc,Ddisc] = ssdata(SYSd);
    n = size(Adisc,1);
    m_ent = size(Bdisc,2);
    p = size(Cdisc,1);
    A11 = Adisc(1:2,1:2);
    A12 = Adisc(1:2,3);
    A21 = Adisc(3,1:2);
    A22 = Adisc(3,3);
    B1 = Bdisc(1:2,:);
    B2 = Bdisc(3,:);

    sigma_obred = 4/te_obred;
    p_obred = exp(-sigma_obred*Ts);
    L_obred = place(A22', A12', p_obred)';
    A_obred = A22-L_obred*A12;
    B_obred = [B2-L_obred*B1 A21-L_obred*A11+(A22-L_obred*A12)*L_obred];
    C_obred = eye(n-p,n-p);
    D_obred = [zeros(n-p,m_ent) L_obred];
    SYS_obred = ss(A_obred, B_obred, C_obred, D_obred, Ts);

    Aa = [Adisc zeros(n,p); -Cdisc eye(p)];
    Ba = [Bdisc; -Ddisc];

    sigma_realim = 4/te_realim;
    p_ctrint = exp(-sigma_realim*Ts)*[1 1.01 1.001 1.0001 1.00001];
    Ka = place(Aa, Ba, p_ctrint);

    sim("Segway_ModeloNL_ctrInt_obRed.slx");
    …
end
```

2. Resultados

```
…
```

Script 10. Script "param_Segway_ctrInt_obRed.mlx"

4.3.2.2. Resultados

Este control ofrece resultados muy satisfactorios de acciones de control [Ilustración 42] y variables controladas [Ilustración 43] para las cuatro simulaciones realizadas, ya que se alcanza la velocidad lineal exigida en los 2.5 segundos impuestos en la realimentación y sin ningún tipo de sobreoscilación, y el balanceo del péndulo apenas sufre una ligera inclinación máxima de 0.2°.

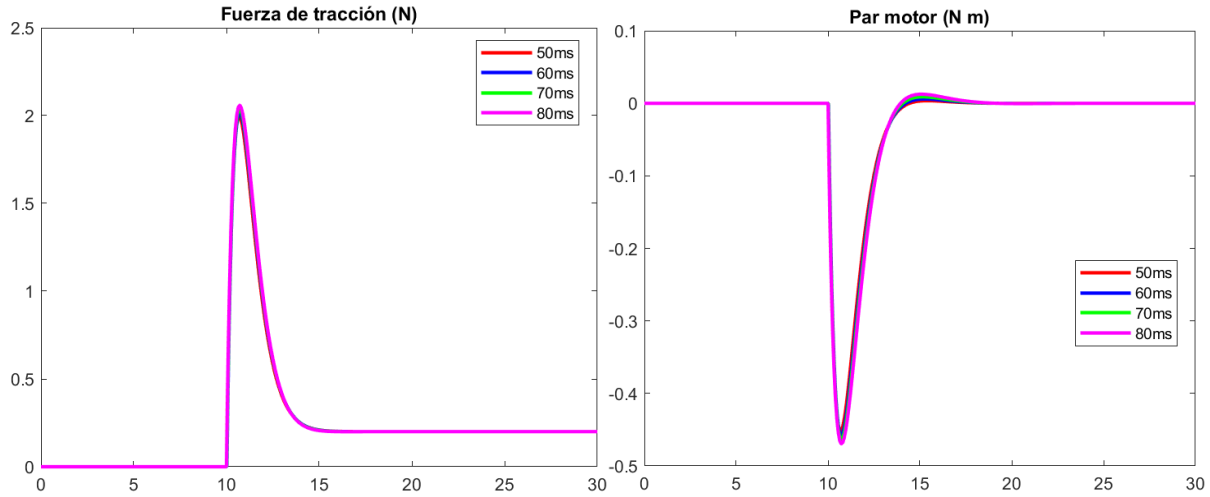

Ilustración 42. Ob.Red+Ctr.Int simulación – Acc. Control: F. tracción y par motor (Elaboración propia)

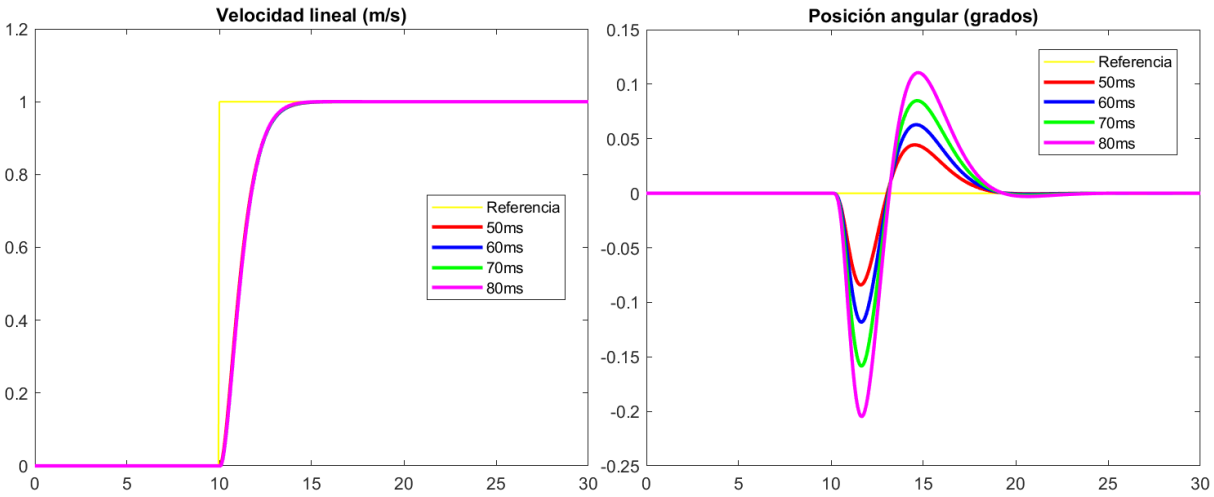

Ilustración 43. Ob.Red+Ctr.Int simulación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

4.3.3. Fase experimental en Raspberry Pi y SoMachine

Para la programación del controlador, se ha hecho uso de la herramienta PLC Coder de MATLAB/Simulink.

4.3.3.1. Controlador en SoMachine

Se ha desarrollado un proyecto "Segway_SoMachine_ctrInt_obRed_80ms.project" [Script 11] en SoMachine en el que se ha programado el control en lenguaje ST siguiendo el mismo procedimiento que en la fase de simulación. Como puede apreciarse, se ha omitido gran parte del contenido por ser idéntico al del control PID, mostrando únicamente aquellos métodos novedosos del controlador.

```
// DECLARACIÓN DE VARIABLES
PROGRAM POU
VAR
	…
	Y_ctrInt_k: ARRAY[0..1] OF LREAL;
	CtrInt_states: ARRAY[0..1] OF LREAL;
	…
	A_ob: LREAL := 0.72614903707369083;
	B_ob: ARRAY[0..3] OF LREAL := [-0.011625615191541681, -0.089213770680626261, 0.0984867846285391, -
0.39864877663281684];
	D_ob: ARRAY[0..3] OF LREAL := [0.0, 0.0, -0.35114596845916046, 3.7567276401137546];
	Ka: ARRAY[0..9] OF LREAL := [9.1318253525384367,-2.1620033301791755,4.5658808313737733,-
12.475579076485175,2.9819692632090833,-3.8227347260097435,-0.56036488583583366,0.12982332757947296,-
0.18168679486365671,0.21497149080924838];
END_VAR
…
// ARRANQUE DEL CONTROLADOR
	…
ELSIF manual = FALSE THEN
	…
	// Control integral
	E_k[0] := Ref[0] - Y_k[0];
	E_k[1] := Ref[1] - Y_k[1];

	Y_ctrInt_k[0] := CtrInt_states[0];
	Y_ctrInt_k[1] := CtrInt_states[1];

	CtrInt_states[0] := E_k[0] - (-CtrInt_states[0]);
	CtrInt_states[1] := E_k[1] - (-CtrInt_states[1]);

	// Observador de orden reducido
	rtb_Observador := (D_ob[2]*Y_k[0]) + Observador_DSTATE;
	Observador_DSTATE := ((((A_ob*Observador_DSTATE) + (B_ob[0]*U_k[0])) + (B_ob[1]*U_k[1])) +
(B_ob[2]*Y_k[0])) + (B_ob[3]*Y_k[1]);
	Y_obRed_k := (D_ob[3]*Y_k[1]) + rtb_Observador;

	// Realimentación del estado
	U_realim[0] := Y_k[0];
	U_realim[1] := Y_k[1];
	U_realim[2] := Y_obRed_k;
	U_realim[3] := Y_ctrInt_k[0];
	U_realim[4] := Y_ctrInt_k[1];

	Y_realim[0] := Ka[0]*U_realim[0] + Ka[2]*U_realim[1] + Ka[4]*U_realim[2] + Ka[6]*U_realim[3] +
Ka[8]*U_realim[4];
	Y_realim[1] := Ka[1]*U_realim[0] + Ka[3]*U_realim[1] + Ka[5]*U_realim[2] + Ka[7]*U_realim[3] +
Ka[9]*U_realim[4];

	// Cálculo de las acciones de control
	U_k[0] := 0.0 - Y_realim[0];
	U_k[1] := 0.0 - Y_realim[1];
	…
END_IF
```

Script 11. Proyecto "Segway_SoMachine_ctrInt_obRed_80ms.project"

4.3.3.2. Resultados

Los resultados experimentales en acciones de control [Ilustración 44] y variables controladas [Ilustración 45] son favorables, siéndolo quizás menos en la posición angular debido a ligeras oscilaciones presentes en torno a la referencia, dándose para todos los períodos de muestreo.

En la comparativa entre simulación, fase intermedia y experimentación para Ts = 80 ms [Ilustración 46] [Ilustración 47], las acciones de control muestran una gran similitud, y es por ello que se aprecian unos grandes resultados en la velocidad lineal y la posición angular, la cual sufre un muy ligero incremento debido a los problemas de cuantificación existentes.

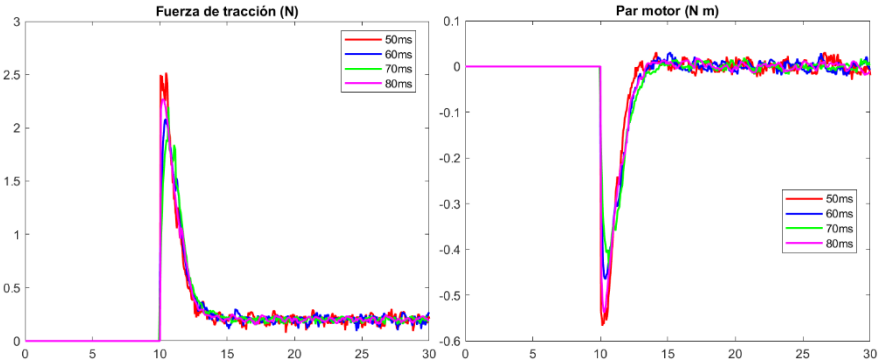

Ilustración 44. Ob.Red+Ctr.Int experimentación – Acc. Control: F. tracción y par motor (Elaboración propia)

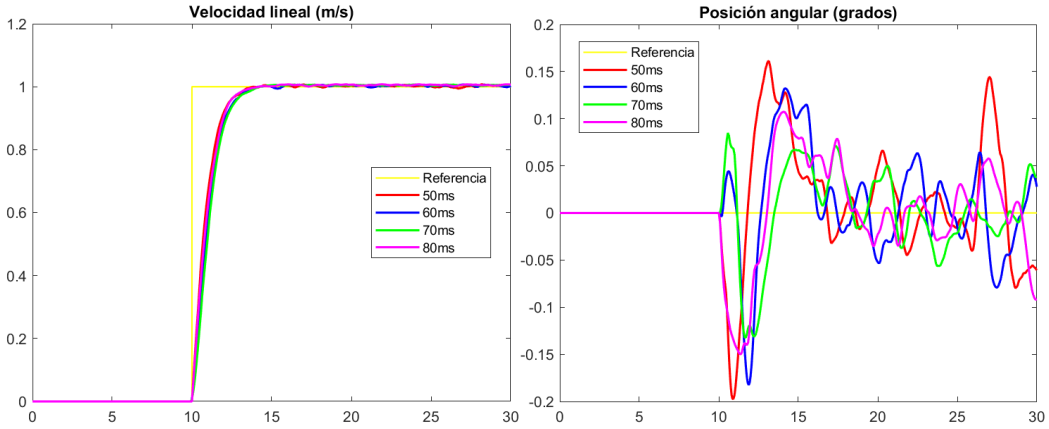

Ilustración 45. Ob.Red+Ctr.Int experimentación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

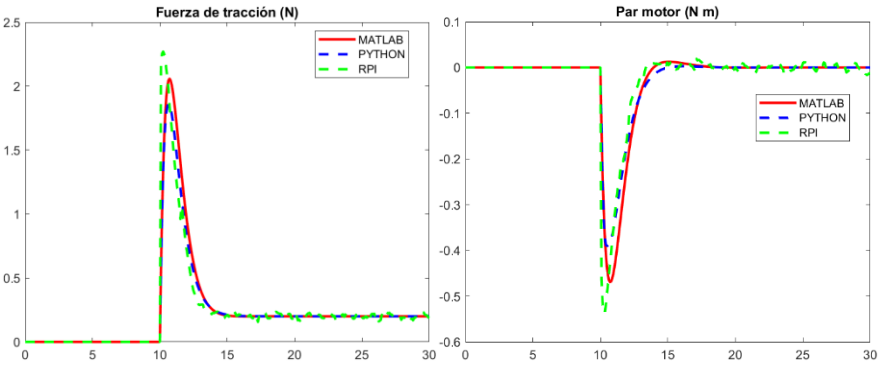

Ilustración 46. Ob.Red+Ctr.Int - comparativa acc. Control entre simulación y experimentación (Elaboración propia)

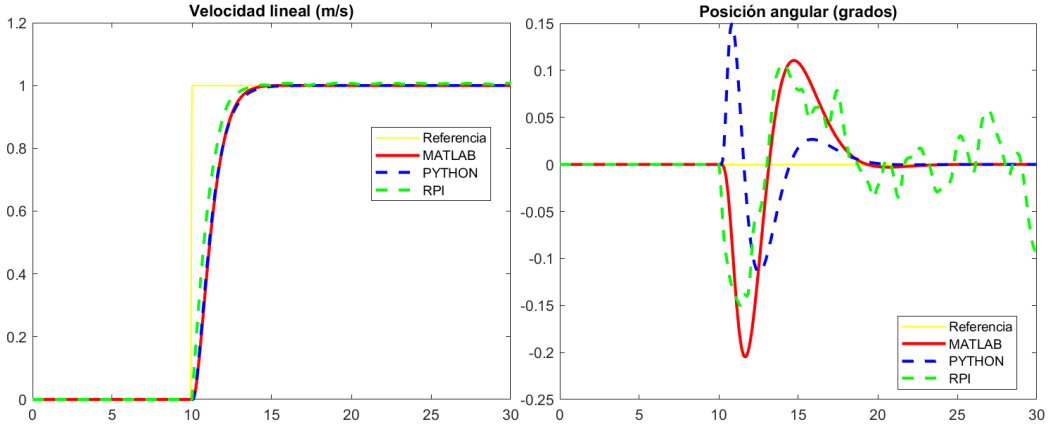

Ilustración 47. Ob.Red+Ctr.Int - comparativa var. Ctr entre simulación y experimentación (Elaboración propia)

4.4. Observador de orden completo + Control Integral

4.4.1. Desarrollo teórico y aplicación sobre el Segway

Este control, al igual que en el apartado 4.2, se basa en los principios de realimentación del estado y de estimación del vector de estados mediante un observador de orden completo, pero en este caso se aplica el control integral, mencionado en el apartado 4.3, sobre el error del sistema, sin necesidad de prealimentar las referencias. Por tanto, esta técnica es similar a la anterior, pero esta vez se utiliza un observador de orden completo en lugar de uno reducido.

Por ello, la ampliación del vector de entradas de la realimentación, U_{Realim}, considera los siguientes parámetros:

$$U_{Realim} = \begin{bmatrix} Y_{ObCom_1} \\ Y_{ObCom_2} \\ Y_{ObCom_3} \\ Y_{CtrInt_1} \\ Y_{CtrInt_2} \end{bmatrix}$$

4.4.2. Fase de simulación en MATLAB

4.4.2.1. Controlador en MATLAB

Al igual que en el anterior control, se opta por tiempos de establecimiento [Tabla 6] del observador más pequeños que el del propio modelo con el fin de obtener su estabilización, y se escogen los de la realimentación según el comportamiento apreciado gráficamente en las salidas:

Tabla 6. Ob.Com+Ctr.Int: Tiempos de establecimiento (Elaboración propia)

TIEMPOS DE ESTABLECIMIENTO DEL OBSERVADOR Y DE LA MATRIZ DE REALIMENTACIÓN				
	$T_s = 50\ ms$	$T_s = 60\ ms$	$T_s = 70\ ms$	$Ts = 80\ ms$
$te_{Ob.Com}$	1	1	1	1
te_{Realim}	2.4	2.7	2.5	2.5

El control en simulación se ha desarrollado en el fichero “Segway_ModeloNL_ctrInt_obCom.slx” [Ilustración 48] de Simulink donde se aplican todos los conceptos mostrados anteriormente, los cuales han sido programados previamente en el script “param_Segway_ctrInt_obCom.mlx” [Script 12] de MATLAB.

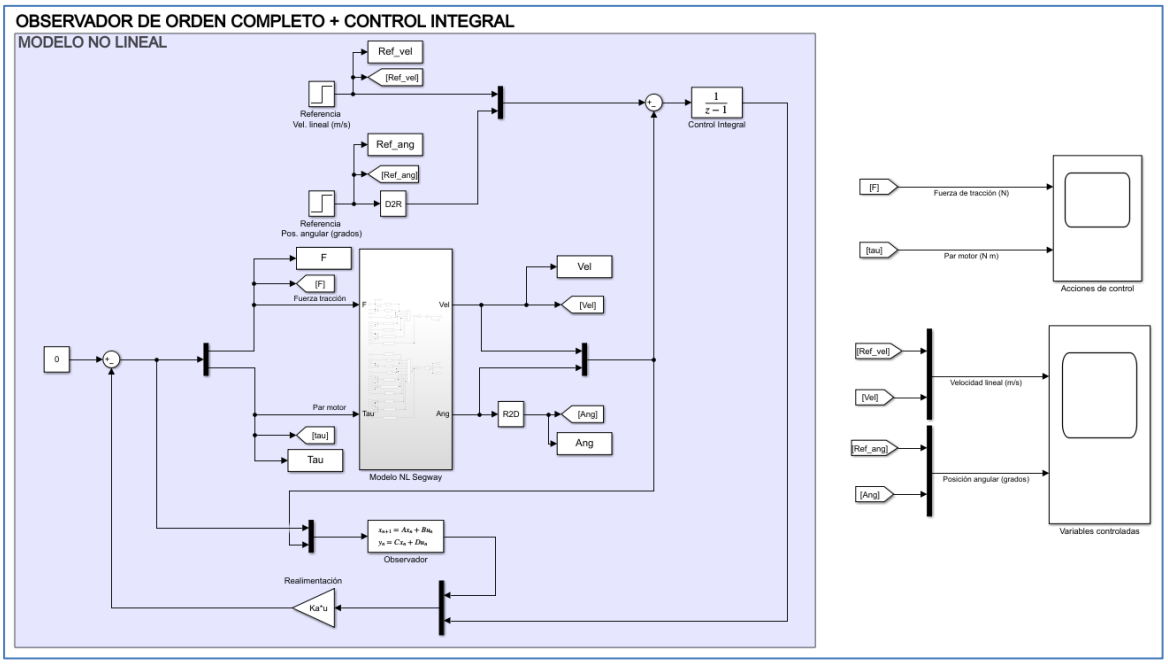

Ilustración 48. Fichero "Segway_ModeloNL_ctrInt_obCom.slx" (Elaboración propia)

1. Parámetros y simulación en MATLAB

```
…
if graf_MAT ~= "SI"
    Ts = input("Elige un período de muestreo: ");
    switch Ts
        case 0.05
            te_obcom = 1;
            te_realim = 2.4;
        …
    end

    SYSd = c2d(SYS,Ts,'zoh');
    [Adisc,Bdisc,Cdisc,Ddisc] = ssdata(SYSd);
    n = size(Adisc,1);
    m_ent = size(Bdisc,2);
    p = size(Cdisc,1);

    Aa = [Adisc zeros(n,p); -Cdisc eye(p)];
    Ba = [Bdisc; -Ddisc];

    sigma_obcom = 4/te_obcom;
    p_obcom = exp(-sigma_obcom*Ts)*[1 1.01 1.02];
    L_obcom = place(Adisc', Cdisc', p_obcom)';
    A_obcom = Adisc-L_obcom*Cdisc;
    B_obcom = [Bdisc-L_obcom*Ddisc L_obcom];
    C_obcom = eye(n,n);
    D_obcom = [zeros(n,m_ent) zeros(n,p)];
    SYS_obcom = ss(A_obcom, B_obcom, C_obcom, D_obcom, Ts);

    sigma_realim = 4/te_realim;
    p_ctrint = exp(-sigma_realim*Ts)*[1 1.01 1.02 1.03 1.04];
    Ka = place(Aa, Ba, p_ctrint);

    sim("Segway_ModeloNL_ctrInt_obCom.slx");
    …
end
```

2. Resultados

```
…
```

Script 12. Script "param_Segway_ctrInt_obCom.mlx"

4.4.2.2. Resultados

Este control ofrece resultados muy satisfactorios de acciones de control [Ilustración 49] y variables controladas [Ilustración 50] para las cuatro simulaciones realizadas, ya que se alcanza la velocidad lineal exigida en los 2.5 segundos impuestos en la realimentación y sin ningún tipo de sobreoscilación, y el balanceo del péndulo apenas sufre una ligera inclinación máxima de 0.3°.

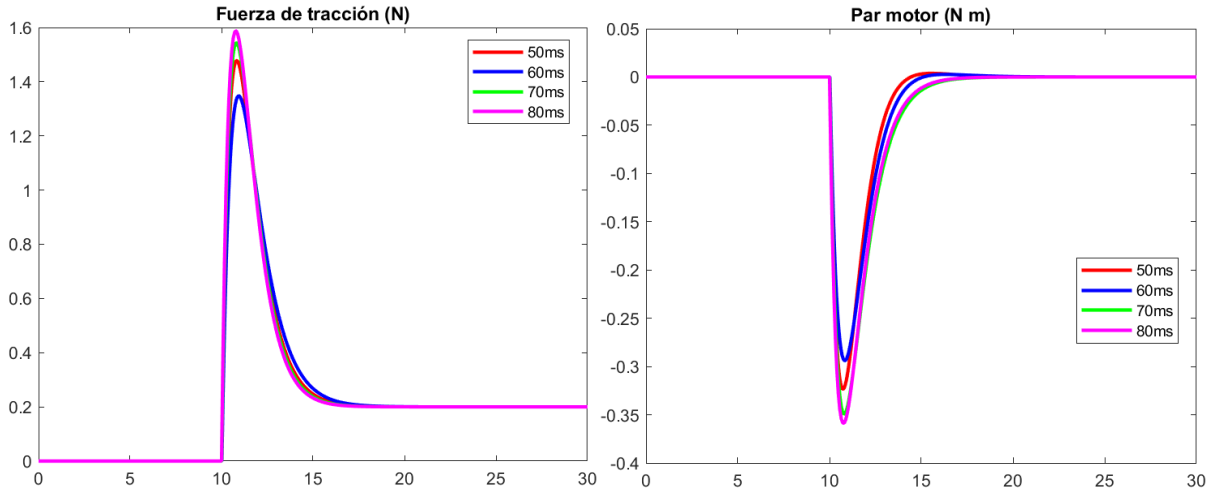

Ilustración 49. Ob.Com+Ctr.Int simulación – Acc. Control: F. tracción y par motor (Elaboración propia)

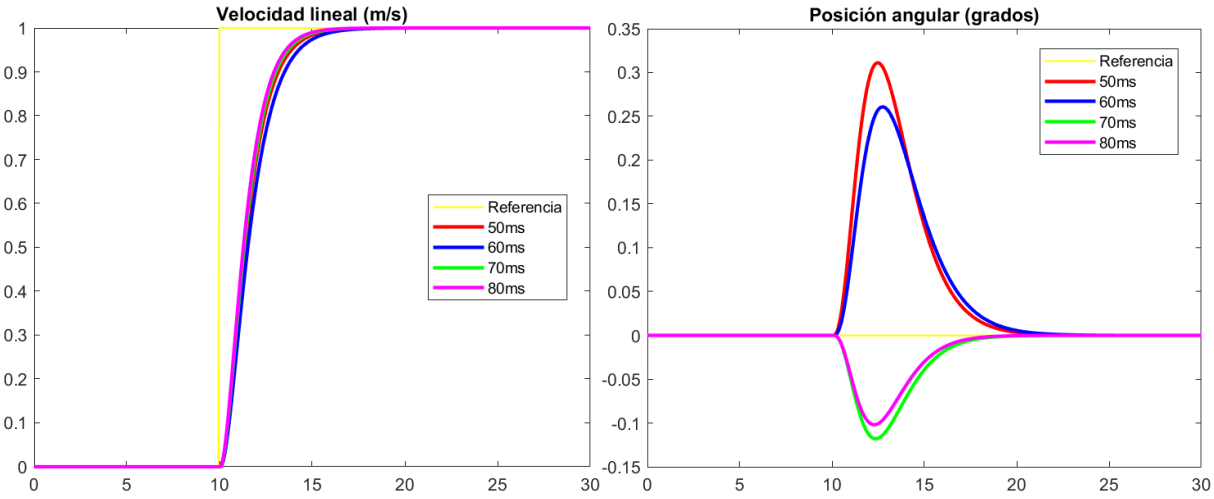

Ilustración 50. Ob.Com+Ctr.Int simulación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

4.4.3. Fase experimental en Raspberry Pi y SoMachine

Para la programación del controlador, se ha hecho uso de la herramienta PLC Coder de MATLAB/Simulink.

4.4.3.1. Controlador en SoMachine

Se ha desarrollado un proyecto "Segway_SoMachine_ctrInt_obCom_80ms.project" [Script 13] en SoMachine en el que se ha programado el control en lenguaje ST siguiendo el mismo procedimiento que en la fase de simulación. Como puede apreciarse, se ha omitido gran parte del contenido por ser idéntico al del control PID, mostrando únicamente aquellos métodos novedosos del controlador.

```
// DECLARACIÓN DE VARIABLES
PROGRAM POU
VAR
	…
	A_ob: ARRAY[0..8] OF LREAL :=
[0.7298290254583083,0.038264197778461795,0.14115633081094545,0.027638163905995272,0.437801861723609,-
1.0930696927721588,-0.0075587911540941173,0.080867565642392752,1.032600695151366];
	B_ob: ARRAY[0..11] OF LREAL := [0.031964790878767663,-0.0010701652444498326,-
0.026870241994574487,0.026870241994560316,-0.0046155701596880391,-0.11598858782232246,0.2637780163659269,-
0.038050164729571727,-0.13578228241202789,-0.21742805714942304,0.594798833427757,1.912320286265158];
	C_ob: ARRAY[0..8] OF LREAL := [1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0];
	Ka: ARRAY[0..9] OF LREAL := [7.7638463262781467,-1.8366076368974698,1.5773710486388701,-
10.756037186818565,1.9447560671612703,-3.2506252772720536,-0.40519166287367531,0.09301075120533564,-
0.020383246767100457,0.11819196900507697];
END_VAR
…
// ARRANQUE DEL CONTROLADOR
	…
ELSIF manual = FALSE THEN
	…
	// Control integral
	…
	// Observador de orden completo
	rtb_Observador[0] := (C_ob[6]*Observador_DSTATE[2]) + ((C_ob[3]*Observador_DSTATE[1]) +
(C_ob[0]*Observador_DSTATE[0]));
	xnew[0] := (B_ob[9]*Y_k[1]) + ((B_ob[6]*Y_k[0]) + ((B_ob[3]*U_k[1]) + ((B_ob[0]*U_k[0]) +
((A_ob[6]*Observador_DSTATE[2]) + ((A_ob[3]*Observador_DSTATE[1]) + (A_ob[0]*Observador_DSTATE[0])))));
	rtb_Observador[1] := (C_ob[7]*Observador_DSTATE[2]) + ((C_ob[4]*Observador_DSTATE[1]) +
(C_ob[1]*Observador_DSTATE[0]));
	xnew[1] := (B_ob[10]*Y_k[1]) + ((B_ob[7]*Y_k[0]) + ((B_ob[4]*U_k[1]) + ((B_ob[1]*U_k[0]) +
((A_ob[7]*Observador_DSTATE[2]) + ((A_ob[4]*Observador_DSTATE[1]) + (A_ob[1]*Observador_DSTATE[0])))));
	rtb_Observador[2] := (C_ob[8]*Observador_DSTATE[2]) + ((C_ob[5]*Observador_DSTATE[1]) +
(C_ob[2]*Observador_DSTATE[0]));
	xnew[2] := (B_ob[11]*Y_k[1]) + ((B_ob[8]*Y_k[0]) + ((B_ob[5]*U_k[1]) + ((B_ob[2]*U_k[0]) +
((A_ob[8]*Observador_DSTATE[2]) + ((A_ob[5]*Observador_DSTATE[1]) + (A_ob[2]*Observador_DSTATE[0])))));

	Observador_DSTATE[0] := xnew[0];
	Observador_DSTATE[1] := xnew[1];
	Observador_DSTATE[2] := xnew[2];
	Y_obCom_k[0] := rtb_Observador[0];
	Y_obCom_k[1] := rtb_Observador[1];
	Y_obCom_k[2] := rtb_Observador[2];
	// Realimentación del estado
	U_realim[0] := Y_obCom_k[0];
	U_realim[1] := Y_obCom_k[1];
	U_realim[2] := Y_obCom_k[2];
	U_realim[3] := Y_ctrInt_k[0];
	U_realim[4] := Y_ctrInt_k[1];

	Y_realim[0] := Ka[0]*U_realim[0] + Ka[2]*U_realim[1] + Ka[4]*U_realim[2] + Ka[6]*U_realim[3] +
Ka[8]*U_realim[4];
	Y_realim[1] := Ka[1]*U_realim[0] + Ka[3]*U_realim[1] + Ka[5]*U_realim[2] + Ka[7]*U_realim[3] +
Ka[9]*U_realim[4];
	// Cálculo de las acciones de control
	U_k[0] := 0.0 - Y_realim[0];
	U_k[1] := 0.0 - Y_realim[1];
	…
END_IF
```

Script 13. Proyecto "Segway_SoMachine_ctrInt_obCom_80ms.project"

4.4.3.2. Resultados

Los resultados experimentales en acciones de control [Ilustración 51] y variables controladas [Ilustración 52] son favorables, siéndolo quizás menos en la posición angular debido a ligeras oscilaciones presentes en torno a la referencia, dándose asó en todos los períodos de muestreo.

En la comparativa entre simulación, fase intermedia y experimentación para Ts = 80 ms [Ilustración 53] [Ilustración 54], las acciones de control muestran una gran similitud, y es por ello que se aprecian grandes resultados en la velocidad lineal y la posición angular, la cual sufre un muy ligero incremento y le cuesta un poco más alcanzar la referencia debido a los problemas de cuantificación existentes.

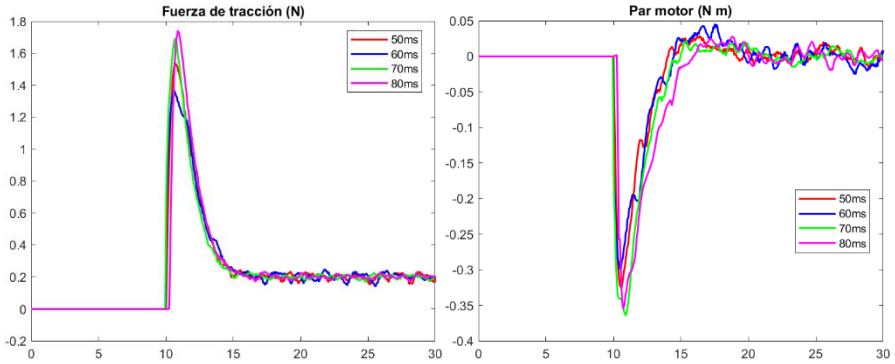

Ilustración 51. Ob.Com+Ctr.Int experimentación – Acc. Control: F. tracción y par motor (Elaboración propia)

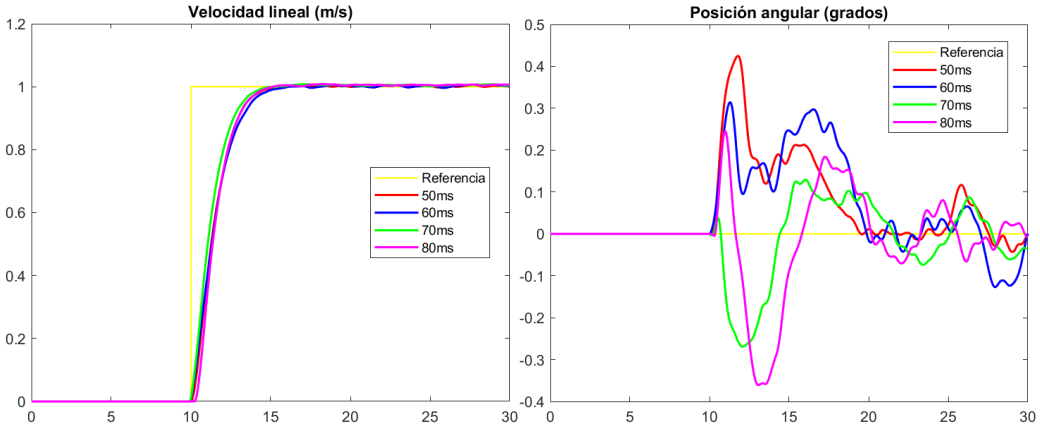

Ilustración 52. Ob.Com+Ctr.Int experimentación – Var.Ctr.Vel.lineal y pos.angular (Elaboración propia)

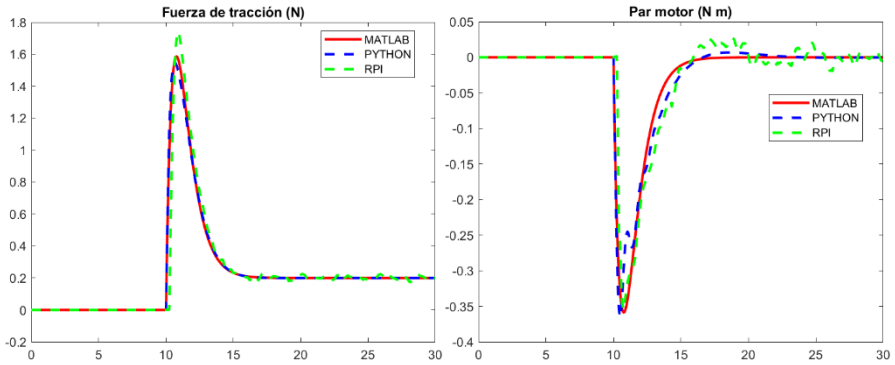

Ilustración 53. Ob.Com+Ctr.Int – Comparativa acc. Control entre simulación y experimentación (Elaboración propia)

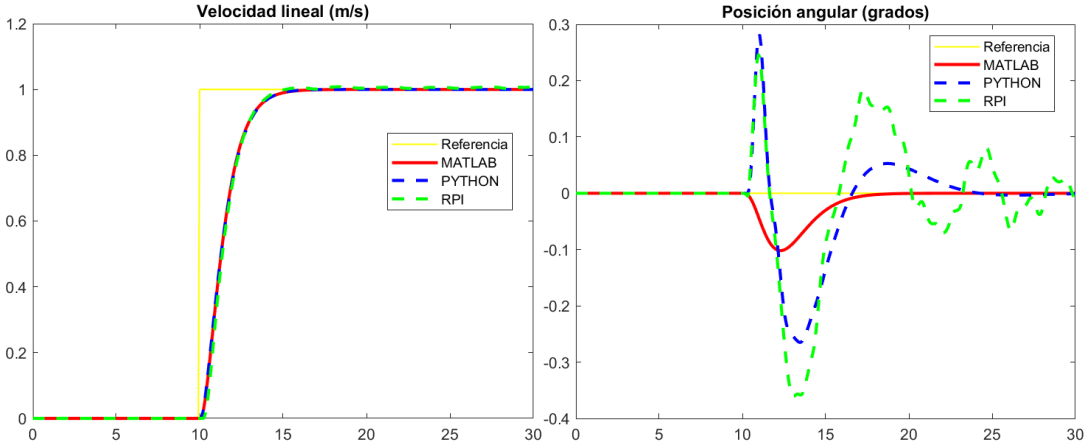

Ilustración 54. Ob.Com+Ctr.Int – Comparativa var. Ctr entre simulación y experimentación (Elaboración propia)

4.5. Observador de orden reducido + LQR + Control Integral

4.5.1. Desarrollo teórico y aplicación sobre el Segway

Este control es semejante al implementado en el apartado 4.3, ya que se hace uso de un observador de orden reducido, una matriz de realimentación del estado y un control integral del error del sistema. Sin embargo, la diferencia entre uno y otro radica en la forma de calcular la matriz K_{Realim}, ya que anteriormente se hacía imponiendo una dinámica al sistema mediante la asignación de polos estables, mientras que ahora se dispone de una expresión de un índice de coste J que debe minimizarse mediante el diseño del control.

A este método se le denomina control óptimo, y la matriz de realimentación recibe el nombre de regulador lineal cuadrático (LQR).

4.5.1.1. Regulador Lineal Cuadrático (LQR)

Como se ha mencionado anteriormente, se desea combinar este regulador LQR con un observador de estado de orden reducido y un control integral, por lo que el índice de costes del control, teniendo en cuenta que se trabaja con variables discretas, debe adoptar la siguiente expresión:

$$J = \sum_{k=0}^{\infty} \left(x_{a_k}^T \cdot Q_a \cdot x_{a_k} + u_k^T \cdot R \cdot u_k + 2 \cdot x_{a_k}^T \cdot M_a \cdot u_k \right) = \sum_{k=0}^{\infty} \left(\begin{bmatrix} x_{a_k}^T & u_k^T \end{bmatrix} \cdot \begin{bmatrix} Q_a & M_a \\ M_a^T & R \end{bmatrix} \cdot \begin{bmatrix} x_{a_k} \\ u_k \end{bmatrix} \right)$$

Siendo “a” la abreviatura de sistema ampliado por la salida del control integral:

$$x_{a_k} = \begin{bmatrix} x_k \\ y_{CtrInt_k} \end{bmatrix}$$

De esta forma, Q_a es la matriz de ponderación de los estados del sistema ampliado, R es la matriz de ponderación de las entradas y M_a es la que pondera la relación entre estados, siendo todas ellas matrices diagonales. Asimismo, cuanto mayor sea el valor de la matriz que multiplica a la variable en cuestión, mayor relevancia cobra ésta en el control del proceso.

Sin embargo, para implementar dicho control óptimo es necesaria comprobar previamente las siguientes condiciones del sistema ampliado $SYS_a = [A_a, B_a, C_a, D_a]$:

- Todas las ponderaciones de los estados son positivas o nulas: $Q_a \geq 0$.
- Todas las ponderaciones de las entradas son positivas: $R > 0$.
- Para índices mixtos, todas las ponderaciones son positivas: $\begin{bmatrix} Q_a & M_a \\ M_a^T & R \end{bmatrix} > 0$.
- El sistema debe ser totalmente controlable, para así aplicar realimentación del estado: $rank\big(ctrb(SYS_a)\big) = núm.\, estados\ SYS_a$.
- El par $(A_a, \sqrt{Q_a})$ debe ser observable, para que todos los estados estén ponderados directa o indirectamente: $rank\left(obsv\left(\left(A_a, \sqrt{Q_a}\right)\right)\right) = núm.\, estados\ SYS_a$.

Si todas las condiciones se dan, es posible calcular la matriz K_{Realim} mediante el comando ‘dlqr’ para modelos discretizados.

4.5.2. Fase de simulación en MATLAB

4.5.2.1. Controlador en MATLAB

En las matrices de ponderación [Tabla 7], en el caso de los errores del sistema se opta por darle siempre una mayor importancia a la posición angular, ya que se trata del parámetro que provoca más inestabilidad en el Segway, mientras que las ponderaciones de las entradas no cobran gran relevancia, ya que no se han apreciado grandes diferencias en la respuesta del sistema según se pondere más una u otra.

Tabla 7. Matrices de ponderación y tiempo de establecimiento del observador (Elaboración propia)

MATRICES DE PONDERACIÓN Y TIEMPO DE ESTABLECIMIENTO DEL OBSERVADOR				
	$T_s = 50\ ms$	$T_s = 60\ ms$	$T_s = 80\ ms$	$Ts = 100\ ms$
Q_v	$\begin{bmatrix} 0.004 & 0 \\ 0 & 0.1 \end{bmatrix}$	$\begin{bmatrix} 0.004 & 0 \\ 0 & 0.07 \end{bmatrix}$	$\begin{bmatrix} 0.004 & 0 \\ 0 & 0.02 \end{bmatrix}$	$\begin{bmatrix} 0.004 & 0 \\ 0 & 0.005 \end{bmatrix}$
R	$\begin{bmatrix} 0.01 & 0 \\ 0 & 0.005 \end{bmatrix}$	$\begin{bmatrix} 0.01 & 0 \\ 0 & 0.005 \end{bmatrix}$	$\begin{bmatrix} 0.01 & 0 \\ 0 & 0.005 \end{bmatrix}$	$\begin{bmatrix} 0.01 & 0 \\ 0 & 0.005 \end{bmatrix}$
te_{ObRed}	1.5	1.5	1.5	1.5

El control en simulación se ha desarrollado en el fichero "Segway_ModeloNL_LQR_ctrInt_obRed.slx" [Ilustración 55] de Simulink donde se aplican todos los conceptos mostrados anteriormente, los cuales han sido programados previamente en el script "param_Segway_LQR_ctrInt_obRed.mlx" [Script 14] de MATLAB.

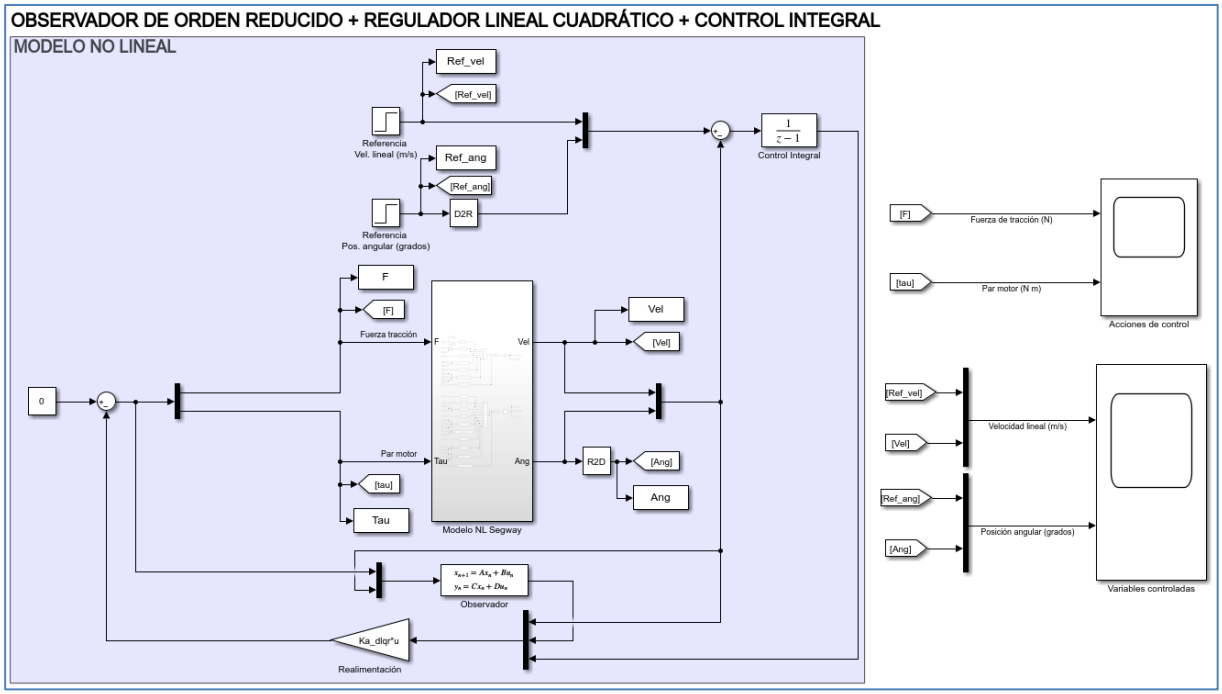

Ilustración 55. Fichero "Segway_ModeloNL_LQR_ctrInt_obRed.slx" (Elaboración propia)

1. Parámetros y simulación en MATLAB

```
…
if graf_MAT ~= "SI"
    Ts = input("Elija un período de muestreo: ");
    switch Ts
        case 0.05
            Qv = diag([0.004 0.1]);
            R = diag([0.01 0.005]);
        …
    end
    SYSd = c2d(SYS,Ts,'zoh');
    [Adisc,Bdisc,Cdisc,Ddisc] = ssdata(SYSd);
    n = size(Adisc,1);
    m_ent = size(Bdisc,2);
    p = size(Cdisc,1);
    A11 = Adisc(1:2,1:2);
    A12 = Adisc(1:2,3);
    A21 = Adisc(3,1:2);
    A22 = Adisc(3,3);
    B1 = Bdisc(1:2,:);
    B2 = Bdisc(3,:);

    te_obred = 1.5;
    sigma_obred = 4/te_obred;
    p_obred = exp(-sigma_obred*Ts);
    L_obred = place(A22', A12', p_obred)';
    A_obred = A22-L_obred*A12;
    B_obred = [B2-L_obred*B1 A21-L_obred*A11+(A22-L_obred*A12)*L_obred];
    C_obred = eye(n-p,n-p);
    D_obred = [zeros(n-p,m_ent) L_obred];
    SYS_obred = ss(A_obred, B_obred, C_obred, D_obred, Ts);

    Qx = zeros(n);
    Mxu = zeros(n,m_ent);
    Qa = [Qx zeros(n,p); zeros(p,n) Qv];
    Ra = R;
    Ma = [Mxu; zeros(p,m_ent)];
    Qxu_a = [Qa Ma; Ma' Ra];

    Aa_disc = [Adisc zeros(n,p); -Cdisc eye(p)];
    Ba_disc = [Bdisc; -Ddisc];
    Ca_disc = [Cdisc zeros(p); zeros(p,n+p)];
    Da_disc = [Ddisc; zeros(p)];
    SYSa_disc = ss(Aa_disc,Ba_disc,Ca_disc,Da_disc,Ts);
    disp('Valores propios de Qa >= 0:'), eig(Qa)
    disp('Valores propios de Ra > 0:'), eig(Ra)
    disp('Valores propios de Qxu_a > 0:'), eig(Qxu_a)
    disp('Controlabilidad del sistema SYSa_disc:'), rank(ctrb(SYSa_disc))
    disp('Observabilidad del par (Aa_disc,sqrt(Qa)):'), rank(obsv(Aa_disc,sqrt(Qa)))
    Ka_dlqr = dlqr(Aa_disc,Ba_disc,Qa,Ra,Ma);

    sim("Segway_ModeloNL_LQR_ctrInt_obRed.slx");
    …
end
```

2. Resultados

```
…
```

Script 14. Script “param_Segway_LQR_ctrInt_obRed.mlx”

4.5.2.2. Resultados

Este control ofrece resultados bastante aceptables de acciones de control [Ilustración 56] y variables controladas [Ilustración 57] para las cuatro simulaciones realizadas, ya que se alcanza la velocidad lineal exigida en, aproximadamente, 4 segundos con una sobreoscilación previa del 5%, y el balanceo del péndulo sufre una inclinación máxima de 2.5°, más notoria que en anteriores controles pero igualmente casi imperceptible.

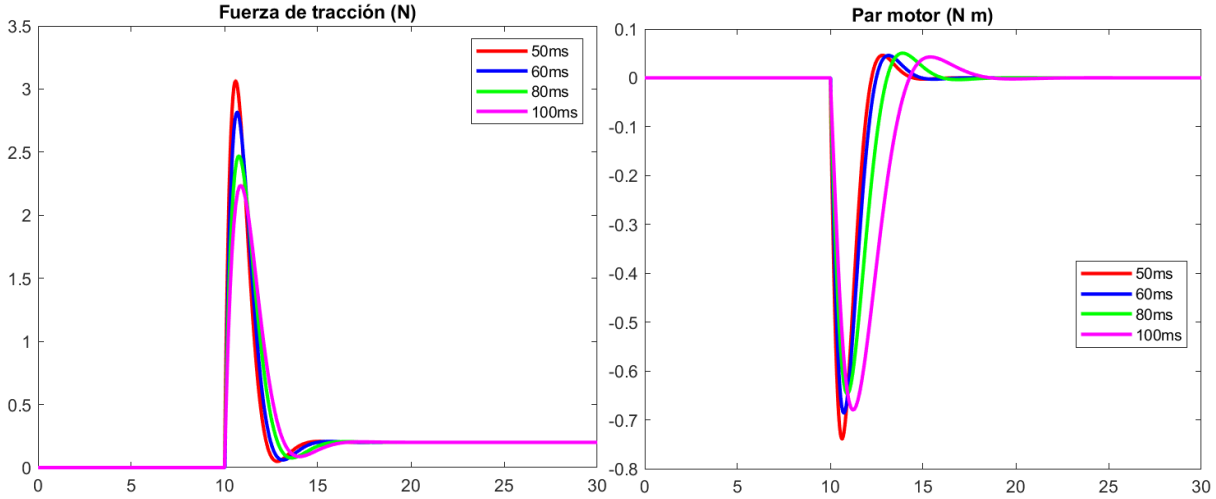

Ilustración 56. Ob.Red+LQR+Ctr.Int simulación – Acc. Control: F. tracción y par motor (Elaboración propia)

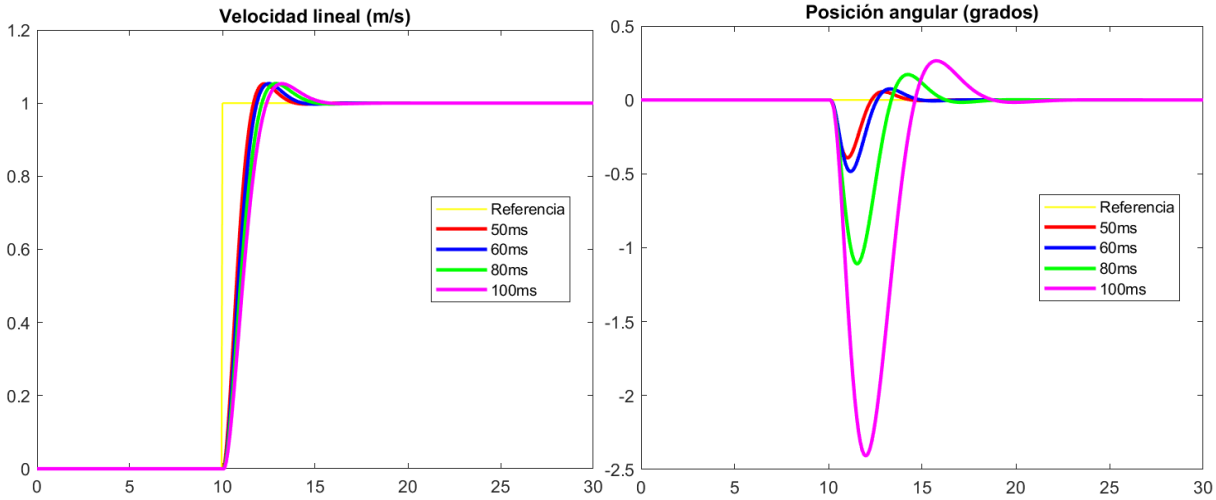

Ilustración 57. Ob.Red+LQR+Ctr.Int simulación – Var. Ctr: Vel. Lineal y pos. Angular (Elaboración propia)

4.5.3. Fase experimental en Raspberry Pi y SoMachine

Para la programación del controlador, se ha hecho uso de la herramienta PLC Coder de MATLAB/Simulink.

4.5.3.1. Controlador en SoMachine

Se ha desarrollado un proyecto "Segway_SoMachine_LQR_ctrInt_obRed_100ms.project" [Script 15] en SoMachine en el que se ha programado el control en lenguaje ST siguiendo el mismo procedimiento que en la fase de simulación. Como puede apreciarse, se ha omitido gran parte del contenido por ser idéntico al del control PID, mostrando únicamente aquellos métodos novedosos del controlador.

```
// DECLARACIÓN DE VARIABLES
PROGRAM POU
VAR
	…
	A_ob: LREAL := 0.76592833836464869;
	B_ob: ARRAY[0..3] OF LREAL := [-0.016252929470901498, -0.11495721477063719, 0.078643630661699437,
0.164405872857450 67];
	D_ob: ARRAY[0..3] OF LREAL := [0.0, 0.0, -0.32209385895235776, 2.7665028819660993];
	Ka: ARRAY[0..9] OF LREAL := [5.9922455894526214,-1.6460418954113503,0.031065893625485709,-
16.466805976368658,1.0230111991452961,-4.7154131730908571,-
0.569944225357037,0.093619786599838842,0.0453874698766572,0.68365392698931871];
END_VAR
…
// ARRANQUE DEL CONTROLADOR
	…
ELSIF manual = FALSE THEN
	…
	// Control integral
	…
	// Observador de orden reducido
	rtb_Observador := D_ob[2]*Y_k[0] + Observador_DSTATE;
	Observador_DSTATE := A_ob*Observador_DSTATE + B_ob[0]*U_k[0] + B_ob[1]*U_k[1] + B_ob[2]*Y_k[0] +
B_ob[3]*Y_k[1];
	Y_obRed_k := D_ob[3]*Y_k[1] + rtb_Observador;

	// Realimentación del estado
	U_realim[0] := Y_k[0];
	U_realim[1] := Y_k[1];
	U_realim[2] := Y_obRed_k;
	U_realim[3] := Y_ctrInt_k[0];
	U_realim[4] := Y_ctrInt_k[1];

	Y_realim[0] := Ka[0]*U_realim[0] + Ka[2]*U_realim[1] + Ka[4]*U_realim[2] + Ka[6]*U_realim[3] +
Ka[8]*U_realim[4];
	Y_realim[1] := Ka[1]*U_realim[0] + Ka[3]*U_realim[1] + Ka[5]*U_realim[2] + Ka[7]*U_realim[3] +
Ka[9]*U_realim[4];

	// Cálculo de las acciones de control
	U_k[0] := 0.0 - Y_realim[0];
	U_k[1] := 0.0 - Y_realim[1];
	…
END_IF
```

Script 15. Proyecto "Segway_SoMachine_LQR_ctrInt_obRed_100ms.project"

4.5.3.2. Resultados

Los resultados experimentales en acciones de control [Ilustración 58] y variables controladas [Ilustración 59] son ciertamente correctos, ya que, para todos los períodos de muestreo, el comportamiento de la velocidad lineal y de la posición angular es muy similar al de la fase de simulación.

En concreto, en la comparativa entre simulación, fase intermedia y experimentación para Ts = 100 ms [Ilustración 60] [Ilustración 61], apenas se aprecian diferencias entre sí, por lo que puede concluirse que el control óptimo basado en regulador LQR ha sido exitoso en su puesta en práctica aun existiendo problemas de cuantificación y de tiempos de computación.

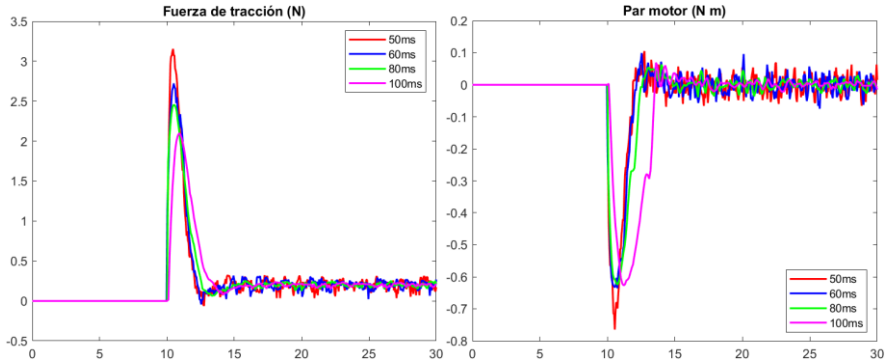

Ilustración 58. Ob.Red+LQR+Ctr.Int experimentación – Acc. Control: F. tracción y par motor (Elaboración propia)

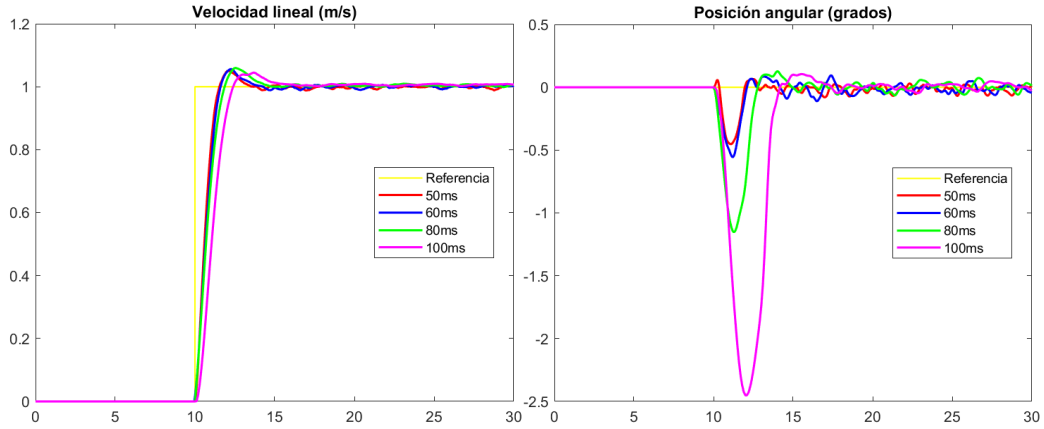

Ilustración 59. Ob.Red+LQR+Ctr.Int experimentación – Var. Ctr: Vel. Lineal y pos. Angular (Elaboración propia)

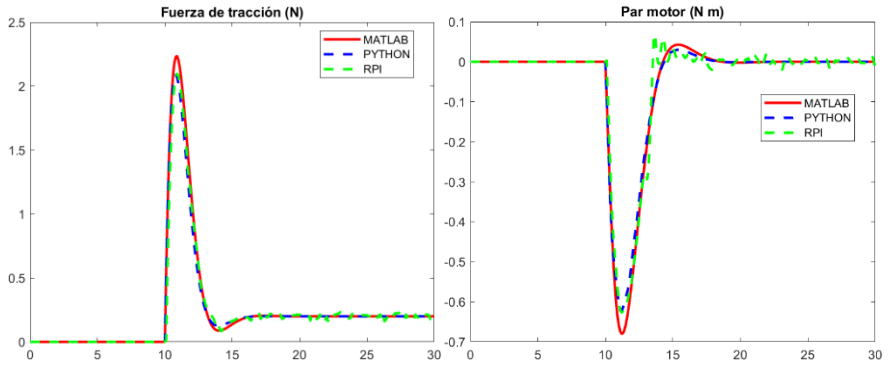

Ilustración 60. Ob.Red+LQR+Ctr.Int – Comparativa acc. Control entre simulación y experimentación (Elaboración propia)

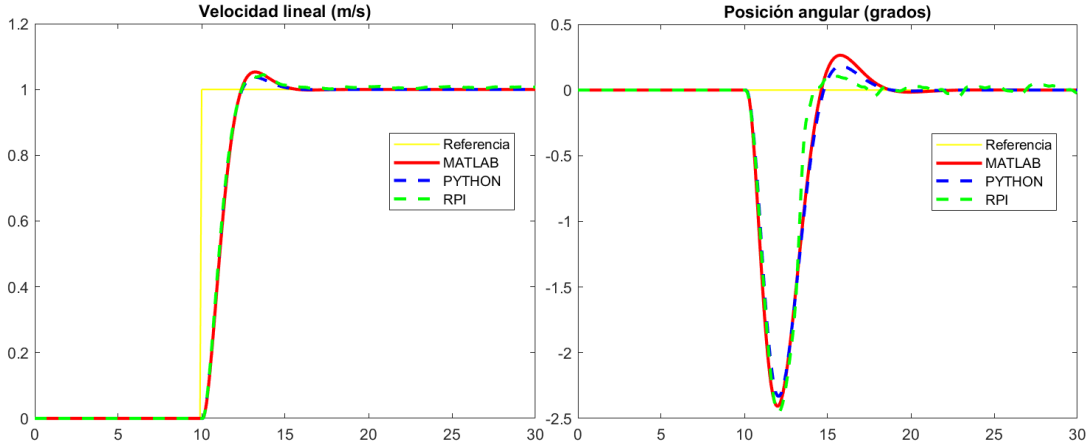

Ilustración 61. Ob.Red+LQR+Ctr.Int – Comparativa var. Ctr entre simulación y experimentación (Elaboración propia)

4.6. Observador de Kalman + LQR + Control Integral

4.6.1. Desarrollo teórico y aplicación sobre el Segway

Este control tiene la misma estructura que el de observador de orden completo, realimentación del estado y control integral mostrado en el apartado 4.4. Sin embargo, el método de cálculo de la matriz de realimentación K_{Realim} es el utilizado en el apartado 4.5, es decir, es un regulador lineal cuadrático basado en control óptimo. Asimismo, el observador de orden completo es básicamente un filtro de Kalman, que consiste en la estimación de estados de un modelo lineal minimizando la medida del error cuadrático, es decir, la varianza del error de estimación. Comúnmente, se le denomina estimador de estado óptimo o LQE.

4.6.1.1. Observador de estado óptimo de Kalman

Este tipo de observador suele utilizarse en sistemas con ruido de proceso y de medida, es decir, el espacio de estados del Segway debe definirse como:

$$\begin{cases} x_{k+1} = A_d \cdot x_k + B_d \cdot u_k + G_d \cdot w_k \\ y_k = C_d \cdot x_k + D_d \cdot u_k + v_k \end{cases}$$

Donde "w" es el ruido del proceso, es decir, el que está implícito en el modelo, y "v" es el ruido de la medida, es decir, el que se adhiere al medir las salidas del proceso. Estos ruidos deben considerarse en forma de varianzas, o de matrices de varianzas si se trata de un sistema multivariable, como es el caso del Segway. De esta forma, el objetivo de este estimador es calcular la ganancia del observador, $L_{ObKalman}$, que minimice un índice de coste interpretado como la suma del error de las varianzas para cada estado, y que se calcula, en el caso del Segway como modelo lineal discreto, como:

$$Q = \begin{bmatrix} var_{w_1} & 0 & 0 \\ 0 & var_{w_2} & 0 \\ 0 & 0 & var_{w_3} \end{bmatrix}, \quad R = \begin{bmatrix} var_{v_1} & 0 \\ 0 & var_{v_2} \end{bmatrix}, \quad N = \begin{bmatrix} covar_{w_1,v_1} & covar_{w_1,v_2} \\ covar_{w_2,v_1} & covar_{w_2,v_2} \\ covar_{w_3,v_1} & covar_{w_3,v_2} \end{bmatrix}$$

Es decir, Q es la matriz de varianzas de ruido del proceso, R es la matriz de varianzas de ruido del proceso y N es la matriz de covarianzas entre ambos ruidos. Cuantos menores sean los valores de estas varianzas, el observador será más rápido y el filtrado mucho peor.

$$L_{ObKalman} = dlqe(A_d, G_d, C_d, Q, R, N)$$

Una vez obtenida esta matriz, es fácilmente calculable el espacio de estados del observador de Kalman, ya que se hace como un simple observador de orden completo:

$$A_{ObKalman} = A_d - L_{ObKalman} \cdot C_d$$

$$B_{ObKalman} = [B_d - L_{ObKalman} \cdot D_d \quad L_{ObKalman}]$$

$$C_{ObKalman} = I_{numX}$$

$$D_{ObKalman} = [0_{numX,numU} \quad 0_{numX,numY}]$$

$$SYS_{ObKalman} = (A_{ObKalman}, B_{ObKalman}, C_{ObKalman}, D_{ObKalman})$$

4.6.2. Fase de simulación en MATLAB

4.6.2.1. Controlador en MATLAB

En las matrices de ponderación [Tabla 8], en el caso de los errores del sistema se opta por darle siempre

una mayor importancia a la posición angular, ya que se trata del parámetro que provoca más inestabilidad en el Segway, pero conforme se incrementa el período de muestreo se ha requerido reducir este parámetro debido a que se producían inestabilidades. Por otro lado, las ponderaciones de las entradas no cobran gran relevancia, ya que no se han apreciado grandes diferencias en la respuesta del sistema según se pondere más una u otra.

Respecto a los valores de varianzas de los ruidos, se ha considerado que estas nulas, ya que el modelo del Segway no presenta ningún tipo de ruido implícito ni en la medida de sus salidas, y de esta forma se consigue el observador más rápido posible mediante esta técnica.

Tabla 8. Matrices de ponderación y de varianza del ruido (Elaboración propia)

MATRICES DE PONDERACIÓN Y DE VARIANZA DEL RUIDO				
	$T_s = 50\ ms$	$T_s = 60\ ms$	$T_s = 70\ ms$	$Ts = 80\ ms$
Q_v	$\begin{bmatrix} 0.004 & 0 \\ 0 & 0.1 \end{bmatrix}$	$\begin{bmatrix} 0.004 & 0 \\ 0 & 0.02 \end{bmatrix}$	$\begin{bmatrix} 0.004 & 0 \\ 0 & 0.003 \end{bmatrix}$	$\begin{bmatrix} 0.004 & 0 \\ 0 & 0.0015 \end{bmatrix}$
R	$\begin{bmatrix} 0.01 & 0 \\ 0 & 0.005 \end{bmatrix}$	$\begin{bmatrix} 0.01 & 0 \\ 0 & 0.005 \end{bmatrix}$	$\begin{bmatrix} 0.05 & 0 \\ 0 & 0.005 \end{bmatrix}$	$\begin{bmatrix} 0.05 & 0 \\ 0 & 0.005 \end{bmatrix}$
var_w	0	0	0	0
var_v	$1E^{-6}$	$1E^{-6}$	$1E^{-6}$	$1E^{-6}$

El control en simulación se ha desarrollado en el fichero “Segway_ModeloNL_Kalman_LQR_ctrInt.slx” [Ilustración 62] de Simulink donde se aplican todos los conceptos mostrados anteriormente, los cuales han sido programados previamente en el script “param_Segway_Kalman_LQR_ctrInt.mlx” [Script 16] de MATLAB.

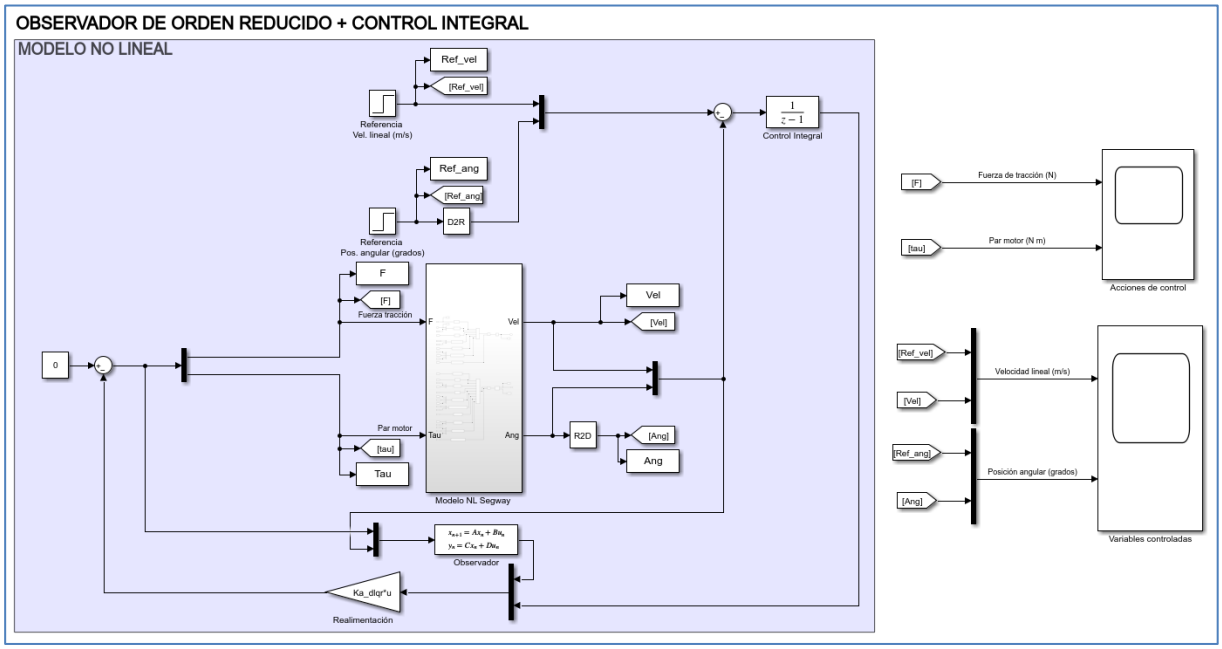

Ilustración 62. Fichero “Segway_ModeloNL_Kalman_LQR_ctrInt.slx” (Elaboración propia)

1. Parámetros y simulación en MATLAB

```
…
if graf_MAT ~= "SI"
    Ts = input("Elija un período de muestreo: ");
    switch Ts
        case 0.05
            Qv = diag([0.004 0.1]);
            R = diag([0.01 0.005]);
        …
    end

    SYSd = c2d(SYS,Ts,'zoh');
    [Adisc,Bdisc,Cdisc,Ddisc] = ssdata(SYSd);
    n = size(Adisc,1);
    m_ent = size(Bdisc,2);
    p = size(Cdisc,1);

    Qx = zeros(n);
    Mxu = zeros(n,m_ent);
    Qa = [Qx zeros(n,p); zeros(p,n) Qv];
    Ra = R;
    Ma = [Mxu; zeros(p,m_ent)];
    Qxu_a = [Qa Ma; Ma' Ra];

    Aa_disc = [Adisc zeros(n,p); -Cdisc eye(p)];
    Ba_disc = [Bdisc; -Ddisc];
    Ca_disc = [Cdisc zeros(p); zeros(p,n+p)];
    Da_disc = [Ddisc; zeros(p)];
    SYSa_disc = ss(Aa_disc,Ba_disc,Ca_disc,Da_disc,Ts);
    disp('Valores propios de Qa >= 0:'), eig(Qa)
    disp('Valores propios de Ra > 0:'), eig(Ra)
    disp('Valores propios de Qxu_a > 0:'), eig(Qxu_a)
    disp('Controlabilidad del sistema SYSa_disc:'), rank(ctrb(SYSa_disc))
    disp('Observabilidad del par (Aa_disc,sqrt(Qa)):'), rank(obsv(Aa_disc,sqrt(Qa)))
    Ka_dlqr = dlqr(Aa_disc,Ba_disc,Qa,Ra,Ma);

    var_W = 0;
    Q = var_W*eye(n); % Matriz de varianza de ruido del proceso
    var_V = 1E-6;
    R = var_V*eye(p); % Matriz de varianza de ruido de la medida
    N = zeros(n,p); % No existe correlación entre ambos ruidos

    % Filtro de Kalman (Observador óptimo de orden completo)
    Gdisc = eye(n);
    [L_obKalman,~,~,~] = dlqe(Adisc,Gdisc,Cdisc,Q,R,N);
    A_obKalman = Adisc-L_obKalman*Cdisc;
    B_obKalman = [Bdisc-L_obKalman*Ddisc L_obKalman];
    C_obKalman = eye(n,n);
    D_obKalman = [zeros(n,m_ent) zeros(n,p)];
    SYS_obKalman = ss(A_obKalman,B_obKalman,C_obKalman,D_obKalman,Ts);

    sim("Segway_ModeloNL_Kalman_LQR_ctrInt.slx");
    …
end
```

2. Resultados

```
…
```

Script 16. Script “param_Segway_Kalman_LQR_ctrInt.mlx”

4.6.2.2. Resultados

Este control ofrece resultados acordes con lo esperado en lo referido a acciones de control [Ilustración 63] y variables controladas [Ilustración 64] para las cuatro simulaciones realizadas, ya que se alcanza la velocidad lineal exigida en, aproximadamente, 5 segundos con una sobreoscilación previa del 5%, y el balanceo del péndulo sufre una inclinación máxima de 6°, más apreciable visualmente que en otros controles, pero previsible, pues este tipo de observador se centra más en la atenuación de ruidos.

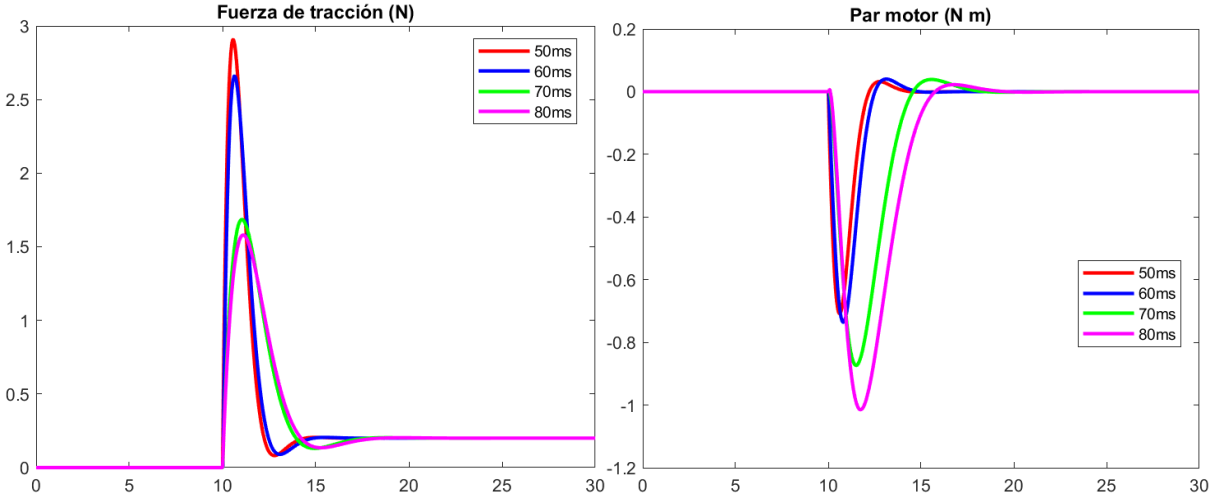

Ilustración 63. Ob.Kalman+LQR+Ctr.Int simulación – Acc. Control: F. tracción y par motor (Elaboración propia)

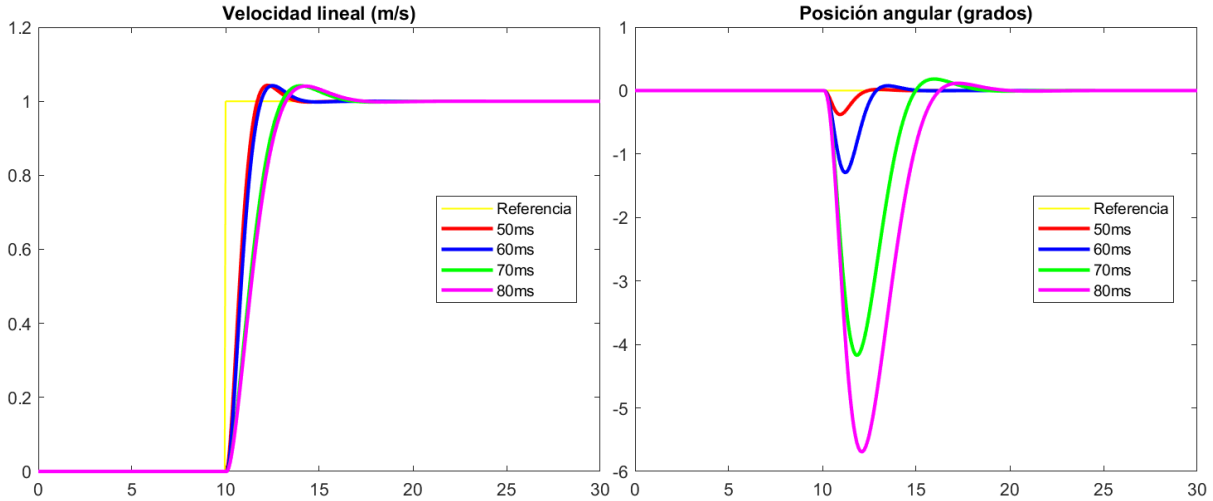

Ilustración 64. Ob.Kalman+LQR+Ctr.Int simulación – Var. Ctr: Vel. Lineal y pos. Angular (Elaboración propia)

4.6.3. Fase experimental en Raspberry Pi y SoMachine

Para la programación del controlador, se ha hecho uso de la herramienta PLC Coder de MATLAB/Simulink.

4.6.3.1. Controlador en SoMachine

Se ha desarrollado un proyecto “Segway_SoMachine_Kalman_LQR_ctrInt_80ms.project” [Script 17] en SoMachine en el que se ha programado el control en lenguaje ST siguiendo el mismo procedimiento que en la fase de simulación. Como puede apreciarse, se ha omitido gran parte del contenido por ser idéntico al del control PID, mostrando únicamente aquellos métodos novedosos del controlador.

```
// DECLARACIÓN DE VARIABLES
PROGRAM POU
VAR
	…
	A_ob: ARRAY[0..8] OF LREAL := [-0.0062930749207400494,0.00021506761585987481,0.13725829918455001,-
0.18978885867645795,0.032689626907611236,-0.59166324308950435,-
0.0075587911540941173,0.080867565642392752,1.032600695151366];
	B_ob: ARRAY[0..11] OF LREAL := [0.031964790878767663,-0.0010701652444498326,-
0.026870241994560316,0.026870241994560316,-0.0046155701596880391,-0.11598858782232246,0.99990011674497525,-
1.0345669698048011E-6,-0.13188425078563246,-1.0345669698090354E-6,0.99991106824375475,1.4109138365825036];
	C_ob: ARRAY[0..8] OF LREAL := [1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0];
	Ka: ARRAY[0..9] OF LREAL := [4.4673915159026611,-1.3351672793432952,0.49368121855798019,-
15.808287025920206,0.93963040051152047,-4.6960090140781086,-0.266579840986845,-0.0067679112183603139,-
0.0065626877611635508,0.41060953461967176];
END_VAR
…
// ARRANQUE DEL CONTROLADOR
	…
ELSIF manual = FALSE THEN
	…
	// Control integral
	…
	// Filtro de Kalman (Observador de orden completo)
	rtb_Observador[0] := (C_ob[6]*Observador_DSTATE[2]) + ((C_ob[3]*Observador_DSTATE[1]) +
(C_ob[0]*Observador_DSTATE[0]));
	xnew[0] := (B_ob[9]*Y_k[1]) + ((B_ob[6]*Y_k[0]) + ((B_ob[3]*U_k[1]) + ((B_ob[0]*U_k[0]) +
((A_ob[6]*Observador_DSTATE[2]) + ((A_ob[3]*Observador_DSTATE[1]) + (A_ob[0]*Observador_DSTATE[0])))));
	rtb_Observador[1] := (C_ob[7]*Observador_DSTATE[2]) + ((C_ob[4]*Observador_DSTATE[1]) +
(C_ob[1]*Observador_DSTATE[0]));
	xnew[1] := (B_ob[10]*Y_k[1]) + ((B_ob[7]*Y_k[0]) + ((B_ob[4]*U_k[1]) + ((B_ob[1]*U_k[0]) +
((A_ob[7]*Observador_DSTATE[2]) + ((A_ob[4]*Observador_DSTATE[1]) + (A_ob[1]*Observador_DSTATE[0])))));
	rtb_Observador[2] := (C_ob[8]*Observador_DSTATE[2]) + ((C_ob[5]*Observador_DSTATE[1]) +
(C_ob[2]*Observador_DSTATE[0]));
	xnew[2] := (B_ob[11]*Y_k[1]) + ((B_ob[8]*Y_k[0]) + ((B_ob[5]*U_k[1]) + ((B_ob[2]*U_k[0]) +
((A_ob[8]*Observador_DSTATE[2]) + ((A_ob[5]*Observador_DSTATE[1]) + (A_ob[2]*Observador_DSTATE[0])))));

	Observador_DSTATE[0] := xnew[0];
	Observador_DSTATE[1] := xnew[1];
	Observador_DSTATE[2] := xnew[2];
	Y_obCom_k[0] := rtb_Observador[0];
	Y_obCom_k[1] := rtb_Observador[1];
	Y_obCom_k[2] := rtb_Observador[2];
	// Realimentación del estado
	U_realim[0] := Y_obCom_k[0];
	U_realim[1] := Y_obCom_k[1];
	U_realim[2] := Y_obCom_k[2];
	U_realim[3] := Y_ctrInt_k[0];
	U_realim[4] := Y_ctrInt_k[1];
	Y_realim[0] := Ka[0]*U_realim[0] + Ka[2]*U_realim[1] + Ka[4]*U_realim[2] + Ka[6]*U_realim[3] +
Ka[8]*U_realim[4];
	Y_realim[1] := Ka[1]*U_realim[0] + Ka[3]*U_realim[1] + Ka[5]*U_realim[2] + Ka[7]*U_realim[3] +
Ka[9]*U_realim[4];
	// Cálculo de las acciones de control
	U_k[0] := 0.0 - Y_realim[0];
	U_k[1] := 0.0 - Y_realim[1];
	…
END_IF
```

Script 17. Proyecto "Segway_SoMachine_Kalman_LQR_ctrInt_80ms.project"

4.6.3.2. Resultados

Los resultados experimentales en acciones de control [Ilustración 65] y variables controladas [Ilustración 66] son ciertamente correctos, ya que, para todos los períodos de muestreo, el comportamiento de la velocidad lineal y de la posición angular es muy similar al de la fase de simulación.

En concreto, en la comparativa entre simulación, fase intermedia y experimentación para Ts = 80 ms [Ilustración 67] [Ilustración 68], apenas se aprecian diferencias entre sí, por lo que puede concluirse que el control óptimo basado en regulador LQR y observador de Kalman ha sido exitoso.

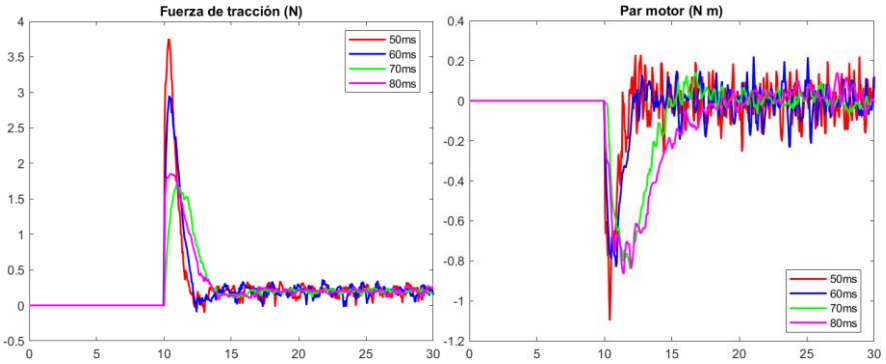

Ilustración 65. Kalman+LQR+Ctr.Int experimentación – Acc. Control: F. tracción y par motor (Elaboración propia)

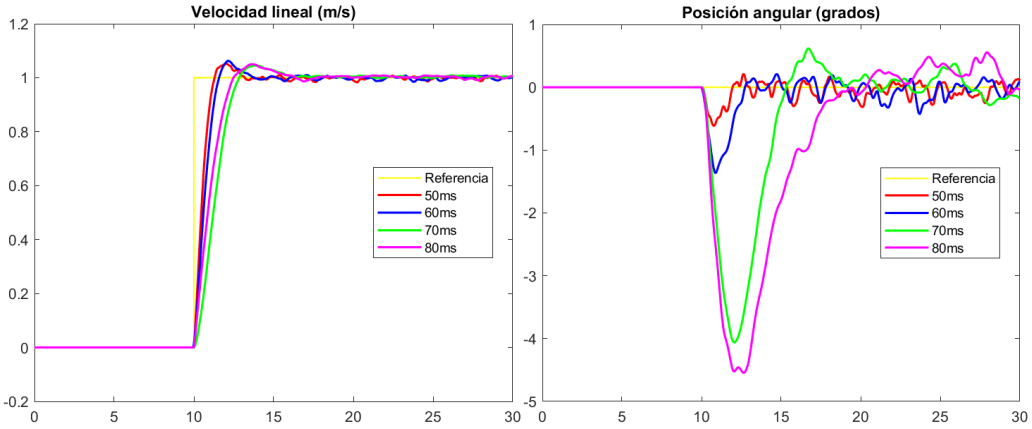

Ilustración 66. Kalman+LQR+Ctr.Int experimentación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

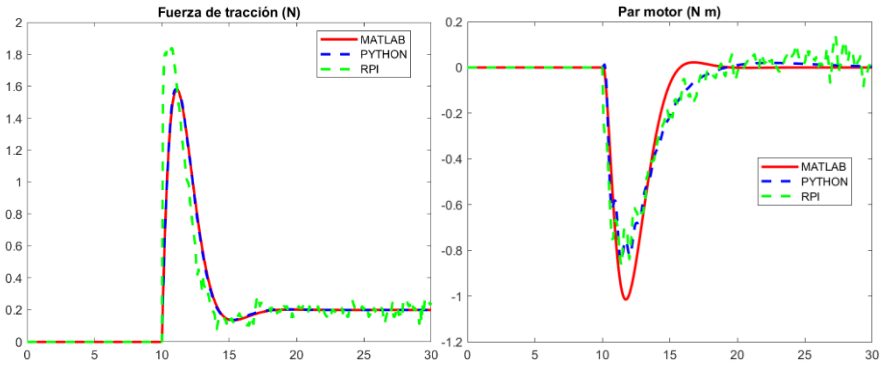

Ilustración 67. Kalman+LQR+Ctr.Int – Comparativa acc. Control entre simulación y experimentación (Elaboración propia)

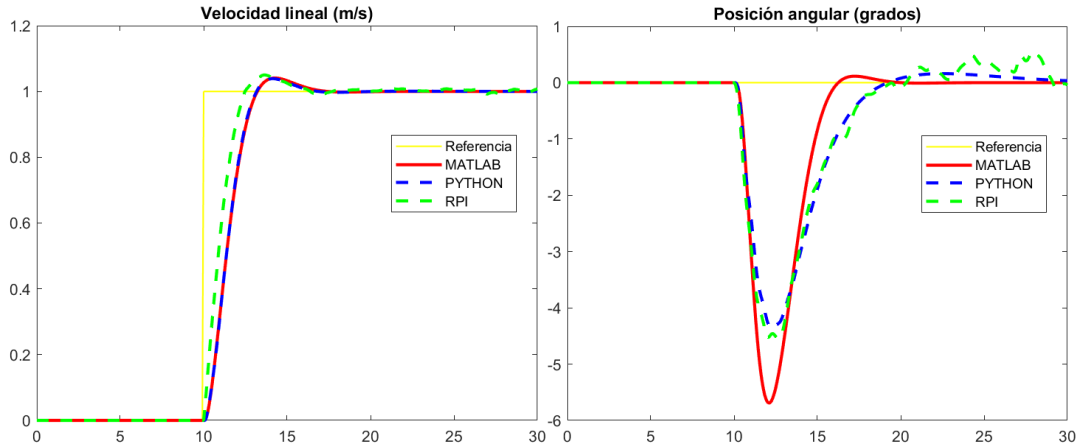

Ilustración 68. Kalman+LQR+Ctr.Int – Comparativa var. Ctr entre simulación y experimentación (Elaboración propia)

Capítulo 5

Control del Segway mediante State-Space Model Predictive Control

Anteriormente se han visto pequeños detalles de los resultados que pueden obtenerse mediante el uso de control óptimo. Sin embargo, esta rama del control engloba muchas otras técnicas que muestran un rendimiento fabuloso, y entre ellas se encuentra el control predictivo, el cual va a ser implementado sobre el Segway en el presente capítulo.

Gran parte del contenido mostrado a continuación se ha basado en la teoría de MPC [24], DMC [25] y SS MPC [26] de la asignatura de Control Industrial Avanzado.

5.1. Antecedentes del control basado en MPC

Los inicios del control óptimo se remontan a la época de 1960 cuando se crea el primer regulador lineal cuadrático (LQR), que garantizaba la estabilidad gracias a horizontes infinitos. Sin embargo, este controlador estaba muy limitado pues no consideraba restricciones ni incertidumbre y sólo podía aplicado cuando se disponía de modelos precisos.

Por ello, no fue hasta finales de 1970 y principios de 1980 que se asentaron las bases del control predictivo basado en modelos (en adelante, MPC) y se empezó a trabajar con índices de coste cuadráticos y con ciertas restricciones en acciones de control y variables controladas. De esta forma, empezó a utilizarse este control en plantas de vapor, unidades de crackeo, columnas de destilación, hornos industriales.

Desde entonces, se han desarrollado multitud de variantes del control predictivo que hacen que siga utilizándose a día de hoy en una gran cantidad de procesos complejos, normalmente no lineales y sobreparametrizados.

5.2. Desarrollo teórico y aplicación sobre el Segway

El MPC es una técnica avanzada de control utilizada para lidiar con procesos multivariable que presentan dinámicas complejas, siendo frecuente el uso de modelos lineales obtenidos mediante identificación, y se basa en la realización de predicciones sobre cómo debe comportarse el proceso, de manera que se pueda alcanzar la referencia sin un gran gasto energético, es decir, se busca minimizar constantemente el índice de coste del proceso de manera, generando así acciones de control según su estado actual. En el caso del Segway, no es necesaria una identificación previa, pues ya se ha obtenido un modelo lineal expresado en espacio de estados, y por este mismo motivo se va a aplicar la técnica State-Space Model Predictive Control (en adelante, SS MPC).

5.2.1. Parámetros del controlador

En esta técnica se requiere fijar previamente una serie de parámetros, entre los que se encuentran:

- El horizonte de predicción, "**p**", que es el número de predicciones realizadas, las cuales se expresan en forma de ecuaciones.
- El horizonte de control, "**c**", que es el número de futuros movimientos de control que se van a realizar dentro del horizonte de predicción, por lo que "c" debe ser siempre menor o igual que "p". Estos dos parámetros y el número de estados, entradas y salidas fijan el tamaño de las matrices del controlador, por lo que debe encontrarse un equilibrio entre estos dos criterios, ya que grandes horizontes pueden suponer un cálculo computacional excesivo, pero a la inversa pueden suponer una mala predicción del futuro.
- El vector de factores de peso de las salidas, "$\boldsymbol{\alpha}$", que realiza un balance en el controlador entre esfuerzo de control y error en la referencia. Cuanto mayor es el valor del factor, menor es el error de posición en régimen permanente, pero la salida puede sobreoscilar más. En el caso MIMO, una variable con un factor menor que las demás se traduce en una menor importancia de control de la misma.
- El vector de factores de peso de los movimientos de control, "$\boldsymbol{\lambda}$", de manera que cuanto mayor es el peso de una variable, menores son sus movimientos de control, es decir, el controlador utiliza menos esta variable para controlar el proceso.

Mediante estos cuatro parámetros y las matrices del modelo lineal discreto en espacio de estados del Segway se construyen las matrices dinámicas "P", "Q", "G" y "H" necesarias para el diseño del controlador.

5.2.2. Diseño del controlador

El método SS MPC asume que no hay perturbaciones o ruidos de medida y que se tiene una medida del vector de estados completo. En el caso del Segway, se cumple la primera condición, pero no la segunda, pues sólo se tiene una medida directa de los dos primeros estados. Por este motivo, se hace uso de un observador de estado de orden reducido que estime el valor del estado desconocido, tal y como se hizo en algunas de las técnicas del anterior capítulo.

Una vez calculada la estimación del estado, es posible realimentar el estado completo hasta la función en la que se aloja el bucle de control, la cual recibe este vector y las referencias del modelo como parámetros de entrada.

Dentro de la función del controlador, se comienza calculando las predicciones de la respuesta libre del sistema "Y_{free}", que es la respuesta del sistema considerando exclusivamente el estado actual y los efectos de las acciones de control pasadas, y asumiendo que no hay cambios en las entradas actuales y futuras:

$$Y_{free_{p\cdot numY,1}} = P_{p\cdot numY,numX} \cdot x_k + Q_{p\cdot numY,numU} \cdot u_{k-1} + bias_k$$

Donde "x_k" es el vector de estados del instante actual, "u_{k-1}" es el vector de acciones de control del instante anterior y "$bias_k$" es la diferencia entre las salidas reales del sistema y las predicciones de las mismas.

Por otra parte, debe calcularse la predicción de la respuesta forzada del sistema "Y_{forced}", que recibe este nombre porque, considerando los efectos de las entradas actuales y futuras sobre los incrementos de las mismas, obtiene la respuesta del mismo a estos cambios cuando se encuentra en régimen permanente:

$$Y_{forced_{p\cdot numY,1}} = G_{p\cdot numY,c\cdot numU} \cdot \Delta u_{c\cdot numU,1}$$

Habiendo calculado la respuesta libre y la forzada, es posible calcular la nueva predicción de la salida "Y_{future}" como la suma de ambas respuestas:

$$Y_{future_{p\cdot numY,1}} = Y_{free} + Y_{forced}$$

Como puede apreciarse, en este vector columna se destinan "p" filas para cada salida, y para el cálculo del nuevo "bias" sólo es necesario el primer valor de estos subvectores, es decir, la primera predicción de cada salida. Estos valores se almacenan en un nuevo vector "Y_{pred}" que en el caso del Segway es:

$$Y_{pred_{numY,1}} = \begin{bmatrix} Y_{future_{1,1}} \\ Y_{future_{p+1,1}} \end{bmatrix}$$

Así, se puede calcular el "bias" actual, que ya se guarda para la siguiente iteración del bucle:

$$bias_{k_{numY}} = Y_k - Y_{pred}$$

Por otro lado, es necesario transformar las referencias en un vector columna "SP", de tamaño $p\cdot num_Y$ x 1, con el objetivo de calcular el vector "Error" de predicción del error del sistema, es decir, la diferencia entre la referencia deseada y la predicción de la respuesta libre:

$$Error_{p\cdot numY} = SP - Y\_free$$

Finalmente, se calculan los futuros cambios o incrementos en las acciones de control:

$$\Delta u_{k_{c\cdot numU,1}} = H \cdot Error$$

Y con ello es posible calcular las nuevas acciones de control aplicadas sobre el modelo del proceso como la suma de las acciones de control del instante anterior y estos incrementos en las mismas:

$$u_k = u_{k-1} + \Delta u_k$$

Esta secuencia de procesos debe repetirse en cada iteración del bucle de control, provocando así constantes cambios en los horizontes de predicción y de control y dando lugar a la optimización del proceso.

5.3. Fase de simulación en MATLAB

5.3.1. Controlador en MATLAB

Respecto a los parámetros del controlador SS MPC [Tabla 9], se ha optado por calcular 90 predicciones y 35 futuros movimientos de control, darle un mayor uso a τ que a F y utilizar un observador más rápido que la dinámica del sistema. Asimismo, para Ts = 50 ms se ha considerado que, para la posición angular, es más importante el error de posición que la sobreoscilación que pueda producirse, contrastando así con los otros períodos de muestreo donde la relevancia de la sobreoscilación es mayor. Esto se ha decidido así porque, a medida que se aumenta el período de muestreo, es más probable que un sistema se vuelva inestable, y más en un control con un cálculo matricial tan denso.

Tabla 9. Parámetros del controlador SS-MPC (Elaboración propia)

PARÁMETROS DEL CONTROLADOR SS-MPC			
	$T_s = 50\ ms$	$T_s = 60\ ms$	$T_s = 70\ ms$
c	35	35	35
p	90	90	90
α	[1.5 3]	[1.5 1]	[1.5 1]
λ	[10 5]	[10 5]	[10 5]
te_{ObRed}	1	1	1

El control en simulación se ha desarrollado en el fichero “Segway_ModeloNL_SSMPC.slx” [Ilustración 69] de Simulink donde se aplican todos los conceptos mostrados anteriormente. Por otra parte, en el script “SSMPC_Segway_Ini.mlx” [Script 18] de MATLAB se ha realizado el cálculo matricial, el diseño del observador de orden reducido y la inicialización de las variables utilizadas, mientras que en la función “SSMPC_Segway_Control.mlx” [Script 19] se ha implementado el bucle de control detallado anteriormente. Se han utilizado las funciones “crea_AlfaM.m” [Script 37], “crea_LambdaM.m” [Script 38], “crea_P.m” [Script 39], “crea_Q.m” [Script 40] y “crea_G.m” [Script 41] para el cálculo matricial, encontrándose todas ellas en el Anexo I.

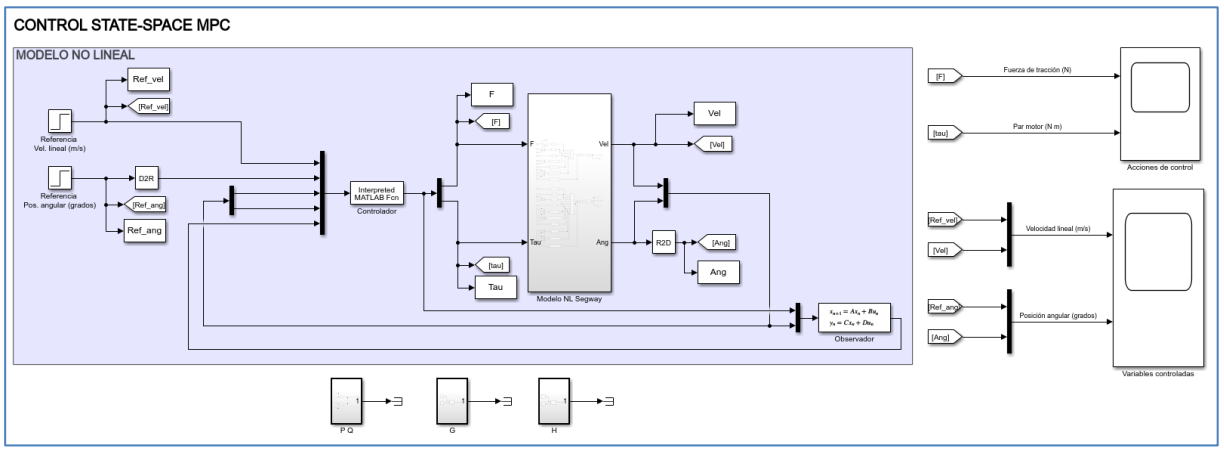

Ilustración 69. Fichero “Segway_ModeloNL_SSMPC.slx” (Elaboración propia)

1. Parámetros y simulación en MATLAB

```
…
%% Variables globales
global p c num_y num_u Y_pred Y_meas Y_free Y_future Error;
global P Q G H Bias_k Bias_vect_k U_k Deltau_k Deltau_util_k SP_vect;

graf_MAT = input("¿Mostrar SÓLO los resultados de los diferentes Ts en MATLAB?: ",'s');
if graf_MAT ~= "SI"
    Ts = input("Elija un período de muestreo: ");
    switch Ts
        case 0.05
            c = 35; p = 90; alfa = [1.5 3]; lambda = [10 5];
        …
    End

    %% Discretización del modelo lineal y núm. X, U, Y
    SYS_d = c2d(SYS, Ts, 'zoh');
    [Adisc, Bdisc, Cdisc, Ddisc] = ssdata(SYS_d);
    num_x = size(Adisc, 1);
    num_u = size(Bdisc, 2);
    num_y = size(Cdisc, 1);

    %% Cálculo matricial
    P = crea_P(Adisc,Cdisc,num_y,num_x,p);
    Q = crea_Q(Adisc,Bdisc,Cdisc,num_y,num_u,p);
    G = crea_G(Q,num_y,num_u,p,c);
    alfaM = crea_AlfaM(alfa,p,num_y);
    lambdaM = crea_LambdaM(lambda,c,num_u);
    H = (G'*alfaM*G+lambdaM)\(G'*alfaM);

    %% Observador de orden reducido
    A11 = Adisc(1:2,1:2);
    A12 = Adisc(1:2,3);
    A21 = Adisc(3,1:2);
    A22 = Adisc(3,3);
    B1 = Bdisc(1:2,:);
    B2 = Bdisc(3,:);
    te_obred = 1;
    sigma_obred = 4/te_obred;
    p_obred = exp(-sigma_obred*Ts);
    L_obred = place(A22', A12', p_obred)';
    A_obred = A22-L_obred*A12;
    B_obred = [B2-L_obred*B1 A21-L_obred*A11+(A22-L_obred*A12)*L_obred];
    C_obred = eye(num_x-num_y);
    D_obred = [zeros(num_x-num_y,num_u) L_obred];
    SYS_obred = ss(A_obred, B_obred, C_obred, D_obred, Ts);

    %% Inicialización de variables
    U_k = zeros(num_u,1);
    Deltau_k = zeros(c*num_u,1);
    Deltau_util_k = zeros(num_u,1);
    SP_vect = zeros(p*num_y,1);
    Y_pred = zeros(num_y,1);
    Y_meas = zeros(num_y,1);
    Y_free = zeros(p*num_y,1);
    Y_future = zeros(p*num_y,1);
    Error = zeros(p*num_y,1);
    Bias_k = zeros(num_y,1);
    Bias_vect_k = zeros(p*num_y,1);

    %% Simulación en MATLAB
    sim("Segway_ModeloNL_SSMPC.slx");
    …
end
```

2. Resultados

```
…
```

Script 18. Script “SSMPC_Segway_Ini.mlx”

```
function salidas = SSMPC_Segway_Control(entradas)
    global p c num_y num_u;
    global Y_pred Y_meas Y_free Y_future Error;
    global P Q G H Bias_k Bias_vect_k;
    global U_k Deltau_k Deltau_util_k SP_vect;

    Y_k = [entradas(3); entradas(4)];
    X_k = [entradas(3); entradas(4); entradas(5)];

    Y_free = P*X_k + Q*U_k + Bias_vect_k;
    Y_future = Y_free + G*Deltau_k;
    for i = 1:1:num_y
        Y_pred(i,1) = Y_future((i-1)*p+1,1);
    end
    Y_meas = Y_k;
    Bias_k = Y_meas - Y_pred;
    for i = 1:1:num_y
        Bias_vect_k((i-1)*p+1:i*p,1) = ones(p,1)*Bias_k(i);
    end
    SP = [entradas(1); entradas(2)];
    for i = 1:1:num_y
        SP_vect((i-1)*p+1:i*p,1) = ones(p,1)*SP(i);
    end
    Error = SP_vect - Y_free;
    Deltau_k = H*Error;
    for j = 1:1:num_u
        Deltau_util_k(j,1) = Deltau_k((j-1)*c+1,1);
    End
    U_k = U_k + Deltau_util_k;
    salidas = U_k;
end
```

Script 19. Función "SSMPC_Segway_Control.mlx"

5.3.2. Resultados

Este control ofrece resultados acordes con lo esperado en lo referido a acciones de control [Ilustración 70] y variables controladas [Ilustración 71] para las tres simulaciones realizadas, ya que se alcanza la velocidad lineal exigida en aproximadamente 5 segundos, con una sobreoscilación previa del 5% en Ts = 70 ms, y el balanceo del péndulo sufre una inclinación máxima de 1.2°, lo cual es casi inapreciable.

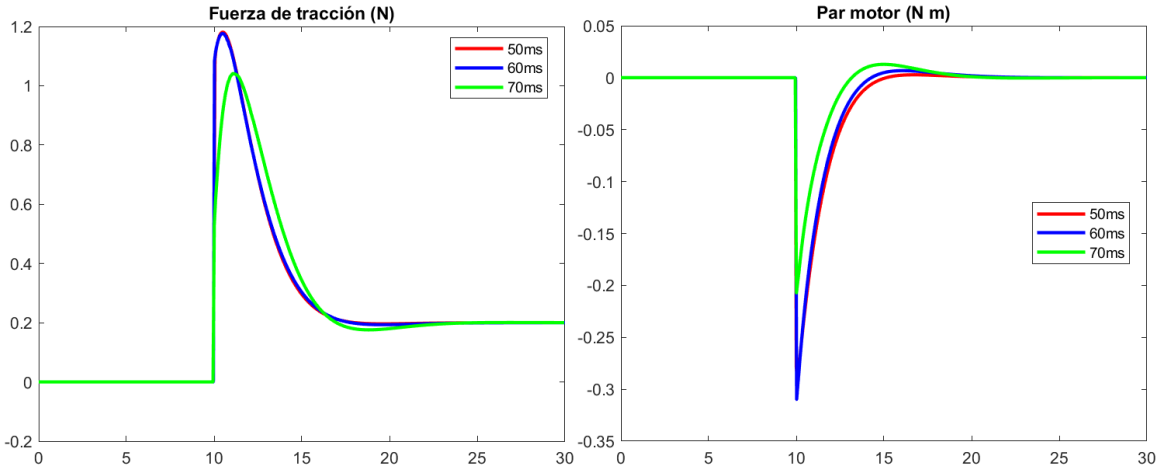

Ilustración 70. SS MPC simulación – Acc. Control: F. tracción y par motor (Elaboración propia)

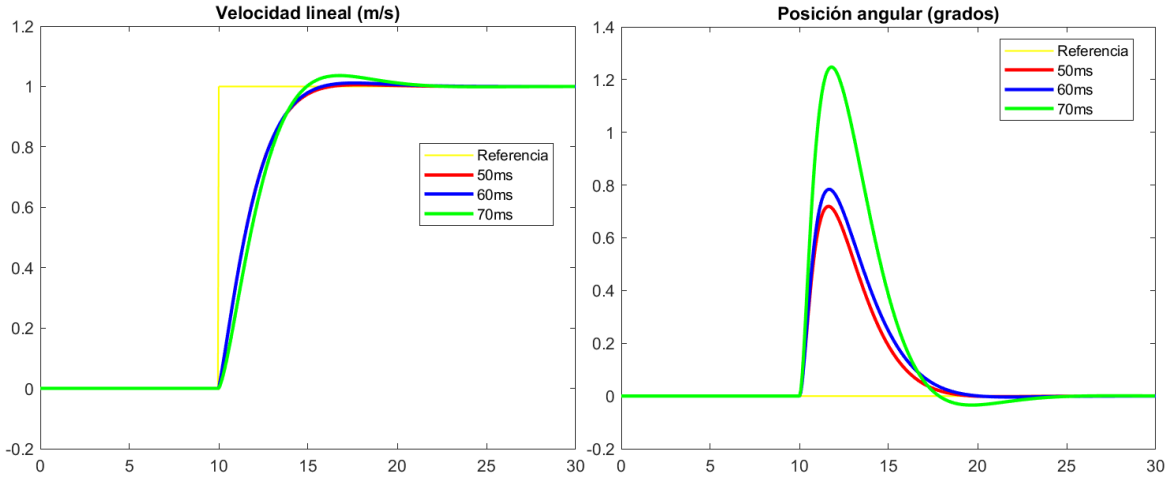

Ilustración 71. SS MPC simulación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

5.4. Fase experimental en Raspberry Pi y SoMachine

Para la programación del controlador, se ha hecho uso de la herramienta PLC Coder de MATLAB/Simulink.

5.4.1. Controlador en SoMachine

Se ha desarrollado un proyecto "Segway_SoMachine_SSMPC_70ms.project" [Script 20] en SoMachine en el que se ha programado el control en lenguaje ST siguiendo el mismo procedimiento que en la fase de simulación. Como puede apreciarse, se ha omitido gran parte del contenido por ser idéntico a los demás, mostrando únicamente aquellos métodos novedosos del controlador.

```
// DECLARACIÓN DE VARIABLES
…
// ARRANQUE DEL CONTROLADOR
	…
ELSIF manual = FALSE THEN
	…
	// Observador de orden reducido
	rtb_Observador := D_ob[2]*Y_k[0] + Observador_DSTATE;
	Observador_DSTATE := A_ob*Observador_DSTATE + B_ob[0]*U_k[0] + B_ob[1]*U_k[1] + B_ob[2]*Y_k[0] +
B_ob[3]*Y_k[1];
	Y_obRed_k := D_ob[3]*Y_k[1] + rtb_Observador;
	// Inserción de las referencias, las salidas y el estado observado en el control SS MPC
	X_k[0] := Y_k[0];
	X_k[1] := Y_k[1];
	X_k[2] := Y_obRed_k;
	// Implementación del bucle de control
	FOR i := 0 TO 179 DO
		MatrixMultiply[i] := 0.0;
		MatrixMultiply[i] := (P[i]*X_k[0]) + MatrixMultiply[i];
		MatrixMultiply[i] := (P[i+180]*X_k[1]) + MatrixMultiply[i];
		MatrixMultiply[i] := (P[i+360]*X_k[2]) + MatrixMultiply[i];
		MatrixMultiply1[i] := 0.0;
		MatrixMultiply1[i] := (Q[i]*U_k[0]) + MatrixMultiply1[i];
		MatrixMultiply1[i] := (Q[i+180]*U_k[1]) + MatrixMultiply1[i];
		Y_free[i] := MatrixMultiply[i] + MatrixMultiply1[i];
		Y_future[i] := 0.0;
		FOR i_0 := 0 TO 69 DO
			Y_future[i] := G[(180*i_0)+i]*DeltaU_k[i_0] + Y_future[i];
		END_FOR;
		Y_future[i] := Y_future[i] + Y_free[i];
	END_FOR;
	Y_pred[0] := Y_future[0];
	Y_pred[1] := Y_future[90];
	Bias_k[0] := Y_k[0] - Y_pred[0];
	Bias_k[1] := Y_k[1] - Y_pred[1];
```

```
        FOR i_1 := 0 TO 89 DO
               Bias_vect_k[i_1] := Bias_k[0];
               SP_vect[i_1] := Ref[0];
               Error[i_1] := SP_vect[i_1] - Y_free[i_1];
        END_FOR;
        FOR i_2 := 90 TO 179 DO
               Bias_vect_k[i_2] := Bias_k[1];
               SP_vect[i_2] := Ref[1];
               Error[i_2] := SP_vect[i_2] - Y_free[i_2];
        END_FOR;
        FOR i_3 := 0 TO 69 DO
               DeltaU_k[i_3] := 0.0;
               FOR i_4 := 0 TO 179 DO
                      DeltaU_k[i_3] := H[(70*i_4)+i_3]*Error[i_4] + DeltaU_k[i_3];
               END_FOR;
        END_FOR;
        DeltaU_util_k[0] := DeltaU_k[0];
        DeltaU_util_k[1] := DeltaU_k[35];
        // Cálculo de las acciones de control
        U_k[0] := U_k[0] + DeltaU_util_k[0];
        U_k[1] := U_k[1] + DeltaU_util_k[1];
        …
END_IF
```

Script 20. Proyecto "Segway_SoMachine_SSMPC_70ms.project"

5.4.2. Resultados

Los resultados experimentales en acciones de control [Ilustración 72] y variables controladas [Ilustración 73] han sido sorprendentemente buenos para todos los períodos de muestreo, ya que se esperaba una inestabilidad mucho mayor debido al gran cálculo computacional requerido en cada iteración del bucle. Sin embargo, el comportamiento de la velocidad lineal y de la posición angular del Segway es fabuloso durante toda la experimentación, independientemente del período de muestreo.

Por si esto fuera poco, en la comparativa entre simulación, fase intermedia y experimentación para Ts = 70 ms [Ilustración 74] [Ilustración 75] los resultados experimentales llegan a mejorar incluso a los de la simulación, por lo que puede concluirse que el control predictivo basado en modelos en espacio de estados ha sido un éxito rotundo.

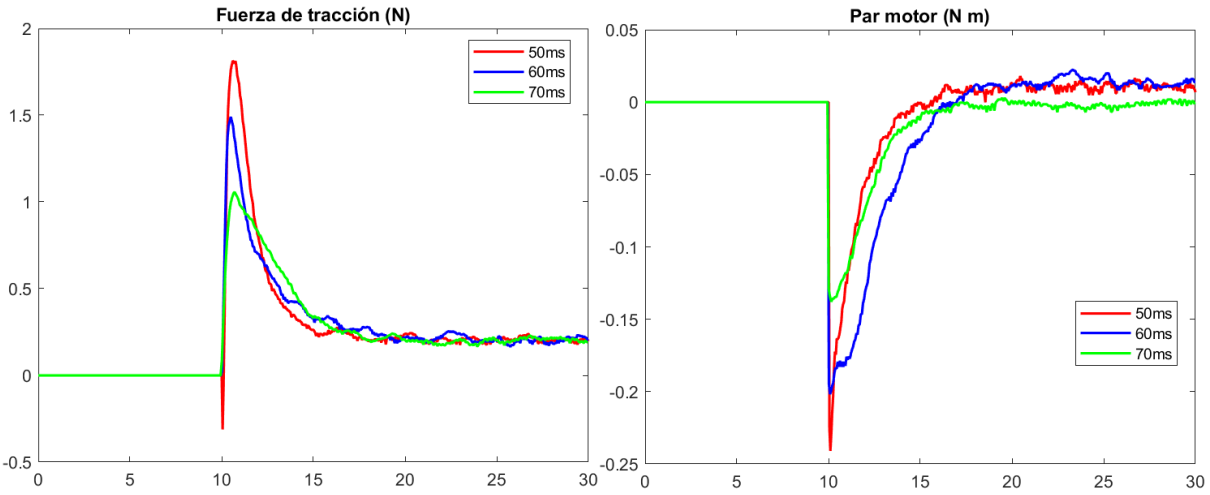

Ilustración 72. SS MPC experimentación – Acc. Control: F. tracción y par motor (Elaboración propia)

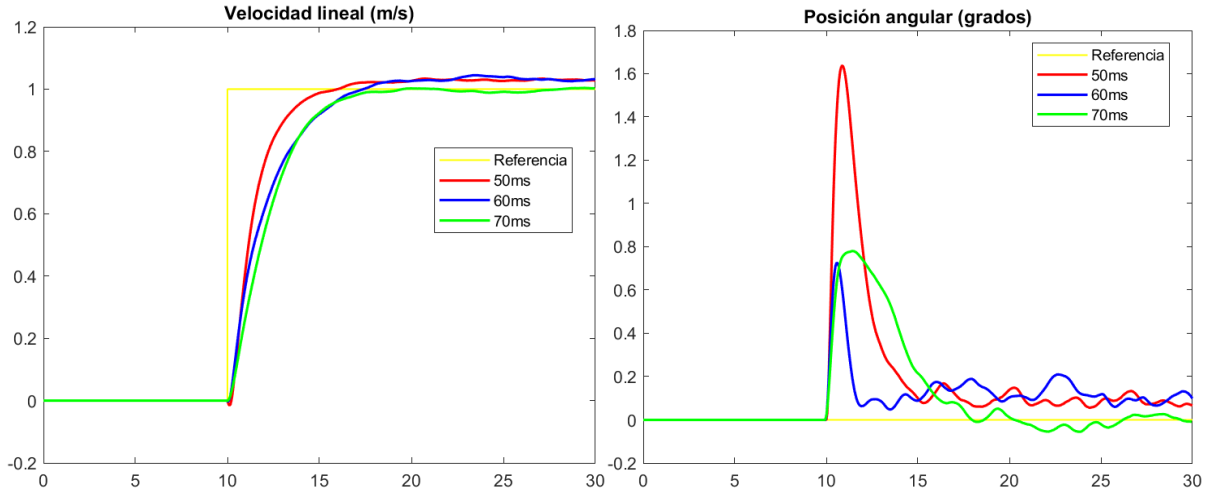

Ilustración 73. SS MPC experimentación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

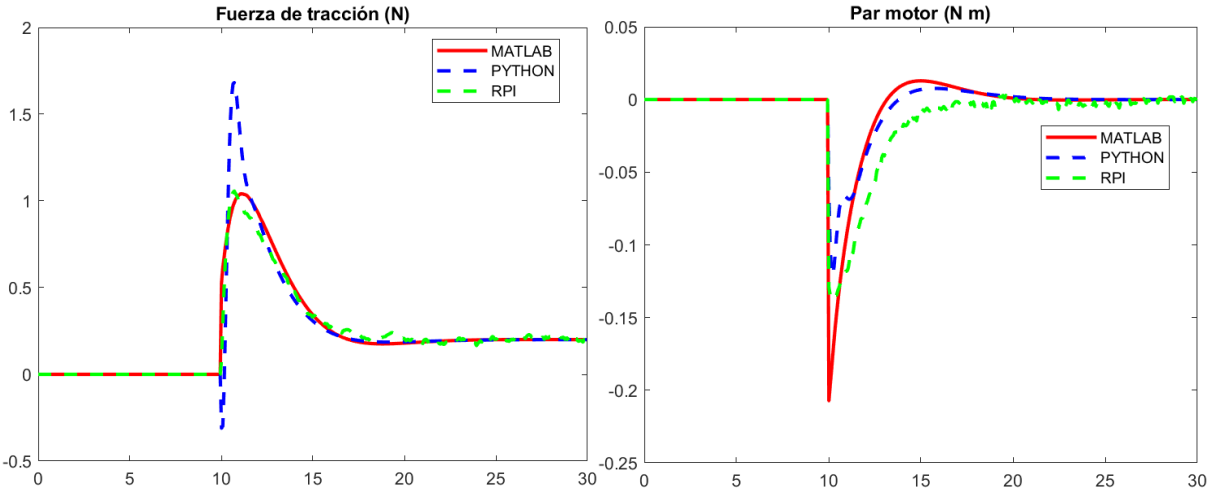

Ilustración 74. SS MPC – Comparativa acc. Control entre simulación y experimentación (Elaboración propia)

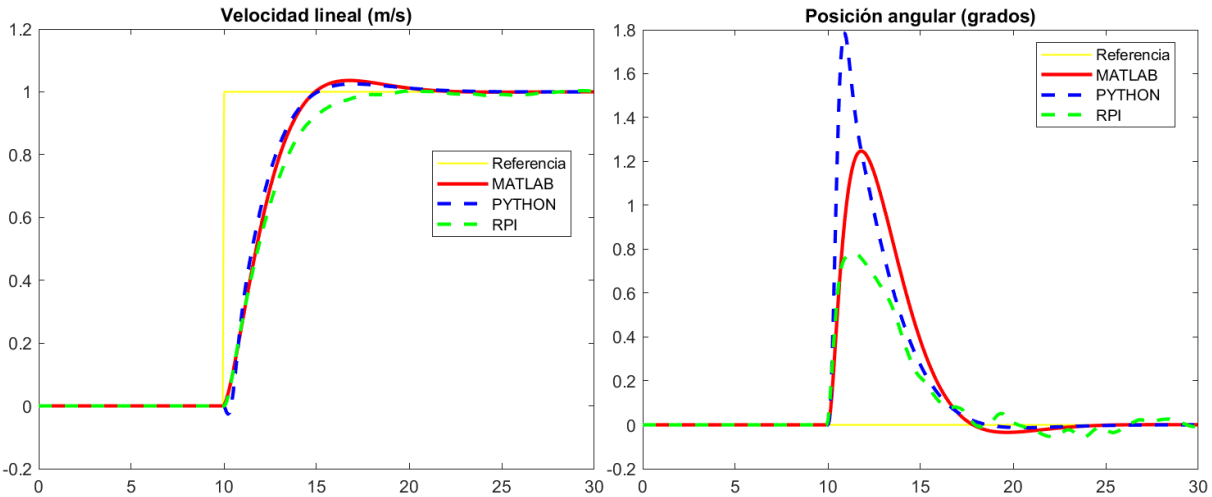

Ilustración 75. SS MPC – Comparativa var. Ctr entre simulación y experimentación (Elaboración propia)

Capítulo 6

Control del Segway mediante control robusto

El control robusto, a diferencia de todos los vistos anteriormente, tiene en cuenta diferentes tipos de incertidumbre que puedan afectar al comportamiento del proceso, por lo que se prioriza la robustez del mismo en detrimento de otros criterios como el tiempo de establecimiento o el error de posición.

Gran parte del contenido mostrado a continuación se ha basado en la teoría de control multivariable durante la asignatura de Identificación y Control de Sistemas Complejos.

6.1. Control $\mathcal{H}_\infty$

6.1.1. Desarrollo teórico y aplicación sobre el Segway

El modelo lineal del Segway dispone de un polo continuo en el plano positivo, es decir, es un proceso inestable, por lo que es necesario realizar una serie de comprobaciones y modificaciones sobre el mismo antes de implementar el control.

6.1.1.1. Análisis SVD estático de controlabilidad

En primer lugar, se realiza un análisis SVD estático de controlabilidad [Ilustración 76] con el objetivo de conocer la capacidad de rapidez y dimensionamiento del modelo de tamaño 2x2, observándose que sólo puede controlarse correctamente un parámetro con una frecuencia de corte de 0.316 rad/s, lo que se traduce en un tiempo de establecimiento de unos 10 segundos, más conservador que el obtenido utilizando los polos. Asimismo, el vector de ganancia relativa RGA y el índice de Niederlinski NI muestran que los emparejamientos F. Tracción – Vel. Lineal y Par motor – Pos. Angular son los correctos, ya que sus ganancias relativas son iguales a 1 y su índice es positivo, por lo que lo establecido en capítulos anteriores es correcto.

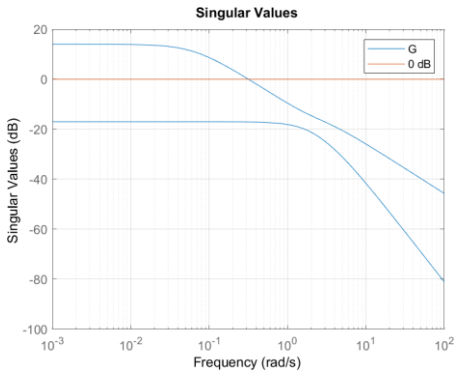

Ilustración 76. Análisis SVD del modelo lineal del Segway (Elaboración propia)

Respecto a la ganancia estática del modelo, su ganancia mínima es menor que 1, por lo que aplicar técnicas de control robusto no garantizaría el seguimiento de la referencia. Por otra parte, el condicionamiento del modelo es igual a 35.3160, valor muy por encima de los 20 fijados como límite y de los 5 recomendables, lo que se traduce en una gran dificultad para invertir la matriz del modelo. No obstante, el condicionamiento del modelo podría reducirse hipotéticamente hasta 1, lo que significa que mediante un buen escalado puede mejorarse notablemente.

6.1.1.2. Escalado del sistema

Considerando incrementos de ±1 N para la fuerza de tracción y ±1.8 Nm para el par motor, introducidos en la matriz diagonal "Eu", y de ±1.7 m/s para la velocidad lineal y ±1° para la posición angular, introducidos en la matriz diagonal "Ey", y haciendo uso de la expresión $\mathrm{SYS_{esc}} = \mathrm{E_y^{-1}} \cdot \mathrm{SYS} \cdot \mathrm{E_u}$ es posible obtener un modelo escalado a partir del original.

Como puede apreciarse en el nuevo análisis SVD [Ilustración 77], ahora es capaz de controlar dos parámetros simultáneamente con una frecuencia de corte de 0.177 rad/s, es decir, en un tiempo de establecimiento de 17.76 segundos. Los emparejamientos según RGA y NI permanecen igual, pero ahora la ganancia mínima es mayor que 1 y el condicionamiento es de 4.9645, mostrando una gran mejoría respecto al inicio. Se ha decidido apurar el condicionamiento hasta un valor de 5, habiendo podido reducirlo más, porque es un rango en el que se puede obtener una mejor respuesta de las salidas acosta de una menor robustez.

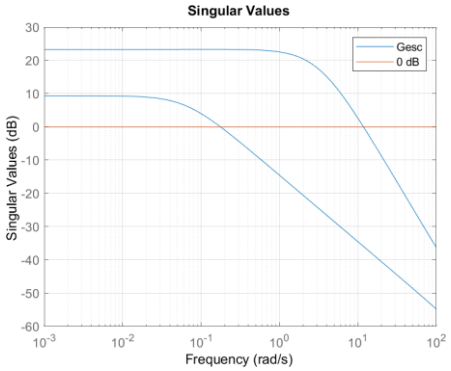

Ilustración 77. Análisis SVD del modelo escalado del Segway (Elaboración propia)

El análisis SVD del modelo original y el diseño del modelo escalado se han realizado en el script "Inicializacion.mlx" [Script 21] de MATLAB, del cual se omite un tercer apartado que se mostrará en apartados posteriores:

```
clearvars;
close all;
1. Modelo lineal en espacio de estados
rga=@(M) M.*inv(M');
q=@(M) max([sum(abs(M)) sum(abs(M'))]);
cotamincond=@(M) q(M)+sqrt(q(M)^2-1);
s = tf('s');
load ModeloL_Segway.mat;
SYS = ModeloL_Segway;
G = zpk(tf(SYS));
Gest = dcgain(G);
```

```
2. Selección de variables a controlar y actuadores
2.1. Análisis SVD de controlabilidad estático
sigma(G,tf(1)), grid on; legend('G','0 dB');
w1_SYS = 0.316; te1_SYS = pi/w1_SYS;
rga_SYS = rga(Gest);
NI_SYS = NiederlinskiIndex(Gest, [1 2]);
[U1,S1,V1] = svd(Gest); S1;
cond_SYS = cond(Gest);
cotamin_SYS = cotamincond(rga_SYS);
2.2. Escalado del sistema
incr_Y1 = 1.7; incr_Y2 = 1.0*(pi/180); incr_U1 = 1.0; incr_U2 = 1.8;
Ey = diag([incr_Y1 incr_Y2]);
Eu = diag([incr_U1 incr_U2]);
SYSesc = Ey\(SYS*Eu);
Gesc = zpk(tf(SYSesc));
Gesc_est = dcgain(Gesc);
sigma(Gesc,tf(1)), grid on; legend('Gesc','0 dB');
w1_0dB = 0.1768754; te1 = pi/w1_0dB;
rga_SYSesc = rga(Gesc_est);
NI_SYSesc = NiederlinskiIndex(Gesc_est, [1 2]);
[Uesc,Sesc,Vesc] = svd(Gesc_est); Sesc;
cond_SYSesc = cond(Gesc_est);
cotamin_SYSesc = cotamincond(rga_SYSesc);
rng('default'); NumSistemasPrueba = 70;
muchosSYS = usample(SYSi,NumSistemasPrueba);
mSYSf = frd(muchosSYS,logspace(-1,3,60));
[SYSr,Info] = ucover(mSYSf,SYS,3);
SYSesc_i = Ey\(SYSr*Eu);
```

Script 21. Script "Inicializacion.mlx", apartados 1 y 2

6.1.1.3. Control basado en la norma H∞

Este tipo de control se basa en minimizar la norma ∞ del sistema en bucle cerrado formado por la planta generalizada ponderada y el controlador calculado, es decir, reducir su pico de respuesta en frecuencia con el objetivo de que no se sobrepasen ciertos límites. Concretamente, se busca que la norma sea inferior a 1 con el objetivo de obtener estabilidad en el rango de frecuencias seleccionado. Por todo ello, es necesario diseñar una planta a partir del modelo escalado del Segway.

Para el diseño de la planta generalizada no ponderada, previo al de la ponderada, se han realizado pruebas para diferentes tipos de estructuras y se ha considerado que la mejor configuración es con las referencias y unas posibles perturbaciones de las acciones de control como entradas exógenas, las acciones de control como entradas manipuladas, los errores entre variables controladas y sus referencias y las acciones de control como variables controladas, y otra vez estos errores como medidas físicas enviadas al controlador. En resumen:

$$\begin{bmatrix} e_1 \\ e_2 \\ u_1 \\ u_2 \\ \hline e_{1_K} \\ e_{2_K} \end{bmatrix} = \begin{bmatrix} I_{2x2} & -SYS_{esc} & -SYS_{esc} \\ 0_{2x2} & 0_{2x2} & I_{2x2} \\ I_{2x2} & -SYS_{esc} & -SYS_{esc} \end{bmatrix} \cdot \begin{bmatrix} ref_1 \\ ref_2 \\ du_1 \\ du_2 \\ \hline u_1 \\ u_2 \end{bmatrix} = GPNW \cdot \begin{bmatrix} ref_1 \\ ref_2 \\ du_1 \\ du_2 \\ \hline u_1 \\ u_2 \end{bmatrix}$$

Una vez obtenida esta planta con rechazo de perturbaciones a la entrada, es posible realizar la ponderación de las entradas y las salidas en las matrices W_{in} y W_{out}, respectivamente, de manera que la planta ponderada pueda expresarse como:

$$GPW = W_{out} \cdot GPNW \cdot W_{in} = \begin{bmatrix} W_{error} & 0 & 0 \\ 0 & W_u & 0 \\ 0 & 0 & I_{2x2} \end{bmatrix} \cdot GPNW \cdot \begin{bmatrix} W_{ref} & 0 & 0 \\ 0 & W_{du} & 0 \\ 0 & 0 & I_{2x2} \end{bmatrix}$$

Es decir, sólo se ponderan las entradas exógenas y las variables controladas. En el caso del Segway, W_{error} es un filtro paso-bajo en el que se minimizan a un 0.1% los errores a frecuencias por debajo de la de corte del modelo escalado [Ilustración 78], W_u es una matriz identidad, es decir, un peso constante que limita las acciones de control a todas las frecuencias, W_{ref} es otra matriz identidad y W_{du} es otro peso constante que limita el tamaño del ruido del proceso a un 0.05.

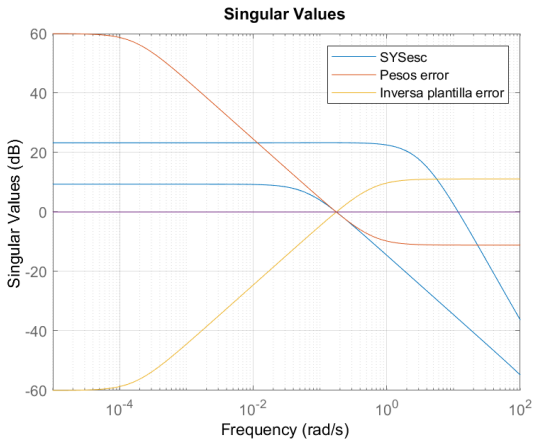

Ilustración 78. Control H∞: Filtro del error aplicado sobre el sistema escalado (Elaboración propia)

Así, aplicando el comando 'hinfsyn' sobre la GPW y estableciendo el número de medidas y de acciones de control se obtiene un controlador expresado en espacio de estados continuo que devuelve una norma ∞ inferior a 1, lo que se traduce en estabilidad robusta en todo el rango de frecuencias fijado previamente, es decir, por debajo de la frecuencia de corte. Finalmente, este controlador es discretizado para poder aplicarlo sobre el modelo no lineal escalado en la fase experimental.

6.1.2. Fase de simulación en MATLAB

6.1.2.1. Controlador en MATLAB

El control en simulación se ha desarrollado en el fichero "Segway_ModeloNL_controlHinf.slx" [Ilustración 79] de Simulink, donde también se aplica el escalado sobre el modelo no lineal y se aplican los conceptos mostrados anteriormente, los cuales han sido programados previamente en el script "param_Segway_controlHinf.mlx" [Script 22] de MATLAB.

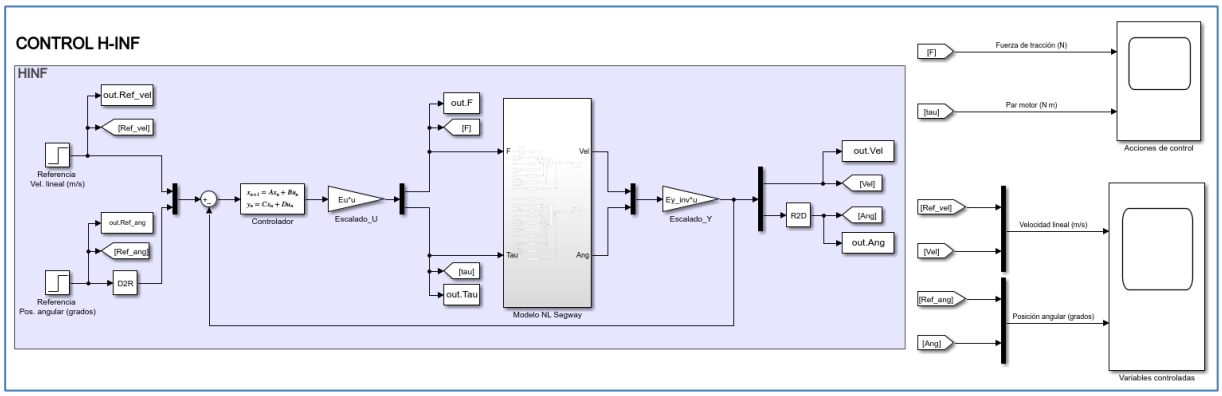

Ilustración 79. Fichero "Segway_ModeloNL_controlHinf.slx" (Elaboración propia)

1. Parámetros y simulación en MATLAB

```
…
incr_Y1 = 1.7; incr_Y2 = 1.0*(pi/180); incr_U1 = 1.0; incr_U2 = 1.8;
Ey = diag([incr_Y1 incr_Y2]);
Ey_inv = inv(Ey);
Eu = diag([incr_U1 incr_U2]);
SYSesc = Ey\(SYS*Eu);
w1_0dB = 0.1768754;
% Planta Generalizada No Ponderada
col1 = [eye(2); zeros(2); eye(2)];
col2 = [-eye(2); zeros(2); -eye(2)]*SYSesc;
col3 = [zeros(2); eye(2); zeros(2)] + [-eye(2); zeros(2); -eye(2)]*SYSesc;
GPNW = minreal([col1 col2 col3]);
GPNW.InputName = {'r1','r2','du1','du2','u1','u2'};
GPNW.OutputName = {'e1','e2','u1','u2','e1_K','e2_K'};
graf_MAT = input("¿Mostrar SÓLO los resultados de los diferentes Ts en MATLAB?: ",'s'); % Responder
SI en caso de desear mostrarlos
if graf_MAT ~= "SI"
    Ts = input("Elija un período de muestreo: ");
    switch Ts
        case 0.05
            errorposiciondeseado = 0.001;
            anchodebandadecontrol = 0.1761; % w1_0dB = 0.1768754
            plantillaerror = makeweight(errorposiciondeseado,anchodebandadecontrol,3.6);
            Werror = (1/plantillaerror)*eye(2);
            Wref = eye(2);
            Wu = eye(2);
            Wdu = 0.05*eye(2);
        …
    end
    % Ponderación de entradas y salidas
    Win = blkdiag(Wref,Wdu,eye(2));
    Wout = blkdiag(Werror,Wu,eye(2));
    % Planta Generalizada Ponderada
    GPW = minreal(Wout*GPNW*Win);
    % Controlador que minimiza la norma infinito
    [K_hinf,CL_hinf,GAM_hinf,INFO_hinf] = hinfsyn(GPW,2,2,'TOLGAM',1e-4);
    GAM_hinf
    Kd_hinf = c2d(K_hinf,Ts,'zoh');

    Hinfsim = sim("Segway_ModeloNL_controlHinf.slx");
    …
```

2. Resultados

```
…
```

Script 22. Script “param_Segway_controlHinf.mlx”

6.1.2.2. Resultados

Los resultados de las acciones de control [Ilustración 80] y de las variables controladas [Ilustración 81] en simulación son ciertamente correctos para los diferentes períodos de muestreo, ya que se alcanza la referencia de la velocidad lineal sin sobreoscilación en aproximadamente 20 segundos, más lento que en todos los demás controles pero lógico teniendo en cuenta que es más relevante la robustez del proceso, y la posición angular del péndulo apenas sufre una inclinación máxima de 2°, lo cual es prácticamente imperceptible, volviendo a su referencia en unos 10 segundos.

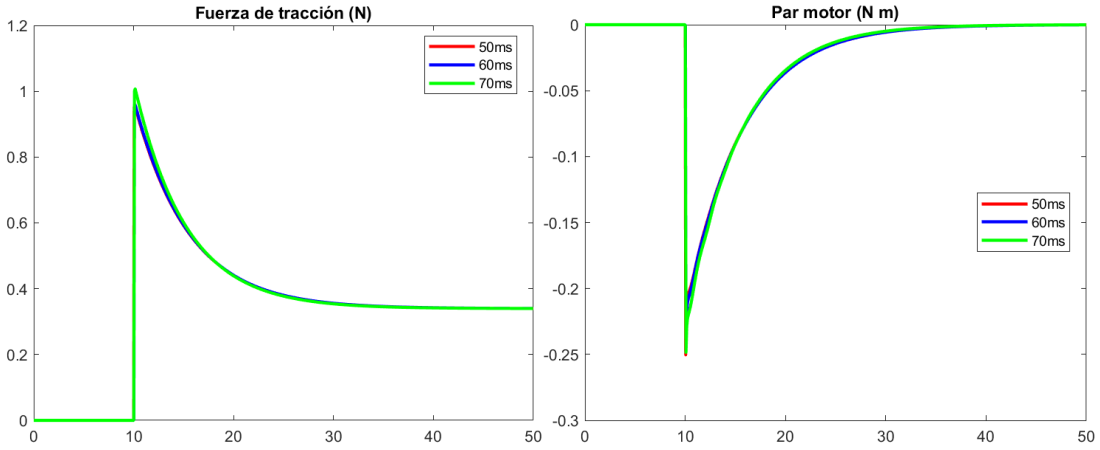

Ilustración 80. H∞ simulación – Acc. Control: F. tracción y par motor (Elaboración propia)

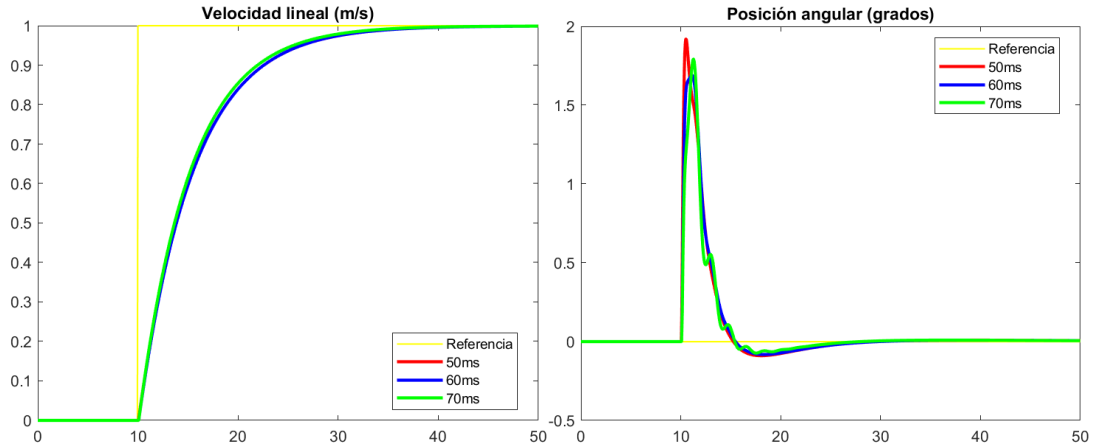

Ilustración 81. H∞ simulación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

6.1.3. Fase experimental en Raspberry Pi y SoMachine

Para la programación del controlador, se ha hecho uso de la herramienta PLC Coder de MATLAB/Simulink.

6.1.3.1. Controlador en SoMachine

Se ha desarrollado un proyecto "Segway_SoMachine_controlHinf_70ms.project" [Script 23] en SoMachine en el que se ha programado el control en lenguaje ST siguiendo el mismo procedimiento que en la fase de simulación. Como puede apreciarse, se ha omitido gran parte del contenido por ser idéntico a los demás, mostrando únicamente aquellos métodos novedosos del controlador.

```
// DECLARACIÓN DE VARIABLES
PROGRAM POU
VAR
	…
	Controlador_DSTATE: ARRAY[0..4] OF LREAL;
	xnew: ARRAY[0..4] OF LREAL;
	rtb_Controlador: ARRAY[0..1] OF LREAL;
	A_ctr: ARRAY[0..24] OF LREAL := [0.99998722349195379,-5.1580706565017813,-7.4754730277819687E-15,-
25.212330708234365,-500.50653374792449,2.2650705784722975E-19,-0.047986323389633226,-6.15666336405648 64E-
15,0.95865941997861837,-6.9053675855998868,-5.9424757712525873E-18,27.374632553602169,0.99998722349211677,-
24.678501083414989,182.46773717521191,7.9317409716222447E-20,-0.03945131893610173,3.2499432035250913E-
17,0.22836780345976943,-5.3193708938474948,-1.666944945273617E-21,0.0042632128468015774,4.7229237588471392E-
17,0.010806469120039169,0.55214516127520008];
	B_ctr: ARRAY[0..9] OF LREAL := [3.0273171784068056E-
20,0.35439909765928318,0.019444320228128621,0.14235107123631013,-
1.5463853561525118,0.0024305400285161843,0.022174126783538764,-6.0935156251238087E-
17,3.2473828608858009,2.2243285302843017];
	C_ctr: ARRAY[0..9] OF LREAL := [-9.440131020873439,-4.4503748687076241,-
7.0241206543823456,0.95589728569768861,183.59701255121226,-24.925647948595426,0.0164096620665753,-
0.06989379920993477,0.053494075575583382,-0.011494235817390258];
END_VAR
// ARRANQUE DEL CONTROLADOR
	…
ELSIF manual = FALSE THEN
	…
	// Cálculo del error entre referencia y salida
	E_k[0] := Ref[0] - Y_k[0];
	E_k[1] := Ref[1] - Y_k[1];
	// Cálculo de las acciones de control
	rtb_Controlador[0] := C_ctr[0]*Controlador_DSTATE[0] + C_ctr[2]*Controlador_DSTATE[1] +
C_ctr[4]*Controlador_DSTATE[2] + C_ctr[6]*Controlador_DSTATE[3] + C_ctr[8]*Controlador_DSTATE[4];
	rtb_Controlador[1] := C_ctr[1]*Controlador_DSTATE[0] + C_ctr[3]*Controlador_DSTATE[1] +
C_ctr[5]*Controlador_DSTATE[2] + C_ctr[7]*Controlador_DSTATE[3] + C_ctr[9]*Controlador_DSTATE[4];

	xnew[0] := A_ctr[0]*Controlador_DSTATE[0] + A_ctr[5]*Controlador_DSTATE[1] +
A_ctr[10]*Controlador_DSTATE[2] + A_ctr[15]*Controlador_DSTATE[3] + A_ctr[20]*Controlador_DSTATE[4] +
B_ctr[0]*E_k[0] + B_ctr[5]*E_k[1];
	xnew[1] := A_ctr[1]*Controlador_DSTATE[0] + A_ctr[6]*Controlador_DSTATE[1] +
A_ctr[11]*Controlador_DSTATE[2] + A_ctr[16]*Controlador_DSTATE[3] + A_ctr[21]*Controlador_DSTATE[4] +
B_ctr[1]*E_k[0] + B_ctr[6]*E_k[1];
	xnew[2] := A_ctr[2]*Controlador_DSTATE[0] + A_ctr[7]*Controlador_DSTATE[1] +
A_ctr[12]*Controlador_DSTATE[2] + A_ctr[17]*Controlador_DSTATE[3] + A_ctr[22]*Controlador_DSTATE[4] +
B_ctr[2]*E_k[0] + B_ctr[7]*E_k[1];
	xnew[3] := A_ctr[3]*Controlador_DSTATE[0] + A_ctr[8]*Controlador_DSTATE[1] +
A_ctr[13]*Controlador_DSTATE[2] + A_ctr[18]*Controlador_DSTATE[3] + A_ctr[23]*Controlador_DSTATE[4] +
B_ctr[3]*E_k[0] + B_ctr[8]*E_k[1];
	xnew[4] := A_ctr[4]*Controlador_DSTATE[0] + A_ctr[9]*Controlador_DSTATE[1] +
A_ctr[14]*Controlador_DSTATE[2] + A_ctr[19]*Controlador_DSTATE[3] + A_ctr[24]*Controlador_DSTATE[4] +
B_ctr[4]*E_k[0] + B_ctr[9]*E_k[1];

	Controlador_DSTATE[0] := xnew[0];
	Controlador_DSTATE[1] := xnew[1];
	Controlador_DSTATE[2] := xnew[2];
	Controlador_DSTATE[3] := xnew[3];
	Controlador_DSTATE[4] := xnew[4];
	// Escalado de las acciones de control
	U_k[0] := rtb_Controlador[0]*1.0;
	U_k[1] := rtb_Controlador[1]*1.8;
	…
END_IF
```

Script 23. Proyecto "Segway_SoMachine_controlHinf_70ms.project"

6.1.3.2. Resultados

Pese a que los resultados en simulación eran ciertamente esperanzadores, lo cierto es que el comportamiento del proceso en la fase experimental ha dejado mucho que desear. Como puede apreciarse, tanto en las acciones de control [Ilustración 82] como en las variables controladas [Ilustración 83] el control llega a funcionar bien durante unos instantes de tiempo, pero posteriormente se inestabiliza y ya es incapaz de recuperar unos valores normales. Este suceso se da especialmente en la posición angular, alcanzándose valores muy altos que se traducirían en una

hecatombe para el Segway. Por ello, puede concluirse que el control H∞ no ha resultado favorecedor.

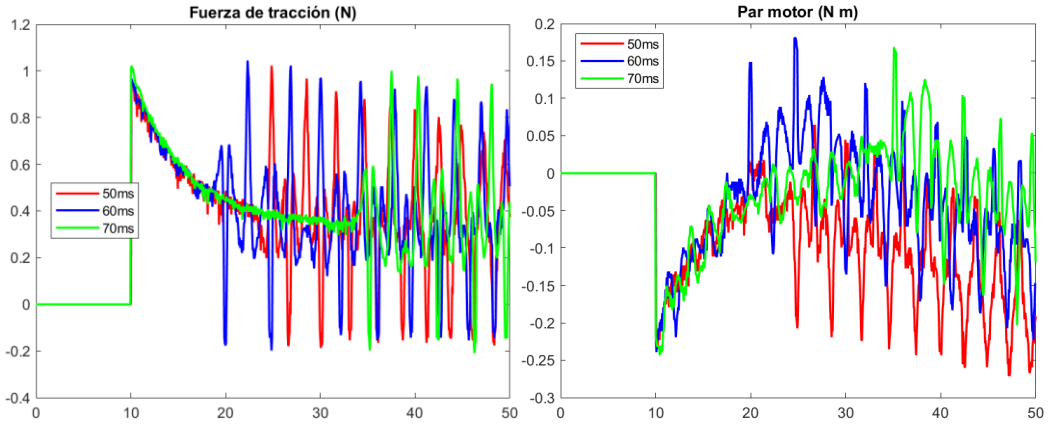

Ilustración 82. H∞ experimentación – Acc. Control: F. tracción y par motor (Elaboración propia)

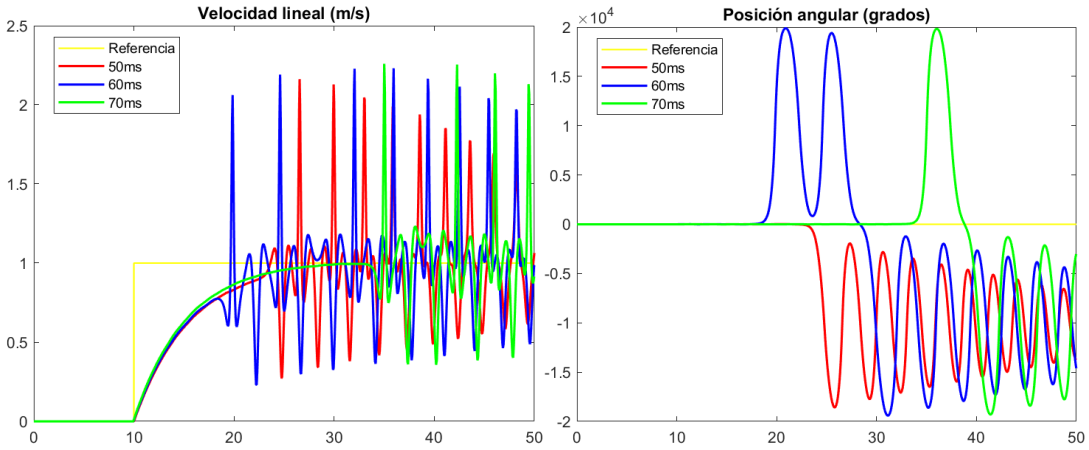

Ilustración 83. H∞ experimentación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

No se conocen con precisión los problemas que han podido llevar al control a esta situación, pero uno de ellos podría ser el tiempo que tarda en ejecutarse el bucle de control en SoMachine [Ilustración 84], siendo éste desde la lectura de variables controladas hasta el envío de las acciones de control. Para ello, se han realizado dos lecturas del tiempo en microsegundos al inicio y al final del bucle mediante la función 'SysTimeGetUs', siendo el tiempo de ejecución la diferencia entre éstas. Así, puede apreciarse que el bucle se ejecuta en tan sólo 0.12 milisegundos, lo cual es mucho más rápido que los 50 ms fijados como tiempo en el que deben leerse los valores de variables controladas. Por otro lado, la tecnología de la RPI no permite trabajar a una velocidad tan elevada, por lo que es imposible que siga este ritmo y sea lo que está propiciando esta inestabilidad tan brusca.

```
Lectura2 0 := SysTimeGetUs(pUsTime2 9168377409);
Tiempo_iter_int 118 := pUsTime2 9168377409 - pUsTime1 9168377291;
Tiempo_iter 0.118 := ULINT_TO_LREAL(Tiempo_iter_int 118)/1000.0;
Tiempo_tot 119 := Tiempo_tot 119 + Tiempo_iter 0.118;
IF (Tiempo_iter 0.118 > Tiempo_iter_max 0.128) AND (j2 968 <= 99) THEN
    Tiempo_iter_max 0.128 := Tiempo_iter 0.118;
ELSIF (Tiempo_iter 0.118 < Tiempo_iter_min 0.117) AND (j2 968 <= 99) THEN
    Tiempo_iter_min 0.117 := Tiempo_iter 0.118;
END_IF
IF j2 968 = 99 THEN
    Tiempo_iter_media 0.12 := Tiempo_tot 119/100.0;
END_IF
j2 968 := j2 968 + 1;
```

Ilustración 84. Tiempo máximo, mínimo y medio de ejecución del bucle de control H∞ (Elaboración propia)

6.2. Control $\mathcal{H}_2$

6.2.1. Desarrollo teórico y aplicación sobre el Segway

Este tipo de control se basa en minimizar la norma 2 del sistema en bucle cerrado formado por la planta generalizada ponderada y el controlador calculado, es decir, reducir la varianza de las variables controladas, ya ponderadas, cuando las entradas de perturbación tienen varianza igual a 1. Por ello, es necesario diseñar una planta a partir del modelo escalado del Segway.

Para el diseño de la planta generalizada no ponderada, previo al de la ponderada, se han realizado pruebas para diferentes tipos de estructuras y se ha considerado que la mejor configuración es con las referencias y unas posibles perturbaciones de las acciones de control como entradas exógenas, las acciones de control como entradas manipuladas, los errores entre variables controladas y sus referencias, las acciones de control y las salidas como variables controladas, y otra vez los errores como medidas físicas enviadas al controlador. En resumen:

$$\begin{bmatrix} e_1 \\ e_2 \\ u_1 \\ u_2 \\ y_1 \\ y_2 \\ \hline e_{1_K} \\ e_{2_K} \end{bmatrix} = \begin{bmatrix} I_{2x2} & -SYS_{esc} & -SYS_{esc} \\ 0_{2x2} & 0_{2x2} & I_{2x2} \\ 0_{2x2} & SYS_{esc} & SYS_{esc} \\ I_{2x2} & -SYS_{esc} & -SYS_{esc} \end{bmatrix} \cdot \begin{bmatrix} ref_1 \\ ref_2 \\ du_1 \\ du_2 \\ \hline u_1 \\ u_2 \end{bmatrix} = GPNW \cdot \begin{bmatrix} ref_1 \\ ref_2 \\ du_1 \\ du_2 \\ \hline u_1 \\ u_2 \end{bmatrix}$$

Una vez obtenida esta planta con rechazo de perturbaciones a la entrada, es posible realizar la ponderación de las entradas y las salidas en las matrices W_{in} y W_{out}, respectivamente, de manera que la planta ponderada pueda expresarse como:

$$GPW = W_{out} \cdot GPNW \cdot W_{in} = \begin{bmatrix} W_{error} & 0 & 0 & 0 \\ 0 & W_u & 0 & 0 \\ 0 & 0 & W_y & 0 \\ 0 & 0 & 0 & I_{2x2} \end{bmatrix} \cdot GPNW \cdot \begin{bmatrix} W_{ref} & 0 & 0 \\ 0 & W_{du} & 0 \\ 0 & 0 & I_{2x2} \end{bmatrix}$$

Es decir, sólo se ponderan las entradas exógenas y las variables controladas. En el caso del Segway, W_{error} es un filtro paso-bajo en el que se minimizan a un 0.1% los errores a frecuencias por debajo de la de corte del modelo escalado, W_u es un peso constante que limita el tamaño de las acciones de control a 0.714, W_y limita las salidas a 0.2, W_{ref} limita las referencias a 0.714 y W_{du} es otro peso constante que limita el tamaño del ruido del proceso a un 0.05.

Así, aplicando el comando 'h2syn' sobre la GPW y estableciendo el número de medidas y de acciones de control, las cuales son 2 y 2, respectivamente, se obtiene un controlador expresado en espacio de estados continuo que devuelve una norma 2 inferior a 1, lo que se traduce en estabilidad para todo el rango de frecuencias fijado previamente, es decir, por debajo de la frecuencia de corte. Finalmente, este controlador es discretizado para poder aplicarlo sobre el modelo no lineal escalado en la fase experimental.

6.2.2. Fase de simulación en MATLAB

6.2.2.1. Controlador en MATLAB

El fichero Simulink de control en simulación basado en esta técnica tiene una estructura idéntica a la del fichero "Segway_ModeloNL_controlHinf.slx" [Ilustración 79]. Asimismo, todos los conceptos

anteriores se han programado en el script “param_Segway_controlH2.mlx” [Script 24].

1. Parámetros y simulación en MATLAB

```
…
% Planta Generalizada No Ponderada
col1 = [eye(2); zeros(4,2); eye(2)];
col2 = [-eye(2); zeros(2); eye(2); -eye(2)]*SYSesc;
col3 = [zeros(2); eye(2); zeros(4,2)] + [-eye(2); zeros(2); eye(2); -eye(2)]*SYSesc;
GPNW = minreal([col1 col2 col3]);
GPNW.InputName = {'r1','r2','du1','du2','u1','u2'};
GPNW.OutputName = {'e1','e2','u1','u2','y1','y2','e1_K','e2_K'};

graf_MAT = input("¿Mostrar SÓLO los resultados de los diferentes Ts en MATLAB?: ",'s');
if graf_MAT ~= "SI"
    Ts = input("Elija un período de muestreo: ");
    switch Ts
        case 0.05
            Wref = 0.714*eye(2);
            Wu = 0.714*eye(2);
            Wy = 0.2*eye(2);
            Wdu = 0.05*eye(2);
            errorposiciondeseado = 0.001;
            anchodebandadecontrol = w1_0dB; % 0.1768754
            GanFerr = (1/errorposiciondeseado);
            tau = GanFerr/anchodebandadecontrol;
            plantillaerror = (GanFerr/(tau*s+1));
            Werror = plantillaerror*eye(2);
        …
    end
    % Ponderación de entradas y salidas
    Win = blkdiag(Wref,Wdu,eye(2));
    Wout = blkdiag(Werror,Wu,Wy,eye(2));
    % Planta Generalizada Ponderada
    GPW = minreal(Wout*GPNW*Win);

    [K_hdos,CL_hdos,GAM_hdos,INFO_hdos] = h2syn(GPW,2,2);
    GAM_hdos
    Kd_hdos = c2d(K_hdos,Ts,'zoh');
    H2sim = sim("Segway_ModeloNL_controlH2.slx");
    …
```

2. Resultados

```
…
```

Script 24. Script “param_Segway_controlH2.mlx”

6.2.2.2. Resultados

Los resultados de las acciones de control [Ilustración 85] y de las variables controladas [Ilustración 86] en simulación son ciertamente correctos para los diferentes períodos de muestreo, ya que se alcanza la referencia de la velocidad lineal con una ligera sobreoscilación del 5% en aproximadamente 15 segundos y la posición angular del péndulo apenas sufre una inclinación máxima de 2°, lo cual es prácticamente imperceptible, volviendo a su referencia en unos 10 segundos.

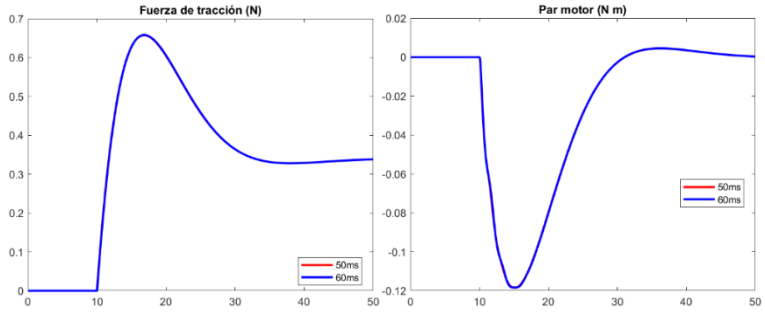

Ilustración 85. H2 simulación – Acc. Control: F. tracción y par motor (Elaboración propia)

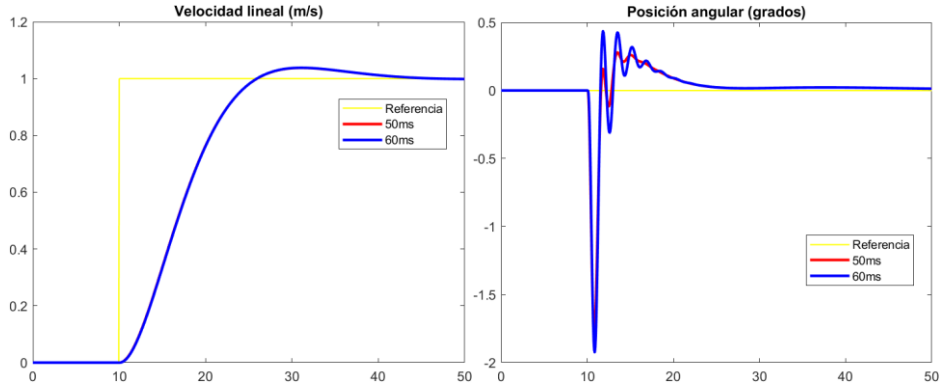

Ilustración 86. H2 simulación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

6.2.3. Fase experimental en Raspberry Pi y SoMachine

Para la programación del controlador, se ha hecho uso de la herramienta PLC Coder de MATLAB/Simulink.

6.2.3.1. Controlador en SoMachine

Se ha desarrollado un proyecto "Segway_SoMachine_controlH2_50ms.project" [Script 25] en SoMachine en el que se ha programado el control en lenguaje ST siguiendo el mismo procedimiento que en la fase de simulación. Como puede apreciarse, se ha omitido gran parte del contenido por ser idéntico a los demás, mostrando únicamente aquellos métodos novedosos del controlador.

```
// DECLARACIÓN DE VARIABLES
PROGRAM POU
VAR
	…
	A_ctr: ARRAY[0..24] OF LREAL := [0.99999115626910606,-3.3908903249984972E-20,0.0054800838876459491,-
0.00029304637484542069,-1.6490104911256457E-5,-1.0084483582608895E-21,0.99999115626910606,-0.00846572096669295,-
0.033896593084487059,-0.00262726729849802,6.0354295861435743E-22,-1.2882825118305639E-
17,0.98503208051421509,0.0031961479919276137,0.00054044766345553324,-8.113611764337775E-20,-3.1854200511577917E-
16,-0.0864574684649484,0.63757702879865108,0.030472896569052926,2.6858072350549045E-18,1.023186389117908E-14,-
0.57540541363549691,-2.3520679805896716,0.51987593019945733];
	B_ctr: ARRAY[0..9] OF LREAL := [0.024999889453200884,-1.6061255897019503E-18,-2.514203133879581E-5,-
6.3870919026049432E-5,-3.5674351737256208E-5,-3.3865498602618372E-
20,0.024999889453200749,0.00093960577021059532,0.0045769858972146658,0.00406707828949739];
	C_ctr: ARRAY[0..9] OF LREAL := [0.4902160835251812,-0.067598643050223062,-0.067659492328908277,-
0.49068399453718653,-1.0975795767150813,0.22534113811296924,-0.26921478591383946,-4.2712308540546342,-
4.2690459647628778,-33.572362708938243];
END_VAR
// ARRANQUE DEL CONTROLADOR
	…
ELSIF manual = FALSE THEN
	…
	// Cálculo del error entre referencia y salida
	E_k[0] := Ref[0] - Y_k[0];
	E_k[1] := Ref[1] - Y_k[1];
	// Cálculo de las acciones de control
	rtb_Controlador[0] := C_ctr[0]*Controlador_DSTATE[0] + C_ctr[2]*Controlador_DSTATE[1] +
C_ctr[4]*Controlador_DSTATE[2] + C_ctr[6]*Controlador_DSTATE[3] + C_ctr[8]*Controlador_DSTATE[4];
	rtb_Controlador[1] := C_ctr[1]*Controlador_DSTATE[0] + C_ctr[3]*Controlador_DSTATE[1] +
C_ctr[5]*Controlador_DSTATE[2] + C_ctr[7]*Controlador_DSTATE[3] + C_ctr[9]*Controlador_DSTATE[4];

	xnew[0] := A_ctr[0]*Controlador_DSTATE[0] + A_ctr[5]*Controlador_DSTATE[1] +
A_ctr[10]*Controlador_DSTATE[2] + A_ctr[15]*Controlador_DSTATE[3] + A_ctr[20]*Controlador_DSTATE[4] +
B_ctr[0]*E_k[0] + B_ctr[5]*E_k[1];
	xnew[1] := A_ctr[1]*Controlador_DSTATE[0] + A_ctr[6]*Controlador_DSTATE[1] +
A_ctr[11]*Controlador_DSTATE[2] + A_ctr[16]*Controlador_DSTATE[3] + A_ctr[21]*Controlador_DSTATE[4] +
B_ctr[1]*E_k[0] + B_ctr[6]*E_k[1];
	xnew[2] := A_ctr[2]*Controlador_DSTATE[0] + A_ctr[7]*Controlador_DSTATE[1] +
A_ctr[12]*Controlador_DSTATE[2] + A_ctr[17]*Controlador_DSTATE[3] + A_ctr[22]*Controlador_DSTATE[4] +
B_ctr[2]*E_k[0] + B_ctr[7]*E_k[1];
```

```
        xnew[3] := A_ctr[3]*Controlador_DSTATE[0] + A_ctr[8]*Controlador_DSTATE[1] +
A_ctr[13]*Controlador_DSTATE[2] + A_ctr[18]*Controlador_DSTATE[3] + A_ctr[23]*Controlador_DSTATE[4] +
B_ctr[3]*E_k[0] + B_ctr[8]*E_k[1];
        xnew[4] := A_ctr[4]*Controlador_DSTATE[0] + A_ctr[9]*Controlador_DSTATE[1] +
A_ctr[14]*Controlador_DSTATE[2] + A_ctr[19]*Controlador_DSTATE[3] + A_ctr[24]*Controlador_DSTATE[4] +
B_ctr[4]*E_k[0] + B_ctr[9]*E_k[1];

        Controlador_DSTATE[0] := xnew[0];
        Controlador_DSTATE[1] := xnew[1];
        Controlador_DSTATE[2] := xnew[2];
        Controlador_DSTATE[3] := xnew[3];
        Controlador_DSTATE[4] := xnew[4];
        // Escalado de las acciones de control
        U_k[0] := rtb_Controlador[0]*1.0;
        U_k[1] := rtb_Controlador[1]*1.8;
        …
END_IF
```

Script 25. Proyecto "Segway_SoMachine_controlH2_50ms.project"

6.2.3.2. Resultados

Pese a que los resultados en simulación eran ciertamente esperanzadores, lo cierto es que el comportamiento de las acciones de control [Ilustración 87] y de las variables controladas [Ilustración 88] del proceso en la fase experimental ha dejado mucho que desear. Como puede apreciarse, la velocidad lineal ha sido fiel a lo que se esperaba según simulación, ya que ha alcanzado su referencia sin ninguna dificultad. Sin embargo, la posición angular no es capaz de estabilizarse en ningún momento, oscilando así constantemente entre los 10° y los -10°. Esto puede ser debido a errores de cuantificación en el par motor, que sigue una trayectoria similar a la simulada, pero con un principio de inestabilidad mucho mayor. Por tanto, puede concluirse que el control H_2 no ha resultado favorecedor.

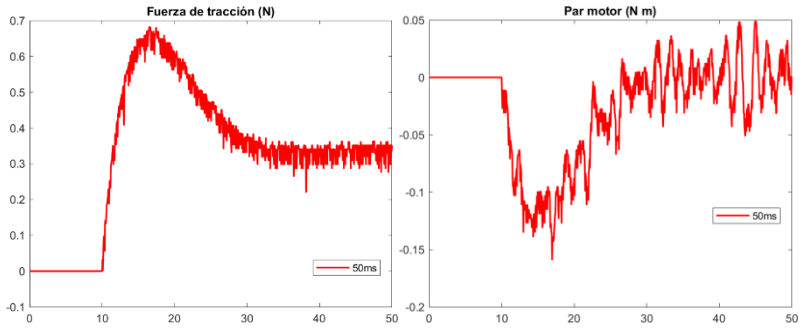

Ilustración 87. H2 experimentación – Acc. Control: F. tracción y par motor (Elaboración propia)

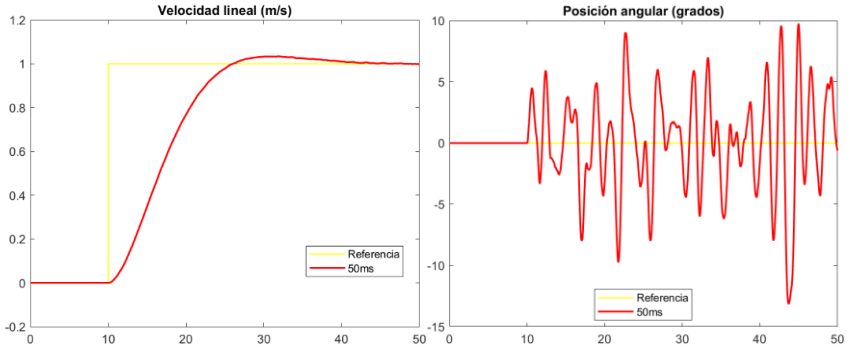

Ilustración 88. H2 experimentación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

Al igual que en el anterior control, uno de los problemas que estén provocando este comportamiento puede ser el tiempo de ejecución del bucle de control [Ilustración 89], el cual presenta un valor medio

de 0.12 milisegundos, muy por debajo de los 50 ms de período de muestreo. Este suceso, unido a las limitaciones que presenta la RPI, puede ser uno de los detonantes de esta conducta.

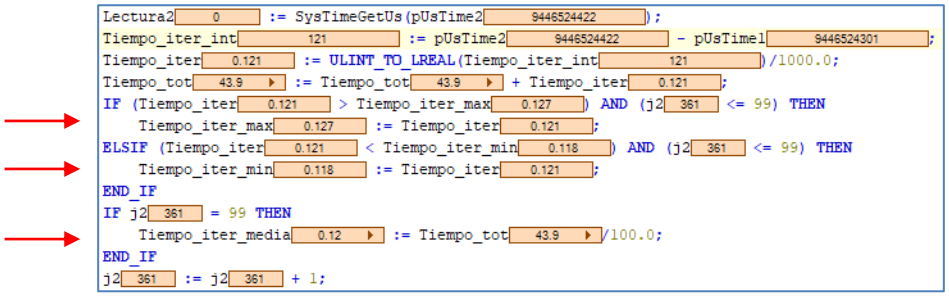

Ilustración 89. Tiempo máximo, mínimo y medio de ejecución del bucle de control H2 (Elaboración propia)

6.3. Control Mixed Sensitivity

6.3.1. Desarrollo teórico y aplicación sobre el Segway

El control basado en la técnica Mixed Sensitivity es un caso particular del control H∞, ya que trata de minimizar la norma ∞ del bucle de control estándar de un grado de libertad de un modelo que tiene cierta incertidumbre, es decir, y = G·u = G·K·(ref-y), no siendo necesario diseñar una planta generalizada ponderada. No obstante, con el objetivo de seguir hablando en los mismos términos, puede diseñarse una planta en la que no es necesaria la ponderación de las entradas:

$$\begin{bmatrix} e_{1_w} \\ e_{2_w} \\ u_{1_w} \\ u_{2_w} \\ y_{1_w} \\ y_{2_w} \\ \hline e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} W_1 & 0 & 0 & 0 \\ 0 & W_2 & 0 & 0 \\ 0 & 0 & W_3 & 0 \\ 0 & 0 & 0 & I_{2x2} \end{bmatrix} \cdot \begin{bmatrix} I_{2x2} & -SYS_{esc} \\ 0_{2x2} & I_{2x2} \\ 0_{2x2} & SYS_{esc} \\ I_{2x2} & -SYS_{esc} \end{bmatrix} \cdot \begin{bmatrix} ref_1 \\ ref_2 \\ \hline u_1 \\ u_2 \end{bmatrix}$$

Como puede apreciarse, se está considerando un sistema con 1 entrada, las referencias, y 3 salidas, los errores, las acciones de control y las variables controladas, teniendo cada uno de ellos sus filtros W_1, W_2 y W_3, respectivamente. De esta forma, W_1 tiene relación con las prestaciones nominales del sistema, W_2 con la incertidumbre aditiva y W_3 con la incertidumbre multiplicativa, y en el caso de que se obtenga un controlador que, formando un sistema en bucle cerrado con el modelo, devuelva una norma ∞ menor que 1, se garantizan prestaciones nominales y estabilidad robusta, pero no prestaciones robustas.

En el caso del Segway, para W_1 se opta por un filtro paso-bajo con un error de posición del 0.1% a frecuencias bajas y ancho de banda igual a la frecuencia de corte del modelo escalado, mientras que W_2 y W_3 son filtros constantes cuyos valores varían según la opción escogida [Tabla 10].

Tabla 10. Control Mixed Sensitivity: Filtros W_2 y W_3 (Elaboración propia)

FILTROS DEL CONTROL MIXED-SENSITIVITY (Ts = 50 ms)				
	Opción 1	***Opción 2***	***Opción 3***	***Opción 4***
W_2	$0.6612 \cdot I_{2x2}$	$0.95 \cdot I_{2x2}$	$0.95 \cdot I_{2x2}$	$0.97 \cdot I_{2x2}$
W_3	$0.66 \cdot I_{2x2}$	$0.287 \cdot I_{2x2}$	$0.315 \cdot I_{2x2}$	$0.248 \cdot I_{2x2}$

De esta forma, aplicando el comando 'mixsyn' sobre el sistema escalado, SYS_{esc}, e insertando los respectivos filtros de los errores, las acciones de control y las variables controladas del sistema se obtiene un controlador expresado en espacio de estados continuo que devuelve una norma ∞ inferior a 1, por lo que se aseguran las prestaciones nominales y la estabilidad robusta en todo el rango de frecuencias fijado previamente. Finalmente, este controlador es discretizado para poder aplicarlo sobre el modelo no lineal escalado en la fase experimental, es decir, convertido a lenguaje ST.

6.3.2. Fase de simulación en MATLAB

6.3.2.1. Controlador en MATLAB

El fichero Simulink de control en simulación basado en esta técnica tiene una estructura idéntica a la del fichero "Segway_ModeloNL_controlHinf.slx" [Ilustración 79]. Asimismo, todos los conceptos anteriores se han programado en el script "param_Segway_controlMixsyn.mlx" [Script 26].

1. Parámetros y simulación en MATLAB

```
…
incr_Y1 = 1.7; incr_Y2 = 1.0*(pi/180); incr_U1 = 1.0; incr_U2 = 1.8;
Ey = diag([incr_Y1 incr_Y2]);
Ey_inv = inv(Ey);
Eu = diag([incr_U1 incr_U2]);
SYSesc = Ey\(SYS*Eu);
w1_0dB = 0.1768754;

graf_MAT = input("¿Mostrar SÓLO los resultados de los diferentes Ts en MATLAB?: ",'s');
if graf_MAT ~= "SI"
    Ts = 0.05;
    opcion = input("Elige una opción: ");
    switch opcion
        case 1
            Wu = 0.6612*eye(2);
            Wy = 0.66*eye(2);
            errorposiciondeseado = 0.001;
            anchodebandadecontrol = w1_0dB; % 0.1768754
            plantillaerror = makeweight(errorposiciondeseado,anchodebandadecontrol,3.5);
            Werror = (1/plantillaerror)*eye(2);
        …
    end

    if (opcion == 1) || (opcion == 2)
        [K_mxs,CL_mxs,GAM_mxs,INFO_mxs] = mixsyn(SYSesc,Werror,Wu,Wy);
    elseif (opcion == 3) || (opcion == 4)
        [K_mxs,CL_mxs,GAM_mxs,INFO_mxs] = mixsyn(SYSesc,Werror,Wu,Wy,'TOLGAM',1e-4);
    end
    GAM_mxs
    Kd_mxs = c2d(K_mxs,Ts,'zoh');

    Mixsynsim = sim("Segway_ModeloNL_controlMixsyn.slx");
    …
```

2. Resultados

```
…
```

Script 26. Script "param_Segway_controlMixsyn.mlx"

6.3.2.2. Resultados

Los resultados de las acciones de control [Ilustración 90] y de las variables controladas [Ilustración 91] en simulación son ciertamente correctos para los diferentes períodos de muestreo, ya que se alcanza la referencia de la velocidad lineal sin sobreoscilación en aproximadamente 15 segundos, más lento que en todos los demás controles pero lógico teniendo en cuenta que es más relevante la robustez del proceso, y la posición angular del péndulo apenas sufre una inclinación máxima de 3°, lo cual es

prácticamente imperceptible, volviendo a su referencia en unos 10 segundos.

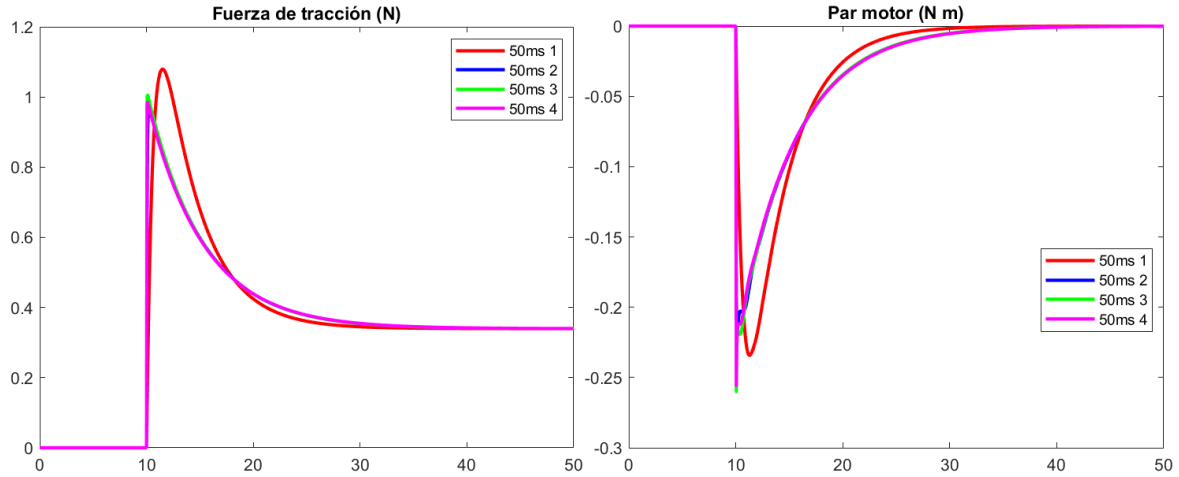

Ilustración 90. Mixed-Sen simulación – Acc. Control: F. tracción y par motor (Elaboración propia)

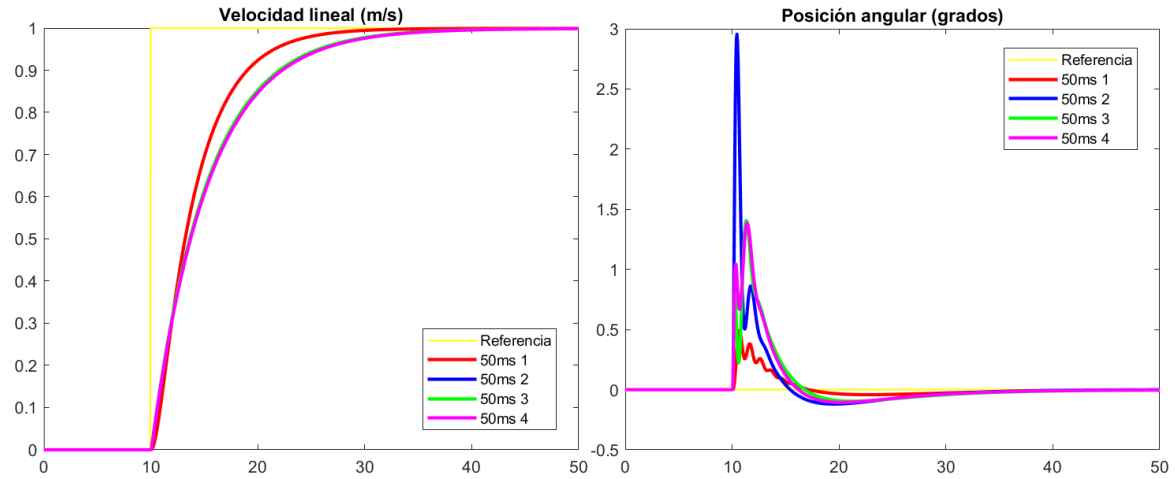

Ilustración 91. Mixed-Sen simulación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

6.3.3. Fase experimental en Raspberry Pi y SoMachine

Para la programación del controlador, se ha hecho uso de la herramienta PLC Coder de MATLAB/Simulink.

6.3.3.1. Controlador en SoMachine

Se ha desarrollado un proyecto "Segway_SoMachine_controlMixsyn_50ms4.project" [Script 27] en SoMachine en el que se ha programado el control en lenguaje ST siguiendo el mismo procedimiento que en la fase de simulación. Como puede apreciarse, se ha omitido gran parte del contenido por ser idéntico a los demás, mostrando únicamente aquellos métodos novedosos del controlador.

```
// DECLARACIÓN DE VARIABLES
PROGRAM POU
VAR
	…
	A_ctr: ARRAY[0..24] OF LREAL := [0.99999152067162334,8.2494781898942581E-7,-
3.8895950049506638,102.34269070514904,-101.82080255030863,1.8885692171205331E-
7,0.99995408097988447,83.023098040526747,-1668.6047904807408,5933.331307687763,-5.3359834391090839E-
9,1.0579442209283193E-6,-1.3458297311638696,47.147467796200793,-167.63966719267626,-1.6547748119333426E-
10,3.2911989383783035E-8,-0.056560001048697031,1.9344460554084333,-5.0146904041787064,-1.0725327573166327E-
12,7.5408035894777032E-10,0.0092988536689302143,-0.30684658724212072,0.34097183213035409];
	B_ctr: ARRAY[0..9] OF LREAL := [-1.9792608464758106E-
11,0.0071428308554949737,0.24194793112554161,0.24469106896129947,-
0.66344251233552876,0.0017857067633799328,1.3761938197082963E-10,0.022809000124016807,-0.604259413450249,-
7.8588633350056361];
	C_ctr: ARRAY[0..9] OF LREAL := [-205.78866460171886,22.443255952649071,9302.638643973225,-
1259.0061753751997,-262.82482529690822,35.57182088551626,-8.1713501237169073,1.143395401199216,-
0.18138974461153781,0.053577512043033813];
END_VAR

// ARRANQUE DEL CONTROLADOR
	…
ELSIF manual = FALSE THEN
	…
	// Cálculo del error entre referencia y salida
	E_k[0] := Ref[0] - Y_k[0];
	E_k[1] := Ref[1] - Y_k[1];

	// Cálculo de las acciones de control
	rtb_Controlador[0] := C_ctr[0]*Controlador_DSTATE[0] + C_ctr[2]*Controlador_DSTATE[1] +
C_ctr[4]*Controlador_DSTATE[2] + C_ctr[6]*Controlador_DSTATE[3] + C_ctr[8]*Controlador_DSTATE[4];
	rtb_Controlador[1] := C_ctr[1]*Controlador_DSTATE[0] + C_ctr[3]*Controlador_DSTATE[1] +
C_ctr[5]*Controlador_DSTATE[2] + C_ctr[7]*Controlador_DSTATE[3] + C_ctr[9]*Controlador_DSTATE[4];

	xnew[0] := A_ctr[0]*Controlador_DSTATE[0] + A_ctr[5]*Controlador_DSTATE[1] +
A_ctr[10]*Controlador_DSTATE[2] + A_ctr[15]*Controlador_DSTATE[3] + A_ctr[20]*Controlador_DSTATE[4] +
B_ctr[0]*E_k[0] + B_ctr[5]*E_k[1];
	xnew[1] := A_ctr[1]*Controlador_DSTATE[0] + A_ctr[6]*Controlador_DSTATE[1] +
A_ctr[11]*Controlador_DSTATE[2] + A_ctr[16]*Controlador_DSTATE[3] + A_ctr[21]*Controlador_DSTATE[4] +
B_ctr[1]*E_k[0] + B_ctr[6]*E_k[1];
	xnew[2] := A_ctr[2]*Controlador_DSTATE[0] + A_ctr[7]*Controlador_DSTATE[1] +
A_ctr[12]*Controlador_DSTATE[2] + A_ctr[17]*Controlador_DSTATE[3] + A_ctr[22]*Controlador_DSTATE[4] +
B_ctr[2]*E_k[0] + B_ctr[7]*E_k[1];
	xnew[3] := A_ctr[3]*Controlador_DSTATE[0] + A_ctr[8]*Controlador_DSTATE[1] +
A_ctr[13]*Controlador_DSTATE[2] + A_ctr[18]*Controlador_DSTATE[3] + A_ctr[23]*Controlador_DSTATE[4] +
B_ctr[3]*E_k[0] + B_ctr[8]*E_k[1];
	xnew[4] := A_ctr[4]*Controlador_DSTATE[0] + A_ctr[9]*Controlador_DSTATE[1] +
A_ctr[14]*Controlador_DSTATE[2] + A_ctr[19]*Controlador_DSTATE[3] + A_ctr[24]*Controlador_DSTATE[4] +
B_ctr[4]*E_k[0] + B_ctr[9]*E_k[1];

	Controlador_DSTATE[0] := xnew[0];
	Controlador_DSTATE[1] := xnew[1];
	Controlador_DSTATE[2] := xnew[2];
	Controlador_DSTATE[3] := xnew[3];
	Controlador_DSTATE[4] := xnew[4];

	// Escalado de las acciones de control
	U_k[0] := rtb_Controlador[0]*1.0;
	U_k[1] := rtb_Controlador[1]*1.8;
	…
END_IF
```

Script 27. Proyecto “Segway_SoMachine_controlMixsyn_50ms4.project”

6.3.3.2. Resultados

Pese a que los resultados en simulación eran ciertamente esperanzadores, lo cierto es que el comportamiento de las acciones de control [Ilustración 92] y de las variables controladas [Ilustración 93] del proceso en la fase experimental ha dejado mucho que desear. Como puede apreciarse, la velocidad lineal ha sido fiel a lo que se esperaba según simulación, ya que ha alcanzado su referencia sin ninguna dificultad. Sin embargo, la posición angular no es capaz de estabilizarse en ningún momento, oscilando así constantemente entre los 10° y los -10°. Esto puede ser debido a errores de

cuantificación en el par motor, que sigue una trayectoria similar a la simulada, pero con un principio de inestabilidad mucho mayor. Por tanto, puede concluirse que el control Mixed-Sensitivity no ha resultado favorecedor.

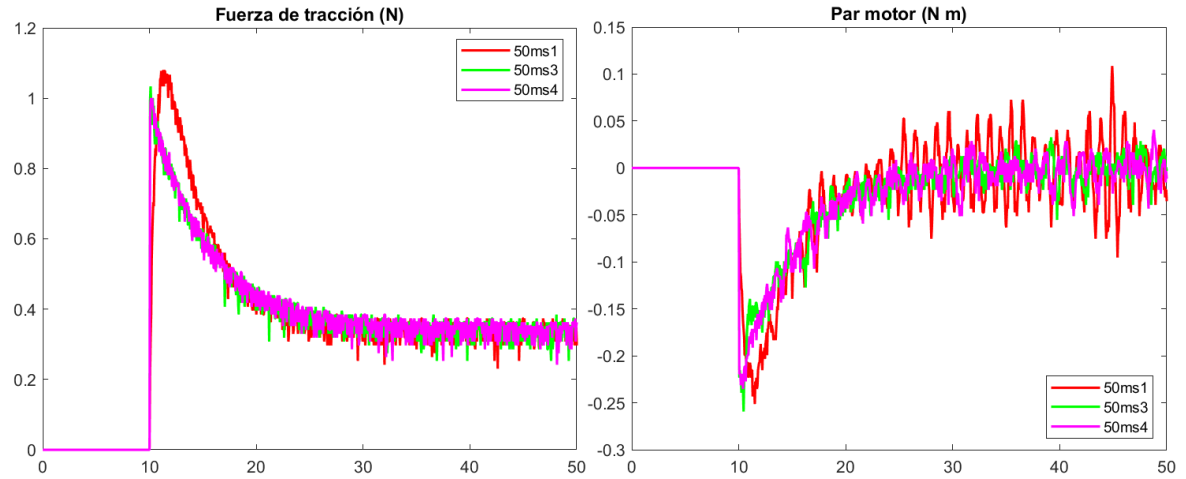

Ilustración 92. Mixed-Sen experimentación – Acc. Control: F. tracción y par motor (Elaboración propia)

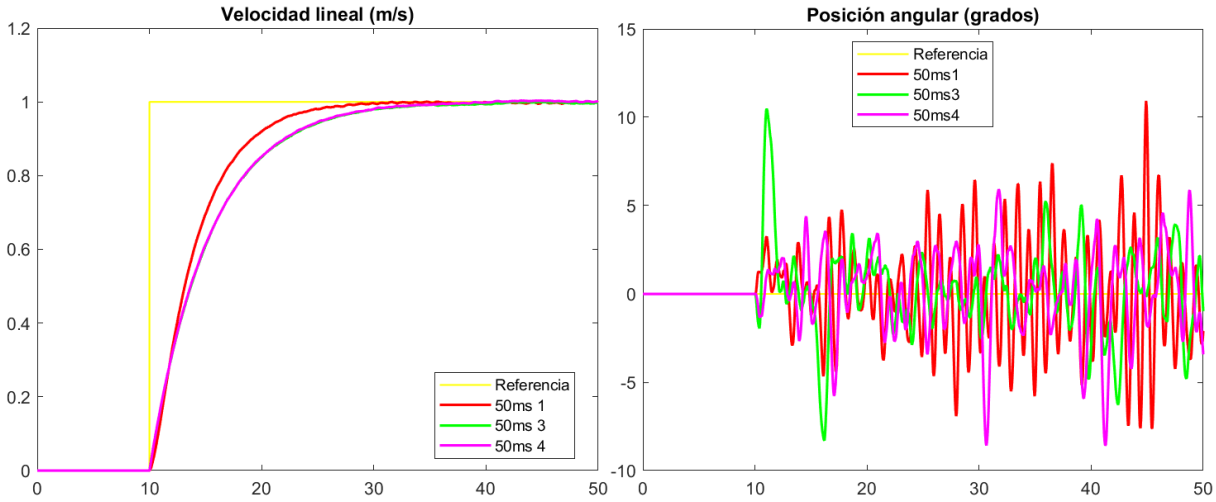

Ilustración 93. Mixed-Sen experimentación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

Al igual que en el anterior control, uno de los problemas que estén provocando este comportamiento puede ser el tiempo de ejecución del bucle de control [Ilustración 94], el cual presenta un valor medio de 0.12 milisegundos, muy por debajo de los 50 ms de período de muestreo. Este suceso, unido a las limitaciones que presenta la RPI, puede ser uno de los detonantes de esta conducta.

```
Lectura2 0 := SysTimeGetUs(pUsTime2 9774999435 );
Tiempo_iter_int 121 := pUsTime2 9774999435 - pUsTime1 9774999314 ;
Tiempo_iter 0.121 := ULINT_TO_LREAL(Tiempo_iter_int 121 )/1000.0;
Tiempo_tot 23.3 := Tiempo_tot 23.3 + Tiempo_iter 0.121 ;
IF (Tiempo_iter 0.121 > Tiempo_iter_max 0.129 ) AND (j2 193 <= 99) THEN
    Tiempo_iter_max 0.129 := Tiempo_iter 0.121 ;
ELSIF (Tiempo_iter 0.121 < Tiempo_iter_min 0.116 ) AND (j2 193 <= 99) THEN
    Tiempo_iter_min 0.116 := Tiempo_iter 0.121 ;
END_IF
IF j2 193 = 99 THEN
    Tiempo_iter_media 0.12 := Tiempo_tot 23.3 /100.0;
END_IF
j2 193 := j2 193 + 1;
```

Ilustración 94. Tiempo máximo, mínimo y medio de ejecución del bucle de control Mixed-Sensitivity (Elaboración propia)

6.4. Control basado en PIDs diseñados con Hinfstruct

6.4.1. Desarrollo teórico y aplicación sobre el Segway

6.4.1.1. Modelado del sistema incierto

Para llevar a cabo este tipo de control y los dos siguientes es necesario el modelado del Segway como un sistema lineal incierto, es decir, considerar que los parámetros fijos del modelo también pueden contener una incertidumbre, lo que provoca cierta desviación respecto a su valor establecido inicialmente. En este caso, mediante el comando 'ureal' se han establecido los siguientes parámetros inciertos [Tabla 11]:

Tabla 11. Parámetros inciertos del modelo del Segway (Elaboración propia)

PARÁMETROS INCIERTOS DEL MODELO DEL SEGWAY		
	Valor nominal	***Incertidumbre***
$\boldsymbol{M}$	$2.5\ Kg$	3%
$\boldsymbol{m}$	$0.6\ Kg$	3%
$\boldsymbol{l}$	$1.2\ m$	3%
$\boldsymbol{r}$	$0.2\frac{N}{m/s}$	3%
$\boldsymbol{g}$	$9.81\ {}^{m}/_{s^2}$	1%

Una vez escogidos estos parámetros, es el momento de calcular el modelo linealizado incierto. No obstante, debe realizarse de forma que su número de ocurrencias en MATLAB sea el menor posible ya que, en el caso de ser elevado, posteriormente podría atraparse en un mínimo local durante el cálculo iterativo utilizado para el diseño de algunos controladores, no encontrando así una solución óptima. Así, tras diferentes pruebas se ha considerado que el método óptimo es el siguiente:

En primer lugar, se diseña en forma matricial el sistema integrado por las ecuaciones (2.1) y (2.2):

$$Mat_{izq} \cdot dX = Mat_{der} \rightarrow \begin{bmatrix} (M+m) & -lm \cdot \sin(\theta) \cdot \dot{\theta} & lm \\ 0 & 1 & 0 \\ lm \cdot \cos(\theta) & -lm \cdot \dot{x} \cdot \sin(\theta) & l^2 \cdot m \end{bmatrix} = \begin{bmatrix} \ddot{x} \\ \dot{\theta} \\ \ddot{\theta} \end{bmatrix} \cdot \begin{bmatrix} F - r \cdot \dot{x} \\ \dot{\theta} \\ glm \cdot \sin(\theta) - \tau \end{bmatrix}$$

De esta forma, sobre la matriz Mat_{der} debe aplicarse el jacobiano, es decir, la derivación parcial de cada una de sus posiciones respecto a sus variables de estado y respecto a sus acciones de control, lo cual se realiza mediante el comando 'jacobian', y posteriormente sustituir las variables de estado por sus valores de equilibrio, los cuales eran nulos para todas ellas. Por otra parte, la matriz Mat_{izq} no debe derivarse, pero sí que se debe sustituir el vector de estados por su punto de equilibrio:

$$Mat_{der_{dA_{eq}}} = \begin{bmatrix} -r & 0 & 0 \\ 0 & 0 & 1 \\ 0 & glm & 0 \end{bmatrix}, \quad Mat_{der_{dB_{eq}}} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & -1 \end{bmatrix}, \quad Mat_{izq_{eq}} = \begin{bmatrix} M+m & 0 & lm \\ 0 & 1 & 0 \\ lm & 0 & l^2 m \end{bmatrix}$$

De este modo, en estas tres matrices de equilibrio es necesario sustituir manualmente las variables simbólicas por sus parámetros inciertos. Tras haberlas implementado, es posible calcular las matrices A y B del modelo lineal incierto en espacio de estados, $SYS_i = [A_i, B_i, C_i, D_i]$, con el mínimo número de ocurrencias:

$$A_i = Mat_{izq_{eq}}^{-1} \cdot Mat_{der_{dA_{eq}}}, \qquad B_i = Mat_{izq_{eq}}^{-1} \cdot Mat_{der_{dB_{eq}}}, \qquad C_i = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \qquad D_i = 0_{2x2}$$

A partir de este sistema incierto se opta por generar 70 muestreos aleatorios mediante el comando 'usample', los cuales son convertidos en modelos de datos de respuesta en frecuencia establecidos en un intervalo entre 0.1 y 1000 rad/s. Estos modelos son utilizados para realizar, mediante el comando 'ucover', un ajuste SYS_r del modelo incierto con un filtro de orden 3, de manera que agrupa todas las incertidumbres en una y las acota superiormente.

Finalmente, haciendo uso de las matrices de escalado y de este ajuste se obtiene el modelo escalado incierto del Segway, $SYS_{esc_i} = E_y^{-1} \cdot SYS_r \cdot Eu$.

Todos estos conceptos se han implementado en el tercer apartado del script "Inicializacion.mlx" [Script 28] mencionado anteriormente:

3. Modelado del sistema incierto

3.1. Modelo no lineal en espacio de estados

```
syms M m l g r F tau ang vel_ang ace_ang vel ace;
param = [M m l r g];
Ec1 = F == (M+m)*ace + m*l*cos(ang)*ace_ang - m*l*sin(ang)*(vel_ang)^(2) + r*vel;
Ec2 = m*g*l*sin(ang) - tau == m*(l)^(2)*ace_ang + m*l*ace*cos(ang) - m*l*vel*vel_ang*sin(ang);
modelo = [Ec1; Ec2];
[Ace, Ace_ang] = solve(modelo, ace, ace_ang);
Vel_ang = vel_ang;

dX = [Ace; Vel_ang; Ace_ang];
U = [F tau]; X = [vel ang vel_ang]; Y = [X(1); X(2)];

F_eq = 0; tau_eq = 0; vel_eq = 0; ang_eq = 0; velang_eq = 0;
U_eq = [F_eq tau_eq]; Y_eq = [vel_eq ang_eq]; X_eq = [vel_eq ang_eq velang_eq];
```

3.2. Parámetros inciertos

```
M_p = ureal('MasaPlataforma',2.5,'Perc',3);
m_p = ureal('MasaPendulo',0.6,'Perc',3);
l_p = ureal('LongitudPendulo',1.2,'Perc',3);
r_p = ureal('Frozamiento',0.2,'Perc',3);
g_p = ureal('Gravedad',9.81,'Perc',1);
param_p = [M_p m_p l_p r_p g_p];
```

3.3. Cálculo del modelo linealizado

```
MatIzq_m = [(M+m) -l*m*ang*vel_ang l*m; 0 1 0; l*m -l*m*vel*ang l^2*m];
MatDer_m = [F - r*vel; vel_ang; g*l*m*ang - tau];
X_m = [ace; vel_ang; ace_ang];
MatDer_dA_m = subs(jacobian(MatDer_m, [vel ang vel_ang]),X,X_eq);
MatDer_dB_m = jacobian(MatDer_m, [F tau]);
MatIzq = [(M_p+m_p) -l_p*m_p*ang_eq*velang_eq l_p*m_p; 0 1 0; l_p*m_p -l_p*m_p*vel_eq*ang_eq
l_p^2*m_p];

MatDer_dA = [-r_p 0 0; 0 0 1; 0 g_p*l_p*m_p 0];
A = MatIzq\MatDer_dA; A.NominalValue;
MatDer_dB = [1 0; 0 0; 0 -1];
B = MatIzq\MatDer_dB; B.NominalValue;
C = [1 0 0; 0 1 0];
D = zeros(2);
SYSi = ss(A,B,C,D); SYS = SYSi.NominalValue;

rng('default'); NumSistemasPrueba = 70;
muchosSYS = usample(SYSi,NumSistemasPrueba);
mSYSf = frd(muchosSYS,logspace(-1,3,60));
[SYSr,Info] = ucover(mSYSf,SYS,3);
SYSesc_i = Ey\(SYSr*Eu);
```

Script 28. Script "Inicializacion.mlx", apartado 3

6.4.1.2. Diseño de PIDs mediante hinfstruct

El control mediante la técnica hinfstruct se basa en el diseño de controladores PID que minimicen la norma ∞ por debajo de 1, de manera que se asegure la estabilidad robusta y, si se cumplen ciertas condiciones que no se abordan en este capítulo, prestaciones robustas. No obstante, ahora se realiza mediante cálculo iterativo, el cual puede provocar largos períodos de ejecución a costa de obtener controladores mucho más precisos, y por lo tanto menos conservadores. En este caso, es necesario el diseño de una planta generalizada ponderada.

En primer lugar, se ha diseñado una planta generalizada no ponderada donde únicamente se consideran las referencias como entradas exógenas, es decir, no hay rechazo de perturbaciones de entrada:

$$\begin{bmatrix} e_1 \\ e_2 \\ u_1 \\ u_2 \\ \hline e_{1_K} \\ e_{2_K} \end{bmatrix} = \begin{bmatrix} I_{2x2} & -SYS_{esc_i} \\ 0_{2x2} & I_{2x2} \\ I_{2x2} & -SYS_{esc_i} \end{bmatrix} \cdot \begin{bmatrix} ref_1 \\ ref_2 \\ \hline u_1 \\ u_2 \end{bmatrix}$$

De esta forma, es posible construir la planta generalizada ponderada utilizando filtros en los errores, acciones de control y referencias:

$$GPW = W_{out} \cdot GPNW \cdot W_{in} = \begin{bmatrix} W_{error} & 0 & 0 \\ 0 & W_u & 0 \\ 0 & 0 & I_{2x2} \end{bmatrix} \cdot GPNW \cdot \begin{bmatrix} W_{ref} & 0 \\ 0 & I_{2x2} \end{bmatrix}$$

Al igual que en los anteriores controles, W_{error} es un filtro paso-bajo con error de posición 0.1% a frecuencias por debajo de la de corte del modelo escalado, mientras que W_u y W_{ref} son pesos constantes que varían según el período de muestreo escogido [Tabla 12].

Tabla 12. Control Hinfstruct: Filtros W_{ref} y W_u (Elaboración propia)

FILTROS DEL CONTROL HINFSTRUCT				
	$\boldsymbol{Ts = 50\ ms}$	$\boldsymbol{Ts = 60\ ms}$	$\boldsymbol{Ts = 80\ ms}$	$\boldsymbol{Ts = 100\ ms}$
$\boldsymbol{W_{ref}}$	$0.979 \cdot I_{2x2}$	$0.979 \cdot I_{2x2}$	$0.979 \cdot I_{2x2}$	I_{2x2}
$\boldsymbol{W_u}$	$0.978 \cdot I_{2x2}$	$0.978 \cdot I_{2x2}$	$0.978 \cdot I_{2x2}$	I_{2x2}

Una vez obtenida la GPW, debe diseñarse un controlador de PIDs genéricos con el comando 'tunablePID', optándose por la configuración $PIDS = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} PID & P \\ P & PID \end{bmatrix}$. Con este controlador y la GPW se implementa el sistema en bucle cerrado, el cual se utiliza para obtener finalmente, mediante el comando 'hinfstruct', un controlador que devuelve una norma ∞ menor que 1, asegurando así la estabilidad robusta del sistema. Finalmente, este controlador es discretizado para poder aplicarlo sobre el modelo no lineal escalado en la fase experimental, es decir, convertido a lenguaje ST.

6.4.2. Fase de simulación en MATLAB

6.4.2.1. Controlador en MATLAB

El fichero Simulink de control en simulación basado en esta técnica tiene una estructura idéntica a la

del fichero "Segway_ModeloNL_controlHinf.slx" [Ilustración 79]. Asimismo, todos los conceptos anteriores se han programado en el script "param_Segway_controlHinfstruct.mlx" [Script 29].

1. Parámetros y simulación en MATLAB

```
…
col1 = [eye(2); zeros(2); eye(2)];
col2 = [zeros(2); eye(2); zeros(2)] - [eye(2); zeros(2); eye(2)]*SYSesc_i;
GPNW = minreal([col1 col2]);
GPNW.InputName = {'r1','r2','u1','u2'};
GPNW.OutputName = {'e1','e2','u1','u2','e1_K','e2_K'};

graf_MAT = input("¿Mostrar SÓLO los resultados de los diferentes Ts en MATLAB?: ",'s'); % Responder
SI en caso de desear mostrarlos
if graf_MAT ~= "SI"
    Ts = input("Elija un período de muestreo: ");
    switch Ts
        case 0.05
            Wref = 0.979*eye(2);
            Wu = 0.978*eye(2);
            errorposiciondeseado = 0.001;
            anchodebandadecontrol = 0.19; % 0.1768754
            plantillaerror = makeweight(errorposiciondeseado,anchodebandadecontrol,3.6);
            Werror = (1/plantillaerror)*eye(2);
        …
    end

    GPW = blkdiag(Werror,Wu,eye(2))*GPNW*blkdiag(Wref,eye(2));

    tic
    C11 = tunablePID('ReguladorPID0','pid');
    C12 = tunablePID('ReguladorPID1','p');
    C21 = tunablePID('ReguladorPID2','p');
    C22 = tunablePID('ReguladorPID3','pid');
    PIDs = [C11 C12; C21 C22];
    TunableCL = lft(GPW.NominalValue,PIDs);
    opt = hinfstructOptions('Display','final','RandomStart',5);
    [CL_PID,GAM_PID,~] = hinfstruct(TunableCL,opt);
    GAM_PID
    toc

    K_hinfstruct = getValue(PIDs,CL_PID.Blocks);
    Kd_hinfstruct = c2d(K_hinfstruct,Ts,'zoh');

    Hinfstructsim = sim("Segway_ModeloNL_controlHinfstruct.slx");
    …
```

2. Resultados

```
…
```

Script 29. Script "param_Segway_controlHinfstruct.mlx"

6.4.2.2. Resultados

Los resultados de las acciones de control [Ilustración 95] y de las variables controladas [Ilustración 96] en simulación son ciertamente correctos para los diferentes períodos de muestreo, ya que se alcanza la referencia de la velocidad lineal sin sobreoscilación en aproximadamente 20 segundos, más lento que en todos los demás controles pero lógico teniendo en cuenta que es más relevante la robustez del proceso, y la posición angular del péndulo apenas sufre una inclinación máxima de 0.8°, todavía menor que en el resto de controles robustos, volviendo a su referencia en unos 20 segundos.

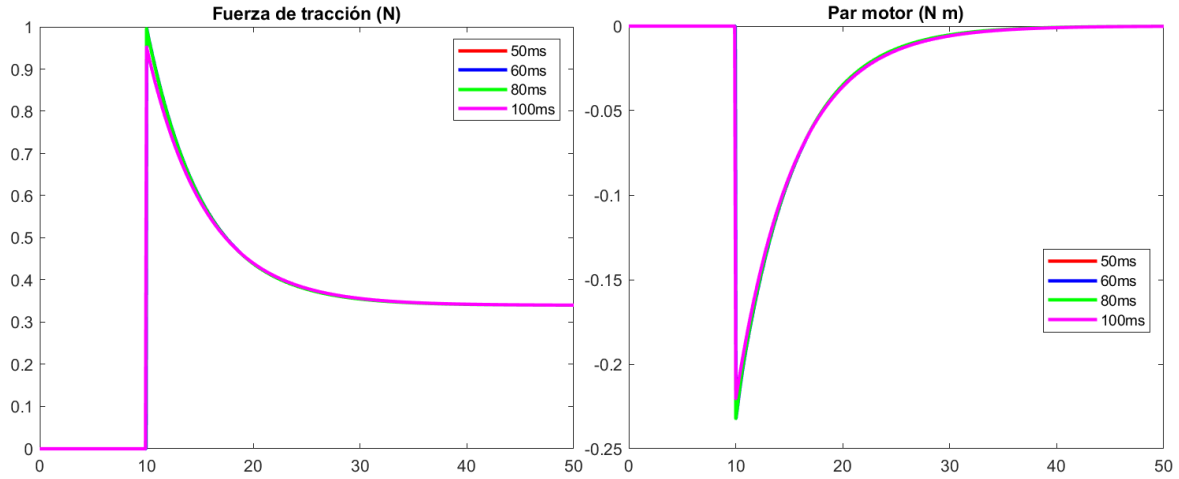

Ilustración 95. Hinfstruct simulación – Acc. Control: F. tracción y par motor (Elaboración propia)

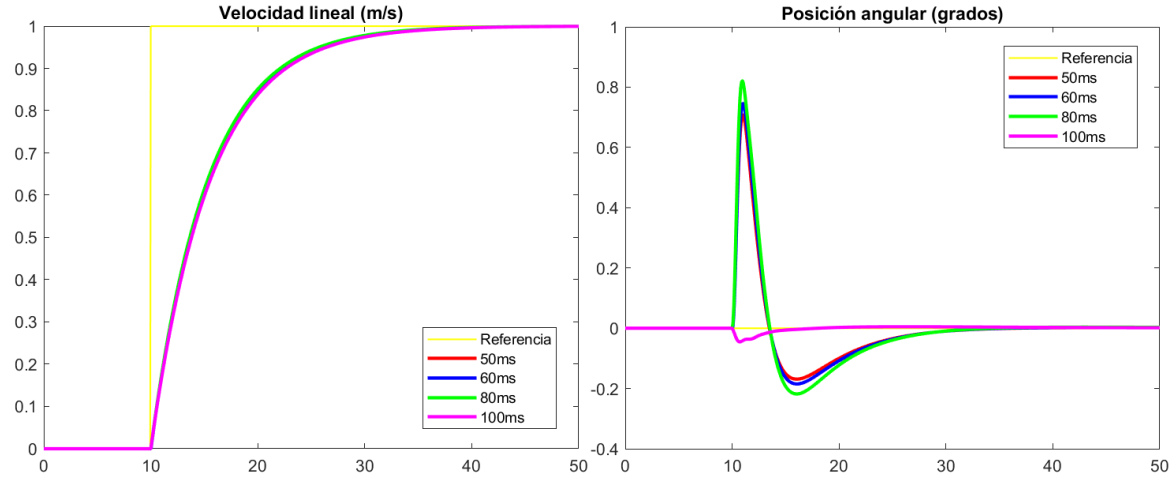

Ilustración 96. Hinfstruct simulación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

6.4.3. Fase experimental en Raspberry Pi y SoMachine

Para la programación del controlador, se ha hecho uso de la herramienta PLC Coder de MATLAB/Simulink.

6.4.3.1. Controlador en SoMachine

Se ha desarrollado un proyecto "Segway_SoMachine_controlHinfstruct_60ms.project" [Script 30] en SoMachine en el que se ha programado el control en lenguaje ST siguiendo el mismo procedimiento que en la fase de simulación. Como puede apreciarse, se ha omitido gran parte del contenido por ser idéntico a los demás, mostrando únicamente aquellos métodos novedosos del controlador.

```
// DECLARACIÓN DE VARIABLES
PROGRAM POU
VAR
	…
	A_ctr: ARRAY[0..15] OF LREAL :=
[1.0,0.0,0.0,0.0,0.0,0.99988103532127315,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.58816062121097];
	B_ctr: ARRAY[0..7] OF LREAL := [0.06,-0.00011896467872687099,0.0,0.0,0.0,0.0,0.06,-0.41183937878903004];
	C_ctr: ARRAY[0..7] OF LREAL := [0.0642639965721041,0.0,-0.0038528553499561579,0.0,0.0,-
0.016538229066037212,0.0,-0.18245633570198333];
	D_ctr: ARRAY[0..3] OF LREAL := [0.99621992844359641,-0.12895959309430208,-0.014913236246936435,-
0.28694057139108908];
END_VAR

// ARRANQUE DEL CONTROLADOR
	…
ELSIF manual = FALSE THEN
	…
	// Cálculo del error entre referencia y salida
	E_k[0] := Ref[0] - Y_k[0];
	E_k[1] := Ref[1] - Y_k[1];

	// Cálculo de las acciones de control
	rtb_Controlador[0] := (D_ctr[2]*E_k[1]) + ((D_ctr[0] *E_k[0]) +
((C_ctr[6]*Controlador_DSTATE[3]) + ((C_ctr[4]*Controlador_DSTATE[2]) +
((C_ctr[2]*Controlador_DSTATE[1]) + (C_ctr[0]*Controlador_DSTATE[0])))));
	rtb_Controlador[1] := (D_ctr[3]*E_k[1]) + ((D_ctr[1] *E_k[0]) +
((C_ctr[7]*Controlador_DSTATE[3]) + ((C_ctr[5]*Controlador_DSTATE[2]) +
((C_ctr[3]*Controlador_DSTATE[1]) + (C_ctr[1]*Controlador_DSTATE[0])))));

	xnew[0] := (B_ctr[4]*E_k[1]) + ((B_ctr[0]*E_k[0]) + ((A_ctr[12]*Controlador_DSTATE[3]) +
((A_ctr[8]*Controlador_DSTATE[2]) + ((A_ctr[4]*Controlador_DSTATE[1]) +
(A_ctr[0]*Controlador_DSTATE[0])))));
	xnew[1] := (B_ctr[5]*E_k[1]) + ((B_ctr[1]*E_k[0]) + ((A_ctr[13]*Controlador_DSTATE[3]) +
((A_ctr[9]*Controlador_DSTATE[2]) + ((A_ctr[5]*Controlador_DSTATE[1]) +
(A_ctr[1]*Controlador_DSTATE[0])))));
	xnew[2] := (B_ctr[6]*E_k[1]) + ((B_ctr[2]*E_k[0]) + ((A_ctr[14]*Controlador_DSTATE[3]) +
((A_ctr[10]*Controlador_DSTATE[2]) + ((A_ctr[6]*Controlador_DSTATE[1]) +
(A_ctr[2]*Controlador_DSTATE[0])))));
	xnew[3] := (B_ctr[7]*E_k[1]) + ((B_ctr[3]*E_k[0]) + ((A_ctr[15]*Controlador_DSTATE[3]) +
((A_ctr[11]*Controlador_DSTATE[2]) + ((A_ctr[7]*Controlador_DSTATE[1]) +
(A_ctr[3]*Controlador_DSTATE[0])))));

	Controlador_DSTATE[0] := xnew[0];
	Controlador_DSTATE[1] := xnew[1];
	Controlador_DSTATE[2] := xnew[2];
	Controlador_DSTATE[3] := xnew[3];

	// Escalado de las acciones de control
	U_k[0] := rtb_Controlador[0]*1.0;
	U_k[1] := rtb_Controlador[1]*1.8;
	…
END_IF
```

Script 30. Proyecto "Segway_SoMachine_controlHinfstruct_60ms.project"

6.4.3.2. Resultados

Pese a que los resultados en simulación eran ciertamente esperanzadores, lo cierto es que el comportamiento de las acciones de control [Ilustración 97] y de las variables controladas [Ilustración 98] del proceso en la fase experimental ha dejado mucho que desear. Como puede apreciarse, la velocidad lineal ha sido fiel a lo que se esperaba según simulación, ya que ha alcanzado su referencia sin ninguna dificultad. Sin embargo, la posición angular no es capaz de estabilizarse en ningún momento, oscilando así constantemente entre los 10° y los -10°. Esto puede ser debido a errores de cuantificación en el par motor, que sigue una trayectoria similar a la simulada, pero con un principio de inestabilidad mucho mayor. Por tanto, puede concluirse que el control Hinfstruct no ha resultado favorecedor.

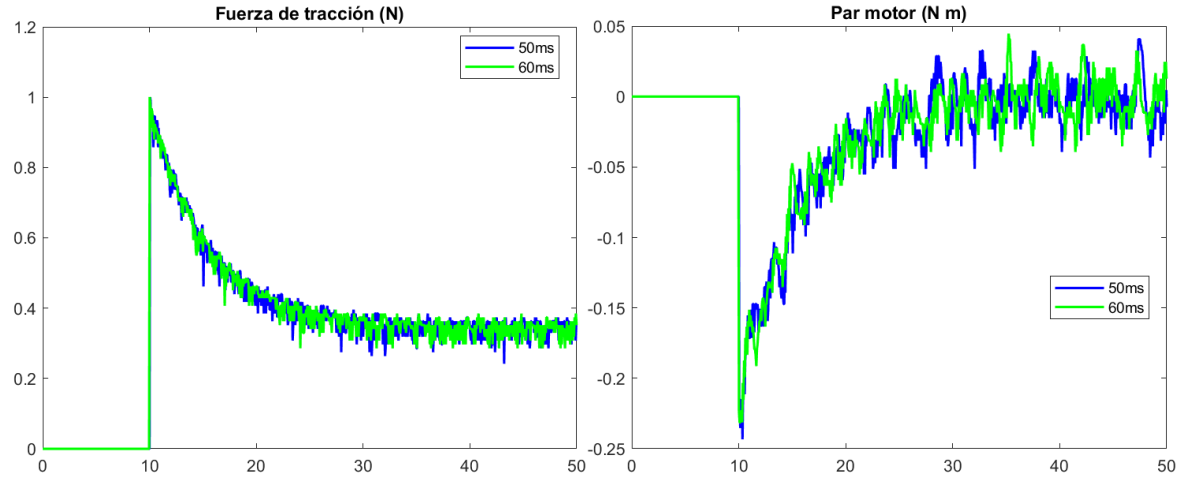

Ilustración 97. Hinfstruct experimentación – Acc. Control: F. tracción y par motor (Elaboración propia)

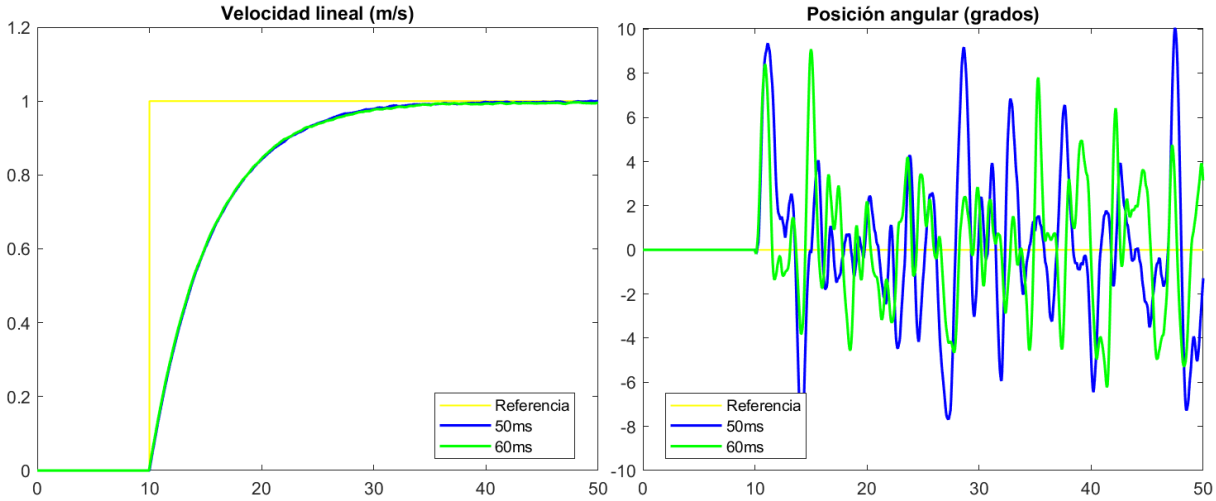

Ilustración 98. Hinfstruct experimentación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

Al igual que en el anterior control, uno de los problemas que estén provocando este comportamiento puede ser el tiempo de ejecución del bucle de control [Ilustración 99], el cual presenta un valor medio de 0.106 milisegundos, muy por debajo de los períodos de muestreo experimentados. Este suceso, unido a las limitaciones que presenta la RPI, puede ser uno de los detonantes de esta conducta.

```
Lectura2 0 := SysTimeGetUs(pUsTime2 10755158395);
Tiempo_iter_int 103 := pUsTime2 10755158395 - pUsTime1 10755158292;
Tiempo_iter 0.103 := ULINT_TO_LREAL(Tiempo_iter_int 103)/1000.0;
Tiempo_tot 16.7 := Tiempo_tot 16.7 + Tiempo_iter 0.103;
IF (Tiempo_iter 0.103 > Tiempo_iter_max 0.113) AND (j2 158 <= 99) THEN
    Tiempo_iter_max 0.113 := Tiempo_iter 0.103;
ELSIF (Tiempo_iter 0.103 < Tiempo_iter_min 0.101) AND (j2 158 <= 99) THEN
    Tiempo_iter_min 0.101 := Tiempo_iter 0.103;
END_IF
IF j2 158 = 99 THEN
    Tiempo_iter_media 0.106 := Tiempo_tot 16.7 /100.0;
END_IF
j2 158 := j2 158 + 1;
```

Ilustración 99. Tiempo máximo, mínimo y medio de ejecución del bucle de control Hinfstruct (Elaboración propia)

6.5. Control DK-Iteration/µ-Synthesis

6.5.1. Desarrollo teórico y aplicación sobre el Segway

Este método de control, antiguamente conocido como DK-Iteration o iteración multiplicador-controlador y actualmente como µ-Synthesis, se basa en un cálculo iterativo que, combinando la norma ∞ y el análisis µ, trata de reducir por debajo de 1 la norma ∞, asegurando así estabilidad robusta y, si se dan ciertas condiciones, prestaciones robustas del sistema en bucle cerrado. Puesto que requiere iteración, la obtención del controlador es más lenta que en otros controles como Mixed Sensitivity, y es necesario relajar un poco las prestaciones de los filtros utilizados.

Para utilizar esta técnica, es necesario diseñar una planta generalizada no ponderada tal y como se hizo en el apartado (6.4). Sin embargo, al diseñar la ponderada no es necesario realizar la ponderación de las entradas, ya que este método considera por sí mismo la incertidumbre del proceso:

$$GPW = W_{out} \cdot GPNW = \begin{bmatrix} W_{error} & 0 & 0 \\ 0 & W_u & 0 \\ 0 & 0 & I_{2x2} \end{bmatrix} \cdot GPNW$$

Respecto a los filtros, en W_{error} ha sido necesario reducir el ancho de banda un poco por debajo de la frecuencia de corte, manteniendo el error de posición de 0.1% a baja frecuencia, y W_u es un peso constante que limita el tamaño de las acciones de control a todas las frecuecias.

Asimismo, al comando 'musyn' se le puede indicar que comience el proceso iterativo desde cierto punto de partida, que en el caso del Segway ha sido el controlador obtenido anteriormente en Mixed Sensitivity. De esta forma, se ha obtenido un controlador que mejora la respuesta en frecuencia y de las salidas del sistema respecto a dicha técnica y que asegura estabilidad robusta, pero a costa de un mayor tiempo de cálculo.

Sin embargo, este controlador contiene 23 variables de estado, el cual es un orden muy elevado que, en la conversión a la fase experimental, puede derivar en problemas de control debidos al gran coste computacional. No obstante, utilizando el comando 'balred' es posible obtener los valores singulares de Hankel del controlador [Ilustración 100], los cuales muestran que puede realizarse una reducción a orden 6 sin cometer demasiado error y siendo fiel al controlador inicial. Finalmente, este controlador es discretizado para poder aplicarlo sobre el modelo no lineal escalado en la fase experimental, es decir, convertido a lenguaje ST.

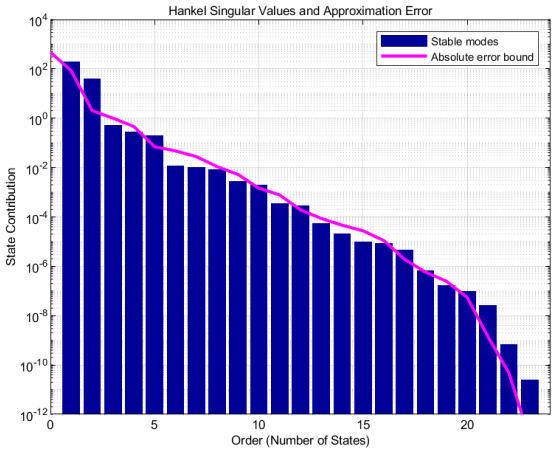

Ilustración 100. Control µ-Synthesis: Valores singulares de Hankel del controlador (Elaboración propia)

6.5.2. Fase de simulación en MATLAB

6.5.2.1. Controlador en MATLAB

El fichero Simulink de control en simulación basado en esta técnica tiene una estructura idéntica a la del fichero "Segway_ModeloNL_controlHinf.slx" [Ilustración 79]. Asimismo, todos los conceptos anteriores se han programado en el script "param_Segway_controlMusyn.mlx" [Script 31].

1. Parámetros y simulación en MATLAB

```
…
col1 = [eye(2); zeros(2); eye(2)];
col2 = [zeros(2); eye(2); zeros(2)] - [eye(2); zeros(2); eye(2)]*SYSesc_i;
GPNW = minreal([col1 col2]);
GPNW.InputName = {'r1','r2','u1','u2'};
GPNW.OutputName = {'e1','e2','u1','u2','e1_K','e2_K'};

graf_MAT = input("¿Mostrar SÓLO los resultados de los diferentes Ts en MATLAB?: ",'s'); % Responder
SI en caso de desear mostrarlos
if graf_MAT ~= "SI"
    Ts = input("Elija un período de muestreo: ");
    switch Ts
        case 0.04
            errorposiciondeseado = 0.001;
            anchodebandadecontrol = w1_0dB; % w1_0dB = 0.1768754
            plantillaerror = makeweight(errorposiciondeseado,anchodebandadecontrol,3.5);
            Werror = (1/plantillaerror)*eye(2);
            Wu = 0.9*eye(2);
    end

    GPW = blkdiag(Werror,Wu,eye(2))*GPNW_U;

    tic;
    [K_mu,GAM_mu,INFO_mu] = musyn(GPW,2,2,K_mxs);
    toc;
    GAM_mu
    figure; balred(K_mu);
    K_mu_red = balred(K_mu,6);
    Kd_mu_red = c2d(K_mu_red,Ts,'zoh');

    Musynsim = sim("Segway_ModeloNL_controlMusyn.slx");

    …
```

2. Resultados

```
…
```

Script 31. Script "param_Segway_controlMusyn.mlx"

6.5.2.2. Resultados

Los resultados de las acciones de control [Ilustración 101] y de las variables controladas [Ilustración 102] en simulación son ciertamente correctos para los diferentes períodos de muestreo, ya que se alcanza la referencia de la velocidad lineal sin sobreoscilación en aproximadamente 20 segundos, más lento que en todos los demás controles pero lógico teniendo en cuenta que es más relevante la robustez del proceso, y la posición angular del péndulo apenas sufre una inclinación máxima de 1.6°, lo cual es prácticamente imperceptible, volviendo a su referencia en unos 20 segundos.

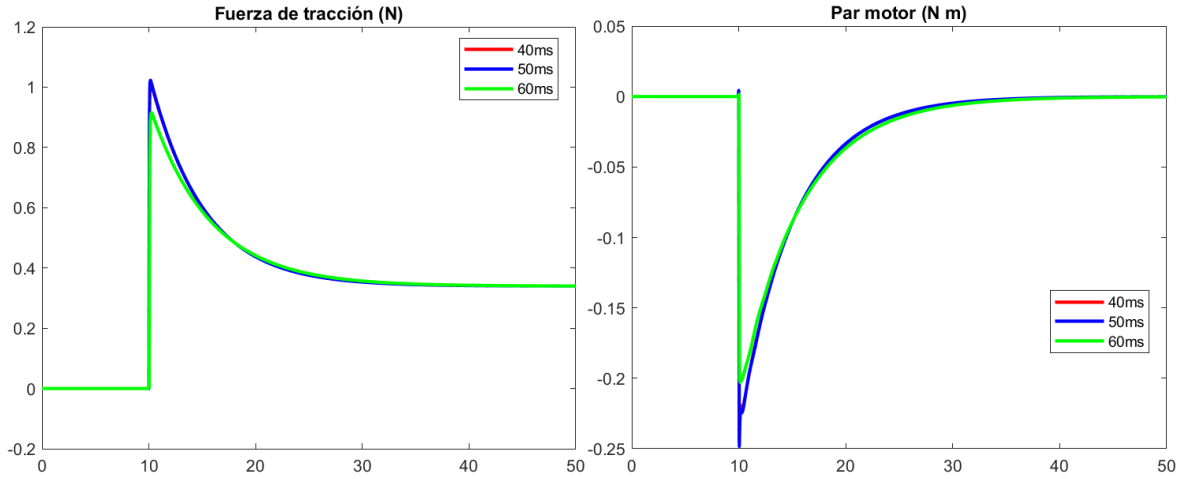

Ilustración 101. Mu-Syn simulación – Acc. Control: F. tracción y par motor (Elaboración propia)

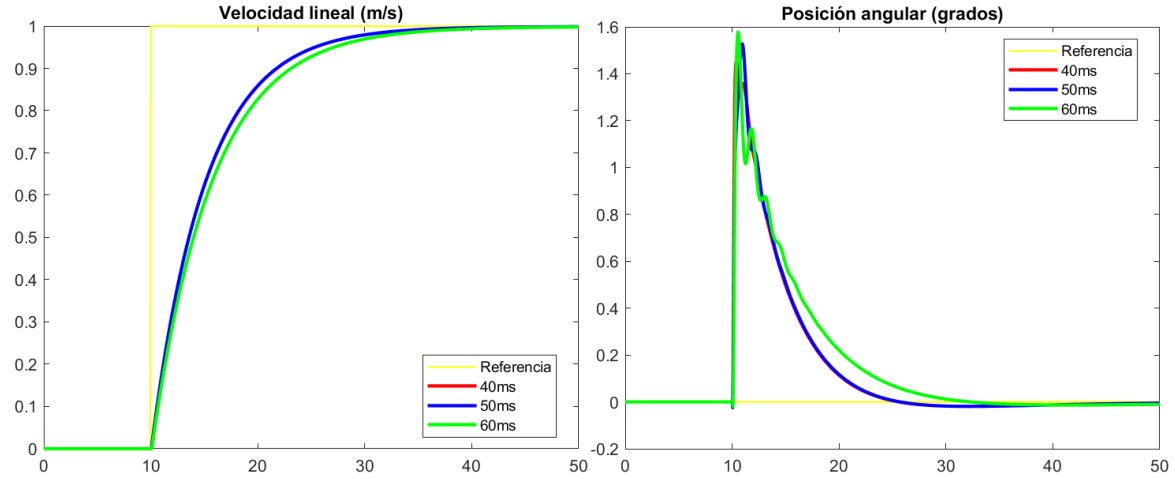

Ilustración 102. Mu-Syn simulación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

6.5.3. Fase experimental en Raspberry Pi y SoMachine

Para la programación del controlador, se ha hecho uso de la herramienta PLC Coder de MATLAB/Simulink.

6.5.3.1. Controlador en SoMachine

Se ha desarrollado un proyecto "Segway_SoMachine_controlMusyn_60ms.project" [Script 32] en SoMachine en el que se ha programado el control en lenguaje ST siguiendo el mismo procedimiento que en la fase de simulación. Como puede apreciarse, se ha omitido gran parte del contenido por ser idéntico a los demás, mostrando únicamente aquellos métodos novedosos del controlador.

```
// DECLARACIÓN DE VARIABLES
PROGRAM POU
VAR
	…
	A_ctr: ARRAY[0..24] OF LREAL := [1.0015640452828343,-0.00058971730483510976,-
0.005393286144000177,0.0066041293281130144,-0.027820408337310334,0.00010066407455547934,0.99967727519931926,-
0.024085487598193026,-0.0038641942580022624,-0.0010892754198569864,4.85676056392776E-5,0.0015438702878679961,-
0.043322715766266118,-
0.57552485547788768,0.098513195878874874,0.008882924305701298,0.020127399218992263,0.21799052741678812,0.50002177
357834565,-0.060624357009569127,0.047274954358854406,-0.0089326190032915781,-0.093821126339670449,-
0.015505088549713482,0.20224360494813187];
	B_ctr: ARRAY[0..9] OF LREAL := [-0.27315259571310785,-0.00980405300619071,-0.0030898443309312303,-
0.0034988428561967206,-0.032460324933487709,0.0092488067705904561,-0.27507190951227717,-0.10806489238963468,-
0.01746441817774402,0.0078766492703021326];
	C_ctr: ARRAY[0..9] OF LREAL := [-0.59503822782960092,0.052054948407606043,-
0.054029127105853447,0.044530274703515685,-0.48977195501578552,3.3504680641087421,-
2.7258657132863244,2.3791435195528341,-17.298840509136859,2.4220608046929422];
	D_ctr: ARRAY[0..3] OF LREAL := [0.0031925105732936005,0.000897065180671856,-0.00017980733522470049,-
0.0091808999266617085];
END_VAR

// ARRANQUE DEL CONTROLADOR
	…
ELSIF manual = FALSE THEN
	…
	// Cálculo del error entre referencia y salida
	E_k[0] := Ref[0] - Y_k[0];
	E_k[1] := Ref[1] - Y_k[1];
	// Cálculo de las acciones de control
	rtb_Controlador[0] := C_ctr[0]*Controlador_DSTATE[0] + C_ctr[2]*Controlador_DSTATE[1] +
C_ctr[4]*Controlador_DSTATE[2] + C_ctr[6]*Controlador_DSTATE[3] + C_ctr[8]*Controlador_DSTATE[4] +
D_ctr[0]*E_k[0] + D_ctr[2]*E_k[1];
	rtb_Controlador[1] := C_ctr[1]*Controlador_DSTATE[0] + C_ctr[3]*Controlador_DSTATE[1] +
C_ctr[5]*Controlador_DSTATE[2] + C_ctr[7]*Controlador_DSTATE[3] + C_ctr[9]*Controlador_DSTATE[4] +
D_ctr[1]*E_k[0] + D_ctr[3]*E_k[1];

	xnew[0] := A_ctr[0]*Controlador_DSTATE[0] + A_ctr[5]*Controlador_DSTATE[1] +
A_ctr[10]*Controlador_DSTATE[2] + A_ctr[15]*Controlador_DSTATE[3] + A_ctr[20]*Controlador_DSTATE[4] +
B_ctr[0]*E_k[0] + B_ctr[5]*E_k[1];
	xnew[1] := A_ctr[1]*Controlador_DSTATE[0] + A_ctr[6]*Controlador_DSTATE[1] +
A_ctr[11]*Controlador_DSTATE[2] + A_ctr[16]*Controlador_DSTATE[3] + A_ctr[21]*Controlador_DSTATE[4] +
B_ctr[1]*E_k[0] + B_ctr[6]*E_k[1];
	xnew[2] := A_ctr[2]*Controlador_DSTATE[0] + A_ctr[7]*Controlador_DSTATE[1] +
A_ctr[12]*Controlador_DSTATE[2] + A_ctr[17]*Controlador_DSTATE[3] + A_ctr[22]*Controlador_DSTATE[4] +
B_ctr[2]*E_k[0] + B_ctr[7]*E_k[1];
	xnew[3] := A_ctr[3]*Controlador_DSTATE[0] + A_ctr[8]*Controlador_DSTATE[1] +
A_ctr[13]*Controlador_DSTATE[2] + A_ctr[18]*Controlador_DSTATE[3] + A_ctr[23]*Controlador_DSTATE[4] +
B_ctr[3]*E_k[0] + B_ctr[8]*E_k[1];
	xnew[4] := A_ctr[4]*Controlador_DSTATE[0] + A_ctr[9]*Controlador_DSTATE[1] +
A_ctr[14]*Controlador_DSTATE[2] + A_ctr[19]*Controlador_DSTATE[3] + A_ctr[24]*Controlador_DSTATE[4] +
B_ctr[4]*E_k[0] + B_ctr[9]*E_k[1];

	Controlador_DSTATE[0] := xnew[0];
	Controlador_DSTATE[1] := xnew[1];
	Controlador_DSTATE[2] := xnew[2];
	Controlador_DSTATE[3] := xnew[3];
	Controlador_DSTATE[4] := xnew[4];
	// Escalado de las acciones de control
	U_k[0] := rtb_Controlador[0]*1.0;
	U_k[1] := rtb_Controlador[1]*1.8;
	…
END_IF
```

Script 32. Proyecto "Segway_SoMachine_controlMusyn_60ms.project"

6.5.3.2. Resultados

Pese a que los resultados en simulación eran ciertamente esperanzadores, lo cierto es que el comportamiento de las acciones de control [Ilustración 103] y de las variables controladas [Ilustración 104] del proceso en la fase experimental ha dejado mucho que desear. Como puede apreciarse, la velocidad lineal ha sido fiel a lo que se esperaba según simulación, ya que ha alcanzado su referencia sin ninguna dificultad. Sin embargo, la posición angular no es capaz de estabilizarse en ningún

momento, oscilando así constantemente entre los 10° y los -10°. Esto puede ser debido a errores de cuantificación en el par motor, que sigue una trayectoria similar a la simulada, pero con un principio de inestabilidad mucho mayor. Por tanto, puede concluirse que el control µ-Synthesis no ha resultado favorecedor.

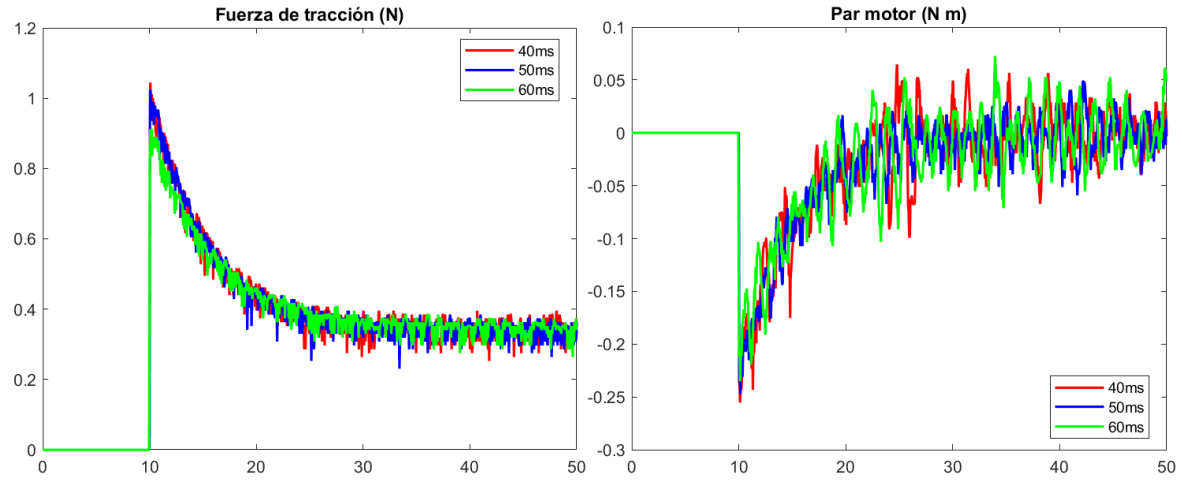

Ilustración 103. Mu-Syn experimentación – Acc. Control: F. tracción y par motor (Elaboración propia)

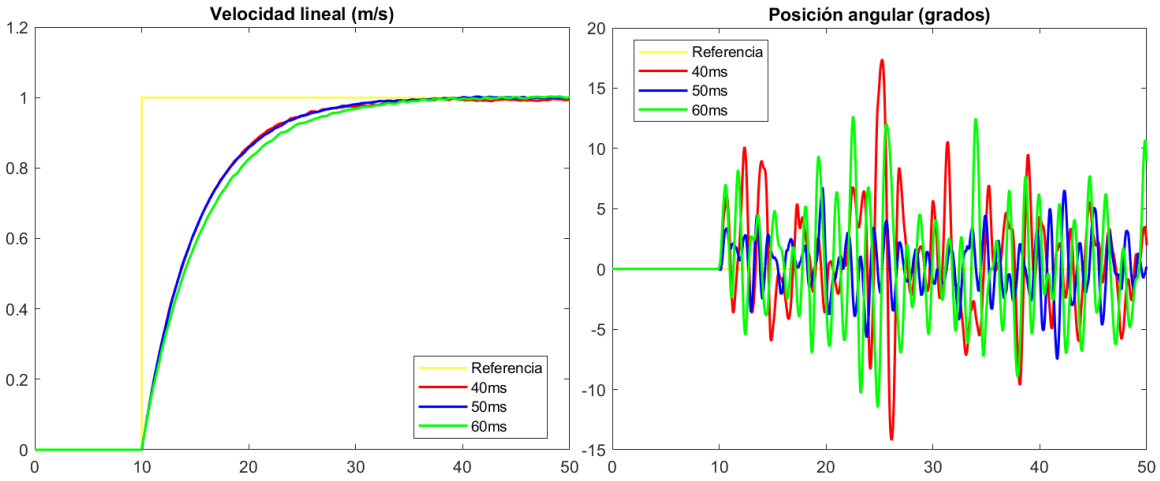

Ilustración 104. Mu-Syn experimentación – Var. Ctr.: Vel. Lineal y pos. Angular (Elaboración propia)

Al igual que en el anterior control, uno de los problemas que estén provocando este comportamiento puede ser el tiempo de ejecución del bucle de control [Ilustración 105], el cual presenta un valor medio de 0.124 milisegundos, muy por debajo de los períodos de muestreo experimentados. Este suceso, unido a las limitaciones que presenta la RPI, puede ser uno de los detonantes de esta conducta.

```
Lectura2 0 := SysTimeGetUs(pUsTime2 10348057419);
Tiempo_iter_int 124 := pUsTime2 10348057419 - pUsTime1 10348057295;
Tiempo_iter 0.124 := ULINT_TO_LREAL(Tiempo_iter_int 124 )/1000.0;
Tiempo_tot 18.2 := Tiempo_tot 18.2 + Tiempo_iter 0.124;
IF (Tiempo_iter 0.124 > Tiempo_iter_max 0.132 ) AND (j2 146 <= 99) THEN
    Tiempo_iter_max 0.132 := Tiempo_iter 0.124;
ELSIF (Tiempo_iter 0.124 < Tiempo_iter_min 0.122 ) AND (j2 146 <= 99) THEN
    Tiempo_iter_min 0.122 := Tiempo_iter 0.124;
END_IF
IF j2 146 = 99 THEN
    Tiempo_iter_media 0.124 := Tiempo_tot 18.2 /100.0;
END_IF
j2 146 := j2 146 + 1;
```

Ilustración 105. Tiempo máximo, mínimo y medio de ejecución del bucle de control µ-Synthesis (Elaboración propia)

Capítulo 7

Conclusiones

7.1. Conclusiones

A partir del modelo no lineal del Segway disponible inicialmente, se ha obtenido un modelo linealizado del mismo que ha mostrado una dinámica muy similar pese a la simplificación de muchos de sus términos.

Asimismo, se ha desarrollado una gran cantidad de tipos de control, cada uno de ellos basado en unos principios teóricos y criterios diferentes, gracias al uso de este modelo lineal. Estos controles han demostrado en simulación una gran capacidad de estabilización de la velocidad lineal del Segway y de la posición angular de su mando direccional, y en gran medida también ha sido así en la fase experimental, a pesar de las limitaciones de velocidad que pueda ofrecer la Raspberry Pi, en combinación con un PLC que ha ejecutado excelentemente los controladores, y de los problemas de cuantificación sufridos al trabajar con variables analógicas.

Por otra parte, se han apreciado las distintas funcionalidades de que dispone la Raspberry Pi pese a su reducido precio, ya que ha sido capaz de recrear con creces el comportamiento que debe tener el Segway.

Por todo ello, y pese a algunos resultados nada favorecedores, puede considerarse que el presente Trabajo Final de Máster ha resultado exitoso en su ejecución, ya que ha sido posible poner en marcha un eficiente sistema de control en el PLC y el modelo Segway se ha programado satisfactoriamente mediante HIL (Hardware in the loop) en una Raspberry Pi.

Parte II

Bibliografía

Bibliografía

[1] Página Web. *CVEM Segway Professional: Segway i2 SE, 2021* [fecha de consulta: 9 de junio de 2021]. Disponible en <https://cvem.es/producto/segway-i2-se/>.

[2] Página Web. *Segway Professional: Nuestra historia, 2021* [fecha de consulta: 9 de junio de 2021]. Disponible en <https://www.segway.es/es/comunicacion/nuestra-historia/>.

[3] Página Web. *Raspberry Pi, products: Raspberry Pi 4 Model B, 2021* [fecha de consulta: 9 de junio de 2021]. Disponible en <https://www.raspberrypi.org/products/>.

[4] Página Web. *GEEKNETIC: ¿Qué es Raspberry Pi y para qué sirve?, 2020* [fecha de consulta: 10 de junio de 2021]. Disponible en <https://www.geeknetic.es/Raspberry-Pi/que-es-y-para-que-sirve>.

[5] Página Web. *MathWorks: MATLAB, 2021* [fecha de consulta: 19 de mayo de 2021]. Disponible en <https://es.mathworks.com/products/matlab.html>.

[6] Página Web. *MathWorks: Simulink, 2021* [fecha de consulta: 19 de mayo de 2021]. Disponible en <https://es.mathworks.com/products/simulink.html>.

[7] Página Web. *Spyder IDE: Spyder, 2020* [fecha de consulta: 20 de mayo de 2021]. Disponible en <https://www.spyder-ide.org/>.

[8] Página Web. *Schneider Electric: Controlador M241 40 ES Relé Ethernet, 2021* [fecha de consulta: 8 de junio de 2021]. Disponible en <https://www.se.com/es/es/product/TM241CE40R/%22controlador-m241-40-es-rel%C3%A9-ethernet%22/?range=62129-modicon-m241&selected-node-id=12692219234>.

[9] Página Web. *Schneider Electric: EcoStruxure Machine Expert, 2021* [fecha de consulta: 8 de junio de 2021]. Disponible en <https://www.se.com/es/es/product-range-presentation/2226-ecostruxure-machine-expert-%28somachine%29/>.

[10] Página Web. *Evolución Histórica de la Ingeniería de Control: Primeros ejemplos históricos de sistemas de control, 1999* [fecha de consulta: 13 de junio de 2021]. Disponible en <http://automata.cps.unizar.es/regulacionautomatica/historia.PDF>.

[11] Página Web. *PC Componentes: Xiaomi Mi Electric Scooter 1S Patinete Eléctrico Negro, 2021* [fecha de consulta: 13 de junio de 2021]. Disponible en <https://www.rmhobby.es/patinetes-electricos/449-patinete-electrico-lion-85-8437013067188.html>.

[12] Archivo PDF. *Ingeniería de Control (MUII): TAREA 1. Modelado de un SegWay, 201*9 [fecha de consulta: 11 de agosto de 2021].

[13] Archivo PDF. *Herramientas de Modelado y Simulación de Procesos (MUII): Sistema: Péndulo invertido sobre carro móvil, 2020* [fecha de consulta: 30 de agosto de 2021].

[14] Archivo PDF. *Identificación y Control de Sistemas Complejos (MUII): Robust Computer Control of an Inverted Pendulum, 2020* [fecha de consulta: 30 de agosto de 2021].

[15] Archivo PDF. *Ingeniería de Control (MUII): Tema 2: Representación de Sistemas Multivariables, 2019* [fecha de consulta: 2 de septiembre de 2021].

[16] Página Web. *NOVUS: Artículo: Control PID: rompiendo la barrera del tiempo: Historia, 2021* [fecha de consulta: 2 de septiembre de 2021]. Disponible en <https://www.novusautomation.com/site/default.asp?Idioma=34&TroncoID=053663&SecaoID=0&SubsecaoID=0&Template=../artigosnoticias/user_exibir.asp&ID=638091>.

[17] Página Web. *PROGRAMACIONSIEMENS.com, 2021* [fecha de consulta: 2 de septiembre de 2021]. Disponible en <https://programacionsiemens.com/pid-en-step7/>.

[18] Página Web. *Picuino: Controlador PID, 2021* [fecha de consulta: 2 de septiembre de 2021]. Disponible en <https://www.picuino.com/es/arduprog/control-pid.html>.

[19] Archivo PDF. *Ingeniería de Control (MUII): Tema 3a: Análisis de Sistemas Multivariables. Controlabilidad y Observabilidad, 2019* [fecha de consulta: 10 de septiembre de 2021].

[20] Archivo PDF. *Ingeniería de Control (MUII): Tema 4: Realimentación del Estado, 2019* [fecha de consulta: 10 de septiembre de 2021].

[21] Archivo PDF. *Ingeniería de Control (MUII): Tema 5: Observadores y Filtros: Sensores Virtuales, 2019* [fecha de consulta: 10 de septiembre de 2021].

[22] Archivo PDF. *Ingeniería de Control (MUII): Tema 6: Control Óptimo, 2019* [fecha de consulta: 10 de septiembre de 2021].

[23] Archivo PDF. *Ingeniería de Control (MUII): Tema 7: Filtro de Kalman, 2019* [fecha de consulta: 10 de septiembre de 2021].

[24] Archivo PDF. *Control Industrial Avanzado (MUII): Tema 1: Basics of Industrial Model Predictive Control, 2019* [fecha de consulta: 11 de septiembre de 2021].

[25] Archivo PDF. *Control Industrial Avanzado (MUII): Tema 2: Dynamic Matrix Control (DMC), 2019* [fecha de consulta: 11 de septiembre de 2021].

[26] Archivo PDF. *Control Industrial Avanzado (MUII): Tema 6: State Space MPC, 2019* [fecha de consulta: 11 de septiembre de 2021].

[27] Página Web. *Modelado, análisis, identificación y control de sistemas multivariables. Antonio Sala Piqueras, 2021* [fecha de consulta: 12 de septiembre de 2021]. Disponible en <http://personales.upv.es/asala/DocenciaOnline/Cursos/Apuntes.html>.

[28] Página Web. *Lenovo: Lenovo Legion 5Pi, 2021* [fecha de consulta: 12 de septiembre de 2021]. Disponible en <https://www.lenovo.com/es/es/laptops/legion-laptops/legion-5-series/Lenovo-

Legion-5Pi-15IMH05/p/82AY001YSP>.

[29] Página Web. *Schneider Electric: Controlador M241 40 ES Relé Ethernet, 2021* [fecha de consulta: 12 de septiembre de 2021]. Disponible en <https://www.se.com/es/es/product/TM241CE40R/%22controlador-m241-40-es-rel%C3%A9-ethernet%22/?range=62129-modicon-m241&selected-node-id=12692219234>.

[30] Página Web. *Raspberry Pi: Products: Raspberry Pi 4 Model B, 2021* [fecha de consulta: 12 de septiembre de 2021]. Disponible en <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/>.

[31] Página Web. *MathWorks: Pricing and Licensing: Student license, 2021* [fecha de consulta: 12 de septiembre de 2021]. Disponible en <https://es.mathworks.com/pricing-licensing.html?prodcode=ML&intendeduse=comm>.

[32] Página Web. *Microsoft: Microsoft 365 Personal, 2021* [fecha de consulta: 12 de septiembre de 2021]. Disponible en <https://www.microsoft.com/es-es/microsoft-365/p/microsoft-365-personal/cfq7ttc0k5bf?activetab=pivot%3aoverviewtab>.

Parte III

Presupuesto

Capítulo 8

Presupuesto

8.1. Necesidad del presupuesto

El presente proyecto de ingeniería ha requerido un gran esfuerzo del autor con el fin de realizar cálculos precisos y obtener resultados satisfactorios, pero también ha requerido el uso de mano de obra, hardware y software, conllevando cada uno de ellos unos costes económicos.

Por este motivo, es necesaria la elaboración de un presupuesto que considere los recursos empleados, tales como los honorarios del autor de este proyecto y del supervisor del mismo, la amortización del ordenador, el coste del hardware utilizado o las licencias de software, entre otros.

8.2. Cuadro de precios básicos

A continuación, se definen los precios básicos [Tabla 13] de los distintos recursos físicos e intangibles utilizados:

8.2.1. Mano de obra

La mano de obra de este proyecto está formada por un Ingeniero Industrial, siendo éste el encargado del diseño e implementación de todos los sistemas de control y de la redacción de todos los documentos necesarios. Tiene un coste de 30€/h, acorde a un titulado de estas características.

8.2.2. Hardware

La fase de simulación del proyecto se ha desarrollado, principalmente, en un ordenador portátil adquirido por 936.75€ [28]. Considerándose que su período de amortización es de 5 años, que 2021 cuenta con 251 días laborables – sin tener en cuenta fines de semana ni festivos – y que se ha seguido una jornada laboral de 8 horas/día, se obtiene un coste de amortización de 0.09€/h según la siguiente fórmula:

$$Coste\ horario\left(\text{€}/_{h}\right) = \frac{Coste(\text{€})}{Días\ laborables\left(días/_{\text{año}}\right) \cdot Jornada\ laboral\left(h/_{día}\right) \cdot Amortización(\ años)}$$

Por otro lado, en la fase experimental del sistema control ha sido vital el uso de un autómata programable, modelo M241 40 ES Relé Ethernet [29], y de una Raspberry Pi, modelo 4 B [30], para la implementación del controlador y del modelo no lineal del Segway. Éstos han sido adquiridos por 426.94€ y 29.63€, respectivamente, los cuales han sido cargados íntegramente por no considerarse su

amortización a largo plazo. En otras palabras, a diferencia del ordenador, al cual se le puede seguir dando otros usos en un futuro, estos dispositivos se utilizan única y exclusivamente en el control del Segway.

8.2.3. Software

La fase de simulación del proyecto se realiza gracias al programa MATLAB, cuya licencia estándar es anual y tiene un coste de 800€ [31]. De esta forma, considerando que su período de amortización es de 1 año y siguiendo el mismo procedimiento que antes, se obtiene un coste de 0.398€/h.

Por otra parte, los controladores se implementan en el programa SoMachine, versión 4.3, cuya licencia estándar es gratuita. Por tanto, no es posible computar un coste horario de la misma.

Finalmente, la redacción de los documentos que justifiquen el desarrollo del proyecto se realiza en el programa Microsoft Office, cuya licencia estándar es anual y tiene un coste de 69€ [32], por lo que considerando un período de amortización de 1 año y operando como anteriormente se obtiene un coste de 0.03€/h.

Tabla 13. Cuadro de precios básicos (Elaboración propia)

Código	Unidad	Descripción	Precio (€)
O1	h	Ingeniero Industrial	30,00
H1	h	Ordenador personal	0,09
H2	ud	Autómata programable	426,94
H3	ud	Raspberry Pi	29,63
S1	h	Licencia de MATLAB	0,398
S2	h	Licencia de Microsoft Office	0,03
S3	h	Licencia de SoMachine	0,00

8.3. Cuadro de precios unitarios descompuestos

Haciendo uso de los precios básicos mostrados anteriormente, se muestran las unidades de obra que han dado lugar a la realización del proyecto, todas ellas distinguidas por capítulos. También se ha considerado un 2% en concepto de costes directos complementarios en cada una de estas unidades, de manera que se puedan contabilizar algunos gastos secundarios como la conexión a Internet, la luz u otros materiales fungibles.

8.3.1. Capítulo 1: Trabajos previos

En el capítulo 1 [Tabla 14] se han tenido en cuenta los trabajos previos a la elaboración del proyecto, los cuales consisten en el estudio y aprendizaje de los programas MATLAB y SoMachine, de manera que se le pueda sacar el máximo rendimiento a sus diferentes funcionalidades y se agilicen ciertos cálculos. Para ello, el Ingeniero Industrial encargado de implementar el sistema de control debe utilizar su ordenador personal y formarse en el uso de estos softwares adquiridos previamente.

Tabla 14. Unidades de obra del capítulo 1 (Elaboración propia)

Código	Ud.	Descripción	Rdto.	Precio	Importe (€)
CAPÍTULO 1: Trabajos previos					
1.1	h	Estudio del programa MATLAB			
Estudio y aprendizaje del programa MATLAB y de sus diferentes funcionalidades					
O1	h	Ingeniero Industrial	1,00	30,00	30,00
H1	h	Ordenador personal	1,00	0,09	0,09
S1	h	Licencia de MATLAB	1,00	0,40	0,40
CD Comp	%	Costes Directos Complementarios	0,02	30,49	0,61
			Coste Total		**31,10**
1.2	h	Estudio del programa SoMachine			
Estudio y aprendizaje del programa SoMachine y de sus diferentes funcionalidades					
O1	h	Ingeniero Industrial	1,00	30,00	30,00
H1	h	Ordenador personal	1,00	0,09	0,09
S3	h	Licencia de SoMachine	1,00	0,00	0,00
CD Comp	%	Costes Directos Complementarios	0,02	30,09	0,60
			Coste Total		**30,70**

8.3.2. Capítulo 2: Elaboración del proyecto

El capítulo 2 [Tabla 15] se ha destinado a la elaboración del proyecto, y se divide en tres subcapítulos:

- Fase de simulación, donde el ingeniero hace uso de su ordenador personal y diseña y simula en MATLAB los controladores que estime oportunos. Tiene un coste de 31.10€/h.
- Fase experimental, donde el ingeniero implementa en SoMachine los controles que han ofrecido buenos resultados en simulación y establece la comunicación entre el modelo del Segway de la Raspberry Pi y el controlador oportuno del autómata programable. Tiene un coste de 30.60€/h.
- Redacción de la memoria, donde el ingeniero, una vez finalizadas las fases de simulación y experimentación, argumenta y justifica los resultados obtenidos en un documento de texto editado mediante la herramienta Word de Microsoft Office. Tiene un coste de 30.73€/h.
- Sistema de control, donde se considera el coste unitario de los dispositivos hardware utilizados en la implementación del sistema de control. Tiene un coste total de 465.70€/ud.

Tabla 15. Unidades de obra del capítulo 2 (Elaboración propia)

Código	Ud.	Descripción	Rdto.	Precio	Importe (€)
CAPÍTULO 2: Elaboración del proyecto					
2.1	h	Fase de simulación			
		Diseño y simulación de los controles en MATLAB			
O1	h	Ingeniero Industrial	1,00	30,00	30,00
H1	h	Ordenador personal	1,00	0,09	0,09
S1	h	Licencia de MATLAB	1,00	0,40	0,40
CD Comp	%	Costes Directos Complementarios	0,02	30,49	0,61
			Coste Total		**31,10**
2.2	h	Fase experimental			
		Experimentación con los controles mediante el uso del PLC y la Raspberry Pi			
O1	h	Ingeniero Industrial	1,00	30,00	30,00
S3	h	Licencia de SoMachine	1,00	0,00	0,00
CD Comp	%	Costes Directos Complementarios	0,02	30,00	0,60
			Coste Total		**30,60**
2.3	h	Redacción de la memoria			
		Redacción de los diferentes documentos del proyecto			
O1	h	Ingeniero Industrial	1,00	30,00	30,00
H1	h	Ordenador personal	1,00	0,09	0,09
S2	h	Licencia de Microsoft Office	1,00	0,03	0,03
CD Comp	%	Costes Directos Complementarios	0,02	30,13	0,60
			Coste Total		**30,73**
2.4	ud	Sistema de control			
		Dispositivos utilizados en la implementación del sistema de control			
H2	ud	Autómata programable	1,00	426,94	426,94
H3	ud	Raspberry Pi	1,00	29,63	29,63
CD Comp	%	Costes Directos Complementarios	0,02	456,57	9,13
			Coste Total		**465,70**

8.4. Mediciones y cuadro de presupuestos parciales

A continuación, se presentan los presupuestos parciales [Tabla 16] de cada capítulo mencionado anteriormente, los cuales se han calculado teniendo en cuenta que se han invertido un total de 300 horas en el desarrollo y ejecución de este proyecto y que se hace uso de un único sistema de control.

Como puede apreciarse, se dedicó un 11.67% y un 15% del tiempo total al estudio de MATLAB y SoMachine, respectivamente, siendo esta diferencia debida a que el autor estaba más familiarizado con un programa que con otro, siendo básicamente un inexperto en el uso de lenguaje ST. No obstante, el ingeniero también requirió formarse en otras herramientas internas de MATLAB que facilitasen la conversión de un lenguaje a otro, motivo por el que su porcentaje es ciertamente elevado.

Por otra parte, la elaboración del proyecto ha ocupado un 73.33% del tiempo total, donde se necesitó

invertir más tiempo en la fase experimental que en la de simulación debido a problemas de cuantificación y de tiempos de computación inesperados. No obstante, la fase de redacción de la memoria ha resultado ser la más costosa de todo el proyecto.

Tabla 16. Cuadro de presupuestos parciales (Elaboración propia)

Código	Ud.	Descripción	Medición	Precio	Importe (€)
CAPÍTULO 1: Trabajos previos					
1.1	h	Estudio del programa MATLAB	35,00	31,10	1088,55
1.2	h	Estudio del programa SoMachine	45,00	30,70	1381,28
			Coste Total		**2469,84**
CAPÍTULO 2: Elaboración del proyecto					
2.1	h	Fase de simulación	65,00	31,10	2021,60
2.2	h	Fase experimental	75,00	30,60	2295,00
2.3	h	Redacción de la memoria	80,00	30,73	2458,42
2.4	ud	Sistema de control	1,00	465,70	465,70
			Coste Total		**7240,72**

8.5. Resumen del presupuesto

Finalmente, se muestra un resumen del presupuesto total del proyecto [Tabla 17], en el cual se realiza la suma total de los diferentes capítulos, obteniendo así el Presupuesto de Ejecución Material (PEM).

Posteriormente, se aplica sobre el PEM un 13% y un 6% en concepto de gastos generales y beneficio industrial, respectivamente, siendo la suma de estos tres el Presupuesto de Ejecución por Contrata (PEC).

Finalmente, se aplica un IVA del 21% sobre el PEC, alcanzando así un Presupuesto Base de Licitación (PBL) total que asciende a:

TRECE MIL NOVECIENTOS OCHENTA Y DOS EUROS CON VEINTITRÉS CÉNTIMOS.

Tabla 17. Resumen del presupuesto total del proyecto (Elaboración propia)

RESUMEN	
CAPÍTULO 1: Trabajos previos	2.469,84 €
CAPÍTULO 2: Elaboración del proyecto	7.240,72 €
PRESUPUESTO DE EJECUCIÓN MATERIAL (PEM)	**9.710,56 €**
GASTOS GENERALES (13%)	1.262,37 €
BENEFICIO INDUSTRIAL (6%)	582,63 €
PRESUPUESTO DE EJECUCIÓN POR CONTRATA (PEC)	**11.555,56 €**
IVA (21%)	2.426,67 €
PRESUPUESTO BASE DE LICITACIÓN (PBL)	**13.982,23 €**

Parte IV

Anexos

Anexo I: Scripts

1. Capítulo 2: Segway

1.1. Script "main.py"

```
# IMPORTACION DE LIBRERIAS Y PROCESOS
import time
…
import segway_odeint
# CONFIGURACION DE LA RPI
#TAD
print("HIL Started")
bus = SMBus(1)
# Configuracion direcciones i2c
adc = MCP3424(0x68, 0x68, 12) #hay que poner la misma direccion repetida en el adc para que no de error
dac = MCP4728(0x60)
### Print adc, dac
GPIO.setup(5, GPIO.OUT, initial=GPIO.LOW) #dac en modo continuo
dig = Adafruit_MCP230XX(address = 0x20, num_gpios = 8) # MCP23008
# Set pins 0 y 5 como entradas
push1 = 0
dig.config(push1, dig.INPUT)
dig.pullup(push1,1)
push2 = 5
dig.config(push2, dig.INPUT)
dig.pullup(push2,1)
# El resto como salidas
led1 = 1
dig.config(led1, dig.OUTPUT)
led2_rojo = 2
dig.config(2, dig.OUTPUT)
led2_azul = 3
dig.config(3, dig.OUTPUT)
led2_verde = 4
dig.config(4, dig.OUTPUT)
# El pin 6 y 7 estan sin uso
dig.config(6, dig.OUTPUT)
dig.config(7, dig.OUTPUT)
for i in [1,2,3,4]:
    dig.output(i, 0)
toggle = 0 # Para hacer parpadear led run
adc1 = 1
adc2 = 2
dac1 = 1
dac2 = 2
# APERTURA DE FICHEROS DONDE SE ALMACENAN VARIABLES
##valores = open('controlPID_50ms.txt','w')
…
##valores.close()
# FUNCION PRINCIPAL
def HILTask():
    global toggle
    time_in = time.time()
    # Aqui va el controlador y todo lo que tenga que hacer
```

```
    if(globals.reset == 1):
        rc_euler.reset()
        …
        segway_odeint.reset()
        time.sleep(0.5)
        globals.reset=0
    if (globals.run == 1):
        if(globals.proceso == 'rc'):
            globals.uv = adc.read_voltage(adc1)
            globals.u = rc_euler.escala_u(globals.uv)
            globals.y = rc_euler.step(globals.u)
            globals.yv = rc_euler.escala_y(globals.y)
            dac.single_internal(dac1,globals.yv,False)
            Ts = rc_euler.Ts
        …
        elif (globals.proceso == 'segway'):
            start_time = time.time()
            # LECTURA DE TENSIONES
            globals.Fv = adc.read_voltage(adc1)
            globals.tauv = adc.read_voltage(adc2)
            # FUNCION escala_u
            globals.F, globals.tau = segway_odeint.escala_u(globals.Fv, globals.tauv, globals.Fmax,
globals.taumax, globals.Fmin, globals.taumin)
            # APLICACIÓN DE ACCION DE CONTROL
            if globals.tiempo <= 10.0:
               globals.F = 0.0
               globals.tau = 0.0
            else:
               globals.F = globals.F - segway_odeint.media_F
               globals.tau = globals.tau - segway_odeint.media_tau
            print("Fuerza de traccion: "+str(globals.F)+" N"+"\n")
            print("Par motor: "+str(globals.tau)+" N m"+"\n")
            # FUNCION step
            globals.vellin, globals.posang = segway_odeint.step(globals.F, globals.tau)
            print("Velocidad lineal: "+str(globals.vellin)+" m/s"+"\n")
            print("Posicion angular: "+str(globals.posang)+" grados"+"\n")
            # ALMACENAMIENTO DE VARIABLES EN FICHEROS
##          with open("controlPID_50ms.txt",'a') as valores:
##          with open("controlPID_60ms.txt",'a') as valores:
##          …
##             if globals.tiempo <= 50.0:
##                valores.write(str(globals.tiempo)+' '+str(globals.F)+' '+str(globals.tau)+'
'+str(globals.vellin)+' '+str(globals.posang)+'\n')
            # FUNCION escala_y
            globals.vellinv, globals.posangv = segway_odeint.escala_y(globals.vellin, globals.posang,
globals.vellin_max, globals.posang_max, globals.vellin_min, globals.posang_min)
            # ENVIO DE TENSIONES
            dac.single_internal(dac1, globals.vellinv, False)
            dac.single_internal(dac2, globals.posangv, False)
            # ACTUALIZACION DEL TIEMPO
            Ts = segway_odeint.Ts
            globals.z = globals.z + 1.0
            globals.tiempo = Ts*globals.z
            print("Tiempo: "+str(globals.tiempo)+" seg"+"\n")
        else:
            print ("Sin proceso")
            Ts = 1
    else:
        Ts = 1
    if toggle == 1:
        dig.output(led1, 1)
        toggle=0
    else:
        dig.output(led1, 0)
        toggle = 1
    tnow = time.time()
    delaytime = Ts - (tnow-time_in)#Basetime 1 second,
    timer = Timer(delaytime, HILTask)
    timer.start()

HILTask()
Web = WebServer(8000)
Web.Start()
```

Script 33. Script “main.py”

2. Capítulo 3: Control del Segway mediante controladores PID

2.1. Script “param_Segway_PID.mlx”

1. Parámetros y simulación en MATLAB

```
clearvars; close all;
M = 2.5; m = 0.6; l = 1.2; r = 0.2; g = 9.81;
escalon = 10;
graf_MAT = input("¿Mostrar SÓLO los resultados de los diferentes Ts en MATLAB?: ",'s'); % Responder
SI en caso de desear mostrarlos
if graf_MAT ~= "SI"
    Ts = input("Elija un período de muestreo: ");
    switch Ts
        case 0.05
            Kp11 = 5.625; Ki11 = 0.3637; Kp12 = -1.303; Kp22 = -56.9300; Ki22 = -52.3083; Kd22 = -
9.7622; tiempo = 30.0;
        case 0.06
            Kp11 = 5.625; Ki11 = 0.3637; Kp12 = -1.2; Kp22 = -56.9300; Ki22 = -52.3083; Kd22 = -
9.7622; tiempo = 30.0;
        case 0.07
            Kp11 = 5.625; Ki11 = 0.3637; Kp12 = -1.2; Kp22 = -45.432; Ki22 = -48.703; Kd22 = -
9.2775; tiempo = 25.0;
        otherwise
            disp("Período de muestreo no disponible");
    end
    sim("Segway_ModeloNL_PID.slx");
    tiempo_MAT = Vel.Time;
    Ref_vel = Ref_vel.Data;
    Ref_ang = Ref_ang.Data;
    Vel_MAT = Vel.Data;
    Ang_MAT = Ang.Data;
    F_MAT = F.Data;
    Tau_MAT = Tau.Data;
elseif graf_MAT == "SI"
    Ts_v = [0.05; 0.06; 0.07];
    for i = 1:1:3
        Ts = Ts_v(i);
        switch Ts
        case 0.05
            Kp11 = 5.625; Ki11 = 0.3637; Kp12 = -1.303; Kp22 = -56.9300; Ki22 = -52.3083; Kd22 = -
9.7622; tiempo = 30.0;
        case 0.06
            Kp11 = 5.625; Ki11 = 0.3637; Kp12 = -1.2; Kp22 = -56.9300; Ki22 = -52.3083; Kd22 = -
9.7622; tiempo = 30.0;
        case 0.07
            Kp11 = 5.625; Ki11 = 0.3637; Kp12 = -1.2; Kp22 = -45.432; Ki22 = -48.703; Kd22 = -
9.2775; tiempo = 25.0;
        end
        sim("Segway_ModeloNL_PID.slx");
        if Ts == 0.05
            tiempo_MAT_50 = Vel.Time;
            Ref_vel_50 = Ref_vel.Data;
            Ref_ang_50 = Ref_ang.Data;
            Vel_MAT_50 = Vel.Data;
            Ang_MAT_50 = Ang.Data;
            F_MAT_50 = F.Data;
            Tau_MAT_50 = Tau.Data;
        elseif Ts == 0.06
            tiempo_MAT_60 = Vel.Time;
            Vel_MAT_60 = Vel.Data;
            Ang_MAT_60 = Ang.Data;
            F_MAT_60 = F.Data;
            Tau_MAT_60 = Tau.Data;
        elseif Ts == 0.07
            tiempo_MAT_70 = Vel.Time;
            Vel_MAT_70 = Vel.Data;
            Ang_MAT_70 = Ang.Data;
            F_MAT_70 = F.Data;
```

```
            Tau_MAT_70 = Tau.Data;
        end
    end
end
```

2. Resultados

```
if graf_MAT ~= "SI"
    switch Ts
        case 0.05
            fileID = fopen('PID_Python_50ms.txt','r');
            fileID2 = fopen('PID_50ms.txt','r');
        case 0.06
            fileID = fopen('PID_Python_60ms.txt','r');
            fileID2 = fopen('PID_60ms.txt','r');
        case 0.07
            fileID = fopen('PID_Python_70ms.txt','r');
            fileID2 = fopen('PID_70ms.txt','r');
    end
    formatSpec = '%f %f %f %f %f';
    sizeA = [5 inf];

    valores_Python = fscanf(fileID,formatSpec,sizeA);
    tiempo_PY = valores_Python(1,:)';
    F_PY = valores_Python(2,:)';
    Tau_PY = valores_Python(3,:)';
    Vel_PY = valores_Python(4,:)';
    Ang_PY = valores_Python(5,:)';
    fclose(fileID);

    valores_RPI = fscanf(fileID2,formatSpec,sizeA);
    tiempo_RPI = valores_RPI(1,:)';
    F_RPI = valores_RPI(2,:)';
    Tau_RPI = valores_RPI(3,:)';
    Vel_RPI = valores_RPI(4,:)';
    Ang_RPI = valores_RPI(5,:)';
    fclose(fileID2);

    graf_PY = input("¿Mostrar resultados de Python?: ",'s'); % Responder SI en caso de desear
mostrarlos
    graf_RPI = input("¿Mostrar resultados de la RPI?: ",'s'); % Responder SI en caso de desear
mostrarlos

    Velocidad = figure(1);
    plot(tiempo_MAT,Ref_vel,'y','LineWidth',1);
    hold on;
    plot(tiempo_MAT,Vel_MAT,'r','LineWidth',2);
    if graf_PY == "SI"
        hold on;
        plot(tiempo_PY,Vel_PY,'b--','LineWidth',2);
    end
    if graf_RPI == "SI"
        hold on;
        plot(tiempo_RPI,Vel_RPI,'g--','LineWidth',2);
    end
    title('Velocidad lineal (m/s)');
    if (graf_PY == "SI") && (graf_RPI == "SI")
        legend('Referencia','MATLAB','PYTHON','RPI','Location','Best');
    elseif (graf_PY == "SI") && (graf_RPI == "NO")
        legend('Referencia','MATLAB','PYTHON','Location','Best');
    elseif (graf_PY == "NO") && (graf_RPI == "SI")
        legend('Referencia','MATLAB','RPI','Location','Best');
    elseif (graf_PY == "NO") && (graf_RPI == "NO")
        legend('Referencia','MATLAB','Location','Best');
    end
    hold off;
    Angulo = figure(2);
    plot(tiempo_MAT,Ref_ang,'y','LineWidth',1);
    hold on;
```

```
    plot(tiempo_MAT,Ang_MAT,'r','LineWidth',2);
    if graf_PY == "SI"
        hold on;
        plot(tiempo_PY,Ang_PY,'b--','LineWidth',2);
    end
    if graf_RPI == "SI"
        hold on;
        plot(tiempo_RPI,Ang_RPI,'g--','LineWidth',2);
    end
    title('Posición angular (grados)');
    if (graf_PY == "SI") && (graf_RPI == "SI")
        legend('Referencia','MATLAB','PYTHON','RPI','Location','Best');
    elseif (graf_PY == "SI") && (graf_RPI == "NO")
        legend('Referencia','MATLAB','PYTHON','Location','Best');
    elseif (graf_PY == "NO") && (graf_RPI == "SI")
        legend('Referencia','MATLAB','RPI','Location','Best');
    elseif (graf_PY == "NO") && (graf_RPI == "NO")
        legend('Referencia','MATLAB','Location','Best');
    end
    hold off;
    Fuerza = figure(3);
    plot(tiempo_MAT,F_MAT,'r','LineWidth',2);
    if graf_PY == "SI"
        hold on;
        plot(tiempo_PY,F_PY,'b--','LineWidth',2);
    end
    if graf_RPI == "SI"
        hold on;
        plot(tiempo_RPI,F_RPI,'g--','LineWidth',2);
    end
    title('Fuerza de tracción (N)');
    if (graf_PY == "SI") && (graf_RPI == "SI")
        legend('MATLAB','PYTHON','RPI','Location','Best');
    elseif (graf_PY == "SI") && (graf_RPI == "NO")
        legend('MATLAB','PYTHON','Location','Best');
    elseif (graf_PY == "NO") && (graf_RPI == "SI")
        legend('MATLAB','RPI','Location','Best');
    elseif (graf_PY == "NO") && (graf_RPI == "NO")
        legend('MATLAB','Location','Best');
    end
    hold off;
    Par = figure(4);
    plot(tiempo_MAT,Tau_MAT,'r','LineWidth',2);
    if graf_PY == "SI"
        hold on;
        plot(tiempo_PY,Tau_PY,'b--','LineWidth',2);
    end
    if graf_RPI == "SI"
        hold on;
        plot(tiempo_RPI,Tau_RPI,'g--','LineWidth',2);
    end
    title('Par motor (N m)');
    if (graf_PY == "SI") && (graf_RPI == "SI")
        legend('MATLAB','PYTHON','RPI','Location','Best');
    elseif (graf_PY == "SI") && (graf_RPI == "NO")
        legend('MATLAB','PYTHON','Location','Best');
    elseif (graf_PY == "NO") && (graf_RPI == "SI")
        legend('MATLAB','RPI','Location','Best');
    elseif (graf_PY == "NO") && (graf_RPI == "NO")
        legend('MATLAB','Location','Best');
    end
    hold off;
elseif graf_MAT == "SI"
    Velocidad = figure(1);
    plot(tiempo_MAT_50,Ref_vel_50,'y','LineWidth',1); hold on;
    plot(tiempo_MAT_50,Vel_MAT_50,'r','LineWidth',2); hold on;
    plot(tiempo_MAT_60,Vel_MAT_60,'b','LineWidth',2); hold on;
    plot(tiempo_MAT_70,Vel_MAT_70,'g','LineWidth',2);
    title('Velocidad lineal (m/s)');
    legend('Referencia','50ms','60ms','70ms','Location','Best'); hold off;
```

```
    Angulo = figure(2);
    plot(tiempo_MAT_50,Ref_ang_50,'y','LineWidth',1); hold on;
    plot(tiempo_MAT_50,Ang_MAT_50,'r','LineWidth',2); hold on;
    plot(tiempo_MAT_60,Ang_MAT_60,'b','LineWidth',2); hold on;
    plot(tiempo_MAT_70,Ang_MAT_70,'g','LineWidth',2);
    title('Posición angular (grados)');
    legend('Referencia','50ms','60ms','70ms','Location','Best'); hold off;

    Fuerza = figure(3);
    plot(tiempo_MAT_50,F_MAT_50,'r','LineWidth',2); hold on;
    plot(tiempo_MAT_60,F_MAT_60,'b','LineWidth',2); hold on;
    plot(tiempo_MAT_70,F_MAT_70,'g','LineWidth',2);
    title('Fuerza de tracción (N)');
    legend('50ms','60ms','70ms','Location','Best'); hold off;

    Par = figure(4);
    plot(tiempo_MAT_50,Tau_MAT_50,'r','LineWidth',2); hold on;
    plot(tiempo_MAT_60,Tau_MAT_60,'b','LineWidth',2); hold on;
    plot(tiempo_MAT_70,Tau_MAT_70,'g','LineWidth',2);
    title('Par motor (N m)');
    legend('50ms','60ms','70ms','Location','Best'); hold off;
end
```

Script 34. Script "param_Segway_PID.mlx"

2.2. Proyecto "Segway_SoMachine_PID.project"

```
// DECLARACIÓN DE VARIABLES
PROGRAM POU
VAR
	Ref: ARRAY[0..1] OF LREAL := [1.0, 0.0];
	Y_k_volt: ARRAY[0..1] OF LREAL;
	Y_k: ARRAY[0..1] OF LREAL;
	E_k: ARRAY[0..1] OF LREAL;
	U_k: ARRAY[0..1] OF LREAL;
	U_k_volt: ARRAY[0..1] OF LREAL;
	U11_k: LREAL;
	U12_k: LREAL;
	U22_k: LREAL;
	Kp11: LREAL;
	Ki11: LREAL;
	Kp12: LREAL;
	Kp22: LREAL;
	Ki22: LREAL;
	Kd22: LREAL;
	Ts: LREAL := 0.06;
	manual: BOOL := TRUE;
	iter1: BOOL := TRUE;
	Integrador_DSTATE11: LREAL;
	rtb_Tsamp22: LREAL;
	Integrador_DSTATE22: LREAL;
	UD_DSTATE22: LREAL;
	U_max: ARRAY[0..1] OF LREAL := [7.0, 0.2];
	U_min: ARRAY[0..1] OF LREAL := [-0.1, -2.0];
	Y_max: ARRAY[0..1] OF LREAL := [1.2, 0.3];
	Y_min: ARRAY[0..1] OF LREAL := [-0.1, -0.3];
	j: INT := 0;
	Y_media: ARRAY [0..1] OF LREAL := [-0.0021492600000000738, 0.0057021600000000047];
	Y_suma: ARRAY [0..1] OF LREAL := [0.0, 0.0];
	Y_media_calculo: ARRAY [0..1] OF LREAL;
END_VAR
// ELECCIÓN DE PARÁMETROS DE LOS PID
IF iter1 = TRUE THEN
	IF Ts = 0.05 THEN
		Kp11 := 5.625;
		Ki11 := 0.3637;
		Kp12 := -1.303;
		Kp22 := -56.9300;
		Ki22 := -52.3083;
		Kd22 := -9.7622;
	ELSIF Ts = 0.06 THEN
		Kp11 := 5.625;
```

```
			Ki11 := 0.3637;
			Kp12 := -1.2;
			Kp22 := -56.9300;
			Ki22 := -52.3083;
			Kd22 := -9.7622;
		ELSIF Ts = 0.07 THEN
			Kp11 := 5.625;
			Ki11 := 0.3637;
			Kp12 := -1.2;
			Kp22 := -45.432;
			Ki22 := -48.703;
			Kd22 := -9.2775;
		END_IF
		iter1 := FALSE;
END_IF
// LECTURA DE LAS VARIABLES CONTROLADAS EN VOLTIOS
Y_k_volt[0] := WORD_TO_LREAL(GVL.VC1_volt)/1000.0;
Y_k_volt[1] := WORD_TO_LREAL(GVL.VC2_volt)/1000.0;
// CÁLCULO DE LAS MEDIAS DE LAS VARIABLES CONTROLADAS
// Cambio en las variables controladas de voltios a sus respectivas unidades (m/s, grados)
//Y_k[0] := (Y_max[0] - ((10.0-Y_k_volt[0])*(Y_max[0]-Y_min[0]))/10.0);
//Y_k[1] := (Y_max[1] - ((10.0-Y_k_volt[1])*(Y_max[1]-Y_min[1]))/10.0);
//IF j <= 499 THEN
		//Y_suma[0] := Y_suma[0] + Y_k[0];
		//Y_suma[1] := Y_suma[1] + Y_k[1];
		//j := j + 1;
//END_IF
//IF j = 500 THEN
		//Y_media_calculo[0] := Y_suma[0]/500.0;
		//Y_media_calculo[1] := Y_suma[1]/500.0;
		//j := j + 1;
//END_IF
// CONVERSIÓN DE LAS VARIABLES CONTROLADAS DE VOLTIOS A SUS UNIDADES
// Cambio en las variables controladas de voltios a sus respectivas unidades (m/s, grados)
Y_k[0] := (Y_max[0] - ((10.0-Y_k_volt[0])*(Y_max[0]-Y_min[0]))/10.0) - Y_media[0];
Y_k[1] := ((Y_max[1] - ((10.0-Y_k_volt[1])*(Y_max[1]-Y_min[1]))/10.0) - Y_media[1]);
// ARRANQUE DEL CONTROLADOR
IF manual = TRUE THEN
		// Imposición inicial en las acciones de control (N, N m) para evitar que el modelo se inestabilice
		U_k[0] := 0.0;
		U_k[1] := 0.0;

		// Cambio en las acciones de control de sus respectivas unidades (N, N m) a voltios
		U_k_volt[0] := 10.0 - (10.0*(U_max[0]-U_k[0]))/(U_max[0]-U_min[0]);
		U_k_volt[1] := 10.0 - (10.0*(U_max[1]-U_k[1]))/(U_max[1]-U_min[1]);

		// Escritura de las acciones de control en voltios
		GVL.AC1_volt := LREAL_TO_WORD(U_k_volt[0]*1000.0);
		GVL.AC2_volt := LREAL_TO_WORD(U_k_volt[1]*1000.0);
ELSIF manual = FALSE THEN
		// Anti-windup de las variables controladas
		IF Y_k[0] > Y_max[0] THEN
			Y_k[0] := Y_max[0];
		ELSIF Y_k[0] < Y_min[0] THEN
			Y_k[0] := Y_min[0];
		END_IF
		IF Y_k[1] > Y_max[1] THEN
			Y_k[1] := Y_max[1];
		ELSIF Y_k[1] < Y_min[1] THEN
			Y_k[1] := Y_min[1];
		END_IF
		// Conversión de grados sexagesimales a radianes
		Y_k[1] := Y_k[1]*(3.14159265358979323846264338327950288419716939937510/180.0);
		// Cálculo del error entre referencia y salida
		E_k[0] := Ref[0] - Y_k[0];
		E_k[1] := Ref[1] - Y_k[1];
		// Cálculo de la acción de control "fuerza de tracción (F)"
		U11_k := Kp11*E_k[0] + Integrador_DSTATE11;
		Integrador_DSTATE11 := (Ki11*Ts*E_k[0]) + Integrador_DSTATE11;
		U12_k := Kp12*E_k[0];
		U_k[0] := U11_k;
		// Cálculo de la acción de control "par motor (tau)"
		U12_k := Kp12*E_k[0];
```

```
        rtb_Tsamp22 := (Kd22/Ts)*E_k[1];
        U22_k := (Kp22*E_k[1] + Integrador_DSTATE22) + (rtb_Tsamp22 - UD_DSTATE22);
        UD_DSTATE22 := rtb_Tsamp22;
        Integrador_DSTATE22 := (Ki22*Ts*E_k[1]) + Integrador_DSTATE22;
        U_k[1] := U12_k + U22_k;
        // Anti-windup de las acciones de control
        IF U_k[0] > U_max[0] THEN
               U_k[0] := U_max[0];
        ELSIF U_k[0] < U_min[0] THEN
               U_k[0] := U_min[0];
        END_IF

        IF U_k[1] > U_max[1] THEN
               U_k[1] := U_max[1];
        ELSIF U_k[1] < U_min[1] THEN
               U_k[1] := U_min[1];
        END_IF
        // Cambio en las acciones de control de sus respectivas unidades (N, N m) a voltios
        U_k_volt[0] := 10.0 - (10.0*(U_max[0]-U_k[0]))/(U_max[0]-U_min[0]);
        U_k_volt[1] := 10.0 - (10.0*(U_max[1]-U_k[1]))/(U_max[1]-U_min[1]);
        // Escritura de las acciones de control en voltios
        GVL.AC1_volt := LREAL_TO_WORD(U_k_volt[0]*1000.0);
        GVL.AC2_volt := LREAL_TO_WORD(U_k_volt[1]*1000.0);
END_IF
```

Script 35. Script “Segway_SoMachine_PID.mlx”

2.3. Script “PID_SoMachine.mlx”

PIDs en SoMachine

```
clearvars; close all;
fileID1 = fopen('PID_50ms.txt','r');
fileID2 = fopen('PID_60ms.txt','r');
fileID3 = fopen('PID_70ms.txt','r');
escalon = 10;
tiempo = 30;

sim("Refs_PID.slx");
tiempo_MAT = Ref_vel.Time;
Ref_vel = Ref_vel.Data;
Ref_ang = Ref_ang.Data;

formatSpec = '%f %f %f %f %f';
sizeA = [5 inf];

valores_RPI_50ms = fscanf(fileID1,formatSpec,sizeA);
tiempo_RPI_50ms = valores_RPI_50ms(1,:)';
F_RPI_50ms = valores_RPI_50ms(2,:)';
Tau_RPI_50ms = valores_RPI_50ms(3,:)';
Vel_RPI_50ms = valores_RPI_50ms(4,:)';
Ang_RPI_50ms = valores_RPI_50ms(5,:)';

valores_RPI_60ms = fscanf(fileID2,formatSpec,sizeA);
tiempo_RPI_60ms = valores_RPI_60ms(1,:)';
F_RPI_60ms = valores_RPI_60ms(2,:)';
Tau_RPI_60ms = valores_RPI_60ms(3,:)';
Vel_RPI_60ms = valores_RPI_60ms(4,:)';
Ang_RPI_60ms = valores_RPI_60ms(5,:)';

valores_RPI_70ms = fscanf(fileID3,formatSpec,sizeA);
tiempo_RPI_70ms = valores_RPI_70ms(1,:)';
F_RPI_70ms = valores_RPI_70ms(2,:)';
Tau_RPI_70ms = valores_RPI_70ms(3,:)';
Vel_RPI_70ms = valores_RPI_70ms(4,:)';
Ang_RPI_70ms = valores_RPI_70ms(5,:)';

Velocidad = figure(1);
plot(tiempo_MAT,Ref_vel,'y','LineWidth',1); hold on;
plot(tiempo_RPI_50ms,Vel_RPI_50ms,'r','LineWidth',1.5); hold on;
plot(tiempo_RPI_60ms,Vel_RPI_60ms,'b','LineWidth',1.5); hold on;
plot(tiempo_RPI_70ms,Vel_RPI_70ms,'g','LineWidth',1.5);
title('Velocidad lineal (m/s)');
legend('Referencia','50ms','60ms','70ms','Location','Best'); hold off;

Angulo = figure(2);
plot(tiempo_MAT,Ref_ang,'y','LineWidth',1); hold on;
plot(tiempo_RPI_50ms,Ang_RPI_50ms,'r','LineWidth',1.5); hold on;
plot(tiempo_RPI_60ms,Ang_RPI_60ms,'b','LineWidth',1.5); hold on;
plot(tiempo_RPI_70ms,Ang_RPI_70ms,'g','LineWidth',1.5);
title('Posición angular (grados)');
legend('Referencia','50ms','60ms','70ms','Location','Best'); hold off;

Fuerza = figure(3);
plot(tiempo_RPI_50ms,F_RPI_50ms,'r','LineWidth',1.5); hold on;
plot(tiempo_RPI_60ms,F_RPI_60ms,'b','LineWidth',1.5); hold on;
plot(tiempo_RPI_70ms,F_RPI_70ms,'g','LineWidth',1.5);
title('Fuerza de tracción (N)'); legend('50ms','60ms','70ms','Location','Best'); hold off;

Par = figure(4);
plot(tiempo_RPI_50ms,Tau_RPI_50ms,'r','LineWidth',1.5); hold on;
plot(tiempo_RPI_60ms,Tau_RPI_60ms,'b','LineWidth',1.5); hold on;
plot(tiempo_RPI_70ms,Tau_RPI_70ms,'g','LineWidth',1.5);
title('Par motor (N m)'); legend('50ms','60ms','70ms','Location','Best'); hold off;
```

Script 36. Script “PID_SoMachine.mlx”

3. Control del Segway mediante State-Space Model Predictive Control

3.1. Función "crea_AlfaM.m"

```
function alfaM=crea_AlfaM(alfa,p,num_y)
    alfaM=zeros(p*num_y);
    for i=1:1:num_y
        alfaM((i-1)*p+1:i*p,(i-1)*p+1:i*p)=alfa(i)*eye(p);
    end
end
```

Script 37. Función "crea_AlfaM.m"

3.2. Función "crea_LambdaM.m"

```
function lambdaM=crea_LambdaM(lambda,c,num_u)
    lambdaM=zeros(c*num_u);
    for j=1:1:num_u
        lambdaM((j-1)*c+1:j*c,(j-1)*c+1:j*c)=lambda(j)*eye(c);
    end
end
```

Script 38. Función "crea_LambdaM.m"

3.3. Función "crea_P.m"

```
function P = crea_P(A,C,num_y,num_x,p)
    P = zeros(p*num_y,num_x);
    for i = 1:1:num_y
        for j = 1:1:p
            P((i-1)*p+j,:) = C(i,:)*A^(j);
        end
    end
end
```

Script 39. Función "crea_P.m"

3.4. Función "crea_Q.m"

```
function Q = crea_Q(A,B,C,num_y,num_u,p)
    Q = zeros(p*num_y,num_u);
    for i = 1:1:num_y
        for j = 1:1:p
            for h = 1:1:j
                Q((i-1)*p+j,:) = Q((i-1)*p+j,:) + C(i,:)*A^(h-1)*B;
            end
        end
    end
end
```

Script 40. Función "crea_Q.m"

3.5. Función "crea_G.m"

```
function G = crea_G(Q,num_y,num_u,p,c)
    G = zeros(p*num_y,c*num_u);
    for i = 1:1:num_y
        for j = 1:1:c
            G((i-1)*p+1:i*p,j:j+1) = [zeros(j-1,num_u); Q((i-1)*p+1:i*p-j+1,:)];
        end
    end
end
```

Script 41. Función "crea_G.m"

Anexo II: Manual de usuario

A continuación, se establece una guía detallada sobre cómo programar un controlador concreto en el PLC mediante el programa SoMachine, versión 4.3:

1. Crear un nuevo proyecto

En primer lugar, ya dentro del programa SoMachine 4.3 debe crearse un nuevo proyecto. Para ello, debe pulsarse sobre "Nuevo proyecto" >> "Proyecto vacío" >> "Crear proyecto" (previamente, se le puede dar un nombre al mismo en la parte superior) >> Diseño de aplicaciones: Controlador: "Programar uno o varios controladores" [Ilustración 106].

Ilustración 106. Cómo crear un controlador en SoMachine (Elaboración propia)

2. Agregar un dispositivo

En segundo lugar, debe añadirse el modelo de autómata programable o PLC con el que se va a trabajar. Para ello, en la ventana "Aplicaciones" debe pulsarse con el botón derecho del ratón sobre el título del proyecto >> "Agregar el dispositivo". En la ventana "Agregar dispositivo" [Ilustración 107], deben escogerse las siguientes opciones:

- Acción: "Agregar el dispositivo".
- Fabricante: "Schneider Electric".
- Controlador lógico >> M241 >> TM241CE40R.

Finalmente, debe pulsarse sobre el botón "Agregar el dispositivo".

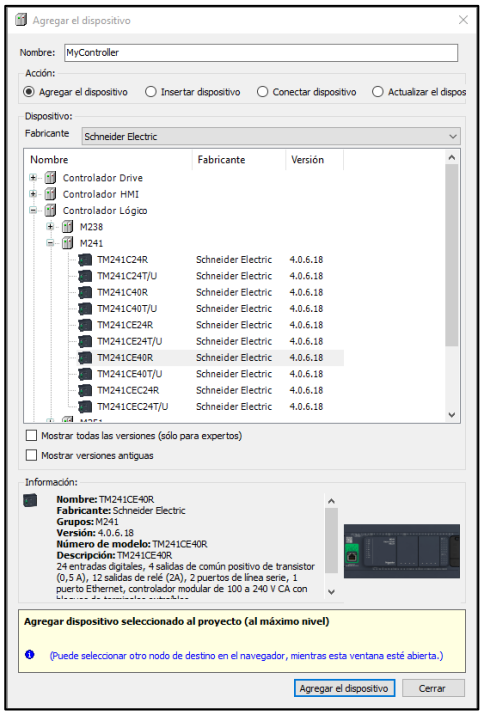

Ilustración 107. Cómo agregar un dispositivo PLC en SoMachine (Elaboración propia)

3. Agregar un POU

En tercer lugar, es necesario añadir una Unidad de Organización del Programa (POU) donde declarar las variables utilizadas y programar el código del controlador. Para ello, en la ventana "Aplicaciones", concretamente en la pestaña "Application (My Controller:…)" >> "Agregar objeto" >> "POU…". En la ventana "Agregar POU" [Ilustración 108] se escogen las siguientes opciones:

- Tipo: "Programa".
- Lenguaje de implementación: "Texto estructurado (ST)". No obstante, existen otros lenguajes tales como Diagrama de Bloques Funcionales o Diagrama de Contactos.

Finalmente, debe pulsarse sobre "Agregar".

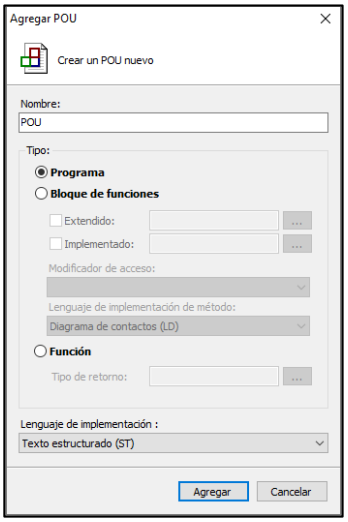

Ilustración 108. Cómo agregar un POU en SoMachine (Elaboración propia)

4. Agregar variables globales

Puesto que se requiere la comunicación con otro dispositivo, es necesario la declaración de variables globales que sean enviadas o leídas. Para ello, en la ventana Aplicaciones, concretamente en la pestaña Application (MyController) >> GVL. En la ventana abierta [Ilustración 109], deben declararse las variables como tipo WORD:

Ámbito	Nombre	Dirección	Tipo de datos	Inicialización	Comentario	Atributos
VAR_GLOBAL	**VC1_volt**		WORD			
VAR_GLOBAL	**VC2_volt**		WORD			
VAR_GLOBAL	**AC1_volt**		WORD			
VAR_GLOBAL	**AC2_volt**		WORD			

Ilustración 109. Cómo declarar variables globales en SoMachine (Elaboración propia)

Tras ello, en la ventana Dispositivos, debe pulsarse con el botón derecho sobre la opción “Cartridge_1 (Cartridge)” >> “Agregar el dispositivo…”. En la ventana abierta [Ilustración 110], dentro de “Módulos de E/S analógicas TMC” >> Entradas >> TMC4AI2. Finalmente, debe pulsarse sobre el botón “Agregar el dispositivo”.

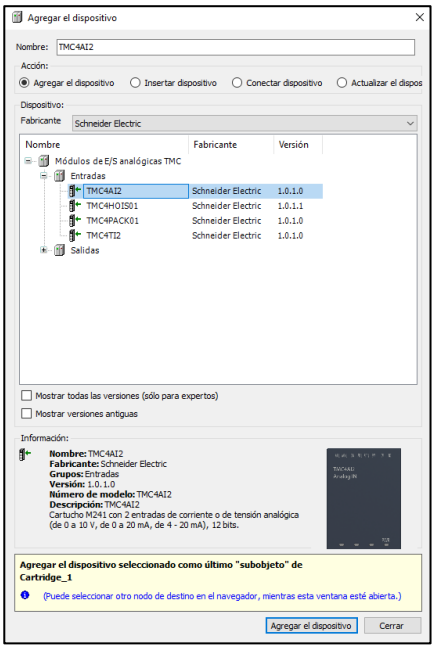

Ilustración 110. Cómo agregar un dispositivo que albergue las entradas del PLC en SoMachine (Elaboración propia)

Una vez agregado, se pulsa sobre “TMC4AI2” >> “Entradas” [Ilustración 111]. Aquí deben agregarse las variables globales que reciben los valores del otro dispositivo para ser leídos en el controlador. En este caso, deben declararse del tipo INT.

Variable	Asignación	Canal	Dirección	Tipo	Valor predeterminado	Unidad	Descripción
Entradas							
Application.VC1_volt		IW0	~~%IW3~~	INT			
Application.VC2_volt		IW1	~~%IW4~~	INT			
Diagnostic							

Ilustración 111. Cómo declarar variables globales como entradas en SoMachine (Elaboración propia)

Análogamente, debe pulsarse con el botón derecho sobre “Cartridge_2 (Cartridge)” >> “Agregar el dispositivo…”. En la ventana abierta, dentro de “Módulos de E/S analógicas TMC” >> Salidas >>

TMC4AQ2. Finalmente, debe pulsarse sobre el botón “Agregar el dispositivo”.

Una vez agregado, se pulsa sobre “TMC4AQ2” >> “Salidas”. Aquí deben agregarse las variables globales utilizadas para enviar los valores del controlador al otro dispositivo, y deben declararse como INT.

5. Programación del POU

Tras acceder a la unidad del POU a través de la ventana “Aplicaciones” >> “Application (MyController)” >> “POU (PRG)”, se abre una nueva pestaña en la que se pueden apreciar dos estructuras claramente diferenciadas. En la parte superior, se declaran las variables locales utilizadas en la programación del controlador [Ilustración 112], donde se deben completar principalmente los campos:

- “Nombre”: nombre de la variable local.
- “Tipo de datos”: se debe asignar el tipo de variable, sea este un entero ‘INT’, real ‘LREAL’, vector o matriz de decimales ‘ARRAY[0..x] OF LREAL’, booleano ‘BOOL’, etc.
- “Inicialización”: donde existe la posibilidad de asignar un valor inicial a las variables.

Ámbito	Nombre	Dirección	Tipo de datos	Inicialización	Comentario	Atributos
VAR	**U_k_volt**		ARRAY[0..1] OF LREAL			
VAR	**U11_k**		LREAL			
VAR	**U12_k**		LREAL			
VAR	**U22_k**		LREAL			
VAR	**Kp11**		LREAL			
VAR	**Ki11**		LREAL			
VAR	**Kp12**		LREAL			
VAR	**Kp22**		LREAL			
VAR	**Ki22**		LREAL			
VAR	**Kd22**		LREAL			
VAR	**Ts**		LREAL	0.06		
VAR	**manual**		BOOL	TRUE		
VAR	**j**		INT	0		
VAR	**Integrador_DSTATE11**		LREAL			
VAR	**rtb_Tsamp22**		LREAL			
VAR	**Integrador_DSTATE22**		LREAL			
VAR	**UD_DSTATE22**		LREAL			
VAR	**U_max**		ARRAY[0..1] OF LREAL	[7.0, 0.2]		

Ilustración 112. Cómo declarar variables locales en SoMachine (Elaboración propia)

Tras declarar las variables, en la parte inferior se realiza la programación del controlador [Ilustración 113], donde se pueden realizar conversiones de variables globales a números reales, utilizar estructuras IF, bucles FOR, etc.

```
// Lectura de las variables controladas en voltios
Y_k_volt[0] := WORD_TO_LREAL(GVL.VC1_volt)/1000.0;
Y_k_volt[1] := WORD_TO_LREAL(GVL.VC2_volt)/1000.0;

// Cambio en las variables controladas de voltios a sus respectivas unidades (m/s, grados)
Y_k[0] := (Y_max[0] - ((10.0-Y_k_volt[0])*(Y_max[0]-Y_min[0]))/10.0);
Y_k[1] := (Y_max[1] - ((10.0-Y_k_volt[1])*(Y_max[1]-Y_min[1]))/10.0);

IF j <= 499 THEN
    Y_suma[0] := Y_suma[0] + Y_k[0];
    Y_suma[1] := Y_suma[1] + Y_k[1];
    j := j + 1;
END_IF
IF j = 500 THEN
    Y_media_calculo[0] := Y_suma[0]/500.0;
    Y_media_calculo[1] := Y_suma[1]/500.0;
    j := j + 1;
END_IF
```

Ilustración 113. Ejemplo de programación del POU en SoMachine (Elaboración propia)

6. Inserción del período de muestreo

Puesto que el sistema de control tiene un carácter cíclico, es necesario insertar un período de muestreo que determine la duración de cada iteración. Para ello, debe pulsarse en la ventana “Aplicaciones” >> “Application (MyController) >> “Configuración de tareas” >> “MAST”. En la ventana abierta [Ilustración 114], deben completarse los siguientes campos:

- Prioridad: la que estime oportuna el usuario.
- Tipo: “Cíclico”.
- Intervalo: período de muestreo escogido, en unidades de milisegundo.
- Watchdog: desactivado.
- “Agregar llamada” >> “Application” >> “POU”.

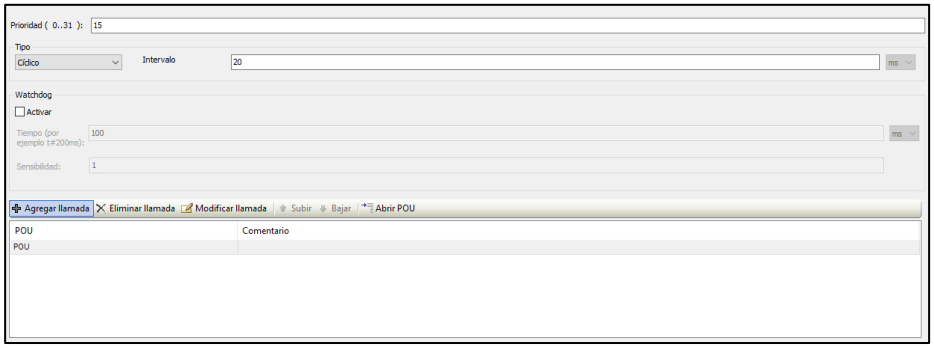

Ilustración 114. Cómo insertar el período de muestreo en SoMachine (Elaboración propia)

De esta forma, el controlador ya está preparado para ser puesto en marcha.

7. Supervisión de variables

Es probable que, durante la utilización del controlador, sea necesario modificar manualmente el valor de algunas variables con el objetivo de cumplir ciertas condiciones. Para ello, es necesario acceder desde la parte superior del programa en “Ver” >> “Supervisar” >> “Supervisar 1”. Tras este paso, se abre una nueva ventana en la parte inferior [Ilustración 115], la cual está formada por los siguientes campos:

- Expresión: donde se escoge la variable a modificar. Para ello, debe pulsarse sobre los puntos suspensivos >> “MyController” >> “Application” >> “POU”.
- Tipo de datos: tipo de variable, aparece asignado según se haya hecho en la declaración.
- Valor: estado actual de la variable.
- Valor preparado: estado al que se desea llevar a la variable. Para ello, debe pulsarse con el botón derecho sobre la casilla >> “Forzar valores”. En el caso de tener valores preparados de diversas variables y pulsar el botón, se modifican todas ellas.

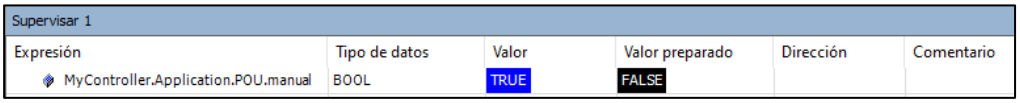

Ilustración 115. Cómo supervisar el valor de una variable en SoMachine (Elaboración propia)