Implementación de plataforma Hardware in the Loop para la enseñanza del control sobre sistema embebido de bajo costo

Brian Alexis Cárdenas Castañeda
Alyson Valeria Castiblanco Castañeda

Universidad ECCI
Escuela Colombiana de Carreras Industriales
Facultad de Ingeniería
Bogotá, Colombia
2022

Implementación de plataforma Hardware in the Loop para la enseñanza del control sobre sistema embebido de bajo costo

Brian Alexis Cárdenas Castañeda
Alyson Valeria Castiblanco Castañeda

Tesis como requisito parcial para optar al título de:

Ingeniero Mecatrónico

Director (a):

MSc., Ingeniero Jorge Eduardo Cote Ballesteros

Codirector (a):

MSc., Ingeniero Jhon Edisson Rodríguez Castellanos

Línea de Investigación:

Control Automático

Grupo de Investigación/Semillero:

Semillero RV-CAR

Universidad ECCI

Escuela Colombiana de Carreras Industriales

Facultad de Ingeniería

Bogotá, Colombia

2022

Siento una enorme gratitud por los que me dijeron que no. Gracias a ellos lo hice yo mismo.

Albert Einstein

Agradecimientos

Al Físico Daniel David Castiblanco Castañeda, por sus orientaciones y contribución en los temas relacionados con el modelado numérico, corazón de este proyecto.

Al Ingeniero Edilberto Carlos Vivas González, por sus contribuciones al esclarecimiento del desarrollo de uno de los componentes del proyecto.

A Leidy Sophia Sandoval Camargo, quien dio forma a la unicidad y sentido de las ideas plasmadas en este documento en estilo y forma de escritura.

A nuestras madres, por ser motivación y motor para seguir por el camino de la perseverancia y amor por la investigación

Resumen

En el ámbito del control industrial, la posibilidad de contar con una representación virtual de cualquier planta, ya sea por motivos económicos o tiempo de desarrollo, está en auge, puesto que con esta herramienta, es posible aumentar las posibilidades de considerar cualquier escenario de comportamiento y así proveer una sintonización más adecuada y más certera de los controladores, esta técnica se conoce como Hardware In The Loop. Dentro del ámbito académico, sería más que apropiado traer todas estas ventajas en los componentes prácticos de las asignaturas de Control, ya que se observarían ahorros económicos, de tiempos de desarrollo e incluso de problemas de singularidad que presentan programas como Matlab. Es por eso que a lo largo de este documento, se explorará con detalle el proceso de desarrollo de una solución basada en esta técnica, la cual busca simular el comportamiento de dos plantas electromecánicas (Péndulo de Furuta y Helicóptero de dos grados de libertad) a través del diseño e implementación en Python del método numérico de Runge- Kutta de cuarto orden y esta simulación se hace visible a través de una interfaz HMI capaz de permitir una interacción visual con el usuario y proyectar señales medibles, todo desarrollado a través de un modelo Vista- Controlador. Finalmente, se hace la validación a través de la sintonización de dos controladores PID, los cuales demuestran la funcionalidad del sistema y sugerencias relacionadas con la sintonización de estos controladores.

Palabras clave: Método numérico, sistema, algoritmo, interfaz, simulación. Controlador, HIL.

Abstract

In the Industrial Control field, the possibility to find with a virtual representation of any system, due to economical or development time reasons is increasing, because with this tool, is possible to considerate more operation scenarios and with that give better controller sintonizations, this technique is known as Hardware In The Loop. In the academic field, it would be excellent to bring all these advantages in the practical topics of the subjects related with control with the same economical and time development advantages, also to fix singularity errors that is possible to find in programs like Matlab. In the following document, it will be explored with detail the two electromechanical plants behavior (Furuta's Pendulum and Two Degrees of Freedom Helicopter) through the development and implementation in Python of fourth order Runge- Kutta numeric method projected by a HMI interface able to have a visual interaction with the user and give measured signals, everything developed through View- Controller Model. Finally, the tests are projected through two PID controller sintonization, that can demonstrate how it works the system, with some suggestions relate with these sintonizations.

Keywords: Numeric Method, system, algorithm, interface, simulation, controller, HIL.

Contenido

Lista de figuras

Pág.

Lista de tablas

Pág.

Lista de Símbolos y abreviaturas

Símbolos con letras latinas

Símbolo	Término	Unidad SI	Definición
B_p	Fricción viscosa del eje Pitch	N/V	**Tabla 3-2**
B_y	Fricción viscosa del eje Yaw	N/V	**Tabla 3-2**
b	Fricción viscosa considerada en el Péndulo de Furuta	Nm/(rad/s)	**Tabla 3-1**
F_{cpp}	Constante debida a la fricción de Coloumb Pitch-Pitch	Nm/V	**Tabla 3-2**
F_{cpy}	Constante debida a la fricción de Coloumb Pitch-Yaw	Nm/V	**Tabla 3-2**
F_{cyp}	Constante debida a la fricción de Coloumb Yaw-Pitch	Nm/V	**Tabla 3-2**
F_{cyy}	Constante debida a la fricción de Coloumb Yaw-Yaw	Nm/V	**Tabla 3-2**
F_g	Fuerza gravitacional	N	Sección 2.2
F_p	Fuerza perpendicular en Pitch	N	Sección 2.2

Símbolo	Término	Unidad SI	Definición
F_y	Fuerza perpendicular en Yaw	N	Sección 2.2
g	Aceleración de la gravedad	$\frac{\mathrm{m}}{\mathrm{s}^2}$	9,81
h	Altura del brazo del helicóptero por debajo del fuselaje	m	**Tabla 3-2**
J_{eqp}	Momento de inercia equivalente alrededor de Pitch	kgm^2	**Tabla 3-2**
J_{eqy}	Momento de inercia equivalente alrededor de Yaw	kgm^2	**Tabla 3-2**
J_m	Momento de inercia equivalente del sistema del Péndulo de Furuta	kgm^2	**Tabla 3-1**
K	Energía cinética total del Péndulo de Furuta	J	Ec.$K = \frac{1}{2}\left[J_m\dot{\varphi}^2 + \frac{1}{3}m_a l_a\dot{\varphi}^2 + m_p\left({l_a}^2 + \frac{1}{3}{l_p}^2\sin^2\theta\right)\dot{\varphi}^2 + m_p l_a l_p\cos\theta\dot{\varphi}\dot{\theta}\right] + \frac{1}{2}\left[\frac{1}{3}m_p{l_p}^2\dot{\theta}^2 + M\left({l_a}^2 + {l_p}^2\sin^2\theta\right)\dot{\varphi}^2 + 2Ml_a l_p\cos\theta\dot{\varphi}\dot{\theta} + M{l_p}^2\dot{\theta}^2\right]$ (2.5
K_e	Constante de fuerza electromotriz del motor	Vs/rad	**Tabla 3-1**
K_t	Constante de par del motor	Nm/A	**Tabla 3-1**
K_{pp}	Constante de voltaje a torque en Pitch por voltaje aplicado en Pitch	Nm/V	**Tabla 3-2**
K_{py}	Constante de voltaje a	Nm/V	**Tabla 3-2**

Símbolo	Término	Unidad SI	Definición
	torque en Pitch por voltaje aplicado en Yaw		
K_{yp}	Constante de voltaje a torque en Yaw por voltaje aplicado en Pitch	Nm/V	**Tabla 3-2**
K_{yy}	Constante de voltaje a torque en Yaw por voltaje aplicado en Yaw	Nm/V	**Tabla 3-2**
l_a	Longitud del brazo horizontal del péndulo de Furuta	m	**Tabla 3-1**
l_p	Longitud del brazo vertical del péndulo de Furuta	m	**Tabla 3-1**
l_{mc}	Distancia del centro de masa a lo largo del fuselaje del helicóptero	m	**Tabla 3-2**
M	Masa de balance	kg	**Tabla 3-1**
m_a	Masa de la barra horizontal del Péndulo de Furuta	kg	**Tabla 3-1**
m_p	Masa del péndulo	kg	**Tabla 3-1**
M_{heli}	Masa total del helicóptero	kg	**Tabla 3-2**
P	Energía potencial	J	E c.$P = M_{heli}g(l_{mc}\sin\theta - h\cos\theta)$ (2.19

Símbolo	Término	Unidad SI	Definición
	total del helicóptero		
R	Resistencia del motor	Ω	**Tabla 3-1**
r_x, r_y, r_z	Posición radial en x, y, z del Péndulo de Furuta	Rad	Ec.$r_x(r_a, r_p) = r_a \cos\varphi - r_p \sin\varphi \sin\theta$ (2.1,$r_y(r_a, r_p) = r_a \sin\varphi + r_p \cos\varphi \sin\theta$ (2.2, $r_z(r_a, r_p) = r_p \cos\theta$ (2.3
r_a, r_p	Posición radial de los brazos horizontal y vertical del Péndulo de Furuta	Rad	Sección 2.1
T_t	Energía cinética total del helicóptero	J	Ec.$T_t = \frac{1}{2} M_{heli}\left[\left(h^2 + \cos^2\theta\left({l_{mc}}^2 - h^2\right) + l_{mc} h \sin 2\theta\right)\dot{\varphi}^2 + \left({l_{mc}}^2 + h^2\right)\dot{\theta}^2\right]$ (2.20
T_0^2	Matriz de transformación homogénea del helicóptero	1	Ec.$T_0^2 = \begin{bmatrix} \cos\varphi\cos\theta & \sin\varphi & -\cos\varphi\sin\theta & l_{mc}\cos\varphi\cos\theta + h\cos \\ -\cos\theta\sin\varphi & \cos\varphi & \sin\varphi\sin\theta & -l_{mc}\cos\theta\sin\varphi - h\sin \\ \sin\theta & 0 & \cos\theta & l_{mc}\sin\varphi - h\cos \\ 0 & 0 & 0 & 1 \end{bmatrix}$ (2.14
U	Energía potencial total del Péndulo de Furuta	J	Ec.$U = \frac{1}{2} g m_p l_p \cos\theta + g M l_p \cos\theta$ (2.6$(J_{eqp} + M_{heli}({l_{mc}}^2 + h^2))\ddot{\theta} + M_{heli}[\frac{sin2\theta({l_{mc}}^2 - h^2)}{2} - l_{mc} h cos2\theta]\dot{\varphi}^2 + M_{heli} g(l_{mc} cos\theta + h sin\theta) + B_p\dot{\theta} = \left(K_{pp} V_{mp} + F_{cpp}\right) + (K_{py} Vmy + F_{cpy})$ (2.21
v	Magnitud de la velocidad del Péndulo de Furuta y el Helicóptero	rad/s	Ec.$v^2(r_a, r_p) = ({r_a}^2 + {r_p}^2 \sin^2\theta)\dot{\varphi}^2 + 2 r_a r_p \cos\theta\dot{\varphi}\dot{\theta} + {r_p}^2\dot{\theta}^2$ (2.4, $v^2 = \left(h^2 + \cos^2\theta\left({l_{mc}}^2 - h^2\right) + l_{mc} h \sin 2\theta\right)\dot{\varphi}^2 + \left({l_{mc}}^2 + h^2\right)\dot{\theta}^2$ (2.18
v_x, v_y, v_z	Componentes x, y, z de la velocidad del Péndulo de Furuta	rad/s	Ec.$v^2(r_a, r_p) = ({r_a}^2 + {r_p}^2 \sin^2\theta)\dot{\varphi}^2 + 2 r_a r_p \cos\theta\dot{\varphi}\dot{\theta} + {r_p}^2\dot{\theta}^2$ (2.4
$V(t)$	Voltaje de entrada del sistema	V	Sección 2.1

Símbolo	Término	Unidad SI	Definición
V_{mp}	Voltaje de entrada del motor en Pitch	V	Sección 2.2
V_{my}	Voltaje de entrada del motor en Yaw	V	Sección 2.1
X_{mc}, Y_{mc}, Z_m	Posición en x, y, z del centro de masa del Helicóptero	m	Ec. $X_{mc} = (l_{mc}\cos\theta + h\sin\theta)\cos\varphi$ (2.15, $Y_{mc} = (-l_{mc}\cos\theta - h\sin\theta)\sin\varphi$ (2.16, $Z_{mc} = l_{mc}\sin\theta - h\cos\theta$ (2.17

Símbolos con letras griegas

Símbolo	Término	Unidad SI	Definición
α	Parámetro físico del Péndulo de Furuta	kg	Ec. $\alpha = J_m + \left(M + \frac{1}{3}m_a + m_p\right) l_a{}^2$ (2.9
β	Parámetro físico	kg	Ec. $\beta = \left(M + \frac{1}{3}m_p\right) l_p{}^2$ (2.10

Símbolo	Término	Unidad SI	Definición
	del Péndulo de Furuta Paramétrico		
γ	del Péndulo de Furuta Paramétrico	kg	Ec. $\gamma = \left(M + \frac{1}{2}m_p\right) l_a l_p$ (2.11
δ	del Péndulo de Furuta Paramétrico	$\frac{kg}{s^2}$	Ec. $\delta = \left(M + \frac{1}{2}m_p\right) g l_p$ (2.12

Símbolo	Término	Unidad SI	Definición
τ_{φ}	Nm	Nm	Ec. $\tau_{\varphi}(t) = \frac{K_t}{R}V(t) - \frac{K_e K_t}{R}\dot{\varphi}(t)$ (2.13 Modelo aproximado del torque del motor del Péndulo de Furuta
θ	Por sición del brazo vertical	rad	Sección 2.1 y 2.2

Símbolo	Término	Unidad SI	Definición
	cal del Péndulo de Furuta y eje Pitch del Helicóptero		
φ	Posición horizontal del brazo horizontal del Péndulo de		Sección 2.1 y 2.2

Símbolo	Término	Unidad SI	Definición
$\dot{\theta}$	Velocidad del brazo horizontal del Péndulo de Furuta y eje Yaw del Helicóptero	rad/s	Ec. $\ddot{\theta} = \frac{\beta(\alpha+\beta sin^2\theta)cos\theta sin\theta\dot{\varphi}^2+2\beta\gamma cos^2\theta sin\theta\dot{\varphi}\dot{\theta}}{\alpha\beta-\gamma^2+(\beta^2+\gamma^2)sin^2\theta} + \frac{-\gamma^2 cos\theta sin\theta\dot{\theta}^2+\delta(\alpha+\beta sin^2\theta)sin\theta-\gamma^2 cos\theta(\tau_\varphi-b\dot{\varphi})}{\alpha\beta-\gamma^2+(\beta^2+\gamma^2)sin^2\theta}$ (3.4, $\ddot{\theta} = -\frac{M_{heli}\left[\frac{sin2\theta(l_{mc}{}^2-h^2)}{2}-l_{mc}hcos2\theta\right]\dot{\phi}^2}{\left(J_{eqp}+M_{heli}(l_{mc}{}^2+h^2)\right)} - \frac{M_{heli}g(l_{mc}cos\theta+hsin\theta)+B_p\dot{\theta}-(K_{pp}V_{mp}+F_{cpp})-(K_{py}V_{my}+F_{cpy})}{\left(J_{eqp}P+M_{heli}(l_{mc}{}^2+h^2)\right)}$ (3.7

Símbolo	Término	Unidad SI	Definición
	del Helicóptero		
	Velocidad del brazo horizontal del Péndulo de Furuta y eje Yaw del Helicóptero	Ec. (3.2, REF _Ref91752427 \h * MERGEFORMAT (3.8	$\ddot{\varphi} = \frac{\beta(\tau_\varphi - b\dot{\varphi}) - \beta\gamma cos^2\theta sin\theta\dot{\varphi}^2 - 2\beta^2 cos\theta sin\theta\dot{\varphi}\dot{\theta}}{\alpha\beta - \gamma^2 + (\beta^2+\gamma^2)sin^2\theta} + \frac{\beta\gamma sin\theta\dot{\theta}^2 - \gamma\delta cos\theta sin\theta}{\alpha\beta - \gamma^2 + (\beta^2+\gamma^2)sin^2\theta}$ $\ddot{\varphi} = Mhel\left[sin2\theta h2 - lmc2 + 2lmchcos2\theta\right]\theta\varphi Jeqy + Mhelicos2\theta lmc2 - h2 + lmchsin2\theta + h2 - By\varphi - KypVmp + Fcypcos\theta - (KyyVmy + Fcyy)cos\theta Jeqy + Mhelicos2\theta lmc2 - h2 + lmchsin2\theta + h2$

Símbolo	Término	Unidad SI	Definición
$\ddot{\theta}$	Aceleración del brazo horizontal del Péndulo de Furuta y eje Yaw del Helicóptero	rad/s^2	Ec. $\ddot{\theta} = \frac{\beta(\alpha+\beta sin^2\theta)cos\theta sin\theta\dot{\varphi}^2+2\beta\gamma cos^2\theta sin\theta\dot{\varphi}\dot{\theta}}{\alpha\beta-\gamma^2+(\beta^2+\gamma^2)sin^2\theta} + \frac{-\gamma^2 cos\theta sin\theta\dot{\theta}^2+\delta(\alpha+\beta sin^2\theta)sin\theta-\gamma^2 cos\theta(\tau_\varphi-b\dot{\varphi})}{\alpha\beta-\gamma^2+(\beta^2+\gamma^2)sin^2\theta}$ (3.4, $\ddot{\theta} = -\frac{M_{heli}\left[\frac{sin2\theta\left(l_{mc}{}^2-h^2\right)}{2}-l_{mc}hcos2\theta\right]\dot{\varphi}^2}{\left(J_{eqp}+M_{heli}\left(l_{mc}{}^2+h^2\right)\right)} - \frac{M_{heli}g(l_{mc}cos\theta+hsin\theta)+B_p\dot{\theta}-(K_{pp}V_{mp}+F_{cpp})-(K_{py}V_{my}+F_{cpy})}{\left(J_{eqp}P+M_{heli}\left(l_{mc}{}^2+h^2\right)\right)}$ (3.7
$\ddot{\varphi}$	Aceleración del	rad/s^2	Ec. $\ddot{\varphi} = \frac{\beta(\tau_\varphi-b\dot{\varphi})-\beta\gamma cos^2\theta sin\theta\dot{\varphi}^2-2\beta^2 cos\theta sin\theta\dot{\varphi}\dot{\theta}}{\alpha\beta-\gamma^2+(\beta^2+\gamma^2)sin^2\theta} + \frac{\beta\gamma sin\theta\dot{\theta}^2-\gamma\delta cos\theta sin\theta}{\alpha\beta-\gamma^2+(\beta^2+\gamma^2)sin^2\theta}$ (3.2

Símbolo	Término	Unidad SI	Definición
	brazo horizontal del Péndulo de Furuta y eje Yaw del Helicóptero		

Abreviaturas

Abreviatura	Término
PVI	Problema de Valor Inicial
ED	Ecuación(es) Diferencial(es)
RK4O	Runge- Kutta de Cuarto Orden

Introducción

Dentro del campo académico de la enseñanza en ingeniería, el componente de validación de esas competencias teóricas y empalmarlas con contextos reales es de alta importancia, pues desde esta arista es posible ofrecer al estudiante una visión integral de las problemáticas que se pueden encontrar tanto en el campo industrial como de investigación. Sin embargo este aspecto se puede ver obstaculizado debido a razones económicas concernientes al desarrollo de prototipos, los cuales también emplean mucho tiempo de fabricación, y a esto se suma las dificultades de disponer de equipos especializados por situaciones de distancia o de aislamiento como las observadas durante el COVID- 19, por lo que se hace necesario pensar en una solución virtual que pueda permitir el desarrollo de estos aspectos, y en este punto se considera la técnica Hardware In The Loop, la cual proyecta una simulación de la planta que se desea explorar de tal forma que, al momento de trabajar con la sintonización de controladores, estos puedan recibir este comportamiento y realizar estas tareas de sintonización como si estuvieran enfrentándose a la planta real (Innovations, 2020).

A lo largo de las últimas dos décadas, se han hecho exploraciones sobre estos temas, denotando ejemplos como el caso donde se hace una compilación de diferentes plantas desarrolladas con fines de enseñanza y acondicionarlas en uno solo software para su consulta académica en la Universidad Autónoma de Occidente (Girón Rodriguez & Naranjo Grisales, 2018), y otro software similar desarrollado a través de Matlab, permite observar el comportamiento de diferentes plantas desde su no linealidad y los diferentes resultados con diferentes técnicas de control (Cárdenas, 1998). En el año 2010, la Universidad de Educación A Distancia de España desarrolló una alternativa de software que podía ser manipulado de manera remota a través de un sitio web (Andújar Márquez & Mateo Sanguino, 2010), y aquí se desarrollaron un sistema Heatflow, uno de tanques y otro de un motor de corriente contínua (Santana et al., 2010).

En el primer capítulo de este documento, se evidenciará el marco teórico que sustenta los antecedentes, trabajos previos y los conceptos generales necesarios para comprender a cabalidad el desarrollo del proyecto en todas sus etapas. En el segundo capítulo se mostrarán los modelos matemáticos de los dos sistemas que serán simulados en el proyecto, proporcionando una visión general de cómo se llegaron a esos modelos (análisis geométrico, planteamiento de ecuaciones de movimiento, análisis energético a través de las Ecuaciones de Lagrange y sistema de Ecuaciones Diferenciales).

En el tercer capítulo, se expone de manera detallada el algoritmo implementado para simular el comportamiento en el tiempo de los dos sistemas en el capítulo anterior a través del método de Runge- Kutta de Cuarto Orden. A través de sus secciones, se explica el procedimiento de separación del sistema de ED (se llamará de ahora en adelante las Ecuaciones Diferenciales) de segundo orden acopladas en un sistema de cuatro ED de primer orden puras para cada uno de los modelos, la explicación detallada de la variante del método para este sistema junto con la modificación que hace posible su ejecución de manera indefinida y su implementación en Python y la Raspberry 3B+, así como su validación a través de Matlab y Simulink.

En el cuarto capítulo, se explora el desarrollo de la interfaz HMI del Hardware In The Loop. Se expondrán las diferentes pantallas con las que el usuario se encontrará cuando haga uso del programa (pantalla principal, instrucciones y pantallas de simulación) y su desarrollo en la parte del Back-end (ingreso de datos, desarrollo del sistema de alertas que indica si se ingresan los datos de manera correcta, implementación del algoritmo y el uso de hilos para despliegue de gráficas en tiempo real y conversión de valores en señales eléctricas por medio del uso de un conversor Digital- Análogo con protocolo I2C), así como las pruebas efectuadas para corroborar su correcto funcionamiento.

En el quinto capítulo, se observará la validación del Hardware In The Loop en lazo cerrado a través del diseño de un controlador PID bajo la estructura paralela y discretización trapezoidal, donde uno de ellos (Helicóptero de dos grados de libertad) se sintoniza con ayuda de la herramienta de autosintonización de Simulink, mientras que en el otro caso, (Péndulo de Furuta), se registra un intento experimental y con ello, las razones de por qué se deben explorar a futuro controladores más robustos en función de los resultados obtenidos.

Finalmente, en el sexto capítulo se consideran las conclusiones generales del proyecto, así como el trabajo futuro con miras a posibles mejoras.

Planteamiento del problema

En el ámbito de la enseñanza**,** las carreras de ingeniería generalmente complementan el desarrollo teórico de sus asignaturas con un componente práctico**,** que consiste en sesiones de experimentación con elementos reales, que permiten a los estudiantes validar lo aprendido en sus clases y así reforzar los conocimientos ya adquiridos.

No obstante, la falta de herramientas en los laboratorios tradicionales en la Universidad ECCI, y la disponibilidad de los pocos que hay es una situación que dificulta el pleno proceso de aprendizaje de asignaturas como Control Industrial, Control Multivariado e Instrumentación Avanzada desde los temas enfocados al análisis del comportamiento físico de sistemas mecánicos orientados a su control por sintonización de controladores en la carrera de Ingeniería Mecatrónica además de retrasar las actividades y proyectos de los estudiantes, quienes han evidenciado estos inconvenientes desde hace unos años.

A pesar de que se ha intentado a lo largo de este tiempo de dar solución a esta problemática implementando simulaciones en programas como Matlab, Simulink y Labview e instando a la construcción de prototipos que permitan observar el comportamiento físico de las plantas planteadas, se hace necesario buscar alternativas a estos métodos propuestos con el fin de mejorar la forma en que se imparten los conceptos de esta disciplina, partiendo de tres limitantes: la primera, concerniente a los prototipos físicos y como se ha planteado anteriormente, está ligado al aspecto económico y de tiempo que requiere construirlos y por el apartado del software como Matlab, Simulink y LabView, está ligado a la compleja adquisición de las licencias para trabajar con ellos y un tema más específico relacionado con las singularidades que se pueden presentar con valores muy altos durante la simulación, situación que puede ser subsanada a través del desarrollo en plataformas de bajo costo que permiten restringir estas condiciones para que no se

presenten y es por esto, que la solución a este planteamiento puede estar orientada desde el proyecto desarrollado en este documento.

Justificación

Debido a la falta de disponibilidad de los recursos físicos en los laboratorios para hacer las sesiones experimentales, es necesario implementar alternativas de aprendizaje como complemento a lo enseñado en el plan de estudios correspondiente a las asignaturas de Control, y para ello, la tecnología ofrece herramientas que posibilitan que un entorno virtual sea una opción factible para dar solución a dicho problema.

Dentro de las ventajas que se pueden encontrar gracias al desarrollo de entornos virtuales para apoyar el aprendizaje académico mediante la simulación del comportamiento físico de las plantas propuestas, es que los tiempos de desarrollo de las actividades académicas concernientes a las prácticas de laboratorio se reducirían al no depender de la disponibilidad de los equipos, pues solo se necesitaría el acceso a una computadora o a hardware como monitor, teclado o mouse para la ejecución del programa y así hacer correctamente los procedimientos solicitados por el docente. Por otro lado, se evitaría el daño de equipos de laboratorio causados por malas conexiones o manipulación incorrecta, ya que el estudiante, aparte de simular los sistemas las veces que desee, no tendrá el temor de dañar algún elemento importante que se le facilite, dificultando su proceso de validación de los conocimientos adquiridos.

Ahora bien, dentro de los aspectos que subsana considerablemente este proyecto se encuentran las ya mencionadas ventajas económicas y de tiempo de desarrollo de una planta física que permiten que los estudiantes se enfoquen especialmente en el desarrollo de controladores, eje central de las asignaturas anteriormente mencionadas a través de un software intuitivo y amigable con el usuario donde solo este debe ingresar los valores y el programa genera la simulación del comportamiento físico. Por otro lado, gracias a que este software está implementado sobre una plataforma de bajo costo (las razones de por qué se llega a esta premisa se encuentran en próximos apartados) es una ventaja bastante

grande a comparación de la adquisición de licencias limitadas y de alto precio económico como Matlab o Simulink o incluso, la adquisición de un equipo de cómputo en concreto y con altas capacidades de procesamiento, pues el software fue desarrollado sobre una plataforma de uso libre, disponible para cualquier tipo de usuario interesado. Finalmente, los problemas de singularidad y valores altos (aspectos que también se consideran en capítulos próximos) que es una fuerte limitante en los cálculos de soluciones presentes en los programas anteriormente mencionados, cuestión que es solucionada en el software gracias a las restricciones que se implementan y que suprimen estos errores dentro de la simulación.

Por último, esta herramienta de aprendizaje beneficiaría a los estudiantes de ingeniería de últimos semestres de carreras como la Ingeniería Mecatrónica, Ingeniería Biomédica e Ingeniería Electrónica, quienes cursan las asignaturas correspondientes al control (Control Industrial, Sistemas Dinámicos y Control Multivariado), enriqueciendo el plan de estudios de estas y aportando el complemento práctico que requieren.

Objetivos

Objetivo General

Desarrollar una planta virtual basada en la técnica de Hardware In The Loop para la simulación de sistemas de control no lineales sobre plataformas de bajo costo, que cuente con una interfaz gráfica como puente de interacción entre los usuarios y los procesos simulados, con el objetivo de presentar una alternativa a los laboratorios tradicionales en las universidades.

Objetivos Específicos

- Recopilar información acerca de proyectos similares desarrollados con anterioridad, así como de herramientas tanto matemáticas como de ingenierías necesarias para la creación de la propuesta.

- Implementar el algoritmo que establece la respuesta en el tiempo de los procesos de control, mediante el método numérico de Runge- Kutta de cuarto orden en tiempo real y de manera física mediante una señal medible.

- Desarrollar una interfaz gráfica que permita la comunicación entre usuario y dispositivo.

- Implementar un controlador tipo PID que permita la validación del funcionamiento del software en lazo cerrado.

1. Capítulo 1: Marco Referencial

1.1 Antecedentes

El uso de los entornos de experimentación más conocidos como laboratorios ha sido una práctica extendida a lo largo de todos los ambientes académicos, pues la necesidad de que el estudiante pueda comprobar los resultados teóricos enseñados en sus respectivas asignaturas con lo que sucede en la realidad, hace parte de una formación integral que le permite conocer de primera mano los resultados de ciertas acciones practicadas por el alumno de acuerdo a su instrucción y así, mejorar su experiencia de aprendizaje en su campo (Santana et al., 2010).

Ahora bien, hasta la segunda mitad del siglo pasado, los laboratorios tradicionales, es decir, ambientes presenciales donde los estudiantes acudían durante cierta cantidad de tiempo en compañía de un docente para hacer sus prácticas, predominó en las aulas como método complementario de enseñanza (Giron Rodriguez & Naranjo Grisales, 2018), pero a través de los años, la demanda de estudiantes crece, y el acceso a los laboratorios y a los equipos de las universidades se vuelve cada vez más escaso, y específicamente en el terreno de la ingeniería, estos insumos suelen ser aún más costosos y de acceso limitado, por lo que el proceso de formación puede verse seriamente afectado (Pedro et al., 1940).

Debido a esto, el desarrollo de herramientas virtuales que puedan ejercer las mismas funciones de un laboratorio físico con los mismos beneficios se ha convertido en objeto de estudio de muchas universidades a lo largo del mundo, pues el hecho de poder acceder a prácticas de laboratorio desde cualquier lugar, en cualquier momento y las veces que sean necesarias resulta ser una idea atractiva para reforzar los conocimientos de todos los estudiantes involucrados (Andújar Márquez & Mateo Sanguino, 2010). Los primeros pasos sobre este tema se dieron en 1984, cuando se comienza a considerar los instrumentos

virtuales como un producto obtenido de la programación y, por lo tanto, imprescindibles para el desarrollo de "Laboratorios virtuales". Durante la década de los años 80 y principios de los 90, diferentes instituciones universitarias en los Estados Unidos se ocuparon del desarrollo de entornos de prueba accesibles directamente desde un ordenador. Tal es el caso de la Universidad de Bucknell, la cual creó en 1991 un sistema de procesado digital y conexión a Internet, y en 1992 ya se formalizó el término de "Laboratorio virtual" para estos entornos de aprendizaje, ahora desarrollados desde la programación orientada a objetos y cuyo primer representante fue el prototipo MWS (Microscopist's WorkStation), que después se convertiría en el sistema CMDA (ColLaboratory for Microscopio Digital Anatomy). Y en el transcurso de los años 90, se hicieron artículos, experimentos y conferencias sobre el tema, resaltando como conclusión los beneficios de las prácticas virtuales como herramienta para optimizar los tiempos de aprendizaje y en uso de equipos en las universidades, además de sofisticar las técnicas de programación de estos entornos pasando a ambientes de desarrollo con Visual Basic o Java, e incluso recurriendo a técnicas como CGI, ISAPI o COM+ para hacer la experiencia lo más real posible (Giron Rodriguez & Naranjo Grisales, 2018).

1.2 Marco teórico

1.2.1 Introducción a los métodos numéricos

Gran parte de los sistemas o fenómenos físicos, químicos, económicos o estadísticos pueden ser descritos de una manera acertada mediante la relación de variables que, asociadas de una manera ordenada y coherente, pueden predecir el comportamiento de dichos sistemas, a esta relación se le conoce como ECUACIÓN. Ahora bien, muchos de estos sistemas, describen pequeñas variaciones que requieren un análisis de instantes muy pequeños de tiempo, por lo que se involucra el concepto de DERIVADA, y toda aquella ecuación que involucre las derivadas de las variables dependientes (a medida que cambia la variable independiente (la mayoría de las ocasiones es el tiempo) cambia el comportamiento de la variable) y las variables independientes (variables que no varían en función de otros parámetros), se le conoce como ECUACIÓN DIFERENCIAL. Estas descripciones matemáticas de los sistemas, han permitido entender el mundo actual y desarrollar diferentes elementos o campos de estudio que mejoren la calidad de vida de la

sociedad y aportar aún más a las diferentes áreas de conocimiento general. Para hallar la solución de este tipo de ecuaciones, existen diferentes métodos analíticos que, con condiciones específicas dadas, permiten hallar ya sea una solución general (ecuación perteneciente a una familia de ecuaciones que describe el comportamiento de la ecuación al ser resuelta, por lo que, no se cuenta con valores específicos para unas constantes que la volverían particular) o una solución particular (ecuación que cuenta con valores específicos en sus constantes, lo cual ya no la asociaría simplemente a una familia de funciones, que se obtiene al plantear condiciones iniciales que afecten la conducta de la ecuación). Sin embargo, en la práctica, no siempre es posible hallar la solución EXACTA de las ecuaciones que se puedan plantear, pues, ya sea porque su cálculo es muy complejo, o no se conozca el método indicado, o de lleno, no exista, es necesario recurrir a otras técnicas que permitan no necesariamente llegar a su solución exacta sino mas bien, a una muy aproximada. Para ello, se recurre a una sucesión de operaciones numéricas que arrojan un valor muy cercano a la realidad. Entonces, cuando se hacen esta sucesión de cálculos (actualmente con la ayuda de los ordenadores), al final se obtiene una aproximación. Esta técnica se le conoce como MÉTODO NUMÉRICO(Departamento EDAN, 2019).

Los métodos numéricos permiten, mediante el uso de ordenadores, resolver una gran cantidad de problemas matemáticos, de ingeniería y científicos de una manera rápida y muy cercana a la realidad que a su vez, trae beneficios a la mejora de las habilidades del manejo del software existente, a la comprensión de los principios científicos básicos y el análisis matemático (VetNumVMC, 2019). Para que la solución por medio del uso de ordenadores sea correcto, esta debe pasar por cinco fases a considerar: Una ESPECIFICACIÓN DEL PROBLEMA (Identificación de variables involucradas, limitaciones y los resultados esperados), ANÁLISIS (la formulación de la solución transformada en el ALGORITMO, que son los pasos que debe seguir la computadora para llegar a dicha solución), PROGRAMACIÓN (momento en que el programador traduce los pasos en código que pueda ejecutar el ordenador), VERIFICACIÓN (al código ya constituido se le practican gran cantidad de pruebas para detectar errores mediante ejercicios de los cuales ya se conozca su solución), DOCUMENTACIÓN Y PRODUCCIÓN (donde se hace un instructivo de uso para que cualquier usuario pueda utilizar el programa sin problema y finalmente se ingresan los parámetros de las ecuaciones a probar para obtener los resultados) (VetNumVMC, 2019).

Dentro de los diferentes procedimientos matemáticos en que los métodos numéricos tienen aplicación se encuentran: derivadas, integrales, operaciones con matrices, interpolaciones, ajuste de curvas, polinomios y ecuaciones diferenciales, siendo este último, el objeto de estudio de este trabajo.

1.2.2 Método de Runge- Kutta de cuarto orden

Se define Método de Runge- Kutta a un procedimiento numérico que busca hallar solución a una determinada ecuación diferencial con Problema de Valor Inicial de la forma de la Ecuación 1.1:

$$y' = f(x, y), y(x_0) = y_0 \quad (1.1)$$

Mediante hallar una sucesión de puntos que se aproximen a la curva de solución, en otras palabras, conserva el fundamento de lo que es un método numérico. Debido a su facilidad de aplicación, es uno de los métodos más difundidos para la solución de dichas ecuaciones, pues este procedimiento, el cual viene en distintos órdenes, se deduce a partir del desarrollo de $y(x_n + h)$ en serie de Taylor con residuo. Esta herramienta permite que se halle una función aproximada a través de una sucesión polinómica como en la evidenciada en la Ecuación 1.2 (Rocha, 2005):

$$y(x_n + 1) = y(x_n + h) = y(x_n) + hy'(x_n) + \frac{h^2}{2!}y^n(x_n) + \frac{h^3}{3!}y^m(x_n) + \cdots + \frac{h^{k+1}}{(k+1)!}y^{(k+1)}(c) \quad (1.2)$$

Donde c se encuentra entre x_n y $x_n + h$. Ahora bien, bajo las condiciones convenientemente dadas de que $k = 1$ y el residuo $\frac{h^2}{2!}y^n(c)$ es muy pequeño, se obtendría la conocida fórmula de iteración de otro método numérico más conocido como MÉTODO DE EULER, evidenciado en la Ecuación 1.3 (Zill, 1997):

$$y(n + 1) = y_n + hy'_n = y_n + hf(x_n, y_n) \quad (1.3)$$

Como se mencionó anteriormente, la serie de Taylor planteada de esta forma puede dar una solución aproximada a cualquier ecuación propuesta, pero la precisión depende de dos cosas: del valor de n, el cual proporciona el número de términos de la serie. Entre más términos se involucren, se observa que el residuo es más pequeño y más aproximada es la solución (debe recordarse que el residuo y el error de truncamiento de la solución están

directamente relacionados) y del valor de h el cual representa el tamaño del paso, es decir cada cuanto se hará el análisis de las iteraciones dentro del intervalo de observación. Entre más cercanos estén los puntos de análisis (el paso más pequeño), mayor cantidad de valores se obtendrán y por ende, más aproximada será la solución (Rocha, 2005). Es entonces, que se entiende el concepto de orden en el método numérico de Runge- Kutta, pues no es más que presentar un polinomio de Taylor del grado que depende a su vez del orden que se desea dar al método. Hay que tener en cuenta, que muy pocos términos del polinomio arrojarían un error cada vez mayor, mientras que muchos términos darían una aproximación más certera, pero requeriría de mayor cantidad de operaciones y directamente, mayor poder computacional. El MÉTODO DE RUNGE- KUTTA DE CUARTO ORDEN satisface estos dos parámetros: ofrece una buena aproximación de la función solución del PVI y con una cantidad de operaciones que una pequeña computadora podría manejar con facilidad.

El método de Runge- Kutta de cuarto orden, desde la experiencia propia del proyecto, tiene tres variantes: una para Ecuaciones Diferenciales de primer orden, una para Ecuaciones Diferenciales de segundo orden y una adicional para un sistema de Ecuaciones Diferenciales de segundo orden acopladas (un sistema compuesto por dos ED de segundo orden cada una y ambas cuentan con las variables independientes de la otra o sus derivadas). La tercera variante es la de interés, pues, como se mencionará en el próximo apartado, las dos plantas de análisis cuentan con esta característica particular.

Un sistema de Ecuaciones Diferenciales de segundo orden tiene las siguientes características, como se observa en las ecuaciones 1.4 y 1.5, con condiciones iniciales relacionadas en las ecuaciones 1.6, 1.7 y 1.8:

$$\frac{d^2x}{dt} = f(t, x, v_x, y, v_y) \tag{1.4}$$

$$\frac{d^2x}{dt} = g(t, x, v_x, y, v_y) \tag{1.5}$$

$$x(t_0) = x_0 \tag{1.6}$$

$$\left(\frac{dx}{dt}\right)_{t_0} = v_{x0} \tag{1.7}$$

$$x(t_0) = x_0 \tag{1.8}$$

Se puede observar que el sistema de ecuaciones diferenciales de segundo orden está compuesto por funciones que contienen las variables dependientes (generalmente se refieren a la posición) y sus derivadas. Es menester aclarar que es indispensable contar con sus respectivas condiciones iniciales.

Para hacer uso del método Runge- Kutta para resolver una ED de segundo orden, es necesario dividir esta ecuación en dos ecuaciones más sencillas de primer orden, pues debe recordarse que el método de Runge-Kutta está diseñado exclusivamente para resolver ecuaciones de primer orden puras (Zill, 1997), por lo que, si se cuenta con una ED de segundo orden, debe hacerse esta división, a través de las llamadas Variables de estado, tal y como se muestra en la ecuación 1.9:

$$\frac{dx}{dt} = v_x;\ \frac{dv_x}{dt} = f(t, x, y) \tag{1.9}$$

Entonces, ya teniendo esta división, se adaptaría la siguiente variante como lo muestran las ecuaciones 1.10 y 1.11:

$$x_{n+1} = x_n + \frac{1}{6}(k_1 + 2k_2 + 3k_3 + k_4) \tag{1.10}$$

$$y_{n+1} = y_n + \frac{1}{6}(m_1 + 2m_2 + 3m_3 + m_4) \tag{1.11}$$

Donde las k se calculan así, según las ecuaciones 1.10.1, 1.10.2, 1.10.3 y 1.10.4:

$$k_1 = hf(t_n, x_n, y_n) \tag{1.10.1}$$

$$k_2 = hf\left(t_n + \frac{h}{2}, x_n + \frac{k_1}{2}, y_n + \frac{m_1}{2}\right) \tag{1.10.2}$$

$$k_3 = hf\left(t_n + \frac{h}{2}, x_n + \frac{k_2}{2}, y_n + \frac{m_2}{2}\right) \tag{1.10.3}$$

$$k_4 = hf(t_n + h, x_n + k_3, y_n + m_3) \tag{1.10.4}$$

Mientras que las m se calculan así, según las ecuaciones 1.11.1, 1.11.2, 1.11.3 y 1.11.4:

$$m_1 = hg(t_n, x_n, y_n) \tag{1.11.1}$$

$$m_2 = hg\left(t_n + \frac{h}{2}, x_n + \frac{k_1}{2}, y_n + \frac{m_1}{2}\right) \quad (1.11.2)$$

$$m_3 = hg\left(t_n + \frac{h}{2}, x_n + \frac{k_2}{2}, y_n + \frac{m_2}{2}\right) \quad (1.11.3)$$

$$m_4 = hg(t_n + h, x_n + k_3, y_n + m_3) \quad (1.11.4)$$

Ahora bien, esto es para una sola ED de segundo orden. Si se cuenta con dos ED de este tipo acopladas, se tendría como resultado un sistema de cuatro ED de primer orden en función de sus variables de estado, tal y como se muestra en las ecuaciones 1.12 y 1.13:

$$\frac{dx}{dt} = v_x;\ \frac{dv_x}{dt} = f(t, x, v_x, y, v_y) \quad (1.12)$$

$$\frac{dy}{dt} = v_y;\ \frac{dv_y}{dt} = g(t, x, v_x, y, v_y) \quad (1.13)$$

Finalmente, ya teniendo esta división, se aplicará una variante del método de Runge- Kutta que permite calcular un sistema de cuatro ecuaciones a la vez, haciendo uso de las ecuaciones 1.14, 1.15, 1.16 y 1.17:

$$x_{n+1} = x_n + \frac{1}{6}(k_1 + 2k_2 + 3k_3 + k_4) \quad (1.14)$$

$$\dot{x}_{n+1} = \dot{x}_n + \frac{1}{6}(m_1 + 2m_2 + 3m_3 + m_4) \quad (1.15)$$

$$y_{n+1} = y_n + \frac{1}{6}(l_1 + 2l_2 + 3l_3 + l_4) \quad (1.16)$$

$$\dot{y}_{n+1} = \dot{y}_n + \frac{1}{6}(b_1 + 2b_2 + 3b_3 + b_4) \quad (1.17)$$

Donde las k se calcularían según las ecuaciones 1.14.1, 1.14.2, 1.14.3 y 1.14.4:

$$k_1 = hf(t_n, x_n, \dot{x}_n, y_n, \dot{y}_n) \quad (1.14.1)$$

$$k_2 = hf\left(t_n + \frac{h}{2}, x_n + \frac{k_1}{2}, \dot{x}_n + \frac{m_1}{2} y_n + \frac{l_1}{2}, \dot{y}_n + \frac{b_1}{2}\right) \quad (1.14.2)$$

$$k_3 = hf\left(t_n + \frac{h}{2}, x_n + \frac{k_2}{2}, \dot{x}_n + \frac{m_2}{2} y_n + \frac{l_2}{2}, \dot{y}_n + \frac{b_2}{2}\right) \quad (1.14.3)$$

$$k_4 = hf(t_n + h, x_n + k_3, \dot{x}_n + m_3, y_n + l_3, \dot{y}_n + b_3) \quad (1.14.4)$$

Las m se calcularían según las ecuaciones 1.15.1, 1.15.2, 1.15.3 y 1.15.4:

$$m_1 = hg(t_n, x_n, \dot{x}_{n,} y_n, \dot{y}_n) \quad (1.15.1)$$

$$m_2 = hg\left(t_n + \frac{h}{2}, x_n + \frac{k_1}{2}, \dot{x}_n + \frac{m_1}{2} y_n + \frac{l_1}{2}, \dot{y}_n + \frac{b_1}{2}\right) \quad (1.15.2)$$

$$m_3 = hg\left(t_n + \frac{h}{2}, x_n + \frac{k_2}{2}, \dot{x}_n + \frac{m_2}{2} y_n + \frac{l_2}{2}, \dot{y}_n + \frac{b_2}{2}\right) \quad (1.15.3)$$

$$m_4 = hg(t_n + h, x_n + k_3, \dot{x}_n + m_3, y_n + l_3, \dot{y}_n + b_3) \quad (1.15.4)$$

Las l se calcularían según las ecuaciones 1.16.1, 1.16.2, 1.16.3 y 1.16.4:

$$l_1 = hz(t_n, x_n, \dot{x}_{n,} y_n, \dot{y}_n) \quad (1.16.1)$$

$$l_2 = hz\left(t_n + \frac{h}{2}, x_n + \frac{k_1}{2}, \dot{x}_n + \frac{m_1}{2} y_n + \frac{l_1}{2}, \dot{y}_n + \frac{b_1}{2}\right) \quad (1.16.2)$$

$$l_3 = hz\left(t_n + \frac{h}{2}, x_n + \frac{k_2}{2}, \dot{x}_n + \frac{m_2}{2} y_n + \frac{l_2}{2}, \dot{y}_n + \frac{b_2}{2}\right) \quad (1.16.3)$$

$$l_4 = hz(t_n + h, x_n + k_3, \dot{x}_n + m_3, y_n + l_3, \dot{y}_n + b_3) \quad (1.16.4)$$

Y las b se calcularían según las ecuaciones 1.17.1, 1.17.2, 1.17.3 y 1.17.4:

$$b_1 = hj(t_n, x_n, \dot{x}_{n,} y_n, \dot{y}_n) \quad (1.17.1)$$

$$b_2 = hj\left(t_n + \frac{h}{2}, x_n + \frac{k_1}{2}, \dot{x}_n + \frac{m_1}{2} y_n + \frac{l_1}{2}, \dot{y}_n + \frac{b_1}{2}\right) \quad (1.17.2)$$

$$b_3 = hj\left(t_n + \frac{h}{2}, x_n + \frac{k_2}{2}, \dot{x}_n + \frac{m_2}{2} y_n + \frac{l_2}{2}, \dot{y}_n + \frac{b_2}{2}\right) \quad (1.17.3)$$

$$b_4 = hj(t_n + h, x_n + k_3, \dot{x}_n + m_3, y_n + l_3, \dot{y}_n + b_3) \quad (1.17.4)$$

Aquí, se tendrían cuatro soluciones: dos para las variables dependientes, y dos para las derivadas de esas variables.

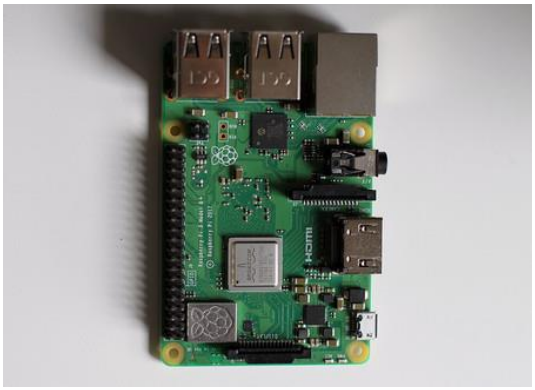
1.2.3 Raspberry PI

La Raspberry Pi inicio como proyecto en el año 2006 por el fundador Eben Upton (Universidad de Cambridge) Junto con Rob Mullins y Alan Mycroft quienes percibieron en este embebido no como un producto de consumo, sino como estímulo de las ciencias de la computación en estudiantes a través de diminutos ordenadores, idealizado como un producto económico y portable. (Eugenio Lopez Aldea, 2017)

Desde su primer lanzamiento en el año 2012 el objetivo se ha centrado sobre la promoción en el campo académico, tanto en universidades como en colegios; contando para ese mismo año con distribuciones de la Raspberry Pi modelo B en diferentes tiendas y para el año 2013 el modelo A que sería más asequible. (Valera et al., 2014)

Entrando ya en el modelo utilizado en el desarrollo del proyecto es la Raspberry Pi 3 modelo B+. Que en comparación con la Raspberry Pi modelo 3 que a nivel de hardware no cuenta con muchos cambios sino más que todo en cambios que afectan la conectividad.

Figura 1.1: Raspberry Pi 3 Modelo B+. (Pastor, 2018)



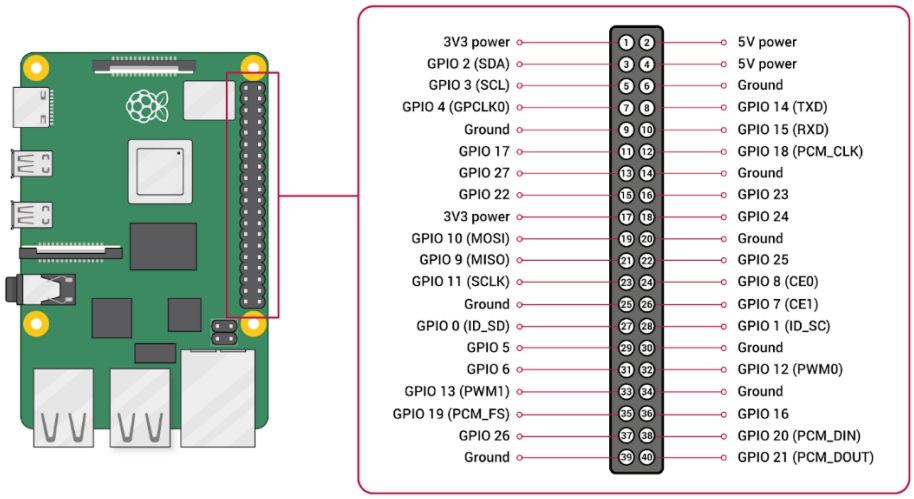
Un avistamiento general de las principales características de la Raspberry Pi 3 modelo B+ se puede ver en la tabla N°1.

Tabla 1-1.1: Características Raspberry Pi 3B+ (Pastor, 2018)

Procesador	Broadcom BCM2837B0, Cortex-A53 64-bit SoC
Frecuencia de Reloj	1.4 GHz
Memoria	1GB LPDDR2 SDRAM
Conectividad Inalambrica	2.4GHz/5GHz IEEE 802.11b/g/n/ac
Conectividad de Red	Gigabit Ethernet over USB 2.0 (300Mbps de máximo teórico)
Puertos	HDMI 4 x USB 2.0 CSI DSI Micro SD Micro USB Power-Over Ethernet (PoE)
Precio	$39.75

La Raspberry Pi 3 modelo B+ cuenta con 40 pines que pueden ser usados como entradas y salidas de datos, sin embargo, todos estos son exclusivamente pines digitales; por lo que para poder leer o sacar datos analógicos es necesario utilizar un conversor DAC.

Figura 1.2: Pines de la Raspberry Pi 3B+. (*Aprendiendo Arduino*, n.d.)



1.2.4 Conversor DAC MPC 4725.

Un convertidor DAC modelo MPC 4725 *(Digital Analog Convert*) es un dispositivo que en síntesis se encarga de convertir señales digitales y por medio del protocolo I2C pasarlas a señales continuas, que cuenta con una memoria no volátil. (*Electronic Components Datasheet Search*, n.d.)

Este dispositivo es muy útil, cuando se utiliza como soporte para tarjetas embebidas que no cuentan con entradas de lectura digital, haciéndose necesario una transformación en la naturaleza de la señal

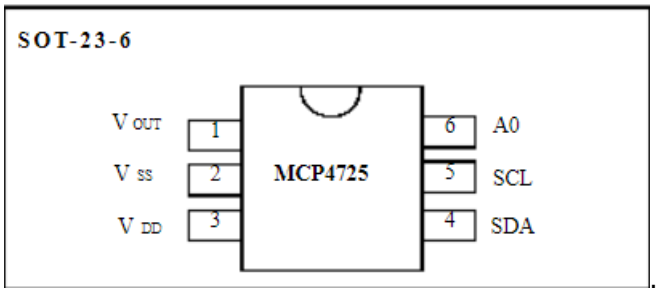
Un avistamiento general de las principales características del DAC MPC 4725 se puede ver en la tabla 1-2.

Tabla 1-2: Características Módulo MPC 4725. (*Electronic Components Datasheet Search*, n.d.)

Resolución de 12 Bit
Memoria no Volátil
Pin de Dirección A0
Voltaje de Operación de 2.7V a 5.5V
Interfaz I2C

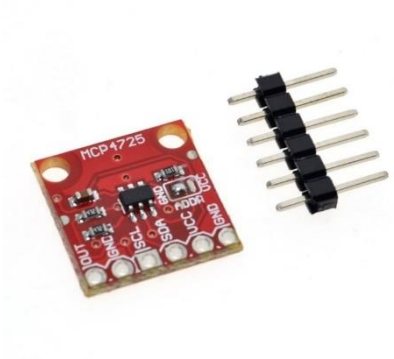
El conversor DAC MPU4725 cuenta con 6 pines, véase la Figura 1.3.

Figura 1.3: Pines del DAC MPC 4725. (*Electronic Components Datasheet Search*, n.d.)



El pin 3 corresponde a la alimentación del módulo, el pin 1 corresponde a la salida de la señal análoga los puertos 4 y 5 permiten la transmisión de los datos por el protocolo I2C, y por último los pines 2 y 6 van a GND en el esquema. La Figura X, se puede ver el módulo comercial.

Figura 1.4: Modulo Comercial MPC 4725. (*AliExpress*, n.d.)



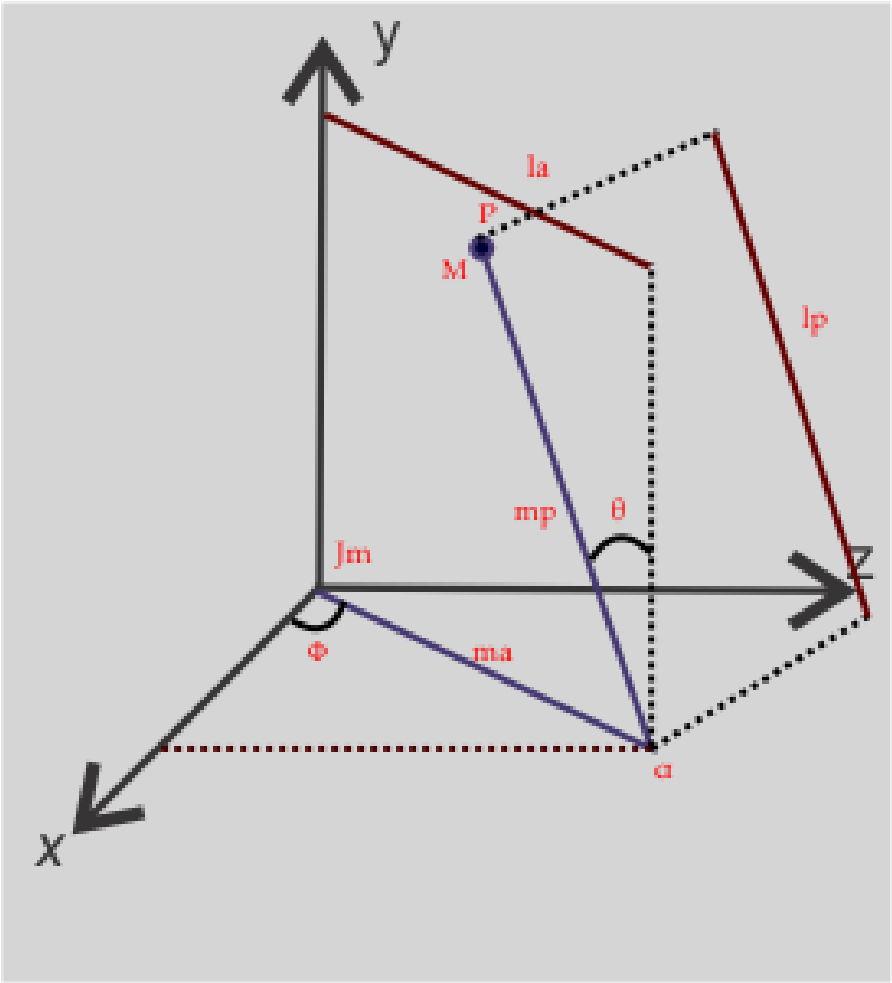
2. Capítulo 2: Modelos matemáticos del Péndulo de Furuta y Helicóptero de Dos Grados de Libertad

En el Capítulo 1 se explicó con detalle la deducción matemática y las ecuaciones que componen el método de Runge- Kutta de Cuarto Orden para un sistema de dos ED de segundo orden acopladas, con el objetivo de encontrar su solución en el dominio del tiempo. Ya teniendo este tema en contexto, en este capítulo se hará una explicación de los dos modelos matemáticos simulados a través del Hardware In The Loop: el Péndulo de Furuta y el Helicóptero de Dos Grados de Libertad.

2.1 Péndulo de Furuta

El Péndulo de Furuta, o Péndulo Invertido Rotacional, es un mecanismo electromecánico que cuenta con la siguiente estructura: se compone principalmente de dos brazos, uno horizontal de longitud l_a y con una masa m_a. A este brazo horizontal, se le añade el eje de un motor cuyo momento de inercia J_m representa el equivalente de la inercia de los brazos y del mismo motor. El movimiento de este brazo se encuentra descrito en los ejes x, y mediante un ángulo φ positivo si va en el sentido horario. Por otra parte, el brazo vertical de longitud l_p y de masa m_p se encuentra unido al brazo horizontal en su otro extremo, y en la punta de este se encuentra una masa M. El movimiento de este brazo se encuentra entre los ejes x, z y se describe a través del ángulo θ, positivo si se mueve en sentido horario. Ambos brazos se cuentan con una masa homogéneamente distribuida, tal y como lo muestra la Figura 2-1 (Navaridas, 2017):

Figura 2.1: Péndulo Invertido Rotacional.



A través del análisis geométrico del péndulo en sus ejes para calcular sus ecuaciones de movimiento, se obtienen una serie de ecuaciones que describen dicho movimiento en los ejes x, y, z. Estas ecuaciones son las 2.1, 2.2 y 2.3 respectivamente:

$$r_x(r_a, r_p) = r_a \cos\varphi - r_p \sin\varphi \sin\theta \quad (2.1)$$

$$r_y(r_a, r_p) = r_a \sin\varphi + r_p \cos\varphi \sin\theta \quad (2.2)$$

$$r_z(r_a, r_p) = r_p \cos\theta \quad (2.3)$$

Para poder obtener el modelo completo a través del tiempo, es necesario hacer un minucioso análisis que resultaría extenso de realizarse por métodos convencionales como las Leyes de Newton, por lo que se hace uso de las ecuaciones cinéticas del sistema, obteniendo la magnitud de la velocidad del sistema derivando con respecto al tiempo las Ecuaciones 2.1, 2.2 y 2.3, aplicando e un análisis vectorial simple para dar como resultado la Ecuación 2.4:

$$v_x{}^2(r_a, r_p) = r_a{}^2 \sin^2\varphi\dot{\varphi}^2 + r_a r_p \sin\varphi\cos\varphi\sin\theta\dot{\varphi}^2 + r_a r_p \sin^2\varphi\cos\theta\dot{\varphi}\dot{\theta} + r_a r_p \sin\varphi\cos\varphi\sin\theta\dot{\varphi}^2 + r_p{}^2\cos^2\varphi\sin^2\theta\dot{\varphi}^2 + r_p{}^2\sin\varphi\cos\varphi\sin\theta\cos\theta\dot{\varphi}\dot{\theta} + r_a r_p \sin^2\varphi\cos\theta\dot{\varphi}\dot{\theta} + r_p{}^2\sin\varphi\cos\varphi\sin\theta\cos\theta\dot{\varphi}\dot{\theta} + r_p{}^2\sin^2\varphi\cos^2\theta\dot{\varphi}^2$$

$$v_y{}^2(r_a, r_p) = r_a{}^2\cos^2\varphi\dot{\varphi}^2 - r_a r_p \sin\varphi\cos\varphi\sin\theta\dot{\varphi}^2 + r_a r_p \cos^2\varphi\cos\theta\dot{\varphi}\dot{\theta} - r_a r_p \sin\varphi\cos\varphi\sin\theta\dot{\varphi}^2 + r_p{}^2\sin^2\varphi\sin^2\theta\dot{\varphi}^2 - r_p{}^2\sin\varphi\cos\varphi\sin\theta\cos\theta\dot{\varphi}\dot{\theta} + r_a r_p \cos^2\varphi\cos\theta\dot{\varphi}\dot{\theta} - r_p{}^2\sin\varphi\cos\varphi\sin\theta\cos\theta\dot{\varphi}\dot{\theta} + r_p{}^2\cos^2\varphi\cos^2\theta\dot{\varphi}^2$$

$$v_z{}^2(r_a, r_p) = -r_p{}^2\sin^2\theta\dot{\theta}^2$$

$$v^2(r_a, r_p) = (r_a{}^2 + r_p{}^2\sin^2\theta)\dot{\varphi}^2 + 2r_a r_p\cos\theta\dot{\varphi}\dot{\theta} + r_p{}^2\dot{\theta}^2 \quad (2.4)$$

Con la Ecuación 2.4, se hace un análisis energético del sistema, necesario para aplicar las Ecuaciones de Lagrange de manera posterior. Este análisis energético, que da como resultado dos expresiones físicas (energía cinética y potencial), ya permiten incorporar los parámetros físicos del sistema (longitud de las articulaciones, sus masas, masa de balance, entre otros) y relacionarlos para dar como resultado las Ecuaciones 2.5 y 2.6:

$$K = \frac{1}{2}\left[J_m\dot{\varphi}^2 + \frac{1}{3}m_a l_a\dot{\varphi}^2 + m_p\left(l_a{}^2 + \frac{1}{3}l_p{}^2\sin^2\theta\right)\dot{\varphi}^2 + m_p l_a l_p\cos\theta\dot{\varphi}\dot{\theta}\right] + \frac{1}{2}\left[\frac{1}{3}m_p l_p{}^2\dot{\theta}^2 + M\left(l_a{}^2 + l_p{}^2\sin^2\theta\right)\dot{\varphi}^2 + 2Ml_a l_p\cos\theta\dot{\varphi}\dot{\theta} + Ml_p{}^2\dot{\theta}^2\right] \quad (2.5)$$

$$U = \frac{1}{2}gm_p l_p\cos\theta + gMl_p\cos\theta \quad (2.6)$$

Donde la Ecuación 2.5 es la energía cinética total del sistema y la Ecuación 2.6 es la energía potencial total del sistema. Finalmente a través las ecuaciones de Lagrange, las cuales hacen uso de estas ecuaciones energéticas junto con las mecánicas para dar como

resultado un sistema de dos ED de segundo orden acopladas, tal y como lo muestran las ecuaciones 2.7 y 2.8 (Navaridas, 2017):

$$(\alpha + \beta sin^2\theta)\ddot{\varphi} + \gamma cos\theta\ddot{\theta} + 2\beta sin\theta cos\theta\dot{\varphi}\dot{\theta} - \gamma sin\theta\dot{\theta}^2 = \tau_\varphi \quad (2.7)$$

$$\gamma cos\theta\ddot{\varphi} + \beta sin\theta cos\theta\gamma cos\theta\dot{\varphi}^2 - \delta sin\theta = \tau_\theta \quad (2.8)$$

Donde:

$$\alpha = J_m + \left(M + \frac{1}{3}m_a + m_p\right){l_a}^2 \quad (2.9)$$

$$\beta = \left(M + \frac{1}{3}m_p\right){l_p}^2 \quad (2.10)$$

$$\gamma = \left(M + \frac{1}{2}m_p\right)l_a l_p \quad (2.11)$$

$$\delta = \left(M + \frac{1}{2}m_p\right)g l_p \quad (2.12)$$

$$\tau_\varphi(t) = \frac{K_t}{R}V(t) - \frac{K_e K_t}{R}\dot{\varphi(t)} \quad (2.13)$$

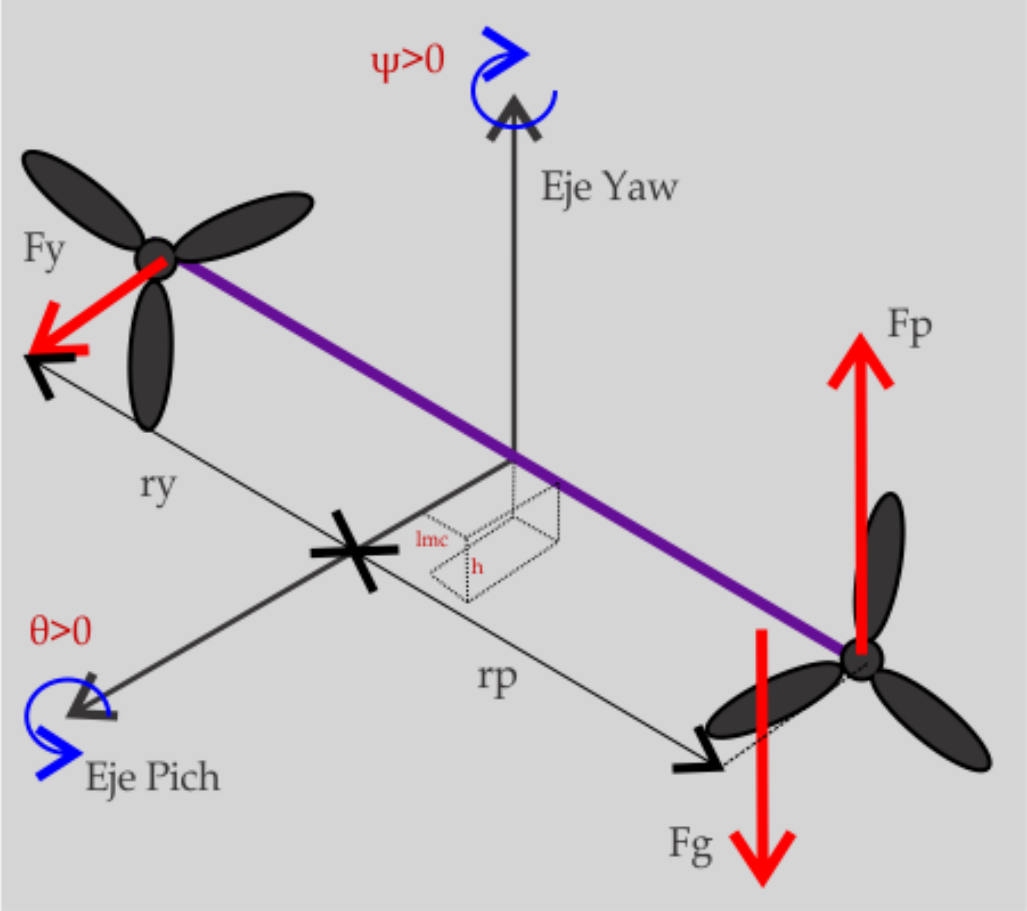
Alpha (Ecuación 2.9), Beta (Ecuación 2.10), Gamma (Ecuación 2.11) y Delta (Ecuación 2.12) son ecuaciones que simplifican las ecuaciones 2.7 y 2.8 en su expresión, mientras que la ecuación 2.13, es un modelo simplificado de un motor eléctrico en función de la velocidad del brazo horizontal φ, que tiene como parámetros unas constantes propias del motor eléctrico (K_t, K_e, R) y una entrada de voltaje $V(t)$ que actuará como entrada del sistema (escalón, rampa, seno, etc.).

2.2 Helicóptero de Dos Grados de Libertad

El Helicóptero de Dos Grados de Libertad es un sistema electromecánico que se describe en su estructura de la siguiente manera: el modelo cuenta con dos hélices que permiten al helicóptero moverse en dos ángulos: un ángulo que hace que el brazo que une las dos hélices se mueva alrededor del eje horizontal θ o Pitch y otro que hace que este brazo gire alrededor del eje vertical φ o Yaw. El ángulo θ es positivo cuando gira en sentido antihorario y el ángulo φ es positivo cuando gira en sentido horario. El brazo horizontal en su longitud se divide en dos distancias: una que va desde el centro hacia la hélice que hace girar el mecanismo en el eje Yaw r_p y una distancia que va desde el centro hasta la hélice que hace girar al mecanismo en el eje Pitch r_y. En ambos ejes, se genera un torque si se aplica unas fuerzas perpendiculares F_p y F_y respectivamente, ambas positivas en sus

respectivos sentidos. Finalmente, hay una fuerza gravitacional F_g que hace que la nariz del helicóptero baje, mientras que el centro de masa se encuentra a una distancia l_{mc} a lo largo del fuselaje y una altura h por debajo de dicho fuselaje(González Vivas, 2011), tal como se muestra en la Figura 2-2.

Figura 2.2: Helicóptero de Dos Grados de Libertad.



A diferencia del Péndulo de Furuta, este modelo matemático cuenta con dos entradas para sus respectivos ejes, y su análisis se reduce a una matriz de transformación de estos que busca las ecuaciones de la cinemática en los ejes x, y, z. Esta matriz, considera los dos ejes de movimiento y describe su rotación y su traslación a lo largo de estos espacios, y viene descrita por la Ecuación 2.14:

$$T_0^2 = Rot_{x0,\varphi} Rot_{y1,\theta} \begin{bmatrix} 1 & 0 & 0 & l_{mc} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -h \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_0^2 = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 & 0 \\ -\sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & l_{mc} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -h \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_0^2 = \begin{bmatrix} \cos\varphi\cos\theta & \sin\varphi & -\cos\varphi\sin\theta & l_{mc}\cos\varphi\cos\theta + h\cos\varphi\sin\theta \\ -\cos\theta\sin\varphi & \cos\varphi & \sin\varphi\sin\theta & -l_{mc}\cos\theta\sin\varphi - h\sin\varphi\sin\theta \\ \sin\theta & 0 & \cos\theta & l_{mc}\sin\varphi - h\cos\theta \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.14)$$

De la matriz resultante 2.14, se toman los elementos 1, 2 y 3 de la cuarta columna, los cuales describen la posición en coordenadas cartesianas del centro de masa del helicóptero en x (Ecuación 2.15), y (Ecuación 2.16) y z (Ecuación 2.17):

$$X_{mc} = (l_{mc}\cos\theta + h\sin\theta)\cos\varphi \quad (2.15)$$

$$Y_{mc} = (-l_{mc}\cos\theta - h\sin\theta)\sin\varphi \quad (2.16)$$

$$Z_{mc} = l_{mc}\sin\theta - h\cos\theta \quad (2.17)$$

A estas ecuaciones cartesianas, se complementa el cálculo del centro de masa de la estructura, que nos es más que la suma de las masas de cada una de las secciones del helicóptero (hélices, barras, etc.) así como sus respectivos centros de masa. Posteriormente, estas ecuaciones se derivan con respecto al tiempo y la magnitud de la velocidad del sistema se expresa mediante la Ecuación 2.18:

$$v^2 = \left(h^2 + \cos^2\theta\left({l_{mc}}^2 - h^2\right) + l_{mc}h\sin 2\theta\right)\dot{\varphi}^2 + \left({l_{mc}}^2 + h^2\right)\dot{\theta}^2 \quad (2.18)$$

Una vez obtenida esta ecuación, se hace un análisis energético del sistema, donde se obtienen las ecuaciones que representan la energía cinética total y la energía potencial total expresadas a través de las Ecuaciones 2.19 y 2.20 respectivamente:

$$P = M_{heli}g(l_{mc}\sin\theta - h\cos\theta) \quad (2.19)$$

$$T_t = \frac{1}{2}M_{heli}\left[\left(h^2 + \cos^2\theta\left({l_{mc}}^2 - h^2\right) + l_{mc}h\sin 2\theta\right)\dot{\varphi}^2 + \left({l_{mc}}^2 + h^2\right)\dot{\theta}^2\right] \quad (2.20)$$

Finalmente, a través de un análisis de las Ecuaciones 2.19 y 2.20 mediante las ecuaciones de Lagrange, se obtiene un sistema de dos ED de segundo orden acopladas, tal y como lo muestran las ecuaciones 2.21 y 2.22(González Vivas, 2011):

$$(J_{eqp} + M_{heli}({l_{mc}}^2 + h^2))\ddot{\theta} + M_{heli}[\frac{sin2\theta({l_{mc}}^2 - h^2)}{2} - l_{mc}hcos2\theta]\dot{\varphi}^2 + M_{heli}g(l_{mc}cos\theta + hsin\theta) + B_p\dot{\dot{\theta}} = \left(K_{pp}V_{mp} + F_{cpp}\right) + (K_{py}V_{my} + F_{cpy}) \quad (2.21)$$

$$[J_{eqy} + M_{heli}[cos^2\theta\left({l_{mc}}^2 - h^2\right) + l_{mc}hsin2\theta + h^2]]\ddot{\varphi} + M_{heli}[sin2\theta\left(h^2 - {l_{mc}}^2\right) + 2l_{mc}hcos2\theta]\dot{\theta}\dot{\varphi} + B_y\dot{\varphi} = \left(K_{yp}V_{mp} + F_{cyp}\right)cos\theta + (K_{yy}V_{my} + F_{cyy})cos\theta \quad (2.22)$$

Las ecuaciones 2.21 y 2.22 se componen principalmente de constantes relacionadas con el helicóptero obtenidas de manera experimental(González Vivas, 2011) y unos voltajes de entrada V_{mp} y V_{my}, los cuales actúan como entradas (tanto por el lado de Pitch como de Yaw) del sistema (escalón, rampa, seno, etc.)

3. Capítulo 3: Simulación en el tiempo del Péndulo de Furuta y Helicóptero de dos grados de libertad (Uso del método de RK4O)

En

3.1 Variables de estado de los sistemas

En

3.1.1 Variables de estado del Péndulo de Furuta

P
a
r
a

e
l

$$X = [\varphi \quad \dot{\varphi} \quad \theta \quad \dot{\theta}]^T$$

Las Ecuaciones 2.7 y 2.8, al ser un sistema de dos ecuaciones con dos incógnitas, se

c
a
s
p
i
d
e
l

$$\frac{d\varphi}{dt} = \dot{\varphi} \tag{3.1}$$

l
P
é
n
d

$$\ddot{\varphi} = \frac{\beta(\tau_{\varphi}-b\dot{\varphi})-\beta\gamma cos^2\theta sin\theta\dot{\varphi}^2-2\beta^2 cos\theta sin\theta\dot{\varphi}\dot{\theta}}{\alpha\beta-\gamma^2+(\beta^2+\gamma^2)sin^2\theta} + \frac{\beta\gamma sin\theta\dot{\theta}^2-\gamma\delta cos\theta sin\theta}{\alpha\beta-\gamma^2+(\beta^2+\gamma^2)sin^2\theta} \tag{3.2}$$

$$\frac{d\theta}{dt} = \dot{\theta} \tag{3.3}$$

$$\ddot{\theta} = \frac{\beta(\alpha+\beta sin^2\theta)cos\theta sin\theta\dot{\varphi}^2+2\beta\gamma cos^2\theta sin\theta\dot{\varphi}\dot{\theta}}{\alpha\beta-\gamma^2+(\beta^2+\gamma^2)sin^2\theta} + \frac{-\gamma^2 cos\theta sin\theta\dot{\theta}^2+\delta(\alpha+\beta sin^2\theta)sin\theta-\gamma^2 cos\theta(\tau_{\varphi}-b\dot{\varphi})}{\alpha\beta-\gamma^2+(\beta^2+\gamma^2)sin^2\theta} \tag{3.4}$$

3.1.2 Variables de estado del Helicóptero de dos grados de libertad

Para el Helicóptero

$$X = [\theta \quad \varphi \quad \dot{\theta} \quad \dot{\varphi}]^T$$

Para las Ecuaciones 2.21 y 2.22, se realiza el mismo procedimiento que con el Péndulo de Furuta, se resuelve el sistema de dos ecuaciones con dos incógnitas para finalmente obtener

$$\frac{d\theta}{dt} = \dot{\theta} \tag{3.5}$$

$$\frac{d\varphi}{dt} = \dot{\varphi} \tag{3.6}$$

$$\ddot{\theta} = -\frac{M_{heli}\left[\frac{sin2\theta(l_{mc}{}^2-h^2)}{2}-l_{mc}hcos2\theta\right]\dot{\varphi}^2}{\left(J_{eqp}+M_{heli}(l_{mc}{}^2+h^2)\right)} - \frac{M_{heli}g(l_{mc}cos\theta+hsin\theta)+B_p\dot{\theta}-(K_{pp}V_{mp}+F_{cpp})-(K_{py}V_{my}+F_{cpy})}{\left(J_{eqp}P+M_{heli}(l_{mc}{}^2+h^2)\right)} \tag{3.7}$$

$$\ddot{\varphi} = -\frac{M_{heli}[sin2\theta(h^2-l_{mc}{}^2)+2l_{mc}hcos2\theta]\dot{\theta}\dot{\varphi}}{\left[J_{eqy}+M_{heli}[cos^2\theta(l_{mc}{}^2-h^2)+l_{mc}hsin2\theta+h^2]\right]} - \frac{B_y\dot{\varphi}-(K_{yp}V_{mp}+F_{cyp})cos\theta-(K_{yy}V_{my}+F_{cyy})cos\theta}{\left[J_{eqy}+M_{heli}[cos^2\theta(l_{mc}{}^2-h^2)+l_{mc}hsin2\theta+h^2]\right]} \tag{3.8}$$

3.2 Desarrollo del algoritmo Runge- Kutta de Cuarto orden

El algoritmo diseñado para este proyecto reproduce el método numérico de Runge- Kutta de Cuarto Orden para los dos sistemas explicados con anterioridad (Péndulo de Furuta y Helicóptero de dos grados de libertad). Para ello, partiendo de lo visto en la sección 3.1, al obtener el sistema de cuatro ecuaciones en los dos casos, se procede a diseñar el algoritmo que pueda simular sus comportamientos en el tiempo tanto en posición como en velocidad. A pesar que cada sistema cuenta con particularidades concernientes a estas, el algoritmo en modo general es el mismo. Las consideraciones específicas para cada sistema se contemplarán en las siguientes subsecciones.

El método de Runge- Kutta, debido a sus características matemáticas, resulta ser un método finito, es decir, requiere de manera obligatoria un tiempo inicial de simulación y un tiempo final. Sin embargo, la aplicación para este proyecto, que se busca una simulación en tiempo real, se hace una modificación adicional que permite que se efectúe la simulación de manera “infinita” (o hasta que el usuario decida finalizarla) estableciendo un periodo de tiempo constante que cuando finaliza, le pregunta al usuario si desea continuar o no. Esto garantiza, que no solo se ejecute indefinidamente, sino que el método de Runge-Kutta cuente con un intervalo constante y por ende, una cantidad determinada de operaciones que le permita ejecutarse a cabalidad. El diagrama de flujo general del algoritmo se puede observar en el Anexo A y su descripción es la siguiente:

1. Ingresar los parámetros físicos del sistema (masa, longitud, momento de inercia, entrada o entradas del sistema, etc.). Así mismo, se ingresan las condiciones iniciales de las variables dependientes y sus derivadas ($\theta, \varphi, \dot{\theta}, \dot{\varphi}$). Tener en consideración el sistema de ED que se debe solucionar y que el tiempo final es diez segundos después del tiempo inicial.

2. Calcular el número de iteraciones restando el tiempo inicial con el final y el resultado dividirlo entre el paso.

3. Declarar vectores vacíos que contienen los valores de $t, \theta, \varphi, \dot{\theta}, \dot{\varphi}$.

4. Inicia la simulación.

5. Los valores de $t, \theta, \varphi, \dot{\theta}, \dot{\varphi}$ se almacenan en la siguiente posición de cada vector (primera si es la primera iteración) y en unas variables auxiliares usadas para el método $(t_0, \theta_0, \varphi_0, \dot{\theta}_0, \dot{\varphi}_0)$.

6. Calcular $k_1, m_1, l_1, b_1, k_2, m_2, l_2, b_2, k_3, m_3, l_3, b_3, k_4, m_4, l_4, b_4$ como en las Ecuaciones 1.14.1, 1.14.2, 1.14.3 y 1.14.4 (en el caso de k_n), 1.15.1, 1.15.2, 1.15.3 y 1.15.4 (en el caso de m_n), 1.16.1, 1.16.2, 1.16.3 y 1.16.4 (en el caso de l_n) y 1.17.1, 1.17.2, 1.17.3 y 1.17.4 (en el caso de b_n), teniendo en cuenta la consideración de evaluar la función de cada letra correspondiente $f(t_n, x_n, \dot{x}_n, y_n, \dot{y}_n), g(t_n, x_n, \dot{x}_n, y_n, \dot{y}_n), z(t_n, x_n, \dot{x}_n, y_n, \dot{y}_n), j(t_n, x_n, \dot{x}_n, y_n, \dot{y}_n)$.

7. Declarar un vector auxiliar que albergue los valores del siguiente paso como en las Ecuaciones 1.14.1, 1.14.2, 1.14.3 y 1.14.4 (en el caso de k_n), 1.15.1, 1.15.2, 1.15.3 y 1.15.4 (en el caso de m_n), 1.16.1, 1.16.2, 1.16.3 y 1.16.4 (en el caso de l_n) y 1.17.1, 1.17.2, 1.17.3 y 1.17.4 (en el caso de b_n), se hace uso de las variables auxiliares del paso 5 para los cuatro pasos calculados en el paso 6.

8. Calcular los valores de $\theta, \varphi, \dot{\theta}, \dot{\varphi}$ de la iteración como en las Ecuaciones 1.14, 1.15, 1.16 y 1.17. Aumenta en uno la variable auxiliar que monitorea las iteraciones

9. Al mismo tiempo, graficar el comportamiento de las cuatro variables mencionadas en el paso 2 versus el tiempo.

10. Repetir los pasos 5 al 9 hasta que se iguale el número de iteraciones calculados en el paso 4. Cuando llegue allí, responder si quiere continuar o no. Si continúa, la variable auxiliar que monitorea las iteraciones aumenta en uno y se repiten nuevamente los pasos 5 al 9. Si no, la simulación finalizará.

11. Cuando la variable auxiliar sea múltiplo del número de iteraciones, responder si desea continuar o no. Se ejecutarán los pasos del 5 al 9 hasta que el usuario decida no continuar. Si no, la simulación finalizará.

3.2.1 Consideraciones específicas al algoritmo RK4O para el Péndulo de Furuta

Para la formulación del algoritmo RK4O para el Péndulo de Furuta, existen ciertas condiciones específicas para tener en consideración al momento de ejecutarlo. Las condiciones son las siguientes:

1. En el paso número 1, los parámetros físicos que deben ingresarse son los siguientes.

Tabla 3-1. Parámetros físicos y condiciones iniciales del Péndulo de Furuta (Navaridas, 2017).

Parámetro	Valor
m_a	0 kg
l_a	0,109 m
m_p	0,0033 m
l_p	0,1832 m
M	0 kg
g	9,81 $\frac{m}{s^2}$
J_m	0,00097 kgm^2
R	9 Ω
K_t	0,26 Nm/A
K_e	0,26 Nm/A
b	0,0022 Nm/(rad/s)
θ	0 rad
$\dot{\theta}$	0 rad/s
φ	0,1 rad
$\dot{\varphi}$	0 rad/s

Según Navarridas (Navaridas, 2017), los parámetros físicos del péndulo en cuestión pertenecen al péndulo fabricado por Quanser, quienes en su manual de usuario

especifican los valores para esta estructura, mientras que las condiciones iniciales del sistema fueron establecidas de manera intencional, especialmente en el brazo vertical, donde se ubica en un ángulo de 0,1 radianes para evitar que inicie directamente en una posición que le costaría sostener de inmediato. En cuando a los parámetros del motor, fueron hallados de manera experimental por el autor con base en el motor Lego XL. Para la entrada del sistema, se considerará solo una, la cual va relacionada al motor ubicado en el brazo horizontal.

2. En el paso número 2, el tamaño del paso utilizado fue de 0,121. Este paso fue asignado para este modelo de manera experimental, con el objetivo de obtener el comportamiento más fiel al sistema real, por ello es que este valor cuenta con tres cifras significativas. Las comparaciones relacionadas con el comportamiento físico del sistema se contemplarán en la sección …

3.2.2 Consideraciones específicas al algoritmo RK4O para el Helicóptero de dos grados de libertad

Al igual que el péndulo de Furuta, el Helicóptero de dos grados de libertad cuenta con unas consideraciones específicas que deben ser tenidas en cuenta a la hora de formular el algoritmo para este sistema. Las consideraciones son las siguientes:

1. En el paso 1, los valores de los parámetros físicos son los siguientes:

Tabla 3-2. Parámetros físicos y condiciones iniciales del Helicóptero de dos grados de libertad (Gonzalez Vivas, 2011).

Parámetro	Valor
B_p	0,01325 N/V
B_y	0,8513 N/V
M_{heli}	1,3872 kg
J_{eqp}	0,0332 kgm^2
J_{eqy}	0,0371 kgm^2
l_{mc}	0,0122 m
h	0,00714 m

θ	0 rad
$\dot{\theta}$	0 rad/s
φ	0 rad
$\dot{\varphi}$	0 rad/s

Según Vivas (Gonzalez Vivas, 2011), el modelo de helicóptero analizado fue el fabricado por Quanser, sin embargo, los parámetros anteriormente mencionados fueron hallados de manera experimental por el autor para obtener mayor fidelidad en el modelo propuesto, encontrando que algunos de estos son diferentes a los propuestos por la empresa fabricante, por lo que al igual que el autor, este proyecto se decantará por los hallados de manera experimental. Así mismo, si bien el algoritmo considera una entrada, en este modelo en particular se considerarán dos: una para Pitch y otra para Yaw que puede ser escalón, rampa o seno.

2. Para el paso 2, el tamaño del paso se estableció en 0,01. Este valor se obtuvo de manera experimental al hacer las pruebas del algoritmo, hallando que si se establecía un valor más grande (0,1 por ejemplo), el sistema se desestabiliza y no tiene el comportamiento esperado. Esto se justifica debido a que cuando en tiempo discreto se toma un tiempo de muestreo muy grande (superior a $1s$) o muy pequeño (del orden de $10^{-4}\,s$, el sistema se puede desestabilizar y no recuperarse. Adicionalmente, el número de iteraciones se mantuvo en 100 de manera forzada alterando el intervalo de simulación de 0 a 1 segundos, ya que, si se mantiene en 10 segundos (ver paso 1 del algoritmo), el número de iteraciones aumenta considerablemente y el gasto computacional es mayor, por lo que afectaría seriamente los procesos del hardware que se explicarán con detalle en el siguiente capítulo.

3.3 Implementación del algoritmo RK4O en Python y validación en Matlab y Simulink

El algoritmo explicado con anterioridad en la sección 3.2 bajo las condiciones especificadas en la sección 3.3 es implementado en el lenguaje de programación Python, ya que este cuenta con las herramientas necesarias para el desarrollo de procesamiento de datos, además de un tiempo de desarrollo más corto debido a sus propiedades de lenguaje de alto nivel. A su vez, este código es ejecutado en la Raspberry PI 3B+, hardware de bajo costo utilizado para el proyecto debido a su compatibilidad con Python y su capacidad de procesamiento. Comparando con otras tarjetas que fueron candidatas como la ESP32 (Guerra Carmenate, 2022) o STM32F205 (STMicroelectronics, 2020), las propiedades y características de las Raspberry en cuanto a procesamiento y facilidad de desarrollo son superiores, tal y como se muestra en la **Tabla 3-3**.

Tabla 3-3. Tabla comparativa de características entre las tarjetas embebidas candidatas (Eugenio Lopez Aldea, 2017; Guerra Carmenate, 2022; STMicroelectronics, 2020)

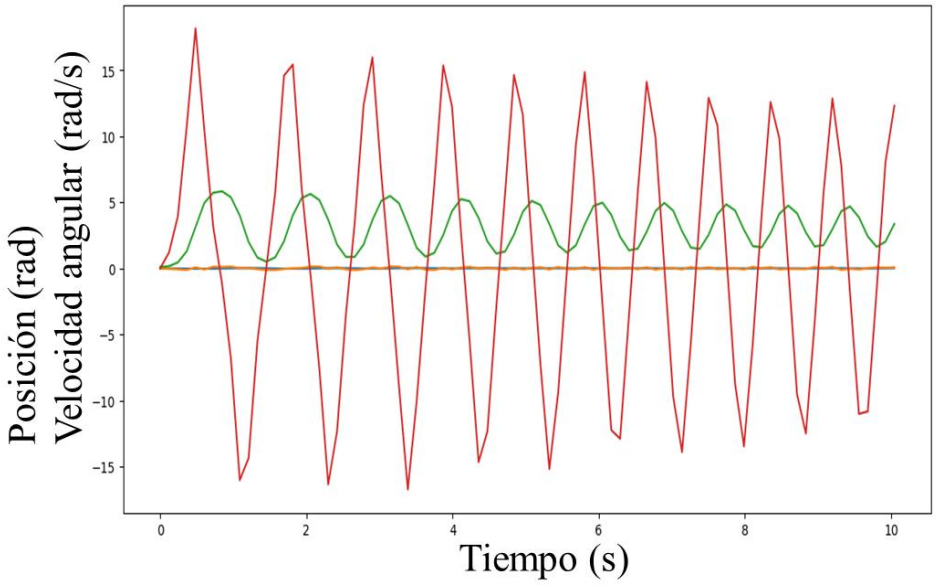
Tarjeta Embebida	Procesador	Pines GPIO	Desarrollo con Python
Raspberry PI 3B+	Broadcom BCM2837B0, Cortex-A53 64-bit SoC, 1GB LPDDR2 SDRAM	40	Soporte complete con Python 3, con acceso completo a Tkinter.
ESP32	Xtensa Dual-Core 32-bit LX6 con 600 DMIPS, 448 KB	34	MicroPython con soporte básico para el desarrollo de interfaces GUI
STM32F205	Arm® 32-bit Cortex®-M3 CPU (120 MHz max)	50	MicroPython con soporte básico para el desarrollo de interfaces GUI

Teniendo en cuenta que cada modelo posee condiciones especiales para su ejecución, se escribe un código para cada modelo con el objetivo de evitar errores en la

simulación producidos por la generalización de condiciones específicas tales como el tamaño del paso, parámetros físicos y número de entradas. Los códigos de cada modelo se encuentran especificados en el Anexo A, donde la implementación del algoritmo se lleva a cabo en una función específica llamada **Runge_Kutta_1** (para el caso del Péndulo de Furuta) y **Runge_Kutta_2** (para el caso del Helicóptero de dos grados de libertad). Los demás aspectos tales como las gráficas animadas y la recepción y transmisión de los datos calculados se ahondarán con detalle en el siguiente capítulo.

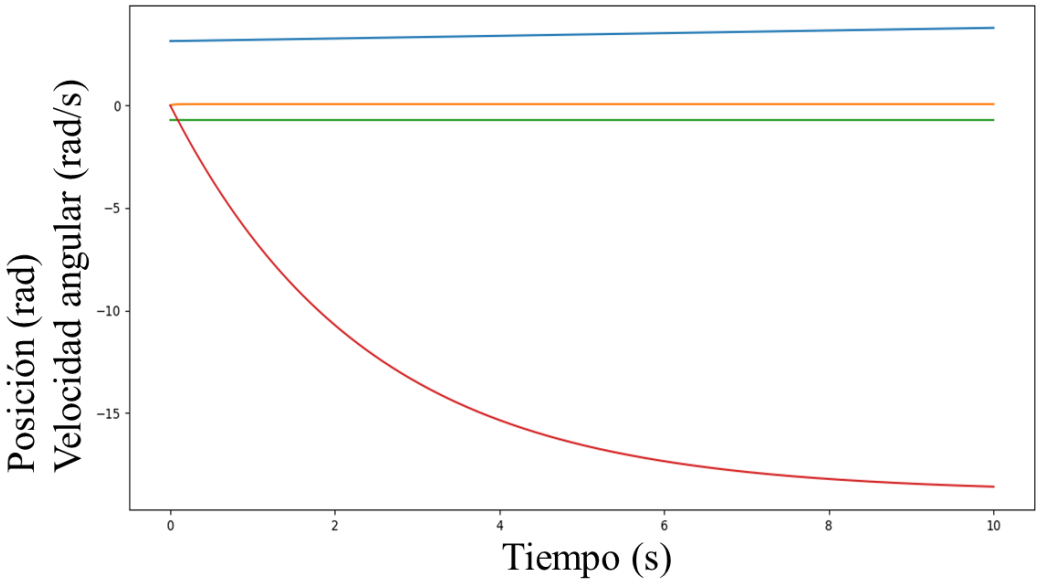
Para el desarrollo de los códigos, se hace uso del IDE específico para Python con el que cuenta la Raspberry PI 3B+, el Python 3 IDLE. Allí, se disponen de las librerías necesarias para el procesamiento y almacenamiento de los datos en arrays (numpy), funciones matemáticas para las operaciones de las ecuaciones (math) y su posterior graficación (matplotlib), tal como se muestra en la Figura **Figura 3.1** y **Figura 3.2** para el Péndulo de Furuta y el Helicóptero de dos grados de libertad respectivamente.

Figura 3.1: Gráfica en el tiempo del Péndulo de Furuta.



Nota: Línea verde: posición de brazo vertical. Línea roja: velocidad de brazo vertical. Línea azul: Posición de brazo horizontal. Línea naranja: velocidad en brazo horizontal.

Figura 3.2. Gráfica en el tiempo del Helicóptero de dos grados de libertad.



Nota: Línea verde: velocidad en pitch. Línea roja: velocidad en Yaw. Línea azul: Posición de Yaw. Línea naranja: Posición en Pitch.

3.3.1 Validación del Pendulo de Furuta en Simulink

Una vez se implementa y se ejecuta en la Raspberry PI 3B+, se hace la validación del algoritmo para cada modelo a través del montaje de estos en dos softwares especializados: Simulink.

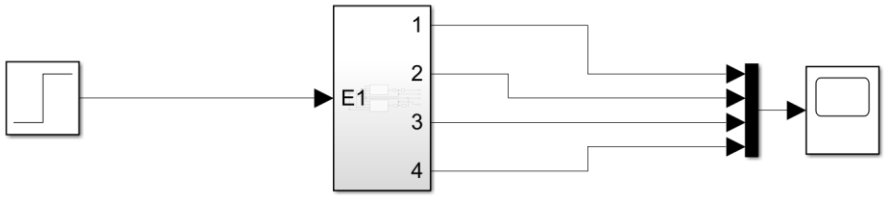
En este caso, Simulink cuenta con un bloque especial (Matlab Function) que sirve para la definición de funciones personalizadas. Este bloque le permite escribir desde el “script” de Matlab una función para ser utilizada en el modelo de Simulink., siendo posible representar las ecuaciones de cada sistema de una forma más versátil.

El bloque permite especificar entradas y salidas. Para el uso del bloque se debe determinar el nombre de la función y cuáles van a ser las entradas y salidas. Ya dentro del editor del bloque de funtion Matlab, aparece una función pre determinada y ahí es donde se va construyendo la función que se necesita. En este punto se implementó un bloque para cada par del sistema de ecuaciones diferenciales.

Cabe mencionar que para que la simulación del modelo quede correcta también es necesario ajusta el “Solver” con que el solucionador de Simulink ejecuta la solución. Para los dos sistemas se aplicó la misma configuración del “solver”. Siendo esta un tipo de paso fijo con solucionador Ode4 (Runge Kutta), en el modelado del Péndulo de Furuta el valor del paso fijo fue de 0.121.

En la Figura **Figura 3.3** se podrá ver el montaje correspondiente al modelado en Simulink, estando dentro del subsistema el bloque de Function Matlab.

Figura 3.3: Modelo del Péndulo de Furuta en Simulink.



Abordando ya la construcción de la función dentro del bloque ya mencionado, se genera el código en C que desde Simulink se ejecutara luego. Presentando a continuación el código, es de resaltar que es muy parecido al del script de Matlab.

En primer lugar, se tiene la función predeterminada a la que se le asigna un nombre, los atributos de la función, siendo las entradas los atributos.

```
function salida1=fcn(theta,vtheta,vphi,E1)
```

Después, se ingresaron todos los valores correspondientes a parámetros del sistema, la ecuación correspondiente al comportamiento del motor y demás ecuaciones que simplifican parámetros de las ecuaciones del modelo.

```
ma=0;
la=0.109;
mp=0.0033;
```

```
lp=0.1832;
M=0;
G=9.81
Jm=0.00097;
R=9;
Kt=0.26;
Ko=0.26;
b=0.0022;
T0= ((Kt/R)*E1)-((Ko*Kt)/R)*(vphi);

alpha= Jm + ((M+(1/3)*ma+mp)*(la)^2)
betha=((M+(1/3)*mp)*((lp)^2))
gamma=((M+(1/2)*mp)*(la)*(lp))
deltha=((M+(1/2)*mp)*(G)*(lp))
```

Por último, se ingresa la ecuación diferencial en la salida de la función del bloque. En el caso del Péndulo de Furuta se crearon dos bloques de Matlab Function independientes para cada par de sistemas de ecuaciones, a consecuencia, se cada ecuación representativa de cada uno de los ejes se ingresa en bloques apartes.

Esta línea corresponde a la salida de la posición en Phi.

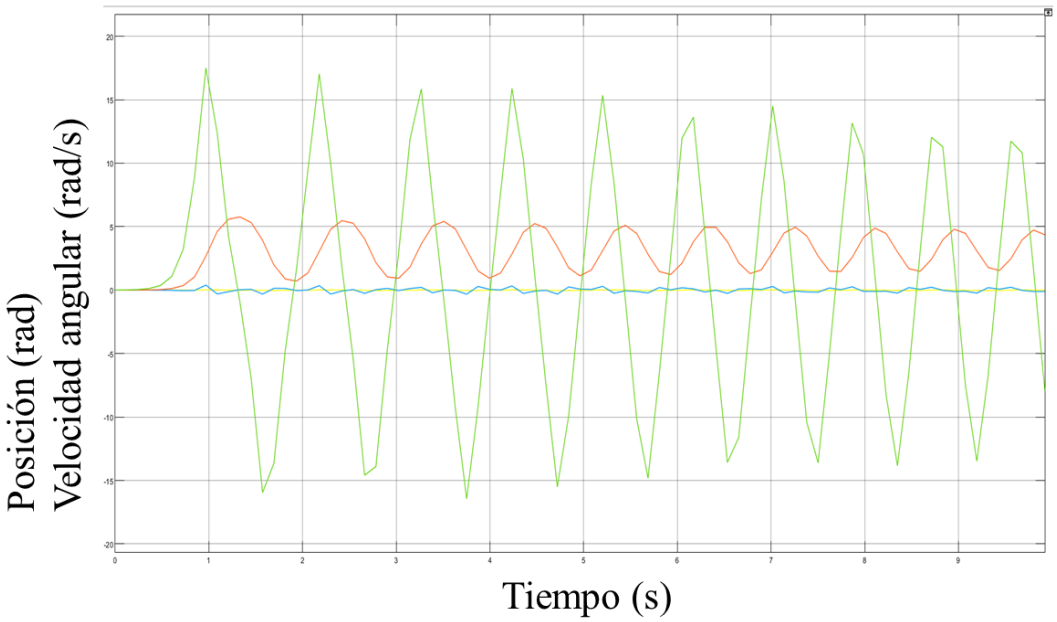
```
salida1=((betha*(T0-b*vphi)-
betha*gamma*((cos(theta))^2)*(sin(theta)*((vphi)^2))-
2*((betha)^2)*(cos(theta)*(sin(theta)*(vphi)*(vtheta))))/((alpha*betha)-
((gamma)^2)+((betha^2)+(gamma^2))*(sin(theta))^2)) +
(((betha)*(gamma)*(sin(theta)*(vtheta^2))-
(gamma)*(deltha)*(cos(theta))*(sin(theta)))/((alpha*betha)-
((gamma)^2)+((betha^2)+(gamma^2))*(sin(theta))^2))
```

Y esta segunda línea corresponde a la salida de la posición en Theta.

```
salida2=(((betha*(alpha+betha*sin(theta)^2)*cos(theta)*(sin(theta)*(vphi
^2))+2*(betha)*(gamma)*(cos(theta)^2)*(sin(theta)*(vphi)*(vtheta)))/((al
pha*betha)-((gamma)^2)+((betha^2)+(gamma^2))*(sin(theta))^2))) + ((-
(gamma^2)*cos(theta)*(sin(theta)*(vtheta)^2)+deltha*(alpha+betha*(sin(th
eta)^2))*sin(theta)-(gamma^2)*cos(theta)*(T0-b*vphi))/((alpha*betha)-
((gamma)^2)+((betha^2)+(gamma^2))*(sin(theta))^2))
```

Finalmente se determinó que se obtuvo un comportamiento homogéneo en las simulaciones de los tres diferentes softwares (Python, Matlab, Simulink). Concluyendo de ese modo que la implementación del algoritmo RK4O para el Péndulo de Furuta quedo correctamente implementado dentro del hardware de bajo costo. En la Figura 3.5 se evidencian los resultados en Simulink.

Figura 3.4: Gráfica del comportamiento del Péndulo de Furuta en Simulnk.



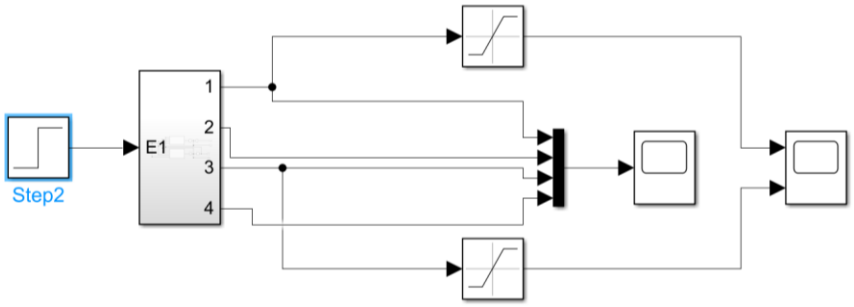
Nota: Línea amarilla: posición en brazo horizontal. Línea azul: velocidad en brazo horizontal. Línea verde: Posición en brazo vertical. Línea Roja: velocidad en brazo vertical.

En referencia al sentido físico de las gráficas, se debe tener presente que, en el Péndulo de Furuta físicamente inicia con el brazo correspondiente al ángulo Theta en una posición vertical, y que al descolgarse se genera una oscilación que va disminuyendo que hasta que alcanza una posición de equilibro que correspondería a Pi radianes, que es exactamente lo que sucede en la Figura 3-6 en la gráfica de color verde, mientras que la posición de Phi, que corresponde al brazo horizontal, gráfica de color amarilla, no presenta un cambio de posición.

Después de comprobado el sistema en lazo abierto sin ninguna restricción, se agrega un bloque de “Saturation” en modelo de Simulink. Esto debido a que en la vida real El funcionamiento del Péndulo de Furuta no tendrá toda esa libertad en el movimiento. En la **Figura 3-6** se podrá observar el esquema del modelado con el nuevo bloque.

Para cada bloque de “Saturation” se utiliza un valor de saturación diferente, ya que las restricciones para el ángulo “Theta” y el ánulo “Phi” cambian. Para el caso del eje horizontal la restricción que se ajusto fue de -360 a 360 grados y para el caso del eje vertical la restricción que se ajusto fue de -5 a 5 grados.

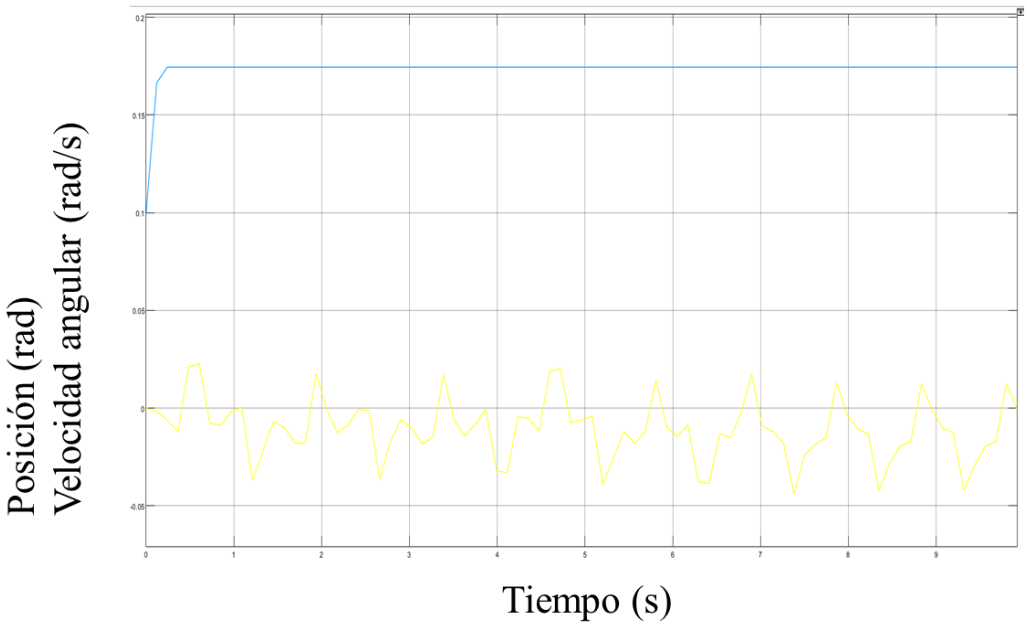
Figura 3.5: Modelo del Péndulo de Furuta con Saturación.



En este punto se valida si dichos bloques de saturación cumplen su cometido. Para eso se puede ver en la Figura 3.7 y la Figura 3.8 El resultado de la validación de la prueba. Para la primera figura se aplicó un step de 0 por lo que para el eje que corresponde al ángulo “Phi” no hay una variación en la posición, sin embargo, el eje que corresponde al ángulo vertical, puede verse que debido a la dinámica varia la posición llegando al punto de la saturación en un corto tiempo.

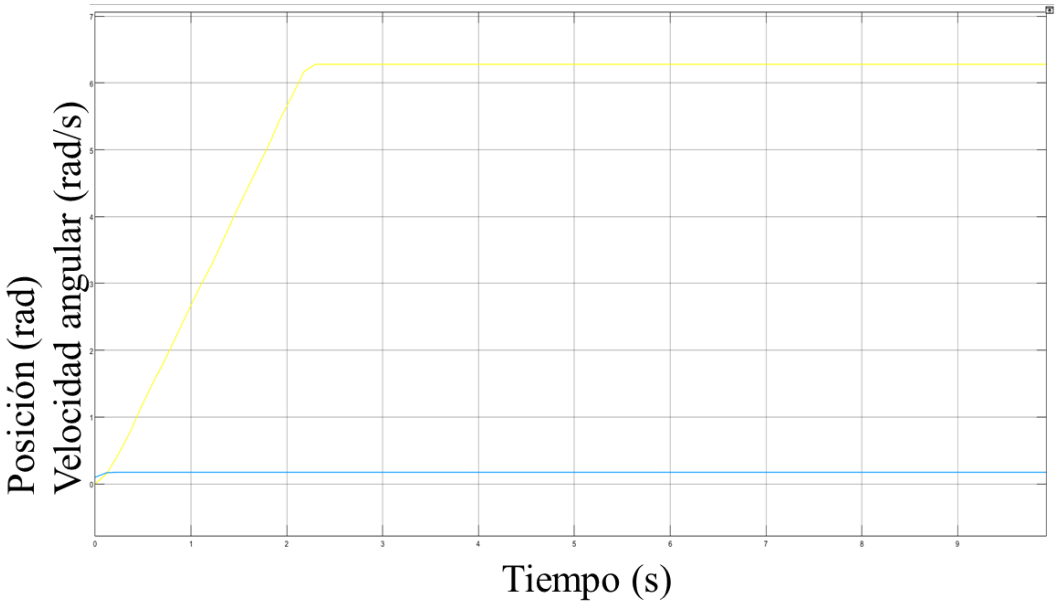
Para la segunda figura se aplicó un step de 1, acá los dos ángulos llegan al punto de saturación.

Figura 3.6: Gráfica del comportamiento del Péndulo de Furuta con Bloque de Saturation y Step de 0.



Nota: Línea amarilla: posición en brazo horizontal. Línea azul: posición en brazo vertical.

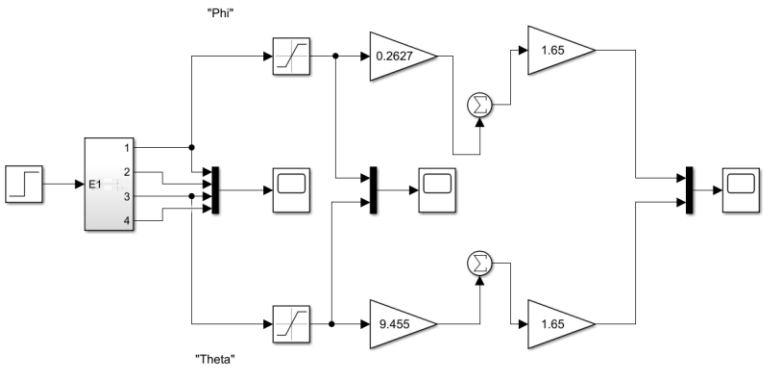
Figura 3.7: Gráfica del comportamiento del Péndulo de Furuta con Bloque de Saturation y Step de 1.



Nota: Línea amarilla: posición en brazo horizontal. Línea azul: posición en brazo vertical. Eje X: tiempo en segundos. Eje Y: Respuesta de posición en radianes o velocidad en radianes sobre segundo.

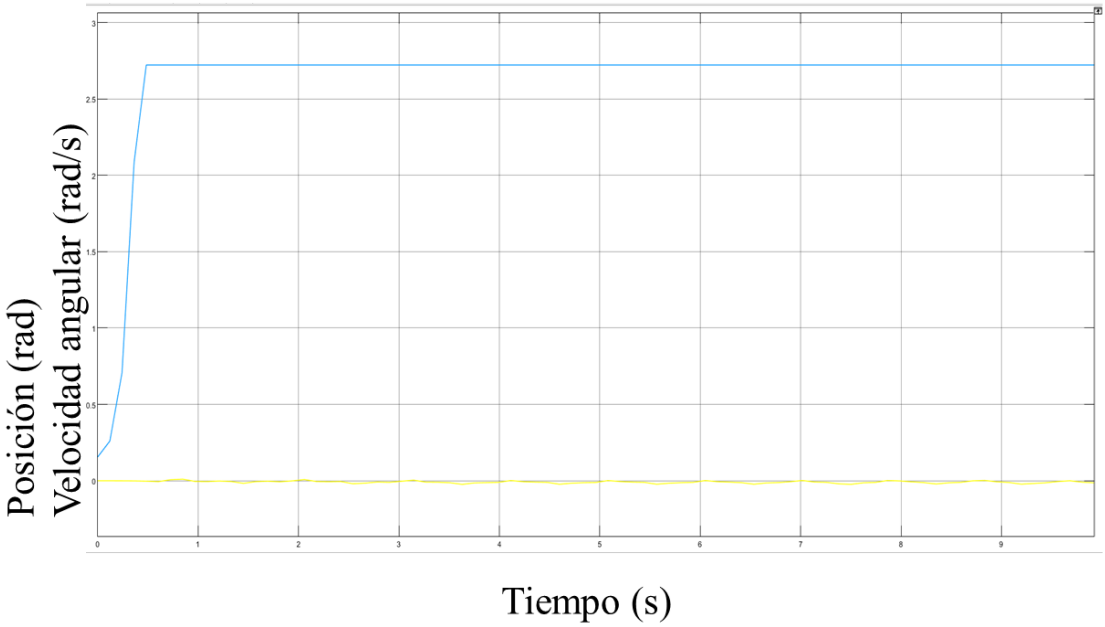
Para terminar el proceso de verificación de los resultados, se debió hacer un ajuste en la salida, para que la interpretación del modelo de la Figura 3.6, representa una salida en radianes, por lo que se realizó una conversión de esos valores en radianes a voltaje, implementando la ecuación que se verá en el modelo de la Figura 3.9. El proceso para determinar los valores para realizar dicha conversión se encuentra en la sección 4.2.5.1.2.

Figura 3.8: Modelo del Péndulo de Furuta en Voltaje.



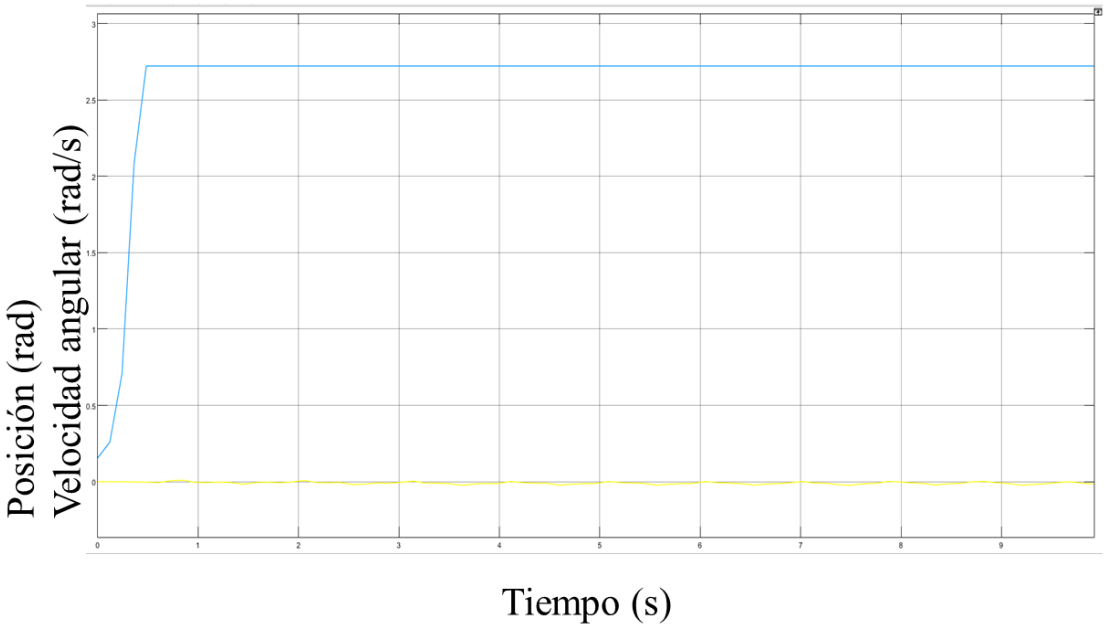
Ya en este punto se puede ver el comportamiento del Péndulo de Furuta en función del voltaje con un step de 0 y un step de 1 en las Figura 3.10 y Figura 3.11 correspondientemente.

Figura 3.9: Gráfica Final del Comportamiento del Péndulo de Furuta en Voltaje – Step de 0.



Nota: Línea amarilla: posición en brazo horizontal. Línea azul: posición en brazo vertical. Eje X: tiempo en segundos. Eje Y: Respuesta de posición en radianes o velocidad en radianes sobre segundo.

Figura 3.10: Gráfica Final del Comportamiento del Péndulo de Furuta en Voltaje - Step de 1.



Nota: Línea amarilla: posición en brazo horizontal. Línea azul: posición en brazo vertical.

3.3.2 Validación del Helicóptero de dos grados de libertad en Simulink.

Para el modelado del sistema del helicóptero de dos grados de libertad se utilizó el mismo principio que en el modelo del Péndulo de Furuta. Ya que, para implementar las ecuaciones del modelo, también se hizo uso del bloque de Function Matlab. La diferencia en este montaje radica que en ya no se utilizaron dos de los bloques ya mencionados, sino que, en vez de eso, se simplifico la estructura ingresando las dos ecuaciones en un solo bloque, como se verá en la Figura 3.12 y Figura 3.13.

Para la simulación del Helicóptero de dos grados de libertad se debió ajustar la configuración del “Solver” por un paso tipo de paso fijo con un valor de 0.1

Figura 3.11: Gráfica del Modelo del Helicoptero de dos grados de libertad en Simulink.

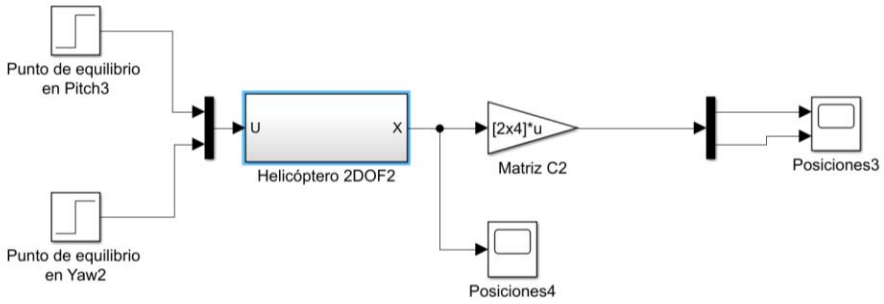
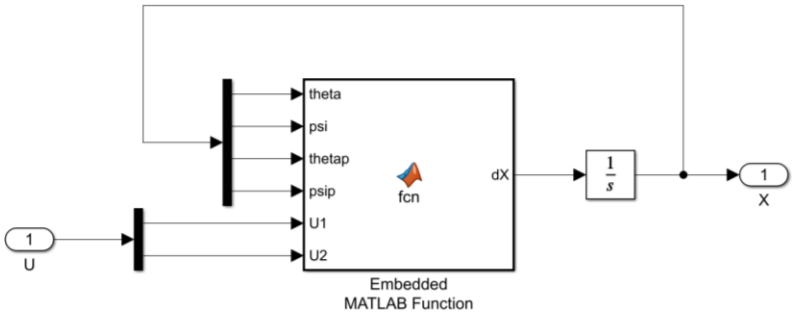


Figura 3.12: Bloque de Matlab Funtion del Helicoptero de dos grados de libertad.



Conociendo ya el diseño, se implementa el código en C que se ejecutara desde el simulador de Simulink.

Luego de tener ya la función por defecto que genera el bloque, se nombra la función y se le indica cuales van a ser los argumentos de la misma.

```
function dX= fcn(theta,psi,thetap,psip,U1,U2)
```

Después se ingresan todo lo correspondiente a los parámetros que van a regir el sistema, tales como parámetros físicos, y constantes de voltajes

```
%Parámetros Físicos del Helicóptero:
    Mheli=1.3872; % [Kg] masa del cuerpo movil del Quanser
    Lmc=0.0122; % [m] Posicion en x del centroide del helicoptero
    h=0.00714; %[m] posicion en z del centroide
    g=9.81; % [m/s2] gravedad
    JeqP=0.0332;%[kg*m^2] momento de inercia con respecto al eje de
pitch
    Jshaft=0.0039; % [Kg*m^2] momento de inercia del eje de yaw con
respecto a yaw
    JeqY=JeqP+Jshaft;%[kg*m^2] momento de inercia con respecto al eje de
yaw
    Bp=0.01325; %[N*m*s/rad] Coeficiente de friccion en el eje de pitch
    By=0.8513;%[Kg*m*s/rad] Coeficiente de friccion en el eje de Yaw

    %Constantes de Voltaje a Torque
    Kpp=0.02638;%Nm/V
    Kpy=0.001894;%Nm/V
    Kyp=0.002096;%Nm/V
    Kyy=0.01871;%Nm/V


    %Frcciones de coulomb:
    Fcpp=-0.2080; %constante %Nm/V
    Fcpy=0.0064;  %constante %Nm/V
    Fcyp=-0.0072;  %constante %Nm/V
    Fcyy=0.03004*2.65; %constante %Nm/V


    %Torques en función de voltajes:
    %las constantes a,b,c y d aparecen debido a la fricción de coulomb
    Tpp=Kpp*U1+Fcpp;
    Tpy=Kpy*U2+Fcpy;
    Typ=(Kyp*U1+Fcyp)*cos(theta);
    Tyy=(Kyy*U2+Fcyy)*cos(theta);
```

En este punto a diferencia del modelo del péndulo, se ingresó el sistema de ecuaciones en un único bloque, quedando de la siguiente manera.

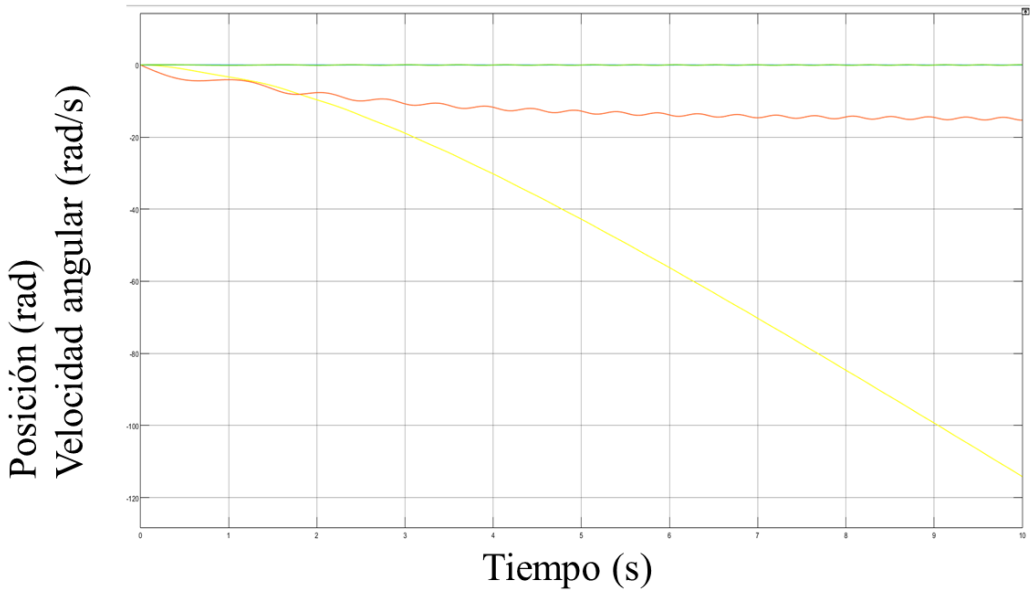
```
    dX1 = thetap;
    dX2 = psip;
    dX3 = (-Mheli*psip^2*(sin(2*theta)*(Lmc^2-h^2)/2-
Lmc*h*cos(2*theta))-Mheli*g*(Lmc*cos(theta)+h*sin(theta))-
Bp*thetap+Tpp+Tpy)/(JeqP+Mheli*(Lmc^2+h^2));
    dX4 = (-Mheli*(sin(2*theta)*(h^2-
Lmc^2)+2*Lmc*h*cos(2*theta))*thetap*psip-
By*psip+Typ+Tyy)/(JeqY+Mheli*(cos(theta)^2*(Lmc^2-
h^2)+Lmc*h*sin(2*theta)+h^2));
```

Por último, lo que retorna en la salida es justamente la operación de evaluar esas ecuaciones con los parámetros.

```
dX=[dX1; dX2; dX3; dX4];
```

Finalmente se determinó que se obtuvo un comportamiento homogéneo en las simulaciones de los tres diferentes softwares (Python, Matlab, Simulink). Concluyendo de ese modo que la implementación del algoritmo RK4O para el Helicóptero de dos grados de libertad quedo correctamente implementado dentro del hardware de bajo costo. En la Figura 3.14 se evidencian los resultados en Simulink.

Figura 3.13: Gráfica del comportamiento del Helicoptero de dos grados de libertad en Simulnk.



Nota: Línea amarilla: posición en Pitch. Línea roja: velocidad en Pitch. Línea azul: posición en Yaw. Línea verde: velocidad en Yaw.

En referencia al sentido físico de las gráficas, se debe tener presente que, en el Helicóptero de dos grados de libertad si bien el punto de equilibrio en Pitch es con el brazo completamente horizontal, debido a la condición inicial se descuelga hasta el punto de saturación que se verá, que es exactamente lo que se ve en la Figura 3-15 en la gráfica de color amarilla, mientras que en la gráfica que corresponde a la posición de Yaw no presenta ningún cambio.

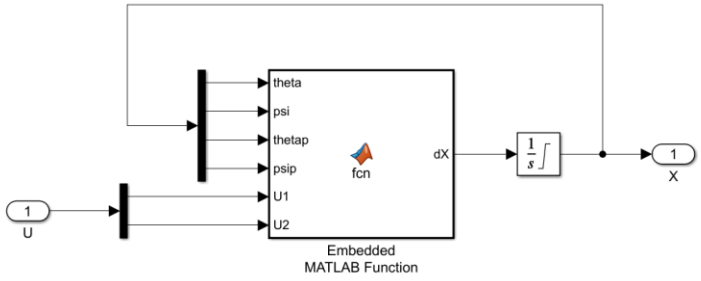
Después de comprobado el sistema en lazo abierto sin ninguna restricción, se agrega un bloque de “Saturation” en modelo de Simulink. Esto debido a que en la vida real El

funcionamiento del Helicóptero de dos grados de libertad no tendrá la libertad de movimiento que presenta en la simulación.

Para cada bloque de “Saturation” se utiliza un valor de saturación diferente, ya que las restricciones para el eje “Pitch” y el eje “Yaw” cambian. Para el caso del eje Yaw la restricción que se ajusto fue de -360 a 360 grados y para el caso del eje Pitch la restricción que se ajusto fue de -40.5 a 40.5 grados.

Una particularidad en este caso, es que no se utilizó un bloque independiente para la saturación, sino que, desde mismo bloque de integración, se pudo ajustar la saturación. En la Figura 3.15 se podrá observar el esquema del montaje.

Figura 3.14: Modelado del Helicóptero con bloque de Integrados con Saturador.



En este punto se valida si dichos bloques de saturación cumplen su cometido. Para eso se puede ver en la Figura 3.16 y la Figura 3.17 El resultado de la validación de la prueba. Partiendo de que el eje de Pitch asume un valor de 0 en la posición cuando se encuentra completamente horizontal.

Para la Figura 3.16 se aplicó un step de 0 por lo que para el eje que corresponde al ángulo “Pitch” no hay un cambio en la posición en referencia a la condición inicial en la que comienza, conociendo que esta es en -40.5.

Para la Figura 3.17 se aplicó un step de 16, acá los dos ángulos llegan al punto de saturación de acuerdo a los valores de los límites que se ajustaron.

Figura 3.15: Gráfica del comportamiento del Helicóptero de dos grados de libertad con Bloque de Saturación y Step de 0.

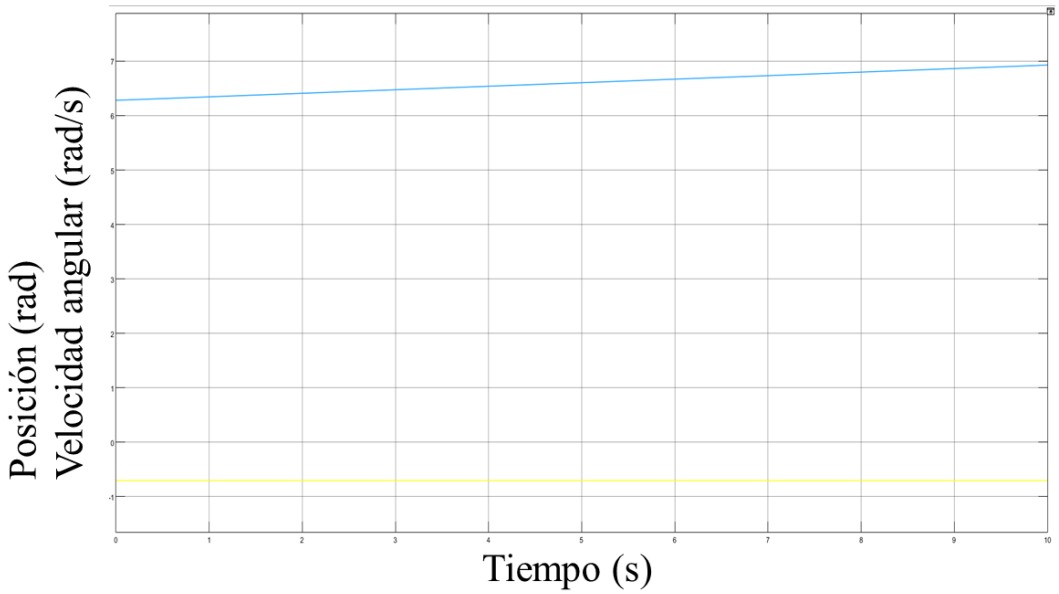
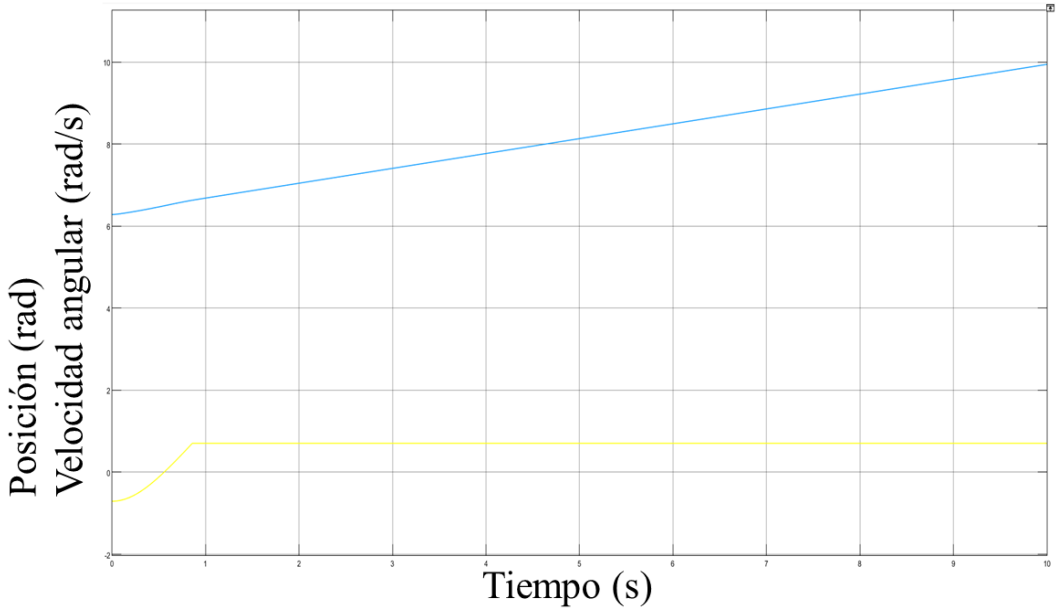
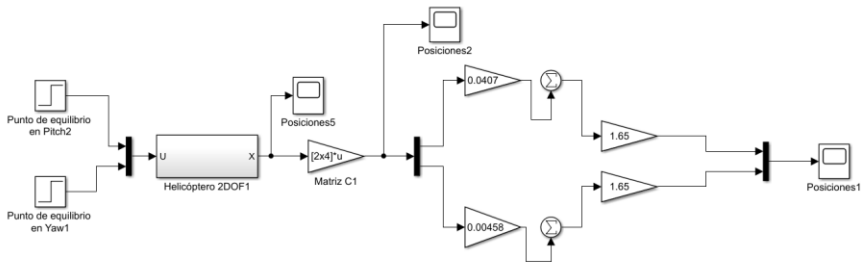


Figura 3.16: Gráfica del comportamiento del Helicóptero de dos grados de libertad con Bloque de Saturación y Step de 16.



Para terminar el proceso de verificación de los resultados, se debió hacer un ajuste en la salida, para que la interpretación del modelo de la Figura 3.17, representa una salida en radianes, por lo que se realizó una conversión de esos valores en radianes a voltaje, implementando la ecuación que se verá en el modelo de la Figura 3.18. El procedimiento de cómo se calcularon estos valores se puede visualizar con más detalle en la sección 4.2.5.1.2.

Figura 3.17: Modelo del Helicóptero de dos grados de libertad en Voltaje.



Ya en este punto se puede ver el comportamiento del Helicóptero de dos grados de libertad en función del voltaje con un step de 0 y un step de 16 en las Figura 3.10 y Figura 3.11 correspondientemente.

Figura 3.18: Gráfica Final del Comportamiento del Helióptero de dos grados de libertad en Voltaje - Step de 0.

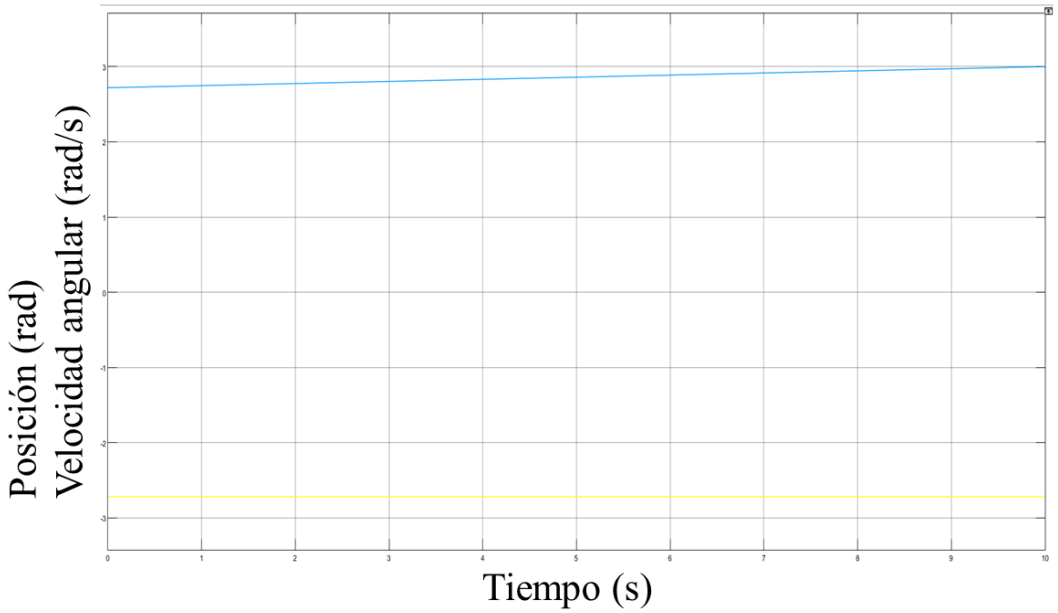
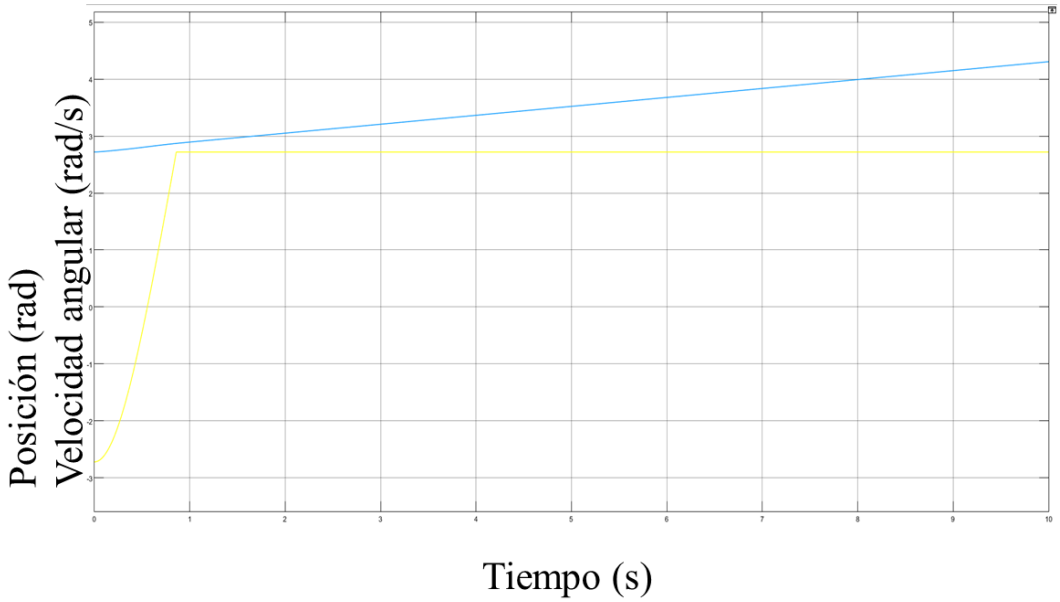


Figura 3.19: Gráfica Final del Comportamiento del Helióptero de dos grados de libertad en Voltaje - Step de 16.



4. Capítulo 4: Human-machine Interface.

4.1 Diseño e implementación del Front-end: Interfaz de usuario e ingreso de parámetros

Otro apartado importante es el que se verá en el capítulo 4, en el que se hablara sobre la human-machine interface (HMI) que se desarrolló para la interacción de los usuarios con el algoritmo. Pensada para que los usuarios tengan una experiencia agradable, la interfaz gráfica se basó en las normas ISA (101), que son normas para el diseño HMI.

Si bien a nivel internacional no había existido una estandarización acerca del diseño de las interfaces HMI, no es hasta el año 2005 que el comité ISA-SP 101 estables estándares, prácticas y recomendaciones que permiten normalizar el campo de las interfaces hombre-maquina. (Penin, n.d.)

Con el objetivo de:

- Disminuir errores en los diseños
- Reducir tiempos de aprendizaje.
- Reducir Costos en los diseños.

Es así que, bajo estas normativas, se trabajó la interfaz HMI del algoritmo para los dos sistemas de control expuestos y explicados previamente. normas tales como, los colores de fondo como los colores de los objetos, el tipo de fuente y la presentación visual de la información.

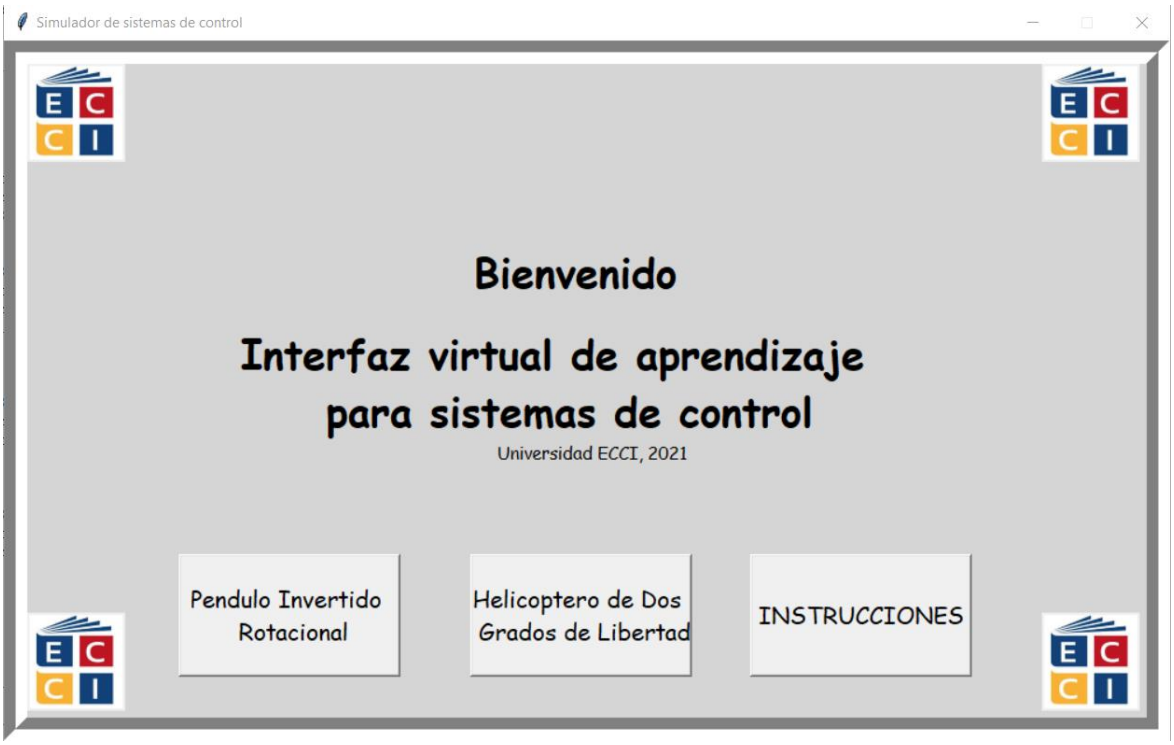
En primera instancia esta lo que es el color del fondo de la pantalla, según los criterios de las normas ISA 101 es recomendable que cada una de las pestañas o pantallas tengan el mismo color de fondo, así como un sinóptico de la planta en general, de igual forma es importante la ubicación de los elementos, tales como la información y presentarla de la

manera más sencilla que se pueda, y de forma tal, que a la hora de la lectura se haga un barrido de los datos.

La siguiente característica que se tuvo en cuenta fue el tipo de fuente. Para este punto de la presentación de la información dentro de la interfaz del proyecto, se tuvo especial cuidado en presentar solo los datos necesariamente a ingresar, ya que, si bien cada sistema cuenta con mucha información que se consideraría importante, no se pretende saturar la interfaz con todos eso datos. Así como tampoco, se utilizaron tipos de letras especiales, que generen conflicto en otros dispositivos ni colores llamativos.

Teniendo claro esto, ahora se procede a realizar la descripción de cada una de las ventanas que componen la interfaz HMI del proyecto. La primera ventana corresponde a la ventana de “Bienvenida”, ahí se podrá ver El título del proyecto con tres botones, dirigidos cada uno al correspondiente sistema electromecánicos y el último a una pestaña de instrucciones, como se muestra en la Figura **Figura 4.1**.

Figura 4.1: Pantalla Principal de la Interfaz HMI.



Las Figuras **Figura 4.2** y **Figura 4.3** son las ventanas que corresponden al simulador del Péndulo Invertido Rotacional y al Helicóptero de Dos Grados de Libertad respectivamente.

Acá se solicita que el usuario que asigne de manera manual los datos que pertenece a cada uno de los parámetros que rigen comportamiento de los sistemas.

Figura 4.2: Interfaz del Simulador del Péndulo Invertido Rotacional.

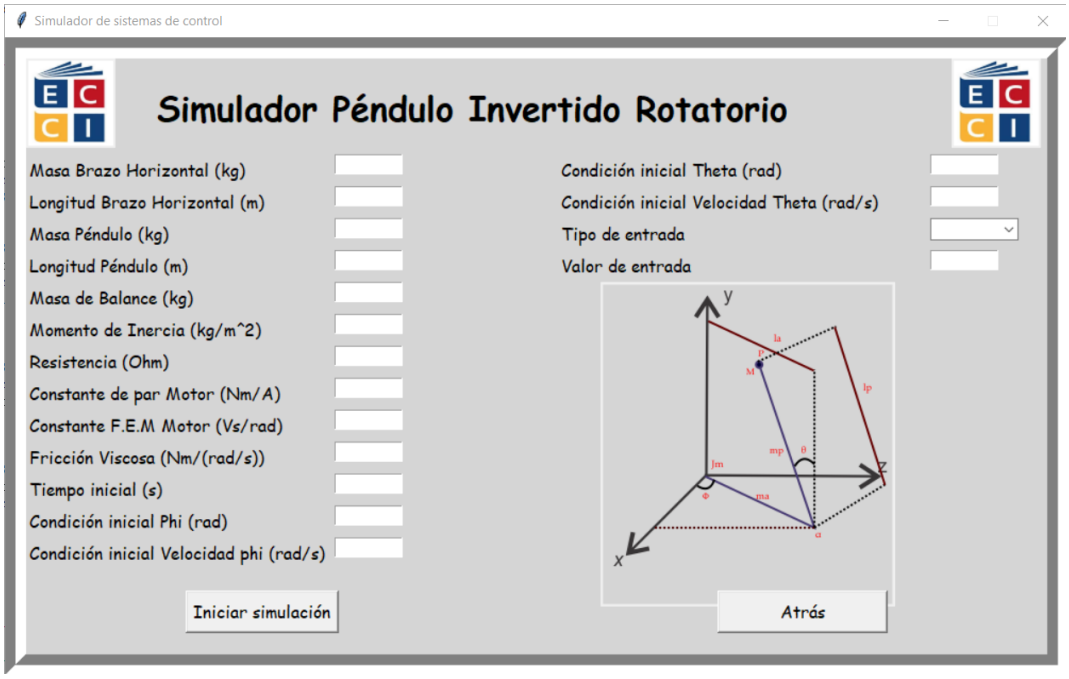
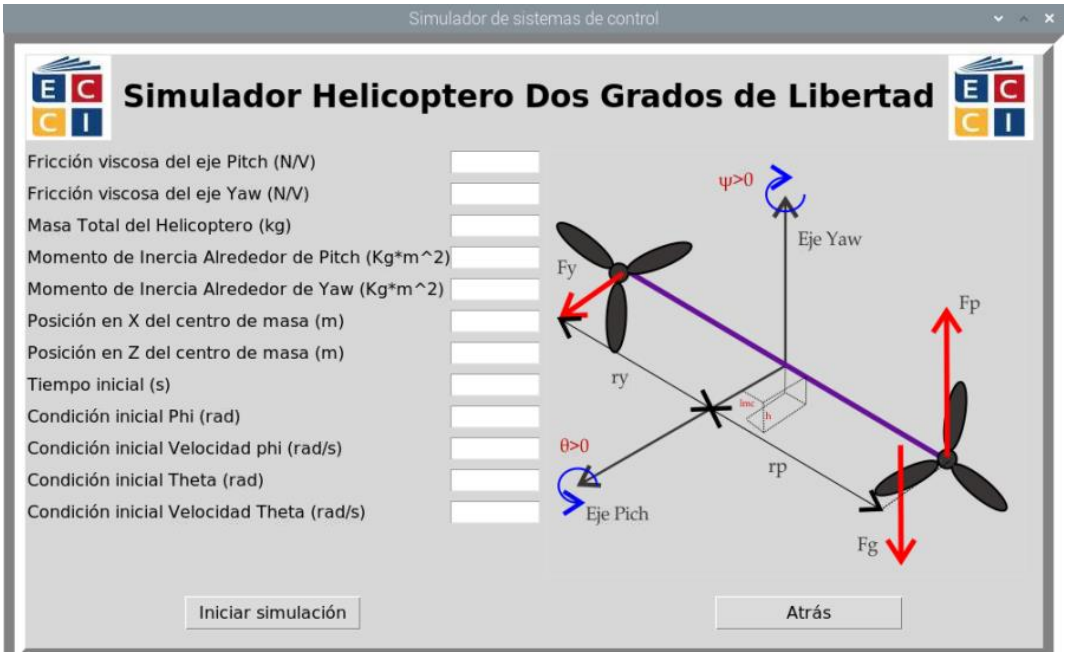
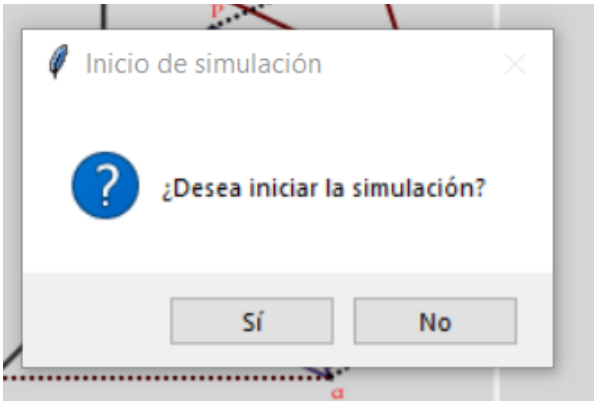


Figura 4.3: Interfaz del Simulador del Helicóptero de Dos Grados de Libertad.



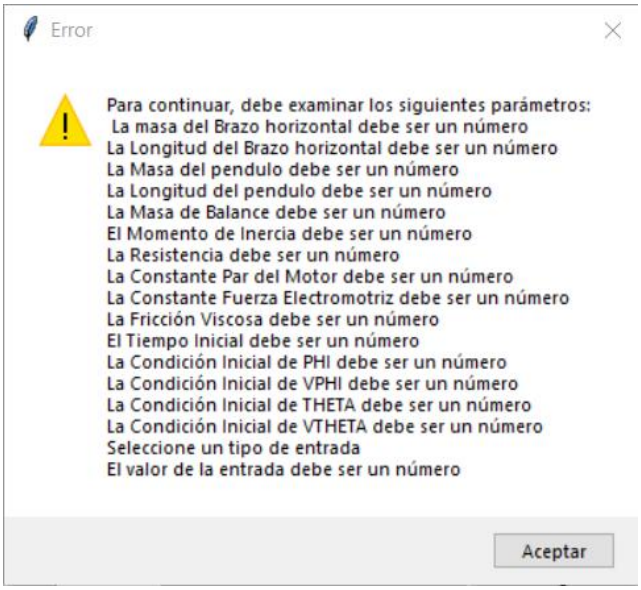
Como el marco de la secuencia de la información es de vital importancia a la hora de haber realizado el diseño de la Interfaz, la interfaz HMI del Hardware in the Loop, sigue una secuencia que determina el orden de los sucesos dentro del proceso e incluso subprocesos como lo es el de la Figura **Figura 4.4**, que indica el inicio de la ejecución del programa después de haber validado los datos correctamente.

Figura 4.4: Ventana de Confirmación para Ejecutar el Programa.



Otra norma que se siguió es la “Seguridad” Que pertenece a una “alarma” o ventana de advertencia en dado caso de que no se esté siguiendo el proceso adecuadamente. En este caso, si el usuario ingresa un parámetro incorrectamente, el proceso es detenido temporalmente hasta que sea corregido. Esta alarma se presenta en forma de una ventana emergente, como la mostrada en la Figura **Figura 4.5**.

Figura 4.5: Ventana de Advertencia.



4.2 Desarrollo del Back-End de la Interfaz virtual

4.2.1 Estructura general de la Interfaz virtual

En la subsección 4.1, se explicó el funcionamiento de la interfaz virtual de aprendizaje desde la visión del “usuario”, es decir, las pantallas que este observa, el ingreso de los datos, así como las advertencias e instructivo que permiten que la navegación sea cómoda e intuitiva. Como complemento a este apartado, se explicará la estructura desde el lado del “desarrollador”, para tener una visión más detallada de esta interfaz.

La interfaz virtual, al igual que el algoritmo RK4O, fue implementada en su totalidad bajo Python, haciendo uso de la librería Tkinter, especializada en la creación de interfaces gráficas. Al ser una interfaz con muchas funcionalidades (pantalla principal con acceso a los mecanismos y las instrucciones, ingreso de datos, salida por pantalla y por puerto I2C, entre otras), produce que el desarrollo de esta aumente en código a medida que se añaden estos aspectos, por lo que su construcción se ejecuta similar a una “página web” en el sentido de que cada una de las pantallas se manejan como archivos separados, garantizando así un diseño modular. Al trabajar de este modo, no solo se asegura que sea más sencillo detectar errores y posteriormente hacer su corrección sin afectar las demás

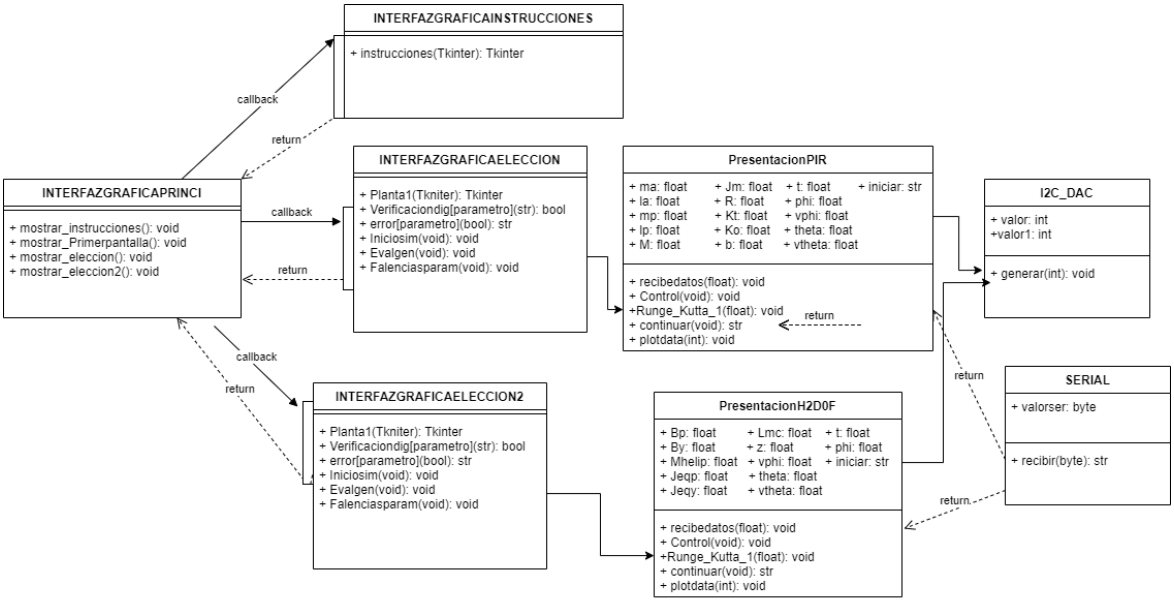
funcionalidades, sino también permitir la constante actualización o incorporación de nuevos aspectos en un tiempo menor.

Para que las pantallas puedan ser visibles al usuario, hay que entender el concepto principal de la creación de una pantalla en Tkinter: todas las ventanas con las que pueda contar una interfaz, son hijas de una pantalla madre, es decir, se crea una pantalla raíz que alberga toda las demás y esta pantalla principal, es la que siempre va a permanecer desplegada. En el caso de esta interfaz, la pantalla “raíz” sería la evidenciada en la Figura 4-1. Esta pantalla cuenta con tres botones cuya función es invocar a las pantallas restantes a través de una clase llamada Functools, que cuenta con un método “partial” que permite desplegar todas las propiedades de la pantalla si se presenta una llamada o “callback” de la pantalla madre, llamada que se efectúa una vez se hace clic sobre los botones.

Gracias a esta metodología de despliegue, se garantiza no solamente que la navegación sea más cómoda y accesible al contar con todas las opciones en una sola pantalla, sino que también posibilita el retorno entre pantallas en dado caso de que el usuario desee hacerlo, facultad que no sería posible de otro modo, ya que si se trataran por separado, eso obligaría a las pantallas a eliminarse para desplegar la siguiente y cuando esto sucede, no se puede volver a invocar de nuevo, a menos que se reinicie el programa. La constitución detallada de cada una de las pantallas de la interfaz virtual se explicará en las siguientes subsecciones.

Para ilustrar mejor su desarrollo, en la **Figura 4.6** se muestra el diagrama UML que demuestra la relación entre las diferentes clases que hacen posible el funcionamiento del programa.

Figura 4.6. Diagrama UML Hardware In The Loop diseñado.



4.2.2 Pantalla de bienvenida

Como se explicó en la subsección anterior, la pantalla de la Figura 4-1 es la pantalla de bienvenida. En ella, se hacen visibles tres botones que proyectan las pantallas de las dos plantas electromecánicas y una pantalla adicional de instrucciones. Esta pantalla, al ser la primera que evidencia el usuario, es la pantalla “raíz”, es decir, la que desplegará las demás cuando sean invocadas.

El código fuente de la pantalla de bienvenida se constituye principalmente de la creación de un objeto de la clase Tkinter que actuará como la pantalla raíz junto con sus parámetros (tamaño, color, botones, etc.). Al ser creada la pantalla principal, se crean otras tres funciones adicionales que permiten visibilizar la pantalla que se requiera al presionar su correspondiente botón. Cada función trabaja bajo la misma estructura: llama a todas las pantallas de la interfaz, las empaqueta a través del método “pack” (debe recordarse que todas las pantallas son creadas como objetos de la clase Tkinter) y para que solo la pantalla sea deseada esté visible, las pantallas restantes no solo se empaquetan, sino que también se ocultan de manera temporal a través del método “pack_forget”. Dentro de este archivo, se invocan los demás archivos que contienen el código de cada una de las pantallas a modo de “clases” a través del comando “import” y esto se evidencia al inicio. Finalmente, para ejecutar todas las propiedades de las pantallas, se hace el llamado a la clase de la

pantalla que se desea desplegar y se le suministran dos parámetros: la pantalla donde se desea que se despliegue (siempre va a ser la pantalla raíz) y la pantalla que invoca a la pantalla deseada (que también es la pantalla principal).

4.2.3 Pantalla de Instrucciones

La pantalla de instrucciones contiene un sencillo instructivo que le permite al usuario tener una noción de como navegar a través de la interfaz. Explica de manera sencilla las pantallas que encontrará, cómo debe ingresar los datos, iniciar la simulación, qué sucede si no ingresa correctamente un dato, entre otros aspectos que el usuario debe considerar si desea utilizar el programa.

Al igual que la pantalla principal, esta pantalla de instrucciones es creada como un objeto de la clase Tkinter junto con todas sus propiedades. Al ser una pantalla “hija”, todo el diseño está contenido en una función, que a su vez, actúa como método cuando se desea desplegar en la pantalla principal. Al inicio de esta función, se pregunta si la pantalla fue llamada a través de un “callback”. Si es así, las propiedades de la pantalla son traspasadas a la principal a través del método “partial” para que sea visible. Adicionalmente, cuenta con un botón de retroceso que una vez se presiona, invoca a través del “callback”, la pantalla que la llamó, es decir, la pantalla principal, garantizando que sea posible siempre regresar a esta. Finalmente, esta función debe retornar la pantalla creada en este archivo para que sea visible cuando sea llamada desde el archivo principal.

4.2.4 Pantallas de las plantas electromecánicas: Péndulo de Furuta y Helicóptero de dos grados de libertad

Para el caso de las dos pantallas de Ias plantas electromecánicas simuladas (ver Figuras 4-2 y 4-3), se cuenta con una estructura similar, por lo que se ahondarán de manera general. Después de ser creadas como objetos de la clase Tkinter, contener todas las propiedades en una función y que estas sean traspasadas a través de la llamada “callback” de la pantalla principal, lo que se puede visualizar cuando se hacen visibles son las cajas de texto que reciben los datos numéricos por parte del usuario, la imagen de referencia correspondiente a la planta y dos botones (retroceso e inicio de simulación).

Una vez el usuario ingresa los datos, este lo hace en las cajas de texto, que almacenan lo ingresado en una variable. El tipo de entrada puede ingresarse en un menú desplegable que también captura cuál se eligió en otra variable para posteriormente ser traspasados y poder efectuar la simulación. Como se desea que la interfaz sea lo más intuitiva y comprensible para el usuario, se incorporan unas funciones adicionales que permiten al programa detectar si los datos se ingresaron correctamente o si se ingresaron. En caso de que esto no suceda, arroja unas alertas que le indican al usuario qué parámetros no fueron ingresados o no son correctos, y no le permitirá continuar con la simulación.

Ahora bien, para que estas funciones puedan ejecutarse, se incorporan al programa una serie de funciones que evalúan el valor ingresado y verifican si este es un número o si la caja de texto tiene una cadena vacía. Si esto sucede, la variable global que acumula los errores concatena la falencia de cada parámetro para que al final, despliegue en un cuadro de advertencias lo que debe corregir o lo que debe ingresar para poder iniciar la simulación (ver Figura 4-5). Cuando el usuario haya ingresado correctamente los datos, la variable global de errores permanecerá vacía y es allí cuando se despliega un cuadro de diálogo que le permite al usuario iniciar la simulación (ver Figura 4-4). Si el usuario acepta, los valores numéricos ingresados son convertidos a coma flotante (el valor almacenado en la caja de texto se conserva como tipo cadena) y se transfieren a un nuevo archivo “clase” que ejecutará la simulación. Finalmente, la función retorna la pantalla que se desea visualizar y la despliega en la pantalla raíz. Las imágenes desplegadas se almacenan en una lista que permite que sean visibles sin que se conviertan en “datos basura”, lo que impedirían que se hagan visibles al ser ventanas “hijas” de la ventana principal.

4.2.5 Simulación de las plantas electromecánicas

Como se mencionó en la sección 4.2.4, una vez los parámetros de las dos plantas son ingresados a través del método “recibe_datos”, el cual se encuentra presente en las clases que ejecutan la simulación de los datos, estos son recibidos y es allí donde se procesan los datos para luego ser simulados y arrojar las repuestas en el tiempo esperadas. Es importante acotar que los códigos de este menester se encuentran en el anexo A, y se procederá en esta sección a explica con detalle su funcionamiento y el diagrama UML del software se puede encontrar en la Figura…

Antes de empezar el análisis, es importante acotar que el lector encontrará dentro de la estructura de programación que se crearon dos clases para la misma función, una para cada planta. La razón del por qué se hizo esto radica en que, a pesar de que la funcionalidad es la misma, cada planta cuenta con sus particularidades específicas en lo concerniente a la cantidad de entradas (una para el Péndulo de Furuta y dos para el Helicóptero), además de los pasos de ejecución del método y los tiempos internos de recepción de datos y cómo estos son recibidos por el software al momento de cerrar el lazo de control. Las diferencias se explorarán a través de las siguientes subsecciones.

4.2.5.1 Aspectos generales de las clases tipo "Presentación"

Las clases tipo "Presentación" (PresentacionPIR y PresentacionH2DOF respectivamente) son las encargadas de procesar los datos y transformarlos en unas gráficas que muestran el comportamiento en el tiempo de las posiciones y velocidades de las plantas, enviar los datos a una clase posterior para transformarlos en señales análogas, además de recibir las señales de control que permiten cerrar el lazo y permitir que las plantas puedan seguir las referencias deseadas por el usuario.

Es importante acotar que cada una de estas funciones, debido a que es un Hardware In The Loop, deben ejecutarse al mismo tiempo y de manera coordinada, por lo cual se recurre a una herramienta muy útil de programación presente en todos los programas de uso cotidiano: los hilos, los cuales pueden definirse como múltiples tareas que se ejecutan al mismo tiempo a través de una ejecución- bloqueo generada por el procesador por espacios de tiempo cortos, produciendo esta precepción ante el usuario. Para lograr esto, se recurre a la clase Thread la cual permite que las funciones que se desean ejecutar dentro de la clase lo puedan lograr. El funcionamiento de los hilos en Python trabaja bajo la premisa de que siempre debe de escogerse un proceso principal, y los demás serán procesos secundarios que se invocarán mediante el método Thread. A continuación, se relacionan proceso principal y secundarios en la **Tabla 4-1**-1.

Tabla 4-1. Relación de procesos y qué tipo de tarea es al interior de la clase.

Tarea	Nombre tarea en el código	Tipo de tarea
Graficación y animación de la simulación	plotdata()	Tarea principal
Ejecución del método numérico y llamado a clase que crea las señales análogas	Runge_Kutta_1() o Runge_Kutta_2() según el caso	Tarea secundaria
Recepción de la acción de control por protocolo serial	Control()	Tarea secundaria

Siguiendo la premisa anteriormente mencionada, la tarea principal es ejecutada como proceso principal, no siendo invocado como objeto de la clase Thread, las razones de esta decisión, se considerarán de manera posterior. Mientras que los procesos son secundarios como objetos dentro de la clase Thread siendo el "target" el nombre de la tarea dentro del código, además de sus parámetros. Una vez son creados, se inician los hilos con el método start() y una vez no se desea continuar con el proceso, el método join() detiene y destruye los hilos para liberar el espacio de memoria que ocupan.

```
thread=Thread(target=Runge_Kutta_1, args=(ma, la, mp, lp, M, Jm, R, Kt,
Ko, b, t, phi, vphi, theta, vtheta, iniciar))
thread1=Thread(target=Control)
thread.start()
thread1.start()
ani=animation.FuncAnimation(fig,plotdata, interval=200, repeat=False)
plt.show()
thread.join()
thread1.join()
```

Como se puede observar, la tarea principal no es invocada como hilo debido a que su estructura siempre se correrá en un loop indefinido y por ello, es más conveniente para el desarrollador que sea la tarea principal, mientras que las demás si son invocadas como hilos.

4.2.5.1.1 Método plotdata( )

Uno de los requerimientos principales dentro de la interfaz HMI es que para mejorar la experiencia del usuario este pueda evidenciar el comportamiento de las plantas y sus

cambios en el tiempo de manera gráfica en pantalla. Para cumplir con este requerimiento, se hace un desarrollo de tal forma que el resultado de los cálculos se vea plasmados en cuatro pantallas (posiciones y velocidades) que animan en el tiempo más real posible estos comportamientos, permitiendo al usuario observar de manera más gráfica estos temas.

Ahora bien, para ello, se hace uso de la clase *matplotlib.animation* y se creará un objeto de esa clase para que se pueda ejecutar de manera posterior.

```
ani=animation.FuncAnimation(fig,plotdata, interval=200, repeat=False)
```

Un objeto de la clase matplotlib.animation recibe como atributos una figura matplotlib que se crea previamente (tamaño de la ventana, cuántas gráficas se desean, sus rótulos, etc.), la función que toma la trama de datos y hace posible la animación (esta trama se actualiza cada cierto tiempo y se regula a través de una variable *n* que se itera de manera indefinida, el intervalo en milisegundos en el que se actualizan los frames generados y si se desea que la gráfica se repita, esto para truncar errores de uso relacionados con volver a usar la interfaz para simular nuevamente.

La tarea que se encarga de tomar las muestras y enviarlas a la animación se llama plotdata( ), la cual inicia tomando los datos resultado del método Runge- Kutta y los almacena en otras cinco listas que se usarán para animar el comportamiento en el tiempo y los recibe de la siguiente forma:

- Lista *tempo*: recibe el intervalo de tiempo en el que viene siendo graficado el comportamiento en el tiempo.
- Lista *data1:* recibe la lista resultado1 (lista que almacena la variable φ)
- Lista *data2:* recibe la lista resultado2 (lista que almacena la variable velocidad de φ)
- Lista *data3:* recibe la lista resultado3 (lista que almacena la variable θ)
- Lista *data4:* recibe la lista resultado4 (lista que almacena la variable velocidad de θ)

Como se había mencionado en capítulos anteriores, el método de Runge- Kutta en cualquiera de sus variantes no es infinito, por lo cual se pregunta constantemente si desea continuar, y cuando el usuario no desea continuar, los datos dejan de ser

generados y como esta clase ejecuta las animaciones de manera indefinida, genera un error de ejecución relacionado con no recibir más datos y por ello, no poder animar más. Este error se subsana examinando constantemente si la longitud de la lista recibida de tiempo (variable equiparable con la variable iterativa *n)* es igual o mayor a *n*, lo cual para el programa significa que no se recibirán más datos y ya no se simularán más y para evitar este error si esto sucede, las listas que reciben los datos a animar recibirán valores de 0 para seguir percibiendo animación aun cuando el método ha finalizado.

```
def plotdata(n):
    global auxiliartiempo
    global flag
    orr=len(lineatiempo)
    aux2=n+1
    #print(orr)
    #print(aux2)
    #print(flag)
    if(aux2>=orr):
        auxiliartiempo=auxiliartiempo+0.1
        tempo.append(auxiliartiempo)
        data1.append(0)
        data2.append(0)
        data3.append(0)
        data4.append(0)
        flag=False
    else:
        flag=True
        auxiliartiempo=lineatiempo[n]
        tempo.append(lineatiempo[n])
        data1.append(resultado1[n])
        data2.append(resultado2[n])
        data3.append(resultado3[n])
        data4.append(resultado4[n])
```

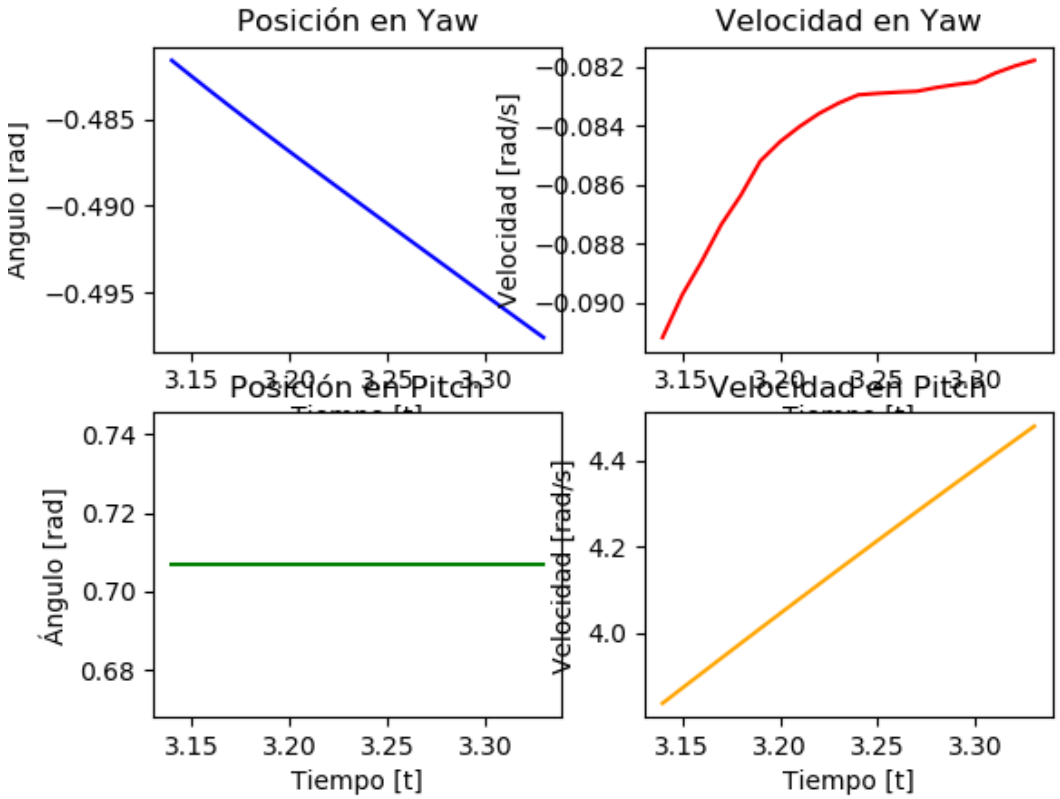
Posteriormente, se envían las tramas para ser graficadas a través de “ventanas de datos”, las cuales no son más que la toma de cierta cantidad de muestras para que la percepción de animación sea más real. Estas “ventanas” para todos los datos son regidas a través de la variable global *Samples* inicializada con el valor de 20. Esto quiere decir que las ventanas enviarán cada 20 datos para ser graficados y una vez suceda esto, las pantallas que los proyectan serán limpiadas para que grafiquen las siguientes ventanas de datos a través de método clear( ).

```
tempod=tempo[-Samples:]
data1d=data1[-Samples:]
data2d=data2[-Samples:]
```

```
data3d=data3[-Samples:]
data4d=data4[-Samples:]
ax1.clear()
ax2.clear()
ax3.clear()
ax4.clear()
```

Finalmente, estos datos son animados de manera indefinida creando el objeto visto al inicio de esta subsección, teniendo como resultado lo visto en la **Figura 4.7**.

Figura 4.7. Muestra de pantalla animada del Hardware In The Loop. Ejemplo de la pantalla del Helicóptero de dos grados de libertad-



4.2.5.1.2 Método Runge_Kutta_1( ) y Método Runge_Kutta_2( )

Este método es el encargado de ejecutar y crear la simulación del comportamiento en el tiempo tanto de las posiciones como de las velocidades de las plantas electromecánicas simuladas a través del algoritmo de Runge- Kutta de cuarto orden, profundizado en el Capítulo 3, Sección 3.2.

Una vez los datos son procesados y calculados, se obtiene la respuesta del sistema simulado, tal como se evidencia en las Figuras 3-1 y **Figura 3.2** del Capítulo 3.

Ahora bien, una vez el método obtiene los resultados, deben considerarse las restricciones físicas con la que las plantas cuentan en sus particularidades, pues si los datos deben ser transformados en señales medibles, es necesario que estas restricciones se consideren dentro del modelo, además de ser lo más fieles posible al comportamiento de la planta real. Es por esto que, dependiendo de la planta en cuestión, estos intervalos de operación se realizan y al igual que lo considerado en la validación en Simulink en el Capítulo 3, Sección 3.3, se implementa un saturador en cada una de las salidas del sistema, y de acuerdo con estas restricciones, se le pide al software que mantenga el valor límite en el caso que supere estos intervalos. Las particularidades de cada planta se considerarán en la siguiente subsección y a continuación, la **Tabla 4-2** ilustra los límites físicos de cada rango de acuerdo con los trabajos escritos previos citados en capítulos anteriores:

Tabla 4-2. Límites físicos de las dos plantas simuladas

Planta	Límite físico en φ	Límite físico en θ
Péndulo de Furuta	-2π rad, 2π rad	-0.0872665 rad, 0.0872665 rad
Helicóptero de dos grados de libertad	-2π rad, 2π rad	-0.7068 rad, 0.7068 rad

Una vez obtenidos estos límites, se hace una linealización sobre estos rangos de operación, pues el conversor Digital- Análogo que se implementa para convertir los valores en señales medibles tiene un rango de conversión limitado de 0 a 4095, por lo cual se hace la consideración de que el valor inferior es igual a 0 y el valor superior sería igual a 4095, por lo cual, considerando estos valores como una ecuación lineal se toman los siguientes puntos como en la **Tabla 4-3**:

Tabla 4-3. Valores equivalentes entre el valor de posición y los rangos de resultado del conversor DAC.

Planta	Valores equivalentes (φ, VDAC)	Valores equivalentes (θ, VDAC)
Péndulo de Furuta	(-2π rad, 0), (2π rad, 4095)	(-0.0872665 rad, 0), (0.0872665 rad, 4095)
Helicóptero de dos grados de libertad	(-2π rad, 0), (2π rad, 4095)	(-0.7068 rad, 0), (0.7068 rad, 4095)

Al realizar el procedimiento para calcular la ecuación de la recta para describir este comportamiento (tomar los valores y calcular la pendiente, calcular el punto de corte cuando la posición es igual a 0), las ecuaciones resultantes son las proyectadas en la **Tabla 4-4**.

Tabla 4-4. Ecuaciones de conversión de los valores digitales a señales análogos.

Planta	Ecuación para φ	Ecuación para θ
Péndulo de Furuta	$Val_{DAC} = 325.87\varphi + 2047.49$	$Val_{DAC} = 2896.85\theta + 2047.49$
Helicóptero de dos grados de libertad	$Val_{DAC} = 325.87\varphi + 2047.49$	$Val_{DAC} = 23462.61\theta + 2047.49$

Finalmente, estos valores son convertidos a valores enteros para ser procesados por el DAC y dar como resultado los valores análogos en voltios mediante el llamado a la clase *I2C_DAC* mediante el método generar( ), no sin antes crear un objeto de esta clase. A continuación, se proyecta el ejemplo con el Péndulo de Furuta:

```
valordac1=int((23462.61*theta)+2047.49)
valordac2=int((325.87*(phi))+2047.49)

valordac1d=I2C_DAC(valordac1, valordac2)
valordac1d.generar()
```

Dentro de cada uno de los códigos, existe una particularidad en el eje φ, debido a que este tiene una restricción de ±2π rad, si se mantiene en este valor, físicamente no se estaría moviendo, por lo cual se le agregó un detalle adicional donde, al llegar este valor, con ayuda de una variable auxiliar retorne a 0 y pueda volver a rotar:

```
valordac2=int((325.87*(phi-auxiliar1))+2047.49)
```

Al final del método, se pregunta al usuario si desea continuar, tal como se había planteado en el Capítulo 3 donde se explica el funcionamiento del algoritmo. Si no desea hacerlo, el método se dejará de ejecutar y el hilo se cerrará.

4.2.5.1.3 Método Control( )

Este método es el encargado de recibir la acción de control desde el controlador externo (calculado por una tarjeta Arduino DUE) mediante protocolo serial, estos valores son recibidos por la clase *SERIAL* que retorna a su vez los resultados, son convertidos a valores de coma flotante para que sean almacenados en las variables globales *valor* (en el caso del péndulo) y *E1* y *E2* (en el caso del Helicóptero) y de este modo, se cierre el lazo de control.

```
lectura=SERIAL()
valor=float(lectura.recibir())
```

4.2.5.2 Consideraciones específicas para cada una de las plantas simuladas de la clase "Presentación"

En la subsección 4.2.5.1, se habló acerca de que, a pesar de que las funcionalidades son las mismas para ambas clases tipo "Presentacion", la razón de crear dos clases diferentes radica en la necesidad de subsanar posibles problemas o parámetros específicos de cada una de las plantas y asegurarse de este modo que cada módulo trabaje correctamente y que estos errores puedan ser corregidos sin tanto esfuerzo. Así mismo, en la sección 4.2.5.1.2 se comentó acerca de ciertas particularidades en la ejecución del método, tanto en valores de paso y número de entradas. Estos parámetros son explicados con mayor profundidad en la sección 3.2.1 y 3.2.2 del Capítulo 3.

Finalmente, una última consideración que se tiene en cuenta y que es particular en cada planta, es la acción de control que se genera y cómo esta recibe los datos y los distingue para poder cerrar el lazo. En el caso del Péndulo de Furuta, solo se haría la recepción de un solo valor, por lo que no sería necesario hacer una discriminación de los datos y estos serían convertidos directamente a un valor de coma flotante el cual ingresaría

al valor de la entrada, cerrando así el lazo. También se agregó una consideración adicional a los valores de la acción de control, pues cuando estos valores son muy altos, se arroja una cadena que notifica esto, por lo cual se sugiere al código que convierta este valor en 0 y que continúe con la acción de control.

```
def Control():
    global flag
    global valor
    while(flag):
        lectura=SERIAL()
        if('nan' in lectura.recibir()):
            valor=0.0
        else:
            valor=float(lectura.recibir())
```

En el caso del Helicóptero de dos grados de libertad, se reciben dos acciones de control, las cuales son transmitidas por puerto serial y al contar con una sola vía, es necesario discriminar los valores para que la clase sepa a cuál entrada pertenece cada acción de control, es por esto que desde el controlador externo se envían las acciones de control distinguidas por letras (A para Pitch y B para Yaw). Una vez estos valores son recibidos, el método pregunta qué letra contiene, si es A, suprime la letra, convierte este valor a coma flotante y lo asigna a *E1,* variable de entrada de Pitch y si es B, igualmente suprime la letra, convierte el valor a coma flotante y lo asigna a *E2,* valor de entrada de Yaw.

```
def Control():
        global flag
        global E1
        global E2
        while(flag):
            lectura=SERIAL()

            if('A' in lectura.recibir()):
                E1=float(lectura.recibir().replace("A",""))
            if('B' in lectura.recibir()):
                E2=float(lectura.recibir().replace("B",""))
```

4.2.6 Clase I2C_DAC

En la sección 4.2.5.1.2, se explicó que una vez que los valores son calculados y linealizados, se envían a un objeto creado de la clase I2_DAC, el cual se analizará con más detalle en esta subsección.

Inicialmente, se hace un llamado a los módulos que hacen posible la comunicación I2C *board* y *busio,* estableciendo las direcciones I2C que ocuparán cada uno de los DAC utilizados. Debido a que por defecto los DAC contaban con la misma dirección, la única alternativa al respecto es cambiarlas físicamente soldando un apartado especial del DAC que permiten cambiar esta dirección y por ello, se pueden tener las direcciones 0x60 y 0x61. Posteriormente, se hace uso de la clase *adafruit_MCP4725* para establecer comunicación con los dos DAC y se establece el valor máximo de lectura a través del método *raw_value( ).*

```
i2c = busio.I2C(board.SCL, board.SDA)
dac = adafruit_mcp4725.MCP4725(i2c, address=0x60)
dac1 = adafruit_mcp4725.MCP4725(i2c, address=0x61)
dac.raw_value = 4095
dac1.raw_value = 4095
```

Posteriormente, se hace el constructor que recibe los valores de las posiciones y con estos valores, a través del método *generar( ),* se tiene como resultado los valores análogos que son expresados en voltaje en el exterior.

```
class I2C_DAC:
    def __init__(self, value, value1):
        self.value=value
        self.value1=value1

    def generar(self):
        #print(self.value)
        #print(self.value1)
        dac.raw_value = self.value
        dac1.raw_value = self.value1
```

4.2.7 Clase SERIAL

Finalmente, cuando el controlador genera la acción de control, los valores son enviados por protocolo serial, el cual es recibido por la Raspberry a través de la clase *SERIAL,* la cual los recibe y los trata para luego ser procesados desde las clases *Presentación.*

Para ello, se invoca la clase serial de Python para poder llamar el puerto al que está conectado el controlador a una tasa de baudios de 115200, debido a que esta tasa está configurados los controladores de manera posterior y así se asegura la mayor rapidez de transmisión de los datos posible.

```
ser = serial.Serial("/dev/ttyACM0", baudrate=115200)
```

Cuando los datos son recibidos a través del puerto, llegan como datos tipo byte, por lo cual debe ser decodificados a un formato string tipo ascii para poder ser procesados y finalmente este valor convertido es el retornado en las clases *Presentacion*.

```
class SERIAL:
    def __init__(self):
        self.valorser=ser.readline()

    def recibir(self):
        rd=self.valorser.decode("ascii")
        return(rd)
```

5. Resultados de la validación del funcionamiento del Hardware In The Loop

5.1 Validación del Péndulo de Furuta con PID

Luego de comprobar que el funcionamiento del software con las simulaciones en Simulink de cada uno de los sistemas de control ya trabajados, se plantea hacer las ultimas verificaciones. Esto es cerrar el lazo con un controlador PID, revisando y analizando los resultados obtenidos y comprobando la funcionalidad o no del controlador frente a los sistemas no lineales.

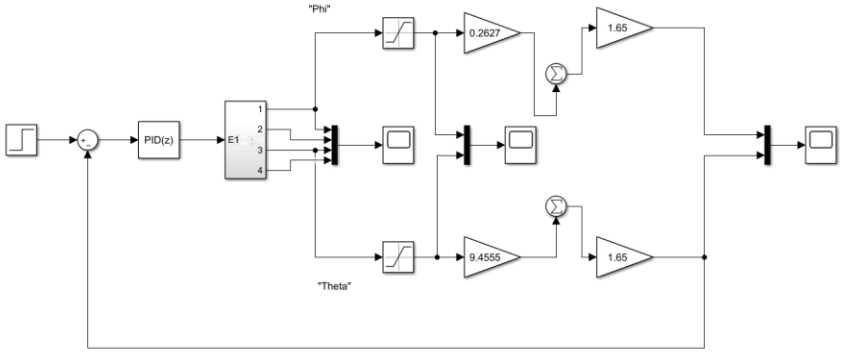
Continuando con el modelo ya implementado en Simulink del Péndulo de Furuta, lo que resta por hacer es incorporar un bloque PID para poder así cerrar el lazo y mirar la acción de control.

El bloque de PID controller permite implementar un controlador con diferentes tipos y estructuras de controlador, tales como configuraciones (PID, PI, PD), variar el dominio en el tiempo (continuo o discreto) y dentro de la segunda opción permite manipular el integrador y el método de discretización, permitiendo escoger entre (Euler hacía adelante, Euler hacía atrás y Trapezoidal)

Debido a la naturaleza de los datos en el Hardware in the Loop, dentro de las verificaciones de los sistemas en Simulink se optó por dominio en el tiempo discreto y el método de filtrado e integración una configuración Trapezoidal. Lista la configuración del bloque PID, la Figura 5.1 muestra el modelado del sistema en lazo cerrado.

Cabe resaltar que los valores de las ganancia proporcional, integradora y derivativa, fueron ajustadas por medio de la herramienta “Tune” que posee Simulink.

Figura 5.1:Modelo del Péndulo de Furuta en Lazo Cerrado



En este punto se puede ya realizar la simulación final, para poder analizar la viabilidad de implementar un controlador de tipo PID en este sistema. De la interpretación de los resultados se puede decir que debido a la gran inestabilidad y no linealidad del sistema el PID debe efectuar un esfuerzo de control bastante grande, causando que la simulación se detenga por errores de “singularidad”, es decir, que el programa toma valores numéricos bastantes altos que el software de Simulink no puede manejar.

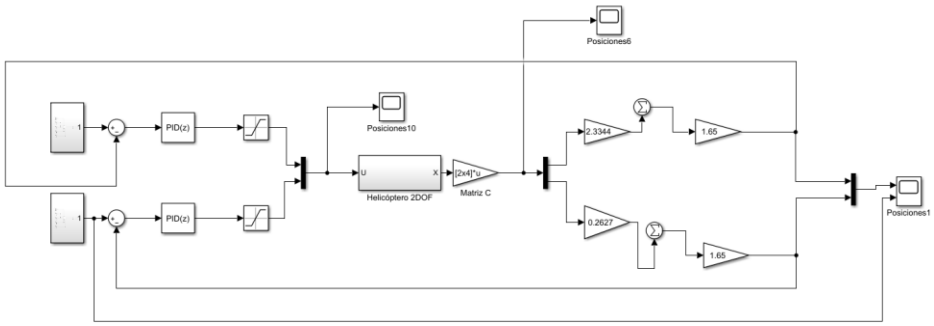
Aunque se realizaron varias pruebas con diferentes tipos de controladores, el error sobre la ejecución no permitía una simulación prologada en el tiempo. Lo cual llega a dictar en primer lugar la limitación del software frente a este tipo de cálculos matemáticos y en segundo lugar que, debido a alta inestabilidad del modelo, se debe considerar el implementar otro tipo de controlador.

5.2 Validación del Helicóptero de dos grados de libertad con PID

Para la validación de los resultados del Helicóptero de dos grados de libertad en lazo cerrado se debió tomar en cuenta las mismas consideraciones que en el péndulo, aunque son sistemas diferentes, en el sentido de la configuración del bloque PID Controller. Esto quiere decir, que para ambos sistemas se optó por un dominio en el tiempo discreto con un método de discretización e integrador Trapezoidal.

Acá una vez más se hace uso de la herramienta con que cuenta Simulink “Tune” para generar de manera automática los valores correspondientes del proporcional, integrador y derivador. Quedando el modelo completo en lazo cerrado como se ve en la Figura 5-2

Figura 5.2: Modelo del Helicóptero de dos grados de libertad en Lazo Cerrado



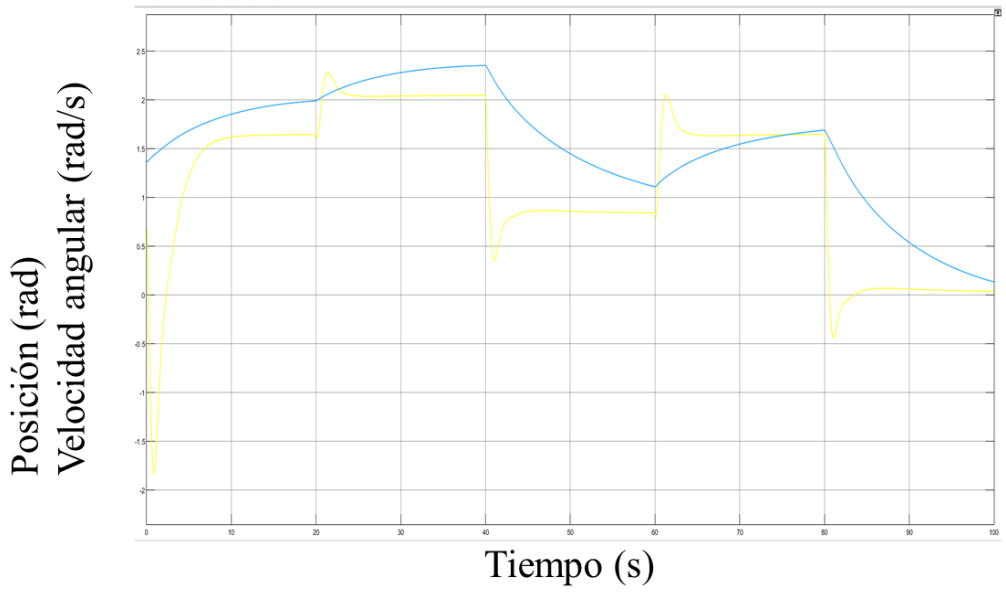
En el caso del helicóptero de dos grados de libertad se debió implementar dos controladores PID, uno para cada eslabón que controla el movimiento de Pitch y Yaw por aparte. Es por eso que en la Figura 5.2 se ve como se cierran dos lazos independientes. Por lo que aun que cada bloque de PID Controller tenga la misma configuración, los valores de ajuste de las ganancias son diferentes.

Adicionalmente, cuenta con un bloque de saturation por si los valores del resultado del control se tornan muy altos, sean los más fehacientes a los admitidos por las características de los motores que se encuentran dentro del modelado del helicóptero.

En este punto, se puede ya ejecutar la simulación del modelo del Helicóptero en lazo cerrado y poder analizar la viabilidad del controlador PID en este sistema. Para eso se puede observar la Figura 5-3 que corresponde a la gráfica de la respuesta del sistema con los dos controladores implementados.

Como se puede ver en la Figura 5-3, a diferencia del Péndulo de Furuta, el Helicóptero no presenta algún error de “singularidad” por problemas en los cálculos matemáticos de la simulación.

Figura 5.3: Gráfica Final del Comportamiento del Helióptero de dos grados de libertad con PID



Nota: Línea amarilla: posición en Pitch. Línea azul: posición en Yaw. Línea roja: Seguimiento de referencia.

Acá, para poder observar mejor la acción de control del PID y como este se comporta antes los cambios, para ello se ajustaron las entradas con un cambio de referencia, y así poder ver si tanto Pitch como Yaw con capaces de seguir dicha referencia.

Aunque el controlador implementado es funcional, como se puede ver y que Pitch responde mejor que Yaw a los cambios de referencia de la entrada, se puede considerar la ideas de implementar otro tipo de controlador para una mayor eficiencia.

5.3 Implementación del controlador PID en el HIL del Péndulo de Furuta

5.3.1 Diseño del controlador PID en Arduino Due

Para la implementación del controlador externo, se usa la tarjeta embebida Arduino Due, la cual se eligió debido a que Arduino desde el software cuenta con amplia documentación para el desarrollo del PID, además de que esta modalidad de Arduino cuenta con un potente procesador ARM CortexM3 de 32- Bit (Arduino, n.d.) a través de la

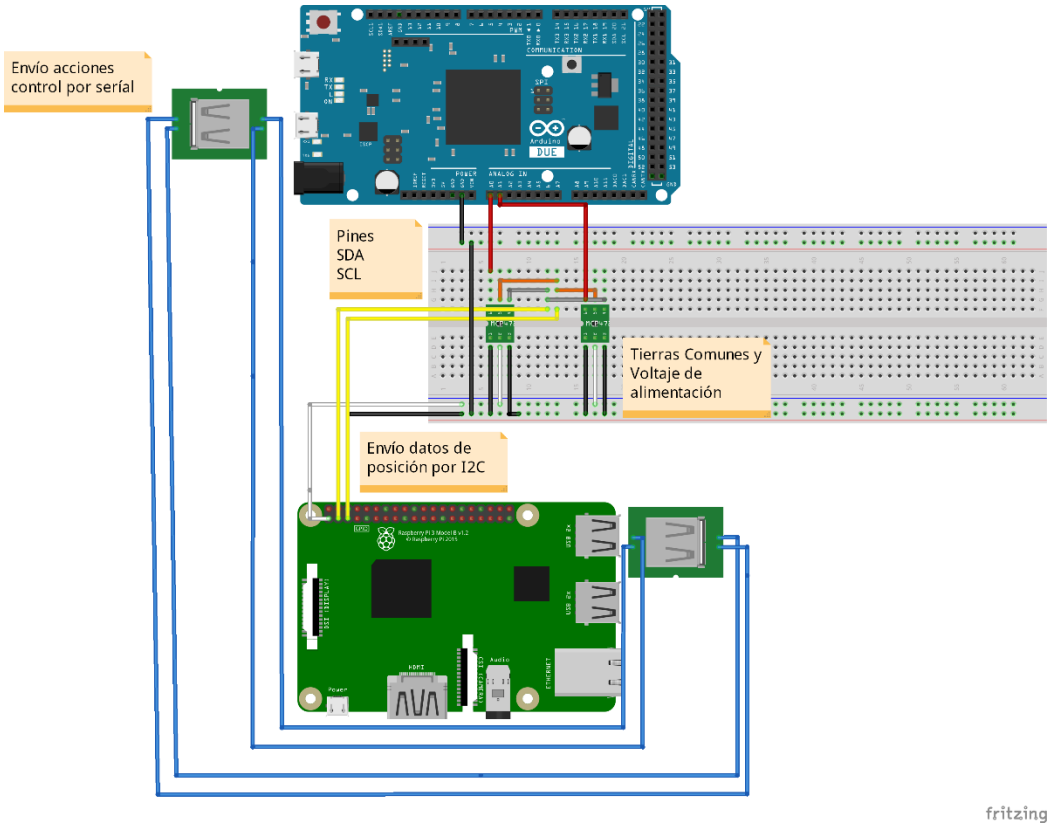
librería ArduPID, la cual cuenta con una configuración paralela con método de discretización Trapezoidal. En esta librería, se crea un objeto de esta clase, se fija el Set Point y las ganancias P, I y D de acuerdo con pruebas experimentales y los valres calculados por Simulink, así como la entrada producto de la lectura análoga de los valores de posición en voltaje, para que al final calcule la acción de salida y la almacene en la variable Output, la cual se usa para enviar los datos por puerto serial.

Posteriormente, en la sección *setup* se establecen los valores necesarios para poder parametrizar adecuadamente el PID: establecer el tiempo de muestro entre 1ms y 100ms a través del método *setSampleTime( ),* los límites máximos del voltaje de la acción de control, considerados por el autor Navaridas (Navaridas, 2017) para el Motor Lego XL (en el caso del Péndulo de Furuta) y para los motores Faulhaber Serie 2842 y Pittman Modelo 9234 (en el caso del Helicóptero de dos grados de Libertad) según los autores Porras y Guatibonza (Porras Rodríguez & Guatibonza Pérez, 2020) mediante el método *setOutputLimits( )* y finalmente unos valores Wind- Up que impide que se acumule acción de error integrativa con valores grandes a través del método *setWindUpLimits( ).* Finalmente, en la función Loop, se computa el controlador dando como resultado los valores de acción de control y los envía por protocolo serial. Los códigos completos el lector lo puede encontrar en el Anexo A…

5.3.2 Implementación del controlador a las simulaciones

Finalmente, una vez se sintonizan los PID, son conectados directamente al puerto USB de la Raspberry, para que la acción de control sea recibida como se planteó en la sección 4.2.5.1.2.

Figura 5.4. Montaje Físico del Hardware in the Loop.



El montaje físico que se utilizó para implementar el Hardware in the Loop se puede ver en la figura 5.4. Las conexiones necesarias los protocolos de comunicación serial e I2C se puedan aplicar y la forma en que se cerro el lazo entre la Raspberry Pi 3B+ y el Arduino Due.

Al igual que los resultados en Simulink, los vistos desde el HIL son variados. Por un lado, el Helicóptero de Dos Grados de Libertad presenta un seguimiento de referencia gracias a los valores sintonizados previamente, pero este proceso es muy lento por los tiempos de procesamiento del software, por lo que tendría que pasar mucho tiempo antes de ver la acción en estado estacionario, aunque si se puede observar un intento de acción tal y como se muestra en las Figuras **Figura 5.5**, **Figura 5.6** y **Figura 5.7**.

Figura 5.5. Inicio del control del Helicóptero de dos grados de libertad. Nótese que el eje Pitch empieza a elevarse tratando de seguir la referencia.

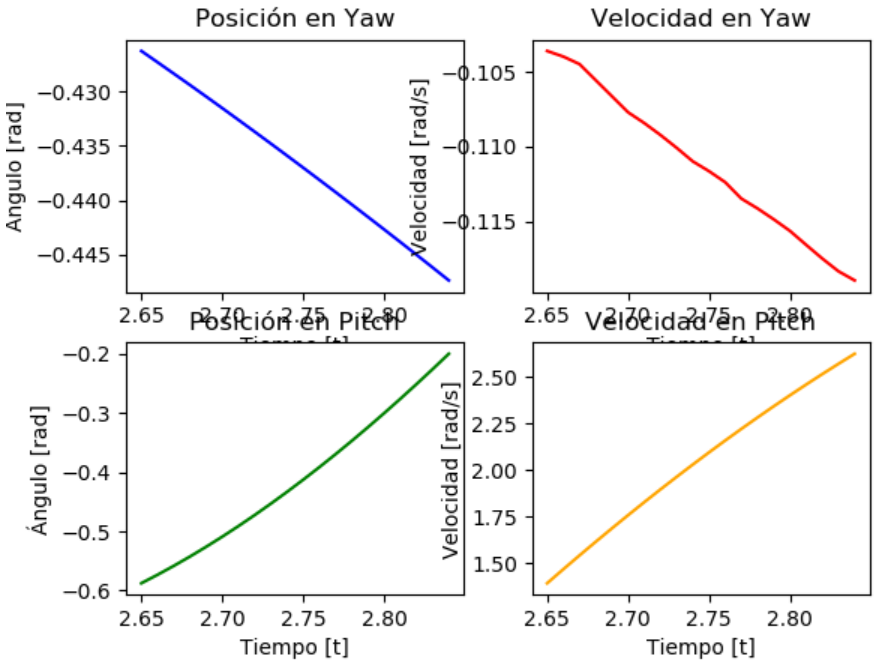


Figura 5.6. Etapa media del control del Helicóptero de dos grados de libertad. Nótese

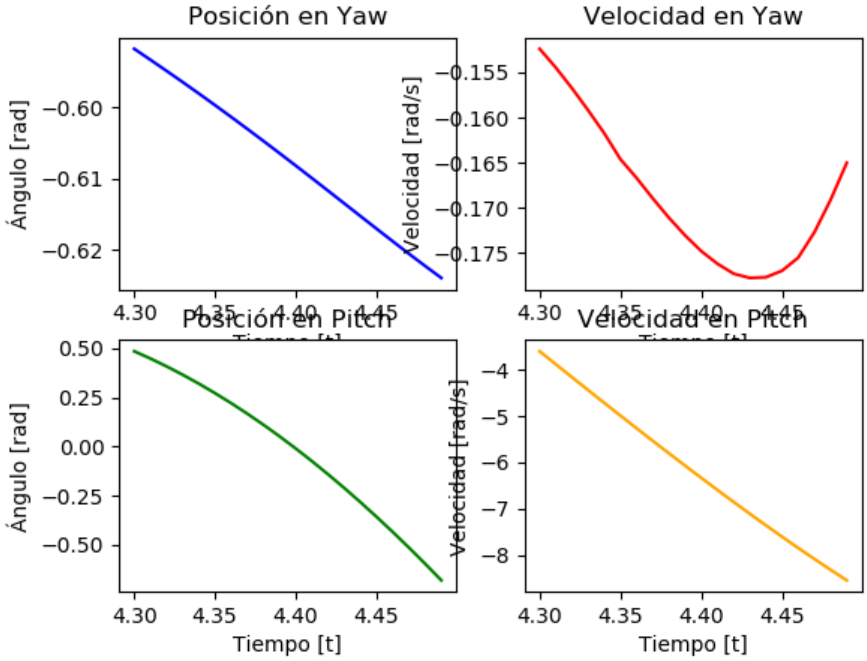
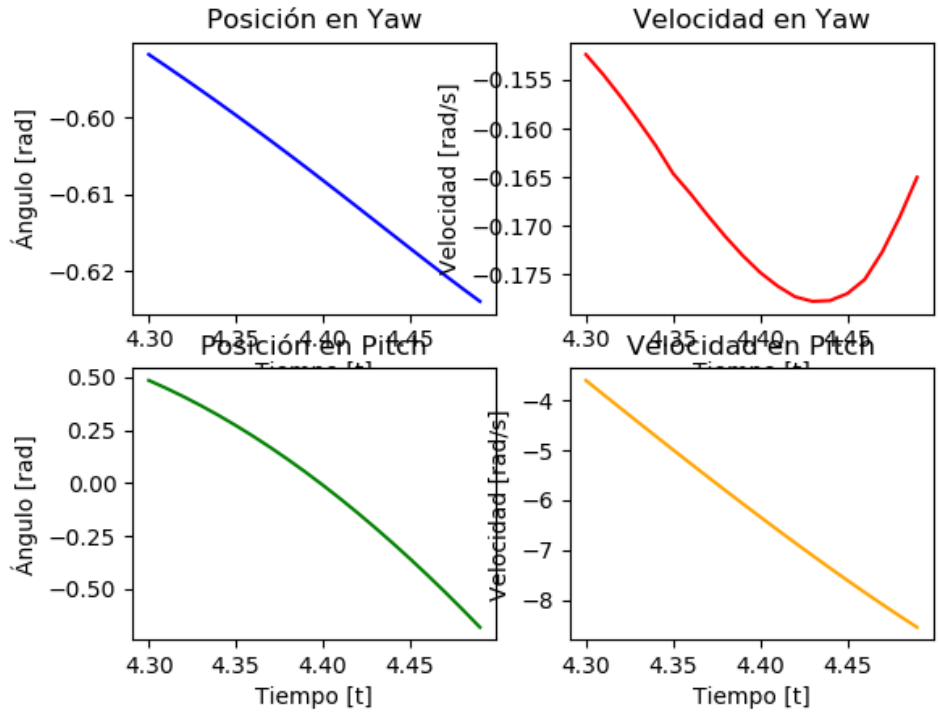
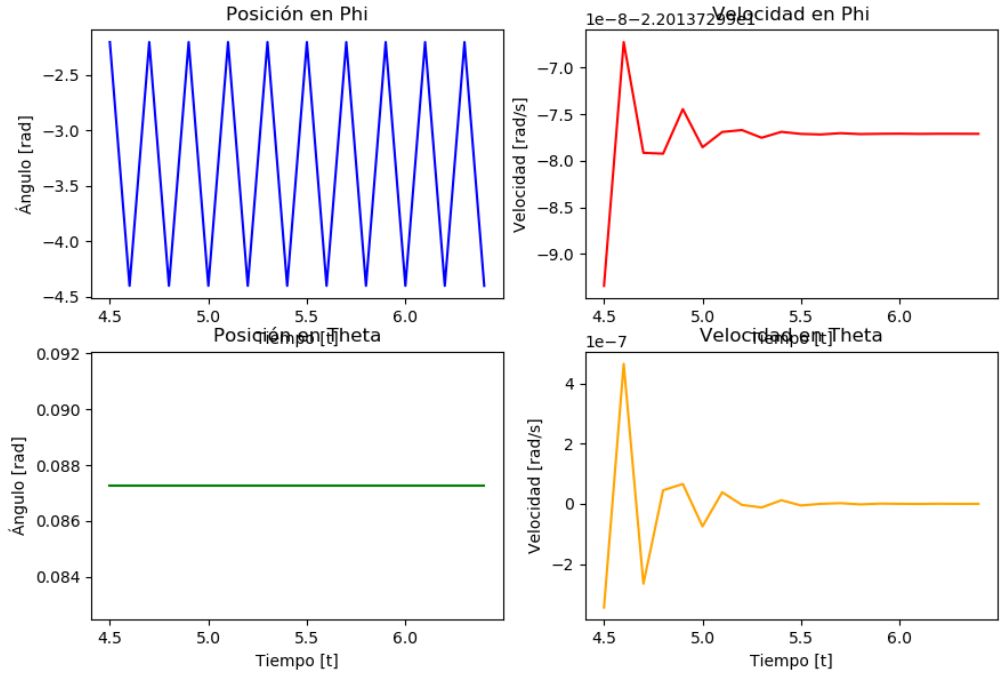


Figura 5.7. Continuación del control. Se evidencia que baja la referencia, tratando seguirla.



En contraste, el Péndulo de Furuta, gracias a que se aplicaron restricciones desde el controlador externo y desde el modelo implementado, impidieron que se observaran estas acciones de singularidad vistas a lo largo de este capítulo. Sin embargo, a pesar de que intenta mantener una acción de control, al ser un esfuerzo demasiado alto (del orden de $2.5 * 10^{25}$) y con las restricciones puestas, estos esfuerzos no son suficientes para llegar a la referencia deseada, tal y como se muestra en la Figura **Figura 5.8**.

Figura 5.8. Control de lazo cerrado del Péndulo de Furuta. Nótese el esfuerzo de control por parte de Phi, pero no suficiente para Theta.



Es por esto, que si bien se recibe la acción de control y se procesa siguiendo un cambio de referencia, es importante acotar dos limitantes: La primera, está relacionada con los tiempos de procesamiento y recepción de los datos ya que, al tener que ejecutar varios procesos al mismo tiempo, esto conlleva a que haya un retardo de entre 20 y 30 segundos en tiempo real de la recepción y envío de la acción de control, por lo cual no se vería de inmediato, y la segunda, viene ligada a la acción de control. Si bien es posible intentar validar el funcionamiento de las plantas a través de un PID, debido a la naturaleza de estos sistemas, se recomienda explorar otro tipo de controladores que si puedan ejercer una acción de control más óptima y rápida, aunque el cierre del lazo se evidencia de manera exitosa.

6. Conclusiones y trabajo futuro

6.1 Conclusiones

En la primera parte de este proyecto de investigación, al ser un apartado poco explorado dentro del área educativa, se procedió a realizar un minucioso levantamiento de la información relacionada con sistemas simulados o remotos enfocados al refuerzo de las competencias de desarrollo práctico de los estudiantes de ingeniería de las asignaturas de Control Industrial y Automatización. Dentro de esta exploración se encontraron ejemplos desarrollados en los últimos 20 años, con enfoques que van desde la replicación del comportamiento de las plantas en el tiempo haciendo uso de herramientas como Simulink o Labview, hasta programas remotos que permiten manipular plantas como un sistema de tanques para su control. Sin embargo, el hecho de apoyarse de un método numérico como solución al modelo matemático de cada una de estas plantas en su sentido más puro (es decir, sin ayuda de algún software especializado al respecto y diseñando por completo este algoritmo), además de graficar los datos en tiempo casi real y que se vean visualizados en señales medibles, ha sido un terreno aún menos explorado, por lo cual resultado de esto el enfoque hacia esta perspectiva pedagógica es novedoso.

Posteriormente, apoyados con documentación relacionada con los métodos numéricos y las Ecuaciones Diferenciales, se procedió a diseñar el algoritmo del método numérico que solucionaría el sistema de ED que caracterizaron a estas plantas. Sobre este apartado se acotan dos cosas: La primera, está ligada a los modelos matemáticos de cada una de estas plantas (refiérase al Péndulo de Furuta y el Helicóptero de Dos Grados de Libertad), pues al ser sistemas mecánicos complejos y con una dinámica peculiarmente difícil de tratar, se veían representados con un gran sistema de Ecuaciones (12 a 13 desde el análisis mecánico Newtoniano), por lo cual se decidió profundizar en los modelos y su solución a través del análisis geométrico y energético mediante las Ecuaciones de Euler-

Lagrange para al final obtener en ambos casos un sistema de dos ED de segundo orden acopladas. Se recurrió a la documentación de los autores citados en este documento (Navarridas, 2017) y (Vivas, 2011) para este modelado. Sin embargo, la profundización que se dio en el estudio de estos modelos permitió comprender aún mejor la dinámica de estas plantas, comprender su relación entre entradas y salidas, además de buscar la mejor solución para cada uno a través del diseño del algoritmo apropiado para estos sistemas. Por otra parte, la creación y la implementación del método de Runge- Kutta de cuarto orden para estos sistemas, resultó ser uno de los pasos más largos dentro de la investigación, debido a que las condiciones matemáticas de las ecuaciones requerían una modificación especial para su solución, pues se necesitaba solucionar un sistema de cuatro ecuaciones con cuatro incógnitas, una por cada variable de estado (posiciones y velocidades) y se validó mediante herramientas como Simulink y Matlab una vez fue implementado en Python. Con respecto a esto, se evidenció que fue conveniente trabajar los dos algoritmos por separado, debido a que cada sistema tiene peculiaridades tales como el tamaño del paso y el número de entradas y condiciones iniciales. Estos parámetros deben tenerse muy en cuenta, especialmente la asignación del paso, ya que, si no se obtiene un paso adecuado, el sistema puede desestabilizarse ya que se sigue trabajando en tiempo discreto y no se llegará a los resultados esperados, por lo que las pruebas experimentales permitieron encontrar las condiciones más adecuadas de simulación, con una precisión del 95%, ya que se considera un modelo ideal y por lo tanto, no se consideran perturbaciones en el modelo matemático, cuestión que se puede ver desde las plantas reales y que representaría mayor robustez matemática en dichos modelos.

En la siguiente parte, y ya con el algoritmo adecuado, se enfocaron los esfuerzos en el desarrollo de la interfaz HMI, la cual fue pensada con el objetivo de que fuera amigable y fácil de usar para el futuro usuario. Por lo tanto, se recurrió a la normativa ISA-101 para el diseño de interfaces de este tipo como apoyo para la distribución de botones, menús y colores del programa. Otra cuestión que debe resaltarse a lo largo de esta investigación fue el fuerte enfoque que se le otorgó al desarrollo del software desde su componente de programación, pues en lugar de apelar a una programación completamente secuencial, se tomó como inspiración el desarrollo de aplicaciones del mercado donde tanto su frontend como su backend, son creados por archivos modulares y orientados a objetos, lo cual permite no solo que el desarrollo sea más organizado, sino que también permite encontrar errores más fácilmente sin necesidad de modificar la

estructura completa, además de mejorar los tiempos de ejecución de los procesos, pues se garantiza que todo se ejecute al mismo tiempo sin errores de este tema. Adicionalmente, siempre se tuvo en constante consideración que la experiencia del usuario fuera la mejor, y esto se ve reflejado a través de las gráficas animadas y plasmar estos valores en señales medibles gracias al uso de dos conversores Digitales- Análogos mediante el protocolo I2C. Como era necesario que todas estas tareas fueran ejecutadas al mismo tiempo se aplicó la técnica de ejecución de tareas por hilos para de esta forma garantizar que todos estos procesos se ejecuten de manera correcta. Sin embargo, después de optimizar lo mejor posible este desarrollo, hay que acotar que existen ciertas limitaciones en su ejecución y en los tiempos en que lo realiza, pues se evidencia que si todos los procesos se ejecutan de manera conjunta, los tiempos de procesamiento son extremadamente altos, teniendo como consecuencia que los datos no sean obtenidos en tiempo estrictamente real, sino después de aproximadamente 20 a 30 segundos en este tiempo, entonces se puede concluir que si se obtiene y se visualizan los cambios tanto desde la interfaz como desde las señales medibles, pero no en tiempo completamente real, teniendo en cuenta este retardo de 20 a 30 segundos.

Por último, se realiza la validación del Hardware In The Loop mediante el diseño de un controlador PID, el cual recibiendo las señales medibles del software calculara la acción de control desde un controlador externo mediante una librería cuya configuración era paralela con discretización trapezoidal, y esta acción de control es retornada a la planta mediante protocolo serial, cerrando así el lazo. Se realizaron múltiples pruebas en Simulink para ambas plantas, teniendo resultados opuestos. Mientras que el helicóptero responde al controlador dentro de la simulación y aunque se demora en ver este resultado en el Hardware In The Loop debido a las limitaciones mencionadas anteriormente, se evidencia que sigue la referencia estipulada aunque con retardo en los intervalos anteriormente mencionados, en contraste el Péndulo de Furuta, debido a su dinámica altamente inestable (incapaz de mantenerse completamente vertical en tiempos inferiores a 1 s), en las simulaciones hechas se evidenció que en caso de realizarse con un PID, el esfuerzo de control es muy alto, y esto incurre en una singularidad de ejecución por valores muy altos (del orden de $2.5 * 10^{25}$), y por ello, la acción de control no es suficiente, por lo tanto, se podrían explorar controles más robustos, pero con la certeza de que estas acciones de control serán recibidas por el sistema y se verán plasmadas.

6.2 Trabajo futuro

A pesar de que se logró un desarrollo interesante del Hardware In The Loop propuesto a través del uso de un método numérico que cuenta con el equilibrio entre capacidad de cómputo y precisión, además de poder ejecutar procesos robustos de cálculo, recepción y transmisión de datos, existen limitaciones de tiempos de ejecución, pues estos son muy lentos si se desea ver una acción en tiempo real. Por lo tanto, es una buena alternativa profundizar en cómo optimizar aún más estos tiempos en pos de observar las acciones en tiempo real, además de considerar retomar este desarrollo sobre una plataforma más actual, como la Raspberry PI 4, la cual tiene grandes avances en procesamiento y cómputo, y podría dar una significativa mejora a la ejecución del programa.

Adicionalmente, se deja la puerta abierta a explorar formas más robustas de control que puedan satisfacer los problemas relacionados con el seguimiento de referencia en las plantas tales como el Control Difuso o las técnicas LQR, la cual es poco explorada dentro de sistemas embebidos de bajo coste, pues el software diseñado puede recibir estas acciones de control y podrían observarse resultados más rápidos y óptimos.

A. Anexo: Códigos y Diagrama de Flujo del Hardware in the Loop.

PresentacionPIR

```
import PENDULOINVERTIDODEFINITIVO
from I2C_DAC import I2C_DAC
from SERIAL import SERIAL
import numpy as np
import matplotlib.pyplot as plt
import math as m
from numpy import sin, pi, cos, sqrt
import tkinter as tk
from tkinter import messagebox
from threading import Thread
from matplotlib.lines import Line2D
import time
import matplotlib.animation as animation
data=[]
tempo=[]
data1=[]
data2=[]
data3=[]
data4=[]
lines=[]
Samples=20
Sampletime=10
numdata=4
resultado1=[]
resultado2=[]
resultado3=[]
resultado4=[]
lineatiempo=[]
auxiliartiempo=0
flag=True
valor=0.0
def recibedatos(ma, la, mp, lp, M, Jm, R, Kt, Ko, b, t, phi, vphi,
theta, vtheta, iniciar):

    fig=plt.figure()
    ax1=fig.add_subplot(2,2,1)
    ax1.title.set_text("Posición en Phi")
    ax1.set_xlabel("Tiempo [t]")
```

```
    ax1.set_ylabel("Ángulo [rad]")

    ax2=fig.add_subplot(2,2,2)
    ax2.title.set_text("Velocidad en Phi")
    ax2.set_xlabel("Tiempo [t]")
    ax2.set_ylabel("Velocidad [rad/s]")


    ax3=fig.add_subplot(2,2,3)
    ax3.title.set_text("Posición en Theta")
    ax3.set_xlabel("Tiempo [t]")
    ax3.set_ylabel("Ángulo [rad]")

    ax4=fig.add_subplot(2,2,4)
    ax4.title.set_text("Velocidad en Theta")
    ax4.set_xlabel("Tiempo [t]")
    ax4.set_ylabel("Velocidad [rad/s]")

    def Control():
        global flag
        global valor
        while(flag):
            #pass
            #print('control')
            lectura=SERIAL()
            if('nan' in lectura.recibir()):
                valor=0.0
                #print(0.0)
            if('V' in lectura.recibir()):
                print(lectura.recibir())
            else:
                valor=float(lectura.recibir())
                #print(valor)



    def Runge_Kutta_1(ma, la, mp, lp, M, Jm, R, Kt, Ko, b, t, phi, vphi,
theta, vtheta, iniciar):
        global valor
        i=0
        auxiliar=0
        auxiliar1=0
        rampa=0
        tempo.clear()
        data1.clear()
        data2.clear()
        data3.clear()
        data4.clear()
        lineatiempo.clear()
        resultado1.clear()
        resultado2.clear()
        resultado3.clear()
        resultado4.clear()
```

```
        G=9.81
        h=0.1
        t1=t+10
        vphi0=0
        f="vphi"
        g="((betha*(nombre-b*vphi)-
betha*gamma*((cos(theta))**2)*(sin(theta)*((vphi)**2))-
2*((betha)**2)*(cos(theta)*(sin(theta)*(vphi)*(vtheta))))/((alpha*betha)
-((gamma)**2)+((betha**2)+(gamma**2))*(sin(theta))**2)) +
(((betha)*(gamma)*(sin(theta)*(vtheta**2))-
(gamma)*(deltha)*(cos(theta))*(sin(theta)))/((alpha*betha)-
((gamma)**2)+((betha**2)+(gamma**2))*(sin(theta))**2))"
        W="vtheta"

Br="(((betha*(alpha+betha*sin(theta)**2)*cos(theta)*(sin(theta)*(vphi**2
))+2*(betha)*(gamma)*(cos(theta)**2)*(sin(theta)*(vphi)*(vtheta)))/((alp
ha*betha)-((gamma)**2)+((betha**2)+(gamma**2))*(sin(theta))**2))) + ((-
(gamma**2)*cos(theta)*(sin(theta)*(vtheta)**2)+deltha*(alpha+betha*(sin(
theta)**2))*sin(theta)-(gamma**2)*cos(theta)*(nombre-
b*vphi))/((alpha*betha)-
((gamma)**2)+((betha**2)+(gamma**2))*(sin(theta))**2))"

        alpha= Jm + ((M+(1/3)*ma+mp)*(la)**2)
        betha=((M+(1/3)*mp)*((lp)**2))
        gamma=((M+(1/2)*mp)*(la)*(lp))
        deltha=((M+(1/2)*mp)*(G)*(lp))


        T0= "((Kt/R)*valor)-((Ko*Kt)/R)*(vphi0)"


        n=m.ceil((t1-t)/h)

        pt=[]
        pphi=[]
        pvphi=[]
        ptheta=[]
        pvtheta=[]
        pT0=[]
        while iniciar == 's':
            auxiliar1=phi
            t0=t
            phi0=phi
            vphi0=vphi
            theta0=theta
            vtheta0=vtheta
            nombre=eval(T0)
            pt.append(t)
            pphi.append(phi)
            pvphi.append(vphi)
            ptheta.append(theta)
            pvtheta.append(vtheta)
            pT0.append(T0)
```

```
            rampa=(t-auxiliar)*0.1
            K1=(h)*eval(f)
            M1=(h)*eval(g)
            L1=(h)*eval(W)
            B1=(h)*eval(Br)

            k2=[t0 + (h/2), phi0 + (1/2)*K1, vphi0 + (1/2)*M1, theta0 +
(1/2)*L1, vtheta0 + (1/2)*B1]
            t= k2[0]
            phi= k2[1]
            vphi= k2[2]
            theta= k2[3]
            vtheta= k2[4]

            K2=(h)*eval(f)
            M2=(h)*eval(g)
            L2=(h)*eval(W)
            B2=(h)*eval(Br)

            k3=[t0 + (h/2), phi0 + (1/2)*K2, vphi0 + (1/2)*M2, theta0 +
(1/2)*L2, vtheta0 + (1/2)*B2]
            t= k3[0]
            phi= k3[1]
            vphi= k3[2]
            theta= k3[3]
            vtheta= k3[4]

            K3=(h)*eval(f)
            M3=(h)*eval(g)
            L3=(h)*eval(W)
            B3=(h)*eval(Br)

            k4=[t0 + h, phi0 + K3, vphi0 + M3, theta0 + L3, vtheta0 +
B3]
            t= k4[0]
            phi= k4[1]
            vphi= k4[2]
            theta= k4[3]
            vtheta= k4[4]

            K4=(h)*eval(f)
            M4=(h)*eval(g)
            L4=(h)*eval(W)
            B4=(h)*eval(Br)


            phi=phi0 + (1/6)*(K1 + 2*K2 + 2*K3 + K4)
            vphi=vphi0 + (1/6)*(M1 + 2*M2 + 2*M3 + M4)
            theta=theta0 + (1/6)*(L1 + 2*L2 + 2*L3 + L4)
            vtheta=vtheta0 + (1/6)*(B1 + 2*B2 + 2*B3 + B4)

            t=t0+h

            i=i+1
```

```
            lineatiempo.append(t)

            if (phi>6.2831):
                resultado1.append(phi-auxiliar1)
                phi=phi-auxiliar1
                resultado2.append(vphi)
                valordac2=int((325.87*(phi-auxiliar1))+2047.49)
            elif(phi<-6.2831):
                resultado1.append(phi-auxiliar1)
                phi=phi-auxiliar1
                resultado2.append(vphi)
                valordac2=int((325.87*(phi-auxiliar1))+2047.49)
            else:
                resultado1.append(phi)
                resultado2.append(vphi)
                valordac2=int((325.87*phi)+2047.49)


            if(theta>0.0872665):
                resultado3.append(0.0872665)
                resultado4.append(vtheta)
                #valordac1=int((11733.52*0.1745)+2047.49)
                valordac1=int((23462.61*0.0872665)+2047.49)

            elif(theta<-0.0872665):
                resultado3.append(-0.0872665)
                resultado4.append(vtheta)
                #valordac1=int((11733.52*-0.0872665)+2047.49)
                valordac1=int((23462.61*-0.0872665)+2047.49)

            else:
                resultado3.append(theta)
                resultado4.append(vtheta)
                #valordac1=int((11733.52*theta)+2047.49)
                valordac1=int((23462.61*theta)+2047.49)


            valordac1d=I2C_DAC(valordac1, valordac2)
            valordac1d.generar()


            while i%(n) == 0:
                valorcont=continuar()
                if valorcont=="yes":
                    iniciar='s'
                    auxiliar=t
                    i=i+1
                else:
                    iniciar='n'
                    i=i+1
            time.sleep(0.2)


    def continuar():
```

```
        valorc=messagebox.askquestion("Continuar simulación","¿Desea
continuar la simulación?")
        return valorc

    def plotdata(n):
        global auxiliartiempo
        global flag
        orr=len(lineatiempo)
        aux2=n+1
        if(aux2>=orr):
            auxiliartiempo=auxiliartiempo+0.1
            tempo.append(auxiliartiempo)
            data1.append(0)
            data2.append(0)
            data3.append(0)
            data4.append(0)
            flag=False
        else:
            flag=True
            auxiliartiempo=lineatiempo[n]
            tempo.append(lineatiempo[n])
            data1.append(resultado1[n])
            data2.append(resultado2[n])
            data3.append(resultado3[n])
            data4.append(resultado4[n])

        tempod=tempo[-Samples:]
        data1d=data1[-Samples:]
        data2d=data2[-Samples:]
        data3d=data3[-Samples:]
        data4d=data4[-Samples:]
        ax1.clear()
        ax2.clear()
        ax3.clear()
        ax4.clear()
        ax1.title.set_text("Posición en Phi")
        ax1.set_xlabel("Tiempo [t]")
        ax1.set_ylabel("Ángulo [rad]")
        ax2.title.set_text("Velocidad en Phi")
        ax2.set_xlabel("Tiempo [t]")
        ax2.set_ylabel("Velocidad [rad/s]")
        ax3.title.set_text("Posición en Theta")
        ax3.set_xlabel("Tiempo [t]")
        ax3.set_ylabel("Ángulo [rad]")
        ax4.title.set_text("Velocidad en Theta")
        ax4.set_xlabel("Tiempo [t]")
        ax4.set_ylabel("Velocidad [rad/s]")
        ax1.plot(tempod,data1d, color="blue")
        ax2.plot(tempod,data2d, color="red")
        ax3.plot(tempod,data3d, color="green")
        ax4.plot(tempod,data4d, color="orange")


    thread=Thread(target=Runge_Kutta_1, args=(ma, la, mp, lp, M, Jm, R,
```

```
Kt, Ko, b, t, phi, vphi, theta, vtheta, iniciar))
    thread1=Thread(target=Control)
    thread.start()
    thread1.start()
    ani=animation.FuncAnimation(fig,plotdata, interval=200,
repeat=False)
    plt.show()
    thread.join()
    thread1.join()
```

PresentacionH2DOF

```
import PENDULOINVERTIDODEFINITIVO
from I2C_DAC import I2C_DAC
from SERIAL import SERIAL
import numpy as np
import matplotlib.pyplot as plt
import math as m
from numpy import sin, pi, cos, sqrt
import tkinter as tk
from tkinter import messagebox
from threading import Thread
from matplotlib.lines import Line2D
import time
import matplotlib.animation as animation

data=[]
tempo=[]
data1=[]
data2=[]
data3=[]
data4=[]
lines=[]
Samples=20
Sampletime=10
numdata=4
resultado1=[]
resultado2=[]
resultado3=[]
resultado4=[]
lineatiempo=[]
auxiliartiempo=0
flag=True
E1=0.0
E2=0.0
def
recibedatos(Bp,By,Mhelip,Jeqp,Jeqy,Lmc,z,t,phi,vphi,theta,vtheta,iniciar
):

    fig=plt.figure()
    ax1=fig.add_subplot(2,2,1)
    ax1.title.set_text("Posición en Yaw")
    ax1.set_xlabel("Tiempo [t]")
    ax1.set_ylabel("Ángulo [rad]")
```

```
    ax2=fig.add_subplot(2,2,2)
    ax2.title.set_text("Velocidad en Yaw")
    ax2.set_xlabel("Tiempo [t]")
    ax2.set_ylabel("Velocidad [rad/s]")


    ax3=fig.add_subplot(2,2,3)
    ax3.title.set_text("Posición en Pitch")
    ax3.set_xlabel("Tiempo [t]")
    ax3.set_ylabel("Ángulo [rad]")

    ax4=fig.add_subplot(2,2,4)
    ax4.title.set_text("Velocidad en Pitch")
    ax4.set_xlabel("Tiempo [t]")
    ax4.set_ylabel("Velocidad [rad/s]")

    def Control():
        global flag
        global E1
        global E2
        recepcion=[]
        while(flag):
            lectura=SERIAL()

            if('A' in lectura.recibir()):
                E1=float(lectura.recibir().replace("A",""))
            if('B' in lectura.recibir()):
                E2=float(lectura.recibir().replace("B",""))


    def
Runge_Kutta_2(Bp,By,Mheli,Jeqp,Jeqy,Lmc,z,t,phi,vphi,theta,vtheta,inicia
r):
        global E1
        global E2
        i=0
        auxiliar=0
        auxiliar1=0
        rampa=0
        tempo.clear()
        data1.clear()
        data2.clear()
        data3.clear()
        data4.clear()
        lineatiempo.clear()
        resultado1.clear()
        resultado2.clear()
        resultado3.clear()
        resultado4.clear()
        G=9.81
        h=0.01
        t1=t+1
        Kpp=0.02638
        Kpy=0.001894
        Kyp=0.002096
```

```
        Kyy=0.01871
        Fcpp=-0.208
        Fcpy=0.0064
        Fcyp=-0.0072
        Fcyy=0.0796


        f="vtheta"
        g="vphi"
        B="(-Mheli*vphi**2*(sin(2*theta)*(Lmc**2-z**2)/2-
Lmc*z*cos(2*theta))-Mheli*G*(Lmc*cos(theta)+z*sin(theta))-
Bp*vtheta+Tpp+Tpy)/(Jeqp+Mheli*(Lmc**2+z**2))"
        W="(-Mheli*(sin(2*theta)*(z**2-
Lmc**2)+2*Lmc*z*cos(2*theta))*vtheta*vphi-
By*vphi+Typ+Tyy)/(Jeqy+Mheli*(cos(theta)**2*(Lmc**2-
z**2)+Lmc*z*sin(2*theta)+z**2))"


        n=m.ceil((t1-t)/h)

        pt=[]
        pphi=[]
        pvphi=[]
        ptheta=[]
        pvtheta=[]

        while iniciar == 's':

            auxiliar1=phi
            if(theta>0.7068):
                theta=0.7068
            elif(theta<-0.7068):
                theta=-0.7068

            if (phi>6.2831):
                phi=phi-auxiliar1
            elif(phi<-6.2831):
                phi=phi-auxiliar1

            valordac2=int((325.87*phi)+2047.49)
            valordac1=int((2896.85*theta)+2047.49)
            t0=t
            phi0=phi
            vphi0=vphi
            theta0=theta
            vtheta0=vtheta

            Tpp=eval("Kpp*E1+Fcpp")
            Tpy=eval("Kpy*E2+Fcpy")
            Typ=eval("(Kyp*E1+Fcyp)*cos(theta)")
            Tyy=eval("(Kyy*E2+Fcyy)*cos(theta)")


            pt.append(t)
```

```
            pphi.append(phi)
            pvphi.append(vphi)
            ptheta.append(theta)
            pvtheta.append(vtheta)
            rampa=(t-auxiliar)*0.01

            K1=(h)*eval(f)
            M1=(h)*eval(g)
            L1=(h)*eval(B)
            B1=(h)*eval(W)

            k2=[t0 + (h/2), theta0 + (1/2)*K1, phi0 + (1/2)*M1, vtheta0
+ (1/2)*L1, vphi0 + (1/2)*B1]
            t= k2[0]
            phi= k2[2]
            vphi= k2[4]
            theta= k2[1]
            vtheta= k2[3]

            K2=(h)*eval(f)
            M2=(h)*eval(g)
            L2=(h)*eval(B)
            B2=(h)*eval(W)

            k3=[t0 + (h/2), theta0 + (1/2)*K2, phi0 + (1/2)*M2, vtheta0
+ (1/2)*L2, vphi0 + (1/2)*B2]
            t= k2[0]
            phi= k2[2]
            vphi= k2[4]
            theta= k2[1]
            vtheta= k2[3]

            K3=(h)*eval(f)
            M3=(h)*eval(g)
            L3=(h)*eval(B)
            B3=(h)*eval(W)

            k4=[t0 + h, theta0 + K3, phi0 + M3, vtheta0 + L3, vphi0 +
B3]
            t= k2[0]
            phi= k2[2]
            vphi= k2[4]
            theta= k2[1]
            vtheta= k2[3]

            K4=(h)*eval(f)
            M4=(h)*eval(g)
            L4=(h)*eval(B)
            B4=(h)*eval(W)


            theta=theta0 + (1/6)*(K1 + 2*K2 + 2*K3 + K4)
            phi=phi0 + (1/6)*(M1 + 2*M2 + 2*M3 + M4)
            vtheta=vtheta0 + (1/6)*(L1 + 2*L2 + 2*L3 + L4)
            vphi=vphi0 + (1/6)*(B1 + 2*B2 + 2*B3 + B4)
```

```
            t=t0+h

            i=i+1

            lineatiempo.append(t)

            if (phi>6.2831):
                resultado1.append(phi-auxiliar1)
                resultado2.append(vphi)
            elif(phi<-6.2831):
                resultado1.append(phi-auxiliar1)
                resultado2.append(vphi)
            else:
                resultado1.append(phi)
                resultado2.append(vphi)

            if(theta>0.7068):
                resultado3.append(0.7068)
                resultado4.append(vtheta)
            elif(theta<-0.7068):
                resultado3.append(-0.7068)
                resultado4.append(vtheta)
            else:
                resultado3.append(theta)
                resultado4.append(vtheta)


            valordac1d=I2C_DAC(valordac1, valordac2)
            valordac1d.generar()


            while i%(n) == 0:
                valorcont=continuar()
                if valorcont=="yes":
                    iniciar='s'
                    auxiliar=t
                    i=i+1
                else:
                    iniciar='n'
                    i=i+1
            time.sleep(0.1)

    def continuar():
        valorc=messagebox.askquestion("Continuar simulación","¿Desea
continuar la simulación?")
        return valorc

    def plotdata(n):
        global auxiliartiempo
        global flag
        orr=len(lineatiempo)
        aux2=n+1
```

```
        if(aux2>=orr):
            flag=False
            auxiliartiempo=auxiliartiempo+0.1
            tempo.append(auxiliartiempo)
            data1.append(0)
            data2.append(0)
            data3.append(0)
            data4.append(0)
        else:
            flag=True
            auxiliartiempo=lineatiempo[n]
            tempo.append(lineatiempo[n])
            data1.append(resultado1[n])
            data2.append(resultado2[n])
            data3.append(resultado3[n])
            data4.append(resultado4[n])


        tempod=tempo[-Samples:]
        data1d=data1[-Samples:]
        data2d=data2[-Samples:]
        data3d=data3[-Samples:]
        data4d=data4[-Samples:]
        ax1.clear()
        ax2.clear()
        ax3.clear()
        ax4.clear()
        ax1.title.set_text("Posición en Yaw")
        ax1.set_xlabel("Tiempo [t]")
        ax1.set_ylabel("Ángulo [rad]")
        ax2.title.set_text("Velocidad en Yaw")
        ax2.set_xlabel("Tiempo [t]")
        ax2.set_ylabel("Velocidad [rad/s]")
        ax3.title.set_text("Posición en Pitch")
        ax3.set_xlabel("Tiempo [t]")
        ax3.set_ylabel("Ángulo [rad]")
        ax4.title.set_text("Velocidad en Pitch")
        ax4.set_xlabel("Tiempo [t]")
        ax4.set_ylabel("Velocidad [rad/s]")
        ax1.plot(tempod,data1d, color="blue")
        ax2.plot(tempod,data2d, color="red")
        ax3.plot(tempod,data3d, color="green")
        ax4.plot(tempod,data4d, color="orange")



    thread=Thread(target=Runge_Kutta_2,
args=(Bp,By,Mhelip,Jeqp,Jeqy,Lmc,z,t,phi,vphi,theta,vtheta,iniciar))
    thread.start()
    thread1=Thread(target=Control)
    thread1.start()
    ani=animation.FuncAnimation(fig,plotdata, interval=5,repeat=False)
    plt.show()
    thread.join()
    thread1.join()
```

PID_PIR

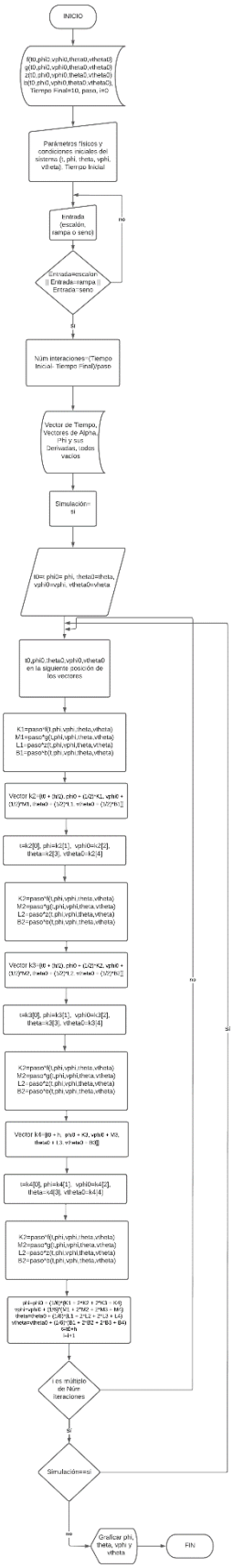
```
#include "ArduPID.h"
ArduPID myController;

double input;
double output;

double setpoint = 1.65;
double p = 5000;
double i = 0;
double d = 24137687.8676344;

void setup()
{
  Serial.begin(115200);
  myController.begin(&input, &output, &setpoint, p, i, d);

  myController.setOutputLimits(-7.4, 7.4);
  myController.setBias(0);
  myController.setWindUpLimits(-100000, 100000);

  myController.start();

}
void loop()
{
  String salida="V";
  input = (analogRead(A0)*3.3)/1023;
  salida+=input;
  myController.compute();
  Serial.println(output);
  Serial.println(salida);
  delay(700);
}
```

PID_H2DOF

```
#include "ArduPID.h"

ArduPID myController;
ArduPID myController1;

//INPUT PITCH
double input;
double output;
//-----------------------------------------------------------------------
//INPUT YAW
double input1;
double output1;
```

```
//SET POINT PITCH
double setpoint = 1.65;
double p = 3.04816532853804;
double i = 1.66257814737297;
double d = 1.2685393111935;
//-----------------------------------------------------------------------
//SET POINT YAW
double setpoint1 = 1.65;
double p1 = 12.3502508171481;
double i1 = 0.227131707099136;
double d1 = 0.80502513762335;

void setup()
{
  Serial.begin(115200);
  myController.begin(&input, &output, &setpoint, p, i, d);
  myController1.begin(&input1, &output1, &setpoint1, p1, i1, d1);

  myController.setSampleTime(10);
  myController.setOutputLimits(-22, 22);
  myController.setBias(0);
  myController.setWindUpLimits(-100000, 100000);

//-----------------------------------------------------------------------
-------------------------------------------------
  myController1.setSampleTime(100);
  myController1.setOutputLimits(-24, 24);
  myController1.setBias(0);
  myController1.setWindUpLimits(-100000, 100000);

  myController.start();
  myController1.start();
}

void loop()
{
  input = (analogRead(A0)*3.3)/1023;
  input1=(analogRead(A1)*3.3)/1023;
  String salida="A";
  String salida1="B";

  myController.compute();
  myController1.compute();

  salida+=output;
  salida1+=output1;
  Serial.println(salida);
  Serial.println(salida1);
  delay(100);

}
```

Diagrama de Flujo Método Runge Kutta.


INICIO
Parámetros físicos y condiciones iniciales del sistema (L, phi, theta, vphi, vtheta). Tiempo Inicial
Entrada (escalón, rampa o seno)
no
Entrada=escalon || Entrada=rampa || Entrada=seno
si
Vector de Tiempo, Vectores de Alpha, Phi y sus Derivadas, todos vacios
Simulación= si
t0= phi0= phi, theta0=theta, vphi0=vphi, vtheta0=vtheta
t0,phi0,theta0,vphi0,vtheta0 en la siguiente posición de los vectores
K1=paso*f(t,phi,vphi,theta,vtheta)
M1=paso*g(t,phi,vphi,theta,vtheta)
L1=paso*z(t,phi,vphi,theta,vtheta)
B1=paso*b(t,phi,vphi,theta,vtheta)
Vector k2=[t0 + 0/2, phi0 + L/2*K1, vphi0 + M/2*M1, theta0 + L/2*L1, vtheta0 + B/2*B1]
t=k2[0], phi=k2[1], vphi0=k2[2], theta=k2[3], vtheta0=k2[4]
K2=paso*f(t,phi,vphi,theta,vtheta)
M2=paso*g(t,phi,vphi,theta,vtheta)
L2=paso*z(t,phi,vphi,theta,vtheta)
B2=paso*b(t,phi,vphi,theta,vtheta)
Vector k3=[t0 + 0/2, phi0 + L/2*K2, vphi0 + M/2*M2, theta0 + L/2*L2, vtheta0 + B/2*B2]
t=k3[0], phi=k3[1], vphi0=k3[2], theta=k3[3], vtheta0=k3[4]
K3=paso*f(t,phi,vphi,theta,vtheta)
M3=paso*g(t,phi,vphi,theta,vtheta)
L3=paso*z(t,phi,vphi,theta,vtheta)
B3=paso*b(t,phi,vphi,theta,vtheta)
Vector k4=[t0 + h, phi0 + K3, vphi0 + M3, theta0 + L3, vtheta0 + B3]
t=k4[0], phi=k4[1], vphi0=k4[2], theta=k4[3], vtheta0=k4[4]
K4=paso*f(t,phi,vphi,theta,vtheta)
M4=paso*g(t,phi,vphi,theta,vtheta)
L4=paso*z(t,phi,vphi,theta,vtheta)
B4=paso*b(t,phi,vphi,theta,vtheta)
phi=phi0 + (1/6)*(K1 + 2*K2 + 2*K3 + K4)
vphi=vphi0 + (1/6)*(M1 + 2*M2 + 2*M3 + M4)
theta=theta0 + (L/6)*(L1 + 2*L2 + 2*L3 + L4)
vtheta=vtheta0 + (L/6)*(B1 + 2*B2 + 2*B3 + B4)
t=t0+h
i es múltiplo de Num Iteraciones
si
Simulación==si
no
Graficar phi, theta, vphi y vtheta
FIN

Bibliografía

AliExpress. (n.d.).

Andújar Márquez, J. M., & Mateo Sanguino, T. J. (2010). Diseño de Laboratorios Virtuales y/o Remotos. Un Caso Práctico. *Revista Iberoamericana de Automática e Informática Industrial RIAI*, *7*(1), 64–72. https://doi.org/10.1016/s1697-7912(10)70009-1

Aprendiendo Arduino. (n.d.).

Arduino. (n.d.). *Arduino Due | Arduino.cl - Compra tu Arduino en Línea*. Retrieved July 30, 2022, from https://arduino.cl/producto/arduino-due/

Departamento EDAN, U. de S. (2019). Tema 4. Métodos numéricos. In *MATEMÁTICAS APLICADAS A LA BIOLOGÍA- GRADO EN BIOLOGÍA* (pp. 159–181).

Electronic Components Datasheet Search. (n.d.).

Eugenio Lopez Aldea. (2017). *Raspberry Pi Fundamentos y Aplicaciones*.

Giron Rodriguez, J., & Naranjo Grisales, A. F. (2018). *Diseño e implementación de un laboratorio virtual usando seis plantas dispuestas en el Laboratorio de automática de la Universidad Autónoma de Occidente*. Universidad Autónoma de Occidente.

Gonzalez Vivas, C. E. (2011). *Control del Helicoptero 2d Usando Metodos de Control Robusto h ∞*. Universidad Nacional de Colombia.

Guerra Carmenate, J. (2022). *ESP32 Wifi y Bluetooth en un solo chip*. https://programarfacil.com/esp8266/esp32/

Navaridas, F. C. (2017). *Control de un péndulo invertido rotatorio con hardware de bajo coste*. Universidad de La Rioja.

Pastor, J. (2018). *Xataka*.

Pedro, M. I. A., Medina, L., Saba, M. I. G. H., Hernández, M. S. I. J., Ladrón, M. C. E., & Durán, D. G. (1940). *Los Laboratorios Virtuales 1 y Laboratorios Remotos en la Enseñanza de la Ingeniería*. *August 2017*, 24–30.

Penin, A. R. (n.d.). *Sistemas SCADA*.

Porras Rodríguez, C. C., & Guatibonza Pérez, J. A. (2020). *Sistema Para El Desarrollo De Estrategias De Control on / Off , Pid Y Lqr Aplicadas Al Helicóptero Quanser 2 Dof Del Laboratorio De Control De La Facultad Tecnológica.*

Santana, I., Ferre, M., Hernández, L., Aracil, R., Rodríguez, Y., & Pinto, E. (2010). Aplicación del Sistema de Laboratorios a Distancia en Asignaturas de Regulación Automática. *Revista Iberoamericana de Automática e Informática Industrial RIAI*, *7*(1), 46–53. https://doi.org/10.1016/s1697-7912(10)70007-8

STMicroelectronics. (2020). *Datasheet - STM32F205xx STM32F207xx Arm®-based 32-bit MCU*. *July*. https://www.st.com/resource/en/datasheet/stm32f207ig.pdf

Valera, A., Soriano, A., & Vall, M. (2014). *A. Valera ∗ , A. Soriano, M. Vall´ es*. *11*, 363–376. https://doi.org/10.1016/j.riai.2014.09.002

Zill, D. G. (1997). Capítulo 9. Métodos numéricos. In *Ecuaciones Diferenciales con aplicaciones de modelado* (pp. 400–436).