UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

"DISEÑO E IMPLEMENTACIÓN EN "HARDWARE-IN-THE-LOOP" DEL SISTEMA CARRO-PÉNDULO CON BASE EN UN PROCESADOR DIGITAL DE SEÑALES"

TESIS

PARA OBTENER EL TÍTULO DE

MAESTRO EN ROBÓTICA

PRESENTA:

MARCELINO MARTÍNEZ ARAGÓN

DIRECTOR:

DR. FERMÍN HUGO RAMÍREZ LEYVA

CO-DIRECTOR:

DR. JOSÉ ANIBAL ARIAS AGUILAR

HUAJUAPAN DE LEÓN, OAXACA, MAYO DE 2019

Tesis presentada ante los sinodales:
Dra. Esther Lugo González
Dr. Jorge Luis Barahona Ávalos
Dr. Edgardo Yescas Mendoza
M.C. Felipe Santiago Espinosa

Director:
Dr. Fermín Hugo Ramírez Leyva
Co-director:
Dr. José Aníbal Arias Aguilar

A mis padres:
Marcelino y Rosa

A mis hermanos:
Noe Pedro, Eliazar y Elvira.
Sandra, Roselba y Noe.

A mi esposa:
Karen.

Índice

Índice de figuras

Índice de tablas

Introducción

La teoría de control es el estudio del comportamiento de sistemas dinámicos. Al sistema a controlar se le denomina planta, ésta puede tener cualquier número de salidas o variables a controlar. Las variables podrían ser, por ejemplo, presión, velocidad, posición, temperatura, flujo, etc. El trabajo en el laboratorio es indispensable en la educación de ingeniería de control, sin la experiencia proporcionada por los ejercicios de laboratorio, los estudiantes no comprenderían plenamente sus aplicaciones y limitaciones [1]. A pesar de sus virtudes, la formación de un laboratorio convencional tiene inconvenientes, tales como espacio, costo, disponibilidad, límite de plantas, etc.

Es posible implementar controladores reales sobre hardware en interacción con modelos físicos simulados en tiempo real, que permitan dar una idea muy aproximada al comportamiento de un sistema [2]. Este tipo de implementación es comúnmente llamada Hardware-In-The-Loop(HIL), un simulador HIL es una técnica de prueba que simula el comportamiento de entradas y salidas de un sistema físico que es conectada a un controlador en tiempo real [3], ésta sustituye a la planta real en un laboratorio.

El sistema carro-péndulo invertido es un ejemplo muy utilizado para la enseñanza de la teoría de control. Consiste en un péndulo colocado sobre una articulación fija de un carro, el péndulo oscila libremente y el carro, con ayuda de una guía, se mueve de forma horizontal. El objetivo del control es llevar al péndulo en una posición vertical superior.

En este trabajo se propone la implementación de un sistema carro-péndulo invertido usando la técnica de simulación HIL con la finalidad de tener una planta virtual de bajo costo, en la que se puedan aplicar diversas técnicas de control con la seguridad de que la salida del sistema sea lo más cercano a la realidad. Para incrementar el realismo se va a tener una interfaz gráfica en la que visualmente se mostrarán los movimientos del sistema y su configuración.

Estado del arte

Se hizo una investigación de los trabajos existentes en la literatura con caracteristicas comunes con la presente investigacion, los cuales se describen a continuacion.

D. López describe un método de análisis de tolerancia a fallos, donde el sistema es un carro-péndulo invertido modelado por software en MATLAB/Simulink [4]. El controlador es descrito en el lenguaje de descripción hardware VHDL y es emulado en un FPGA. Con esto se implementa un entorno HIL, en el que el hardware del FPGA y el software se unen creando un sistema híbrido, con gran potencial para el estudio de la tolerancia a fallos en circuitos electrónicos. En la Figura I.1, se representa el esquema HIL con el que se llevó acabo el trabajo.

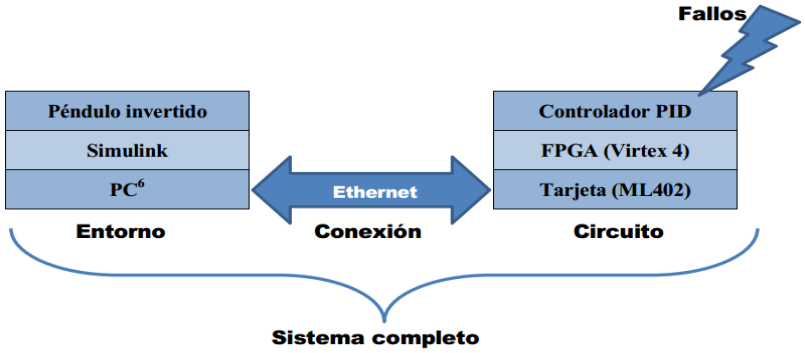

Figura I.1: Esquema HIL utilizado [4]

En la Figura I.2, se muestra el diagrama esquemático utilizado. Las entradas definen las referencias de posición del carro. El controlador se diseñó en un bloque de Simulink que permite añadir código VHDL. Se utiliza el modelo del carro-péndulo invertido que proporciona MATLAB, añadiéndole algunas modificaciones.

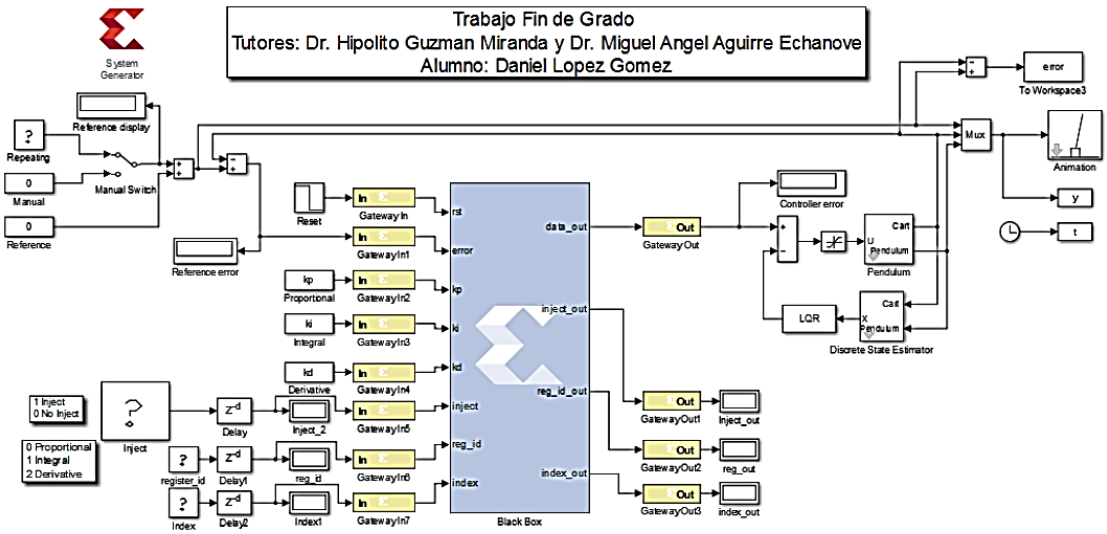

Figura I.2: Entorno de simulación HIL, para el estudio de tolerancia a fallos [4]

D. López desarrolló un simulador HIL donde el controlador es embebido en un FPGA y la planta del carro-péndulo invertido es simulada en Matlab/Simulink. En este trabajo la planta será embebida en un microcontrolador y el control se realiza en Matlab/Simulink, además el simulador HIL propuesto permite conectarse a cualquier hardware, de forma que el algoritmo de control pueda implementarse en una tarjeta Arduino, un FPGA, Raspberry, etc.

P. Feng *et al,* implementaron un control de movimiento de un péndulo invertido rotatorio con HIL [5]. El sistema de control está realizado en MATLAB/Simulink. Cuenta con el modelo matemático del péndulo invertido simple y doble. Los resultados muestran que este sistema es una plataforma ideal para la investigación y aplicar la teoría de control.

En el trabajo de P.Feng *et al,* usaron MATLAB/Simulink para el control y la simulación de la planta. Este software tiene módulos de aplicación en tiempo real y demás herramientas para el diseño de control, de igual forma en esta tesis se utiliza MATLAB/Simulink para diseñar un par de algoritmos de control. Aunque Feng implementa el control en un péndulo giratorio (o de Furuta), este trabajo se enfoca en el carro-péndulo invertido.

A. Turnau *et al.*, pusieron en práctica un control óptimo de tiempo para el sistema carro-péndulo en tiempo real [6]. La simulación HIL lo usaron para hallar experimentalmente un par de constantes de la ecuación de control. Dado que las pruebas se hicieron en la planta real, no se muestra el proceso de implementación del simulador HIL. Queda como antecedente la utilidad de ésta técnica para realizar diversos tipos de control en el carro-péndulo.

El control SwingUp publicado en el trabajo de Turnau se usa de guía (junto a [7]) para el control de balanceo del carro-péndulo presentado en este documento, debido a que es de los pocos trabajos donde implementan el algoritmo de control directamente en la planta real.

El carro-péndulo invertido, es llamado un sistema subactuado puesto que los sistemas o vehículos mecánicos subactuados, son sistemas con un número menor de actuadores que grados de libertad a ser controlados[8]. En los siguientes trabajos se muestran algunas aplicaciones del uso de la técnica HIL en sistemas de este tipo.

Subramanian *et al.*, trabajaron en la verificación en tiempo real de un algoritmo para evitar obstáculos en 3D de un *flat-fish* (pez marino plano) subactuado tipo AUV (vehículo submarino autónomo), usando la herramienta de simulación HIL [9]. Los modelos son desarrollados en el entorno MATLAB/Simulink y la simulación HIL se

lleva a cabo utilizando el entorno dSPACE. Los resultados muestran que la simulación HIL es una herramienta para la verificación de algoritmos de control y el algoritmo de evitación de obstáculos desarrollado puede ser usado en tiempo real para el *flat-fish* tipo AUV.

En la Figura I.3, se muestra la estructura detallada de la simulación HIL usada para la verificación del algoritmo de evitación de obstáculos. Utiliza la tarjeta de control dSPACE DS1104, el software "control desk" es usado para probar y experimentar en tiempo real el modelo HIL. Se colocó un sensor para detectar y medir la distancia entre los obstáculos y el AUV. El modelo de simulación recibe las posiciones inicial y final como entrada de usuario y recibe las posiciones de los obstáculos del sensor de datos, el planificador genera la trayectoria deseada, se compara la trayectoria deseada con la real, obteniendo un error de seguimiento, entonces el controlador ajusta los valores de velocidad y ángulos, estas señales se trasmiten como voltajes analógicos a la unidad actuadora, que consta de tres motores para controlar los ejes del AUV.

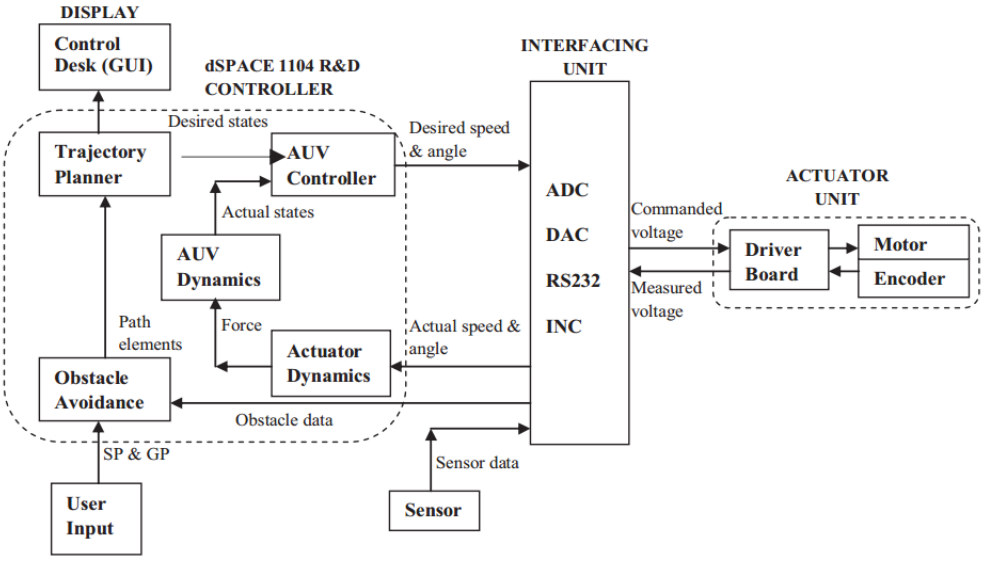

Figura I.3: Estructura del sistema de simulación HIL [8]

D. Lee *et al.* presentaron un sistema de control adaptativo basado en visión para un quadrotor subactuado [10]. El algoritmo propuesto es validado con un sistema de visión integrado HIL, este sistema se compone de una cámara, aviónica, simulador de vuelo y monitor. Como se muestra en la Figura I.4 y la Figura I.5, el plano de imagen virtual representa una imagen del objetivo de la cámara montada sobre el quadrotor. El control de vuelo del quadrotor calcula las entradas usando la

información visual de un sensor de visión real. Entonces la respuesta del quadrotor es simulado usando el simulador de vuelo implementado en xPC.

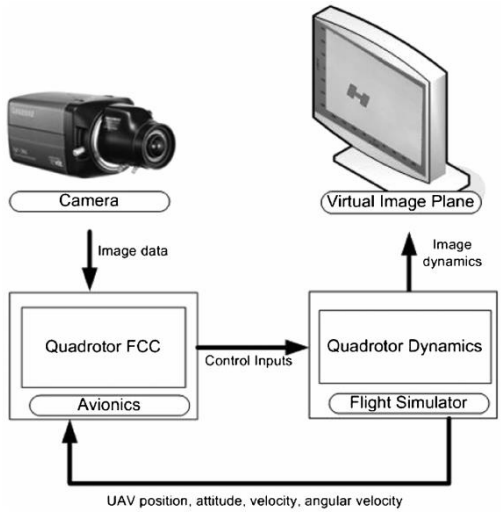

Figura I.4: Diagrama de bloques para el sistema de visión integrada HIL [9]

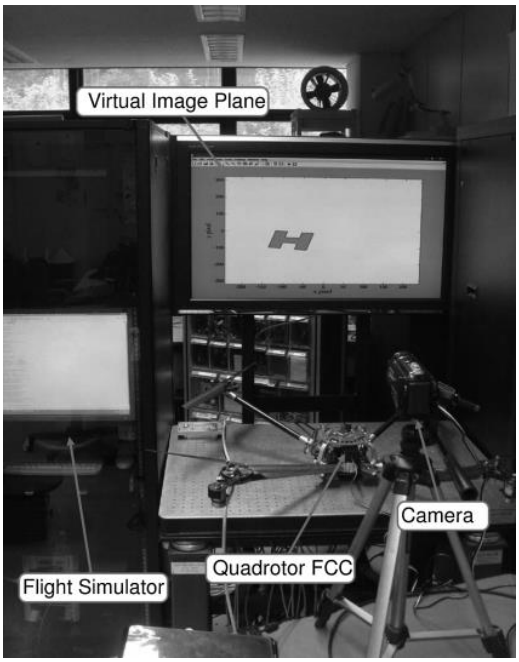

Figura I.5: Visión integrada del sistema HIL [9]

Existen muchas plataformas para implementar un simulador HIL, se pueden dividir en dos grandes grupos, sistemas basados en PC y sistemas embebidos. En el primer grupo se usa hardware de una computadora personal, la cual es equipada con una tarjeta de adquisición de datos [11]. En el grupo de sistemas embebidos se

encuentran diversas tecnologías como son DSPs, microcontroladores, FPGA, estos suelen ser más económicos y muchos fabricantes proporcionan tarjetas de evaluación de sus productos, lo que hace ampliamente disponibles, tal como lo es el kit de desarrollo Hercules Launchpad LAUNCHXL2-RM57L de Texas Instruments [12], el cual se utilizara en este trabajo para el desarrollo del simulador HIL.

Objetivos de la tesis

Objetivo general

Diseñar e implementar un simulador HIL que simule el comportamiento dinámico de un sistema carro-péndulo invertido, en el que muestre el movimiento de traslación y rotación del sistema en un entorno virtual, con el fin de probar diferentes técnicas de control para su estabilización.

Objetivos específicos

1. Recopilar información sobre HIL e Interfaz gráfica.
2. Simular el sistema carro-péndulo en Matlab/Simulink y LabVIEW.
3. Desarrollar el simulador HIL, usando la tarjeta Hercules LAUNCHXL2-RM57L con un filtro que acondicione las señales.
4. Mostrar una interfaz gráfica donde se visualice los movimientos de la planta en función de la entrada de control, en una computadora personal.
5. Probar dos técnicas de control para comparar la eficiencia del emulador y la planta real.
6. Redactar el documento de tesis.

Descripción del documento

La estructura de la tesis está formada por 5 capítulos y dos apéndices, y su contenido es el siguiente:

En el capítulo 1 se describe el sistema carro-péndulo invertido, su modelado matemático y las partes que conforma un simulador HIL.

En el capítulo 2 se presenta el hardware utilizado (Hercules RM57Lx) para el desarrollo del simulador; los módulos utilizados de la tarjeta, y además se describe el circuito de acondicionamiento de señales.

En el capítulo 3 se muestra el diseño del software, se describe la forma de programación de la interfaz gráfica desarrollada en LabVIEW y Unity, la manera de interactuar entre el usuario y la interfaz.

En el capítulo 4 se presentan los resultados obtenidos durante las pruebas del controlador junto al simulador del Carro-Péndulo-Invertido (CPI).

En el capítulo 5 se redactan las conclusiones del trabajo de tesis y los trabajos a futuro.

Se incluyen dos apéndices, en el A se expone el código utilizado para programar la tarjeta Hercules RM57Lx. En el apéndice B se escribe el código fuente de los scripts utilizado en Unity 3D.

Capítulo 1
Marco teórico

El péndulo invertido es uno de los experimentos de laboratorio más populares, usado para ilustrar las técnicas de control no lineal. Este sistema tiene aplicaciones en diversas áreas de la ingeniería, tales como el control de cohetes y el control antisísmico de edificios [8]. Por otro lado, la simulación HIL es una técnica que es usada cada vez más en el desarrollo y pruebas de sistemas embebidos complejos en tiempo real. El propósito de la simulación HIL es proporcionar una plataforma efectiva para desarrollar y probar sistemas embebidos en tiempo real [13].

En este capítulo se hace referencia al marco teórico, donde se da a conocer los conceptos básicos acerca de las partes que conforman un simulador Hardware-In-The-Loop (HIL), posteriormente se presenta el modelo dinámico del sistema carro-péndulo.

1.1. Sistema Carro-Péndulo-Invertido.

El sistema Carro-Péndulo Invertido, también conocido como péndulo invertido, es un sistema mecánico que consta de una barra (péndulo) montado sobre una plataforma móvil (carro). El péndulo no tiene control directo sobre sí misma, naturalmente tiende a caerse desde cualquier posición. El carro solo puede moverse horizontalmente y es accionada por una fuerza aplicada en la misma dirección.

Un sistema de este tipo se cuenta en el laboratorio de posgrado de la Universidad Tecnológica de la Mixteca, de la marca Feedback, modelo 33-005-PCI,

tal como lo muestra la Figura 1.1, actualmente tiene un costo de $385,752 pesos. La configuración del carro-péndulo consiste de una barra montada en un carro de tal manera que el péndulo pueda oscilar libremente en el plano vertical. El carro es accionado por un motor de CD el cual mueve una banda dentada y este a su vez al carro. Para hacer girar y equilibrar el péndulo, el carro es empujado hacia atrás y hacia delante en un carril de longitud limitada [14].

El problema de control del péndulo es llevarlo a su posición de equilibrio inestable (que es la posición vertical), lograr colocarlo lo más pronto posible y con pocas oscilaciones. Después de alcanzar la posición deseada el sistema debería mantenerse en este estado a pesar de las perturbaciones externas [15].

Figura 1.1: Carro-péndulo invertido de la compañía Feedback [13]

1.2. Modelo dinámico del carro-péndulo

El primer paso para controlar un sistema y en particular para poder comprender el comportamiento del sistema carro-péndulo, es obtener su modelo dinámico. La dinámica del carro-péndulo trata con las formulaciones matemáticas de las ecuaciones de movimiento del péndulo, ésta son un conjunto de ecuaciones diferenciales que describen su conducta dinámica, con el cual es posible hacer su simulación en computadora, y a partir de él, diseñar el controlador.

El modelo dinámico se puede obtener a partir de leyes físicas conocidas, tales como las leyes de la mecánica Newtoniana y Lagrangiana [16]. Es importante hacer notar que, para la simulación HIL, se obtendrá el modelo dinámico no lineal del

sistema con la finalidad de que se tenga a la salida una respuesta más cercana a la real.

Metodología Euler Lagrange

Se obtendrá el modelo dinámico del carro-péndulo utilizando el método de Euler-Lagrange partiendo de la ecuación (1.1), después se incluirá (acoplará) el modelo dinámico del motor de DC (Direct Current), finalmente se hace un cambio a variables de estado. El Lagrangiano de un sistema dinámico es una función que resume la dinámica del sistema. Se define como la energía cinética (K) menos la energía potencial (P).

$$L = K - P \tag{1.1}$$

Si se conoce el Lagrangiano de un sistema, se pueden obtener las ecuaciones de movimiento mediante la sustitución directa de la expresión en la ecuación de Euler-Lagrange (véase la ecuación (1.2)). Ésta se utiliza para describir cualquier sistema mecánico por medio de coordenadas generalizadas de posición, y el Lagrangiano se utiliza para los sistemas conservativos [17].

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}(q,\dot{q})\right) - \frac{\partial L}{\partial q}(q,\dot{q}) + \frac{\partial D}{\partial \dot{q}} = \tau \tag{1.2}$$

Donde $q = (q_1, \dots, q_n)^T$ representan las variables generalizadas, una por cada grado de libertad del sistema, $\tau = (\tau_1, \dots, \tau_n)^T$ denota las fuerzas que son aplicadas externamente al sistema y D es la función de disipación de Rayleigh [8].

En este caso, las variables generalizadas son: x que indica la posición del carro y θ que representa la posición del péndulo, esto es $q = [x, \theta]^T$. Por lo tanto, se tiene:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} + \frac{\partial D}{\partial \dot{x}} = F \tag{1.3}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} + \frac{\partial D}{\partial \dot{\theta}} = 0 \tag{1.4}$$

Donde F es la fuerza que produce el motor de DC para mover el carro. En la Figura 1.2 se muestran las fuerzas que actúan en el carro-péndulo, considerando los siguientes parámetros.

Tabla 1: Parámetros del sistema carro-péndulo invertido

Parámetro	Descripción	Valor	Unidades
$\boldsymbol{M}$	Masa del carro.	1.12	kg
$\boldsymbol{m}$	Masa del péndulo.	0.11	kg
$\boldsymbol{l}$	Distancia de la articulación al centro de gravedad del péndulo.	0.3434	m
$\boldsymbol{I_P}$	Momento de inercia del péndulo en relación con su centro de gravedad.	0.0136	kgm^2
$\boldsymbol{g}$	Aceleración debida a la gravedad.	9.81	m/s^2
$\boldsymbol{x}$	Distancia del centro de masa del carro respecto a su posición inicial.		m
$\boldsymbol{\theta}$	Ángulo que forma el péndulo con respecto a la vertical.		rad
$\boldsymbol{F}$	Fuerza aplicada sobre el carro		N
$\boldsymbol{F_p}$	Fricción del péndulo	0.007	Nms/rad
$\boldsymbol{F_c}$	Fricción del carro	0.05	Ns/m

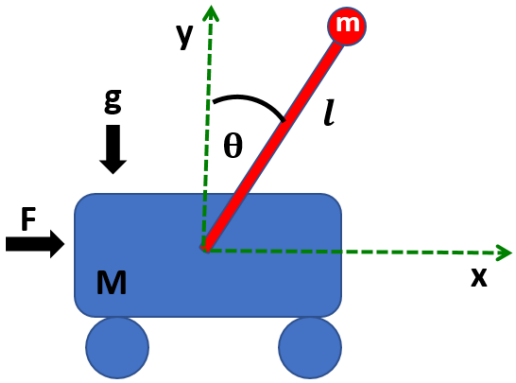

Figura 1.2: Sistema carro-péndulo

Las coordenadas del centro de masa del péndulo son:

$$x_1 = x + l\sin(\theta) \tag{1.5}$$

$$y_1 = l\cos(\theta) \tag{1.6}$$

Derivando con respecto al tiempo las ecuaciones de posición del centro de masa (1.5) y (1.6) se obtienen las ecuaciones de velocidad.

$$\dot{x}_1 = \dot{x} + l\cos(\theta)\dot{\theta} \tag{1.7}$$

$$\dot{y}_1 = -l\sin(\theta)\dot{\theta} \tag{1.8}$$

La energía cinética del carro es K_1:

$$K_1 = \frac{1}{2}M\dot{x}^2 \tag{1.9}$$

La energía cinética del péndulo es K_2:

$$K_2 = \frac{1}{2}m\dot{x}_1^{\,2} + \frac{1}{2}m\dot{y}_1^{\,2} + \frac{1}{2}I_P\dot{\theta}^2 \tag{1.10}$$

$$K_2 = \frac{1}{2}m(\dot{x}_1^{\,2} + \dot{y}_1^{\,2}) + \frac{1}{2}I_P\dot{\theta}^2 \tag{1.11}$$

Elevando las ecuaciones (1.7) y (1.8) al cuadrado, se obtiene:

$$\begin{aligned} \dot{x}_1^{\,2} &= \left[\dot{x} + l\cos(\theta)\dot{\theta}\right]^2 \\ &= \dot{x}^2 + 2\dot{x}l\cos(\theta)\dot{\theta} + l^2\cos^2(\theta)\dot{\theta}^2 \\ \dot{y}_1^{\,2} &= l^2\sin^2(\theta)\dot{\theta}^2 \end{aligned} \tag{1.12}$$

Sumando,

$$\begin{aligned} \dot{x}_1^{\,2} + \dot{y}_1^{\,2} &= \dot{x}^2 + 2\dot{x}l\cos(\theta)\dot{\theta} + l^2\cos^2(\theta)\dot{\theta}^2 + l^2\sin^2(\theta)\dot{\theta}^2 \\ &= \dot{x}^2 + 2\dot{x}l\cos(\theta)\dot{\theta} + l^2\dot{\theta}^2\left[\cos^2(\theta) + \sin^2(\theta)\right] \end{aligned}$$

Entonces la suma queda simplificada

$$\dot{x}_1^{\,2} + \dot{y}_1^{\,2} = \dot{x}^2 + 2\dot{x}l\cos(\theta)\dot{\theta} + l^2\dot{\theta}^2 \tag{1.13}$$

Sustituyendo (1.13) en (1.11) se obtiene.

$$\begin{aligned} K_2 &= \frac{1}{2} m(\dot{x}_1^{\,2} + \dot{y}_1^{\,2}) + \frac{1}{2} I_P \dot{\theta}^2 \\ &= \frac{1}{2} m\left[\dot{x}^2 + 2\dot{x}l\cos(\theta)\dot{\theta} + l^2\dot{\theta}^2\right] + \frac{1}{2} I_P \dot{\theta}^2 \end{aligned} \tag{1.14}$$

La energía cinética total está dada por la suma de las energías cinéticas de las ecuaciones (1.9) y(1.14):

$$\begin{aligned} K &= K_1 + K_2 \\ &= \frac{1}{2} M\dot{x}^2 + \frac{1}{2} m\left[\dot{x}^2 + 2\dot{x}l\cos(\theta)\dot{\theta} + l^2\dot{\theta}^2\right] + \frac{1}{2} I_P \dot{\theta}^2 \\ &= \frac{1}{2}(M+m)\dot{x}^2 + m\dot{x}l\cos(\theta)\dot{\theta} + \frac{1}{2}(I_P + ml^2)\dot{\theta}^2 \end{aligned} \tag{1.15}$$

La energía potencial del sistema está dada por:

$$P = mgl\cos(\theta) \tag{1.16}$$

Considerando la ecuación de disipación de Rayleigh:

$$D = \frac{1}{2} F_c \dot{x}^2 + \frac{1}{2} F_p \dot{\theta}^2 \tag{1.17}$$

Haciendo la resta correspondiente a la ecuación (1.1), se obtiene:

$$L = \frac{1}{2}(M+m)\dot{x}^2 + m\dot{x}l\cos(\theta)\dot{\theta} + \frac{1}{2}(I_P + ml^2)\dot{\theta}^2 - mgl\cos(\theta) \tag{1.18}$$

Obteniendo las derivadas parciales:

$$\frac{\partial L}{\partial \dot{x}} = (M+m)\dot{x} + ml\cos(\theta)\dot{\theta} \tag{1.19}$$

$$\frac{\partial L}{\partial x} = 0 \tag{1.20}$$

$$\frac{\partial D}{\partial \dot{x}} = F_c \dot{x} \tag{1.21}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) = (M+m)\ddot{x} + ml\left[-\sin(\theta)\dot{\theta}^2 + \cos(\theta)\ddot{\theta}\right] \tag{1.22}$$

$$\frac{\partial L}{\partial \dot{\theta}} = (ml^2 + I_P)\dot{\theta} + ml\dot{x}\cos(\theta) \tag{1.23}$$

$$\frac{\partial L}{\partial \theta} = -ml\dot{x}\sin(\theta)\dot{\theta} + mgl\sin(\theta) \tag{1.24}$$

$$\frac{\partial D}{\partial \dot{\theta}} = F_p\dot{\theta} \tag{1.25}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) = (ml^2 + I_P)\ddot{\theta} + ml\left[\ddot{x}\cos(\theta) - \dot{x}\sin(\theta)\dot{\theta}\right] \tag{1.26}$$

Sustituyendo (1.20), (1.21) y (1.22) en la ecuación (1.3) se obtiene:

$$(M+m)\ddot{x} + ml\left[-\sin(\theta)\dot{\theta}^2 + \cos(\theta)\ddot{\theta}\right] + F_c\dot{x} = F \tag{1.27}$$

Sustituyendo (1.24), (1.25) y (1.26) en la ecuación (1.4) se obtiene:

$$\begin{aligned}
(ml^2 + I_P)\ddot{\theta} + ml\left[\ddot{x}\cos(\theta) - \dot{x}\sin(\theta)\dot{\theta}\right] + ml\dot{x}\sin(\theta)\dot{\theta} - mgl\sin(\theta) + F_p\dot{\theta} = 0 \\
(ml^2 + I_P)\ddot{\theta} + ml\ddot{x}\cos(\theta) - ml\dot{x}\sin(\theta)\dot{\theta} + ml\dot{x}\sin(\theta)\dot{\theta} - mgl\sin(\theta) + F_p\dot{\theta} = 0 \\
(ml^2 + I_P)\ddot{\theta} + ml\left[\ddot{x}\cos(\theta)\right] - mgl\sin(\theta) + F_p\dot{\theta} = 0
\end{aligned} \tag{1.28}$$

Las ecuaciones (1.27) y (1.28) pertenecen al sistema del carro-péndulo, como se puede ver son no lineales. El motor de CD (ver Figura 1.3) es quien produce la fuerza que mueve al carro-péndulo, por lo cual es necesario incluirlo en el modelo. La armadura esta modelada como un circuito con resistencia R_a conectada en serie con una inductancia L_a y una fuente de voltaje que representa la fuerza contraelectromotriz en la armadura [18], las variables y parámetros del motor se definen en la Tabla 2.

El modelado del motor de corriente directa requiere de dos ecuaciones, una ecuación mecánica y otra eléctrica. Estas ecuaciones están acopladas, la ecuación mecánica (1.31), modela el movimiento del rotor. La ecuación eléctrica (1.30) se obtiene de la suma de los voltajes en cada elemento que conforma la malla.

Se supondrá, que la constante eléctrica (K_e) y la constante mecánica (K_m) son iguales en magnitud para mantener la relación electro-mecánica. Entonces se tiene:

$$k_e = k_m = k \tag{1.29}$$

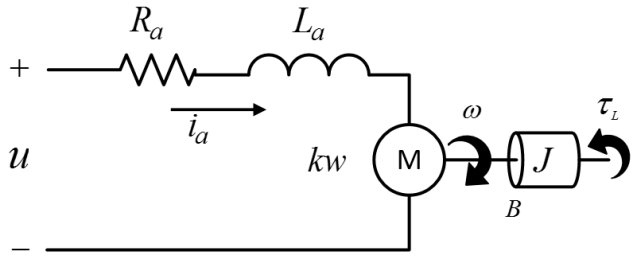

Figura 1.3: Dinámica del motor de CD

$$L_a \frac{di_a}{dt} = u - R_a i_a - k\omega \tag{1.30}$$

$$J \frac{dw}{dt} = k i_a - B\omega - \tau_L \tag{1.31}$$

Tabla 2: Parámetros del motor de DC del sistema carro-péndulo.

Parámetros	Descripción	Valor	Unidades
$\boldsymbol{R_a}$	Resistencia de armadura: Es el valor de la resistencia al paso de corriente de la carcasa del motor.	3.9	Ω
$\boldsymbol{L_a}$	Inductancia del circuito de armadura: Inductancia del embobinado que está dentro del rotor.	8.9E-3	mH
$\boldsymbol{K_e}$	Constante eléctrica: Relaciona la velocidad angular de la flecha y el voltaje generado.	53.5E-3	Vs/rad
$\boldsymbol{K_m}$	Constante mecánica: Relaciona la corriente eléctrica del circuito y el torque generado.	53.5E-3	Nm/A
$\boldsymbol{k}$	Simboliza la igualdad entre $\boldsymbol{K_e = K_m}$	53.5E-3	

Tabla 3: Parámetros del motor de DC del sistema carro-péndulo (continuación).

Parámetros	Descripción	Valor	Unidades
$\boldsymbol{B}$	Coeficiente de fricción viscosa rotacional: Representa la oposición al movimiento del motor.	50E-6	Nms/rad
$\boldsymbol{J}$	Momento de inercia: Es la inercia rotacional asociada a la geometría del rotor.	0.11E-4	kgm^2
$\boldsymbol{i_a}$	Corriente del circuito de armadura.		mA
$\boldsymbol{\omega}$	Velocidad angular: Velocidad angular a la que gira el rotor.		rad/s
$\boldsymbol{\tau_L}$	Par de carga constante desconocido.		Nm
$\boldsymbol{u}$	Entrada de control: Es el voltaje que se aplica en las terminales del motor.		V
$\boldsymbol{R}$	Radio del disco montado en el eje del motor	0.02616	m

El par de carga se genera a partir de la fuerza aplicada al disco por el radio del disco:

$$\tau_L = RF \tag{1.32}$$

Para simplificar el modelo, se va a suponer que la inductancia es muy pequeña en (1.30), por lo cual se llega a lo siguiente.

$$L_a \to 0 \Rightarrow L_a \frac{di_a}{dt} = 0 \tag{1.33}$$

$$u - R_a i_a - k\omega = 0$$
$$R_a i_a = u - k\omega$$
$$i_a = \frac{u - k\omega}{R_a}$$

Se sustituye i_a , en (1.31), se produce lo siguiente:

$$J\frac{d\omega}{dt} = k\left[\frac{u - k\omega}{R_a}\right] - B\omega - RF$$
$$J\frac{d\omega}{dt} = \left(\frac{k}{R_a}\right)u - \left(\frac{k^2}{R_a} + B\right)\omega - RF \tag{1.34}$$

Se sabe que la posición está en función del radio del disco del motor (R) y el ángulo α (desplazamiento angular) como lo muestra la Figura 1.4.

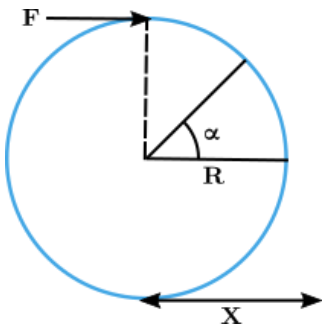

Figura 1.4: Desplazamiento angular del disco

Entonces la posición es dada por:

$$x = R\alpha \Leftrightarrow \alpha = \frac{x}{R}$$
$$\dot{x} = R\dot{\alpha} \Leftrightarrow \dot{\alpha} = \frac{\dot{x}}{R} = \omega \tag{1.35}$$
$$\frac{d\omega}{dt} = \frac{\ddot{x}}{R} = \ddot{\alpha}$$

Por lo tanto, se sustituye (1.35) en(1.34), y como consecuencia se tiene el resultado:

$$J\frac{\ddot{x}}{R} = \left(\frac{k}{R_a}\right)u - \left(\frac{k^2}{R_a} + B\right)\frac{\dot{x}}{R} - RF \tag{1.36}$$

Despejando F de la ecuación (1.36)

$$F = -\left(\frac{k^2}{R_a} + B\right)\frac{\dot{x}}{R^2} - \frac{J}{R^2}\ddot{x} + \frac{k}{RR_a}u \tag{1.37}$$

Se igualan las dinámicas del carro-péndulo (1.27) y del motor(1.37), obteniendo:

$$(M+m)\ddot{x} + ml\left[\ddot{\theta}\cos(\theta) - \dot{\theta}^2\sin(\theta)\right] + F_c\dot{x} = -\left(\frac{k^2}{R_a} + B\right)\frac{\dot{x}}{R^2} - \frac{J}{R^2}\ddot{x} + \frac{k}{RR_a}u \quad (1.38)$$

Con esto se obtienen dos ecuaciones del modelado dinámico del sistema carro-péndulo invertido:

$$\left[\frac{J}{R^2} + (M+m)\right]\ddot{x} + \left[\frac{k^2}{R^2R_a} + \frac{B}{R^2} + F_c\right]\dot{x} + ml\left[\ddot{\theta}\cos(\theta) - \dot{\theta}^2\sin(\theta)\right] = \frac{k}{RR_a}u \quad (1.39)$$

$$(ml^2 + I)\ddot{\theta} + ml\left[\ddot{x}\cos(\theta)\right] - mgl\sin(\theta) + F_p\dot{\theta} = 0 \quad (1.40)$$

Para facilitar los cálculos se asignan las siguientes constantes:

$$a_1 = \left[\frac{J}{R^2} + (M+m)\right] \quad (1.41)$$

$$a_2 = \left[\frac{k^2}{R^2R_a} + \frac{B}{R^2} + F_c\right] \quad (1.42)$$

$$a_3 = ml \quad (1.43)$$

$$a_4 = ml^2 + I \quad (1.44)$$

$$a_5 = mgl \quad (1.45)$$

$$b_1 = \frac{k}{RR_a} \quad (1.46)$$

Sustituyendo estas constantes en las ecuaciones (1.39) y (1.40), se tiene :

$$a_1\ddot{x} + a_2\dot{x} + a_3\left[\ddot{\theta}\cos(\theta) - \dot{\theta}^2\sin(\theta)\right] = b_1u \quad (1.47)$$

$$a_4\ddot{\theta} + a_3\left[\ddot{x}\cos(\theta)\right] - a_5\sin(\theta) + F_p\dot{\theta} = 0 \quad (1.48)$$

Despejando $\ddot{x}$ de (1.47).

$$\ddot{x} = -\frac{a_2}{a_1}\dot{x} - \frac{a_3}{a_1}\left[\ddot{\theta}\cos(\theta) - \dot{\theta}^2\sin(\theta)\right] + \frac{b_1}{a_1}u \tag{1.49}$$

Se sustituye este último en (1.48).

$$a_4\ddot{\theta} + a_3\cos(\theta)\left[-\frac{a_2}{a_1}\dot{x} - \frac{a_3}{a_1}\left[\ddot{\theta}\cos(\theta) - \dot{\theta}^2\sin(\theta)\right] + \frac{b_1}{a_1}u\right] - a_5\sin(\theta) + F_p\dot{\theta} = 0 \tag{1.50}$$

$$\begin{aligned}\left[a_4 - \frac{a_3^2}{a_1}\cos^2(\theta)\right]\ddot{\theta} &= -F_p\dot{\theta} - \frac{a_3^2}{a_1}\dot{\theta}^2\sin(\theta)\cos(\theta) + a_5\sin(\theta) + \\ &\frac{a_2a_3}{a_1}\cos(\theta)\dot{x} - \frac{a_3b_1}{a_1}\cos(\theta)u\end{aligned} \tag{1.51}$$

Haciendo un cambio de variable

$$\beta = a_4 - \frac{a_3^2}{a_1}\cos^2(\theta) \tag{1.52}$$

Se despeja la aceleración angular $\ddot{\theta}$ de (1.51)

$$\ddot{\theta} = -\frac{F_p}{\beta}\dot{\theta} - \frac{a_3^2}{a_1\beta}\dot{\theta}^2\sin(\theta)\cos(\theta) + \frac{a_5}{\beta}\sin(\theta) + \frac{a_2a_3}{a_1\beta}\cos(\theta)\dot{x} - \frac{a_3b_1}{a_1\beta}\cos(\theta)u \tag{1.53}$$

Sustituyendo $\ddot{\theta}$ en (1.49) y finalmente se obtiene la ecuación para la aceleración lineal:

$$\ddot{x} = -\frac{a_2}{a_1}\dot{x} - \frac{a_3}{a_1}\left[\begin{pmatrix} -\frac{F_p}{\beta}\dot{\theta} - \frac{a_3^2}{a_1\beta}\dot{\theta}^2\sin(\theta)\cos(\theta) + \\ \frac{a_5}{\beta}\sin(\theta) + \frac{a_2a_3}{a_1\beta}\cos(\theta)\dot{x} - \frac{a_3b_1}{a_1\beta}\cos(\theta)u \end{pmatrix}\cos(\theta) - \dot{\theta}^2\sin(\theta)\right] + \frac{b_1}{a_1}u \tag{1.54}$$

Un sistema no lineal puede ser representado en forma de variables de estado como se muestra en la ecuación (1.55).

$$\begin{aligned}\dot{x} &= f(x) + g(x)u \\ y &= h(x)\end{aligned} \tag{1.55}$$

Donde $x \in \mathbb{R}^n, u \in \mathbb{R}$ los cuales son las variables de estado y de control respectivamente. $y \in \mathbb{R}^n$ denota la variable de salida f y g son los vectores función. Tomando como referencia la ecuación (1.55) se definen las variables de estado como:

$$\begin{aligned} x_1 &= x \\ x_2 &= \dot{x} \\ x_3 &= \theta \\ x_4 &= \dot{\theta} \end{aligned} \tag{1.56}$$

De acuerdo con las variables de estado antes mencionadas se obtienen las siguientes ecuaciones que representan el modelo de estado no lineal:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{a_2}{a_1} x_2 - \frac{a_3}{a_1}\left[\left(\begin{array}{c} -\frac{F_p}{\beta} x_4 - \frac{a_3^2}{a_1\beta} x_4^2 \sin(x_3)\cos(x_3) + \\ \frac{a_5}{\beta}\sin(x_3) + \frac{a_2 a_3}{a_1\beta}\cos(x_3)x_2 - \frac{a_3 b_1}{a_1\beta}\cos(x_3)u \end{array}\right)\cos(x_3) - x_4^2\sin(x_3)\right] + \frac{b_1}{a_1}u \\ \dot{x}_3 &= x_4 \\ \dot{x}_4 &= -\frac{F_p}{\beta} x_4 - \frac{a_3^2}{a_1\beta} x_4^2 \sin(x_3)\cos(x_3) + \frac{a_5}{\beta}\sin(x_3) + \frac{a_2 a_3}{a_1\beta}\cos(x_3)x_2 - \\ & \frac{a_3 b_1}{a_1\beta}\cos(x_3)u \end{aligned} \tag{1.57}$$

1.3. Simulación de un sistema

Los simuladores son programas de computadora que predicen el comportamiento dinámico de los sistemas. Estos paquetes de software se basan en el modelado matemático de los elementos que constituyen los sistemas y de las señales que los excitan. La validez de los simuladores depende de la aproximación que hay entre los modelos matemáticos de los componentes y sus verdaderos comportamientos físicos. Por lo que una mayor sofisticación de los modelos supondrá que se aproxime más verazmente al comportamiento físico, produciendo casi nulas diferencias entre lo indicado en la simulación y en su implementación física. Por tanto, no es de extrañar la importancia que tienen los simuladores en las empresas [19].

Algunos ejemplos de simuladores comerciales son PSpice, MATLAB/Simulink, Proteus, los cuales son paquetes de software utilizados en la ingeniería para simular un sistema.

1.4. Simulador Hardware-In-The-Loop

La simulación HIL es una técnica de prueba que simula el comportamiento de entradas y salidas de un sistema físico que es conectada a un controlador en tiempo real [3]. Constituye una plataforma efectiva, ya que incluye toda la complejidad de la planta que controla el sistema embebido. Con esta técnica la planta o máquina física es remplazado por una simulación.

La simulación HIL es un tipo de simulación en tiempo real. Se utiliza para probar diseños de controladores, para mostrar cómo responde el control en tiempo real. La Figura 1.5, muestra una configuración típica de simulación HIL, el sistema es un tanque de reacción con agitación continua (TRAC) [20], es utilizado en la fabricación de productos químicos de una cierta concentración. El controlador (PLC) es conectado a una tarjeta de adquisición de datos (DAQ) que se encarga de recibir la señal de control y enviarla a la planta, de igual forma, recibir las variables de la planta y enviarlas al controlador. Usa una computadora personal para simular y visualizar (LabVIEW) en tiempo real el comportamiento del modelo matemático de la planta. En esta simulación HIL el modelo matemático de la planta es implementado en software de una computadora personal y el controlador en Hardware (PLC), sin embargo, en ocasiones el algoritmo de control es implementado en software de una computadora personal y el modelo matemático de la planta en Hardware.

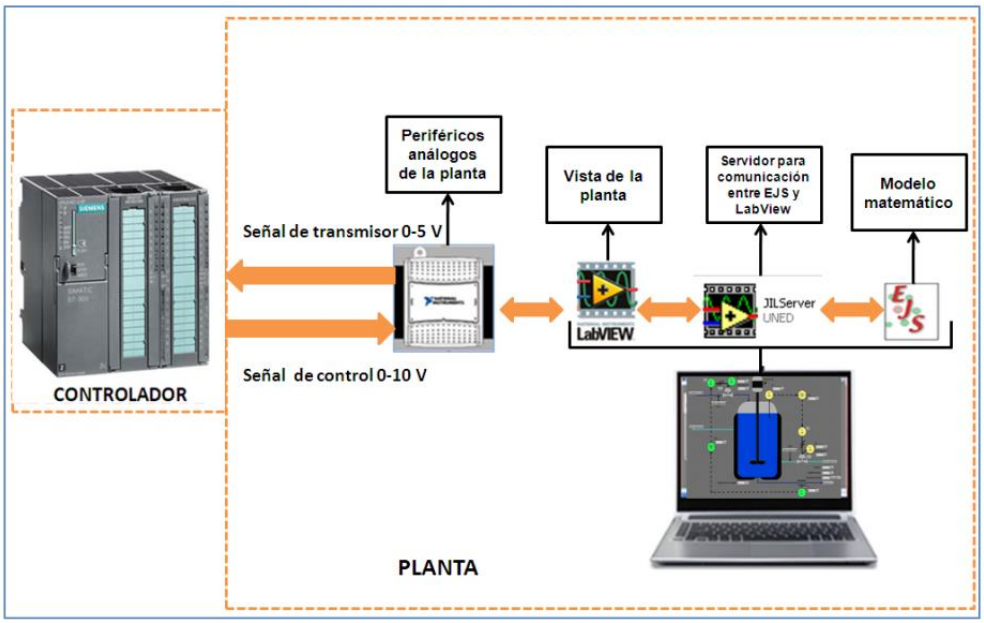

Figura 1.5: Configuración típica de simulación HIL [18]

Como trabaja el simulador HIL

La simulación HIL incluye la simulación de sensores y actuadores. Estas simulaciones sirven de interfaz entre el modelo de la planta y el sistema de control bajo prueba, como se observa en la Figura 1.6. El valor de cada sensor está controlado por el modelo de planta y es leído por el sistema embebido. Del mismo modo, el sistema embebido bajo prueba ejecuta su algoritmo de control por medio de las señales de los actuadores. Igualmente, cambios en las señales de control provocan cambios en los valores de las variables en el modelo de simulación de la planta.

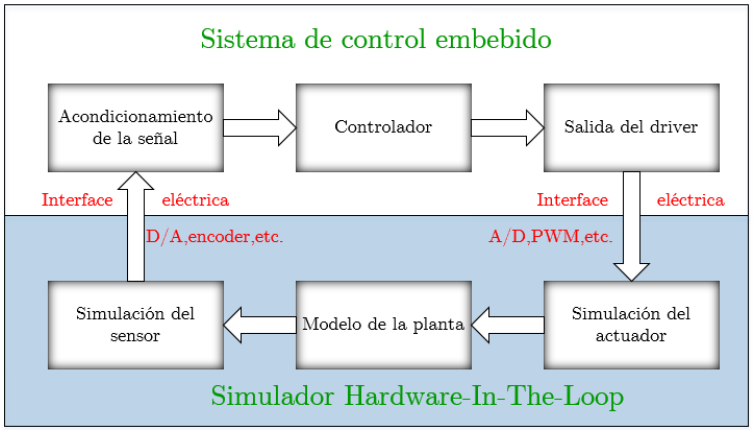

Figura 1.6: Esquema de un simulador HIL

Un sistema físico que sea frágil, costoso o difícil de conseguir puede ser probado en interacción con un controlador usando la simulación HIL. En un sistema de control convencional, el control recibe las señales de los sensores que miden algunas variables de salida de la planta y entrega las señales de control a los actuadores que alimentan la planta, un simulador HIL debe ser capaz de generar y adquirir señales que podrían entregar el sistema físico con las mismas características de amplitud y velocidad de cambio, esto es, debería poder simular los sensores, el acople de los actuadores y de ser necesario las impedancias de acople al sistema de control. Una simulación HIL no reemplaza las pruebas con una planta real pero ayuda a evaluar alternativas de diseño e identificar errores que pueden reducir costos al tener un menor impacto sobre el sistema final [2].

El modelado matemático de la planta de un simulador HIL se escribe en ecuaciones diferenciales, para obtener una solución aproximada es necesario recurrir a los métodos numéricos, uno de los métodos numéricos más sencillos para llevar a cabo esta tarea es el método de Euler.

1.5. Método de Euler

El modelo dinámico del sistema carro-péndulo consta de ecuaciones diferenciales, tal que se necesitan resolver para llevar al péndulo a su posición deseada.

A veces no es posible obtener la solución de una ecuación diferencial, pero sí se puede encontrar una aproximación satisfactoria. Estas aproximaciones se hallan usando métodos numéricos, el método de Euler o método de las tangentes es uno de los métodos más sencillos para obtener la solución [21]. Cabe señalar que este método se aplica para encontrar la solución a ecuaciones diferenciales ordinarias (EDO), esto es, cuando la función involucra solo una variable independiente.

Para describir el método de Euler, se parte del problema de valor inicial, dadas las condiciones $y(t_0) = y_0$, donde t es la variable independiente.

$$\begin{aligned} &\frac{dy}{dt} = f(t, y) \\ &y(t_0) = y_0 \end{aligned} \tag{1.58}$$

Como es dado $f(t, y)$, se puede trazar su campo de pendientes en el plano *t-y.* La idea del método es empezar en el punto (t_0, y_0) en el campo de pendientes y dar pequeños pasos dictados por las tangentes de ésta.

Se elige un tamaño de paso Δt(pequeño), en cada paso se mueve Δt unidades a lo largo del eje t. El tamaño de Δt determina la exactitud de la solución, así como el número de cálculos que son necesarios para obtener la aproximación.

Se parte en (t_0, y_0), el primer paso es hacia el punto (t_1, y_1) donde $t_1 = t_0 + \Delta t$ y (t_1, y_1) es el punto sobre la línea que pasa por (t_0, y_0) y cuya pendiente es proporcionada por el campo de pendientes en(t_0, y_0). De la misma manera se utiliza el campo de pendientes en el punto (t_k, y_k) para calcular el siguiente punto (t_{k+1}, y_{k+1}). La secuencia de valores y_0, y_1, y_2 sirve como una aproximación a la solución en los tiempos t_0, t_1, t_2, etc. Geométricamente el método produce una secuencia de pequeños segmentos de línea que conectan (t_k, y_k) con (t_{k+1}, y_{k+1}).

Para poner en práctica el método de Euler, se necesita una ecuación que determine (t_{k+1}, y_{k+1}) a partir de (t_k, y_k). Para encontrar t_{k+1}, primero se especifica el tamaño de paso Δt y entonces

$$t_{k+1} = t_k + \Delta t \tag{1.59}$$

Para obtener y_{k+1}, se utiliza la ecuación diferencial. Se sabe que la pendiente de la solución a la ecuación $dy/dx = f(t,y)$ en el punto (t_k, y_k) es $f(t_k, y_k)$y el método de Euler usa esta pendiente para calcular y_{k+1}. De hecho, determina el punto (t_{k+1}, y_{k+1}) suponiendo que éste se encuentra sobre la línea que pasa por (t_k, y_k) con pendiente $f(t_k, y_k)$ (ver la Figura 1.7).

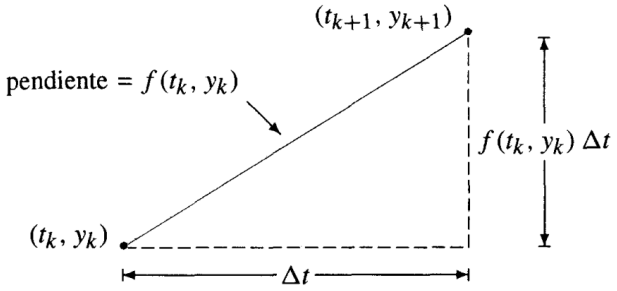

Figura 1.7: Pendiente en el punto (t_k, y_k) [19]

Usando la fórmula para la pendiente queda:

$$\frac{y_{k+1} - y_k}{t_{k+1} - t_k} = f(t_k, y_k) \tag{1.60}$$

De (1.59) se despeja Δt entonces se tiene:

$$\Delta t = t_{k+1} - t_k \tag{1.61}$$

Se sustituye (1.61) en (1.60), quedando:

$$\begin{aligned} \frac{y_{k+1} - y_k}{\Delta t} &= f(t_k, y_k) \\ y_{k+1} - y_k &= f(t_k, y_k)\Delta t \\ y_{k+1} &= y_k + f(t_k, y_k)\Delta t \end{aligned} \tag{1.62}$$

Esta es la fórmula para el método de Euler. En la Figura 1.8 se muestran las pendientes de cada paso.

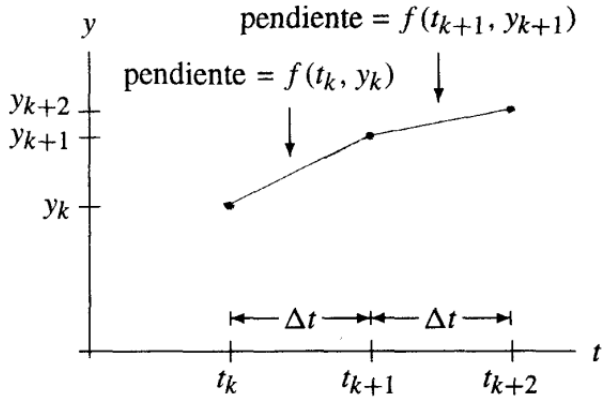

Figura 1.8: Dos pasos sucesivos del método de Euler [19]

Método de Euler resumido [21], para

$$\frac{dy}{dt} = f(t, y) \tag{1.63}$$

Dada la condición inicial $y(t_0) = y_0$ y el tamaño de paso Δt, calcule el punto (t_{k+1}, y_{k+1}) a partir del punto (t_k, y_k) como sigue:

Use la ecuación diferencial para determinar la pendiente $f(t_k, y_k)$. Calcule el siguiente punto (t_{k+1}, y_{k+1}) mediante las fórmulas $t_{k+1} = t_k + \Delta t$ y $y_{k+1} = y_k + f(t_k, y_k)\Delta t$.

El método de Euler se implementa en una computadora o en cualquier sistema embebido, para calcular la solución aproximada en tiempo real.

1.6. Componentes de Hardware y Software

Se requiere una plataforma de simulación configurable (parámetros variables) con posibilidad de funcionar en tiempo real, de interactuar con hardware y con interfaz gráfica de usuario.

Una plataforma de simulación configurable implica que debe tener flexibilidad para cambiar la rutina de simulación. Algunos módulos de hardware pueden tener entradas y/o salidas analógicas, que interactúan con el simulador.

Además, se requiere el funcionamiento de la plataforma en tiempo real. Este requisito es alcanzable, dependiendo de la complejidad del algoritmo de simulación y del paso de integración (resolución de tiempo) utilizado [22].

Los principales componentes para implementar un simulador HIL son: un procesador y su memoria, encargado de la solución de la ecuación diferencial del modelado de la planta; un conjunto de periféricos que forman la interfaz eléctrica con el controlador; por último, una interfaz de usuario que permite la configuración del simulador y modificación de parámetros en línea, tal como lo muestra la Figura 1.9.

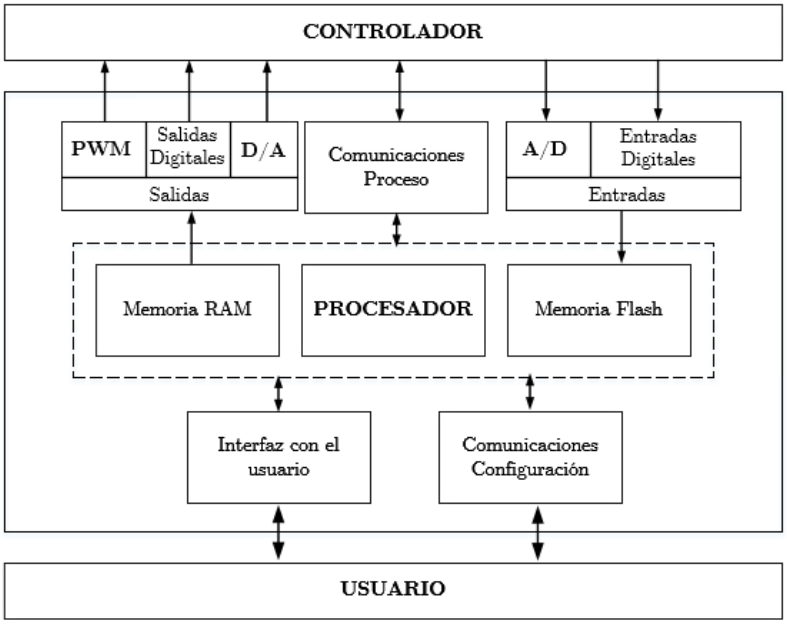

Figura 1.9: Componentes internos de un simulador HIL [10]

1.7. Microcontrolador RM57L843

Un microcontrolador es una computadora en un solo chip. Estos son dispositivos de muy bajo costo y pueden ser usados fácilmente en aplicaciones de control digital. Muchos microcontroladores tienen los recursos necesarios para aplicaciones de control. Por ejemplo, pueden tener convertidores analógico digital (ADC) para que las señales externas puedan ser muestreadas, convertidor digital analógico (DAC) la salida puede ser usada para controlar la planta a través de un actuador. También tienen puertos de entrada y salida en paralelo para que los datos digitales sean leídas o generar una señal desde el microcontrolador [23].

Actualmente la empresa Texas Instruments ha sacado al mercado kits de desarrollo de bajo costo con poderosos microcontroladores con la capacidad de un

procesador digital de señales (DSP), lo que hace factible usarlo para la implementación de sistemas HIL [24].

Hercules Launchpad LAUNCHXL2-RM57L

El kit de desarrollo Hercules™ RM57Lx Launchpad™ de Texas Instruments (ver Figura 1.10) se basa en el microcontrolador RM57L843 de 32 bits el cual tiene un procesador ARM® Cortex®-R5F [12].

El RM57L843 integra dos CPU ARM Cortex-R5F de punto flotante, operando en tiempo real, el cual ofrece una eficiente velocidad de 1.66 DMIPS/MHZ (Dhrystone de millones de instrucciones por segundo por cada MHz) y una frecuencia de operación de hasta 330 MHz [12]. El dispositivo tiene 4MB de flash integrado y 512KB de RAM de datos con un solo bit de corrección de error y doble bit de detección de error (SECDED, single-error correcting, double-error detecting). La memoria flash en este dispositivo es no volátil, eléctricamente borrable y memoria programable. La memoria flash opera con un voltaje de 3.3V.

El modulador de ancho de pulso mejorado (ePWM) puede generar formas de onda de ancho de pulso complejas con una mínima sobrecarga de CPU sin su intervención. El módulo de captura mejorado (eCAP) es esencial en sistemas donde el tiempo de captura de eventos externos es preciso. El eCAP puede ser usado para monitorear la salida ePWM o para simple generación PWM cuando no se necesiten las aplicaciones de captura. El módulo decodificador de cuadratura (eQEP) interactúa directamente con un encoder incremental lineal o rotativo para obtener la posición, dirección, e información de la velocidad, tal como se utiliza en sistemas de movimiento y sistemas de control de posición. El dispositivo tiene dos módulos de conversión analógico-digital de 12 bits, con 41 canales en total. Las principales características son:

- ✓ Diseñado para seguridad crítica industrial y aplicaciones médicas, aplicaciones de automatización y control, control de motores industriales, controladores lógicos programables, generación y distribución de energía.
- ✓ Micropocesador ARM® Cortex-R5F de 32 bits.
- ✓ Frecuencia máxima de operación: 330 MHz.
- ✓ Memoria integrada de 4MB Flash con ECC (código de corrección de errores).
- ✓ 512kB de RAM con ECC.
- ✓ 128kB de flash de datos con ECC para emulación EEPROM.
- ✓ Interrupción en tiempo real (7).

- ✓ Interfaces múltiples de comunicación (Ethernet, CAN, MibSPI, UART).
- ✓ Dos módulos convertidores analógico digital de 12 bits (MibADC).
- ✓ Siete módulos mejorados ePWM, seis módulos de captura (eCAP), dos módulos para el manejo de encoders (eQEP).

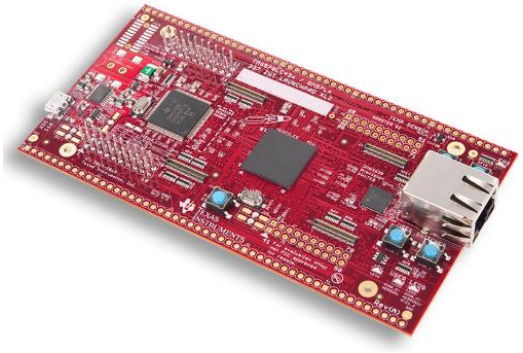

Figura 1.10: Hercules LAUNCHXL2-RM57 [11]

1.8. Interfaz gráfica de usuario

Es necesario desarrollar una interfaz gráfica para el usuario, con el objetivo de proporcionar un entorno visual sencillo, de fácil manejo, para permitir la comunicación con la planta a simular.

El software LabVIEW, de National Instruments, es un entorno de desarrollo integrado que cumple con los requerimientos para implementar una interfaz de usuario, esto en una computadora personal. Unity 3D es una plataforma que permite el diseño de videojuegos en 3D lo que hace una buena opción para diseñar y visualizar los movimientos del sistema carro-péndulo.

1.8.1. LabVIEW

LabVIEW (Laboratory Virtual Instrument Engineering Workbench), es un entorno de programación en el cual se crean programas usando una notación gráfica (Lenguaje G). Es un programa interactivo diseñado específicamente para científicos e ingenieros que desarrollan sistemas de medidas y control. LabVIEW puede incrementar la productividad, programas que toman semanas o meses para escribir usando lenguajes de programación convencional pueden ser completados en horas usando LabVIEW, porque es diseñado específicamente para tomar mediciones, adquisición de datos, analizar datos y presentar resultados al usuario. Como

LabVIEW tiene una interfaz gráfica muy fácil de manejar, se puede crear exactamente el tipo de instrumento virtual que se necesite [25].

En la Figura 1.11, se muestra una interfaz gráfica de usuario desarrollada en LabVIEW 2014, en la parte izquierda se visualizan los controles e indicadores comunes, tales como displays de números y cadenas, botones, barras deslizantes, barras progresivas, pestañas. A la derecha se observan los controles e indicadores específicos de ingeniería, LabVIEW contiene más controles e indicadores que son comunes en aplicaciones científicas y de ingeniería.

Usando los controles incluidos se puede crear VIs con paneles frontales que se asemejan a los instrumentos físicos conocidos, y por lo tanto son más fácil de entender y utilizar por los operadores [26].

LabVIEW es la solución para crear la interfaz gráfica de usuario de la simulación HIL.

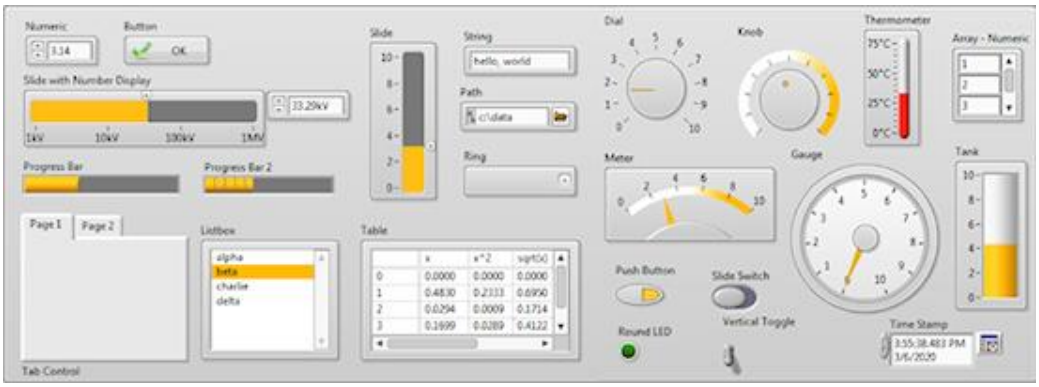

Figura 1.11: Interfaz gráfica de usuario desarrollada en LabVIEW [24]

1.8.2. Unity 3D

Unity 3D es un software para desarrollar videojuegos, permite la creación de juegos en múltiples plataformas como son PlayStation, Xbox, PC, Mac, iOS, Android, etc.

Es una herramienta que permite un increíble realismo con gran facilidad. Aun teniendo una versión gratuita del software, la calidad de los juegos que se pueden desarrollar es excelente, y el uso del programa es sencillo. Estas características convierten a Unity en una muy buena opción para los desarrolladores independientes y los estudios de videojuegos, cualquiera que sea su presupuesto [27].

Fue lanzado oficialmente el 1 de junio de 2005. Este motor permite la creación de juegos, además es posible hacer diseños arquitectónicos. Se pueden crear aplicaciones 3D en tiempo real y multimedia [28].

En la Figura 1.12 se muestra el entorno de trabajo de Unity 3D, en específico se muestran las herramientas para crear una interfaz de usuario.

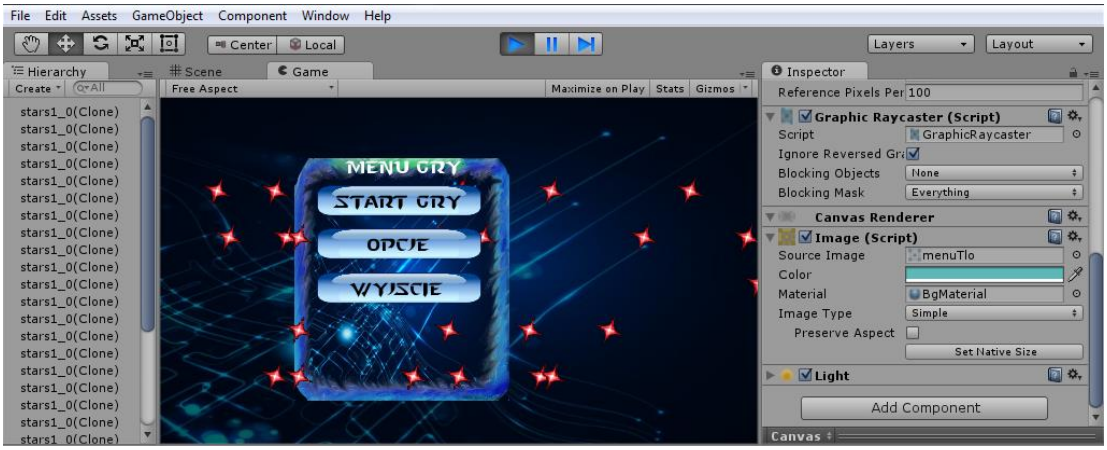

Figura 1.12: Interfaz de usuario diseñada en Unity 3D [29]

1.9. Acondicionamiento de señales.

Muchas aplicaciones implican medidas ambientales o estructurales mediante sensores, como es el caso de la temperatura y las vibraciones. Estos sensores, a su vez, requieren el acondicionamiento de las señales antes de que un dispositivo de adquisición de datos pueda medir con eficacia y precisión la señal. El acondicionamiento de señal es uno de los componentes más importantes de un sistema de adquisición de datos; ya que sin la optimización de las señales del mundo real para el digitalizador que se esté utilizando, no se puede confiar en la exactitud de la medida.

Las necesidades de acondicionamiento de las señales varían ampliamente dependiendo de la funcionalidad del sensor, por lo que ningún instrumento puede proporcionar todo tipo de acondicionamiento para todos los sensores [30].

En la simulación HIL, es necesario acondicionar la señal de la salida del controlador a la entrada del simulador de la planta (y viceversa), con el fin de hacer compatibles los niveles de voltaje. En la Figura 1.13 se muestra el bloque de acondicionamiento de señales como parte del sistema de adquisición. El controlador

envía la señal de control a la simulación de la planta y la planta envía una señal de uno o más sensores simulados, en ambos casos estas señales pasan por el acondicionamiento de señales.

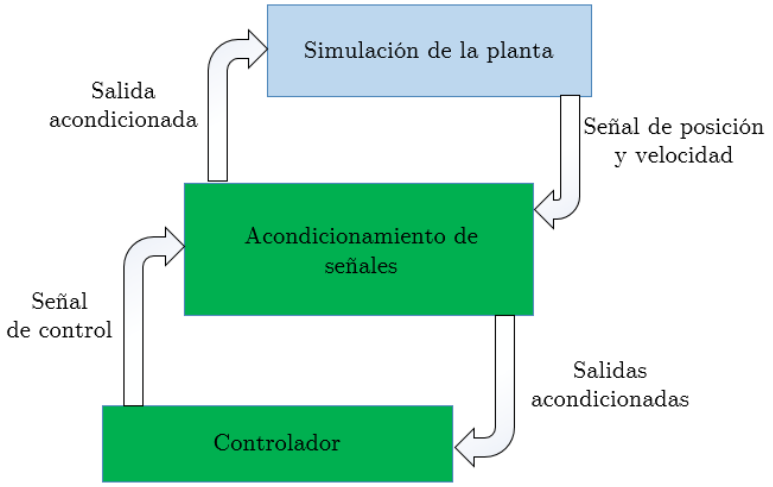

Figura 1.13: Esquema de acondicionamiento de señales

A continuación, se enumeran los bloques más comunes que tiene un sistema de acondicionamiento.

Amplificación

Los amplificadores incrementan el nivel de voltaje para lograr una mejor adaptación al rango del convertidor analógico-digital (ADC), aumentando así la resolución de la medida y la sensibilidad. Además, la localización de los acondicionadores de señal externos más cerca de la fuente de la señal o del transductor, mejora la relación de la señal con respecto al ruido de la medida, mediante el incremento del nivel de voltaje antes de que se vea afectada por el ruido ambiental.

Atenuación

La atenuación, que es lo contrario que la amplificación, se necesita cuando las tensiones que se van a digitalizar están fuera del rango del ADC. Esta forma de acondicionamiento de la señal disminuye la amplitud de la señal de entrada de modo que la señal acondicionada quede dentro del rango de voltaje del ADC. La atenuación es típicamente necesaria cuando se miden voltajes de más de 10 V.

Filtrado

Los filtros rechazan el ruido no deseado dentro de un determinado rango de frecuencias. A menudo, los filtros paso-bajo se utilizan para bloquear el ruido de las medidas eléctricas, tales como el procedente de los 50/60 Hz de la red eléctrica. Otro uso común del filtrado es evitar el "aliasing" de las señales de alta frecuencia. Esto se puede hacer mediante el uso de un filtro "anti-aliasing" que atenúan las señales por encima de la frecuencia de Nyquist.

Aislamiento

Señales de voltaje que están bastante fuera del rango del digitalizador pueden dañar al sistema de medida y al operador. Por esa razón, se requiere generalmente el aislamiento junto con la atenuación para proteger al sistema y al usuario de los voltajes peligrosos o de los picos de voltaje. El aislamiento también puede ser necesario cuando el sensor está en un plano de tierra diferente del sensor de medida, tal como ocurre con un termopar montado en un motor.

Linealización

La linealización es necesaria cuando los sensores producen señales de voltaje que no están linealmente relacionados con las medidas físicas. La linealización consiste en el proceso de interpretación de la señal del sensor, se puede implementar mediante el acondicionamiento de la señal o por medio de software.

Capítulo 2
Diseño del Hardware

En este capítulo se muestra el procedimiento para simular el sistema carro-péndulo (CPI) en la tarjeta de desarrollo Hercules Launchpad RM57L. Se menciona como se vincula el software HAL Code Generator y Code Composer Studio para utilizar la tarjeta. Se describe las características principales y configuración de cada módulo utilizado de la tarjeta.

2.1. Diagrama general de bloques.

En la

Figura 2.1 se muestra el diagrama a bloques que conforma cada etapa del simulador HIL. Los elementos que están fuera del recuadro de "Hercules RM57Lx" indican las etapas de entradas/salidas externas al microcontrolador. El resto lo conforman módulos o bloques internos del microcontrolador.

El módulo ADC digitaliza la señal analógica proveniente de un controlador externo, con base a esta entrada se resuelve el modelo dinámico del CPI basándose en el método numérico de Euler, al resolver las ecuaciones se obtienen la posición del carro y la posición del péndulo, las que son enviadas por puerto serie a una interfaz gráfica donde se visualizan los movimientos del CPI. Adicionalmente, estas posiciones se escalan de forma que generen dos señales PWM las cuales son las variables de entrada del controlador.

El bloque de acondicionamiento de señales puede o no utilizarse, su uso depende del voltaje de operación de donde se procese el algoritmo de control.

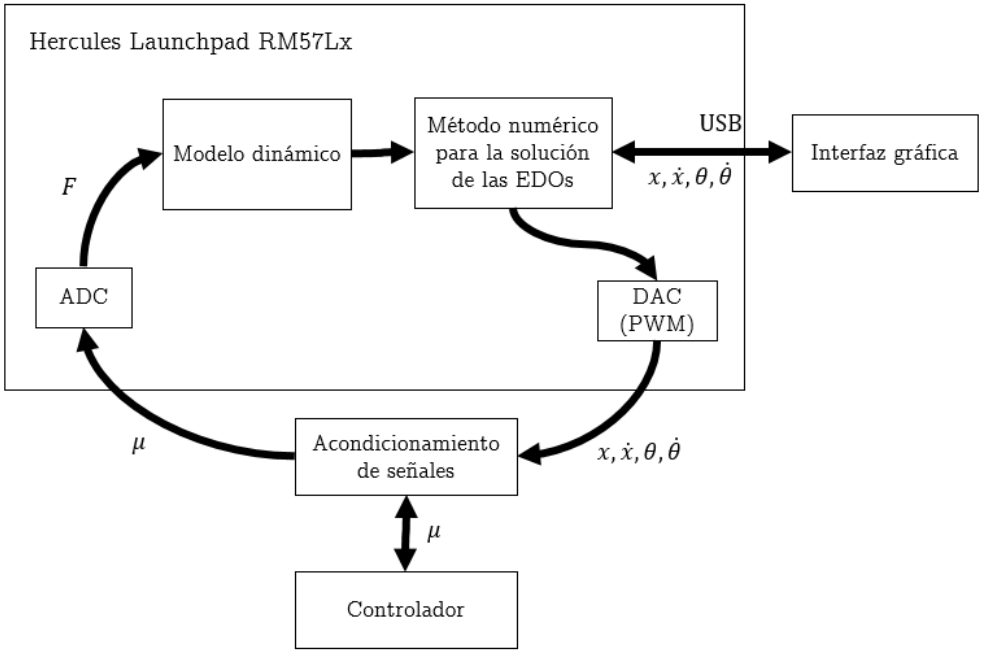

Figura 2.1: Diagrama general del simulador HIL

2.2. Configuración del software CCS y HalcoGEN

El Hercules Launchpad RM57Lx se programa con el ambiente de programación Code Composer Studio (CCS) que proporciona el fabricante Texas Instruments, que se utiliza para programar a todos sus productos. Para el caso del Hercules se requiere instalar un complemento llamado HAL Code Generator en donde se configuran los pines y módulos del MCU. A continuación, se describen los pasos para la creación de proyectos para este kit de desarrollo, los cuales son:

- ✓ Instalar el Software CCS y HAL Code Generator [31].
- ✓ Crear un proyecto en CCS, guardar el proyecto.
- ✓ Crear un proyecto en HAL Code Generator guardando el proyecto en la dirección anterior.
- ✓ Configurar los módulos en HALCoGen requeridos, tales como GIO, PWM, ADC, SCI, HET.
    - o Dar clic en generar el código (F5), entonces se genera el código automáticamente para el microcontrolador seleccionado.
- ✓ Volver a CCS e ir a la siguiente ruta:
    - o Project-Properties-Build- ARM Compiler-Include Options.

- Dar clic en la sección ¨Add directory path¨ y añadir la siguiente dirección:
  - "${workspace_loc:/${ProjName}/NombreDeTuProyecto/include}"
  - Esto cargara los módulos configurados en la etapa anterior.

✓ En CCS buscar la carpeta con el nombre del proyecto hecho en HAL luego ir a la carpeta "source" entonces abrir el archivo "HL_sys_main.c" ese será la función principal (main) del proyecto.

Nota: cada línea de código agregado en CCS es necesario colocarlos entre los comentarios marcados como:

✓ /* USER CODE BEGIN (X) */

✓ /* USER CODE END *

2.3. Hardware Abstraction Layer Code Generator

HALCoGen es un generador de código de inicialización, configuración y controlador, basado en una interfaz gráfica de usuario (GUI) para las tarjetas de desarrollo (Hercules) de Texas Instruments. Permite configurar los periféricos internos, generar y crear-depurar código fuente a través de CCS en la misma carpeta de trabajo [31].

En la Figura 2.2 se muestran todos los bloques del microcontrolador RM57L843. Los bloques de color azul pueden ser configurados por el usuario, los de color gris son reservados.

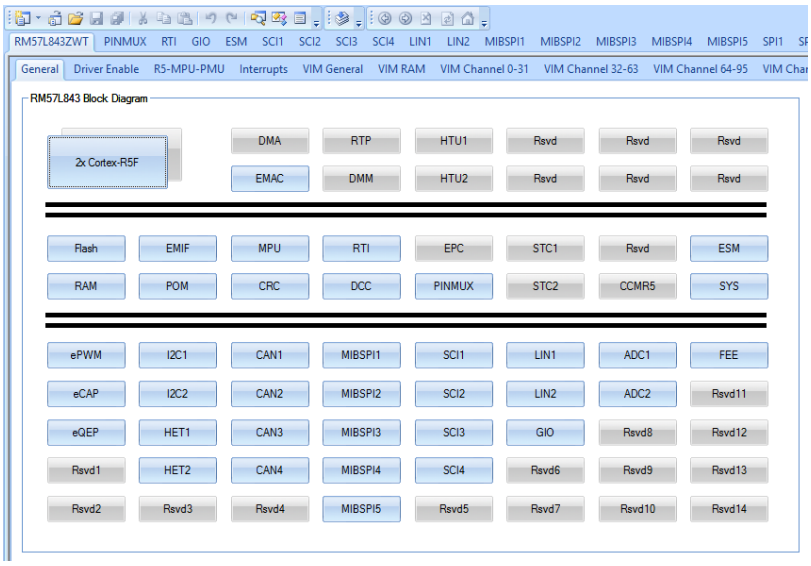

Figura 2.2: Diferentes bloques configurables del MCU RM57L843

2.4. Modulo Convertidor Analógico Digital

Un convertidor analógico digital (ADC) es un circuito o dispositivo electrónico que convierte una señal analógica en una señal digital. Para un simulador HIL es necesario tomar la señal analógica del controlador convertirla a digital para enseguida con éste solucionar el modelo dinámico.

Algunas características del módulo ADC son:

- ✓ Resolución seleccionable entre 12, 10 y 8 bits.
- ✓ Cuenta con dos módulos independientes ADC1, soporta 32 canales; ADC2 soporta 35 canales, de los cuales 16 son compartidas con ADC1 en total suman 41 canales.
- ✓ Pines de voltaje de referencia (AD_{REFHI} y AD_{REFLO}).
- ✓ Tiempo total de muestreo/retención/conversión de 600ns mínimo.

En la Figura 2.3 se presenta el diagrama de bloques del módulo ADC. El multiplexor conecta al canal de entrada seleccionado a la entrada AIN del núcleo del ADC. El secuenciador selecciona el canal que se va a digitalizar, la conversión se basa en el registro de aproximación sucesiva (SAR), esta selección se aplica a todas las conversiones realizadas por el módulo ADC. No es posible convertir algunos canales con una resolución de 12 bits y algunos con una resolución de 10 bits. El rango de voltaje de entrada del módulo ADC es de 0 a 3.3 volts [32].

Una sola conversión de una entrada analógica a una conversión digital ocurre en dos etapas distintas, a continuación se describen:

Periodo de muestreo:

- ✓ El secuenciador genera una señal START al núcleo ADC para señalar el inicio del período de muestreo.
- ✓ La señal de entrada analógica es muestreada directamente en la matriz de condensadores conmutados durante este período.
- ✓ El período de muestreo finaliza un ADCLK (ciclo de reloj para la conversión ADC) completo después del flanco descendente de la señal START.
- ✓ El secuenciador puede controlar la duración del período de muestreo configurando el registro de control de tiempo de muestreo del grupo de conversión (ADEVSAMP, ADG1SAMP, ADG2SAMP). Este registro controla el tiempo durante el cual la señal START permanece en alto.

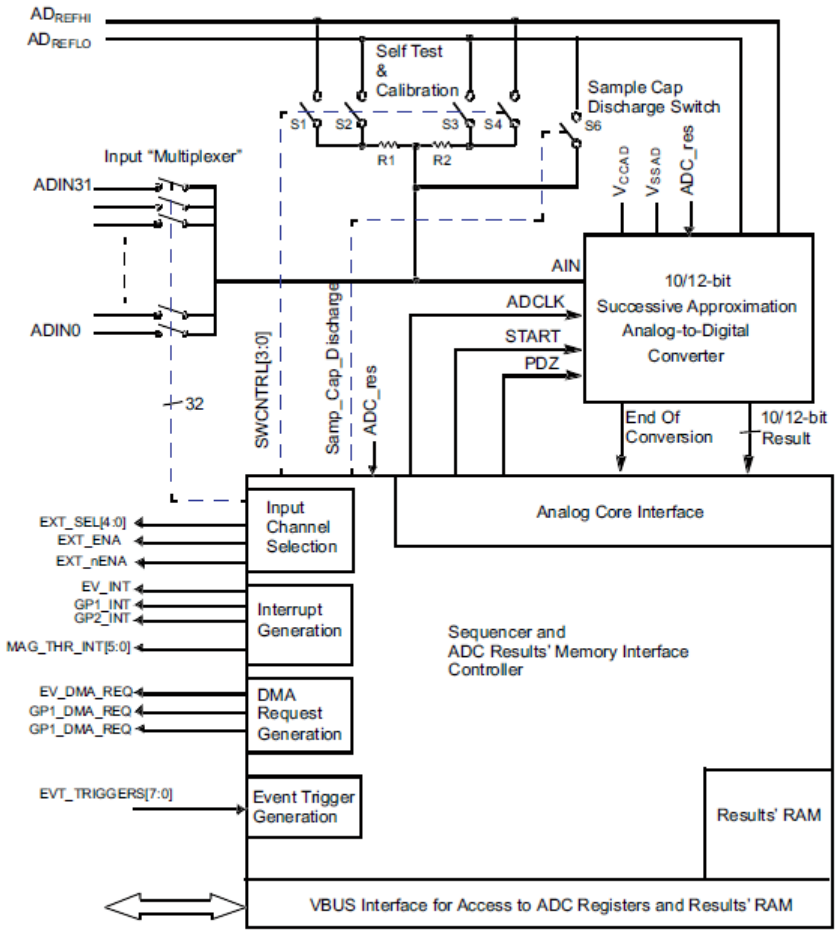

Figura 2.3: Componentes del módulo ADC [28]

Período de conversión:

- ✓ El período de conversión comienza con un ciclo ADCLK completo después del flanco descendente de START.
- ✓ Un bit del resultado de conversión se emite en cada flanco ascendente del ADCLK en el período de conversión, comenzando por el bit más significativo.
- ✓ El periodo de conversión es de 12 ciclos ADCLK en caso de un ADC de 12 bits.
- ✓ El núcleo ADC genera una señal de fin de conversión al secuenciador al final del período de conversión. En ese momento el resultado de conversión está disponible.

El rango de conversión analógica es determinado por los voltajes de referencia: ADREFHI y ADREFLO. ADREFHI es el voltaje de referencia superior y es el voltaje analógico máximo que se puede convertir. ADREFLO es el voltaje de referencia inferior y es el voltaje analógico mínimo que se puede convertir. Tanto en AD_{REFHI}

como en AD_{REFLO} deben elegirse de manera que no excedan las fuentes de alimentación analógicas V_{CCAD} y V_{SSAD}, respectivamente. Los valores de referencia por default son, para AD_{REFHI} 3.3 volts, para AD_{REFLO} 0 volts.

Los voltajes de entrada entre AD_{REFHI} y AD_{REFLO} producen un resultado de conversión dado por la ecuación (2.1) para una resolución de 12 bits.

$$ValorDigital = \frac{4096 * (\text{VoltajeEntrada} - AD_{REFLO})}{AD_{REFHI} - AD_{REFLO}} - 0.5 \qquad (2.1)$$

Configuración del módulo ADC

En la Tabla 4, se describen los pasos para configurar el módulo ADC, los parámetros en el software HALCoGen y las líneas de código para inicializar los módulos en CCS.

Tabla 4: Configuración del módulo ADC1 del MCU RM57L843

Descripción	HALCoGen	CCS
En la pestaña Driver Enable se activan los módulos a utilizar.	RM57L843ZWT→ Driver Enable→ Activar Enable ADC1 driver	*include "HL_adc.h" adcInit();*
Fifo size es el tamaño del buffer para almacenar las conversiones de cada canal, se ingresa 2 para leer dos señales analógicas.	ADC1→ ADC1 Group1→ Fifo Size: Ingresar 2	
Se selecciona la resolución en bits del módulo.	ADC1→ ADC1 Group1→ Data Resolution (Bit) → 12_BIT	
Se selecciona el canal (pin) donde el módulo tomará la señal analógica.	ADC1→ ADC1 Group1→ ADC1 Group 1 Channel Selection: Enable pin 18, Enable pin 17	
Se configura el periodo de conversión (ADCLK).	ADC1 General-Cycle time (ns): 100	

Con un reloj (ADCLK) configurado en 100ns, el tiempo de conversión total de un valor analógico a un valor binario es de 1.600340 μs, el tiempo aumenta a medida

que se utilicen más canales del convertidor. En la Figura 2.4 se puede observar algunos campos de configuración del módulo ADC1.

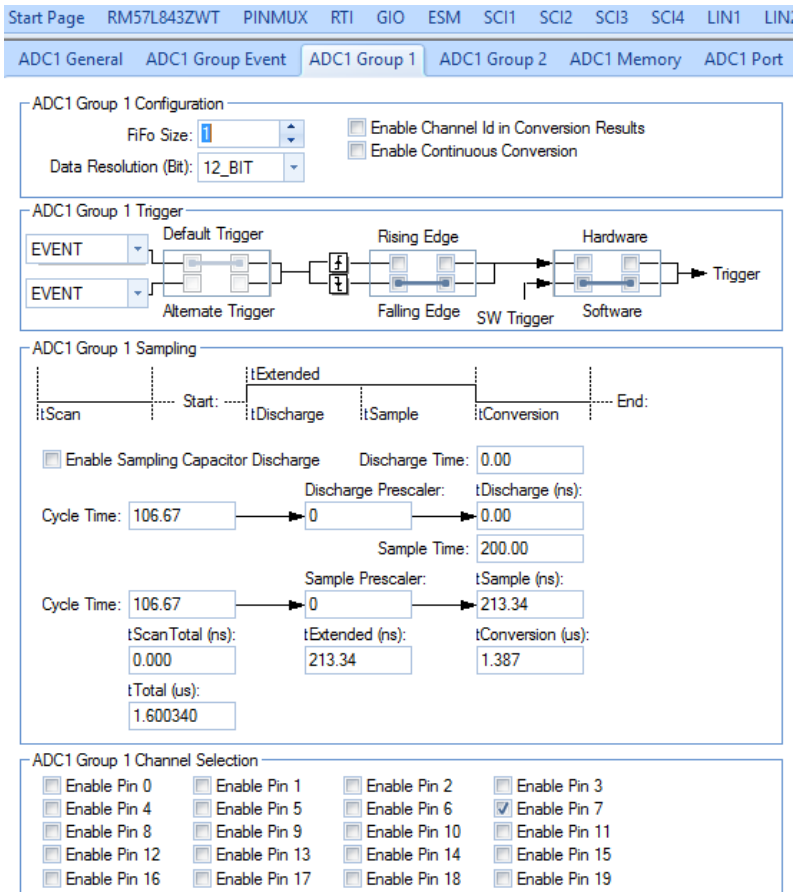

Figura 2.4: Bloques de configuración del módulo ADC1

2.5. Módulo N2HET y GPIO

N2HET (High-End Timer) es un temporizador inteligente avanzado que proporciona sofisticadas funciones de temporización para aplicaciones en tiempo real. El temporizador está controlado por software, se puede utilizar para generar salidas PWM, capturar/comparar entradas o configurar los pines GPIO. El N2HET es especialmente adecuado para aplicaciones que requieren información de múltiples sensores o controlar actuadores con tiempos complejos y precisos [32].

El módulo GPIO tiene el propósito de funcionar como entrada y salida digital, está compuesto por un registro de 8 bits, puede configurarse hasta 145 pines para uso general. Tiene 16 pines con capacidad de interrupción externa.

2.6. Módulo mejorado ePWM

Puesto que el MCU no tiene módulo Convertidor Digital-Analógico (DAC), es necesario emular una señal de salida analógica usando la técnica de modulación de ancho de pulsos (PWM) y un filtro pasa bajas. Para esta etapa se usa uno de los 7 módulos ePWM del MCU RM57L843, se le llama mejorado porque genera formas de onda de ancho de pulso complejas con una intervención mínima de CPU, además es muy flexible, fácil de entender y usar.

El módulo ePWM representa un canal PWM completo compuesto de dos salidas PWM: EPWMxA y EPWMxB. Múltiples módulos ePWM se instancian dentro de un dispositivo como se muestra en la Figura 2.5. Cada instancia de ePWM es idéntica y está indicada por un valor numérico que comienza con 1. Por ejemplo, ePWM1 es la primera instancia y ePWM3 es la tercera instancia en el sistema y ePWMx indica cualquier instancia.

Los módulos ePWM están anidados a través de un esquema de sincronización de reloj que les permite funcionar como un único sistema cuando sea necesario. Los módulos también pueden funcionar independientemente.

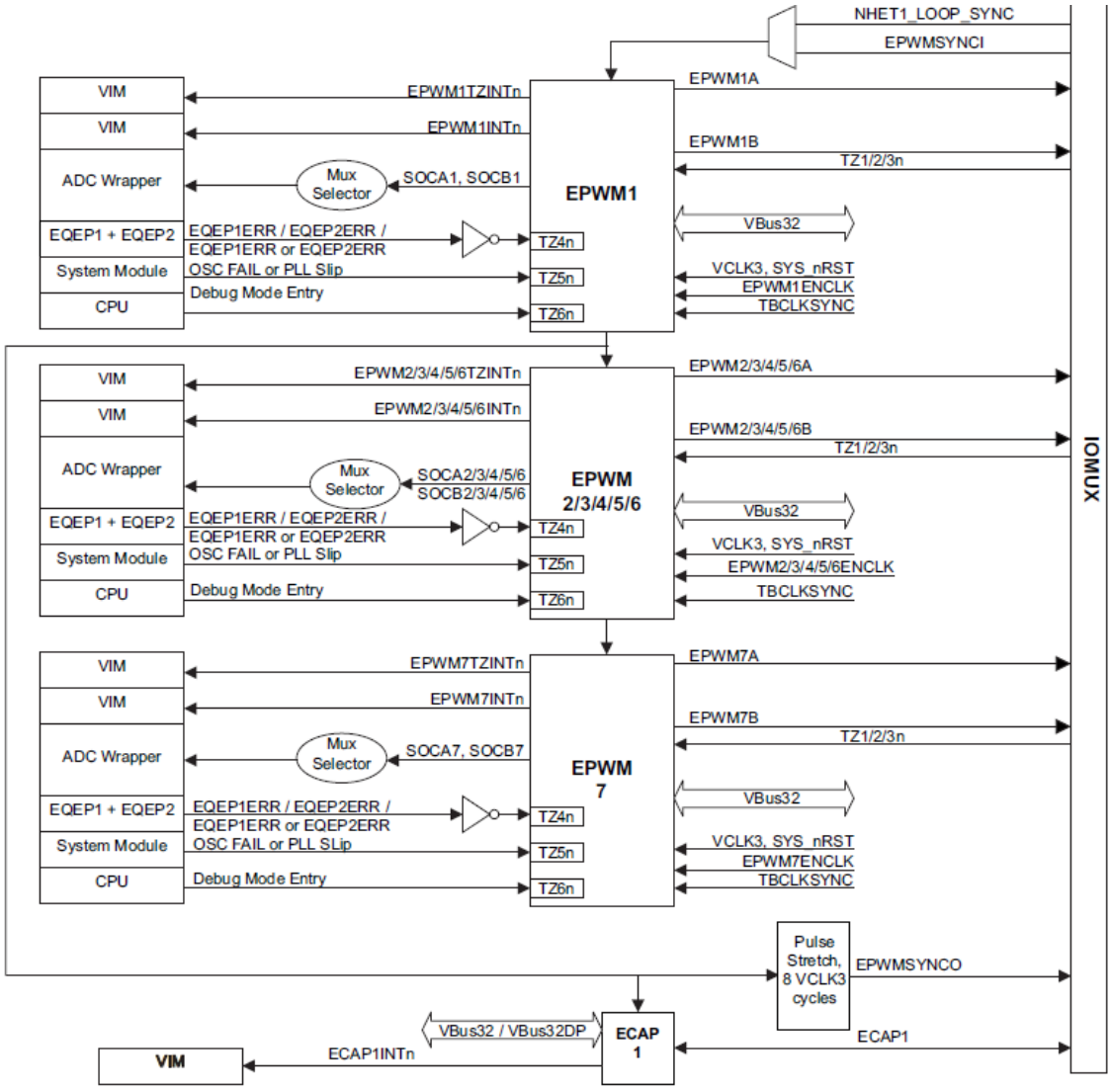

Figura 2.5: Múltiples módulos PWM [28]

Configuración del generador de señal PWM

La Tabla 5, muestra la descripción de la configuración del módulo PWM en HALCoGen y CCS. Se ejemplifica solo un canal (PWM0) pero de la misma forma se configuran los demás canales.

Tabla 5: Configuración del canal PWM0 del MCU

Descripción	HALCoGen	CCS
Se activa uno de los dos módulos HET.	RM57L843ZWT→ Driver Enable→ Activar Enable HET1 driver.	
Se habilita uno de los 7 canales PWM del módulo HET1. Se configura el ciclo de trabajo, el periodo de la señal y se selecciona el pin de salida.	HET1→ Pwm 0-7→PWM 0: ✓ Activar Enable ✓ Duty [%]: 10 ✓ Period [us]: 33 ✓ Pin: 2	*#include "HL_het.h"* *hetInit();*
Se habilita el bit 2 como pin de salida	HET1 - Pin 0-7-Bit 2: -Activa la casilla DIR	

También puede configurarse la polaridad de salida de la señal PWM, polaridad alta o baja, como lo muestra la Figura 2.6.

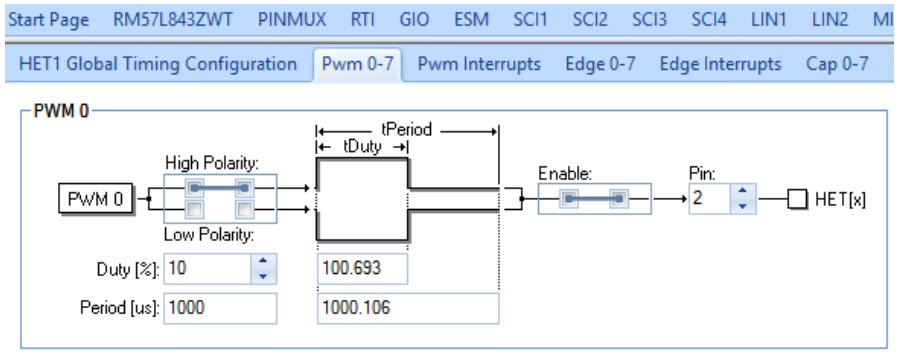

Figura 2.6: Bloques de configuración del canal PWM0

2.7. Módulo de comunicación SCI

El módulo SCI (Interfaz de comunicación serial) es un transmisor-receptor asíncrono universal, puede utilizarse para comunicarse a través del puerto RS-232. El puerto serial envía y recibe bytes de información un bit a la vez, usualmente, la comunicación serial se utiliza para transmitir datos en formato ASCII. Los parámetros más importantes de la comunicación serial son la velocidad de transmisión, bits de datos, bits de parada y la paridad.

Se mencionan algunas características del módulo SCI [32].

- ✓ Comunicación estándar de transmisor-receptor asíncrono universal (UART).

- ✓ Soporta operaciones full o half-duplex.
- ✓ Funciones de recepción y transmisión de doble buffer en modo de compatibilidad.
- ✓ Soporta dos líneas de interrupción habilitadas individualmente: nivel 0 y nivel 1.
- ✓ Modos de comunicación asíncronos o síncronos.
- ✓ Dos formatos de comunicación multiprocesador permiten la comunicación entre dos o más dispositivos.
- ✓ La velocidad de transmisión programable de 24 bits soporta 2^{24} velocidades de baudios diferentes, que proporcionan una velocidad de transmisión de alta precisión.

En la Figura 2.7, se muestra el diagrama de bloques detallado del módulo SCI. Los tres componentes principales son el transmisor, generador de reloj de baudios (Baud clock) y el receptor.

El transmisor (TX):

Contiene dos registros principales para ejecutar el doble buffer:

- ✓ El registro del buffer de datos del transmisor (SCITD) contiene datos cargados por la CPU para ser transferidos al registro de desplazamiento para la transmisión.
- ✓ El registro de desplazamiento del transmisor (SCITXSHF) carga datos desde el buffer de datos (SCITD) y desplaza los datos al pin LINTX, un bit a la vez.

Baud reloj generador: Un generador de baudios programable genera un reloj de baudios escalado de VBUSP CLK.

El receptor (RX):

- ✓ Contiene dos registros principales para ejecutar el doble buffer.
- ✓ El registro de desplazamiento del receptor (SCIRXSHF) desplaza los datos del pin LINRX un bit a la vez y transfiere los datos completados al buffer de datos de recepción.
- ✓ El registro del buffer de datos del receptor (SCIRD) contiene los datos recibidos transferidos desde el registro de desplazamiento del receptor.

El receptor SCI y el transmisor son de doble búfer, y cada uno tienen sus propios bits de habilitación y de interrupción. El receptor y el transmisor pueden ser operados independientemente o simultáneamente en modo full-dúplex.

Para garantizar la integridad de los datos, el SCI comprueba los datos que recibe para los errores de pausa, paridad, desbordamiento y enmarcado. La tasa de bits (baud) es programable a más de 16 millones de velocidades diferentes a través de un registro de selección de baudios de 24 bits.

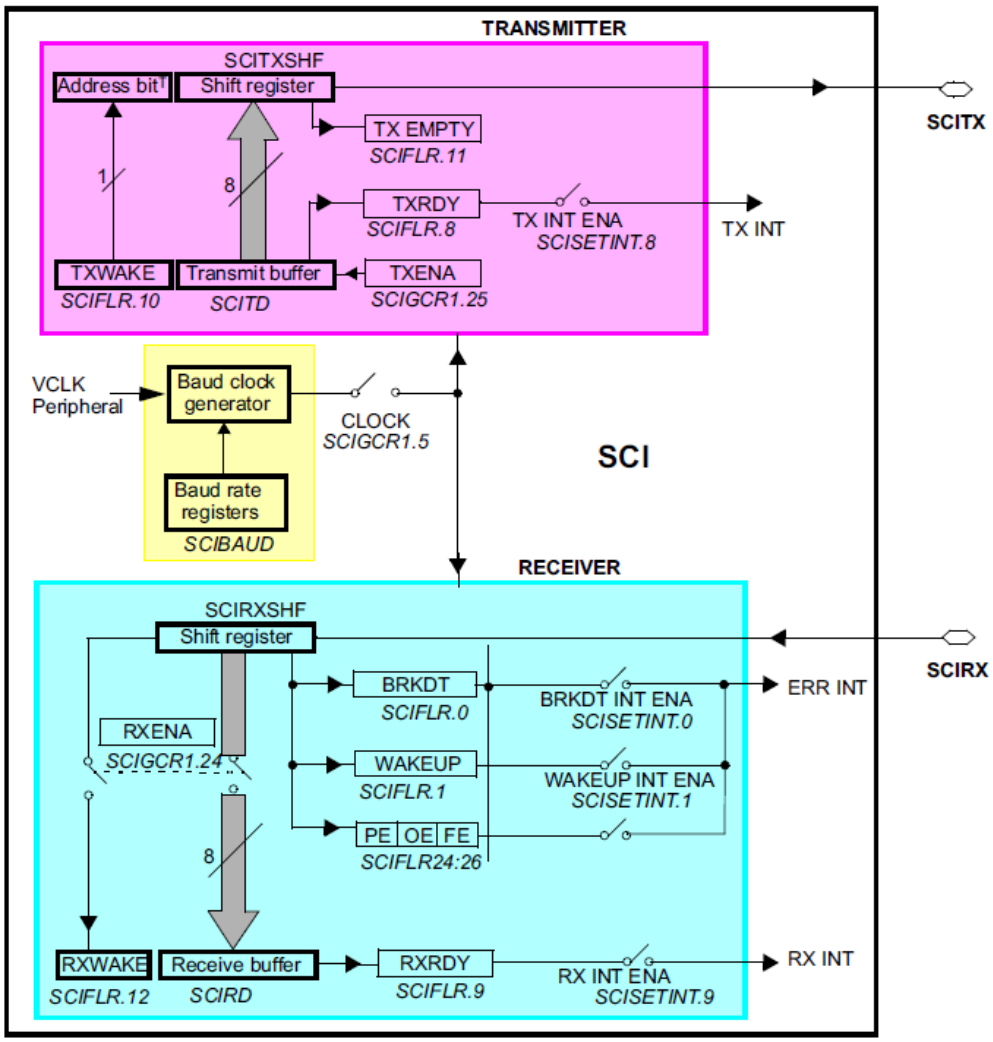

Figura 2.7: Diagrama de bloques del módulo SCI [28]

Configuración del SCI

En la siguiente tabla se muestra los parámetros de configuración de uno de los cuatro canales SCI. Se puede habilitar interrupciones por transmisión o recepción de datos.

Tabla 6: Configuración del módulo SCI1

Descripción	HALCoGen	CCS
Se activa uno de los cuatro módulos SCI.	RM57L843ZWT→ Driver Enable→ Activar Enable SC1 driver.	*#include "HL_sci.h"*
Se configuran los parámetros de comunicación.	SCI1→ SCI Data Format: ✓ Baudrate:56000 ✓ Stop Bits: 2 ✓ Length: 8 ✓ Parity Enable: desactivado	*sciInit();*

En la Figura 2.8 se ven los campos de configuración del puerto serial en el software HALCoGen. En Baudrate se configura la velocidad de transmisión, el software automáticamente calcula la frecuencia del reloj (VCLK1) y el valor del prescaler, que usara para aproximar la frecuencia deseada.

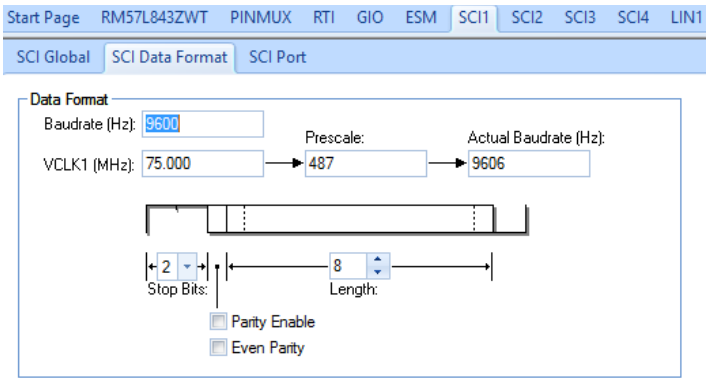

Figura 2.8: Bloques de configuración del canal SCI1

2.8. Módulo de interrupción en tiempo real

El módulo de interrupción en tiempo real (RTI) proporciona la funcionalidad de temporizador para los sistemas operativos. El módulo RTI puede incorporar varios contadores que definen las bases de tiempo necesarias para la programación en el sistema operativo [32].

El módulo RTI tiene las siguientes características:

- ✓ Dos bloques de contador independiente de 64 bits.
- ✓ Cuatro comparadores configurables para generar ticks del sistema operativo. Cada evento puede ser accionado por el bloque de contador 0 o el bloque de contador 1.
- ✓ Activación/desactivación rápida de eventos.
- ✓ Dos funciones de registro temporal (captura) para las interrupciones del sistema o periféricos, una para cada bloque de contador.

En la Figura 2.9 se ilustra el diagrama de bloques de alto nivel del módulo RTI. El módulo RTI tiene dos bloques contadores independientes para generar diferentes bases de tiempo: bloque contador 0 y bloque contador 1.

Una unidad de comparación coteja los contadores con valores programables y genera cuatro interrupciones independientes o solicitudes DMA. Cada uno de los registros de comparación puede ser programado para ser comparado con el bloque contador 0 o con el bloque contador 1.

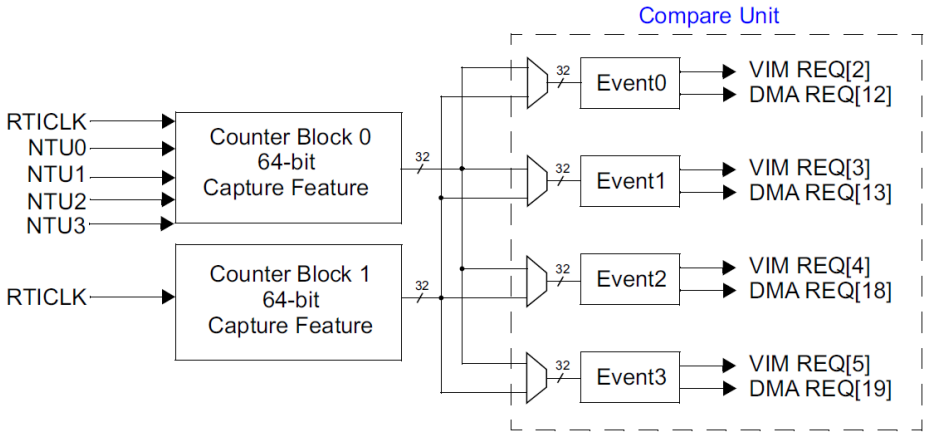

Figura 2.9: Diagrama de bloques del módulo RTI [28]

Configuración del módulo RTI

En la Tabla 7, se presenta la configuración del comparador 0 con el objetivo de producir una interrupción cada 1 ms, este tiempo servirá de base para la solución del modelo dinámico del carro-péndulo.

Tabla 7: Configuración de la interrupción (1ms)

Descripción	HALCoGen	CCS
Se activa uno de los dos módulos RTI.	RM57L843ZWT→ Driver Enable→ Activar Enable RTI driver.	*#include "HL_rti.h"* *rtiInit();* *_enable_IRQ();*
Se configura el periodo de interrupción en milisegundos.	RTI→ RTI1 Compare→ Compare 0: Period:1	

La Figura 2.10 ilustra los campos a configurar del Comparador 0, el campo más importante es la del periodo (Period) donde se ingresa el tiempo (en milisegundos) que transcurrirá para que el contador desborde y se produzca una interrupción.

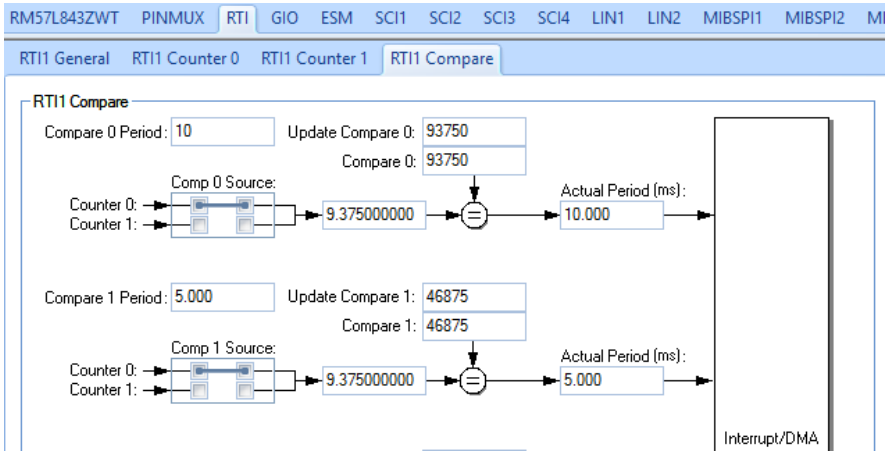

Figura 2.10: Campos de configuración del comparador RTI.

2.9. Acondicionamiento de señales (HW)

Al conectar el simulador HIL al controlador es necesario acondicionar (filtrar) la señal, esto debido que se simula una salida analógica a través de un modulador de ancho de pulso (PWM), y en efecto, se debe mejorar la respuesta de salida.

En la Figura 2.11 se muestra el diagrama a bloques del flujo de datos entre el DAQ, filtro RC y el simulador. El simulador HIL tiene como salidas PWM la posición del carro y péndulo, estas señales entran al filtro RC de forma que a la salida se

obtengan las señales con la frecuencia PWM filtrada y mostrando solo el voltaje continuo, estas ingresan a la tarjeta de adquisición de datos (DAQ) para posteriormente enviarlas por puerto serial al controlador diseñado en MATLAB/Simulink.

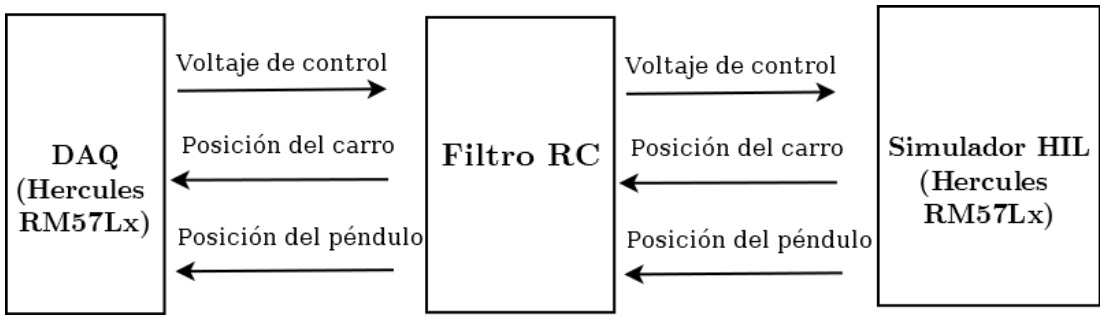

Figura 2.11: Diagrama a bloques de la comunicación del CAS-HIL.

Se diseña un filtro pasa bajas RC con una frecuencia de corte de 1.06KHz como lo muestra la Figura 2.12, donde R es una resistencia con valor de 1.5 kΩ y C es un capacitor cerámico de 0.1 microfaradios. A la entrada del filtro se le conecta el pin de salida del PWM de la tarjeta de adquisición de datos, esta señal PWM tiene una frecuencia de 9Khz.

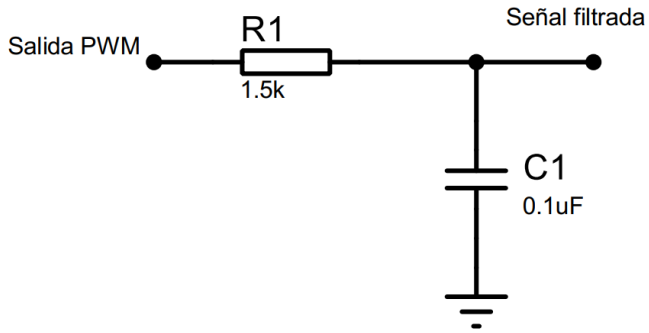

Figura 2.12: Filtro RC

Capítulo 3
Diseño del Software

En este capítulo se describe el proceso de diseño del software del Emulador del Sistema Carro Péndulo Invertido (ESCPI). Se presenta la forma de programación de la tarjeta Hercules RM57Lx, el diseño de la interfaz gráfica hecho en LabVIEW y en Unity 3D. Finalmente se muestran los diferentes diagramas de flujo de los módulos utilizados.

3.1. Modelo de desarrollo

La implementación del ESCPI se basa en el modelo de desarrollo incremental, si bien este método es comúnmente aplicado en ingeniería de software, es útil cuando el programa se debe interconectar con otros elementos como hardware y usuarios. En la Figura 3.1 se observa el proceso de desarrollo del modelo incremental aplicado al ESCPI. Consta de los siguientes: requerimientos iniciales, análisis, diseño, código y pruebas. En las secciones siguientes se analiza cada una de las etapas.

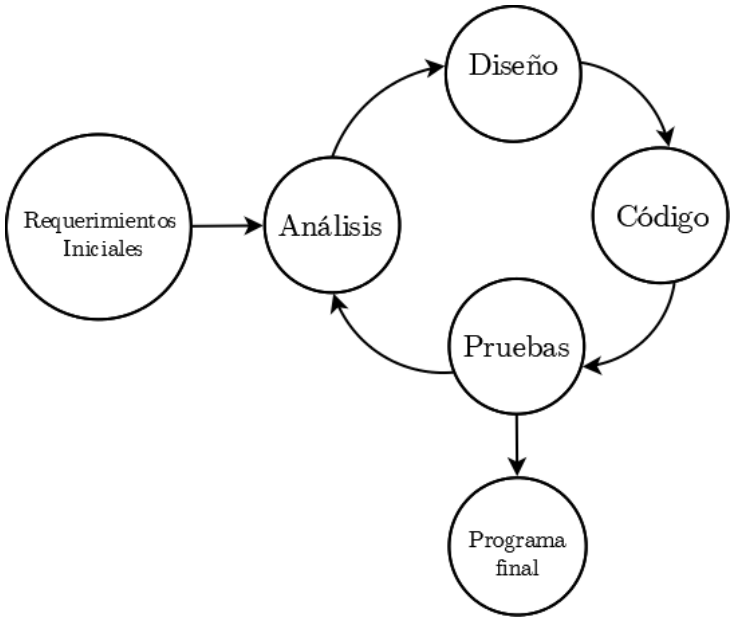

Figura 3.1: Fases del modelo de desarrollo incremental.

3.2. Requerimientos del ESCPI

La función principal del ESCPI es simular el comportamiento de un carro-péndulo invertido con base en un MCU y mostrar su movimiento en un entorno virtual. Para cumplir con esto el ESCPI se divide en 2 programas que son el Software embebido en el MCU y la Interfaz gráfica en Unity. Los requerimientos para llevar a cabo estas funciones son:

Software embebido en el MCU:

- ✓ Leer el voltaje de control mediante el módulo ADC, filtrar y escalar la señal.
- ✓ Resolver el modelo dinámico para calcular las variables de estado (posición del carro y posición del péndulo).
- ✓ Escalar las posiciones, la señal de control y enviarlos por puerto serie.
- ✓ Recibir comandos (letras) para modificar los parámetros del modelo dinámico del ESCPI o bien para cambiar la posición del carro o péndulo.
- ✓ Simular salidas analógicas (posición del carro y péndulo) mediante el módulo PWM.

Interfaz gráfica en Unity:

- ✓ Recibir por puerto serial la posición del carro y péndulo y mostrar los movimientos del carro-péndulo en un ambiente virtual en 3D.

- ✓ Mostrar y graficar la posición del carro, posición del péndulo y la señal de control.
- ✓ Mostrar un menú de opciones donde se pueda cambiar los parámetros del modelo dinámico del ESCPI.
- ✓ Proporcionar al usuario la opción de enviar perturbaciones (mover el carro o péndulo).

3.3. Análisis de software.

En esta fase se analizaron cada uno de los requerimientos con el objetivo de crear una serie de tareas que se interconectan para hacer funcionar de forma adecuada el ESCPI.

3.3.1. Software embebido en el MCU

En la Figura 3.2 se observan las tareas principales que debe realizar el programa que se ejecutara en el microcontrolador. El programa se divide en los siguientes bloques: inicialización, lectura de la señal analógica, modelo dinámico, generar salidas analógicas y comunicación serial.

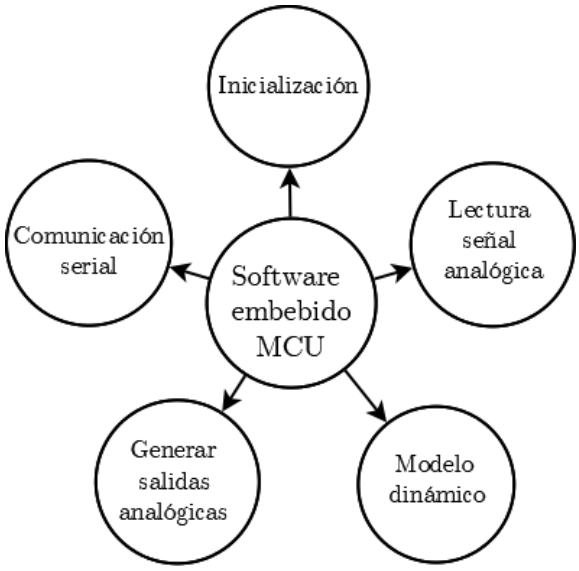

Figura 3.2: Tareas principales del ESCPI

Inicialización. Este bloque tiene como función inicializar todos los módulos que ocupara el programa, como lo son el puerto serial, el módulo ADC, las

interrupciones, el módulo PWM, además de configurar los parámetros del modelo dinámico del CPI (Carro Péndulo Invertido).

Lectura de una señal analógica. Este bloque tiene como tarea, leer continuamente una señal analógica (señal de control) a través del módulo ADC, guardarla en una variable para posteriormente sustituirla en la ecuación del modelo dinámico.

Modelo dinámico. Se generará una interrupción, donde se resolverá el modelo dinámico del CPI con base al método numérico de Euler, obteniéndose la posición del carro y la posición del péndulo.

Generar salidas analógicas. Se usará el módulo ePWM para simular salidas analógicas, la posición del carro (proporcional a metros) y la posición del péndulo (proporcional en grados).

Comunicación serial. En este bloque se recibirá un carácter que proviene de la interfaz gráfica y dado el carácter, el programa tendrá diferentes formas de comportamiento. Se transmitirá por el puerto serial las posiciones del carro-péndulo y la señal de control.

3.3.2. Interfaz gráfica en Unity

El objetivo de la interfaz gráfica es mostrar los movimientos del CPI en un ambiente virtual. En la Figura 3.3, se observan las tareas principales que debe realizar el programa.

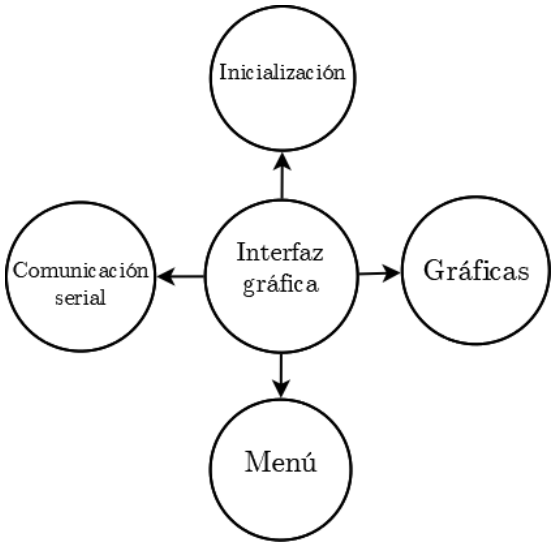

Figura 3.3: Tareas principales de la interfaz gráfica.

Inicialización. En este bloque, se configura el puerto serie a utilizar.

Gráficas. Se recibirá por el puerto serial las posiciones del CPI y se graficaran en tiempo real, mostrándole al usuario tres graficas: posición del carro, posición del péndulo y la entrada de control.

Menú. En la interfaz gráfica, el usuario tendrá acceso a un menú donde podrá modificar los parámetros del modelo dinámico del CPI, elegir el modo de entrada de control (interno o externo), visualizar información sobre los parámetros, etc.

Comunicación serial. En todo momento el programa envía y recibe datos por el puerto serial. Envía un carácter dependiendo del botón en el que haga clic el usuario.

3.4. Diseño del software embebido en la tarjeta Hercules RM57Lx

Un sistema en tiempo real es capaz de realizar varias operaciones simultáneas. Usualmente en el uso de microcontroladores, la ejecución en paralelo se consigue como la sucesión rápida de actividades secuenciales. Se cuenta con varias técnicas para ejecutar tareas en paralelo, una de ellas es mediante interrupciones, con base a esta se pueden comunicar los módulos mediante banderas (flags), variables, contadores, etc.

Para cumplir con los requisitos de un sistema en tiempo real, se realiza la programación usando las diferentes interrupciones de la tarjeta como son los módulos ADC, RTI, SCI. El lenguaje utilizado es C, compilado y ejecutado en el entorno Code Composer Studio.

En la Figura 3.4 se muestran los bloques y el flujo interno de datos del programa embebido en el MCU. Como primera tarea se inicializan los módulos que utilizará el microcontrolador. Constantemente el programa estará verificando la recepción de datos, la lectura de una señal analógica y generando una interrupción. Para mayor control de los tiempos de procesamiento, el programa recibirá un carácter y de acuerdo con éste, hará 6 tareas diferentes: modo sin control, aplicar un control de estabilización interno, controlar el péndulo de acuerdo con una entrada de control externa (señal analógica), modificar los parámetros del CPI, cambiar la posición del péndulo y/o del carro, este último se puede tomar como una perturbación al sistema, es decir, mover al péndulo de su posición de equilibrio.

Posteriormente se resuelve el modelo dinámico del CPI con base al método numérico de Euler, donde las variables de entrada de las ecuaciones son x1a, x2a, x3a, x4a estas son las posiciones y velocidades anteriores (inicializadas en ceros), después de resolver las ecuaciones del modelo se obtienen x1, x2, x3, x4, que son las posiciones y velocidades actuales, dos de ellos, la posición del carro y la posición del péndulo (x1, x3) serán enviados por puerto serial a la interfaz gráfica, además estas posiciones son escaladas (de forma proporcional) en dos salidas analógicas (PWM).

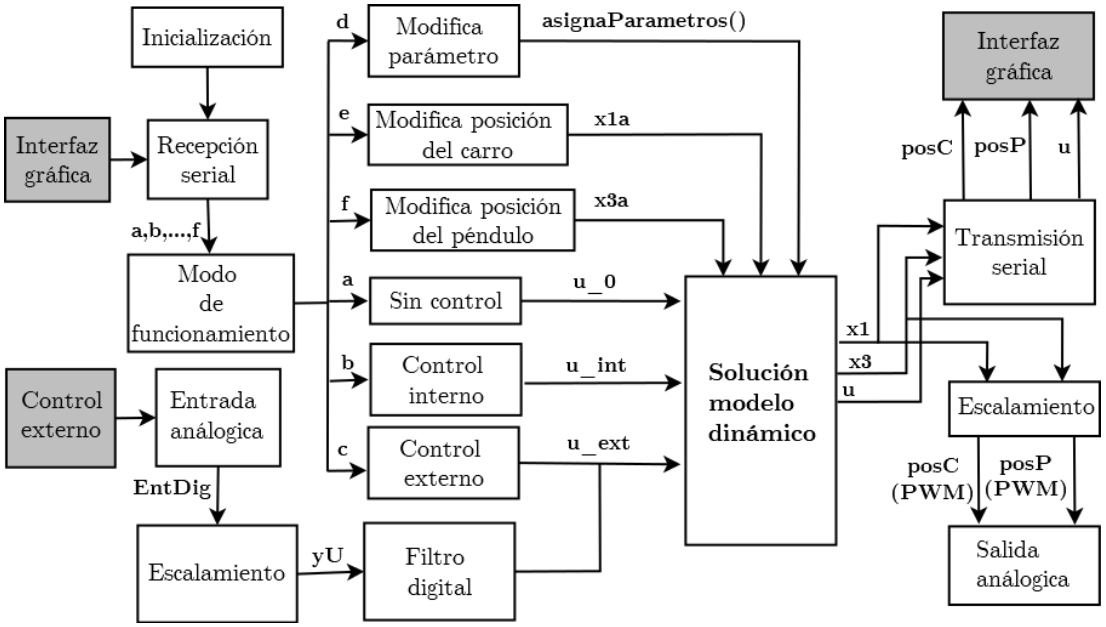

Figura 3.4: Diagrama general del software embebido en el MCU.

La variable de control (u) tiene tres modos de funcionamiento como entrada del modelo dinámico (para más detalles ver la sección 3.6), cada una se describe a continuación:

- ✓ u_0 se activa cuando se recibe el carácter 'a' ésta toma el valor de 0, es decir el modelo dinámico no tendrá entrada de control.
- ✓ u_int se activa cuando el carácter leído es 'b' entonces se ejecuta un algoritmo de control de estabilización (control por realimentación de estados) embebido en el MCU.
- ✓ u_ext se activa cuando el carácter recibido es 'c' entonces se ejecuta el bloque de control externo, la señal analógica se escala y se filtra para actuar como variable de entrada del modelo dinámico del CPI.

A continuación, se describen las diferentes variables usadas en la Figura 3.4.

- ✓ a, b, ..., f son los diferentes caracteres que puede recibir el programa y de acuerdo con éste se ejecuta un modo de funcionamiento.
- ✓ EntDig es una variable (entero) producto de la conversión analógica digital, ésta puede tomar valores de 0 a 4095.
- ✓ yU es la variable resultante del escalamiento de la entrada de control toma valores de -55 a 55 (volts).
- ✓ u es la entrada de control del sistema, ésta señal ya ha sido filtrada digitalmente. Dependiendo del modo de funcionamiento u puede valer 0 (sin control) o bien se calcula de acuerdo con un control de estabilización interno.
- ✓ x1a, x3a representan el estado anterior de la posición del carro y de la posición del péndulo respectivamente.
- ✓ x1, x3 es el estado actual de las posiciones, estas son enviadas a la interfaz gráfica.
- ✓ posC, posP representan la posición del carro y la posición del péndulo en milímetros y grados respectivamente, estas variables se envían por puerto serie a la interfaz gráfica. De la misma forma se escalan para generar una señal analógica a través del módulo PWM.
- ✓ u_0 es la variable de control equivalente a 0 volts.
- ✓ u_int es la variable de control interno, derivado de un controlador de estabilización del péndulo procesado en el mismo microcontrolador.
- ✓ u_ext es la variable de control proveniente de una señal externa.

3.5. Entrada analógica y escalamiento.

El CPI es un sistema subactuado por lo tanto requiere de una señal de control para regir el comportamiento del carro y péndulo. Para obtener esta señal se usa el módulo ADC que convierte la señal analógica del controlador en digital, después se escala o se adapta para representar el dato digital en su valor adecuado.

El rango de voltaje de la entrada analógica de la tarjeta Hercules RM57Lx es de 0 a 3.3 volts, una vez realizada la conversión se obtiene un valor digital que toma valores de 0 a 4095. Con la finalidad de reducir el ruido de entrada del módulo ADC es necesario filtrar la señal. El filtro utilizado es de promedio, toma 50 muestras analógicas y el valor usado es el promedio de éstas. A continuación, se muestra el diagrama de flujo (ver Figura 3.5) de la conversión analógico-digital y escalamiento de la señal de control.

Para escalar la señal se utiliza la ecuación (3.1), donde PromEntDig es el valor digital de entrada, **u** es el valor escalado.

$$u = \frac{110}{4095} * promEntDig - 55 \tag{3.1}$$

De esta forma **u** toma valores en el rango de -55 a 55 que es el voltaje de funcionamiento del motor del sistema CPI. Por ejemplo, si $\mathbf{PromEntDig} = 0$ entonces $\mathbf{u} = -55$. Dado el valor $\mathbf{PromEntDig} = 2048$, $\mathbf{U} = 0.01343$ y si $\mathbf{PromEntDig} = 4095$, $\mathbf{u} = 55$.

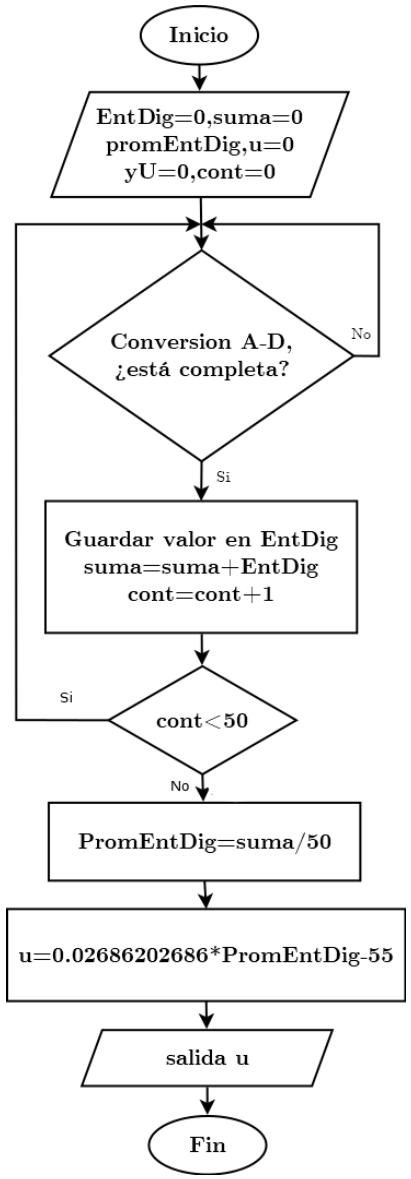

Figura 3.5: Escalamiento de la señal de control.

3.6. Modos de entrada de control

El microcontrolador recibe por puerto serial un carácter que le indica la tarea a realizar, tres de ellos son para los modos de funcionamiento de la entrada de control del emulador.

Sin control. En este este modo el valor de entrada de control es cero, la posición del péndulo es de cero grados, es decir se coloca en una posición vertical superior. Sin control, el péndulo comienza a oscilar hasta detenerse. Se diseñó para fines demostrativos y ver el comportamiento del CPI en lazo abierto.

Control interno. En este modo se pone en marcha un control interno (retroalimentación de todos los estados) de estabilización del péndulo. Este control lo calcula el mismo microcontrolador, se diseñó para fines demostrativos y pruebas.

Control externo. En este modo el microcontrolador recibe la entrada de control a través del módulo ADC. Se diseñó para ver el comportamiento del sistema conforme a un controlador proveniente de una PC, o alguna otra tarjeta de procesamiento.

En la Figura 3.6 se muestra el diagrama de flujo de los modos de funcionamiento. opc es la variable que guarda el carácter recibido por el puerto serial, x3a es la variable de la posición del péndulo, u es la variable de control.

Si opc es igual a 'a' el programa ejecuta el modo sin control entonces x3a toma un valor muy cercano a 0, es decir la posición del péndulo se coloca en la posición vertical superior. u también toma un valor de casi 0, de esta forma al sustituirlo en las ecuaciones del modelo dinámico del CPI, la posición del péndulo comenzara a caer hasta detenerse.

Si la variable opc es igual a 'b', x3a igual toma un valor cercano a 0 pero en este modo, a u se le asigna el valor calculado de un controlador de estabilización, entonces el péndulo no caerá, el control tratará de mantenerlo posicionado en 0 grados.

Si opc es igual a 'c', x3a toma el valor de 3.1416, es decir, el péndulo se coloca en una posición vertical inferior. u toma valores provenientes de un controlador externo usando el módulo ADC del MCU, este último proceso se explica en la sección 3.5.

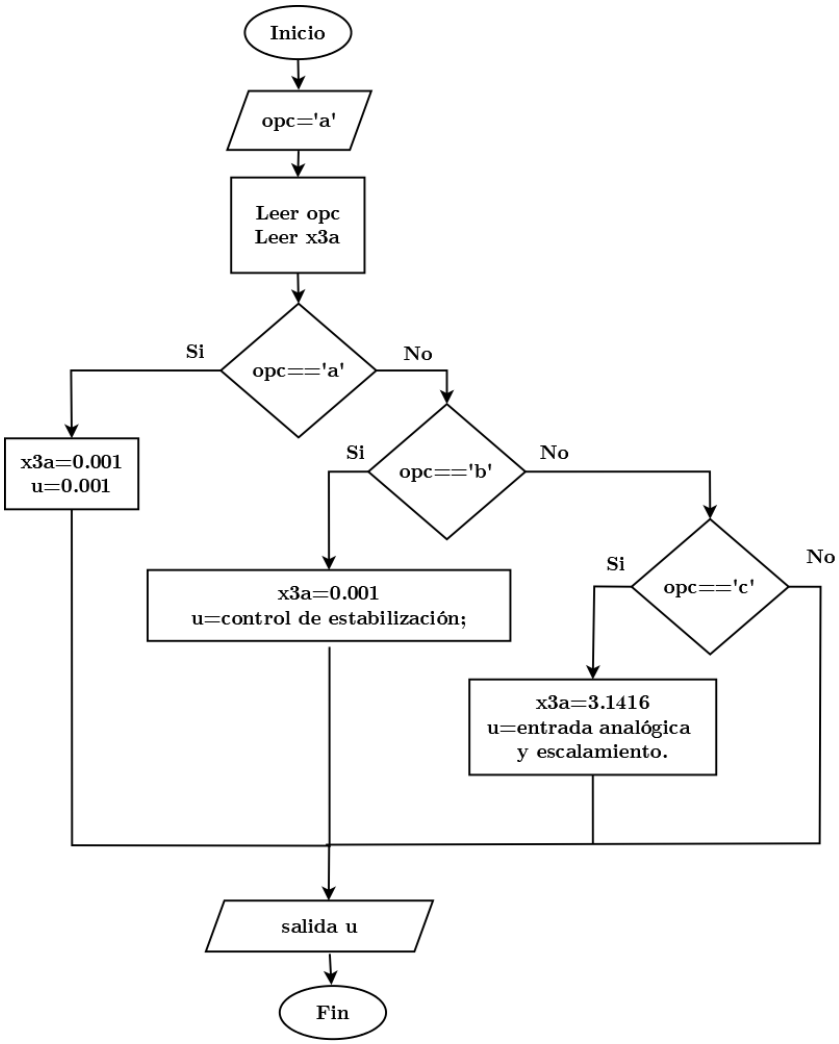

Figura 3.6: Diagrama de flujo de los modos de funcionamiento

3.7. Solución del modelo dinámico

El modelo dinámico describe el comportamiento del CPI en función de una variable de entrada (control), la salidas que interesan son las posiciones del carro y péndulo, para obtenerlas es necesario solucionar (o aproximar) el sistema de ecuaciones diferenciales del modelo dinámico visto en (1.49) y (1.53), para ello se representan la ecuaciones en variables de estado como en (1.57) después se utiliza el método numérico de Euler visto en la sección 1.5, obteniéndose el siguiente sistema de ecuaciones.

$$
\begin{aligned}
x1k &= x1a + h * x2a \\
x2k &= x2a + h * (-(a2 / a1) * x2a - (a3 / a1) * x4k * cos(x3a) + (a3 / a1) * \\
&\quad x4a^2 * sin(x3a) + (b1 / a1) * u) \\
x3k &= x3a + h * x4a \\
x4k &= x4a + h * (-(Fp / \beta) * x4a - (a3^2 / (a1 * \beta)) * x4a^2 * sin(x3a) * \\
&\quad cos(x3a) + ((a5 * sin(x3a)) / \beta) + ((a2 * a3) / (a1 * \beta)) * cos(x3a) * x2a - \\
&\quad ((a3 * b1) / (a1 * \beta)) * cos(x3a) * u);
\end{aligned}
\tag{3.2}
$$

Donde $\boldsymbol{u}$ es la entrada de control; $a1, a2, a3, a4, a5, b1$ son constantes que se definen en las ecuaciones (1.41) a la (1.46) y β en la ecuación (1.52). $x1k, x2k, x3k, x4k$ son las variables de estado actuales por calcular, $x1a, x2a, x3a, x4a$ son las variables de estado anteriores. Es importante señalar que las condiciones iniciales de estas variables son 0.

En la Figura 3.7 se muestra la forma de programación en la tarjeta Hercules RM57Lx. Se genera una interrupción al programa principal cada milisegundo, usando el módulo RTI, entonces el procesador ejecuta un bloque de código donde resuelve la ecuación diferencial obteniéndose las variables $\boldsymbol{x1k}$, $\boldsymbol{x2k}$, $\boldsymbol{x3k}$, $\boldsymbol{x4k}$, posteriormente la posición del péndulo $(x3k)$ que está en radianes es convertido a grados, este valor se almacena en la variable PosP y la posición del carro $(x1k)$ que esta en metros es convertido a milímetros, se almacena en la variable PosC.

Una vez calculadas estas variables, se termina la atención de la interrupción y se devuelve el tiempo de procesamiento al programa principal (main).

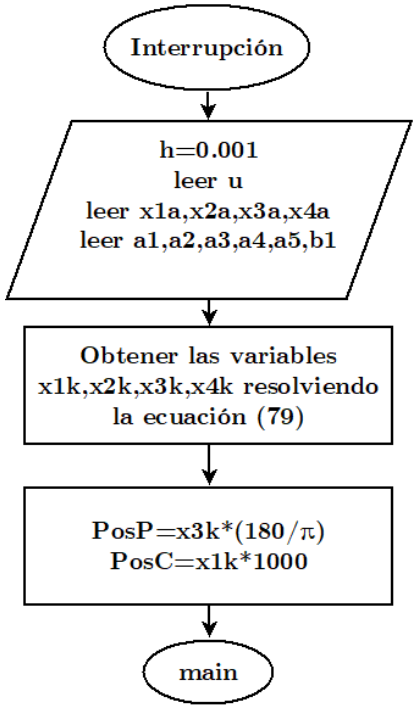

Figura 3.7: Diagrama de flujo del manejo de interrupción.

3.8. Salidas PWM

Para que el ESCPI pueda enviar la señal de posición del carro (PosC) y la posición del péndulo (PosP) a un controlador externo, es necesario simular dos salidas analógicas, para esta tarea se usa el módulo ePWM del microcontrolador.

La ecuación (3.3) se utiliza para obtener un valor proporcional a la variable de entrada, tomando en cuenta la longitud del riel del carro ($\pm 70\,cm$). $PosC$ es la variable de entrada en metros, $pwmPosC$ es el valor porcentual del ciclo de trabajo, se le asigna a la salida PWM de esta forma simula una señal analógica. Por ejemplo, si PosC equivale a 0.5m sustituyéndolo en la ecuación, pwmPosC será 85.7% este valor es el porcentaje del ciclo de trabajo de la señal PWM. Tomando en cuenta que el rango de voltaje de salida de la tarjeta es de 0 a 3.3v, con este porcentaje de ciclo de trabajo a la salida del pin se tiene un voltaje de 2.82V.

$$pwmPosC = 71.42857 * PosC + 50 \tag{3.3}$$

La frecuencia asignada a esta señal PWM es de 10kHz. La resolución es calculada con la ecuación (3.4), donde T es el periodo de la señal PWM y ΔT_{ON} es el salto o intervalo discreto de tiempo [33].

$$R = \frac{T}{\Delta T_{ON}} \tag{3.4}$$

El periodo de la señal T es de 100µs, la duración del salto mínimo de tiempo (ΔT_{ON}) es 0.853us. Por lo tanto, hay 117 valores distintos que puede tomar el ciclo de trabajo de la señal PWM. De otra forma el ciclo de trabajo puede cambiar mínimo cada 0.85%.

$$R = \frac{100us}{0.853us} = 117.2 \tag{3.5}$$

Se utiliza la ecuación (3.6), para obtener el porcentaje (**pwmPosP**) que será aplicado a la señal PWM, **PosP** es la variable de entrada en grados. A manera de ejemplo, si **PosP** tiene el valor de 120 grados, **pwmPosP** es igual a 36%, este porcentaje del periodo de la señal PWM estará en alto. El voltaje del pin de salida equivale a 1.18V.

$$pwmPosP = 0.23255 * PosP + 8.1395 \tag{3.6}$$

3.9. Comunicación serial

La comunicación serial se establece entre la tarjeta Hercules RM57Lx y una computadora personal que ejecuta la interfaz gráfica del ESCPI.

3.9.1. Transmisión de datos

Después de resolver el modelo dinámico del CP, se obtienen las posiciones del carro y del péndulo (PosC, PosP), éstos se transmiten a la interfaz gráfica a través del módulo SCI del microcontrolador. Primero se envía la posición del péndulo enseguida la del carro, separados por una coma (,) y terminando con un carácter de nueva línea (\n), la posición del péndulo se envía en grados y la posición del carro en milímetros. Por ejemplo, una trama se envía de la siguiente forma "90, 200, \n".

3.9.2. Recepción de datos

La interfaz gráfica envía constantemente un carácter al microcontrolador, dependiendo de cuál reciba ejecutara alguna tarea. En la Tabla 8 se muestra la

descripción de cada tarea a realizar dado el carácter recibido, las tres primeras tareas se explican detalladamente en la sección 3.6.

Si el carácter recibido es una 'd', el MCU recibirá una cadena de caracteres separados por un punto y coma (';'), en total son 12 parámetros, al final se recibe el carácter '#' dando a entender el fin de la cadena, estos parámetros se guardan en el microcontrolador. El orden de los parámetros a recibir es: $J,\ R,\ Mc,\ m,\ L,\ Fc,\ Fp,\ Ra,\ g,\ K,\ I,\ D$, en la .

Tabla 1 y Tabla 2 se describen estos parámetros. Por ejemplo, se reciben los datos "0.0011; 0.030; 1.5;0.9; 0.343; 0.07; 0.005; 3.9; 9.81; 0.0535; 0.0132; 0.0003#".

Tabla 8: Descripción de cada tarea.

Carácter	Tarea	Descripción
a	Sin control	La entrada de control $\boldsymbol{u}$ toma el valor de cero (0), se resuelve el modelo dinámico y se obtienen las posiciones del sistema en lazo abierto.
b	Control interno	La entrada de control $\boldsymbol{u}$ está en función de un control de estabilización interno.
c	Control externo	La entrada de control $\boldsymbol{u}$ está en función de una entrada analógica proveniente de un control externo.
d	Modifica parámetros	Recibe los parámetros enviados desde la interfaz gráfica. Estos parámetros se guardan (actualizan) en el MCU.
e	Modifica la posición del carro	Recibiendo este carácter, el microcontrolador actualiza la posición del carro (PosC).
f	Modifica la posición del péndulo	Actualiza la posición del péndulo (PosP)

3.10. Interfaz gráfica

Para poder visualizar el comportamiento del carro-péndulo y configurar sus parámetros, se desarrolló una interfaz en LabVIEW donde se muestran las gráficas de salida. Para visualizar el movimiento de rotación y traslación se diseñó el carro-péndulo en el software Unity 3D.

3.10.1. Interfaz gráfica en LabVIEW

En la Figura 3.8 se observa el diagrama de flujo donde muestra la comunicación entre la tarjeta de desarrollo y la interfaz gráfica diseñada en LabVIEW.

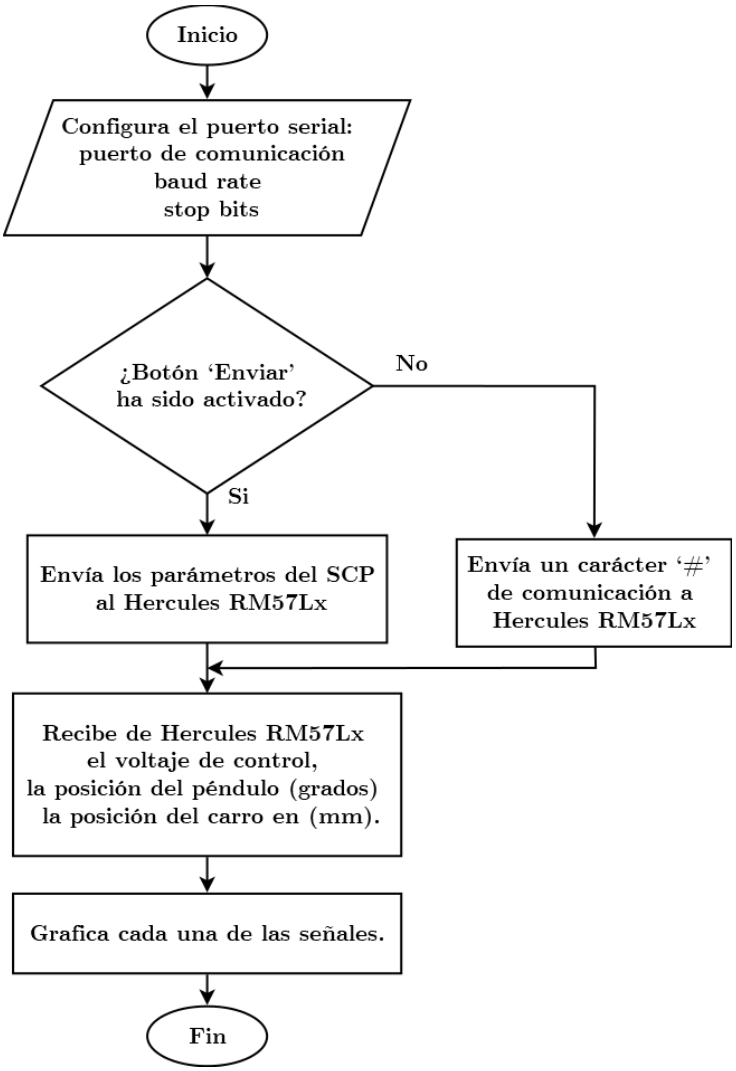

Figura 3.8: Flujo de intercambio de datos de la IG y el microcontrolador

El emulador genera dos variables de salida, la posición del carro y la posición del péndulo los envía a LabVIEW para su procesamiento y visualización. En la Figura 3.9 se muestra la interfaz gráfica, contiene tres pestañas: 'gráficas', muestra la gráfica de la posición del péndulo y carro en tiempo real; 'parámetros' en esta pestaña se configura los valores de los parámetros del carro-péndulo puede cambiarse en tiempo de ejecución, por último se tiene la pestaña 'puerto serial' en donde se configura el puerto de comunicación, los bits de parada, baudios, etc.

En la pestaña ‘Gráficas’ a la izquierda se muestra la gráfica de la posición del péndulo en tiempo real y a la derecha se muestra la posición del péndulo una vez terminada la emulación.

Figura 3.9: Interfaz gráfica del simulador HIL desarrollada en LabVIEW

3.10.2. Interfaz gráfica en Unity

Se diseña una interfaz gráfica en Unity para representar los movimientos de rotación y traslación del ESCPI. Las dimensiones son aproximadas y se tomaron de la planta que se encuentra en el laboratorio de robótica de la Universidad Tecnológica de la Mixteca.

Para el desarrollo de la interfaz se utilizó Unity 5.6.1, para ejecutarlo es necesario tener.

- ✓ Sistema operativo (mínimo): Windows vista SP1, MAC OS X 10.9, Ubuntu 12.04.
- ✓ Tarjeta de video con capacidad para DX10.
- ✓ CPU: compatible con el conjunto de instrucciones SSE.

En la Figura 3.10 se muestran las variables que se transmiten de la tarjeta a la interfaz y viceversa. La interfaz gráfica recibe la posición del carro, posición del péndulo y el voltaje de control, de igual forma envía datos a la tarjeta para modificar parámetros del modelo dinámico del CPI.

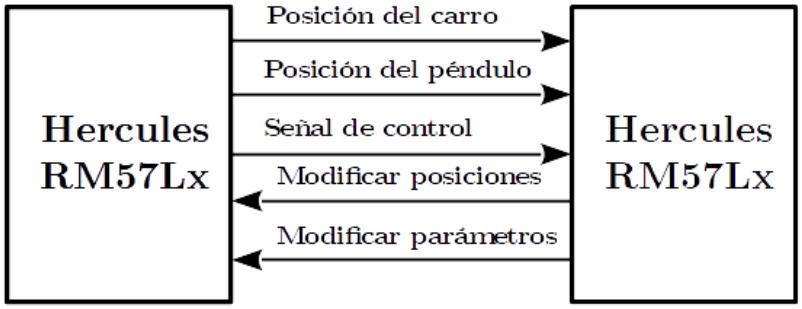

Figura 3.10: Comunicación entre la tarjeta Hercules y Unity

3.11. Descripción general de la interfaz grafica

En la Figura 3.11 se muestra la estructura jerárquica de la GUI, los bloques se describen en los siguientes.

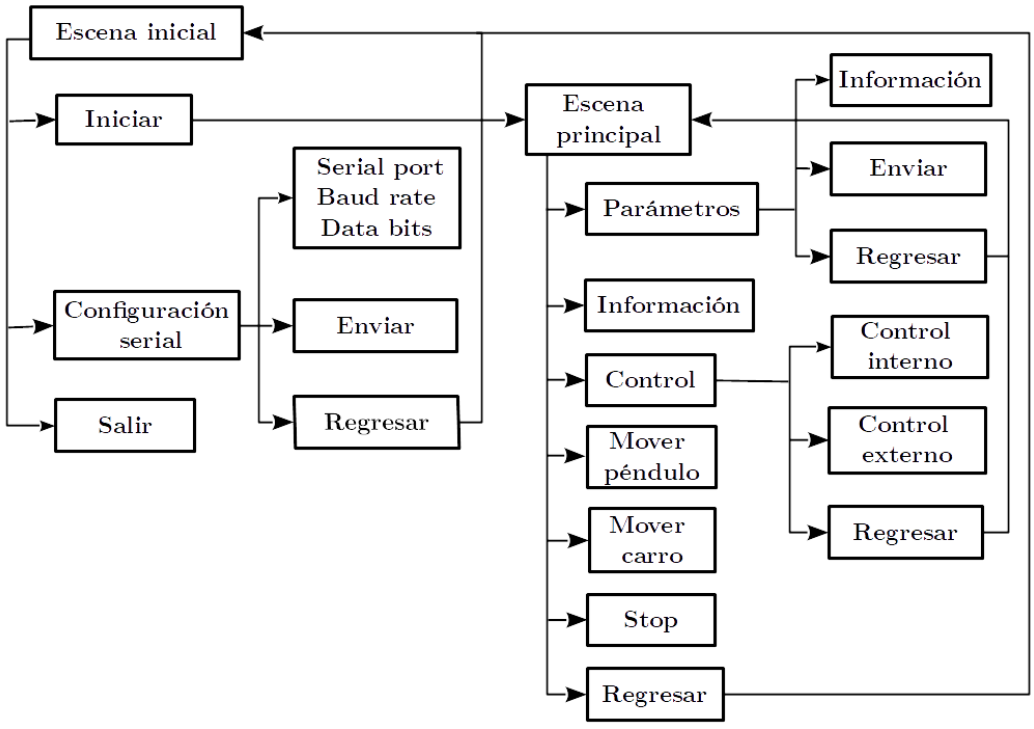

Figura 3.11: Estructura jerárquica de la GUI

Escena inicial: es la escena que aparece en cuanto se ejecuta el programa, es la portada del emulador, cuenta con tres botones (ver Figura 3.12).

Iniciar: al dar clic en este botón se muestra al usuario la escena principal.

Configuración serial: al dar clic aparece un menú, se pueden configurar 3 parámetros de la comunicación serial: **serial port**, en este recuadro se ingresa el puerto de comunicación a utilizar, por default contiene COM3. Este campo debe de ser configurado correctamente y antes de iniciar el emulador de lo contrario aparecerá un mensaje de error en el siguiente escenario. **Baud rate**, este campo se refiere a la velocidad de transmisión, de forma predeterminada el valor de este campo es 115200 bps, cabe mencionar que este parámetro no se debe modificar debido a que la tarjeta Hercules RM57Lx transmite los datos a esa velocidad, se colocó para pruebas y fines informativos. **Data Bits** muestra los bits de datos, de igual forma al anterior este campo no se puede modificar, por default el valor es 8.

Dentro de este menú hay dos opciones más; **Enviar**, al hacer clic en este botón, se envían los parámetros antes mencionados al MCU; **Regresar**, este botón se utiliza para regresar a la escena anterior, en este caso a la escena inicial.

Salir: al hacer clic, se cierra el emulador.

Figura 3.12: Escena inicial del ECPI

Escena principal: en esta escena es donde se muestra el carro péndulo en movimiento, tiene varios botones para transmitir datos al MCU. En la Figura 3.13 se aprecia la interfaz y sus diferentes opciones.

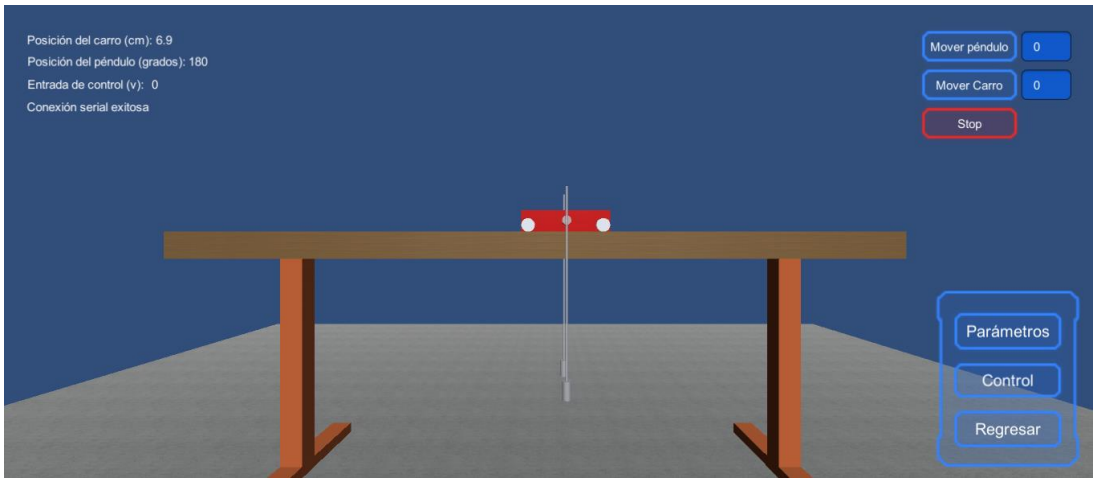

Figura 3.13: Escena principal del ESCPI.

Parámetros: al dar clic en este botón aparecen un conjunto de campos de entradas, son los parámetros del modelo dinámico del CPI donde en caso necesario se pueden cambiar los valores (ver Figura 3.14). Este menú cuenta con un panel de información donde describe cada parámetro visto en la .

Tabla 1, para visualizarlo dar clic en información. Igualmente incluye los botones enviar y regresar.

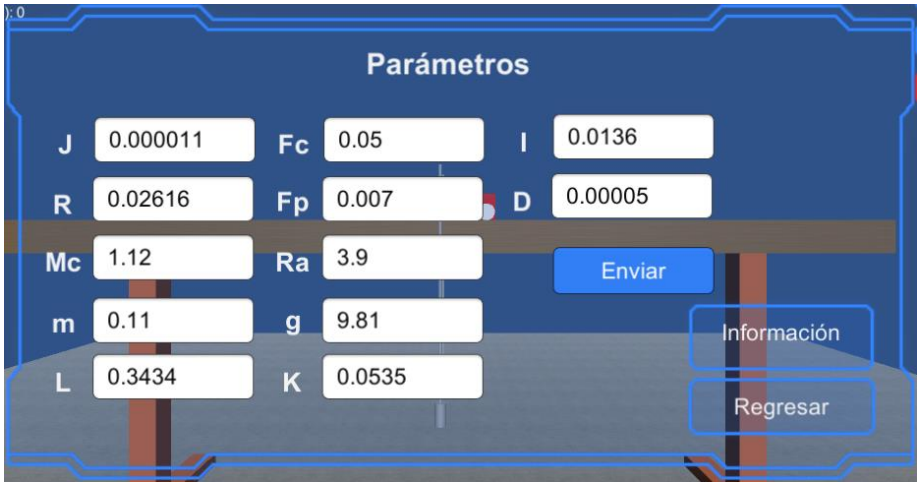

Figura 3.14: Parámetros del modelo dinámico del CPI.

Control: En este menú hay dos botones con diferentes modos de control; **control interno,** utilizado para activar un algoritmo de control interno, es decir, el algoritmo de control lo procesa y calcula el MCU; **control externo**, al hacer clic en este botón, el MCU recibe la entrada de control a través de su módulo convertidor analógico digital.

Mover péndulo: envía la posición del péndulo ingresada en el campo, enseguida el MCU actualiza la posición modificando el comportamiento del modelo dinámico de la planta.

Mover carro: repite el proceso anterior con la posición del carro.

Stop: detiene la simulación colocando todos los estados del CPI a cero, dicho de otra forma, la posición y velocidad del péndulo, la posición y velocidad del carro son cero.

Información: este bloque se refiere a información desplegada en la esquina superior izquierda de la pantalla, se visualiza en tiempo real las posiciones del carro, péndulo, la entrada de control y el estado de conexión entre la GUI y el MCU.

Regresar: regresa a la escena inicial.

Con la ayuda del Asset: Grapher - Graph, Replay, Log [34] pulsando la combinación de teclas Ctrl+G, aparece una pantalla donde se visualizan las gráficas de la posición del péndulo (color cyan), posición del carro (color verde) y la entrada de control (color blanco). Al detener la simulación se guardan los datos (si el usuario lo requiere) de las gráficas en un archivo de Excel para posteriormente reproducirlos en el mismo graficador. En la Figura 3.15 se observa el panel donde se grafican las variables del emulador.

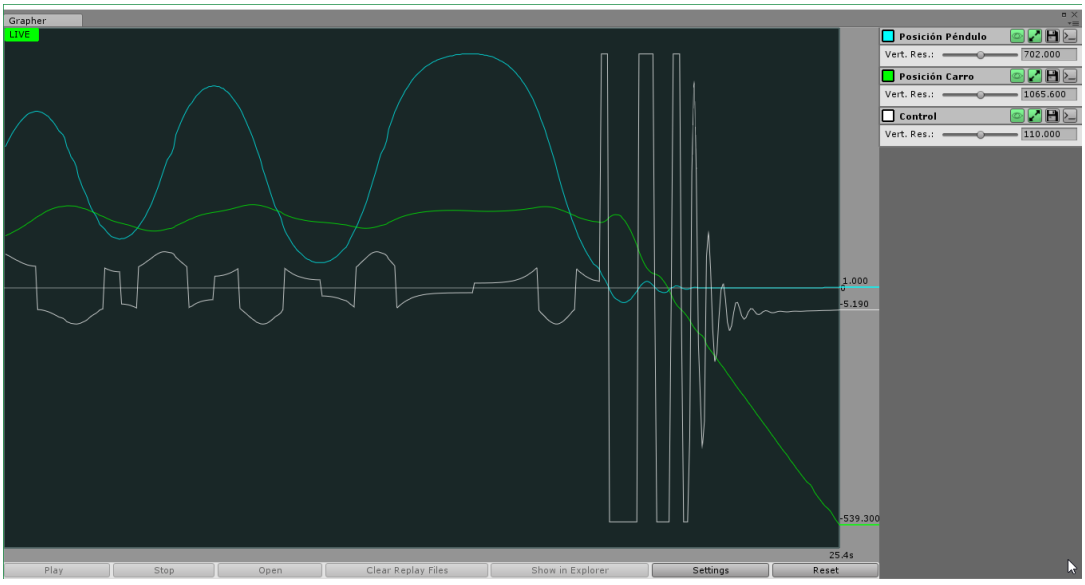

Figura 3.15: Panel para observar las graficas

3.12. Codificación de la GUI del ESCPI

En Unity un juego es dividido en escenas, son espacios vacíos que pueden ser llenados con GameObjects, estos son objetos fundamentales que representan personajes, accesorios y el escenario, su comportamiento es controlado por components.

Se puede crear un "component" utilizando scripts que son documentos que contiene código de programación. Para el desarrollo de esta interfaz gráfica se crearon 6 scripts en el lenguaje de programación C#, a continuación, se describen los scripts más importantes.

Script inicializarPuerto: este script se encarga de leer los campos de la configuración del puerto serial y asignárselos a las variables SP, BR, DB. Además, evita que, al pasar a la siguiente escena, las variables mencionadas sean eliminadas.

Script cambiarEscena: en este programa se realizan los cambios entre la escena principal y la escena inicial.

Script Tras_Rot: es el módulo principal, se agregan las variables y constantes de la comunicación serial, por default se inicializan con las siguientes, puerto COM3, velocidad de transmisión 115200 baudios, bits de datos 8, sin paridad, dos bits de paro. Se leen constantemente las variables de entrada y con base a estos datos sitúa al péndulo y al carro.

En el diagrama de flujo de la Figura 3.16 se muestra la forma de transmisión-recepción de datos entre la tarjeta Hercules RM57Lx y el software Unity. Se inicializa el puerto serial con los valores ingresados, si hubiera algún error al abrir el puerto, mostrará en pantalla en color verde el mensaje "Conexión serial fallida", de lo contrario en color blanco "Conexión serial exitosa".

Si la conexión serial es exitosa, de manera constante se envía la variable opc (un carácter) al MCU, esta variable toma valores de acuerdo con el botón que haya dado clic el usuario, en función del carácter enviado, el MCU ejecutará una tarea (en la Tabla 8 se muestran las diferentes tareas), además, enviará tres variables: la posición del carro, posición del péndulo y la señal de control. Unity recibirá estas posiciones, en seguida dará movimiento al carro y péndulo, también mostrará en pantalla el valor numérico de las variables. Cabe señalar que en la Figura 3.16 los dos "si" que se observa se refiere a que Unity ejecuta dos tareas en paralelo.

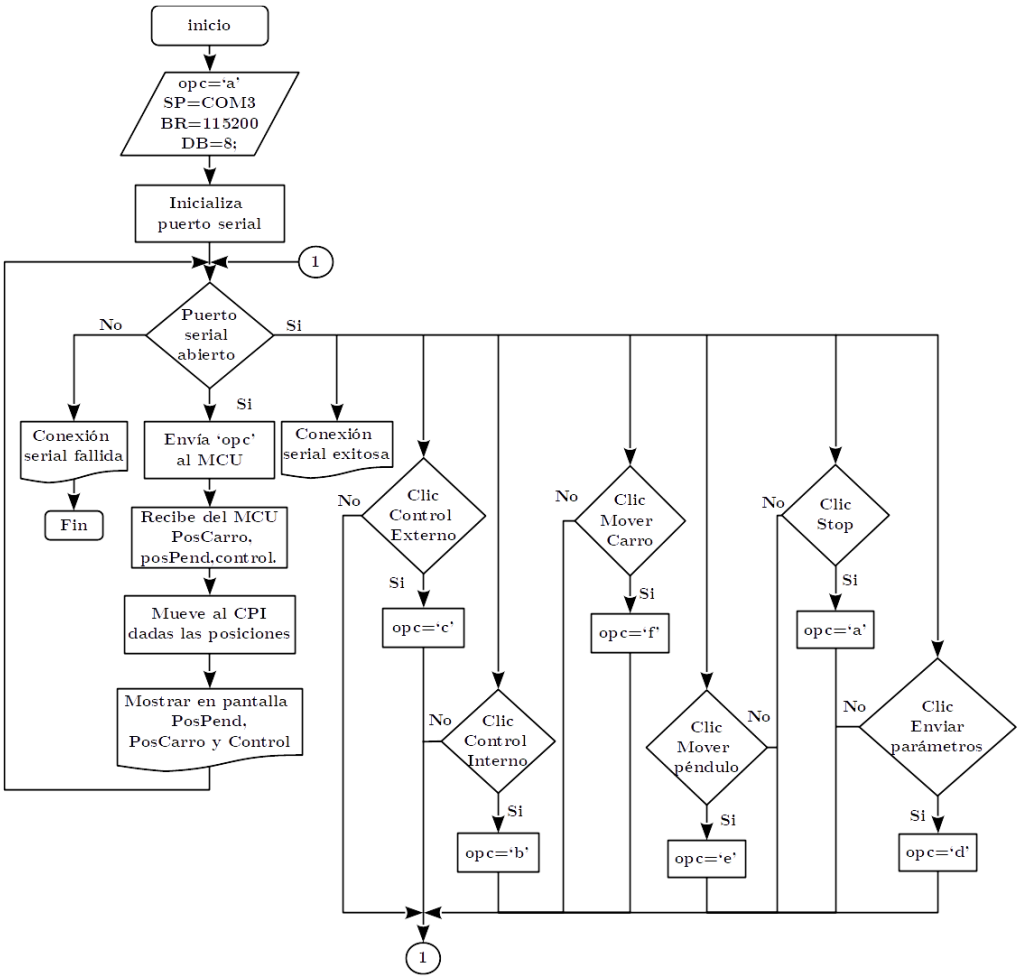

Figura 3.16: Diagrama de flujo de la comunicación entre el MCU y Unity

Capítulo 4
Pruebas y resultados

En este capítulo se presentan las pruebas y resultados realizados en cada etapa del proyecto. Inicialmente, se describe la simulación del modelo dinámico (sin entrada de control) desarrollada en MATLAB/Simulink, posteriormente, se muestra la emulación del mismo modelo, pero la solución se obtiene de la tarjeta de desarrollo Hércules RM57L.

4.1. Simulación del modelo dinámico del CPI

En este proyecto, se emula la planta (el Carro-Péndulo Invertido) y el controlador en hardware. Antes de embeber el modelo del sistema carro-péndulo, se procede a verificar que el modelo dinámico obtenido, reproduce de forma satisfactoria el comportamiento observado en una planta real. Se diseña el modelo en el software MATLAB/Simulink tal como lo muestra la Figura 4.1, el modelo dinámico obtenido anteriormente (ecuación (1.57)) es codificado en un bloque de funciones (MATLAB Function), donde:

u	=	Entrada de control
t(theta)	=	Posición del péndulo
tp (theta punto)	=	Velocidad del péndulo
x (equis)	=	Posición del carro
xp(equis punto)	=	Velocidad del carro

Los parámetros se tomaron de la .

Tabla 1 y Tabla 2, estos indican los valores reales del sistema carro-péndulo modelo 33-005-PCI de la compañía Feedback Instruments.

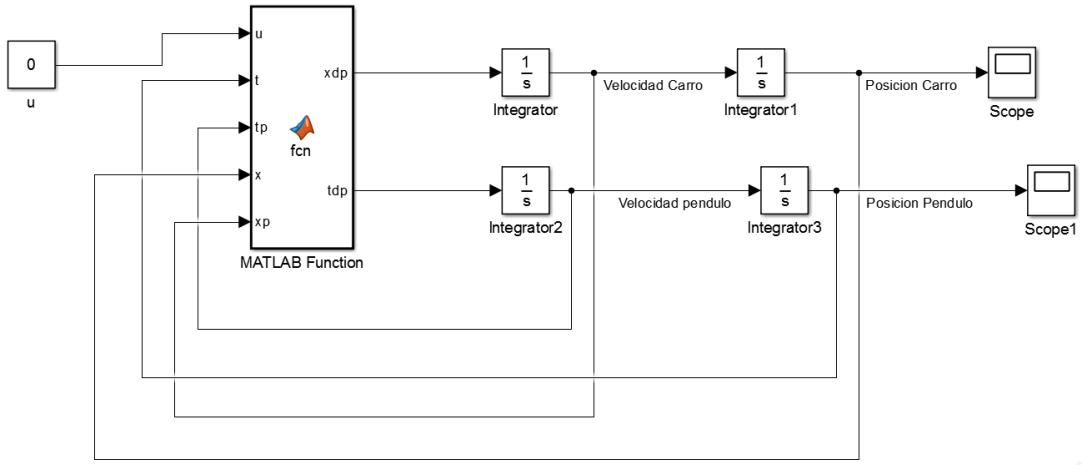

Figura 4.1: Modelo dinámico del sistema carro-péndulo en bloques.

Físicamente si un péndulo con fricción se deja caer desde la vertical superior oscilará bajo la acción gravitatoria y se detendrá en algún momento, con base a esta afirmación, a la entrada de control del modelo se le asignó el valor de cero (u=0), obteniendo la siguiente respuesta de la posición del péndulo. En la Figura 4.2 se observa la gráfica de ésta, que inicia en la posición vertical superior (0 grados), se pone a oscilar y en el transcurso del tiempo se va acercando (deteniendo) al valor de π (180 grados).

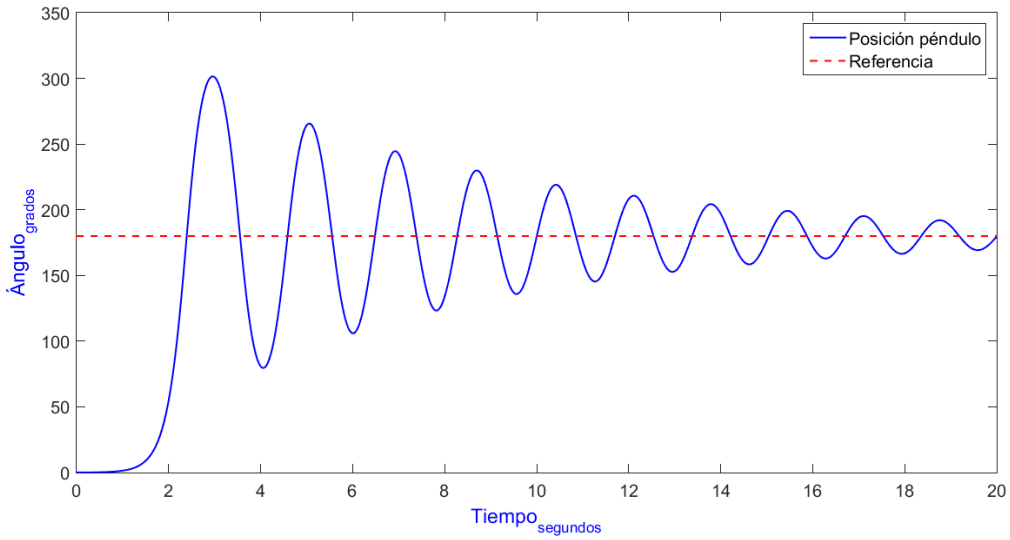

Figura 4.2: Posición del péndulo sin entrada de control.

4.2. Comparación de la simulación en Matlab vs Simulador HIL

Para verificar que funcione adecuadamente el simulador HIL, se compara su respuesta contra la que se obtiene con el modelo editado en MATLAB/Simulink. En las siguientes dos figuras se muestran una comparación entre la respuesta obtenida con MATLAB/Simulink y con el simulador HIL, las condiciones son sin entrada de control y posicionando inicialmente al péndulo en 0 grados. La respuesta corresponde a la posición del péndulo.

Se configuró Simulink para que la solución del modelo sea realizado con el método de Euler. El periodo de muestreo (h) es de 10 ms. Como se puede observar las salidas son similares. Sin embargo, la señal del simulador HIL tarda un poco más en converger a la posición final de 180 grados.

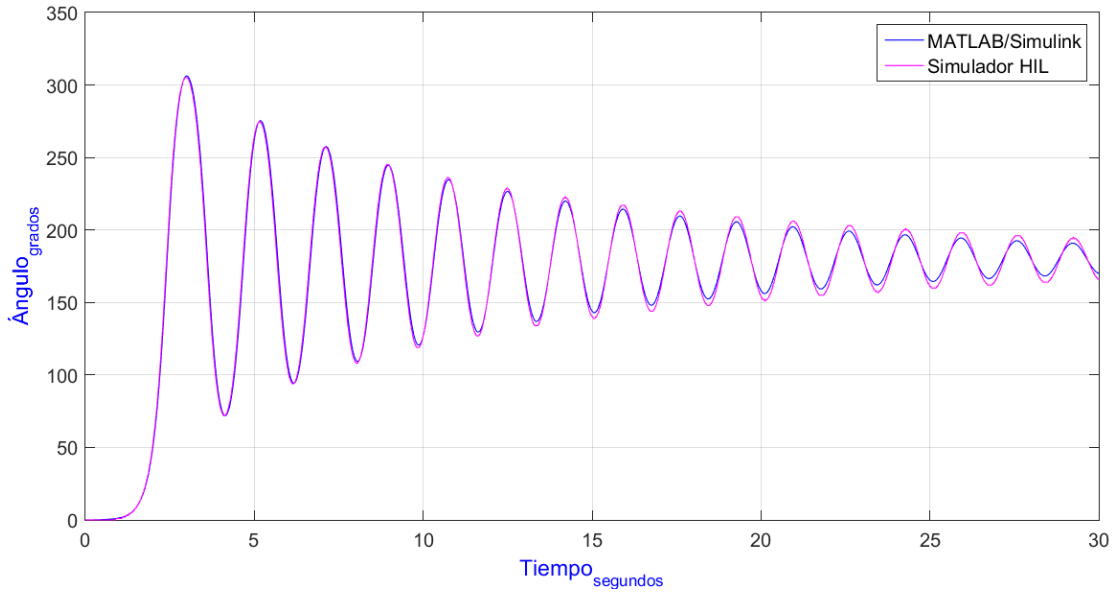

Figura 4.3: Grafica comparativa con un periodo de muestreo de 10 ms.

En la Figura 4.4 se muestra la comparación de señales con un tiempo de muestreo menor (h=1ms), las señales tienen una mayor similitud que con h=10ms, además de que disminuye el tiempo en el que convergen a 180.

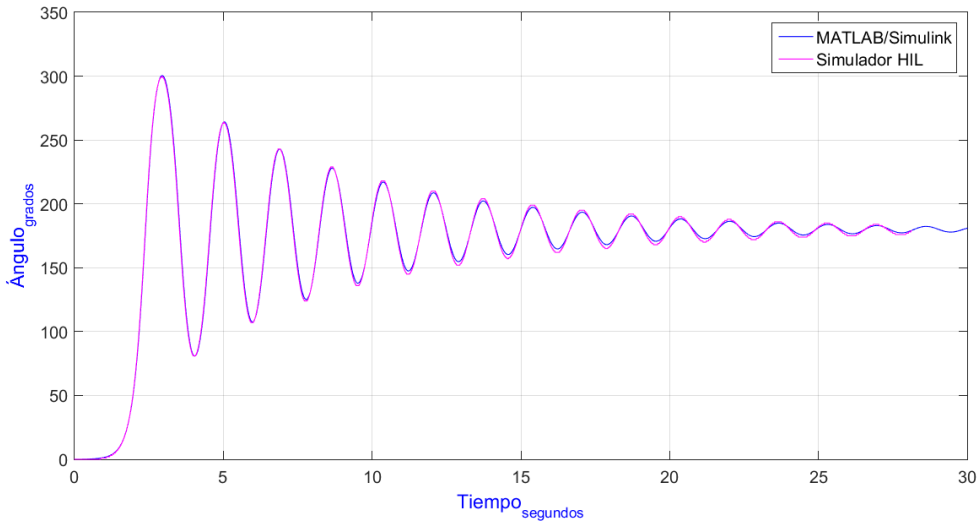

Figura 4.4: Grafica comparativa con un periodo de muestreo de 1 ms.

4.3. Tiempo de ejecución de operaciones básicas

Para que el simulador HIL funcione en tiempo real, es necesario saber el tiempo que toma el procesador en resolver cada operación matemática básica. En la Tabla 9 se muestran los resultados obtenidos, la cual se ponen las operaciones, los operados y el tiempo que tarda en realizar cada operación (en ciclos de reloj y µs). Se utilizó la librería optimizada para procesadores ARM que proporciona el sitio web de Texas Instruments.

La fórmula utilizada para calcular el tiempo es:

$$tiempo = \frac{CiclosDe\,\mathrm{Re}\,loj}{330 \times 10^{6} Hz} \tag{4.1}$$

Ciclos de reloj es, el número de ciclos que se lleva el microcontrolador en realizar algún calculo, este se obtiene al depurar el programa en Code Composer Studio.

Tabla 9: Tiempo de cómputo de operaciones básicas

Operación	A	B	Ciclos de reloj CPU	Tiempo
suma	8.234765	2.976323	53	0.16us
Resta			53	0.16us
Multiplicación			53	0.16us
División			53	0.16us
Coseno			559	1.693us

Contabilizar una por una todas las operaciones que realiza el MCU para el cálculo del modelo dinámico del carro péndulo, es muy engorroso. Por lo cual se utilizó un complemento del software CCS con el que es posible contabilizar, los ciclos de reloj que le lleva realizar cierta función o grupo de funciones. Con lo cual se obtuvo que el tiempo que tarda en resolver las ecuaciones del modelo dinámico, con el método de Euler, por cada iteración y sin entrada de control (u=0) es:

$$Tiempo_{modelo} = \frac{3651}{330 \times 10^6} = 11.063\ us$$

Tiempo en que tarda en convertir un valor analógico a digital.

$$Tiempo_{ADC} = \frac{696}{330 \times 10^6} = 2.10us$$

Tiempo que se toma en generar una salida PWM: 2.4 µs.

4.4. Periodo de muestreo del simulador HIL

El periodo de muestreo afecta la dinámica del sistema discretizado, debe elegirse de forma adecuada para evitar que se inestabilice. Si el valor es muy grande por ejemplo un tiempo de 10ms o mayor, la posición del péndulo tiende a oscilar por más tiempo incluso podría no llegar a la referencia.

En la Figura 4.5 se muestra tres respuestas con diferente periodo de muestreo. En la gráfica de color verde se tiene un periodo de muestreo de 10ms, la turquesa 2ms, la azul 1ms y la roja 0.4ms. Con base en estas respuestas se concluye que con un periodo menor o igual a 3ms la salida del sistema es la misma. A medida que se incrementa este parámetro la salida tiene más oscilaciones (para 10ms). Nótese que la señal azul turquesa y la roja casi no se visualizan debido a que ambas salidas son casi idénticas.

Por lo antes expuesto se eligió un periodo de muestreo de 1ms para la solución del sistema de ecuaciones en el sistema HIL.

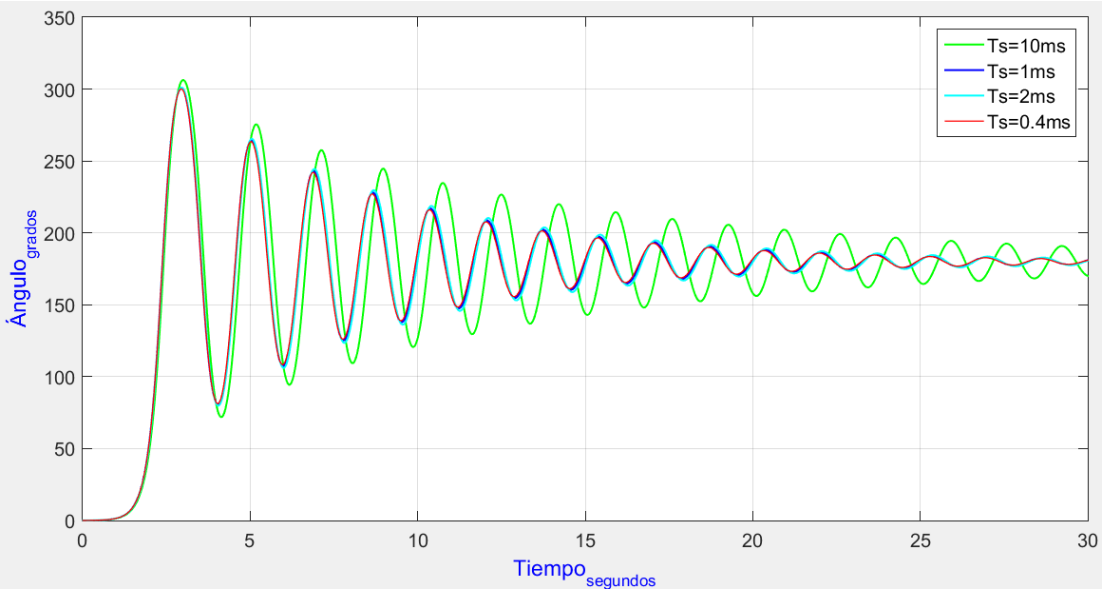

Figura 4.5: Posición del péndulo con base a diferentes tiempos de muestreo.

4.5. Simulador HIL

El simulador HIL del CPI consta de dos componentes principales, la tarjeta (microcontrolador) donde esta embebido el modelo dinámico del CPI y una laptop donde se ejecuta el entorno gráfico, de modo que se visualice los movimientos del CPI. En la Figura 4.6 se muestra el flujo de datos del ESCPI. En función de la entrada de control u la tarjeta Hercules RM57Lx resuelve las ecuaciones del modelo dinámico obteniendo la posición del péndulo y la posición del carro, con base a estas variables se generan dos señales analógicas que están disponibles para conectarse a un controlador, las señales son generadas a través del módulo PWM de la tarjeta. La entrada de control u es una señal que proviene del exterior y es procesada a través del módulo ADC. Es conveniente que las señales se filtren antes de conectarse a una tarjeta de adquisición de datos o a la tarjeta.

Las tres variables se envían por puerto serie a una computadora donde se ejecute la interfaz gráfica, las posiciones enviadas determinaran los movimientos del CPI, la variable de control se envía para visualizarla, analizarla en una gráfica. Esta interfaz gráfica se desarrolló con el software Unity.

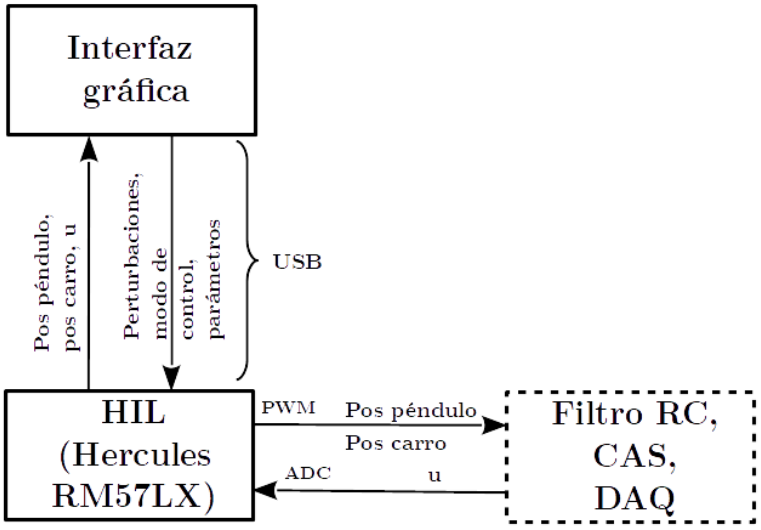

Figura 4.6: Flujo de datos del ECPI

4.6. Conexión del simulador

La tarjeta Hercules RM57Lx cuenta con más de 80 pines de entradas y salidas, el simulador solo ocupa una entrada analógica y dos salidas PWM. En la Figura 4.7 se muestran algunos pines utilizados para interactuar con otros dispositivos.

Los pines temporizadores de gama alta HET2 y HET18 (High-End Timer) se utilizan para generar salidas PWM, la posición del carro y la posición del péndulo respectivamente. El pin ADC7, es utilizado para leer la señal analógica de control. USB Mini es el puerto de comunicación serial.

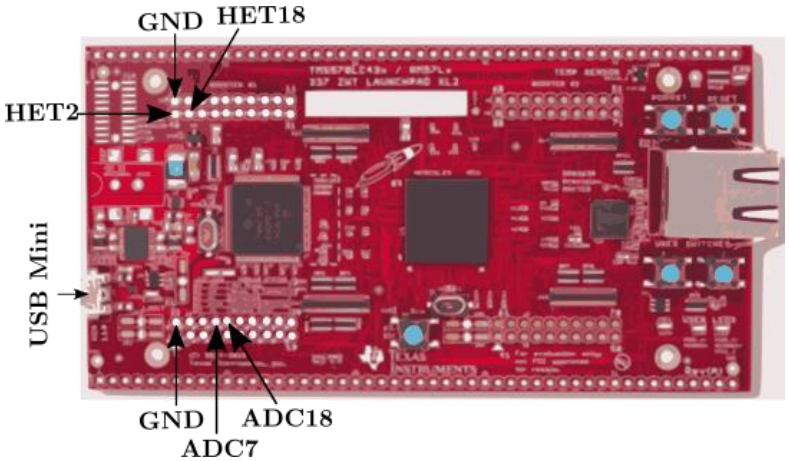

Figura 4.7: Pines de conexión del simulador HIL

Con base al flujo de datos de la Figura 4.6 y mediante el pin **HET18**, se mide en un osciloscopio la señal analógica del péndulo del SCP. En la Figura 4.8 se muestra la señal obtenida, cabe señalar que el SCP no tiene entrada de control (u=0) y el péndulo se posiciona inicialmente en 0 grados, es decir, en la posición vertical superior.

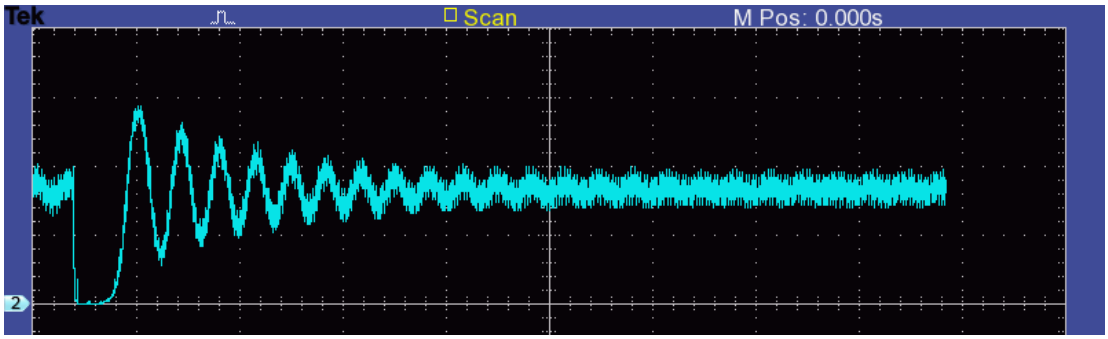

Figura 4.8: Señal analógica de la posición del péndulo.

Como se observa, la señal muestra las mismas oscilaciones de acuerdo con lo escrito en 4.2, sin embargo, la señal contiene bastante ruido por ello se utilizará un filtro digital para reducirlo.

4.6.1. Conexión entre el ESCPI y el controlador

Para demostrar el funcionamiento del emulador HIL se conectó con un controlador diseñado en MATLAB/Simulink, en la Figura 4.9 se muestran los detalles del flujo de datos. Son tres las señales que interactúan, la posición del carro, posición del péndulo y la señal de control. A continuación, se describe la función de cada uno de los bloques.

- ✓ Controlador: es diseñado en Simulink, se ejecuta en una laptop o PC, recibe y envía datos por puerto serie a través de una tarjeta de adquisición de datos (DAQ). Recibe la posición del péndulo y posición del carro. Envía la señal de control.
- ✓ DAQ: Es una tarjeta Hercules RM57Lx usada para adquirir y enviar las señales entre el controlador y el emulador HIL. Esta tarjeta recibe la posición del carro y la posición del péndulo a través de su módulo ADC y envía la señal de control a través de su módulo PWM.
- ✓ Filtro RC: Consiste en una resistencia y un capacitor que en conjunto filtran las señales provenientes del simulador y las que se envían al controlador.

- ✓ HIL: es el emulador HIL del SCPI, los detalles de conexión se definen en la sección 4.6.1.
- ✓ Interfaz gráfica: Es el software desarrollado en Unity, que interactúa con el usuario, da movimiento al CP de acuerdo con los valores recibidos.

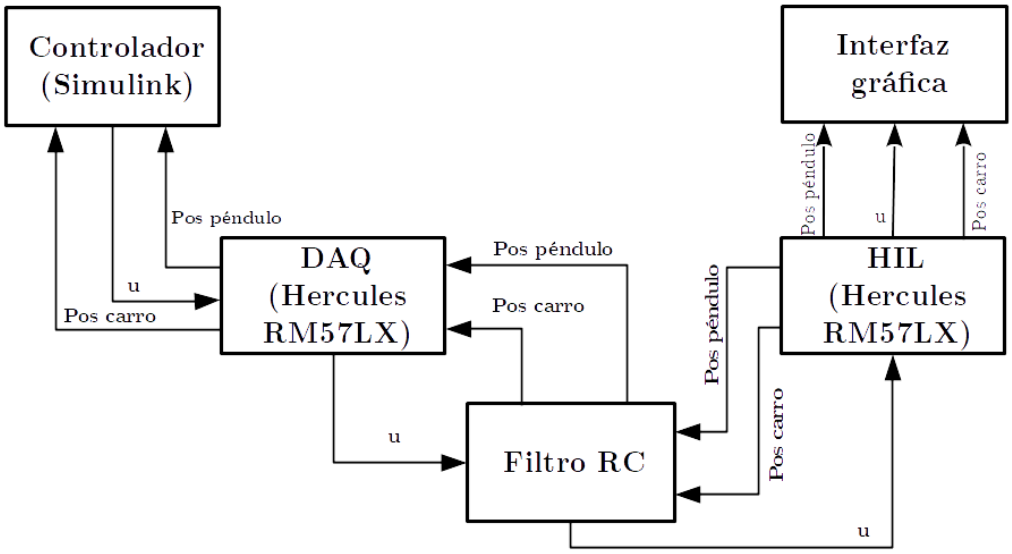

Figura 4.9: Flujo de datos entre el controlador y el simulador HIL

4.6.2. Comunicación entre Simulink y la tarjeta DAQ

Para transmitir y recibir datos entre Simulink y la Hercules RM57LX que se utiliza como DAQ, se conecta a una laptop mediante un cable USB-USB mini.

En MATLAB/Simulink se configuran los módulos, en la Figura 4.10, se presentan los bloques utilizados para leer datos de la DAQ. El objetivo de procesar los datos recibidos es conocer las cuatro variables de estado, posición del carro, posición del péndulo, velocidad del carro y velocidad del péndulo, con estas variables de entrada se diseña cualquier tipo de controlador. A continuación, se describen los módulos utilizados.

- ✓ Query Instrument: este módulo es utilizado para recibir datos a través del puerto serial, recibe una cadena de enteros.
- ✓ Math Function: separa la cadena en dos variables (posición del carro y la posición del péndulo).
- ✓ Voltaje A18, A7: son bloques que escalan las entradas a una salida equivalente al voltaje de operación de la tarjeta de adquisición de datos, esto es de 0V a 3,3V.

- ✓ Voltaje a radianes: convierte el voltaje de entrada a su equivalente en radianes, la misma función realiza el bloque Voltaje a metros.
- ✓ Derivador: deriva la señal de entrada para obtener la velocidad del carro y péndulo.

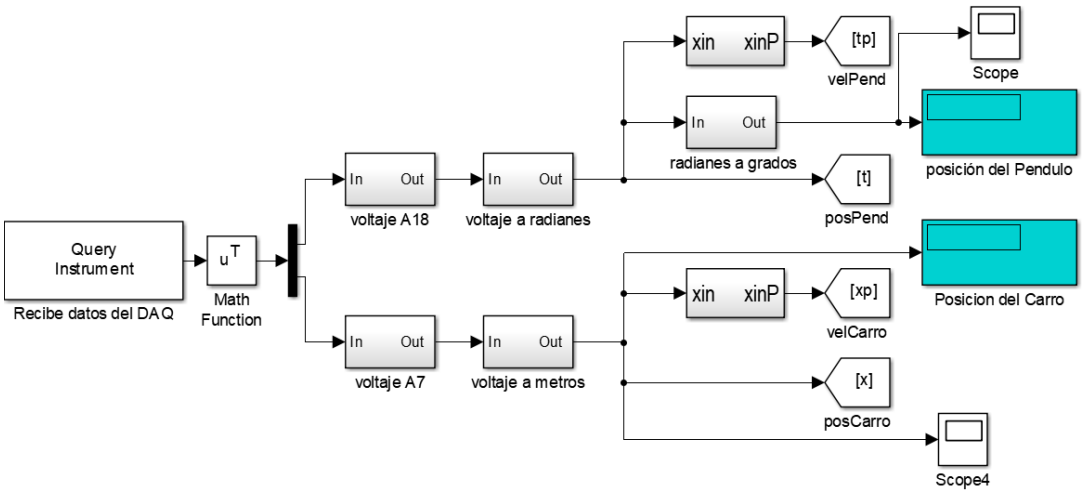

Figura 4.10: Bloques utilizados para recibir datos de la tarjeta DAQ.

Para verificar la conexión y funcionamiento entre el simulador HIL, la tarjeta DAQ y Simulink se grafica la señal de posición del péndulo, ésta se obtiene con base al flujo de datos de la Figura 4.9. Inicialmente y desde la interfaz gráfica se coloca el péndulo en la posición vertical superior, consecuentemente, en un pin de salida PWM de la tarjeta hercules se obtiene la señal analógica de la posición del péndulo, ésta pasa por un filtro pasa bajas RC, después ingresa a la tarjeta de adquisición de datos para finalmente mostrarse en MATLAB/Simulink. En la Figura 4.11 se muestra la señal analógica del péndulo graficada en Simulink, la señal de color azul es la señal ideal que debería replicarse en Simulink, sin embargo, debido al ruido provocada por la emulación de una salida analógica a través de un modulador de ancho de pulsos al filtro pasa bajas RC y demás ruido de la transmisión de la señal, se obtiene una señal demasiado ruidosa (señal de color morado). Para disminuir el ruido de la señal en la tarjeta de adquisición de datos se implementó un filtro digital (véase la sección 3.5).

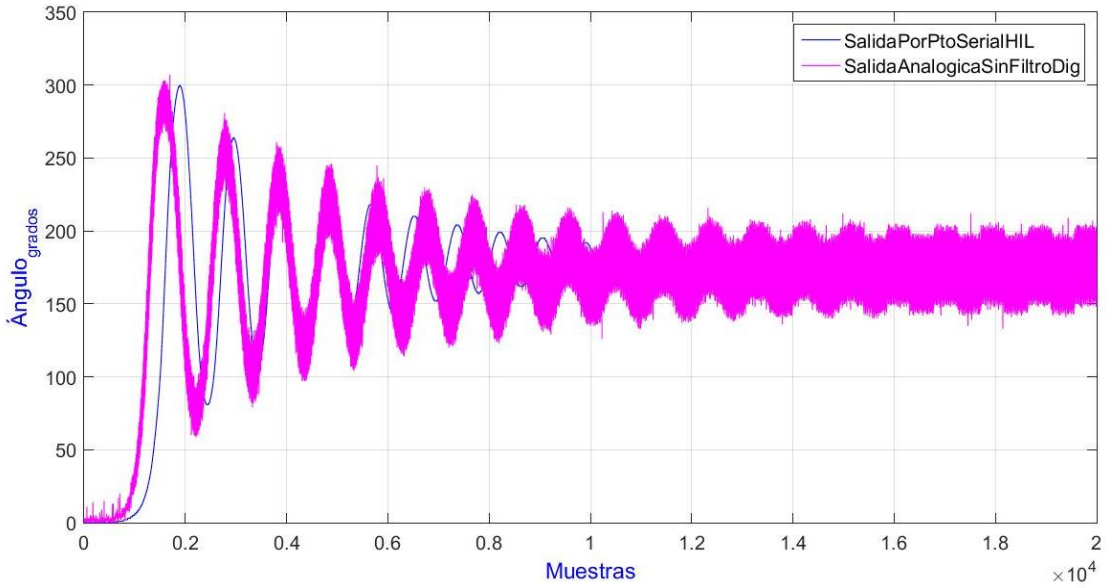

Figura 4.11: Señal analógica del péndulo sin filtro digital.

En la Figura 4.12 se observa la señal de la posición del péndulo después del filtro digital, las oscilaciones no coinciden totalmente con la señal ideal pero tiene menos ruido, de forma que la salida del simulador HIL tenga mayor exactitud y confiabilidad en los cálculos.

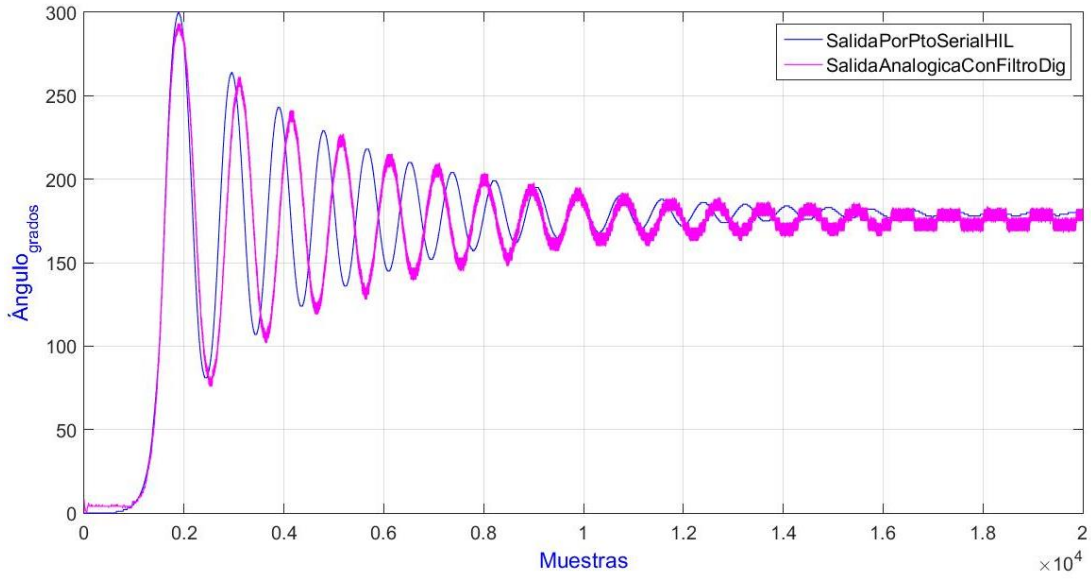

Figura 4.12: Señal analógica del péndulo después del filtro digital.

4.7. Control de levantamiento y estabilización del péndulo

El principal objetivo de un controlador aplicado al SCPI es mantener al péndulo en equilibrio en la posición vertical superior; es común que el usuario al experimentar con la planta coloque manualmente al péndulo en la posición deseada para posteriormente ejecutar el controlador; esto suele ser tedioso de hecho colocar y mantener al péndulo en la posición vertical superior sin vibraciones toma aproximadamente 5 segundos.

El control completo de este sistema consta de dos controladores, el primero llamado de auto-levante (Swing-up) y el segundo que es el de estabilización. Esto se consigue con el movimiento de izquierda a derecha del carro. Ambos controladores se diseñaron y probaron en MATLAB/Simulink.

4.7.1. Control de levantamiento del péndulo (Swing-up)

El objetivo del control de levantamiento del péndulo es hacer que el carro se mueva de izquierda a derecha para balancear al péndulo de forma que sea llevado a la posición vertical superior (punto de equilibrio inestable). Esta forma de balancear se hace por medio de aplicación de energía al carro, es necesario regular la energía dependiendo de la posición y velocidad del péndulo. Para ello se sigue la metodología mostrada por Astrom y Furuta [35], donde utiliza la ecuación de energía (4.2)

$$E = \frac{1}{2} J \left(\dot{\theta}\right)^2 + mgl\left(\cos\left(\theta\right) - 1\right) \tag{4.2}$$

Donde m es la masa del péndulo, J es el momento de inercia de la varilla del péndulo con respecto al pivote (punto de unión entre el carro y péndulo), l es la distancia del pivote al centro de masa del péndulo, $\boldsymbol{\theta}$ es el ángulo positivo, g es la aceleración de la gravedad, estas fuerzas y variables son mostradas en el diagrama de la Figura 1.2. La ley de control está dada en la ecuación (4.3) con la que se obtiene la energía deseada E_0, es decir la energía que toma cuando el péndulo esta su posición vertical superior:

$$u = k\left(E - E_0\right) sign(\dot{\theta}\cos\theta) \tag{4.3}$$

La función $sign()$ realiza el cambio de dirección de la señal de control, $E - E_0$ es la diferencia entre el error actual y la energía del péndulo en la posición deseada,

k es una constante de diseño, donde en la práctica es determinado por los niveles de ruido de la señal.

Con base en los parámetros de la Tabla 1, las ecuaciones (4.2) y (4.3), se implementa en MATLAB/Simulink el controlador de levantamiento del péndulo. En la Figura 4.13 se ven los bloques utilizados, las entradas necesarias son la posición del péndulo y su velocidad, en el subsistema Swing Up se codifican las ecuaciones antes mencionadas. Las constantes E_0 y k se variaron hasta lograr un acercamiento del péndulo al punto de equilibrio inestable, tomando en cuenta el ruido de entrada de la señal y que el controlador no considera la posición del carro. Las constantes finalmente tomaron los siguientes valores, $E_0 = 0.01$ y $k = 20$.

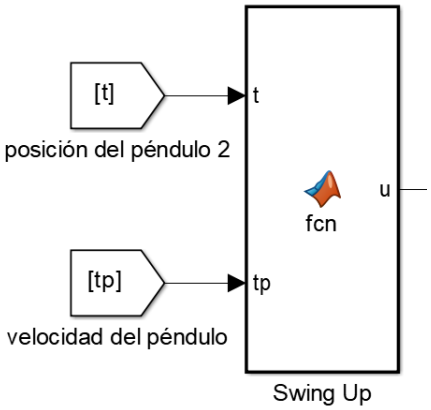

Figura 4.13: Control Swing-UP en MATLAB/Simulink

En la Figura 4.14 se muestra el código ingresado en el subsistema Swing Up, es decir, las ecuaciones (4.2) y (4.3).

```
function u = fcn(t,tp)
J=0.11E-4;
m=0.11;
l=0.3434;
g=9.81;
k=20;
E0=0.01;
u=k*(E-E0)*sign(tp*cos(t));
```

Figura 4.14: Código del subsistema Swing Up

4.7.2. Control de estabilización del péndulo

Se implementó un control de estabilización del péndulo usando retroalimentación de todos los estados. En el diagrama a bloques de la Figura 4.15 se muestra los bloques del controlador diseñado, donde x es el vector de estados es decir $x = [x, \dot{x}, \theta, \dot{\theta}]$ y k es el vector de ganancias $k = [k_0, k_1, k_2, k_3]$. Es importante mencionar que todos los estados son conocidos y que los estados deseados son cero, esto significa que la posición del carro deseada es cero (centro del riel), la posición angular del péndulo es cero (posición vertical superior), las velocidades lineal y angular son cero y en ese instante la entrada de control deseada es cero.

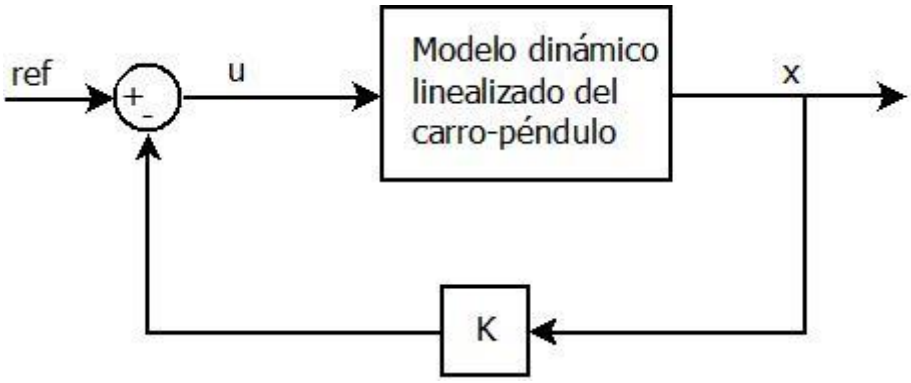

Figura 4.15: Diagrama a boques del control por realimentación de estados.

Para aplicar el control por realimentación de estados, se linealiza el modelo dinámico del carro-péndulo (véase la ecuación (1.57)) alrededor del punto de equilibrio, es decir, donde el péndulo es totalmente vertical ($\theta = 0$). Tomando los parámetros de la planta y haciendo los cálculos en MATLAB, se obtienen las matrices A y B que constituyen un sistema lineal de la forma (4.4).

$$\dot{x} = Ax + Bu \tag{4.4}$$

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & -0.1032 & -0.4418 & 0.0083 \\ 0 & 0 & 0 & 1 \\ 0 & 0.1467 & 14.5739 & -0.2753 \end{pmatrix} \tag{4.5}$$

$$B = \begin{pmatrix} 0 \\ 0.4398 \\ 0 \\ -0.6252 \end{pmatrix} \tag{4.6}$$

La ley de control de realimentación de estados se expresa como la siguiente ecuación:

$$u = -kx = -\left[k_3, k_2, k_1, k_0\right] * \left[x_1, x_2, x_3, x_4\right]^T \tag{4.7}$$

Sustituyendo en la ecuación (4.4)

$$\dot{x} = Ax - Bkx = \left(A - Bk\right)x \tag{4.8}$$

Con ayuda de MATLAB se verifica la controlabilidad del sistema, se calcula la determinante de la matriz para obtener los valores característicos y se iguala a un polinomio característico de cuarto orden de la forma $(s^2 + 2w_n sz + w_n^2)^2$ donde w_n es la frecuencia natural y z es el coeficiente de amortiguamiento, asignando los valores $wn = 100$ y $z = 0.707$ se resuelve el sistema de ecuaciones obteniendo el vector de ganancias: $k_3 = 8.1967$, $k_2 = 1.4249$, $k_1 = 2.2081 \times 10^3$ y $k_0 = 19.9702$.

Este controlador de posición se diseña en Simulink con base a las variables de estado leídas y el vector de ganancias k. En la Figura 4.16 se aprecia el controlador diseñado en MATLAB/Simulink, las variables de entrada $x,\ xp,\ t,\ tp$, que son leídas desde la DAQ (ver Figura 4.10) y generadas por el modelo embebido en la tarjeta Hercules. El subsistema llamado Control pos retro recibe las variables y resuelve la ecuación (4.9), generando la señal de control $\boldsymbol{u}$.

$$u = -k_3 x - k_2 xp - k_1 t - k_0 tp \tag{4.9}$$

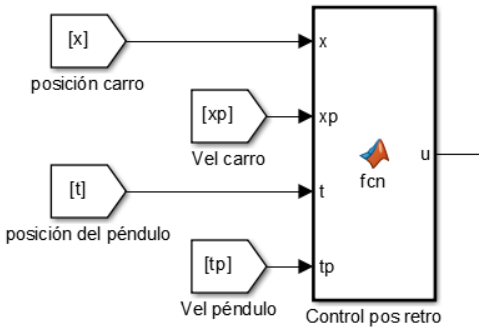

Figura 4.16: Control de posición en MATLAB/Simulink

En la Figura 4.17 se muestra la función que calcula el control de realimentación de estados, es decir la ecuación (4.9).

```
function u = fcn(x,xp,t,tp)
k0= -19.9702;
k1=-2.2081e03;
k2= -1.4249;
k3=-8.1967;
u=-k3*x-k2*xp-k1*t-k0*tp;
```

Figura 4.17: Función del controlador

4.8. Integración de los controladores

Una vez diseñado los controladores, se integran como lo muestra la Figura 4.18, la forma de adquisición de la posición, velocidad; del carro y péndulo se describió en la sección 4.6.2. A continuación, se describen los bloques faltantes.

- ✓ Interval test: en este bloque se define un intervalo en radianes (± 0.523599 rad), recibe la posición del péndulo, la salida es TRUE si el ángulo de entrada está en el intervalo definido en caso contrario es FALSE.
- ✓ Switch: conmuta entre dos entradas dependiendo de una entrada de control si es TRUE deja pasar la primera señal en caso contrario la segunda señal, la operación es parecida a la de un relay.
- ✓ Saturación: se define el mínimo y máximo voltaje de operación, en este caso $\pm 55\ Volts$.
- ✓ Escalamiento: debido a que los datos se enviaran por puerto serie se escala el voltaje de entrada ± 55 a un rango de voltaje de 0 a 255.
- ✓ To Instrument: envía por puerto serie la señal de control a la tarjeta de adquisición de datos, a su vez la envía al emulador del SCPI.

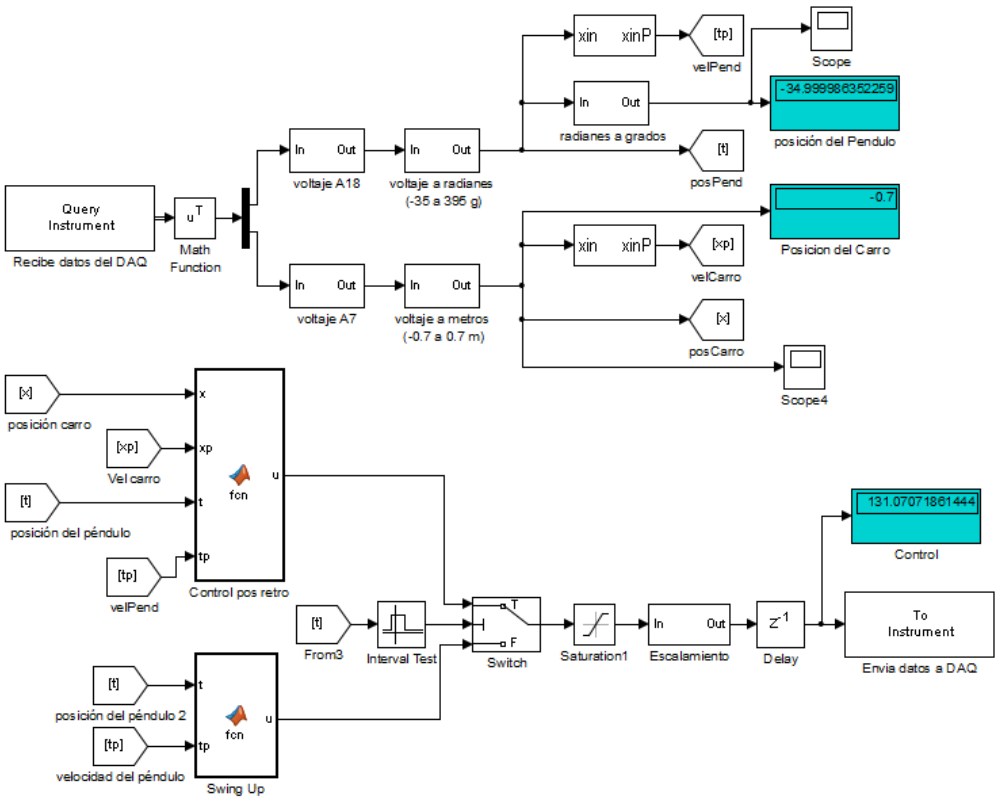

Figura 4.18: Conmutación entre el control de levantamiento y estabilización

Los controladores en conjunto funcionan de la siguiente manera, primero se leen la posición del carro y péndulo, con el bloque derivador se obtienen sus respectivas velocidades. Los controladores se ejecutan simultáneamente, en efecto cada una tiene la señal de control u a la disposición, el bloque Interval Test lee la posición del péndulo, si ésta no está en su rango de $\pm 30°$ es decir lo mas cercano a su posición vertical superior $(0°)$ conmuta a la señal de control de balanceo, una vez que el péndulo este llegando a $0°$ conmuta a la señal de estabilización del péndulo. Este controlador está constantemente recibiendo las posiciones que genera el emulador del CPI y enviando la señal de control al mismo.

4.9. Respuesta de control con el péndulo inicial en 0°

Se prueba el controlador con una posición inicial del péndulo de 0 grados, es decir se lleva manualmente al péndulo de la posición vertical inferior a la superior.

En la Figura 4.19 se muestra la forma de conexión del controlador, filtro y demás componentes para las pruebas realizadas. El controlador de Matlab/Simulink

y la interfaz grafica se ejecutaron en computadores distintas, todos los componentes de conexión se describen en la sección 4.6.1.

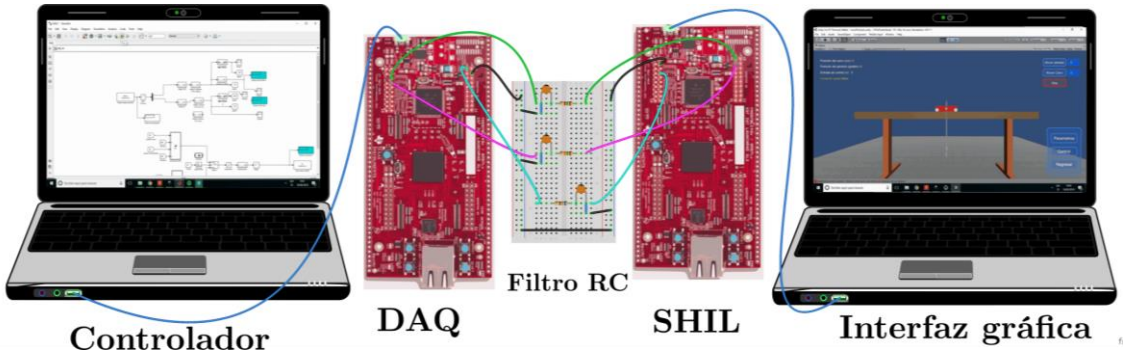

Figura 4.19: Conexión entre el contralor y el simulador HIL.

En la Figura 4.20 se visualizan las gráficas en respuesta al algoritmo de control, estos datos se guardaron en archivos de Excel usando la graficadora del ESCPI, posteriormente con los datos se editaron las graficas en Matlab. La figura muestra tres graficas, la primera (de arriba hacia abajo) es la posicion angular en grados que toma el pendulo en un tiempo t (segundos), la segunda grafica muestra la posicion o desplazamiento en cm del carro en el riel, la tercera y ultima grafica muestra el voltaje de control que se le aplica al ESCPI.

Al iniciar, el péndulo se encuentra en la posición angular de $180°$, al término de dos segundos el péndulo es llevado a $0°$ (en otras palabras, se lleva el péndulo manualmente a la posición de $0°$), inmediatamente el controlador mantiene al péndulo en esa posición o lo más cercano posible. En la misma figura se muestra la gráfica de posición del carro y la señal de control aplicada. Debido a que antes de ejecutar el controlador, la posición del péndulo es $0°$, en esta prueba solo se ejecuta el algoritmo de estabilización.

En la primera grafica se observa como el pendulo se mantiene en la posicion de $0°$ durante 12 segundos aproximadamente, en algunas pruebas el tiempo de estabilización es mayor, en otras tan solo 5 segundos, esto dependiento de la posicion del carro sobre el riel y la velocidad angular que lleva el péndulo. En promedio se mantiene 14 segundos estable.

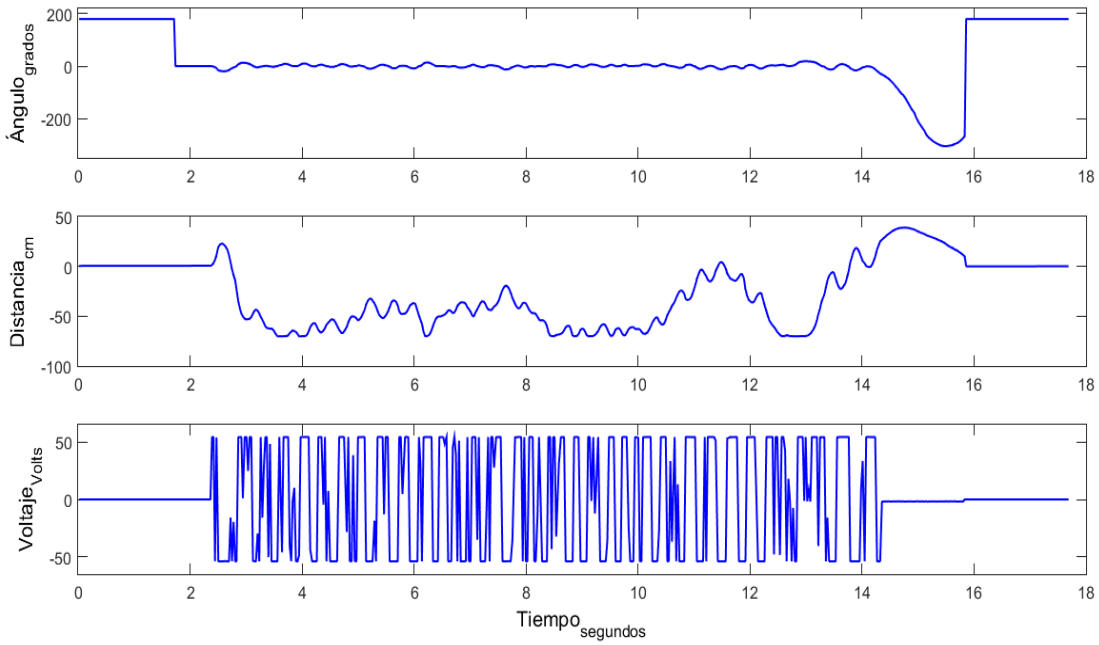

Figura 4.20: Respuesta de control con posición inicial del péndulo en $0°$

4.10. Respuesta de control con auto-balanceo del CPI.

En esta prueba, el péndulo inicia en la posición de $180°$. Inicialmente se ejecuta el controlador de levantamiento, una vez que el péndulo este en el rango de $\pm 15°$ conmuta y ejecuta el controlador de estabilización. Como se indica en la Figura 4.21 el péndulo comienza a oscilar y a los 12 segundos, cuando el pendulo pasa o se acerca a los $0°$ se ejecuta el control de estabilización del pendulo.

La Figura 4.21 muestra tres graficas, la primera (de arriba hacia abajo) es la posicion angular, la segunda grafica muestra el desplazamiento del carro en el riel y ultima grafica muestra el voltaje de control que se le aplica al ESCPI.

Estas graficas se obtuvieron después de algunos intentos, pues no todas las veces lograba estabilizar al péndulo. Con base a la experimentación se determinó que la efectividad del controlador depende de las ganancias asignadas, así como la posición del inicial del carro y la velocidad del péndulo al llegar a la posición vertical superior.

El tiempo de estabilizacion del péndulo es de 22 segundos aproximadamente, suficientes para determinar que el proceso o los datos enviados desde el controlador al simulador HIL funciona correctamente.

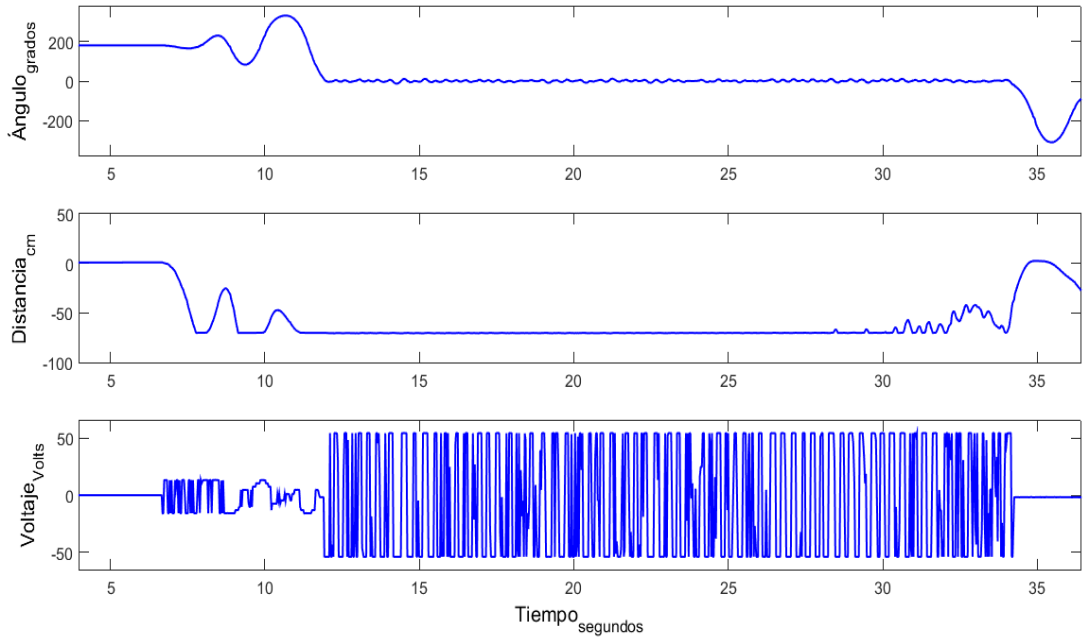

Figura 4.21: Respuesta de control con péndulo inicial en 180°

4.11. Simulador HIL conectado a un controlador embebido en hardware.

Para mostrar la versatilidad del ESCPI, se prueba el simulador usando un controlador embebido en hardware. Se diseña el controlador con base a la ecuación (4.9) de la sección 4.7.2, se utiliza la tarjeta Hercules Launchpad RM57Lx para embeber el algoritmo de control de estabilización del péndulo. El diagrama de conexión se muestra en la Figura 4.22, del simulador HIL salen 4 señales analógicas, la posición del carro y péndulo, la velocidad del carro y péndulo, pasan por un filtro pasa bajas RC de forma que estas señales entren a la tarjeta controladora procese la señal y devuelva la señal de control.

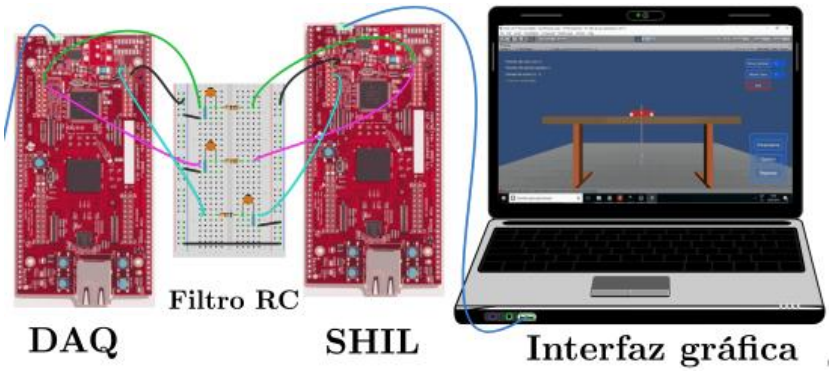

Figura 4.22: Conexión entre el controlador en hardware y el SHIL

En la Figura 4.23 se muestra la gráfica de comportamiento del péndulo, inicialmente el péndulo está en 180 grados, enseguida se coloca en 0 grados y el control lo mantiene oscilando en esa posición hasta que se detenga la simulación, el péndulo se estabiliza por un tiempo aproximado de 30 segundos. Es importante mencionar que esta señal se tomó de los datos obtenidos de la interfaz gráfica del simulador HIL, posteriormente se utilizó MATLAB para graficar la señal.

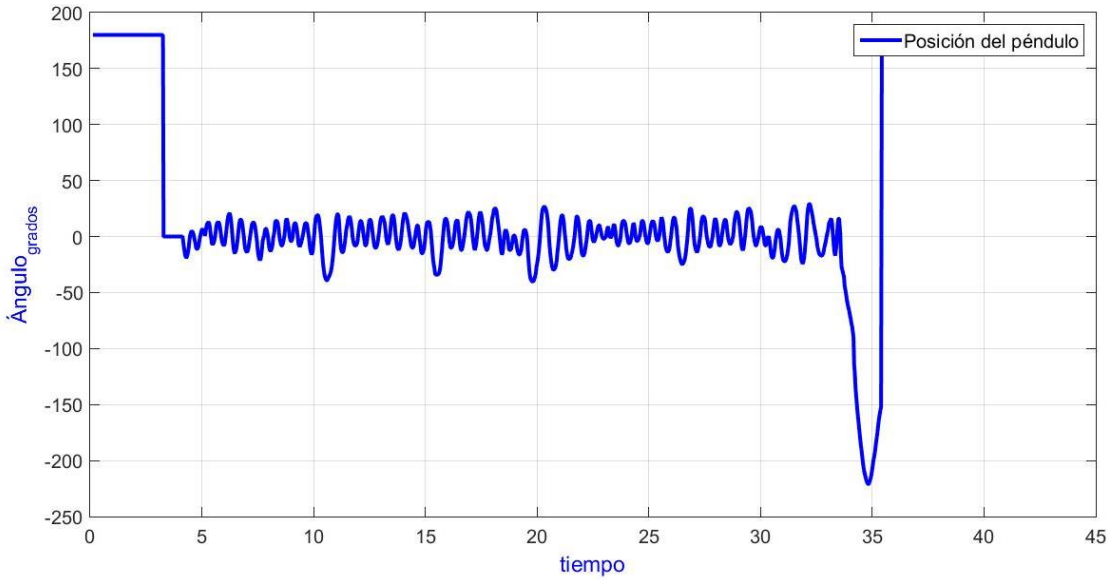

Figura 4.23: Comportamiento del péndulo invertido

Capítulo 5
Conclusiones y trabajos a futuro

Se desarrolló un emulador del sistema carro-péndulo invertido (ESCPI) con su respectiva interfaz gráfica elaborada en Unity y LabVIEW, además un circuito de acondicionamiento de señales (CAS) con amplificadores operacionales básicos.

Se usó la tarjeta Hercules RM57Lx para embeber el modelo dinámico del CPI, la configuración de ésta es algo diferente a las demás tarjetas de Texas Instruments puesto que necesita de un software más para configurar los módulos GIO, ADC, PWM, etc., lo que al principio dificulta su configuración, pero finalmente resulta más fácil y cómodo que configurar los módulos por código (programación).

Es importante tener en cuenta la frecuencia con la que se genera la señal PWM, debido que a mayor frecuencia se tiene menos ruido a la salida del filtro RC, pero con baja resolución, es decir, se tienen pocos valores diferentes de ciclo de trabajo y con esto perder valores de ángulos o de metros del CPI. A menor frecuencia se tiene mayor resolución, pero con mucho más ruido a la salida.

Si bien la tarjeta Hercules RM57Lx tiene gran rendimiento, que no tiene el módulo convertidor digital analógico (DAC) presenta gran desventaja para este proyecto. El ruido que se genera al transmitir los datos de la tarjeta al controlador lo hace menos preciso, en otras palabras, puede no coincidir la señal de control simulada en Matlab a la generada en el simulador HIL.

La interfaz gráfica Unity fue una excelente elección para desarrollar la interfaz gráfica, los gráficos ofrecen alta calidad visual, la emulación se ve en tiempo real, además ofrece la posibilidad de ejecutar el software en un teléfono inteligente

(Smartphone), Tablet, OS Mac, etc. En LabVIEW se desarrolló una interfaz gráfica básica, para visualización de las señales.

Las respuestas de los controladores cumplieron con su objetivo, balancear al carro y mantener al péndulo estable, la efectividad depende de la posición inicial del carro y la velocidad con la que el péndulo llegue a la posición vertical superior.

Como trabajo a futuro se proponen las siguientes mejoras:

Agregar un convertidor digital analógico (DAC) para descartar problemas de ruido en la señal de salida o bien cambiar a una tarjeta que ya cuente con este módulo.

Además de transmitir datos por puerto serie, implementar un protocolo de comunicación WiFi, Bluetooth, de tal forma que la interfaz gráfica se pueda ejecutar en un teléfono inteligente, Tablet, etc.

Probar los controladores con otras tarjetas de adquisición de datos.

Diseñar el controlador en LabVIEW y comparar la velocidad de respuesta en cuanto a transmisión de datos.

Para obtener las velocidades se usó un módulo derivador de Simulink, por lo cual sería conveniente probar nuevos esquemas par el cálculo de derivadas con señales o diseñar dos observadores de estado (velocidad del péndulo y la velocidad del carro).

Bibliografía

[1] S. Usenmez, U. Yaman, M. Dolen, and A. B. Koku, "A new hardware-in-the-loop simulator for control engineering education," *IEEE Glob. Eng. Educ. Conf. EDUCON*. April, pp. 1–8, 2014.

[2] J. C. Martinez and J. Andrade, "Implementación de controladores en Sistemas Retroalimentados usando electrónica embebida y simulación Hardware In The Loop," Universidad Tecnológica de Pereira, Pereira, 2013.

[3] C. Washington and S. Delgado, "Improve Design Efficiency and Test Capabilities with HIL Siimulation," *IEEE Autotestcon*. September, pp. 8–11, 2008.

[4] D. López, "Entorno de simulación Hardware-In-the-Loop para estudios de tolerancia a fallos en sistemas electrónicos complejos," Universidad de Sevilla, 2015.

[5] P. Feng, X. Dingyu, C. Dali, and C. Jianjiang, "Design and Implementation of Rotary Inverted Pendulum Motion Control Hardware-In-The-Loop Simulation Platform," pp. 2328–2333, 2010.

[6] A. Turnau, A. Korytowski, and M. Szymkat, "Time optimal for the pendulum-cart system in real-time," pp. 1249–1254, 1999.

[7] K. J. Åström and K. Furuta, "Swinging up a pendulum by energy control," *Automatica*, vol. 36, no. 2, pp. 287–295, 2000.

[8] I. Fantoni, R. Lozano, and S. Sinha, *Non-linear Control for Underactuated Mechanical Systems*, vol. 55, no. 4. Londres: Springer, 2002.

[9] S. Subramanian, T. George, and A. Thondiyath, "Hardware-in-The-Loop verification for 3D obstacle avoidance algorithm of an underactuated flat-fish type AUV," *2012 IEEE Int. Conf. Robot. Biomimetics, ROBIO 2012 - Conf. Dig.*, pp. 545–550, 2012.

[10] D. Lee, H. Lim, H. J. Kim, Y. Kim, and K. J. Seong, "Adaptive Image-Based Visual Servoing for an Underactuated Quadrotor System," *J. Guid. Control. Dyn.*, vol. 35, no. 4, pp. 1335–1353, 2012.

[11] J. Santana, "Diseño e implementación de un emulador de plantas SISO analógicas," Escuela Politécnica del Ejército, 2006.

[12] T. Instruments, "Hercules RM57Lx LaunchPad Development Kit," 2016. [Online]. Available: http://www.ti.com/tool/LAUNCHXL2-RM57L. [Accessed: 09-May-2016].

[13] Applied Dynamics International, "Solutions in real time," 2016. [Online]. Available: http://www.adi.com/technology/tech-apps/what-is-hardware-in-the-loop-simulation/. [Accessed: 01-Jan-2016].

[14] F. Instruments, “Products: Educational Equipment,” 2017. [Online]. Available: http://www.feedbackinstruments.com/products/education/control_instrumentation/digital_pendulum. [Accessed: 04-May-2017].

[15] F. Instruments, “Digital Pendulum System,” vol. 44, no. 1160. FI Ltd, England, p. 18, 1892.

[16] K. Fu, R. Gonzalez, and C. Lee, *Robotics*. McGraw-Hill, 1987.

[17] R. Kelly and V. Santibáñez, *Control de Movimiento de Robots Manipuladores*. Madrid: Prentice Hall, 2003.

[18] B. . Kuo, *Sistemas de control automático*, 7th ed. Mexico: Prentice Hall, 1995.

[19] C. Platero, “Modelado matemático de los sistemas dinámicos,” *Universidad Politécnica de Madrid*. pp. 73–119, 2008.

[20] D. H. D´Arthenay, “Desarrollo de un simulador de procesos industriales bajo configuración Hardware-in-the-Loop para la práctica-enseñanza de control lógico y regulatorio mediante un PLC,” Universidad Nacional de Colombia, 2015.

[21] P. Blanchard, R. Devaney, and G. Hall, *Ecuaciones Diferenciales*. International Thomson Editores, 1999.

[22] S. A. R. Gonzalez and G. J. Krause, “Plataforma De Simulación Distribuida Con Hardware In The Loop en tiempo real,” *Asoc. Argentina Mecánica Comput.*, vol. XXXII, pp. 1877–1889, 2013.

[23] D. Ibrahim, *Microcontroller Based Applied Digital Control*. Chichester, UK: John Wiley & Sons, Ltd, 2006.

[24] T. Instruments, “TI LaunchPad development kits,” 2016. [Online]. Available: http://www.ti.com/lsds/ti/tools-software/launchpads/launchpads.page#tabs. [Accessed: 21-Oct-2016].

[25] J. Travis and J. Kring, *LabVIEW For Everyone: Graphical Programming Made Easy and Fun*, 3rd ed. Prentice Hall, 2006.

[26] N. Instruments, “NI LabVIEW Data Visualization and User Interface Design,” 2016. [Online]. Available: http://www.ni.com/white-paper/14557/en/. [Accessed: 10-Sep-2016].

[27] N. Arrioja Landa Cosio, *Unity*, Primera. Buenos Aires: Fox Andina, 2013.

[28] I. Ouazzani, “Manual de creación de videojuego con Unity 3D,” Universidad Carlos III de Madrid, 2012.

[29] U. Technologies, “Unity Documentation,” 2017. [Online]. Available: https://docs.unity3d.com/es/current/Manual/UISystem.html.

[30] D. Ashlock and A. Warren, “Guía de acondicionamiento de señales para ingenieros.” p. 14, 2015.

[31] T. Instruments, “How to Create A HALCoGen-Based Project For CCS,” Dallas, 2016.

[32] T. Instruments, "Technical Reference Manual," Dallas, 2010.

[33] F. E. Valdés-Pérez and R. Pallàs-Areny, *Microcontroladores: Fundamentos y Aplicaciones con PIC*. 2007.

[34] N. Coding, "Grapher - Graph, Replay, Log," *Asset store*. [Online]. Available: https://assetstore.unity.com/publishers/14460. [Accessed: 15-Feb-2018].

[35] K. J. Åström and K. Furuta, "Swinging Up a Pendulum by Energy Control," pp. 1–15, 1996.

Apéndice A. Código de programación del ESCPI

Autor: Marcelino Martínez Aragón.

Fecha: 25 de junio de 2018.

Código de la función main.

```
/* Include Files */
/* USER CODE BEGIN (0) */
#include "HL_het.h" //Libreria para PWM
#include "HL_sci.h"
#include "HL_adc.h"
#include "HL_rti.h"
#include "HL_gio.h"
#include "stdlib.h"
#include "arm_math.h"
#include "HL_sys_common.h"
/* USER CODE END */

/* USER CODE BEGIN (1) */
//Variables globales
Double param[12]={0.000011,0.02616,1.12,0.11,0.3434,0.05,0.007,
3.9,9.81,0.0535,0.0136,0.00005};
double x1k,x2k,x3k,x4k,u=0.001,h=0.001;
double x1a=0.001,x2a=0.001,x3a=0.001,x4a=0.001;
double a1,a2,a3,a4,a5,b1;
double const Pii=3.1415926;
static unsigned char recibe;
/* USER CODE END */

/* USER CODE BEGIN (2) */
void GuardarParametros();
void asignaParametros();
void posicionar();
/* USER CODE END */

void main(void)
{
/* USER CODE BEGIN (3) */
    sciInit(); //Inicializa el módulo SCI (Comunicacion Serial)
    adcInit(); //Inicializa el módulo ADC
    hetInit();//Inicializa modulo PWM
```

```
    rtiInit(); //Inicializa módulo de interrupcion RTI
    gioInit(); //Inicializa Pines de E/S

    //Declaración variables locales
    unsigned int TamPosP=0,TamPosC=0,TamU=0;
    unsigned char StrPosP[8],StrPosC[8],opc='z',StrU[8];
    uint32 pwmPosC,pwmPosP;
    bool flag1=0;

    asignaParametros(); //Inicializa los parámetros del CPI
    rtiEnableNotification(rtiREG1,rtiNOTIFICATION_COMPARE0); //activa la
interrupción del comparador 0
    _enable_IRQ();
    rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0); //inicializa el contador del
bloque 0

    while(1) // Ciclo infinito
    {
       sciReceive(sciREG1,1,(unsigned char *)&opc); //Recibe un carácter de la
GUI
       if (opc=='a'&&flag1==0){ //Reset del CPI
            u=0.001;  //Variable de control = 0
            x4a=0.001; //Velocidad del péndulo = 0
            x3a=3.1416;  // Posición péndulo vertical inferior = 180 grados
            x2a=0.001; //Velocidad del carro = 0
            x1a=0.001; //Posición del péndulo = 0
            flag1=1;  //Bandera de reseteo
        }
       else if(opc=='b' )//Control interno (estabilización del péndulo)
           flag1=0;
       else if(opc=='c'){//Control externo (entrada analógica)
           flag1=0;
       else if(opc=='d') // Modifica los parámetros del modelo dinámico del CPI
           GuardarParametros(); //Actualiza los parámetros recibidos de la GUI
       else if(opc=='e'){ // Cambia la posición del péndulo
           posicionar();
           x3a=nume*Pii/180;
           x4a=0.001;
           x2a=0.001;
        }
       else if(opc=='f'){ //Cambia la posición del carro
            posicionar();
            x1a=nume/100;
            x2a=0.001;
        }

        /*Tamaño de las variables a enviar por puerto serie */
        TamPosP = ltoa(PosP,(char *)StrPosP);
        TamPosC = ltoa(PosC,(char *)StrPosC);
        TamU = ltoa(u*100,(char *)StrU);

        /*Envía la posición del carro, posición del péndulo y la variable de
control
         por puerto seria, separados por una coma*/
        sciSend(sciREG1, TamPosC, StrPosC);
```

```
        sciSend(sciREG1, 1, (unsigned char *)",");
        sciSend(sciREG1, TamPosP, StrPosP);
        sciSend(sciREG1, 1, (unsigned char *)",");
        sciSend(sciREG1, TamU, StrU);
        sciSend(sciREG1, 2, (unsigned char *)"\r\n");

        /*Salidas analógicas (PWM) de la posición del carro
         y posición del péndulo*/
        pwmStart(hetRAM1, pwm0);
        pwmPosC=71.42857*x1k+50; //Calculo del porcentaje PWM
        pwmSetDuty(hetRAM1, pwm0, pwmPosC);

        pwmStart(hetRAM1, pwm1);
        pwmPosP=0.2325581395*PosP+8.139534884;
        pwmSetDuty(hetRAM1, pwm1, pwmPosP);

    }

/* USER CODE END */
}

/* USER CODE BEGIN (4) */
/*Funcion que recibe datos de la GUI y actualiza las
 posiciones del carro o pendulo del CPI*/
void posicionar(){
    double temp=0,num;
    bool mult=true,flag;
    sciReceive(sciREG1,1,(unsigned char *)&recibe);//Recibe un caracter de la
GUI
    while(recibe!='#')// El caracter # signidica fin de la cadena
    {
        num=recibe-'0'; //Convierte de ascci a flotante
        if (mult==true)
        {
            if(recibe=='-')
                flag=1;
            else{
                temp=num;
                mult=false;
            }
        }
        else
            temp=temp*10+num;
        sciReceive(sciREG1,1,(unsigned char *)&recibe);
    }

    if (flag==1)
        nume=0-temp;
    else
        nume=temp;
}

/* Funcion para recibir los parametros del CPI, provenientes
 de la GUI y actualizarlos */
void GuardarParametros()
```

```
{
   double temp=0,num;
   bool mult=true;
   int i=1,j=0;
   sciReceive(sciREG1,1,(unsigned char *)&recibe);
   while(recibe!='#')
    {
        while(recibe!=';' && recibe!='#')
        {
            num=recibe-'0'; //convierte de ascci a flotante
            if (recibe=='.' || i>1)
            {
                if(i>1)
                    temp=temp+(num/i);
                i=i*10;
            }
            else
            {
                if (mult==true)
                {
                    temp=num;
                    mult=false;
                }
                else
                    temp=temp*10+num;
            }
            sciReceive(sciREG1,1,(unsigned char *)&recibe);
        }
        param[j]=temp;
        j++;
        i=1;
        mult=true;
        temp=0;

       if (recibe !='#') //no es fin de la cadena
           sciReceive(sciREG1,1,(unsigned char *)&recibe);
       if(recibe=='#') // ya termino de transmitir todos los datos
           asignaParametros();
   }
}

/*Funcion para actualizar los parametros del CPI en tiempo de ejecucion*/
void asignaParametros()
{
    a1=(param[0]/(param[1]*param[1]))+(param[2]+param[3]);
    a2=param[9]*param[9]/(param[7]*param[1]*param[1])+
(param[11]/(param[1]*param[1]))+param[5];
    a3=param[3]*param[4];
    a4=(param[3]*param[4]*param[4])+param[10];
    a5=param[3]*param[8]*param[4];
    b1=param[9]/(param[1]*param[7]);
}

/* USER CODE END */
```

Código de la interrupción.

```
/* Include Files */
#include "HL_esm.h"
#include "HL_adc.h"
#include "HL_gio.h"
#include "HL_sci.h"
#include "HL_het.h"
#include "HL_rti.h"
#include "HL_epc.h"
#include "HL_sys_dma.h"

/* USER CODE BEGIN (0) */
#include  "arm_math.h"
extern double x1k,x2k,x3k,x4k,x1a,x2a,x3a,x4a,h;
extern double a1,a2,a3,a4,a5,b1;
//Ganancias para el control de estabilización del péndulo (retro)
double k0=-19.9702, k1=-2.2081E3, k2=-1.4249,k3=-8.1967 ;
extern double param[12],u,PosP,PosC,yU,VelC;
extern double const Pii;
adcData_t adc_data[1];
extern unsigned char opc;
int EntDig=0, promU=0;
int i,n_m=45; //Número de muestras para el promedio
double sumaU=0,ast;
/* USER CODE END */

void rtiNotification(rtiBASE_t *rtiREG, uint32 notification)
{
/* USER CODE BEGIN (12) */
    /*Si el carácter recibido es b, ejecutar el control
     de estabilización del péndulo*/
    if (opc=='b')
    {
        if(x3k<=0.436332 && x3k>=-0.436332)
            u=-k3*x1a-k2*x2a-k1*x3a-k0*x4a;
        if (u>55)
            u=55;
        if (u<-55)
            u=-55;
    }

    /*Si el carácter recibido es c, ejecuta el control externo
     (entrada analógica) */
    else if (opc=='c')
    {
        for (i=0; i<n_m;i++)
        {
            adcStartConversion(adcREG1, 1U); //Comienza la conversión ADC, se
está usando el ADC1 del micro y el grupo 1
            while(!adcIsConversionComplete(adcREG1, 1U)); // Espera a que
termine la conversion
```

```
            adcGetData(adcREG1, 1U, &adc_data[0]); // Obtiene el valor producto
de la conversión y lo almacena en un apuntador
            EntDig=adc_data[0].value; //Almacena el valor digital
            sumaU=sumaU+EntDig; //Acumulador de las entradas, para después
promediar la entrada de control
        }
        promU=sumaU/n_m;
        u=0.02686202686*promU-55;
        sumaU=0;
    }

    /*Ecuaciones del modelo dinámico del CPI*/
    ast=a4-((a3*a3)/a1)*(arm_cos_f32(x3a)*arm_cos_f32(x3a));
    x1k=x1a+h*(x2a);
    x4k=x4a+h*(-(param[6]/ast)*x4a-((a3*a3)/(a1*ast))*(x4a*x4a)*arm_sin_f32(x3a)
*arm_cos_f32(x3a)+((a5*arm_sin_f32(x3a))/ast)+((a2*a3)/(a1*ast))*arm_cos_f32(x3a
)*x2a-((a3*b1)/(a1*ast))*arm_cos_f32(x3a)*u);
    x2k=x2a+h*(-(a2/a1)*x2a-(a3/a1)*x4k*arm_cos_f32(x3a)+(a3/a1)*(x4a*x4a)*
arm_sin_f32(x3a)+(b1/a1)*u);
    x3k=x3a+h*(x4a);

    //actualizar estados anteriores
    x1a=x1k;
    x2a=x2k;
    x3a=x3k;
    x4a=x4k;

    //Convertir las posiciones en grados y milímetros respectivamente.
    PosP=x3k*(180/Pii); //radianes a grados
    PosC=x1k*1000; //metros a milímetros

/* USER CODE END */
```

Apéndice B. Código de programación de la interfaz gráfica en Unity

Autor: Marcelino Martínez Aragón.

Fecha: 25 de junio de 2018.

Script InicializarPuerto

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System.IO.Ports;

public class InicializarPuerto : MonoBehaviour {

    public InputField SP;
    public InputField BR,DB;
    public string SPNum;
    public int BRNum;
    public int DBNum;
    public static InicializarPuerto iniPuerto;

    /* Mantiene activo el puerto entre las dos
    escenas*/
    void Awake(){
        DontDestroyOnLoad (gameObject);
        if (iniPuerto == null) {
            iniPuerto = this;
            DontDestroyOnLoad (gameObject);
        }
        else if (iniPuerto != this) {
            Destroy (gameObject);
        }
    }

    //Lee los parámetros del puerto serial
    void Start() {
        SP.text = "" + SPNum;
        BR.text = "" + BRNum;
        DB.text = "" + DBNum;
    }
    //Actualiza los parámetros de la comunicación serial
    public void EnviarPuertoCom(){
        SPNum = SP.text;
```

```
        int.TryParse(BR.text, out BRNum);
        int.TryParse(DB.text, out DBNum);
        SP.text = "" + SPNum;
        BR.text = "" + BRNum;
        DB.text = "" + DBNum;
    }

    public void Reset(){
        Application.LoadLevel (0);
    }

}
```

Script Principal

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.IO.Ports;
using UnityEngine.UI;

public class Tras_Rot : MonoBehaviour {

    public Text TxtSerialPort;
    public float PosPendulo=0; //float
    public float PosCarro=0;
    public float control=0;//float
    public Text PP;
    public Text PC;
    public Text Ctrl;

    private string opc="z";
    private GameObject CP;
    private GameObject Pendulo;
    private string PuertoCom;
    private int DataBits,BaudRate;
    private float posCarro,posPend; //control
    private string[] values;
    private float ang;
    public SerialPort sp= new SerialPort();

    Vector3 tempPosCarro;

    /*Código que se ejecuta solo una vez
    Inicializa el puerto de comunicación serial*/
    void Start (){
        GameObject go = GameObject.Find ("GOPuertoSerial");
        InicializarPuerto iniPuerto = go.GetComponent <InicializarPuerto> ();
        PuertoCom = iniPuerto.SPNum;
        BaudRate = iniPuerto.BRNum;
        DataBits = iniPuerto.DBNum;
        sp.PortName = PuertoCom;
        sp.BaudRate = BaudRate;
        try{
```

```
            if (sp.IsOpen == false){
                sp.Open ();
                sp.ReadTimeout = 1;
                sp.Parity = Parity.None;
                sp.DataBits=DataBits;
                sp.StopBits = StopBits.Two;
            }
            if (sp.IsOpen == true){
                TxtSerialPort.text = "Conexión serial exitosa";
            }
        }
        catch(System.Exception)
        {
            TxtSerialPort.text = "Conexión serial fallida";
        }
        CP = GameObject.Find ("GameObjectCP");
        Pendulo = GameObject.Find ("GameObjectPend");
        //Debug.Log (sp.PortName);
    }

    //Actualiza en cada frame
    void Update (){
        if (sp.IsOpen) {
            try {
                 sp.Write (opc);
                //float pos=int.Parse(sp.ReadLine());
                 values = sp.ReadLine ().Split (','); //Lee las posiciones recib
idas desde el MCU
                 posCarro = (float.Parse (values [0]));// Separa los valores sep
arados por coma
                 posPend = (float.Parse (values [1]));
                 control = (float.Parse (values [2]));

                posCarro = posCarro / 10; // la posicion llega en mm, lo convier
te en cm.
                control=control/100; //Escala el valor de control
                MoveObject (posPend, posCarro);
                Ctrl.text = "" + control;
                //Debug.Log (opc);
            } catch (System.Exception) {
            }
        }
        else{
            TxtSerialPort.text = "Conexión serial fallida";
            TxtSerialPort.color = new Color32(96, 253, 0, 255); //color verde
        }
    }

    //Actualiza la posición del carro y del péndulo.
    //Da movimiento al CPI.
    void MoveObject(float angulo, float dist)
    {
        ang = 180 - angulo;
        Pendulo.transform.rotation=Quaternion.Euler(0,0,ang);
        PosPendulo=angulo;
```

```
        PP.text = "" + angulo;
        tempPosCarro = CP.transform.position;
        tempPosCarro.x = dist;
        CP.transform.position = tempPosCarro;
        PosCarro=dist;
        PC.text = "" + dist;
    }

    public void Reset(){
        opc="a";
    }
    public void ContrlInt(){
        opc="b";
    }
    public void ContrlExt(){
        opc="c";
    }
}
```

Script EnviarPerturbaciones

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System.IO.Ports;

public class EnviarPeturbacion : MonoBehaviour {

    public InputField ingPos;
    public InputField ingPosCarro;
    private string envPos;
    private SerialPort serialP;
    private string cadena;

    //Crea una instancia del puerto de comunicación
    void Start (){
        GameObject go = GameObject.Find ("GameObjectCP");
        Tras_Rot trasRot = go.GetComponent <Tras_Rot> ();
        serialP = trasRot.sp;
    }

    //Envía al MCU, la nueva posición del péndulo
    public void EnviarPer(){
        serialP.Write ("e");//mover el péndulo
        envPos = ingPos.text;
        cadena = envPos + "#";
        serialP.Write (cadena);
    }
    //Envía al MCU, la nueva posicíin del carro
    public void EnviarPerCarro(){
        serialP.Write ("f");//mover el carro
        envPos = ingPosCarro.text;
        cadena = envPos + "#";
```

```
        serialP.Write (cadena);

    }
}
```

Script Enviar parametros

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using System.IO.Ports;

public class EnviarParametros : MonoBehaviour {

    public InputField jF;
    public InputField RF;
    public InputField McF;
    public InputField mF;
    public InputField LF;
    public InputField FcF;
    public InputField FpF;
    public InputField RaF;
    public InputField gF;
    public InputField KF;
    public InputField IF,DF;

    private string jNum,RNum,McNum,mNum,INum,FcNum,FpNum,RaNum,gNum,KNum,Lnum,DN
um;
    private string cadena;
    private SerialPort serialP;

    void Start () {
        GameObject go = GameObject.Find ("GameObjectCP");
        Tras_Rot trasRot = go.GetComponent <Tras_Rot> ();
        serialP = trasRot.sp;
    }

    //envía por puerto serie los nuevos parámetros al MCU.
    public void EnviarDatos(){
        serialP.Write ("d"); //p
        jNum = jF.text;
        RNum = RF.text;
        McNum = McF.text;
        mNum = mF.text;
        Lnum = LF.text;
        FcNum = FcF.text;
        FpNum = FpF.text;
        RaNum = RaF.text;
        gNum = gF.text;
        KNum = KF.text;
        INum = IF.text;
        DNum = DF.text;

        cadena = jNum + ";" + RNum + ";" + McNum + ";" + mNum + ";" + Lnum + ";"
```

```
 + FcNum + ";" + FpNum + ";" + RaNum + ";" + gNum + ";" + KNum + ";" +INum+ ";"
+ DNum + ";"+ "#";
        serialP.Write (cadena);
    }
}
```

Script CambiarEscena

```
using UnityEngine;
using System.Collections;
using System.IO.Ports;

public class CambiarEscena : MonoBehaviour {
    private SerialPort serialP;

    /*Cambia de la primera escena a la segunda y viceversa*/
    void Start () {
        GameObject go = GameObject.Find ("GameObjectCP");
        Tras_Rot trasRot = go.GetComponent <Tras_Rot> ();
        serialP = trasRot.sp;
    }

    public void Comenzar(string LevelName){
        Application.LoadLevel ("CarroPendulo");
    }

    public void mainMenu(string LevelName){
        Application.LoadLevel ("GUI");
        serialP.Close ();
    }
    public void Salir(){
        Application.Quit ();
        serialP.Close ();
    }

}
```

Script Graficar

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using System.IO.Ports;
using UnityEngine.UI;

public class Grafica : MonoBehaviour {

    public float posCarro;
    public float posPend;
    public float u;
```

```
    private Tras_Rot trasRot;

    // Grafica las variables
    void Start () {
        GameObject go = GameObject.Find ("GameObjectCP");
        trasRot = go.GetComponent <Tras_Rot> ();
    }


    void Update () {
        posCarro = trasRot.PosCarro;
        posPend = trasRot.PosPendulo;
        u = trasRot.control;
        Grapher.Log(posPend, "Posición Péndulo", Color.cyan);
        Grapher.Log(posCarro, "Posición Carro", Color.green);
        Grapher.Log(u, "Control", Color.white);
    }
}
```