Proyecto Fin de Máster

Ingeniería Electrónica, Robótica y Automática

Diseño y validación de controladores predictivos para un vehículo tipo segway

Autor: Samuel Moya Mateo

Tutor: Daniel Limón Marruedo

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2023

Proyecto Fin de Máster Ingeniería

Electrónica, Robótica y Automática

Diseño y validación de controladores predictivos para un vehículo tipo segway

Autor:

Samuel Moya Mateo

Tutor:

Daniel Limón Marruedo

Catedrático

Dpto. de Ingeniería de Sistemas y Automática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2023

Proyecto Fin de Máster: DISEÑO Y VALIDACIÓN DE CONTROLADORES PREDICTIVOS PARA UN VEHÍCULO TIPO SEGWAY

Autor: Samuel Moya Mateo

Tutor: Daniel Limón Marruedo

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

Presidente:

Vocal/es:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Resumen

Este Proyecto de Final de Máster se centra en continuar con los avances realizados en el diseño y control de un Segway pequeño con capacidad de autoequilibrio. Para lograr esta funcionalidad, se implementarán diversos algoritmos de control, incluyendo el Control Lineal Cuadrático (LQR) y dos variantes del Model Predictive Control (MPC): lineal y no lineal. Estos algoritmos se seleccionan con el objetivo de optimizar el rendimiento del Segway en diferentes condiciones y escenarios.

Además, se emplearán dos herramientas clave en el proceso de diseño y control: SPCIES y CasADI. SPCIES será la herramienta empleada en el desarrollo de un código que permita implementar el MPC lineal en el comportamiento del Segway, mientras que CasADI será la herramienta empleada en la implementación del MPC no lineal.

Mediante diversas simulaciones realizadas en Matlab, se comprobarán y diseñarán los algoritmos de control anteriores comparándolos entre sí.

Otro aspecto a estudiar en este proyecto será la identificación de estados, permitiendo la obtención de matrices del espacio de estados del sistema. Estas matrices se utilizarán para ajustar y validar los parámetros del Segway, verificando su capacidad para corregir errores de equilibrio de manera efectiva.

En resumen, este trabajo aborda la tarea de diseñar un Segway con capacidad de autoequilibrarse, integrando algoritmos de control avanzados y herramientas especializadas. La identificación de estados y la validación experimental contribuirán a evaluar la eficacia del sistema en situaciones prácticas, respaldando así la robustez y rendimiento del Segway desarrollado.

Abstract

This Master's Final Project focuses on continuing the progress made in the design and control of a small self-balancing Segway. To achieve this functionality, various control algorithms will be implemented, including Linear Quadratic Regulation (LQR) and two variants of Model Predictive Control (MPC): linear and non-linear. These algorithms are selected with the aim of optimizing the Segway's performance under different conditions and scenarios.

Additionally, two key tools will be employed in the design and control process: SPCIES and CasADI. SPCIES will be used in developing code to implement linear MPC in the behavior of the Segway, while CasADI will be used in the implementation of non-linear MPC.

Through various simulations conducted in Matlab, the above-mentioned control algorithms will be tested and designed, comparing them to each other. Another aspect to be studied in this project is the identification of states, allowing the derivation of state-space matrices for the system. These matrices will be used to adjust and validate the Segway's parameters, verifying its ability to effectively correct balance errors.

In summary, this work addresses the task of designing a self-balancing Segway, integrating advanced control algorithms and specialized tools. State identification and experimental validation will contribute to evaluating the system's effectiveness in practical situations, thus supporting the robustness and performance of the developed Segway.

Indice

1.- Introducción

Se expondrán el contexto y motivo por el cual se lleva a cabo el proyecto en la sección 1.1, y en la sección 1.2 se expondrán los objetivos que se quieren alcanzar con el mismo.

1.1.- Contexto y justificación del proyecto

Los sistemas de control tienen una gran importancia en nuestro día a día ya que cualquier dispositivo electrónico los utiliza ya sea con mayor o menor complejidad, desde los electrodomésticos que utilizamos diariamente hasta los nuevos modelos de vehículos autónomos. Todos estos sistemas de control son los que guían al sistema electrónico indicándoles cómo deben comportarse, por ejemplo, en el caso de un vehículo autónomo, indicando en que rango de velocidades puede desplazarse, a qué velocidad debe moverse en todo momento, ángulo de giro de las ruedas, etc. En definitiva, el desarrollo de un sistema de control dedicado para un sistema electrónico complejo debe complacer las necesidades solicitadas dentro de unos márgenes de tiempo indicados. Esto es lo que se conoce como un sistema en tiempo real, es decir, un sistema que interacciona con su entorno físico y responde a los estímulos del entorno dentro de un plazo de tiempo determinado.

Existen sistemas de tiempo real crítico (tiempo real «duro»), en los que los plazos de respuesta deben respetarse siempre estrictamente y una sola respuesta tardía a un suceso externo puede tener consecuencias fatales; y sistemas de tiempo real acrítico (tiempo real «suave»), en los que se pueden tolerar retrasos ocasionales en la respuesta a un suceso. [1]

Las características que definen a los sistemas de tiempo real son:

- Determinismo: Es la capacidad de determinar con una alta probabilidad, cuánto es el tiempo que se toma una tarea en iniciarse. Esto es importante porque los sistemas de tiempo real necesitan que ciertas tareas se ejecuten antes de que otras puedan iniciar.
- Responsividad: se enfoca en el tiempo que tarda una tarea en ejecutarse una vez que la interrupción ha sido atendida.
- Controlador: tiene como objetivo hacer que el sistema sea capaz de recopilar los datos necesarios, procesarlos y convertir esa información en una respuesta que haga al sistema capaz de alcanzar la respuesta deseada.
- Confiabilidad: no debe solamente estar libre de fallas, también debe de cumplir que la calidad del servicio que presta no se degradará más allá de un límite determinado de tiempo, esto quiere decir que debe de entregar la respuesta a una solicitud del controlador en una cantidad de tiempo específica.

Muchos son los campos en los que los sistemas en tiempo real tienen lugar, siendo el que interesa para este proyecto el campo de la robótica. Entre la gran cantidad de robots en los que se emplean los sistemas de control, en este caso se hará uso de un Segway (robot con dos ruedas dispuesto en un sistema de péndulo invertido) mostrado en la figura 1.

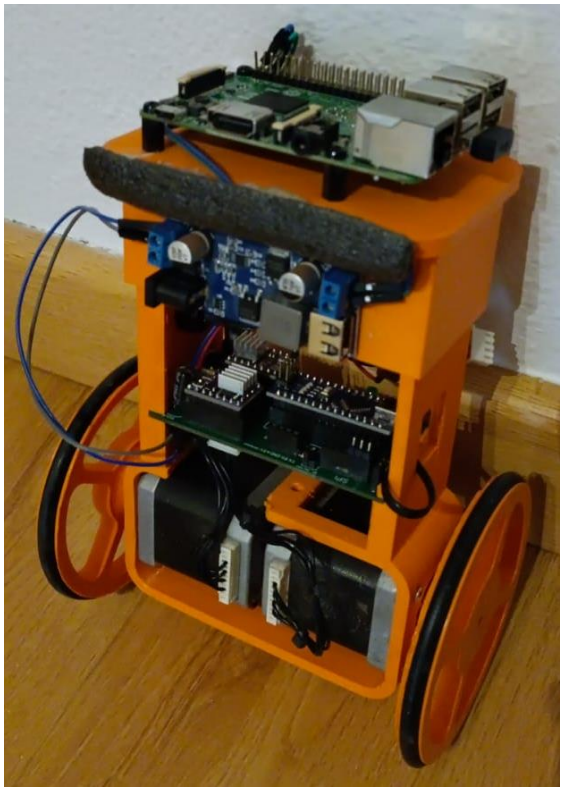

Figura 1. Segway.

El Segway es considerado un sistema de tiempo real crítico, ya que con solo una respuesta tardía el robot podría caerse y no alcanzar la respuesta deseada.

El Segway que se va a emplear para este proyecto ya ha sido empleado por el departamento de Ingeniería de Sistemas y Automática para otros proyectos, por lo que el estudio en profundidad de los elementos que componen este robot ya han sido desarrollados en trabajos anteriores que pueden ser encontrados en la bibliografía, aun así se hará una breve descripción de cada componente ya usado y un desarrollo en profundidad de los componentes nuevos. A su vez se probarán los siguientes sistemas de control sobre el Segway: control PID, MPC lineal y MPC no lineal; todos ellos con sus ventajas e inconvenientes.

1.2.- Objetivos del proyecto

El principal objetivo de este proyecto es analizar los resultados obtenidos en el Segway aplicando los sistemas de control mencionados en el punto anterior en varios escenarios. Los escenarios en los que se va a probar el Segway son:

- Segway inclinado 15º en reposo.
- Segway inclinado 30º en reposo.
- Segway inclinado 45º en reposo.
- Segway inclinado hasta el ángulo máximo en el cual no se caiga el Segway en reposo.

Los resultados que se quieren obtener con cada uno de los algoritmos de control se van a comparar en función de los siguientes:

- Tiempo que tarda el Segway en corregir el error en el ángulo respecto a la referencia.
- Suavidad de la señal de control obtenida.

2.- Fundamentos Teóricos

En la sección 2.1 se expondrán los fundamentos del control en sistemas dinámicos, en la sección 2.2 se expondrán los fundamentos del control LQR, en la sección 2.3 se expondrán los fundamentos del control MPC lineal, en la sección 2.4 se expondrán los fundamentos teóricos del control MPC no lineal y en la sección 2.5 se presentarán las herramientas para la implementación de MPC conocidas como SPCIES y CasADi.

2.1.- Control en sistemas dinámicos

2.1.1.- Sistema dinámico

Un sistema dinámico se refiere a un sistema cuyo estado evoluciona a medida que pasa el tiempo. Estos sistemas pueden encontrarse en situaciones no estacionarias, como máquinas en funcionamiento o procesos físicos en desarrollo. Para comprender y analizar estos sistemas, es esencial identificar sus límites, componentes y relaciones. Esto nos permite construir modelos que intentan representar la estructura y el comportamiento del sistema en cuestión.

Cuando definimos los límites de un sistema dinámico, comenzamos seleccionando los componentes que desempeñan un papel significativo en la generación de los diferentes modos de comportamiento del sistema. Luego, determinamos el entorno o espacio en el que se llevará a cabo nuestro estudio, excluyendo cualquier detalle o aspecto que no sea relevante para nuestros objetivos de análisis.

Es importante señalar que existen dos tipos principales de sistemas dinámicos: aquellos en tiempo continuo y los que operan en tiempo discreto. En este contexto, un sistema dinámico en tiempo discreto, que es el tipo que a menudo se simula en ordenadores, puede describirse mediante una ecuación general, como la siguiente:

$$\begin{cases} x(k+1) = A \cdot x(k) + B \cdot u(k) \\ y(k) = C \cdot x(k) + D \cdot u(k) \end{cases}$$

Donde:

- x: es la matriz de variables de estado del sistema
- y: es la matriz de variables de salida del sistema
- u: es la matriz de variables de control del sistema
- A, B, C y D: son las matrices del espacio de estados del sistema

2.1.2.- Puntos de equilibrio

Supongamos que estamos tratando con un sistema dinámico autónomo, lo que significa que no hay una señal de entrada externa que afecte al sistema. Además, consideremos que este sistema se encuentra en un punto particular llamado «punto de equilibrio». Este punto de equilibrio se caracteriza de la siguiente manera:

- **Punto de Equilibrio Estable**: Se denomina punto de equilibrio estable cuando, si permitimos que el sistema evolucione naturalmente desde una condición inicial cercana al punto de equilibrio, el sistema tiende a volver al punto de equilibrio. En otras palabras, las perturbaciones menores o fluctuaciones en el sistema tienden a ser corregidas con el tiempo, y el sistema finalmente se estabiliza en el punto de equilibrio. Esto indica una especie de «atracción» hacia el punto de equilibrio en el espacio de estado del sistema.

- **Punto de Equilibrio Inestable**: Por otro lado, si, al dejar que el sistema evolucione libremente desde una condición inicial cercana al punto de equilibrio, el sistema se aleja cada vez más del punto de equilibrio, entonces se considera un punto de equilibrio inestable. En este caso, pequeñas perturbaciones tienden a amplificarse y llevar al sistema a estados alejados del punto de equilibrio. Esto sugiere que el punto de equilibrio no es atractivo y que el sistema es muy sensible a las perturbaciones.

En la práctica, la estabilidad de un punto de equilibrio es un concepto crítico en el análisis de sistemas dinámicos. Los puntos de equilibrio estables son deseables en muchos contextos, ya que representan estados de equilibrio a los que un sistema tiende a volver, lo que implica una mayor robustez y previsibilidad en su comportamiento. En contraste, los puntos de equilibrio inestables pueden ser problemáticos, ya que pequeñas perturbaciones pueden llevar al sistema a estados no deseados.

Este análisis de la estabilidad de puntos de equilibrio es fundamental en áreas como la teoría de control, donde se buscan diseñar controladores que estabilicen sistemas dinámicos en puntos de equilibrio específicos. Además, la noción de estabilidad es esencial en diversas disciplinas científicas y de ingeniería para comprender y predecir el comportamiento de sistemas naturales y artificiales. [2]

En la figura 2, se muestran dos sistemas, uno con un punto de equilibrio inestable (izquierda) y otro con punto de equilibrio estable (derecha).

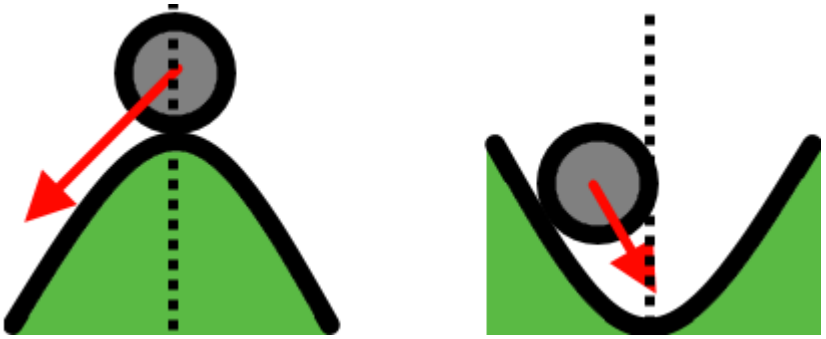

Figura 2. Ejemplo de puntos de equilibrio.

2.1.3.- Control

El control es una técnica que se utiliza para lograr que un sistema alcance un comportamiento deseado mediante el uso del principio de realimentación. En este enfoque, el sistema que busca alcanzar el comportamiento deseado se denomina sistema de control. Un sistema de control generalmente se puede dividir en cuatro componentes clave, cada uno desempeñando un papel esencial en el proceso de control:

- **Medición**: En esta etapa, se realiza la medición de la variable de estado que se desea corregir o controlar en el sistema. La medición proporciona información en tiempo real sobre el estado actual del sistema y se utiliza como base para tomar decisiones en el proceso de control. Esta información es fundamental para comprender cómo el sistema se desvía del comportamiento deseado.
- **Acción**: La acción de control implica la aplicación de un algoritmo o estrategia de control específica para corregir la señal medida anteriormente. Este algoritmo toma la información de la medición y determina cómo deben ajustarse las variables de entrada del sistema para llevarlo hacia el comportamiento deseado. La acción de control es el corazón del proceso de control y es donde se implementa la lógica para tomar decisiones en tiempo real.
- **Referencia**: La referencia representa el valor objetivo de la variable de entrada que se busca alcanzar en el sistema. En otras palabras, es el comportamiento deseado que se desea lograr. El sistema de control se esfuerza por ajustar las variables de entrada de manera que la variable de estado medida se acerque lo más posible a este valor de referencia. La referencia es fundamental

porque define el objetivo del sistema de control.

Estos componentes trabajan en conjunto para garantizar que el sistema de control pueda mantener o llevar al sistema controlado a un estado deseado y mantenerlo allí. La medición proporciona información en tiempo real, la acción de control toma decisiones basadas en esa información y la referencia establece el objetivo que se debe alcanzar.

El control se aplica en una amplia variedad de campos, desde la industria hasta la robótica y la ingeniería eléctrica, y es esencial para lograr un rendimiento óptimo y predecible en sistemas complejos. El diseño de algoritmos de control efectivos y la comprensión de estos componentes son aspectos clave en el desarrollo de sistemas de control exitosos.

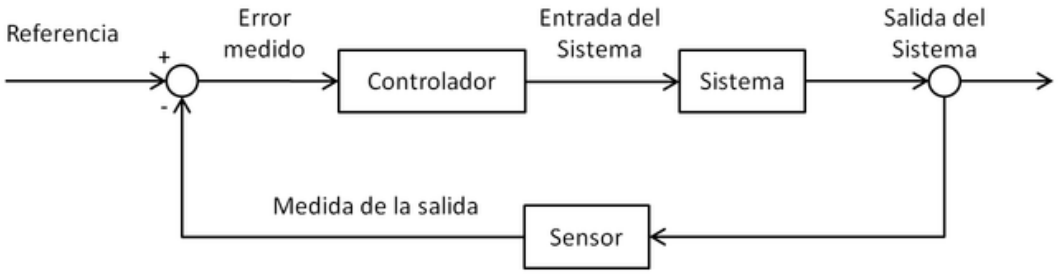

Figura 3. Esquema de un sistema de control.

En el caso que aquí se estudia, el sensor que se utiliza es la placa MPU6050 que proporciona la medida del ángulo que posee el Segway en tiempo real. Los actuadores que se utilizan son los dos motores que se ubican en la parte inferior del Segway. La planta sería el propio Segway sin contar con el sensor y actuadores anteriormente mencionados. Los algoritmos que se van a emplear son: **control LQR**, **MPC lineal** y **MPC no lineal**.

2.2.- Control LQR (Lineal Quadratic Regulator)

Un problema de control óptimo es un problema en el que se busca encontrar la mejor manera de controlar un sistema dinámico para optimizar un cierto criterio de rendimiento a lo largo del tiempo. En este contexto, un sistema dinámico puede ser cualquier entidad que evoluciona con el tiempo, como un proceso industrial.

El objetivo del control óptimo es determinar la secuencia óptima de acciones de control a lo largo del tiempo para minimizar o maximizar una función de costo o rendimiento. Este proceso implica la consideración de variables dinámicas, restricciones operativas y otros factores que pueden afectar el comportamiento del sistema.

Para resolver estos problemas, se utilizan herramientas matemáticas, como el algoritmo conocido como LQR (Lineal Quadrático Regulatorio). Este algoritmo busca la solución (en caso de que exista) de un problema de control óptimo en sistemas lineales en los que no hay restricciones. El algoritmo LQR es, en esencia, una herramienta automatizada para encontrar un controlador de realimentación de estado adecuado.

Existen dos versiones principales que se van a tratar en este proyecto: con horizonte finito y con horizonte infinito. El horizonte en este caso hace referencia a la cantidad de tiempo en el futuro que se va a tener en cuenta para calcular la variable de control a aplicar. [3]

Con **horizonte de predicción finito**

Para un sistema descrito por:

$$\mathrm{x(k+1) = A \cdot x(k) + B \cdot u(k)}$$

Y dada una función de costos cuadrática para el sistema, definida como:

$$\mathrm{J = x_{H_P}^T Q_{H_P} x} + \sum_{k=0}^{H_p-1} \left(x_k^T Q x_k + u_k^T R u_k + 2 x_k^T N u_k\right)$$

Donde *Hp* es el horizonte de predicción.

La ley de control de realimentación que minimiza el valor del costo es:

$$\mathrm{u(k) = -K(k) \cdot x(k)}$$

Donde K viene dado por:

$$\mathrm{K(k) = \left(R + B^T P(k+1) B\right)^{-1} (B^T P(k+1) A + N^T)}$$

Y P se encuentra resolviendo la ecuación diferencial de Riccati en tiempo discreto:

$$P(k-1) = \mathrm{A^T P(k) A} - (A^T P(k) B + N)(R + B^T P(k) B)^{-1}\left(\mathrm{B^T P(k) A + N^T}\right) + \mathrm{Q}$$

Con la condición terminal:

$$\mathrm{P_{H_P} = Q_{H_P}}$$

Con **horizonte de predicción infinito**

Para un sistema descrito por:

$$\mathrm{x(k+1) = A \cdot x(k) + B \cdot u(k)}$$

Y dada una función de costos cuadrática para el sistema, definida como:

$$\mathrm{J} = \sum_{k=0}^{\infty} \left(x_k^T Q x_k + u_k^T R u_k + 2 x_k^T N u_k\right)$$

La ley de control de realimentación que minimiza el valor del costo es:

$$\mathrm{u(k) = -K \cdot x(k)}$$

Donde K viene dado por:

$$\mathrm{K = \left(R + B^T P B\right)^{-1} (B^T P A + N^T)}$$

Y P es la única solución definida positiva de la ecuación algebraica de Riccati en tiempo discreto:

$$P = \mathrm{A^T PA} - (A^T PB + N)(R + B^T PB)^{-1}\left(\mathrm{B^T PA} + \mathrm{N^T}\right) + \mathrm{Q}$$

2.3.- MPC (Model Predictive Control) lineal

El control predictivo por modelo (MPC por sus siglas en inglés) es un método avanzado de control óptimo que es una de las técnicas de control más exitosas en la industria de procesos. Destaca por su capacidad para manejar restricciones en las entradas y estados, su diseño estabilizador, la optimización de un índice de rendimiento y su amplio campo de aplicación. Este enfoque ha demostrado ser eficaz en situaciones donde es crucial abordar restricciones operativas y lograr un rendimiento óptimo, consolidándose como una herramienta fundamental en el ámbito industrial. El MPC utiliza:

- Un modelo dinámico interno del proceso.
- Un historial de movimiento de control pasados.
- Una función de optimización de costos J sobre el horizonte de predicción. [4]

El problema de control óptimo de un MPC es un problema de programación multiparamétrica (es decir, que varios parámetros influyen en el control del sistema) que depende del estado actual x, de una referencia dada (x_r, u_r) y una secuencia de perturbaciones que pueden predecirse d.

Un ejemplo de una función de coste viene dada por:

$$\begin{aligned} \min_{x,\,u} \quad & V_N(x,u) = \sum_{i=0}^{N} \|x(i) - x_r\|_Q^2 + \|u(i) - u_r\|_R^2 + \|x(N) - x_r\|_P^2 \\ \text{s. a.} \quad & x(0) = x \\ & x(j+1) = Ax(j) + Bu(j) + d(j) \\ & A_e x(j) + B_e u(j) + E_e d(j) = 0 \\ & A_x x(j) \le b_x \\ & A_u u(j) \le b_u \\ & A_t(x(N) - x_r) \le b_t \end{aligned}$$

Donde:

A y B son las matrices del espacio de estados que definen el comportamiento del sistema.

A_e y B_e son las matrices que definen las restricciones de igualdad.

A_x y B_x son las matrices que definen las restricciones de menor-igual para el estado actual x.

A_u y B_u son las matrices que definen las restricciones de menor-igual para la entrada u.

Existen multitud de funciones de costes que se pueden utilizar para lograr los resultados deseados, aunque comúnmente se emplea un coste cuadrático como en LQR.

A continuación, se muestra cómo aplicar el algoritmo MPC en un espacio de estados:

Se comienza describiendo la planta:

$$\begin{cases} x_m(k+1) = A_m x_m(k) + B_m u(k) \\ y(k) = C_m x_m(k) \end{cases}$$

La planta debe adaptarse para que pueda cumplir con el propósito del diseño en el que se incluye un integrador. Hay que tener en cuenta que una formulación general de un modelo de espacio de estado tiene un término directo de la señal de entrada u(k) a la salida y(k) como:

$$y(k) = C_m x_m(k) + D_m u(k)$$

Como siempre se asume que la entrada u(k) no afecta la salida y(k) en el mismo tiempo, entonces, se dice que Dm = 0.

Aplicando una operación de diferencia en ambos lados de la ecuación, tenemos que:

$$x_m(k+1) - x_m(k) = A_m\big(x_m(k) - x_m(k-1)\big) + B_m\big(u(k) - u(k-1)\big) \ (1)$$

Podemos expresar esas diferencias en términos de incrementos:

$$\Delta x_m(k+1) = A_m \Delta x_m(k) + B_m \Delta u(k)$$

Nótese que como ahora la entrada en espacio de estados es $\Delta u(k)$ debemos relacionar $\Delta x_m(k)$ con la salida y(k). Esto es fácil creando un nuevo vector de estados:

$$x(k) = \begin{bmatrix} \Delta x_m(k)^T \\ y(k) \end{bmatrix}$$

De esta forma la salida del sistema es descrita por:

$$y(k+1) - y(k) = C_m\big(x_m(k+1) - x_m(k)\big) = C_m \Delta x_m(k+1)$$
$$y(k+1) = C_m \Delta x_m(k+1) + y(k)$$

Reemplazando $\Delta x_m(k)$

$$y(k+1) = C_m A_m \Delta x_m(k) + C_m B_m \Delta u(k) + y(k) \ (2)$$

Colocando juntos la ecuación (1) y (2) tenemos la siguiente representación en espacio de estados:

$$\begin{bmatrix} \Delta x_m(k+1) \\ y(k+1) \end{bmatrix} = \begin{bmatrix} A_m & 0_m^T \\ C_m A_m & 1 \end{bmatrix} \begin{bmatrix} \Delta x_m(k) \\ y(k) \end{bmatrix} + \begin{bmatrix} B_m \\ C_m B_m \end{bmatrix} \Delta u(k)$$
$$y(k) = \begin{bmatrix} 0_m & 1 \end{bmatrix} \begin{bmatrix} \Delta x_m(k) \\ y(k) \end{bmatrix} \ (3)$$

Siendo:

$$x(k+1) = \begin{bmatrix} \Delta x_m(k+1) \\ y(k+1) \end{bmatrix}$$
$$A = \begin{bmatrix} A_m & 0_m^T \\ C_m A_m & 1 \end{bmatrix}$$
$$x(k) = \begin{bmatrix} \Delta x_m(k) \\ y(k) \end{bmatrix}$$
$$B = \begin{bmatrix} B_m \\ C_m B_m \end{bmatrix}$$
$$C = [0_m \quad 1]$$

Donde: $0_m = [0 \quad 0 \quad 0 \quad \dots \quad 0]$ de dimensión n_1 y las matrices A, B, C son conocidas como las matrices aumentadas, las cuales son empleadas para el diseño del control predictivo.

A continuación, se muestra como sucede la predicción de los estados:

$$x(k+1|k) = Ax(k) + B\Delta u(k)$$

$$x(k+2|k) = Ax(k+1|k) + B\Delta u(k+1|k) = A^2 x(k) + AB\Delta u(k) + B\Delta u(k+1|k)$$

...

$$x(k+Np|k) = A^{Np} x(k) + A^{Np-1} B\Delta u(k) + A^{Np-2} B\Delta u(k+1) + \cdots + A^{Np-Nc} B\Delta u(k+N_c-1|k)$$

Donde Np es el horizonte de predicción y Nc es el horizonte de control.

Todas las predicciones son formuladas en terminos de las variables de estado actuales x(k) y del futuro en el movimiento de control Δu(k+j) donde (j = 1, 2, 3, …, Nc – 1). Así se definen los vectores como:

$$Y = [y(k+1) \quad y(k+2) \quad \dots \quad y(k+Np)]^T$$
$$\Delta U = [\Delta u(k) \quad \Delta u(k+1) \quad \dots \quad \Delta u(k+Nc-1)]^T$$

De forma compacta, las predicciones de salida se pueden representar como:

$$Y = Fx(k) + G\Delta u$$

Donde:

$$F = \begin{bmatrix} I \\ A \\ A^2 \\ \dots \\ A^{Np} \end{bmatrix}$$

$$G = \begin{bmatrix} I & 0 & 0 & \dots & 0 \\ B & 0 & 0 & \dots & 0 \\ A \cdot B & B & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ A^{Np-1}B & A^{Np-2}B & A^{Np-3}B & \dots & 0 \end{bmatrix}$$

Función objetivo

Partiendo de la siguiente función objetivo a minimizar:

$$J = (R - y)^T Q_\delta (R - y) + \Delta u^T Q_\lambda \Delta u$$

Donde $R^T = [1, 1, 1, \dots, 1] w(k)$ es el vector que contiene las referencias, w(k) es la referencia actual, Q_δ es la matriz de ponderación del error de seguimiento y Q_λ es la matriz de ponderación del incremento de control.

Para el caso sin restricción solo basta con cumplir con:

$$\frac{\partial \mathrm{J}}{\partial \Delta \mathrm{u}} = 0$$

El incremento de control para el caso sin restricciones se reduce a:

$$\Delta U = (G^T Q_\delta G + Q_\lambda)^{-1} G^T Q_\delta \big(R - Fx(k)\big)$$

Donde el termino $(G^T Q_\delta G + Q_\lambda)^{-1} G^T Q_\delta$ se puede calcular una sola vez y reducirlo a una ganancia:

$$K_1 = (G^T Q_\delta G + Q_\lambda)^{-1} G^T Q_\delta$$

Recordando que solo interesa la primera fila de esta matriz. El incremento de control debe ser calculado en cada interacción es:

$$\Delta U = K_1 \big(R - Fx(k)\big)$$

Caso con restricciones

Para el caso restricto hay que partir de la función de coste:

$$J = (R - y)^T Q_N (R - y) + \Delta u^T R_N \Delta u$$

Y sabiendo que $Y = Fx(k) + G\Delta u$, entonces reemplazando:

$$J = (R - Fx(k) + G\Delta u)^T Q_N (R - Fx(k) + G\Delta u) + \Delta u^T R_N \Delta u$$

Y desarrollando se consigue:

$$J = \Big[\big(R - Fx(k)\big)^T - \Delta u^T G^T\Big] Q_N \big[\big(R - Fx(k)\big) - G\Delta u\big] + \Delta u^T R_N \Delta u$$

$$J = \big(R - Fx(k)\big)^T Q_N \big(R - Fx(k)\big) - 2\big(R - Fx(k)\big)^T Q_N G\Delta u + \Delta u^T G^T Q_N G\Delta u + \Delta u^T R_N \Delta u$$

$$J = \Delta u^T (G^T Q_N G + R_N)\Delta u + 2(Fx(k) - R)^T Q_N G\Delta u + \big(R - Fx(k)\big)^T Q_N \big(R - Fx(k)\big)$$

Donde: $\Delta u^T (G^T Q_N G + R_N)\Delta u + 2(Fx(k) - R)^T Q_N G\Delta u$ representa la parte de la función a optimizar y $\big(R - Fx(k)\big)^T Q_N \big(R - Fx(k)\big)$ representa la parte de la función que no influencia en la optimización.

Multiplicando la expresión anterior por ½:

$$J = \frac{1}{2}\Delta u^T (G^T Q_N G + R_N)\Delta u + (Fx(k) - R)^T Q_N G\Delta u$$

Donde: $H = (G^T Q_N G + R_N)$ y $h^T = (Fx(k) - R)^T Q_N G$

$$\boxed{J = \frac{1}{2}\Delta u^T H \Delta u + F_o^T \Delta u}$$

Para las restricciones existen dos tipos de formas de implementación:

- Restricciones de igualdad: $N \cdot u = n$
- Restricciones de menor-igual: $M \cdot u \leq m$

Donde:

$$N = A_e G + B_e$$

$$n = -A_e F_x x$$

$$M = \begin{bmatrix} M_u \\ M_x G \end{bmatrix}$$

$$m = \begin{bmatrix} m_u \\ m_x - M_x (F_x x) \end{bmatrix}$$

Siendo:

$$A_e = [blkdiag(Ae, Ae, \dots, Ae), 0]$$
$$B_e = blkdiag(Be, Be, \dots, Be)$$
$$E_e = blkdiag(Ee, Ee, \dots, Ee)$$

$$M_u = blkdiag(Au, \dots, Au)$$
$$m_u = (bu, \dots, bu)$$
$$M_x = blkdiag(Ax, Ax, \dots, Ax, At)$$
$$m_x = (bx, \dots, bx, bt - Ax \cdot xr)$$

De esta forma se consigue el problema de optimización a resolver:

$$\boxed{\begin{array}{ll} min & \frac{1}{2}u^T Hu + h^T u \\ s.t. & Nu = n \\ & Mu \leq m \end{array}}$$

2.4.- MPC no lineal

El Control Predictivo por Modelo No Lineal (NMPC por sus siglas en inglés) es una variante del MPC que se caracteriza por el uso de modelos no lineales de sistema en la predicción. Como en MPC lineal, el NMPC requiere de la solución iterativa de problemas de control en cuanto optimización en un horizonte de predicción finito. Mientras estos problemas son convexos en MPC lineal, en NMPC dejan de serlo. Esto supone retos tanto para la teoría estabilidad en NMPC como para la solución numérica. [5]

El NMPC es especialmente útil cuando el modelo lineal no consigue representar bien la dinámica de un sistema y, por tanto, el controlador tiene dificultades en cumplir los objetivos sobre el sistema. Para utilizar el NMPC es necesario hacer uso de la programación no lineal, donde el optimizador deberá

encontrar la trayectoria óptima de nuestro proceso solucionando simultáneamente las restricciones de mi proceso.

El problema de optimización es similar al del MPC lineal:

$$\min \; J(x,u)$$
$$\text{s.t.}$$
$$x_{k+j} = f(x_{k+j-1}, u_{k+j-1})$$
$$y_{k+j} = h(x_{k+k}, u_{k+j})$$
$$y_{min} \leq y_{k+j} \leq y_{max}, \quad j = 1,2,\dots,N$$
$$\Delta u_{min} \leq \Delta u_{k+j} \leq \Delta u_{max}$$

En este caso, se continua minimizando una función cuadrática (min J(x,u)) pero en las restricciones se va tener que solucionar el modelo No Lineal del proceso (x_{k+j}, y_{k+j}) junto con las restricciones de operación del proceso.

Existen diversas estrategias para conseguir solucionar un problema NMPC:

- **SQP (Programación cuadrática secuencial):** con este abordaje se busca resolver todo simultáneamente, desde la integración del modelo no lineal del sistema, las restricciones físicas del sistema y la trayectoria óptima de control a ser aplicada al sistema. También conocida como NLP (Programación No Lineal). El uso de programación no lineal requiere de mucho más procesamiento de cálculo, es por eso por lo que existen abordajes que buscan simplificar el problema haciendo aproximaciones lineales para continuar usando programación cuadrática (PQ).
- **Combinación lineal de soluciones lineales:** se crean diferentes modelos lineales del proceso en diferentes puntos de operación con el objetivo de ir combinando todos esos modelos a lo largo de las trayectorias de las variables.
- **Modelos lineales dinámicos junto con no lineales estáticos:** consiste en combinar elementos del modelo lineal con elementos no lineales, por ejemplo, se puede representar el comportamiento dinámico con un modelo lineal, pero los actuadores representados con un comportamiento no lineal.
- **Modelos locales:** se realizan sucesivas aproximaciones en cada punto de operación a lo largo de la trayectoria, con esto se convierte un problema no lineal, en un problema lineal variante con el tiempo. [6]

2.5.- SPCIES y CasADi

Para la implementación del MPC lineal y el MPC no lineal (NMPC) se hará uso de dos herramientas de código abierto llamadas SPCIES (para MPC lineal) y CasADi (para MPC no lineal).

2.5.1.- SPCIES

SPCIES es una herramienta de Matlab que genera de forma automática soluciones para MPC en diferentes lenguajes de programación. Se admiten varias formulaciones de MPC y se proporcionan soluciones basadas en *Métodos de primer orden*.

Los métodos de primer orden son una categoría de algoritmos de optimización utilizados para predecir y controlar el comportamiento de un sistema dinámico. Estos métodos se caracterizan por utilizar aproximaciones lineales de primer orden para representar la dinámica del sistema y generar las predicciones necesarias para tomar decisiones de control. Sin embargo, los métodos de primer orden pueden no ser adecuados para sistemas con dinámicas no lineales o complejas, ya que las

aproximaciones lineales de primer orden pueden no capturar con precisión el comportamiento del sistema en todas las condiciones. Algunos ejemplos de métodos de primer orden son los algoritmos ADMM, ISTA y FISTA. En la figura 4, se muestra la comparación entre estos métodos comparando el número de iteraciones necesarias con la norma del error.

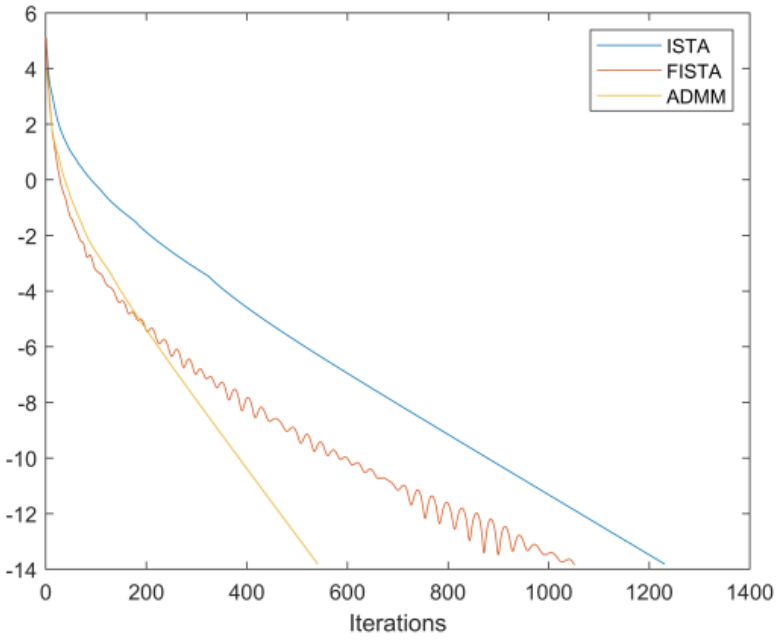

Figura 4. Comparación entre métodos de primer orden.

Actualmente, la herramienta genera código en C y en código Matlab (Mex). En el caso que aquí se estudia, se busca generar el código en C y luego adaptarlo a C++, ya que todavía la herramienta no es capaz de generar código en C++ y conseguir así que funcione correctamente con el funcionamiento que tiene el Segway. [7]

2.5.2.- CasADi

CasADi es una herramienta de código abierto para optimización no lineal y diferenciación algorítmica. Facilita la implementación rápida, pero eficiente, de diferentes métodos para el control numérico óptimo, tanto en un contexto fuera de línea como para el control predictivo de modelos no lineales (NMPC).

Figura 5. Logo de CasADi

Si bien la diferenciación algorítmica todavía constituye una de las funcionalidades principales de la herramienta, desde entonces el alcance de la herramienta se ha ampliado considerablemente, con la adición de soporte para la integración ODE/DAE y el análisis de sensibilidad, programación no lineal e interfaces con otras herramientas numéricas. En su forma actual, es una herramienta de uso general para la optimización numérica basada en gradientes (con un fuerte enfoque en el control óptimo).

Es importante señalar que CasADi no es una herramienta AD convencional, que puede usarse para calcular información derivada del código de usuario existente con poca o ninguna modificación. Si tiene

un modelo existente escrito en C++, Python o MATLAB/Octave, debe estar preparado para volver a implementar el modelo utilizando la sintaxis CasADi.

En segundo lugar, CasADi no es un sistema de álgebra informática. Si bien el núcleo simbólico incluye un conjunto cada vez mayor de herramientas para manipular expresiones simbólicas, estas capacidades son muy limitadas en comparación con una herramienta CAS adecuada.

A pesar de estas limitaciones, CasADi es una herramienta que será útil para este proyecto y que puede proporcionar las soluciones buscadas.

3.- Descripción del Segway

En la sección 3.1 se presentarán de forma general todos los componentes que forman el Segway, en la sección 3.2 se describen los componentes previamente utilizados en el proyecto de Jose Ignacio Cámara. Dentro de esta sección, se puede encontrar el punto 3.2.1 donde se describe el funcionamiento y uso de una Raspberry Pi, en la sección 3.2.2 se describe el funcionamiento y uso de una placa Arduino Nano, en la sección 3.2.3 se describen los motores paso a paso que funcionan como actuadores del sistema, en la sección 3.2.4 se describe la placa MPU6050 utilizada para obtener las entradas del sistema (ángulo, velocidad angular y velocidad angular de las ruedas), en la sección 3.2.5 se describe la batería empleada en el Segway, en la sección 3.2.6 se describe la PCB utilizada para conectar todos los elementos del sistema. Por otro lado, en el apartado 3.3 se exponen los elementos nuevos que se han agregado en este proyecto, donde en la sección 3.3.1 se define el controlador de los motores paso a paso DRV8825, en la sección 3.3.2 se expone el modelo JZK utilizado para conectar las placas Arduino Nano y Raspberry Pi y en la sección 3.3.3 se describe la comunicación serial que se va a emplear en este proyecto.

3.1.- Componentes del Segway

Como se ha mencionado anteriormente, el Segway está formado por una serie de componentes que ya fueron desarrollados en algunos de los trabajos realizados anteriormente con este Segway. Estos trabajos pueden consultarse en la bibliografía de este documento. [8]

El Segway está formado por un conjunto de componentes que juntos hacen que el sistema funcione correctamente. Las conexiones entre los distintos componentes se han realizado a través de cables de cobre, las pistas de la placa PCB (que se mencionará en la siguiente sección) y a través del puerto serie. La disposición de los componentes puede entenderse viendo la figura 6.

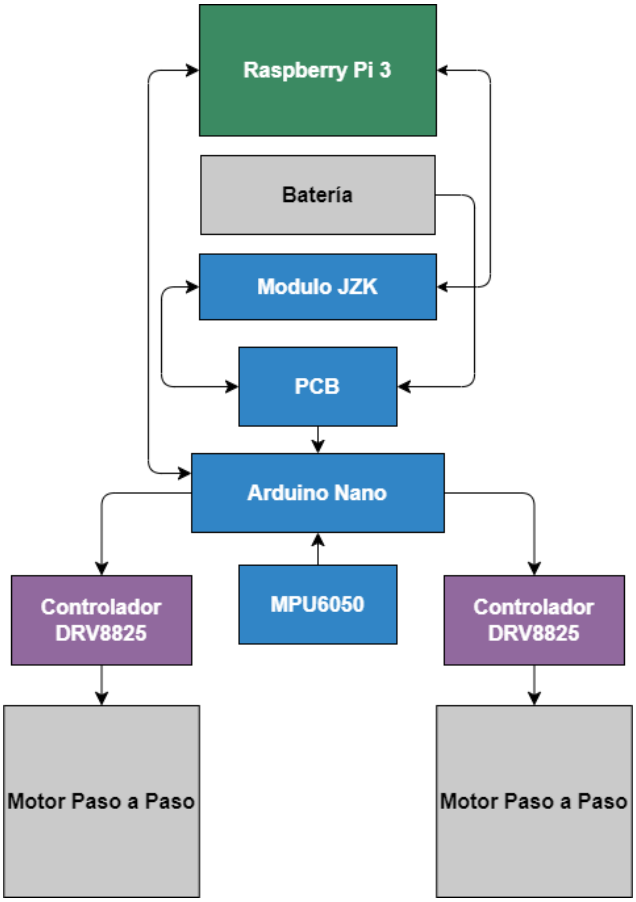

Figura 6. Diagrama del esquema de los componentes del Segway.

3.2.- Componentes previamente implementados

3.2.1.- Raspberry Pi

Completamente carente de partes móviles, la Raspberry Pi es un computador de una sola placa que a la fecha de la preparación de este proyecto, se distribuye comercialmente en cuatro versiones, aunque todos están basados en SoC (*System on a Chip, Sistema en un Chip*) de la misma familia, tienen características ligeramente distintas como se verá más adelante. [8]

En este proyecto se hará uso de una placa Raspberry Pi modelo 3, como la mostrada en la figura 7.

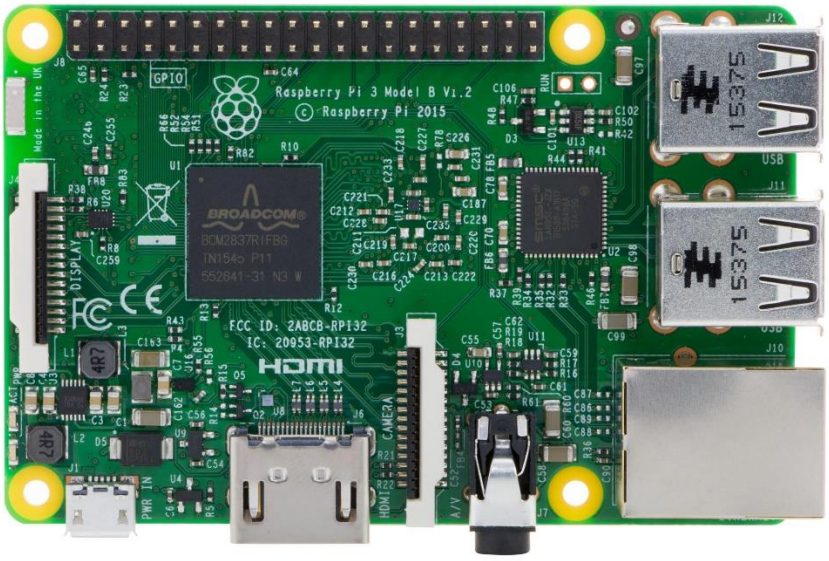

Figura 7. Placa Raspberry Pi 3.

De esta placa se hará uso de los puertos USB ubicados en la parte superior derecha que puede verse en la figura 7, para conectarse mediante puerto serie a la placa Arduino Nano (a la que se le hará mención posteriormente). Además, se han utilizado esos puertos durante el desarrollo del proyecto para conectar periféricos como teclado y ratón.

El sistema operativo está instalado en una placa microSD que va conectada a la placa Raspberry PI. Este sistema operativo se ha instalado con el software que se expondrá en la sección X.

La placa Raspberry Pi 3 estará siendo alimentada a 3,3V mediante los pines 1 y 6 que se muestran en la Figura 8.

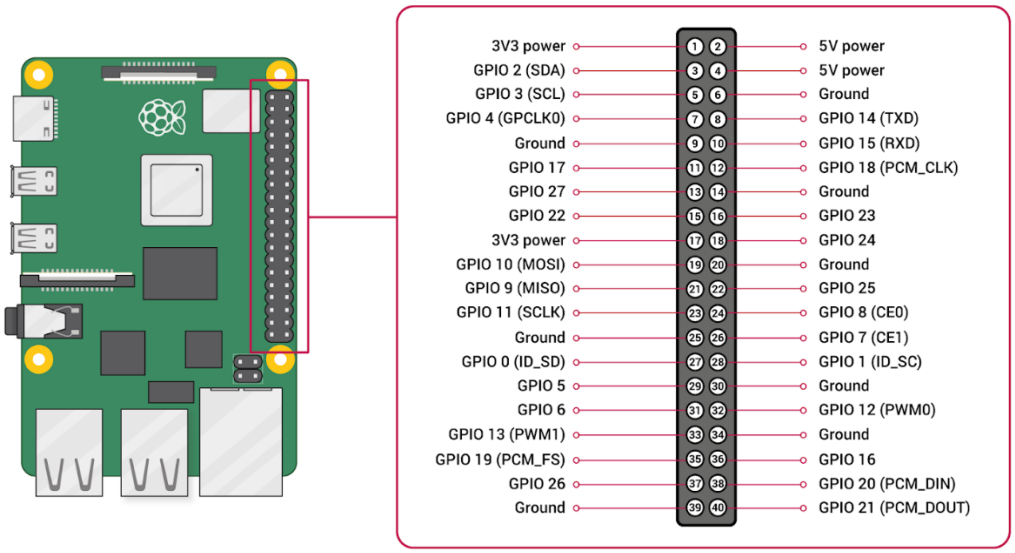

Figura 8. GPIO de la placa Raspberry Pi 3.

Esta placa se empleará para calcular las salidas aplicándoles los diversos algoritmos de control que se expondrán posteriormente y le enviará a la placa Arduino Nano el valor necesario para que el sistema funcione correctamente.

3.2.2.- Arduino Nano

El Arduino es un circuito integrado de software y hardware libre, fácil de utilizar y es capaz de realizar operaciones matemáticas a gran velocidad, utiliza un microcontrolador Atmel. En el corazón de todo Arduino hay una unidad de microcontrolador Atmel (MCU). La mayoría de las placas de Arduino utiliza un microcontrolador AVR ATMega. El lenguaje de programación de Arduino que proporciona acceso a periféricos del microcontrolador, incluidos convertidores de analógico a digital (ADC), buses de comunicación (I2C y SPI), pines de entrada y salida para fines generales, e interfaces de serie. [9]

Hay muchas placas de Arduino y los fabricantes están lanzando placas nuevas con diferentes características, pero en este proyecto se hará uso de la versión conocida como Arduino Nano, cuyo aspecto es el mostrado en la Figura 9.

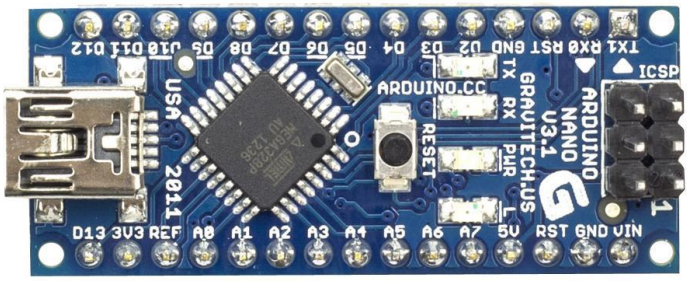

Figura 9. Placa Arduino Nano.

La placa Arduino Nano se encargará de obtener mediante el MPU6050 el ángulo de giro del Segway y enviarle esa información a la placa Raspberry Pi 3. A su vez, obtendrá el dato desde la Raspberry Pi 3 y enviará el valor de velocidad de giro de los motores que corresponda.

3.2.3.- Motores Paso a Paso

Como actuadores del sistema se harán uso de dos motores paso a paso. En concreto, usaremos dos motores Nema 17 como los que se muestran en la figura 10.

Figura 10. Motor paso a paso Nema 17.

El motor Nema 17 es un dispositivo electrónico que se encarga de realizar movimientos precisos,

potentes y controlados a partir de impulsos eléctricos. La velocidad de rotación será proporcional a la frecuencia con la que recibe los impulsos eléctricos. Recibe su nombre debido a las dimensiones de su base, 1.7 x 1.7 pulgadas. [8]

Cuenta con 200 pasos completos por vuelta lo que equivale a 1.8 grados cada paso ($\frac{360º}{200\ pasos} = 1{,}8º/pasos$). Cada paso puede dividirse en hasta 8 micropasos, con lo que tendríamos 1600 micropasos por vuelta, aumentando con ello la precisión y la suavidad del giro. Para poder producir esta división del paso, necesitaremos el uso de un controlador dedicado al motor. A continuación, podemos apreciar los pasos necesarios para barrer una vuelta de 360 grados.

	Micropasos	Pulsos por vuelta	Ángulo de avance
Paso	1	200	1,8º
Medio paso	2	400	0,9º
Cuarto de paso	4	800	0,45º
Octavo de paso	8	1600	0,225º

Tabla 1. División de pasos del motor Nema 17.

3.2.4.- MPU6050

El MPU-6050 es una unidad de medidas inerciales (IMU) de seis grados de libertad, que combina un acelerómetro de tres ejes y un giroscopio de tres ejes. Este es uno de los IMUs más empleados por su gran calidad y precio. Normalmente, se encuentra integrado en módulos como el GY-521 que incorporan la electrónica necesaria para el correcto uso de la IMU y facilita el acceso a sus pines. Aunque la tensión de alimentación del PMU6050 es de bajo voltaje entre 2.4 y 3.6V, este módulo incluye un regulador de voltaje que permite alimentar directamente a 5V. [10] En la figura 11 se muestra una placa MPU6050.

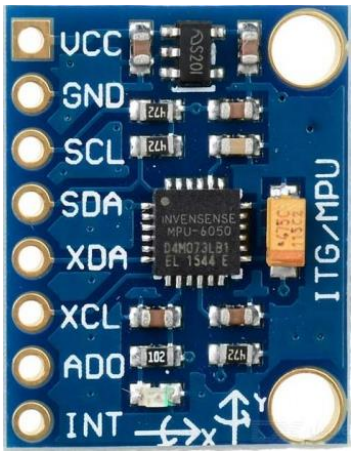

Figura 11. Placa MPU6050.

Esta placa será la que obtenga el ángulo de giro en dos dimensiones del Segway tal y como se muestra en la figura 12.

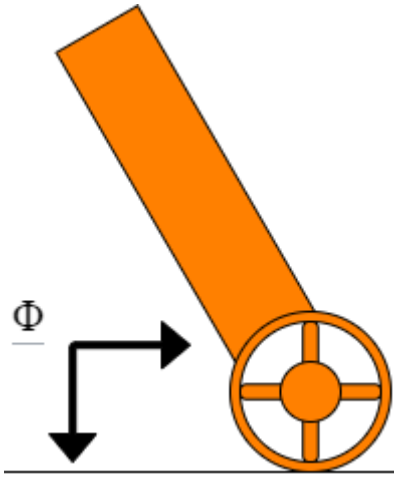

Figura 12. Representación del ángulo de inclinación del Segway.

3.2.5.- Batería

La batería elegida será de polímero de litio, popularmente conocida como batería Li-Po. En el mercado podemos encontrar baterías Li-Po con diferentes especificaciones. Se ha optado por el uso de la batería X-COPTER POWER que consta de 3 celdas. Nos proporciona una capacidad de 1.500 mAh y una tensión nominal de 11.1 V. En la figura 13, se muestra la batería usada en el proyecto.

Figura 13. Batería LIPO X-Copter Power.

Tanto los motores paso a paso, como las placas Arduino y Raspberry Pi serán alimentados a través de esta batería que empleará un regulador de tensión para que les llegué a cada placa su tensión de alimentación correspondiente. Esta batería irá ubicada en la parte superior del Segway, escondida en el interior del chasis.

3.2.6.- PCB

Una PCB es una placa de circuito impreso (Printed Circuit Board) que esta formada por pistas o buses que interconectan distintas áreas de la placa. En este caso las partes que va a interconectar son los pines de cada una de las placas anteriormente mencionadas. La disposición y estructura de la PCB usada puede ser consultada en los trabajos anteriores a este. [8]

3.3.- Componentes nuevos agregados

3.3.1.- Controlador DRV8825

El controlador de motores paso a paso DRV8825 se va a utilizar para poder realizar el control deseado sobre los motores mencionados anteriormente. El dispositivo cuenta con dos puentes H que permite que el motor pueda girar en ambos sentidos, además de poder controlar la cantidad de pasos por pulso que realiza el motor. En las figuras 14 y 15, se muestra el controlador DRV8825 que se ha utilizado en el proyecto (uno por motor), y su esquema de funcionamiento. [11]

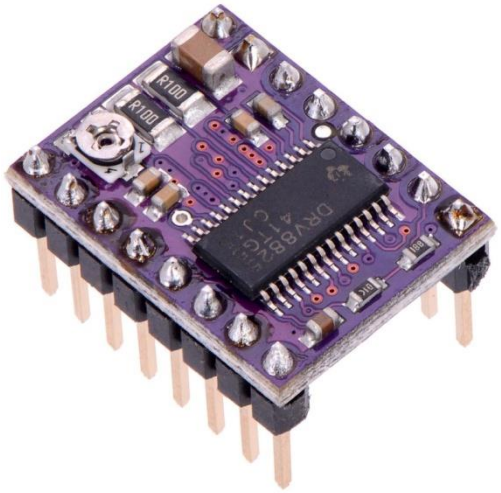

Figura 14. Controlador DRV8825.

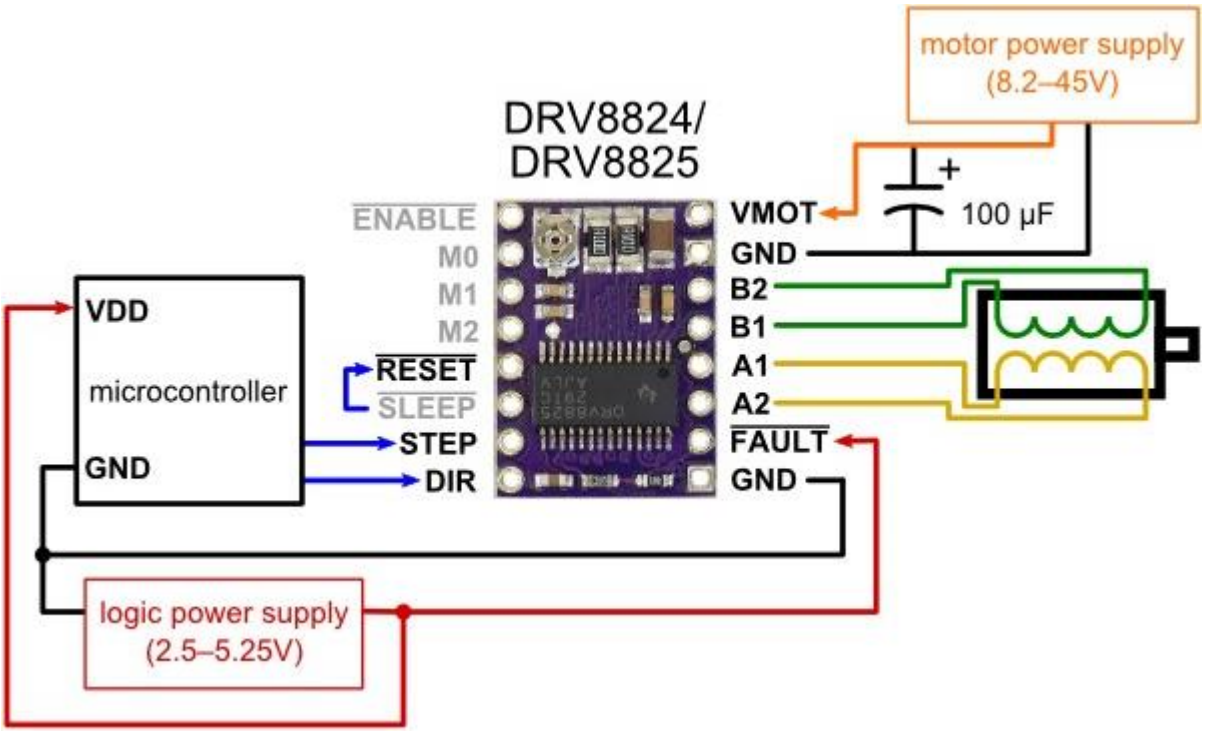

Figura 15. Esquema de funcionamiento del controlador DRV8825.

3.3.2.- Modulo JZK 24V / 12V a 5V

El módulo JZK 24V/12V a 5V es, como su propio nombre indica, un módulo de conversión de voltaje diseñado para convertir una tensión de 24/12 V (la tensión que proporciona la batería) a una tensión de 5V (la utilizada para alimentar las placas Arduino Nano y Raspberry Pi). Además de esta función, ofrece muchas otras utilidades que no van a ser usadas en este proyecto. En la figura 16, se muestra el

módulo JZK 24/12V a 5V utilizado en este proyecto.

Figura 16. JZK 24/12V a 5V.

3.3.3.- Protocolo de comunicación serie

La **comunicación serie** es el proceso de envío de datos de un bit a la vez, de forma secuencial, sobre un canal de comunicación o un bus. En cambio, en la «comunicación en paralelo» todos los bits de cada símbolo se envían al mismo tiempo, y por ello debe haber al menos tantas líneas de comunicación como bits tenga la información a transmitir.

Es la forma de comunicación más sencilla que existe entre receptor y emisor, ya que las dificultades de sincronización que aparecen en la comunicación en paralelo, sumado con el costo del cable, hacen que la comunicación en serie sea considerada mejor opción para comunicaciones de larga distancia. [12]

En la figura 17 se muestra un esquema de cómo funciona la comunicación serie:

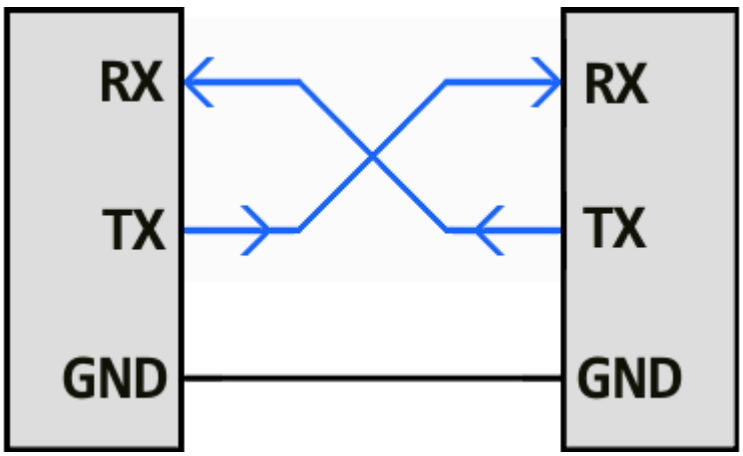

Figura 17. Comunicación serie.

La comunicación serie permite enviar información usando las placas Raspberry Pi y Arduino en paquetes de un byte. En este proyecto, la única información que se transmitirá por comunicación serie será el ángulo giro actual del Segway y el tiempo de espera entre un pulso y otro que decidirá la

velocidad de giro de los motores.

Se quieren hacer dos estudios con la información que se puede enviar por comunicación serie, por un lado se quiere comprobar la complejidad que tiene el código al transmitir dos bytes en lugar de uno, y por otro, comprobar si el comportamiento del Segway varia de forma notable entre la transmisión de un byte y la transmisión de dos bytes.

Con un byte se permite mandar:

$$1 \text{ byte} \rightarrow 2^8 = 256 \text{ valores}$$
$$2 \text{ bytes} \rightarrow 2^{16} = 65536 \text{ valores}$$

El ángulo de giro del Segway puede tomar valores entre -90º y 90º. En las tabla 2, se pueden observar cómo se muestrean los valores para 1 y 2 bytes.

Con 1 byte	-90º	-89,30º	-88,59º	…	88,59º	89,30º	90º
Con 2 bytes	-90º	-89,9973º	-89,9945º	…	89,9945º	89,9973º	90º

Tabla 2. Muestreo de los valores de ángulo de giro con 1 y 2 bytes.

3.4.- Antecedentes en el control del Segway

En este proyecto se parte desde el trabajo realizado por José Ignacio Cámara Molina. [13]

En aquel proyecto, el objetivo consistía en controlar un Segway parecido al de este trabajo mediante LQR y MPC lineal. Para ello se hizo uso de los componentes expuestos en la sección 4 de este trabajo.

Entre los cambios que se han realizado esta la implementación del envío de datos mediante puerto serie en lugar de I2C. Entre las principales diferencias entre ambos protocolos se encuentran:

- La comunicación por puerto serie permite un mayor rango de distancia de comunicación que con I2C.
- La comunicación por puerto serie tiene mayor resistencia al ruido electromagnético.
- La comunicación por puerto serie es ampliamente compatible con muchos dispositivos.
- Una desventaja de la comunicación serial, es que requiere del uso de mayor cableado que en I2C o en este caso de la utilización de un cable USB.
- Además, la comunicación por puerto serie tiene menor velocidad de transmisión en comparación con I2C.

Este último punto no será inconveniente en principio debido a que la velocidad de transmisión que aquí se emplea es igual a la del proyecto anterior: 115200 baudios (115,2 Kbits/s ó 14,4 KB/s)

En el trabajo anterior se enviaba desde la placa Arduino Nano a la placa Raspberry dos datos de tipo «float», el cual son 4 bytes por cada dato. En este trabajo, la placa MPU6050 que proporciona los datos para el cálculo del ángulo de giro y la velocidad angular está conectada directamente a la placa Arduino Nano en lugar de a la placa Raspberry Pi. Por tanto, ahora esos datos deben enviarse también desde Arduino. Los datos proporcionados por el MPU6050 que deben enviarse a la placa Raspberry Pi son:

$$\left.\begin{array}{r} \text{ax} \rightarrow \text{int} \rightarrow 2 \text{ bytes} \\ \text{gx} \rightarrow \text{int} \rightarrow 2 \text{ bytes} \\ \text{gy} \rightarrow \text{int} \rightarrow 2 \text{ bytes} \\ \text{gz} \rightarrow \text{int} \rightarrow 2 \text{ bytes} \\ \text{omega} \rightarrow \text{int} \rightarrow 2\text{bytes} \end{array}\right\} 10 \text{ bytes}$$

El dato que se envía desde la Raspberry Pi a la placa Arduino Nano es la aceleración de las ruedas, el cual se trata de una variable tipo «float» de 4 bytes.

Ahora se procederá a calcular que velocidad de transmisión se necesita para realizar la comunicación:

$$\text{Total de Bytes para comunicar: } 10 \text{ bytes} + 4 \text{ bytes} = 14 \text{ bytes}$$
$$\text{Tiempo para realizar la comunicación: } 20 \text{ ms}$$

Velocidad para comunicar:

$$\frac{14 \text{ bytes}}{20 \text{ ms}} = 700 \ \frac{\text{bytes}}{\text{s}} = 0{,}7 \ \frac{\text{KB}}{\text{s}}$$

Se puede comprobar que la velocidad que proporcionan los 115200 baudios es superior a la velocidad necesaria para transmitir los datos.

Para lograr esta velocidad de comunicación es necesario además instalar un parche al sistema operativo de la placa Raspberry Pi para obtener un sistema operativo en tiempo real.

En el trabajo anterior se utilizó el parche llamado «xenomai». Xenomai es una herramienta de desarrollo en tiempo real que trabaja con el Kernel de Linux para proporcionar la ejecución de códigos y otras funcionalidades en tiempo real. [14]

Para este trabajo se ha apostado por otro parche conocido como «Preempt_RT». Este parche tiene el objetivo de aumentar la previsibilidad y reducir las latencias del Kernel modificando directamente el código del Kernel existente. Es importante señalar que el parche no garantiza tiempos de respuesta absolutos en todas las situaciones, pero este dato en nuestro caso no afecta, ya que el tiempo de respuesta requerido no es tan restrictivo. [15]

3.5.- Implementaciones previas de control LQR y MPC en el Segway

En el proyecto anterior se implementó el control LQR de una forma similar a como se implementará en este trabajo. Primero se calculará la matriz de realimentación K mediante un código de Maltab para posteriormente implementar esta misma matriz en el código del Segway.

Para implementar el control MPC lineal en el trabajo anterior, se crearon todas las matrices necesarias de forma manual en el código del Segway. En este caso, estas matrices creadas manualmente se utilizarán en la simulación de la sección 7, con el propósito de comprobar si la herramienta SPCIES consigue un desarrollo similar.

4.- Identificación de Espacios de Estados

4.1.- Modelado del sistema

A pesar de que, en proyectos anteriores realizados con el Segway, ya se habían calculado diferentes modelos del sistema, en este proyecto se calculará de nuevo mediante el método de regresión por mínimos cuadrados.

El objetivo principal de esta estimación es validar este método para otros sistemas en los que no esté claro cómo se comporta el sistema o, dicho de otro modo, no se conocen las matrices de su espacio de estados que definen el comportamiento del sistema. Por otro lado, el segundo objetivo a lograr es poder prescindir del cálculo de la velocidad angular de las ruedas que se debe realizar. Si se consigue cumplir este objetivo, el sistema sería capaz de funcionar correctamente aplicando únicamente dos variables en la matriz de estados del sistema (ángulo y velocidad angular) que son perfectamente medibles mediante los sensores descritos en la sección anterior.

Suponiendo que el Segway fuera un sistema de «caja negra», este método serviría para poder relacionar los valores de salida con los valores de entrada mediante un espacio de estados.

Este método es utilizado para estimar los coeficientes de las matrices de estado que describen la relación entre las variables independientes, las variables dependientes y las entradas del sistema. [16]

En este caso:

$$X_{k+1} = A \cdot X_k + B \cdot u_k$$
$$Y_k = C \cdot X_k + D \cdot u_k$$

Siendo:

$$X_k = \begin{pmatrix} \phi \\ \dot{\phi} \end{pmatrix}$$

$$u_k = (\ddot{\theta})$$

$$Y_k = (\phi)$$

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

$$B = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

$$C = (c_1 \quad c_2)$$

$$D = (d)$$

Con:

- ϕ: ángulo de giro del Segway.
- $\dot{\phi}$: velocidad angular del Segway.

– $\ddot{\theta}$: aceleración angular de las ruedas del Segway.

En el proyecto anterior, se estimó el espacio de estados obtenido a través de la linealización de las ecuaciones que definen el comportamiento del sistema:

$$a = 2ML^2 + MRL$$
$$b = (3m_r + M)R^2 + MRL$$
$$c = MgL$$

Siendo:

- M: masa del Segway sin las ruedas.
- L: longitud desde el eje hasta el centro de gravedad.
- R: radio de las ruedas.
- m_r: masa de una rueda del Segway.
- g: gravedad.

El espacio de estados se definió como:

$$\begin{bmatrix} \dot{\Phi} \\ \ddot{\Phi} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ c/a & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \Phi \\ \dot{\Phi} \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ -b/a \\ 1 \end{bmatrix} \ddot{\theta}$$

4.2.- Conceptos básicos de la identificación de espacios de estados

Se denomina identificación de sistemas a la obtención de información relevante a partir de un conjunto de datos medidos. Es decir, dado el sistema de estados:

$$X_{k+1} = A \cdot X_k + B \cdot u_k$$

Obtener las matrices A y B que aproximen de la mejor forma los valores medidos de X_k y u_k.

El proceso de identificación de sistemas consta de las siguientes fases:

- Considerar el contexto de la aplicación.
- Propósito y formulación del problema.
- Planificación experimental.
- Conjunto de modelos.
- Identificación y métodos de identificación
- Validación del modelo.

Para este caso, solo interesan conocer los dos últimos pasos.

Existen varios métodos de identificación de sistemas. En este proyecto se ha optado por utilizar la regresión lineal por mínimos cuadrados que será expuesta en la sección 5.3.

La validación del modelo consiste en comprobar, una vez obtenidas las matrices A y B, que para una variable de estado de partida y una variable de entrada conocida, se puede prever con una exactitud aceptable el comportamiento del sistema.

4.3.- Metodología de regresión lineal aplicada al Segway

Lo que se quiere conseguir, es obtener las matrices A y B a través del método de regresión por mínimos cuadrados, conociendo los valores de entrada u y variables de estado x. Empleando el método de regresión por mínimos cuadrados se requiere la siguiente forma:

$$\gamma = \nu \cdot \beta + n$$

Expandiendo el modelo de espacio de estados se obtiene:

$$\begin{pmatrix} x_{1,\ k+1} \\ \cdots \\ x_{n,\ k+1} \end{pmatrix} = \begin{pmatrix} \varphi_{11} & \cdots & \varphi_{1n} \\ \cdots & & \cdots \\ \varphi_{n1} & \cdots & \varphi_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_{1,\ k} \\ \cdots \\ x_{n,\ k} \end{pmatrix} + \begin{pmatrix} \lambda_{11} & \cdots & \lambda_{1m} \\ \cdots & & \cdots \\ \lambda_{n1} & \cdots & \lambda_{nm} \end{pmatrix} \cdot \begin{pmatrix} u_{1,\ k} \\ \cdots \\ u_{m,\ k} \end{pmatrix}$$

Pudiendo también representarse de la siguiente forma:

$$x_{i,\ k+1} = [x_{1,\ k} \quad \cdots \quad x_{n,\ k} \quad u_{1,\ k} \quad \cdots \quad u_{m},\ k] \cdot \begin{bmatrix} \varphi_{i,\ 1} \\ \cdots \\ \varphi_{i,\ n} \\ \lambda_{i,\ 1} \\ \cdots \\ \lambda_{i,\ m} \end{bmatrix}$$

Para todos los valores que se muestrean de cada variable i-esima:

$$\begin{pmatrix} x_{i,\ k+1} \\ \cdots \\ x_{i,\ k+r} \end{pmatrix} = \begin{pmatrix} x_{1,\ k} & \cdots & x_{n,\ k} & u_{1,\ k} & \cdots & u_{m,\ k} \\ \cdots & & \cdots & \cdots & & \cdots \\ x_{1,\ k+r-1} & \cdots & x_{n,\ k+r-1} & u_{1,\ k+r-1} & \cdots & u_{m,\ k+r-1} \end{pmatrix} \cdot \begin{pmatrix} \varphi_{i,\ 1} \\ \cdots \\ \varphi_{i,\ n} \\ \lambda_{i,\ 1} \\ \cdots \\ \lambda_{i,\ m} \end{pmatrix}$$

Siendo la ecuación anterior equivalente a la siguiente:

$$\gamma_i = \nu \cdot \beta_i$$

Se puede obtener el valor de β_i a través de la siguiente ecuación:

$$\nu \cdot \beta_i = \gamma_i$$

$$\nu^{-1} \cdot \nu \cdot \beta_i = \nu^{-1} \cdot \gamma_i$$

$$\beta_i = \nu^{-1} \cdot \gamma_i$$

$$\beta_i = \left(\nu^T \cdot \nu\right)^{-1} \cdot \nu^T \cdot \gamma_i$$

Siendo las matrices A y B:

$$A = \begin{pmatrix} \varphi_{1,\ 1} & \varphi_{2,\ 1} & \varphi_{3,\ 1} \\ \varphi_{1,\ 2} & \varphi_{2,\ 2} & \varphi_{3,\ 2} \\ \varphi_{1,\ 3} & \varphi_{2,\ 3} & \varphi_{3,\ 3} \end{pmatrix}$$

$$B = \begin{pmatrix} \lambda_{1,\,1} \\ \lambda_{1,\,2} \\ \lambda_{1,\,3} \end{pmatrix}$$

En el caso de este proyecto se han tomado varias muestras del ángulo de giro tanto para el caso en el que los motores están parados como para casos en los que los motores se mueven en aceleración constante. Se ha estimado que las matrices obtenidas sean el valor medio de todos los valores obtenidos con las diferentes muestras que se han realizado. El código empleado para obtener las matrices A y B, se puede consultar en el Anexo de este proyecto en la sección 11.1.

Los valores obtenidos de las matrices A y B son:

$$A = \begin{pmatrix} 1{,}0029 & 0{,}0432 \\ 1.3199 & 1.0332 \end{pmatrix}$$

$$B = \begin{pmatrix} -0.0003 \\ -0.0109 \end{pmatrix}$$

4.4.- Resultados de la identificación de espacios de estados

A continuación, en las figuras 18 y 19, se muestran los datos obtenidos de las muestras realizadas con el segway en comparación con los valores obtenidos a través de las matrices obtenidas anteriormente.

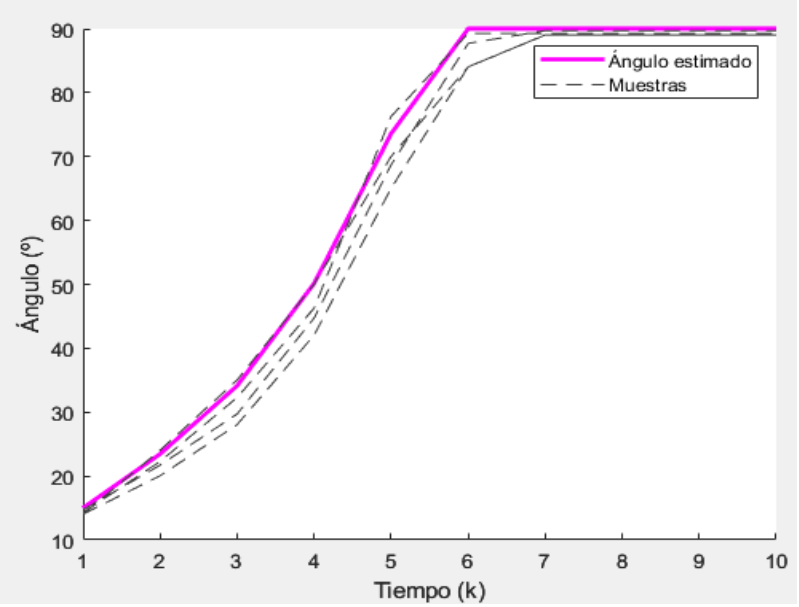

Figura 18. Estimación del ángulo sin entrada u.

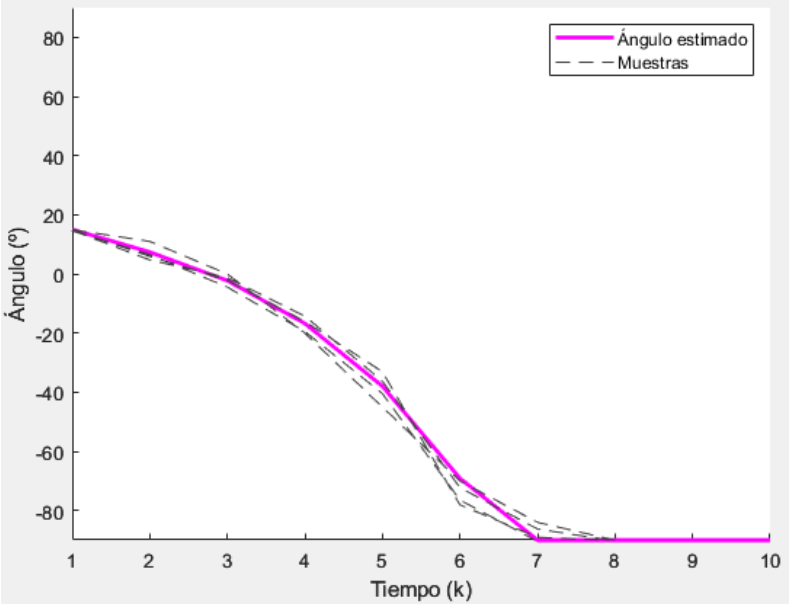

Figura 19. Estimación del ángulo aplicando una entrada constante u = -5.

En las figuras 20 y 21 se puede observar como las matrices obtenidas se aproximan al comportamiento lineal del Segway para las variables de estado de velocidad angular del Segway y velocidad angular de las ruedas.

Ya una vez habiendo obtenido las matrices del espacio de estados del sistema se puede aplicar los diferentes algoritmos de control, primero en simulación y luego en el propio segway.

5.- Implementación del control LQR

5.1.- Diseño del controlador LQR

Para diseñar el controlador LQR primero se deben definir los parámetros del sistema:

Longitud hasta el CDG (en m.)	0,05
Masa de la rueda (en kg.)	0,068
Masa del Segway sin ruedas (en kg.)	1,16
Radio de las ruedas (en m.)	0,0495
Gravedad (en m/s^2)	9,81

Con estos parámetros se podrán calcular las matrices del espacio de estados que definen el comportamiento del sistema. Para calcular las matrices del modelo lineal del sistema se debe proceder como se comentó en la sección 2.2.

$$A = \begin{bmatrix} 0 & 1 & 0 \\ c/a & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ -b/a \\ 1 \end{bmatrix}$$

$$C = [1 \quad 0 \quad 0]$$

$$D = [0]$$

Estas matrices están definidas en la región del tiempo. Para discretizarlas se puede emplear el siguiente código en Matlab:

```
%% Parámetros del Segway
L = 0.05; % Longitud del Segway (en m)
mr = 0.068; % Masa de la rueda (en kg)
Masa = 1.228 - mr; % Masa total del Segway (en kg)
Radio = 0.0495; % Radio de la rueda (en m)
g = 9.81; % Gravedad (en m/s^2)

a = 2*Masa*L^2 + Masa*Radio*L;
b = (3*mr + Masa)*Radio^2 + Masa*Radio*L;
c = Masa*g*L;

%% Matrices del sistema
A = [ 0   , 1 , 0;
     c/a , 0 , 0;
      0   , 0 , 0];

B = [  0 ;
     -b/a;
       1 ];

C =[1 0 0];

D = 0;

Mc = ss(A,B,C,D);
Md = c2d(Mc,0.02);

[Ad,Bd,Cd,Dd] = ssdata(Md);
```

Para proseguir con el diseño del controlador, se deben seleccionar los valores de ponderación de errores de las variables de estado y de las variables de entrada (Q y R, respectivamente):

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

$$R = [0{,}1]$$

Ya habiendo obtenido las matrices del espacio de estados y las matrices de ponderación de errores, se puede calcular el valor de realimentación K y además la matriz de Riccati que será útil para el cálculo del controlador MPC.

```
%% Obtención del vector de realimentación K
Q = [10 0 0 ; 0 1 0 ; 0 0 10]; % Peso de las variables de estado
R = 0.1; % Peso de la variable de entrada

[K,S,P] = dlqr(Ad,Bd,Q,R),
```

La función *dlqr* es la encargada de calcular la matriz de ganancias óptimas *K* de manera que se cumpla la ley de realimentación que se comentó en la sección 2.2.

$$u[n] = -K \cdot x[n]$$

El valor de K es el valor que minimiza la función cuadrática de coste:

$$J(u) = \sum_{n+1}^{\infty} \left(x[n]^T \cdot Q \cdot x[n] + u[n]^T \cdot R \cdot u[n] + 2 \cdot x[n]^T \cdot N \cdot u[n] \right)$$

Siendo el modelo de espacio de estados en tiempo discreto el que se calculo anteriormente que cumple con:

$$x[n+1] = A \cdot x[n] + B \cdot u[n]$$

Además de la ganancia de realimentación de estados K, la función también proporciona, como se ha comentado, la solución de horizonte infinito S de la ecuación de Riccati de tiempo discreto asociada.

$$A^T \cdot S \cdot A - S - (A^T \cdot S \cdot B + N) \cdot (B^T \cdot S \cdot B + R)^{-1} \cdot (B^T \cdot S \cdot A + N^T) + Q = 0$$

5.2.- Simulaciones

Tanto para este controlador como para los posteriores, el procedimiento será simular mediante Matlab el comportamiento del sistema. Se realizarán tres simulaciones por cada controlador, la primera será una simulación utilizando las matrices linealizadas del espacio de estados, por lo que se espera el mejor resultado posible, la segunda simulación se hará empleando la K obtenida a partir de las matrices calculadas mediante la identificación de estados, pero con la misma entrada calculada en la primera simulación. La tercera simulación se realizará empleando la misma entrada *u* obtenida en la primera simulación (K obtenida a partir de las matrices linealizadas) calculando el comportamiento del sistema mediante la ecuación no lineal.

1ª simulación (K obtenida con las matrices linealizadas)

En la figura 20, se muestra el resultado de la 1ª simulación para un ángulo inicial de -15º.

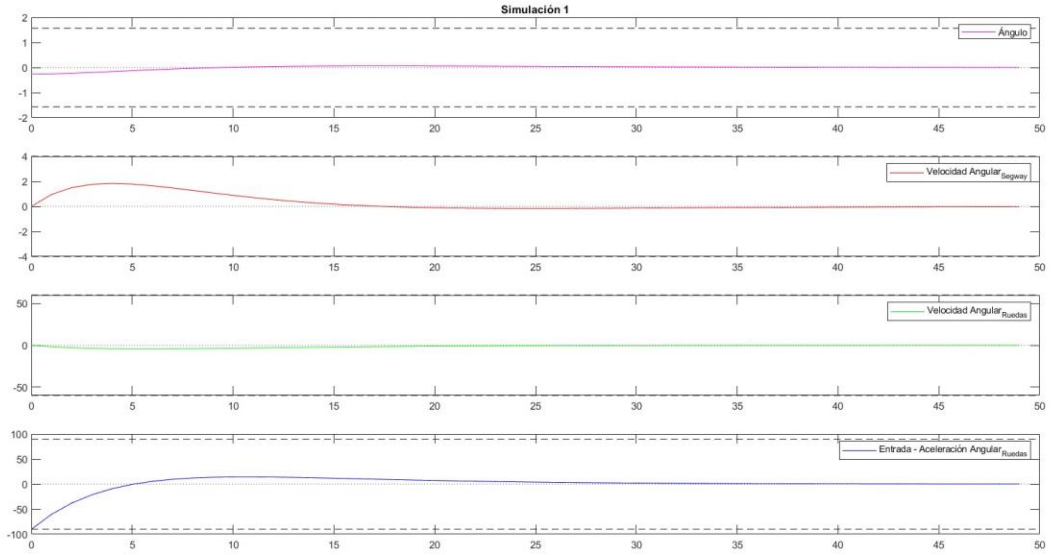

Figura 20. 1ª simulación de LQR con ángulo inicial de -15º.

En la figura 21, se muestra el resultado de la 1ª simulación para un ángulo inicial de 30º.

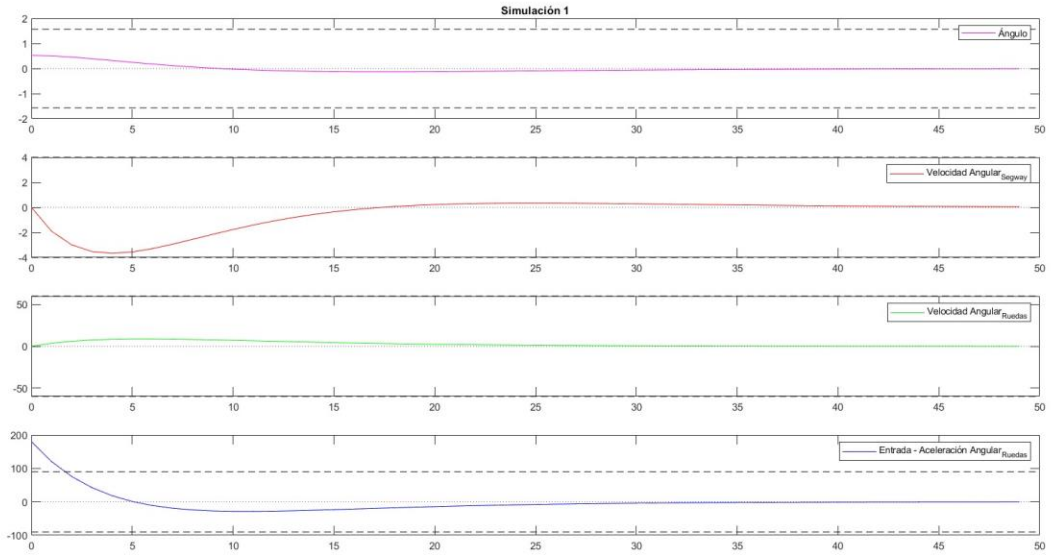

Figura 21. 1ª simulación de LQR con ángulo inicial de 30º.

En la figura 22, se muestra el resultado de la 1ª simulación para un ángulo inicial de -45º.

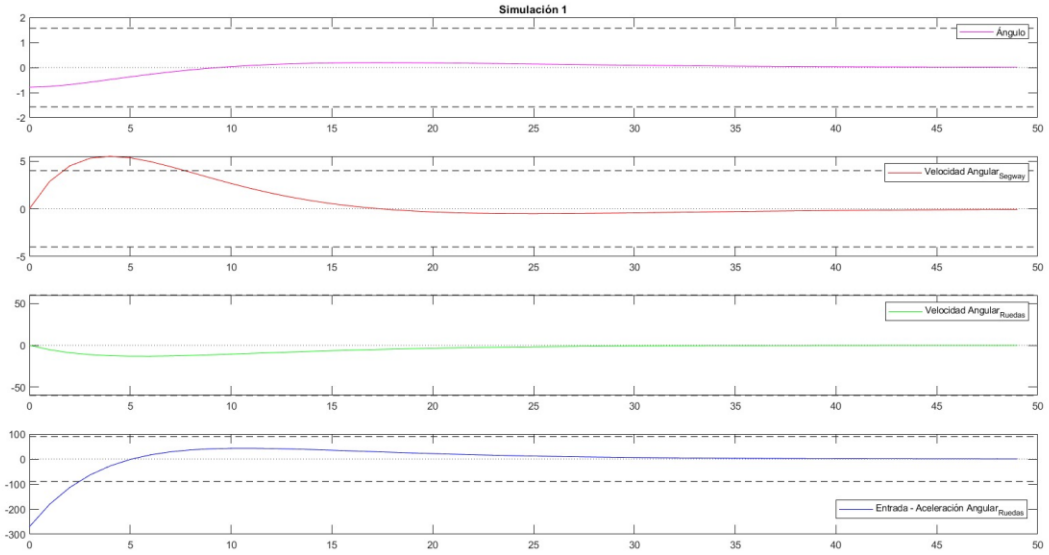

Figura 22. 1ª simulación de LQR con ángulo inicial de -45º.

Como se puede observar, en las simulaciones para +30º y -45º, el valor que toma la entrada 'u' sobrepasa los límites definidos para el sistema (en este caso [-90 , 90])

Teniendo en cuenta que el LQR no valora las restricciones del sistema, si se realizan ambas simulaciones teniendo en cuenta los límites de forma manual, se obtiene:

En la figura 23, se muestra la simulación realizada teniendo en cuenta las restricciones con un ángulo inicial de 30º.

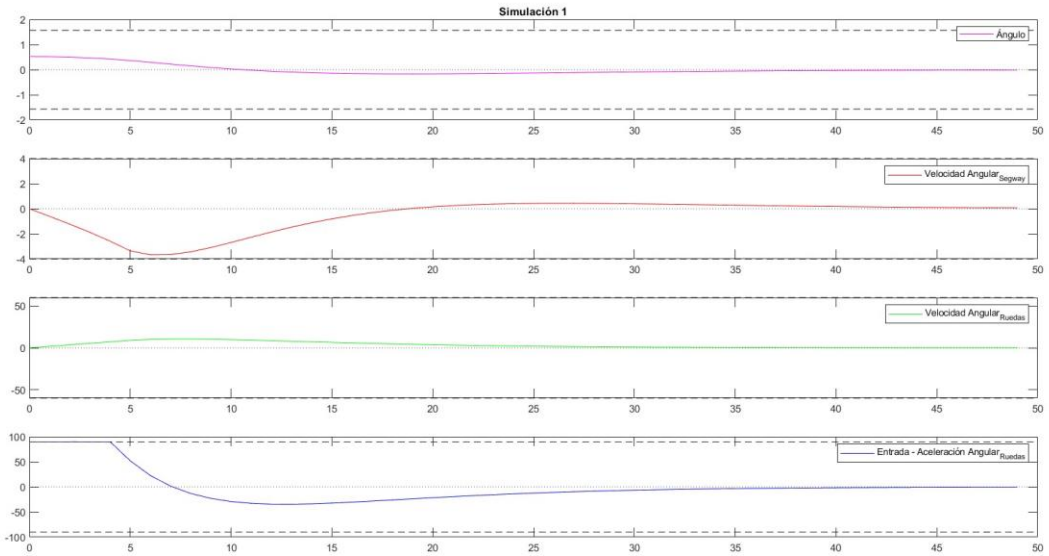

Figura 23. Simulación de LQR con restricciones para un ángulo inicial de 30º.

En la figura 24, se muestra la simulación realizada teniendo en cuenta las restricciones con un ángulo inicial de -45º.

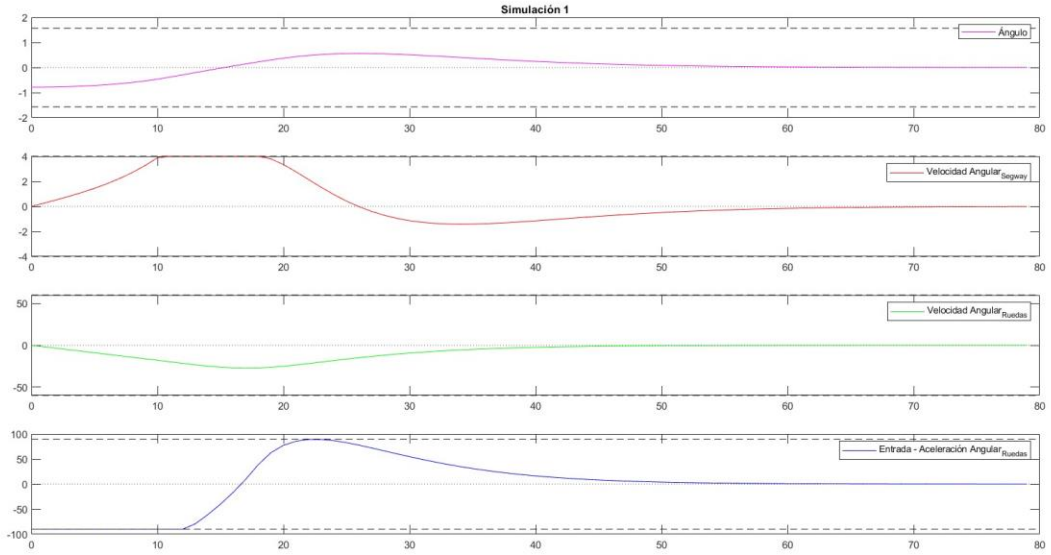

Figura 24. Simulación de LQR con restricciones para un ángulo inicial de -45º.

Se puede observar que a pesar de las restricciones el sistema es capaz de seguir la referencia.

2ª simulación (con las matrices obtenidas de la identificación de sistemas)

En la figura 25, se muestra el resultado de la 2ª simulación para un ángulo inicial de -15º.

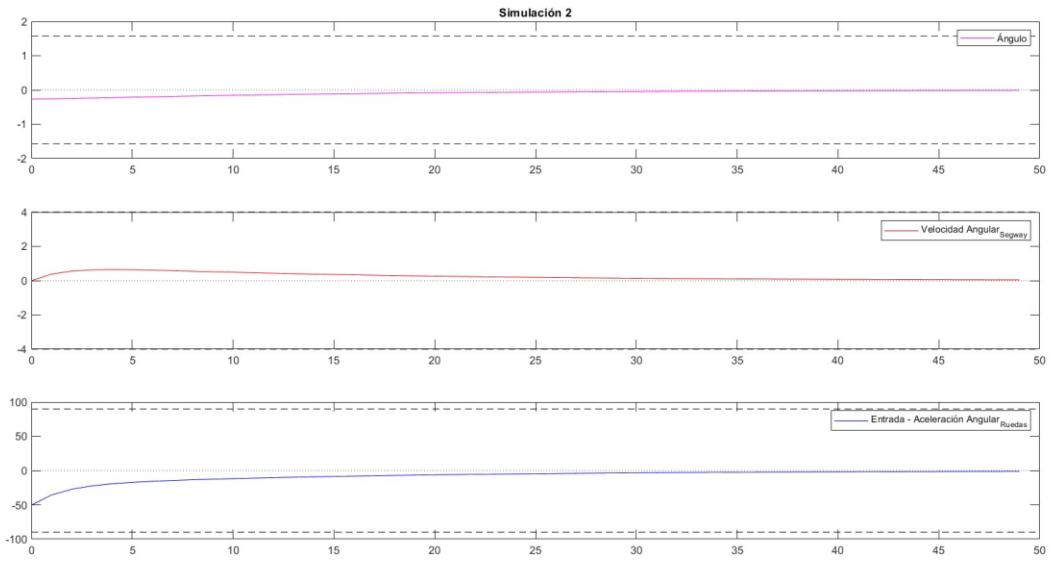

Figura 25. 2ª simulación usando K de identificación de estados con ángulo inicial de -15º.

En la figura 26, se muestra el resultado de la 2ª simulación para un ángulo inicial de 30º.

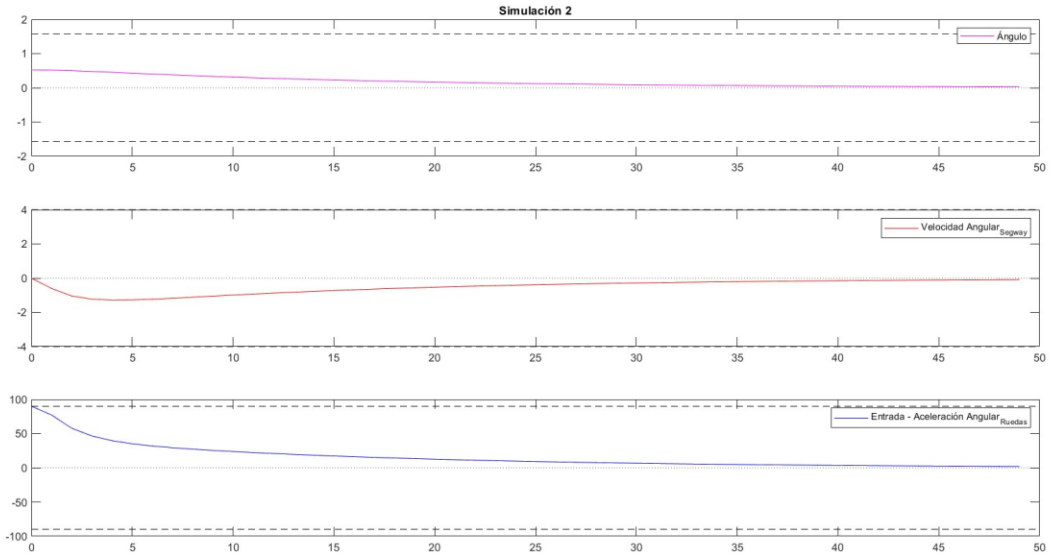

Figura 26. 2ª simulación usando K de identificación de estados con ángulo inicial de 30º.

En la figura 27, se muestra el resultado de la 2ª simulación para un ángulo inicial de -45º.

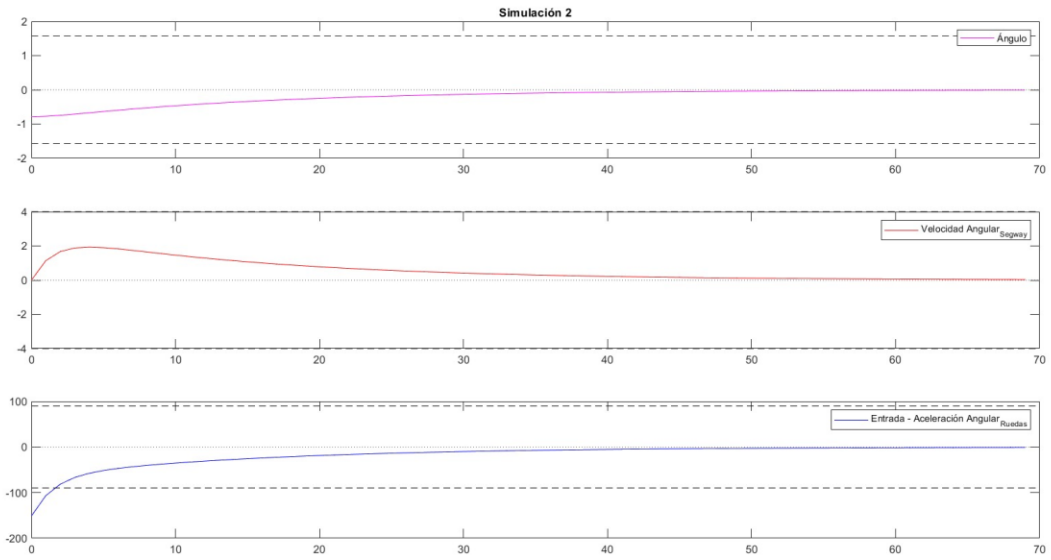

Figura 27. 2ª simulación usando K de identificación de estados con ángulo inicial de -45º.

De la misma forma que para la 1ª simulación, cuando el ángulo inicial es de -45º, el valor de la variable de entrada 'u' sobrepasa los límites definidos en las restricciones. Una vez más, se muestran las simulaciones obtenidas aplicando las restricciones.

En la figura 28, se muestra la simulación realizada teniendo en cuenta las restricciones con un ángulo inicial de 30º.

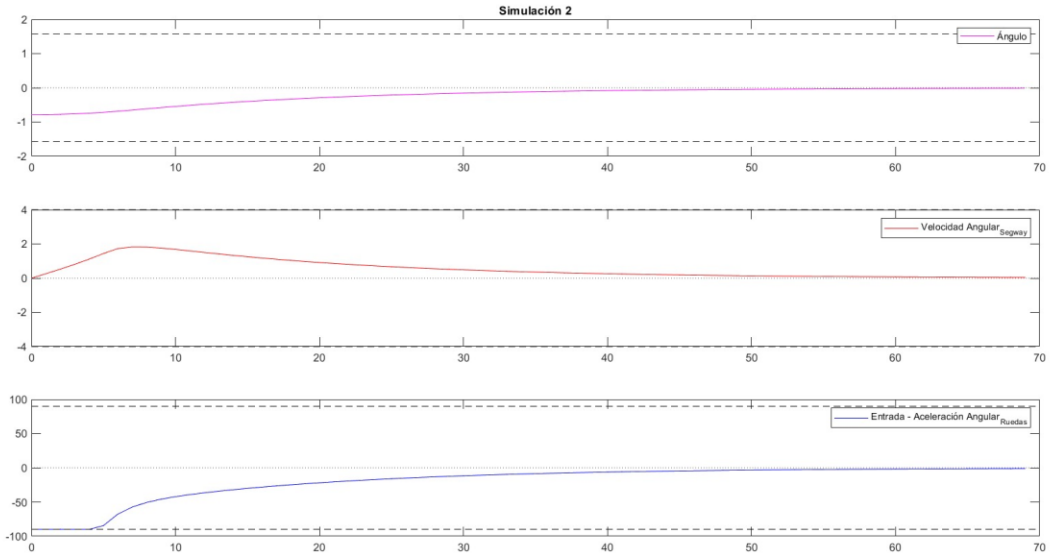

Figura 28. Simulación con restricciones usando K de identificación de estados con ángulo inicial de -45º.

En la figura 29, se muestra una comparación entre los resultados obtenidos en la simulación 1 y 2 para un ángulo inicial de -15º.

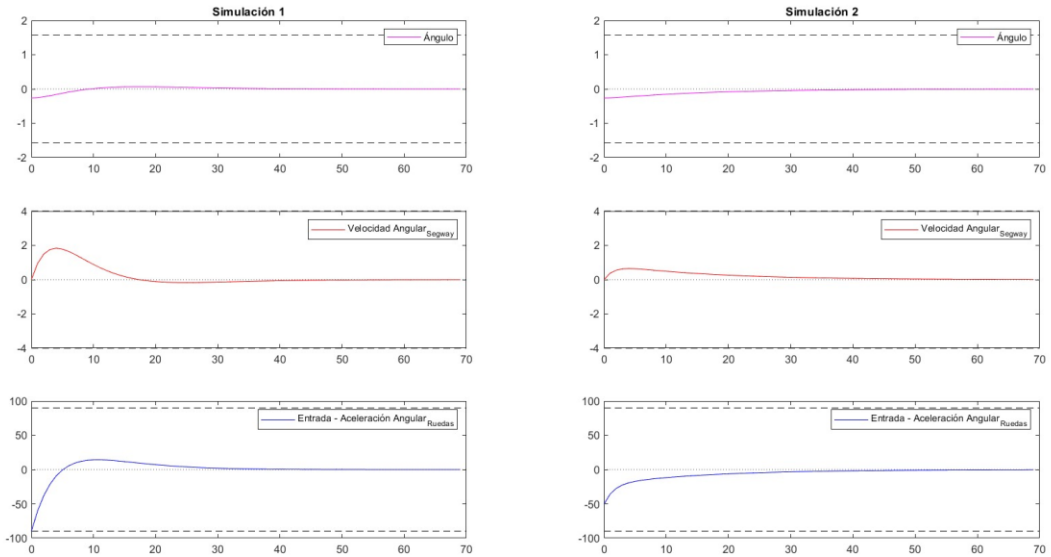

Figura 29. Comparación entre simulaciones 1 y 2 con ángulo inicial de -15º.

En la figura 30, se muestra una comparación entre los resultados obtenidos en la simulación 1 y 2 para un ángulo inicial de 30º.

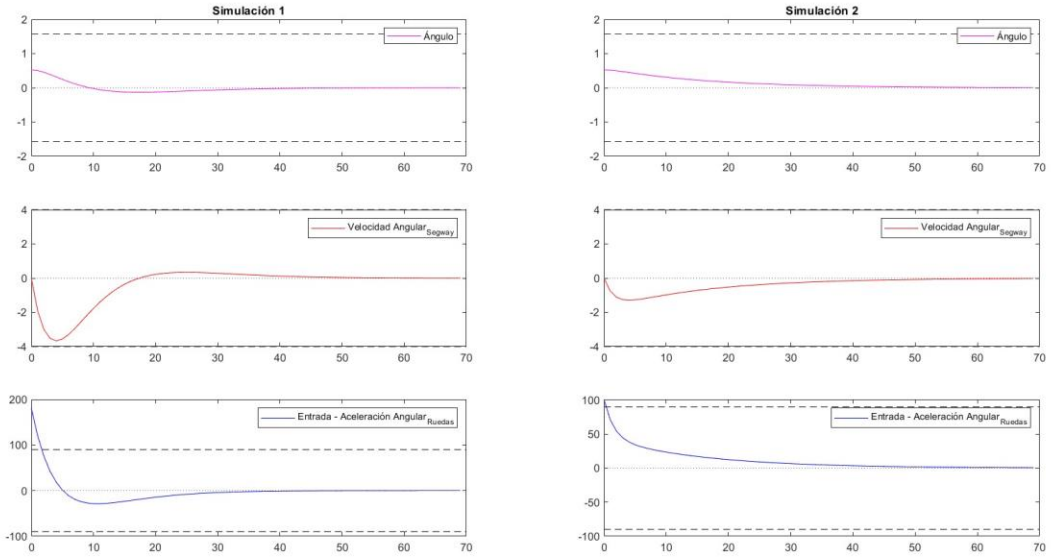

Figura 30. Comparación entre simulaciones 1 y 2 con ángulo inicial de -30º.

En la figura 31, se muestra una comparación entre los resultados obtenidos en la simulación 1 y 2 para un ángulo inicial de -45º.

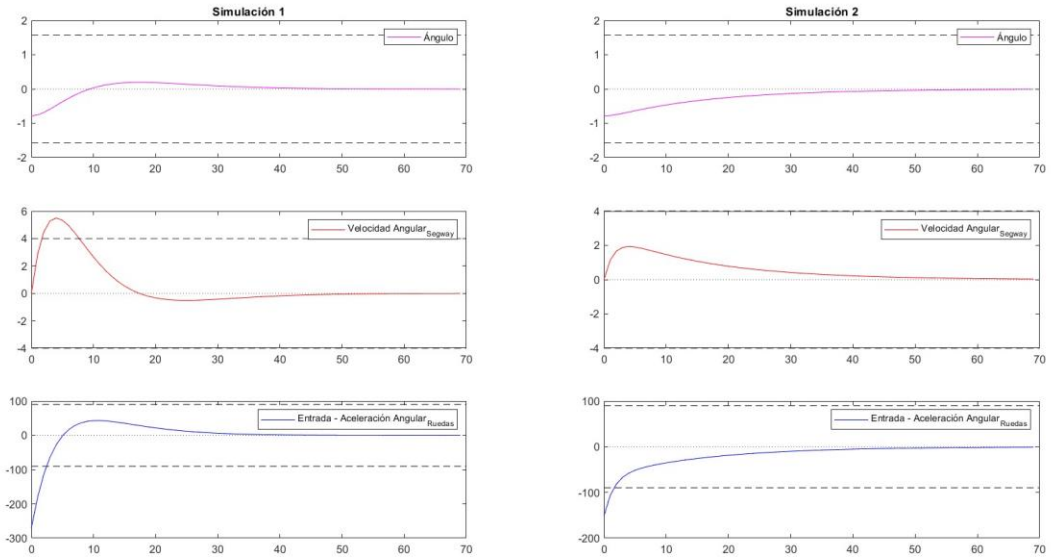

Figura 31. Comparación entre simulaciones 1 y 2 con ángulo inicial de -45º.

Se puede observar que aplicando la K obtenida con la identificación de estados, el sistema alcanza la referencia también, sin embargo, emplea más tiempo para ello.

3ª simulación (con la ecuación no lineal del sistema)

Ahora se expondrán las simulaciones realizadas aplicando la misma entrada «u» obtenida en las simulaciones anteriores, pero siendo aplicada en la ecuación no lineal del sistema.

En la figura 30, se muestra el resultado de la 3ª simulación usando la K tanto de la 1ª simulación como de la 2ª simulación para un ángulo inicial de -15º.

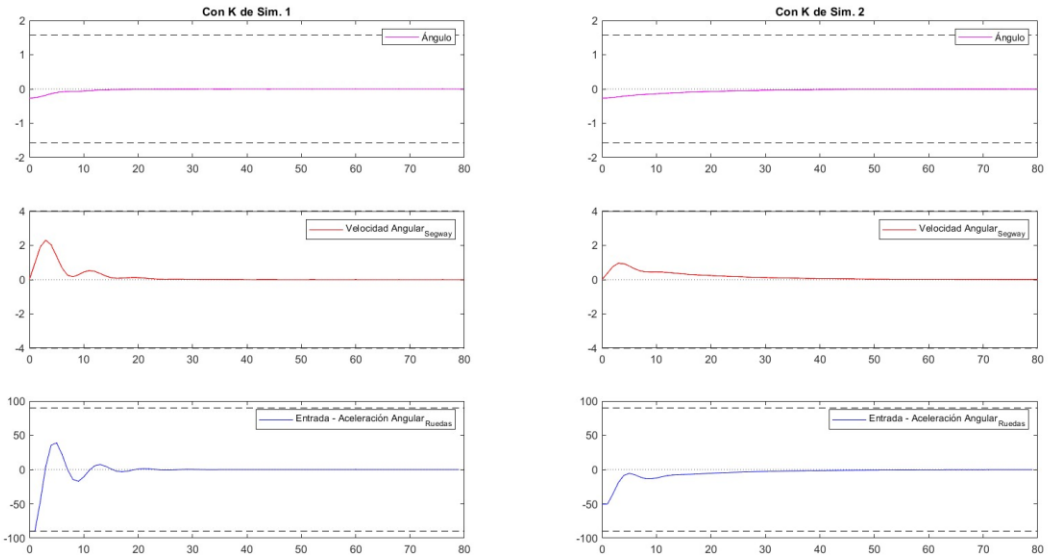

Figura 32. 3ª simulación con modelo no lineal con un ángulo inicial de -15º.

En la figura 31, se muestra el resultado de la 3ª simulación usando la K tanto de la 1ª simulación como de la 2ª simulación para un ángulo inicial de 30º.

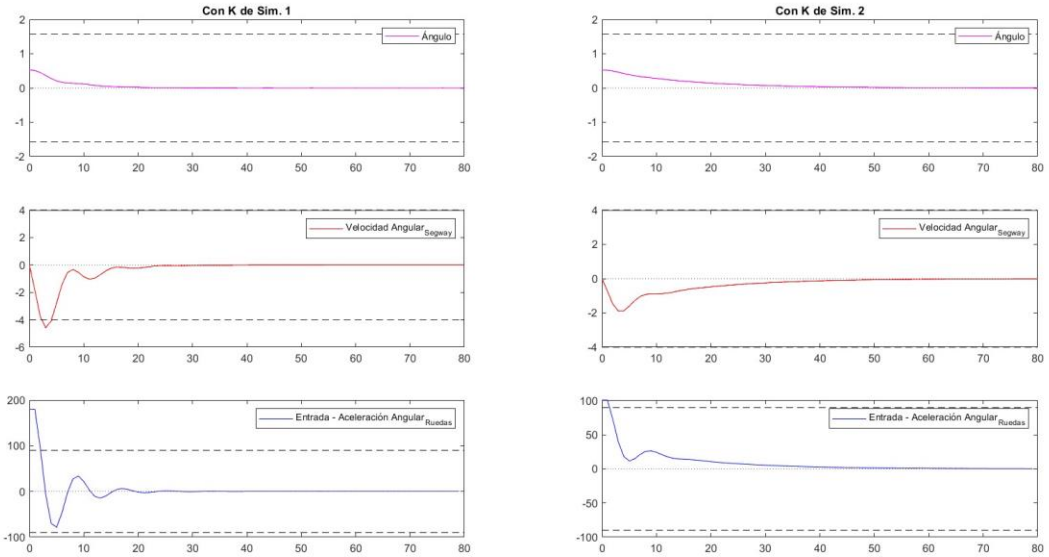

Figura 33. 3ª simulación con modelo no lineal con un ángulo inicial de 30º.

En la figura 32, se muestra el resultado de la 3ª simulación usando la K tanto de la 1ª simulación como de la 2ª simulación para un ángulo inicial de -45º.

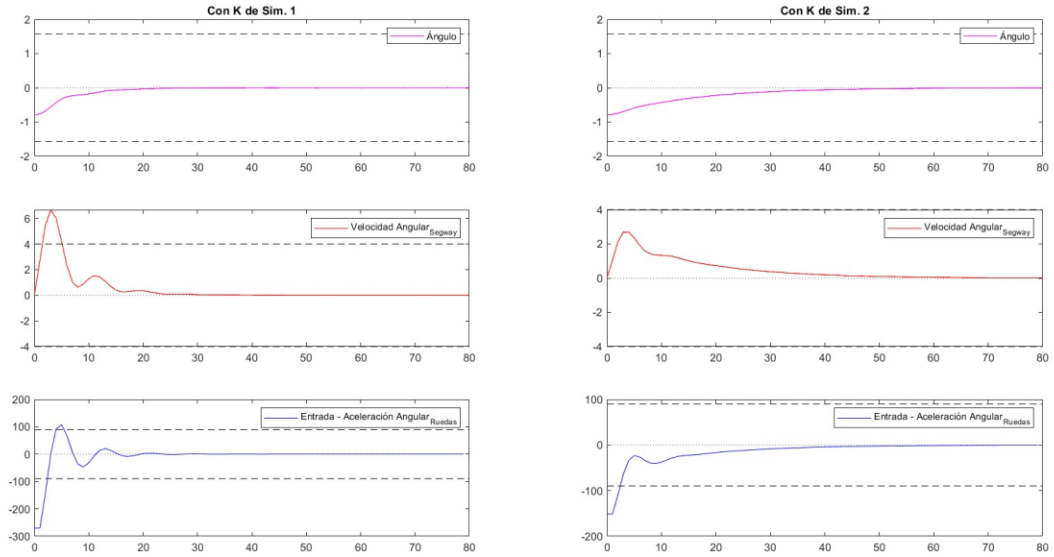

Figura 34. 3ª simulación con modelo no lineal con un ángulo inicial de -45º.

Como se puede observar, la señal de control no puede corregir el ángulo obtenido. Aplicando las restricciones como en los casos anteriores:

En la figura 33, se muestra la simulación realizada teniendo en cuenta las restricciones con un ángulo inicial de 30º.

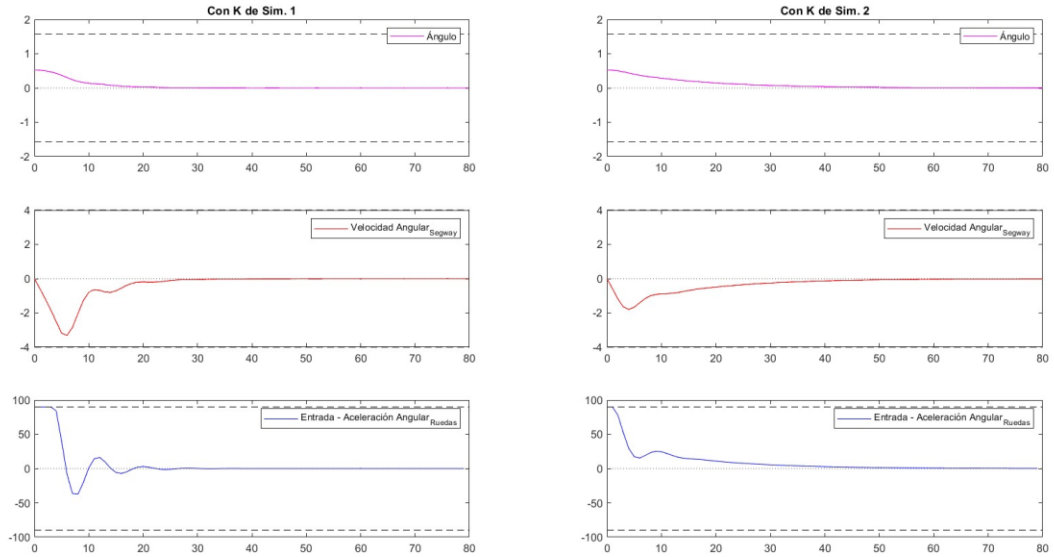

Figura 35. Simulación con restricciones con modelo no lineal con un ángulo inicial de 30º.

En la figura 34, se muestra la simulación realizada teniendo en cuenta las restricciones con un ángulo inicial de -45º.

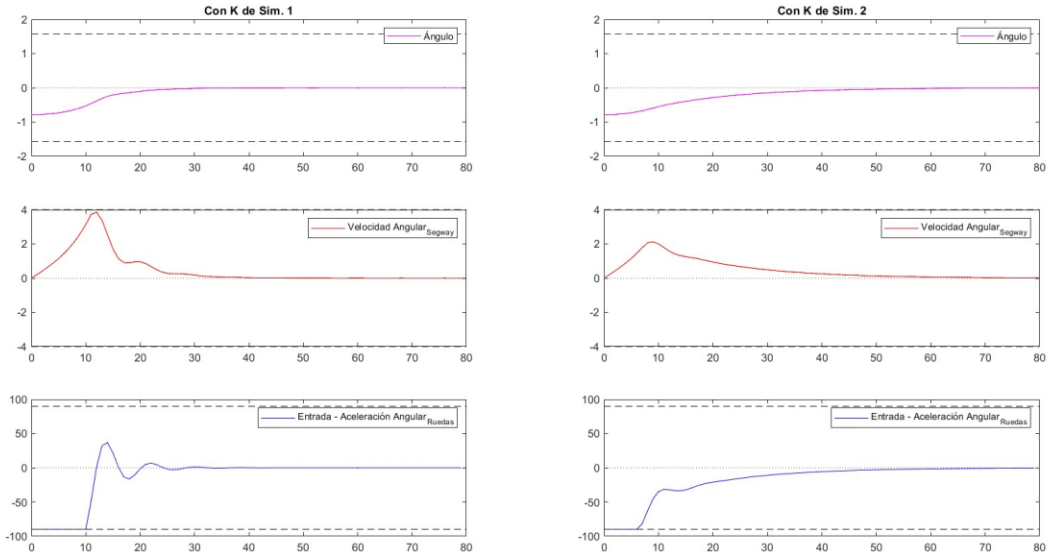

Figura 36. Simulación con restricciones con modelo no lineal con un ángulo inicial de -45º.

Todas las simulaciones

En la siguiente figura 35, se muestran todas las simulaciones anteriores comparadas cuando el ángulo inicial es -15º.

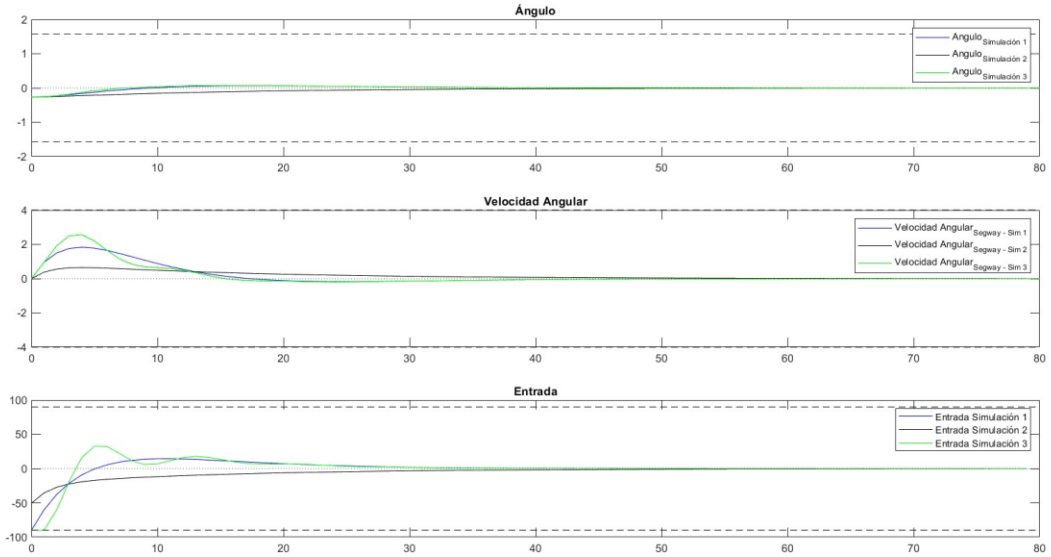

Figura 37. Comparativa entre todas las simulaciones anteriores para un ángulo inicial de -15º.

En la figura 36, se muestran todas las simulaciones anteriores comparadas cuando el ángulo inicial es 30º.

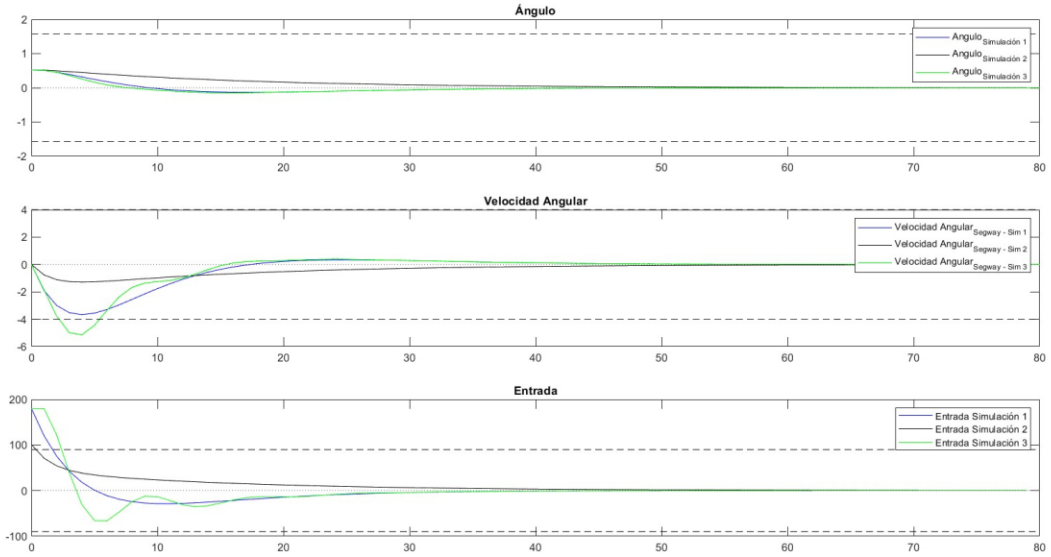

Figura 38. Comparativa entre todas las simulaciones anteriores para un ángulo inicial de 30º.

En la figura 37, se muestran todas las simulaciones anteriores comparadas cuando el ángulo inicial es -45º.

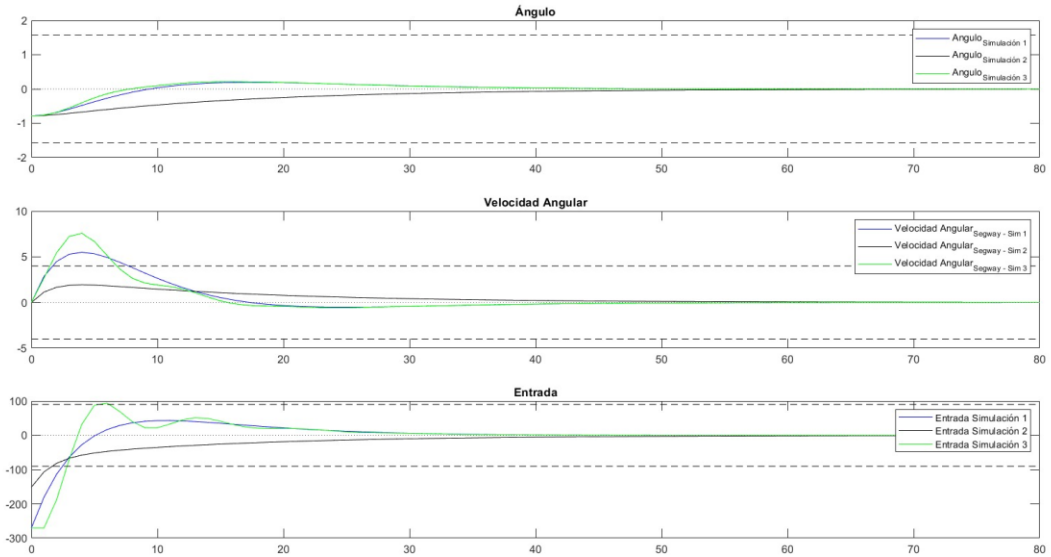

Figura 39. Comparativa entre todas las simulaciones anteriores para un ángulo inicial de -45º.

En la sección 9, se comentan las gráficas obtenidas en estas simulaciones.

5.3.- Resultados obtenidos

Para implementar el control LQR en el código en C++ en la Raspberry Pi se hará uso de la librería *WiringPi* y *WiringSerial*. Estas librerías permiten obtener y enviar datos a través del puerto serie. Para lograrlo se hacen uso de las siguientes funciones:

- void serialPutchar (int fd, unsigned char c): envía el byte «c» a través del Puerto serie «fd».
- int serialOpen (char *device, int baud): abre el Puerto serie a través del directorio «tty» correspondiente y a una velocidad de «baudios». Para este caso se hará uso de una velocidad de 115200 baudios. El valor que devuelve es el valor «fd» usado en el resto de las funciones.
- int serialDataAvail (int fd): devuelve el número de bytes en espera en el buffer de entrada del puerto serie «fd».
- int serialGetchar (int fd): devuelve el siguiente byte disponible en el buffer del puerto serie «fd».

A diferencia del proyecto anterior, en este se hará uso del puerto serie, por ello es necesario cambiar el bucle que ejecuta la función «Manejador». Ahora el algoritmo de ejecución será el siguiente:

Obtención de «ax», «ay» y «az» del MPU6050
Obtención de «gy» del MPU6050
Obtención de $\dot{\theta}$
Cálculo del controlador correspondiente
Envío del dato de aceleración a Arduino
} 20 ms

1.- Obtención de «ax», «ay» y «az» del MPU6050.

```
case 1:
    for (int i = 0; i < sizeof(miEstructura); i++) {
        buffer[i] = serialGetchar(fd);
    }
    memcpy(&datos, buffer, sizeof(miEstructura));

    ang_x = uint2int(datos.angByte_x);
    ang_y = uint2int(datos.angByte_y);
    ang_z = uint2int(datos.angByte_z);

    ax_escalado = round((ang_x - A_OFF_X)*1000.0 / ACCEL_SENS)/1000.0;
    ay_escalado = round((ang_y - A_OFF_Y)*1000.0 / ACCEL_SENS)/1000.0;
    az_escalado = round((ang_z - A_OFF_Z)*1000.0 / ACCEL_SENS)/1000.0;

  break;
```

En este fragmento de código y en los siguientes, el método de obtención de los valores se hará byte a byte a través de un bucle «for». Una vez recibidos todos los bytes, se podrá reconstruir la estructura enviada desde Arduino.

5.- Implementación del control LQR

La función «uint2int» es la siguiente:

```
int uint2int(uint16_t valor) {
      int salida = (int)valor;
      if (salida > 32768) {
            salida -= 65536;
      }
      return salida;

  }
```

Es una función que convierte un dato tipo «uint8_t» (es decir, un byte en código) a un número entero.

2.- Obtención de «gy» del MPU6050.

```
case 2:
      for (int i = 0; i < sizeof(uint16_t); i++) {
        buffer[i] = serialGetchar(fd);
    }
    memcpy(&gyByte, buffer, sizeof(uint16_t));

    gy = uint2int(gyByte);
    gy_escalado = round((gy - G_OFF_Y)*1000.0 / GYRO_SENS)/1000.0;

    accel_ang_y = -atan2(ax_escalado, (sqrt((az_escalado*az_escalado) +
(ay_escalado*ay_escalado))))*(180/PI);
    giros_ang_y = filtro(accel_ang_y,gy_escalado);

  break;
```

En este fragmento de código se obtiene «gy» a partir del mismo método usado en el fragmento de código anterior. Luego mediante una ecuación se obtiene el valor del ángulo y de la velocidad angular que pertenecen a la matriz de las variables de estado.

La función «filtro» es la siguiente:

```
float filtro(float accel, float gxy) {
      dtt = (double)(clock() - iniciall)/CLOCKS_PER_SEC;
      giros_ang_y_prev = TAU*(giros_ang_y_prev + dtt*gxy) + (1-TAU)*accel;
      iniciall = clock();
      return(giros_ang_y_prev);

  }
```

Es un filtro complementario que obtiene el valor del ángulo y velocidad angular a través tanto del acelerómetro como del giroscopio. De esta forma el ruido obtenido se reduce considerablemente.

3.- Obtención de $\dot{\theta}$.

```
case 3:
      for (int i = 0; i < sizeof(uint16_t); i++) {
        buffer[i] = serialGetchar(fd);
    }
    memcpy(&omegaByte, buffer, sizeof(uint16_t));

    omega = uint2float(omegaByte);

  break;
```

De la misma forma que en los casos anteriores, se obtiene el dato a través del bucle. Ahora se hará uso de una función llamada «uint2float» que permite convertir un dato tipo «uint8_t» (byte) a una variable tipo «float». Esta función se define a continuación:

```
float uint2float(uint16_t valor) {
      float salida = (float)valor;
      salida -= 32768;
      salida = salida/100;
      return salida;

  }
```

4.- Cálculo del control LQR.

En este caso el control que se va a aplicar es el LQR. Para ello se hará uso del vector de realimentación K calculado anteriormente. En el siguiente fragmento de código se define como se implementa:

5.- Implementación del control LQR

```
case 4:
    alpha = calculo_lqr(omega, giros_ang_y);
break;

...

float calculo_lqr(float A, float gir) {
      phi = (gir + offset)*DEG_TO_RAD;
      dphi = gy_escalado*DEG_TO_RAD;
      dtheta = A;
    u = phi*K[0] + dphi*K[1] + (A)*K[2];
    return (u);

  }
```

5.- Envío del dato de aceleración a Arduino.

```
case 5:
    for (int i = 0; i < 4; i++) {
        serialPutchar(fd, *((char *)&alpha + i));
    }
    ciclo = 0;

  break;
```

En este caso se enviará en cada iteración del bucle «for» el byte correspondiente de la variable «alpha» calculada en el paso anterior.

Para la obtención de los datos, se ha hecho uso de la terminal de Raspberry Pi OS donde se irán mostrando en pantalla los datos en tiempo real de las diferentes variables. Estos datos son transferidos a un ordenador y se grafican en Matlab. Los resultados que se han obtenido se muestran a continuación.

En la figura 38 se muestran los resultados obtenidos para un ángulo inicial de 0º.

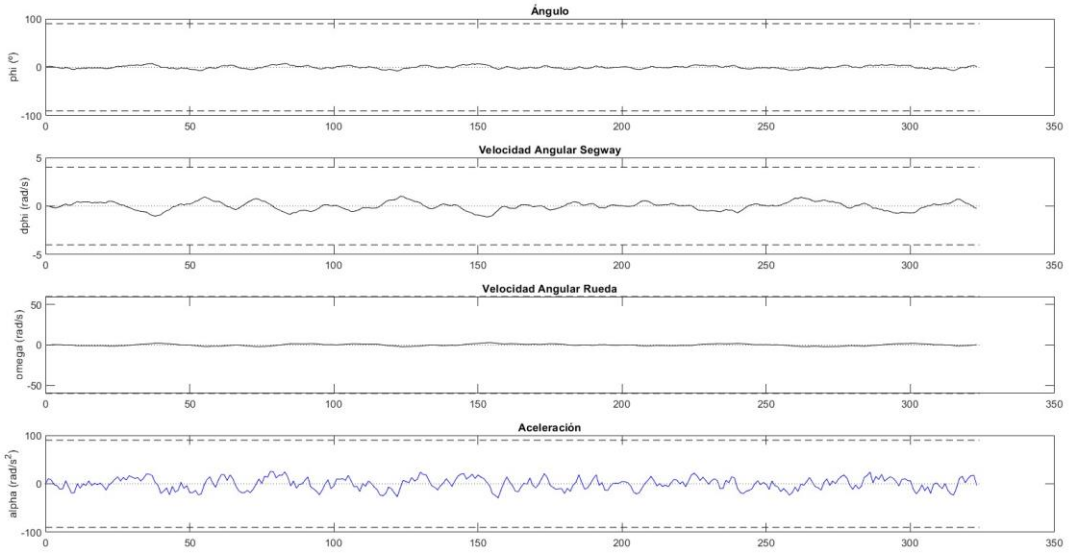

Figura 40. Resultados obtenidos usando LQR para un ángulo inicial de 0º.

En la figura 39 se muestran los resultados obtenidos para un ángulo inicial de 15º.

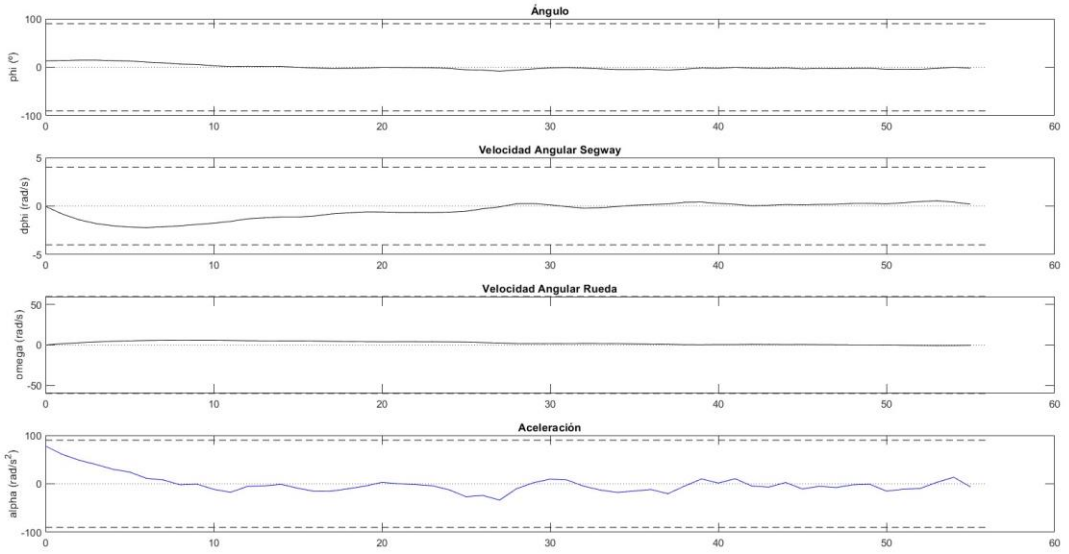

Figura 41. Resultados obtenidos usando LQR para un ángulo inicial de 15º.

En la figura 40 se muestran los resultados obtenidos para un ángulo inicial de 30º.

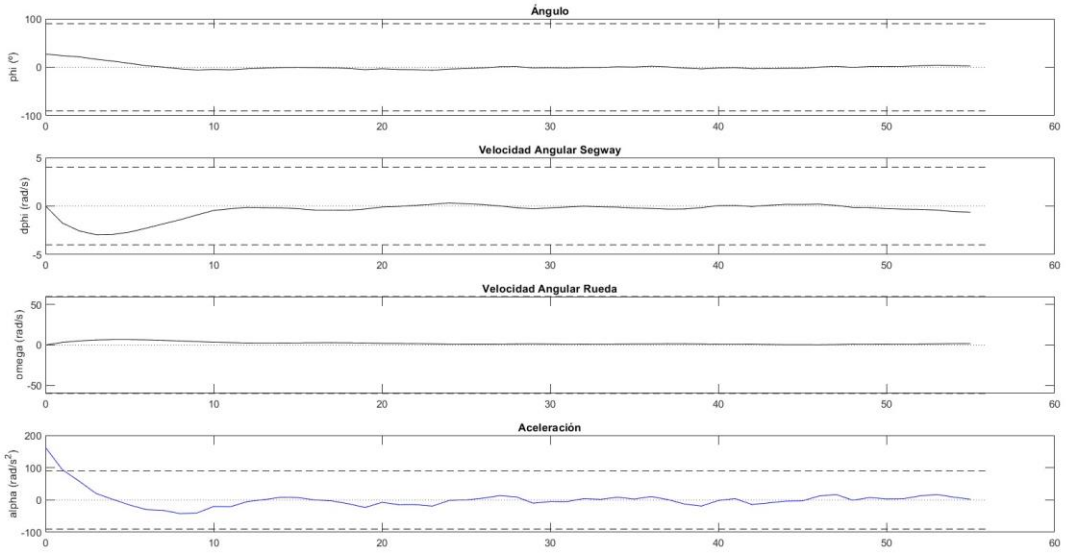

Figura 42. Resultados obtenidos usando LQR para un ángulo inicial de 30º.

Para un ángulo inicial de 45º. Hay que tener en cuenta que, en la mayoría de los ensayos realizados el Segway no pudo corregir el ángulo.

6.- Implementación del control MPC lineal

6.1.- Diseño del controlador MPC lineal con SPCIES

Para el diseño del controlador MPC lineal con SPCIES el procedimiento será diseñar en primer lugar un MPC lineal sin emplear SPCIES. Se comprobará que empleando el MPC lineal sin restricciones se obtiene un resultado similar al obtenido con LQR. Para conseguir que el MPC sea completamente equivalente al LQR, hay que diseñar un controlador cuyo horizonte de control sea igual al horizonte de predicción. En este caso no se hará uso de la restricción terminal, lo que se conoce como Lax MPC. A su vez, se empleará un coste terminal $\left(\mathrm{x(N)^T \cdot P \cdot x(N)}\right)$.

La restricción terminal es aquella en la que se establece que el estado final del sistema será un valor concreto:

$$\begin{array}{ll} min & \frac{1}{2}u^T Hu + h^T u + \boxed{x(N)^T \cdot P \cdot x(N)} \\ s.t. & Nu = n \\ & Mu \leq m \\ & \boxed{x(N) = X_f} \end{array}$$

Donde X_f es el valor final que se desea para la variable de entrada y P es la matriz de Riccati mencionada anteriormente. Al no tener restricción terminal, la restricción marcada no estará definida.

Las restricciones se han aplicado haciendo uso de las matrices Au, bu, Ax y bx, para poder calcular las matrices M, m, N y n

$$\mathrm{Au \cdot u \leq bu}$$
$$\mathrm{Ax \cdot x \leq bx}$$

Se quieren implementar las siguientes restricciones:

$$-90 \leq \mathrm{u} \leq +90$$
$$-\frac{\mathrm{pi}}{2} \leq \mathrm{x_1} \leq +\frac{\mathrm{pi}}{2}$$
$$-4 \leq \mathrm{x_2} \leq +4$$
$$-60 \leq \mathrm{x_3} \leq +60$$

Por tanto, las matrices deben ser:

$$\mathrm{Au} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad \mathrm{bu} = \begin{bmatrix} 90 \\ 90 \end{bmatrix}$$

$$\mathrm{Ax} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix} \quad \mathrm{bx} = \begin{bmatrix} \pi/2 \\ \pi/2 \\ 4 \\ 4 \\ 60 \\ 60 \end{bmatrix}$$

Y así, se consiguen las siguientes restricciones:

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} \cdot u \leq \begin{bmatrix} 90 \\ 90 \end{bmatrix} \rightarrow \begin{matrix} u \leq 90 \\ -u \leq 90 \end{matrix} \rightarrow \begin{matrix} u \leq 90 \\ u \geq -90 \end{matrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leq \begin{bmatrix} \pi/2 \\ \pi/2 \\ 4 \\ 4 \\ 60 \\ 60 \end{bmatrix} \rightarrow \begin{matrix} x_1 \leq \pi/2 \\ -x_1 \leq \pi/2 \\ x_2 \leq 4 \\ -x_2 \leq 4 \\ x_3 \leq 60 \\ -x_3 \leq 60 \end{matrix} \rightarrow \begin{matrix} x_1 \leq \pi/2 \\ x_1 \geq \pi/2 \\ x_2 \leq 4 \\ x_2 \geq 4 \\ x_3 \leq 60 \\ x_3 \geq 60 \end{matrix}$$

En Matlab se define de la siguiente forma:

```
% Longitud de las matrices del espacio de estados
n = size(Ad, 1);
m = size(Bd, 2);

% Restricciones del sistema
LBx = -[pi/2; 4; 60];
UBx =  [pi/2; 4; 60];
LBu = -[90];
UBu =  [90];

% Descripción del sistema

  sys = struct('A', Ad, 'B', Bd, 'LBx', LBx, 'UBx', UBx, 'LBu', LBu, 'UBu',
  UBu);
```

En este caso, no existen restricciones de igualdad, por tanto, ahora se deben calcular unas matrices auxiliares M_u, m_u, M_x y m_x, para proseguir con el cálculo de M, m, N y n:

$$M_u = blkdiag(Au, \dots, Au)$$
$$m_u = [bu, \dots, bu]$$

$$M_x = blkdiag(Ax, Ax, \dots, Ax)$$
$$m_x = [bx, \dots, bx]$$

Entonces las matrices M y m son:

$$M = \begin{bmatrix} M_u \\ M_x G \end{bmatrix} \quad m = \begin{bmatrix} m_u \\ m_x - M_x \cdot F_x \cdot x \end{bmatrix}$$

La herramienta SPCIES permite crear todas estas matrices de forma automática a través de unos parámetros de diseños que deben ser seleccionados:

- rho: es el valor que se le da al parámetro de penalización del algoritmo ADMM. Cuanto mayor sea este valor, mayor relevancia tendrán los errores obtenidos en la obtención del resultado esperado.
- k máx: es el máximo número de iteraciones que el solucionador realizará para encontrar la solución

óptima. Si se sobrepasan las iteraciones el sistema devolverá el mejor valor encontrado.

– Tol: es la tolerancia del solucionador. Cuando mayor sea la tolerancia, más imprecisa será la solución óptima encontrada.

```
%% Generar el solucionador
solver_options.rho = 15; % Valor del parámetro de penalización
solver_options.k_max = 1000; % Número máximo de iteraciones
solver_options.tol = 1e-3; % Tolerancia

% Guardar el archivo que contiene el MPC
options.save_name = 'mpc_solver';
options.directory = '';
options.time = true; % Contar el tiempo internamente

spcies_clear;

% Generar el archivo MEX
spcies_gen_controller('sys', sys, 'param', param, 'solver_options',
solver_options,'options', options, 'platform', 'Matlab', 'type', 'laxMPC');
```

6.2.- Simulaciones

Como se ha mencionado, en primer lugar, se busca que el valor de las variables de estado sea igual tanto en el LQR como en el MPC. En la figura 41, se muestra como ambas simulaciones reflejan el mismo resultado:

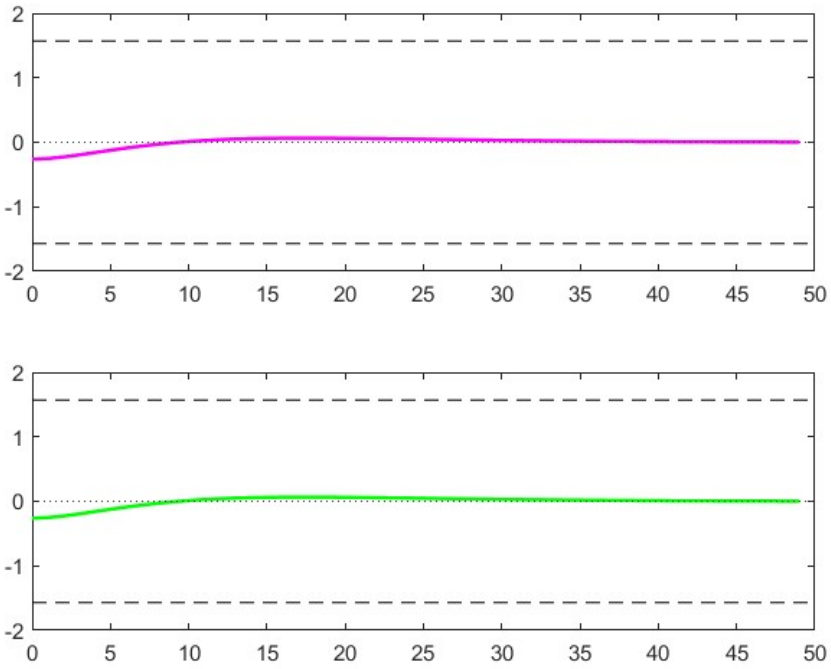

Figura 43. Comparativa entre la simulación obtenida en LQR y MPC.

Como se puede observar, los valores obtenidos son iguales. Hay que tener en cuenta que en este caso no se han aplicado restricciones al sistema. En las siguientes simulaciones, se mostrarán las gráficas de desarrollo del sistema partiendo de un estado inicial de -15º, 30º y -45º. En todas ellas, se aplicarán las restricciones para comparar.

En la figura 42, se muestra la simulación comparando el resultado obtenido con LQR y MPC lineal teniendo en cuenta las restricciones con un ángulo inicial de -15º.

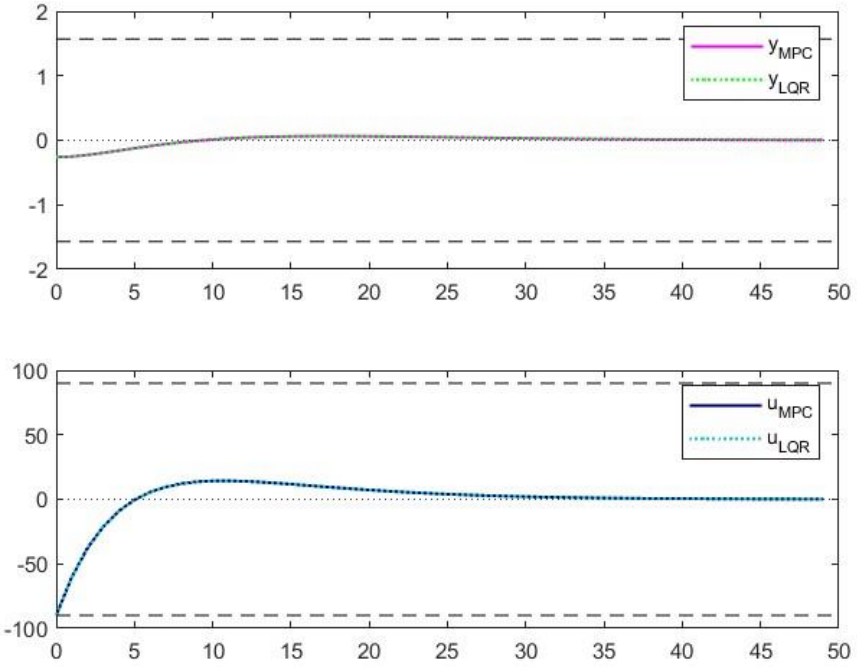

Figura 44. Comparativa entre la simulación obtenida en LQR y MPC con un ángulo inicial de -15º.

En la figura 43, se muestra la simulación comparando el resultado obtenido con LQR y MPC lineal teniendo en cuenta las restricciones con un ángulo inicial de 30º.

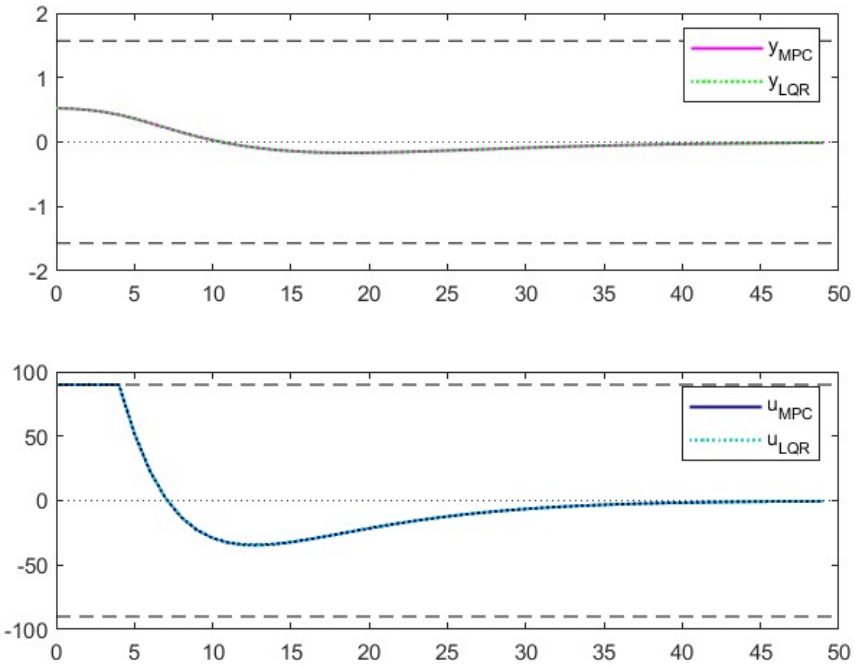

Figura 45. Comparativa entre la simulación obtenida en LQR y MPC con un ángulo inicial de 30º.

En la figura 44, se muestra la simulación comparando el resultado obtenido con LQR y MPC lineal teniendo en cuenta las restricciones con un ángulo inicial de -45º.

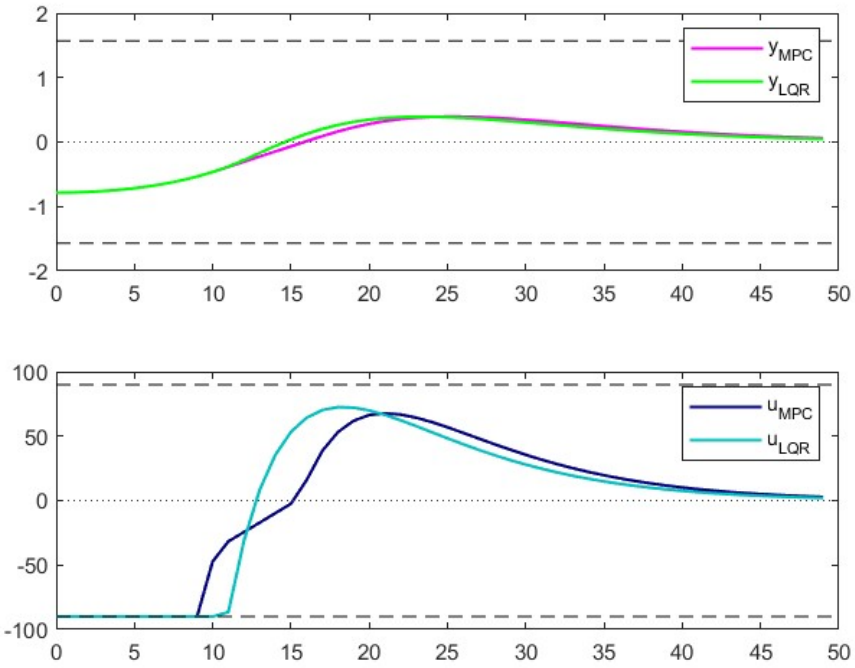

Figura 46. Comparativa entre la simulación obtenida en LQR y MPC con un ángulo inicial de -45º.

Se puede observar que el comportamiento del sistema no varía mucho entre el LQR y el MPC.

A continuación, se mostrarán los resultados obtenidos utilizando la herramienta SPCIES y comparándola con los resultados obtenidos en el MPC diseñado manualmente.

En la figura 45, se muestra la comparación entre ambos métodos con un ángulo inicial de -15º.

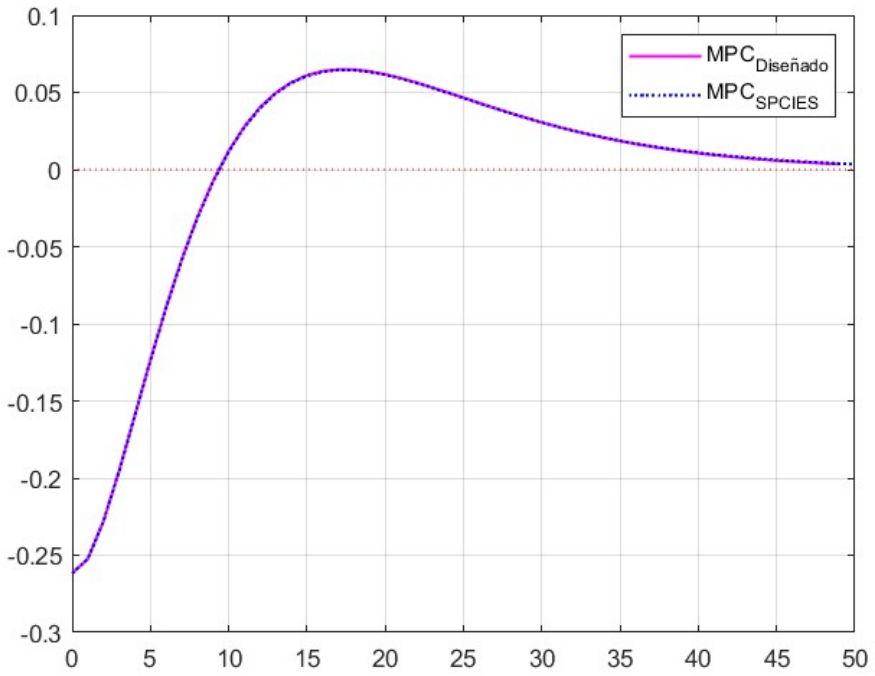

Figura 47. Comparativa entre las simulaciones de SPCIES y MPC programado con un ángulo inicial de -15º.

En la figura 46, se muestra la comparación entre ambos métodos con un ángulo inicial de 30º.

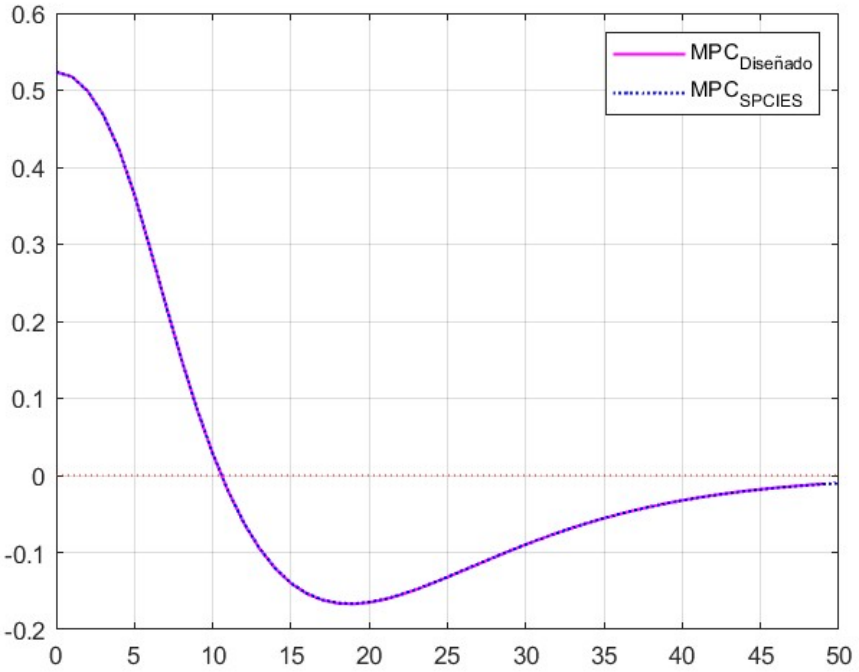

Figura 48. Comparativa entre las simulaciones de SPCIES y MPC programado con un ángulo inicial de 30º.

En la figura 47, se muestra la comparación entre ambos métodos con un ángulo inicial de -45º.

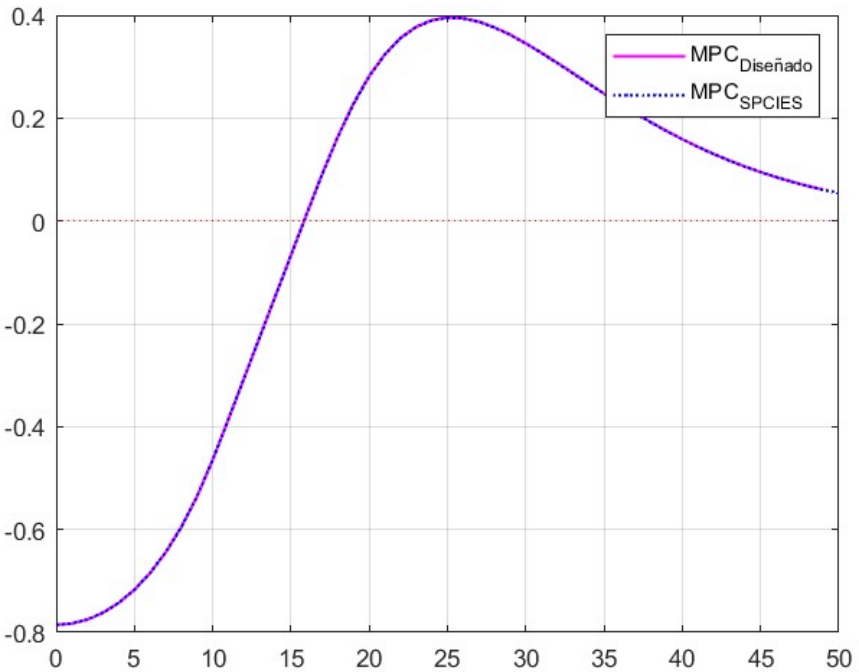

Figura 49. Comparativa entre las simulaciones de SPCIES y MPC programado con un ángulo inicial de -45º.

Se puede apreciar que las simulaciones obtenidas son exactamente iguales. Esto puede cambiar si se modifican los parámetros de diseño por unos más restrictivos. Por ejemplo, aumentando la tolerancia y

disminuyendo el número de iteraciones máximas que puede realizar:

En la figura 48, se muestra la simulación obtenida comparando ambos métodos, pero aplicando parámetros más restrictivos en SPCIES.

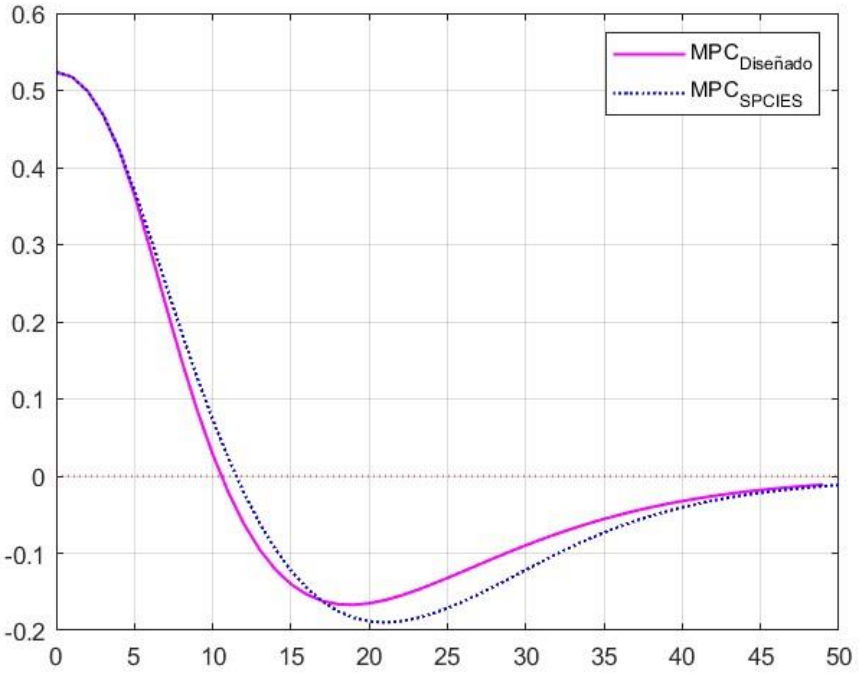

Figura 50. Comparativa entre las simulaciones de SPCIES y MPC programado, pero con parámetros más restrictivos en SPCIES.

De hecho, si los parámetros se vuelven demasiado restrictivos, el sistema no podrá alcanzar la solución mejor solución posible, como se muestra en la figura 49.

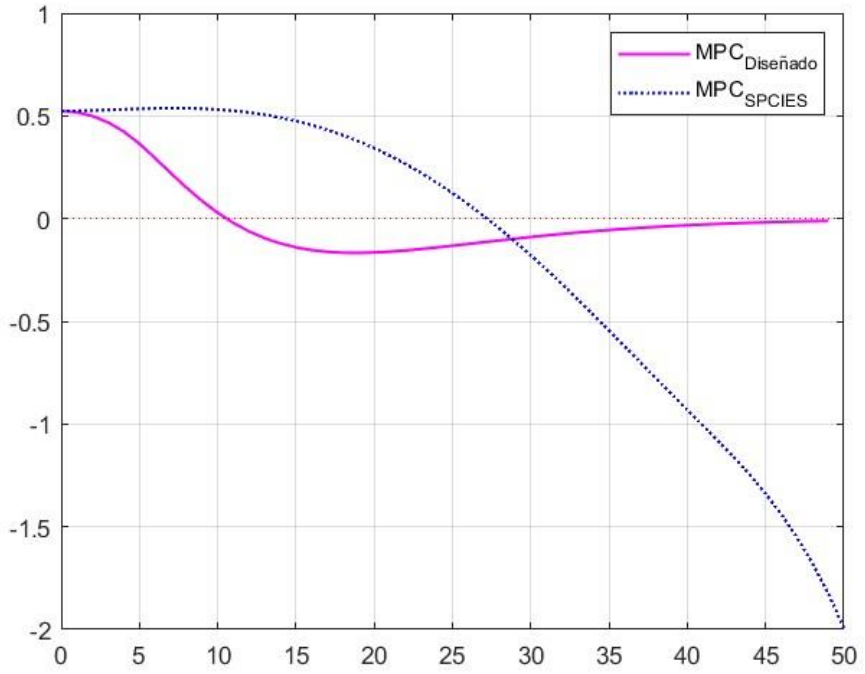

Figura 51. Comparativa entre las simulaciones de SPCIES y MPC programado, pero con parámetros aún más restrictivos en SPCIES.

Ahora se va a comparar las simulaciones obtenidas empleando la herramienta SPCIES para el modelo del sistema lineal y el modelo no lineal.

En la figura 50, se muestra la comparación entre las simulaciones del modelo lineal y no lineal empleando SPCIES con un ángulo inicial de -15º.

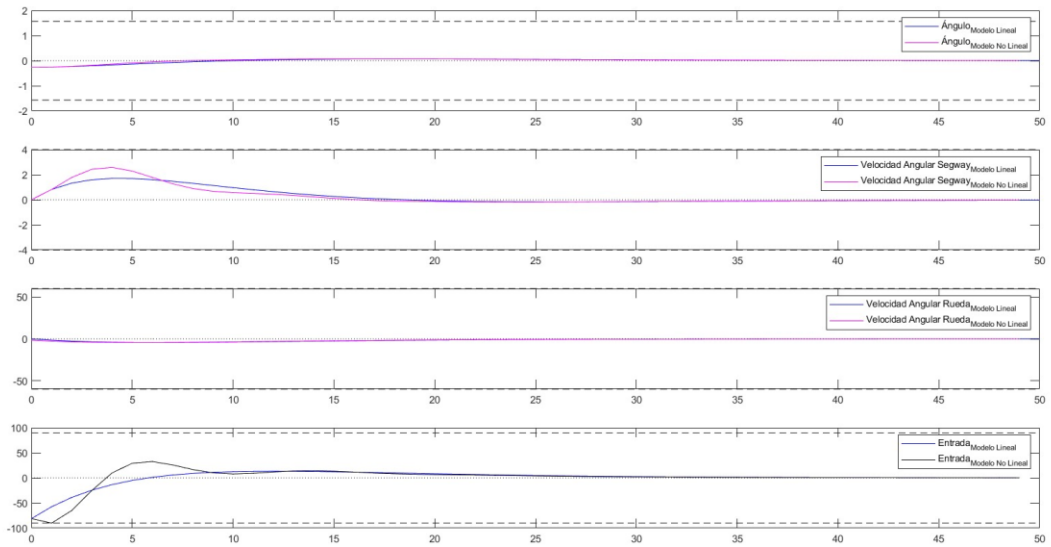

Figura 52. Comparativa entre simulaciones usando SPCIES en el modelo lineal y no lineal con un ángulo inicial de -15º.

En la figura 51, se muestra la comparación entre las simulaciones del modelo lineal y no lineal empleando SPCIES con un ángulo inicial de 30º.

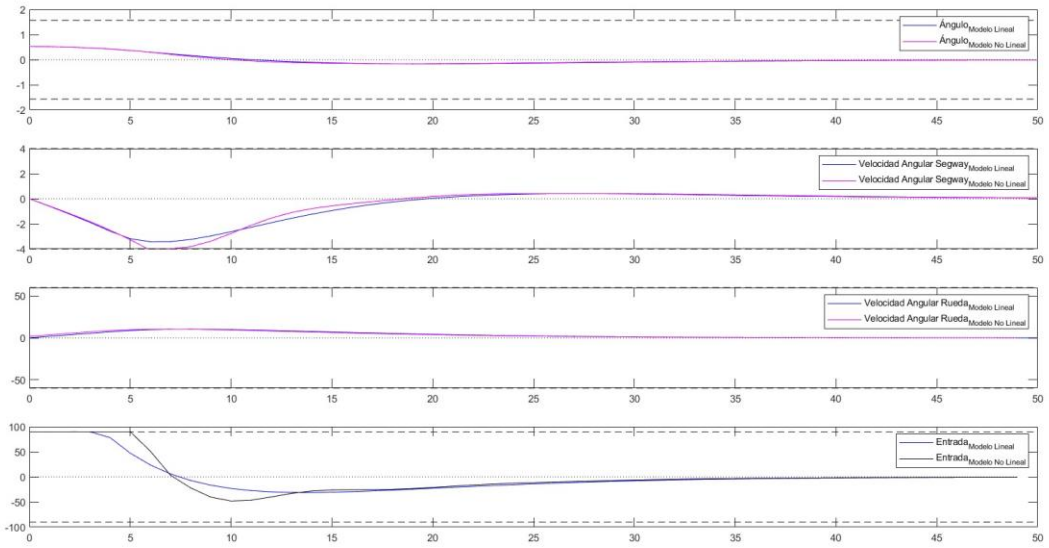

Figura 53. Comparativa entre simulaciones usando SPCIES en el modelo lineal y no lineal con un ángulo inicial de 30º.

En la figura 52, se muestra la comparación entre las simulaciones del modelo lineal y no lineal empleando SPCIES con un ángulo inicial de -45º.

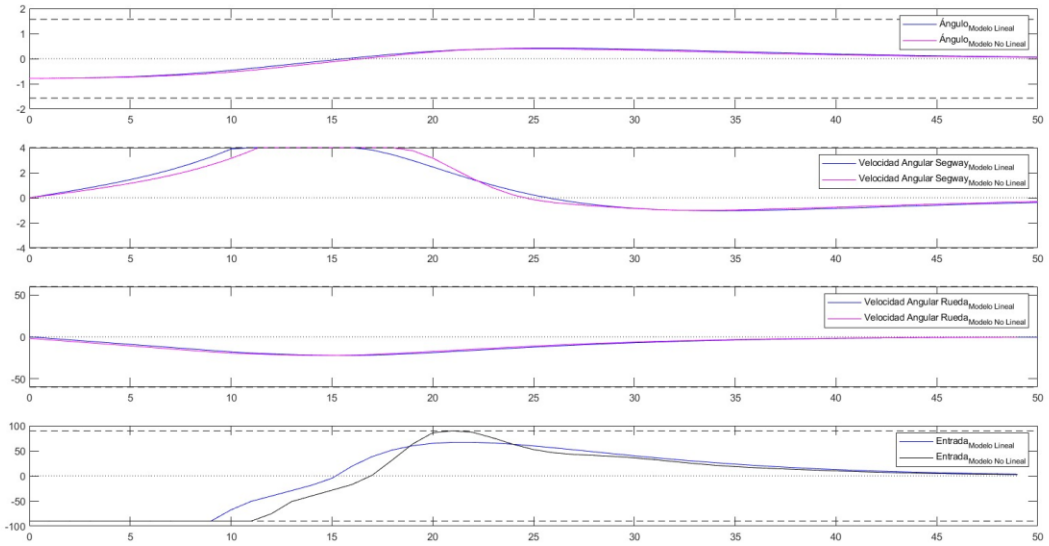

Figura 54. Comparativa entre simulaciones usando SPCIES en el modelo lineal y no lineal con un ángulo inicial de -45º.

Se puede observar que a pesar de que la herramienta SPCIES trabaja con un modelo lineal, el modelo no lineal es capaz de corregir la posición del Segway.

A continuación, se va a comprobar si el MPC diseñado con las matrices obtenidas de la identificación de estados puede corregir adecuadamente el error en tiempo real:

En la figura 55, se muestra la comparación entre las simulaciones del modelo lineal y no lineal empleando SPCIES con las matrices de la identificación de estados con un ángulo inicial de -15º.

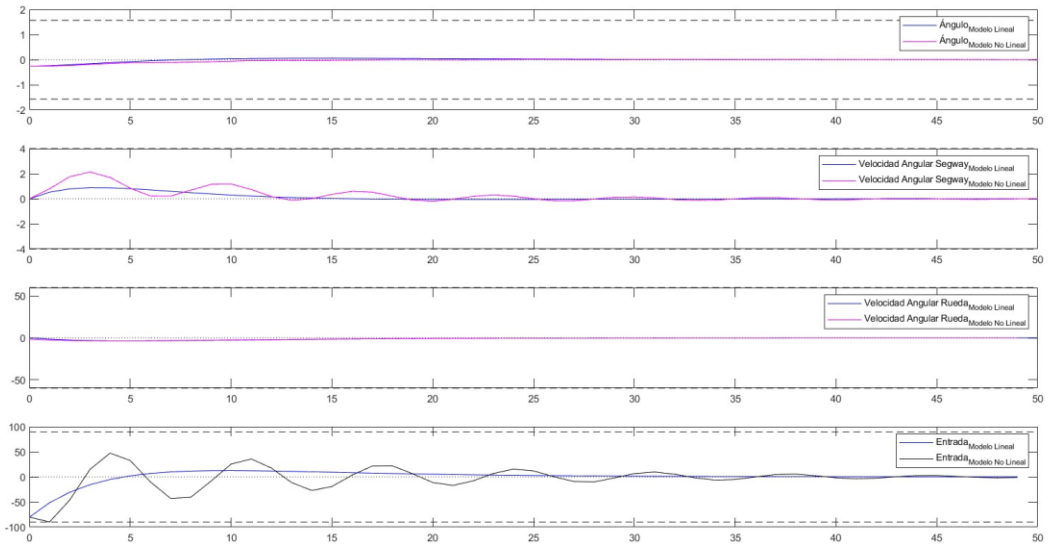

Figura 55. Comparativa entre simulaciones usando SPCIES en el modelo lineal y no lineal usando las matrices obtenidas por identificación de estados con un ángulo inicial de -15º.

En la figura 56, se muestra la comparación entre las simulaciones del modelo lineal y no lineal empleando SPCIES con las matrices de la identificación de estados con un ángulo inicial de 30º.

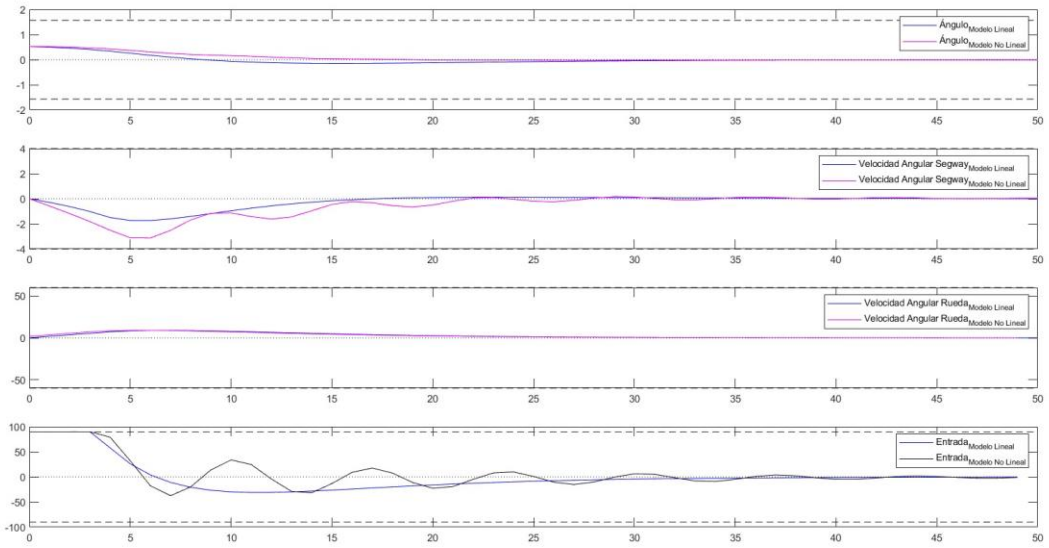

Figura 56. Comparativa entre simulaciones usando SPCIES en el modelo lineal y no lineal usando las matrices obtenidas por identificación de estados con un ángulo inicial de 30º.

En la figura 57, se muestra la comparación entre las simulaciones del modelo lineal y no lineal empleando SPCIES con las matrices de la identificación de estados con un ángulo inicial de -45º.

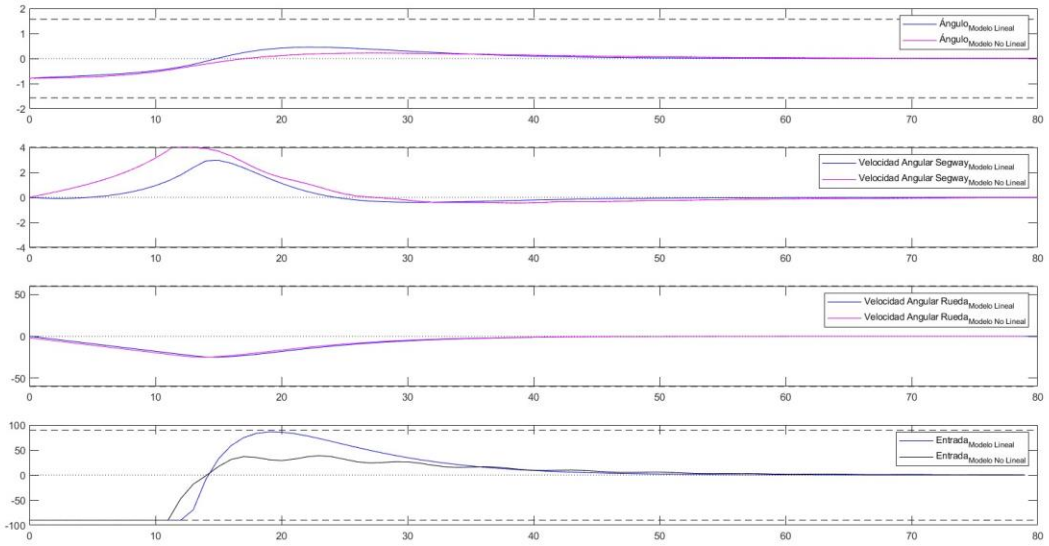

Figura 57. Comparativa entre simulaciones usando SPCIES en el modelo lineal y no lineal usando las matrices obtenidas por identificación de estados con un ángulo inicial de -45º.

Se puede observar que a pesar de que los datos oscilan, finalmente convergen en el punto de referencia deseado.

6.3.- Resultados obtenidos

Para implementar el control MPC lineal en la Raspberry Pi es necesario obtener los archivos con extensión «.c» y «.h» que se han creado al ejecutar el código de SPCIES de Matlab con la opción 'C':

```
% Generar el archivo MEX
spcies_gen_controller('sys', sys, 'param', param, 'solver_options',
solver_options,'options', options, 'platform', 'C', 'type', 'laxMPC');
```

Los archivos obtenidos tienen como nombre «mpc_solver.h» y «mpc_solver.c». El archivo «mpc_solver.h» debe ser incluido junto al resto de cabeceras del código.

El paso 4 anteriormente visto se calcula de la siguiente forma:

```
case 4:
    laxMPC_ADMM(x0SP, xrSP, urSP, u_optSP, pointer_kSP, e_flagSP, solSP);
    alpha = u_optSP[0];
break;
```

Las variables que deben introducirse en la función «laxMPC_ADMM» son:

- double x0SP: es el valor inicial de la matriz de variables de estado. Para este caso, será la matriz que contiene los valores leídos en ese instante de tiempo desde la placa Arduino.
- double xrSP: es la matriz que contiene las referencias que se quieren alcanzar de cada variable de estado.
- double urSP: es la matriz que contiene la referencia que se quiere alcanzar de la entrada.

- double u_optSP: es la matriz que contiene el valor de la aceleración óptima. Este valor es el que se enviará a la placa Arduino.
- int pointer_kSP: es el número de iteracciones necesarias para calcular el valor de u_optSP.

Las otras dos variables son variables internas que emplea el propio solver.

El resultado obtenido se muestra a continuación:

En la figura 58 se muestra el resultado obtenido empleando SPCIES para un ángulo inicial de 0º.

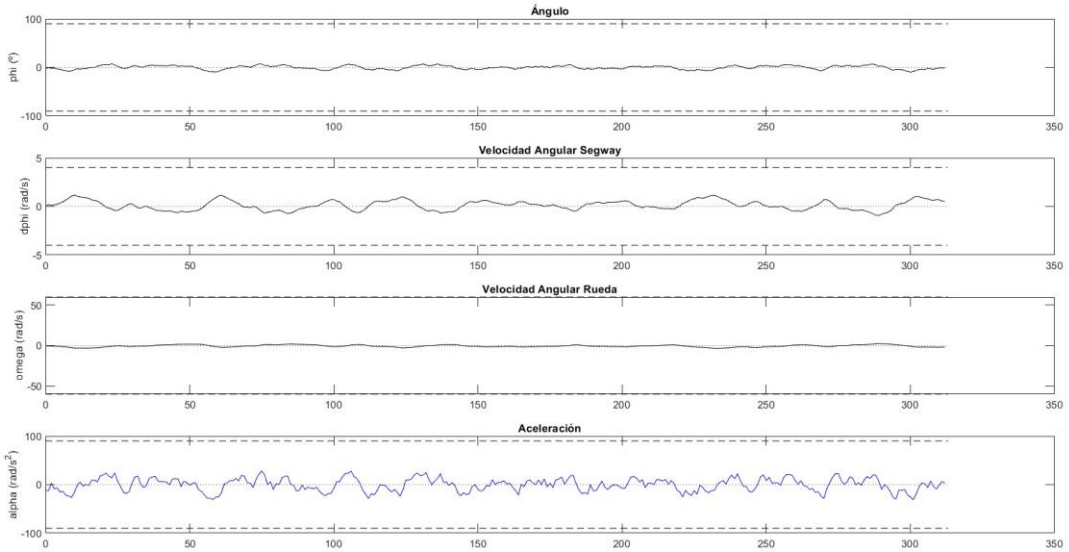

Figura 58. Resultado obtenido empleando SPCIES para un ángulo inicial de 0º.

En la figura 59, se muestra el resultado obtenido empleando SPCIES para un ángulo inicial de 15º.

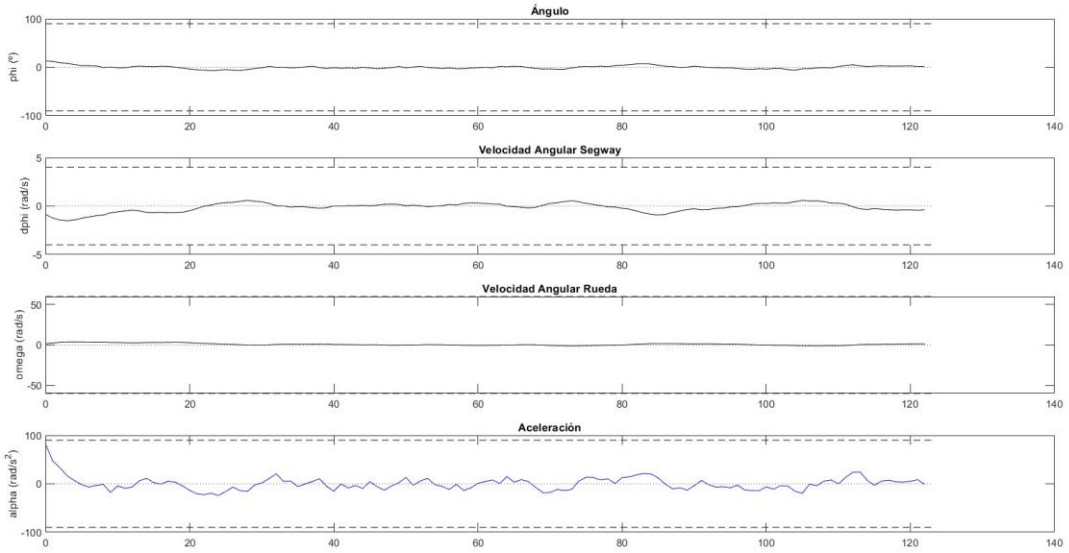

Figura 59. Resultado obtenido empleando SPCIES para un ángulo inicial de 15º.

En la figura 60, se muestra el resultado obtenido empleando SPCIES para un ángulo inicial de 30º.

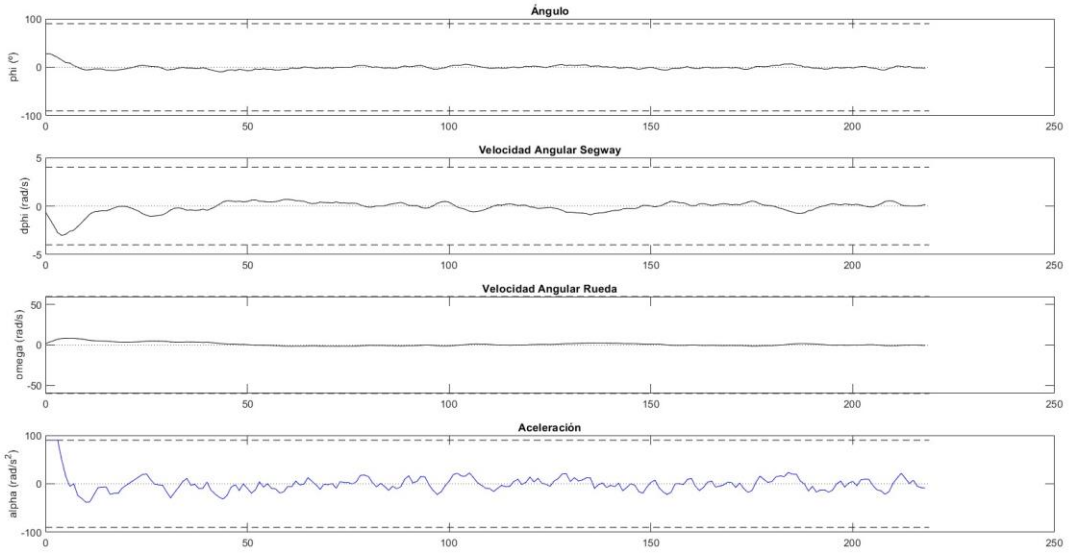

Figura 60. Resultado obtenido empleando SPCIES para un ángulo inicial de 30º.

En la figura 61, se muestra el resultado obtenido empleando SPCIES para un ángulo inicial de 45º.

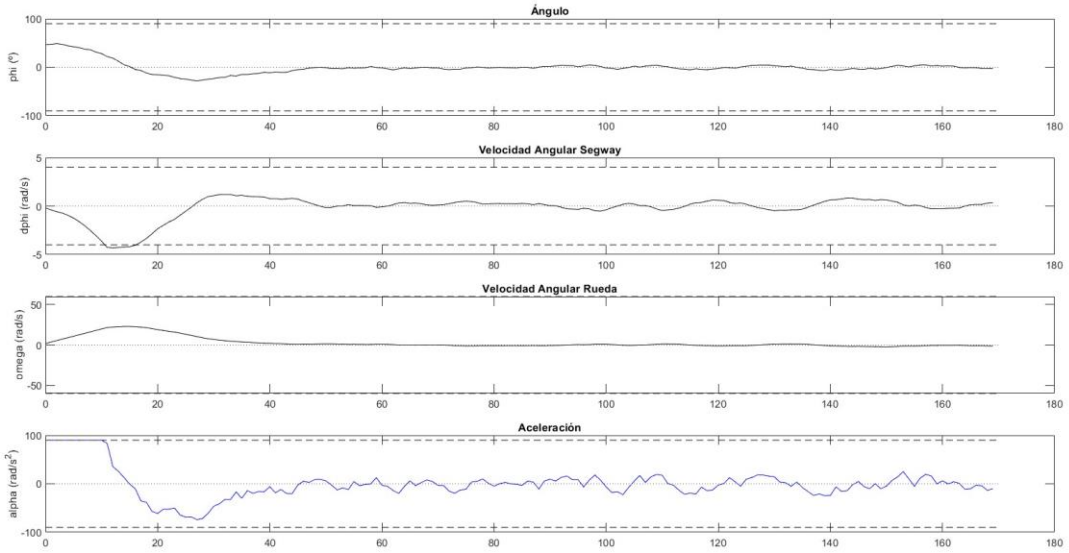

Figura 61. Resultado obtenido empleando SPCIES para un ángulo inicial de 45º.

7.- Implementación del control MPC no lineal

7.1.- Diseño del controlador MPC no lineal con CasADi

Para diseñar el MPC no lineal con CasADi es necesario en primer lugar importar la librería que contiene las funciones que serán necesarias.

En el código que se va a ejecutar será necesario definir las variables de estado y variables de control mediante una función específica de CasADi. En este caso, el espacio de estados estará formado por dos variables de estado en lugar de tres, siendo el ángulo de giro y la velocidad angular del Segway las variables que componen ahora la matriz de variables de estado.

La función de costes a minimizar será:

$$J = \sum_{i=1}^{N} \varphi_i^2 + \sum_{i=1}^{N} \dot{\varphi}_i^2 + \sum_{i=1}^{N} u_i^2$$

CasADI hará uso de una función que necesita el valor de las variables de estado y variables de control actuales y devuelve la variable de estado en el instante siguiente. Para ello se necesita discretizar la función que define el comportamiento no lineal del Segway. Esto puede lograrse usando las siguientes funciones de Matlab.

$$[VF, Sbs] = \text{odeToVectorField}(ode)$$

$$\text{odsefcn} = \text{matlabFunction}(VF, 'vars', \{'t', 'Y'\})$$

$$[t,x] = \text{ode45}(\text{odsefcn}, [k_1, k_2], [x_1, x_2])$$

Siendo:

$$ode = (a + b \cdot \cos(\varphi)) \cdot \ddot{\varphi} + (c + b \cdot \cos(\varphi)) \cdot u - b \cdot \dot{\varphi} \cdot \sin(\varphi) - d \cdot \sin(\varphi) = 0$$

Donde:

$$a = 2 \cdot m \cdot L^2$$
$$b = m \cdot R \cdot L$$
$$c = R^2 \cdot (3 \cdot m_r + m)$$
$$d = m \cdot g \cdot L$$

La función que debe proporcionarse a CasADi es:

$$f(x,u) = \begin{bmatrix} x_2 \\ -\dfrac{(c \cdot u - d \cdot \sin(x_1)) - b \cdot \sin(x_1) \cdot x_2 + b \cdot \cos(x_1) \cdot u}{a + b \cdot \cos(x_1)} \end{bmatrix}$$

Para el caso que aquí se estudia:

$$\varphi = x_1$$
$$\dot{\varphi} = x_2$$

La función devuelve, por tanto:

$$\dot{\varphi} = x_2 = \dot{\varphi}$$

$$\ddot{\varphi} = -\frac{(c \cdot u - d \cdot \sin(\varphi)) - b \cdot \sin(\varphi) \cdot \dot{\varphi} + b \cdot \cos(\varphi) \cdot u}{a + b \cdot \cos(\varphi)}$$

Esta función se aplicará en un algoritmo de integración de Runge-Kutta de 4 fases de integración para proporcionar el valor de las variables de estado en el instante siguiente. El método de Runge-Kutta es un método iterativo de resolución de ecuaciones diferenciales. [17]

El método se ha aplicado de la siguiente forma:

Paso 1:

$$k_1 = f(x_i, u_i)$$

Paso 2:

$$k_2 = f\left(x_i + \frac{\Delta t}{2} \cdot k_1, u_i\right)$$

Paso 3:

$$k_3 = f\left(x_i + \frac{\Delta t}{2} \cdot k_2, u_i\right)$$

Paso 4:

$$k_4 = f(x_i + \Delta t \cdot k_3, u_i)$$

Siendo las variables de estado del instante siguiente:

$$x_{i+1} = x_i + \frac{\Delta t}{6} \cdot (k_1 + k_2 + 2 \cdot k_3 + k_4)$$

Para implementar CasADi en Matlab se ha hecho uso del código detallado en el Anexo de este proyecto.

```
%% Importar la librería que incluye CasADi
import casadi.*

N = 25; % Horizonte de predicción

opti = casadi.Opti(); % Problema de optimización

tFinal = 50; % Número de iteracciones
```

```
% Variables de decisión
X = opti.variable(3,N+1); % Variables de estado
U = opti.variable(1,N);   % Variable de control

% Función objetivo
opti.minimize(sumsqr(X) + sumsqr(U));
```

7.- Implementación del control MPC no lineal

Se va a trabajar con un horizonte de predicción de 25.

Se declaran las variables de estado y de control junto con la función objetivo. Para la función objetivo se ha utilizado la función «sumsqr» que permite realizar la siguiente operación:

$$\text{sumsqr}(x) = \sum_{i=0}^{N} x_i^2$$

Siendo por tanto la función objetivo:

$$J = \sum_{i=0}^{N} x_i^2 + \sum_{i=0}^{N} u_i^2$$

```
% Restricciones dinámicas
    f = @(x,u) [x(2);
                -(c*u - d*sin(x(1)) - b*sin(x(1))*x(2) + b*cos(x(1))*u)/(a +
b*cos(x(1)));
                x(3) + dt*u]; % dx/dt = f(x,u)
```

Las restricciones dinámicas se definirán mediante una función cuyo contenido es el resultado de aplicar la función «odeToVectorField» a la expresión que define el comportamiento no lineal del sistema:

$$(a + b \cdot \cos\ \Phi) \cdot \ddot{\Phi} + [c + b \cdot \cos \Phi] \cdot u - b \cdot \dot{\Phi} \cdot \sin \Phi - d \cdot \sin \Phi = 0$$

Esta función convierte ecuaciones diferenciales de orden mayor que uno a ecuaciones diferenciales de orden uno.

```
for k=1:N % Obtención del estado siguiente mediante Runge Kutta
        k1 = f(X(:,k),         U(:,k));
        k2 = f(X(:,k)+dt/2*k1, U(:,k));
        k3 = f(X(:,k)+dt/2*k2, U(:,k));
        k4 = f(X(:,k)+dt*k3,   U(:,k));
        x_next = X(:,k) + dt/6*(k1+2*k2+2*k3+k4);
        opti.subject_to(X(:,k+1)==x_next); % Restricción que define el
comportamiento del Segway
    end
```

Sobre esa misma función se aplicará el algoritmo de Runge-Kutta mencionado anteriormente.

```
% Restricciones de valor inicial
opti.subject_to(X(1,1) == xNMPC(1,kk)); % phi
opti.subject_to(X(2,1) == xNMPC(2,kk)); % dphi
opti.subject_to(X(3,1) == xNMPC(3,kk)); % omega

% Restricciones de valores máximos y mínimos
opti.subject_to(-pi/2<=X(1,:)<=pi/2); % phi
opti.subject_to(-4<=X(2,:)<=4); % dphi
opti.subject_to(-60<=X(3,:)<=60); % dphi
opti.subject_to(-90<=U<=90); % u (alpha)
```

Se definen las restricciones del sistema.

Se llama al solver, que en este caso se ha utilizado el «ipopt» (Interior Point Optimizer), que es uno de los solvers más utilizados en la optimización de sistemas no lineales.

7.2.- Simulaciones

A continuación, se mostrarán las simulaciones realizadas para los diferentes ángulos de inicio que se han comentado anteriormente.

```
% Llamar al solver
opti.solver('ipopt');
sol = opti.solve();
```

En la figura 62, se muestra el desarrollo del Segway en la simulación con un ángulo inicial de -15º.

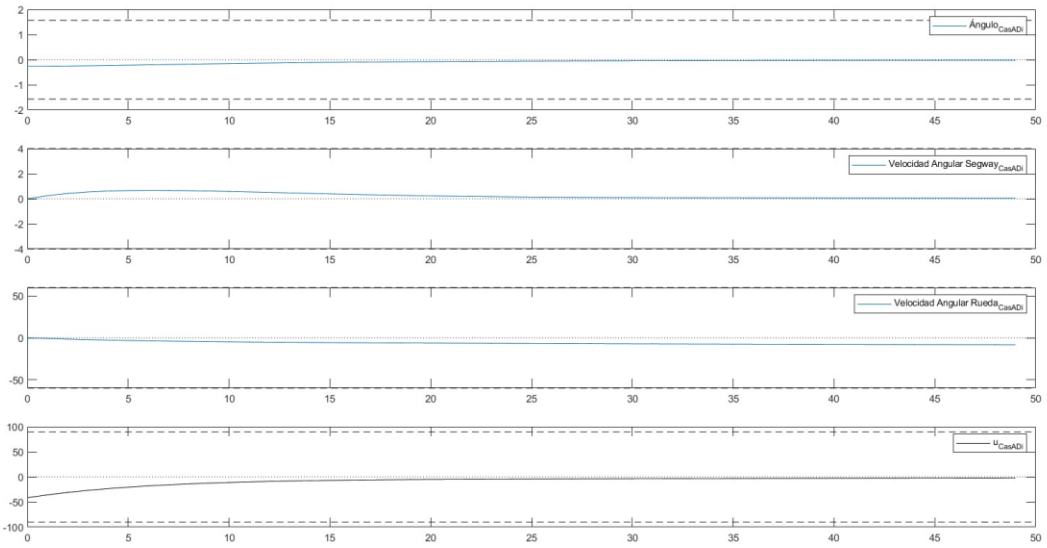

Figura 62. Resultados obtenidos aplicando CasADi con un ángulo inicial de -15º.

En la figura 63, se muestra el desarrollo del Segway en la simulación con un ángulo inicial de 30º.

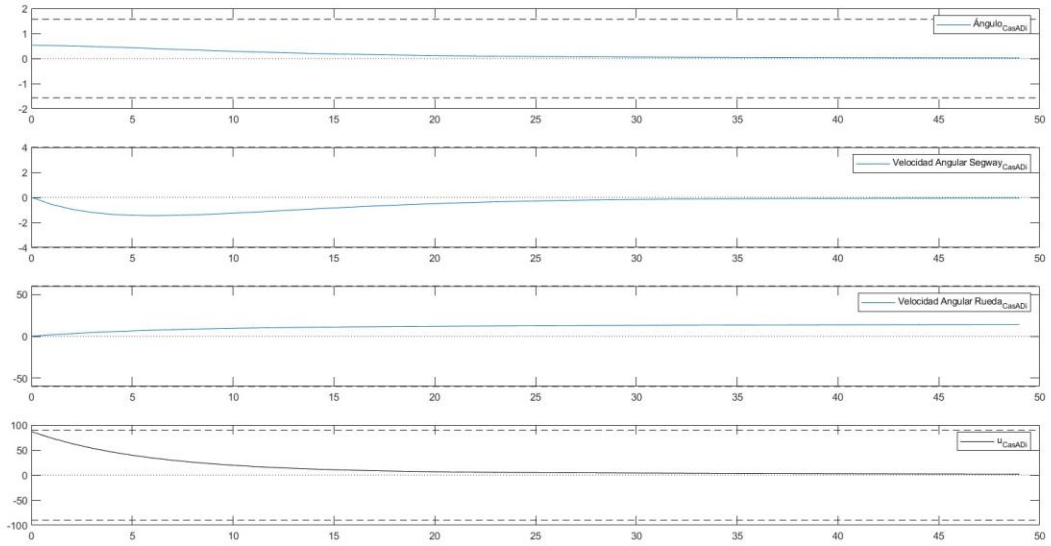

Figura 63. Resultados obtenidos aplicando CasADi con un ángulo inicial de 30º.

En la figura 64, se muestra el desarrollo del Segway en la simulación con un ángulo inicial de -45º.

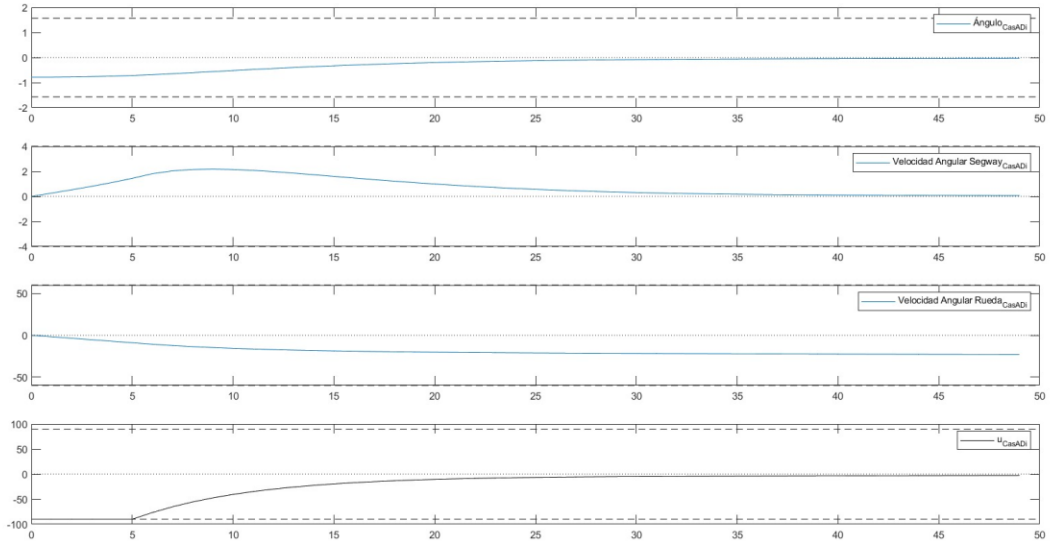

Figura 64. Resultados obtenidos aplicando CasADi con un ángulo inicial de -45º.

Se puede observar cómo sigue de forma correcta la referencia dada. En las siguientes simulaciones se muestra en una sola gráfica como se desarrollan los controladores anteriores con el modelo no lineal.

En la figura 65, se muestra la simulación del LQR, MPC lineal y no lineal para un ángulo inicial de -15º empleando el modelo no lineal del sistema.

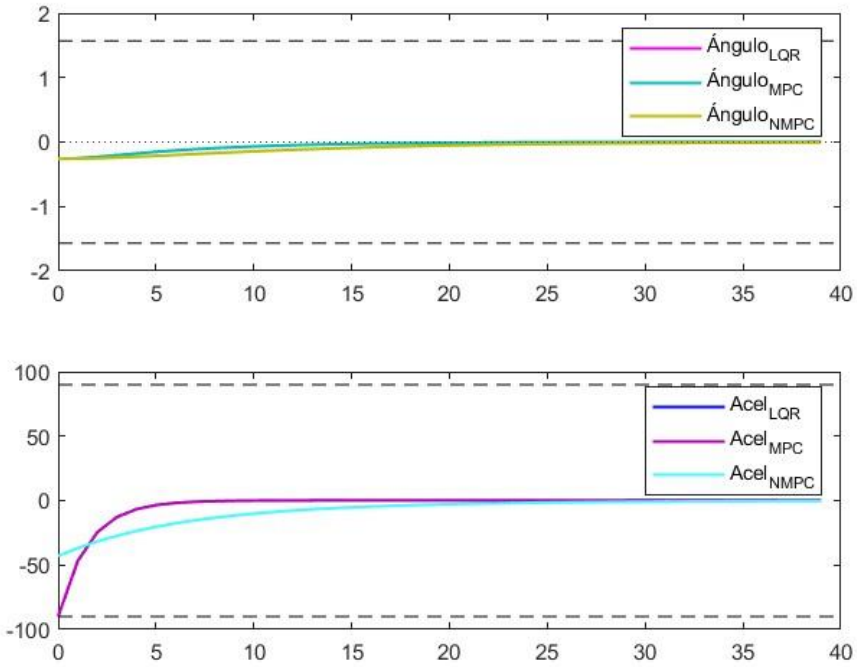

Figura 65. Comparativa entre las simulaciones usando LQR, MPC lineal y MPC no lineal con ángulo inicial de -15º.

En la figura 66, se muestra la simulación del LQR, MPC lineal y no lineal para un ángulo inicial de 30º empleando el modelo no lineal del sistema.

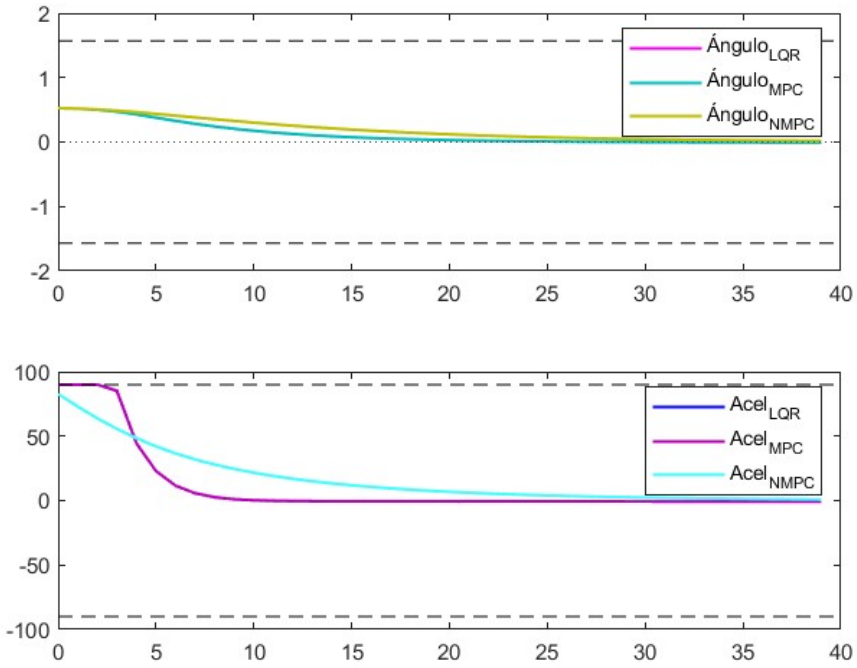

Figura 66. Comparativa entre las simulaciones usando LQR, MPC lineal y MPC no lineal con ángulo inicial de 30º.

En la figura 67, se muestra la simulación del LQR, MPC lineal y no lineal para un ángulo inicial de -45º

empleando el modelo no lineal del sistema.

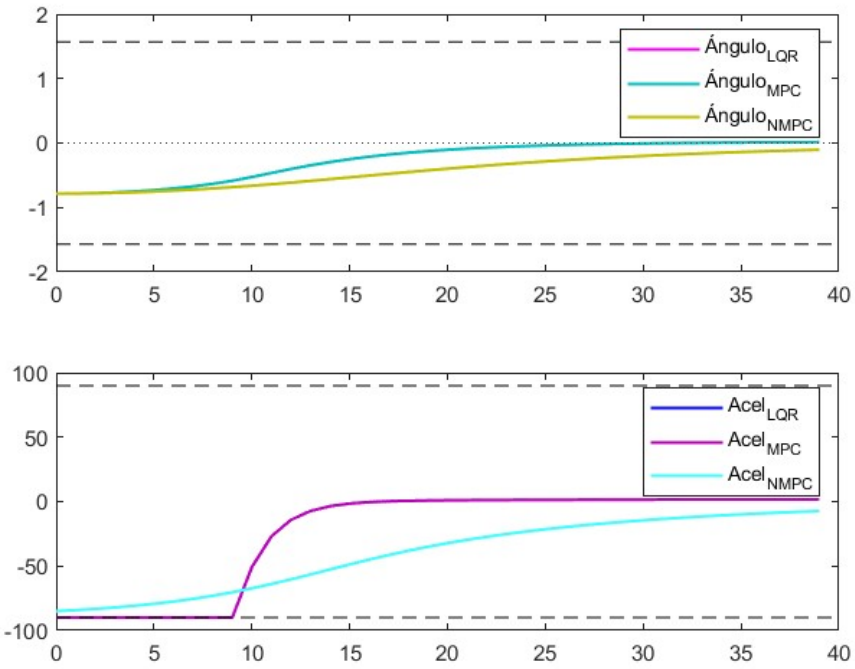

Figura 67. Comparativa entre las simulaciones usando LQR, MPC lineal y MPC no lineal con ángulo inicial de -45º.

Se puede observar que, si bien el MPC no lineal implementado con CasADi tarda más en alcanzar la referencia, también hay que tener en cuenta que la entrada presenta una curva más suave.

7.3.- Resultados obtenidos

Para implementar CasADi en la Raspberry Pi ha sido necesario instalar otro sistema operativo. Esto es debido a que el usuario «*zehuilu*» ha proporcionado una forma de implementar esta herramienta que no funciona con el sistema operativo Raspberry Pi Os. En este caso se ha instalado Ubuntu 20.04 LTS para Raspberry Pi con el parche Preempt_RT para implementar el sistema operativo en tiempo real.

Para agilizar aún más el sistema operativo, se ha prescindido del entorno gráfico. Esto puede realizarse mediante la ejecución de la siguiente instrucción en la terminal de Ubuntu:

```
sudo systemctl set-default multi-user.target

sudo reboot
```

Para implementar CasADi en C++ se ha hecho uso de la librería de uso libre creada por «zehuili». [18]

Primero debe incluirse la librería CasADi.

```
#include <casadi/casadi.hpp>

  using namespace casadi;
```

Por otro lado, se han de definir todos los parámetros del Segway.

```
// Parámetros del Segway
const double l = 0.05; // Longitud del Segway (en m)
const double mr = 0.068; // Masa de la rueda (en kg)
const double masa = 1.228 - mr; // Masa total del Segway (en kg)
const double radio = 0.0495; // Radio de la rueda (en m)
const double = 9.81; // Gravedad (en m/s^2)

// Parámetros para calcular la función no lineal
double a = 2*masa*l*l;
double b = masa*radio*l;
double c = radio*radio*(3*mr+masa);
double d = masa*g*l;

uint8_t N = 50; // number of control intervals


  const double dt = 0.02; // length of a control interval
```

Ahora se declararán las variables de CasADi.

```
casadi::MX f(const casadi::MX& x, const casadi::MX& u) {
      return vertcat(x(1), -(c*u - d*sin(x(0)) - b*sin(x(0))*x(1) +
b*cos(x(0))*u)/(a + b*cos(x(0))), x(2) + dt*u);
}

casadi::Opti opti = casadi::Opti();

casadi::Slice all;

casadi::MX X = opti.variable(3, N + 1);
casadi::MX U = opti.variable(1, N);
casadi::MX T = opti.variable();

casadi::MX k1 = f(X(all,0), U(all,0));
casadi::MX k2 = f(X(all,0)+dt/2*k1, U(all,0));
casadi::MX k3 = f(X(all,0)+dt/2*k2, U(all,0));
casadi::MX k4 = f(X(all,0)+dt*k3, U(all,0));
casadi::MX x_next = X(all,0) + dt/6*(k1+2*k2+2*k3+k4);

casadi::Dict opts_dict = casadi::Dict();

casadi::OptiSol sol = opti.solve();


  casadi::DM u_optimal_e = sol.value(U);
```

Siendo el algoritmo de control:

```
case 4:
        X = opti.variable(3, N + 1);
            U = opti.variable(1, N);
            T = opti.variable();

            opti.minimize(T);

            for (int k = 0; k < N; k++) {
                  k1 = f(X(all,k), U(all,k));
                  k2 = f(X(all,k)+dt/2*k1, U(all,k));
                  k3 = f(X(all,k)+dt/2*k2, U(all,k));
                  k4 = f(X(all,k)+dt*k3, U(all,k));
                  x_next = X(all,k) + dt/6*(k1+2*k2+2*k3+k4);
                  opti.subject_to(X(all,k+1)==x_next);
            }

            opti.subject_to(X(0,0) == ang);
            opti.subject_to(X(1,0) == dang);
            opti.subject_to(X(2,0) == theta);
            opti.subject_to(-3.151592/2.0 <= X(0,all) <= 3.151592/2.0);
            opti.subject_to(-4 <= X(1,all) <= 4);
            opti.subject_to(-60 <= X(2,all) <= 60);
            opti.subject_to(-90 <= U <= 90);

            opti.subject_to(T >= 0);

            opti.set_initial(T, 1);

            opts_dict = casadi::Dict();
            opts_dict["ipopt.print_level"] = 0;
            opts_dict["ipopt.sb"] = "yes";
            opts_dict["print_time"] = 0;

            opti.solver("ipopt",opts_dict);

            sol = opti.solve();

            u_optimal_e = sol.value(U);

            alpha = (double)evalf(u_optimal_e(0,0));

  break;
```

Aplicando la herramienta CasADi sobre el código creado en la Raspberry Pi se han obtenido los siguientes datos.

Para un ángulo de inicio de 0º.

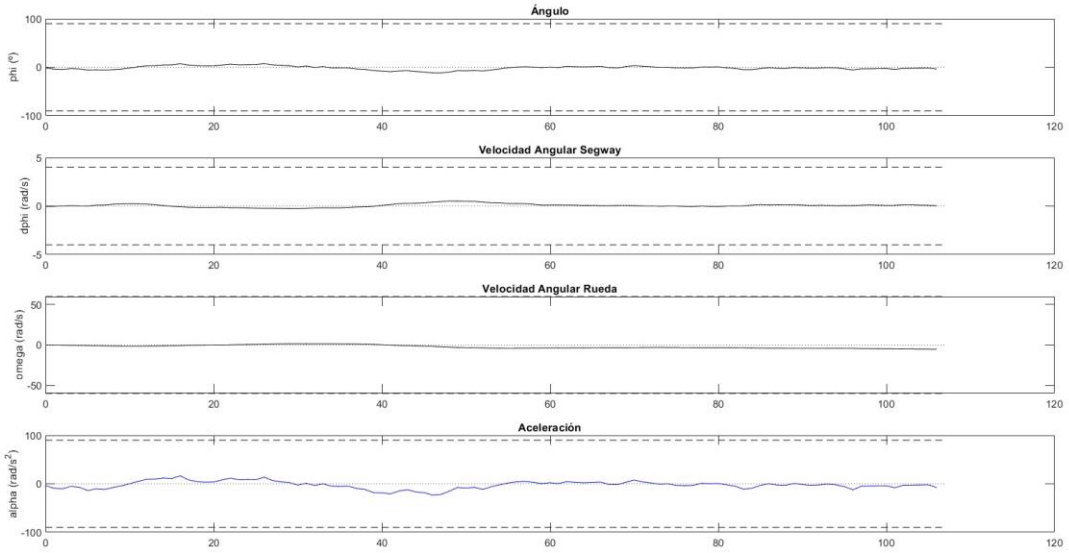

Figura 68. Resultados obtenidos empleando CasADi para un ángulo de 0º.

Para un ángulo de inicio de 15º.

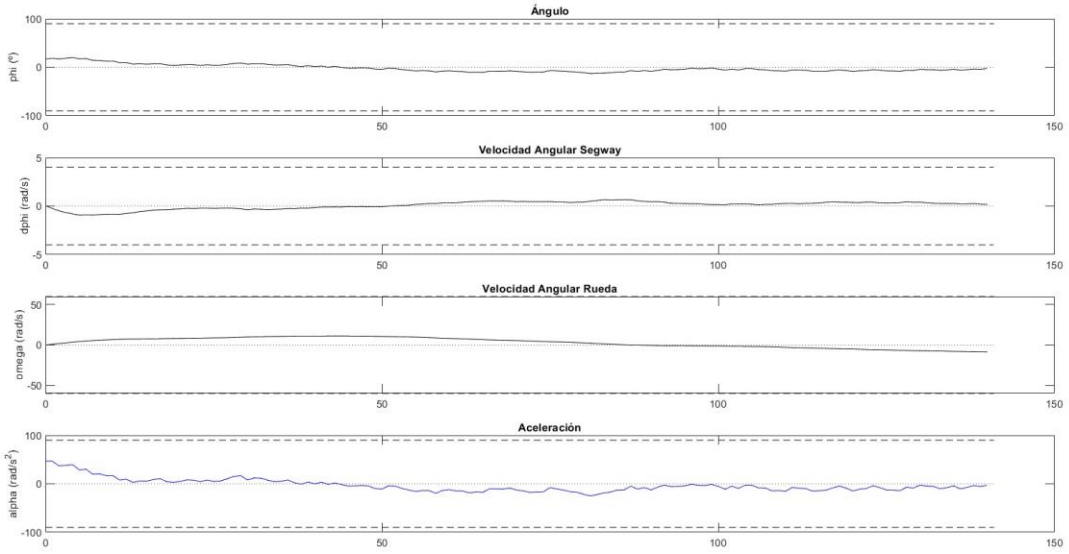

Figura 69. Resultados obtenidos empleando CasADi para un ángulo de 15º.

Para un ángulo de inicio de 30º.

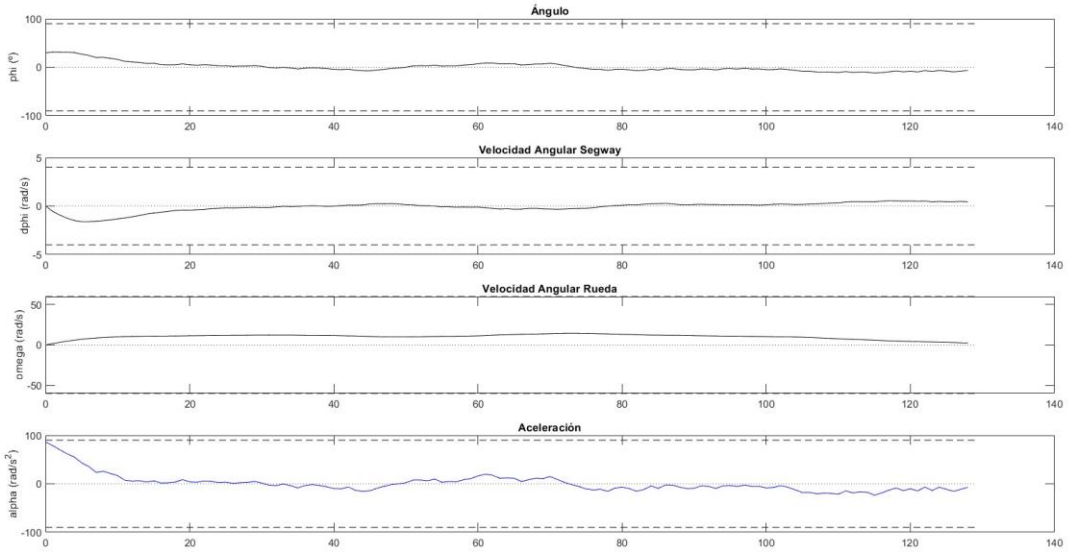

Figura 70. Resultados obtenidos empleando CasADi para un ángulo de 30º.

Para un ángulo de inicio de 45º.

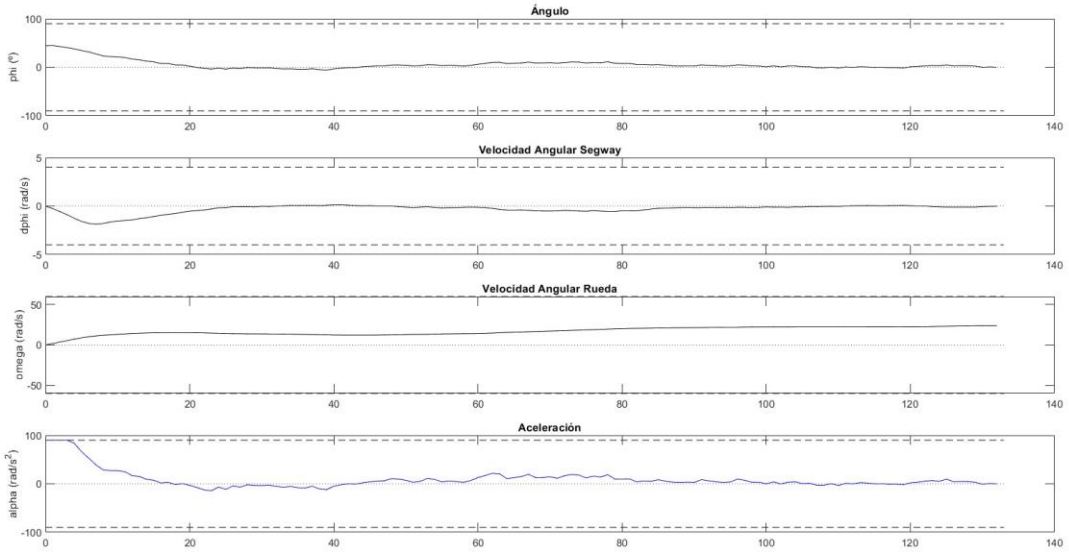

Figura 71. Resultados obtenidos empleando CasADi para un ángulo de 45º.

8.- Comparativa de resultados

A continuación, se mostrarán diferentes gráficas comparando los resultados obtenidos empleando los algoritmos LQR, MPC lineal y MPC no lineal, y además se analizarán los resultados obtenidos con los parámetros obtenidos de la identificación de estados.

En la figura 72, se muestra la comparación de los resultados obtenidos para un ángulo inicial de 15º.

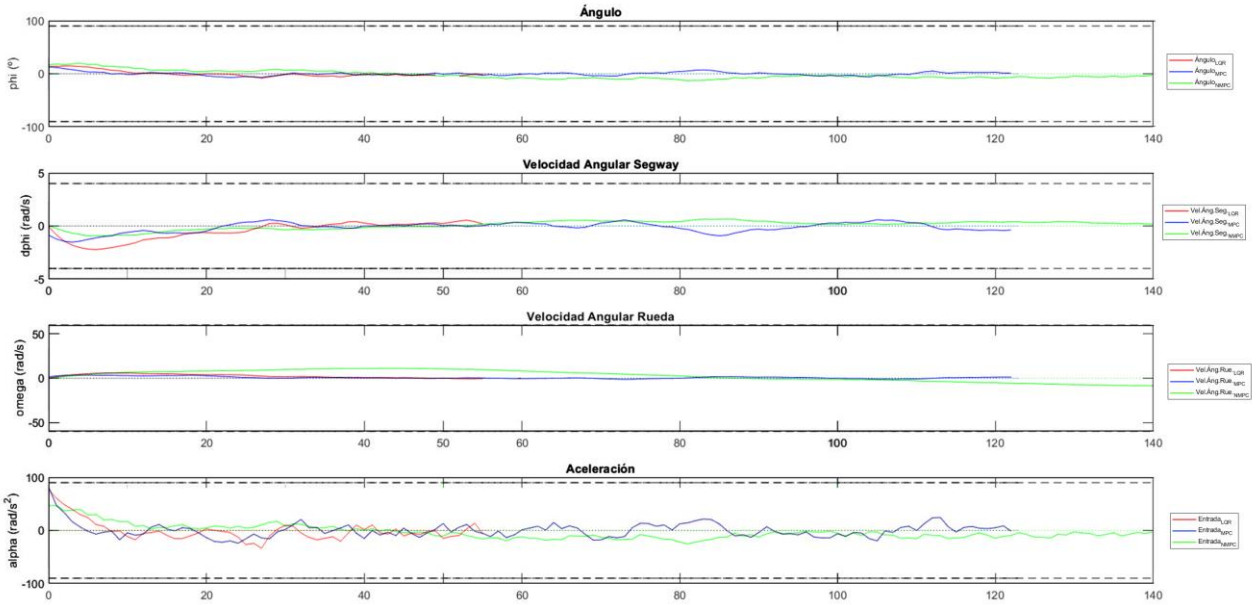

Figura 72. Comparativa de resultados obtenidos con LQR, MPC lineal y NMPC para 15º.

En la figura 73, se muestra la comparación de los resultados obtenidos para un ángulo inicial de 30º.

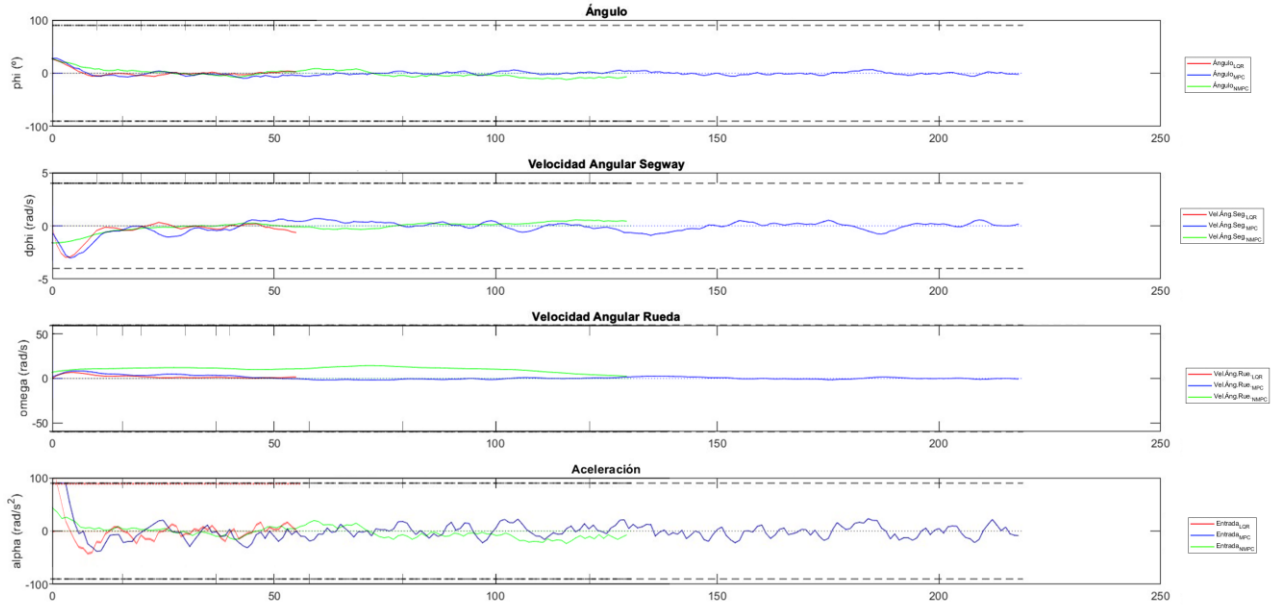

Figura 73. Comparativa de resultados obtenidos con LQR, MPC lineal y NMPC para 30º.

En la figura 74, se muestra la comparación de los resultados obtenidos para un ángulo inicial de 45º.

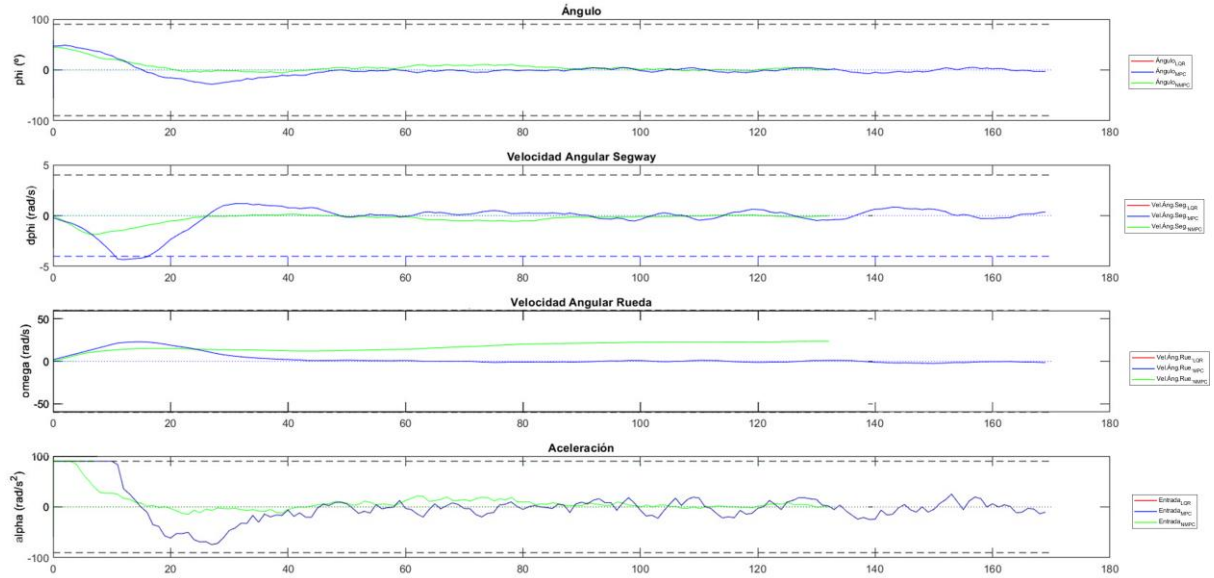

Figura 74. Comparativa de resultados obtenidos con LQR, MPC lineal y NMPC para 45º.

Por otro lado, los resultados obtenidos experimentalmente con la identificación de estados no lograron en la mayoría de los escenarios alcanzar la referencia de forma correcta. Esto probaría que a pesar de que en la simulación no haya habido errores, en la práctica si pueden generarse errores aplicando las matrices obtenidas de la identificación de estados.

9.- Conclusiones

9.1.- Análisis

En este proyecto se han alcanzado varios de los objetivos que se plantearon inicialmente. En primer lugar, se ha logrado programar y ejecutar un código tanto en SPCIES y en CasADi. Los resultados obtenidos con ambas herramientas han sido los esperados, salvo por el hecho de que se esperaba una corrección mejor con CasADi al implementar un control MPC no lineal.

En cuanto a la comparativa anteriormente mencionada, se pueden sacar la conclusión de que en el resultado obtenido en el Segway con la herramienta CasADi, se puede observar que las señales alcanzan la referencia con mayor suavidad, pero a su vez, el tiempo que tarda en alcanzar la referencia aumenta. Esto puede ser útil en aplicaciones en las que se le de prioridad a la suavidad en contra de la rapidez de respuesta.

Por otro lado, la identificación de estados ha conseguido obtener unas matrices del espacio de estados que consiguen alcanzar también la referencia nula en la simulación, sin embargo, en la práctica lo ha logrado en pocas ocasiones.

Esto puede ser debido a múltiples factores que no se han tenido en cuenta a la hora de realizar la simulación como podrían ser, en primer lugar, que las matrices obtenidas son una aproximación del modelo lineal del sistema y a pesar de que con el modelo lineal también se han conseguido buenos resultados, con una aproximación no lo suficientemente rigurosa estos resultados no puedan alcanzarse. Otro motivo podría ser la acumulación de errores en las medidas tomadas para realizar la identificación de estados. Al haberse realizado mediante varias muestras, estas muestras han ido acumulando el error de las mediciones.

9.2.- Posibles ampliaciones

Dentro de las posibles ampliaciones que se pueden realizar se encuentran:

- Desarrollar e implementar un método que pueda ejecutar la herramienta CasADi en el sistema operativo Raspberry Pi OS.
- Comprobar la diferencia de tiempo de cómputo entre una ejecución de SPCIES aplicada en Python y en C++, y de la misma forma con CasADi.
- Implementar un módulo Bluetooth en el Segway para poder controlarlo remotamente.

10.- Bibliografía

[1] Villarroel Salcedo, J. L. (2014). *Sistemas de Tiempo Real.*

[2] Gene F. Franklin, J. David Powell, Abbas Emami-Naeini. (2013). *Feedback Control of Dynamic Systems*.

[3] Hernández, G., Bordons Alba, C., Marcos, D., & Montero, C. (2015). Control de estabilidad basado en MPC para un vehículo eléctrico con motores en rueda. *Jornadas de automática (2015), p 887-894*

[4] Lehtomaki, N.; Sandell, N.; Athans, M. (1981). "*La robustez da como resultado diseños de control multivariable basados en gaussiano lineal-cuadrático*" . Transacciones IEEE sobre control automático. 26 (1): 75–93.

[5] Medina, M. A. P. L., Saba, M. G. H., de Guevara Durán, M. E. L., & Silva, M. J. H. (2011). *Controladores PID y controladores difusos*. Revista de ingeniería industrial, 5(1).

[6] An excellent overview of the state of the art (in 2008) is given in the proceedings of the two large international workshops on NMPC, by Zheng and Allgower (2000) and by Findeisen, Allgöwer, and Biegler (2006).

[7] BIEGLER, L. (1984). Solution of Dynamic Optimizatíon Problems by Successive Quadratic Programming and Orthogonal Collocation. *Comput. chem. Engng.*, 243-248.

[8] Cámara Molina, J., Alvarado Aldea, I., & Krupa García, P. (2020). *Diseño, realización y control de un robot autoequilibrado de bajo coste basado en una Raspberry Pi.*

[9] Gerum, P. (2004). *Xenomai-Implementing a RTOS emulation framework on GNU/Linux*. White Paper, Xenomai, 81.

[10] Reghenzani, F., Massari, G., & Fornaciari, W. (2019). *The real-time linux kernel: A survey on preempt_rt.* ACM Computing Surveys (CSUR), 52(1), 1-36.

[11] Salcedo Tovar, M. L. (2015). *Minicomputador educacional de bajo costo raspberry pi: primera parte.* Revista Ethos Venezolana, *7*(1), 28-45.

[12] Abrajan Arias, C. O. (2020). *Diseño y construcción de un robot seguidor de línea evasor de obstáculos empleando Arduino Nano.*

[13] Carnuccio, E., Valiente, W., Volker, M., De Luca, G., García, G., Giulianelli, D. A., & Barillaro, S. (2017). *Desarrollo de un Prototipo detector de caídas utilizando la placa Intel Galileo Generación I y el sensor MPU6050.* In XXIII Congreso Argentino de Ciencias de la Computación (La Plata, 2017)

[14] Texas Instruments. (2019). *DRV8825: Stepper Motor Controller IC*. Texas Instruments. https://www.ti.com/lit/ds/symlink/drv8825.pdf.

[15] Dawoud, D. S., & Dawoud, P. (2022). *Serial Communication Protocols and Standards*. CRC Press.

[16] Valdéz Blanco, D. (2010). Regresión por mínimos cuadrados parciales. *Revista Varianza, 18.*

[17] J. Arrieta, R. Ferreira, R. Pardo y A. Rodríguez-Bernal. "*Análisis Numérico de Ecuaciones Diferenciales Ordinarias*". Paraninfo, Madrid, 2020. ISBN 9788428344418, ISBN 8428344418

[18] https://github.com/zehuilu/Tutorial-on-CasADi-with-CPP

11.- Anexos

11.1.- Código para identificación del sistema

El código empleado en la identificación de sistemas para estimar el valor de las matrices de estado A y B es:

```
%% Descripción
% Con este código se pretende obtener las matrices A y B de nuestro espacio de
estados
% Para ello se hará uso del método de mínimos cuadrados (regresión lineal)
% Se usarán una serie de muestras del ángulo obtenido a una entrada constante

%% Referencia: http://paginapessoal.utfpr.edu.br/avargas/courses-
1/identificacao_sistemas/identificacao_sistemas/1_materialApoioStateSpace.pdf

%% Inicialización
clear all, clc, cla, hold off, close all

%% Toma de datos
% Muestras del ángulo obtenido
mue = [
        % Implementar las muestras obtenidas
      ]';

%% Cálculo de las matrices A y B
% Es necesario calcular la siguiente expresión: Yi = U · Bi
% Donde:
% Yi = [x(i,k+1) , x(i,k+2) , ... , x(i,k+N)]'      *Donde N es el número de
muestras obtenidas
%
% U = [ x(1,k)   ,  x(2,k)   ,  u(k)   ;
%       x(1,k+1) , x(2,k+1) , u(k+1) ;
%         ...    ,   ...    ,  ...    ;
%      x(1,k+N-1),x(2,k+N-1),u(k+N-1)]
%
% Bi = [A(i,1) , A(i,2) , B(i,1)]

% Declaración de la matriz de variables de estado
xp = mue(1:3,:);
N = length(xp); % Número de muestras

% Declaración de la matriz de variables de entrada
%u = -0.5*ones(1,N); % Entrada constante de 10
u = mue(4,:);

% Declaración de la matriz Yi anteriormente referenciada (para i = 1 | i = 2)
Yi1 = xp(1,2:N);
Yi2 = xp(2,2:N);
Yi3 = xp(3,2:N);
```

```
% Declaración de dos matrices auxiliares iguales que x , u pero sin tomar el
valor final
xi = xp(:,1:N-1);
ui = u(1:N-1)';

% Declaración de la matriz U
U = [xi',ui];

% Declaración de la matriz Bi
Bi = inv(U'*U)*U';

Bi1 = Bi*Yi1'; % Parte correspondiente a angular (x1)
Bi2 = Bi*Yi2'; % Parte correspondiente a velocidad angular (x2)
Bi3 = Bi*Yi3'; % Parte correspondiente a omega (x3)

% Cálculo de las matrices A y B
A = [Bi1(1:3,:)';Bi2(1:3,:)';Bi3(1:3,:)']
B = [Bi1(4,:);Bi2(4,:);Bi3(4,:)]

% Declaración de una matriz de espacios de estados para comparar el valor
obtenido con las muestras
x = zeros(3,N);
x(:,1) = xp(:,1);

%% Real
% Parámetros del Segway
L = 0.05; % Longitud del Segway (en m)
mr = 0.068; % Masa de la rueda (en kg)
Masa = 1.228 - mr; % Masa total del Segway (en kg)
Radio = 0.0495; % Radio de la rueda (en m)
g = 9.81; % Gravedad (en m/s^2)

a = 2*Masa*L^2 + Masa*Radio*L;
b = (3*mr + Masa)*Radio^2 + Masa*Radio*L;
c = Masa*g*L;

% Matrices del sistema
Ac = [ 0   , 1 , 0;
     c/a , 0 , 0;
      0   , 0 , 0];

Bc = [ 0 ;
     -b/a;
       1 ];

Cc =[1 0 0];

Dc = 0;

Mc = ss(Ac,Bc,Cc,Dc);
Md = c2d(Mc,0.02);
[Ad,Bd,Cd,Dd] = ssdata(Md);

%% Iteración para calcular x estimada
for k = 1:1:N-1
    x(:,k+1) = A*x(:,k) + B*u(k);
end
```

```
%% Mostrar en gráficas
t = 1:1:N;

figure(1);
hold on
plot(t,x(1,1:N),'m');

figure(2);
hold on
plot(t,x(2,1:N),'m');

figure(3);
hold on

plot(t,x(3,1:N),'m');
```

11.2.- Código para simulación usando LQR

El código empleado en la simulación para el control LQR es:

```
%% Descripción
% Implementación de un algoritmo de control LQR al espacio de estados

%% Inicialización
clear all, clc, cla, hold off, close all

%% Parámetros del Segway
L = 0.05; % Longitud del Segway (en m)
mr = 0.068; % Masa de la rueda (en kg)
Masa = 1.228 - mr; % Masa total del Segway (en kg)
Radio = 0.0495; % Radio de la rueda (en m)
g = 9.81; % Gravedad (en m/s^2)

a = 2*Masa*L^2 + Masa*Radio*L;
b = (3*mr + Masa)*Radio^2 + Masa*Radio*L;
c = Masa*g*L;

%% Matrices del sistema linealiado
A = [ 0   , 1 , 0;
     c/a , 0 , 0;
      0   , 0 , 0];

B = [  0 ;
     -b/a;
       1 ];

C =[1 0 0];

D = 0;

Mc = ss(A,B,C,D);
Md = c2d(Mc,0.02);
[Ad,Bd,Cd,Dd] = ssdata(Md);
```

```
%% Matrices del sistema de la identificación de sistemas
A_Ident = [0.1240   -0.0188   -0.0096
           1.0779    0.8323    0.0481
           0.1651   -0.2437    0.9877];

B_Ident = [-0.0004
           -0.0019
            0.0113];

%% Obtención del vector de realimentación K
Q = [10 0 0 ; 0 1 0 ; 0 0 10]; % Peso de las variables de estado
R = 0.1; % Peso de la variable de entrada
ref = 0; % Referencia

[K,S,P] = dlqr(Ad,Bd,Q,R),

%% Iteración para calcular x estimada
tFinal = 40; % Numero de muestras
x = zeros(3,tFinal); % Variables de estado
u = zeros(1,tFinal); % Variable de entrada
y = zeros(1,tFinal); % Variable de salida

x(:,1) = [-45;0;0]*pi/180; % Valor inicial

xIdent = x; % Variable de estados para las matrices obtenidas por identificación
de sistemas

%% Parámetros para calcular la función no lineal
yNL = [];
xNL = x(:,1)'; % Ángulo obtenido con el modelo no lineal

syms ang(t)
dang = diff(ang); % dang/dt
d2ang = diff(ang,2); % d2ang/dt2

% Parámetros de la función no lineal
a = 2*Masa*L^2;
b = Masa*Radio*L;
c = Radio^2*(3*mr+Masa);
d = Masa*g*L;

%% Bucle
for k = 1:1:tFinal
    %x(1,k) = restric(x(1,k),-pi/2,pi/2); % Restringir el ángulo
    %x(2,k) = restric(x(2,k),-4,4); % Restringir la velocidad angular

    u(:,k) = -K*x(:,k); % Calculo de la entrada

    %u(1,k) = restric(u(1,k),-90,90); % Restringir la entrada

    y(:,k) = Cd*x(:,k) + Dd*u(:,k); % Calculo de la salida
    x(:,k+1) = Ad*x(:,k) + Bd*u(:,k); % Calculo del siguiente valor de angulo y
velocidad angular

    xIdent(:,k+1) = A_Ident*xIdent(:,k) + B_Ident*u(:,k);
```

```
ode1 = (a + b*cos(ang))*d2ang + (c + b*cos(ang))*u(:,k) - b*dang*sin(ang) -
d*sin(ang) == 0; % Función no lineal que describe el comportamiento del Segway

    % Calculo del angulo obtenido a traves de la entrada (modelo no lineal)
    [VF,Sbs] = odeToVectorField(ode1);
    odsefcn = matlabFunction(VF,'vars',{'t','Y'});
    [tNL,xNL] = ode45(odsefcn,[(k-1)*0.02,(k)*0.02],[xNL(end,1),xNL(end,2)]);

    for in = 1:1:length(tNL)
        xNL(in,1) = restric(xNL(in,1),-pi/2,pi/2); % Restringir el ángulo
        xNL(in,2) = restric(xNL(in,2),-4,4); % Restringir la velocidad angular
    end

    %hold on
    %plot(tNL/0.02,yNL(:,1),'k','LineWidth',1.5); % Graficar los valores
comprendidos entre [0.02 * (k-1) , 0.02 * k]

    yNL = [yNL;[tNL,xNL]];
end
yNL = yNL';

%% Mostrar en gráficas
t = 0:1:tFinal-1;
figure(1);
hold on
subplot(2,1,1);
plot(t,y,'m');
hold on
plot(t,xIdent(1,1:end-1),'Color',[0,0.75,0.75]);
plot(yNL(1,:)./0.02,yNL(2,:),'Color',[0.75,0.25,0.25]);
line([0,tFinal],[-pi/2,-pi/2],'Color',[0,0,0],'LineStyle','--');
line([0,tFinal],[pi/2,pi/2],'Color',[0,0,0],'LineStyle','--');
line([0,tFinal],[ref,ref],'Color',[0.25,0.25,0.25],'LineStyle',':');
legend('Ángulo_{M. Lineal}','Ángulo_{Identificación}','Ángulo_{M. No
Lineal}','','','');
subplot(2,1,2);
plot(t,u,'b');
hold on
line([0,tFinal],[-90,-90],'Color',[0,0,0],'LineStyle','--');
line([0,tFinal],[90,90],'Color',[0,0,0],'LineStyle','--');
legend('Entrada');
hold off

%% Funcion creada para acotar las variables del sistema
function salida = restric(valor,valorMin,valorMax)
    if valor > valorMax
        valor = valorMax;
    elseif valor < valorMin
        valor = valorMin;
    end
    salida = valor;

end
```

11.3.- Código para simulación usando MPC con SPCIES

El código empleado en Matlab para simular MPC con SPCIES es:

```
%% Descripción
% Implementación del MPC lineal al espacio de estados

%% Inicialización
clear, clc,

%% Parámetros del Segway
L = 0.05; % Longitud del Segway (en m)
mr = 0.068; % Masa de la rueda (en kg)
Masa = 1.228 - mr; % Masa total del Segway (en kg)
Radio = 0.0495; % Radio de la rueda (en m)
g = 9.81; % Gravedad (en m/s^2)

a = 2*Masa*L^2 + Masa*Radio*L;
b = (3*mr + Masa)*Radio^2 + Masa*Radio*L;
c = Masa*g*L;

%% Matrices del sistema
A = [ 0   , 1 , 0;
     c/a , 0 , 0;
      0   , 0 , 0];

B = [  0 ;
     -b/a;
       1 ];

C =[1 0 0];

D = 0;

Mc = ss(A,B,C,D);
Md = c2d(Mc,0.02);
[Ad,Bd,Cd,Dd] = ssdata(Md);

% Longitud de las matrices del espacio de estados
n = size(Ad, 1);
m = size(Bd, 2);

% Restricciones del sistema
LBx = -[pi/2; 4; 60];
UBx =  [pi/2; 4; 60];
LBu = -[90];
UBu =  [90];

% Descripción del sistema
sys = struct('A', Ad, 'B', Bd, 'LBx', LBx, 'UBx', UBx, 'LBu', LBu, 'UBu',
UBu);
```

```
%% Diseño del controlador MPC
% Valores para calcular la matriz de Riccati

Q = [10 0 0 ; 0 1 0 ; 0 0 10]; % Peso de las variables de estado
R = 0.1; % Peso de la variable de entrada
ref = 0; % Referencia
[~, T] = dlqr(sys.A, sys.B, Q, R); % Calculo de T
N = 3; % Horizonte de predicción (debe ser similar al número de variables de
estado)

% Los parámetros anteriores deben ser guardados en una estructura
param = struct('Q', Q, 'R', R, 'T', T, 'N', N);

%% Generar el solucionador
solver_options.rho = 15; % Valor del parámetro de penalización
solver_options.k_max = 1000; % Número máximo de iteraciones
solver_options.tol = 1e-3; % Tolerancia

% Guardar el archivo que contiene el MPC
options.save_name = 'mpc_solver';
options.directory = '';
options.time = true; % Contar el tiempo internamente

spcies_clear;

% Generar el archivo MEX
spcies_gen_controller('sys', sys, 'param', param, 'solver_options',
solver_options,...
    'options', options, 'platform', 'Matlab', 'type', 'laxMPC');

%% Usar el archivo MEX
% Condiciones de la simulación
num_iter = 50; % Número máximo de iteraciones
x0 = [20;0;0]*pi/180; % Valor inicial

ur = 0*ones(m, 1); % Referencia para u
xr = 0*ones(n, 1); % Referencia para x

% Variables para almacenar los valores obtenidos durante el bucle
hX = zeros(n, num_iter); % Evolución de las variables de estado
hU = zeros(m, num_iter); % Señal de control aplicada
hT = zeros(1, num_iter); % Tiempo de cómputo
hK = zeros(1, num_iter); % Número de iteraciones del solucionador
hE = zeros(1, num_iter); % Bandera de salida

% Inicio de la simulación
x = x0; % Establecer el valor inicial
```

```
%% Bucle
for i = 1:num_iter
    % Ejecutar la función MPC
    [u, hK(i), hE(i), info] = mpc_solver(x, xr, ur);
    hT(i) = info.run_time;

    % Simular el sistema con las matrices A y B
    x = sys.A*x + sys.B*u;

    % Guardar los valores de 'x' y 'u'
    hX(:, i) = x;
    hU(:, i) = u;
end

%% Mostrar los resultados
figure(1); clf(1);

% Variables de estado
subplot(2, 1, 1);
object_num = 1;
plot(0:num_iter, [x0(object_num) hX(object_num,:)], 'b');
hold on;
plot(0:num_iter, xr(object_num)*ones(1, num_iter+1), 'r:');
line([0,num_iter],[-pi/2,-pi/2],'Color',[0,0,0],'LineStyle','--');
line([0,num_iter],[pi/2,pi/2],'Color',[0,0,0],'LineStyle','--');
xlabel('Sample time');
ylabel(['Position of object ' num2str(object_num)]);
grid on;

% Entrada
subplot(2, 1, 2);
force_num = 1;
plot(0:num_iter-1, hU(force_num, :), 'b');
hold on;
plot(0:num_iter-1, ur(force_num)*ones(1, num_iter), 'r:');
line([0,num_iter],[-90,-90],'Color',[0,0,0],'LineStyle','--');
line([0,num_iter],[90,90],'Color',[0,0,0],'LineStyle','--');
xlabel('Sample time');
ylabel(['Control input #' num2str(force_num)]);
grid on;
```

11.4.- Código para simulación usando MPC con CasADi

El código empleado en Matlab para realizar las simulaciones con CasADi es el siguiente:

```
%% Descripción
% Implementación de un algoritmo de control MPC no lineal usando CasADi al
espacio de estados
clear all, clc,

%% Parámetros del Segway
L = 0.05; % Longitud del Segway (en m)
mr = 0.068; % Masa de la rueda (en kg)
Masa = 1.228 - mr; % Masa total del Segway (en kg)
Radio = 0.0495; % Radio de la rueda (en m)
g = 9.81; % Gravedad (en m/s^2)

a = 2*Masa*L^2 + Masa*Radio*L;
b = (3*mr + Masa)*Radio^2 + Masa*Radio*L;
c = Masa*g*L;

%% Matrices del sistema
A = [ 0   , 1 , 0;
     c/a , 0 , 0;
      0   , 0 , 0];

B = [  0 ;
     -b/a;
       1 ];

C =[1 0 0];

D = 0;

dt = 0.02; % Muestreo

Mc = ss(A,B,C,D);
Md = c2d(Mc,0.02);

[Ad,Bd,Cd,Dd] = ssdata(Md);

%% Parámetros para calcular la función no lineal
a = 2*Masa*L^2;
b = Masa*Radio*L;
c = Radio^2*(3*mr+Masa);
d = Masa*g*L;

%% Importar la librería que incluye CasADi
import casadi.*

N = 25; % Horizonte de predicción

opti = casadi.Opti(); % Problema de optimización

tFinal = 50; % Número de iteracciones
```

```
%% Variables del espacio de estados
xNMPC = zeros(3,tFinal);
yNMPC = zeros(1,tFinal);
uNMPC = zeros(1,tFinal);

xNMPC(:,1) = [-15 ; 0 ; 0].*pi/180; % Valor inicial de X

%% Bucle
for kk = 1:1:tFinal
    % Variables de decisión
    X = opti.variable(3,N+1); % Variables de estado
    U = opti.variable(1,N);   % Variable de control

    % Función objetivo
    opti.minimize(sumsqr(X) + sumsqr(U));

    % Restricciones dinámicas
    f = @(x,u) [x(2);
                -(c*u - d*sin(x(1)) - b*sin(x(1))*x(2) + b*cos(x(1))*u)/(a +
b*cos(x(1)));
                x(3) + dt*u]; % dx/dt = f(x,u)

    for k=1:N % Obtención del estado siguiente mediante Runge Kutta
        k1 = f(X(:,k),         U(:,k));
        k2 = f(X(:,k)+dt/2*k1, U(:,k));
        k3 = f(X(:,k)+dt/2*k2, U(:,k));
        k4 = f(X(:,k)+dt*k3,   U(:,k));
        x_next = X(:,k) + dt/6*(k1+2*k2+2*k3+k4);
        opti.subject_to(X(:,k+1)==x_next); % Restricción que define el
comportamiento del Segway
    end

    % Restricciones de valor inicial
    opti.subject_to(X(1,1) == xNMPC(1,kk)); % phi
    opti.subject_to(X(2,1) == xNMPC(2,kk)); % dphi
    opti.subject_to(X(3,1) == xNMPC(3,kk)); % omega

    % Restricciones de valores máximos y mínimos
    opti.subject_to(-pi/2<=X(1,:)<=pi/2); % phi
    opti.subject_to(-4<=X(2,:)<=4); % dphi
    opti.subject_to(-60<=X(3,:)<=60); % dphi
    opti.subject_to(-90<=U<=90); % u (alpha)

    % Llamar al solver
    opti.solver('ipopt');
    sol = opti.solve();

    % Procesar el valor de 'u' obtenido
    u_optimal = sol.value(U);
    uNMPC(:,kk) = u_optimal(1);
    xNMPC(:,kk+1) = Ad*xNMPC(:,kk) + Bd*uNMPC(:,kk);
    yNMPC(:,kk) = Cd*xNMPC(:,kk) + Dd*uNMPC(:,kk);

  end
```

```
%% Muestrear los resultados
figure
subplot(2,1,1);
plot(xNMPC(1,1:end-1));
grid on
hold on
%plot(xNMPC(2,1:end-1));
plot([0,tFinal],[0,0],'LineStyle',':','Color',[0,0,0]);
line([0,tFinal],[-pi/2,-pi/2],'Color',[0,0,0],'LineStyle','--');
line([0,tFinal],[pi/2,pi/2],'Color',[0,0,0],'LineStyle','--');
legend('Ang_{CasADi}','','','');
subplot(2,1,2);
plot(uNMPC(1,:),'k');
grid on
legend('u_{CasADi}');
```

11.5.- Código de la Raspberry Pi para implementar LQR

```
//DECLARACIÓN DE CABECERAS
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <iostream>
#include <wiringPi.h>
#include <wiringSerial.h>
#include <chrono>
#include <math.h>

using namespace std::chrono;

//Para implementar el control LQRI se varía la cuarta componente del vector K a
0.3
float offset = 3.5;
float K[] = {-344.0464,-42.5310,-7.6876,0.0};

//FUNCIONES
float calculo_lqr(float A, float B, float gir);
float filtro(float accel, float gxy);

//CAMBIO DE RADIANES A GRADOS
#define DEG_TO_RAD 0.01745329252
float PI = 3.1415;
int cont;

//VARIABLES PARA COMUNICAR CON ARDUINO
float omega, omega_r;
uint16_t omegaByte, omega_rByte;
float alpha;
float aceleracion;
int fd;
```

```
int gy;
float gyRad, ang_x, ang_y, ang_z;
float gy_escalado, ax_escalado, ay_escalado, az_escalado;

#define A_OFF_X 0.0
#define A_OFF_Y 0.0
#define A_OFF_Z 0.0

#define ACCEL_SENS 835.1

#define G_OFF_X -3.55
#define G_OFF_Y   0.65
#define G_OFF_Z 3.7

#define GYRO_SENS 131.0

float angulo = 0.0;
float accel_ang_y;

float giros_ang_y;
float giros_ang_y_prev;
float angAnterior = 0.0;

float TAU = 0.98;

float dt;
long tiempo_prev;

uint16_t gyByte;

#pragma pack(push, 1)
struct miEstructura {
    uint16_t angByte_x = 0;
    uint16_t angByte_y = 0;
    uint16_t angByte_z = 0;
};
#pragma pack(pop)

//VARIABLES PARA CONTROL LQR
float ivr = 0, u;
float dtheta_r, phi,dphi,dtheta;

//VARIABLES PARA TEMPORIZADOR Y SENAL
timer_t temporizador; //declaración del nombre del temporizador
struct itimerspec periodo; //estructura para indicar cada cuanto tiempo se lanza
el temporizador
struct timespec inicial;
struct timespec tiempo; //estructura para definirla duración del temporizador
struct sigevent aviso; //establezco una senal que avise que se ha cumplido el
temporizador
sigset_t sign; //conjunto de señales

struct sigaction act;
```

```
//VARIABLE EJECUTIVO CICLICO
int ciclo=0;
double dtt;
clock_t iniciall;
int flag=0;

int uint2int(uint16_t valor) {
      int salida = (int)valor;
      if (salida > 32768) {
            salida -= 65536;
      }
      return salida;
}

float uint2float(uint16_t valor) {
      float salida = (float)valor;
      salida -= 32768;
      salida = salida/100;
      return salida;
}

auto start = 0;
auto stop = 0;
auto duration = 0;

static bool yaEjecutado = false;
char vaciar;

const float beta = 0.15;

unsigned char buffer[sizeof(miEstructura)];
miEstructura datos;

//FUNCIÓN MANEJADOR
void Manejador(int signo, siginfo_t *info, void*context) {
    ciclo++;
    //CADA VEZ QUE SE ACTIVA LA SENAL, CICLO AUMENTA EN UNA UNIDAD Y HACE QUE LE
CORRESPONDA EN EL SWITCH-CASE
    switch(ciclo) {
        case 1:
            for (int i = 0; i < sizeof(miEstructura); i++) {
                buffer[i] = serialGetchar(fd);
            }
            memcpy(&datos, buffer, sizeof(miEstructura));

            ang_x = uint2int(datos.angByte_x);
            ang_y = uint2int(datos.angByte_y);
            ang_z = uint2int(datos.angByte_z);

            ax_escalado = round((ang_x - A_OFF_X)*1000.0 / ACCEL_SENS)/1000.0;
            ay_escalado = round((ang_y - A_OFF_Y)*1000.0 / ACCEL_SENS)/1000.0;
            az_escalado = round((ang_z - A_OFF_Z)*1000.0 / ACCEL_SENS)/1000.0;
        break;
```

```
case 2:
                    for (int i = 0; i < sizeof(uint16_t); i++) {
                buffer[i] = serialGetchar(fd);
            }
            memcpy(&gyByte, buffer, sizeof(uint16_t));

            gy = uint2int(gyByte);
            gy_escalado = round((gy - G_OFF_Y)*1000.0 / GYRO_SENS)/1000.0;

            accel_ang_y = -atan2(ax_escalado, (sqrt((az_escalado*az_escalado) +
(ay_escalado*ay_escalado))))*(180/PI);
            giros_ang_y = filtro(accel_ang_y,gy_escalado);

            std::cout << phi/DEG_TO_RAD <<  "  " << dphi/DEG_TO_RAD << "  " <<
omega << "  " << alpha << std::endl;
            std::cout << serialDataAvail(fd) << std::endl;
        break;

        case 3:
                    for (int i = 0; i < sizeof(uint16_t); i++) {
                buffer[i] = serialGetchar(fd);
            }
            memcpy(&omegaByte, buffer, sizeof(uint16_t));

            omega = uint2float(omegaByte);

            //std::cout << "omega: " << omega << std::endl;
        break;

        case 4:
            alpha = calculo_lqr(omega, 0, giros_ang_y);
            if (alpha < -90) {
                        alpha = -90;
                  } else if (alpha > 90) {
                        alpha = 90;
                  }
            //aceleracion += alpha;
            //std::cout << alpha << std::endl;
        break;

        case 5:
            for (int i = 0; i < 4; i++) {
               serialPutchar(fd, *((char *)&alpha + i));
            }
            ciclo = 0;
        break;
    }

  }
```

```
//FUNCIÓN PRINCIPAL MAIN
int main(void) {
    //CODIGO DE CONEXION CON ARDUINO
    // Abrir comunicacion serial
    fd = serialOpen("/dev/ttyUSB0", 115200);

    if (fd < 0) {
      std::cerr << "Error al abrir puerto serie" << std::endl;
      return 1;
    }

    if (!yaEjecutado) {
            while (serialDataAvail(fd) > 0) {
                  vaciar = serialGetchar(fd);
            }
            yaEjecutado = true;
      }
    //TRATAMIENTO DE SENALES
    sigemptyset(&sign); //crea una mascara vacía
    sigaddset(&sign, SIGUSR1); //anade la senal SIGALARM al conjunto
    pthread_sigmask(SIG_UNBLOCK,&sign, NULL); //bloqueo la senal para el proceso
    /* programo la senar SIGVTALRM para que salte la funcion manejador
    cuando se cumpla el temporizador */
    act.sa_sigaction=Manejador;
    sigemptyset(&(act.sa_mask));
    act.sa_flags=SA_SIGINFO;
    sigaction(SIGUSR1, &act, NULL);
    //configuro el temporizador
    inicial.tv_sec=1;
    tiempo.tv_nsec=4000000; //cada 4 ms salta el manejador
    periodo.it_value=inicial; //indico que el primer valor lo lance 1 segundo
despues de llamarlo
    periodo.it_interval=tiempo; //indico que los peridos sean cada 4 ms
    aviso.sigev_notify=SIGEV_SIGNAL;
    aviso.sigev_signo=SIGUSR1;
    timer_create(CLOCK_REALTIME, &aviso, &temporizador); //creo el temporizador
llamado tempo, de tiempo real y que avise con la senal configurada en el segundo
argumento de entrada

    //inicio el temporizador
    timer_settime(&temporizador, TIMER_ABSTIME, &periodo , NULL);
    while(1){}
    exit(0);
}
//función que calcula la acción de control
float calculo_lqr(float A, float B, float gir) {
      phi = (gir + offset)*DEG_TO_RAD;
      dphi = gy_escalado*DEG_TO_RAD;
      dtheta = A;
      ivr = ivr + B - A;
    u = phi*K[0] + dphi*K[1] + (B+A)*K[2] + ivr*K[3];
    return (u);
}
float filtro(float accel, float gxy) {
      dtt = (double)(clock() - inicial1)/CLOCKS_PER_SEC;
      giros_ang_y_prev = TAU*(giros_ang_y_prev + dtt*gxy) + (1-TAU)*accel;
      inicial1 = clock();
      return(giros_ang_y_prev);
}
```

11.6.- Código de la Raspberry Pi para implementar MPC con SPCIES

```
//DECLARACIÓN DE CABECERAS
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <iostream>
#include <wiringPi.h>
#include <wiringSerial.h>
#include <chrono>
#include "mpc_solver.h"

using namespace std::chrono;

//Para implementar el control LQRI se varía la cuarta componente del vector K a
0.3
float offset = 3.5;
float K[] = {-344.0464,-42.5310,-7.6876,0.0};

//FUNCIONES
float calculo_lqr(float A, float B, float gir);
float filtro(float accel, float gxy);

//CAMBIO DE RADIANES A GRADOS
#define DEG_TO_RAD 0.01745329252
float PI = 3.1415;
int cont;

//VARIABLES PARA COMUNICAR CON ARDUINO
float omega, omega_r;
uint16_t omegaByte, omega_rByte;
float alpha;
float aceleracion;
int fd;

int gy;
float gyRad, ang_x, ang_y, ang_z;
float gy_escalado, ax_escalado, ay_escalado, az_escalado;

#define A_OFF_X 0.0
#define A_OFF_Y 0.0
#define A_OFF_Z 0.0

#define ACCEL_SENS 835.1

#define G_OFF_X -3.55
#define G_OFF_Y   0.65
#define G_OFF_Z 3.7

#define GYRO_SENS 131.0

float angulo = 0.0;
float accel_ang_y;

float giros_ang_y;
float giros_ang_y_prev;
float angAnterior = 0.0;
```

```
float TAU = 0.98;

float dt;
long tiempo_prev;

uint16_t gyByte;

#pragma pack(push, 1)
struct miEstructura {
    uint16_t angByte_x = 0;
    uint16_t angByte_y = 0;
    uint16_t angByte_z = 0;
};
#pragma pack(pop)

//VARIABLES PARA CONTROL LQR
float ivr = 0, u;
float dtheta_r, phi,dphi,dtheta;

//VARIABLES PARA TEMPORIZADOR Y SENAL
timer_t temporizador; //declaración del nombre del temporizador
struct itimerspec periodo; //estructura para indicar cada cuanto tiempo se lanza
el temporizador
struct timespec inicial;
struct timespec tiempo; //estructura para definirla duración del temporizador
struct sigevent aviso; //establezco una senal que avise que se ha cumplido el
temporizador
sigset_t sign; //conjunto de senales
struct sigaction act;

double x0SP[] = {0.0,0.0,0.0}
double xrSP[] = {0.0,0.0,0.0}
double u_optSP[] = {0.0}
double urSP[] = {0.0}
int pointer_kSP[] = {0}
int e_flagSP[] = {0}
solution solSP;

//VARIABLE EJECUTIVO CICLICO
int ciclo=0;
double dtt;
clock_t iniciall;
int flag=0;

int uint2int(uint16_t valor) {
      int salida = (int)valor;
      if (salida > 32768) {
            salida -= 65536;
      }
      return salida;

  }
```

```
float uint2float(uint16_t valor) {
      float salida = (float)valor;
      salida -= 32768;
      salida = salida/100;
      return salida;
}

auto start = 0;
auto stop = 0;
auto duration = 0;

static bool yaEjecutado = false;
char vaciar;

const float beta = 0.15;

unsigned char buffer[sizeof(miEstructura)];
miEstructura datos;

//FUNCIÓN MANEJADOR
void Manejador(int signo, siginfo_t *info, void*context) {
    ciclo++;
    //CADA VEZ QUE SE ACTIVA LA SENAL, CICLO AUMENTA EN UNA UNIDAD Y HACE QUE LE
CORRESPONDA EN EL SWITCH-CASE
    switch(ciclo) {
        case 1:
            for (int i = 0; i < sizeof(miEstructura); i++) {
                buffer[i] = serialGetchar(fd);
            }
            memcpy(&datos, buffer, sizeof(miEstructura));

            ang_x = uint2int(datos.angByte_x);
            ang_y = uint2int(datos.angByte_y);
            ang_z = uint2int(datos.angByte_z);

            ax_escalado = round((ang_x - A_OFF_X)*1000.0 / ACCEL_SENS)/1000.0;
            ay_escalado = round((ang_y - A_OFF_Y)*1000.0 / ACCEL_SENS)/1000.0;
            az_escalado = round((ang_z - A_OFF_Z)*1000.0 / ACCEL_SENS)/1000.0;
        break;

        case 2:
                for (int i = 0; i < sizeof(uint16_t); i++) {
                buffer[i] = serialGetchar(fd);
            }
            memcpy(&gyByte, buffer, sizeof(uint16_t));

            gy = uint2int(gyByte);
            gy_escalado = round((gy - G_OFF_Y)*1000.0 / GYRO_SENS)/1000.0;

            accel_ang_y = -atan2(ax_escalado, (sqrt((az_escalado*az_escalado) +
(ay_escalado*ay_escalado))))*(180/PI);
            giros_ang_y = filtro(accel_ang_y,gy_escalado);

                  phi = (giros_ang_y + offset)*DEG_TO_RAD;
                  dphi = gy_escalado*DEG_TO_RAD;
```

```
std::cout << phi/DEG_TO_RAD <<  "  " << dphi/DEG_TO_RAD << "  " << omega << "  "
<< alpha << std::endl;
            std::cout << serialDataAvail(fd) << std::endl;
        break;

        case 3:
                  for (int i = 0; i < sizeof(uint16_t); i++) {
               buffer[i] = serialGetchar(fd);
            }
            memcpy(&omegaByte, buffer, sizeof(uint16_t));

            omega = uint2float(omegaByte);

            //std::cout << "omega: " << omega << std::endl;
        break;

        case 4:
                  laxMPC_ADMM(x0SP, xrSP, urSP, u_optSP, pointer_kSP, e_flagSP,
solSP);
                  alpha = u_optSP[0];
            break;


        case 5:
            for (int i = 0; i < 4; i++) {
               serialPutchar(fd, *((char *)&alpha + i));
            }
            ciclo = 0;
        break;
    }
}

//FUNCIÓN PRINCIPAL MAIN
int main(void) {
    //CODIGO DE CONEXION CON ARDUINO
    // Abrir comunicacion serial
    fd = serialOpen("/dev/ttyUSB0", 115200);

    if (fd < 0) {
      std::cerr << "Error al abrir puerto serie" << std::endl;
      return 1;

       }
```

```
if (!yaEjecutado) {
            while (serialDataAvail(fd) > 0) {
                  vaciar = serialGetchar(fd);
            }
            yaEjecutado = true;
      }

    //TRATAMIENTO DE SENALES
    sigemptyset(&sign); //crea una mascara vacía
    sigaddset(&sign, SIGUSR1); //anade la senal SIGALARM al conjunto
    pthread_sigmask(SIG_UNBLOCK,&sign, NULL); //bloqueo la senal para el proceso
    /* programo la senar SIGVTALRM para que salte la funcion manejador
    cuando se cumpla el temporizador */
    act.sa_sigaction=Manejador;
    sigemptyset(&(act.sa_mask));
    act.sa_flags=SA_SIGINFO;
    sigaction(SIGUSR1, &act, NULL);

    //configuro el temporizador
    inicial.tv_sec=1;
    tiempo.tv_nsec=4000000; //cada 4 ms salta el manejador
    periodo.it_value=inicial; //indico que el primer valor lo lance 1 segundo
despues de llamarlo
    periodo.it_interval=tiempo; //indico que los peridos sean cada 4 ms
    aviso.sigev_notify=SIGEV_SIGNAL;
    aviso.sigev_signo=SIGUSR1;
    timer_create(CLOCK_REALTIME, &aviso, &temporizador); //creo el temporizador
llamado tempo, de tiempo real y que avise con la senal configurada en el segundo
argumento de entrada

    //inicio el temporizador
    timer_settime(&temporizador, TIMER_ABSTIME, &periodo , NULL);
    while(1){}
    exit(0);

  }



float filtro(float accel, float gxy) {
      dtt = (double)(clock() - iniciall)/CLOCKS_PER_SEC;
      giros_ang_y_prev = TAU*(giros_ang_y_prev + dtt*gxy) + (1-TAU)*accel;
      iniciall = clock();
      return(giros_ang_y_prev);
}
```

11.- Anexos

11.7.- Código de la Raspberry Pi para implementar MPC con CasADi

```
//DECLARACIÓN DE CABECERAS
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <iostream>
#include <unistd.h>
#include <chrono>
#include <math.h>
#include <fcntl.h>
#include <termios.h>
#include <casadi/casadi.hpp>

using namespace std::chrono;

//Para implementar el control LQRI se varía la cuarta componente del vector K a
0.3
float offset = 3.5;
float K[] = {-344.0464,-42.5310,-7.6876,0.0};

//FUNCIONES
float calculo_lqr(float A, float B, float gir);
float filtro(float accel, float gxy);

//CAMBIO DE RADIANES A GRADOS
#define DEG_TO_RAD 0.01745329252
float PI = 3.1415;
int cont;

//VARIABLES PARA COMUNICAR CON ARDUINO
float omega, omega_r;
uint16_t omegaByte, omega_rByte;
float alpha;
float aceleracion;
int fd;

int gy;
float gyRad, ang_x, ang_y, ang_z;
float gy_escalado, ax_escalado, ay_escalado, az_escalado;

#define A_OFF_X 0.0
#define A_OFF_Y 0.0
#define A_OFF_Z 0.0

#define ACCEL_SENS 835.1

#define G_OFF_X -3.55
#define G_OFF_Y   0.65
#define G_OFF_Z 3.7

#define GYRO_SENS 131.0
```

```
float angulo = 0.0;
float accel_ang_y;

float giros_ang_y;
float giros_ang_y_prev;
float angAnterior = 0.0;

float TAU = 0.98;

long tiempo_prev;

uint16_t gyByte;

#pragma pack(push, 1)
struct miEstructura {
    uint16_t angByte_x = 0;
    uint16_t angByte_y = 0;
    uint16_t angByte_z = 0;
};
#pragma pack(pop)

//VARIABLES PARA CONTROL LQR
float ivr = 0, u;
float dtheta_r, phi,dphi,dtheta;

//VARIABLES PARA TEMPORIZADOR Y SENAL
timer_t temporizador; //declaración del nombre del temporizador
struct itimerspec periodo; //estructura para indicar cada cuanto tiempo se lanza
el temporizador
struct timespec inicial;
struct timespec tiempo; //estructura para definirla duración del temporizador
struct sigevent aviso; //establezco una senal que avise que se ha cumplido el
temporizador
sigset_t sign; //conjunto de senales
struct sigaction act;

//VARIABLE EJECUTIVO CICLICO
int ciclo=0;
double dtt;
clock_t iniciall;
int flag=0;

int uint2int(uint16_t valor) {
      int salida = (int)valor;
      if (salida > 32768) {
            salida -= 65536;
      }
      return salida;
}

float uint2float(uint16_t valor) {
      float salida = (float)valor;
      salida -= 32768;
      salida = salida/100;
      return salida;
}
```

```
auto start = 0;
auto stop = 0;
auto duration = 0;

static bool yaEjecutado = false;
char vaciar;

const float beta = 0.15;

unsigned char buffer[sizeof(miEstructura)];
miEstructura datos;

// CASADI
const double L = 0.05;
const double mr = 0.068;
const double Masa = 1.228 - mr;
const double Radio = 0.0495;
const double g = 9.81;

double a = 2*Masa*L*L;
double b = Masa*Radio*L;
double c = Radio*Radio*(3*mr + Masa);
double d = Masa*g*L;

int N = 25;

double ang = -0.2618;
double dang = 0;
double theta = 0;

double Ad[3][3] = {{1.013152476 , 0.0200876 , 0},{1.31812321 , 1.013152476 ,
0},{0 , 0 , 1}};
double Bd[3][1] = {{-0.000143622254477},{-0.01439362607983},{0.02}};

double dt = 0.02;

double u_opt = 0.0;

casadi::MX f(const casadi::MX& x, const casadi::MX& u) {
      return vertcat(x(1), -(c*u - d*sin(x(0)) - b*sin(x(0))*x(1) +
b*cos(x(0))*u)/(a + b*cos(x(0))), x(2) + dt*u);
}

casadi::Opti opti = casadi::Opti();

casadi::Slice all;

casadi::MX X = opti.variable(3, N + 1);
casadi::MX U = opti.variable(1, N);
casadi::MX T = opti.variable();

casadi::MX k1 = f(X(all,0), U(all,0));
casadi::MX k2 = f(X(all,0)+dt/2*k1, U(all,0));
casadi::MX k3 = f(X(all,0)+dt/2*k2, U(all,0));
casadi::MX k4 = f(X(all,0)+dt*k3, U(all,0));

casadi::MX x_next = X(all,0) + dt/6*(k1+2*k2+2*k3+k4);
```

```
casadi::Dict opts_dict = casadi::Dict();

casadi::OptiSol sol = opti.solve();

casadi::DM u_optimal_e = sol.value(U);

//FUNCIÓN MANEJADOR
void Manejador(int signo, siginfo_t *info, void*context) {
    ciclo++;
    //CADA VEZ QUE SE ACTIVA LA SENAL, CICLO AUMENTA EN UNA UNIDAD Y HACE QUE LE
CORRESPONDA EN EL SWITCH-CASE
    switch(ciclo) {
        case 1:
            for (int i = 0; i < sizeof(miEstructura); i++) {
            read(fd, &buffer, sizeof(buffer));
            }
            memcpy(&datos, buffer, sizeof(miEstructura));

            ang_x = uint2int(datos.angByte_x);
            ang_y = uint2int(datos.angByte_y);
            ang_z = uint2int(datos.angByte_z);

            ax_escalado = round((ang_x - A_OFF_X)*1000.0 / ACCEL_SENS)/1000.0;
            ay_escalado = round((ang_y - A_OFF_Y)*1000.0 / ACCEL_SENS)/1000.0;
            az_escalado = round((ang_z - A_OFF_Z)*1000.0 / ACCEL_SENS)/1000.0;
        break;

        case 2:
          for (int i = 0; i < sizeof(uint16_t); i++) {
            read(fd, &buffer, sizeof(buffer));
            }
            memcpy(&gyByte, buffer, sizeof(uint16_t));

            gy = uint2int(gyByte);
            gy_escalado = round((gy - G_OFF_Y)*1000.0 / GYRO_SENS)/1000.0;

            accel_ang_y = -atan2(ax_escalado, (sqrt((az_escalado*az_escalado) +
(ay_escalado*ay_escalado))))*(180/PI);
            giros_ang_y = filtro(accel_ang_y,gy_escalado);

            std::cout << phi/DEG_TO_RAD <<  "  " << dphi/DEG_TO_RAD << "  " <<
omega << "  " << alpha << std::endl;
            //std::cout << serialDataAvail(fd) << std::endl;
        break;

        case 3:
          for (int i = 0; i < sizeof(uint16_t); i++) {
            read(fd, &buffer, sizeof(buffer));
            }
            memcpy(&omegaByte, buffer, sizeof(uint16_t));

            omega = uint2float(omegaByte);

            //std::cout << "omega: " << omega << std::endl;
        break;
```

```
case 4:
                X = opti.variable(3, N + 1);
            U = opti.variable(1, N);
            T = opti.variable();

            opti.minimize(T);

            for (int k = 0; k < N; k++) {
                k1 = f(X(all,k), U(all,k));
                k2 = f(X(all,k)+dt/2*k1, U(all,k));
                k3 = f(X(all,k)+dt/2*k2, U(all,k));
                k4 = f(X(all,k)+dt*k3, U(all,k));
                x_next = X(all,k) + dt/6*(k1+2*k2+2*k3+k4);
                opti.subject_to(X(all,k+1)==x_next);
            }

            opti.subject_to(X(0,0) == ang);
            opti.subject_to(X(1,0) == dang);
            opti.subject_to(X(2,0) == theta);
            opti.subject_to(-3.151592/2.0 <= X(0,all) <= 3.151592/2.0);
            opti.subject_to(-4 <= X(1,all) <= 4);
            opti.subject_to(-60 <= X(2,all) <= 60);
            opti.subject_to(-90 <= U <= 90);

            opti.subject_to(T >= 0);
            opti.set_initial(T, 1);
            opts_dict = casadi::Dict();
            opts_dict["ipopt.print_level"] = 0;
            opts_dict["ipopt.sb"] = "yes";
            opts_dict["print_time"] = 0;

            opti.solver("ipopt",opts_dict);

            sol = opti.solve();

            u_optimal_e = sol.value(U);

            alpha = (double)evalf(u_optimal_e(0,0));
        break;

        case 5:
            for (int i = 0; i < 4; i++) {
               write(fd, (char *)&alpha + i, sizeof(alpha));
            }
            ciclo = 0;
        break;
    }
}
//FUNCIÓN PRINCIPAL MAIN
int main(void) {
    //CODIGO DE CONEXION CON ARDUINO
    // Abrir comunicacion serial
    fd = open("/dev/ttyUSB0", O_RDWR);

    if (fd < 0) {
            std::cerr << "Error al abrir puerto serie" << std::endl;
            return 1;
    }
```

```
struct termios tty;

      cfsetispeed(&tty, B115200);
      cfsetospeed(&tty, B115200);

    //TRATAMIENTO DE SENALES
    sigemptyset(&sign); //crea una mascara vacía
    sigaddset(&sign, SIGUSR1); //anade la senal SIGALARM al conjunto
    pthread_sigmask(SIG_UNBLOCK,&sign, NULL); //bloqueo la senal para el proceso
    /* programo la senar SIGVTALRM para que salte la funcion manejador
    cuando se cumpla el temporizador */
    act.sa_sigaction=Manejador;
    sigemptyset(&(act.sa_mask));
    act.sa_flags=SA_SIGINFO;
    sigaction(SIGUSR1, &act, NULL);

    //configuro el temporizador
    inicial.tv_sec=1;
    tiempo.tv_nsec=4000000; //cada 4 ms salta el manejador
    periodo.it_value=inicial; //indico que el primer valor lo lance 1 segundo
despues de llamarlo
    periodo.it_interval=tiempo; //indico que los peridos sean cada 4 ms
    aviso.sigev_notify=SIGEV_SIGNAL;
    aviso.sigev_signo=SIGUSR1;
    timer_create(CLOCK_REALTIME, &aviso, &temporizador); //creo el temporizador
llamado tempo, de tiempo real y que avise con la senal configurada en el segundo
argumento de entrada

    //inicio el temporizador
    timer_settime(&temporizador, TIMER_ABSTIME, &periodo , NULL);
    while(1){}
    exit(0);
}

float filtro(float accel, float gxy) {
      dtt = (double)(clock() - iniciall)/CLOCKS_PER_SEC;
      giros_ang_y_prev = TAU*(giros_ang_y_prev + dtt*gxy) + (1-TAU)*accel;
      iniciall = clock();
      return(giros_ang_y_prev);
}
```