Universidad Autónoma de Baja California

Instituto de Ingeniería

Maestría y Doctorado en Ciencias e Ingeniería

Análisis e implementación de algoritmo de control para péndulo rotatorio usando el método Hardware-in-the-Loop

Tesis que para obtener el grado de:

Maestro en Ingeniería

Presenta:

Ing. Horacio Alain Millán Guerrero

Director de tesis:

Dr. Jesús Caro Gutiérrez

Co-director de tesis:

Dr. Fabian N. Murrieta Rico

Mexicali, B.C. 14 de junio de 2022

Dedicatoria

A mi familia.

Agradecimientos

Agradezco a la Universidad Autónoma de Baja California, a su instituto y a su facultad de ingeniería por haberme formado como ingeniero a lo largo de estos últimos años.

Agradezco a mis maestros y miembros del comité por su conocimiento, apoyo y retroalimentación brindados durante la realización de esta investigación. Además, agradezco también al Dr. Lars Lindner, al Dr. Fabian Murrieta y al Dr. Jesús Caro por haberme apoyado arduamente en esta investigación con su conocimiento, experiencia y rigurosa atención a los detalles.

Esta investigación no habría sido posible sin el apoyo económico del Consejo Nacional de Ciencia y Tecnología (CONACyT) ni el equipo brindado por los laboratorios del Instituto de Ingeniería de la Universidad Autónoma de Baja California.

Agradezco a todos los maestros que me han compartido su conocimiento, experiencia y han tenido un impacto positivo en mi persona a lo largo de mi vida.

Finalmente, agradezco a mi familia por su apoyo incondicional, su infinita confianza en mí y su constante ejemplo de superación.

Resumen

El control de sistemas caóticos sub-actuados como el péndulo rotatorio no es una tarea sencilla. Cuando la planta se fabrica físicamente, esta cuenta con fenómenos no lineales como fricción estática y backlash. Agregado a todo lo anterior, cuando se diseña un controlador, resulta difícil realizar experimentos debido a que la aceleración gravitatoria siempre está presente. En ocasiones es difícil comprobar si existe algún problema en la implementación del controlador, en la fabricación de la planta, en las mediciones realizadas por los sensores o alguna combinación de las anteriores. En esta investigación se desarrolló un simulador Hardware-in-the-Loop (**HIL**) que permite analizar y verificar con exactitud si la implementación del algoritmo de control es correcta. El **HIL** resuelve en tiempo real las ecuaciones diferenciales que describen el comportamiento dinámico de la planta junto con las señales eléctricas que la planta real produciría. Un ejemplo de dichas señales son las señales de cuadratura de los sensores de posición angular. El controlador se dice verificado si la planta **HIL** logra ser estabilizada exitosamente. Con el **HIL** también es posible observar estados no observables del sistema. Esto permite comprobar y cuantificar el rendimiento de observadores o algoritmos de estimación de estados implementados en el controlador. Además de las ventajas anteriormente mencionadas, con el **HIL** se pueden ver fenómenos que ocurren debido al muestreo, a la cuantización, a la resolución aritmética, etc. Con el **HIL** es posible variar fácilmente las condiciones experimentales como la aceleración de la gravedad, las condiciones iniciales y los parámetros del sistema. En esta investigación se verificó el controlador con el **HIL**, se compararon las simulaciones con la planta real, se cuantificó el rendimiento de diferentes estimadores de velocidad comparándolos con los estados internos del **HIL**, se simuló la planta y sus sensores y se replicaron fenómenos que ocurren con la planta real.

Índice

Listas

Lista de símbolos

Tabla 1: Tabla de símbolos.

Símbolo	**Descripción**	**Símbolo**	**Descripción**
α	Constante del péndulo	M	Masa M
β	Constante del péndulo	m_a	Masa del brazo
c	Fricción Viscosa	m_p	Masa de la barra
δ	Constante del péndulo	n	Número de muestra
$dpos$	Diferencial de posición	ϕ	Posición del eje ϕ
dt	Diferencial de tiempo	$\dot{\phi}$	Velocidad del eje ϕ
ϵ	Error	R_a	Resistencia eléctrica del motor
f	Función (cualquiera)	r_x	Posición x de la barra
F_s	Frecuencia de muestreo	r_y	Posición y de la barra
F_v	Fricción Viscosa	r_z	Posición z de la barra
$F_{v\phi}$	Fricción Viscosa en eje phi	τ_ϕ	Torque en eje ϕ
$F_{v\theta}$	Fricción Viscosa en eje theta	τ_θ	Torque en eje θ
γ	Constante del péndulo	θ	Posición del eje θ
I_s	Momento de inercia del péndulo simple	$\dot{\theta}$	Velocidad del eje θ
L	Longitud del péndulo simple	m	Masa del péndulo simple
J	Momento de inercia del péndulo rotatorio	T_s	Periodo de muestreo
k_1	Constante de RK4	v	Velocidad
k_2	Constante de RK4	v_x	Velocidad en x de la barra
k_3	Constante de RK4	v_y	Velocidad en y de la barra
k_4	Constante de RK4	v_z	Velocidad en z de la barra
K_b	Constante de Back-EMF	$\mathbf{x}$	Vector de estados
K_{ma}	Constante de Torque	x_1	Estado 1. Pos ϕ
$\mathcal{L}$	Transformada de Laplace	x_2	Estado 2. Vel ϕ

Continúa en siguiente página

Tabla 1 – Continuación de página anterior

Símbolo	**Descripción**	**Símbolo**	**Descripción**
l_a	Longitud del brazo	x_3	Estado 3. Pos θ
L_a	Inductancia del motor	x_4	Estado 4. Vel θ
λ	Fricción no lineal	y_n	Resultado RK4
l_p	Longitud de la barra	u	Entrada
$\boldsymbol{A}$	Matriz de sistema	J_c	Función de costo
$\boldsymbol{B}$	Matriz de entrada	$\boldsymbol{x_0}$	Estado inicial
$\boldsymbol{C}$	Matriz de salida	$\boldsymbol{x_{REF}}$	Estado de referencia
$\boldsymbol{D}$	Matriz de transmisión directa	t	Tiempo
$\boldsymbol{K}$	Ganancia óptima de control	g	Aceleración gravitatoria

Algunos símbolos pueden tener los subíndices. Por ejemplo, el subíndice, $\square_d$, denota que la variable o constante descrita por el símbolo $\square$ es discreta.

De igual forma también se pueden tener superíndices. Por ejemplo, el superíndice, $\hat{\square}$, denota que la variable $\square$ es estimada como en el caso del filtro Kalman.

Lista de figuras

Lista de tablas

1. Introducción

El péndulo rotatorio es un sistema electro-mecánico conformado por un brazo que gira en un plano horizontal (paralelo al piso) con una barra conectada en la orilla. La barra gira en un plano vertical perpendicular al plano horizontal (ver Fig. 1). La barra gira libremente con un rodamiento y el brazo gira por el torque aplicado con un motor. Fue creado por Katsuhisa Furuta en 1992 en el Instituto Tecnológico de Tokio.

El péndulo rotatorio es un sistema con 2 grados de libertad, un grado de libertad por cada eje de giro. Se modela matemáticamente como dos torques; el torque del brazo y el torque de la barra [5]. La barra solo puede girarse indirectamente aplicando torque al brazo, además, el péndulo no puede seguir trayectorias arbitrarias por razones que se explicarán más adelante. Estas limitantes lo convierten en un sistema sub-actuado.

El modelo matemático puede utilizarse para diseñar un controlador que estabilice el péndulo a un estado de equilibrio inestable, por ejemplo, con ángulo de cero grados, es decir, con la barra apuntando hacia arriba.

El péndulo rotatorio, al igual que otros sistemas similares, es un excelente sistema para probar algoritmos de control y realizar experimentos. Se pueden diseñar controladores con redes neuronales [6], controladores no-lineales [7], e incluso utilizar visión por computadora para retroalimentar el sistema sin la necesidad de utilizar sensores angulares en los ejes de giro [8].

Para que el controlador realice su trabajo correctamente, es necesario medir el estado del sistema de una forma tan precisa como sea razonable. En la práctica existen fenómenos de muestreo que impiden medir con precisión el estado del sistema a bajas velocidades, fenómenos de comportamiento no-lineal del péndulo rotatorio y fenómenos del motor y driver. Todos estos fenómenos producen un comportamiento complejo del sistema que resulta en un mal desempeño del controlador. Si a esto se le agrega la imperfección de la construcción física del péndulo rotatorio, la tarea de estabilizar el péndulo rotatorio resulta complicada. Para estabilizar el péndulo rotatorio es necesario eliminar; o por lo menos atenuar, los problemas anteriormente mencionados.

Diseñar controladores para un sistema altamente no-lineal y caótico, así como realizar experimentos resulta complicado. Para tener un ambiente de experimentación completamente bajo control, donde todos los parámetros y estados del sistema puedan ser observados y/o modificados es conveniente utilizar una planta simulada con *Hardware-in-the-loop*.

La planta *Hardware-in-the-loop* simula el comportamiento del péndulo rotatorio en tiempo real y produce a la salida las mismas señales que el péndulo rotatorio real produciría. Utilizando este método se puede verificar la correcta implementación de algoritmos de control [9] así como realizar análisis más confiables y experimentos repetibles.

La planta *Hardware-in-the-loop* resuelve en tiempo real las ecuaciones diferenciales del péndulo rotatorio utilizando métodos numéricos. Las ecuaciones se resuelven teniendo en cuenta que las variables de entrada de la planta pueden variar en función del tiempo, tal como en la planta real [10].

Algunas de las ventajas de utilizar una planta *Hardware-in-the-loop* son el control completo de los estados y parámetros del sistema, observabilidad total del sistema y control de las condiciones experimentales como gravedad, tiempo, etc. Además de lo anterior, el diseño y experimentación se vuelve mucho más eficiente ya que fácilmente se pueden diseñar diferentes iteraciones de prototipos del controlador y la planta sin tener que construirla o modificarlas, solo hasta que se obtenga el desempeño deseado [11].

Se pueden realizar diferentes niveles de simulación con el *Hardware-in-the-loop*: Nivel señal [12], donde solo se simula la información de las entradas y salidas del sistema; nivel energía [13], donde se tiene en cuenta la conservación de energía, es decir, la planta simulada consume la misma energía eléctrica que la planta real y finalmente, nivel mecánico [14], donde la planta simulada además de consumir la misma energía requiere ser actuada por actuadores mecánicos, tal como la planta real. Cada nivel simula la planta con más fidelidad que el anterior. En esta tesis la planta *Hardware-in-the-loop* será simulada a nivel señal.

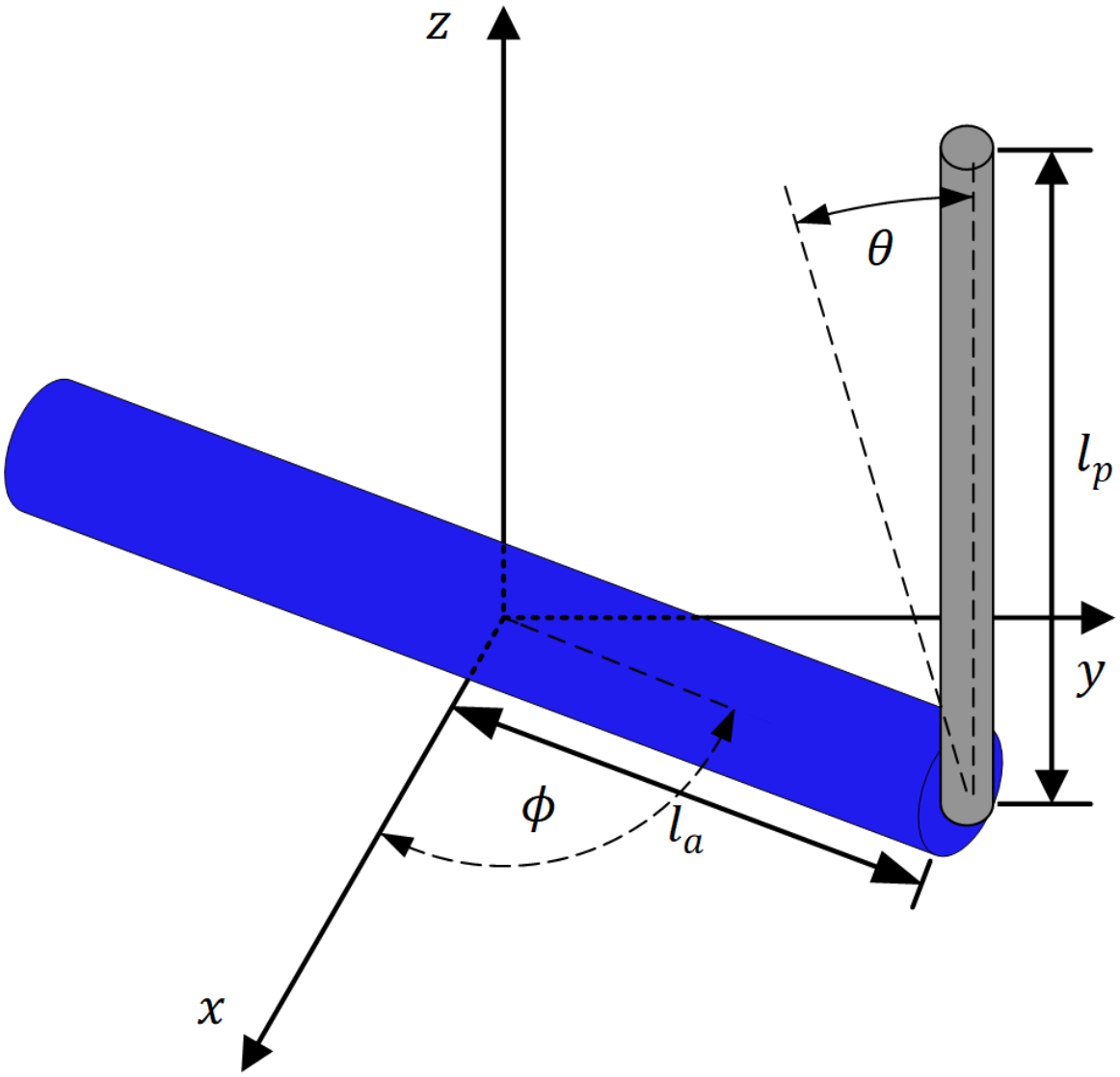


Figura 1: Diagrama del péndulo rotatorio.

2. Planteamiento del problema

El primer problema, la medición incorrecta y ruidosa de velocidad, ocurre cuando el péndulo está en equilibrio o cerca del equilibrio con la barra apuntando hacia arriba. Como en ese estado la velocidad es baja, los periodos entre pulsos del codificador rotatorio son mayores que los periodos de muestreo para medir la posición, esto se debe a que físicamente el codificador rotatorio manda una señal cuadrada con periodo mayor al periodo de muestro del μC. Esto causa que la velocidad medida sea 0 o la velocidad diferente a 0 más pequeña, $\frac{1}{T_s}$, dependiendo de la probabilidad de que el sistema tome una muestra justo después de cuando la posición del codificador rotatorio cambie. Donde T_s es el periodo de muestreo. El hecho de que la velocidad sea 0 o múltiplos de $\frac{1}{T_s}$ se debe a que la velocidad se calcula $\dot{\theta} = \frac{\theta_n - \theta_{n-1}}{T_s}$, donde el numerador es un número discreto debido a la naturaleza discreta del codificador rotatorio.

La probabilidad de detectar el cambio de estado del codificador rotatorio es una función de la posición del codificador rotatorio, su velocidad angular y el tren de impulsos de muestreo del μC.

El segundo problema, la experimentación y análisis de desempeño resultan una tarea difícil debido a que el sistema es no-lineal, caótico y la fuerza gravitatoria constantemente lo afecta. Esto causa que los experimentos a menudo no sean repetibles y complica la verificación del desempeño del algoritmo de control.

El tercer problema, la naturaleza no-lineal y caótica del péndulo además de los fenómenos que aparecen al construir la planta físicamente como por ejemplo la fricción, *backlash*, etc.

3. Justificación y uso de los resultados

Se desea implementar un algoritmo de control (controlador) y/u observador que estime la velocidad de forma continua en base a mediciones que se toman cada T_s segundos y diseñar un algoritmo y/o circuito que permita controlar con precisión un motor DC. El controlador del motor DC estará diseñado para estabilizar el péndulo rotatorio de una forma óptima. La implementación se realizará en un μC. Se encuentran más detalles al respecto en la sección 9.

Una vez se realice la implementación del controlador en el μC, se verificará su comportamiento y rendimiento utilizando una planta simulada con el método *Hardware-in-the-loop* y también con la planta real.

El conocimiento y técnica adquiridos al finalizar esta tesis también podrán ser aplicados a otros sistemas dinámicos que requieran medir posiciones, velocidades y/o controlar uno o más motores DC; especialmente si se requiere trabajar a bajas velocidades con una alta precisión de control del motor DC. Algunos de los sistemas que se benefician con este tipo de controlador son:

- Sistemas de láseres para modelado 3D.
- Sistemas utilizados en la industria de la manufactura (CNCs, etc).
- Instrumentación médica.
- Sistemas de control didácticos como el péndulo rotatorio.
- Sistemas de control para telescopios.
- Robots.

4. Objetivos de la investigación

4.1. Objetivo general

Diseñar, analizar, implementar en un μC y verificar con el *Hardware-in-the-loop* un algoritmo de control que mantenga estable el péndulo rotatorio en su punto de equilibrio inestable ($\theta = 0\ rad$).

4.2. Objetivos específicos

1. Diseño de algoritmo de estimación de velocidades.
2. Modelado y simulación del péndulo rotatorio.
3. Diseño 3D, fabricación y estimación de parámetros del péndulo real.
4. Diseño, implementación, verificación con **HIL** y análisis del algoritmo de control.

5. Hipótesis

Simular el péndulo rotatorio utilizando la técnica *Hardware-in-the-loop* debería permitir analizar y verificar algoritmos de control implementados en el μC teniendo mejores capacidades de experimentación, iteración de prototipos de controladores y observación ideal de variables no-observables.

En las secciones 8.6.2, 9.4.4 y 9.4.5 se compara la técnica **HIL** con *Simulink* y la planta física. En estas secciones se demuestra que el **HIL** permite verificar los algoritmos de control. Además, en las secciones 9.4.6 y 9.4.9 se demuestran las mejores capacidades de experimentación del **HIL** al simular también los encoders y observar estados no-observables.

6. Metodología

A continuación se muestra una figura y se describe punto a punto la metodología utilizada para lograr cumplir con los objetivos de esta investigación.

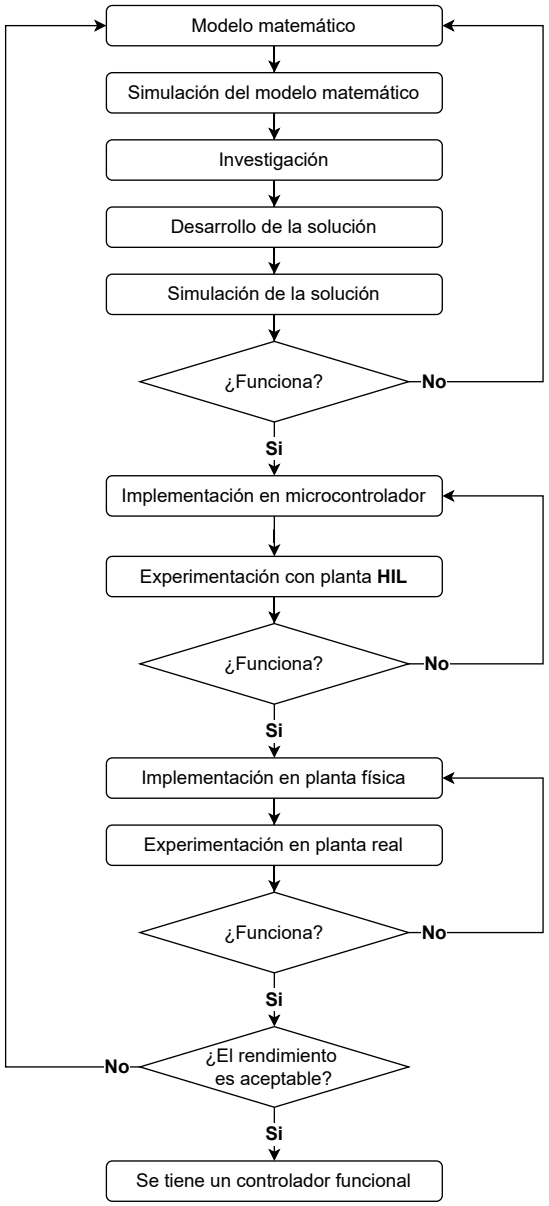


Figura 2: Diagrama de bloques de la metodología.

- Modelo matemático: Se requiere tener modelos matemáticos que describan la dinámica del péndulo, fenómenos entre el μC y el codificador rotatorio, etc. Estos modelos matemáticos ayudarán a analizar el comportamiento del sistema e identificar áreas de mejora.
- Simulación: Teniendo el modelo matemático se puede simular el comportamiento del sistema. Una gran ventaja de la simulación es observar todas las variables del sistema perfectamente sin errores de medición, también se pueden simular condiciones extremas o de frontera como por ejemplo a velocidades extremadamente bajas o altas para observar que fenómenos ocurren y como atenuarlos/eliminarlos.
- Investigación: En base al conocimiento obtenido en la simulación y con el modelo matemático se tendrá más información para poder tomar una decisión sobre cómo proceder. Teniendo identificadas las causas de los problemas se puede investigar con mayor precisión cuál es la solución.
- Desarrollo de la solución: Con todo el conocimiento adquirido se puede diseñar una nueva estrategia de medición o estimación de velocidad que tenga un rendimiento superior a la estrategia anterior.
- Simulación de la solución: Una vez diseñada la nueva estrategia de control se puede simular el comportamiento de la nueva solución.
- Implementación en μC: Se carga al μC el código con el nuevo controlador.
- Experimentación con planta *Hardware-in-the-loop*: Se verificará el rendimiento del controlador estabilizando la planta *Hardware-in-the-loop*. Si el rendimiento es satisfactorio se procede al siguiente paso.
- Implementación física: Una vez verificado que en efecto la nueva estrategia de control es una mejora de la anterior se procede al siguiente paso.
- Experimentación con planta real: Se probará el rendimiento del controlador estabilizando la planta real. El sistema real probablemente tenga un comportamiento similar al simulado con algunas diferencias. En esta etapa se realizarán pequeños ajustes por factores inesperados o que no hayan sido tomados en cuenta en pasos anteriores.

7. Fundamento teórico

7.1. Modelo matemático del péndulo rotatorio

7.1.1. Modelo 3D y declaración de parámetros

Primero, para modelar matemáticamente el péndulo se requiere especificar sus parámetros.

Se utilizan las mismas variables como en el artículo de Magnus Gäfvert, *Modelling the Furuta Pendulum* [5]

- l_a es la longitud del brazo desde el eje de giro hasta la barra
- l_p es la longitud de la barra
- ϕ es el ángulo del brazo en el plano XY
- θ es el ángulo de la barra donde $\theta = 0°$ sí la barra apunta hacia arriba

También, se tienen las masas del brazo, m_a, la masa de la barra, m_p y el momento de inercia , J, visto desde el eje del motor en el eje Z.

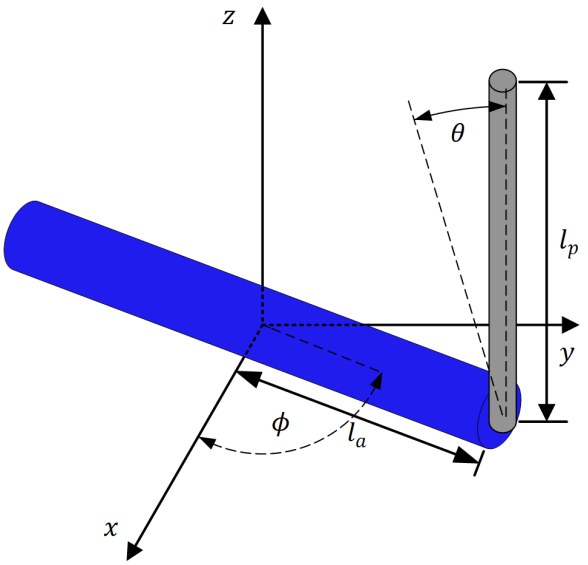


Figura 3: Diagrama del péndulo rotatorio.

7.1.2. Posición de la punta de la barra

Su posición en coordenadas cartesianas se puede obtener con trigonometría en función de los ángulos ϕ y θ y de las distancias l_a y l_p:

$$r_x(la, lp, \phi, \theta) = l_a cos\phi - l_p sen\theta sen\phi \tag{1}$$

$$r_y(la, lp, \phi, \theta) = l_a sen\phi + l_p cos\phi sen\phi \tag{2}$$

$$r_z(la, lp, \phi, \theta) = l_p cos\theta \tag{3}$$

7.1.3. Velocidad de la punta de la barra

Para obtener la velocidad de la punta de la barra simplemente se derivan las posiciones con respecto al tiempo de manera que:

$$v_x(la, lp, \phi, \theta) = -l_a sen\phi\dot{\phi} - l_p cos\theta sen\phi\dot{\theta} - l_p sen\theta cos\phi\dot{\phi} \tag{4}$$

$$v_y(la, lp, \phi, \theta) = l_a cos\phi\dot{\phi} + l_o cos\theta cos\phi\dot{\theta} - l_p sen\theta sen\phi\dot{\phi} \tag{5}$$

$$v_z(la, lp, \phi, \theta) = -l_p sen\theta\dot{\theta} \tag{6}$$

7.1.4. Ecuaciones de movimiento

Una vez obtenidas las ecuaciones de posición y velocidad se determinan las ecuaciones de energía cinética y energía potencial de cada uno de los elementos del péndulo, se forma el Lagrangiano y se llega a las siguientes ecuaciones de movimiento [5]:

$$(\alpha + \beta sen^2\theta)\ddot{\phi} + \gamma cos\theta\ddot{\theta} + 2\beta cos\theta sen\theta\dot{\phi}\dot{\theta} - \gamma sen\theta\dot{\theta}^2 = \tau_\phi \tag{7}$$

$$\gamma \cos\theta\ddot{\phi} + \beta\ddot{\theta} - \beta \cos\theta sen\theta\dot{\phi}^2 - \delta sen\theta = \tau_\theta \tag{8}$$

Donde α,β,γ y δ son constantes en función de los parámetros del péndulo utilizadas para simplificar las ecuaciones [5].

Para simular el comportamiento del péndulo resulta más conveniente representar las ecuaciones de movimiento en términos de sus velocidades y aceleraciones [5].

$$\frac{d}{dt}\phi = \dot{\phi} \tag{9}$$

$$\frac{d}{dt}\dot{\phi} = \frac{1}{\alpha\beta - \gamma^2 + (\beta^2 + \gamma^2)\, sen^2\theta}\Big[\beta\gamma\left(sen^2\theta - 1\right) sen\theta\dot{\phi}^2 - 2\beta^2 \cos\theta sen\theta\dot{\phi}\dot{\theta} + \beta\gamma sen\theta\dot{\theta}^2 - \gamma\delta \cos\theta sen\theta + \beta\tau_\phi - \gamma \cos\theta\tau_\theta\Big] \tag{10}$$

$$\frac{d}{dt}\theta = \dot{\theta} \tag{11}$$

$$\frac{d}{dt}\dot{\theta} = \frac{1}{\alpha\beta - \gamma^2 + (\beta^2 + \gamma^2)\, sen^2\theta}\Big[\beta\left(\alpha + \beta sen^2\theta\right)\cos\theta sen\theta\dot{\phi}^2 + 2\beta\gamma\left(1 - sen^2\theta\right) sen\theta\dot{\phi}\theta - \gamma^2 \cos\theta sen\theta\theta^2 + \delta\left(\alpha + \beta sen^2\theta\right) sen\theta - \gamma\cos\theta\tau_\phi + \left(\alpha + \beta sen^2\theta\right)\tau_\theta\Big] \tag{12}$$

7.2. Modelo matemático linealizado del péndulo rotatorio

Dependiendo de los requerimientos y objetivos de control resulta útil contar con un modelo linealizado del péndulo rotatorio. El sistema se linealizó alrededor del punto:

$$\boldsymbol{x_0} = [0\ 0\ 0\ 0] \tag{13}$$

con torque $\boldsymbol{\tau} = [0\ 0]$.

Las ecuaciones de estados y salida son las siguientes:

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u} \tag{14}$$

$$\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u} \tag{15}$$

Donde:

$\boldsymbol{x}$ = Vector de estados

$\dot{\boldsymbol{x}}$ = Derivada del vector de estados respecto al tiempo

$\boldsymbol{y}$ = Vector de salida

$\boldsymbol{A}$ = Matriz del sistema

$\boldsymbol{B}$ = Matriz de entrada

$\boldsymbol{C}$ = Matriz de salida

$\boldsymbol{D}$ = Matriz de transmisión directa

Utilizando el primer término de la expansión de serie de Taylor con $\delta = \boldsymbol{x} - \boldsymbol{x_0}$:

$$\frac{d(\delta)}{dt} = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_0 \delta\boldsymbol{x} + \left.\frac{\partial f}{\partial \tau}\right|_0 = \boldsymbol{A}\delta\boldsymbol{x} + \boldsymbol{B}\tau \tag{16}$$

Donde f representa la dinámica del sistema (ver sec. 7.1.4). Esto nos lleva a obtener la matriz de sistema, $\boldsymbol{A}$ y la matriz de entrada, $\boldsymbol{B}$:

$$\boldsymbol{A} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{\delta\gamma}{\alpha\beta-\gamma^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{\alpha\delta}{\alpha\beta-\gamma^2} & 0 \end{pmatrix} \tag{17}$$

$$\boldsymbol{B} = \begin{pmatrix} 0 & 0 \\ \frac{\beta}{\alpha\beta-\gamma^2} & -\frac{\gamma}{\alpha\beta-\gamma^2} \\ 0 & 0 \\ -\frac{\gamma}{\alpha\beta-\gamma^2} & \frac{\alpha}{\alpha\beta-\gamma^2} \end{pmatrix} \tag{18}$$

Las matriz $\boldsymbol{C} = \boldsymbol{I}$ donde $\boldsymbol{I}$ es la matriz de identidad y $\boldsymbol{D} = 0$.

Es importante tener en cuenta que si se desea implementar el modelo matemático en un sistema discreto es necesario utilizar las ecuaciones de espacio de estados en dominio discreto:

$$\dot{\boldsymbol{x}}[n+1] = \boldsymbol{A_d}\boldsymbol{x}[n] + \boldsymbol{B_d}\boldsymbol{u}[n] \tag{19}$$

$$\boldsymbol{y}[n] = \boldsymbol{C_d}\boldsymbol{x}[n] + \boldsymbol{D_d}\boldsymbol{u}[n] \tag{20}$$

Donde:

$\boldsymbol{A_d} = e^{\boldsymbol{A}T}$

$\boldsymbol{B_d} = T\boldsymbol{B}$

$\boldsymbol{C_d} = C$

$\boldsymbol{D_d} = D$

Donde T es el periodo de tiempo entre iteraciones en segundos.

7.3. Simulación del comportamiento del péndulo

En base a las 4 ecuaciones en forma diferencial obtenidas en el punto 7.1.4 es posible simular el comportamiento del péndulo. Se utilizó el software *Simulink* [15] para simular el péndulo y se le agregó fricción viscosa al modelo del péndulo para observar su comportamiento.

La fricción viscosa se modeló como un torque en oposición a la dirección de movimiento. La fricción viscosa es una función de la velocidad angular: $\tau_{F_\theta} = F_v \cdot \dot{\theta}$ donde τ_{F_θ} es el torque producido por la fricción y F_v es la constante de fricción viscosa. También se sigue esta misma lógica para el torque en el eje ϕ.

Las condiciones iniciales del péndulo fueron todo igual a 0 con excepción de $\theta = 0{,}15\ rad$.

Se utilizó este ángulo para que el péndulo comience a caer, de lo contrario solo se quedaría estable verticalmente debido a la ausencia de perturbaciones en el modelo matemático.

Se observa en la figura 4 que las posiciones del péndulo tienen el comportamiento esperado, el de una señal sinusoidal con magnitud decadente de forma exponencial. Lo mismo sucede con sus velocidades angulares.

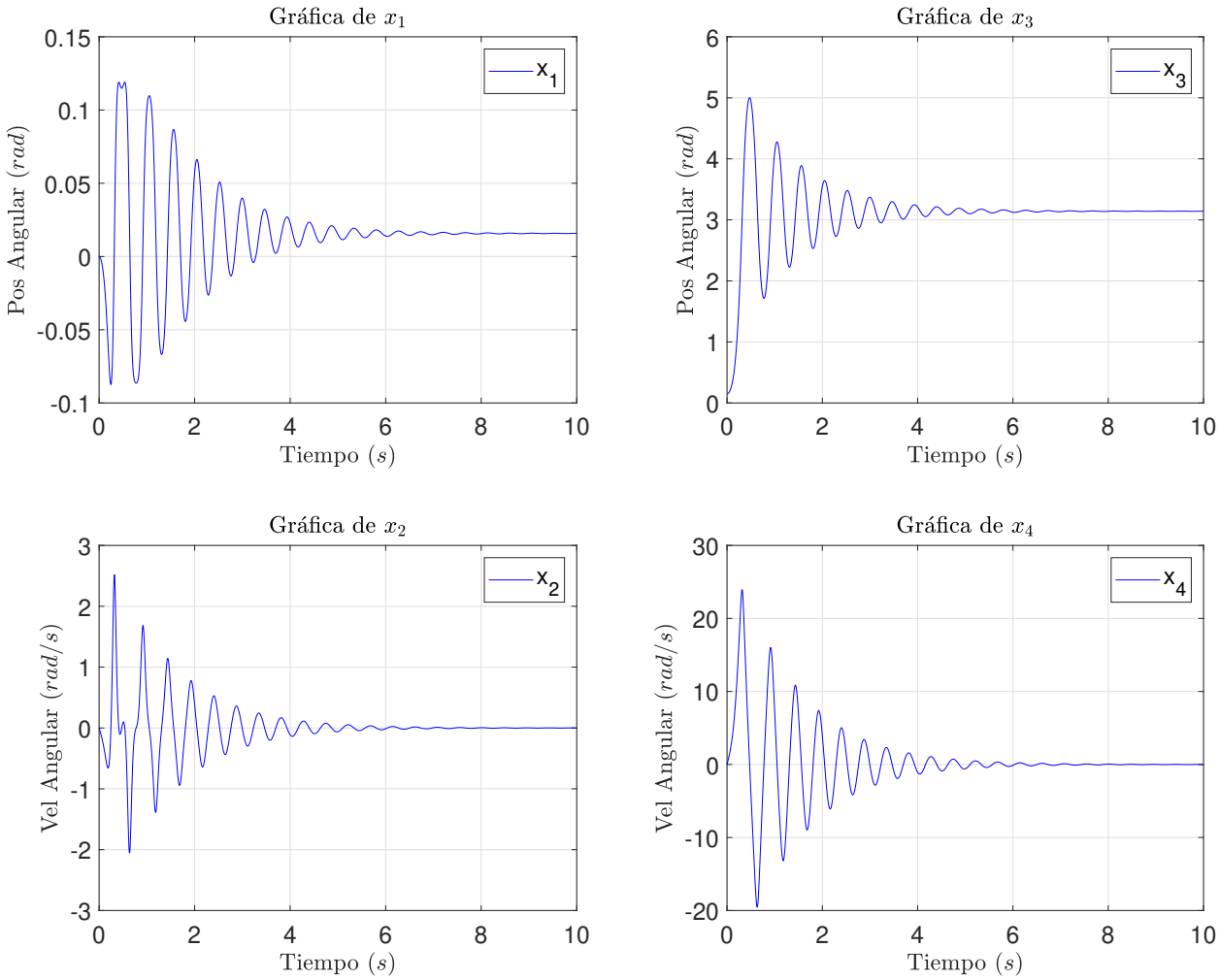


Figura 4: Posición y velocidad de los ángulos del péndulo como funciones del tiempo.

7.4. Señales discretas

7.4.1. Señales discretas en el tiempo

Las señales discretas en el tiempo son representadas mediante vectores o secuencias de números, cada magnitud tiene asociada un número de muestra. Matemáticamente se representan de la siguiente manera.

$$x[n] = x_a(nT) \tag{21}$$

Donde n es un número entero, $x[n]$ es la función discreta, x_a es la magnitud de la señal continua y T es el periodo de muestreo. Como se puede observar, la señal discreta es igual a la continua solo en los instantes de muestreo, para todos lo demás momentos, la señal discreta no existe[16].

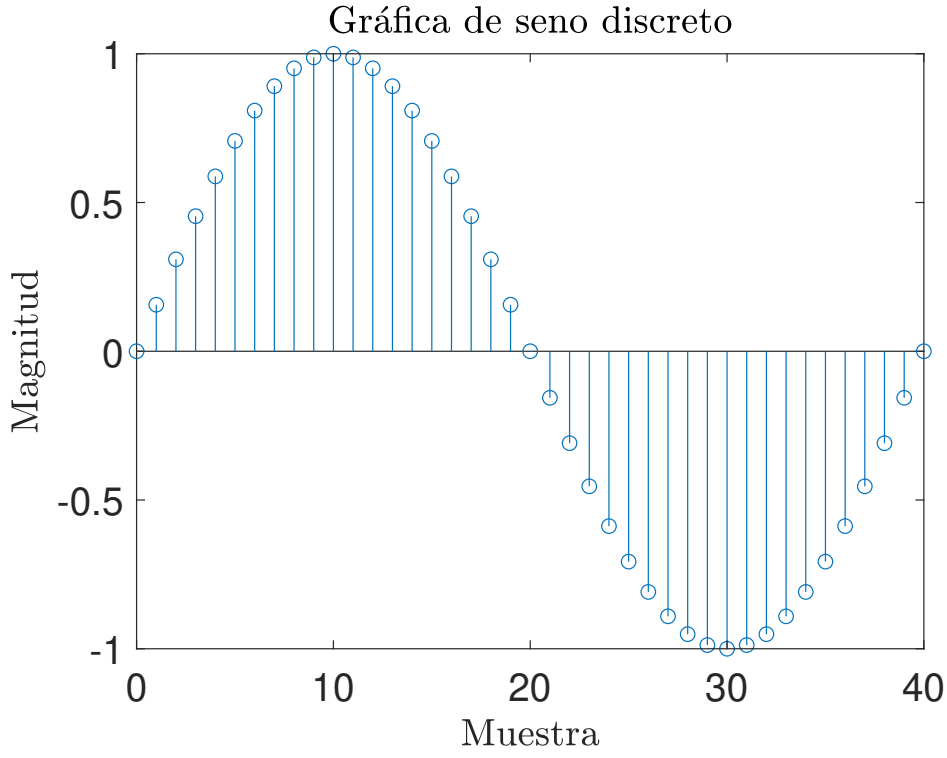


Figura 5: Sinusoide discretizado con 41 muestras.

7.4.2. Teorema del muestreo de Nyquist-Shannon

Para poder reconstruir una señal que fue muestreada con una frecuencia de muestreo inferior a ∞ se requiere estrictamente lo siguiente:

$$\Omega_s = \frac{2\pi}{T} \geq 2\Omega_N \tag{22}$$

donde Ω_s es la frecuencia de muestreo y $2\Omega_N$ es el ratio de Nyquist. Esto significa que la frecuencia de muestreo debe ser, como mínimo, igual a el doble de la frecuencia de la señal continua[1].

Si se considera un tren de impulsos, $s(t)$, como la señal de muestreo, $x_c(t)$, como la señal a muestrear y $x_s(t)$ como la señal muestreada:

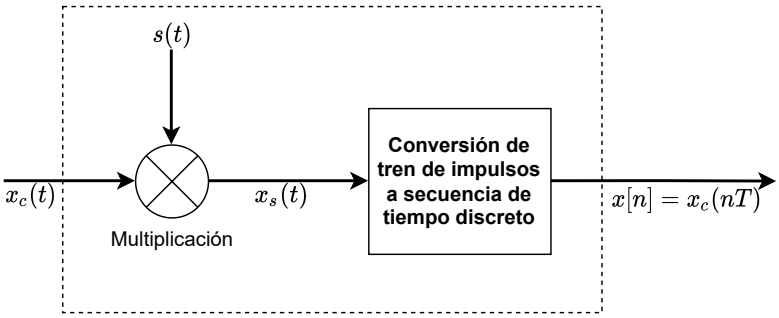


Figura 6: Sistema de muestreo[1].

La señal muestreada en el dominio de tiempo continuo es:

$$x_s(t) = \sum_{n=-\infty}^{\infty} x_c(nT)\delta(t - nT) \tag{23}$$

Convirtiendo al dominio de la frecuencia:

$$X_s(j\Omega) = \sum_{n=-\infty}^{\infty} x_c(nT)e^{-j\Omega Tn} \tag{24}$$

Ahora, teniendo en cuenta que:

$$x[n] = x_c[nT] \tag{25}$$

Y obteniendo la transformada de Fourier discreta $\mathcal{F}(x[n])$:

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} = \mathcal{F}(x[n]) \tag{26}$$

Se puede deducir que $x_s = x_c(t)$ en los instantes de tiempo múltiplos de ΩT:

$$X_s(j\Omega) = X(e^{j\omega})|_{\omega=\Omega T} = X(e^{j\Omega T}) \tag{27}$$

Dado que $x_c(t)$ y $s(t)$ se están multiplicando en el tiempo, se convolucionan en el dominio de la frecuencia:

$$X(e^{j\Omega T}) = \frac{1}{T}\sum_{k=-\infty}^{\infty} X_c(j(\Omega - k\Omega_s)) \tag{28}$$

Si $\omega = \Omega T$ se obtiene:

$$X(e^{j\omega}) = \frac{1}{T}\sum_{k=-\infty}^{\infty} X_c\left[j\left(\frac{\omega}{T} - \frac{2\pi k}{T}\right)\right] \tag{29}$$

Si $X_c(j\Omega)$ tiene un espectro triangular (o cualquier otro) al graficar el espectro de la señal muestreada se puede observar fácilmente porque se debe cumplir que $\Omega_s \geq 2\Omega_N$.

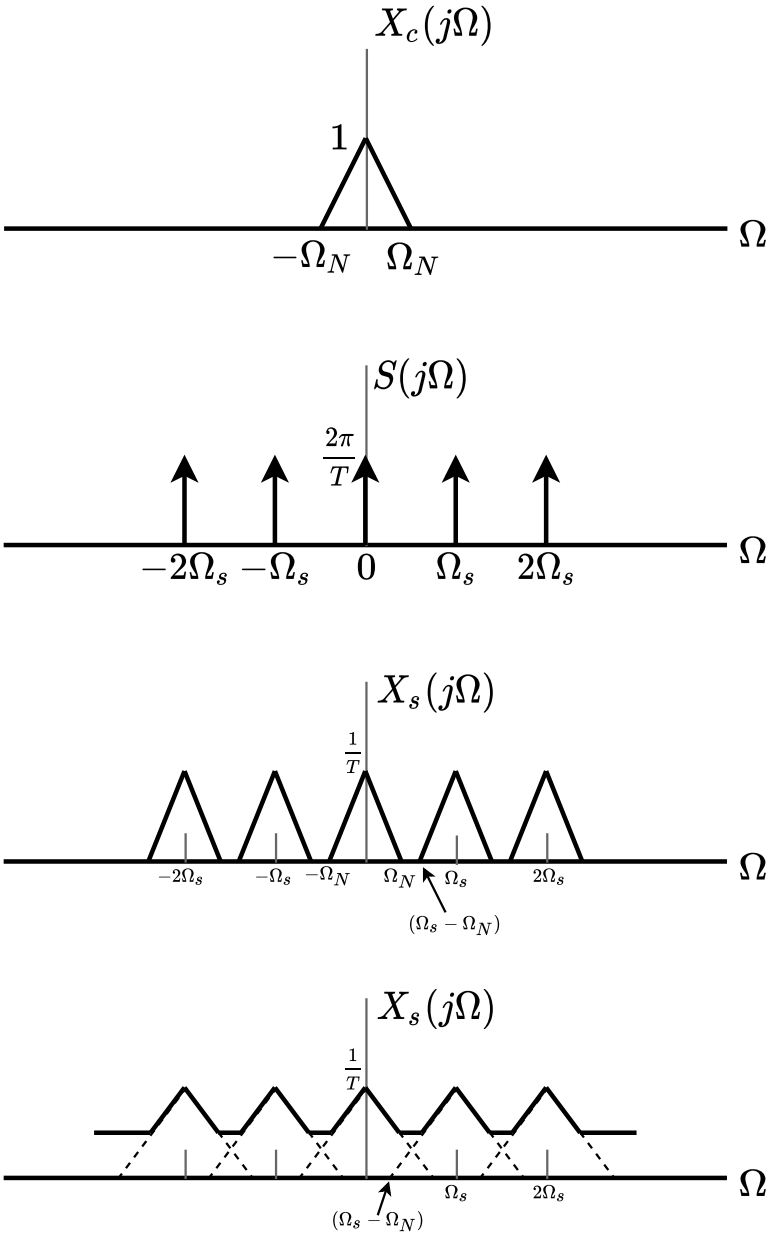


Figura 7: Representación del muestreo en el dominio de la frecuencia.
(a) Espectro de la señal original. (b) Transformada de Fourier de la señal de muestreo. (c) Transformada de Fourier de la señal muestreada con $\Omega_s > 2\Omega_N$. (d) Transformada de Fourier de la señal muestreada con $\Omega_s < 2\Omega_N$ [1].

7.4.3. Transformada Z

En el campo de la ingeniería la transformada Z es una herramienta matemática análoga a la transformada de Laplace, pero utilizada comúnmente para señales discretas. Al igual que la transformada de Laplace en tiempo continuo, la transformada Z, en tiempo discreto, es una generalización de la transformada de Fourier. Una gran ventaja de la transformada Z, al igual que la transformada de Laplace, es que nos permite solucionar ecuaciones de diferencias lineales invariantes en el tiempo de forma algebraica.

Siempre que se utilicen sistemas digitales como un μC para muestrear señales continuas van a surgir señales en tiempo discreto. Si el periodo de muestro T es constante se puede aplicar la transformada Z. Para el caso de la ingeniería y los sistemas causales, se utiliza la transformada Z unilateral. Esta está definida de la siguiente manera:

$$X(z) = Z[x(t)] = Z[x(kT)] = \sum_{k=0}^{\infty} x(kT)z^{-k} \tag{30}$$

Donde Z es la transformada Z, $x(t)$ es la señal en el tiempo continuo, $x(kT)$ es la señal en el tiempo discreto, T es el periodo de muestreo y k es el índice de la muestra [17, 18].

7.4.4. Control discreto

Gracias al avance de la tecnología electrónica es posible controlar el comportamiento de sistemas dinámicos utilizando microcontroladores, computadoras y demás sistemas discretos. Algunas aplicaciones del control discreto van desde un simple control de temperatura [19] o un controlador de un motor de inducción [20] hasta el control de osciladores de Van der Pol [21].

Modelando el sistema dinámico y utilizando la teoría de control digital es posible diseñar controladores que vía software (o también hardware) lleven el estado de un sistema a algún otro estado deseado.

Para garantizar el correcto funcionamiento del controlador ante incertidumbre en las mediciones y en el modelo matemático del sistema usualmente se utiliza retroalimentación o alguna operación similar [22].

Los sistemas de control que utilizan controladores digitales tienen las siguientes ventajas [3]:

- Flexibilidad
- Costo
- Procesamiento digital de señales
- Memoria
- Pueden implementar algoritmos complejos como AI, redes neuronales, algoritmos de corrección de errores, rutinas de auto-calibración, etc.

Una representación simple de un sistema de control digital y por lo tanto discreto es la siguiente.

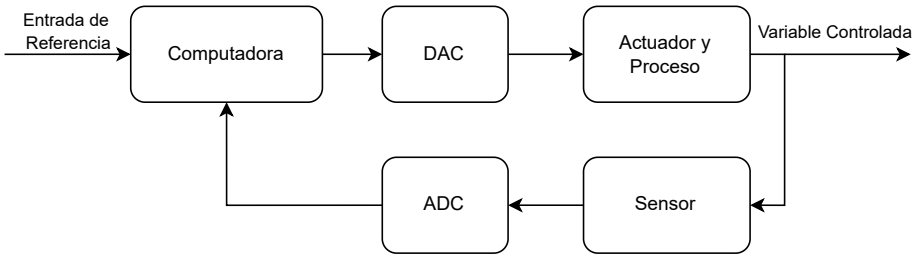


Figura 8: Diagrama a bloques de sistema de control digital con retroalimentación en bucle cerrado [17].

7.4.5. Sample and hold

Los circuitos "sample and hold" son utilizados en sistemas de muestreo y sirven para retener una magnitud analógica como un voltaje y mantenerlo constante mientras un convertidor analógico-digital lo convierte a un valor numérico como un byte para que la computadora o μC puedan realizar operaciones con tal valor [23].

Cuando un μC realiza un muestreo también se podría considerar a la memoria del mismo como un elemento "sample and hold" ya que retendrá el último valor muestreado durante un periodo T_s hasta el siguiente instante de muestreo. Físicamente un circuito (analógico) "sample and hold" puede ser construido de forma sencilla con 2 op-amps, un transistor como switch y un capacitor.

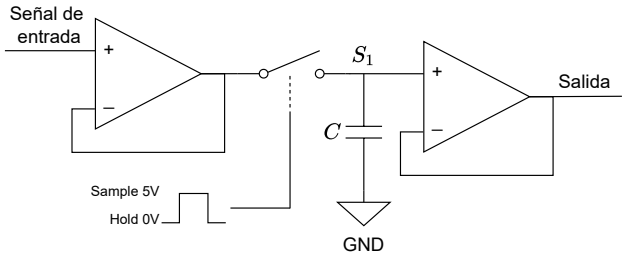


Figura 9: Circuito analógico sample and hold [2].

Matemáticamente esta acción de muestrear y mantener un valor puede ser representada en el dominio de variable compleja s por una ecuación en el dominio de Laplace de orden 0 [3]:

$$H(s) = \frac{1}{s}(1 - e^{-sT_s}) \tag{31}$$

Convirtiendo esta ecuación al dominio del tiempo se obtiene:

$$h(t) = \mathcal{L}^{-1}\{H(s)\} = \sigma(t) - \sigma(t - T_s) \tag{32}$$

Esto no es más que un pulso rectangular de periodo T_s

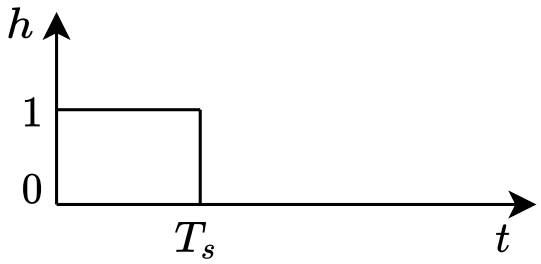


Figura 10: Pulso rectangular de periodo Ts [3].

7.4.6. PWM

La modulación de ancho de pulso (Pulse Width Modulation) [24] es una técnica que se utiliza para controlar la energía promedio de una señal variando el ciclo de trabajo, esto otorga una gran ventaja, con esta técnica los dispositivos digitales como un μC pueden tener voltajes RMS de salida en un rango casi continuo. Este rango está en función de la resolución temporal del μC.

Este tipo de señal se utiliza para controlar la velocidad angular del motor que estabiliza el péndulo rotatorio. Algunas desventajas de utilizar este tipo de señal son: la vibración que induce en el motor, los altos componentes de frecuencia que generan ruido, vibración y mal rendimiento a bajas velocidades del motor.

Estos problemas podrían ser causados por la misma naturaleza de la señal PWM o por la baja frecuencia a la que se implementó (1 kHz en el péndulo) aunque a altas frecuencias (hasta 20 kHz) no se observaron diferencias en el comportamiento del motor.

Algunas de las posibles soluciones pueden ser, utilizar una frecuencia PWM variable en función de la velocidad a la que se desee controlar el motor, aumentar la frecuencia de la señal PWM, cambiar el tipo de modulación a PDM o alguna otra, controlar el motor con voltajes continuos utilizando un circuito push-pull o un DC/DC Buck converter. Existen más posibles soluciones, pero primero es

necesario simularlo y después comprobar físicamente que en efecto la señal PWM sea el problema y no algún otro fenómeno.

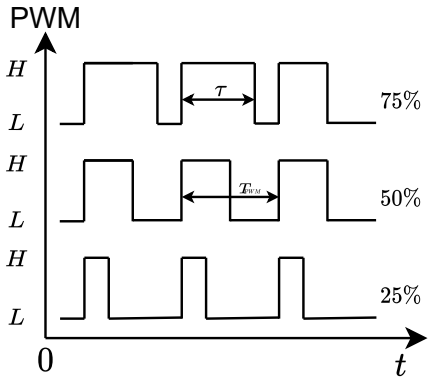


Figura 11: Señales PWM con ciclo de trabajo de 75 %, 50 % y 25 % respectivamente [3].

7.5. Codificador rotatorio

El codificador rotatorio es un dispositivo utilizado para medir el cambio de posición angular de un motor, un eje o algún artefacto que rote [25]. Este dispositivo tiene 2 señales de salida. Cuando el codificador rotatorio gira se puede obtener en sus salidas 2 señales cuadradas desfasadas $\frac{1}{4}$ de onda.

Dependiendo de la frecuencia de las señales y si este desfase es positivo o negativo, se puede determinar la dirección y velocidad del codificador. Algunas veces los codificadores rotatorios cuentan con una tercera salida que sirve para detectar cuando el codificador ha dado una vuelta completa.

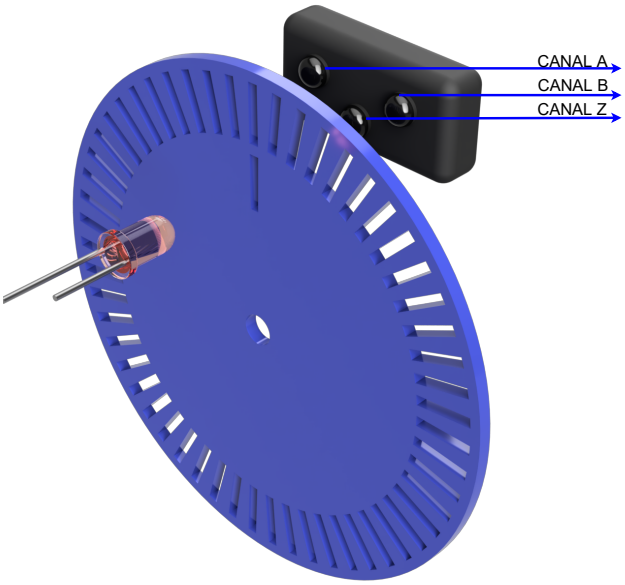


Figura 12: Diagrama de un codificador rotatorio incremental [4].

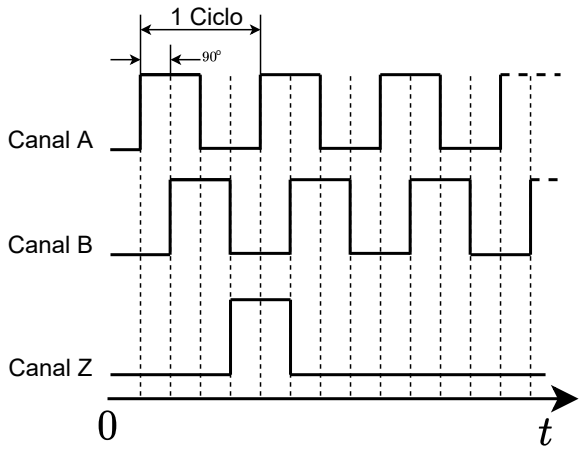


Figura 13: Señales de salida de los canales del codificador rotatorio incremental [3].

7.6. Solucionador de ODEs Runge-Kutta de 4to orden

El método de 4to orden de Runge-Kutta, inventado por Carl Runge y Martin Kutta y abreviado **RK4**, es utilizado para solucionar ecuaciones diferenciales ordinarias. Permite aproximar la solución de ecuaciones diferenciales ordinarias de forma explícita o implícita y con un intervalo de tiempo constante. El método obtiene el mismo resultado que con las series de Taylor, sin la necesidad de tener que resolver analíticamente la **ODE** [26]. Este método de solución iterativa de **ODE** se caracteriza por su simpleza matemática y un bajo costo computacional. Las características anteriores lo hacen ideal para su implementación en código.

7.6.1. Solucionador RK4 paso a paso

Para solucionar las **ODE** paso a paso mientras se tiene en cuenta entradas dinámicas que perturban al sistema, como por ejemplo la señal de control, el método **RK4** es ideal. Existen métodos más simples para solucionar **ODE** como el método de Euler pero debido a su baja exactitud en la aproximación de la solución no suele usarse. El método **RK4** implementado en código mostró una excelente precisión comparado con la simulación de *Simulink* tal como se puede observar en la figura 14.

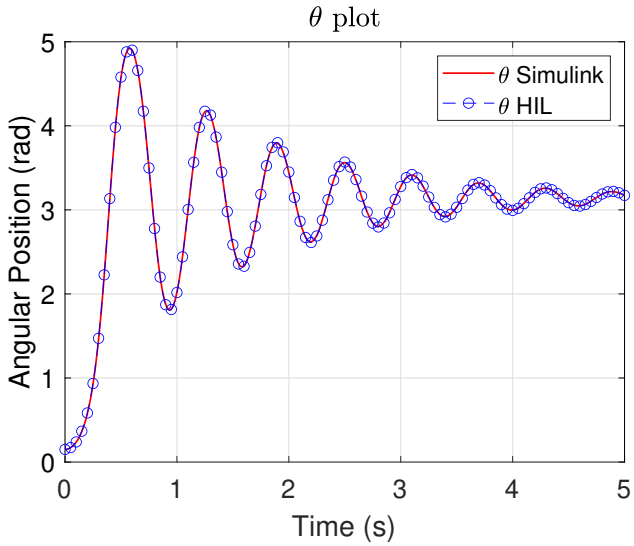


Figura 14: Ángulo θ del péndulo en bucle abierto. Comparación entre la simulación en *Simulink* y el solucionador **RK4**. Condición inicial $\theta = 0{,}15$.

7.7. Microcontroladores

Durante esta investigación se utilizaron μCs de "STMicroelectronics" debido a su alto rendimiento, su documentación, sus librerías de abstracción de hardware y sus capacidades de decodificación vía hardware de las señales de cuadratura de los codificadores rotatorios. También es importante resaltar que estos microcontroladores cuentan con *STM32CUBEIDE* [27], un excelente entorno de desarrollo integrado que permite configurar timers, gpio, protocolos de comunicación, etc, mediante una interfaz gráfica.

Además de todo lo anterior, estos μCs tienen soporte por el proyecto comunitario *PlatformIO* [28]. *PlatformIO* es un esfuerzo de la comunidad por unificar la programación de diferentes tipos y marcas de μCs en un solo entorno de desarrollo compatible basado en *Visual Studio Code* [29]. *PlatformIO* es multiplataforma, multi-arquitectura y *multi-framework*.

Durante el transcurso de esta investigación se utilizó *STM32CUBEIDE* y *PlatformIO* con *VS Code* para programar los μCs.

7.7.1. STM32F103C8

El *STM32F103C8* se utilizó con la tarjeta *blue pill*.

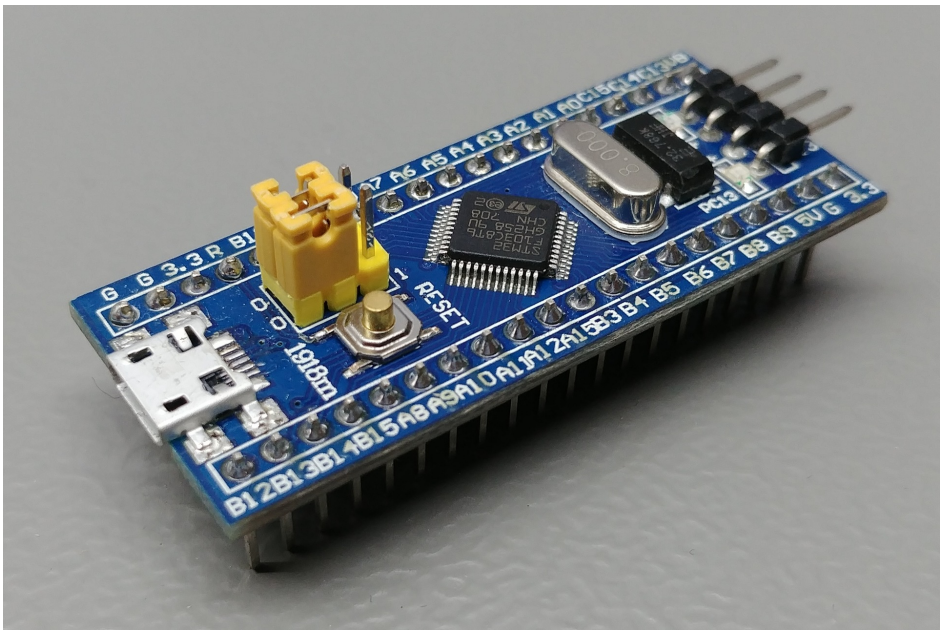


Figura 15: STM32F103C8 en tarjeta *blue pill*.

Algunas características relevantes son las siguientes:

- CPU a 72 MHz
- 2 convertidores A/D
- DMA
- 7 Timers, 4 con decodificación de señal de cuadratura
- 9 Interfaces de comunicación (I^2C, USART, SPI, USB, etc)

Más detalles en la hoja de datos [30].

7.7.2. STM32G474RE

El *STM32FG474RE* se utilizó con la tarjeta *NUCLEO-G474RE*.

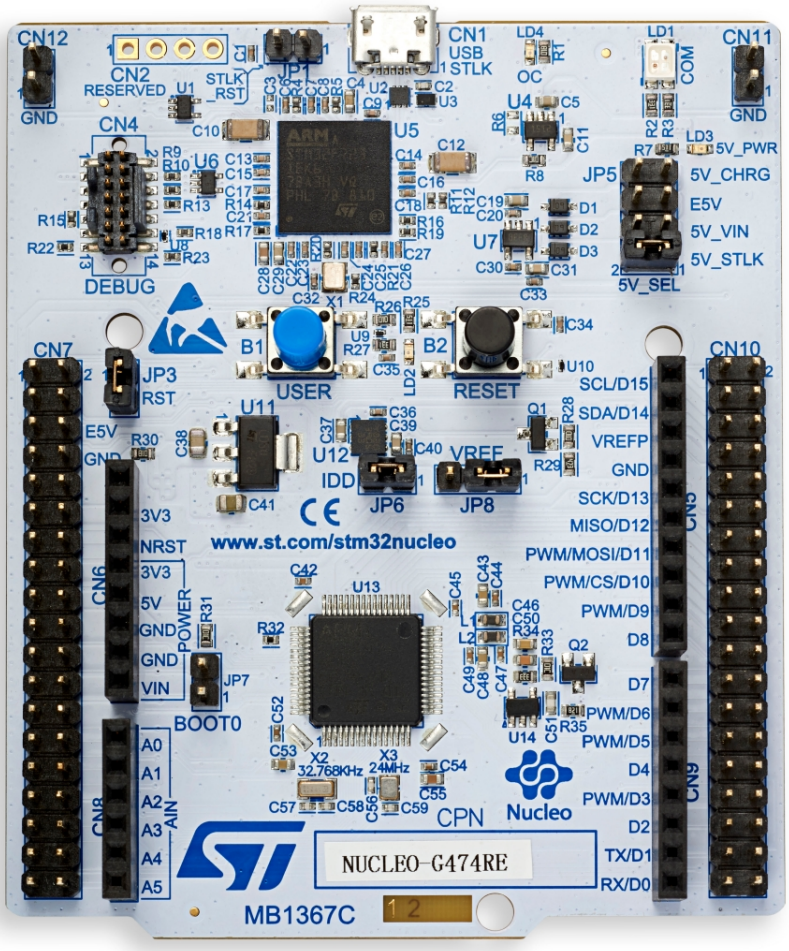


Figura 16: STM32G474RE en tarjeta *NUCLEO-G474RE*.

Algunas características relevantes son las siguientes:

- CPU a 170 MHz
- 5 convertidores A/D
- 7 convertidores D/A
- 7 Comparadores analógicos ultra rápidos
- 6 OP-Amps
- DMA
- 17 Timers, 1 con alta resolución (184 picosegundos), 12 PWM, 2x32bits, 3x16bits, 4 con decodificación de señal de cuadratura, etc.
- 23 Interfaces de comunicación (I^2C, USART, SPI, USB, etc)

Como se puede observar este μC está orientado a señales mixtas y una gran cantidad de protocolos de comunicación. Esto es excelente para los propósitos de esta investigación ya que los convertidores D/A se les puede dar el uso de redundancia para verificar señales con el osciloscopio.

Más detalles en la hoja de datos [31].

7.7.3. STM32H755ZI

El *STM32H755ZI* se utilizó con la tarjeta *NUCLEO-H755ZI*.

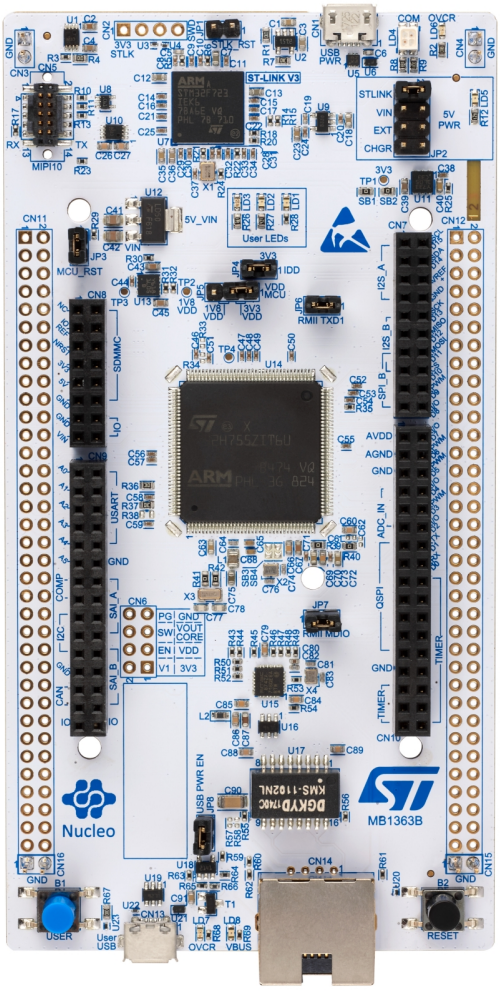


Figura 17: STM32H755ZI en tarjeta *NUCLEO-H755ZI*.

Algunas características relevantes son las siguientes:

- CPU principal a 480 MHz
- CPU secundario a 240 MHz
- Regulador LDO
- DMA
- 11 periféricos analógicos (ADC, DAC, sensor de temperatura, Opamps, etc)
- 22 Timers, 1 con alta resolución (2.1 nanosegundos), RTC, etc.
- 35 Interfaces de comunicación (I^2C, USART, SPI, SAI, CAN, USB, etc)

Como se puede observar este μC está orientado al alto rendimiento, tiene un rápido CPU corriendo a 480MHz y una gran cantidad de protocolos de comunicación, timers y periféricos analógicos. Este μC posee una gran potencia y un 2do CPU.

Más detalles en la hoja de datos [32].

8. Análisis de algoritmo de control

8.1. Estimación de velocidad

La velocidad está definida como el cambio de posición respecto al cambio de tiempo. En el caso del péndulo, las velocidades angulares son calculadas en base al cambio de posición de los codificadores rotatorios; $\dot{\theta} = \frac{\theta[n]-\theta[n-1]}{T_s}$ donde n es la muestra actual de posición y $\dot{\theta}$ es la velocidad angular expresada en pasos por segundo. Para convertir esta velocidad angular a grados por segundo o radianes por segundo es necesario tomar en cuenta los pulsos por revolución del codificador rotatorio.

En el caso del péndulo, para calcular la velocidad se aplica la formula anterior, esta manera de calcular la velocidad tiene una gran desventaja, a bajas velocidades el numerador de la ecuación siempre será un número entero pequeño que al dividirse entre T_s dará como resultado velocidades discretas que varían enormemente entre ellas, las velocidades calculadas serán múltiplos enteros de $\frac{1}{T_s}$.

Este error no es grave a altas velocidades [33] ya que el error relativo se vuelve pequeño conforme se aumenta la velocidad (ver simulación en Fig. 36). El error relativo fue calculado de la siguiente manera: $\epsilon_r = \left|\frac{\dot{\theta}-\dot{\theta_i}}{\dot{\theta_i}}\right| \times 100\,\%$, donde $\dot{\theta}$ es la velocidad calculada por el μC y $\dot{\theta}_i$ es la velocidad medida idealmente (sin errores y con total precisión) que solo puede ser medida en la simulación.

8.1.1. Posibles soluciones

Para reducir este error es posible aumentar el periodo de muestreo para que los múltiplos enteros de velocidad tengan mayor resolución, pero también se vuelve más lento el controlador, de igual manera si se desea un controlador que corrija rápido las perturbaciones se reducirá el periodo de muestreo por lo que el error a bajas velocidades será mayor. También es posible realizar un promedio móvil pero este método reduce la velocidad de respuesta del controlador a cambios repentinos [34].

Este compromiso entre resolución de medición y velocidad de respuesta del controlador representa un problema, ya que se requiere desarrollar alguna otra estrategia de medición que aproveche las capacidades de frecuencia de muestreo de los microcontroladores actuales sin perder precisión al calcular la velocidad.

Una posible solución sería diseñar un observador que estime la velocidad de forma continua. Estimar

la velocidad de forma continua tiene la ventaja de reducir el error relativo a bajas velocidades y evitar "impulsos" en la señal de retroalimentación que generan vibraciones abruptas en el sistema. Este observador podría utilizar información de la corriente y voltaje del motor para ser más preciso. Para medir la velocidad del eje de giro (del péndulo) que no tiene motor, forzosamente se tendrá que diseñar otro observador o algoritmo que estime la velocidad continúa en base a las mediciones discretas del codificador rotatorio.

Otra posible solución es utilizar un supermuestreo que cuente el tiempo entre pulsos de los codificadores rotatorios, esta solución tiene la desventaja de perder resolución a altas velocidades. Se denomina supermuestreo debido a que tiene una frecuencia de muestreo mayor a la utilizada por el ciclo de código del μC, por ejemplo, si el código de estabilización del μC se ejecuta cada 1ms el supermuestreo estará muestreando a 0.1ms, por lo que se harán 10 mediciones por cada vez que se ejecute el código de retroalimentación del péndulo. Otra posible solución podría ser utilizar un algoritmo de medición inteligente que en base a la velocidad decida cuál es la manera óptima de medirla en base a las características del sistema, tales como pulsos por revolución de los codificadores rotatorios, frecuencia máxima y mínima de muestreo, frecuencias máximas y mínimas de los timers, etc. Ya sea medirla convencionalmente con la derivada, con un observador de corriente/voltaje del motor, con estimación continua de velocidad o incluso utilizando varios métodos al mismo tiempo para reducir el error.

8.1.2. Error relativo porcentual

El error relativo porcentual fue calculado con la siguiente ecuación:

$$\epsilon_r = \left| \frac{\dot{\theta} - \dot{\theta}_i}{\dot{\theta}_i} \right| \times 100\,\% \tag{33}$$

Donde $\dot{\theta}$ es la velocidad angular medida o estimada y $\dot{\theta}_i$ es la velocidad angular ideal, es decir, el valor real.

Ver ejemplo en Fig. 37.

8.1.3. Medidor simple

El medidor de velocidad más simple que se puede implementar es el descrito por la siguiente ecuación:

$$\dot{\theta} = \frac{\theta[n] - \theta[n-1]}{T_s} \tag{34}$$

donde n es la muestra actual de posición y θ es la velocidad angular expresada en pasos por segundo.

Esta medición es sencilla de implementar, simplemente se necesita medir la posición actual, restarle la anterior y multiplicarla por la frecuencia de muestreo. La posición empieza por defecto en $2^{15} = 32768$. Esto es para evitar problemas de desbordamiento ya que los timers son de 16 bits. El codificador rotatorio da 4000 pasos por revolución (*ppr*). Para el motor *Pololu* una revolución equivale a 4480, para el motor *MAXON* son 4000. Esta es la razón por la cual en las líneas 28 y 30 del código del anexo 12.8 las posiciones del motor y del codificador rotatorio se dividen por 652 y 637 respectivamente. En otras palabras, dividir entre estas cantidades convierte las unidades de posición de pasos a radianes. Dichas cantidades se obtienen dividiendo $\frac{ppr}{2\pi}$.

8.1.4. Medidor simple con promedio móvil

Como ya se mencionó en la sección 8.1, las velocidades necesariamente deberán ser múltiplos enteros de $\frac{1}{T_s}$ (ver Ec. 35).

$$\dot{\theta} = \frac{\theta[n] - \theta[n-1]}{T_s} \tag{35}$$

Resulta conveniente implementar una forma de medición de velocidad que sea capaz de dar resultados diferentes a múltiplos enteros de $\frac{1}{T_s}$. Esto ayuda a reducir el ruido de la medición ya que si por ejemplo la velocidad real es $\frac{0,5}{T_s}$ la medición será una señal con valor igual a 1 el 50 % del tiempo y 0 el otro 50 %. Es posible implementar un promedio móvil de la forma:

$$P = \frac{1}{N} \sum_{n=0}^{N-1} x[-n] \tag{36}$$

donde P es el promedio, N es el número de muestras a promediar y $x[0]$ es la muestra actual, $x[-1]$ la muestra anterior. Se sigue la misma lógica hasta llegar a la muestra $N-1$.

Cada T_s segundos se realiza la operación de la ecuación 36. El código que realiza este promedio se puede encontrar en el anexo 12.9.

8.1.5. Filtro Kalman Discreto

En algunos casos teniendo conocimiento de las ecuaciones que describen la dinámica del sistema que se desea medir es posible estimar sus estados aún cuando la información proporcionada por los sensores no es 100 % confiable o no es observable. Para el caso particular del péndulo rotatorio resulta útil utilizar el filtro Kalman para estimar las velocidades ya que el método de la subsección anterior tiene un mal rendimiento en bajas velocidades por razones explicadas en la sección 8.6.3. Este filtro se encarga de estimar el estado $\hat{\boldsymbol{x}}[n]$ minimizando óptimamente el error de estimación [35].

Las ecuaciones (en espacio de estados) del filtro son las siguientes:

$$\hat{\boldsymbol{x}}[n] = \hat{\boldsymbol{A}}\hat{\boldsymbol{x}}[n-1] + \hat{\boldsymbol{B}}\boldsymbol{u}[n] \tag{37}$$

$$\hat{\boldsymbol{y}}[n] = \hat{\boldsymbol{C}}\hat{\boldsymbol{x}}[n] \tag{38}$$

Donde:

$\hat{\boldsymbol{x}}$ = Estado estimado

$\hat{\boldsymbol{A}}$ = Matriz del filtro Kalman

$\hat{\boldsymbol{B}}$ = Matriz de entrada del filtro Kalman

$\hat{\boldsymbol{C}}$ = Matriz de observabilidad

$\boldsymbol{u}$ = Entrada

n = Iteración actual

La obtención de las matrices Kalman mencionadas se discutirá en la sección 9 de implementación.

8.1.6. Código del cliente (μC)

El cliente es el 1er μC. El código se ejecuta en un μC STM32G474RE a 170MHz. El código se ejecuta mediante una interrupción generada por un timer cada T_s.

Las posiciones se miden con codificadores rotatorios incrementales que generan señales cuadradas y las velocidades se calculan derivando las posiciones.

Las posiciones se miden vía hardware con los timers del μC y la posición solo se consulta en el código, de manera que no se consumen ciclos del cpu para realizar la medición.

En base a las mediciones se calcula la señal de retroalimentación. La señal de retroalimentación se convierte a un voltaje PWM que alimenta al motor con ayuda de una driver. Esta señal PWM también es generada por un timer.

El código anexo 12.7 tiene los comentarios debajo de cada renglón para facilidad de lectura.

8.1.7. Código de anfitrión

El anfitrión es la PC.

La PC se utiliza para monitorear en tiempo real mediante una gráfica el comportamiento del controlador del péndulo. También se queda guardada la información para su posterior procesamiento y análisis. Cabe mencionar que el anfitrión (PC) no participa en el control, solo lo monitorea. El μC puede funcionar autónomamente y estabilizar el péndulo sin necesidad de un dispositivo anfitrión.

En algunos experimentos con el *Hardware-in-the-loop* la PC además de monitorear también simula la dinámica del péndulo rotatorio. Más información en la sección 8.6.2.

El código anexo 12.6 tiene los comentarios debajo de cada renglón para facilidad de lectura.

8.2. Algoritmos de control

8.2.1. LQR

El regulador cuadrático lineal, con siglas *LQR* en inglés, tiene el objetivo de estabilizar óptimamente el péndulo rotatorio en la posición deseada por el usuario, usualmente $\boldsymbol{x} = [0\ 0\ 0\ 0]$. Estabilización óptima se refiere a llegar desde el estado $\boldsymbol{x_0}$ hasta el estado $\boldsymbol{x_{REF}}$ al menor costo posible. El costo se define con la siguiente función:

$$J_c = \int_0^\infty (\boldsymbol{x^T Q x} + \boldsymbol{u^T R u})\ dt \tag{39}$$

donde $\boldsymbol{u{=}{-}Kx}$, $\boldsymbol{Q}$ pondera el costo de error de los estados y $\boldsymbol{R}$ pondera el costo de actuación de la señal de control. En otras palabras, si $\boldsymbol{Q} < \boldsymbol{R}$ el costo de actuación es mayor que el del error, un ejemplo de esta situación sería un dron con limitado combustible donde es tolerable un poco de error para ahorrar combustible. Un ejemplo del caso opuesto ($\boldsymbol{Q} > \boldsymbol{R}$) puede ser un dispositivo médico crítico donde el error es intolerable sin importar el costo de actuación. Para el caso específico del péndulo se utilizó $\boldsymbol{Q{=}R{=}I}$ donde $\boldsymbol{I}$ es la matriz de identidad.

Para obtener la ganancia óptima de control, $\boldsymbol{K}$, es necesario resolver para $\boldsymbol{S}$ en la ecuación algebraica de Ricatti [36]:

$$0 = \boldsymbol{A^T S{+}SA{-}SBR^{-1}B^T S{+}Q} \tag{40}$$

Después se tiene:

$$\boldsymbol{R^{-1}B^T Sx = Kx} \tag{41}$$

Resolviendo para $\boldsymbol{K}$ se obtiene:

$$\boldsymbol{K} = [-0{,}3162\ \ -0{,}3777\ \ -6{,}8057\ \ -0{,}6340] \tag{42}$$

También es posible obtener la matriz $\boldsymbol{K}$ utilizando *MATLAB* con el comando `lqr(SYS,Q,R)`.

Retomando las ecuaciones 17 y 18 se tiene:

$$\boldsymbol{A} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{\delta\gamma}{\alpha\beta-\gamma^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{\alpha\delta}{\alpha\beta-\gamma^2} & 0 \end{pmatrix}$$

$$\boldsymbol{B} = \begin{pmatrix} 0 & 0 \\ \frac{\beta}{\alpha\beta-\gamma^2} & -\frac{\gamma}{\alpha\beta-\gamma^2} \\ 0 & 0 \\ -\frac{\gamma}{\alpha\beta-\gamma^2} & \frac{\alpha}{\alpha\beta-\gamma^2} \end{pmatrix}$$

Sustituyendo las constantes algebraicas por los parámetros del péndulo se tiene:

$$\boldsymbol{A} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -19{,}1726 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 181{,}6485 & 0 \end{pmatrix} \tag{43}$$

Y tomando en cuenta que solo se actúa τ_ϕ, se elimina la 2da columna de $\boldsymbol{B}$ de manera que:

$$\boldsymbol{B} = \begin{pmatrix} 0 \\ 1{,}1347 \\ 0 \\ -1{,}7385 \end{pmatrix} \tag{44}$$

$$\boldsymbol{C} = \boldsymbol{I} \tag{45}$$

$$\boldsymbol{D} = [0\ 0\ 0\ 0]^T \tag{46}$$

Si se comprueba la estabilidad del sistema se tiene:

$$\boldsymbol{G} = \boldsymbol{C}(s\boldsymbol{I} - \boldsymbol{A})^{-1}\boldsymbol{B} + \boldsymbol{D} \tag{47}$$

Donde $\boldsymbol{G}$ es la función de transferencia del sistema:

$$\boldsymbol{G} = \begin{pmatrix} \frac{69112851-453860s^2}{-20s^2(20s^2-3633)} \\ \frac{69112851-453860s^2}{-20s(20s^2-3633)} \\ \frac{-34770}{20s^2-3633} \\ \frac{-34770s}{20s^2-3633} \end{pmatrix} \tag{48}$$

Observando los denominadores de las 4 funciones de transferencia y utilizando la fórmula general se puede determinar que los polos son [0 0 13,48 $-$ 13,48]. Para evitar este tedioso procedimiento se puede utilizar la función `eig(A)` *(eigenvalues)* de *MATLAB*.

Este sistema es inestable debido a los polos positivos. Esto es esperado ya que aún no se le aplica el filtro **LQR**. Para aplicar (matemáticamente) el controlador **LQR** basta con modificar la matriz $\boldsymbol{A}$ para que incluya la retroalimentación con la ganancia $\boldsymbol{K}$:

$$\boldsymbol{A_{LQR}} = \boldsymbol{A} - \boldsymbol{BK} \tag{49}$$

Obteniendo:

$$\boldsymbol{A_{LQR}} = \begin{pmatrix} 0 & 1{,}0 & 0 & 0 \\ 358{,}8 & 428{,}5 & 7703{,}0 & 719{,}4 \\ 0 & 0 & 0 & 1{,}0 \\ -549{,}8 & -656{,}6 & -11650{,}0 & -1102{,}0 \end{pmatrix} \tag{50}$$

Ahora, obteniendo los polos con el comando `eig(A_lqr)` se obtienen los polos:

$$\begin{bmatrix} -656{,}7 \\ -1{,}0 \\ -8{,}021 + 4{,}342i \\ -8{,}021 - 4{,}342i \end{bmatrix} \tag{51}$$

Como se puede observar, ahora el péndulo rotatorio es matemáticamente estable ya que todos los polos son negativos. Por lo tanto, si el péndulo construido y el controlador implementado tienen un comportamiento similar al descrito por el modelo matemático el péndulo debe estabilizarse.

Ahora, discretizando $\boldsymbol{A}$,$\boldsymbol{B}$,$\boldsymbol{C}$ y $\boldsymbol{D}$ para $T_s = 100\ \mu s$ utilizando las ecuaciones 19 y 20 se obtiene:

$$\boldsymbol{A_d} = \begin{pmatrix} 1 & 0{,}0001 & -9{,}586\text{e}{-8} & -3{,}195\text{e}{-12} \\ 0 & 1 & -0{,}001917 & -9{,}586\text{e}{-8} \\ 0 & 0 & 1 & 0{,}0001 \\ 0 & 0 & 0{,}01816 & 1 \end{pmatrix} \tag{52}$$

$$\boldsymbol{B_d} = \begin{pmatrix} 5{,}673\text{e}{-6} \\ 0{,}1135 \\ -8{,}692\text{e}{-6} \\ -0{,}1738 \end{pmatrix} \tag{53}$$

$$\boldsymbol{C_d} = \boldsymbol{C} \tag{54}$$

$$\boldsymbol{D_d} = \boldsymbol{D} \tag{55}$$

Discretizando también $\boldsymbol{K}$ se obtiene $\boldsymbol{K_d}$:

$$\boldsymbol{K_d} = [-0{,}3059 \ -0{,}3653 \ -6{,}5878 \ -0{,}6136] \tag{56}$$

De forma análoga al procedimiento en dominio continuo se obtienen los polos discretos de $\boldsymbol{A_d}$:

$$\begin{bmatrix} 1{,}0000 \\ 1{,}0000 \\ 1{,}0013 \\ 0{,}9987 \end{bmatrix} \tag{57}$$

Recordando que en el dominio $\mathcal{Z}$ la región de convergencia es todo valor dentro del círculo de radio 1 con centro en el origen se puede observar que este sistema es inestable. Este de igual forma es esperado ya que $\boldsymbol{A_d}$ no está retroalimentado con el controlador **LQR**.

$$\boldsymbol{A_{d_{LQR}}} = \boldsymbol{A_d} - \boldsymbol{B_d}\boldsymbol{K_d} \tag{58}$$

$$\boldsymbol{A_{d_{LQR}}} = \begin{pmatrix} 1{,}0000 & 0{,}0001 & 0{,}0000 & 0{,}0000 \\ 0{,}0347 & 1{,}0414 & 0{,}7454 & 0{,}0696 \\ -0{,}0000 & -0{,}0000 & 0{,}9999 & 0{,}0001 \\ -0{,}0532 & -0{,}0635 & -1{,}1269 & 0{,}8933 \end{pmatrix} \tag{59}$$

Con polos en:

$$\begin{bmatrix} 0{,}9364 \\ 0{,}9999 \\ 0{,}9992 + 0{,}0004339i \\ 0{,}9992 - 0{,}0004339i \end{bmatrix} \tag{60}$$

Como se puede observar los polos están dentro de la región de convergencia por lo que el sistema es estable.

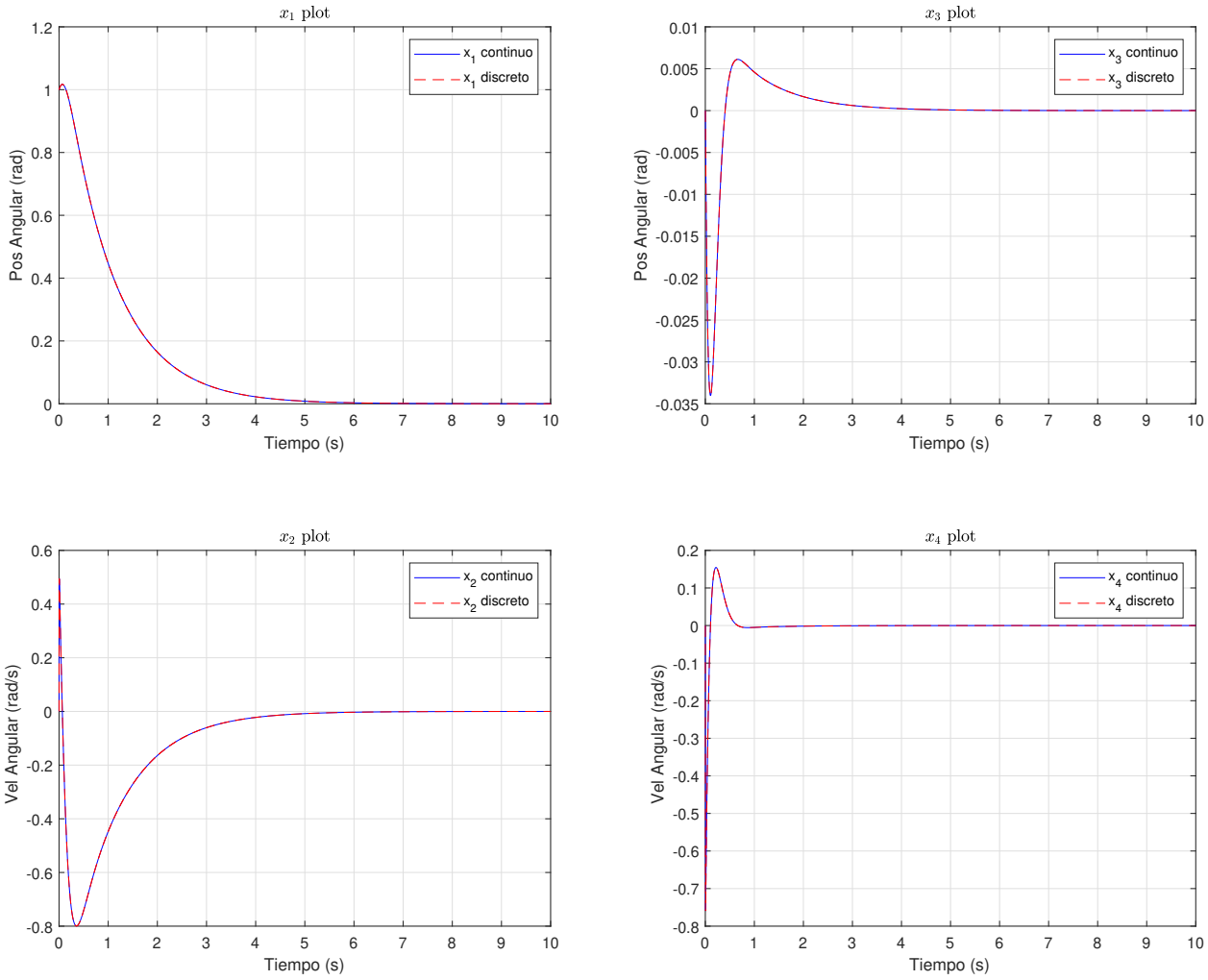


Figura 18: Comparación entre simulación de espacio de estados continuo y discreto.

En la figura 18 se simuló en *Simulink* el comportamiento con respecto al tiempo del péndulo rotatorio en espacio de estados en dominio continuo y discreto. El estado inicial es $\boldsymbol{x_0} = [1\ 0\ 0\ 0]$ y se busca llevar el sistema al estado $\boldsymbol{x_{REF}} = [0\ 0\ 0\ 0]$.

Como se puede observar el comportamiento es idéntico. El sistema continuo se simula en base a las matrices $\boldsymbol{A_{LQR}}$,$\boldsymbol{B}$,$\boldsymbol{C}$ y $\boldsymbol{D}$ (ecuaciones 50, 44, 45 y 46) con el vector de retroalimentación $\boldsymbol{K}$ (ecuación 56).

El sistema discreto se simula en base a las matrices $\boldsymbol{A_{d_{LQR}}}$,$\boldsymbol{B_d}$,$\boldsymbol{C_d}$ y $\boldsymbol{D_d}$ (ecuaciones 59, 53, 54 y 55) con el vector de retroalimentación $\boldsymbol{K_d}$ (ecuación 56).

8.3. Modelo 3D del péndulo rotatorio

8.3.1. Modelo en *SolidWorks*

El modelo 3D fue diseñado y ensamblado en *SolidWorks*.

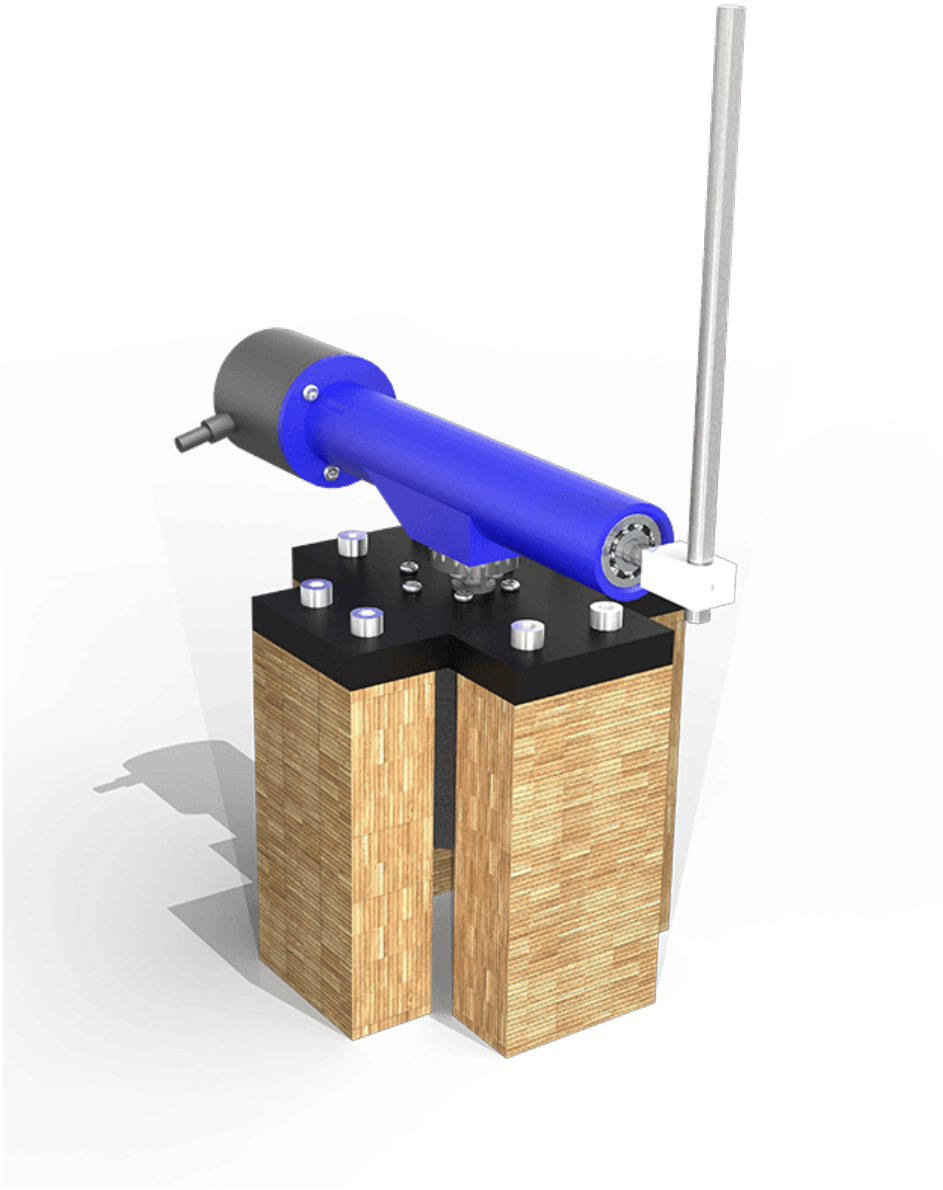


Figura 19: Modelo 3D del péndulo rotatorio.

8.3.2. Modelo Real

El modelo real fue impreso en 3D en plástico PLA con la excepción de algunas piezas fáciles de adquirir como los ejes metálicos o algunas piezas cruciales como los baleros y el conector que une el cuerpo del péndulo con el eje del motor.

Figura 20: Modelo real del péndulo rotatorio V4.

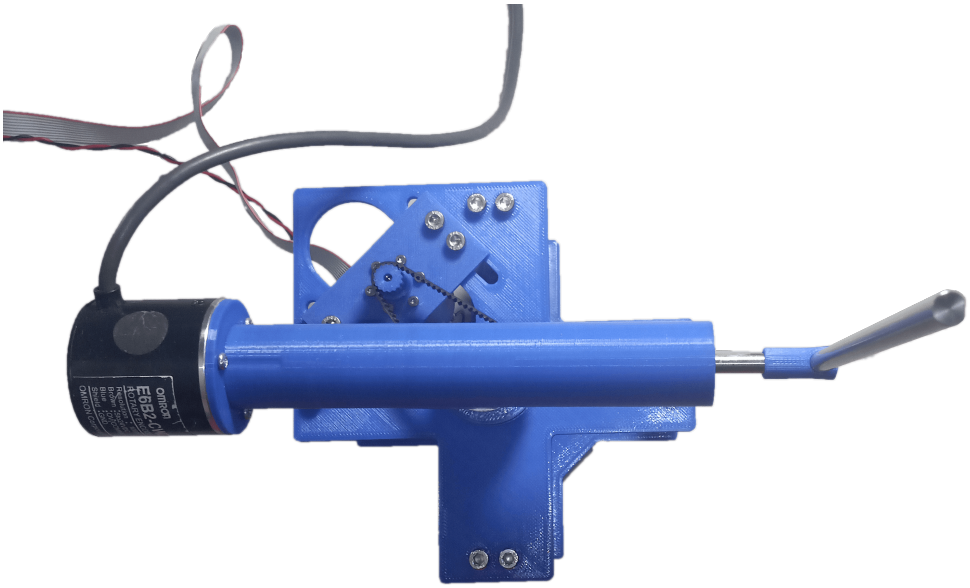


Figura 21: Modelo real del péndulo rotatorio V5.

Las versiones 1 a 3 no se muestran en este documento debido a que no tuvieron un rendimiento satisfactorio. Simplemente sirvieron como punto de partida para construir las versiones posteriores.

8.4. Modelo en *Simulink* del péndulo rotatorio

En teoría si se realizan mediciones del péndulo rotatorio real y se varían los parámetros de fricción de la simulación se pueden caracterizar las 4 fricciones, las 2 del eje ϕ y las 2 del eje θ.

En la práctica no es posible medir de forma exitosa las contribuciones de las fricciones del eje ϕ. Esto se debe a fenómenos mecánicos no modelados y altamente aleatorios provocados por el cable del codificador rotatorio.

Modelando solo la fricción en θ se obtiene el siguiente diagrama de bloques.

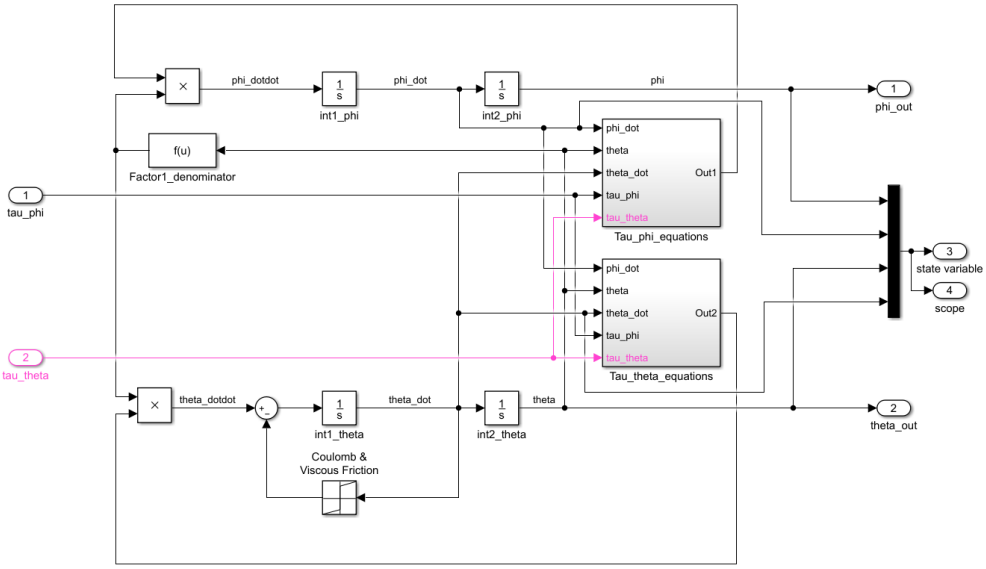


Figura 22: Modelo a bloques del péndulo rotatorio completo.

Observando la gráfica del ángulo θ respecto al tiempo se puede comprobar que los parámetros fricción viscosa y fricción de Coulomb obtenidos en la caracterización realizada del eje θ en el punto 9.3.2.2 fueron implementados correctamente en el péndulo rotatorio completo.

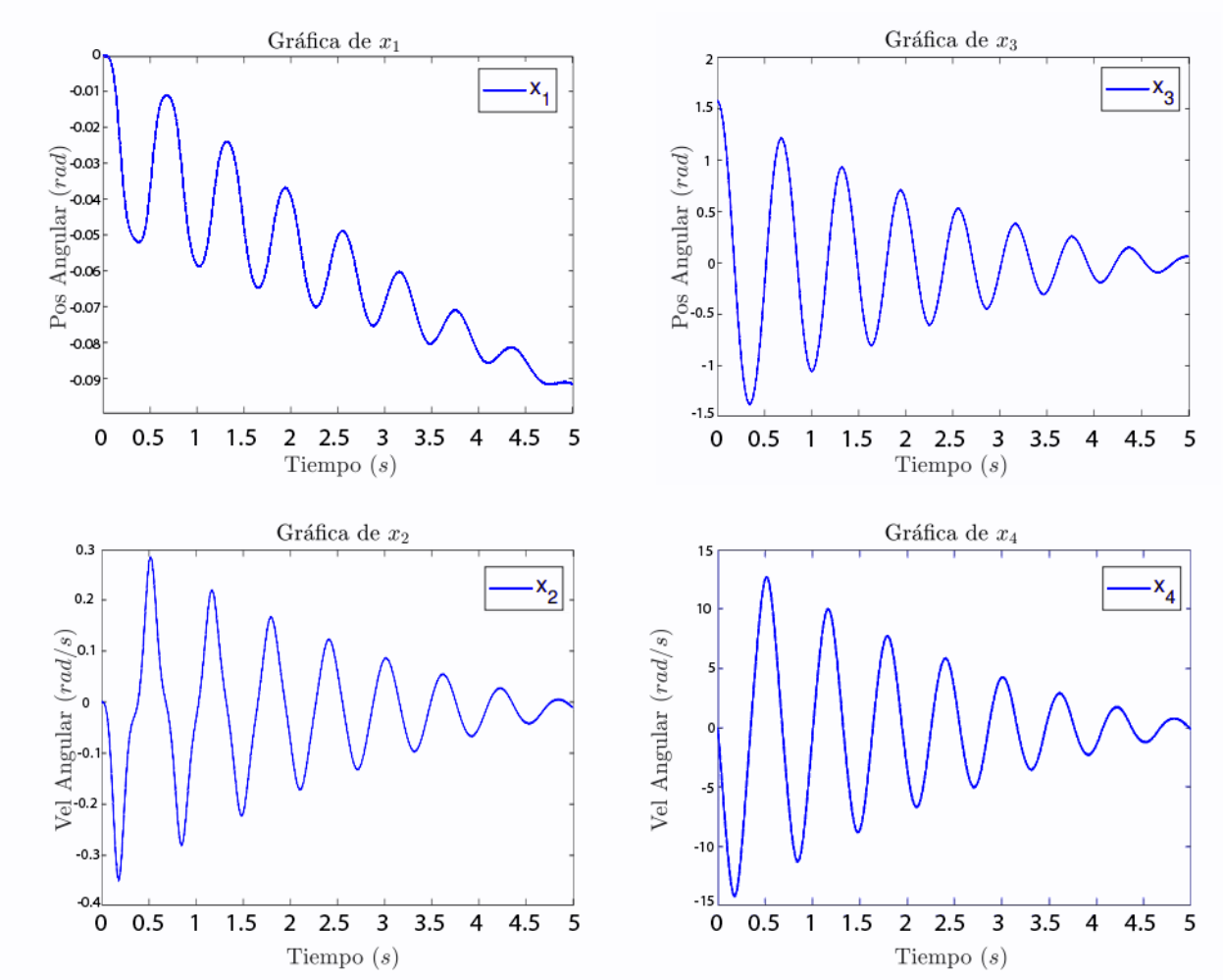


Figura 23: Respuesta en el tiempo del péndulo rotatorio con condiciones iniciales $\theta = \pi/2$.

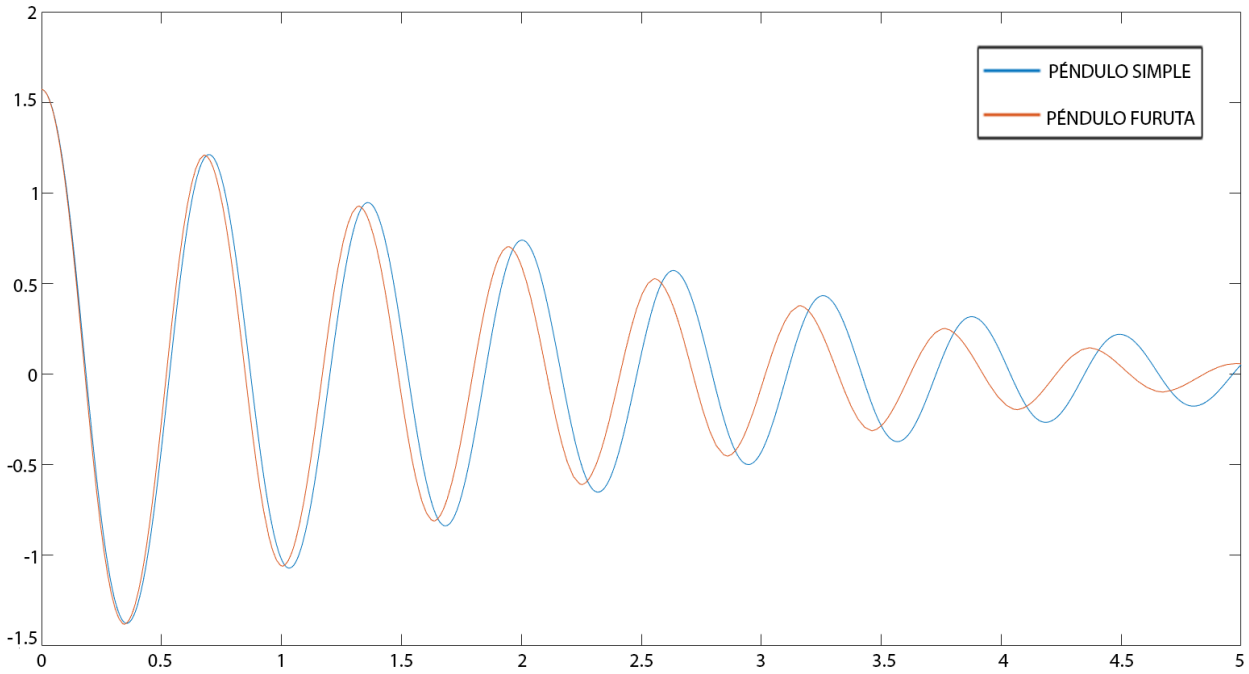


Figura 24: Respuesta en el tiempo del péndulo rotatorio comparado con el péndulo simple. Ambos con condiciones iniciales $\theta = \pi/2$.

Conclusión: Se puede observar que el péndulo simple y el péndulo rotatorio tienen un comportamiento similar con las fricciones modeladas. La diferencia entre las gráficas se debe a que el péndulo rotatorio disipa energía en forma de movimiento del eje ϕ, por eso se detiene antes.

8.5. Fenómenos no modelados

Además del comportamiento matemático ya establecido en el punto 7.1, existen fenómenos no lineales derivados de la construcción imperfecta del péndulo y de la no linealidad del motor. Estos fenómenos pueden alterar en gran forma el comportamiento del péndulo y provocar un comportamiento no previsto.

Algunos de estos fenómenos son los siguientes:

- Deslizamiento de la correa de transmisión
- Precesión de los ejes
- Cable del codificador rotatorio

8.5.1. Backlash

El *backlash* de los engranajes, también llamado *juego*, es un fenómeno no lineal provocado por el espacio entre los dientes de los engranajes, este espacio genera una zona muerta en la respuesta mecánica del sistema, produce componentes de alta frecuencia debido a los pequeños impactos entre dientes y contribuye negativamente al control de posición y velocidad del sistema.

"Backlash is the most important non-linearity that limits the performance of speed control in industrial drives and is an impediment to position control as well."[37]

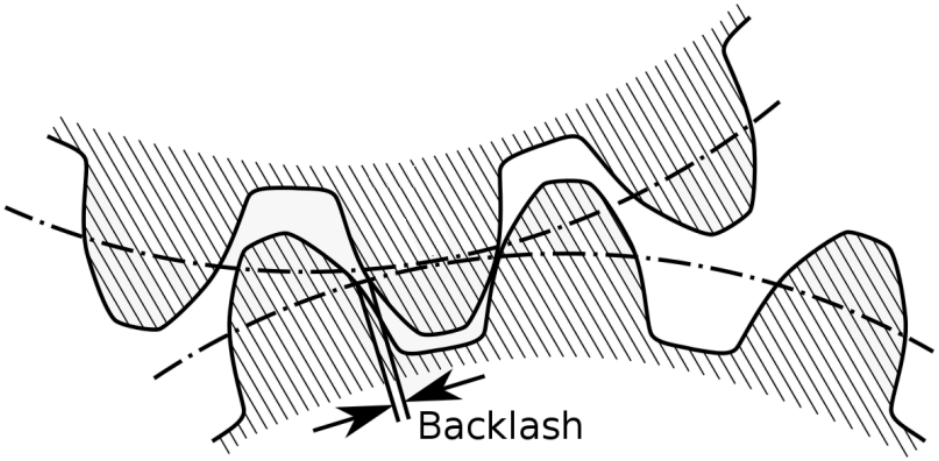


Figura 25: Juego entre engranajes, también llamado *backlash*.

8.5.1.1. Caracterización del backlash

Para la caracterización del backlash se utilizó un láser conectado al eje de la salida de la caja de engranajes a una distancia de 240 *cm* de una pared y se midió el cateto opuesto del triángulo formado al mover los engranajes desde su punto mínimo de backlash hasta el máximo.

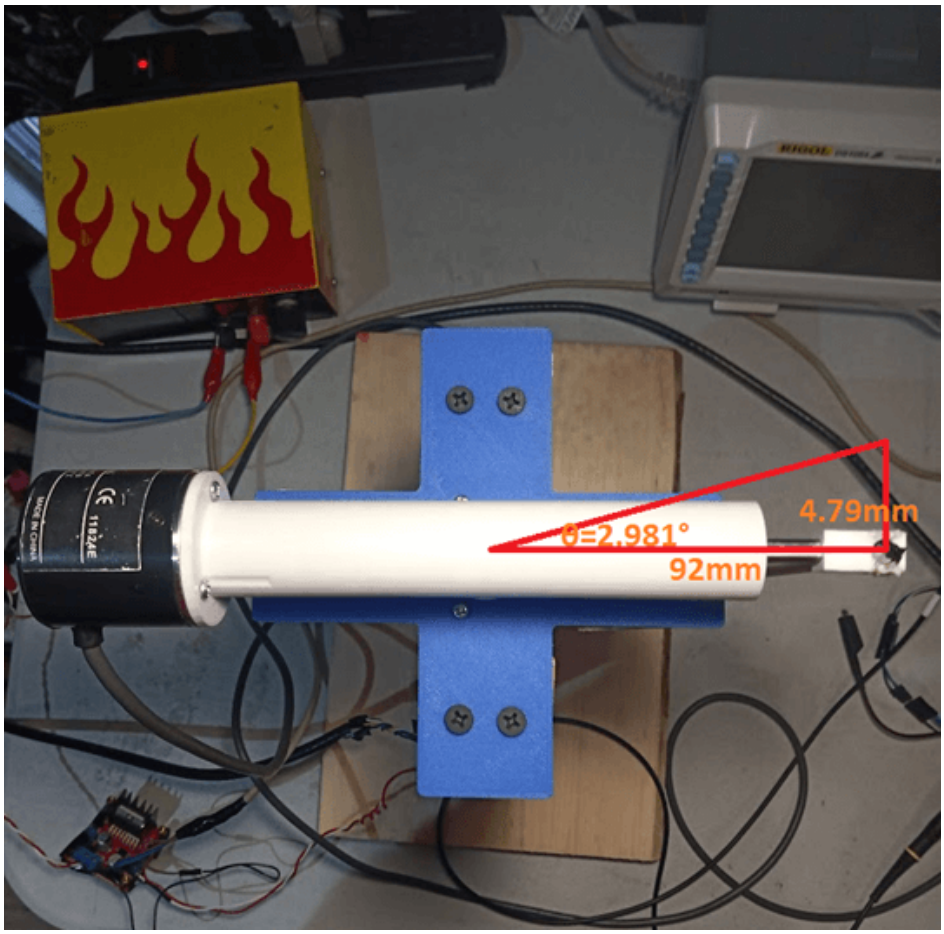


Figura 26: Caracterización del backlash. Péndulo V4.

El resultado del experimento fue un ángulo de 2,981° ≈ 3° de backlash. Aunque parece poco, 3° son suficientes para producir un comportamiento inestable oscilatorio en un sistema que por el contrario tendría un comportamiento suave y estable, tal y como la simulación muestra.

8.5.2. Soluciones a los Fenómenos no modelados

8.5.2.1. Reducción de backlash utilizando DSP

Es posible atenuar el backlash utilizando procesamiento digital de señales; abreviado **DSP**, en inglés, aplicando un filtro rechaza bandas que atenúe las frecuencias de resonancia del backlash, para identificarlas se realizó un experimento.

El experimento consistió en dejar que el controlador intente (sin éxito) estabilizar el péndulo en su posición vertical. Debido a los efectos del backlash el controlador osciló alrededor del valor deseado. La señal en el tiempo no es relevante por el momento ya que el interés es identificar la frecuencia de resonancia del backlash.

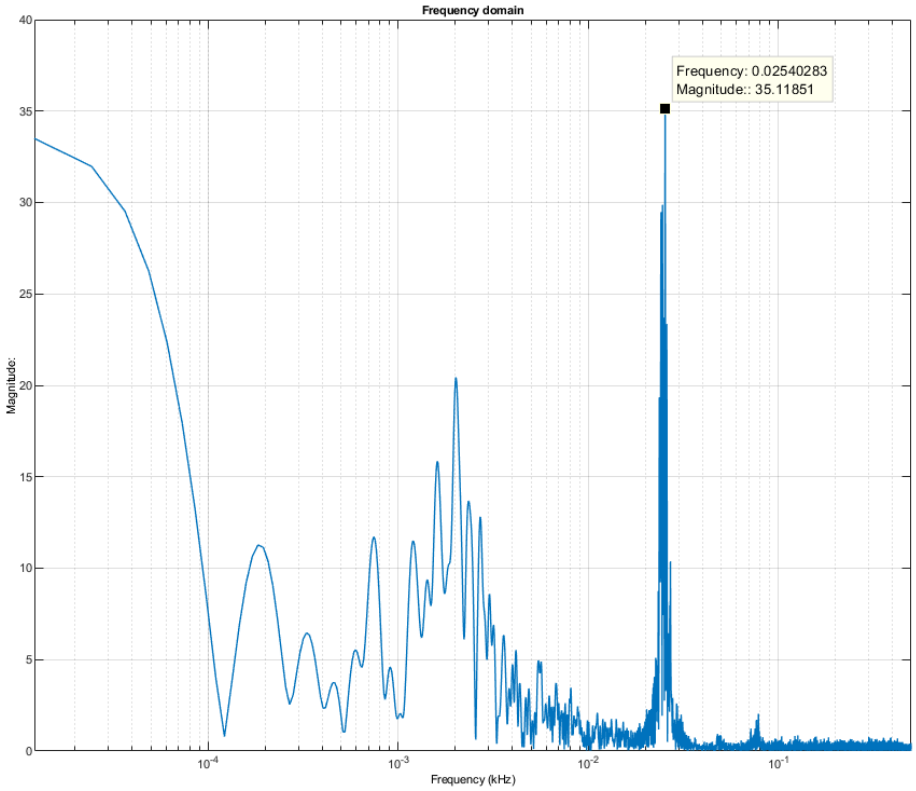


Figura 27: Espectro de frecuencias del experimento del péndulo.

Como se puede observar en la figura 27, la frecuencia de resonancia del backlash es de alrededor de 25 Hz

Para atenuar la resonancia del backlash y el controlador se implementó un filtro de respuesta finita. El filtro es de orden 321 por lo que se espera un retraso respecto a la entrada de 160 ms, si bien no es un retraso tan grande; si es un retraso considerable para un sistema como el péndulo rotatorio.

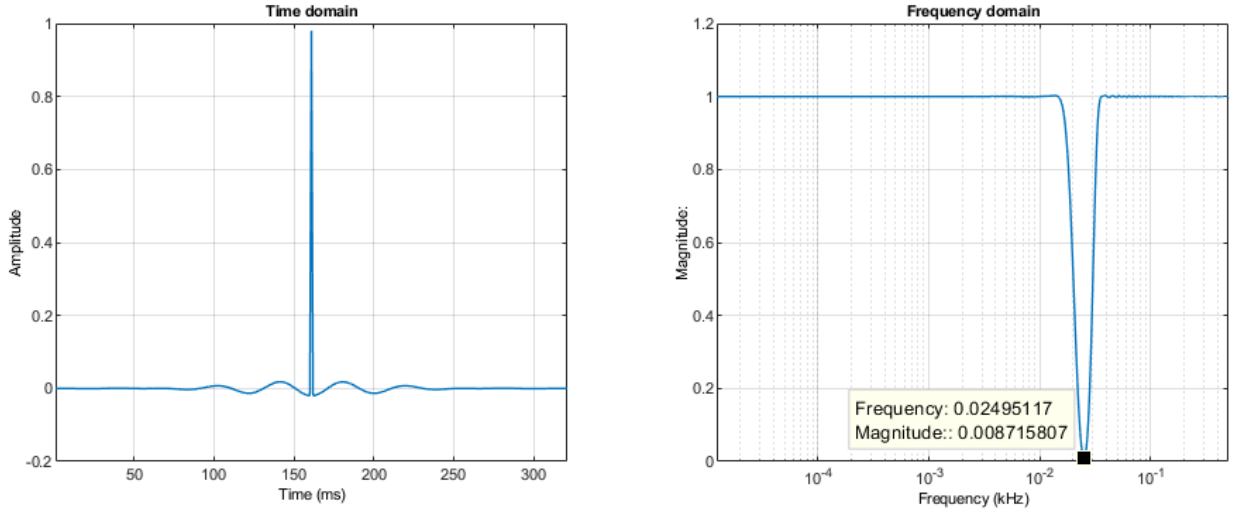


Figura 28: Filtro rechaza backlash.

Al aplicar el filtro a la señal del experimento anterior se observa que el backlash es atenuado prácticamente al 100 %. El pico a 24 Hz en la señal azul es inexistente en la señal naranja.

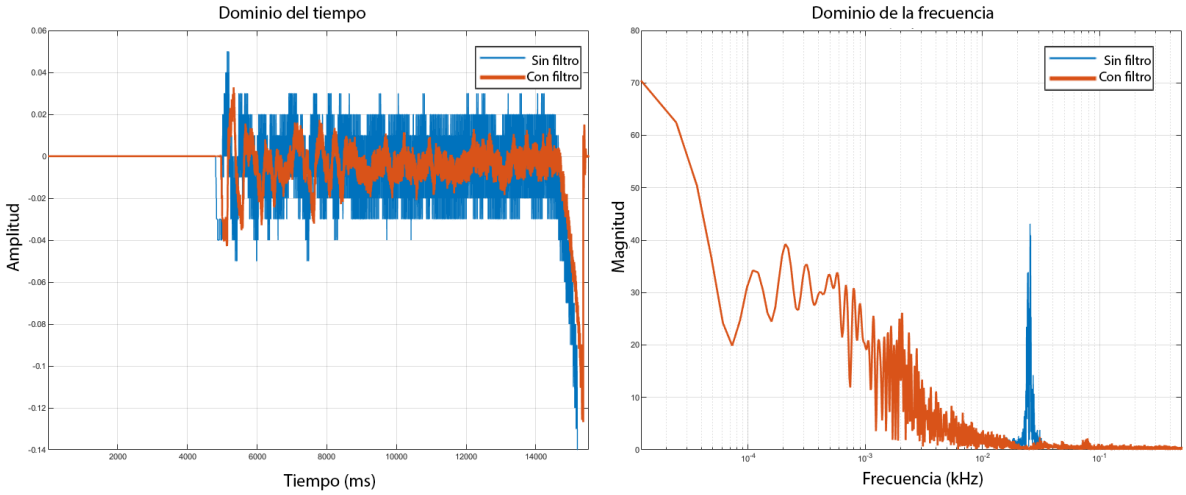


Figura 29: Comparación con y sin filtro rechaza backlash.

Si bien matemáticamente el backlash se atenúa casi por completo, en la práctica al implementar el filtro en el μC aún existen fenómenos transitorios de baja frecuencia del backlash, tales como el rebote que siempre ocurre al iniciar y al terminar un movimiento.

En conclusión, para este método de atenuación si se logró atenuar exitosamente el backlash y se mejoró el comportamiento del controlador aunque aún hay espacio para mejoras en el tiempo de respuesta del controlador y efectos transitorios del backlash.

8.5.2.2. Reducción de backlash re-diseñando péndulo

Debido a las complicaciones producidas por el backlash de los engranajes y los fenómenos (aparentemente) aleatorios que estos generaban en versiones anteriores del péndulo, se decidió por diseñar una nueva versión del péndulo mejorada que no requiera de engranajes.

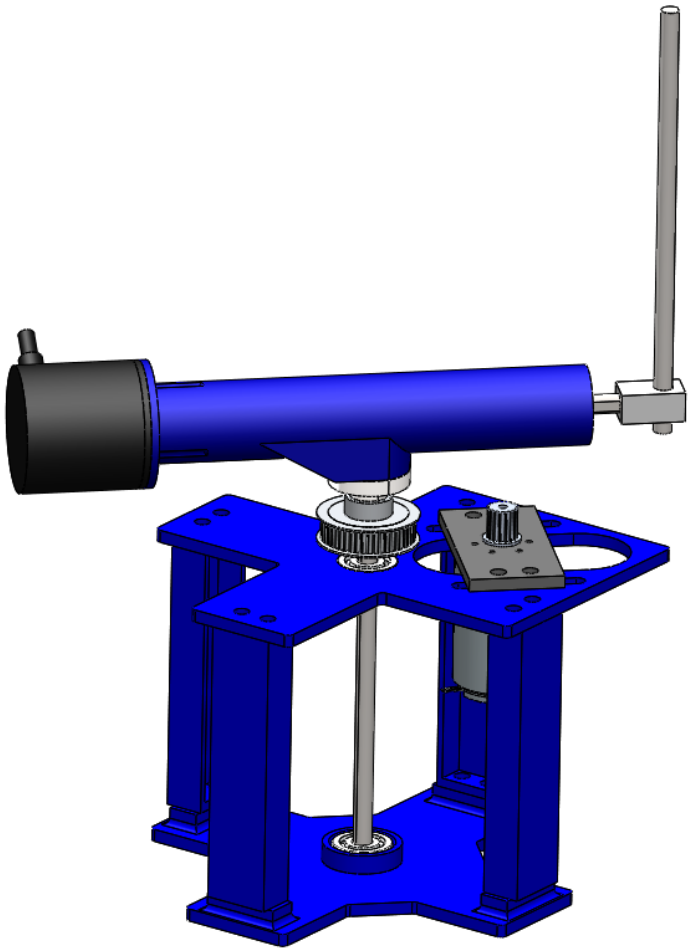


Figura 30: Péndulo rotatorio V5.

Este nuevo péndulo elimina el backlash por completo utilizando una banda o correa de transmisión 2GT-6 con una relación 50:16. El nuevo modelo está diseñado para poder ser ajustado en caso de ser necesario. También se utiliza un eje ϕ largo con baleros al comienzo y al final para eliminar problemas de precesión del eje existentes en las versiones anteriores.

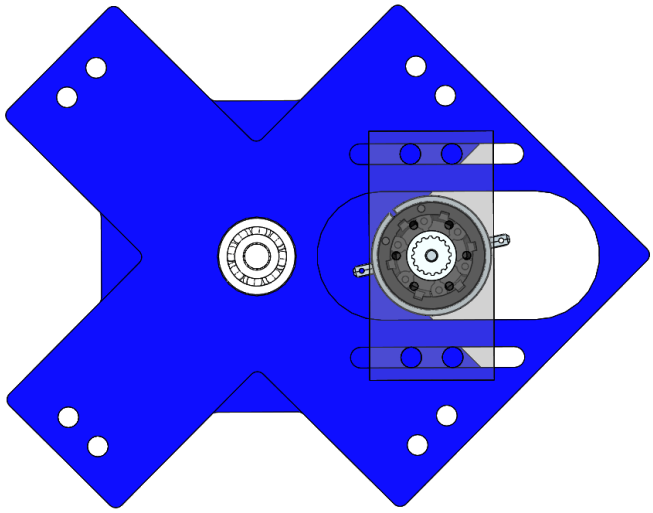


Figura 31: Péndulo rotatorio V5, vista desde arriba.

En conclusión el resultado final fue satisfactorio y se logró eliminar por completo el backlash de manera que ya es posible profundizar en temas relacionados al control, vibraciones inducidas por el controlador y otros tipos de fenómenos relacionados a la medición y no a problemas por un mal diseño mecánico.

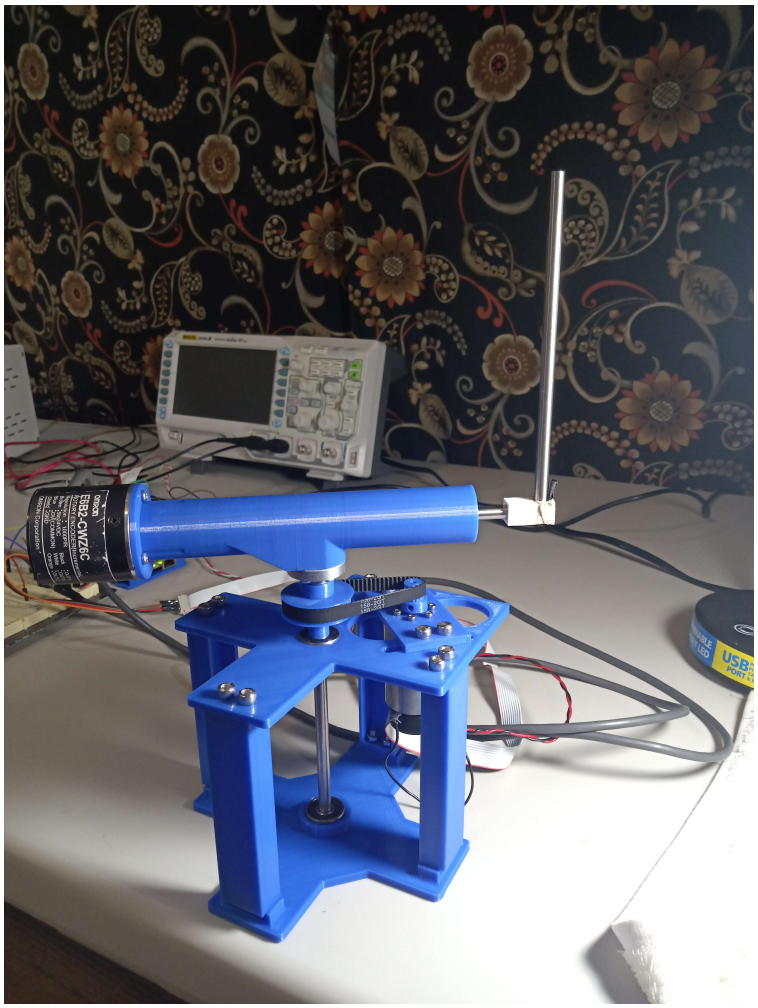


Figura 32: Péndulo rotatorio V5, modelo real.

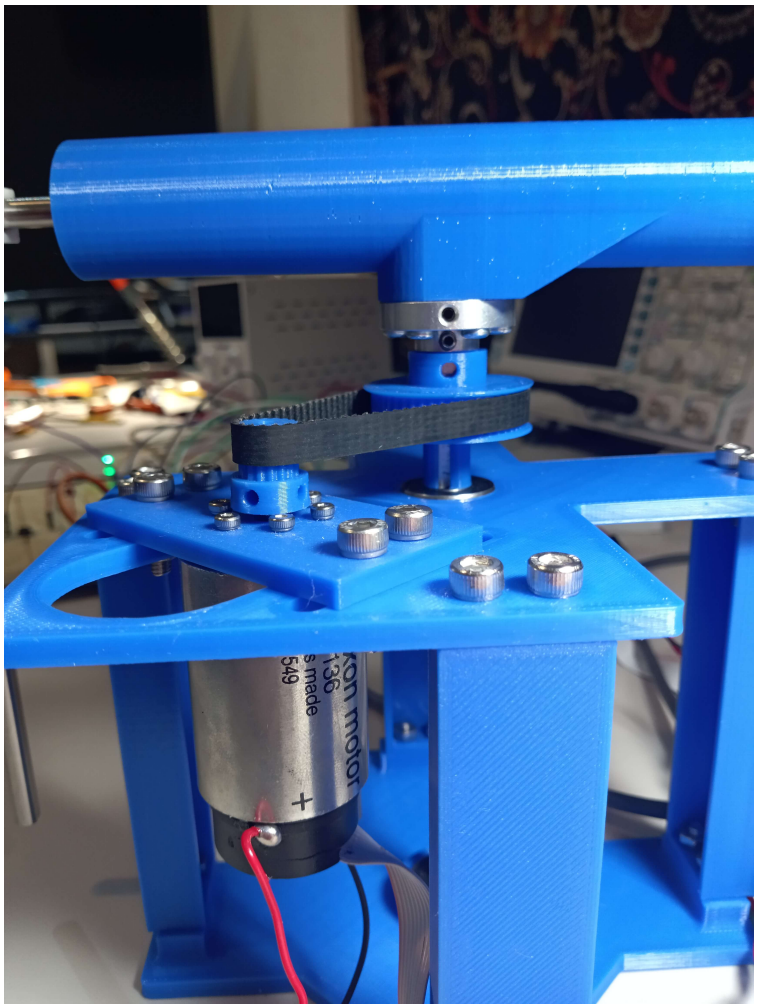


Figura 33: Péndulo rotatorio V5 real, acercamiento al mecanismo de transferencia de torque.

8.6. Simulaciones

8.6.1. Aproximación de la solución del péndulo usando RK4

Para resolver las ecuaciones del péndulo se utilizó el método **RK4**. El método **RK4** calcula 4 pendientes en diferentes instantes (dentro del intervalo del paso) cada paso. Después, las 4 pendientes son ponderadas y promediadas.

El péndulo rotatorio puede ser modelado matemáticamente usando un sistema de ecuaciones que consta de 4 ecuaciones diferenciales. Cada ecuación diferencial es solucionada utilizando el método **RK4**. Esto requiere un total de 20 operaciones matemáticas; 16 operaciones cada paso y 4 promedios.

El sistema de ecuaciones (simplificado) del péndulo que será solucionado por el método **RK4** es el siguiente:

$$\dot{x}_1 = \frac{d}{dt}\phi = \dot{\phi} \tag{61}$$

$$\dot{x}_2 = \frac{d}{dt}\dot{\phi} = f_{x_2}(\dot{\phi}, \theta, \dot{\theta}, \tau_\phi, \tau_\theta) \tag{62}$$

$$\dot{x}_3 = \frac{d}{dt}\theta = \dot{\theta} \tag{63}$$

$$\dot{x}_4 = \frac{d}{dt}\dot{\theta} = f_{x_4}(\dot{\phi}, \theta, \dot{\theta}, \tau_\phi, \tau_\theta) \tag{64}$$

Donde ϕ y θ representan las posiciones angulares, τ_ϕ y τ_θ representan los torques de los ejes de rotación y f_ϕ y f_θ representan la dinámica de los ejes del péndulo rotatorio.

Cada n iteración cada ecuación diferencial es solucionada utilizando el método **RK4**:

$$y_{n+1} = y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4) \tag{65}$$

Donde:

$$k_1 = f(t_n, y_n) \tag{66}$$

$$k_2 = f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1 h) \tag{67}$$

$$k_3 = f(t_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2 h) \tag{68}$$

$$k_4 = f(t_n + h, y_n + k_3 h) \tag{69}$$

NOTA: f puede ser f_{x_2} o f_{x_4} dependiendo de cual **ODE** se esté solucionando.

Después de calcular las 16 pendientes, se calculan los 4 promedios usando la ecuación 65. Resulta útil cambiar de notación de y_n a x_{s_n} para observar fácilmente a cual ecuación diferencial nos estamos refiriendo:

De forma general se tiene:

$$x_{s_{n+1}} = x_{s_n} + \frac{1}{6}h(k_{1s} + 2k_{2s} + 2k_{3s} + k_{4s}) \tag{70}$$

Donde s representa que ecuación diferencial se está resolviendo y n representa la iteración actual. Por ejemplo, $x_{4_{n+1}}$ corresponde a la solución de la 4ta ecuación de estado; la ecuación 64.

8.6.2. Hardware-In-The-Loop

El *Hardware-in-the-loop* (**HIL**) es un método en el que la planta es simulada en tiempo real en un dispositivo de cómputo. Se puede categorizar en diferentes niveles de simulación:

- Nivel señal o información: Solo se simula el comportamiento dinámico de la planta sin tomar en cuenta sus requerimientos energéticos. Por ejemplo, en este nivel un motor simulado de 100 $Watts$ podría ser controlado por un μC sin necesidad de alimentación eléctrica externa. Este nivel es útil para verificar si la implementación del algoritmo de control es correcta.

- Nivel energía: En este nivel se toma en cuenta la conservación de energía de la planta de manera que se requiere circuitería eléctrica que simule y disipe la energía que la planta real consumiría. Siguiendo el ejemplo anterior del motor se requeriría un circuito que disipe la potencia requerida por el motor tal como si fuera el motor real. Este nivel es útil para comprobar si el circuito de alimentación del controlador, puente H o cualquier circuito de potencia de la planta real funciona correctamente.

- Nivel mecánico: En este nivel además de todo lo anterior se simula también la forma de transmitir energía del controlador a la planta. Por ejemplo, si (siguiendo el ejemplo del motor) el motor tendrá una carga de 3 $kg \cdot m^2$ (o alguna carga dinámica) el motor del controlador de la planta se conecta a un freno controlado por el *Hardware-in-the-loop* de manera que el freno aplique una carga mecánica al motor similar a la que la carga real aplicaría. Este nivel es útil para experimentos donde se desee observar fenómenos como por ejemplo el sobrecalentamiento de actuadores, resonancias mecánicas, etc. Este nivel es el último y lo único más preciso que este nivel es un experimento con la planta real.

El *Hardware-in-the-loop*, en esta tesis en particular, simula el péndulo rotatorio. Este (HIL) es implementado a nivel señal y funciona resolviendo numéricamente las ecuaciones diferenciales que describen su movimiento (ecuaciones 9 a 12). Las ecuaciones se resuelven en tiempo real teniendo en cuenta las perturbaciones producidas por el controlador, es decir, la señal de control (ver Fig. 34).

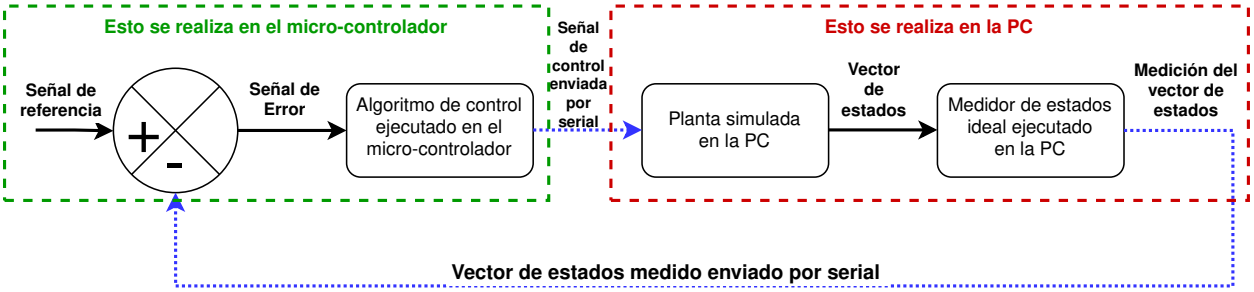


Figura 34: Diagrama a bloques del bucle de retroalimentación del *Hardware-in-the-loop*. Las flechas azules representan comunicación serial.

El solucionador de ecuaciones diferenciales (RK4) fue implementado en *MATLAB* y también en *C++*. La planta *Hardware-in-the-loop* simulada en la PC se comunica vía serial con el μC. El código está disponible en el anexo 12.12.

También se implementó el solucionador **RK4** en un 2do μC utilizando la previa implementación en *C++*. Más información en la sección 9.4.6.

MATLAB[38] es un software y lenguaje de programación utilizado para realizar cálculos numéricos, algoritmos, analizar datos, crear modelos de sistemas, etc. La documentación se encuentra en el libro "MATLAB Primer" [39], así como también en la web.

8.6.2.1. Comparación entre *Simulink* y *Hardware-in-the-loop*

Para validar la implementación del *Hardware-in-the-loop* utilizando **RK4** y la implementación del controlador en el STM32 se realizó un experimento en bucle cerrado. Los resultados fueron los siguientes:

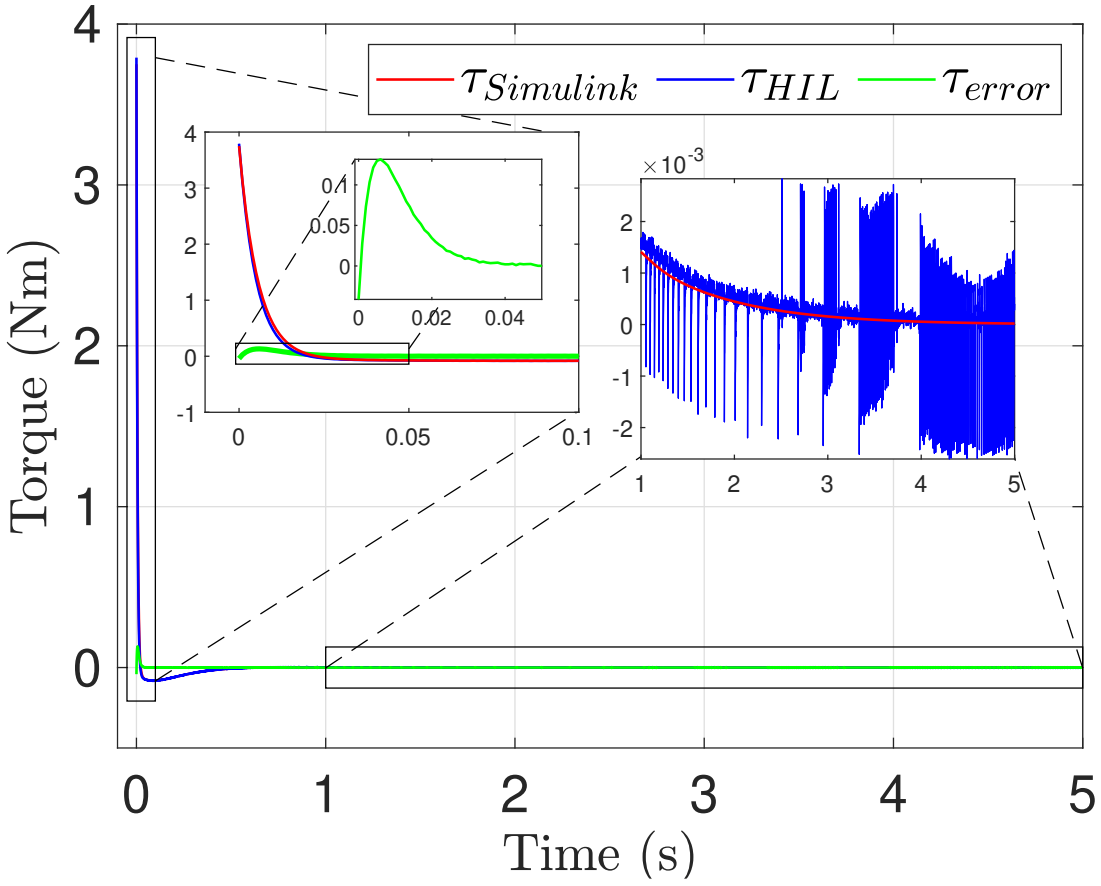


Figura 35: Comparación en bucle cerrado entre *Hardware-in-the-loop* y *Simulink*.

Como se puede observar en la figura 35, el máximo error entre ambas simulaciones, alrededor de $t = 0{,}01s$, es menor al 5 %. También, cuando el péndulo alcanza el estado estacionario se pueden observar oscilaciones de alta frecuencia y baja magnitud en τ_{HIL}, tal como muestra el subplot de la derecha. Estas oscilaciones se deben principalmente a 2 efectos. El primer efecto son los errores de cuantización debido a que la transferencia de información entre la PC y el STM32 se realiza en formato *string* decimal. Este primer error se puede solucionar mandando el signo, la mantisa

y el exponente de los números en vez de el número en sí. Esto con el propósito de que el STM32 reconstruya el número sin perdidas por truncamiento del punto decimal. Para más información ver sección 2.1 de la referencia [26].

El segundo problema se debe a un efecto conocido como (*"chattering"*), *"parloteo"*. El *"chattering"* es producido debido a la frecuencia de muestreo finita del μC [40].

8.6.3. Simulación de muestreo

A continuación, en la figura 36, se muestra una simulación del fenómeno que causa los errores de medición a bajas velocidades. Cómo se puede observar en la imagen, a velocidades bajas los

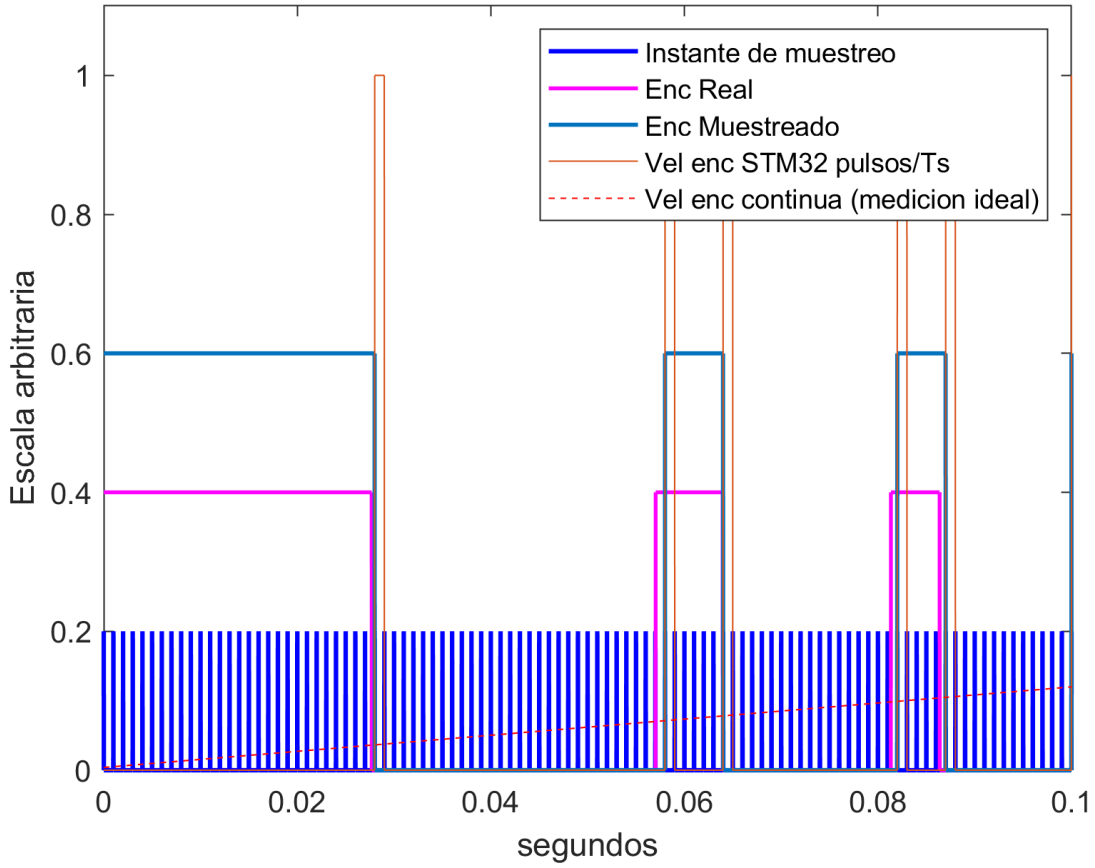


Figura 36: Barrido de velocidad angular (Simulación).

periodos de pulsos (rosa) son bastantes mayores que los periodos entre muestreos (impulsos azules), esto provoca que en la mayor parte del tiempo la velocidad angular detectada sea 0 y solo existan mediciones de velocidad de $\frac{1}{T_s}$ durante periodos cortos (señal color café). La señal cuadrada de color azul es la posición del codificador rotatorio que el sistema de medición detecta y la línea color rosa es la posición real del codificador rotatorio (necesitaríamos una frecuencia de muestreo infinita para obtener esa señal).

Por último, la señal roja punteada que se encuentra al fondo de los impulsos de muestreo es la velocidad angular continua del codificador rotatorio, esta señal es imposible de medir y solo puede ser simulada, se necesitaría un codificador rotatorio de resolución infinita y una frecuencia de muestreo infinita para obtener tal señal ya que es continua.

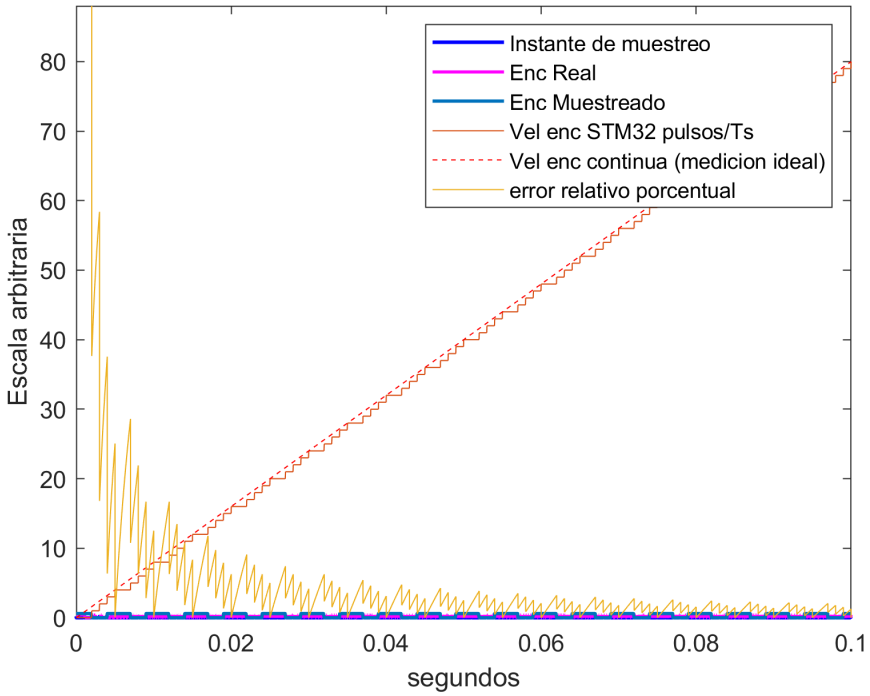


Figura 37: Barrido de velocidad angular de 1 a 80 pulsos/Ts. (Simulación).

La figura 37 muestra el efecto del error a bajas velocidades. En esta ocasión se hizo un barrido de velocidad donde se va incrementando la velocidad angular desde 1 hasta 80 pulsos por periodo de muestreo. La línea puntiaguda amarilla representa el error relativo porcentual de la velocidad medida por el μC STM32 (línea naranja) con respecto a la velocidad medida idealmente (línea roja). Como se puede observar, el error porcentual (línea amarilla) decrece conforme se aumenta la velocidad, aproximando 0 cuando la velocidad es muy alta. Este problema de medición a bajas velocidades puede ser contrarrestado contando el tiempo entre pulsos o utilizando un observador que estime el estado del sistema entre pulsos [41].

8.6.4. Simulación del backlash

Para comprender más a profundidad el fenómeno del backlash se simuló en *MATLAB* su comportamiento. El código está en el anexo 12.10. Como se puede observar en la figura 38, el backlash produce un comportamiento no lineal de la salida.

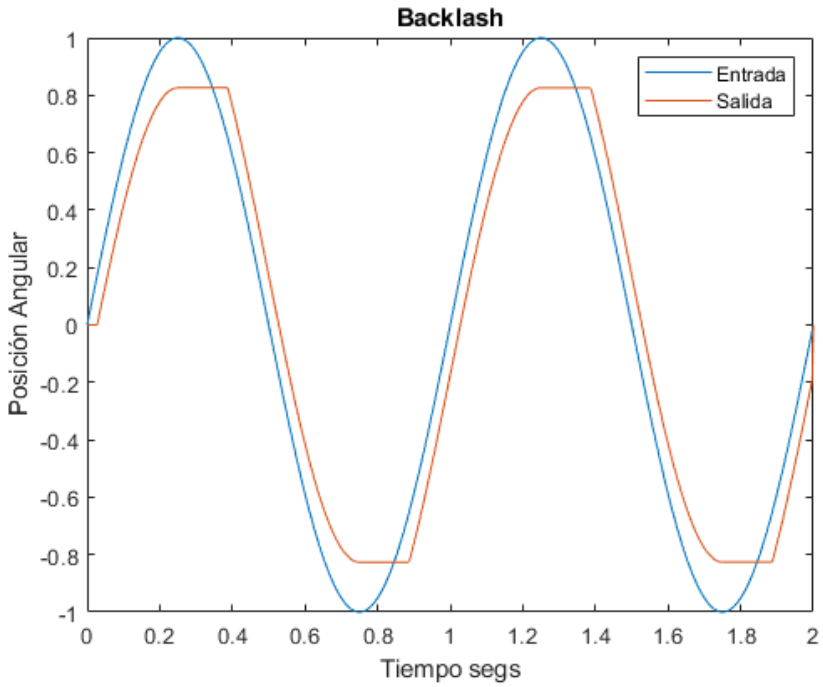


Figura 38: Efecto del backlash en la salida.

En la simulación de la figura 38 la señal de entrada (referencia) es un sinusoide con frecuencia $f = 1$ Hz. A la salida se puede observar que se reduce al rango del actuador y se genera un desfase de la salida respecto a la entrada.

En la sección 8.5.2.1 se atenúan los efectos del backlash sin modificar mecánicamente el sistema. Esto se realiza mediante el procesamiento digital de señales con ayuda de un filtro.

9. Implementación del algoritmo de control

9.1. Motor

En el transcurso de esta investigación se utilizaron 2 motores.

El 1er motor utilizado fue un *Pololu* modelo *70:1 Metal Gearmotor 37Dx70L mm 12V*. Este motor se utilizó en el péndulo rotatorio V4 y anteriores.

El 2do motor utilizado se comenzó a utilizar en el péndulo rotatorio V5. El motor es un *Maxon* modelo *RE-max 29 226783*.

9.1.1. Pololu 70:1 Metal Gearmotor

El motor *Pololu 70:1 Metal Gearmotor 37Dx70L mm 12V* cuenta con un codificador rotatorio de 64 pulsos por revolución, el codificador rotatorio está conectado al motor y no a la salida de la caja de engranajes por lo que la verdadera resolución angular $64 \cdot 70 = 4480$

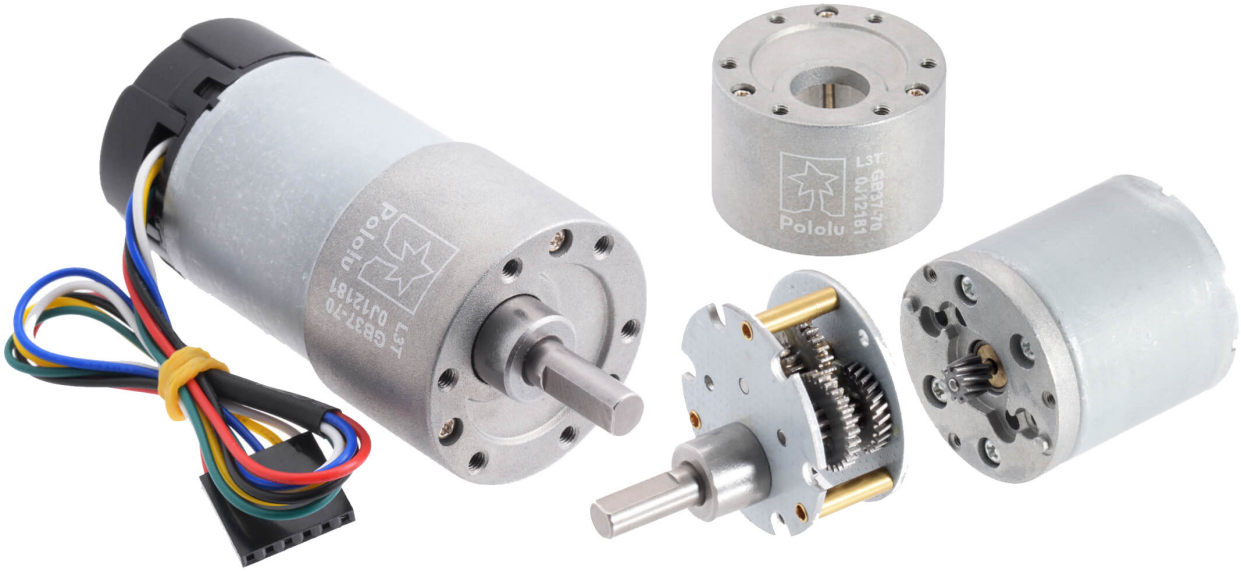


Figura 39: Motor y caja de engranajes Pololu.

Más información del motor y sus parámetros disponible en la sección 9.3.1.5 y en la hoja de datos del mismo [42].

9.1.2. Maxon RE-max 29 226783

El motor *Maxon RE-max 29 226783* tiene escobillas construidas con metales preciosos, tiene un voltaje nominal de 48V, un torque nominal de 25,4 mNm y una corriente nominal de 153 mA. El motor utilizado en esta investigación además cuenta con un codificador rotatorio modelo *Encoder MR 225780* de 1000 ppr (4000 efectivos) [43].

Figura 40: Motor *Maxon RE-max 29 226783*.

Más información del motor y sus parámetros disponible en la sección 9.3.1.6 y en la hoja de datos del mismo [44].

9.2. Sensores

Para la medición de la posiciones angulares se utilizan 2 diferentes codificadores rotatorios de cuadratura. Uno de marca *OMRON* situado en un extremo del brazo del péndulo rotatorio y el otro integrado en el motor.

9.2.1. OMRON E6B2-CWZ6C

Para la medición de la posición angular de la barra se utilizó un codificador rotatorio incremental modelo *E6B2-CWZ6C* de la marca *OMRON*.

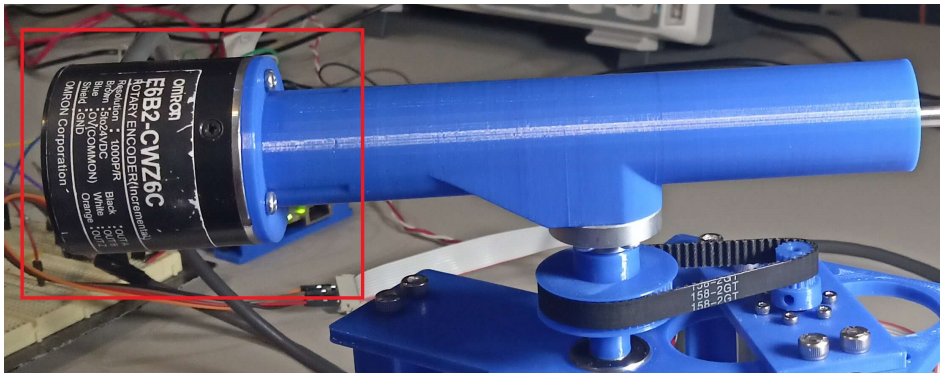


Figura 41: Codificador rotatorio modelo *OMRON E6B2-CWZ6C* en el péndulo rotatorio V5.

- Resolución de 1000 ppr (4000 cuando se utilizan los flancos de ambos canales)
- 5VDC de alimentación
- Salida de colector abierto NPN

Más información en la hoja de datos [45].

9.2.2. Codificadores integrados Maxon y Pololu

Para la medición de la posición angular del brazo se utiliza el codificador rotatorio integrado del motor.

	Codificador Pololu	Codificador Maxon
PPR	4480 ppr	4000 ppr
Resolución	80,35E−3 °	90E−3 °
Resolución	1,4E−3 rad	1,57E−3 rad

Tabla 2: Comparativa entre codificadores rotatorios
Comparativa entre codificadores rotatorios integrados de los motores *Pololu* y *Maxon*.

9.3. Caracterización del péndulo rotatorio

9.3.1. Caracterización del Motor del péndulo

La estimación de parámetros consiste en aproximar las constantes de un sistema mediante un algoritmo matemático que intenta reducir al mínimo la diferencia entre la simulación del sistema y datos medidos proporcionados por el usuario.

Estimar los parámetros de un sistema requiere 3 tareas previas a la estimación; modelado, medición y preparación de las mediciones. Las mediciones no requirieron ser preparadas ya que se automatizó este proceso. Se utilizó *Simulink* para la estimación de parámetros. Al software se le indicó reducir al mínimo el error cuadrático. Se utilizó el método de mínimos cuadrados no lineales y el algoritmo reflexivo de región de confianza [46].

9.3.1.1. Modelado del motor

El motor se modeló en *Simulink* con fricción viscosa en función de la velocidad. Los parámetros del motor son los siguientes:

- R_a = Resistencia eléctrica del motor.
- L_a = Inductancia del motor.
- K_{ma} = Constante de torque.
- J = Momento de inercia del motor.
- c = Constante de fricción viscosa.
- K_b = Constante de Back-EMF.

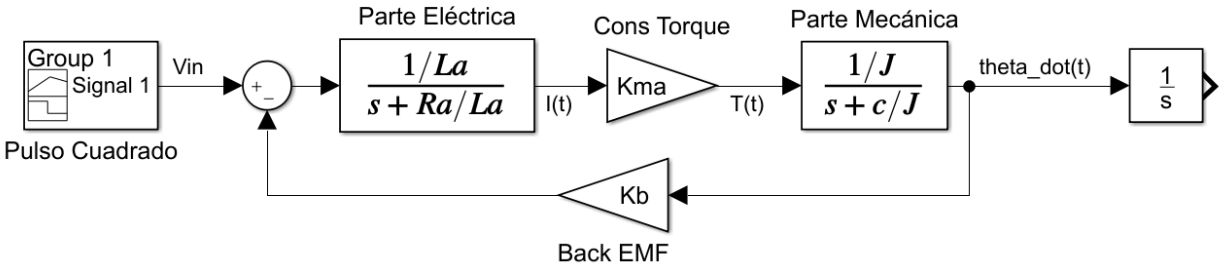


Figura 42: Diagrama de bloques del motor en *Simulink*.

Dado que las unidades del SI son consistentes por conservación de energía: $K_{ma} = K_b$

9.3.1.2. Mediciones del motor

Para la medición de velocidad angular se realizaron 2 mediciones. Se midió la velocidad angular y la corriente del motor. Se utilizó un osciloscopio con una resistencia *"shunt"* de 1Ω en serie para medir la corriente.

La velocidad angular se manda en tiempo real por VCOM USB a la computadora y la medición de la corriente con el osciloscopio se guardó en un USB en un archivo .csv que posteriormente se lee en la PC.

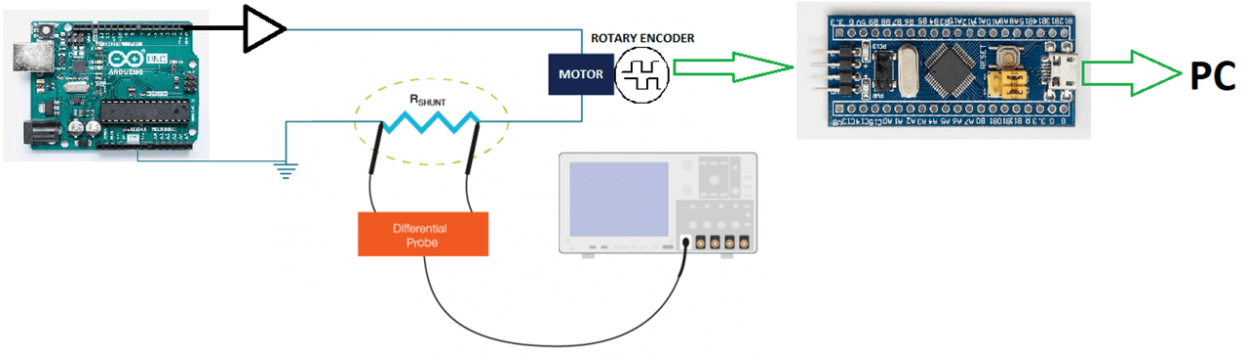


Figura 43: Diagrama electrónico de la medición.

La señal de entrada al motor es la siguiente:

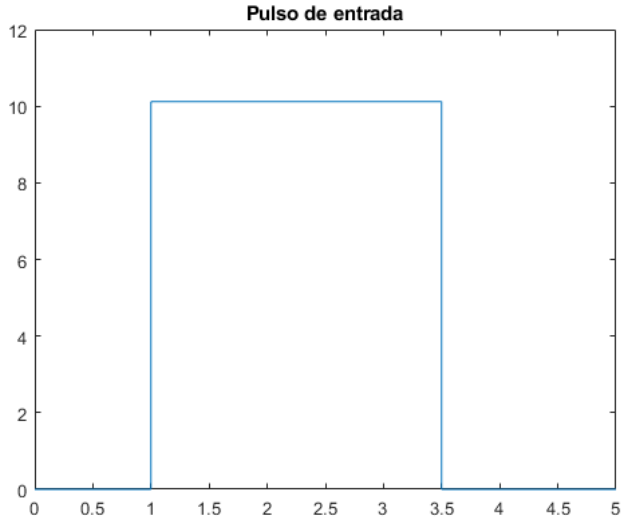


Figura 44: Voltaje suministrado al motor.

9.3.1.3. Preparación de las mediciones

Antes de introducir las señales medidas a *Simulink* es necesario importarlas a MATLAB, verificar que tengan la misma duración, que el vector de tiempo sea correcto y que sean vectores de columna de tamaño 1 por n; no renglones de n por 1.

La velocidad se capturó de forma automatizada.

La corriente del motor se capturó en un archivo .csv que se copió a la PC. Para importarlo a MATLAB el archivo se sitúa en la misma carpeta que el archivo con el código de MATLAB. El código utilizado fue el siguiente:

```
clc
clear all
data = csvread('canal1.csv',2,0);
dt = csvread('canal1.csv',1,3);
dt = dt(1);
ttemp=(data(1:end,1)*dt);
CH1temp=(data(1:end,2));

tini=5;
tfin=10;

t=(1:(tfin-tini)/dt)'*dt;
CH1=CH1temp(1+tini/dt:tfin/dt);

subplot(1,2,1);
plot(ttemp,CH1temp)
title('CH1 OSC (CORRIENTE) ANTES DE RECORTAR')
subplot(1,2,2);
plot(t,CH1)
title('CH1 DESPUES DE RECORTAR')
```

Figura 45: Código MATLAB para preparar la medición.

El código importa las mediciones, crea la matriz *data* (sin cabecera) y obtiene el periodo de muestra, *dt*, para posteriormente convertir la matriz a vectores de columna y calcular el vector de tiempo. También, es muy importante extraer una sección de la medición que coincida con la duración de la simulación, en este caso extraeremos desde el segundo 5 hasta el 10. Al final se grafican ambas señales.

Ya que ambos vectores de columna, t y CH1 son correctos y están en el WORKSPACE de MATLAB se procede a la estimación de parámetros en simulink.

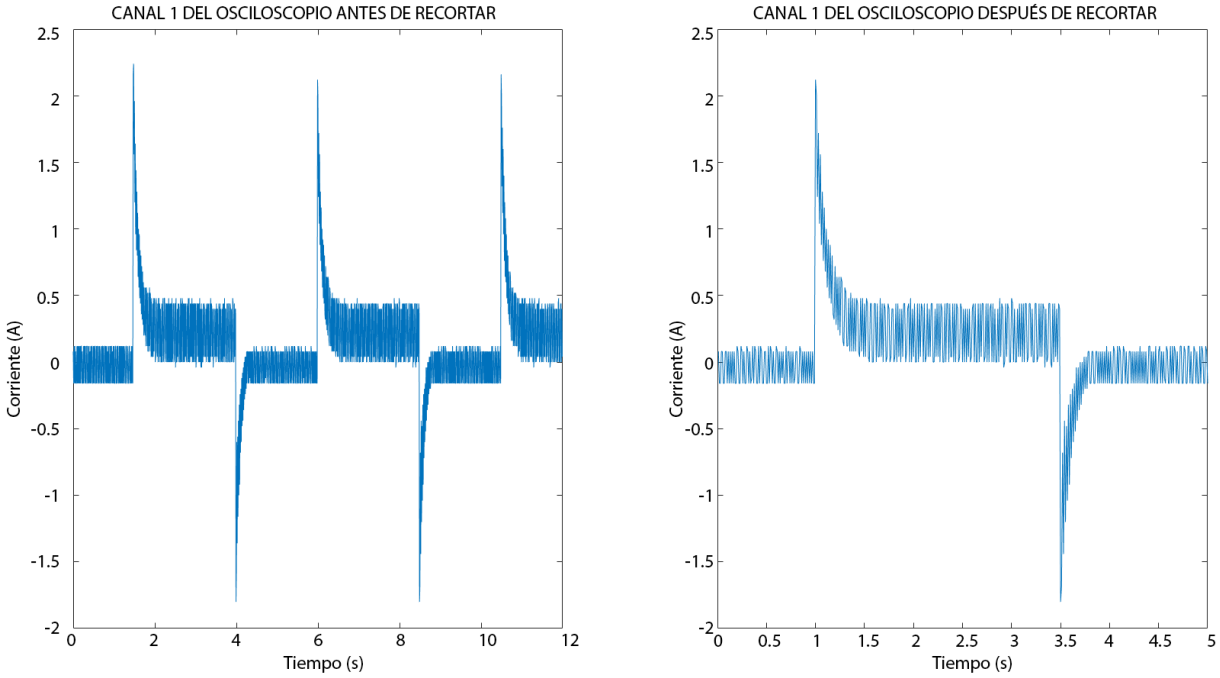


Figura 46: Comparación de señal de corriente antes y después de prepararla.

Para la preparación de la señal de velocidad angular se siguió un procedimiento similar.

9.3.1.4. Estimación de parámetros sin carga

Una vez teniendo preparado el vector de tiempo y ambas señales con magnitudes apropiadas; amperes para la corriente y radianes por segundo para la velocidad, se procede a abrir el modelo del motor en Simulink. Es importante que las constantes del motor (R_a,L_a, etc.) se encuentren en el Workspace del modelo del motor de *Simulink*.

Ya teniendo las constantes, en las pestañas de arriba, en la ventana de *Simulink*, se selecciona Análisis/Estimación de Parámetros. Una ventana se abrirá y se procede a "Seleccionar parámetros", se abrirá una nueva ventana y nuevamente seleccionamos "Seleccionar parámetros". Ahora, dado que se midieron físicamente la inductancia y resistencia del motor, solo se estimarán las constantes J,K_{ma} y C.

Se selecciona un intervalo para los parámetros de 0 a ∞ ya que estos no pueden tener valores negativos.

Ahora se selecciona en la parte de arriba de la ventana, "Nuevo experimento", se seleccionan en el modelo de *Simulink* las señales medidas de salida $I(t)$ y $\dot{\phi}$ dándoles clic y se presiona OK. Después, se introducen los vectores de tiempo y velocidad para las señales $\dot{\phi}$ y $I(t)$.

Los vectores resultantes deben ser $[t, \dot{\phi}(t)]$ y $[t, I(t)]$. Donde $\dot{\phi}$ y $I(t)$ son la velocidad y corriente medidas respectivamente. Una vez hecho lo anterior se procede a "estimar" los parámetros. El algoritmo de *Simulink* intentará reducir el error cuadrático entre la simulación y la medición utilizando la técnica de descenso por gradiente. A este porcedimiento se le conoce como mínimos cuadrados.

Una vez realizada la estimación se debería obtener una simulación muy cercana a la medición. Estos fueron los resultados:

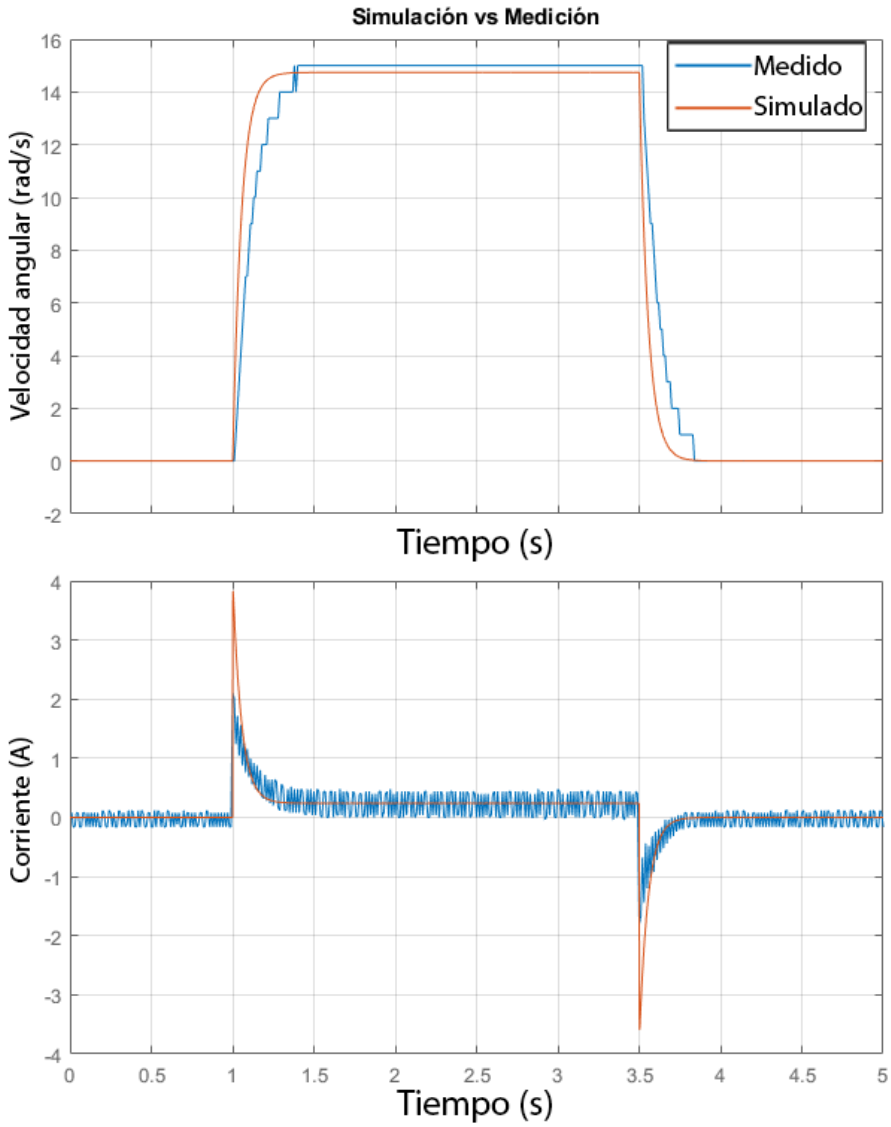


Figura 47: Simulación vs Medición.

En la figura 47 se puede observar que la señal de corriente medida coincide con la señal de corriente simulada por *Simulink*. Esto significa que la estimación de parámetros fue exitosa y el motor simulado tiene un comportamiento similar al real.

También se puede observar el comportamiento de los parámetros durante la estimación:

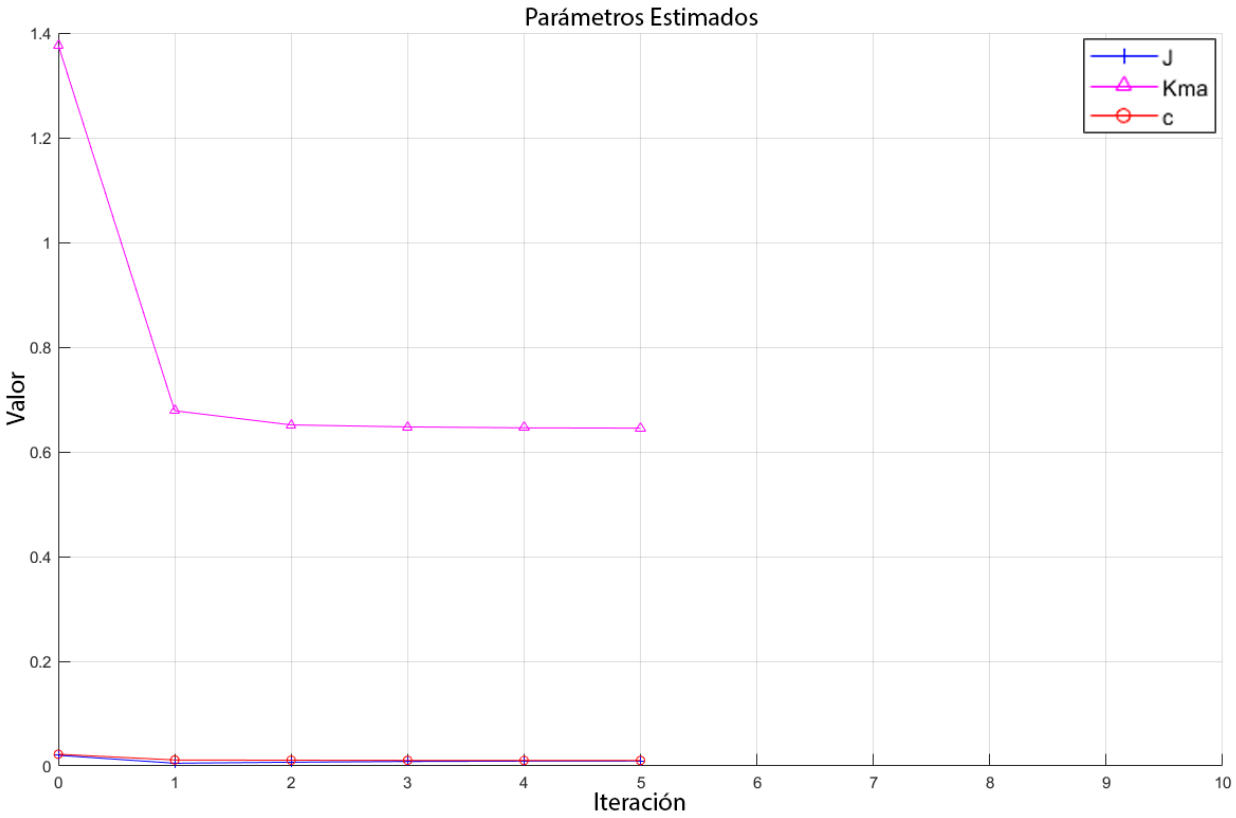


Figura 48: Parámetros durante cada iteración.

9.3.1.5. Parámetros del Motor Pololu

Con el procedimiento del punto anterior se obtuvieron los siguientes parámetros del motor *Pololu 70:1 Metal Gearmotor 37Dx70L mm 12V with 64 CPR Encoder*:

- $J = 20.409 \times 10^{-3}\ \mathrm{kg\,m^2}$
- $K_{ma} = 1{,}3768\ \mathrm{N\,m\,A^{-1}}$
- $F_{v\phi} = 22.425 \times 10^{-3}\ \mathrm{N\,m\,s\,rad^{-1}}$

Y los que fueron medidos físicamente:

- $R_a = 2.54\,\Omega$
- $L_a = 1.584 \times 10^{-3}\ \mathrm{H}$

9.3.1.6. Parámetros del Motor Maxon

También se caracterizó un segundo motor. El motor es un *MAXON 226783* con un codificador rotatorio de 1000 ppr, el modelo del codificador es *MAXON 225780*

- $J = 1.18 \times 10^{-6}\ \mathrm{kg\,m^2}$
- $K_{ma} = 0{,}168\ \mathrm{N\,m\,A^{-1}}$
- $F_{v\phi} = 1.243 \times 10^{-6}\ \mathrm{N\,m\,s\,rad^{-1}}$

Y los que fueron medidos físicamente:

- $R_a = 104\,\Omega$
- $L_a = 8.48 \times 10^{-3}\ \mathrm{H}$

9.3.2. Caracterización del eje θ del péndulo rotatorio

Debido a la complejidad del péndulo rotatorio resulta conveniente caracterizar por separado cada uno de los ejes de giro. Para caracterizar el eje de giro θ se va a mantener fijo el eje de giro ϕ de manera que $\phi = \dot{\phi} = 0$. Mantener fijo al eje ϕ convierte al péndulo rotatorio en un péndulo simple.

9.3.2.1. Modelado matemático del péndulo simple

En el modelo del péndulo simple se tomará en cuenta la fricción de Coulomb, la fricción viscosa y se modelará con la ecuación diferencial no-lineal tomando en cuenta el seno del ángulo y el péndulo como un péndulo de barra. A este tipo de péndulo también se le conoce como *rod pendulum* en inglés.

$$\ddot{\theta} + \lambda\dot{\theta} + \frac{mgL}{I_s} sen(\theta) = 0 \tag{71}$$

donde:

- $\lambda = sgn(\dot{\theta}) \cdot (F_{v\theta} \cdot |\dot{\theta}| + F_{c\theta})$ es la fricción no-lineal
- $F_{v\theta}$ es la fricción viscosa [$\mathrm{N\,m\,s\,rad^{-1}}$]
- $F_{c\theta}$ es la fricción de Coulomb [$\mathrm{N\,m^{-1}}$]
- m es la masa del péndulo [kg]
- $g = 9{,}81$ es la aceleración de la gravedad [$\mathrm{m\,s^{-2}}$]
- L es la longitud del péndulo [m]
- θ es la posición angular del eje [rad]
- I_s es el momento de inercia del péndulo de barra calculado analíticamente [$\mathrm{kg\,m^2}$]

9.3.2.2. Caracterización de péndulo simple con *Simulink*

Para la caracterización se realizó un experimento con el péndulo real. El experimento consistió en dejar caer el péndulo θ desde una posición angular $\theta = 90°$ hasta $\theta = 0$; todo esto manteniendo $\phi = \dot{\phi} = 0$ en todo momento.

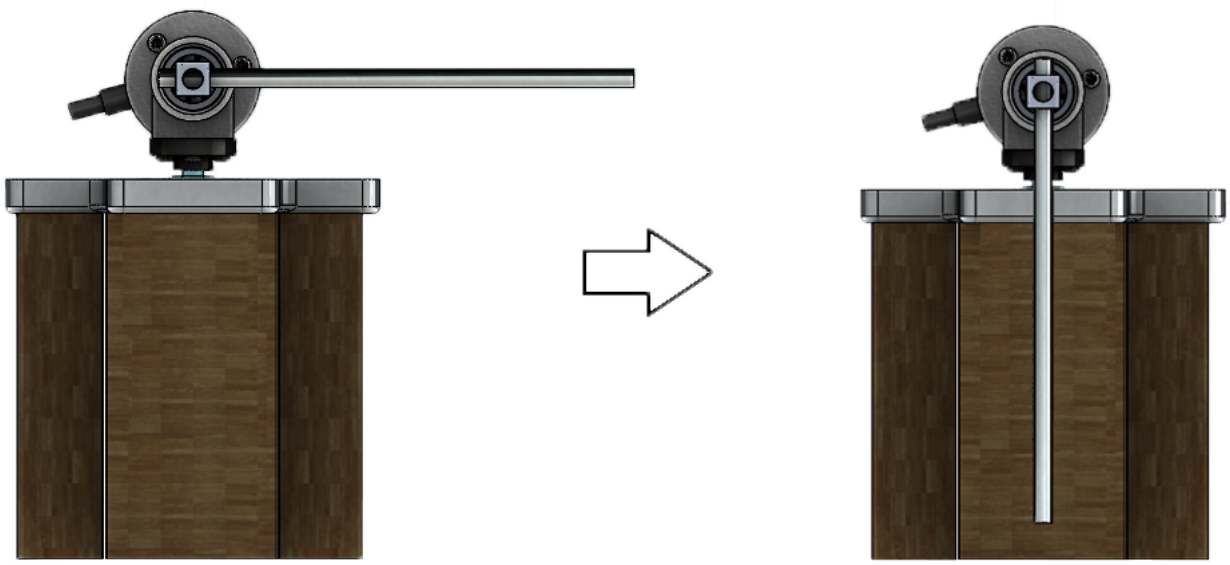


Figura 49: Posición inicial y posición final del experimento.

Para caracterizar el péndulo simple se utilizó el método de variación de parámetros de *Simulink* que varía los parámetros del modelo simulado hasta que se reduce el error cuadrático entre la simulación y la medición al mínimo posible.

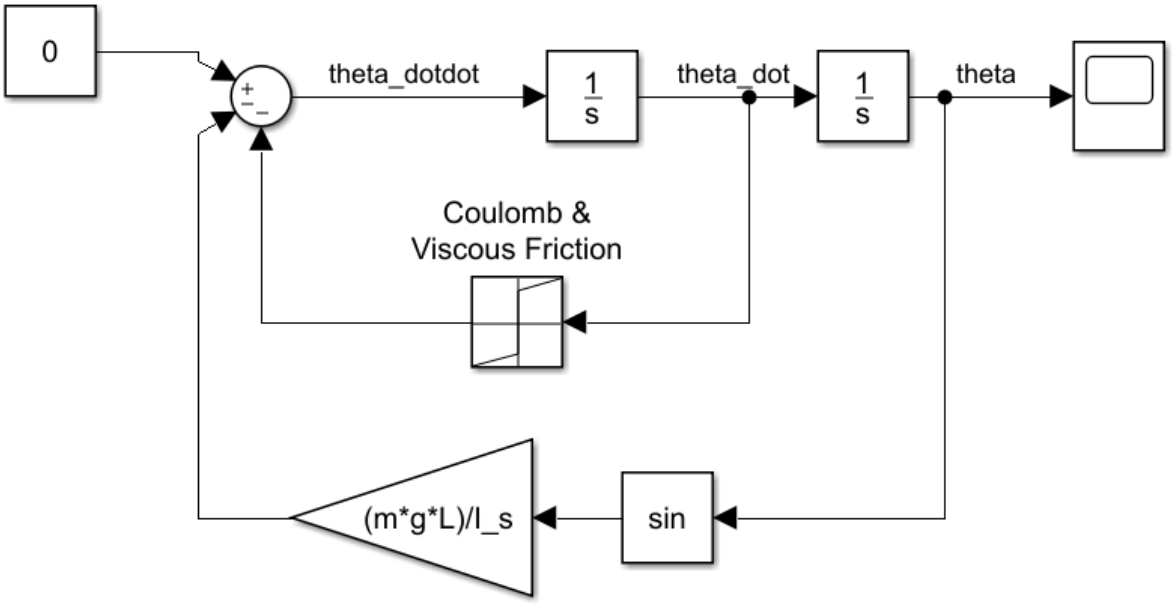


Figura 50: Diagrama de bloques en *Simulink*.

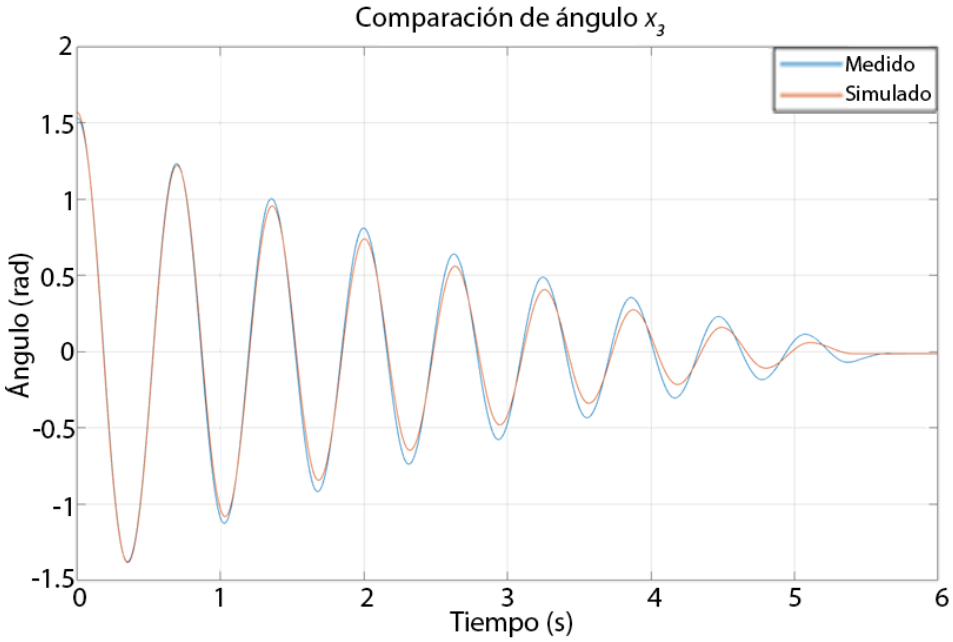


Figura 51: Comparación de péndulo simple real y modelado en *Simulink*.

Al finalizar la variación de parámetros se caracterizó la fricción del péndulo simple:

- $F_{c\theta} = 1{,}4\ \mathrm{N\,m^{-1}}$
- $F_{v\theta} = 0{,}525\ \mathrm{N\,m\,s\,rad^{-1}}$

Las constantes medidas directamente son:

- $L = 0{,}13\ m$
- $m = 0{,}03\ kg$
- $g = 9{,}81\ \mathrm{m/s^2}$

Las constantes que fueron calculadas a partir de las mediciones son:

- $I_s = 0{,}0003789\ \mathrm{kg\,m^2}$

9.4. Estandarización y diseño de experimentos

La estandarización del experimentos ayuda a reducir la probabilidad de que errores humanos ocurran ya que facilita familiarizarse con el sistema que se está estudiando. Algunos de los errores que se busca evitar son la medición errónea de señales, errores de signo, sistemas de coordenadas ambiguos, condiciones experimentales desiguales, etc.

Tener un ambiente de experimentación estandarizado permite realizar experimentos más confiables, esto permite observar diferencias más sutiles del desempeño del controlador, fenómenos causados por discretización de magnitudes y diferencias de comportamiento entre la planta física y la planta *Hardware-in-the-loop*.

Para tener condiciones experimentales igualitarias se instalará la planta con su base tan paralela al piso como sea posible (perpendicular al vector gravitatorio) y se utilizará un código de colores de cables para reducir equivocaciones. También se verificará que los torques, voltajes y direcciones de giro coincidan en la simulación y en el laboratorio con la planta real.

Agregado a lo anterior se utilizará redundancia de medición con un osciloscopio para verificar que los datos enviados a la PC coincidan con los mostrados en el osciloscopio.

9.4.1. Factores influyentes

En el caso específico del péndulo rotatorio, algunos de los factores influyentes controlables detectados son los siguientes:

- Frecuencia de muestreo del μC
- Voltaje pico del PWM
- Frecuencia del PWM
- Resolución de duty cycle del PWM
- Resolución aritmética del μC (int8, int16, int32, float)

Los parámetros físicos del péndulo rotatorio también son factores influyentes. Por simplicidad estos no serán modificados frecuentemente. Los parámetros físicos influyentes son los siguientes:

- Parámetros físicos del péndulo: La longitud, masa, ángulo de la base respecto al piso, centro de masa y distribución de masa del péndulo así como efectos elásticos del cable del codificador rotatorio.
- Parámetros mecánicos y eléctricos del motor: Momento de inercia, fricción, resistencia eléctrica de la bobina, backlash del eje, caja de engranes, etc.

En los experimentos se utilizarán los siguientes valores de los factores influyentes salvo que se mencione lo contrario.

- Frecuencia de muestreo del μC = 10 kHz
- Voltaje pico del PWM = 24 V
- Frecuencia del PWM = 20 kHz
- Resolución de duty cycle del PWM = 2^{16}
- Resolución aritmética del μC = float

Las razones por las que se seleccionaron dichos valores son las siguientes:

- Frecuencia de muestreo del μC: Es la frecuencia más alta a la que aún se tiene suficiente margen de tiempo para realizar otras tareas además del bucle de control. También es un múltiplo de 10 por lo que se facilitan los cálculos. Si se aumenta la frecuencia, T_s podría ser no-constante. Se comprobó con osciloscopio y analizador lógico que cada iteración ocurra cada 100 μs.
- Voltaje pico del PWM: Es el máximo voltaje de la driver.
- Frecuencia del PWM: Es la frecuencia más baja a la que las vibraciones de la bobina del motor están fuera del rango de audición humano.
- Resolución de duty cycle del PWM: Es la resolución máxima del TIMER de 16 bits del μC.
- Resolución aritmética del μC: Se utilizó float para utilizar todas las magnitudes y cálculos en **S.I**. Esto ayuda a mantener la intuición de las variables y otorga portabilidad al código del controlador. Es necesario realizar las conversiones pertinentes de unidades discretas a unidades **S.I** siempre que se modifiquen parámetros que afecten las unidades discretas, como por ejemplo, la resoluciones de los encoders, la frecuencia de muestreo, etc.

9.4.2. Medición y estimación

Las mediciones que se toman son las siguientes:

- Posición: La posición angular del eje del motor (ϕ) se mide con el μC vía hardware usando los temporizadores. Los temporizadores fueron configurados como contadores que aumentan en 1 su valor si el codificador gira 1 paso hacia un lado y viceversa.
- Velocidad: La velocidad se calcula en base a la medición de las posiciones con la siguiente fórmula implementada en el código del μC: $\dot{\theta} = \frac{\theta[n]-\theta[n-1]}{T_s}$ donde n es la muestra actual de posición y $\dot{\theta}$ es la velocidad angular expresada en pasos por segundo. Lo mismo se realiza para ϕ. Dependiendo del experimento la velocidad se podría calcular de diferente forma (especificada en el experimento).
- Señal de control: La señal de control se calcula en base al vector de retroalimentación de estados. Esta señal que representa una magnitud de voltaje continuo se convierte a una señal PWM con un voltaje RMS equivalente para energizar el motor con ayuda del puente H.

9.4.3. Procedimiento de medición

La medición es realizada por el μC y se manda en tiempo real por protocolo serial virtual RS-232 mediante USB a la PC para su posterior análisis. Además, cuando se realizan cambios al algoritmo de medición los valores medidos se mandan a una salida analógica del μC para medirlos utilizando el osciloscopio. Esto se hace con fines de redundancia para validar que los datos enviados desde el μC a la PC vía serial sean consistentes con los medidos por el osciloscopio.

Primero se establece comunicación serial con la PC para grabar los datos. Después se sitúa el péndulo con las condiciones iniciales deseadas y se presiona el botón designado para comenzar la prueba. El botón inicia el experimento deseado, ya sea una respuesta al escalón, seguir una señal cuadrada, experimentos en bucle abierto, etc. Se puede automatizar la medición siempre que el ángulo del péndulo sea $\theta = n\pi$ donde $n \in \mathbb{Z}$.

En la PC se cuenta con un código en MATLAB que grafica toda la información en tiempo real y además la guarda para posterior análisis (ver An. 12.6).

Una vez que el péndulo ya está situado en una posición $\theta = n\pi$ con $n \in \mathbb{Z}$ el procedimiento de medición paso a paso es el siguiente:

1. El usuario comienza a ejecutar el programa teniendo ya todo el sistema conectado.
2. La PC envía el vector de retroalimentación vía serial al STM32. (Virtual-COM por USB).
3. El STM32 consulta la posición de los codificadores rotatorios. La primera lectura resulta en una posición igual a 0 grados debido a que así se inicializa.
4. El STM32 calcula/estima la velocidad.
5. El STM32 realiza la multiplicación del vector de estados, que contiene las velocidades y posiciones de ambos codificadores rotatorios por el vector de retroalimentación para obtener la señal de control. Dependiendo del signo de la señal de control se invertirá la polaridad del motor con los pines correspondientes del STM32.
6. Una vez obtenido el torque el STM32 procede a "convertir" el torque a voltaje utilizando la función de transferencia del motor caracterizado (ya implícita en el código del micro-controlador). Las constantes del motor caracterizado ya están incluidas en el código.

7. Una vez obtenido el voltaje requerido para el torque deseado el STM32 procede a convertirlo a una señal de voltaje PWM.
8. En el STM32 el pin con la señal PWM está conectado a los pines *"enable"* 1 y 2 del driver del motor de manera que se controla el voltaje RMS.
9. El STM32 convierte las velocidades, posiciones y torque en un STRING de texto.
10. Se envía el STRING vía serial a la PC cada iteración.
11. Se repiten los pasos anteriores (con excepción del 1ro) cada periodo de muestreo T_s.

Los tipos de datos (single, double, float, etc) que se utilizan están codificados según el estándar "IEEE754" [47] al igual que las conversiones y su manejo en operaciones aritméticas.

9.4.4. Diseño de experimento: Experimento de similitud en bucle abierto

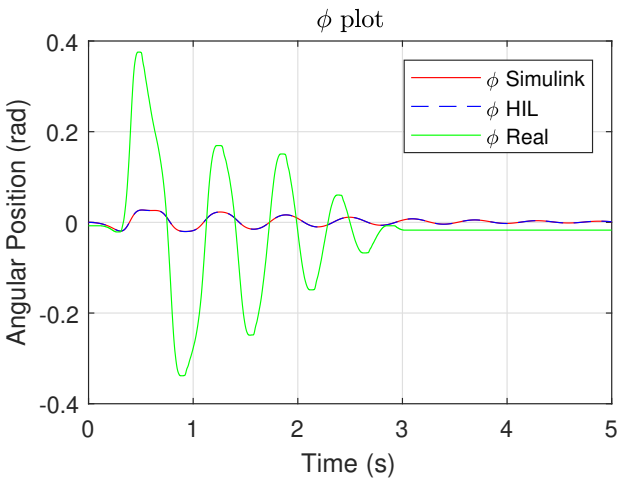
Para corroborar que se realizó correctamente el modelado matemático del péndulo rotatorio, la construcción física del péndulo, el diseño del algoritmo de control y su implementación en el micro-controlador se compararon las respuestas en el tiempo utilizando las mismas condiciones iniciales.

Se realizaron 3 tipos de pruebas:

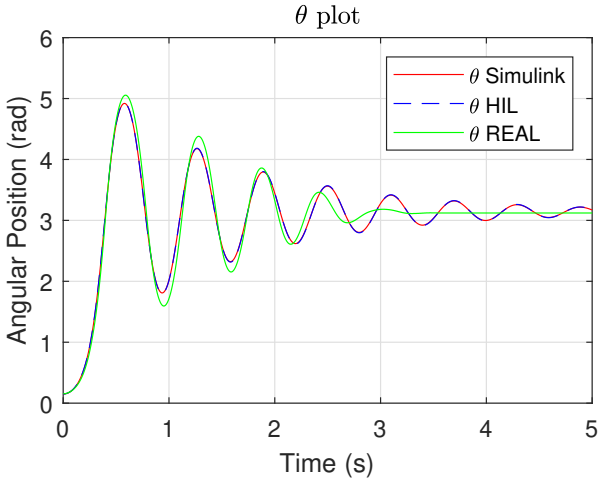
- Prueba de simulación completa: Esta simulación consiste en simular la planta y, si es deseado; el controlador. Esta simulación es la más sencilla ya que el ambiente es completamente controlado y no existen errores de medición. Para esta simulación se utilizó *Simulink*.
- Prueba de simulación parcial **(HIL)**: En esta simulación se simula solo la planta, el controlador es real. Se manda el vector de estado cada periodo de muestreo T_s al μC mediante la interfaz serial.
- Prueba real: Esta consiste en realizar el mismo experimento que en los puntos anteriores pero utilizando el controlador y planta reales.

Para la comparación en bucle abierto se realizó un experimento dejando caer el péndulo hasta llegar a reposo con las siguientes condiciones iniciales. Para $t = 0$:

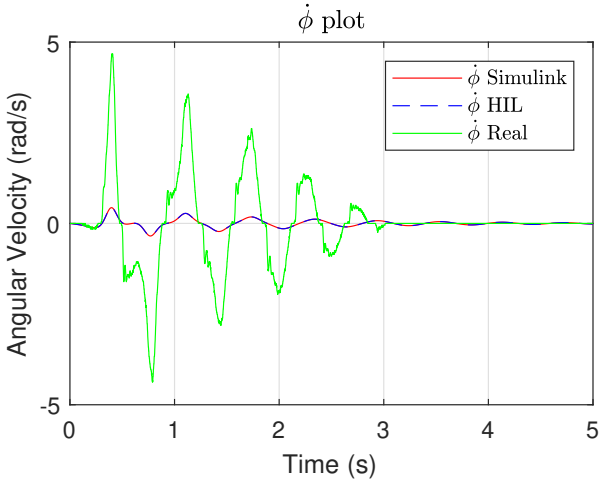
$$\begin{aligned} \phi(0) &= 0 \text{ rad} \\ \dot{\phi}(0) &= 0 \text{ rad} \\ \theta(0) &= 0{,}15 \text{ rad} \\ \dot{\theta}(0) &= 0 \text{ rad} \end{aligned} \tag{72}$$



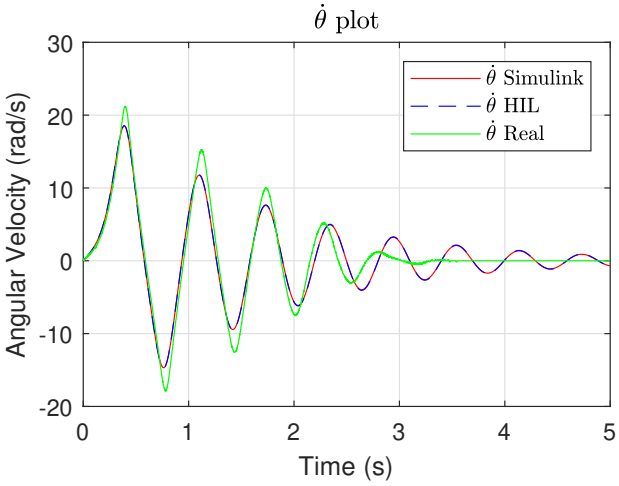
(a) Comparación de ángulo ϕ



(b) Comparación de ángulo θ



(c) Comparación del velocidad $\dot{\phi}$



(d) Comparación de velocidad $\dot{\theta}$

Figura 52: Comparación de las posiciones y velocidades angulares en bucle abierto la planta simulada en *Simulink*, la planta simulada *Hardware-in-the-loop* y la planta real.

9.4.4.1. Conclusiones de la comparación de bucle abierto

Como se puede observar en las figuras 52b y 52d, los valores obtenidos de θ y $\dot{\theta}$ con *Simulink* y con el *Hardware-in-the-loop* son prácticamente idénticos, esto confirma que la implementación del *Hardware-in-the-loop* fue realizado correctamente.

Sin embargo, los valores obtenidos por el sistema real en las figuras 52a y 52c comienzan con magnitudes muy similares a las del *Hardware-in-the-loop* y *Simulink* aunque conforme pasa el tiempo estos valores divergen (alrededor del segundo 2.5). Esto se debe a que la fricción estática comienza a ser un factor importante a bajas velocidades.

Los valores de ϕ y $\dot{\phi}$ se comportan de manera similar en el sistema real aunque con una magnitud diferente, esto se puede deber a un error en la magnitud de un parámetro o un error de medición del ángulo ϕ. Actualmente se está investigando la causa de este problema ya que las mediciones de ϕ parecen ser correctas por lo que el error debería de estar en el modelo matemático del péndulo que se utiliza en *Simulink* y en el *Hardware-in-the-loop* y no en el péndulo real.

Si se normalizan las magnitudes de la figura 52a se puede observar que las señales son muy similares (ver Fig. 53).

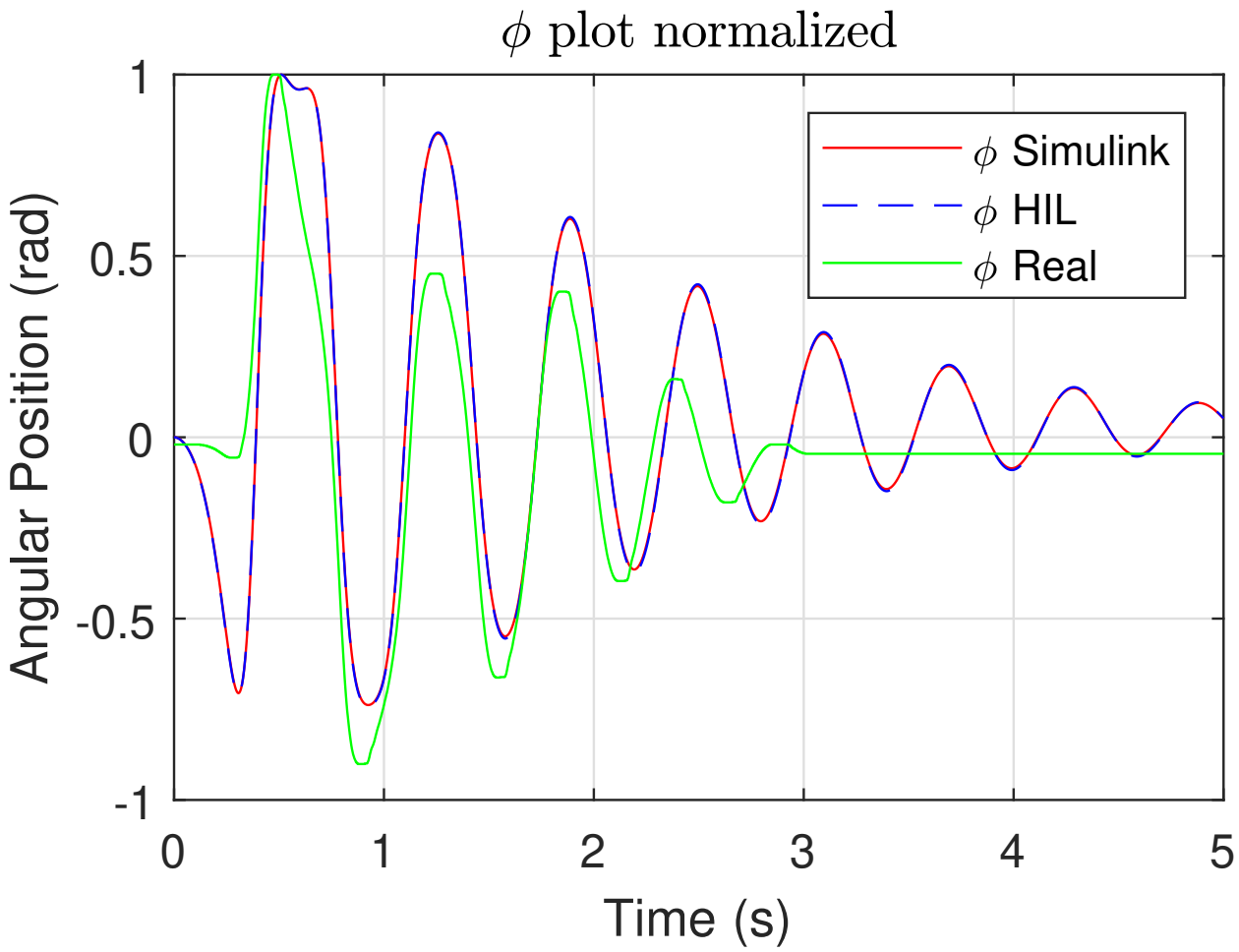


Figura 53: Comparación de la posición angular normalizada ϕ entre la simulación completa en *Simulink*, la simulación parcial con el *Hardware-in-the-loop* y la prueba real. Se puede observar la influencia de la fricción no lineal en la respuesta del sistema real.

9.4.5. Diseño de experimento: Experimento de similitud en bucle cerrado

Se busca determinar la similitud entre el comportamiento dinámico del modelo matemático del péndulo rotatorio y el péndulo real construido físicamente, comparar el tiempo de estabilización y determinar las causas de las diferencias.

Las variables a medir son:

- Las 4 variables de estado: ϕ, $\dot{\phi}$, θ, $\dot{\theta}$ $[rad]$ y $[rad/s]$
- La señal de control: τ $[Nm]$
- El tiempo de transiente: T_T $[s]$

El tiempo de transiente se define como la cantidad de tiempo en segundos que el péndulo rotatorio y el controlador requieren para alcanzar un valor estacionario donde $-0{,}05 < \theta < 0{,}05$ rad.

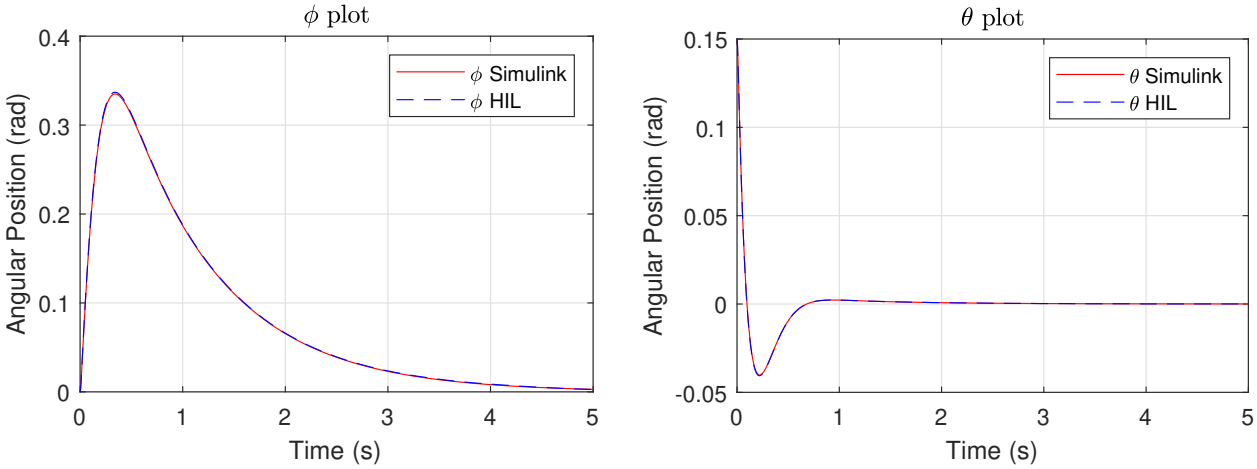
Los parámetros a variar serán los siguientes:

- Ángulo inicial θ
- Frecuencia de muestreo
- Amplitud del PWM

El experimento consiste en posicionar el péndulo en las condiciones iniciales deseadas y activar el controlador, todo lo que ocurra a partir de $t = 0$ se mide tal como el punto 9.4.3 explica.

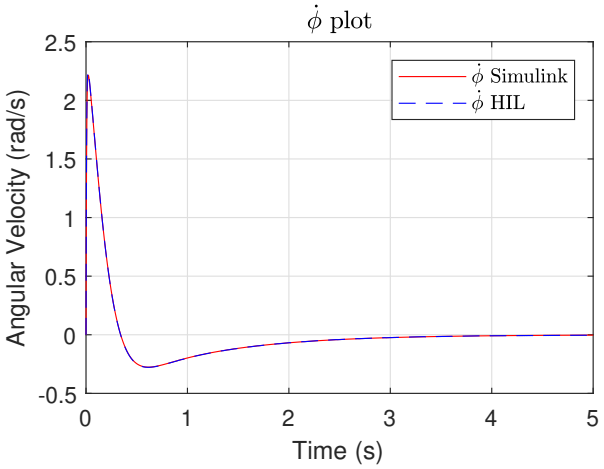
En esta serie de experimentos se cerró el bucle para que el controlador intente estabilizar el péndulo rotatorio. En este experimento se lograron identificar problemas de truncamiento y oscilaciones de muy baja amplitud y alta frecuencia.

Con el *Hardware-in-the-loop* se comprobó que la implementación del algoritmo de control fuera correcta al compararse con *Simulink* y se pudo demostrar que el controlador estabiliza el péndulo rotatorio de una manera rápida y eficiente.

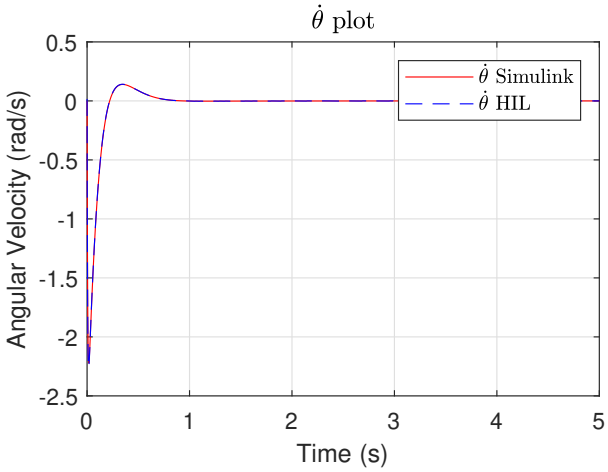


(a) Comparación de ángulo ϕ

(b) Comparación de ángulo θ

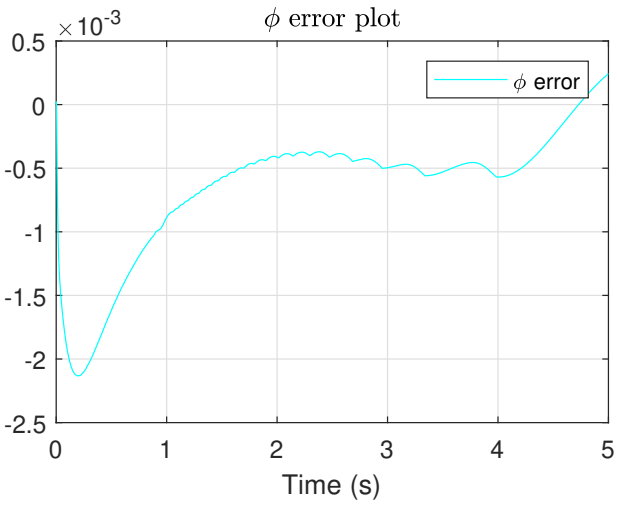


(c) Comparación del velocidad $\dot{\phi}$

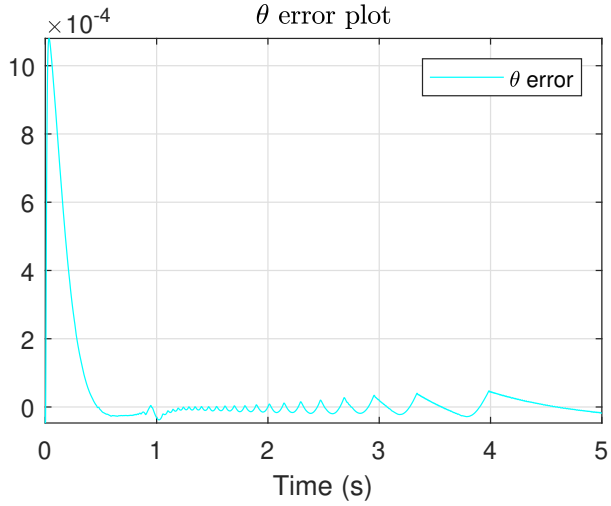


(d) Comparación de velocidad $\dot{\theta}$

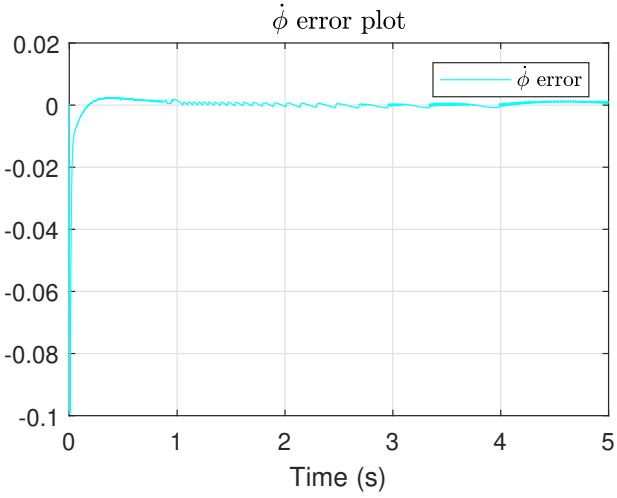
Figura 54: Comparación de las posiciones y velocidades angulares en bucle cerrado entre el péndulo rotatorio completamente simulado en *Simulink* y el parcialmente simulado con el *Hardware-in-the-loop*.



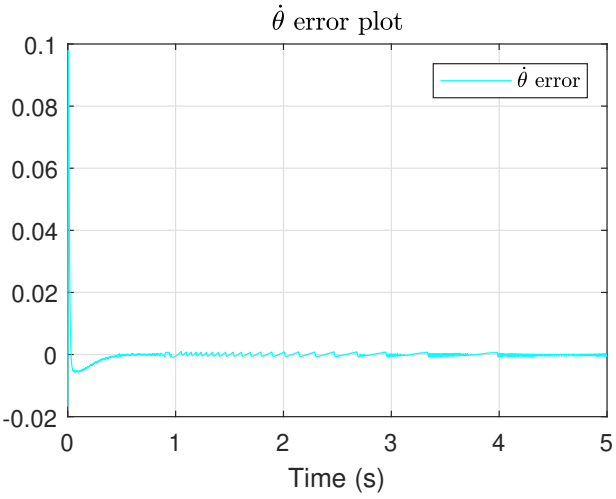
(a) Error de ángulo ϕ



(b) Error de ángulo θ



(c) Error de velocidad $\dot{\phi}$



(d) Error de velocidad $\dot{\theta}$

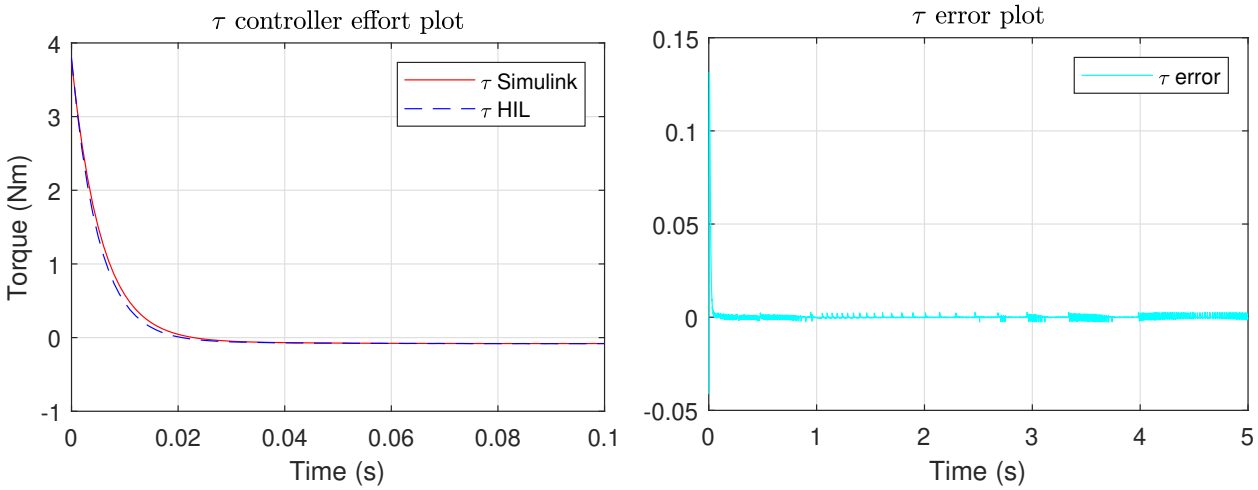
Figura 55: Error absoluto de las posiciones y velocidades angulares en bucle cerrado del péndulo simulado con el *Hardware-in-the-loop* respecto al de *Simulink*.

La fórmula utilizada para calcular el error absoluto es:

$$\epsilon_a = |\text{Valor de } \textit{Simulink} - \text{Valor del HIL}| \tag{73}$$

Como se puede observar en la figura 55 los errores tienden a 0 por lo que con este experimento se comprueba que el algoritmo de control implementado en el micro-controlador funciona correctamente de la misma manera que el controlador simulado en *Simulink*.

Una gran ventaja del *Hardware-in-the-loop* es que se pueden observar estados que en la práctica no son observables o solo son observables indirectamente, un ejemplo de esto es el torque, τ, aplicado al eje ϕ para estabilizar el péndulo. Además de poder medir cualquier variable del péndulo, la medición es ideal; esto se elimina la incertidumbre de medición y facilita la detección de errores en el controlador o en su implementación.



(a) Comparación de torques τ

(b) Error absoluto de torque τ

Figura 56: Observación de estado no observable utilizando el *Hardware-in-the-loop*.

En la figura 56 se puede observar que el torque aplicado por ambos controladores es prácticamente el mismo. En la gráfica del error absoluto de torque se pueden observar oscilaciones de baja magnitud y alta frecuencia después del segundo 2. Estas oscilaciones son causadas por la resolución finita del μC que producen un efecto de *"chattering"*. El error absoluto de torque fue calculado usando la fórmula 73.

9.4.5.1. Conclusiones de la comparación de bucle cerrado

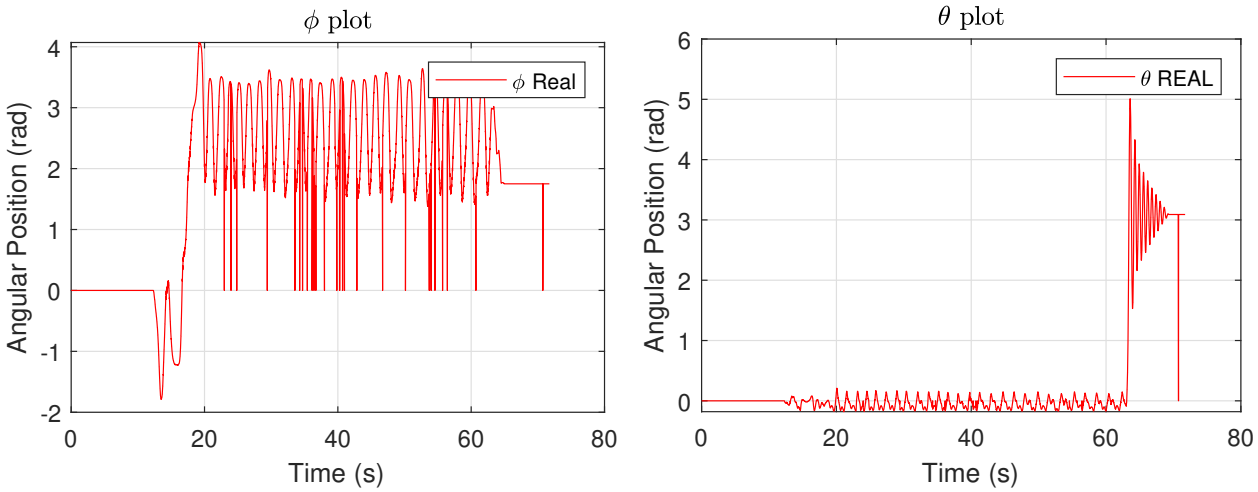
La implementación del algoritmo de control en el micro-controlador fue completamente exitosa, el error es insignificante comparado con las perturbaciones e incertidumbres de medición del péndulo real por lo que se puede afirmar que el controlador funciona correctamente.

Al agregar la 3ra comparación con el péndulo real el péndulo logra estabilizarse pero se queda oscilando alrededor de un punto arbitrario, a esto se le llama ciclo límite. El péndulo tiene un gran desempeño para evitar caerse a pesar de las oscilaciones de estado estacionario.

Se puede observar en la figura 57 que la prueba inicia alrededor del segundo 16, después el péndulo logra equilibrar exitosamente θ con vibraciones de baja magnitud, menores a 0,3 rad, mientras que el ángulo ϕ oscila bastante alrededor de π. El péndulo se detuvo por el usuario alrededor del segundo 60.

Las oscilaciones de ϕ parecen ser causadas por el cable del codificador rotatorio, este cable tiene una masa y rigidez considerable. El cable parece estar formando un oscilador debido a sus propiedades elásticas.

Como el controlador no permite que el péndulo se caiga cuando el cable "jala" al péndulo, el controlador lo estabiliza en sentido opuesto, esto ocasiona que el cable se "comprima" y empuje al péndulo, lo que a su vez ocasiona que el controlador actúe en sentido opuesto y el ciclo se repite.



(a) Ángulo ϕ

(b) Ángulo θ

Figura 57: Péndulo real estabilizado oscilando cerca de π. El péndulo se detuvo por el usuario alrededor de $t = 60$.

También se detectó que el plano de la mesa donde está el péndulo no es completamente perpendicular a la dirección de la fuerza gravitatoria, esto podría provocar que sea imposible equilibrar el péndulo para ciertos valores de ϕ donde el plano está inclinado. Esto no parece ser tan influyente como el problema del cable pero es posible que también contribuya negativamente.

Con toda esta experimentación se logró comprobar que el controlador cumple con los objetivos de control (ver sec. 8.2) y que fue implementado correctamente. También se programó un simulador *Hardware-in-the-loop* que tiene grandes ventajas, se lograron identificar problemas sutiles como la inclinación de la base del péndulo rotatorio, pequeños errores de truncamiento, se detectó que el cable tiene un gran efecto negativo en la estabilización del péndulo y se logró estabilizar el péndulo por tiempo indefinido hasta que el usuario lo detiene.

NOTA: Experimentos posteriores comprobaron que las oscilaciones son causadas en mayor parte por la fricción no lineal del eje θ. Es necesario un controlador no-lineal o algún método de controlador que tenga en cuenta la fricción no-lineal.

9.4.6. Diseño de experimento: Experimento con HIL en bucle cerrado con simulación de encoders

Es cómodo simular el péndulo en la PC utilizando *MATLAB* y cerrar el bucle vía serial con el μC para comprobar la implementación y correcto funcionamiento del controlador. A pesar de esto, existen fenómenos que ocurren al utilizar sensores (como el codificador rotatorio) para medir los estados del sistema. Estos fenómenos pueden modificar el comportamiento esperado del controlador.

Algunos de los fenómenos más importantes son la discretización de los estados y las implicaciones que esto tiene al estimar la velocidad. Hay más información al respecto en la sección 8.1.

Para realizar una simulación que tenga en cuenta estos fenómenos se implementará el mismo *Hardware-in-the-loop* que se ha estado usando en *MATLAB* pero implementado en un 2do μC.

El procedimiento será el siguiente:

- Se leen entradas del sistema (voltaje PWM del motor)
- Se calculan estados del sistema iterando las ecuaciones en espacio de estados discreto utilizando las ecuaciones 37 y 38 implementadas en el μC.
- Se mapean posiciones angulares a pulsos de codificadores rotatorios
- Se repite procedimiento

Nótese que ahora el bucle entre el controlador y la planta no se cierra mediante comunicación serial como en el caso del *Hardware-in-the-loop* con *MATLAB*. Ahora se cierra el bucle mediante la señal de voltaje/torque del motor y las señales de los codificadores rotatorios. Tal como se hace con la planta real.

La señal de entrada es un voltaje PWM que cuya frecuencia y ciclo de trabajo se lee con un timer del μC operando en modo "entrada PWM". Las salidas son las posiciones angulares que son señales de cuadratura discretas en los pines digitales del μC.

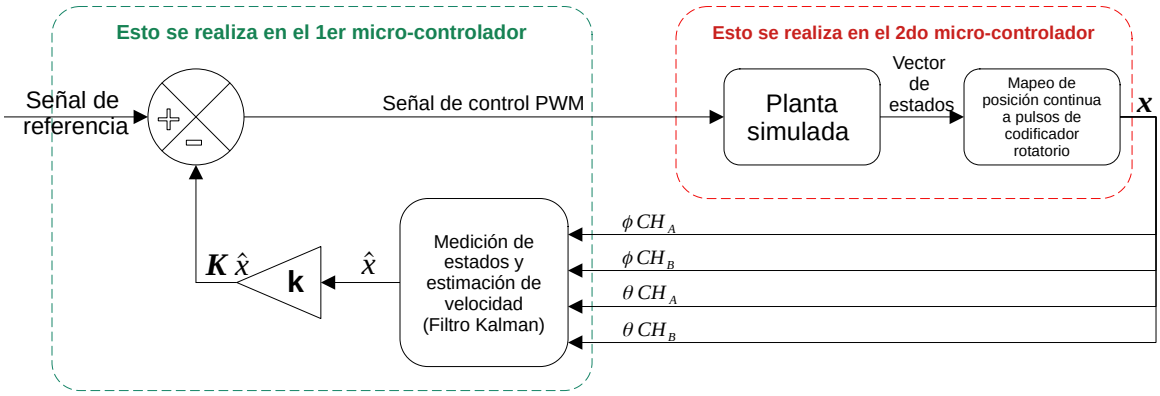


Figura 58: Diagrama a bloques del *Hardware-in-the-loop* en el μC
Diagrama a bloques del bucle de retroalimentación del *Hardware-in-the-loop* simulado por el μC.

Las funciones matemáticas que mapean la posición a una señal de cuadratura discreta; tal como el codificador rotatorio real lo hace; son las siguientes:

$$CH_A = sign(sen(p)) \tag{74}$$

y

$$CH_B = sign(cos(p)) \tag{75}$$

Donde:

$$p = 0{,}5N\phi \tag{76}$$

y CH_A y CH_B son respectivamente el canal A y el canal B del codificador rotatorio simulado.

La variable N representa los pulsos por revolución (ppr) del codificador rotatorio y ϕ representa una posición angular continua.

Nótese que el péndulo consta de 2 codificadores rotatorios por lo que habrá 4 señales CH.

Es importante recordar que para poder simular adecuadamente el comportamiento de los codificadores rotatorios se debe cumplir con la siguiente ecuación:

$$f_{HIL} >= f_{nyquist} \tag{77}$$

Donde:

$$f_{nyquist} = \frac{2\phi_{max}N}{2\pi} \tag{78}$$

Donde ϕ_{max} es la máxima velocidad angular que el péndulo alcanzará; si esa velocidad se supera se necesitará aumentar f_{HIL} de lo contrario CH_A y CH_B no representarán fielmente el comportamiento del codificador rotatorio real.

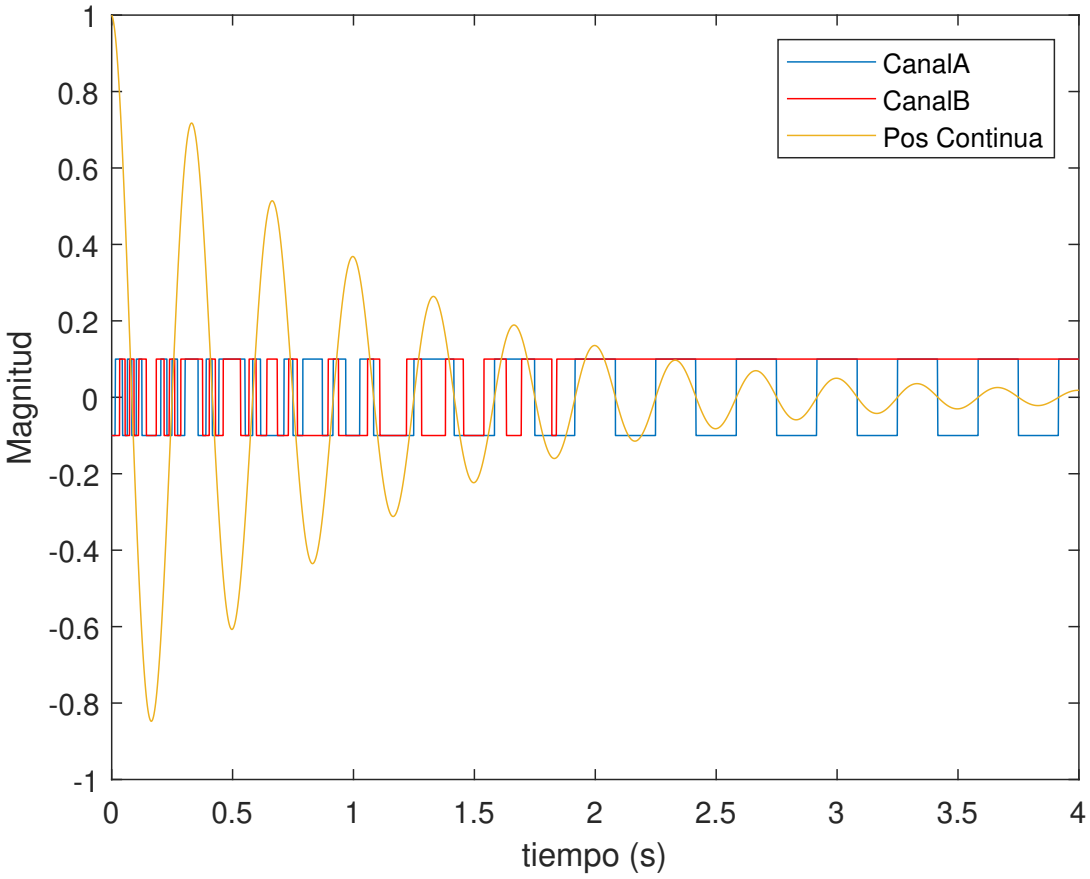


Figura 59: Mapeo de posición continua a pulsos de cuadratura.

En la figura 59 se muestra un ejemplo de mapeo de posición continua a pulsos de cuadratura utilizando las ecuaciones mencionadas previamente. Se simula un sinusoide de 3 Hz exponencialmente decadente que representa la posición angular en radianes. En la figura se simula un encoder con $N = 20$ ppr para facilidad de observación de la gráfica. También se cambió la magnitud de los canales A y B para facilidad de observación.

Tomando en cuenta todo lo anteriormente mencionado la planta se simulará en un 2do μC con una frecuencia de simulación f_{HIL} de 10 kHz y la planta *Hardware-in-the-loop* será controlada en bucle cerrado con el 1er μC con la frecuencia de muestreo (del controlador) y frecuencias de PWM siguientes:

Frec. Muestreo	Frec. PWM
10,000 Hz	20 kHz

Tabla 3: Frecuencias de muestreo y de PWM del controlador.

Las condiciones iniciales son las siguientes:

$$\boldsymbol{x_0} = [1\ 0\ 0\ 0] \tag{79}$$

Se busca llevar al sistema al estado de referencia:

$$\boldsymbol{x_{REF}} = [0\ 0\ 0\ 0] \tag{80}$$

El vector de retroalimentación de estados es el siguiente:

$$\boldsymbol{K_d} = [-0{,}3059\ \ -0{,}3653\ \ -6{,}5878\ \ -0{,}6136] \tag{81}$$

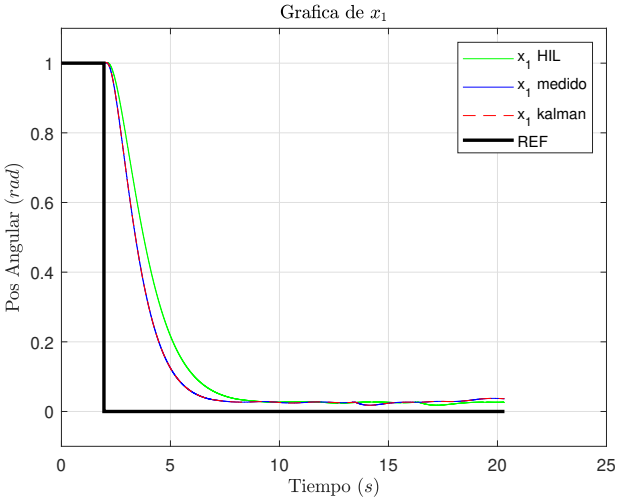
Para estimar las velocidades se utilizó un filtro Kalman con las matrices:

$$\hat{\boldsymbol{A}} = \begin{bmatrix} 0{,}999887 & 0{,}000099 & 0{,}000264 & 0 \\ -0{,}000407 & 1 & 0{,}001871 & 0 \\ 0{,}000264 & 0 & 0{,}997333 & 0{,}000099 \\ 0{,}003562 & 0 & -0{,}017787 & 1 \end{bmatrix} \tag{82}$$

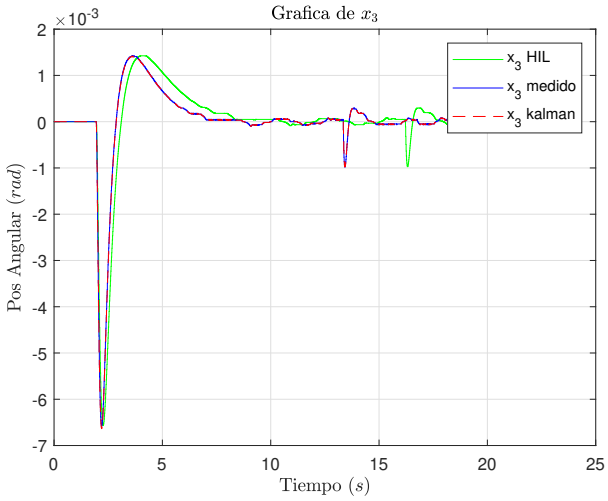
$$\hat{\boldsymbol{B}} = \begin{bmatrix} 0 & 0{,}0001 & 0 & -0{,}0003 & 0 \\ 0{,}1135 & 0{,}0004 & 0 & -0{,}0038 & 0 \\ 0 & -0{,}0003 & 0 & 0{,}0027 & 0 \\ -0{,}1738 & -0{,}0036 & 0 & 0{,}0360 & 0 \end{bmatrix} \tag{83}$$

La resolución de los encoders simulados para x_1 y x_3 es 48000 y 384000 *ppr* respectivamente.

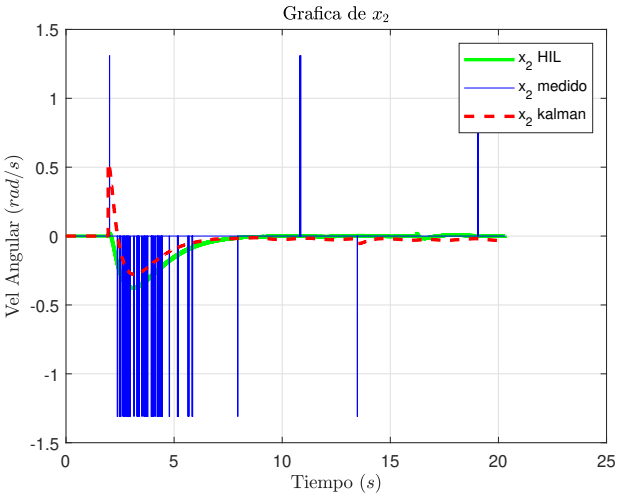
Los resultados fueron los siguientes:



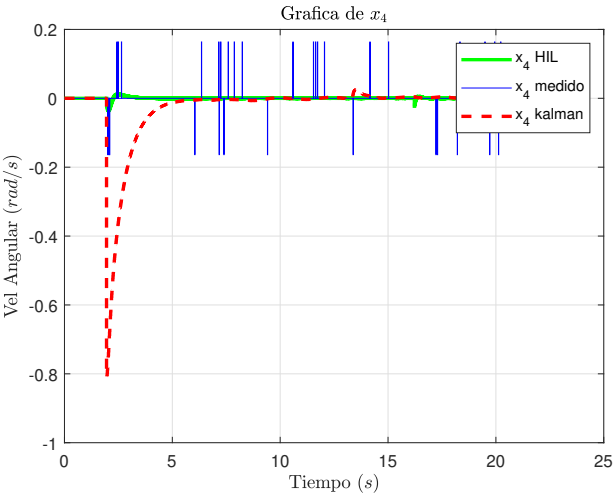
(a) Referencia y comparación de ángulos x_1.



(b) Comparación de ángulos x_3.



(c) Comparación de velocidades x_2.



(d) Comparación de velocidades x_4.

Figura 60: Comparación entre los estados ideales del *Hardware-in-the-loop*, los estados medidos por el controlador y los estados estimados utilizando el filtro Kalman.

9.4.6.1. Conclusiones del experimento

- En x_1 (ver Fig. 60a) existe error estacionario. Esto se debe a que el vector de retroalimentación, $\boldsymbol{K}$ (ver Ec. 81), tiene una ganancia mucho mayor para el error de la barra del péndulo (x_3) comparado con los demás estados. Cuando los estados se acercan a 0 (velocidades cercanas a 0) se forma un mínimo local en el que es prioritario mantener el error de x_3 en 0 a pesar del error de de x_1. A esto hay que agregar que la resolución finita de los encoders produce pequeños impulsos en la señal de control que dificultan llevar los estados a exactamente 0.

- En la gráfica se observa desfase entre las posiciones verdaderas y las medidas. Experimentos posteriores demostraron que el desfase no existe en realidad; este aparece debido a que no hay sincronización entre la comunicación serial de controlador→PC y de **HIL**→PC. Los estados de ambos μCs se mandan a la PC cada 10 ms **pero no necesariamente al mismo tiempo** ya que cada μC tiene su propio reloj. Se comprobó que las señales ocurren al mismo tiempo con el osciloscopio (ver Fig. 61). Esto no tiene ningún efecto en el experimento; simplemente hay que tenerlo en cuenta al graficar las señales en la PC.

- En x_2 (ver Fig. 60c) el filtro Kalman hace un excelente trabajo estimando la velocidad salvo los errores de estimación iniciales que se discuten en los próximos puntos. Sin el filtro Kalman el controlador se vuelve inestable debido a que los impulsos en las señales x_2 *medido* y x_4 *medido* producen impulsos de gran amplitud en la señal de control que desestabilizan el péndulo rotatorio.

- En x_2 (ver Fig. 60c) se puede observar que x_2 *medido* es una señal de impulsos debido al cálculo de velocidad con derivada discreta (ver Ec. 34) mientras que x_2 HIL y x_2 $Kalman$ son suaves y continuas. Esto es un resultado excelente ya que se evitan los impulsos en la señal de control (ver Fig. 58).

- En x_3 (ver Fig. 60b) la posición x_3 se mide y se estima correctamente respecto al valor verdadero x_3 HIL salvo los pequeños impulsos alrededor de los segundos 14 y 17. Teniendo en cuenta que la magnitud está en *mili rad* esas oscilaciones son despreciables.

- En x_4 (ver Fig. 60d) se observan impulsos al igual que en x_2 y de igual forma la estimación con el filtro Kalman elimina las discontinuidades y produce una señal x_4 *kalman* con un

comportamiento similar a x_4 HIL. Si se observan las señales de los filtros Kalman de x_4 y x_2 estas producen un impulso de gran magnitud al inicio del experimento (ver señal **REF** en Fig. 60a); esto se debe a que el filtro Kalman detecta la 1ra velocidad medida y al no tener más información no puede aproximar los estados reales. Este comportamiento es normal ya que así es la naturaleza matemática del filtro; conforme pasa el tiempo las aproximaciones son más cercanas al valor real del **HIL** tal como las gráficas muestran.

- Si se simulan encoders de menor resolución el sistema tiene un mayor error estacionario y oscilaciones de mayor magnitud. El comportamiento con bajas resoluciones es similar al del experimento de la figura 57a realizado con el péndulo rotatorio real, además de las conclusiones de ese experimento la naturaleza de la resolución de los encoders y las señales de cuadratura parecen influir de manera significativa la estabilidad del sistema. Conforme se reduce la resolución de los encoders este fenómeno aumenta hasta que el sistema se hace inestable. El sistema se hace inestable con encoders menores a 4000 ppr.

NOTA: El código en $C++$ de ambos μCs, el que controla la planta y el que la simula se encuentran respectivamente en los anexos 12.14 y 12.15.

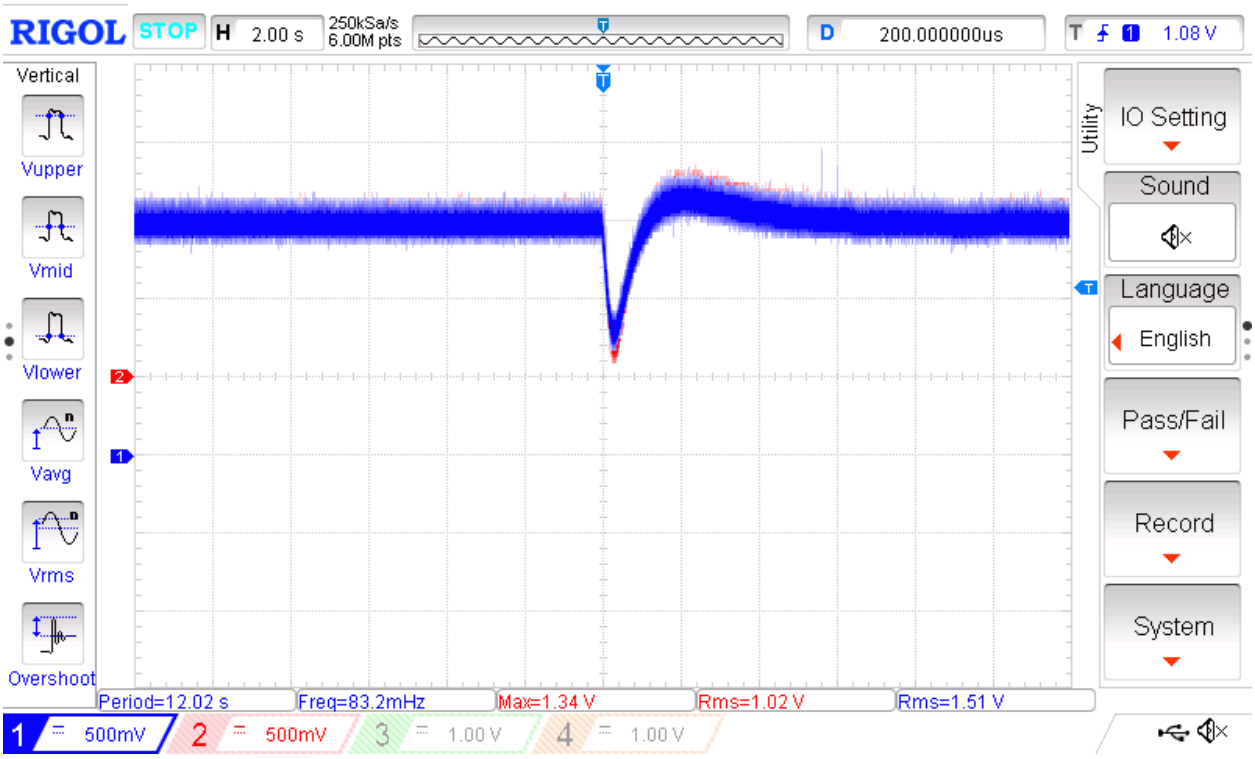


Figura 61: Medición de estado x_3 en el osciloscopio.

En la figura 61 se observa x_3 HIL en azul y x_3 *medida* rojo. En ambos μCs (**HIL** y controlador) además de enviar los estados por serial a la PC también se manda el ángulo x_3 a los convertidores digital-analógico. Las señales tienen $1V$ de offset ya que solo es posible reproducir valores positivos con las **DAC**s. Si se compara con la figura 60c se puede observar que el desfase es inexistente. La formula utilizada para el voltaje de las **DAC**s es la siguiente:

$$V_{DAC} = x_3 \cdot 124 + 1 \tag{84}$$

La constante 124 es para aprovechar al máximo la limitada resolución y rango de voltaje de las **DAC**s. A la señal se le suma $1V$ para poder mostrar ángulos negativos. Esto nos deja con una razón de 4E$-$3 *rad* por división del osciloscopio. En otras palabras, en el osciloscopio $1V$ corresponde a 0 *rad*.

9.4.7. Procedimientos optimizados

Realizar algunos procedimientos es una tarea laboriosa. Se optimizaron los siguientes procedimientos para tener una experimentación más eficiente:

- Programación y comunicación serial por el mismo puerto USB: Los datos de la medición son transmitidos por un puerto serial virtual mediante el mismo cable USB con el que se programa el μC. Esto elimina las desconexiones que antes se requerían para medir o programar. Esto ahorra mucho tiempo y permite tener una comunicación serial sin interrupciones.
- Medición automatizada: Se pueden realizar n experimentos de forma automatizada gracias a la comunicación serial entre la PC y el μC. Se manda un *string* de 'inicio' al μC para que este inicie el experimento y se capturen los datos en la PC. Se puede hacer cuantas veces sea requerido y al final se obtienen experimentos con la misma duración que no requieren preparación de datos. A estos experimentos se pueden promediar o se les puede calcular el error, varianza, etc.
- Estimación de parámetros del péndulo: Un proceso de vital importancia y que consumía mucho tiempo por los 2 puntos anteriores es estimar los parámetros del péndulo rotatorio. Ahora es posible estimar los parámetros rápidamente en *Simulink* utilizando las mediciones correspondientes obtenidas de forma automatizada. Esto es de gran ayuda ya que se pueden cambiar las características físicas del péndulo y encontrar los nuevos parámetros para modelarlo matemáticamente. Esto es de extrema importancia ya que otorga flexibilidad para modificar o cambiar las propiedades físicas del péndulo de ser necesario.

9.4.8. Código de colores

Para facilitar las mediciones con osciloscopio, multímetro o cualquier otro dispositivo es conveniente establecer un código de colores utilizado por cualquier circuito del péndulo, ya sea la driver del motor, conexiones al μC, sensores, etc.

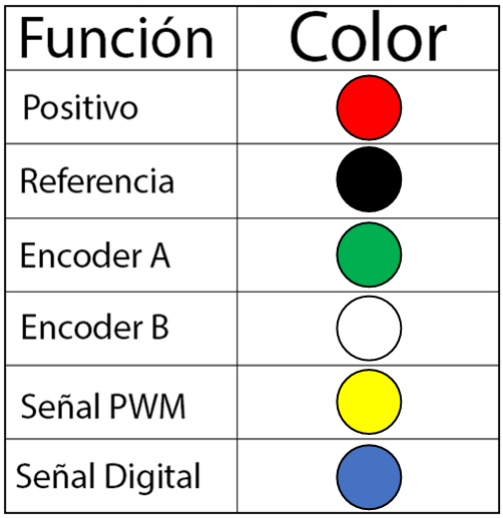


Figura 62: Código de colores que se utilizará para las conexiones eléctricas del péndulo.

Cabe destacar que el cable de señal digital (azul) no es ninguna señal digital específica, simplemente es el color de cable designado para cualquier señal digital, a excepción de las señales de los codificadores y PWM. Este color de cable se designó para LEDs, display LCD, señal serial, etc.

9.4.9. Comparación de estimación de velocidad

Comparación entre velocidad verdadera, medida, estimada con filtro Kalman y estimada con promedio móvil. La velocidad verdadera es enviada del **HIL** a la PC vía serial. La velocidad medida es calculada por el μC utilizando la ecuación 35. La velocidad Kalman utiliza el filtro Kalman (matrices 82 y 83) y el promedio móvil se calcula utilizando la ecuación 36.

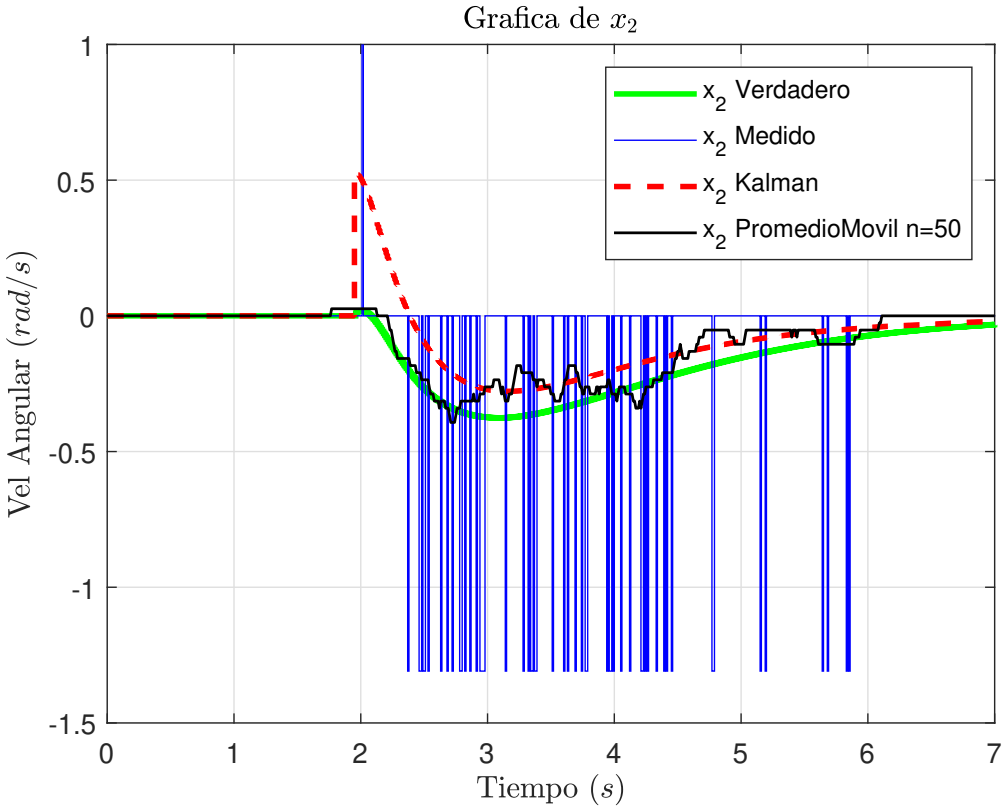


Figura 63: x_2 verdadera, medida y estimada con filtro Kalman y promedio móvil.

En la figura 63 se observa en azul la velocidad medida por el μC sin ningún procesamiento y en negro la misma velocidad pero con un promedio móvil de $n = 50$ muestras.

Se puede observar una mejora significativa en cuanto a la cantidad de ruido que existe en la señal de velocidad respecto a la medición directa (señal azul). La velocidad estimada con promedio móvil tiene un comportamiento aproximado al valor verdadero. La velocidad estimada por el filtro Kalman aproxima con mejor fidelidad la señal verdadera; conforme pasa el tiempo el filtro Kalman reduce el error de estimación, además, el comportamiento suave de la señal reduce impulsos en la señal de control. El promedio móvil es fácil de implementar pero el filtro Kalman da los mejores resultados. Si se tiene suficiente poder computacional y tiempo para implementación, el filtro Kalman es la mejor opción.

10. Resultados y discusión

Los resultados, discusiones y conclusiones se muestran individualmente para cada simulación y experimento de la investigación. A continuación se muestra una tabla donde se enumera cada sección donde hubo resultados, discusión y/o conclusiones. Cabe destacar que se puede hacer clic en los numerales para ir a la sección deseada.

También se tienen las conclusiones generales de la investigación y el trabajo futuro en la sección 11.

Índice	**Nombre de la sección**	**Numeral**
1	LQR	8.2.1
2	Modelo en *Simulink* del péndulo rotatorio	8.4
3	Backlash	8.5.1
4	Caracterización del backlash	8.5.1.1
5	Reducción de backlash utilizando DSP	8.5.2.1
6	Reducción de backlash re-diseñando el péndulo	8.5.2.2
7	Hardware-in-the-Loop	8.6.2
8	Comparación entre *Simulink* y *Hardware-in-the-loop*	8.6.2.1
9	Simulación de muestreo	8.6.3
10	Simulación del backlash	8.6.4
11	Estimación de parámetros de carga	9.3.1.4
12	Diseño de experimento: Experimento de similitud en bucle abierto	9.4.4
13	Conclusiones de la comparación de bucle abierto	9.4.4.1
14	Diseño de experimento: Experimento de similitud en bucle cerrado	9.4.5
15	Conclusiones de la comparación de bucle abierto	9.4.5.1
16	Diseño de experimento: Experimento con **HIL** en bucle cerrado con simulación de encoders	9.4.6
17	Conclusiones del experimento	9.4.6.1
18	Comparación de estimación de velocidad	9.4.9

Tabla 4: Tabla de resultados y discusiones.

11. Conclusiones generales y trabajo futuro

11.1. Conclusiones

En esta tesis se tuvo la hipótesis de simular el péndulo rotatorio mediante el uso del *Hardware-in-the-loop* para analizar y verificar los algoritmos de control al ser implementados en el μC. El resultado de la investigación fue exitoso por las siguientes razones.

El modelo matemático y los parámetros físicos estimados y medidos del péndulo rotatorio modelan correctamente en ambos dominios de tiempo, discreto y continuo, el comportamiento real del sistema sin importar si se simula utilizando el *Hardware-in-the-loop*, *Simulink*, *C++* o *MATLAB*.

Se construyeron plantas reales impresas en 3D para hacer comparaciones con las simulaciones. Las plantas ayudaron a tener en cuenta fenómenos que solo existen cuando se construye físicamente el sistema. Con el *Hardware-in-the-loop* se puede verificar el correcto funcionamiento del controlador aunque la planta real no tenga comportamiento matemático ideal. Esto permite aislar, detectar y corregir problemas en la construcción física de la planta. Eventualmente la planta convergerá a un comportamiento similar o muy similar al de la planta modelada matemáticamente.

A lo largo de la sección de implementación del algoritmo de control (ver sec. 9) se verifica la implementación del algoritmo de control tal como se planteó en la hipótesis. También se pudo tener acceso a variables que no son directamente observables en la planta real utilizando las ventajas del *Hardware-in-the-loop*.

Los problemas de medición discreta de la velocidad del péndulo rotatorio planteados en la sección 2 fueron simulados (en software) al principio de la investigación (ver sec. 8.6.3) y posteriormente replicados en Hardware con el *Hardware-in-the-loop* (ver sec. 9.4.9). También se experimentó con soluciones a este fenómeno como el promedio móvil y el filtro Kalman tal como la figura 63 muestra.

El *Hardware-in-the-loop* logró simular el péndulo rotatorio resolviendo las ecuaciones dinámicas y actualizando el estado del sistema cada 100 μs. La simulación de mayor fidelidad incluye también la simulación de la señales de cuadratura de ambos codificadores rotatorios. Esta simulación es capaz de replicar los fenómenos de medición discreta de velocidad por parte del controlador.

El péndulo rotatorio real y simulado se estabilizan (ver sec. 9.4.5) exitosamente. Las diferencias que existen en el comportamiento del sistema real y el simulado se deben a fenómenos no lineales que existen al construir físicamente la planta real. Algunos de estos fenómenos son backlash, precesión de los ejes, deslizamiento de la correa de transmisión, resonancias mecánicas del chasis del péndulo, no-linealidad del motor, etc. Estos fenómenos no pueden ser eliminados utilizando el controlador actual debido a que el controlador es lineal.

El péndulo rotatorio simulado con el *Hardware-in-the-loop* con sensores ideales fue estabilizado exitosamente utilizando el controlador LQR implementado en el μC y tuvo el mismo comportamiento que el sistema y controlador simulados completamente en *Simulink*. Al agregar la simulación de pulsos de cuadratura (sensores no ideales) se replicaron los fenómenos que ocurren a bajas velocidades en la planta real.

En general en esta investigación se cumplieron exitosamente la hipótesis, los objetivos generales y los objetivos específicos.

11.2. Trabajo futuro

El trabajo de esta investigación abarca los campos de la electrónica, física clásica, control, mecánica y software.

Principalmente existen 3 objetos de estudio en esta investigación. El péndulo rotatorio, el controlador y el *Hardware-in-the-loop*. En cada uno de estos objetos de estudio existe el siguiente trabajo futuro:

Para el péndulo rotatorio el trabajo futuro puede ser:

- Fabricar un péndulo de metal, con engranes anti-backlash o con un motor *direct drive*. 3 sensores angulares de alta resolución, uno en cada de giro y otro sensor más en el eje del motor. El último sensor es con fines de redundancia para poder cuantificar el deslizamiento que pueda existir entre el eje del motor y el eje ϕ.
- Construir un péndulo rotatorio que incluya una base de metal plana de masa mucho mayor a la del péndulo que esté perfectamente balanceada y perpendicular al vector de gravedad.
- Utilizar rodamientos cerámicos de baja fricción para que los fenómenos de fricción no lineal sean mínimos o incluso despreciables.

Para el controlador el trabajo futuro puede ser:

- Diseñar un controlador no-lineal que sea capaz de estabilizar el péndulo rotatorio de forma exitosa incluso con los fenómenos no-lineales mencionados anteriormente.
- Diseñar un controlador que automáticamente estime los parámetros de la planta a partir del modelo matemático del péndulo y el comportamiento esperado. Esto se puede realizar utilizando algún algoritmo de optimización que encuentre los parámetros que hagan coincidir el comportamiento del modelo matemático con el comportamiento medido.
- Diseñar un controlador no-lineal de manera que se pueda estabilizar el péndulo con condiciones iniciales lejanas al punto de equilibrio. Un controlador de este tipo podría realizar el movimiento de *"swing-up"* de $\theta = \pi$ hasta $\theta = 0$ de forma naturalmente matemática, es decir, sin ningún tipo de discontinuidad o caso especial para "levantar" el péndulo.

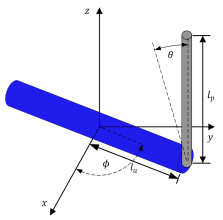
Para el *Hardware-in-the-loop* el trabajo futuro puede ser:

- Implementar el **HIL** en un **FPGA** de manera que la simulación se itere en el orden de los nanosegundos para poder simular fenómenos de alta frecuencia como vibraciones, rebotes, etc.
- Abstraer el **HIL** para poder utilizarlo en cualquier simulación siempre y cuando se tengan las ecuaciones diferenciales que describe la dinámica del sistema o las matrices de estado. El **HIL** abstraído puede simular cualquier sistema incluyendo sensores con el unico requerimiento de que la resolución temporal y la resolución del tipo de datos (float, int, byte) sea suficiente dependiendo del problema a simular.
- Con un **FPGA** y un μC es posible hacer un simulador en tiempo real con diferentes tipos de salidas, digitales y analógicas, que permitan medir información o estados del sistema con equipo de instrumentación de alta velocidad como osciloscopios, analizadores lógicos, etc. También el simulador puede tener entradas analógicas y digitales que permitan introducir información al sistema o cerrar el bucle en tiempo real. La idea es que este simulador en hardware tenga una gran cantidad de entradas y salidas en tiempo real con gran ancho de banda para permitir al usuario enfocarse en la experimentación.

12. Anexos

12.1. Convenciones

Tabla 5: Tabla de convenciones.

Índice	Información de la convención	Notas
1	Se utilizan unidades del **S.I** salvo que se mencione lo contrario	
2	Se usarán indistintamente los términos "péndulo rotatorio", "péndulo de Furuta" y "péndulo" para referirse al péndulo de la figura 1	
3	Se usará el término "péndulo simple" para referirse al péndulo no-rotatorio de un solo ángulo; similar al de un viejo reloj	
4	Se define como "brazo" el cilindro azul que gira en el plano XY	Ver figura 1. En la práctica el brazo fue impreso en 3D con PLA
5	Se define como "barra" el cilindro gris conectado al final del brazo del péndulo	Ver figura 1. En la práctica la barra es una barra de metal
6	El vector gravitatorio apunta en dirección $-Z$	El vector de gravedad apunta en dirección $-Z$ con una magnitud de 9,81 m/s^2
7	Se define la dirección "arriba" como la dirección $+Z$	
8	Se define la dirección "abajo" como la dirección $-Z$	

Continúa en siguiente página

Tabla 5 – Continuación de página anterior

Índice	Información de la convención	Notas
9	Se define el vector de estado $\boldsymbol{x} = [x_1\ x_2\ x_3\ x_4] = [\phi\ \dot{\phi}\ \theta\ \dot{\theta}\ ]$	Se utiliza la notación x_n en el contexto de espacio de estados
10	Se definen los estados ϕ y x_1 como la posición angular del brazo del péndulo.	En la figura 1 se muestra ϕ
11	Se definen los estados $\dot{\phi}$ y x_2 como la velocidad angular del brazo del péndulo.	
12	Se definen los estados θ y x_3 como la posición angular de la barra del péndulo.	En la figura 1 se muestra θ
13	Se definen los estados $\dot{\theta}$ y x_4 como la velocidad angular del brazo del péndulo.	
14	Aplicar un voltaje positivo al motor genera rotación en sentido anti-horario visto desde arriba del péndulo	
15	El ángulo θ incrementa positivamente si la barra del péndulo gira en sentido antihorario tal como se muestra en la figura 1	Ver ángulo θ en la figura 1
16	Se define como "péndulo real" o "péndulo físico" al péndulo rotatorio de plástico PLA impreso 3D con el que se realizan los experimentos	
17	Se define como "péndulo virtual", "péndulo *Hardware-in-the-loop*" o "clon virtual" al péndulo que se está simulando en el μC o en MATLAB	

12.2. Acrónimos

Tabla 6: Tabla de acrónimos.

Índice	Acrónimo	Descripción
1	**AC**	Alternating Current
2	**ADC**	Analog to Digital Converter
3	**CNC**	Computer Numerical Control
4	**COM**	Communication Port
5	**CPR**	Counts per revolution
6	**CPU**	Central Processing Unit
7	**DAC**	Digital to analog Converter
8	**DC**	Direct Current
9	**DMA**	Direct Memory Access
10	**DSP**	Digital Signal Processing
11	**FIR**	Finite Impulse Response
12	**HIL**	Hardware-in-the-Loop
13	$\boldsymbol{I^2C}$	Inter-integrated Circuit
14	**LCD**	Liquid Crystal Display
15	**LDO**	Low drop-out
16	**LED**	Light Emitting Diode
17	**LQG**	Linear Quadratic Gaussian
18	**LQR**	Linear Quadratric Regulator
19	μ**C**	Microcontroller unit
20	**ODE**	Ordinary Differential Equation(s)
21	**PC**	Personal Computer
22	**PLA**	Polylactic Acid
23	**PWM**	Pulse Width Modulation
24	**REF**	Reference Signal
25	**RK4**	Runge-Kutta 4th Order Method

Continúa en siguiente página

Tabla 6 – Continuación de página anterior

Índice	Acrónimo	Descripción
26	**RTC**	Real-time Clock
27	**SPI**	Serial Peripheral Interface
28	**SS**	State Space
29	**USB**	Universal Serial Bus
30	**VCOM**	Virtual Communication Port

12.3. Lista de ecuaciones

12.4. Lista de marcas

A continuación se muestra una tabla con la lista de las marcas utilizada para esta investigación. Las descripciones enlazan a la página web de la marca si se le da clic.

Tabla 7: Lista de marcas.

Índice	**Marca**	**Descripción**
1	*Maxon*	Fabricante de motores Suizo
2	*Pololu*	Minorista de componentes electrónicos y de ingeniería
3	*MATLAB*	Plataforma de programación y cálculo numérico
4	*Simulink*	Entorno de diseño de sistemas multidominio con diagramas de bloque
5	*STM*	Compañía que desarrolla circuitos integrados

12.5. Diagrama de tiempos del cliente

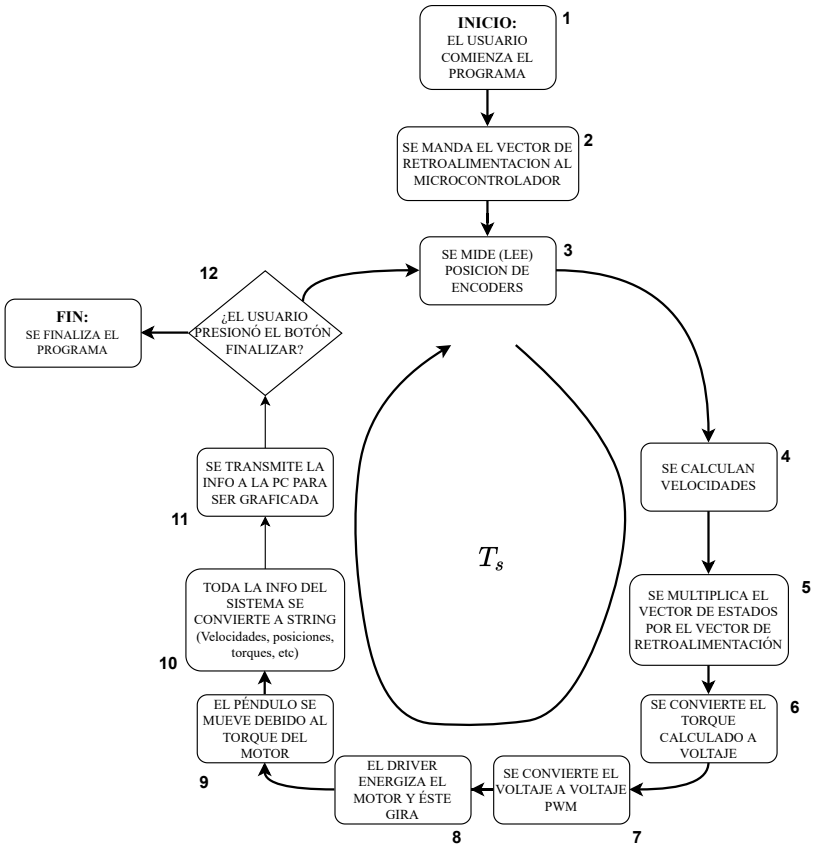


Figura 64: Diagrama de tiempos del código en el micro-controlador.

12.6. Código del Anfitrión

```
%Se limpian variables:
clc
clear all
cla reset
estado = 0;
inicio = 0;

LS=60;       %Se establece Limite superior en s
Ts = 0.010; %Se establece Periodo de muestreo del MCU
Fs = 1/Ts;   %Se calcula f de muestreo
Vrang=200;   %Rango Vertical (Magnitud grafica)
title('Vel(t)'); %Nombre de la grafica
xlabel('ms');     %Nombre del eje X
% Eje y de la izquierda.
ylabel('Magnitud');      %Nombre
set(gca,'ycolor','r')    %Color
set(gca,'YLim',[-Vrang Vrang]) %Rango
set(gca,'XLim',[0 LS*Fs])      %Dominio

instrfind %Se cierran conexiones serial previas
if inicio == 0
if ~isempty(instrfind)
    fclose(instrfind);
    delete(instrfind);
end

s = serial('COM3'); %Se especifica puerto serial del MCU
set(s,'BaudRate',1382400);
set(s, 'TimeOut', 60)
fopen(s);

axes(gca) %Se configuran graficas en tiempo real:
if estado == 0
h1 = animatedline('Color','r');
h2 = animatedline('Color','b');
h3 = animatedline('Color','m');
h4 = animatedline('Color','c');
h5 = animatedline('Color','g');
h6 = animatedline('Color','k');
legend('Pos_{Mot}', 'Vel_{Motor}','Pos_{Enc}','Vel_{Enc}','Filtro','ADC');
set(gca,'ycolor','b')
estado =1;
end
parar=0;
inicio = 1;
set(gca,'XLim',[0 LS*Fs]) %Se situa grafica al inicio y se limpia grafica:
clearpoints(h1)
clearpoints(h2)
clearpoints(h3)
clearpoints(h4)
clearpoints(h5)
clearpoints(h6)

%Estos bucles leen los datos recibidos por el MCU.
%Cada Ts segundos se recibe nueva informacion y se grafica
%Cuando la grafica se llena se borra y se comienza una nueva con nuevos valores en el eje
for z = 1:100        %LS*z
    if parar == 1
        break
    end
    for k = 1:LS*Fs
        SerialStr = fscanf(s, '%f'); %Se reciben datos...
        if length(SerialStr) == 6
        %Se crear vector tiempo y se leen datos...
        t(1,k+LS*Fs*(z-1))=(k-1)+LS*Fs*(z-1);
        pos_MOT(1,k+LS*Fs*(z-1))=(SerialStr(1,1));
        vel_MOT(1,k+LS*Fs*(z-1))=(SerialStr(1,2));
        pos_ENC(1,k+LS*Fs*(z-1))=(SerialStr(1,3));
        vel_ENC(1,k+LS*Fs*(z-1))=(SerialStr(1,4));
        filtro(1,k+LS*Fs*(z-1))=(SerialStr(1,5));
```

```
71          ADC(1,k+LS*Fs*(z-1))=(SerialStr(1,6));
72          %Se agregan los datos recibidos por serial:
73          addpoints(h1,(k+LS*Fs*(z-1)),((SerialStr(1,1))));   %Posicion Mot rads
74          addpoints(h2,(k+LS*Fs*(z-1)),SerialStr(1,2));       %Velocidad Motor
75          addpoints(h3,(k+LS*Fs*(z-1)),((SerialStr(1,3))));   %Posicion Enc rads
76          addpoints(h4,(k+LS*Fs*(z-1)),SerialStr(1,4));       %Velocidad Encoder
77          addpoints(h5,(k+LS*Fs*(z-1)),(SerialStr(1,5)));     %Filtro
78          addpoints(h6,(k+LS*Fs*(z-1)),(SerialStr(1,5)));     %K
79          if parar == 1    %Si parar==1 el programa se detiene y se cierra.
80              break
81          end
82          drawnow limitrate
83          end
84      end
85  if parar ~= 1
86  clearpoints(h1)
87  clearpoints(h2)
88  clearpoints(h3)
89  clearpoints(h4)
90  clearpoints(h5)
91  clearpoints(h6)
92  set(gca,'XLim',[LS*Fs*z LS*Fs*z+LS*Fs])
93  end
94  end
95
96  fclose(s);
97  delete(s)
98  clear s
99  end
```

Figura 65: Código en MATLAB del 'data logger'.

12.7. Código del Cliente

```
/* USER CODE BEGIN*/
char msj[] = "";
//String que se utilizara para enviar pos y vels a la PC
                float dp_ENC = 0;
                //Diferencia de pasos del Encoder (del pendulo) =0
                float dp_MOT = 0;
                //Diferencia de pasos del motor =0
                float pos_ENC=32768,vel_ENC=0,pos_ENC_rads=0,pos_ENC_grad=0;
                //Posiciones centradas en (2^16)/2 para evitar overflow.
                float pos_MOT=32768,ace_MOT=0,vel_MOT=0,vel_MOTante=0,pos_MOT_rads=0,pos_MOT_grad=0;
                //Posiciones centradas y velocidades iniciales 0.
                char texto[] = "ESTADO INICIAL";
                //Se envia string por COM para saber cuando inicia todo.
                float voltaje=0,PWM=0,Duty=0;
                //Condiciones iniciales =0.
                float Km=0.608,Ra=2.54,J=0.0116;
                //Constantes de Torque, Resistencia y momento de inercia del motor.
                unsigned int PWMint=0;
                //Voltaje PWM=0.

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
        //Funcion por interrupcion que se ejecuta cada 1ms.
{
        if(htim->Instance==TIM3) {
                //Se comprueba que sea la interrucion por el timer correcto.
                dp_MOT=((TIM1->CNT)-pos_MOT);
                //Se calcula diferencia de pos del motor en pasos
                dp_ENC=((TIM2->CNT)-pos_ENC);
                //Se calcula diferencia de pos del encoder en pasos
                pos_MOT = (TIM1->CNT);
                //Se lee pos del motor en pasos
                pos_ENC = (TIM2->CNT);
                //Se lee pos del encoder en pasos
                pos_MOT_rads = (pos_MOT-32768)/652;
                //Se descentraliza y se convierte a rads/s. 652 Por la resolucion del encoder del motor.
                pos_ENC_rads = (pos_ENC-32768)/637;
                //Se descentraliza y se convierte a rads/s. 637 Por la resolucion del encoder del pendulo.
                pos_MOT_grad = pos_MOT_rads*57.3;
                //Se convierte a grados para debugging.
                pos_ENC_grad = pos_ENC_rads*57.3;
                //Se convierte a grados para debugging.
                vel_MOT = (dp_MOT/652)*Fs;
                //Se calcula vel motor en rads/seg      // *1000 porque Fs=1000Hz ;; Ts=1ms
                vel_ENC = (dp_ENC/637)*Fs;
                //Se calcula vel del encoder en rads/seg        // *1000 porque Fs=1000Hz ;; Ts=1ms
                voltaje = -1*pos_MOT_rads-2.159*vel_MOT-40.97*pos_ENC_rads-3.897*vel_ENC;
                //Senal retroalimentacion en Volts-DC (Calculada con teoria de control)
                PWM = fabs(voltaje*4*Fs);
                //Volts-DC a Volts-PWM. 4*Fs por que 12V = 48*Fs bits del timer.
                PWMint = (int)PWM;
                //Se convierte float a int para el timer.
                Duty = (PWMint/4.8*Fs);
                //Se calcula el duty cycle para debugging.
                if (PWMint>48*Fs)
                //Se evita desbordamiento del timer.
                {       PWMint = 48*Fs; }
                if (pos_ENC>33268 || pos_ENC<32268) {
                //Si se exceden +-45grados del pendulo se apaga automaticamente por seguridad.
                PWMint = 0; }
                TIM4->CCR1 = PWMint;
                //12V => 48*Fs en CCR1 = maxV           //12V = 48*Fs = Timer lleno 100% Duty cycle.
                direccion(voltaje);
                //Cambia la polaridad del motor

                HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_10);
                //LED Heartbeat para medir frec del ciclo
                sprintf(texto, "%2.3f %2.3f %2.3f %2.3f %3.3f \r\n", //Continua abajo..../
          (float)pos_MOT_rads, (float)vel_MOT, (float)pos_ENC_rads, (float)vel_ENC, (float)voltaje);
                // ^ Prepara string a transmitir ^
                CDC_Transmit_FS((uint8_t *)texto, strlen(texto));
```

```
71                  //Se transmite string via USB CDC VCOM
72 }
73          }
74 void direccion(int filtro)
75          //Se cambia la direccion del motor dependiendo del signo.
76 {
77          if (filtro < 0) {
78                  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET);
79                  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_RESET); }
80                  else {
81                  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_RESET);
82                  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_SET); }
83          }
84 /* USER CODE END*/
```

Figura 66: Código en C++ del μC de la retroalimentación.

12.8. Código del medidor de velocidad simple

```
1  float dp_ENC = 0;
2  //Diferencia de pasos del Encoder (del pendulo) = 0
3  float dp_MOT = 0;
4  //Diferencia de pasos del motor = 0
5  float pos_ENC = 32768, vel_ENC = 0, pos_ENC_rads = 0,
6        pos_ENC_grad = 0;
7  //Posiciones centradas en (2^16)/2 para evitar overflow.
8  float pos_MOT = 32768, ace_MOT = 0, vel_MOT = 0,
9        vel_MOTante = 0, pos_MOT_rads =   0,
10       pos_MOT_grad = 0;
11 //Posiciones centradas y velocidades iniciales 0.
12
13 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
14 //Funcion por interrupcion que se ejecuta cada 1ms.
15 {
16         if (htim->Instance == TIM3) {
17                 //Se comprueba que sea la interrucion por...
18                 //el timer correcto.
19                 dp_MOT = ((TIM1->CNT) - pos_MOT);
20                 //Se calcula diferencia de pos del motor en pasos
21                 dp_ENC = ((TIM2->CNT) - pos_ENC);
22                 //Se calcula diferencia de pos del encoder en pasos
23                 pos_MOT = (TIM1->CNT);
24                 //Se lee pos del motor en pasos
25                 pos_ENC = (TIM2->CNT);
26                 //Se lee pos del encoder en pasos
27                 pos_MOT_rads = (pos_MOT - 32768) / 652;
28                 //Se descentraliza y se convierte a rads/s...
29                 //652 Por la res del encoder del motor.
30                 pos_ENC_rads = (pos_ENC - 32768) / 637;
31                 //Se descentraliza y se convierte a rads/s...
32                 //637 Por la res del encoder del pendulo.
33                 pos_MOT_grad = pos_MOT_rads * 57.3;
34                 //Se convierte a grados para debugging.
35                 pos_ENC_grad = pos_ENC_rads * 57.3;
36                 //Se convierte a grados para debugging.
37                 vel_MOT = (dp_MOT / 652) * Fs;
38                 //Se calcula vel motor en rads/seg
39                 vel_ENC = (dp_ENC / 637) * Fs;
40                 //Se calcula vel encoder en rads/seg
41         }
42 }
```

Figura 67: Código que calcula la velocidad.

12.9. Código del promedio móvil

```
1  N = 10; //Muestras a promediar
2  n = 0;  //Indice enpieza en 0
3
4  float prommovil(float vel) {//Funcion promedio movil
5          x[n] = vel;      //Se recibe el ultimo valor de x[n]
6          n++;             //Se incrementa indice
7          if (n > N) {//Si indice > N se vuelve a 0
8                  n = 0;
9          }
10         P = 0;           //Se limpia promedio anterior
11         for (int k = 0; k < N; k++) {//Sumatoria de n=0 a n=N-1
12                 P = P + x[k];
13         }
14         P = P / N;       // 1/N
15         return P;        //Se retorna el promedio
16 }
```

Figura 68: Código del promedio móvil.

12.10. Codigo de Simulación de Backlash

```
%Este programa simula el backlash a la salida de...
%engranajes con backlash respecto a la senal de entrada.

clc
n=720;                  %Numero de iteraciones
t=linspace(0,2,n);      %Vector de tiempo
in=sin(2*pi*1*t);       %Senal de entrada

indot=diff(in)/(2*pi)*n/2;  %derivada de la entrada
indot(1,end+1)=indot(1,end);%correccion de tamano


m=1/1;                  %Relacion de engranajes
CL=sin(10*pi/180);      %10 grados de backlash
CR=CL;                  %Misma cantidad de backlash...
                        %para ambos sentidos de giro

out=zeros(1,n);         %Se pre-asignan n espacios en memoria
for i=2:n-1

if      indot(1,i)<0 && out(1,i-1)>m*(in(1,i)+CL)
   out(1,i)=m*(in(1,i)+CL);
elseif indot(1,i)>0 && out(1,i-1)<m*(in(1,i)-CR)
   out(1,i)=m*(in(1,i)-CR);
else
   out(1,i)=out(1,i-1);
end

end
plot(t,in)

%Configuracion de la grafica...
hold on
plot(t,out)
title('Backlash')
xlabel('Tiempo segs');
ylabel('Posicion Angular');
legend('Entrada','Salida');
```

Figura 69: Código en MATLAB de la simulación de backlash.

12.11. Filtro FIR rechaza backlash

Vector del filtro orden 321. Los índices incrementan de 0 a 9 en el primer renglón.

0.000303	0.000304	0.000298	0.000285	0.000264	0.000236	0.000201	0.000159	0.000111	5.75E-05
-5.15E-18	-6.06E-05	-0.00012	-0.00019	-0.00025	-0.00031	-0.00036	-0.00041	-0.00045	-0.00048
-0.0005	-0.0005	-0.0005	-0.00048	-0.00045	-0.0004	-0.00034	-0.00027	-0.00019	-9.59E-05
-1.38E-17	9.92E-05	0.000199	0.000296	0.000387	0.00047	0.000541	0.000598	0.00064	0.000664
0.00067	0.000657	0.000626	0.000579	0.000516	0.000441	0.000357	0.000267	0.000174	8.42E-05
6.54E-18	-7.44E-05	-0.00014	-0.00018	-0.00021	-0.00022	-0.00021	-0.00018	-0.00013	-6.95E-05
2.56E-18	7.43E-05	0.000148	0.000214	0.000268	0.000301	0.00031	0.000287	0.00023	0.000135
-6.83E-18	-0.00017	-0.00039	-0.00063	-0.0009	-0.00119	-0.00149	-0.00178	-0.00207	-0.00232
-0.00253	-0.00268	-0.00276	-0.00276	-0.00267	-0.00248	-0.00219	-0.00179	-0.00129	-0.00069
-4.29E-17	0.000762	0.001581	0.002436	0.003303	0.004156	0.004967	0.005707	0.006349	0.006863
0.007226	0.007413	0.007407	0.007194	0.006765	0.006118	0.005258	0.004194	0.002946	0.001538
1.87E-18	-0.00163	-0.00331	-0.00501	-0.00666	-0.00822	-0.00964	-0.01089	-0.0119	-0.01265
-0.0131	-0.01322	-0.013	-0.01244	-0.01152	-0.01026	-0.00869	-0.00684	-0.00473	-0.00244
-9.52E-17	0.002517	0.005047	0.007522	0.009876	0.012041	0.013954	0.015557	0.016801	0.017644
0.018054	0.018013	0.017511	0.016554	0.015158	0.013355	0.011184	0.008698	0.005958	0.003034
-3.78E-17	-0.00307	-0.00608	-0.00897	-0.01165	-0.01405	-0.01612	-0.01778	-0.019	-0.01975
0.97999	-0.01975	-0.019	-0.01778	-0.01612	-0.01405	-0.01165	-0.00897	-0.00608	-0.00307
-3.78E-17	0.003034	0.005958	0.008698	0.011184	0.013355	0.015158	0.016554	0.017511	0.018013
0.018054	0.017644	0.016801	0.015557	0.013954	0.012041	0.009876	0.007522	0.005047	0.002517
-9.52E-17	-0.00244	-0.00473	-0.00684	-0.00869	-0.01026	-0.01152	-0.01244	-0.013	-0.01322
-0.0131	-0.01265	-0.0119	-0.01089	-0.00964	-0.00822	-0.00666	-0.00501	-0.00331	-0.00163
1.87E-18	0.001538	0.002946	0.004194	0.005258	0.006118	0.006765	0.007194	0.007407	0.007413
0.007226	0.006863	0.006349	0.005707	0.004967	0.004156	0.003303	0.002436	0.001581	0.000762
-4.29E-17	-0.00069	-0.00129	-0.00179	-0.00219	-0.00248	-0.00267	-0.00276	-0.00276	-0.00268
-0.00253	-0.00232	-0.00207	-0.00178	-0.00149	-0.00119	-0.0009	-0.00063	-0.00039	-0.00017
-6.83E-18	0.000135	0.00023	0.000287	0.00031	0.000301	0.000268	0.000214	0.000148	7.43E-05
2.56E-18	-6.95E-05	-0.00013	-0.00018	-0.00021	-0.00022	-0.00021	-0.00018	-0.00014	-7.44E-05
6.54E-18	8.42E-05	0.000174	0.000267	0.000357	0.000441	0.000516	0.000579	0.000626	0.000657
0.00067	0.000664	0.00064	0.000598	0.000541	0.00047	0.000387	0.000296	0.000199	9.92E-05
-1.38E-17	-9.59E-05	-0.00019	-0.00027	-0.00034	-0.0004	-0.00045	-0.00048	-0.0005	-0.0005
-0.0005	-0.00048	-0.00045	-0.00041	-0.00036	-0.00031	-0.00025	-0.00019	-0.00012	-6.06E-05
-5.15E-18	5.75E-05	0.000111	0.000159	0.000201	0.000236	0.000264	0.000285	0.000298	0.000304
0.000303									

Tabla 8: Vector del filtro rechaza-backlash.

12.12. Código del anfitrión del HIL

Este código se programó en *MATLAB*. Se ejecuta en la PC y se encarga de simular la planta y enviar los resultados por serial al cliente, el cliente regresa la señal de control y el proceso se repite.

```
%SE ESPECIFICAN CONDICIONES INICIALES %
x1=0;          %angulo      phi
x2=0;          %velocidad   phidot
x3=0.15;          %angulo      theta
x4=0;          %velocidad   thetadot
tau_x1=0;     %Torque inicial en eje phi
tau_x3=0;     %Torque inicial en eje theta

%SE ESPECIFICA t0 y dt %
t=0;           %Tiempo inicial de simulacion
dt=0.001;     %Step-size

%SE INICIALIZA PUERTO SERIAL %
s = serial('COM8');           %Puerto
set(s,'BaudRate',921600);     %Tasa de baudios
set(s, 'TimeOut', 3)          %Time-out de conexion
fopen(s);                     %Iniciar comunicacion

%SE RESUELVEN LAS EDs %
iters = 15000;   %Se simularan 15 segundos
for k=1:iters    %Bucle que resuelve numericamente las EDs

    %SE RESUELVEN LA 1RA ED %
    k11 = dt*f1(tau_x1,tau_x3,x1,x2,x3,x4,t);
    k21 = dt*f2(tau_x1,tau_x3,x1,x2,x3,x4,t);
    k31 = dt*f3(tau_x1,tau_x3,x1,x2,x3,x4,t);
    k41 = dt*f4(tau_x1,tau_x3,x1,x2,x3,x4,t);

    %%SE RESUELVEN LA 2DA ED %
    k12 = dt*f1(tau_x1,tau_x3,x1+0.5*k11,x2+0.5*k21,x3+0.5*k31,x4+0.5*k41,t+0.5*dt);
    k22 = dt*f2(tau_x1,tau_x3,x1+0.5*k11,x2+0.5*k21,x3+0.5*k31,x4+0.5*k41,t+0.5*dt);
    k32 = dt*f3(tau_x1,tau_x3,x1+0.5*k11,x2+0.5*k21,x3+0.5*k31,x4+0.5*k41,t+0.5*dt);
    k42 = dt*f4(tau_x1,tau_x3,x1+0.5*k11,x2+0.5*k21,x3+0.5*k31,x4+0.5*k41,t+0.5*dt);

    %SE RESUELVEN LA 3RA ED %
    k13 = dt*f1(tau_x1,tau_x3,x1+0.5*k12,x2+0.5*k22,x3+0.5*k32,x4+0.5*k42,t+0.5*dt);
    k23 = dt*f2(tau_x1,tau_x3,x1+0.5*k12,x2+0.5*k22,x3+0.5*k32,x4+0.5*k42,t+0.5*dt);
    k33 = dt*f3(tau_x1,tau_x3,x1+0.5*k12,x2+0.5*k22,x3+0.5*k32,x4+0.5*k42,t+0.5*dt);
    k43 = dt*f4(tau_x1,tau_x3,x1+0.5*k12,x2+0.5*k22,x3+0.5*k32,x4+0.5*k42,t+0.5*dt);

    %SE RESUELVEN LA 4TA ED %
    k14 = dt*f1(tau_x1,tau_x3,x1+0.5*k13,x2+0.5*k23,x3+0.5*k33,x4+0.5*k43,t+0.5*dt);
    k24 = dt*f2(tau_x1,tau_x3,x1+0.5*k13,x2+0.5*k23,x3+0.5*k33,x4+0.5*k43,t+0.5*dt);
    k34 = dt*f3(tau_x1,tau_x3,x1+0.5*k13,x2+0.5*k23,x3+0.5*k33,x4+0.5*k43,t+0.5*dt);
    k44 = dt*f4(tau_x1,tau_x3,x1+0.5*k13,x2+0.5*k23,x3+0.5*k33,x4+0.5*k43,t+0.5*dt);

    %SE PROMEDIAN LAS 4 PENDIENTES PARA CADA ED %
    x1 = x1 + (k11+2*k12+2*k13+k14)/6;   %phi
    x2 = x2 + (k21+2*k22+2*k23+k24)/6;   %phi_dot
    x3 = x3 + (k31+2*k32+2*k33+k34)/6;   %theta
    x4 = x4 + (k41+2*k42+2*k43+k44)/6;   %theta_dot

    %SE MANDAN POR SERIAL LOS 4 ESTADOS %
    str = sprintf('%2.4f %2.4f %2.4f %2.4f\n',x1,x2,x3,x4);
    fprintf(s,'%s',str);

    %SE RECIBE LA SENAL DE CONTROL %
        tau_x1 = fscanf(s,'%f');

    %SE INCREMENTA t Y SE REPITE EL PROCESO %
    t = t + dt;
end


%Las siguientes 4 funciones corresponden a las...
%4 ecuaciones diferenciales que describen el comportamiento...
%del pendulo de Furuta. Estas se resuelven con el metodo RK4.
function dx1dt = f1(tau_x1,tau_x3,x1,x2,x3,x4,t)
dx1dt = x2;
end

function dx2dt = f2(tau_x1,tau_x3,x1,x2,x3,x4,t)
cMax=1.24317e-6;
Fr_x1=0.0028873;
Fr_x3=0.000252566;
g=9.81;
JMax=0.00742;
KmMax=0.168;
```

```
la=0.105;
LaMax=0.00848;
lp=0.135;
M=0;
ma=0.17;
mp=0.03;
phi0=0;
phidot0=0;
RMax=104;
theta0=0;
thetadot0=0;

alpha=( JMax + (M + 2/3*ma + mp)*(la)^2);
beta=((M + 1/3*mp)*(lp)^2);
gamma=((M + 1/2*mp)*la*lp);
delta=((M + 1/2*mp)*g*lp);

dx2dt = ((tau_x1 - Fr_x1*x2) +  gamma*x4^2*sin(x3) -...
    (gamma*cos(x3)*(beta*cos(x3)*sin(x3)*x2^2 +...
    tau_x3 - Fr_x3*x4 + delta*sin(x3)))/beta -...
    2*beta*x2*x4*cos(x3)*sin(x3))/...
    (alpha + beta*sin(x3)^2 - (gamma^2*cos(x3)^2)/beta);
end

function dx3dt = f3(tau_x1,tau_x3,x1,x2,x3,x4,t)
dx3dt = x4;
end

function dx4dt = f4(tau_x1,tau_x3,x1,x2,x3,x4,t)
cMax=1.24317e-6;
Fr_x1=0.0028873;
Fr_x3=0.000252566;
g=9.81;
JMax=0.00742;
KmMax=0.168;
la=0.105;
LaMax=0.00848;
lp=0.135;
M=0;
ma=0.17;
mp=0.03;
phi0=0;
phidot0=0;
RMax=104;
theta0=0;
thetadot0=0;

alpha=( JMax + (M + 2/3*ma + mp)*(la)^2);
beta=((M + 1/3*mp)*(lp)^2);
gamma=((M + 1/2*mp)*la*lp);
delta=((M + 1/2*mp)*g*lp);

dx4dt = ((alpha + beta*sin(x3)^2)*(tau_x3 -...
    Fr_x3*x4 + delta*sin(x3) + beta*x2^2*...
    cos(x3)*sin(x3) - (gamma*cos(x3)*(gamma*sin(x3)*...
    x4^2 - 2*beta*x2*cos(x3)*sin(x3)*x4 +...
    (tau_x1 - Fr_x1*x2)))/(alpha + beta*sin(x3)^2)))/(alpha*beta +...
    beta^2*sin(x3)^2 + gamma^2*sin(x3)^2 - gamma^2);
end
```

12.13. Código del cliente del HIL

```
1  //Esta funcion se llama cuando se recibe info:
2  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
3  {
4          //Se recibe 1 byte por el UART (puerto VCOM)
5          HAL_UART_Receive_IT(&huart3, UART3_rxBuffer, 1);
6          //Se concatenan bytes para formar string
7          strcat(recbuff, UART3_rxBuffer);
8
9          //Si se detecta salto de linea -> haz lo siguiente:
10         if (strstr(recbuff, "\n") != NULL)
11         {
12                 //char numbers[] = "1.1 2.2" ==> float x1, x2   |       x1=1.1, x2=2.2
13                 double x1, x2, x3, x4;
14                 x1 = strtod(recbuff, &rxBuffPrtEnd);             //Convierte string a double
15                 x2 = strtod(rxBuffPrtEnd, &rxBuffPrtEnd);        //Convierte string a double
16                 x3 = strtod(rxBuffPrtEnd, &rxBuffPrtEnd);        //Convierte string a double
17                 x4 = strtod(rxBuffPrtEnd, NULL);                         //Convierte string a double
18
19                 //Vector de retroalimentacion, este equilibra al pendulo
20                 double vect_retro = -1*x1-1.2*x2-25*x3-2.5*x4;
21                 //Convierte a string ASCII el numero vect_retro
22                 sprintf(texto, "%1.8f\r\n", (double) vect_retro);
23                 //Manda el string decimal vect_retro a la PC
24                 HAL_UART_Transmit(&huart3, texto, strlen(texto), 10);
25                 //Limpia el recbuff, todos los valores = {0}
26                 memset(&recbuff[0], 0, sizeof(recbuff));
27         }
28
29         //Cleans the interrupt flag
30         USART3->ICR = USART3->ICR | 0b00001000;
31         //Limpia el buffer UART, todos los valores = {0}
32         memset(&UART3_rxBuffer[0], 0, sizeof(UART3_rxBuffer));
33 }
```

12.14. Código del controlador con estimación Kalman

Este código se programó en *C++* y se ejecuta en el μC *STM32G474RE*. El μC lee las señales de cuadratura de los codificadores rotatorios, las decodifica, estima las velocidades utilizando el filtro Kalman, calcula la señal de control, la convierte a una señal PWM y la envía a la salida para cerrar el bucle.

```
#include "mbed.h"
#include <conf_alain.h> //Configure GPIOs and TIMERs...
#include <pendulo.h>    //Add Rotary pendulum parameters
#include <alainclass.h> //Add alainclass class

AnalogOut aout(PA_5);
DigitalOut doutPA1(PA_1);      // Hearbeat signal to oscilloscope
DigitalIn boton(USER_BUTTON); // Board Blue Button
DigitalOut mot_in1(PC_2);      // to Motor input 1
DigitalOut mot_in2(PC_3);      // to Motor input 2
DigitalOut exp_en(PB_0);       // to HIL experiment enable OUTPUT
DigitalIn exp_in(PB_5);        // experiment enable INPUT from HIL
PwmOut pwm(PA_0);              // Initialise PWM output on pin PA0

Thread thread(osPriorityNormal, 32 * 1024); // Start Thread1 with a stack size of...
Thread thread2(osPriorityNormal, 8 * 1024); // Start thread2...
Thread thread3(osPriorityNormal, 8 * 1024); // Start thread3...
Ticker ctrlISR;                             // Generate software interrupt every x secs...
Timer t;                                    // Start a timer

void print_thread();        // Declare print thread fnc
void ctrl_loop();           // Declare fnc to be called by ctrlISR Ticker
void rx_interrupt();        // Declare fnc that handles RxSerial interrupts
void kalman(float torquin); // Declare fnc that handles RxSerial interrupts

int main()
{
    HAL_Delay(300); //Gives time to settle down
    gpioconf(); // Alain's custom GPIO conf for TIM inputs
    timconf();  // Alain's TIMER conf as ENCODER

    serial_port.baud(2000000);                  // Baud rate = 20Mbaud/s ~= 2Mbps
    serial_port.format(8, SerialBase::None, 1); // 8bits, no parity, 1stop bit.
    printf("Serial port initialized\n");
    ThisThread::sleep_for(200ms);

    serial_port.attach(&rx_interrupt, SerialBase::RxIrq); // Attatch interrupt to rx_interrupt function
    printf("ISR attatched\n");

    t.start(); // Start timer

    pwm.period_us(50); // PWM period = 50 usec | 20Khz

    thread.start(print_thread); // Start print thread

    std::chrono::microseconds Fs_us(1000000 / (int)round(Fs)); // Create ns datatype
    ctrlISR.attach(&ctrl_loop, Fs_us);                         // Start interrupt every Fs_us ns

    while (true) // Do nothing but dont sleep so clocks dont stop
    {
        ThisThread::sleep_for(1000ms); //Clocks may stop using this function if deep_sleep not disabled
    }
}

void ctrl_loop()
{
    readflag = 0; // Block access from other threads

    dpMOT = ((TIM3->CNT) - posMOT); // dPos mot
    dpENC = ((TIM4->CNT) - posENC); // dPos enc
    posMOT = TIM3->CNT;             // Read MOT pos
    posENC = TIM4->CNT;             // Read ENC pos

    posMOTrads = 1 + (posMOT - 32768) / K_mot; // Converts to radians and x_0 = [1 0 0 0]
    posENCrads = (posENC - 32768) / K_enc;     //

    velMOT = dpMOT * Fs / K_mot; //Converts to radians
    velENC = dpENC * Fs / K_enc;

    u = -(posMOTrads * -305.8605e-003 // LQR GAIN
        + KF_x[1] * -365.3179e-003
        + posENCrads * -6.5878e+000
```

```
73          + KF_x[3] * -613.5863e-003);
74
75      float maxNm = 0.001;            // Max torque normalized to 100% PWM duty Cycle
76      u = saturate(maxNm, u);         // Saturate torque at +-maxNm
77      vpwm = fabs(u / maxNm);         // Pwm requires positive integers
78      direccion(u, mot_in1, mot_in2); // Sign of u defines direction
79
80      if (boton == 1 || exp_in == 1) // Controller doesn't start until
81      {
82          float usat =    -(KF_x[0] * -305.8605e-003  //LQR GAIN with estimated x^hat
83                          + KF_x[1] * -365.3179e-003
84                          + KF_x[2] * -6.5878e+000
85                          + KF_x[3] * -613.5863e-003);
86          kalman(usat);   // call function to estimate states usign Kalman filter
87      }
88      else    {    u = 0.0;    }
89      if (boton == 1) // Only actuate motor/pwm if user presses blue button
90      {
91          inicio = 1;       // Start Flag
92          pwm.write(vpwm); // PWM dutycycle
93          exp_en = 1;       // Experiment_enable flag
94      }
95      else
96      {
97          pwm.write(0.0f);
98          exp_en = 0;
99          inicio = 0;
100     }
101     readflag = 1;        //Allow other threads to read
102     dtime++;             //Discrete time reference
103     doutPA1 = !doutPA1; //Heartbeat signal to oscilloscope
104 }
105
106 void print_thread()
107 {
108     printf("Hola soy el Sr.Pendulo\n");
109     ThisThread::sleep_for(10ms);
110     while (true)
111     {
112         aout = ((posENCrads * 124.1) + 1) / 3.3; // 0.1rad -> 1V | Analog Output to oscilloscope
113         printf("x1%09.6f\tx2%09.6f\tx3%09.6f\tx4%09.6f\tuu%06.6f\t
114                 x1%09.6f\tx2%09.6f\tx3%09.6f\tx4%09.6f\txE%09.6f\r\n"
115                 ,posMOTrads, velMOT, posENCrads, velENC, u,
116                  KF_x[0], KF_x[1], KF_x[2], KF_x[3], (float)exp_en);
117         ThisThread::sleep_for(10ms); // Print every 10ms | 100Hz
118     }
119 }
120
121 void rx_interrupt()
122 {
123     USART2->CR1 |= 0b100000; // Clear RXNE event flag by writing 1 to RXNEIE
124     USART2->ICR |= 0b1000;
125     // Toggle the LED.
126     // led = !led;
127     // Read the data to clear the receive interrupt.
128     // if (serial_port.read(&c, 1))
129     // {
130     //     // Echo the input back to the terminal.
131     //     serial_port.write(&c, 1);
132     // }
133     // ThisThread::sleep_for(1ms);
134 }
135
136 void kalman(float torquin)
137 {
138 KF_x[0] = posMOTrads; //Update measurements
139 KF_x[2] = posENCrads;
140 // Kalman estimate A*x + B*u
141 KF_x[0] = KFA[0] * KF_x[0] + KFA[1] * KF_x[1] + KFA[2] * KF_x[2] + KFA[3] * KF_x[3] +
142           KFB[0] * torquin +
143           KFB[1] * KF_x[0] + KFB[2] * KF_x[1] + KFB[3] * KF_x[2] + KFB[4] * KF_x[3];
144
145 KF_x[1] = KFA[4] * KF_x[0] + KFA[5] * KF_x[1] + KFA[6] * KF_x[2] + KFA[7] * KF_x[3] +
146           KFB[5] * torquin +
147           KFB[6] * KF_x[0] + KFB[7] * KF_x[1] + KFB[8] * KF_x[2] + KFB[9] * KF_x[3];
148
149 KF_x[2] = KFA[8] * KF_x[0] + KFA[9] * KF_x[1] + KFA[10] * KF_x[2] + KFA[11] * KF_x[3] +
150           KFB[10] * torquin +
151           KFB[11] * KF_x[0] + KFB[12] * KF_x[1] + KFB[13] * KF_x[2] + KFB[14] * KF_x[3];
152
153 KF_x[3] = KFA[12] * KF_x[0] + KFA[13] * KF_x[1] + KFA[14] * KF_x[2] + KFA[15] * KF_x[3] +
154           KFB[15] * torquin +
155           KFB[16] * KF_x[0] + KFB[17] * KF_x[1] + KFB[18] * KF_x[2] + KFB[19] * KF_x[3];
156 }
```

12.15. Código del HIL con simulacion de encoder

Este código se programó en *C++* y se ejecuta en el μC *STM32H755ZI*. El μC lee la señal PWM enviada por el controlador (ver anexo anterior), la convierte a torque, simula como este torque actúa en la planta simulada y mapea la posición de radianes continuos a 4 señales de cuadratura simulando al 100 % el comportamiento de los codificadores rotatorios reales (comprobado por osciloscopio).

```
/* USER CODE BEGIN */
float e = 2.71828182845904553488848081484903;
float t = 0.0, f = 0.0;
void usdelay(uint32_t us); //Se declaran funciones
void usdelay_until(uint32_t us);
int encoderx1A(float pos);
int encoderx1B(float pos);
int encoderx3A(float pos);
int encoderx3B(float pos);

float KF_x[4] = {0, 0, 0, 0}; //Conficiones iniciales
float x0[4] = {1, 0, 0, 0};
float x[4] = {0, 0, 0, 0};

float Am[] = //[A_d] discretizado a 100us/paso
{1.0   , 0.0001, -9.5863e-8, -3.1954e-12,
0      , 1.0   , -0.0019173, -9.5863e-8,
0      , 0     , 1.0       , 0.0001,
0      , 0     , 0.018165  , 1.0};

//[B_d] discretizado a 100us/paso
float Bm[] = {5.6733e-6, 0.11347, -8.6925e-6, -0.17385};

// MATRICES KALMAN :
float KFA[] = // Matriz Kalman A @100us
{0.99989, 0.000099994, 0.00026418, 1.3213e-8,
-0.00040717, 1.0, 0.0018712, 9.3601e-8,
0.00026426, 1.3216e-8, 0.99733, 0.000099867,
0.0035623, 1.7816e-7, -0.017787, 1.0};

float KFB[] = // Matriz Kalman B @100us
{5.6723e-6, 0.00011203, 6.5439e-23, -0.00026427, -4.6232e-22,
0.11347, 0.00040717, 6.2994e-23, -0.0037884, -2.3505e-21,
-8.6843e-6, -0.00026426, 2.0195e-22, 0.0026677, 1.6892e-21,
-0.17385, -0.0035623, 7.7975e-21, 0.035952, 2.2772e-20};

int ite = 0; //Banderas, contadores, etc...
int dac = 0;
float Duty = 0.0;
char Test[96] = {"0"};
float u = 0.0;
int pwmsign = 1;
int pwmstat = 0;
int longitud = 0;
float exp_en = 0.0;

while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    TIM2->CNT = 0;//Se inicia contador de tiempo en 0us
    int CC1IFx = TIM4->SR & (1 << 1);
    if (CC1IFx != 0) // Si CC1IF Flag (1th bit) != 0 se lee dutycycle PWM
    {

        float ICvalue1 = HAL_TIM_ReadCapturedValue(&htim4, TIM_CHANNEL_1);
        float ICvalue2 = HAL_TIM_ReadCapturedValue(&htim4, TIM_CHANNEL_2);
        if (ICvalue1 != 0)
        {
            if ((HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_6)) == 1)
            {
                pwmsign = 1;
            }
            else
            {
                pwmsign = -1;
            }
            Duty = (ICvalue2 * 100.0 / ICvalue1) * pwmsign;
        }
    }
    else
```

```
73      {
74          if ((HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_6)) == 1)
75          {
76              pwmsign = 1;
77          }
78          else
79          {
80              pwmsign = -1;
81          }
82          pwmstat = HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_12);
83          Duty = 100.0 * pwmstat * pwmsign;
84      }
85
86      HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_0); // Heartbeat LED
87      HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_8); // Heartbeat Oscilloscope
88
89      if ((HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_5)) == 0)
90      {//Si controlador desactivado
91          exp_en = 0.0;
92          u = 0.0;
93          Duty = 0.0;
94      }
95      else
96      {//Si se detecta senal de control proviniente del mcu #1
97          u = 0.01 * Duty * 0.001; //Lee senal de control PWM
98          exp_en = 1.0; //Experimento habilitado
99      }
100
101     if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == 1)
102     { //Si se presiona boton se cierra bucle con estimacion perfecta
103       //normalmente no se utiliza, solo si el controlador falla.
104         HAL_GPIO_WritePin(GPIOD, GPIO_PIN_0, 1);
105         u =
106         -(
107         x[0] * -305.8605e-003 +
108         x[1] * -365.3179e-003 +
109         x[2] * -6.5878e+000 +
110         x[3] * -613.5863e-003
111         );
112
113         if (u > 0.001) //Saturacion
114         {
115             u = 0.01;
116         }
117         if (u < -0.001)
118         {
119             u = -0.01;
120         }
121     }
122     else
123     { //Avisa a controlador que pause
124         HAL_GPIO_WritePin(GPIOD, GPIO_PIN_0, 0);
125     }
126     if (ite == 0)
127     {
128         x[0] = x0[0];         //Condiciones iniciales de ss
129         x[1] = x0[1];         //ss = state-space
130         x[2] = x0[2];
131         x[3] = x0[3];
132
133         KF_x[0] = x0[0];      //Condiciones inicial de filtro Kalman
134         KF_x[1] = x0[1];
135         KF_x[2] = x0[2];
136         KF_x[3] = x0[3];
137     }
138     else
139     { //Simula planta en ss
140         x[0] = Am[0] * x[0] + Am[1] * x[1] +
141         Am[2] * x[2] + Am[3] * x[3] +
142         Bm[0] * u;
143
144         x[1] = Am[4] * x[0] + Am[5] * x[1] +
145         Am[6] * x[2] + Am[7] * x[3] +
146         Bm[1] * u;
147
148         x[2] = Am[8] * x[0] + Am[9] * x[1] +
149         Am[10] * x[2] + Am[11] * x[3] +
150         Bm[2] * u;
151
152         x[3] = Am[12] * x[0] + Am[13] * x[1] +
153         Am[14] * x[2] + Am[15] * x[3] +
154         Bm[3] * u;
155
156         if (HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == 1)
157         {
158             KF_x[0] = x[0];
159             KF_x[2] = x[2];
160
```

```
161             KF_x[0] = KFA[0] * KF_x[0] + KFA[1] * KF_x[1] +
162             KFA[2] * KF_x[2] + KFA[3] * KF_x[3] +
163                       KFB[0] * u + KFB[1] * KF_x[0] +
164                       KFB[2] * KF_x[1] + KFB[3] * KF_x[2] +
165                       KFB[4] * KF_x[3];
166
167             KF_x[1] = KFA[4] * KF_x[0] + KFA[5] * KF_x[1] +
168             KFA[6] * KF_x[2] + KFA[7] * KF_x[3] +
169                       KFB[5] * u + KFB[6] * KF_x[0] +
170                       KFB[7] * KF_x[1] + KFB[8] * KF_x[2] +
171                       KFB[9] * KF_x[3];
172
173             KF_x[2] = KFA[8] * KF_x[0] + KFA[9] * KF_x[1] +
174             KFA[10] * KF_x[2] + KFA[11] * KF_x[3] +
175                       KFB[10] * u + KFB[11] * KF_x[0] +
176                       KFB[12] * KF_x[1] + KFB[13] * KF_x[2] +
177                       KFB[14] * KF_x[3];
178
179             KF_x[3] = KFA[12] * KF_x[0] + KFA[13] * KF_x[1] +
180             KFA[14] * KF_x[2] + KFA[15] * KF_x[3] +
181                       KFB[15] * u + KFB[16] * KF_x[0] +
182                       KFB[17] * KF_x[1] + KFB[18] * KF_x[2] +
183                       KFB[19] * KF_x[3];
184         }
185     }
186     if (ite %100 == 0)
187     {
188
189         sprintf(Test,
190                 "x1%09.6f\tx2%09.6f\tx3%09.6f\tx4%09.6f\t
191                  uu%06.6f\txE%09.6f\r\n"
192                 , x[0], x[1], x[2], x[3], u, exp_en);
193         HAL_UART_Transmit(&huart3, Test, strlen(Test), 10);
194
195     }
196     //x1 Encoder channel A & B
197     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_9, (GPIO_PinState)encoderx1A(x[0]));
198     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_10, (GPIO_PinState)encoderx1B(x[0]));
199
200     //x3 Encoder channel A & B
201     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_11, (GPIO_PinState)encoderx3A(x[2]));
202     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_12, (GPIO_PinState)encoderx3B(x[2]));
203
204     //Tambien manda x3 por salida analogica a osciloscopio
205     dac = (int)((x[2] * 124.1) * (4096 / 4) + 1024); // 0.1rad -> 1V
206     HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_1, DAC_ALIGN_12B_R, dac);
207
208     t += 100e-6; //Incrementa tiempo en 100us
209
210     ite++; //Incrementa iteracion
211     usdelay_until(100); //Espera hasta que sean 100us
212
213
214 }
215 /* USER CODE END 3 */
216
217 /* USER CODE BEGIN 4 */
218 //Esta funcion crea un retardo
219 //en microsegundos
220 void usdelay(uint32_t us) {
221         TIM2->CNT = 0;
222         while (TIM2->CNT < us) {
223                 asm("NOP");
224         }
225 }
226
227 //Esta funcion crea un retardo
228 //en 2 partes en microsegundos
229 //ideal para alta precision
230 void usdelay_until(uint32_t us) {
231         while (TIM2->CNT < us) {
232                 asm("NOP");
233         }
234 }
235
236 //Las funciones encoder simulan el encoder de n ppr
237 //calculando el signo(sen(pos)) o signo(cos(pos))
238 int encoderx1A(float pos) {
239         if (sin(12000.0 * pos) < 0)
240                         {
241                 return 0;
242         } else {
243                 return 1;
244         }
245 }
246
247 int encoderx1B(float pos) {
248                 if (cos(12000.0 * pos) < 0)
```

```
249                         {
250                 return 0;
251         } else {
252                 return 1;
253         }
254 }
255
256 int encoderx3A(float pos) {
257                 if (sin(96000.0 * pos) < 0)
258                         {
259                 return 0;
260         } else {
261                 return 1;
262         }
263 }
264
265 int encoderx3B(float pos) {
266                 if (cos(96000.0 * pos) < 0)
267                         {
268                 return 0;
269         } else {
270                 return 1;
271         }
272 }
```

Referencias

[1] A. V. Oppenheim, *Discrete-time signal processing*. Pearson Education India, 1999.

[2] P. Horowitz, W. Hill, and I. Robinson, *The art of electronics 3rd edition*. Cambridge university press Cambridge, 2015.

[3] L. Lindner, *Theoretical method to increase the speed of continuous mapping in a three-dimensional laser scanning system using servomotors control*. Editorial UABC, 2021.

[4] P. E. Sandin, *Robot Mechanism and Mechanical Devices*. McGraw Hill Professional, 2003.

[5] M. Gäfvert, "Modelling the furuta pendulum," 1998.

[6] B. Allotta, L. Pugi, and F. Bartolini, "Reinforcement neural network for the stabilization of a furuta pendulum," in *Proceedings of EUCOMES 08*, pp. 287–294, Springer, 2009.

[7] A. Wadi, J.-H. Lee, and L. Romdhane, "Nonlinear sliding mode control of the furuta pendulum," in *2018 11th International Symposium on Mechatronics and its Applications (ISMA)*, pp. 1–5, 2018.

[8] J. Lupton, B. Ortolano, J. Keane, and M. Reynolds, "Stabilizing an inverted pendulum with a virtual video sensor," *International Journal of Applied Engineering Research*, vol. 16, no. 6, pp. 497–501, 2021.

[9] P. Sarhadi and S. Yousefpour, "State of the art: hardware in the loop modeling and simulation with its applications in design, development and implementation of system and control software," *International Journal of Dynamics and Control*, vol. 3, no. 4, pp. 470–479, 2015.

[10] P. Kaps and P. Rentrop, "Generalized runge-kutta methods of order four with stepsize control for stiff ordinary differential equations," *Numerische Mathematik*, vol. 33, no. 1, pp. 55–68, 1979.

[11] H. K. Fathy, Z. S. Filipi, J. Hagena, and J. L. Stein, "Review of hardware-in-the-loop simulation and its prospects in the automotive area," in *Modeling and simulation for military applications*, vol. 6228, pp. 117–136, SPIE, 2006.

[12] R. Isermann, J. Schaffnit, and S. Sinsel, "Hardware-in-the-loop simulation for the design and testing of engine-control systems," *Control Engineering Practice*, vol. 7, no. 5, pp. 643–653, 1999.

[13] X. Wu, S. Lentijo, A. Deshmuk, A. Monti, and F. Ponci, "Design and implementation of a power-hardware-in-the-loop interface: a nonlinear load case study," in *Twentieth Annual IEEE Applied Power Electronics Conference and Exposition, 2005. APEC 2005.*, vol. 2, pp. 1332–1338 Vol. 2, 2005.

[14] B. Fischer, C. Mehler, A. Zuga, and M. Shan, "Control design for mechanical hardware-in-the-loop operation of dynamometers for testing full-scale drive trains," *Wind Energy*, vol. 19, no. 10, pp. 1889–1901, 2016.

[15] "Simulink documentation." https://www.mathworks.com/help/simulink/. Published by: MathWorks.

[16] B. P. Enrique René, "Apuntes de la unidad de aprendizaje procesamiento digital de señales," 2017.

[17] M. S. Fadali and A. Visioli, *Digital control engineering: analysis and design*. Academic Press, 2013.

[18] K. Ogata, *Sistemas de control en tiempo discreto*. Pearson educación, 1996.

[19] P. F. Khan, S. Sengottuvel, R. Patel, K. Gireesan, R. Baskaran, and A. Mani, "Design and implementation of a discrete-time proportional integral (pi) controller for the temperature control of a heating pad," *SLAS TECHNOLOGY: Translating Life Sciences Innovation*, vol. 23, no. 6, pp. 614–623, 2018.

[20] G. Tarchała and T. Orłowska-Kowalska, "Discrete sliding mode speed control of induction motor using time-varying switching line," *Electronics*, vol. 9, no. 1, p. 185, 2020.

[21] R. T. Siewe, U. S. Domguia, and P. Woafo, "Microcontroller control/synchronization of the dynamics of van der pol oscillators submitted to disturbances," *International Journal of Nonlinear Sciences and Numerical Simulation*, vol. 19, no. 2, pp. 153–163, 2018.

[22] M. S. Fadali and A. Visioli, *Digital control engineering: analysis and design*. Academic Press, 2013.

[23] L. Dai and R. Harjani, "Cmos switched-op-amp-based sample-and-hold circuit," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 1, pp. 109–113, 2000.

[24] M. Barr, "Pulse width modulation," *Embedded Systems Programming*, vol. 14, no. 10, pp. 103–104, 2001.

[25] K. K. Masami Shirai, Masato Noguchi, "Incremental rotary encoder, and a surveying instrument incorporating a magnetic incremental rotary encoder," U.S. Patent US6622391B1, Sep. 2003.

[26] E. W. Cheney and D. R. Kincaid, *Numerical mathematics and computing*. Cengage Learning, 2012.

[27] "Stm32cube ide integrated development environment for stm32." https://www.st.com/en/development-tools/stm32cubeide.html#overview. Published by: ST.

[28] "Platformio." https://docs.platformio.org/en/latest/what-is-platformio.html. Published by: PlatformIO.

[29] "Visual studio code." https://code.visualstudio.com. Published by: Microsoft.

[30] STMicroelectronics, *STM32F103C8 Datasheet*, 2015. Rev. 17.

[31] STMicroelectronics, *STM32G474RE Datasheet*, 2021. Rev. 6.

[32] STMicroelectronics, *STM32H755ZI Datasheet*, 2019. Rev. 1.

[33] A. F. Ilmiawan, D. Wijanarko, A. H. Arofat, H. Hindersyah, and A. Purwadi, "An easy speed measurement for incremental rotary encoder using multi stage moving average method," in *2014 International Conference on Electrical Engineering and Computer Science (ICEECS)*, pp. 363–368, 2014.

[34] A. C. Negrea, M. Imecs, I. l. Incze, A. Pop, and C. Szabo, "Error compensation methods in speed identification using incremental encoder," in *2012 International Conference and Exposition on Electrical and Power Engineering*, pp. 441–445, 2012.

[35] R. F. Stengel, *Optimal control and estimation*. Courier Corporation, 1994.

[36] J. Willems, "Least squares stationary optimal control and the algebraic riccati equation," *IEEE Transactions on automatic control*, vol. 16, no. 6, pp. 621–634, 1971.

[37] M. Nordin, J. Galic', and P.-O. Gutman, "New models for backlash and gear play," *International journal of adaptive control and signal processing*, vol. 11, no. 1, pp. 49–63, 1997.

[38] "Matlab documentation." https://www.mathworks.com/help/matlab/. Published by: MathWorks.

[39] MathWorks, *Matlab Primer*. Mathworks, 2021.

[40] V. Utkin and H. Lee, "Chattering problem in sliding mode control systems," in *International Workshop on Variable Structure Systems, 2006. VSS'06.*, pp. 346–350, IEEE, 2006.

[41] L. Kovudhikulrungsri and T. Koseki, "Precise speed estimation from a low-resolution encoder by dual-sampling-rate observer," *IEEE/ASME Transactions on Mechatronics*, vol. 11, no. 6, pp. 661–670, 2006.

[42] Pololu, *Pololu 37D 12V 70:1 Metal Gearmotors Datasheet*, 2019. Rev. 1.1.

[43] Maxon, *Maxon Encoder MR 225780 Datasheet*, 2020.

[44] Maxon, *Maxon Re-max 29 226783 Datasheet*, 2016.

[45] Omron, *Omron E6B2-CWZ6C Datasheet*, 2017.

[46] J. J. Moré and D. C. Sorensen, "Computing a trust region step," *SIAM Journal on scientific and statistical computing*, vol. 4, no. 3, pp. 553–572, 1983.

[47] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.