



Universidad Nacional Experimental del Táchira

Vicerrectorado Académico

Decanato de Docencia

Departamento de Ingeniería Electrónica

Trabajo de Aplicación Profesional

Proyecto Especial de Grado

Laboratorio virtual de sistemas de control clásicos y difusos utilizando software libre

Autor: Kleiver J. Carrasco M.

C.I: V-24.743.884

kleiver.carrasco@unet.edu.ve

Tutor: MSc. Juan R. Vizcaya R.

jvizcaya@unet.edu.ve

San Cristóbal, enero de 2020



Universidad Nacional Experimental del Táchira

Vicerrectorado Académico

Decanato de Docencia

Departamento de Ingeniería Electrónica

Trabajo de Aplicación Profesional

Proyecto Especial de Grado

Laboratorio virtual de sistemas de control clásicos y difusos utilizando software libre

Proyecto Especial de Grado presentado como requisito parcial para optar
al título de Ingeniero en Electrónica

Autor: Kleiver J. Carrasco M.

C.I: V-24.743.884

kleiver.carrasco@unet.edu.ve

Tutor: MSc. Juan R. Vizcaya R.

jvizcaya@unet.edu.ve

San Cristóbal, enero de 2020



Universidad Nacional Experimental del Táchira

Vicerrectorado Académico

Decanato de Docencia

Departamento de Ingeniería Electrónica

Trabajo de Aplicación Profesional

Proyecto Especial de Grado

Aprobación del Tutor para presentación del Proyecto Especial de Grado

Yo, Juan Rafael Vizcaya Rojas, en mi carácter de Tutor del Proyecto Especial de Grado titulado “**Laboratorio virtual de sistemas de control clásicos y difusos utilizando software libre**”, presentado por el bachiller Kleiver Jesús Carrasco Márquez titular de la cédula de identidad No. V-24.743.884 por medio de la presente autorizo su presentación ante el Jurado que se designe, en virtud de considerar que reúne los requisitos y méritos necesarios para ser sometido a presentación pública.

Tutor
Juan Rafael Vizcaya Rojas
C.I. V-7.348.191

AGRADECIMIENTOS

(Aun por redactar)

UNIVERSIDAD NACIONAL EXPERIMENTAL DEL TÁCHIRA
VICERRECTORADO ACADÉMICO
DECANATO DE DOCENCIA
CARRERA DE INGENIERÍA ELECTRÓNICA

**Laboratorio virtual de sistemas de control clásicos y difusos
utilizando software libre**

Autor: Kleiver J. Carrasco M.
Tutor: MSc. Juan R. Vizcaya R.
Fecha: Enero, 2020

RESUMEN

La investigación realizada se llevó acabo con la finalidad de crear un software enfocado en el área de los sistemas de control. El Laboratorio Virtual de sistemas de control clásicos y difusos apunta a utilizarse de forma similar a MATLAB y SciLab, particularmente, con el propósito de hacer uso del mismo en el área del control de procesos haciendo uso de software libre, específicamente, el lenguaje de programación libre Python. Con la finalidad de cumplir con los objetivos planteados se hizo uso de librerías para Python externas de control de procesos, lógica difusa, calculo numérico, entre otras, además, se realizó la implementación de algoritmos de solución de ecuaciones diferenciales para la simulación de los sistemas de control con controladores difusos. El resultado de la investigación fue una aplicación confiable y práctica para el análisis, diseño y simulación de controladores clásicos y difusos manteniendo un enfoque simple y de uso rápido que logra competir en cierta medida con otras herramientas de corte similar, y a su vez, dejando abierta la posibilidad de implementar mejoras a futuro o la creación de nuevas funciones.

Descriptores: Sistemas de control, logica difusa, Python, Laboratorio Virtual

ÍNDICE

	Pág.
CARTA DE APROBACIÓN PARA PRESENTACIÓN	iii
AGRADECIMIENTOS	iv
RESUMEN	v
INTRODUCCIÓN	1
CAPÍTULO I EL PROBLEMA	3
Planteamiento del problema	3
Objetivos de la investigación	5
Justificación e importancia	5
Alcance y limitaciones	6
CAPÍTULO II MARCO TEÓRICO	7
Antecedentes	7
Bases teóricas	8
CAPÍTULO III MARCO METODOLÓGICO	44
Tipo de investigación	44
Diseño de la investigación	44
Modalidad	45
Fases de la investigación	45
CAPÍTULO IV RESULTADOS	47
Estructura general	47
Análisis de sistemas de control	48
Entonación de controladores PID	51
Diseño de controladores difusos	53
Simulación de sistemas de control	56
Comparación con MATLAB y SciLab	61

Pág.

CAPÍTULO V CONCLUSIONES Y RECOMENDACIONES	81
Conclusiones	81
Recomendaciones	82
REFERENCIAS	84
ANEXOS	88

ÍNDICE DE FIGURAS

FIGURA	Pág.
1 Ejemplo de un sistema en lazo abierto	19
2 Ejemplo de un sistema en lazo cerrado	20
3 Ejemplo de gráfica del lugar de las raíces	21
4 Ejemplo de análisis de estabilidad en el plano complejo	22
5 Ejemplo de análisis de estabilidad con diagrama de Nyquist	23
6 Ejemplo 1 márgenes de ganancia y fase	25
7 Ejemplo 2 márgenes de ganancia y fase	25
8 Ejemplo de un sistema en lazo cerrado con controlador	26
9 Modelado por curva de reacción	30
10 Ejemplo de un conjunto difuso	32
11 Esquema de control: P difuso	39
12 Esquema de control: PD difuso	39
13 Esquema de control: PI difuso	40
14 Esquema de control: PID difuso	40
15 Esquema de control: Programador de ganancias	41
16 Esquema de control: PID clásico mas P difuso	41
17 Diagrama general de la aplicación	47
18 Comparación de análisis/MATLAB - sistema continuo 1	66
19 Controlador difuso 1 para la comparación	68
20 Comparación - Respuesta del controlador difuso 1	71
21 Comparación - Respuesta del controlador difuso 2 - Laboratorio Virtual	71
22 Comparación - Respuesta del controlador difuso 2 - MATLAB	72
23 Comparación - Respuesta del controlador difuso 2 - SciLab	72
24 Simulación del controlador PID difuso en tiempo continuo	75
25 Simulación del controlador PI difuso en tiempo continuo	76

26	Respuesta de controladores en tiempo discreto - PDG y PI difuso	79
27	Interfaz gráfica para el análisis de sistemas de control	88
28	Interfaz gráfica para la entonación de controladores PID	89
29	Interfaz gráfica - difusa - estructura	91
30	Interfaz gráfica - difusa - entradas/salidas	92
31	Interfaz gráfica - difusa - reglas	92
32	Interfaz gráfica - difusa - prueba y respuesta	93
33	Interfaz gráfica para la simulación de sistemas de control	95
34	Interfaz gráfica - simulación - Barras de configuración	96
35	Esquema de control implementado: PD difuso	104
36	Esquema de control implementado: PI difuso	104
37	Esquema de control implementado: PID difuso	104
38	Esquema de control implementado: PD difuso mas PI difuso	105
39	Esquema de control implementado: PI difuso más derivada	105
40	Esquema de control implementado: PD difuso más integrador	105
41	Esquema de control implementado: Programador de ganancias	106
42	Esquema de control implementado: PID clásico más P difuso	106
43	Comparación de análisis/MATLAB - sistema continuo 2	123
44	Comparación de análisis/MATLAB - sistema continuo 3	124
45	Comparación de análisis/SciLab - sistema continuo 1	125
46	Comparación de análisis/SciLab - sistema continuo 2	126
47	Comparación de análisis/MATLAB - sistema discreto 1	127
48	Comparación de análisis/MATLAB - sistema discreto 2	128
49	Comparación de análisis/MATLAB - sistema discreto 3	129
50	Comparación de análisis/SciLab - sistema discreto 1	130

FIGURA**Pág.**

- | | | |
|----|--|-----|
| 51 | Comparación de simulación de sistemas de control continuos - 1 | 138 |
| 52 | Comparación de simulación de sistemas de control continuos - 2 | 139 |
| 53 | Comparación de simulación de sistemas de control discretos - 1 | 139 |
| 54 | Comparación de simulación de sistemas de control discretos - 2 | 140 |

ÍNDICE DE TABLAS

TABLA	Pág.
1 Regla de entonación de Zigler-Nichols	30
2 Regla de entonación de Cohen-Coon	31
3 Sistemas para la comparación de análisis de sistemas de control	61
4 Comparación de análisis - tiempo continuo	62
5 Comparación de análisis - tiempo discreto	63
6 Comparación de diseño de controladores difusos - controlador 1	69
7 Comparación de diseño de controladores difusos - controlador 2	70
8 Parámetros para el Laboratorio Virtual en tiempo continuo	73
9 Parámetros para MATLAB en tiempo continuo	74
10 Parámetros para SciLab en tiempo continuo	74
11 Tiempos de ejecución para la función de simulación de sistemas de control en tiempo continuo	77
12 Parámetros de simulación en tiempo discreto	78
13 Tiempos de ejecución para la función de simulación de sistemas de control en tiempo discreto	79

ÍNDICE DE CÓDIGOS

CÓDIGO	Pág.
1 Pseudo código - Análisis de sistemas de control	48
2 Pseudo código - Entonación de controladores PID	51
3 Pseudo código - Diseño de controladores difusos	53
4 Pseudo código - Simulación de sistemas de control	57
5 Pseudo código - Runge-Kutta explícitos	60
6 Pseudo código - Runge-Kutta embebidos	60
7 Calculo de los margenes de ganancia y fase	98
8 Transformación equivalente entre funciones de membresía	101
9 Formato para guardar controlador	107
10 Extracción de datos del archivo FIS - YAPFLM	110
11 Procesado de los datos del FIS	112
12 Exportar archivos FIS	113
13 Tamaño de paso variable para Runke-Kutta explícitos	121
14 Tamaño de paso variable para Runke-Kutta embebidos	122
15 Función para calcular la norma RMS	122

INTRODUCCIÓN

El presente trabajo plantea la creación de un software enfocado en el área de los sistemas de control. Así como MATLAB es un laboratorio de matrices y SciLab es un laboratorio científico, el Laboratorio Virtual de sistemas de control clásicos y difusos apunta a algo similar con el propósito de utilizarse en el área de control por parte de ingenieros que puedan estar buscando una alternativa simple y gratuita para abarcar los problemas del control de procesos.

Los sistemas de control han incrementado su importancia en el tiempo, lo cual se debe a la necesidad de automatizar los procesos de producción de bienes y servicios para ofrecer productos de mejor calidad a un mejor precio. Los procesos a controlar para la producción son, con frecuencia, complejos y no lineales, por tanto, es común el uso de controladores diferentes al clásico PID, una de estas alternativas se basa en emplear controladores con base en la lógica difusa para contrarrestar los efectos no lineales del proceso a controlar.

Así mismo, la problemática a resolver es la necesidad existente de software para el análisis, diseño y simulación de sistemas de control, el cual, existe de manera reducida de forma gratuita y libre. Opciones como SciLab y Octave no pueden competir con entornos cerrados y de pago como MATLAB en cuanto a opciones y desempeño, adicionalmente, la posibilidad de implementar controladores difusos es tratado como una función opcional y de poca prioridad que no viene por defecto en las alternativas de software mencionados.

Con el objetivo de solventar esta problemática se plantea la creación de un Laboratorio Virtual utilizando el lenguaje de programación libre y gratuito Python, el cual posee librerías externas aptas para la tarea expuesta, todo lo anterior, con la finalidad de otorgar la posibilidad de analizar, diseñar y simular sistemas de control clásicos y difusos por medio de una interfaz gráfica sin la necesidad de instalar elementos externos o escribir alguna línea de código de programación a la vez que se realiza de forma rápida y sencilla.

A continuación, se describe de forma más específica la problemática que atiende esta investigación. Pasado este capítulo, se exponen los conceptos teóricos necesarios para poder implementar un software con las características ya descritas que logre competir con otras herramientas de corte similar, luego, se continua con la metodología empleada para llevar a cabo con éxito la realización del Laboratorio Virtual. Así mismo, se presenta la estructura final del software realizado y los resultados que se obtuvieron en forma de una comparación numérica y grafica entre el Laboratorio Virtual, MATLAB y SciLab. Finalmente, se presentan las conclusiones obtenidas y las recomendaciones que se consideraron de prioridad para mejorar el Laboratorio Virtual a futuro.

CAPÍTULO I

EL PROBLEMA

Planteamiento del problema

Desde hace muchos años que el hombre dedicó parte de sus esfuerzos a ofrecer servicios y producir bienes para el consumo de las personas, antaño, los procesos de producción eran fáciles de implementar y de complejidad reducida, por tanto, se podían controlar de forma manual utilizando instrumentos y herramientas simples, pero Creus (2010) afirma que: “[...] la gradual complejidad con qué éstos se han ido desarrollando ha exigido su automatización progresiva por medio de los instrumentos de medición y control”(p. 1).

Así mismo, la calidad de vida de las personas ha mejorado gracias a que ahora la producción de bienes y servicios se realizan de forma más eficiente. Parte de este incremento de eficiencia se debe a la incorporación de nuevas tecnologías que traen consigo ventajas como rapidez, precisión y mejoras en la automatización. La automatización por medio de controladores analógicos y electrónicos ha desempeñado un papel importante en esta mejora, tanto así, que se ha convertido en parte integral en los sistemas de vehículos espaciales, robóticos, procesos modernos de fabricación y en cualquier operación industrial (Ogata, 2003).

Siguiendo este orden de ideas, es necesario mencionar que analizar los sistemas de control puede llegar a ser, en ocasiones, una tarea difícil de realizar si no se tienen los conocimientos necesarios o si no se utilizan las herramientas adecuadas, uno de los motivos es que “[...] los antecedentes matemáticos requeridos incluyen temas tales como la teoría de la variable compleja, ecuaciones diferenciales y en diferencias, transformada de Laplace y transformada z [...]”(Kuo, 1996, p. 21).

Considerando lo anterior, es natural pensar que un modo de abarcar el análisis, diseño y simulación de sistemas de control es por medio de las computadoras. Ogata (2003) sugiere que gran parte del tiempo dedicado será verificando el comportamiento del sistema mediante un análisis, es por esto que recomienda utilizar un programa de computadora como MATLAB para que realice gran parte del cálculo matemático necesario en los estudios de sistemas de control, no obstante, se debe aclarar que MATLAB es un software que, aunque potente, permanece cerrado y de pago.

En adición a lo anterior, se puede pensar en utilizar herramientas libres como Octave y Scilab, que proporcionan un número elevado de funciones matemáticas, pero se requiere de saber programar, además, no ofrecen un entorno gráfico para la entonación de controladores, lo cual puede llegar a ser problemático para algunos ingenieros, por otro lado, suelen ofrecer soluciones aisladas entre sí e integrarlas suele ser tedioso y problemático. Suárez (2014) afirma que: “Si lograr el dominio de la herramienta computacional es un reto en sí mismo deja de ser una herramienta práctica [...]”(p. 6).

Finalmente, se debe tener en cuenta que las herramientas libres y gratuitas no tienen la posibilidad de diseñar controladores a base de lógica difusa de forma intuitiva y sencilla, lo cual puede conllevar un desperdicio de tiempo considerable en comparación con el uso de una interfaz gráfica para el diseño del controlador, es por esto que la mayoría de las herramientas gratuitas están limitadas a usarse, de manera práctica, solo en teoría clásica de control.

En base a la problemática expuesta, surgen las siguientes preguntas: ¿Es posible realizar un laboratorio para el análisis de sistemas de control con software libre?, ¿Cumpliría con los requisitos para analizar, diseñar y simular sistemas de control? y ¿Cómo se desempeñaría en comparación con otras herramientas?, preguntas que se responderán con el desarrollo de esta investigación y que se utilizaran para guiar el rumbo de la misma.

Objetivos de la investigación

Objetivo general

Desarrollar un laboratorio virtual de sistemas de control clásicos y difusos utilizando software libre.

Objetivos específicos

1. Estudiar los sistemas de control clásicos.
2. Estudiar el diseño de controladores difusos tipo Mamdani.
3. Codificar las rutinas de análisis, diseño y simulación de sistemas de control necesarias.
4. Realizar la interfaz gráfica de un laboratorio de sistemas de control virtual.
5. Comparar los resultados obtenidos con dos herramientas de corte similar.

Justificación e importancia

Actualmente hay una dependencia muy alta de MATLAB a la hora de trabajar con cálculo numérico, así mismo, es el software más usado en la UNET para analizar, diseñar y simular sistemas de control, con el desarrollo de un laboratorio de control utilizando software libre se puede eliminar parcialmente dicha dependencia, logrando así que herramientas externas sean usadas solo cuando se den casos más particulares o complejos.

Con el desarrollo del laboratorio de control se quiere tener una herramienta que cumpla con los requisitos actuales para el análisis, diseño y simulación de sistemas de control de forma libre, gratuita, rápida y sencilla sin tener que invertir demasiado tiempo en aprender la herramienta, y si en lo que importa, diseñar un sistema de control preciso y confiable. Por otro lado, se espera desarrollar la herramienta de forma que su interfaz gráfica acepte otros módulos, i.e., permitirá a aquel que lo desee expandir las funcionalidades del laboratorio de sistemas de control sin que se tenga que rehacer todo,

esto permitirá mantener actualizado y útil la herramienta, además, se estará generando una importancia metodológica al crear la posibilidad de realizar otras investigaciones alrededor de la misma para agregar nuevas funcionalidades y expandir el laboratorio virtual.

Alcance y limitaciones

Con esta investigación se realizará una interfaz gráfica que permita realizar cuatro funciones principales, la primera, análisis de procesos en el dominio del tiempo y en el dominio de la frecuencia como: respuesta al escalón, respuesta al impulso, bode, entre otras, la segunda, entonación de controladores PID de forma manual y automática, la tercera, será el diseño de controladores difusos tipo Mamdani generales y para esquemas específicos de sistemas de control, y finalmente, la simulación de sistemas de control utilizando controladores PID y controladores difusos para esquemas específicos.

Hay que aclarar que, para modelar los sistemas de control en cada una de las funcionalidades mencionadas se le dará la opción al usuario para representarlos como funciones de transferencia o ecuaciones de espacio de estado y se podrá especificar si el modelo está en tiempo continuo o en tiempo discreto. El análisis de sistemas de control es un tema muy amplio y requiere de una gran cantidad de funciones que se pueden realizar utilizando software libre, no obstante, se considera que con este alcance se estaría logrando cumplir con las necesidades fundamentales.

El software que se utilizará para realizar la interfaz gráfica y los cálculos correspondientes será el lenguaje de programación Python junto con su set de bibliotecas externas. Aunque la biblioteca de Python para análisis de sistemas de control es potente y brindan varias herramientas de forma directa, se debe dejar claro que no cumplen con todas las funciones requeridas, por tanto, algunas rutinas de simulación se deberán codificar de cero con ayuda de bibliotecas de cálculo numérico, y otras, harán uso de la biblioteca de control con código complementario.

CAPÍTULO II

MARCO TEÓRICO

Antecedentes

Para esta investigación se buscaron antecedentes de alcance nacional e internacional que ayuden a guiar, sustentar y validar esta investigación.

Casallas, Chacón y Rivera (2005), escribieron para la revista Acción Pedagógica de la Universidad de los Andes un artículo titulado “Desarrollo básico de un Laboratorio Virtual de Control de Procesos basado en Internet”, fue desarrollado conjuntamente entre la UNET y la ULA para ser usado a través de Internet y cuyo objetivo fue desarrollar un laboratorio virtual de control de procesos para la enseñanza a distancia, permitiendo ser utilizado solo por usuarios registrados y en determinados horarios. Aunque la finalidad de la investigación acá propuesta difiere del objetivo de Casallas et al., esta servirá como ayuda para determinar las funciones más utilizadas en el área de control, así como las necesidades típicas de un laboratorio de control virtual.

Adicionalmente, Salazar (2019), realizó en la Universidad Técnica del Norte, Ecuador, una tesis titulada “Diseño de un sistema de riego inteligente para cultivos de hortalizas basado en Fuzzy Logic en la granja la pradera de la Universidad Técnica del Norte.” para la implementación del controlador basado en la lógica difusa utilizó, entre otros equipos, un Rasberry PI para la ejecución de Python, específicamente, hizo uso de la biblioteca Scikit-Fuzzy, la cual le permitió definir las funciones de membresía, establecer las reglas difusas, realizar los procesos de fuzzificación y defuzzificación, todo lo mencionado sera para una arquitectura de controlador difuso tipo Mamdani. Esta tesis servirá como referencia para establecer el diseño de controladores difusos utilizando la biblioteca Scikit-Fuzzy.

Por otro lado, Congo (2018) realizó en la Universidad Tecnológica Israel, Ecuador, una tesis titulada “Aplicaciones del software libre Python para prácticas de laboratorio aplicado a la asignatura de tratamiento digital de señales de la Universidad Tecnológica Israel” cuyo objetivo fue desarrollar por medio del software Python y sus diferentes librerías científicas, la realización de 3 Prácticas de Laboratorio para la asignatura Procesamiento Digital de Señales de la Universidad Tecnológica Israel, al igual que la intención de este trabajo, se realizó una interfaz gráfica con el objetivo de facilitar su uso, en este caso como herramienta para el procesamiento digital de señales. Este trabajo sustenta la viabilidad de realizar una interfaz gráfica con enfoque similar en el campo de la electrónica y se utilizará como guía parcial para establecer su uso en sistemas de control.

Finalmente, Gómez (2009) redactó para la revista Educación en Ingeniería de Colombia, un artículo titulado “Toolbox didáctico para el diseño y análisis de sistemas de control lineal”, cuyo objetivo es describir un toolbox realizado en MATLAB para el análisis de sistemas de control lineales, este toolbox consiste de una interfaz gráfica que permite realizar pruebas a sistemas de control como respuesta escalón, respuesta en frecuencia, análisis de estabilidad, diseño de controladores, entre otras funciones. Este artículo será de utilidad para establecer la interfaz gráfica que se pretende realizar, además, sirvió como punto de partida para determinar las funciones que debería tener un laboratorio virtual de sistemas de control, como punto adicional se puede resaltar que reafirma la utilidad de esta investigación.

Bases teóricas

Para poder realizar el laboratorio virtual de sistemas de control clásicos y difusos será necesario abarcar conocimientos de análisis de sistemas de control, diseño de controladores PID y controladores difusos, a continuación, se presentan los conceptos necesarios para el desarrollo de esta investigación.

Procesos

Es importante partir de la base, es por ellos que empezaremos con los procesos, Sánchez (2003) define los procesos como “[...] un bloque que se identifica porque tiene una o más variables de salida de las cuales es importante conocer y mantener sus valores”(p. 153). Así, un proceso se caracteriza por tener una entrada y una salida (para procesos SISO), dicha salida deberá mantenerse alrededor de un punto de referencia dado, es decir, se debe controlar su salida.

Modelado de procesos en tiempo continuo

Un proceso puede ser representado por ecuaciones diferenciales que determinen su comportamiento dinámico en el dominio del tiempo, no obstante, trabajar con ecuaciones diferenciales es tedioso a nivel matemático, por tanto, se prefiere modelar los procesos utilizando la transformada de Laplace para llevar de una representación dinámica a una algebraica. Esta ecuación algebraica se operará para llevar a una función de transferencia que represente al proceso en el dominio de la frecuencia compleja (Smith y Corripio, 1985).

Transformada de Laplace

La transformada de Laplace de una función dependiente del tiempo $f(t)$ viene dada por la ecuación:

$$F(s) = \mathcal{L}[f(t)] = \int_0^{\infty} f(t)e^{-st}dt \quad (1)$$

si suponemos que $f(t)$ es un proceso cuya representación dinámica viene dada por una ecuación diferencial de la forma:

$$a_2 \frac{d^2y(t)}{dt^2} + a_1 \frac{dy(t)}{dt} + a_0 y(t) = b_0 x(t) \quad (2)$$

Aplicando (1) a (2) y despejando la relación $Y(s)/X(s)$ para el proceso con condiciones iniciales igual a cero (debido a que el proceso debe ser lineal) se obtiene la correspondiente función de transferencia del proceso $H(S)$:

$$H(s) = \frac{Y(s)}{X(s)} = \frac{b_0}{a_2 s^2 + a_1 s + a_0} \quad (3)$$

La demostración completa (Smith y Corripio, 1985, pp. 21-22) no es de interés para este trabajo, pero si su resultado, la función de transferencia puede ser analizada para determinar las características del proceso, como su ganancia, constante de tiempo, tiempo muerto, entre otras.

Ecuaciones de espacio de estado

Las ecuaciones de espacio de estado son un método más moderno para modelar todo tipo de sistemas, no solo físicos, sino también biológicos, económicos, sociales y otros. Así como una ecuación diferencial puede ser representada como una función de transferencia también se puede representar como una ecuación de espacio de estados, por tanto, una función de transferencia también puede ser representada en el espacio de estados y viceversa. Las ecuaciones linealizadas alrededor de un estado de operación (Ogata, 2003, p. 31) son:

$$\dot{x}(t) = Ax(t) + Bu(t) \quad (4)$$

$$y(t) = Cx(t) + Du(t) \quad (5)$$

con:

A: como matriz de estado.

B: como matriz de entrada.

C: como matriz de salida.

D: como matriz de transmisión directa.

Normalmente la notación presentada en las ecuaciones (4) y (5) es preferida dado que son compactas y fáciles de entender, no obstante, es al expandir las ecuaciones

en su forma matricial que podemos ver claramente que la ecuación (4) es un sistema de ecuaciones diferenciales ordinarias:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nm} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix} \quad (6)$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_r \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{r1} & c_{r2} & \cdots & c_{rn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} d_{11} & d_{12} & \cdots & d_{1m} \\ d_{21} & d_{22} & \cdots & d_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ d_{r1} & d_{r2} & \cdots & d_{rm} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix} \quad (7)$$

con:

n : como el número de variables de estado

m : como el número de entradas

r : como el número de salidas

Para resolver sistemas de ecuaciones diferenciales podemos utilizar métodos iterativos como los de Runge-Kuta.

Modelado de procesos en tiempo discreto

Así como para tiempo continuo se utilizan ecuaciones diferenciales, en el tiempo discreto se hace uso de ecuaciones en diferencias. Las ecuaciones en diferencias se pueden utilizar para aproximar a las ecuaciones diferenciales, estas primeras se suelen utilizar porque son más fáciles de programar (Kuo, 1996), por lo cual los procesos pueden ser representados de la siguiente forma:

$$y[k+n] + a_{n-1}y[k+n-1] + \cdots + a_1y[k+1] + a_0y[k] = f[k] \quad (8)$$

Así mismo la ecuación (4) está compuesta de ecuaciones diferenciales, por lo cual, puede ser llevada a una ecuación en diferencias, por tanto, la representación del proceso en el espacio de estados queda como sigue:

$$x[k + 1] = A_d x[k] + B_d u[k] \quad (9)$$

$$y[k] = C_d x[k] + D_d u[k] \quad (10)$$

Transformada z

En tiempo discreto se puede modelar un proceso utilizando la transformada z sobre la ecuación en diferencias del proceso para obtener su representación en el dominio Z, la transformada Z es a las ecuaciones en diferencia lo que la transformada de Laplace es las ecuaciones en tiempo continuo, la ecuación general para la transformada z es:

$$F(z) = \sum_{k=0}^{\infty} f[k] z^{-k} \quad (11)$$

Discretización de una función de transferencia continua

Existen varios métodos para realizar la discretización de una función de transferencia continua, el método más simple suele implicar el uso de un muestreador con un retenedor de orden cero (ZOH). Retenedor de orden cero significa que la señal de entrada es retenida constantemente durante el intervalo de muestreo (Haugen, 2005). La fórmula para realizar la discretización con retenedor de orden cero es la siguiente:

$$H(z) = (1 - z^{-1}) \mathcal{Z} \left[\mathcal{L}^{-1} \left\{ \frac{G(s)}{s} \right\} \Big|_{t=kh} \right] \quad (12)$$

Existen otros modos de llevar una función de transferencia continua al dominio Z como las aproximaciones de Tustin y Euler, estas consisten en sustituir la variable s por una aproximación en el dominio Z y se sustentan en ser aproximaciones numéricas a integrales en el tiempo, Haugen (2005) afirma que la aproximación de Tustin es la más

precisa de las tres, no obstante, sugiere que no existe una diferencia notable entre este y Euler hacia atrás, además, la mayoría de los controladores comerciales vienen con Euler hacia atrás, por lo que puede considerarse una mejor opción a la hora de escoger un método. Las sustituciones correspondientes son:

$$\text{Euler hacia adelante:} \quad s \leftarrow \frac{z - 1}{h} \quad (13)$$

$$\text{Euler hacia atrás:} \quad s \leftarrow \frac{z - 1}{hz} \quad (14)$$

$$\text{Tustin:} \quad s \leftarrow \frac{2z - 1}{hz + 1} \quad (15)$$

Existen más métodos de discretización como el de la invarianza de la respuesta impulso, el cual consiste en hacer coincidir la respuesta impulso en tiempo continuo con la respuesta impulso en tiempo discreto, útil cuando la respuesta impulso es importante (Fernández-Cantí, 2013). También existe la transformación directa de los polos y ceros del sistema, con este método la función de transferencia continua es factorizada para obtener los polos y ceros que serán mapeados al plano discreto (Hori, Cormier y Kanai, 1992). Las ecuaciones para representar ambos métodos son:

$$\text{Transformación del impulso invariante:} \quad H(z) = \sum_{i=1}^n \mathcal{Z} \left[\mathcal{L}^{-1} \left\{ G^i(s) \right\} \Big|_{t=kh} \right] \quad (16)$$

$$\text{Transformación directa:} \quad H(z) = k \frac{(z - 1)^e \prod_m^{m} (z - z_d)}{\prod_n^n (z - p_d)} \quad (17)$$

Respuesta en el tiempo del modelo en el espacio de estados

Para poder obtener la respuesta del sistema en el tiempo podemos utilizar varios métodos de resolución de ODE's o ecuaciones diferenciales ordinarias como la solución exacta, separación de variables o integración numérica aproximada, este último es el mejor para realizar por medio de computadoras. Para realizar la integración numérica y obtener la respuesta del sistema se utilizarán los métodos de Runge-Kutta.

Los métodos de Runge-Kutta son unos métodos iterativos de resolución de ODE's, los mismos se basan en utilizar "indirectamente el algoritmo de Taylor. En general, estos métodos evalúan $f(x, y)$ en más de un punto en la proximidad (x_n, y_n) en lugar de evaluar derivadas de $f(x, y)$ " (Martinez, 1997, p.31). La formulación general de los métodos de Runge-Kutta explícitos es:

$$\begin{aligned}
 k_1 &= h f(t^{(j)}, x^{(j)}) \\
 k_2 &= h f(t^{(j)} + c_2 h, x^{(j)} + a_{21} k_1) \\
 k_3 &= h f(t^{(j)} + c_3 h, x^{(j)} + a_{31} k_1 + a_{32} k_2) \\
 &\vdots \\
 k_s &= h f(t^{(j)} + c_s h, x^{(j)} + a_{s1} k_1 + \cdots + a_{s,s-1} k_{s-1}) \\
 x^{(j+1)} &= x^{(j)} + (b_1 k_1 + \cdots + b_s k_s)
 \end{aligned} \tag{18}$$

Donde s denota el número de escenarios a utilizar, es común expresar (18) utilizando una tabla de Butcher, en honor a John C. Butcher y su artículo de 1964, por tanto, (18) puede representarse de la siguiente forma:

	0					
c_2		a_{21}				
c_3		a_{31}	a_{32}			
\vdots		\vdots	\vdots	\ddots		
c_s		a_{s1}	a_{s2}	\cdots	$a_{s,s-1}$	
		b_1	b_2	\cdots	b_{s-1}	b_s

(19)

Adicionalmente existen los métodos de Runge-Kutta embebidos, los cuales consisten en calcular dos métodos de Runge-Kutta, uno de orden p y otro de orden \hat{p} , donde \hat{p} es normalmente $p - 1$ o $p + 1$ (Hairer, Nørsett y Wanner, 1993). La tabla de Butcher para representar los métodos embebidos es la siguiente:

0						
c_2		a_{21}				
c_3		a_{31}	a_{32}			
\vdots		\vdots	\vdots	\ddots		
c_s		a_{s1}	a_{s2}	\cdots	$a_{s,s-1}$	
<hr/>						
		b_1	b_2	\cdots	b_{s-1}	b_s
<hr/>						
		\hat{b}_1	\hat{b}_2	\cdots	\hat{b}_{s-1}	\hat{b}_s

$$x^{(j+1)} = x^{(j)} + (b_1 k_1 + \cdots + b_s k_s) \quad \text{Orden } p \quad (21)$$

$$\hat{x}^{(j+1)} = x^{(j)} + (\hat{b}_1 k_1 + \cdots + \hat{b}_s k_s) \quad \text{Orden } \hat{p} = p - 1 \text{ o } p + 1 \quad (22)$$

De modo que la integración para el siguiente paso se continua con $x^{(j+1)}$. El objetivo de calcular dos métodos en conjunto es para poder estimar el error y poder realizar un cambio en el tamaño de paso, de este modo, se puede incrementar el tamaño de paso si el error es muy bajo o disminuirlo si el error es inaceptable.

Runge-Kutta's para ecuaciones de espacio de estados

Partiendo de la ecuación (6) en donde podemos observar que $\dot{x}(t)$ viene dado por un conjunto de ecuaciones diferenciales ordinarias, por tanto, es posible aplicar los métodos de Runge-Kutta de forma casi directa, Martinez (1997) desarrolla las expresiones para un Runge-Kutta de orden 4 aplicado en el espacio de estados, estas expresiones pueden ser extendidas para generalizar su aplicación a un numero de escenarios arbitrarios. El vector de estado de forma general se puede expresar como:

$$\begin{aligned}
 x_1(t) &= f_1(t, x_1, x_2, \dots, x_n) = f_1(t, x) \\
 &\vdots \\
 x_n(t) &= f_n(t, x)
 \end{aligned} \quad (23)$$

y cuyas condiciones iniciales se pueden expresar como:

$$\begin{aligned} x_1^{(0)} &= \alpha_1 \\ &\vdots \\ x_n^{(0)} &= \alpha_n \end{aligned} \tag{24}$$

de modo que al sustituir (23) y (24) en (18) se obtenga el nuevo vector de estado:

$$\begin{aligned} k_1 &= h(Ax_n^{(j)} + Bu_m(t)) \\ k_2 &= h(A(x_n^{(j)} + a_{21}k_1) + Bu_m(t + c_2h)) \\ k_3 &= h(A(x_n^{(j)} + a_{31}k_1 + a_{32}k_2) + Bu_m(t + c_3h)) \\ &\vdots \\ k_s &= h(A(x_n^{(j)} + a_{s1}k_1 + \dots + a_{s,s-1}k_{s-1}) + Bu_m(t + c_sh)) \\ x_n^{(j+1)} &= x_n^{(j)} + (b_1k_1 + \dots + b_sk_s) \end{aligned} \tag{25}$$

con:

n : como el número de variables de estado

m : como el número de entradas

j : como la iteración actual

s : como el número de escenarios

finalmente, el vector de estado obtenido con (25) será utilizado en la siguiente iteración para continuar el proceso. Para obtener la salida actual se debe sustituir $x_n^{(j)}$ en (5).

$$y^{(j)} = Cx_n^{(j)} + Du_m(t) \tag{26}$$

Los pasos mencionados se repiten una y otra vez de forma iterativa utilizando un tamaño de paso adecuado, el tamaño de paso determinara el número de iteraciones a realizar. Un tamaño de paso pequeño genera resultados más precisos, pero incrementa el tiempo de computo considerablemente, por otro lado, un tamaño de paso grande puede

generar errores inaceptables con la ventaja de acelerar los cálculos, por tanto, el tamaño de paso es un factor muy importante en los métodos de Runge-Kutta.

Tamaño de paso variable

Como ya se mencionó antes, se puede realizar una adaptación del tamaño de paso en función del error y es posible no solo para los métodos embebidos, sino también para los explícitos, Rogan y Muñoz (s.f.) afirma que:

Los programas adaptativos continuamente monitorean la solución y modifican el paso de tiempo para asegurar que se mantenga la precisión especificada por el usuario. Esos programas pueden hacer algunos cálculos extras para optimizar la elección de τ , en muchos casos este trabajo extra vale la pena. (p.251)

Cálculo del error con dos medios pasos. Este es un método simple pero eficaz de calcular el error, consiste en realizar un paso simple de tamaño h y comparar su respuesta con la respuesta dada por el mismo método al realizar dos pasos de tamaño $\frac{h}{2}$, de este modo el error Δe se puede calcular como la diferencia en valor absoluto de ambas respuestas, $|x_{n,h}^{(j+1)} - x_{n,\frac{h}{2}}^{(j+1)}|$ donde $x_{n,\frac{h}{2}}^{(j+1)}$ es la salida luego de dos medios pasos.

Cálculo del error para métodos embebidos. Los métodos embebidos están hechos para poder realizar este cálculo del error, por tanto, no hay que realizar ningún cálculo extra y el error puede ser calculado de forma directa como:

Nuevo tamaño de paso. Combinando las formulas y algoritmos descritos por Rogan y Muñoz (s.f.), Hairer et al. (1993) y Ritschel (2013) se pueden obtener las fórmulas para el manejo del error a fin de calcular un nuevo tamaño de paso. Lo primero es calcular una escala, este es el único paso que se diferencia entre el método de dos medios pasos y el de los métodos embebidos. Para dos medios pasos la escala es:

$$escala = atol + rtol \cdot \frac{|x_{n,h}^{(j+1)}| + |x_{n,\frac{h}{2}}^{(j+1)}|}{2} \quad (27)$$

para los métodos embebidos:

$$escala = atol + rtol \cdot max \left(\left| x_n^{(j+1)} \right|, \left| x_n^{(j)} \right| \right) \quad (28)$$

A partir de acá las formulas son comunes para ambos métodos. Para continuar, se debe realizar el cálculo de un error normalizado, por lo cual se calcula la norma RMS del error dividido por la escala calculada:

$$\|\Delta e\| = \sqrt{\frac{1}{n} \sum_{i=1}^n \left(\frac{\Delta e_i}{escala_i} \right)^2} \quad (29)$$

donde n denota el número de elementos que componen la respuesta. Finalmente, el nuevo tamaño de paso viene dado por:

$$\begin{cases} h_{t+1} = h_t \cdot maxStepIncrease & Si \quad \|\Delta e\| = 0 \\ h_{t+1} = h_t \cdot sf_1 \cdot \|\Delta e\|^{\frac{-1}{q+1}} & Si \quad \|\Delta e\| \leq 1 \\ h_t = h_t \cdot sf_1 \cdot \|\Delta e\|^{\frac{-1}{q+1}} & Si \quad \|\Delta e\| > 1, \text{ Se descarta la salida} \end{cases} \quad (30)$$

con sf_1 como un factor de seguridad que evita que el nuevo tamaño de paso para un $\|\Delta e\|$ dado se repita indefinidamente, y q como el orden del método utilizado, en el caso de los Runge-Kutta embebidos, q viene dado por el método de menor orden entre x_n y \hat{x}_n , nótese que, si el error se encuentra por arriba de uno, la salida es descartada y se repite el cálculo con un nuevo tamaño de paso. El proceso se repite hasta que el error sea igual o menor que uno.

Respuesta en el tiempo del modelo en el espacio de estados discreto

El cálculo de la respuesta en el tiempo para el modelo en el espacio de estados discreto es mucho más simple, no se requiere ningún método de integración aproximado, lo cual se debe a que el cálculo puede ser realizado de forma directa ya que la fórmula es

el algoritmo a seguir (Haugen, 2005). Tomando en consideración la ecuación (9), basta con sustituir el vector de estado para obtener a $x[k + 1]$ y la salida $y[k]$

Sistemas de control

Sistema en lazo abierto

Un sistema en lazo abierto es aquel cuya salida no es medida en ningún momento en orden de ajustar la entrada, por lo cual provoca que “las perturbaciones que se presentan en el proceso ocasionen que sus efectos se sientan en la salida del proceso, es decir, en el valor de la variable controlada”(Maloney, 2006, p. 350). La mayoría de los sistemas en lazo abierto son estables, pero carecen de utilidad práctica al no poder mantenerlos en un punto de referencia debido a las perturbaciones.

Por otro lado, es importante analizar el comportamiento del sistema en lazo abierto para determinar su comportamiento en lazo cerrado, además, nos permite determinar sus características con el fin de realizar un modelo matemático aproximado del sistema, el cual es útil para realizar la entonación de los controladores sin tener que trabajar de forma directa con el proceso, el cual puede ser peligroso o inaccesible. La representación en diagrama de bloques de un sistema en lazo abierto se puede observar en la Figura 1.



Figura 1. Sistema en lazo abierto. Fuente: Elaboración propia.

Sistema en lazo cerrado

Los sistemas de lazo cerrado son aquellos cuya salida es realimentada a la entrada del sistema, esta realimentación suele ser negativa para que sea estable, no obstante, existen sistemas que se vuelven inestables con el simple hecho de cerrar el lazo, un ejemplo de sistema realimentado se puede observar en la Figura 2.



Figura 2. Sistema en lazo cerrado. Se observan las principales señales de un sistema en lazo cerrado.
Fuente: Elaboración propia.

En la Figura 2 se denotan algunas de las señales que componen a un sistema de control empezando por el valor de referencia o setpoint (sp), la variable medida o variable del proceso (Vp) la cual corresponde a la variable que se desea controlar, las perturbaciones, las cuales son magnitudes físicas que pueden afectar al proceso como la temperatura ambiente, presión, vibraciones, entre otras, y finalmente, la señal de error ($e(t)$) que corresponde a la diferencia entre el valor de referencia y la variable medida (Maloney, 2006).

$$e(t) = sp - Vp \quad (31)$$

Lugar de las raíces

El lugar de las raíces es un método sencillo de diseño que se basa en conocer y manipular la ubicación de los polos del sistema en función de un parámetro de ganancia. Un ajuste en la ganancia puede provocar que los polos cambien de posición en el plano complejo y se ubiquen en una posición deseada o indeseada. Ogata (2003) considera que: “Al diseñar un sistema de control lineal, encontramos que el método del lugar de las raíces resulta muy útil, debido a que indica la forma en la que deben modificarse los polos y ceros en lazo abierto [...]” (p.270). En la Figura 3 se observa un ejemplo de la gráfica del lugar de las raíces.

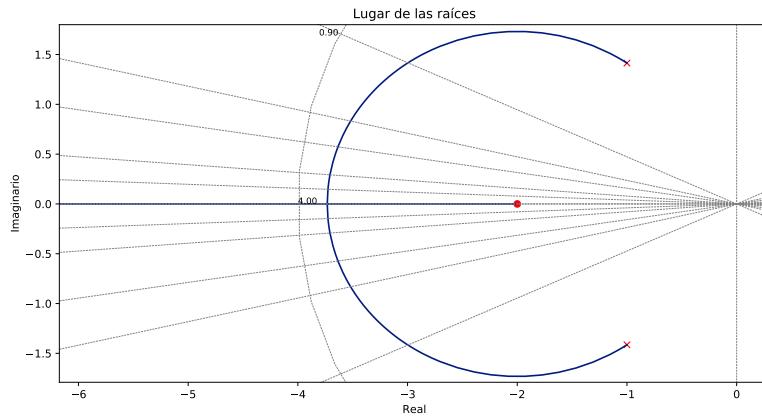


Figura 3. Gráfica del lugar de las raíces. Se puede observar la gráfica del lugar de las raíces para un sistema $G_1(s) = \frac{s+2}{s^2+2s+3}$, el cual tiene ubicado sus polos en el semiplano izquierdo del plano complejo. Fuente: Elaboración propia.

Estabilidad de los sistemas

La estabilidad se ve afectada por la estructura del sistema, por tanto, se debe tomar en cuenta cuando se cierra el lazo del sistema. Ogata (2003) afirma que, desde el punto de vista de estabilidad, el sistema de control en lazo abierto no es un problema importante. Por otra parte, la estabilidad es un gran problema en el sistema de control en lazo cerrado debido a la posibilidad de que se generen oscilaciones. En consecuencia, los análisis de estabilidad se realizan para sistemas en lazo cerrado, el motivo principal es porque los sistemas en lazo abierto tienden a ser estables, además, para realizar el control de un proceso se suele utilizar los sistemas en lazo cerrado.

Análisis de estabilidad en el plano complejo. La estabilidad de un sistema lineal viene dada por la ubicación de sus polos a lo largo del plano complejo s , i.e., se puede determinar si un sistema es estable si no posee ninguno polo en el semiplano derecho del plano complejo s incluyendo el eje coordenado $j\omega$ en tiempo continuo (Ogata, 2003), en el caso de sistemas discretos, los polos deben encontrarse dentro del círculo unitario, la estabilidad marginal es posible si existen uno o más polos justo en el círculo unitario, pero no múltiples, lo cual permite que el análisis de estabilidad se puede realizar de forma analítica y gráfica. En la Figura 4 se puede observar dos sistemas continuos en lazo abierto, $G_1(s)$ que es estable y $G_2(s)$ que es inestable.



Figura 4. Análisis de estabilidad en el plano complejo. Se puede observar como el sistema $G_1(s)$ tiene ubicado sus polos en el semiplano izquierdo del plano complejo, i.e., es estable, por otro lado, el sistema $G_2(s)$ es inestable al poseer polos ubicados en el semiplano derecho del plano complejo. Fuente: Elaboración propia.

Criterio de estabilidad de Nyquist. El criterio de estabilidad de Nyquist determina el estado de estabilidad utilizando los polos en lazo abierto y la respuesta en frecuencia del sistema en lazo abierto. Este criterio “es útil en la ingeniería de control, debido a que permite determinar gráficamente la estabilidad absoluta del sistema en lazo cerrado a partir de las curvas de respuesta en frecuencia en lazo abierto, sin que sea necesario determinar los polos en lazo cerrado”(Ogata, 2003, p. 446). Este criterio tiene la ventaja de poder realizarse tanto con cálculos analíticos como con datos experimentales, Ogata (2003) define tres criterios:

1. El punto $-1 + j0$ no está rodeado. Esto implica que el sistema es estable en lazo cerrado si no hay polos de $G(s)H(s)$ en el semiplano derecho del plano $s^{(1)}$; de lo contrario, el sistema es inestable.
2. El punto $-1 + j0$ queda rodeado una o varias veces en sentido contrario al de las agujas del reloj. En este caso, el sistema es estable en lazo cerrado si el número de rodeos en sentido contrario al de las agujas del reloj es igual al número de polos $G(s)H(s)$ en el semiplano derecho del plano $s^{(1)}$; de lo contrario, el sistema es inestable.

⁽¹⁾En el caso de sistemas discretos se toma en cuenta son los polos fuera del círculo unitario para la evaluación de los criterios.

3. El punto $-1 + j0$ queda rodeado una o varias veces en el sentido de las agujas del reloj. En este caso el sistema es inestable en lazo cerrado.

En la Figura 5 se puede observar los diagramas de Nyquist para los sistemas presentados en la Figura 4, el sistema $G_1(s)$ debe ser evaluado por el criterio número 1, al no poseer ninguno polo en el semiplano derecho el sistema será estable en lazo cerrado asumiendo un $H(s) = 1$, por otro lado, $G_2(s)$ debe ser evaluado por el criterio número 2, debido a que el número de rodeos al punto $-1 + j0$ es igual al número de polos en el semiplano derecho el sistema será estable en lazo cerrado asumiendo un $H(s) = 1$ a pesar de que en lazo abierto sea inestable.

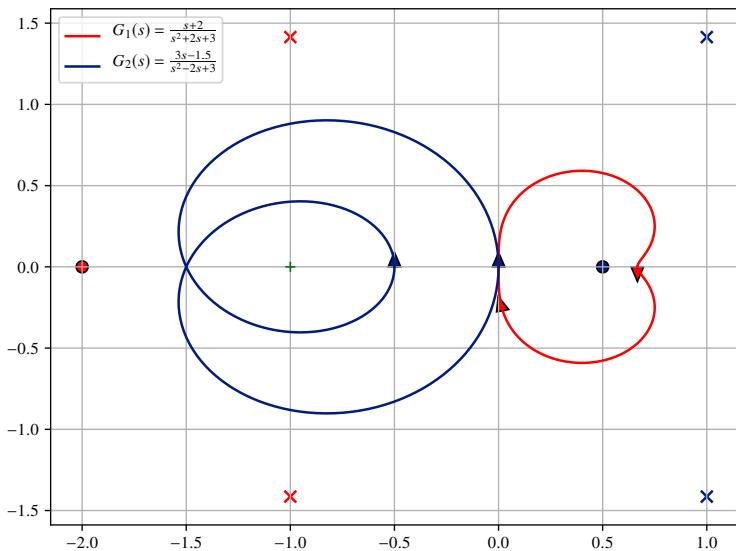


Figura 5. Análisis de estabilidad con diagrama de Nyquist. El sistema $G_1(s)$ al ser evaluado con el criterio número 1 se encuentra que es estable en lazo cerrado, a su vez, el sistema $G_2(s)$ al ser evaluado con el criterio número 2 se encuentra que también será estable en lazo cerrado, en ambos casos se asume un $H(s) = 1$. Fuente: Elaboración propia.

Margen de ganancia y Margen de fase. Los márgenes de ganancia y de fase son una medida de la estabilidad relativa, se toman en cuenta a la hora de realizar el diseño de un controlador debido a que los márgenes se pueden interpretar de modo que orienten en la cantidad de ganancia que se le puede aplicar al sistema en lazo cerrado, adicionalmente, la estabilidad relativa se puede interpretar como que tan estable

es un sistema. Dorf y Bishop (2011) definen el margen de ganancia como: “[...] el incremento en la ganancia del sistema cuando $fase = -180^\circ$ que resultaría en un sistema marginalmente estable con la intersección del punto $-1 + j0$ en el diagrama de Nyquist” (p.655). Así mismo, Dorf y Bishop (2011) definen el margen de fase como:

[...] la cantidad de desplazamiento de fase de $L(j\omega)$ a una unidad de magnitud que resultaría en un sistema marginalmente estable con la intersección del punto $-1 + j0$ en el diagrama de Nyquist. El margen de ganancia y de fase pueden ser fácilmente encontrados utilizando un diagrama de Bode. (p.656)

Análisis de estabilidad con las trazas de Bode. Los diagramas de bode son una forma de representar la respuesta en frecuencia de un sistema tomando en cuenta los cambios de amplitud de $L(j\omega)$ y del ángulo de fase de $L(j\omega)$ respecto a la frecuencia (Nilsson y Riedel, 2005). Para analizar la estabilidad se utilizan el margen de ganancia y el margen de fase del sistema en un diagrama de Bode a modo de trazas, el punto de estabilidad critica en el diagrama de bode pasa a ser su equivalente en dB , el cual es $0dB$.

Un margen de ganancia positivo y de fase positivos indican que el sistema es estable, si el sistema es de fase no mínima, la interpretación anterior deja de ser válida. Aplicando las definiciones dadas por Dorf y Bishop (2011) podemos obtener el margen de ganancia y de fase para los sistemas $G_1(s)$ y $G_2(s)$ y se presentan como ejemplo en la Figura 6 y Figura 7.

- $G_1(s)$: $GM \rightarrow \infty$ y $PM \rightarrow \infty$
- $G_2(s)$: $GM = -3.509dB$ a $1.409 rad/sec$ y $PM = -23.719^\circ$ a $0.807 rad/sec$

Diagrama de Nichols. El diagrama de Nichols es una gráfica del sistema en lazo abierto de tipo magnitud vs fase, en el diagrama se pueden ubicar los márgenes de ganancia y fase, para ayudar en el diseño de controladores se suele superponer una rejilla guía de valores de magnitud y ángulo constantes, esta rejilla lleva el nombre de carta de Nichols o ábaco de Nichols. En el diagrama de nichols el punto crítico $-1 + j0$ se convierte en $(0dB, -180^\circ)$



Figura 6. Márgenes de ganancia y fase de $G_1(s)$. En el diagrama se puede observar que no existe cruce por $0dB$ en magnitud o cruce por -180° en fase, por tanto, el sistema es estable y acepta incrementos teóricos de ganancia infinitos. Fuente: Elaboración propia.



Figura 7. Márgenes de ganancia y fase de $G_2(s)$. El sistema $G_2(s)$ es de fase no mínima, por tanto, la interpretación del margen de ganancia y de fase no es tan simple. En la figura se observar que el margen de ganancia es negativo y aun así, sabemos por su análisis con diagrama de Nyquist que es estable en lazo cerrado. Fuente: Elaboración propia.

Controlador PID

El controlador será el elemento dentro del sistema de control que se encargará de llevar al proceso a un valor de referencia deseado, existen varios esquemas de control para el control de procesos, cada uno con sus ventajas y desventajas, el más común es un sistema en lazo cerrado con controlador, este se puede observar en la Figura 8.

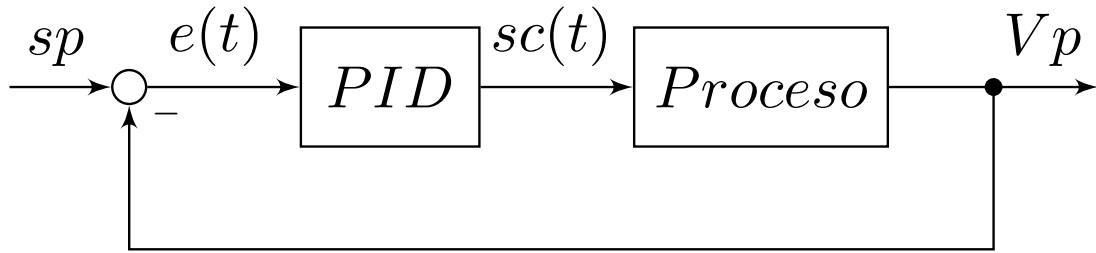


Figura 8. Sistema en lazo cerrado con controlador. El controlador recibe la señal de error y genera una señal de control $sc(t)$. Fuente: Elaboración propia.

En este esquema se observa que el controlador recibe la señal de error producto de tomar la diferencia entre el setpoint y la variable del proceso, este error es utilizado por el controlador para generar una señal de control $sc(t)$. El controlador más usado en la industria es el controlador PID, así lo afirma Kuo (1996): “[...] uno de los controladores más ampliamente empleados es el controlador PID [...] donde las letras son las iniciales de proporcional, integral y derivativo”(p. 671). La ecuación que define a un PID en el dominio del tiempo es la siguiente:

$$sc(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t) \quad (32)$$

y su forma en función de transferencia:

$$G_c(s) = \frac{sc(s)}{e(s)} = K_p + \frac{K_i}{s} + K_d s \quad (33)$$

El controlador PID también puede ser representado de forma discreta, la componente proporcional sigue siendo una constante, la componente integral puede ser

aproximada con varios métodos numéricos para el cálculo de área como la integración trapezoidal (Kuo, 1996). Por último, la derivada se puede aproximar como una diferencia finita regresiva:

$$sc[k] = K_p e[k] + \frac{K_i T}{2} \sum_{i=1}^k (e[i] + e[i-1]) + K_d \frac{e[k] - e[k-1]}{T} \quad (34)$$

En el dominio Z, la componente proporcional se mantiene como constante, la componente integral se lleva al dominio Z utilizando la transformación de Tustin dada por (15) y obtener la representación en el dominio Z de la integración trapezoidal, para el caso de la derivada se utiliza (14) para obtener la representación en el dominio Z de la diferencia finita regresiva.

$$G_c(z) = K_p + \frac{K_i T}{2} \left(\frac{z+1}{z-1} \right) + K_d \frac{z-1}{Tz} \quad (35)$$

Componente proporcional. La componente proporcional se obtiene de multiplicar la señal de error por la ganancia proporcional, lo cual provoca que la señal de control sea proporcional a la señal de error con una relación igual a la ganancia proporcional, si el controlador solo constase de componente proporcional el error en estado estable nunca se eliminaría, dicho de otro modo, el error podría ser muy bajo, pero jamás cero. “Este error se denomina la desviación permanente u ‘offset’. El mismo disminuye si se aumenta el valor de K”(Mata, 1999, p. 54).

Componente integral. La componente integral viene dada por la sumatoria de la señal del error en el tiempo multiplicada por la ganancia integral, por tanto, se obtiene una acumulación en un periodo determinado, lo anterior implica que mientras exista error, la componente integral seguirá aumentando o disminuyendo dependiendo del signo de la señal de error hasta que el error sea cero, la componente integral se suma con la componente proporcional formando una única señal de control.

Debido a que la componente integral depende del tiempo, es posible corregir desviaciones de la variable controlada generadas por perturbaciones externas. Por otro lado, debido a que eventualmente se conseguirá que el error sea igual a cero la componente proporcional no hará ningún aporte al sistema, y la señal de control pasará a ser totalmente generada por la componente integral hasta que exista un cambio en el setpoint o una alteración en la variable medida.

Componente derivativa. La componente derivativa viene dada por el ritmo de cambio de la señal de error multiplicado por la ganancia derivativa, por tanto, cuando el error permanece constante la componente derivativa es igual a cero, la componente derivativa será tan grande como la velocidad con la que cambie el error, lo cual ayuda a evitar que se generen sobre pasos en la variable medida respecto a la variable de referencia.

La componente derivativa es muy sensible al ruido, por tanto, se debe utilizar solo cuando se requiera un cierto grado de anticipación y no exista ruido (Smith y Corripio, 1985). Es recomendable utilizar filtros en orden de disminuir el posible ruido, a su vez, cuando el proceso a controlar es de respuesta rápida se recomienda no agregar componente derivativa o que su ganancia sea muy pequeña. Tomando en cuenta lo anterior, se debe mencionar que la componente derivativa tal como se observa en las ecuaciones (32) y (33) no es implementable al ser no causal, por tanto, la derivada debe ser implementada utilizando una aproximación.

La aproximación actúa como una derivada para las componentes de la señal de baja frecuencia. La ganancia, sin embargo, está limitada a KN . Esto significa que las medidas de ruido de alta frecuencia son amplificadas cuando mucho por un factor KN . (Åström, 2002, p.220)

La función de transferencia de la componente derivativa aproximada es la siguiente:

$$D = K_d \frac{Ns}{s + N} \quad (36)$$

Entonación por método de Ziegler-Nichols en lazo abierto

El método de Ziegler-Nichols en lazo abierto requiere primero aproximar el modelo del proceso como una función de transferencia de primer orden con tiempo muerto.

$$G(s) = \frac{K}{\tau s + 1} e^{-\alpha s} \quad (37)$$

Los parámetros correspondientes pueden ser obtenidos realizando una prueba al escalón y extrayendo los datos correspondientes de la gráfica de la respuesta, un ejemplo se puede observar en la Figura 9. Acá se observa: (a) t_0 que corresponde al inicio del escalón, (b) t_1 inicio de la respuesta por parte del proceso al escalón, (c) t_2 cuando el proceso alcanza el 63 % del cambio total, (d) Δy cambio total del proceso ante el escalón y (e) Δu magnitud de cambio de la entrada. Los parámetros para aproximar el proceso a un modelo de primer orden con tiempo muerto se calculan como sigue:

$$K = \frac{\Delta y}{\Delta u} \quad (38)$$

$$\tau = t_2 - t_1 \quad (39)$$

$$\alpha = t_1 - t_0 \quad (40)$$

Con los parámetros del modelo de primer orden se puede obtener la entonación de un controlador P, PI o PID. Las ganancias K_p , K_i y K_d se pueden obtener sustituyendo los mencionados parámetros en la Tabla 1 correspondiente a la regla de entonación de Ziegler-Nichols. Ogata (2003) comenta que: “Las reglas de sintonía de Ziegler-Nichols [...] se han usado ampliamente para sintonizar controladores PID en sistemas de control de procesos en los que no se conoce con precisión la dinámica de la planta”(p. 572).

Adicionalmente existen otras reglas para la entonación automática de los controladores PID, como el método de Cohen-Coon, al igual que Ziegler-Nichols, requiere de los parámetros del proceso en su aproximación como proceso de primer orden para ser sustituidos en la Tabla 2 y obtener las ganancias respectivas.

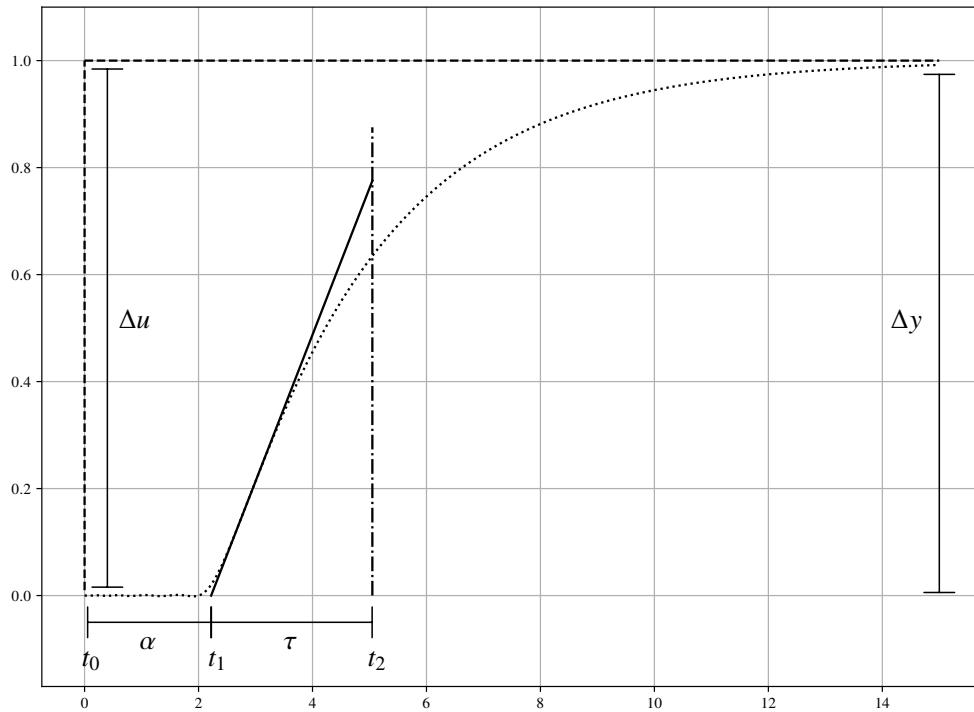


Figura 9. Modelado por curva de reacción. Para realizar este modelado se estudia la respuesta del sistema ante la entrada de la función de heaviside $u(t)$, también llamada función escalón unitario, el modelo obtenido a través de este método es una aproximación del sistema. Fuente: Elaboración propia.

Tabla 1.
Regla de entonación de Zigler-Nichols

Tipo de controlador	K_p	T_i	T_d
P	$\frac{\tau}{\alpha}$	∞	0
PI	$0.9 \frac{\tau}{\alpha}$	$\frac{\alpha}{0.3}$	0
PID	$1.2 \frac{\tau}{\alpha}$	2α	0.5α

Nota. Los valores de K_i y K_d se pueden obtener de la siguiente forma: $K_i = K_p/T_i$; $K_d = K_p T_d$. Fuente: Ogata (2003).

Tabla 2.
Regla de entonación de Cohen-Coon

Tipo de controlador	K_p	T_i	T_d
P	$\frac{1}{K} \left(\frac{\tau}{\alpha} \right) \left[1 + \frac{1}{3} \left(\frac{\alpha}{\tau} \right) \right]$	∞	0
PI	$\frac{1}{K} \left(\frac{\tau}{\alpha} \right) \left[0.9 + \frac{1}{12} \left(\frac{\alpha}{\tau} \right) \right]$	$\alpha \left[\frac{30 + 3 \left(\frac{\alpha}{\tau} \right)}{9 + 20 \left(\frac{\alpha}{\tau} \right)} \right]$	0
PD	$\frac{1}{K} \left(\frac{\tau}{\alpha} \right) \left[\frac{5}{4} + \frac{1}{6} \left(\frac{\alpha}{\tau} \right) \right]$	∞	$\alpha \left[\frac{6 - 2 \left(\frac{\alpha}{\tau} \right)}{22 + 3 \left(\frac{\alpha}{\tau} \right)} \right]$
PID	$\frac{1}{K} \left(\frac{\tau}{\alpha} \right) \left[\frac{4}{3} + \frac{1}{4} \left(\frac{\alpha}{\tau} \right) \right]$	$\alpha \left[\frac{33 + 6 \left(\frac{\alpha}{\tau} \right)}{13 + 8 \left(\frac{\alpha}{\tau} \right)} \right]$	$\alpha \left[\frac{4}{11 + 2 \left(\frac{\alpha}{\tau} \right)} \right]$

Nota. Los valores de K_i y K_d se pueden obtener de la siguiente forma: $K_i = K_p/T_i$; $K_d = K_p T_d$. Fuente: APCO-Inc (s.f.).

Lógica difusa

La lógica difusa es una extensión matemática de la lógica convencional en donde se asigna un grado de pertenencia a un hecho entre un valor verdadero y uno falso, “consiste en que los valores verdaderos (en lógica difusa) o valores de pertenencia (en conjuntos difusos) se indican en un número entre [0.0, 1.0], donde 0.0 representa falsedad total y 1.0 significa verdad absoluta”(Ponce, 2010, p. 4).

La lógica difusa se puede estructurar en tres etapas, la primera etapa consiste en establecer cada una de las variables con etiquetas lingüística que determinen en conjunto el universo de discurso de cada una de ellas, es decir, se crean los conjuntos difusos, en la segunda etapa se definen las reglas de inferencia difusa que determinaran el comportamiento del sistema difuso, y tercera, se obtienen los valores de salida utilizando

las reglas de inferencia y realizando el proceso de defuzzificación que consiste en llevar los grados de pertenencia a un valor nítido o real (Ponce, 2010).

Conjuntos difusos

Un conjunto difuso está compuesto de funciones de membresía, dichas funciones de membresía representan valores para asignar el grado de pertenencia de una variable a un estado en particular y son identificados con etiquetas lingüísticas descriptivas, como alto, caliente, bajo, negativo, entre otras. En la Figura 10 se puede observar un ejemplo de conjunto difuso.

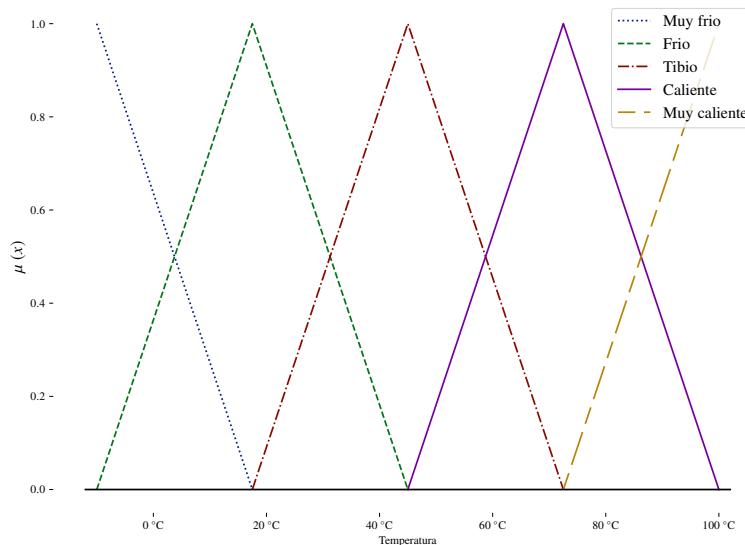


Figura 10. Ejemplo de un conjunto difuso. Este conjunto difuso corresponde a una variable de temperatura que va desde -10°C hasta 100°C. Fuente: Elaboración propia.

Las funciones de membresía pueden tomar varias formas y rangos en orden de poder representar de mejor forma los grados de pertenencia de la variable a una etiqueta en particular, sin embargo, las formas triangulares suelen ser las menos pesadas en memoria y en poder de cómputo (Riid, 2002). La cantidad de etiquetas que se pueden asignar a un conjunto difuso es ilimitada, pero se debe tener en cuenta el nivel de complejidad que se genera a medida que se incrementa su número. A continuación, se muestran algunas formas que pueden tomar las funciones de membresía:

Función de membresía: triangular o trimf.

$$\mu_n(x) = \begin{cases} 0 & Si \quad x \leq a \\ \frac{x-a}{b-a} & Si \quad a \leq x \leq b \\ \frac{c-x}{c-b} & Si \quad b \leq x \leq c \\ 0 & Si \quad x \geq c \end{cases} \quad \text{trimf} \quad (41)$$

Función de membresía: trapezoidal o trapmf.

$$\mu_n(x) = \begin{cases} 0 & Si \quad x \leq a \\ \frac{x-a}{b-a} & Si \quad a \leq x \leq b \\ 1 & Si \quad b \leq x \leq c \\ \frac{d-x}{d-c} & Si \quad c \leq x \leq d \\ 0 & Si \quad x \geq d \end{cases} \quad \text{trapmf} \quad (42)$$

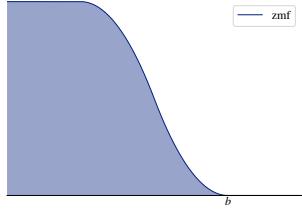
Función de membresía: gaussiana o gaussmf.

$$\mu_n(x) = e^{-\frac{(x-m)^2}{2\sigma^2}} \quad \text{gaussmf} \quad (43)$$

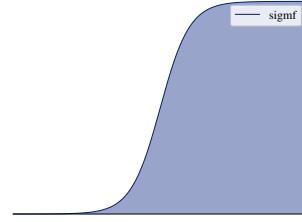
Función de membresía: S o smf.

$$\mu_n(x) = \begin{cases} 0 & Si \quad x \leq a \\ 2 \left(\frac{x-a}{b-a} \right)^2 & Si \quad a \leq x \leq \frac{a+b}{2} \\ 1 - 2 \left(\frac{x-b}{b-a} \right)^2 & Si \quad \frac{a+b}{2} \leq x \leq b \\ 1 & Si \quad x \geq b \end{cases} \quad \text{smf} \quad (44)$$

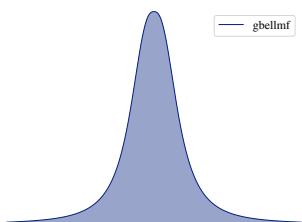
Función de membresía: Z o zmf.

$$\mu_n(x) = \begin{cases} 1 & \text{Si } x \leq a \\ 1 - 2 \left(\frac{x-a}{b-a} \right)^2 & \text{Si } a \leq x \leq \frac{a+b}{2} \\ 2 \left(\frac{x-b}{b-a} \right)^2 & \text{Si } \frac{a+b}{2} \leq x \leq b \\ 0 & \text{Si } x \geq b \end{cases}$$

(45)

Función de membresía: sigmoidal o sigmf.

$$\mu_n(x) = \frac{1}{1 + e^{-c(x-b)}}$$

(46)

Función de membresía: campana generalizada o gbellmf.

$$\mu_n(x) = \frac{1}{1 + \left| \frac{x-c}{a} \right|^{2b}}$$

(47)

Existen otras funciones de membresía que utilizan como base las anteriores mostradas, estas son:

- gauss2mf: Combinación de dos funciones gaussianas.
- dsigmf: Valor absoluto de la diferencia de dos funciones sigmoidales.
- psigmf: Producto de dos funciones sigmoidales.
- pimf: Producto de una smf y una zmf.

Reglas de inferencia

Las reglas de inferencia, al igual que en la lógica convencional, se encargan de realizar una conexión entre los antecedentes y los consecuentes con el fin de obtener una salida, en este caso, entre los conjuntos difusos de entrada y la salida, la cual puede ser un conjunto difuso o una función matemática. Como sugiere Ponce (2010), las reglas pueden formularse de dos formas denominadas modus ponens y modus tollens. Modus ponens utiliza los antecedentes para obtener el consecuente, por otro lado, el modus tollens utiliza los consecuentes recíprocos para obtener el antecedente.

Para realizar el proceso de inferencia se necesitan las normas triangulares (T-Normas) para el caso de lógica AND o las conormas triangulares (T-Conormas) para lógica OR. Sus formas de aplicación más comunes suelen ser el mínimo de los valores de pertenencia para las T-Normas y el máximo de los valores de pertenencia para las T-Conormas, de este modo, se obtienen las operaciones de intersección y unión respectivamente. El proceso de inferencia consta de 6 pasos: fuzzificacion, pareo de proposiciones, conjunción/disyunción de premisas, implicación, agregación y defuzzificación.

Fuzzificacion

El proceso de fuzzificacion es el primer paso para obtener la salida de un controlador difuso. La entrada nítida es transformada en una representación difusa, esto se debe a que las operaciones subsiguientes se realizan entre conjuntos difusos. La forma más común de fuzzificacion para la entrada es el uso de un singleton.

Pareo de proposiciones

Riid (2002) sugiere que en orden de evaluar la proposición de premisas en términos numéricos los valores de membresía μ_{ir} respecto a x deben ser hallados para poder realizar el pareo de proposiciones. El pareo de proposiciones es la relación que existe entre la entrada y una etiqueta lingüística. Riid (2002) afirma que “[...] el singleton

difuso puede considerarse solo una representación diferente de un número nítido, por lo tanto, con fuzzificación singleton, el procedimiento de fuzzificación es incrustado en el pareo de proposiciones” (p.16). Utilizando fuzzificacion singleton el pareo de proposiciones puede representarse como:

$$\tau_{ir} = \mu_{ir}(x_i) \quad (48)$$

donde i es la i-esima entrada al controlador difuso y r es la r-esima regla.

Operación de premisas

Debido a que pueden existir N entradas, también pueden existir N premisas en una regla, las cuales, pueden relacionarse entre sí por medio de operaciones AND en cuyo caso se denomina conjunción de premisas. Por otro lado, también es posible relacionar las premisas utilizando las operaciones OR y se denomina disyunción de premisas. Para realizar estas operaciones se pueden utilizar las T-Normas y las T-Conormas en el caso conjunción y disyunción respectivamente.

$$\text{Conjuncion de premisas: } \tau_r = \bigcap_{i=1}^N \tau_{ir} \quad (49)$$

$$\text{Disyuncion de premisas: } \tau_r = \bigcup_{i=1}^N \tau_{ir} \quad (50)$$

Implicación

La implicación se encarga de relacionar las premisas con los consecuentes. Normalmente se prefiere utilizar conjunción por medio de las T-Normas para realizar la implicación, la cual viene descrita como sigue:

$$F_r(y) = W_r \tau_r \cap \gamma_{jr} \quad (51)$$

donde γ_{jr} es la j-esima Función de membresía de salida asociada con la r-esima regla y W_r es el peso asociado a la r-esima regla (Riid, 2002).

Agregación

El proceso de agregación se realiza luego de haber pasado por la fuzzificación, pareo de proposiciones, conjunción/disyunción de premisas y la implicación. Todos los resultados obtenidos de cada una de las reglas son unificados aplicando un operador OR por medio de una T-Conorma, de este modo, se obtiene un conjunto difuso por cada salida del controlador. Todo el proceso hasta este punto puede ser descrito por las siguientes formulas:

Con conjuncion de premisas:
$$F(y_j) = \bigcup_{r=1}^R \left(W_r \left(\bigcap_{i=1}^N \mu_{ir}(x_i) \right) \cap \gamma_{jr} \right) \quad (52)$$

Con disyuncion de premisas:
$$F(y_j) = \bigcup_{r=1}^R \left(W_r \left(\bigcup_{i=1}^N \mu_{ir}(x_i) \right) \cap \gamma_{jr} \right) \quad (53)$$

donde R es el número total de reglas y N el número total de entradas. Sin embargo, esta salida aún debe pasar primero por un proceso de defuzzificación en orden de llevar la salida difusa a una salida nítida.

Defuzzificación

En esta última etapa también llamada desdiferenciamiento, se obtienen los valores reales de salida. Ponce (2010) lo describe como el “[...] mapeo a escala que convierte el rango de valores de las variables de salida a sus universos de discurso correspondientes. La desdiferenciamiento es la herramienta para obtener la acción de control nítida a partir de una acción de control difusa” (p. 73). Este proceso es matemáticamente costoso y existen distintos métodos para realizarlo, el más común es el método de centro de área.

Método de centro de área. Consiste en calcular el centro de gravedad del conjunto difuso de salida $F(y)$ utilizando los valores y_q del universo de discurso de la salida, para lo cual se utiliza un total de Q elementos para el cálculo y así obtener un valor de salida nítido. El método de centro de área se puede representar en su forma discreta con la siguiente ecuación:

$$Salida = \frac{\sum_{q=1}^Q F(y_q) \cdot y_q}{\sum_{q=1}^Q F(y_q)} \quad (54)$$

Método de bisector de área. Este método ubica un punto en el conjunto difuso de salida y sus valores de pertenencia tal que el área resultante quede dividida en dos partes iguales.

Método de la media del máximo. Este método obtiene una salida a partir de promediar los valores de pertenencia máximos del conjunto difuso de salida.

Método del primero a la izquierda. Este método obtiene una salida a partir de los valores de pertenencia máximos del conjunto difuso de salida, de todos ellos, toma el ubicado más a la izquierda, o lo que es lo mismo, el menor de ellos.

Método del primero a la derecha. Este método obtiene una salida a partir de los valores de pertenencia máximos del conjunto difuso de salida, de todos ellos, toma el ubicado más a la derecha, o lo que es lo mismo, el mayor de ellos.

Sistemas de control difusos

Un controlador difuso es cualquier tipo de controlador que utilice en su interior alguna forma de lógica difusa, al igual que con los controladores clásicos el esquema de control más común es en lazo cerrado. El controlador difuso a su vez puede contener todo tipo de metodologías de control, de hecho, el uso más común es realizar un controlador P, PI, PD o PID, todos difusos. Para realizar los controladores se puede utilizar la estructura Mamdani, la cual asigna conjuntos difusos para las entradas y para las salidas, por tanto, es necesario el proceso de defuzzificación, comúnmente con el método de centro de área.

No existen métodos analíticos claros para el diseño de controladores difusos, por tanto, se debe realizar bajo la experiencia del diseñador, tomando en cuenta las necesidades de control y las dinámicas del proceso. Lo anterior es debido a que el

controlador difuso depende enteramente de la estructura de los conjuntos difusos y de la asignación de las reglas de inferencia (Ponce, 2010). Los siguientes esquemas de control fueron tomados de Ponce (2010), Riid (2002), y Matute y Bernal (2017).

Controlador P difuso. Un controlador P difuso simple puede ser realizado al tomar como entrada la señal de error y como salida la señal de control. El esquema de control se puede observar en la Figura 11.

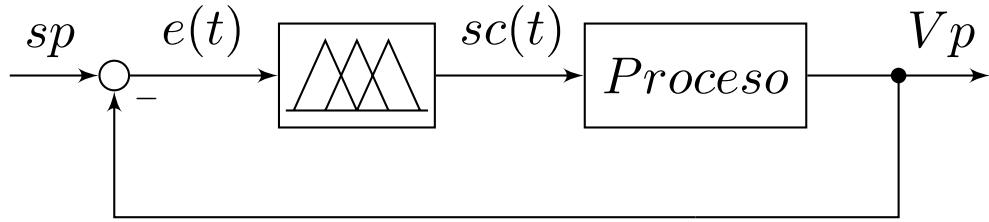


Figura 11. Esquema de control: P difuso. Fuente: Elaboración propia.

Controlador PD difuso. Un controlador PD difuso puede ser realizado al tomar la señal de error y la derivada del error como entradas del controlador y se toma como salida la señal de control. El esquema de control se puede observar en la Figura 12.

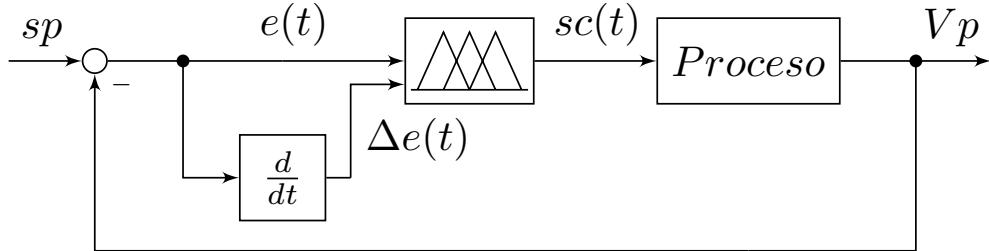


Figura 12. Esquema de control: PD difuso. Fuente: Elaboración propia.

Controlador PI difuso. Un controlador PI difuso se puede obtener al agregar un integrador a la salida de un PD difuso, de este modo $sc(t) = e(t) + \Delta e(t)$ se transforma en $sc(t) = \int_0^t e(\tau)d\tau + e(t)$ la cual será tomada como la señal de control. El esquema de control se puede observar en la Figura 13.

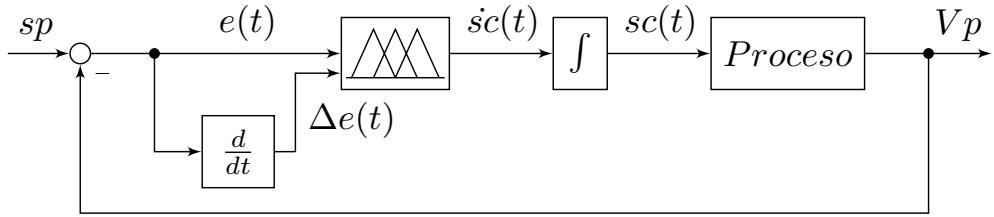


Figura 13. Esquema de control: PI difuso. Fuente: Elaboración propia.

Controlador PID difuso. Un controlador PD difuso puede ser realizado al tomar la señal de error, la derivada del error y la segunda derivada del error como entradas del controlador, la señal de salida es luego integrada de modo que $sc(t) = e(t) + \Delta e(t) + \Delta^2 e(t)$ se transforma en $sc(t) = \int_0^t e(\tau)d\tau + e(t) + \Delta e(t)$ la cual será tomada como la señal de control. El esquema de control para un PID difuso completo se puede observar en la Figura 14.



Figura 14. Esquema de control: PID difuso. Fuente: Elaboración propia.

Existen otras formas de obtener un PID difuso, una de estas formas es realizando la suma de las salidas de un PD difuso y PI difuso, por otro lado, también se puede realizar haciendo uso de componentes no difusos, i.e., sumar la salida de un PD difuso con la integral del error o sumar la salida de un PI difuso con la derivada del error.

Controlador difuso como programador de ganancias. El esquema de control difuso con programador de ganancias consiste en dos controladores, uno difuso y un PID clásico, el controlador difuso se encargará de generar las ganancias del PID de forma dinámica en función de las entradas que reciba, a su vez, el PID será quien gobierne al proceso, por lo cual permite adaptar el controlador PID a los cambios de carga y a los cambios del setpoint. El esquema de control se puede observar en la Figura 15.

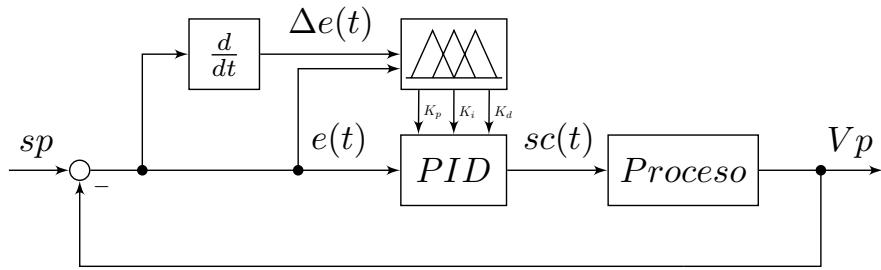


Figura 15. Esquema de control: Programador de ganancias. Fuente: Elaboración propia.

Controlador combinado. Consiste en sumar las salidas de un controlador PID clásico y un controlador difuso, de este modo el controlador PID será quien gobierne al proceso la mayor parte del tiempo. En el caso de utilizar un controlador P difuso y de generarse un cambio en el setpoint o la carga, el controlador difuso procederá a emitir una salida proporcional y adaptada al cambio. El esquema de control se puede observar en la Figura 16.



Figura 16. Esquema de control: PID clásico mas P difuso. Fuente: Elaboración propia.

Python

Python es un lenguaje de programación interpretado y multiparadigma ya que soporta programación orientada a objetos, imperativa y en menor medida funcional. Van Rossum (2017) afirma que Python es un lenguaje de programación potente que puede aprenderse fácilmente. Cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. La elegante sintaxis de Python y su escritura dinámica, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas.

Adicionalmente, Python posee bibliotecas externas para realizar cálculos numéricos complejos, también existen bibliotecas para el análisis de sistemas de control y para el diseño de controladores difusos, por otro lado, existen bibliotecas para salidas gráficas con calidad de publicación, gráficas en tiempo real y en 3D.

Biblioteca NumPy

NumPy es una biblioteca para realizar cálculos computacionales científicos y matemáticos, algunas de sus funciones son: cálculo de vectores n-dimensionales, transformada de Fourier, álgebra lineal, entre otras (Oliphant, 2006)

Biblioteca SciPy

El paquete SciPy es un conjunto de bibliotecas matemáticas, científicas y de ingeniería compuestas por: NumPy, SciPy (biblioteca), SymPy, Matplotlib, IPython y Pandas, la biblioteca SciPy es uno de los núcleos del paquete SciPy. Provee varias rutinas eficientes de uso amigable para el usuario de tipo numérica para realizar integración, interpolación, optimización, álgebra lineal, estadísticas, entre otras (Jones, Oliphant, Peterson et al., 2001), toda la documentación puede ser conseguida en la página oficial de SciPy.

Biblioteca Matplotlib

Matplotlib es una biblioteca para la graficación 2D que produce figuras con calidad de publicación en varios formatos y a través de múltiples ambientes interactivos. Matplotlib puede ser usado en scripting, consolas de comando, IPython, jupyter notebooks y en aplicaciones web (Hunter, 2007). La sintaxis para usar Matplotlib es muy similar al sistema de plots de MATLAB, del mismo modo, configurar parámetros de estilos, fuentes, anchos de línea, entre otros, se realiza de manera similar.

Biblioteca de control

La biblioteca control de python es un conjunto de clases y funciones que implementan las operaciones más comunes en sistemas de control, además, posee un

módulo de compatibilidad para usuarios de MATLAB que emula las funciones y sintaxis de este lenguaje (“Control Systems Library for Python”, 2017). Para crear el modelo de un sistema podemos usar ecuaciones de espacio de estado o funciones de transferencia.

Biblioteca Scikit-Fuzzy

Scikit-Fuzzy es una colección de algoritmos de lógica difusa con la intención de pertenecer al paquete de herramientas científicas SciPy, esta biblioteca permite diseñar y simular controladores difusos con estructura tipo Mamdani (Warner et al., 2016), no posee un modo para realizar lazos de control o compatibilidad con la biblioteca de control de forma directa, de modo que se tendrían que codificar aparte las rutinas que conformen un lazo cerrado de control.

Biblioteca PySide2

PySide2 es una biblioteca que hace de unión entre Qt y Python para la creación de interfaces de usuario. PySide2 es soportado por linux, Mac OS x, windows, entre otros.

Biblioteca PyQtGraph

PyQtGraph es una biblioteca de graficación especializada en gráficas dinámicas y en tiempo real, utiliza de fondo un núcleo Qt, por tanto, requiere el uso de PyQt4, PyQt5, PySide o PySide2.

CAPÍTULO III

MARCO METODOLÓGICO

En este capítulo se abarcará la metodología a emplear en el desarrollo de esta investigación, se definirá el tipo, diseño y modalidad de la investigación, así como las fases de la misma.

Tipo de investigación

Tomando en cuenta los objetivos de la investigación y las bases teóricas que la componen se considera que esta investigación es de tipo proyectiva, esto es debido a que se pretende realizar una propuesta concreta para solventar una problemática, Hurtado de Barrera (2010) afirma que:

La investigación proyectiva tiene como objetivo diseñar o crear propuestas dirigidas a resolver determinadas situaciones. Los proyectos de arquitectura e ingeniería, el diseño de maquinarias, la creación de programas de intervención social, el diseño de programas de estudio, los inventos, la elaboración de programas informáticos, entre otros, siempre que estén sustentados en un proceso de investigación, son ejemplos de investigación proyectiva. (p. 133)

Diseño de la investigación

El diseño de la investigación es no experimental y de tipo transeccional descriptivo, esto es debido a que se describirán los métodos de análisis y diseño de sistemas de control clásicos y difusos, Hernández, Fernández y Lucio (2010) definen la investigación no experimental como:

la investigación que se realiza sin manipular deliberadamente variables. Es decir, se trata de estudios donde no hacemos variar en forma

intencional las variables independientes para ver su efecto sobre otras variables. Lo que hacemos en la investigación no experimental es observar fenómenos tal como se dan en su contexto natural, para posteriormente analizarlos. (p.150)

Modalidad

Esta investigación se encuentra enmarcada en la modalidad de un proyecto factible, debido a que tiene objetivos para atender una necesidad por medio de unas acciones claramente definidas, Dubs de Moya (2002) afirma que:

un proyecto factible consiste en un conjunto de actividades vinculadas entre sí, cuya ejecución permitirá el logro de objetivos previamente definidos en atención a las necesidades que pueda tener una institución o un grupo social en un momento determinado. Es decir, la finalidad del proyecto factible radica en el diseño de una propuesta de acción dirigida a resolver un problema o necesidad previamente detectada en el medio. (pp. 6-7)

Fases de la investigación

Fase 1: Estudio de los sistemas de control clásicos y difusos. En esta fase se procederá a realizar los estudios necesarios en el área de los sistemas de control, esto con la idea de abarcarlos en profundidad y tener un entendimiento claro de su funcionamiento y de la matemática implicada, además, se realizará de forma similar un estudio de controladores difusos con estructura Mamdani y de los esquemas de control difuso.

Fase 2: Codificación de rutinas. Para esta fase con los conocimientos adquiridos de la fase 1, se determinarán que rutinas pueden ser ejecutadas solo con las bibliotecas de python y cuales se deberán codificar de cero, además, se codificarán todas las rutinas necesarias para el funcionamiento del laboratorio virtual, para esto, se hará

uso de las bibliotecas externas de cálculo numérico, control, diseño de controladores difusos y salidas gráficas junto con las que se consideren necesarias.

Fase 3: Interfaz gráfica y enlace con rutinas. En esta fase se realizará la interfaz gráfica para el usuario final, esta interfaz gráfica deberá conectarse y adaptarse a las rutinas previamente codificadas en la fase 2 para su funcionamiento adecuado, en orden de tener diseño acorde se tomarán en cuenta los antecedentes presentados en esta propuesta.

Fase 4: Comparación de resultados. En esta última fase y con el laboratorio de sistemas de control, ya en funcionamiento se procederá a analizar los resultados obtenidos y a compararlos con otras herramientas, la evaluación se realizará en función del resultado esperado, facilidad de implementación, velocidad de ejecución, las ventajas y desventajas de cada una de las herramientas.

CAPÍTULO IV

RESULTADOS

Estructura general

El laboratorio virtual de sistemas de control clásicos y difusos fue pensado de modo que cada una de sus funcionalidades principales sean independientes la una de la otra, no obstante, ciertas rutinas son comunes entre ellas. Por otro lado, se organizó la aplicación de modo que cada funcionalidad principal posea un archivo “handler”, el cual hará de intermediario entre la interfaz gráfica y las rutinas de calculo que se encuentran en un archivo de rutinas. La Figura 17 sirve como referencia de la estructura general utilizada.

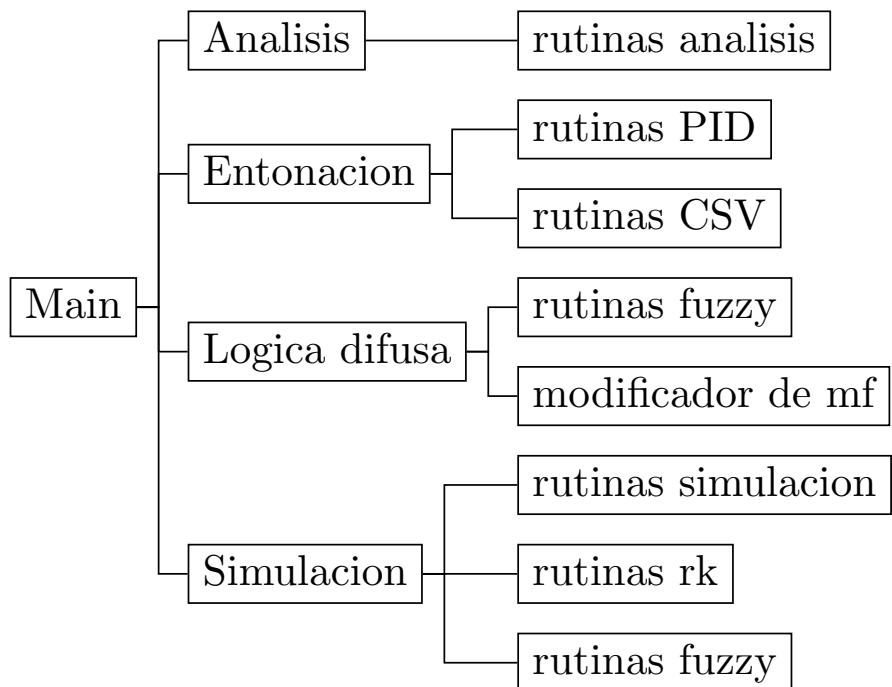


Figura 17. Diagrama general de la aplicación. La estructura general de la aplicación, la segunda columna de izquierda a derecha representa a los archivos handler mientras que la tercera son los archivos de rutinas. Fuente: Elaboración propia.

Análisis de sistemas de control

Con esta función se quiso dar la posibilidad de realizar los análisis en el tiempo y en frecuencia típicos para un proceso ingresado por el usuario, los análisis típicos son: respuesta al escalón, respuesta al impulso, diagrama de bode, diagrama de Nyquist, lugar de las raíces y diagrama de Nichols. El Código 1 muestra un pseudo código de la funcionalidad de análisis de sistemas de control y la interfaz gráfica correspondiente se puede observar en el [Anexo A].

Código 1. Pseudo código para el análisis de sistemas de control

```
WHILE True: # Ciclo principal
    :
    IF validaciones_entradas_análisis:
        validar_entrada_análisis()

    IF usuario_presiona_calcular:
        chequeo_TF_SS()
        creación_proceso()
        escalón()
        impulso()
        bode()
        nyquist()
        lugar_de_las_raíces()
        nichols()
        datos()
    :

```

Inicialmente se valida cada una de las entradas del usuario y se chequea si el botón para realizar los cálculos fue presionado. Una vez presionado el botón de calcular se procede a verificar la representación del sistema y a crearlo para a continuación ejecutar las rutinas de análisis.

Definición del proceso

Un proceso deberá ser ingresado para poder realizar el análisis, la entonación o la simulación del mismo. El sistema puede ser definido con los coeficientes del numerador y el denominador de la función de transferencia o puede ser definido ingresando las

matrices A, B, C y D de la ecuación de espacio de estados. En ambos casos se deben cumplir con los principios matemáticos que correspondan, i.g., la función de transferencia no puede ser impropia.

Tiempo discreto

La opción de discretización permite llevar el proceso ingresado en tiempo continuo a una aproximación en tiempo discreto con solo ingresar el periodo de muestreo y seleccionar el método a utilizar. Se implementaron los siguientes métodos de discretización:

1. Retenedor de orden cero (ZOH)
2. Retenedor de primer orden (FOH)
3. Euler hacia atrás (backward_diff)
4. Euler hacia adelante (Euler)
5. Transformación bilineal (Tustin)
6. Transformación directa de polos y ceros (matched)
7. Transformación del impulso invariante (impulse)

Proceso con atraso o delay

A los procesos ingresados se les puede agregar un atraso, este atraso es manejado de forma distinta dependiendo de la funcionalidad. Para el caso de la funcionalidad de análisis, el delay fue implementado para la respuesta escalón y la respuesta impulso como un desplazamiento en el tiempo equivalente al delay ingresado, para la respuesta en frecuencia se multiplico $e^{-j\omega\alpha}$ a cada una de las evaluaciones de frecuencia con α como el delay ingresado por el usuario. Para el caso del lugar de las raíces, el delay es implementado utilizando una aproximación por PADE.

Si el proceso es definido en tiempo discreto el mismo es multiplicado por $z^{-\alpha/T}$ con la idea de retrasar las muestras en un número equivalente al delay deseado, de este modo, no hay casos especiales dependiendo del tipo de análisis.

Respuesta al escalón

Se obtiene la respuesta en el tiempo del sistema en lazo abierto ante una entrada escalón de amplitud uno, con los datos de la respuesta se realiza la gráfica correspondiente.

Respuesta al impulso

Se obtiene la respuesta en el tiempo del sistema en lazo abierto ante un impulso como entrada al sistema, con los datos de la respuesta se realiza la gráfica correspondiente.

Diagrama de Bode

Se obtiene la respuesta en frecuencia del sistema, con los datos de la respuesta en frecuencia se realiza las gráficas de magnitud y fase. Adicional a las gráficas fue implementado el cálculo de los márgenes de ganancia y de fase, la implementación fue realizada desde cero debido a los resultados inaceptables obtenidos con la implementación de la librería de control. El código para el cálculo de los márgenes de ganancia y fase se puede observar en el [Anexo E].

Diagrama de Nyquist

El diagrama de Nyquist también fue realizado utilizando la librería de control, en el mismo se señala la dirección de la gráfica y el punto $-1 + j0$. Las frecuencias utilizadas son las mismas utilizadas para el diagrama de Bode dado que las mismas son calculadas de manera automática por la librería de control en ambos casos.

Gráfica del lugar de las raíces

La gráfica es realizada utilizando la librería de control, una de las funciones que posee es poder seleccionar cualquier punto en la gráfica y obtener la ganancia y el damping que le corresponden. Para el caso de sistemas discretos se grafica el círculo unitario.

Diagrama de Nichols

A diferencia del diagrama de Nyquist acá se señala el punto ($0dB, -180^\circ$), la librería de control ya viene con la opción para superponer una rejilla que representa a la Carta de Nichols.

Despliegue de datos

Los datos obtenidos se muestran de forma escrita, los datos a desplegar son: el sistema ingresado, los datos de la respuesta al escalón, los márgenes de ganancia y fase, los valores eigen y su respectivo damping.

Entonación de controladores PID

Esta función fue dividida en dos apartados, el primero, es la entonación de un proceso en lazo cerrado con un controlador PID, las ganancias del controlador son ajustables por el usuario, adicionalmente, se puede realizar una entonación automática utilizando el método de Ziegler-Nichols o Cohen-Coon. La segunda opción permite cargar la data de respuesta ante un escalón desde un archivo CSV para realizar la entonación. El Código 2 muestra un pseudo código de la funcionalidad de entonación de controladores PID y la interfaz gráfica correspondiente se puede observar en el [Anexo B].

Código 2. Pseudo código para la entonación de controladores PID

```
WHILE True: # Ciclo principal
    :
    IF validaciones_entradas_entonación:
        validar_entrada_entonación()
    IF usuario_presiona_calcular:
        CASO 1 entonación_manual:
            chequeo_TF_SS()
            creación_proceso()
            entonación()
            escalón()
        CASO 2 entonación_automática:
```

```

    chequeo_TF_SS()
    creación_proceso()
    escalón()
CASO 3 entonación_CSV:
    procesar_CSV()
    entonación()
    graficacion()
    :

```

La forma de definición de procesos presentado en la funcionalidad de análisis de sistemas de control es reutilizada para esta sección. Al igual que para el análisis de sistemas de control, inicialmente se valida cada una de las entradas del usuario y se chequea si el botón para realizar los cálculos fue presionado, una vez presionado el botón de calcular se procede a verificar que tipo de cálculo se va a realizar entre: entonación manual, entonación automática o entonación con archivo CSV.

Entonación de sistema en lazo cerrado con PID

Para el cálculo de la respuesta al escalón se utiliza la librería de control, el esquema implementado es el presentado en la Figura 8. El controlador PID fue definido utilizando (33) y sustituyendo la componente derivativa por (36). Si el sistema es representado en tiempo discreto se utiliza (35) para definir al controlador PID.

Así mismo, para la entonación automática se realiza una prueba escalón en lazo abierto con el propósito de calcular (38), (39) y (40), luego, se sustituyen los valores en la Tablas 1 y 2 con el fin de realizar una entonación por el método Ziegler-Nichols y Cohen-Coon según la selección del usuario.

Entonación utilizando un archivo CSV

Como ya se mencionó, con esta opción se permite cargar un archivo CSV que contenga la data de respuesta a un escalón con la finalidad de obtener su aproximación de primer orden con tiempo muerto y poder realizar una entonación utilizando el método de Ziegler-Nichols o Cohen-Coon. El usuario debe ingresar el límite superior e inferior de la variable del proceso (V_p) y de la entrada al sistema, normalmente, la señal del

elemento final de control (EFC), adicionalmente, debe ingresar el separador del archivo CSV cuyo valor por defecto es una coma (,).

El algoritmo aproximara los parámetros de un modelo de primer orden con tiempo muerto utilizando el punto de máxima pendiente para trazar una recta, este punto quedara como ancla permitiendo al usuario ajustar el tiempo t_1 en caso de que el ajuste automático no genere buenos resultados. El algoritmo solo funciona para respuestas ascendentes, i.g., como la observada en la Figura 9.

El archivo CSV deberá contener un formato de encabezados específico, los encabezados deben contener las palabras clave VP, EFC o TIME para identificar las columnas. El algoritmo intentara buscar las palabras clave dentro del encabezado de cada columna, i.g., datavpsimulacion identificaría a los datos de Vp debido a que incluye la palabra clave Vp, no se hace distinción entre minúsculas y mayúsculas. La columna que contiene los datos de tiempo puede expresar el tiempo en los formatos hh:mm:ss, mm:ss o directamente en segundos. Un ejemplo del formato de un CSV valido se puede observar en el [Anexo F].

Diseño de controladores difusos

Para el diseño de controladores difusos se utiliza de fondo la librería Scikit-Fuzzy, la cual permite por medio de código Python diseñar controladores difusos tipo Mamdani, graficar las funciones de membresía y realizar el proceso de inferencia para el controlador diseñado. La mayoría de la funcionalidad consiste en ser un intermediario entre las entradas del usuario por medio de la interfaz gráfica y el código que requiere Scikit-Fuzzy para poder definir un controlador. Adicional a lo anterior, se implementó la posibilidad de cargar y exportar archivos FIS con ciertas limitaciones tanto en la importación como en la exportación. El Código 3 muestra un pseudo código de la funcionalidad de diseño de controladores difusos y la interfaz gráfica correspondiente se puede observar en el [Anexo C].

Código 3. Pseudo código para el diseño de controladores difusos

```
WHILE True: # Ciclo principal
    :
    IF validaciones_entradas_fuzzy:
        validar_entrada_fuzzy()

    IF usuario_presiona_generar:
        IF esquemas == True:
            empezar_diseño_esquema()
        ELSE
            empezar_diseño_genérico()

    IF usuario_presiona_crear:
        crear_controlador()
        prueba_entradas()
        IF entradas == 1 OR entradas == 2:
            superficie_respuesta()
    :

```

Controladores difusos

Los controladores difusos fueron divididos para su diseño en dos categorías, genéricos y basados en un esquema de control específico.

Controladores genéricos

Los controladores genéricos pueden tener un numero arbitrario de entradas y salidas con un máximo de diez entradas o salidas, al ser genéricos, no se garantiza que puedan ser utilizados en la funcionalidad de simulación de sistemas de control, por tanto, sirve para diseñar cualquier sistema de inferencia difuso con o sin el propósito de utilizarle como controlador.

Funciones de membresía. Scikit-Fuzzy posee todas las funciones de membresías descritas por las ecuaciones (41) a la (47) junto con las funciones compuestas: gauss2mf, dsigmf, psigmf y pimf. Para poder cambiar entre funciones de membresía se codificaron transformaciones aproximadamente equivalentes entre las definiciones de las mismas, se llamó definición a los valores necesarios para poder definir la función

de membresía. El código para realizar las transformaciones se puede observar en el [Anexo G].

Defuzzificacion. Todos los métodos de defuzzificación descritos en Capítulo II son utilizables con Scikit-Fuzzy y pueden ser asignados de forma individual a cada salida del controlador difuso, i.e., cada salida puede utilizar un método de defuzzificación distinto.

Reglas de inferencia. Las reglas pueden plantearse utilizando conjunción de premisas o disyunción de premisas, i.e., lógica AND o lógica OR, a su vez, cada una de las premisas pueden ser negadas. A las salidas se les puede asignar un peso de forma individual, de modo que cada salida puede ser ponderada de forma distinta, las salidas no pueden ser negadas.

Controladores basados en esquemas

Los controladores basados en esquemas poseen todas las características de los controladores difusos a excepción de la posibilidad de seleccionar el número de entradas, el número de salidas y los nombres de las variables de entrada y salida, el motivo es que estos controladores están asociados a un esquema de control específico que debe cumplir con un número de entradas y salidas ya establecido. Los esquemas de control difuso que se pueden diseñar son los presentados en las Figuras 12 a 16 con ciertos cambios que se detallaran, junto con algunos esquemas extras, en la sección correspondiente a la funcionalidad de simulación. Todos los esquemas de control difuso se pueden observar en el [Anexo H].

Guardado de controladores difusos

Archivos JSON

Para guardar el diseño del usuario se utiliza un archivo .JSON con una estructura interna particular y única para funcionar de forma específica con el laboratorio virtual

de sistemas de control, los archivos JSON guardan datos que luego pueden ser cargados de forma directa con la librería JSON de Python. Un ejemplo del formato interno del archivo .JSON para un controlador con una entrada y una salida se puede observar en el [Anexo I].

Archivos FIS

Los archivos FIS son otra forma de cargar el controlador difuso o exportarlo, existen ciertas limitaciones en la carga y en el exportado de archivos FIS producto de las posibilidades que ofrece Scikit-Fuzzy.

Cargar controlador. Para la carga del controlador se utiliza una versión modificada del parsin de archivos FIS de YAPFLM (Yet Another Python Fuzzy Logic Module) de sputnick1124 (2017) para extraer los datos del archivo FIS, luego, otro código es utilizado para llevar los datos extraídos a la estructura presentada en los archivos .JSON. Debido a que Scikit-Fuzzy no permite salidas negadas en las reglas se genera un error en caso de que alguna de las salidas en las reglas se encuentre negada.

Exportar controlador. Para realizar el exportado del controlador difuso se realiza un parsin entre la estructura presentado en los archivos .JSON y la de un archivo FIS. En caso de que alguna regla posea pesos diferentes para cada salida solo se tomara en cuenta el peso asociado a la primera salida, esto es debido a que los archivos FIS no permiten pesos individuales por cada salida, lo mismo aplica al método de defuzzificación, se toma el asignado a la primera salida. El código para realizar la carga y el exportado de archivos FIS se puede observar en el [Anexo J].

Simulación de sistemas de control

Esta funcionalidad permite simular sistemas de control en lazo cerrado con un controlador PID clásico, difuso o ambos, adicional al controlador se permite agregar bloques adicionales como accionador, saturador del accionador y/o sensor. Para la

simulación se utilizan un método de Runge-Kutta como solucionador y obtener la respuesta en el tiempo del sistema, los esquemas simulables están predefinidos. El Código 4 muestra un pseudo código de la funcionalidad de simulación de sistemas de control y la interfaz gráfica correspondiente se puede observar en el [Anexo D].

Código 4. Pseudo código para la simulación de sistemas de control

```
WHILE True: # Ciclo principal
    :
    IF validaciones_entradas_simulación:
        validar_entrada_simulación()

    IF usuario_presiona_calcular:
        chequeo_esquema()
        datos = recolección_datos()
        hilo_de_simulación(datos)
    :

FUNCTION hilo_de_simulación(datos):
    procesar_datos(datos)
    resultados = simular_esquema(método_rk)
    return resultados
```

Al igual que para la funcionalidad de análisis de sistemas de control y entonación de controladores PID se reutiliza la forma de ingresar el proceso, por lo que también se realiza la validación de entradas y se monitorea si el usuario presiona el botón de calcular, cuando el usuario presiona calcular, se procede a realizar la simulación del esquema de control seleccionado en un hilo distinto al de la aplicación principal debido a que los tiempos de simulación para los esquemas difusos pueden ser de varios segundos y presentaría un comportamiento de bloqueo para la ventana de la aplicación.

En caso de que la simulación sea realizada para tiempo discreto, se evaluá (9) en cada iteración para obtener la salida del sistema. Si es el esquema seleccionado es el del PID clásico el controlador implementado es el descrito por (34). Si el sistema es continuo, la respuesta del sistema y del controlador es calculada utilizando el método de Runge-Kutta seleccionado.

Esquemas de simulación

Para la simulación de sistemas de control se implementaron los esquemas de control presentados en el Capítulo II y otros esquemas que se encuentran compuestos de la combinación de los mismos. Como ya se mencionó, los esquemas de control se pueden observar en el [Anexo H] y se listan a continuación:

1. PID clásico
2. PID difuso
3. PI difuso
4. PD difuso
5. PD difuso + PI difuso
6. PI difuso + Derivada clásica
7. PD difuso + Integrador clásico
8. Programador de ganancias
9. PID clásico + Difuso simple (P)

A la derivada de aquellos esquemas que requieran su cálculo se les fue agregado una función de transferencia en serie con la misma para limitar la respuesta a las altas frecuencias, esto es debido a que las simulaciones se van a realizar con un método de Runge-Kutta que requeriría un tamaño de paso muy pequeño para poder aproximar de forma correcta incluso a una derivada aproximada dada por (36), y un incremento en el número de pasos provocaría, a su vez, un incremento en los tiempos de simulación para los esquemas con controladores difusos, y en consecuencia, volviendo la herramienta impráctica.

Solución por Runge-Kutta

Como ya se mencionó, la simulación se llevará a cabo utilizando un método de Runge-Kutta, para ello se implementaron múltiples métodos, el usuario puede seleccionar cualquiera de ellos para realizar la simulación. En total se implementaron nueve Runge-Kutta's explícitos y cuatro embebidos. Las tablas de Butcher para cada uno de los

métodos implementados se pueden observar en el [Anexo K], a continuación, se presenta una lista de los métodos implementados:

1. Explícitos:

- a) Runge-Kutta de orden 2
- b) Runge-Kutta de orden 3
- c) Ralston de orden 3
- d) Heun de orden 3
- e) Strong Stability Preserving Runge-Kutta (SSPRK) de orden 3
- f) Runge-Kutta de orden 4
- g) Runge-Kutta 3/8 de orden 4
- h) Ralston con mínimo error de truncamiento de orden 4
- i) Runge-Kutta de orden 5

2. Embebidos:

- a) Bogacki-Shampine de orden 3(2)
- b) Fehlberg de orden 4(5)
- c) Cash-Karp de orden 4(5)
- d) Dormand-Prince de orden 5(4)

Con el fin mejorar la precisión y mantener el número de pasos al mínimo se implementó un algoritmo de ajuste del tamaño de paso para los métodos de Runge-Kutta, para el caso de los Runge-Kutta explícitos se implementaron las ecuaciones (27), (29) y (30) y para los embebidos las ecuaciones (28) a (30). Los pseudo códigos para los tamaños de paso variable se pueden observar en los Códigos 5 y 6, y los códigos en Python se pueden observar en el [Anexo L].

El cálculo del tamaño de paso se realiza con el bloque que presente mayores variaciones y requiera la mayor precisión, i.e., el menor tamaño de paso, en la mayoría de los casos este bloque es o incluye a la componente derivativa. Para el resto de bloques que requieran ser calculados, se utiliza un Runge-Kutta explicito con tamaño de paso igual al calculado, el Runge-Kutta utilizado varia según el método utilizado.

Código 5. Pseudo código para el ajuste del tamaño de paso de los Runge-Kutta explícitos.

```
FUNCION Tamaño_de_paso_variable(parametros):
    WHILE True:
         $y_h, x_h = calculo_rk(h, x)$ 
         $-, x_{h/2} = calculo_rk(h/2, x)$ 
         $y_{h/2}, x_{h/2} = calculo_rk(h/2, x_{h/2})$ 
         $escala = atol + rtol * (|x_h| + |x_{h/2}|)/2$ 
         $error_norm = norma_rms(|x_h - x_{h/2}|/escala)$ 

        CASO 1 error_norm == 0:
             $h = h * maxStepIncrease$ 

        CASO 2 error_norm <= 1:
             $h = h * min(maxStepIncrease,$ 
             $max(1, sf * error_norm^{(-1/(ordenq+1))}))$ 

        CASO 2 error_norm > 1:
             $h = h * min(1, max(minStepDecrease,$ 
             $sf * error_norm^{(-1/(ordenq+1))}))$ 
            continue
        break
    return  $y_{h/2}, x_{h/2}, h$ 
```

Código 6. Pseudo código para el ajuste del tamaño de paso de los Runge-Kutta embebidos.

```
FUNCION Tamaño_de_paso_variable(parametros):
    WHILE True:
         $y, x_p, x_{\hat{p}} = calculo_rk(h, x)$ 
         $escala = atol + rtol * max(|x_p|, |x|)$ 
         $error_norm = norma_rms(|x_p - x_{\hat{p}}|/escala)$ 

        CASO 1 error_norm == 0:
             $h = h * maxStepIncrease$ 

        CASO 2 error_norm <= 1:
             $h = h * min(maxStepIncrease,$ 
             $max(1, sf * error_norm^{(-1/(ordenq+1))}))$ 

        CASO 2 error_norm > 1:
             $h = h * min(1, max(minStepDecrease,$ 
             $sf * error_norm^{(-1/(ordenq+1))}))$ 
            continue
        break
    return  $y, x_p, h$ 
```

Comparación con MATLAB y SciLab

Análisis de sistemas de control

Para realizar la comparación entre el Laboratorio Virtual, MATLAB y SciLab se seleccionaron tres sistemas distintos, los sistemas fueron seleccionados de modo que abarcaran la mayor parte de las funciones disponibles. En la Tabla 3 se puede observar los sistemas a utilizar, los sistemas discretos se obtuvieron a partir de los continuos utilizando un ZOH, transformación Tustin y un FOH respectivamente, con periodos de muestreo de 0.1s, 0.2s y 0.05s en cada caso.

Tabla 3.
Sistemas para la comparación de análisis de sistemas de control

Nº	Continuo	Discreto
1	$\frac{1}{s^2 + s + 1}$	$\frac{0.004833z + 0.004675}{z^2 - 1.895z + 0.9048}$
2	$A = \begin{bmatrix} -0.5 & -3 & -0.2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ $B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ $C = [1 \ 2 \ 0.5]$ $D = [0]$	$A = \begin{bmatrix} 851.5 & -559.2 & -37.03 \\ 185.2 & 944.1 & -3.703 \\ 18.52 & 194.4 & 999.6 \end{bmatrix}_{10^{-3}}$ $B = \begin{bmatrix} 185.2 \\ 18.52 \\ 1.852 \end{bmatrix}_{10^{-3}}$ $C = [1.116 \ 1.713 \ 0.4777]$ $D = [0.1116]$
3	$\frac{s + 2}{s^2 + 0.5s + 3} e^{-1.5s}$	$z^{-30} \left(\frac{0.02561z^2 + 0.003084z - 0.02375}{z^2 - 1.968z + 0.9753} \right)$

Para poder realizar los cálculos numéricos de diferencia absoluta y diferencia de área se realizó una interpolación con el fin de ajustar el número de muestras entre los vectores de tiempo/frecuencia y los valores correspondientes, i.g., que ambos vectores de salida tengan un valor de salida para $t=0.5s$. En el caso de MATLAB, las funciones de análisis de sistemas de control crean su propio vector de tiempo y frecuencia según sea el caso, por otro lado, en SciLab se deben asignar los valores de forma explícita, por tanto, se utilizó el vector de tiempo del Laboratorio Virtual y el vector de frecuencia de MATLAB.

Con las consideraciones anteriores se realizaron las comparaciones respectivas y se obtuvieron los resultados observados en las Tablas 4 y 5:

Tabla 4.
Comparación de análisis para los sistemas en tiempo continuo

Sistema	MATLAB			SciLab	
	1	2	3	1	2
Respuesta escalón					
Diferencia absoluta	9.271×10^{-6}	2.159×10^{-5}	6.380×10^{-3}	3.030×10^{-12}	4.457×10^{-9}
Diferencia porcentual	5.865×10^{-2}	1.620×10^{-3}	1.738	2.877×10^{-10}	1.867×10^{-7}
RECM	1.409×10^{-6}	2.052×10^{-6}	1.278×10^{-3}	1.142×10^{-12}	1.364×10^{-9}
Diferencia de area	4.224×10^{-7}	9.102×10^{-6}	3.154×10^{-3}	1.862×10^{-12}	8.108×10^{-8}
Distancia de energía	9.796×10^{-5}	4.379×10^{-5}	2.346×10^{-3}	2.892×10^{-8}	4.225×10^{-7}
Respuesta impulso					
Diferencia absoluta	1.117×10^{-5}	4.449×10^{-5}	4.489×10^{-5}	4.473×10^{-12}	1.129×10^{-10}
Diferencia porcentual	6.702×10^{-3}	3.385×10^{-3}	3.514×10^{-3}	8.514×10^{-7}	1.717×10^{-5}
RECM	2.026×10^{-6}	4.125×10^{-6}	7.657×10^{-6}	1.557×10^{-12}	5.246×10^{-11}
Diferencia de area	8.168×10^{-6}	1.290×10^{-5}	1.600×10^{-5}	1.422×10^{-12}	2.720×10^{-9}
Distancia de energía	1.120×10^{-4}	5.985×10^{-5}	1.712×10^{-4}	3.540×10^{-8}	9.027×10^{-8}
Bode - Magnitud					
Diferencia absoluta	3.072×10^{-5}	7.125×10^{-4}	4.807×10^{-4}	3.072×10^{-5}	7.125×10^{-4}
Diferencia porcentual	–	7.032×10^{-3}	5.047×10^{-3}	–	7.032×10^{-3}
RECM	1.204×10^{-5}	1.339×10^{-4}	1.072×10^{-4}	1.204×10^{-5}	1.339×10^{-4}
Diferencia de area	1.259×10^{-3}	2.379×10^{-3}	1.179×10^{-3}	1.259×10^{-3}	2.379×10^{-3}
Distancia de energía	5.439×10^{-4}	1.118×10^{-3}	1.224×10^{-3}	5.439×10^{-4}	1.118×10^{-3}
Bode - Fase					
Diferencia absoluta	2.525×10^{-4}	9.020×10^{-3}	3.272×10^{-3}	2.525×10^{-4}	9.020×10^{-3}
Diferencia porcentual	2.062×10^{-4}	1.161×10^{-2}	1.287×10^{-3}	2.062×10^{-4}	1.161×10^{-2}
RECM	6.375×10^{-5}	1.276×10^{-3}	7.010×10^{-4}	6.375×10^{-5}	1.276×10^{-3}
Diferencia de area	5.777×10^{-4}	2.603×10^{-5}	1.509×10^{-4}	5.777×10^{-4}	2.603×10^{-5}
Distancia de energía	9.920×10^{-4}	2.885×10^{-3}	3.068×10^{-3}	9.920×10^{-4}	2.885×10^{-3}
Margen de ganancia					
Diferencia absoluta	NaN	NaN	1.625×10^{-2}	NaN	NaN
Diferencia porcentual	NaN	NaN	1.689×10^{-1}	NaN	NaN
Margen de fase					
Diferencia absoluta	1.760×10^{-1}	5.918×10^{-2}	2.926×10^{-1}	1.760×10^{-1}	5.793×10^{-2}
Diferencia porcentual	1.955×10^{-1}	8.494×10^{-2}	2.577×10^{-1}	1.955×10^{-1}	8.315×10^{-2}

Sistema	MATLAB			SciLab	
	1	2	3	1	2
Nyquist					
Diferencia absoluta	2.283×10^{-3}	1.466×10^{-2}	6.922×10^{-2}	2.283×10^{-3}	1.466×10^{-2}
Diferencia porcentual	1.662	2.053	2.687×10^2	1.662	2.053
RECM	4.593×10^{-4}	2.131×10^{-3}	1.333×10^{-2}	4.593×10^{-4}	2.131×10^{-3}
Diferencia de area	8.741×10^{-4}	1.253×10^{-2}	8.024×10^{-2}	8.741×10^{-4}	1.253×10^{-2}
Distancia de energía	4.725×10^{-4}	6.415×10^{-3}	8.350×10^{-3}	4.725×10^{-4}	6.415×10^{-3}
Lugar de las raíces					
Distancia de energía	4.268×10^{-1}	4.136×10^{-1}	1.379×10^{-1}	–	–
Nichols					
Diferencia absoluta	1.037	8.484×10^{-2}	7.789×10^{-2}	1.037	8.484×10^{-2}
Diferencia porcentual	1.414	1.023	1.068	1.414	1.023
RECM	2.092×10^{-1}	1.641×10^{-2}	1.846×10^{-2}	2.092×10^{-1}	1.641×10^{-2}
Diferencia de area	4.596×10^{-1}	1.418	1.597×10^2	4.596×10^{-1}	1.418
Distancia de energía	2.369×10^{-2}	3.587×10^{-3}	3.421×10^{-3}	2.369×10^{-2}	3.587×10^{-3}

Tabla 5.
Comparación de análisis para los sistemas en tiempo discreto

Sistema	MATLAB			SciLab
	1	2	3	1
Respuesta escalón				
Diferencia absoluta	8.438×10^{-15}	6.457×10^{-13}	5.895×10^{-14}	1.021×10^{-14}
Diferencia porcentual	8.437×10^{-13}	2.592×10^{-11}	6.456×10^{-12}	1.008×10^{-12}
RECM	3.472×10^{-15}	4.229×10^{-13}	2.053×10^{-14}	4.687×10^{-15}
Diferencia de area	0.000	3.939×10^{-11}	5.009×10^{-13}	3.553×10^{-15}
Distancia de energía	5.361×10^{-9}	3.983×10^{-8}	6.870×10^{-9}	6.034×10^{-9}
Respuesta impulso				
Diferencia absoluta	1.832×10^{-15}	3.916×10^{-14}	3.331×10^{-14}	1.832×10^{-15}
Diferencia porcentual	5.258×10^{-13}	2.352×10^{-11}	2.784×10^{-12}	5.258×10^{-13}
RECM	5.464×10^{-16}	8.837×10^{-15}	9.464×10^{-15}	5.485×10^{-16}
Diferencia de area	2.220×10^{-16}	5.289×10^{-13}	1.354×10^{-14}	2.220×10^{-16}
Distancia de energía	1.759×10^{-9}	4.588×10^{-9}	3.871×10^{-9}	1.769×10^{-9}
Bode - Magnitud				
Diferencia absoluta	3.893×10^{-3}	3.378×10^{-1}	2.067×10^{-2}	3.893×10^{-3}
Diferencia porcentual	5.135×10^{-3}	5.775×10^{-1}	3.383×10^{-2}	5.135×10^{-3}

Sistema	MATLAB			SciLab
	1	2	3	
RECM	6.899×10^{-4}	3.636×10^{-2}	3.492×10^{-3}	6.899×10^{-4}
Diferencia de area	7.633×10^{-3}	3.772×10^{-2}	3.476×10^{-2}	7.633×10^{-3}
Distancia de energía	2.480×10^{-3}	1.036×10^{-2}	4.893×10^{-3}	2.480×10^{-3}
<hr/>				
Bode - Fase				
Diferencia absoluta	4.406×10^{-1}	9.114×10^{-3}	5.692×10^{-1}	4.406×10^{-1}
Diferencia porcentual	1.958×10^{-1}	1.164×10^{-2}	1.039×10^{-2}	1.958×10^{-1}
RECM	5.350×10^{-2}	1.775×10^{-3}	7.215×10^{-2}	5.350×10^{-2}
Diferencia de area	1.140×10^{-1}	8.149×10^{-4}	3.163×10^{-1}	1.140×10^{-1}
Distancia de energía	1.446×10^{-2}	3.770×10^{-3}	1.847×10^{-2}	1.446×10^{-2}
<hr/>				
Margen de ganancia				
Diferencia absoluta	6.090×10^{-2}	NaN	1.624×10^{-2}	5.524×10^{-2}
Diferencia porcentual	2.326×10^{-1}	NaN	1.688×10^{-1}	2.110×10^{-1}
<hr/>				
Margen de fase				
Diferencia absoluta	1.231×10^{-1}	1.454×10^{-1}	4.178×10^{-3}	1.314×10^{-1}
Diferencia porcentual	1.411×10^{-1}	2.087×10^{-1}	3.680×10^{-3}	1.508×10^{-1}
<hr/>				
Nyquist				
Diferencia absoluta	2.284×10^{-3}	1.493×10^{-2}	5.538×10^{-2}	2.284×10^{-3}
Diferencia porcentual	2.026	2.128	3.310×10^2	2.026
RECM	4.328×10^{-4}	2.066×10^{-3}	1.184×10^{-2}	4.328×10^{-4}
Diferencia de area	9.357×10^{-4}	1.303×10^{-2}	6.475×10^{-2}	9.357×10^{-4}
Distancia de energía	4.358×10^{-4}	1.323×10^{-3}	8.612×10^{-3}	4.358×10^{-4}
<hr/>				
Lugar de las raíces				
Distancia de energía	2.601×10^{-1}	2.076×10^{-2}	6.991×10^{-1}	–
<hr/>				
Nichols				
Diferencia absoluta	9.413×10^{-2}	5.167×10^{-1}	1.346×10^{-1}	9.413×10^{-2}
Diferencia porcentual	1.302×10^{-1}	1.147	2.610×10^{-1}	1.302×10^{-1}
RECM	1.643×10^{-2}	2.490×10^{-2}	1.891×10^{-2}	1.643×10^{-2}
Diferencia de area	1.907	1.093	4.307	1.907
Distancia de energía	2.518×10^{-3}	3.993×10^{-3}	3.392×10^{-3}	2.518×10^{-3}

Los datos y sistemas faltantes en las tablas mostradas se deben a limitaciones técnicas de SciLab o la imposibilidad de realizar los cálculos. Ante las limitaciones técnicas de SciLab se optó por descartar la comparación bajo la premisa que la facilidad

de implementación es un factor a tener en cuenta en estas comparativas, a continuación, se presenta una lista con las razones específicas:

- SciLab no posee funciones directas para implementar sistemas con atrasos o aproximarlos con PADE.
- La diferencia máxima se encontraba en un valor de cero (0), por lo que no se puede calcular la diferencia porcentual.
- NaN indica que el margen de ganancia para ese sistema es infinito.
- La función evans de SciLab, la cual se encarga de generar el lugar de las raíces no devuelve ningun tipo de data, por cual no se puede realizar una comparación numérica o gráfica.
- Cuando se realizaron los cálculos para el lugar de las raíces se observó que existía una pobre alineación entre los valores de salida, por tanto, realizar un cálculo de error arrojaba valores no confiables. El problema también fue detectado al realizar la graficacion del área entre una respuesta y la otra, en donde se observaban áreas entre tramos separados del lugar de las raíces.
- La función para discretizar sistemas continuos de SciLab no permite seleccionar un método en particular y utiliza, por defecto, ZOH.

De las Tablas 4 y 5 se puede extraer que existentes diferencias mínimas entre el Laboratorio Virtual y MATLAB/SciLab en términos de precisión, a su vez, se logra percibir la gran dependencia que existe del vector de tiempo y frecuencia. Las métricas de análisis de frecuencia dieron iguales o muy parecidas tanto para MATLAB como para SciLab, esto se debe a que se utilizó el mismo vector de frecuencia para ambos, de modo similar, para la respuesta escalón e impulso se obtuvieron diferencias mínimas entre el Laboratorio Virtual y SciLab por el uso del mismo vector de tiempo, lo anterior también es observable para los sistemas discretos dado que los vectores de tiempo son gobernados por el periodo de muestreo. La comparación grafica con MATLAB se puede observar a modo de ejemplo en la Figura 18 para el sistema continuo uno (1), para observar la comparación grafica de los demás sistemas ir al [Anexo M].

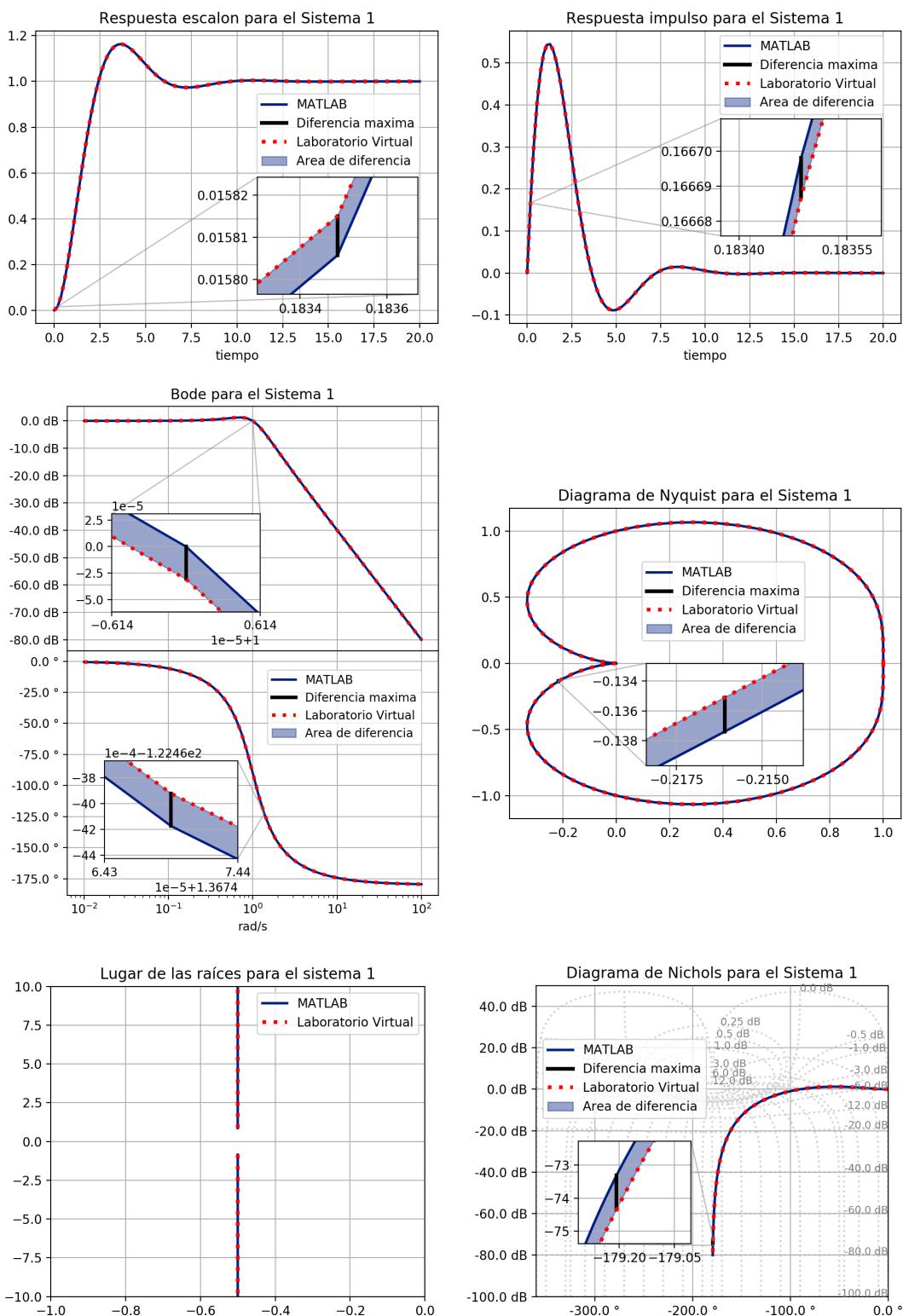


Figura 18. Comparación de análisis/MATLAB - sistema continuo 1. Fuente: Elaboración propia.

La anterior comparativa es basada en precisión utilizando las funciones de MATLAB y SciLab a base de código, no obstante, el Laboratorio Virtual permite realizar los análisis con solo definir el sistema a analizar, este método de análisis también se puede realizar en MATLAB con la aplicación (app) Linear System Analyzer, la cual permite realizar las mismas funciones, en el caso de SciLab no existe una aplicación similar.

Entonación de controladores PID

Para la función de entonación de controladores PID no se realizó una comparación numérica dado que los resultados serán los ya vistos en la función de análisis, por tanto, la comparación se enfocará en la función per se y su forma de uso.

El Laboratorio Virtual permite ajustar las ganancias del controlador PID con solo deslizar los sliders correspondientes, permitiendo visualizar los efectos del aumento de cada una de los componentes del controlador PID, a su vez, permite habilitar y deshabilitar cada componente de forma sencilla permitiendo así, una forma visual de entonación rápida y sin complicaciones. En el caso de MATLAB, existen varias herramientas de entonación, la que presenta mayor similitud es el PID Tuner, el cual permite entonar el controlador basado en el tiempo de respuesta y en el transitorio, no permite ajustar de forma individual cada ganancia, pero si contiene otras funciones no disponibles en el Laboratorio Virtual. En el caso de SciLab, no existe ninguna herramienta que cumpla con la función de entonar controladores PID.

En ambos casos, tanto el Laboratorio Virtual como el PID Tuner de MATLAB permite realizar una entonación automática y realizar ajustes finales de forma manual, el Laboratorio Virtual utiliza los métodos de Ziegler-Nichols y Cohen-Coon, en el caso de MATLAB no se conoce que método es utilizado. Para la identificación y entonación de sistemas a partir de data recolectada, MALTAB posee la aplicación System Identification, la cual tiene una gran cantidad de opciones superiores al Laboratorio Virtual en lo que a identificación de sistemas respecta, no obstante, el Laboratorio Virtual tiene la ventaja

de poseer una entonación automática al mismo tiempo que se realiza la identificación. Nuevamente, SciLab no posee herramientas que puedan realizar identificación de sistemas a partir de la data recolectada, si bien, las anteriores funciones pueden ser implementadas a código, el problema recae en el usuario.

Diseño de controladores difusos

La comparación se realizó utilizando como base dos controladores difusos diseñados utilizando el Laboratorio virtual y exportando su diseño en un archivo FIS. Los controladores fueron diseñados bajo el principio de utilizar gran parte de las posibilidades en términos de funciones de membresía y reglas, por tanto, no serán aptos para el control de procesos, pero si ideales para realizar la comparación con MATLAB y SciLab.

El primer controlador posee una entrada y una salida, la entrada a su vez contiene cinco etiquetas genéricas con una función de membresía distinta cada una, por otro lado, la salida está compuesta de tres etiquetas con funciones de membresía triangulares, el método de defuzzificación seleccionado es el del centro de área (Centroid). En la Figura 19 se pueden observar las funciones de membresía de la entrada y la salida junto con las áreas de activación de reglas para una determinada entrada. El segundo controlador diseñado posee dos entradas y dos salidas con tres etiquetas en todos los casos, el método de defuzzificación seleccionado fue el de primero a la izquierda (SOM). La estructura de los archivos FIS se puede observar en el [Anexo N].



Figura 19. Controlador difuso 1 para la comparación. (a) Entrada del controlador, (b) Salida del controlador. Fuente: Elaboración propia.

En las Tablas 6 y 7 se presentan los set de entradas y salidas para los controladores 1 y 2 respectivamente. Las salidas de MATLAB se obtuvieron utilizando la función evalfis y evalfls para SciLab, en ambos casos se utilizó un parámetro de valores de interpolación de 5001.

Tabla 6.
Comparación para el controlador 1

Entrada1	Salida1		
	Laboratorio Virtual	MATLAB	SciLab
-10	0.00	0.00	-1.30×10^{-6}
-9	4.20×10^{-2}	4.20×10^{-2}	4.20×10^{-2}
-8	1.30×10^{-1}	1.30×10^{-1}	1.30×10^{-1}
-7	2.77×10^{-1}	2.77×10^{-1}	2.77×10^{-1}
-6	4.78×10^{-1}	4.78×10^{-1}	4.78×10^{-1}
-5	6.99×10^{-1}	6.99×10^{-1}	6.99×10^{-1}
-4	8.05×10^{-1}	8.05×10^{-1}	8.05×10^{-1}
-3	5.83×10^{-1}	5.83×10^{-1}	5.83×10^{-1}
-2	1.51×10^{-1}	1.52×10^{-1}	1.52×10^{-1}
-1	-3.00×10^{-2}	-3.02×10^{-2}	-3.02×10^{-2}
0	0.00	0.00	5.53×10^{-16}
1	3.88	3.88	3.88
2	1.41	1.41	1.41
3	9.93×10^{-1}	9.94×10^{-1}	9.94×10^{-1}
4	1.41	1.41	1.41
5	1.47	1.47	1.47
6	1.17	1.17	1.17
7	6.46×10^{-1}	6.46×10^{-1}	6.46×10^{-1}
8	4.48×10^{-1}	4.48×10^{-1}	4.48×10^{-1}
9	2.47×10^{-1}	2.48×10^{-1}	2.48×10^{-1}
10	1.90×10^{-1}	1.91×10^{-1}	1.91×10^{-1}

Tabla 7.
Comparación para el controlador 2

Entradas		Laboratorio Virtual		MATLAB		SciLab	
Entrada1	Entrada2	Salida1	Salida2	Salida1	Salida2	Salida1	Salida2
-1	-10	-1	-100	0	0	-1	-100
-1	-5	0	-100	0	0	0	-100
-1	0	0	-100	0	0	0	-100
-1	5	0	-100	0	0	0	-100
-1	10	0	-100	0	0	0	-100
-0.5	-10	1	-100	1	-100	1	-100
-0.5	-5	1	-100	1	-100	1	-100
-0.5	0	0	-100	0	0	0	-100
-0.5	5	-0.5	-100	0	0	-0.5	-100
-0.5	10	-0.5	-100	0	0	-0.5	-100
0	-10	1	-100	1	-100	1	-100
0	-5	1	-100	1	-100	1	-100
0	0	0	-100	0	0	0	-100
0	5	-0.5	-100	0	0	-0.5	-100
0	10	-1	-100	0	0	-1	-100
0.5	-10	1	-100	1	-100	1	-100
0.5	-5	1	-100	1	-100	1	-100
0.5	0	0	-100	0	0	0	-100
0.5	5	-0.5	-100	0	0	-0.5	-100
0.5	10	0.5	-50	0.5	0	0.5	-50
1	-10	1	-100	1	0	1	-100
1	-5	1	-100	1	0	1	-100
1	0	0	-100	0	0	0	-100
1	5	1	0	1	0	1	0
1	10	1	0	1	0	1	0

Para el controlador 1 la Tabla 6 muestra que la diferencia entre respuestas es despreciable, la diferencia porcentual promedio entre el Laboratorio Virtual y MATLAB/SciLab fue en ambos casos de 1.10×10^{-1} . Las gráficas de respuesta del controlador 1 para las tres herramientas se pueden observar en la Figura 20. En el caso del controlador 2, MATLAB presenta resultados dispares respecto al Laboratorio Virtual y Scilab, esto se debe a que MATLAB no aplica la definición del método de defuzzificación del primero a la izquierda como debería ser y esto provoca que sus resultados sean erróneos e impredecibles, en consecuencia, no se realizó cálculo de diferencia dado que carece de sentido con MATLAB por lo ya descrito, y con SciLab

la diferencia es de cero. Las superficies de respuesta para el controlador 2 se pueden observar en las Figuras 21 a 23.

En términos de la interfaz para el diseño de controladores difusos las tres herramientas presentan un enfoque similar con solo pequeñas variaciones sin mucho impacto, por el lado de funciones disponibles, las variaciones tampoco son grandes, aunque MATLAB y SciLab permiten diseñar controladores Larsen y Tagaki-Sugeno-Khan, estos se encuentran fuera del alcance de este trabajo. Algo a resaltar es la posibilidad del Laboratorio Virtual de asignar métodos de defuzzificación individuales a cada salida, además, permite asignar ponderaciones individuales a cada salida de cada regla, esta función es gracias a las capacidades de Scikit-Fuzzy.

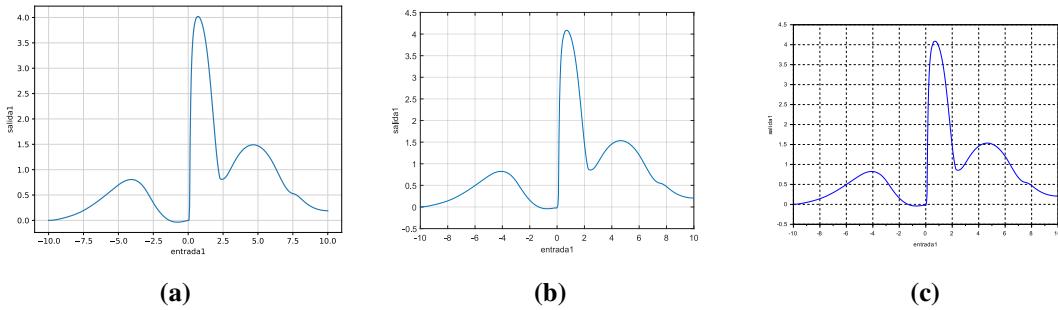


Figura 20. Respuesta del controlador difuso 1. (a) Laboratorio Virtual, (b) MATLAB, (c) SciLab.
Fuente: Elaboración propia.

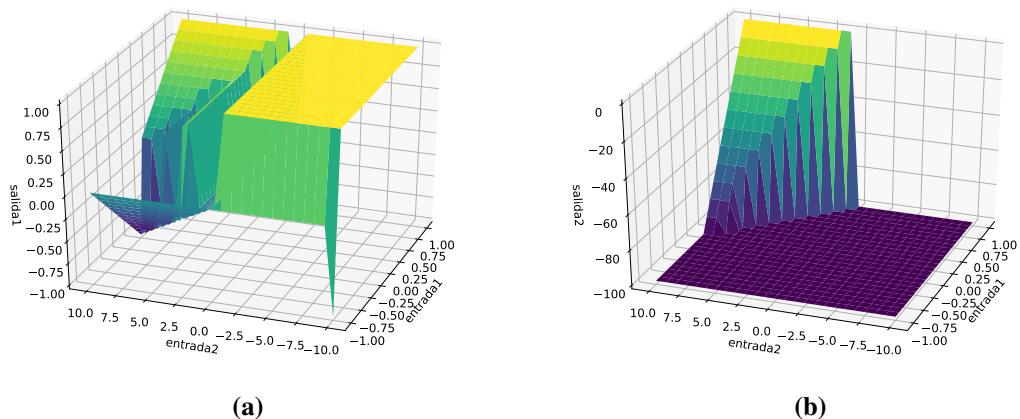


Figura 21. Respuesta del controlador difuso 2 - Laboratorio Virtual. (a) Salida 1, (b) Salida 2.
Fuente: Elaboración propia.

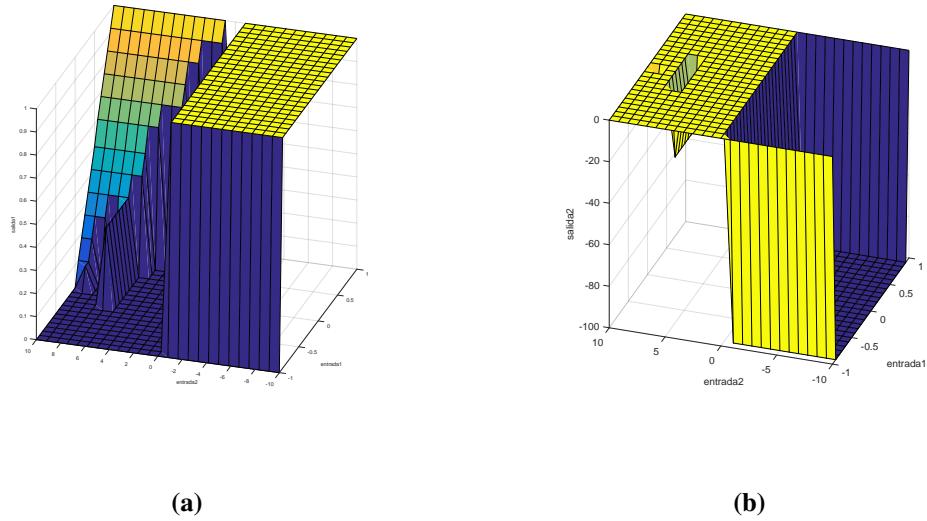


Figura 22. Respuesta del controlador difuso 2 - MATLAB. (a) Salida 1, (b) Salida 2. Se aprecian los efectos de una implementación errónea del SOM por parte de MATLAB. Fuente: Elaboración propia.

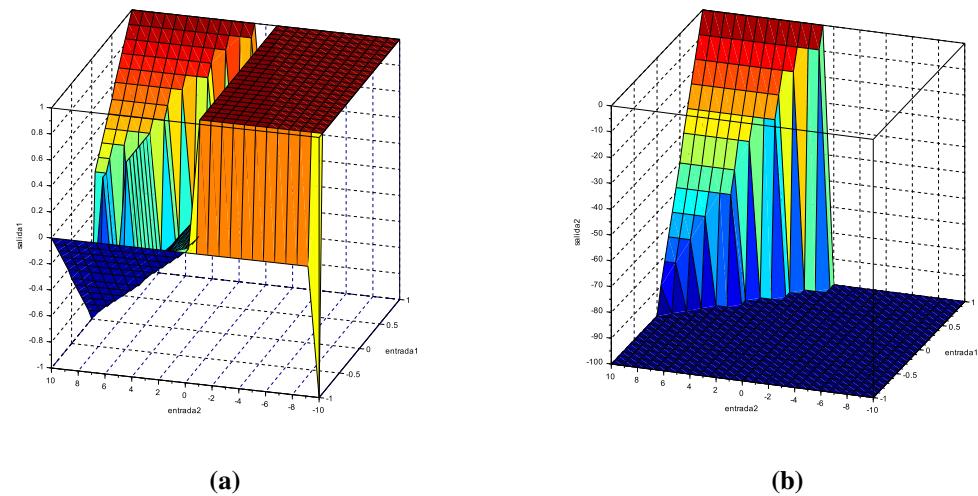


Figura 23. Respuesta del controlador difuso 2 - SciLab. (a) Salida 1, (b) Salida 2. Fuente: Elaboración propia.

Simulación de sistemas de control

Para la comparación de simulación de sistemas de control se seleccionó el sistema número uno (1) de la Tabla 3, los controladores difusos utilizados en cada caso son inadecuados para el control de procesos, i.e., no fueron diseñados con la finalidad de obtener una buena respuesta, la premisa para esta decisión se basa en que la respuesta obtenida debe ser parecida a la obtenida con MATLAB o SciLab, si dicha respuesta es oscilatoria se espera que el Laboratorio Virtual también responda del mismo modo, por tanto, la calidad de los controladores se consideró irrelevante. Los archivos FIS para los controladores utilizados se pueden observar en el [Anexo N].

Los parámetros bajo los que se realizaron las simulaciones como el setpoint, activación o no del filtro de la derivada y/o bloques adicionales, fueron seleccionados de forma arbitraria con la finalidad de probar distintas funciones y obtener respuestas variadas, a su vez, se alternó el uso de distintos solvers para cada herramienta con el fin de observar como afectaban la precisión. A continuación, se presentan las tablas con los parámetros de cada simulación para cada herramienta.

Tabla 8.
Parámetros para el Laboratorio Virtual en tiempo continuo

Controlador	solver	rtol	atol	Otros
P	DOPRI5(4)	1×10^{-4}	1×10^{-6}	$K_p = 1$
PI	DOPRI5(4)	1×10^{-4}	1×10^{-6}	$K_p = K_i = 1$, delay=2s
PID	RK2	1×10^{-6}	1×10^{-6}	$K_p = K_i = K_d = 1$, con filtro
	RK2	1×10^{-6}	1×10^{-6}	$K_p = K_i = K_d = 1$, sin filtro
PID difuso	SSPRK3	1×10^{-6}	1×10^{-6}	Con filtro
	SSPRK3	1×10^{-6}	1×10^{-6}	Sin filtro
	DOPRI5(4)	1×10^{-6}	1×10^{-6}	Sin filtro
PID difuso	DOPRI5(4)	1×10^{-4}	1×10^{-6}	Setpoint variable, con filtro
PI difuso	DOPRI5(4)	1×10^{-4}	1×10^{-7}	Sin filtro
	RK2	1×10^{-6}	1×10^{-6}	Sin filtro

Controlador	solver	rtol	atol	Otros
PI difuso	DOPRI5(4)	1×10^{-4}	1×10^{-6}	Delay=0.5s, con filtro
PI difuso + D	RK4	1×10^{-5}	1×10^{-6}	$K_d = 0.8$, setpoint variable, sin filtro
PD difuso + I	RK2	1×10^{-6}	1×10^{-6}	$K_i = 1$, setpoint variable, sin filtro
PDG difuso	Fehlberg4(5)	1×10^{-4}	1×10^{-6}	Con filtro
PDG difuso	Ralston3	1×10^{-5}	1×10^{-6}	Saturador [0, 20], sin filtro
PID más difuso	RK2	1×10^{-5}	1×10^{-6}	$K_p = 0.2, K_i = K_d = 1$

Tabla 9.
Parámetros para MATLAB en tiempo continuo

Controlador	solver	rtol	atol	Otros
P	Auto(ode45)	Auto	Auto	$K_p = 1$
PI	Auto(ode45)	Auto	Auto	$K_p = K_i = 1$, delay=2s
PID	Auto(ode45)	Auto	Auto	$K_p = K_i = K_d = 1$
PID difuso	Auto(ode45)	Auto	Auto	-
	ode23tb	1×10^{-6}	1×10^{-6}	-
PID difuso	Auto(ode45)	Auto	Auto	Setpoint variable
PI difuso	ode23tb	1×10^{-6}	1×10^{-6}	-
PI difuso	Auto(ode45)	Auto	Auto	Delay=0.5s
PI difuso + D	Auto(ode45)	Auto	Auto	$K_d = 0.8$, setpoint variable
PD difuso + I	Auto(ode45)	Auto	Auto	$K_i = 1$, setpoint variable
PDG difuso	Auto(ode45)	Auto	Auto	-
PDG difuso	ode23tb	1×10^{-6}	1×10^{-6}	Saturador [0, 20]
PID más difuso	Auto(ode45)	Auto	Auto	$K_p = 0.2, K_i = K_d = 1$

Tabla 10.
Parámetros para SciLab en tiempo continuo

Controlador	solver	rtol	atol	Otros
P	BDF-Newton	1×10^{-6}	1×10^{-6}	$K_p = 1$
PI	BDF-Newton	1×10^{-6}	1×10^{-6}	$K_p = K_i = 1$, delay=2s
PID	BDF-Newton	1×10^{-6}	1×10^{-6}	$K_p = K_i = K_d = 1$
PID difuso	BDF-Newton	1×10^{-6}	1×10^{-6}	-
	LSODAR	1×10^{-6}	1×10^{-6}	-

Controlador	solver	rtol	atol	Otros
PID difuso	BDF-Newton	1×10^{-6}	1×10^{-6}	Setpoint variable
PI difuso	LSODAR	1×10^{-6}	1×10^{-6}	-
PI difuso	BDF-Newton	1×10^{-6}	1×10^{-6}	Delay=0.5s
PI difuso + D	BDF-Newton	1×10^{-6}	1×10^{-6}	$K_d = 0.8$, setpoint variable
PD difuso + I	BDF-Newton	1×10^{-6}	1×10^{-6}	$K_i = 1$, setpoint variable
PDG difuso	BDF-Newton	1×10^{-6}	1×10^{-6}	-
PDG difuso	LSODAR	1×10^{-6}	1×10^{-6}	Saturador [0, 20]
PID más difuso	BDF-Newton	1×10^{-6}	1×10^{-6}	$K_p = 0.2, K_i = K_d = 1$

A continuación, se observan las respuestas de las simulaciones que presentaron resultados notables, el resto de figuras se pueden observar en el [Anexo O]. Las gráficas de la simulación para el controlador PID difuso se pueden observar en la Figura 24. La simulación generó resultados interesantes dependiendo de los parámetros y la herramienta empleada, en el caso de MATLAB, se obtuvo una respuesta con oscilaciones que difiere en gran medida del resto al utilizar parámetros por defecto del solver, a su vez, al cambiar el solver a ode23tb para problemas stiff y fijar un valor para la tolerancia relativa y absoluta se corrigió la respuesta, obteniendo así, una salida muy parecida al resto.

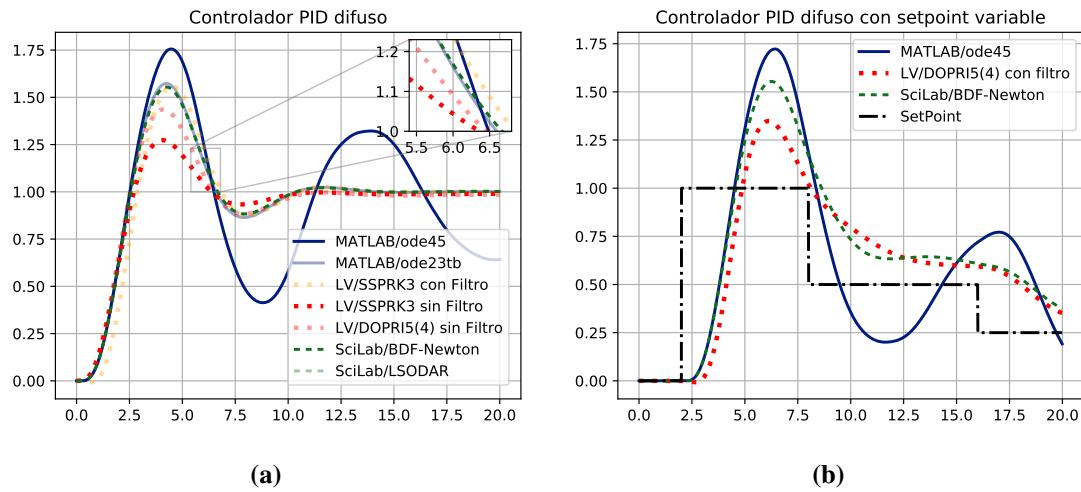


Figura 24. Simulación del controlador PID difuso en tiempo continuo. (a) PID difuso con setpoint básico y múltiples solvers, (b) PID difuso con setpoint variable. Fuente: Elaboración propia.

En el caso del Laboratorio Virtual, es claro que ninguna de sus respuestas es la más precisa, pero mantiene la forma de la respuesta para todos los métodos, por otro lado, también se observan los efectos del filtro para la derivada, su uso genera una mejor respuesta debido a la alta dependencia de las derivadas para este controlador, lo anterior es notable al comparar la respuesta del método SSPRK3 con filtro y sin filtro, no obstante, también se observa que para la simulación presentada en la Figura 24b DOPRI5(4) con el filtro presenta una respuesta menos precisa que sin el filtro.

Los solvers de SciLab son los que presentan la mejor respuesta, de hecho, ambas respuestas se superponen a simple vista, esto es debido a que BDF-Newton es acto para problemas stiff y LSODAR posee una detección automática entre problemas stiff y no stiff que le permite cambiar en cada paso entre BDF para problemas stiff y Adam para problemas no stiff.

Otras simulaciones que presentaron resultados interesantes fueron las asociadas al controlador PI difuso, las cuales se pueden ver en la Figura 25, el Laboratorio Virtual presentó diferencias menores con respecto a MATLAB y SciLab dependiendo del solver utilizado, pero por otro lado, tal como se observa en la Figura 25b, hacer uso del filtro provoca un atraso en la respuesta indeseado.

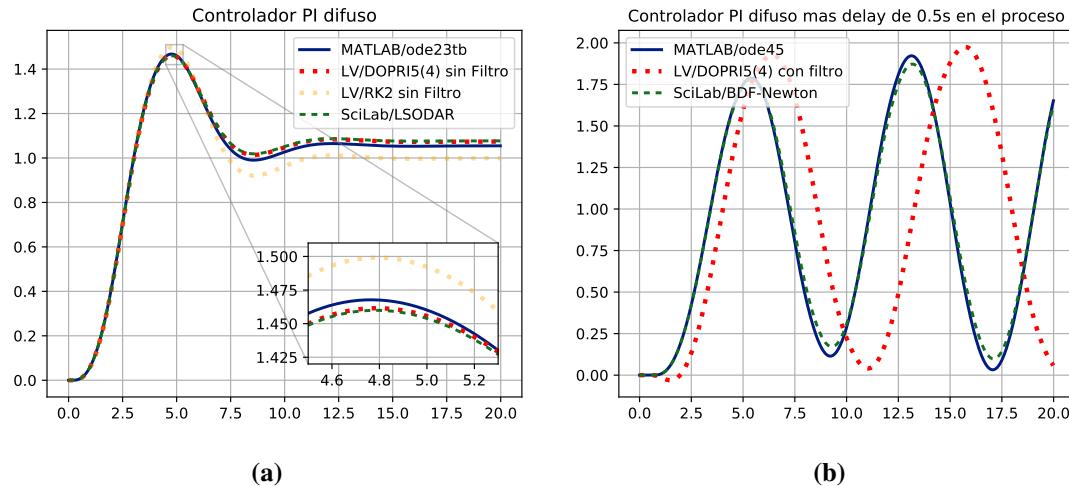


Figura 25. Simulación del controlador PI difuso en tiempo continuo. (a) PI difuso con setpoint básico, (b) PI difuso con 0.5s de delay en el proceso. Fuente: Elaboración propia.

Hay que resaltar que para la simulación de la Figura 24b se realizaron otras pruebas sin filtro y con distintos solvers y no se consiguió obtener una respuesta parecida a la de MATLAB y SciLab, el motivo de esta diferencia es aún desconocido. El resto de simulaciones presentaron diferencias entre las distintas herramientas dentro de lo razonable, por otro lado, los tiempos de ejecución fueron notablemente altos para el caso del Laboratorio Virtual a la hora de utilizar controladores difusos. Los tiempos de ejecución para tiempo continuo se pueden observar en la Tabla 11.

Tabla 11.

Tiempos de ejecución para la función de simulación de sistemas de control en tiempo continuo

Controlador	Tiempo		
	Laboratorio Virtual	MATLAB	SciLab
P	0.530	0.197	2.259
PI	0.666	0.483	2.007
PID	0.672	0.381	1.907
	0.689		
PID difuso	14.430	1.019	11.334
	13.198	1.055	5.972
	9.892	-	-
PID difuso	10.007	1.124	11.018
PI difuso	4.376	0.688	6.177
	8.880		
PI difuso	5.947	0.809	10.825
PI difuso + D	7.995	0.962	12.513
PD difuso + I	19.329	0.687	9.556
PDG difuso	11.079	1.641	8.343
PDG difuso	17.684	1.327	7.486
PID más difuso	4.515	0.837	3.959

Para la simulación en tiempo discreto se seleccionaron arbitrariamente métodos de discretización y periodos de muestreo, a su vez, se realizó una simulación por cada tipo de controlador estableciendo los mismos parámetros extras que para tiempo continuo en aquellos casos que se presentaran, i.g., setpoint avanzado, con lo cual se tendrá a su vez, una comparación directa con sus contrapartes en tiempo continuo. Motivado a que en este caso no hay ningún tipo de solver involucrado los parámetros mostrados en la Tabla 12 son generales para las tres herramientas.

Tabla 12.

Parámetros de simulación en tiempo discreto

Controlador	Método	Periodo de muestreo	Otros
P	ZOH	0.1	$K_p = 1$
PI	Tustin	0.1	$K_p = K_i = 1$, delay=2s
PID	Tustin	0.05	$K_p = K_i = K_d = 1$
PID difuso	Tustin	0.01	Setpoint variable
PI difuso	backward diff	0.05	Delay=0.5s
PI difuso + D	Euler	0.08	$K_d = 0.8$, setpoint variable
PD difuso + I	Impulse	0.1	$K_i = 1$, setpoint variable
PDG difuso	ZOH	0.1	Saturador [0, 20]
PID más difuso	ZOH	0.01	$K_p = 0.2, K_i = K_d = 1$

Las simulaciones en tiempo discreto resultaron, como era de esperarse, mejor que en tiempo continuo, solo dos simulaciones generaron respuestas distintas las cuales fueron para PDG y PI difuso, ambas respuestas se pueden observar en las Figuras 26a y 26b respectivamente. En el caso del programador de ganancias se observa que las tres herramientas generaron respuestas distintas, estando el Laboratorio Virtual entre ambas con un parecido mayor a la dada por MATLAB que a la de SciLab.

Como ya se mencionó antes, la simulación de PI difuso con delay de 0.5s genera grandes diferencias con respecto a MATLAB y SciLab y cuyo motivo es aún desconocido, el tipo de tiempo (continuo o discreto) presentan las mismas discrepancias, por tanto, no proviene de allí el problema.

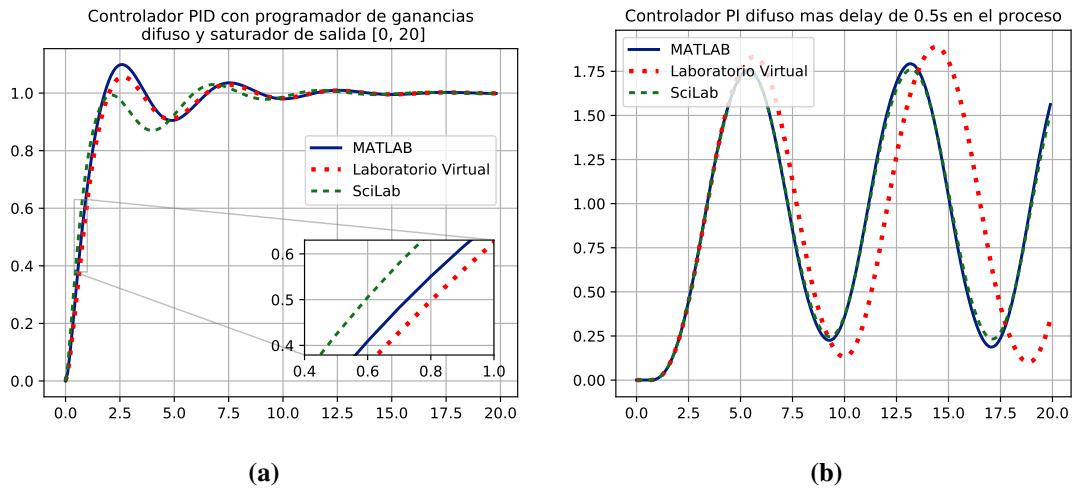


Figura 26. Respuesta de controladores en tiempo discreto. (a) Programador de ganancias difuso con saturador [0,20], (b) PI difuso con 0.5s de delay en el proceso. Fuente: Elaboración propia.

Continuando con la idea, en una prueba independiente se graficó la superficie de respuesta del controlador PI difuso en las tres herramientas y en los tres casos la superficie fue la misma, adicionalmente, se utilizó un mayor número de decimales para definir las matrices de estado sin ningún cambio. El método utilizado para integrar la señal de error y/o señal de salida del controlador tampoco ha presentado problemas en el resto de simulaciones con los distintos controladores, la implementación de Delay funciona perfectamente tal como se observa en controlador PI de la Figura 53 en el [Anexo O], es así que se repite lo ya mencionado, el origen de la diferencia de respuestas para el controlador PI difuso con delay de 0.5s es desconocido. Los tiempos de ejecución para tiempo discreto se pueden observar en la Tabla 13.

Tabla 13.

Tiempos de ejecución para la función de simulación de sistemas de control en tiempo discreto

Controlador	Tiempo		
	Laboratorio Virtual	MATLAB	SciLab
P	0.499	0.808	1.854
PI	0.510	0.591	2.083
PID	0.569	0.526	2.019
PID difuso	14.742	2.521	4.016

Controlador	Tiempo		
	Laboratorio Virtual	MATLAB	SciLab
PI difuso	2.290	0.489	2.923
PI difuso + D	2.085	0.541	3.030
PD difuso + I	1.828	0.561	2.997
PDG difuso	3.135	0.687	3.112
PID más difuso	6.740	0.676	2.755

Los tiempos de ejecución en tiempo discreto son en algunos casos menores para el Laboratorio Virtual al compararlos con SciLab, de forma general se puede afirmar que MATLAB es quien ejecuta las simulaciones en el menor tiempo.

CAPÍTULO V

CONCLUSIONES Y RECOMENDACIONES

A continuación, se presentan las conclusiones derivadas del trabajo realizado así como las recomendaciones que se consideraron pueden mejorar al Laboratorio Virtual por medio trabajos futuros alrededor del mismo.

Conclusiones

1. Todas las funciones implementadas presentaron, en su mayoría, resultados cercanos dentro de una tolerancia a las herramientas alternas que se consideraron, logrando así demostrar que es posible realizar un Laboratorio Virtual de sistemas de control utilizando software libre, particularmente haciendo uso del lenguaje de programación Python en conjunto con un set de librerías de terceros.
2. El Laboratorio Virtual aunque potente, se encuentra aún limitado en funciones respecto a otras herramientas como MATLAB y SciLab, esto es esperable ante el alcance planteado en este trabajo y se considera que herramientas como Simulink y Xcos para la simulación de sistemas de control son aun superiores, por otro lado, el Laboratorio Virtual posee funciones no disponibles en SciLab para el análisis de sistema de control como los distintos métodos de discretización y la implementación de atrasos en el proceso (Delay) y entonación automática de controladores PID.
3. La librería Scikit-Fuzzy cumplió con las expectativas para poder implementar el diseño y simulación de sistemas de control, no obstante, carece de la optimización necesaria para poder realizar simulaciones en lotes, i.e., posee tiempos de ejecución altos para procesar una simulación con múltiples muestras de un tiempo total dado. Lo anterior se entiende dado que Scikit-Fuzzy está pensada para diseñar controladores

difusos que serán implementados en microcontroladores, no obstante, el Laboratorio Virtual logra mejorar los tiempos de ejecución de SciLab en algunas situaciones.

4. El Laboratorio Virtual logra ofrecer las herramientas necesarias para realizar análisis, diseño y simulación de controladores clásicos y difusos de forma simple y rápida sin tener que escribir una línea de código gracias a la interfaz de usuario implementada, la cual, a su vez, puede ser expandida de forma sencilla para implementar más opciones a las funciones ya implementadas en el Laboratorio Virtual.
5. El Laboratorio Virtual posee un gran potencial como herramienta para ingenieros en el área de los sistemas de control y posee un diseño tal que permite seguir siendo desarrollada en esta área, adicionalmente, Python ofrece una gran variedad de librerías externas que pueden emplearse para expandir el Laboratorio Virtual en otros ámbitos sin la necesidad de tocar las funciones ya implementadas.

Recomendaciones

1. Re implementar el sistema de inferencia difuso de Scikit-Fuzzy asegurándose de no modificar los sistemas de diseño de controladores, lo anterior permitiría poder procesar de forma más rápida grandes cantidades de datos con la finalidad de acortar los tiempos de simulación en la función de simulación de sistemas de control sin que, a su vez, se tenga que volver a codificar el archivo “Handler” correspondiente.
2. Implementar la posibilidad de diseñar y simular los sistemas de inferencia Takagi-Sugeno-Khan y Tsukamoto, los cuales son más acordes y utilizados en el área de los sistemas de control, adicionalmente, se puede incluir el controlador Larsen.
3. Añadir una nueva función que incorpore un sistema SCADA con la finalidad de poner a prueba los controladores diseñados de forma experimental, para ello se puede utilizar las distintas librerías existentes que incorporan distintos protocolos de comunicación, i.g., modBuss.

4. Implementar algoritmos de resolución de sistema de ecuaciones diferenciales para problemas stiff como BDF y LSODAR que ofrecen mejores precisiones que las obtenidas con los métodos explícitos y embebidos.
5. Implementar una consola que permita codificar Python dentro del Laboratorio Virtual y que se pueda distribuir utilizando un único archivo ejecutable, de este modo se obtendría una herramienta más completa y potente que permitiría realizar calculo numérico y simbólico, graficacion y codificación general.

REFERENCIAS

- APCO-Inc. (s.f.). *Open Loop Tuning Rules Based on approximate process models*. Recuperado el 24 de septiembre de 2019, desde <http://educyclopedia.karadimov.info/library/pidtune2.pdf>
- Åström, K. (2002). Control system design lecture notes for ME 155a. *Department of Mechanical and Environmental Engineering, University of California Santa Barbara*. Recuperado el 24 de septiembre de 2019, desde http://orion-voyage.com.ua/userfiles/pdf/astrom_-_control_system_design_1288292182.pdf
- Butcher, J. (1964). On Runge-Kutta processes of high order. *Journal of the Australian Mathematical Society*, 4(2), 179-194. Recuperado el 8 de octubre de 2019, desde https://www.cambridge.org/core/services/aop-cambridge-core/content/view/40DFE501CAB781C9AAE1439B6B8F481A/S1446788700023387a.pdf/on_rungekutta_processes_of_high_order.pdf
- Casallas, R., Chacón, R. y Rivera, F. P. (2005). Desarrollo básico de un Laboratorio Virtual de Control de Procesos basado en Internet. *Acción pedagógica*, 14(1), 58-65. Recuperado el 16 de junio de 2019, desde <https://dialnet.unirioja.es/descarga/articulo/2969002.pdf>
- Congo, J. W. (2018). *Aplicaciones del software libre Python para prácticas de laboratorio aplicado a la asignatura de tratamiento digital de señales de la Universidad Tecnológica Israel* (Tesis de pregrado, Universidad Tecnológica Israel, Ecuador). Recuperado el 9 de junio de 2019, desde <http://157.100.241.244/bitstream/47000/1626/1/UISRAEL-EC-ELDT-378.242-2018-042.pdf>
- Control Systems Library for Python. GitHub. Recuperado el 1 de junio de 2019, desde <https://python-control.readthedocs.io/en/0.8.2/>
- Creus, A. (2010). *Instrumentación Industrial* (8.^a ed.). Barcelona, España: Alfaomega.
- Dorf, R. y Bishop, R. (2011). *Modern control systems* (12.^a ed.). New Jersey, United States: Prentice Hall.

Dubs de Moya, R. (2002). El Proyecto Factible: una modalidad de investigación. *Sapiens. Revista Universitaria de Investigación*, 3(2). Recuperado el 15 de junio de 2019, desde <http://www.redalyc.org/pdf/410/41030203.pdf>

Fernández-Cantí, R. (2013). Tema 4. Control digital. En *Sistemas Electrónicos de Control*. Recuperado el 8 de octubre de 2019, desde https://ocw.upc.edu/sites/all/modules/ocw/estadistiques/download.php?file=11608/2011/1/53985/sec_tema_4_control_digital_1314a_ocw-5203.pdf

Gómez, J. C. (2009). Toolbox didáctico para el diseño y análisis de sistemas de control lineal. *Revista Educación en Ingeniería*, 4(8), 155-169. Recuperado el 13 de junio de 2019, desde <http://www.educacioningenieria.org/index.php/edi/article/download/96/86>

Hairer, E., Nørsett, S. P. y Wanner, G. (1993). *Solving ordinary differential equations I, Nonstiff problems* (2.^a ed. rev.). Berlin, Alemania: Springer-Vlg.

Haugen, F. (2005). Discrete-time signals and systems. Recuperado el 18 de septiembre de 2019, desde http://techteach.no/publications/discretetime_signals_systems/discrete.pdf

Hernández, R., Fernández, C. y Lucio, M. d. P., Baptista. (2010). *Metodología de la investigación* (5.^a ed.). México: McGraw-hill.

Hori, N., Cormier, R. y Kanai, K. (1992). Matched pole-zero discrete-time models. *Control Theory and Applications, IEE Proceedings D*, 139(3), 273-278. Recuperado el 8 de octubre de 2019, desde https://www.researchgate.net/publication/3360909_Matched_pole-zero_discrete-time_models

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90-95. Recuperado el 1 de junio de 2019, desde <https://ieeexplore.ieee.org/document/4160265>

Hurtado de Barrera, J. (Ed.). (2010). *Guía para la comprensión holística de la ciencia*. Caracas, Venezuela: Fundación Sypal.

- Jones, E., Oliphant, T., Peterson, P. et al. (2001). SciPy: Open source scientific tools for Python. Recuperado el 1 de junio de 2019, desde <https://www.scipy.org/scipylib/index.html>
- Kuo, B. C. (1996). *Sistemas de control automático* (7.^a ed.). México: Prentice Hall Hispanoamericana.
- Maloney, T. J. (2006). *Electrónica industrial moderna* (5.^a ed.). México: Pearson Educación.
- Martinez, H. (1997). *Métodos Numéricos*. México: Tecnológico de Monterrey. Recuperado el 18 de septiembre de 2019, desde <http://www.mty.itesm.mx/dtie/deptos/cb/cb00854-1/Apuntes/HMA/MN06fED.pdf>
- Mata, N. C. (1999). *Fundamentos prácticos para el control de procesos*. Caracas, Venezuela: MCL CONTROL.
- Matute, A. y Bernal, W. (2017). Técnicas de lógica difusa en ingeniería de control. *Ciencia, Innovación y Tecnología*, 3, 125-134. Recuperado el 29 de septiembre de 2019, desde <https://jdc.edu.co/revistas/index.php/rciyt/article/view/81>
- Nilsson, J. W. y Riedel, S. A. (2005). *Circuitos eléctricos* (7.^a ed.). Madrid, España: Pearson Educación.
- Ogata, K. (2003). *Ingeniería de control moderna* (5.^a ed.). Madrid, España: Pearson Educación.
- Oliphant, T. (2006). NumPy: A guide to NumPy. USA: Trelgol Publishing. Recuperado el 31 de mayo de 2019, desde <http://www.numpy.org/>
- Ponce, P. C. (2010). *Inteligencia artificial con aplicaciones a la ingeniería*. México: Alfaomega.
- Riid, A. (2002). *Transparent Fuzzy Systems: modelling and control* (Tesis doctoral, Tallinn Technical University, Tallinn, Estonia). Recuperado el 14 de junio de 2019, desde https://www.researchgate.net/publication/34752921_Transparent_Fuzzy_Systems_Modeling_and_Control

Ritschel, T. (2013). *Numerical Methods For Solution of Differential Equations* (Tesis de pregrado, Technical University of Denmark, Dinamarca). Recuperado el 5 de septiembre de 2019, desde http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/6607/pdf/imm6607.pdf

Rogan, J. y Muñoz, V. (s.f.). PROGRAMACION Y METODOS NUMERICOS. *Departamento de Física, Universidad de Chile*. Recuperado el 18 de septiembre de 2019, desde https://macul.ciencias.uchile.cl/~vmunoz/homepage/cursos/programacion_avanzada/2012/mfm0-09.pdf

Salazar, L. J. (2019). *Diseño de un sistema de riego inteligente para cultivos de hortalizas basado en Fuzzy Logic en la granja la pradera de la Universidad Técnica del Norte* (Tesis de pregrado, Universidad Técnica del Norte, Ecuador). Recuperado el 8 de junio de 2019, desde <http://repositorio.utn.edu.ec/jspui/bitstream/123456789/9137/1/04%5C%20RED%5C%202019%5C%20TRABAJO%5C%20DE%5C%20GRADO.pdf>

Sánchez, J. A. (2003). *Control avanzado de procesos:(teoría y práctica)*. Madrid, España: Ediciones Díaz de Santos.

Smith, C. A. y Corripio, A. B. (1985). *Principles and practice of automatic process control* (2.^a ed.). New York, United States: Wiley New York.

sputnick1124. (2017). Yet Another Python Fuzzy Logic Module. GitHub. Recuperado el 13 de octubre de 2019, desde <https://github.com/sputnick1124/yapfml>

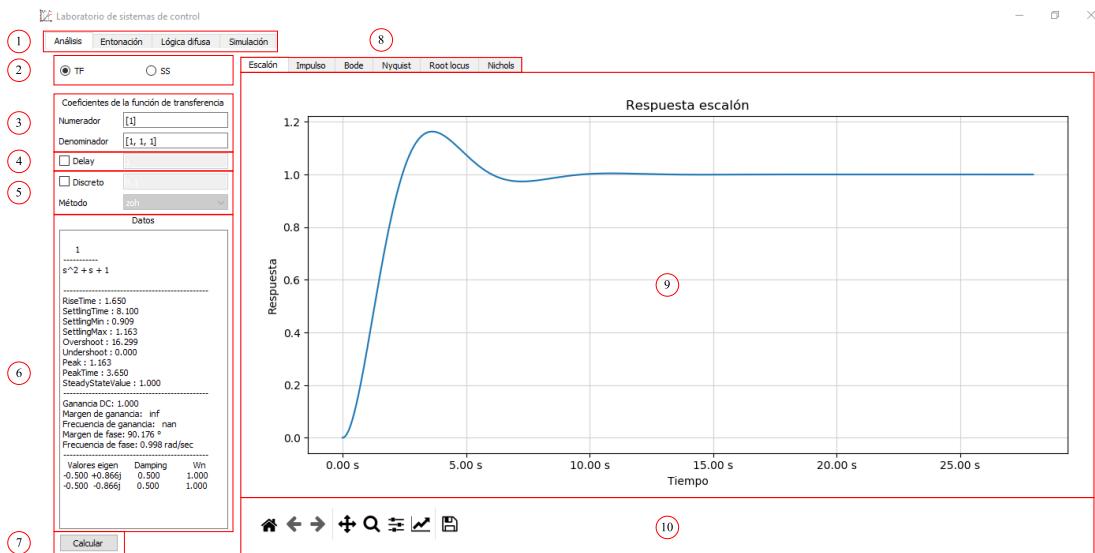
Suárez, M. Á. (2014). *Ciencias e Ingeniería para estudiantes de Python*. Recuperado el 7 de junio de 2019, desde https://www.researchgate.net/publication/313664762_Ciencias_e_Ingenieria_para_estudiantes_de_Python

Van Rossum, G. (2017). El tutorial de Python. Recuperado el 15 de junio de 2019, desde <http://docs.python.org.ar/tutorial/pdfs/TutorialPython3.pdf>

Warner, J., Sexauer, J., twmeggs, S., A. M., Unnikrishnan, A., Castelão, G., . . . Mishra, H. (2016). Scikit-Fuzzy: fuzzy logic library for python. GitHub. Recuperado el 16 de junio de 2019, desde <https://github.com/scikit-fuzzy/scikit-fuzzy>

[Anexo A]

[Interfaz gráfica de la función de análisis de sistemas de control]



(a)

TF SS

Matrices de estado			
A	[[-1, -1], [1, 0]]		
B	[[1], [0]]		
C	[[0, 1]]		
D	[[0]]		

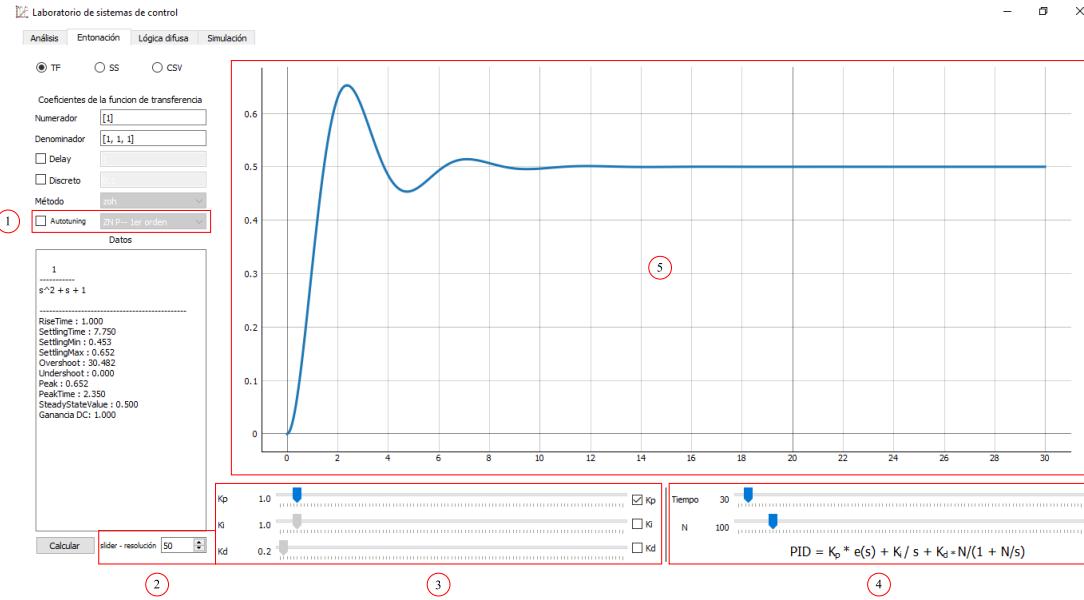
(b)

Figura 27. Interfaz gráfica para el análisis de sistemas de control. (a) Interfaz gráfica general representando al sistema con función de transferencia, (b) Cambios presentes si utiliza la representación con ecuaciones de espacio de estados. Fuente: Elaboración propia.

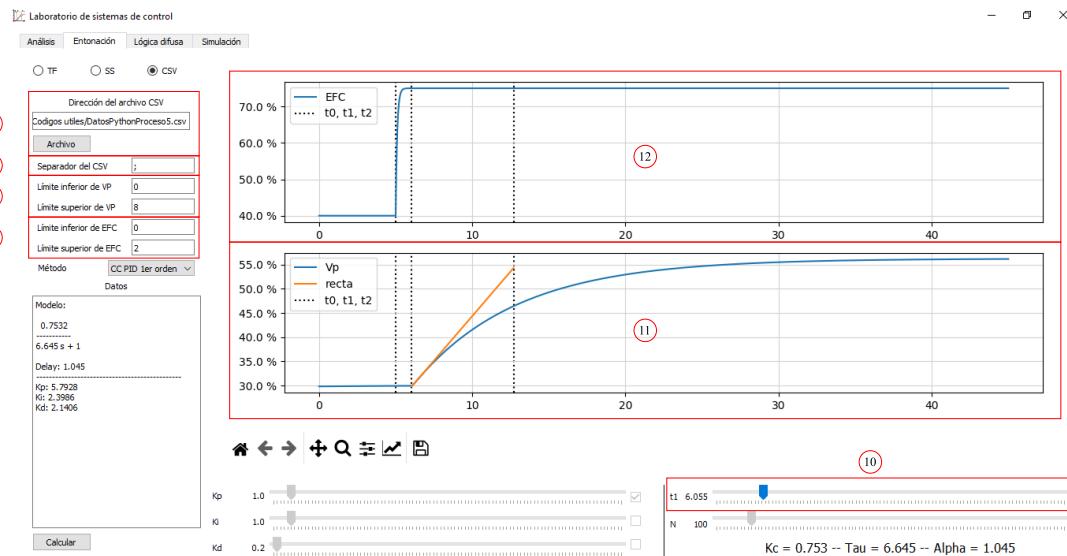
1. Pestañas de funciones
2. Selector de representación
3. Coeficientes de la función de transferencia
4. Agregado de Delay
5. Discretización del proceso
6. Datos del análisis
7. Botón para realizar el análisis
8. Pestañas de gráficas
9. Gráfica con Matplotlib
10. Barra de herramientas de la gráfica
11. Matrices de estados

[Anexo B]

[Interfaz gráfica de la función de entonación de controladores PID]



(a)



(b)

Figura 28. Interfaz gráfica para la entonación de controladores PID. (a) Interfaz gráfica general para la entonación de controladores PID, (b) Interfaz gráfica para la entonación utilizando un archivo CSV. Fuente: Elaboración propia.

1. Función de entonación automática
2. Resolución de los sliders
3. Sliders de ganancias
4. Sliders de tiempo y coeficiente N
5. Gráfica con PyQtGraph
6. Carga del archivo CSV
7. Separador del archivo CSV
8. SPAN de la variable del proceso
9. SPAN de la entrada al proceso (EFC)
10. Slider para ajustar t_1
11. Gráfica de la variable del proceso
12. Gráfica de la entrada al proceso (EFC)

[Anexo C]

[Interfaz gráfica de la función de diseño de controladores difusos]

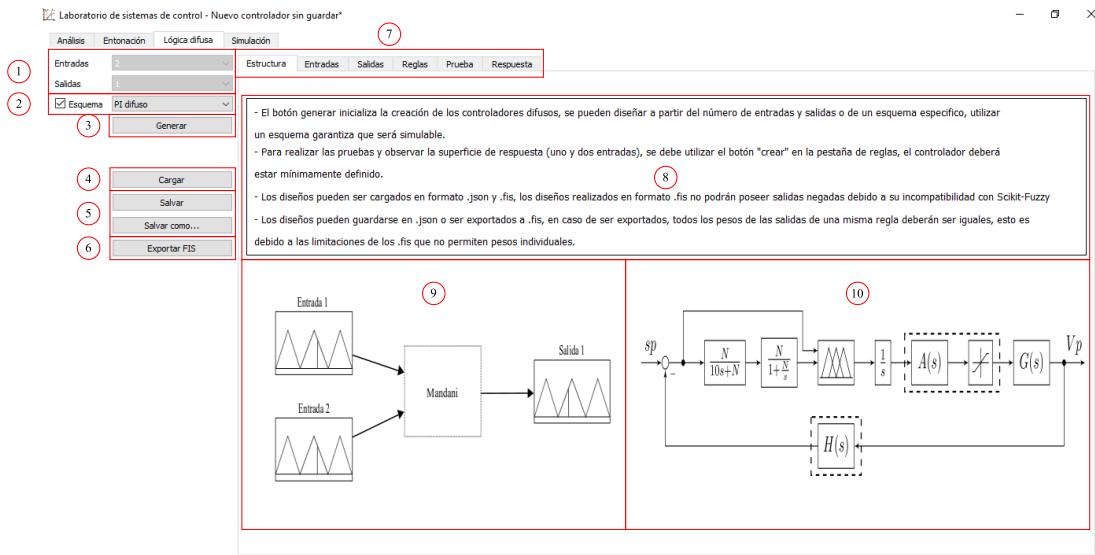
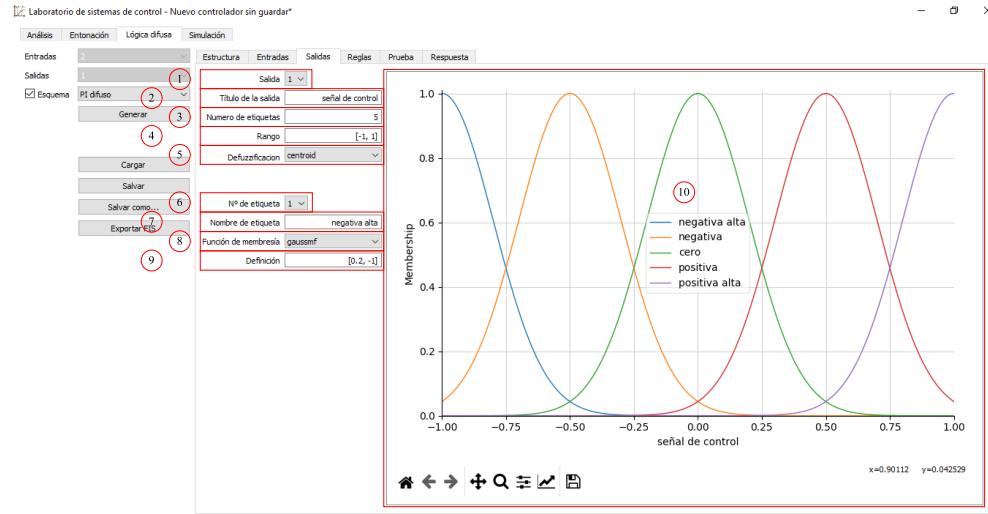
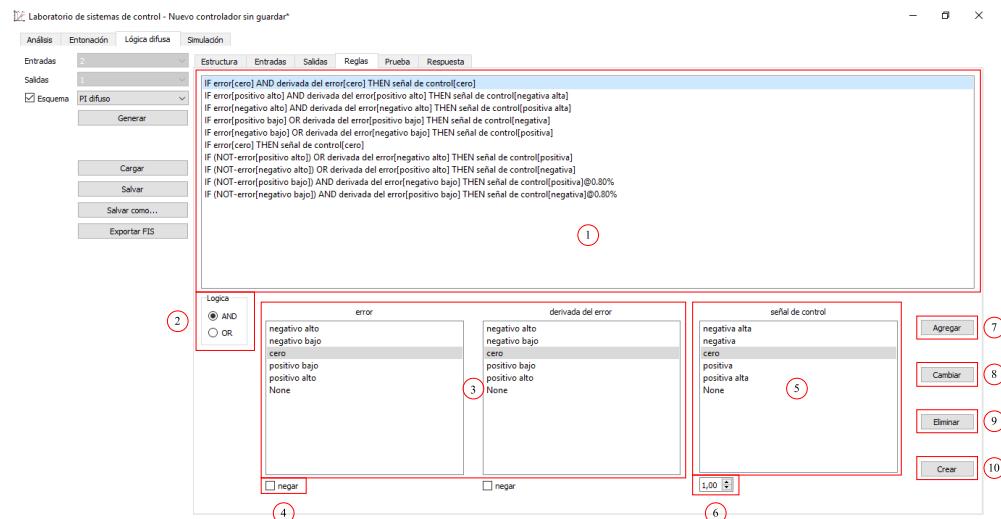


Figura 29. Interfaz gráfica - difusa. Pestaña de estructura. Fuente: Elaboración propia.

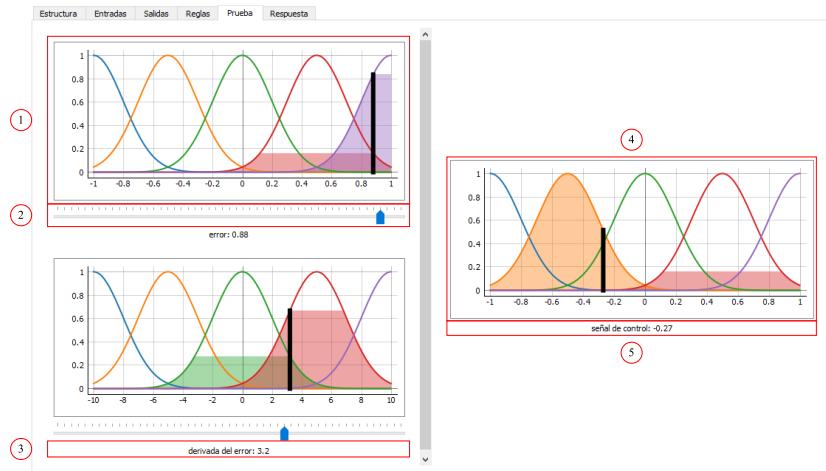
1. Numero de entradas y salidas
2. Selección de esquema de control
3. Botón para iniciar el diseño
4. Botón para cargar un diseño
5. Botones para salvar los diseños
6. Botón para exportar el diseño a FIS
7. Pestañas para el diseño
8. Información general
9. Estructura de entradas y salidas
10. Esquema de control



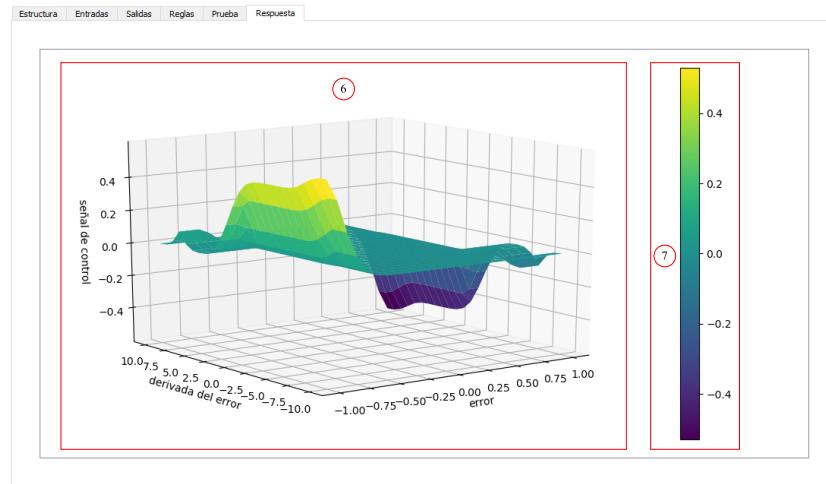
1. Numero de entrada/salida
2. Nombre de la entrada/salida
3. Numero de etiquetas
4. Rango de la entrada/salida
5. Método de defuzzificacion
6. Numero de etiqueta
7. Nombre de la etiqueta
8. Tipo de función de membresía
9. Definición de la función de membresía
10. Gráfica de las funciones de membresía



1. Lista de reglas
2. Lógica de las premisas
3. Etiquetas de las entradas
4. Opción para negar la entrada
5. Etiquetas de las salidas
6. Peso de la salida
7. Botón para agregar una regla
8. Botón para cambiar una regla
9. Botón para eliminar una regla
10. Botón para crear el controlador y realizar pruebas



(a)



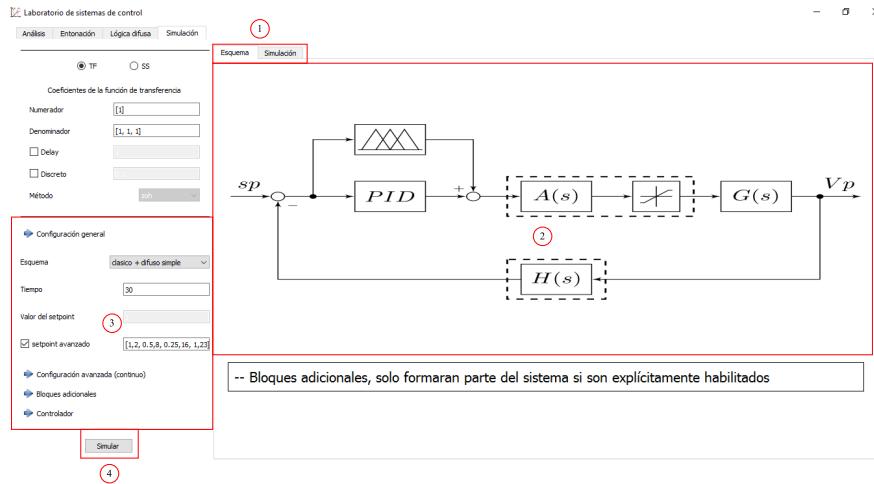
(b)

Figura 32. Interfaz gráfica - difusa. (a) Pestaña para pruebas del controlador, (b) Pestaña para observar la respuesta del controlador, solo visible para controladores con una o dos entradas. Fuente: Elaboración propia.

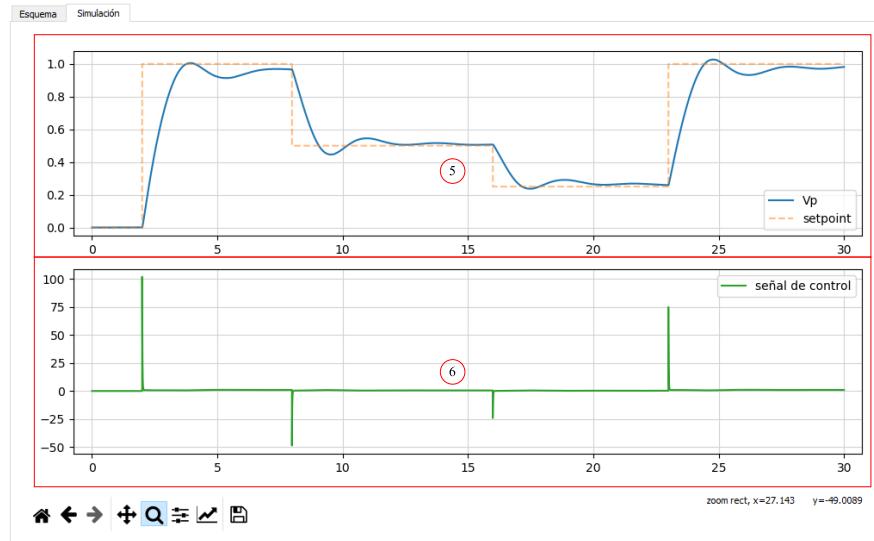
1. Activación de reglas de forma gráfica para las entradas
2. Slider para asignar entrada
3. Valor de entrada
4. Activación de reglas de forma gráfica para las salidas
5. Valor de salida
6. Respuesta del controlador
7. Barra indicadora de altura (para dos entradas)

[Anexo D]

[Interfaz gráfica para la simulación de sistemas de control]



(a)



(b)

Figura 33. Interfaz gráfica para la simulación de sistemas de control. (a) Interfaz gráfica general para la simulación, (b) Pestaña de gráficas de respuesta. Fuente: Elaboración propia.

1. Pestañas de simulación
2. Esquema de control
3. Barras de configuración
4. Botón para simular
5. Gráfica de respuesta del sistema
6. Gráfica de la señal de control

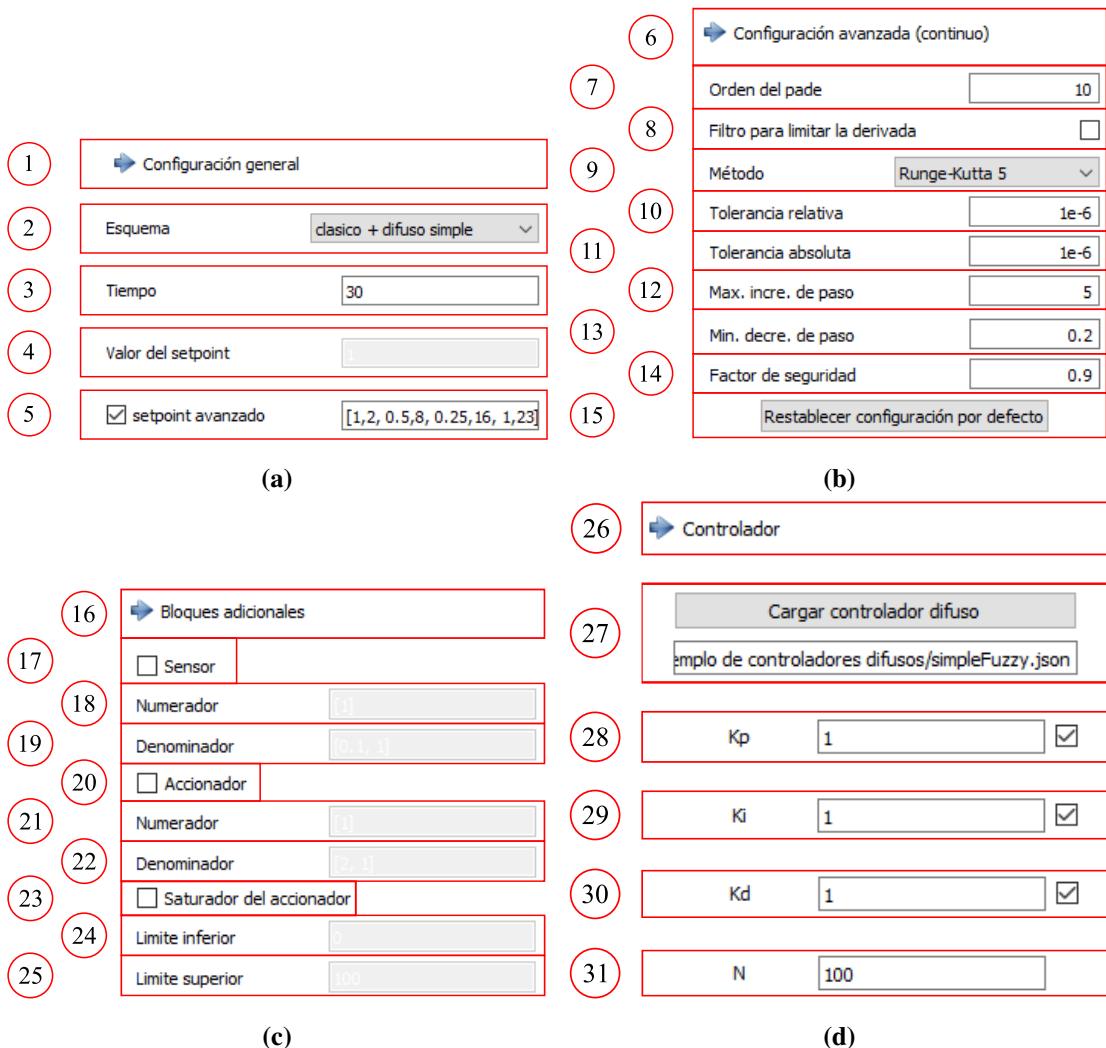


Figura 34. Interfaz gráfica - simulación. (a) Barra de configuración general, (b) Barra de configuración avanzada, (c) Barra de para agregar bloques adicionales, (d) Barra para configurar los controladores. Fuente: Elaboración propia.

1. Barra de configuración general
2. Selección del esquema de control
3. Tiempo de simulación
4. Valor del setpoint
5. Setpoint avanzado (variable)
6. Barra de configuración avanzada
7. Orden del atraso por PADE
8. Activación del filtro para la derivada
9. Selección del método de Runge-Kutta
10. Tolerancia relativa para el paso variable
11. Tolerancia absoluta para el paso variable

- | | |
|---|--|
| 12. Máximo incremento del tamaño de pa- | 21. Numerador del accionador |
| so | 22. Denominador del accionador |
| 13. Mínimo decremento del tamaño de pa- | 23. Activación del saturador |
| so | 24. Límite inferior del saturador |
| 14. Factor de seguridad para el paso varia- | 25. Límite superior del saturador |
| ble | 26. Barra de configuración del controlador |
| 15. Botón para reiniciar la configuración | 27. Controlador difuso |
| 16. Barra de bloques adicionales | 28. Ganancia proporcional del PID |
| 17. Activación del sensor | 29. Ganancia integral del PID |
| 18. Numerador del sensor | 30. Ganancia derivativa del PID |
| 19. Denominador del sensor | 31. Coeficiente N |
| 20. Activación del accionador | |

[Anexo E]

[Código de los márgenes de ganancia y fase]

Código 7. Función para el calculo de los margenes de ganancia y fase.

```
def margenes_ganancias(self, system, mag, phase, omega):
    """
    [Función para obtener el margen de ganancia y el margen de fase]

    :param system: [Representación del sistema]
    :type system: [LTI]
    :param mag: [Magnitud de la respuesta en frecuencia]
    :type mag: [numpyArray]
    :param phase: [Fase de la respuesta en frecuencia]
    :type phase: [numpyArray]
    :param omega: [Frecuencias utilizadas para la respuesta en frecuencia]
    :type omega: [numpyArray]
    """

    gainDb = 20 * np.log10(mag)
    degPhase = phase * 180.0 / np.pi

    # Transformando la fase a : -360 < phase < 360, para +/- 360 phase -> 0
    comp_phase = np.copy(degPhase)
    degPhase = degPhase - (degPhase/360).astype(int) * 360

    # Para evitar la detección de cruces al llevar las fases al rango -360 < phase < 360
    crossHack1 = np.diff(1 * (degPhase > -183) != 0)
    crossHack2 = np.diff(1 * (degPhase > -177) != 0)
    crossHack = ~crossHack1 * ~crossHack2

    # Detección de cruce
    indPhase = np.diff(1 * (gainDb > 0) != 0)
    indGain = np.diff(1 * (degPhase > -180) != 0)
    indGain = indGain * crossHack

    # Calculo de la respuesta en frecuencia para omega = 0 rad/s y pi en caso de ser
    # discreto
    if ctrl.isdtime(system, strict=True):
        zero_freq_response = ctrl.evalfr(system, 1)

        nyquist_freq_response = ctrl.evalfr(system, np.exp(np.pi*1j))
        nyquistMag = np.abs(nyquist_freq_response)
        nyquistPhase = np.angle(nyquist_freq_response)

        if nyquistPhase * 180.0 / np.pi >= 180:
            nyquistPhase = nyquistPhase - 2 * np.pi

        omega = np.insert(omega, len(omega), np.pi/self.dt)
        gainDb = np.insert(gainDb, len(gainDb), 20 * np.log10(nyquistMag))
        degPhase = np.insert(degPhase, len(degPhase), nyquistPhase * 180.0 / np.pi)

    # Verificando "cruce" por -180 grados para la frecuencia de Nyquist
    if np.isclose(nyquistPhase * 180.0 / np.pi, -180):
        indGain = np.insert(indGain, len(indGain), True)
    else:
        indGain = np.insert(indGain, len(indGain), False)
```

```

# Verificando "cruce" por 0 dB para la frecuencia de Nyquist
if np.isclose(20 * np.log10(nyquistMag), 0):
    indPhase = np.insert(indPhase, len(indPhase), True)
else:
    indPhase = np.insert(indPhase, len(indPhase), False)
else:
    zero_freq_response = ctrl.evalfr(system, 0j)

omega = np.insert(omega, 0, 0)
zeroPhase = np.angle(zero_freq_response)
zeroMag = np.abs(zero_freq_response)
if zeroPhase * 180.0 / np.pi >= 180:
    zeroPhase = zeroPhase - 2 * np.pi
gainDb = np.insert(gainDb, 0, 20 * np.log10(zeroMag))
degPhase = np.insert(degPhase, 0, zeroPhase * 180.0 / np.pi)

# Verificando "cruce" por -180 grados para omega = 0 rad/s
if np.isclose(zeroPhase * 180.0 / np.pi, -180):
    indGain = np.insert(indGain, 0, True)
else:
    indGain = np.insert(indGain, 0, False)

# Verificando "cruce" por 0 dB para omega = 0 rad/s
if np.isclose(20 * np.log10(zeroMag), 0):
    indPhase = np.insert(indPhase, 0, True)
else:
    indPhase = np.insert(indPhase, 0, False)

# Marge de ganancia
if len(omega[:-1][indGain]) > 0:
    newGainIndex = np.argmax(np.abs(gainDb[:-1][indGain]))
    omegaGain = omega[:-1][indGain][newGainIndex]
    GainMargin = -gainDb[:-1][indGain][newGainIndex]
else:
    omegaGain = np.nan
    GainMargin = np.infty

# Marge de Fase
if len(omega[:-1][indPhase]) > 0:
    newPhaIndex = min(range(len(degPhase[:-1][indPhase])), key=lambda i: abs(np.abs(degPhase[:-1][indPhase][i]) - 180))
    omegaPhase = omega[:-1][indPhase][newPhaIndex]
    PhaseMargin = 180 + degPhase[:-1][indPhase][newPhaIndex]
else:
    omegaPhase = np.nan
    PhaseMargin = np.infty

return GainMargin, PhaseMargin, omegaGain, omegaPhase

```

[Anexo F]

[Ejemplo de un archivo CSV valido para la entonación]

```
$Date;$Time;VP2;EFC2;SP2
02/22/12;01:21:00.000;0.6312256;5.233645;1.009346
02/22/12;01:21:00.070;0.6312256;5.233645;1.009346
02/22/12;01:21:00.140;0.6312256;5.233645;1.009346
02/22/12;01:21:00.210;0.6312256;5.233645;1.009346
02/22/12;01:21:00.280;0.6312256;5.233645;1.009346
02/22/12;01:21:00.350;0.6312256;5.233645;1.009346
02/22/12;01:21:00.420;0.6312256;5.233645;1.009346
02/22/12;01:21:00.490;0.6312256;5.233645;1.009346
02/22/12;01:21:00.560;0.6312256;5.233645;1.009346
02/22/12;01:21:00.630;0.6312256;5.233645;1.009346
02/22/12;01:21:00.700;0.6312256;5.233645;1.009346
02/22/12;01:21:00.770;0.6312256;5.233645;1.009346
02/22/12;01:21:00.840;0.6313477;5.233645;1.009346
02/22/12;01:21:00.910;0.6313477;5.233645;1.009346
02/22/12;01:21:00.980;0.6313477;5.233645;1.009346
02/22/12;01:21:01.050;0.6313477;5.233645;1.009346
02/22/12;01:21:01.120;0.6313477;5.233645;1.009346
02/22/12;01:21:01.190;0.6313477;5.233645;1.009346
02/22/12;01:21:01.260;0.6313477;5.233645;1.009346
02/22/12;01:21:01.330;0.6313477;5.233645;1.009346
02/22/12;01:21:01.400;0.6313477;5.233645;1.009346
02/22/12;01:21:01.470;0.6313477;5.233645;1.009346
:
:
:
:
:
:
02/22/12;01:21:34.300;1.676147;6.654205;1.009346
02/22/12;01:21:34.370;1.676147;6.654205;1.009346
02/22/12;01:21:34.440;1.676147;6.654205;1.009346
02/22/12;01:21:34.510;1.676147;6.654205;1.009346
02/22/12;01:21:34.580;1.676147;6.654205;1.009346
02/22/12;01:21:34.650;1.676147;6.654205;1.009346
02/22/12;01:21:34.720;1.676147;6.654205;1.009346
02/22/12;01:21:34.790;1.675903;6.654205;1.009346
02/22/12;01:21:34.860;1.675903;6.654205;1.009346
02/22/12;01:21:34.930;1.675903;6.654205;1.009346
02/22/12;01:21:35.000;1.675903;6.654205;1.009346
```

Esta data es válida porque posee tres o más columnas, de las cuales, tres poseen un encabezado con las palabras claves necesarias (VP, EFC y TIME), se está utilizando un separador valido (;), el formato de tiempo es correcto (hh:mm:ss) y la respuesta de la variable del proceso es ascendente.

[Anexo G]

[Código de transformación equivalente entre funciones de membresía]

Código 8. Función para la transformación equivalente entre funciones de membresía.

```
def update_definicionmf(self, old_mf, definicion, new_mf):
    """
    [Función para la transformación equivalente entre funciones de membresía]

    :param old_mf: [Nombre de la antigua función de membresía]
    :type old_mf: [str]
    :param definicion: [Lista con los valores correspondiente a la definición de la antigua
    → función de membresía]
    :type definicion: [list]
    :param new_mf: [Nombre de la nueva función de membresía]
    :type new_mf: [str]
    """

    if old_mf == 'trimf':
        a, b, c = definicion

        if new_mf == 'trimf':
            na, nb, nc = a, b, c
            return [na, nb, nc], '[a, b, c] con: a <= b <= c'

        if new_mf == 'trapmf':
            na, nd = a, c
            nb = (a+b) / 2
            nc = (b+c) / 2
            return [na, nb, nc, nd], '[a, b, c, d] con: a <= b <= c <= d'

        if new_mf == 'gaussmf':
            mean = b
            sigma = (abs(c) + abs(a)) / 8
            return [sigma, mean], '[sigma, media]'

        if new_mf == 'gauss2mf':
            mean1 = (a+b) / 2
            sigma1 = (abs(c) + abs(a)) / 16
            mean2 = (b+c) / 2
            sigma2 = (abs(c) + abs(a)) / 16
            return [sigma1, mean1, sigma2, mean2], '[sigma1, media1, sigma2, media2] con:
            ← media1 <= media2'

        if new_mf == 'smf' or new_mf == 'zmf':
            na = (a+b) / 2
            nb = (b+c) / 2
            return [na, nb], '[a, b] siendo a el inicio del cambio \ny b el final del
            ← cambio'

        if new_mf == 'sigmf':
            nb = b
            nc = 10 / (abs(c) + abs(a))
            return [nc, nb], '[a, b] con:\n a como el ancho del sigmoide, puede ser
            ← negativo\n b el centro del sigmoide'

        if new_mf == 'dsigmf':
            nb1 = (a+b) / 2
```

```

nc1 = 20 / (abs(c) + abs(a))
nb2 = (b+c) / 2
nc2 = -20 / (abs(c) + abs(a))
return [nc1, nb1, nc2, nb2], '[a1, b1, a2, b2] con:\n b1, b2 como los centros
↪ de los sigmoides\n a1, a2 los anchos de los sigmoides, pueden ser
↪ negativos'

if new_mf == 'psigmf':
    nb1 = (a+b) / 2
    nc1 = 20 / (abs(c) + abs(a))
    nb2 = (b+c) / 2
    nc2 = -20 / (abs(c) + abs(a))
    return [nc1, nb1, nc2, nb2], '[a1, b1, a2, b2] con:\n b1, b2 como los centros
↪ de los sigmoides\n a1, a2 los anchos de los sigmoides, pueden ser
↪ negativos'

if new_mf == 'pimf':
    na, nd = a, c
    nb = (a+b) / 2
    nc = (b+c) / 2
    return [na, nb, nc, nd], '[a, b, c, d] con:\n a inicio de la subida y b su final
↪ por el lado izquierdo \n c inicio de la bajada y d su final por el lado
↪ derecho'

if new_mf == 'gbellmf':
    na = abs(c) - abs(a)
    nb = 1 / (a-b)
    nc = b
    return [na, nb, nc], '[a, b, c] con:\n a como el ancho de la campana\n b
↪ pendiente de la campana, puede ser negativa\n c centro de la campana'

if old_mf == 'trapmf':
    a, b, c, d = definicion
    na, nc = a, d
    nb = (c+b) / 2
    return [na, nb, nc], '[a, b, c] con: a <= b <= c'

if old_mf == 'gaussmf':
    b, a = definicion
    na = a - b**4
    nc = a + b**4
    nb = a
    return [na, nb, nc], '[a, b, c] con: a <= b <= c'

if old_mf == 'gauss2mf':
    b, a, d, c = definicion
    na = a - b**4
    nc = c + d**4
    nb = (a+c) / 2
    return [na, nb, nc], '[a, b, c] con: a <= b <= c'

if old_mf == 'smf' or old_mf == 'zmf':
    a, c = definicion
    na = a - (abs(a) + abs(c)) / 4
    nc = c + (abs(a) + abs(c)) / 4
    nb = (a+c) / 2
    return [na, nb, nc], '[a, b, c] con: a <= b <= c'

if old_mf == 'sigmf':
    c, b = definicion
    na = b - abs(c) * 5

```

```

nb = b
nc = b + abs(c) * 5
return [na, nb, nc], '[a, b, c] con: a <= b <= c'

if old_mf == 'dsigmf':
    b, a, d, c = definicion
    na = a - b*1.25
    nb = (a+c) / 2
    nc = c - d*1.25
    return [na, nb, nc], '[a, b, c] con: a <= b <= c'

if old_mf == 'psigmf':
    b, a, d, c = definicion
    na = a - b*1.25
    nb = (a+c) / 2
    nc = c - d*1.25
    return [na, nb, nc], '[a, b, c] con: a <= b <= c'

if old_mf == 'pimf':
    a, b, c, d = definicion
    na, nc = a, d
    nb = (c+b) / 2
    return [na, nb, nc], '[a, b, c] con: a <= b <= c'

if old_mf == 'gbellmf':
    a, b, c = definicion
    na = c - abs(a / c)
    nb = c
    nc = c + abs(a / c)
    return [na, nb, nc], '[a, b, c] con: a <= b <= c'

```

[Anexo H]

[Esquemas de control difuso]

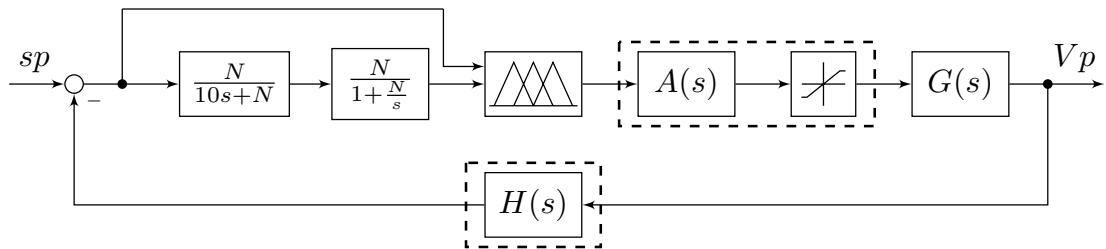


Figura 35. Esquema de control implementado: PD difuso. Fuente: Elaboración propia.

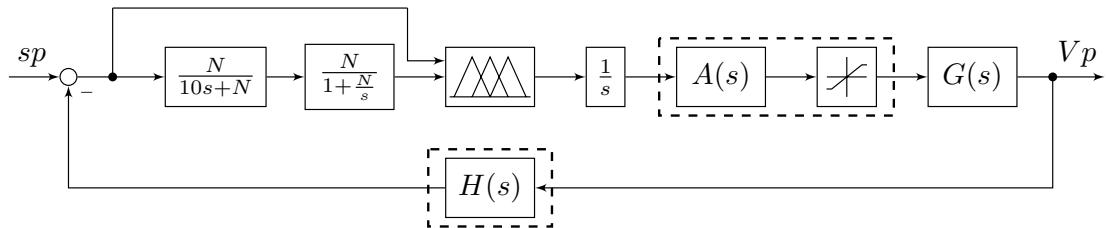


Figura 36. Esquema de control implementado: PI difuso. Fuente: Elaboración propia.

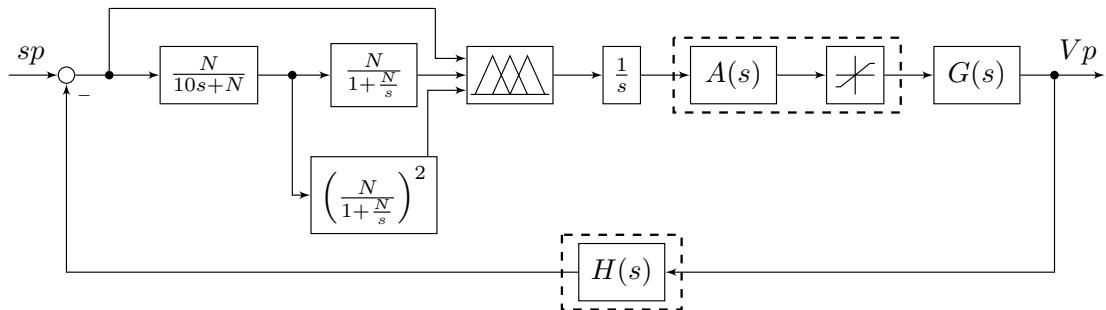


Figura 37. Esquema de control implementado: PID difuso. Fuente: Elaboración propia.

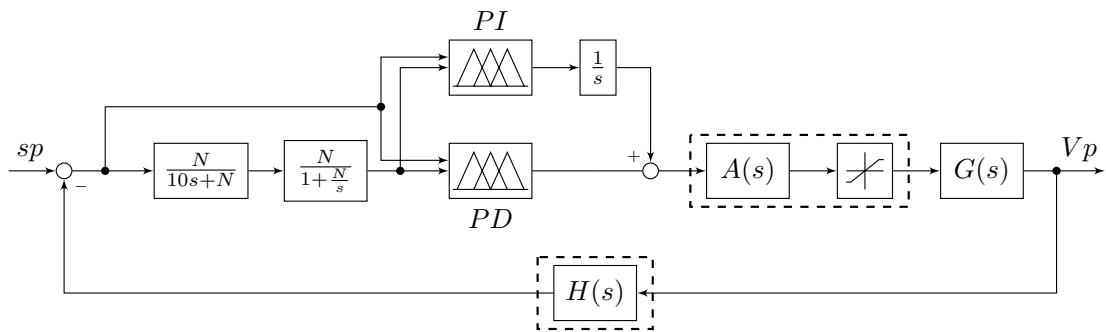


Figura 38. Esquema de control implementado: PD difuso mas PI difuso. Fuente: Elaboración propia.

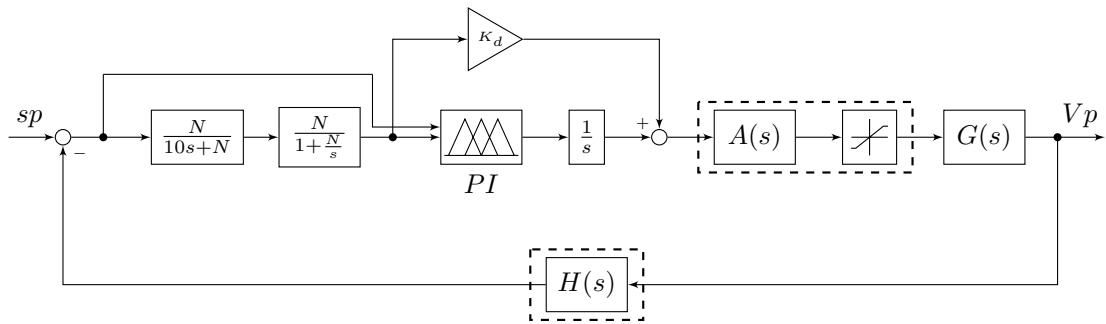


Figura 39. Esquema de control implementado: PI difuso más derivada. Fuente: Elaboración propia.

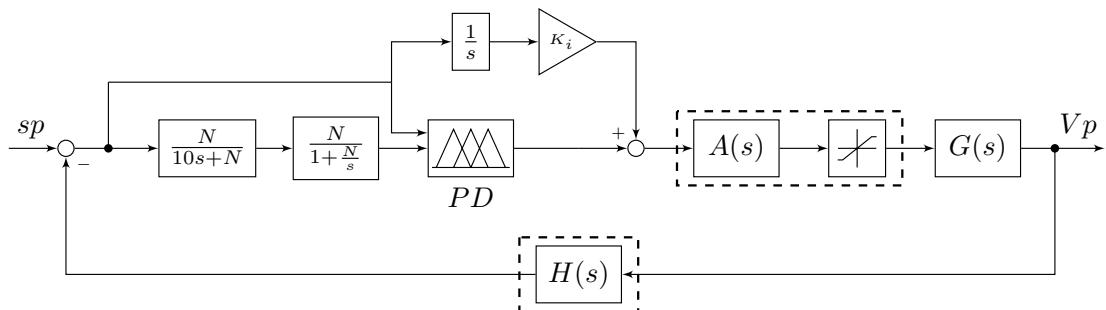


Figura 40. Esquema de control implementado: PD difuso más integrador. Fuente: Elaboración propia.

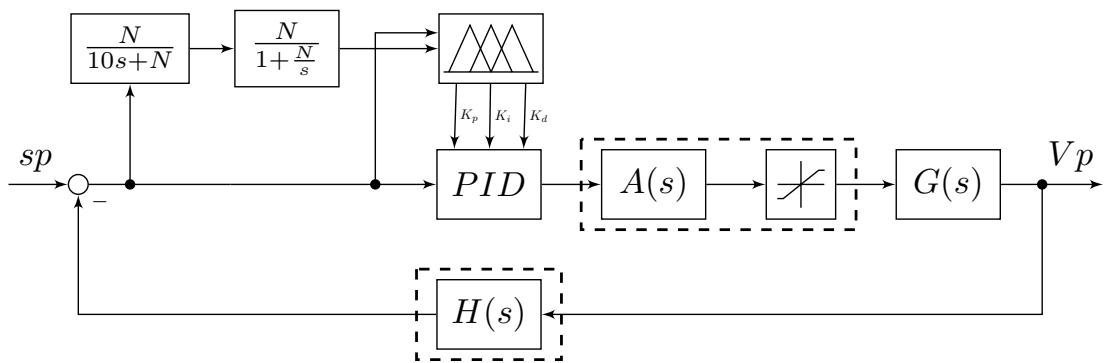


Figura 41. Esquema de control implementado: Programador de ganancias. Fuente: Elaboración propia.

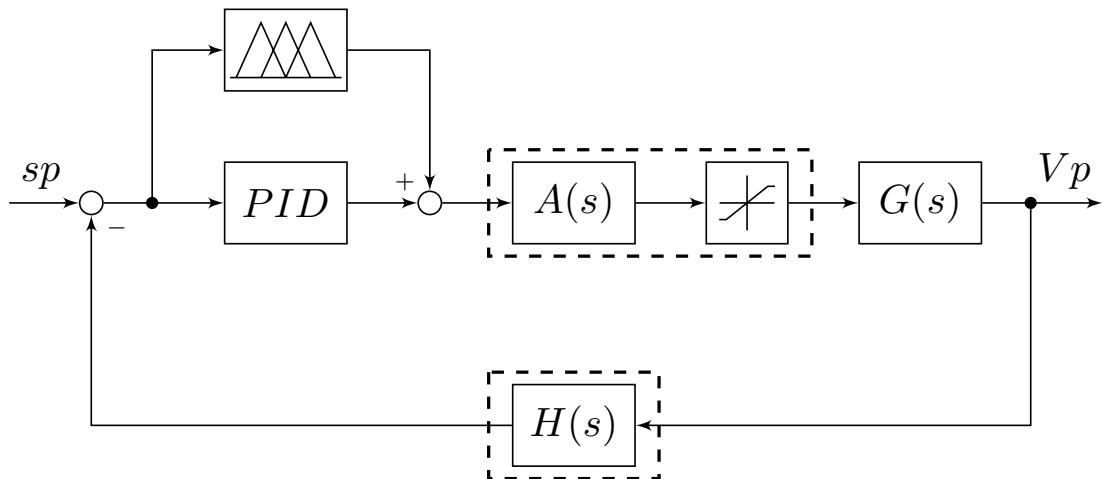


Figura 42. Esquema de control implementado: PID clásico mas P difuso. Fuente: Elaboración propia.

[Anexo I]

[Formato interno para guardar controlador por medio de un archivo .JSON]

Código 9. Formato para un controlador con una entrada, una salida y tres reglas. Las reglas son: una simple, una regla con premisa negada y una regla con salida ponderada en 0.25.

```
[  
  [  
    {  
      "nombre": "entrada1",  
      "numeroE": 3,  
      "etiquetas": [  
        {  
          "nombre": "etiqueta1",  
          "mf": "trimf",  
          "definicion": [  
            -20.0,  
            -10.0,  
            0.0  
          ]  
        },  
        {  
          "nombre": "etiqueta2",  
          "mf": "trapmf",  
          "definicion": [  
            -10.0,  
            -5.0,  
            5.0,  
            10.0  
          ]  
        },  
        {  
          "nombre": "etiqueta3",  
          "mf": "gaussmf",  
          "definicion": [  
            2.5,  
            10.0  
          ]  
        }  
      ],  
      "rango": [  
        -10,  
        10  
      ]  
    },  
    [  
      {  
        "nombre": "salida1",  
        "numeroE": 3,  
        "etiquetas": [  
          {  
            "nombre": "etiqueta1",  
            "mf": "gauss2mf",  
            "definicion": [  
              1.25,  
              -15.0,  
              1.25,  
              0.25  
            ]  
          }  
        ]  
      }  
    ]  
  ]
```

```

        -5.0
    ],
},
{
  "nombre": "etiqueta2",
  "mf": "dsigmf",
  "definicion": [
    1.0,
    -5.0,
    1.0,
    5.0
  ]
},
{
  "nombre": "etiqueta3",
  "mf": "pimf",
  "definicion": [
    0.0,
    5.0,
    15.0,
    20.0
  ]
},
],
"rango": [
  -10,
  10
],
"metodo": "centroid"
}
],
[
[
  [
    [
      [
        [
          "etiqueta1",
          0,
          false
        ]
      ],
      [
        [
          [
            "etiqueta1",
            0,
            1.0
          ]
        ],
        true
      ],
      [
        [
          [
            "etiqueta2",
            0,
            true
          ]
        ],
        [
          [
            "etiqueta2",
            0,
            0
          ]
        ]
      ]
    ]
  ]
]
]

```

```
    1.0
      ]
    ],
  false
  ],
  [
    [
      [
        "etiqueta3",
        0,
        false
      ]
    ],
    [
      [
        "etiqueta3",
        0,
        0.249999999999933
      ],
      false
    ]
  ]
]
```

[Anexo J]

[Códigos para el manejo de archivos FIS]

Código 10. Extracción de datos del archivo FIS - YAPFLM.

```
class FISParser:  
    """  
        [Clase para cargar y exportar archivos .fis, para cargar los archivos FIS las funciones  
        ↳ get_system, get_vars,  
        ↳ get_var y get_rules fueron tomadas de yapflm:  
  
        Yet Another Python Fuzzy Logic Module: https://github.com/sputnick1124/yapflm  
  
        para obtener los datos necesarias del .fis, de allí, se aplica la función fis_to_json  
        ↳ para completar el  
        parsin.  
  
        En el caso de la exportación, se realiza utilizando la función json_to_fis]  
    """  
  
    def __init__(self, file, InputList=None, OutputList=None, RuleEtiquetas=None):  
        """  
            [Constructor de la clase, inicializa las variables a utilizar y selecciona entre  
            ↳ cargar el fis o  
            ↳ exportarlo dependiendo de las variables con las que se cree el objeto]  
  
            :param file: [Dirección del archivo a cargar o exportar]  
            :type file: [str]  
            :param inputlist: [Lista de variables de entrada], defaults to None  
            :type inputlist: [list], optional  
            :param OutputList: [Lista de variables de salida], defaults to None  
            :type OutputList: [list], optional  
            :param RuleEtiquetas: [Lista con la información necesaria para crear las reglas],  
            ↳ defaults to None  
            :type RuleEtiquetas: [list], optional  
        """  
  
        # Cargar archivo .fis  
        if InputList is None and OutputList is None and RuleEtiquetas is None:  
            with open(file, 'r') as infis:  
                self.rawlines = infis.readlines()  
            self.systemList = []  
            self.InputList = []  
            self.OutputList = []  
            self.RuleList = []  
            self.get_system()  
            self.get_vars()  
            self.get_rules()  
        else:  
            # Exportar archivo .fis  
            self.file = file  
            self.InputList = InputList  
            self.OutputList = OutputList  
            self.RuleEtiquetas = RuleEtiquetas  
            self.json_to_fis()  
  
    def get_system(self):  
        """ [Función tomada de yapflm (Yet Another Python Fuzzy Logic Module)] """
```

```

end_sysblock = self.rawlines.index('\n')
systemblock = self.rawlines[1:end_sysblock]
fisargs = map(lambda x: parse('{arg}={val}', x), systemblock)
fissys = {f['arg'].lower(): f['val'].strip("") for f in fisargs}
self.numinputs = int(fissys['numinputs'])
self.numoutputs = int(fissys['numoutputs'])
self.numrules = int(fissys['numrules'])
self.start_varblocks = end_sysblock + 1
self.systemList = fissys

def get_var(self, vartype, varnum, start_line, end_line):
    """ [Función tomada de yapflm (Yet Another Python Fuzzy Logic Module)] """
    varblock = self.rawlines[start_line:end_line]
    fisargs = map(lambda x: parse('{arg}={val}', x), varblock)
    fisvar = {f['arg'].lower(): f['val'].strip("") for f in fisargs}

    if 'input' in vartype:
        self.InputList.append(fisvar)
    elif 'output' in vartype:
        self.OutputList.append(fisvar)

def get_vars(self):
    """ [Función tomada de yapflm (Yet Another Python Fuzzy Logic Module)] """

    start_ruleblock = self.rawlines.index('[Rules]\n')
    var_lines = []
    var_types = []
    flag = 0
    for i, line in enumerate(self.rawlines[self.start_varblocks - 1:start_ruleblock]):
        if flag:
            flag = 0
            vt = parse('{type}{num:d}', line)
            var_types.append((vt['type'].lower(), vt['num']))
        if line == '\n':
            var_lines.append(i + self.start_varblocks - 1)
            flag = 1
    for i, l in enumerate(var_lines[:-1]):
        if 'input' in var_types[i][0]:
            self.get_var('input', var_types[i][1] - 1, l + 2, var_lines[i + 1])
        elif 'output' in var_types[i][0]:
            self.get_var('output', var_types[i][1] - 1, l + 2, var_lines[i + 1])

def get_rules(self):
    """ [Función tomada de yapflm (Yet Another Python Fuzzy Logic Module)] """

    start_ruleblock = self.rawlines.index('[Rules]\n')
    ruleblock = self.rawlines[start_ruleblock + 1:]
    antecedents = ('{a%d:d}' * self.numinputs) % tuple(range(self.numinputs)).strip()
    consequents = ('{c%d:d}' * self.numoutputs) % tuple(range(self.numoutputs))
    p = antecedents + ', ' + consequents + '({w:d}) : {c:d}'
    for rule in ruleblock:
        try:
            p = antecedents + ', ' + consequents + '({w:d}) : {c:d}'
            rp = parse(p, rule)
            r = []
            for inp in range(self.numinputs):
                rpar = rp['a%d' % inp]
                rval = rpar if rpar != 0 else None

```

```

        r.append(rval)
    for outp in range(self.numoutputs):
        rpar = rp['c%d' % outp]
        rval = rpar if rpar != 0 else None
        r.append(rval)
    r += [rp['w'], rp['c'] - 1]
    self.RuleList.append(r)
except:
    p = antecedents + ', ' + consequents + '({w:f}) : {c:d}'
    rp = parse(p, rule)
    r = []
    for inp in range(self.numinputs):
        rpar = rp['a%d' % inp]
        rval = rpar if rpar != 0 else None
        r.append(rval)
    for outp in range(self.numoutputs):
        rpar = rp['c%d' % outp]
        rval = rpar if rpar != 0 else None
        r.append(rval)
    r += [rp['w'], rp['c'] - 1]
    self.RuleList.append(r)

```

Código 11. Procesado de los datos del FIS, esta función pertenece a la clase FISParser.

```

def fis_to_json(self):
    """ [Función para completar la creación del controlador a partir de un archivo .fis]
    """
    # Datos del controlador
    ni = int(self.systemList['numinputs'])
    no = int(self.systemList['numoutputs'])
    nr = int(self.systemList['numrules'])

    InputList = [0] * ni
    OutputList = [0] * no
    RuleEtiquetas = []

    # Creación de las variables de entrada
    for i in range(ni):
        InputList[i] = {
            "nombre": self.InputList[i]['name'],
            "numeroE": int(self.InputList[i]['nummf']),
            "etiquetas": [0] * int(self.InputList[i]['nummf']),
            "rango": ast.literal_eval(
                re.sub("\s+", ", ", self.InputList[i]['range'].strip()))
        }

        for ne in range(int(self.InputList[i]['nummf'])):
            temp_etiqueta = self.InputList[0]['mf' + str(ne + 1)].replace(
                "'", '').split(':')
            temp2 = temp_etiqueta[1].split(',')
            InputList[i]['etiquetas'][ne] = {
                "nombre": temp_etiqueta[0],
                "mf": temp2[0],
                "definicion": ast.literal_eval(re.sub("\s+", ", ", temp2[1].strip())))
    }

```

```

# Creación de las variables de salida
for i in range(no):
    OutputList[i] = {
        "nombre": self.OutputList[i]['name'],
        "numeroE": int(self.OutputList[i]['nummf']),
        "etiquetas": [0] * int(self.OutputList[i]['nummf']),
        "rango": ast.literal_eval(
            re.sub("\s+", "", self.OutputList[i]['range'].strip())),
        "metodo": self.systemList['defuzzmethod']
    }

    for ne in range(int(self.OutputList[i]['nummf'])):
        temp_etiqueta = self.OutputList[i]['mf' + str(ne + 1)].replace(
            "'", '').split(':')
        temp2 = temp_etiqueta[1].split(',')
        OutputList[i]['etiquetas'][ne] = {
            "nombre": temp_etiqueta[0],
            "mf": temp2[0],
            "definicion": ast.literal_eval(re.sub("\s+", "", temp2[1].strip()))
        }

# Creación de las reglas
for numeror, i in enumerate(self.RuleList):
    ril = []
    rol = []

    for j in range(ni):
        if i[j] is not None:
            nombre = InputList[j]['etiquetas'][abs(i[j]) - 1]['nombre']
            numero = j
            negacion = False if i[j] > 0 else True
            ril.append([nombre, numero, negacion])

    for j in range(ni, no + ni):
        if i[j] is not None:
            if i[j] < 0:
                raise TypeError('No se permiten salidas negadas')
            nombre = OutputList[j - ni]['etiquetas'][abs(i[j]) - 1]['nombre']
            numero = j - ni
            peso = float(i[no + ni])
            rol.append([nombre, numero, peso])

    and_condition = True if i[ni + no + 1] == 0 else False
    RuleEtiquetas.append(copy.deepcopy([ril, rol, and_condition]))

return copy.deepcopy(InputList), copy.deepcopy(OutputList),
→ copy.deepcopy(RuleEtiquetas)

```

Código 12. Exportar archivos FIS, esta función pertenece a la clase FISParser.

```

def json_to_fis(self):
    """ [Función para exportar el controlador en formato .fis] """

    # Datos del controlador
    ni = len(self.InputList)
    no = len(self.OutputList)
    nr = len(self.RuleEtiquetas)

    with open(self.file, 'w') as f:

        # Información general del controlador
        f.write(f"[System]\n")
        f.write(f"Name='{self.file.split('/')[-1].split('.')[0]}\\n")
        f.write(f"Type='mamdani'\\n")
        f.write(f"Version=2.0\\n")
        f.write(f"NumInputs={ni}\\n")
        f.write(f"NumOutputs={no}\\n")
        f.write(f"NumRules={nr}\\n")
        f.write(f"AndMethod='min'\\n")
        f.write(f"OrMethod='max'\\n")
        f.write(f"ImpMethod='min'\\n")
        f.write(f"AggMethod='max'\\n")
        f.write(f"DefuzzMethod='{self.OutputList[0]['metodo']}'\\n")
        f.write(f"\\n")

        # Parsin de las entradas del controlador
        for i in range(ni):
            f.write(f"[Input" + str(i + 1) + "]\\n")
            f.write(f"Name='{self.InputList[i]['nombre']}'\\n")
            string_temp = re.sub('\\s+', '',
                                str(self.InputList[i]['rango'])).replace(',', ' ')
            f.write(f"Range={string_temp}\\n")
            f.write(f"NumMFs={self.InputList[i]['numeroE']}\\n")

            for ne in range(self.InputList[i]['numeroE']):
                string_temp = re.sub(
                    '\\s+', '',
                    str(self.InputList[i]['etiquetas'][ne]['definicion'])).replace(
                        ',', ' ')
                f.write(
                    f"MF{ne+1}='{self.InputList[i]['etiquetas'][ne]['nombre']}'"
                    f":'{self.InputList[i]['etiquetas'][ne]['mf']}',{string_temp}\\n"
                )
            f.write(f"\\n")

        # Parsin de las salidas del controlador
        for i in range(no):
            f.write(f"[Output" + str(i + 1) + "]\\n")
            f.write(f"Name='{self.OutputList[i]['nombre']}'\\n")
            string_temp = re.sub('\\s+', '',
                                str(self.OutputList[i]['rango'])).replace(',', ' ')
            f.write(f"Range={string_temp}\\n")
            f.write(f"NumMFs={self.OutputList[i]['numeroE']}\\n")

            for ne in range(self.OutputList[i]['numeroE']):
                string_temp = re.sub(
                    '\\s+', '',
                    str(self.OutputList[i]['etiquetas'][ne]['definicion'])).replace(
                        ',', ' ')
                f.write(

```

```

        f"MF{ne+1}='{self.OutputList[i]['etiquetas'][ne]['nombre']}"
        f"':{self.OutputList[i]['etiquetas'][ne]['mf']}',{string_temp}\n"
    )

f.write(f"\n")

# Pารsin de las reglas del controlador
rules_no_format = []
for i, rule in enumerate(self.RuleEtiquetas):

    inner_rules = []

    # set de entradas
    for nir in range(ni):
        for inputrule in rule[0]:
            if nir == inputrule[1]:
                if not inputrule[2]:
                    for ner, etiqueta in enumerate(self.InputList[nir]['etiquetas']):
                        if etiqueta['nombre'] == inputrule[0]:
                            inner_rules.append(ner + 1)
                            break
                else:
                    for ner, etiqueta in enumerate(self.InputList[nir]['etiquetas']):
                        if etiqueta['nombre'] == inputrule[0]:
                            inner_rules.append(-ner - 1)
                            break
                break
            else:
                continue
        else:
            inner_rules.append(0)
            break

    # set de salidas
    for nor in range(no):
        for outputrule in rule[1]:
            if nor == outputrule[1]:
                for ner, etiqueta in enumerate(self.OutputList[nor]['etiquetas']):
                    if etiqueta['nombre'] == outputrule[0]:
                        inner_rules.append(ner + 1)
                        break
                break
            else:
                continue
        else:
            inner_rules.append(0)

    inner_rules.append(rule[1][0][2])

    if rule[2]:
        inner_rules.append(1)
    else:
        inner_rules.append(2)

    rules_no_format.append(copy.deepcopy(inner_rules))

f.write(f"[Rules]\n")

```

```

# Escribiendo las reglas en el archivo
for i in range(nr):
    rule_str = ""
    for j in range(ni):
        if not j == ni - 1:
            rule_str += str(rules_no_format[i][j]) + " "
        else:
            rule_str += str(rules_no_format[i][j])
    rule_str += ", "
    for j in range(ni, ni + no):
        rule_str += str(rules_no_format[i][j]) + " "
rule_str += f"({str(rules_no_format[i][ni+no])})" + " "
rule_str += f": {str(rules_no_format[i][ni+no+1])}\n"
f.write(rule_str)

return

```

[Anexo K]

[Tablas de Butcher para los métodos de Runge-Kutta]

Métodos explícitos

Runge-Kutta de orden 2

0			
1/2	1/2		
		0	1

(55)

Runge-Kutta de orden 3

0				
1/2	1/2			
1	-1	2		
			1/6	2/3
				1/6

(56)

Heun de orden 3

0				
1/3	1/3			
2/3	0	2/3		
			1/4	0
				3/4

(57)

Ralston de orden 3

0				
1/2	1/2			
3/4	0	3/4		
			2/9	1/3
				4/9

(58)

SSPRK de orden 3

0				
1	1			
1/2	1/4	1/4		
			1/6	1/6
				2/3

(59)

Runge-Kutta de orden 4

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
			1/6	1/3
				1/3
				1/6

(60)

Runge-Kutta 3/8 de orden 4

0					
1/3	1/3				
2/3	-1/3	1			
1	1	-1	1		
	1/8	3/8	3/8	1/8	

(61)

Ralston con mínimo error de truncamiento de orden 4

0					
0.4	0.4				
0.45573725	0.29697761	0.15875964			
1	0.21810040	-3.05096516	3.83286476		
	0.17476028	-0.55148066	1.20553560	0.17118478	

Runge-Kutta de orden 5

0						
1/4	1/4					
1/4	1/8	1/8				
1/2	0	-1/2	1			
3/4	3/16	0	0	9/16		
1	-3/7	2/7	12/7	-12/7	8/7	
	7/90	0	32/90	12/90	32/90	7/90

Métodos embebidos

Bogacki-Shampine de orden 3(2)

	0				
1/2	1/2				
3/4	0	3/4			
1	2/9	1/3	4/9		
	2/9	1/3	4/9	0	
	7/24	1/4	1/3	1/8	

Fehlberg de orden 4(5)

0						
1/4	1/4					
3/8	3/32	9/32				
12/13	1932/2197	-7200/2197	7296/2197			
1	439/216	-8	3680/513	-845/4104		
1/2	-8/27	2	-3544/2565	1859/4104	-11/40	
	25/216	0	1408/2565	2197/4104	-1/5	0
	16/135	0	6656/12825	28561/56430	-9/50	2/55
						(65)

Cash-Karp de orden 4(5)

0						
1/5	1/5					
3/10	3/40	9/40				
3/5	3/10	-9/10	6/5			
1	-11/54	5/2	-70/27	35/27		
7/8	1631/55296	175/512	575/13824	44275/110592	253/4096	
	2825/27648	0	18575/48384	13525/55296	277/14336	1/4
	37/378	0	250/621	125/594	0	512/1771
						(66)

Dormand-Prince de orden 5(4)

0						
1/5	1/5					
3/10	3/40	9/40				
4/5	44/45	-56/15	32/9			
8/9	19372/6561	-25360/2187	64448/6561	-212/729		
1	9017/3168	-355/33	46732/5247	49/176	-5103/18656	
1	35/384	0	500/1113	125/192	-2187/6784	11/84
	35/384	0	500/1113	125/192	-2187/6784	11/84
	5179/57600	0	7571/16695	393/640	-92097/339200	187/2100 1/40
						(67)

[Anexo L]

[Códigos para el tamaño de paso variable]

Código 13. Tamaño de paso variable para Runke-Kutta explícitos.

```
def rk_doble_paso_adaptativo(systema,
                               h_ant,
                               tiempo,
                               tbound,
                               xVectB,
                               entrada,
                               metodo,
                               ordenq,
                               rtol,
                               atol,
                               max_step_increase,
                               min_step_decrease,
                               safety_factor):

    while True:
        # Para asegurar el tiempo máximo
        if tiempo + h_ant >= tbound:
            h_ant = tbound - tiempo
            yS, xVectSn = metodo(*systema, xVectB, h_ant, entrada)
            h_est = h_ant
        else:
            # Paso de tamaño regular
            yB, xVectBn = metodo(*systema, xVectB, h_ant, entrada)

            # Dos pasos de tamaño medio
            yS, xVectSn = metodo(*systema, xVectB, h_ant / 2, entrada)
            yS, xVectSn = metodo(*systema, xVectSn, h_ant / 2, entrada)

            # Ajuste del tamaño de paso
            scale = atol + rtol * (np.abs(xVectBn) + np.abs(xVectSn)) / 2
            delta1 = np.abs(xVectBn - xVectSn)
            error_norm = norm(delta1 / scale)

            if error_norm == 0:
                # Incremento máximo dado el bajo error
                h_est = h_ant * max_step_increase
            elif error_norm <= 1:
                # Incremento normal
                h_est = h_ant * min(max_step_increase,
                                     max(1, safety_factor * error_norm**(-1 / (ordenq+1))))
            else:
                # Decremento normal y se vuelve a calcular la salida
                h_ant = h_ant * min(
                    1,
                    max(min_step_decrease, safety_factor * error_norm**(-1 / (ordenq+1))))
            continue
        break
    return h_ant, h_est, yS, xVectSn
```

Código 14. Tamaño de paso variable para Runke-Kutta embebidos.

```

def rk_embebido_adaptativo(systema,
                            h_ant,
                            tiempo,
                            tbound,
                            xVectr,
                            entrada,
                            metodo,
                            ordenq,
                            rtol,
                            atol,
                            max_step_increase,
                            min_step_decrease,
                            safety_factor):

    while True:
        # Para asegurar el tiempo máximo
        if tiempo + h_ant >= tbound:
            h_ant = tbound - tiempo
            yr, xr, xtemp = metodo(*systema, xVectr, h_ant, entrada)
            h_est = h_ant
        else:
            # Método embebido, la integración se continua con yr y xr
            yr, xr, xtemp = metodo(*systema, xVectr, h_ant, entrada)

        # Ajuste del tamaño de paso
        scale = atol + np.maximum(np.abs(xVectr), np.abs(xr)) * rtol
        delta1 = np.abs(xr - xtemp)
        error_norm = norm(delta1 / scale)

        if error_norm == 0:
            # Incremento máximo dado el bajo error
            h_est = h_ant * max_step_increase
        elif error_norm <= 1:
            # Incremento normal
            h_est = h_ant * min(max_step_increase,
                                max(1, safety_factor * error_norm**(-1 / (ordenq+1))))
        else:
            # Decremento normal y se vuelve a calcular la salida
            h_ant = h_ant * min(
                1,
                max(min_step_decrease, safety_factor * error_norm**(-1 / (ordenq+1))))
            continue
        break
    return h_ant, h_est, yr, xr

```

Código 15. Función para calcular la norma RMS, se realizo un pre compilado con numba para aumentar la velocidad de ejecucion.

```

@cc.export('norm', 'f8(f8[:,::1])')
def norm(x):
    """ [Función para calcular la norma RMS de un vector. Función tomada de SciPy] """
    return np.linalg.norm(x) / x.size**0.5

```

[Anexo M]

[Comparación grafica para el análisis de sistemas de control]

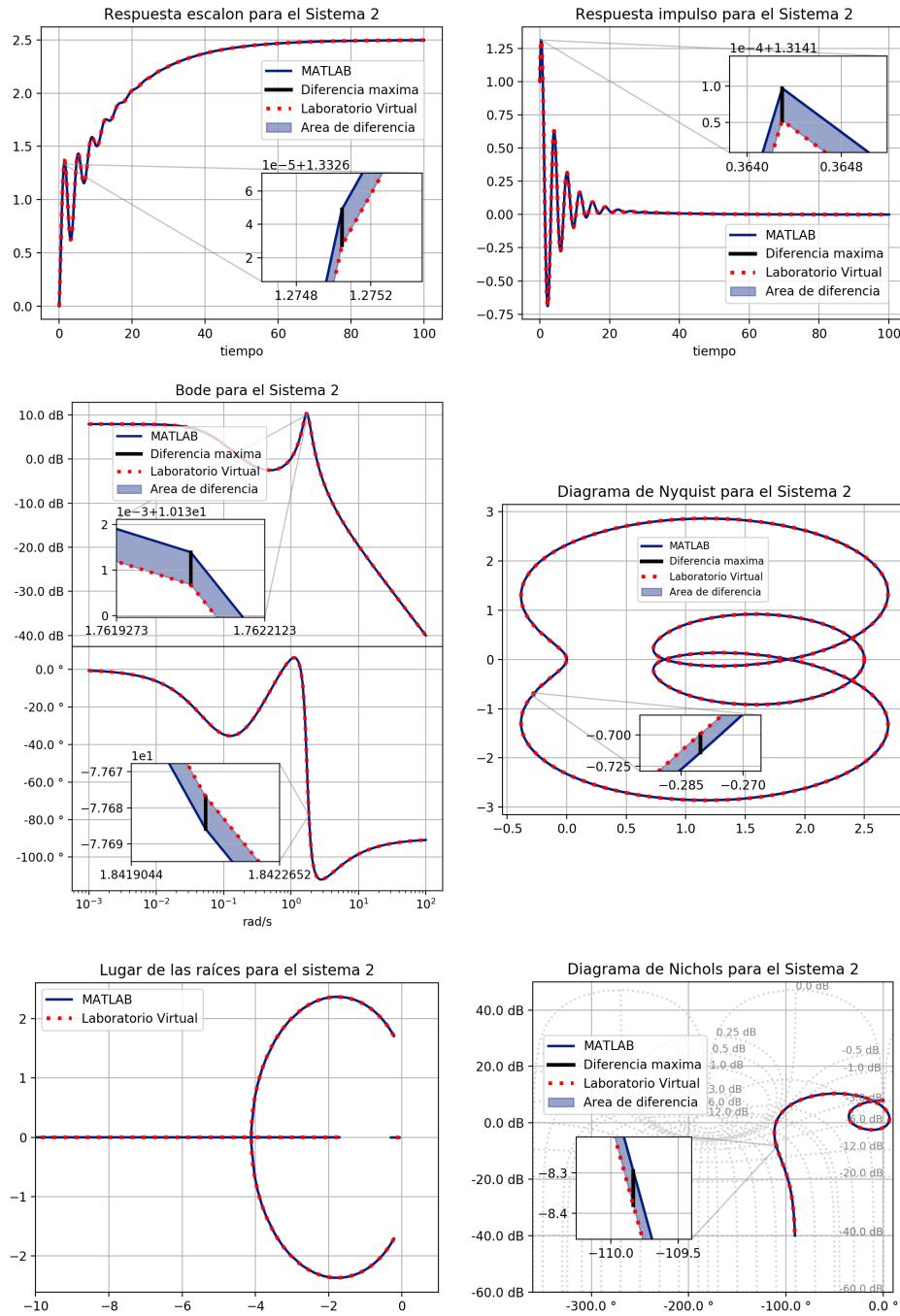


Figura 43. Comparación de análisis/MATLAB - sistema continuo 2. Fuente: Elaboración propia.

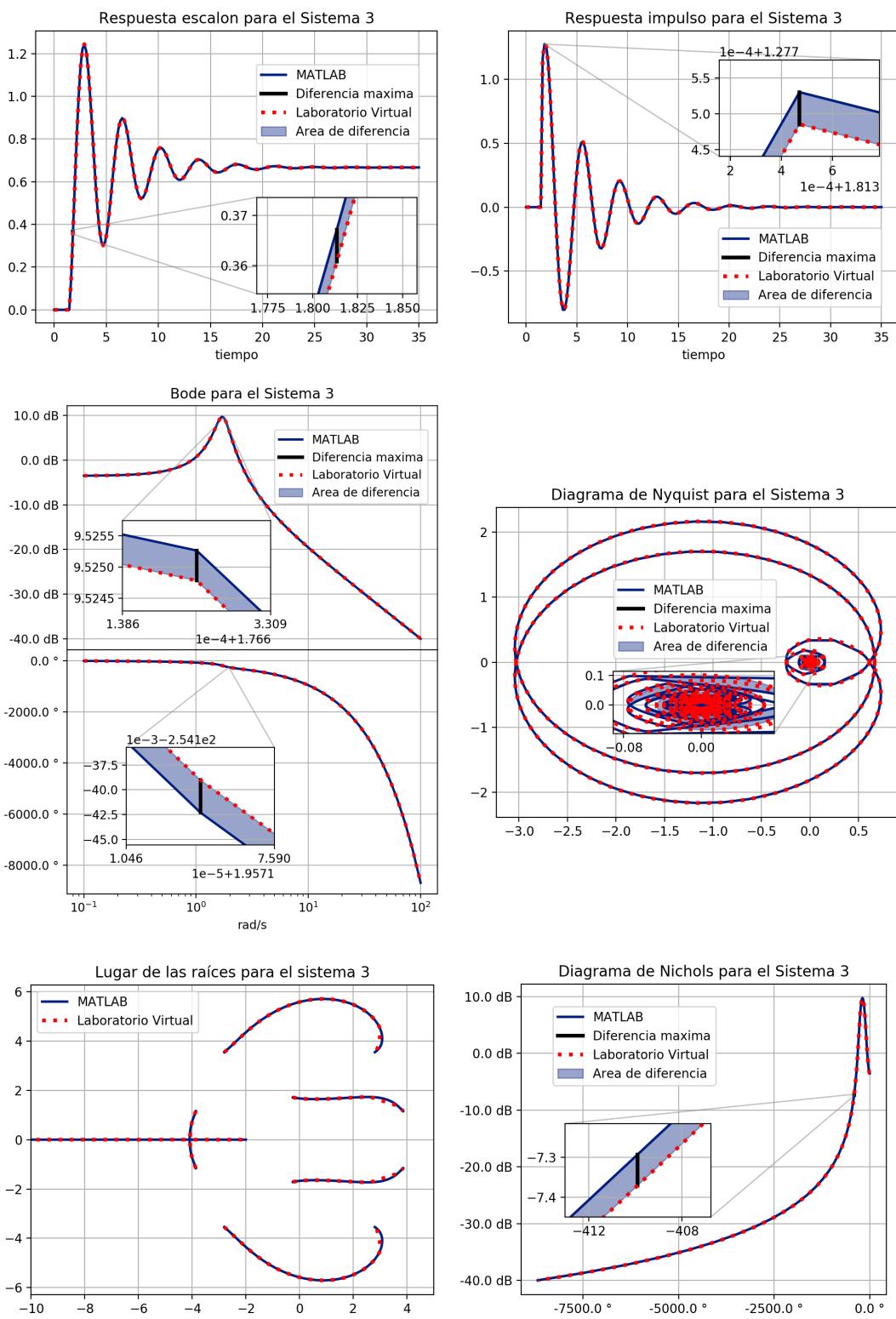


Figura 44. Comparación de análisis/MATLAB - sistema continuo 3. Fuente: Elaboración propia.

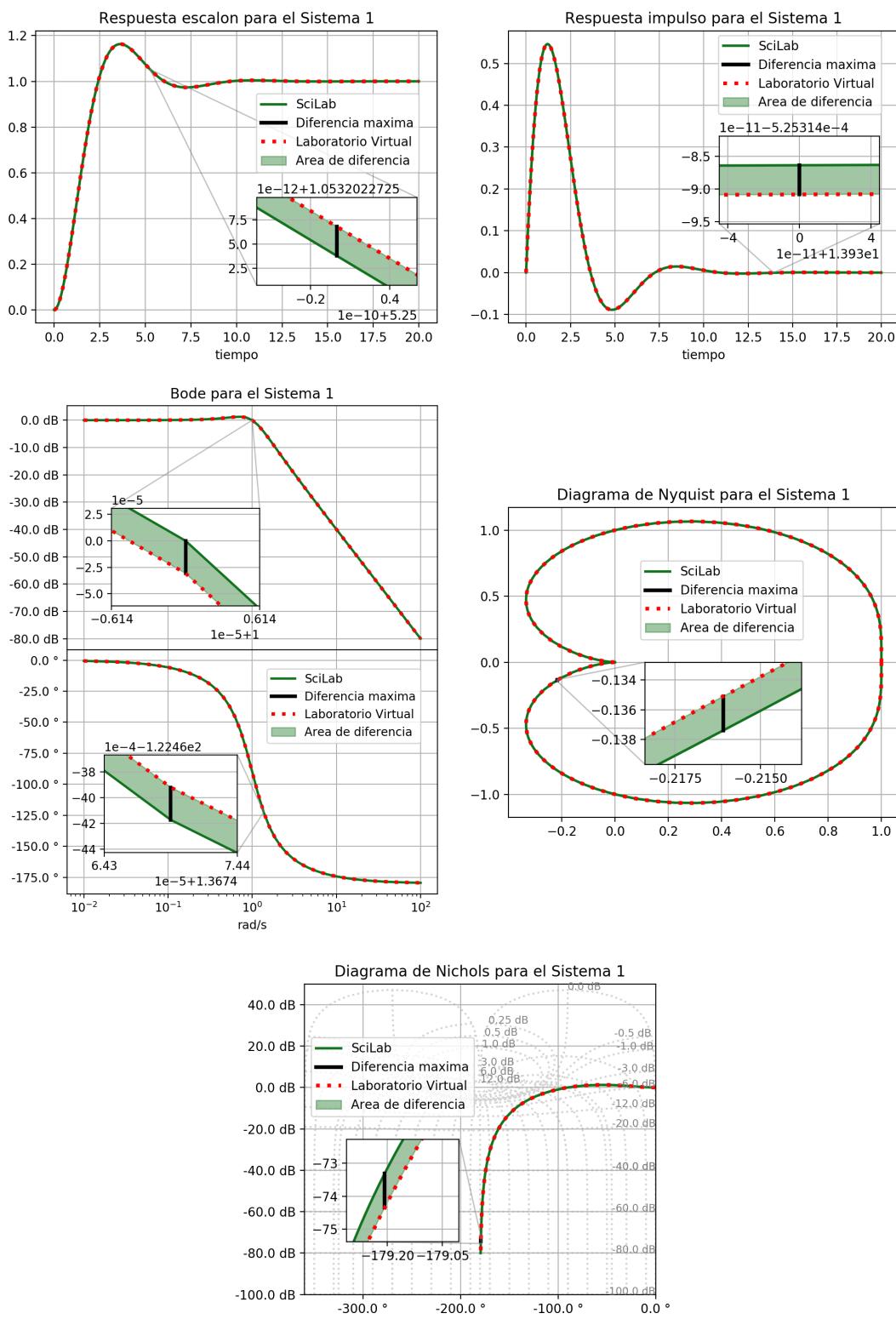


Figura 45. Comparación de análisis/SciLab - sistema continuo 1. Fuente: Elaboración propia.

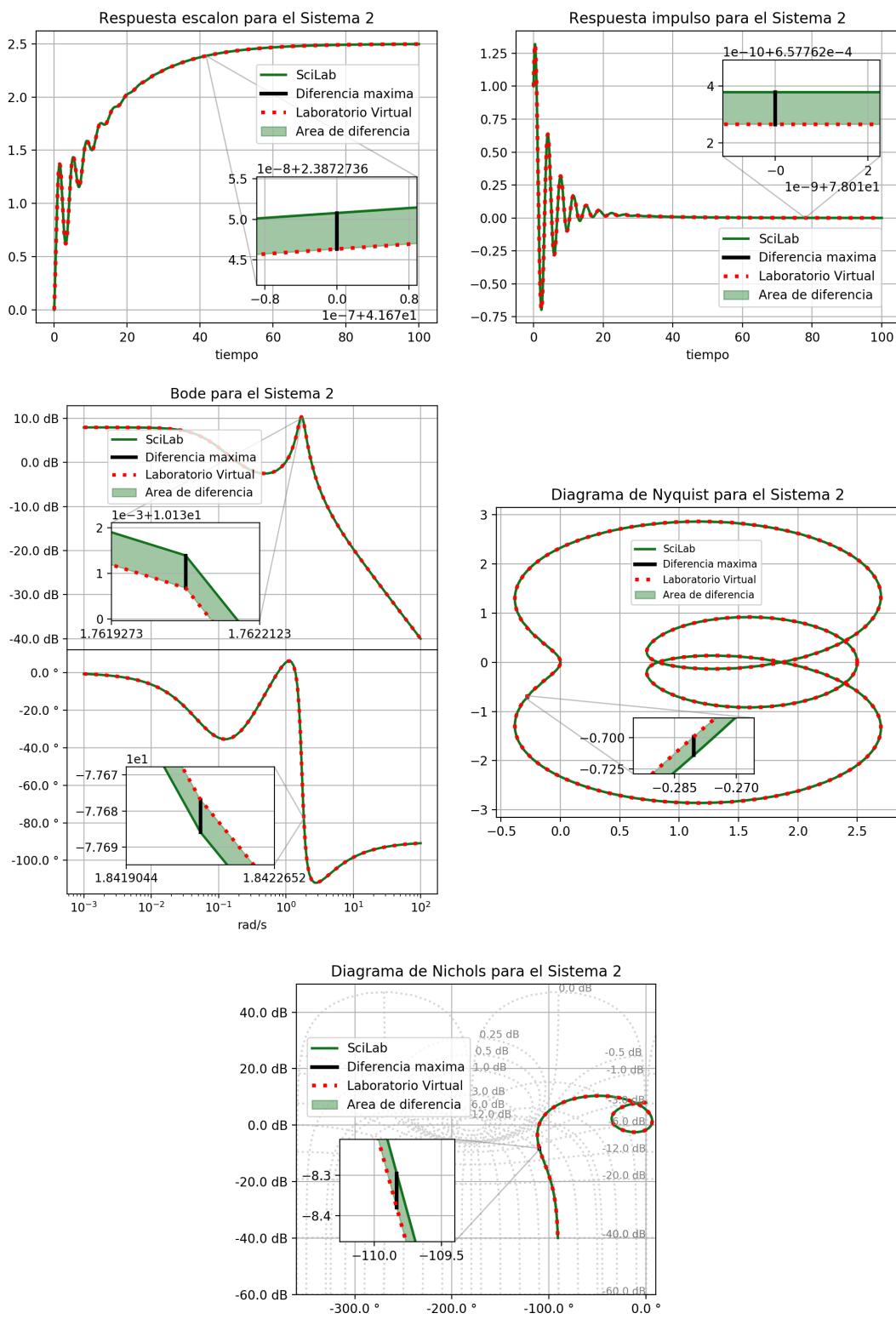


Figura 46. Comparación de análisis/SciLab - sistema continuo 2. Fuente: Elaboración propia.

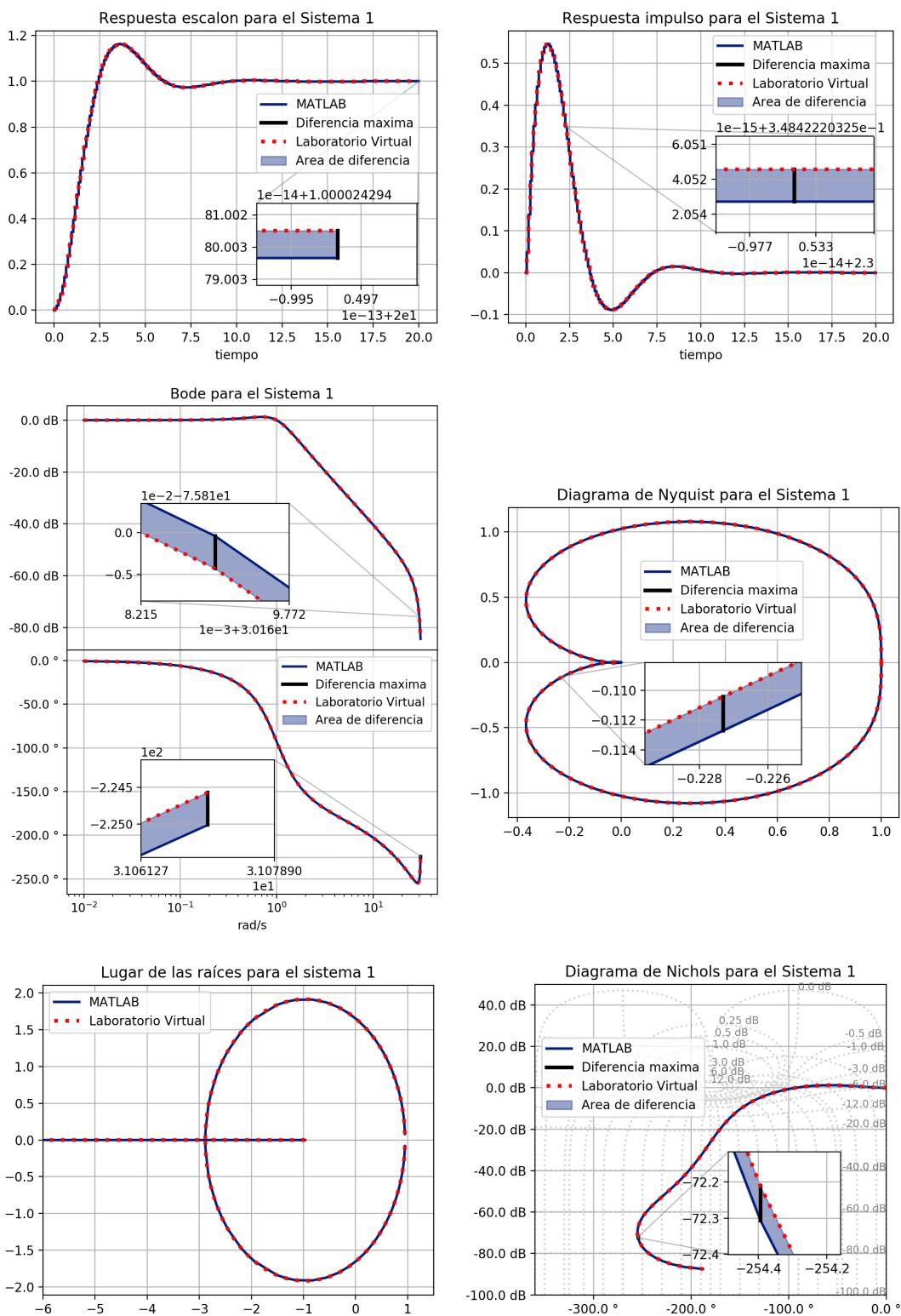


Figura 47. Comparación de análisis/MATLAB - sistema discreto 1. Fuente: Elaboración propia.

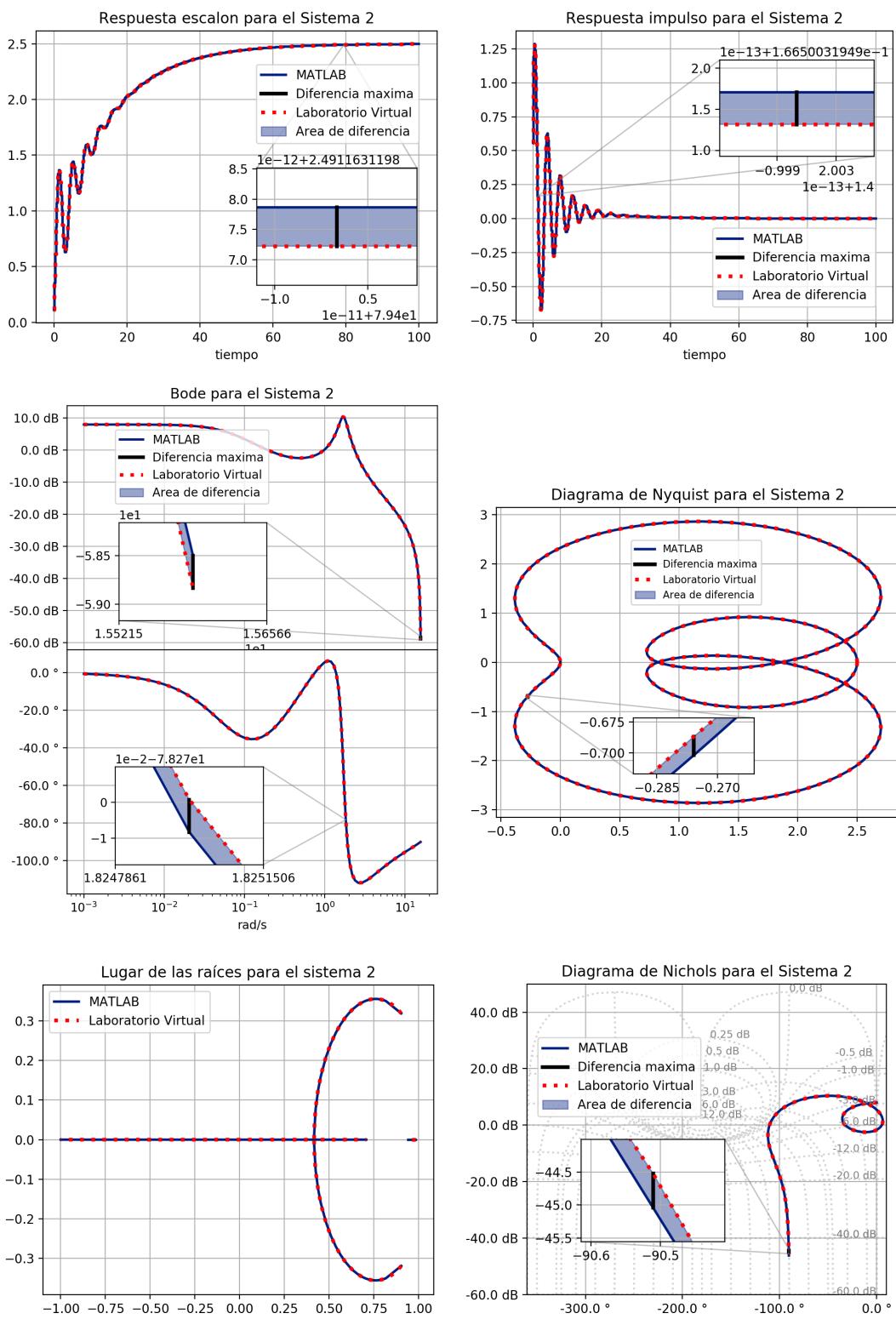


Figura 48. Comparación de análisis/MATLAB - sistema discreto 2. Fuente: Elaboración propia.

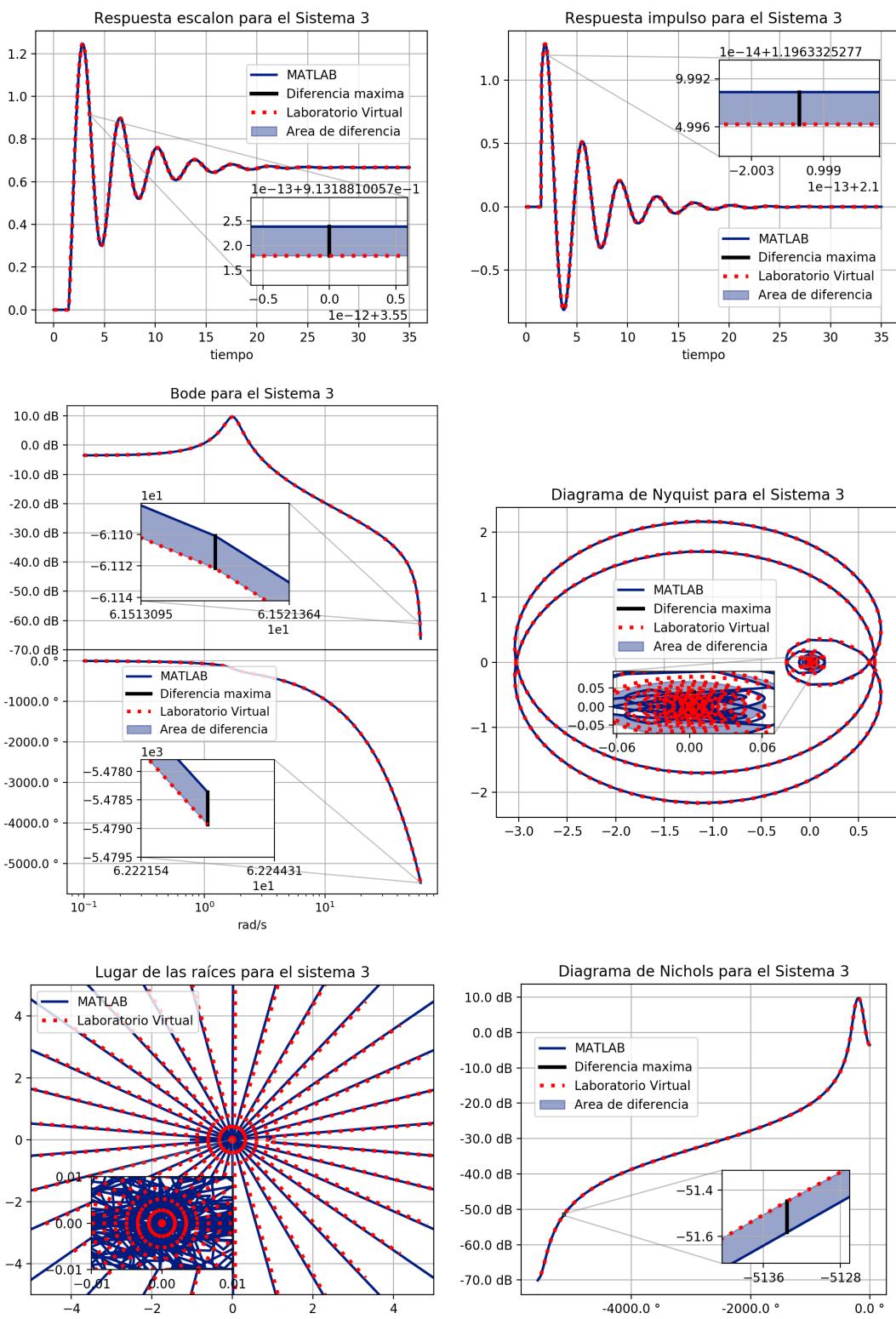


Figura 49. Comparación de análisis/MATLAB - sistema discreto 3. Fuente: Elaboración propia.

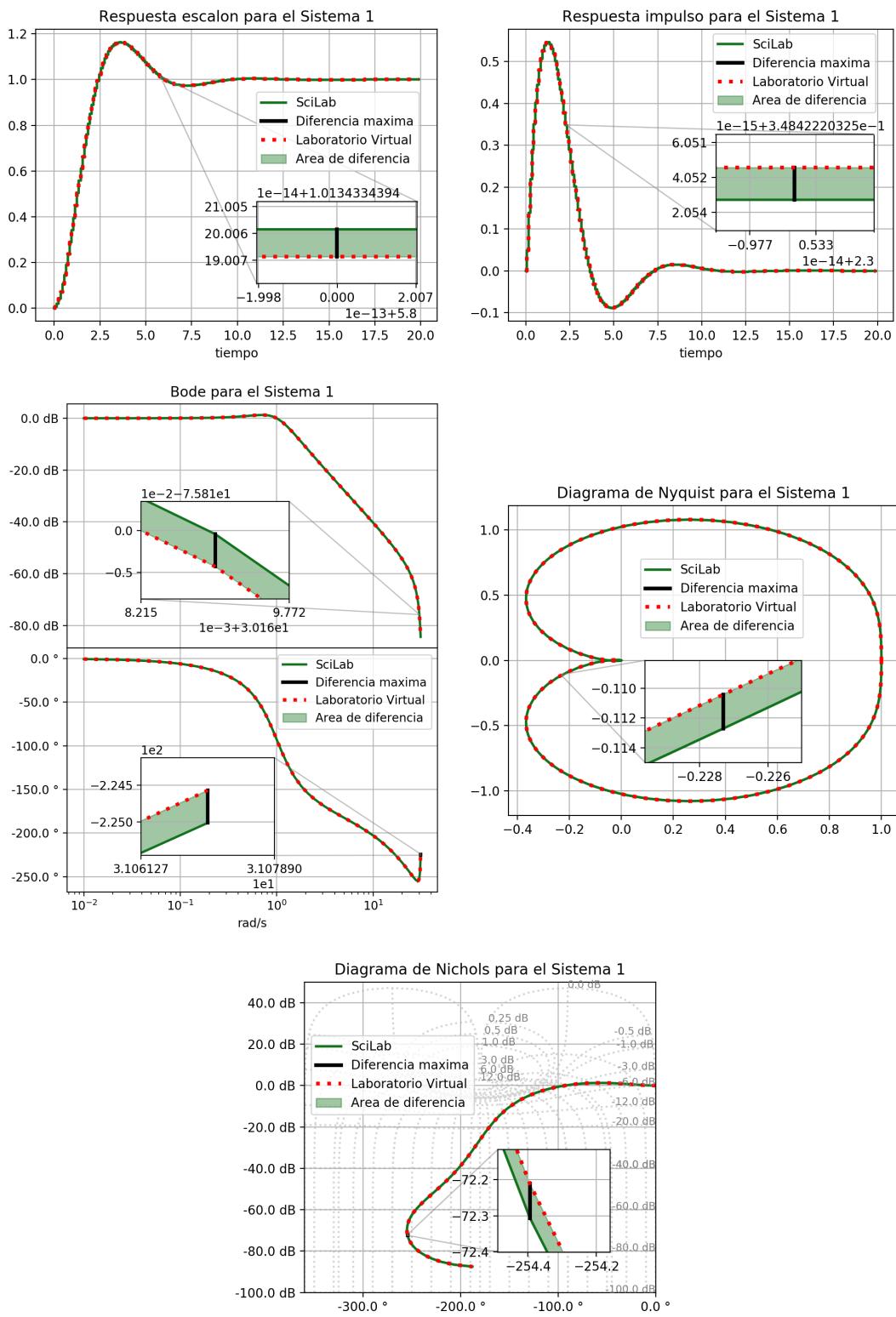


Figura 50. Comparación de análisis/SciLab - sistema discreto 1. Fuente: Elaboración propia.

[Anexo N]

[Estructura FIS para la comparación de diseño de controladores difusos]

Controlador 1:

```
[System]
Name='TomoC1'
Type='mamdani'
Version=2.0
NumInputs=1
NumOutputs=1
NumRules=5
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'

[Input1]
Name='entrada1'
Range=[-10 10]
NumMFs=5
MF1='etiqueta1':'trimf',[-20.0 -10.0 0.0]
MF2='etiqueta2':'trapmf',[-10.0 -7.5 -2.5 0.0]
MF3='etiqueta3':'gaussmf',[2.5 0]
MF4='etiqueta4':'dsigmf',[3 2.5 3 7.5]
MF5='etiqueta5':'smf',[2.5 10]

[Output1]
Name='salida1'
Range=[-10 10]
NumMFs=3
MF1='etiqueta1':'trimf',[-20.0 -10.0 0.0]
MF2='etiqueta2':'trimf',[-10.0 0.0 10.0]
MF3='etiqueta3':'trimf',[0.0 10.0 20.0]

[Rules]
1, 1 (1.0) : 1
2, 2 (1.0) : 2
-3, 3 (1.0) : 2
4, 2 (1.0) : 1
5, 1 (0.7999999999999998) : 1
```

Controlador 2:

```
[System]
Name='TomoC2'
Type='mamdani'
Version=2.0
NumInputs=2
NumOutputs=2
NumRules=9
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
```

```

DefuzzMethod='som'

[Input1]
Name='entrada1'
Range=[-1 1]
NumMFs=3
MF1='etiqueta1':'trimf',[ -2.0 -1.0 0.0]
MF2='etiqueta2':'trimf',[ -1.0 -0.0 1.0]
MF3='etiqueta3':'trimf',[ 0.0 1.0 2.0]

[Input2]
Name='entrada2'
Range=[-10 10]
NumMFs=3
MF1='etiqueta1':'trimf',[ -20.0 -10.0 0.0]
MF2='etiqueta2':'trimf',[ -10.0 0.0 10.0]
MF3='etiqueta3':'trimf',[ 0.0 10.0 20.0]

[Output1]
Name='salida1'
Range=[-1 1]
NumMFs=3
MF1='etiqueta1':'trimf',[ -2.0 -1.0 0.0]
MF2='etiqueta2':'trimf',[ -1.0 -0.0 1.0]
MF3='etiqueta3':'trimf',[ 0.0 1.0 2.0]

[Output2]
Name='salida2'
Range=[-100 100]
NumMFs=3
MF1='etiqueta1':'trimf',[ -200.0 -100.0 0.0]
MF2='etiqueta2':'trimf',[ -100.0 0.0 100.0]
MF3='etiqueta3':'trimf',[ 0.0 100.0 200.0]

[Rules]
1 1, 1 1 (1.0) : 1
1 2, 2 1 (1.0) : 2
1 -3, 3 1 (1.0) : 2
-2 2, 3 2 (1.0) : 2
-2 1, 3 3 (1.0) : 1
2 -3, 3 1 (1.0) : 1
3 1, 1 3 (0.8999999999999999) : 1
3 2, 2 2 (0.7499999999999998) : 1
-3 -3, 3 1 (0.7499999999999998) : 1

```

[Anexo N]

[Estructura FIS para la comparación de simulación de sistemas de control]

Controlador PID difuso:

```
[System]
Name='PIDdifuso'
Type='mamdani'
Version=2.0
NumInputs=3
NumOutputs=1
NumRules=29
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'

[Input1]
Name='error'
Range=[-1 1]
NumMFs=3
MF1='negativo':'trimf',[ -2.0 -1.0 0.0]
MF2='cero':'trimf',[ -1.0 0.0 1.0]
MF3='positivo':'trimf',[ 0.0 1.0 2.0]

[Input2]
Name='d_error'
Range=[-1 1]
NumMFs=3
MF1='negativo':'trimf',[ -2.0 -1.0 0.0]
MF2='cero':'trimf',[ -1.0 0.0 1.0]
MF3='positivo':'trimf',[ 0.0 1.0 2.0]

[Input3]
Name='d2_error'
Range=[-1 1]
NumMFs=3
MF1='negativo':'trimf',[ -2.0 -1.0 0.0]
MF2='cero':'trimf',[ -1.0 0.0 1.0]
MF3='positivo':'trimf',[ 0.0 1.0 2.0]

[Output1]
Name='scontrol'
Range=[-2 2]
NumMFs=3
MF1='negativa':'trimf',[ -3.0 -2.0 0]
MF2='cero':'trimf',[ -2.0 0 2.0]
MF3='positiva':'trimf',[ 0 2.0 3.0]

[Rules]
2 2 3, 2 (1.0) : 1
2 2 2, 2 (1.0) : 1
2 2 1, 2 (1.0) : 1
2 3 1, 3 (1.0) : 1
2 3 2, 2 (1.0) : 1
2 3 3, 2 (1.0) : 1
2 1 3, 1 (1.0) : 1
```

```

2 1 2, 2 (1.0) : 1
2 1 1, 2 (1.0) : 1
3 1 1, 3 (1.0) : 1
3 1 2, 3 (1.0) : 1
3 1 3, 1 (1.0) : 1
3 3 1, 2 (1.0) : 1
3 3 2, 1 (1.0) : 1
3 3 3, 2 (1.0) : 1
3 2 1, 3 (1.0) : 1
3 2 2, 3 (1.0) : 1
3 2 3, 3 (1.0) : 1
1 1 1, 1 (1.0) : 1
1 1 2, 1 (1.0) : 1
1 1 3, 2 (1.0) : 1
1 3 1, 3 (1.0) : 1
1 3 2, 3 (1.0) : 1
1 3 3, 2 (1.0) : 1
1 2 1, 1 (1.0) : 1
1 2 2, 1 (1.0) : 1
1 2 3, 1 (1.0) : 1
-2 1 0, 3 (1.0) : 1
-2 3 0, 1 (1.0) : 1

```

Controlador PI difuso:

```

[System]
Name='PI'
Type='mamdani'
Version=2.0
NumInputs=2
NumOutputs=1
NumRules=9
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'

[Input1]
Name='error'
Range=[-1 1]
NumMFs=3
MF1='negativo':'sigmf',[-12.0 -0.5]
MF2='cero':'dsigmf',[12.0 -0.5 12.0 0.5]
MF3='positivo':'sigmf',[12.0 0.5]

[Input2]
Name='d_error'
Range=[-2 2]
NumMFs=3
MF1='negativo':'sigmf',[-6.0 -1]
MF2='cero':'dsigmf',[6.0 -1 6.0 1]
MF3='positivo':'sigmf',[6.0 1]

[Output1]
Name='s_control'
Range=[-1 1]
NumMFs=3
MF1='negativa':'zmf',[-1 0]

```

```

MF2='cero':'pimf',[-1 0 0 1]
MF3='positiva':'smf',[0 1]

[Rules]
3 1, 3 (1.0) : 1
1 3, 1 (1.0) : 1
3 2, 3 (0.4999999999999956) : 1
1 2, 1 (0.5) : 1
2 1, 2 (0.2) : 1
2 2, 2 (0.2) : 1
2 3, 2 (0.2) : 1
3 3, 1 (1.0) : 1
1 1, 3 (1.0) : 1

```

Controlador PD difuso:

```

[System]
Name='PD'
Type='mamdani'
Version=2.0
NumInputs=2
NumOutputs=1
NumRules=9
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'

[Input1]
Name='error'
Range=[-1 1]
NumMFs=3
MF1='negativo':'gaussmf',[0.5 -1.0]
MF2='cero':'gaussmf',[0.3 0.0]
MF3='positivo':'gaussmf',[0.5 1.0]

[Input2]
Name='d_error'
Range=[-1 1]
NumMFs=3
MF1='negativo':'gaussmf',[0.5 -1.0]
MF2='cero':'gaussmf',[0.3 0.0]
MF3='positivo':'gaussmf',[0.5 1.0]

[Output1]
Name='s_control'
Range=[-1 1]
NumMFs=3
MF1='negativa':'trapmf',[-2.0 -1.5 -0.85 0.0]
MF2='cero':'trapmf',[-1.0 -0.1 0.1 1.0]
MF3='positiva':'trapmf',[0.0 0.85 1.5 2.0]

[Rules]
2 1, 2 (0.5) : 1
2 2, 2 (1.0) : 1
2 3, 2 (0.5) : 1
3 3, 3 (1.0) : 1
3 2, 3 (1.0) : 1

```

```

3 1, 2 (0.25) : 1
1 1, 1 (1.0) : 1
1 2, 1 (1.0) : 1
1 3, 2 (0.25) : 1

```

Controlador programador de ganancias (PDG) difuso:

```

[System]
Name='PDG'
Type='mamdani'
Version=2.0
NumInputs=2
NumOutputs=3
NumRules=9
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'

[Input1]
Name='Error'
Range=[-1 1]
NumMFs=3
MF1='negativo':'trimf',[-2.0 -1.0 0.0]
MF2='cero':'trimf',[-1.0 0.0 1.0]
MF3='positivo':'trimf',[0.0 1.0 2.0]

[Input2]
Name='Derivada del Error'
Range=[-1 1]
NumMFs=3
MF1='negativa':'trimf',[-2.0 -1.0 0.0]
MF2='cero':'trimf',[-1.0 0.0 1.0]
MF3='positiva':'trimf',[0.0 1.0 2.0]

[Output1]
Name='Kp'
Range=[0 2]
NumMFs=3
MF1='Cero':'trimf',[-1 0 1]
MF2='alto':'trimf',[0 1 2]
MF3='muy alto':'trimf',[1 2 3]

[Output2]
Name='Ki'
Range=[0 2]
NumMFs=3
MF1='Cero':'trimf',[-1 0 1]
MF2='alto':'trimf',[0 1 2]
MF3='muy alto':'trimf',[1 2 3]

[Output3]
Name='Kd'
Range=[0 1]
NumMFs=3
MF1='Cero':'trimf',[-1.0 0 0.5]
MF2='alto':'trimf',[0 0.5 1]

```

```

MF3='muy alto':'trimf',[0.5 1 2]

[Rules]
1 1, 1 1 2 (1.0) : 1
1 2, 2 2 2 (1.0) : 1
1 3, 3 3 2 (1.0) : 1
2 1, 1 1 1 (1.0) : 1
2 2, 2 2 1 (1.0) : 1
2 3, 2 2 1 (1.0) : 1
3 1, 3 3 3 (1.0) : 1
3 2, 3 3 2 (1.0) : 1
3 3, 1 1 2 (1.0) : 1

```

Controlador difuso simple:

```

[System]
Name='simpleF'
Type='mamdani'
Version=2.0
NumInputs=1
NumOutputs=1
NumRules=5
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'

[Input1]
Name='error'
Range=[-1 1]
NumMFs=5
MF1='negativo grande':'trimf',[-2 -1 -0.5]
MF2='negativo':'trimf',[-1 -0.5 0]
MF3='cero':'trimf',[-0.5 0 0.5]
MF4='positivo':'trimf',[0 0.5 1.0]
MF5='positivo grande':'trimf',[0.5 1.0 2]

[Output1]
Name='s_fuzzy'
Range=[-1 1]
NumMFs=5
MF1='negativa grande':'trimf',[-2 -1 -0.5]
MF2='negativa':'trimf',[-1 -0.5 0]
MF3='cero':'trimf',[-0.5 0.0 0.5]
MF4='positiva':'trimf',[0 0.5 1.0]
MF5='positiva grande':'trimf',[0.5 1.0 1.667]

[Rules]
5, 5 (1.0) : 1
4, 4 (0.4999999999999956) : 1
3, 3 (0.0999999999999924) : 1
2, 2 (0.4999999999999956) : 1
1, 1 (1.0) : 1

```

[Anexo O]
 [Comparación de simulación de sistemas de control]

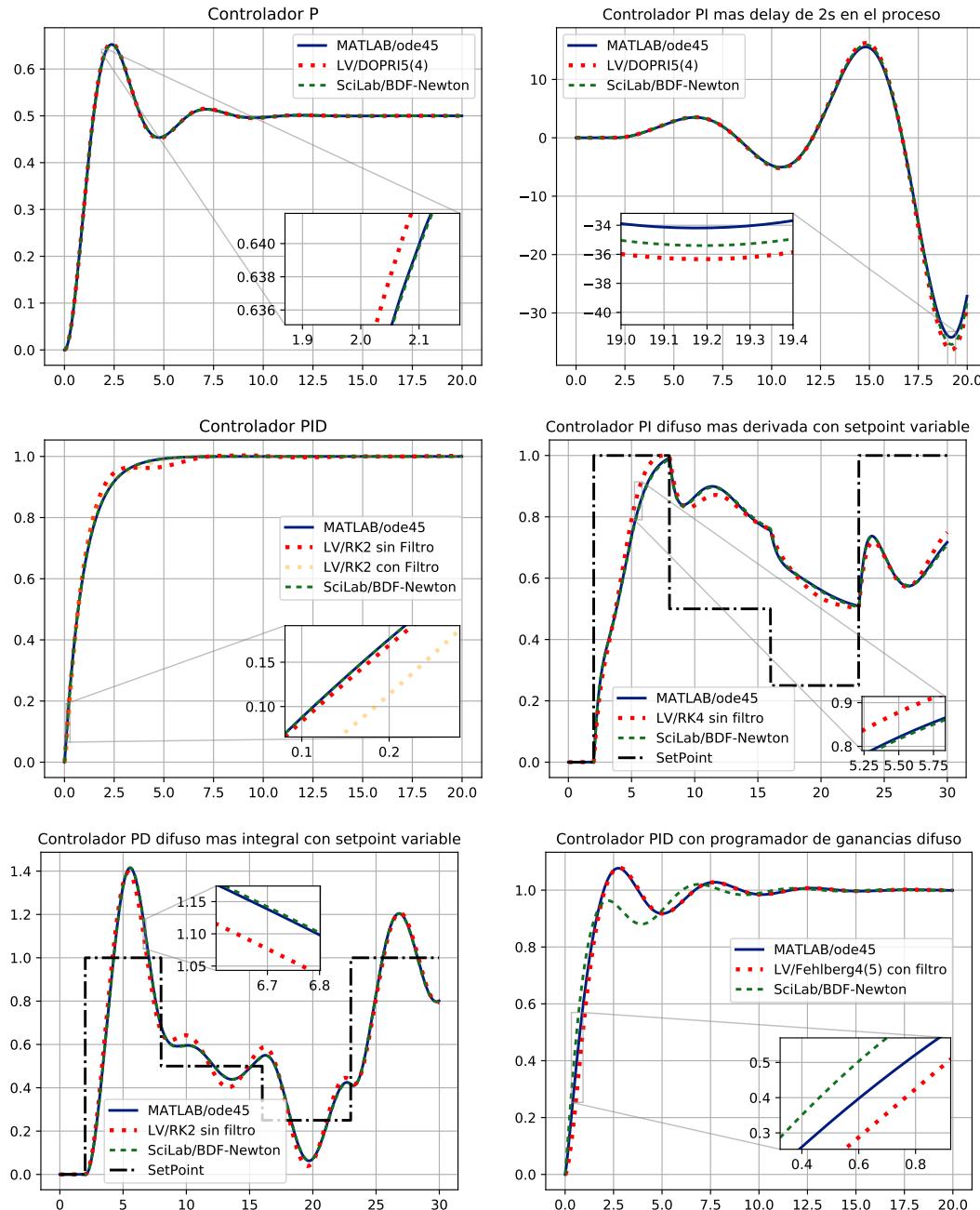


Figura 51. Comparación de simulación de sistemas de control en tiempo continuo - 1. Controladores: P, PI, PID, PI difuso + D, PD difuso más I y PDG. Fuente: Elaboración propia.

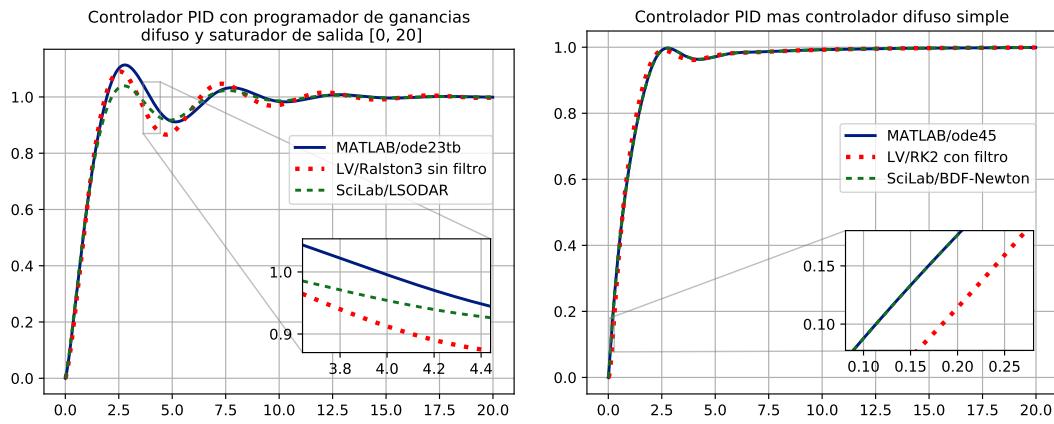


Figura 52. Comparación de simulación de sistemas de control en tiempo continuo - 2. Controladores: PDG con saturador y PID + difuso simple. Fuente: Elaboración propia.

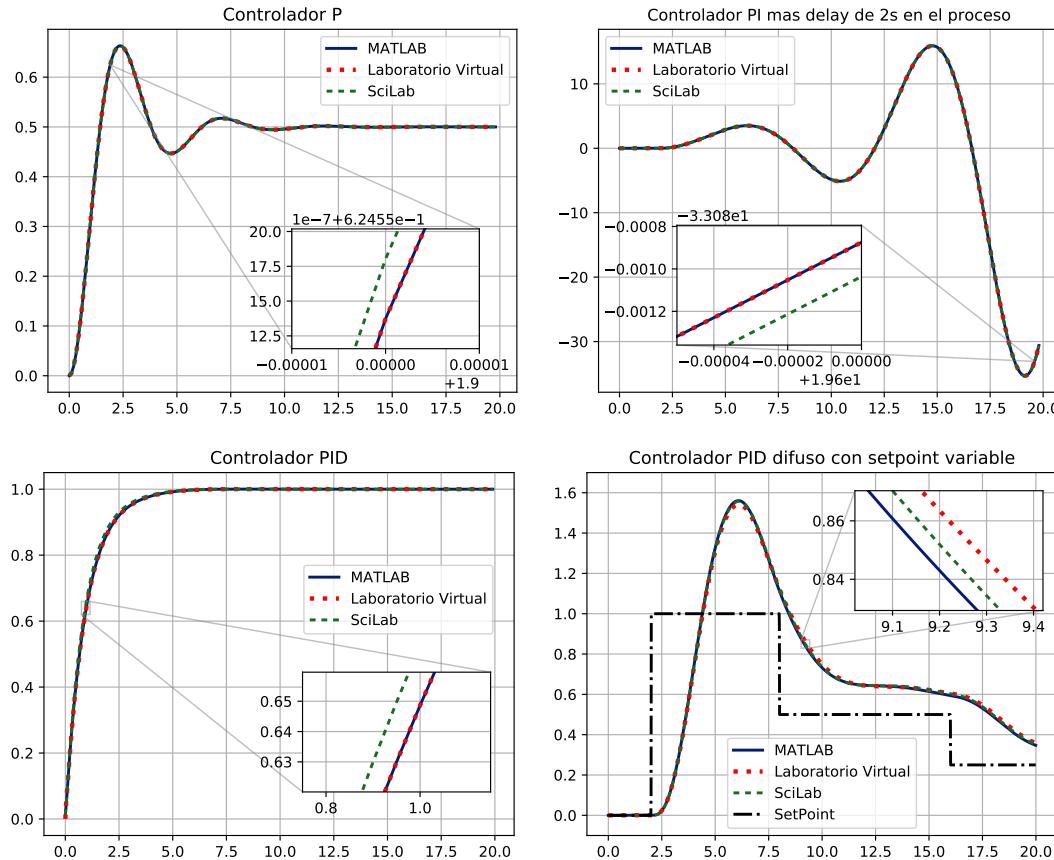


Figura 53. Comparación de simulación de sistemas de control en tiempo discreto - 1. Controladores: P, PI, PID y PID difuso. Fuente: Elaboración propia.

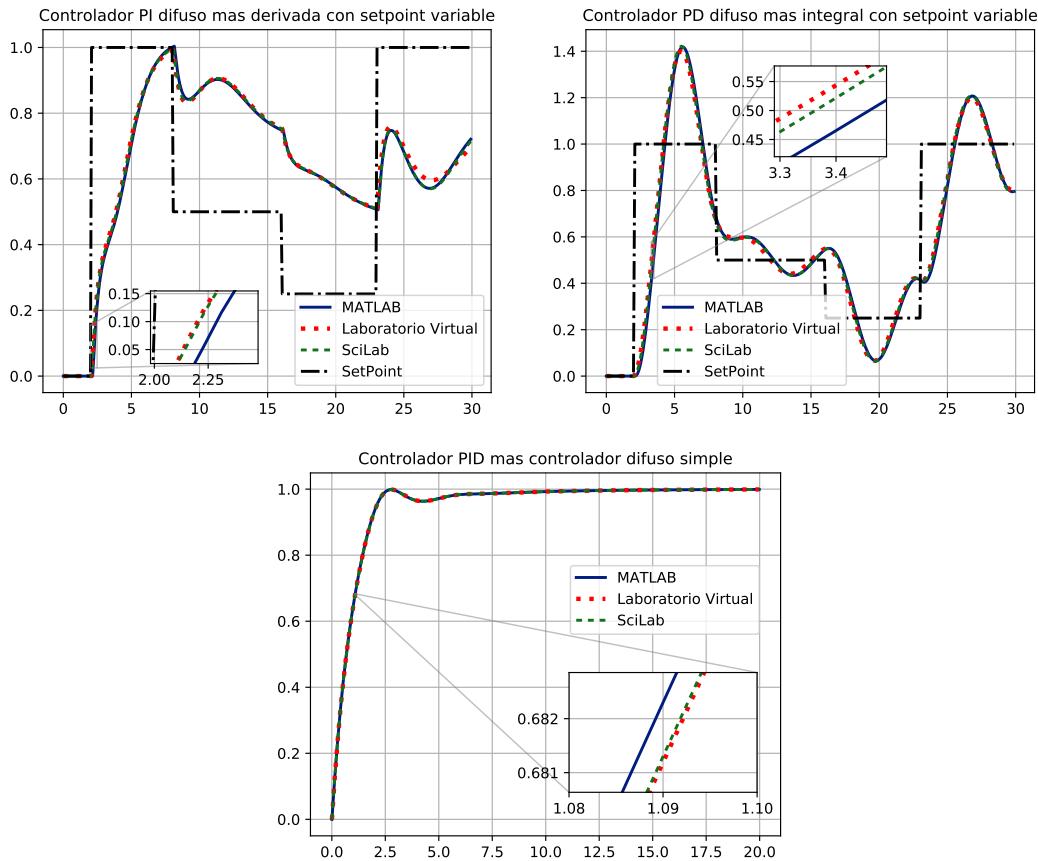


Figura 54. Comparación de simulación de sistemas de control en tiempo discreto - 2. Controladores: PI difuso + D, PD difuso + I y PID + difuso simple. Fuente: Elaboración propia.