

UCUENCA

Universidad de Cuenca

Facultad de Ingeniería

Carrera de Electrónica y Telecomunicaciones

**Diseño e implementación de un controlador PID de ganancia programada
usando técnicas de inteligencia artificial**

Trabajo de titulación previo a la obtención del título de Ingeniero en Electrónica y Telecomunicaciones

Autor:

Michael Alexander Criollo Gordillo

Director:

Luis Ismael Minchala Ávila

ORCID: 0000-0003-0822-0705

Cuenca, Ecuador

2023-03-07

Resumen

En este trabajo se analizan tres estrategias nuevas de control que mejoran la respuesta del sistema y se comparan con el control PID clásico. La eficiencia del controlador PID depende únicamente de sus ganancias K_p , K_i y K_d . Por ello, las estrategias mejoran el proceso de control al encontrar las ganancias óptimas de manera programada. Estos algoritmos se basan en técnicas de inteligencia artificial como la lógica difusa, recocido simulado y el método de búsqueda A*, dando lugar a los controladores FGS-PID, SA-PID y A*-PID. Los algoritmos de control son probados en MATLAB y adaptados para su correcto funcionamiento sobre el sistema embebido STM32 núcleo-144 para controlar el nivel de agua de un sistema multitanque. Para ello, se realizan distintas pruebas de funcionamiento en simulaciones y sobre el sistema real, donde se evalúa la respuesta de la planta ante cambios de referencia, utilizando como índice de desempeño el tiempo de estabilización y porcentaje de sobreimpulso. Asimismo, se comprueba la operación del sistema de control, introduciendo posibles fallas en la planta a través de fugas en los tanques provocados por un agujero en el fondo. Finalmente, los resultados muestran la efectividad de los algoritmos al reducir el tiempo de estabilización y sobreimpulso en la respuesta del sistema multitanque, además de agregar cierta tolerancia a fallas en el sistema. Lo anterior permite el seguimiento de la referencia en presencia de fallas, lo cual no se consigue con el sistema de control PID clásico.

Palabras clave: controlador PID, inteligencia artificial, programación de ganancias, sistema multitanque

Abstract

This paper analyzes three new control strategies that improve system response and its comparison with classic PID control. The efficiency of the PID controller depends only on its gains K_p , K_i and K_d . Therefore, the strategies improve the control process by finding the optimal gains through the programmed algorithm. The programmed gain control algorithms are based on artificial intelligence techniques such as fuzzy logic, the simulated annealing algorithm and the A* search method, resulting in the FGS-PID, SA-PID and A*-PID controllers. The control algorithms are tested in MATLAB and adapted on the STM32 core-144 embedded system, with the objective of controlling the water level of a multitank system. For this purpose, different performance tests are carried out in simulations and on the real system where the response of the plant to reference changes is evaluated. In addition, the settling time and percentage of overshoot are used as performance indexes. Also, the operation of the control system is tested by introducing possible failures in the plant, such as through leaks in the tanks caused by a hole in the bottom. Finally, the results show the effectiveness of the algorithms in reducing the settling time and overshoot in the response of the multitank system, furthermore, adding some tolerance to failures in the system. The aforementioned results allow the tracking of the reference in the presence of faults, which is not achieved with the classical PID control system.

Keywords: PID controller, artificial intelligence, gain scheduling, multitank system

Índice de contenidos

1. Introducción	20
1.1. Identificación del problema	20
1.2. Justificación	20
1.3. Alcance	20
1.4. Objetivos	21
1.4.1. Objetivo general	21
1.4.2. Objetivos específicos	21
1.5. Contribución	21
2. Fundamentos teóricos	23
2.1. Inteligencia artificial	23
2.1.1. Fundamentos de la lógica difusa	23
2.1.2. Algoritmo recocido simulado	30
2.1.3. Algoritmo A*	32
2.2. Algoritmos de control	34
2.2.1. Control proporcional-integral-derivativo (PID)	34
2.2.2. Control on-off	40
2.2.3. Control proporcional de tiempo variable	40
2.2.4. Controlador PID difuso de ganancias programables (FGS-PID)	41
3. Estado del arte	46
3.1. Estado del arte de los algoritmos de inteligencia artificial implementados en sistemas de control	46
3.2. Estado del arte del sistema multitanque y las estrategias de control aplicadas .	48
3.3. Estado del arte de los sistemas de control basados en STM32	49
4. Metodología	51
4.1. Diseño e implementación de los algoritmos de ganancia programada en MATLAB	51
4.1.1. Controlador PID con ganancia programada basado en lógica difusa (FGS- PID)	51
4.1.2. Controlador PID con ganancia programada basado en el algoritmo reco- cido simulado (SA-PID)	58
4.1.3. Controlador PID con ganancia programada basado en el algoritmo A estrella (A*-PID)	61

4.2. Simulación del controlador PID con ganancia programada y sistema multitanque	64
4.2.1. Modelado del sistema multitanque	64
4.2.2. Diseño del sistema de control	66
4.2.3. Simulación del sistema en MATLAB	68
4.3. Implementación y adaptación de los algoritmos de ganancia programada y controladores PID en la placa STM32 núcleo-144	77
4.3.1. Adaptación de las funciones básicas	77
4.3.2. Adaptación de las funciones avanzadas	78
4.4. Integración de la placa STM32 al tablero de control y comunicación con el sistema multitanque	84
4.4.1. Descripción del sistema	85
4.4.2. Diagrama de instrumentación y procesos del sistema	86
4.4.3. Principales componentes del sistema multitanque y comunicación con el tablero de control	87
4.4.4. Descripción del hardware	91
5. Resultados	94
5.1. Resultados en la simulación	94
5.2. Resultados del sistema multitanque	103
5.2.1. Funcionamiento del sistema multitanque y sistema de control	103
5.2.2. Resultados de las pruebas de funcionamiento sin fugas	105
5.2.3. Resultados de las pruebas de funcionamiento con fugas	109
6. Conclusiones y trabajos futuros	116
6.1. Conclusiones	116
6.2. Trabajos futuros	118
Apéndice	120
A. Controlador PID	121
A.1. MATLAB	121
A.1.1. Determinación de la respuesta del sistema en un lazo de control PID	121
A.2. STM32CubeIDE	121
A.2.1. Inicialización de variables y determinación de la señal de control del algoritmo de control PID	121

B. Controlador FGS-PID	123
B.1. MATLAB	123
B.1.1. Implementación del controlador FGS-PID para un sistema representado en espacio de estados	123
B.1.2. Archivo fuzzy_control.fis	123
B.1.3. Implementación del controlador FGS-PID para el control del sistema multitanque en Simulink	125
B.2. STM32CubeIDE	127
B.2.1. Generación de las funciones de membresía y definición de las reglas difusas	127
B.2.2. Evaluación del sistema de inferencia difuso	130
B.2.3. Determinación de las ganancias mediante el algoritmo FGS-PID	133
C. Controlador SA-PID	134
C.1. MATLAB	134
C.1.1. Implementación del algoritmo de control SA-PID	134
C.1.2. Determinación del costo de la función objetivo	134
C.1.3. Determinación de valores vecinos de ganancia	135
C.1.4. Implementación del controlador SA-PID para el control del sistema mul- titanque en Simulink	135
C.2. STM32CubeIDE	138
C.2.1. Inicialización del algoritmo SA-PID	138
C.2.2. Determinación de las ganancias mediante el algoritmo SA-PID	139
C.2.3. Evaluación de la función objetivo	141
C.2.4. Determinación de valores vecinos de ganancia	143
D. Controlador A*-PID	145
D.1. MATLAB	145
D.1.1. Determinación de las ganancias mediante el algoritmo A*-PID	145
D.1.2. Generación de un identificador único a cada nodo	146
D.1.3. Implementación del controlador A*-PID para el control del sistema multi- tanque en Simulink	147
D.2. STM32CubeIDE	151
D.2.1. Inicialización del algoritmo A*-PID	151
D.2.2. Generación de un identificador único a cada nodo	154

D.2.3. Determinación de las ganancias mediante el algoritmo A*-PID	154
--	-----

Bibliografía	159
---------------------	------------

Anexos	166
---------------	------------

Anexo A: Diseño de los tanques, armazón de la planta del sistema multitanque y tablero de control	166
--	-----

Índice de figuras

2.1. Diagrama de bloques de un sistema de control en lazo cerrado.	34
2.2. Repuesta de la planta a un escalón unitario. <i>Imagen adaptada de [1].</i>	37
2.3. Método de la ganancia última, oscilaciones sostenidas. <i>Imagen adaptada de [1].</i>	38
2.4. La respuesta tiempo-temperatura de un controlador On-Off. <i>Imagen adaptada de [2].</i>	40
2.5. Señal PWM, (relación entre t_{on} y t_c). <i>Imagen adaptada de [3].</i>	41
2.6. Controlador PID difuso de ganancias programables. <i>Imagen adaptada de [4].</i>	41
2.7. Funciones de membresía para la variable del error $e(k)$ y derivada del error $\Delta e(k)$. <i>Elaborada con datos de [4].</i>	44
2.8. Funciones de membresía para las variables K_p' y K_d' . <i>Elaborada con datos de [4].</i>	44
2.9. Funciones de membresía para la variable α . <i>Elaborada con datos de [4].</i>	45
4.1. Funciones de membresía para la variable K_i'	54
4.2. Fuzzy Logic Designer de MATLAB.	55
4.3. Edición de las funciones de membresía.	56
4.4. Edición de las reglas difusas.	56
4.5. Comparación del controlador PID y FGS-PID.	58
4.6. Comparación del controlador PID y SA-PID	60
4.7. Espacio de búsqueda	62
4.8. Comparación del controlador PID y A*-PID	64
4.9. Esquema simplificado del sistema multitanque. <i>Imagen adaptada de [5].</i>	65
4.10. Sistema de control PID con ganancia programada y sistema multitanque.	67
4.11. Sistema de control PID con ganancia programada y sistema multitanque en Simulink.	68
4.12. Subsistema Parámetros.	69
4.13. Subsistema Errores	69
4.14. Subsistema del modelo del sistema de multitanque.	70
4.15. Controlador PID.	71
4.16. Bloque del controlador FGS-PID.	73
4.17. Bloque del controlador SA-PID.	74
4.18. Bloque del controlador A*-PID.	75
4.19. Definición del espacio de búsqueda del controlador A*-PID.	75

4.20. Determinación de los nodos sucesores del controlador A*-PID.	76
4.21. Diagrama de flujo del algoritmo SA-PID.	80
4.22. Diagrama de flujo del algoritmo A*-PID.	83
4.23. Esquema general del sistema multitaque y sistema de control.	85
4.24. Diagrama de proceso e instrumentación del sistema multitanque y sistemas de control. <i>Imagen adaptada de [5].</i>	86
4.25. Bomba Favson F3012. <i>Imagen tomada de [5].</i>	87
4.26. Electroválvula Winner WVA4-3. <i>Imagen tomada de [5].</i>	87
4.27. Electroválvula BACOENG 2W-15. <i>Imagen tomada de [6].</i>	88
4.28. Electroválvula U.S. SOLID USSMSV00002. <i>Imagen tomada de [7].</i>	88
4.29. Sensor ultrasónico HC-SR04. <i>Imagen tomada de [8].</i>	89
4.30. Sensor de flujo de agua DIGITEN. <i>Imagen tomada de [9].</i>	89
4.31. Placa STM32F413ZH núcleo-144. <i>Imagen tomada de [10].</i>	90
4.32. Módulo relé de 4 canales. <i>Imagen tomada de [11].</i>	90
4.33. Convertidor de PWM a voltaje analógico 0-10V. <i>Imagen tomada de [12].</i>	91
4.34. Diseño del sistema multitanque. <i>Imagen tomada de [5].</i>	92
4.35. Diseño del tablero de control	93
4.36. Distribución de los terminales de conexión en la parte frontal del tablero de control	93
5.1. Respuesta del sistema en la simulación ante cambios de referencia utilizando el algoritmo de control PID.	95
5.2. Respuesta del sistema en la simulación ante cambios de referencia utilizando el algoritmo de control FGS-PID.	96
5.3. Respuesta del sistema en la simulación ante cambios de referencia utilizando el algoritmo de control SA-PID.	97
5.4. Respuesta del sistema en la simulación ante cambios de referencia utilizando el algoritmo de control A*-PID.	98
5.5. Respuesta del sistema en la simulación ante presencia de fugas en el tanque 1 utilizando un controlador PID.	99
5.6. Respuesta del sistema en la simulación ante presencia de fugas en el tanque 1 utilizando un controlador FGS-PID.	99
5.7. Respuesta del sistema en la simulación ante presencia de fugas en el tanque 1 utilizando un controlador SA-PID.	100
5.8. Respuesta del sistema en la simulación ante presencia de fugas en el tanque 1 utilizando un controlador A*-PID.	100

5.9. Respuesta del sistema en la simulación ante presencia de fugas en el tanque 2 utilizando un controlador PID.	101
5.10. Respuesta del sistema en la simulación ante presencia de fugas en el tanque 2 utilizando un controlador FGS-PID.	102
5.11. Respuesta del sistema en la simulación ante presencia de fugas en el tanque 2 utilizando un controlador SA-PID.	102
5.12. Respuesta del sistema en la simulación ante presencia de fugas en el tanque 2 utilizando un controlador A*-PID.	103
5.13. Sistema multitanque construido.	104
5.14. Tablero de control construido.	104
5.15. Estrategias para reducir el error en la medición.	105
5.16. Sistema de fugas del sistema multitanque construido.	105
5.17. Respuesta del sistema multitanque ante cambios de referencia utilizando el al- goritmo de control PID.	106
5.18. Respuesta del sistema multitanque ante cambios de referencia utilizando el al- goritmo de control FGS-PID.	107
5.19. Respuesta del sistema multitanque ante cambios de referencia utilizando el al- goritmo de control SA-PID.	108
5.20. Respuesta del sistema multitanque ante cambios de referencia utilizando el al- goritmo de control A*-PID.	109
5.21. Respuesta del sistema ante presencia de fugas en el tanque 1 utilizando el algoritmo de control PID.	110
5.22. Respuesta del sistema ante presencia de fugas en el tanque 1 utilizando el algoritmo de control FGS-PID.	110
5.23. Respuesta del sistema ante presencia de fugas en el tanque 1 utilizando el algoritmo de control SA-PID.	111
5.24. Respuesta del sistema ante presencia de fugas en el tanque 1 utilizando el algoritmo de control A*-PID.	112
5.25. Respuesta del sistema ante presencia de fugas en el tanque 2 utilizando el algoritmo de control PID.	112
5.26. Respuesta del sistema ante presencia de fugas en el tanque 2 utilizando el algoritmo de control FGS-PID.	113
5.27. Respuesta del sistema ante presencia de fugas en el tanque 2 utilizando el algoritmo de control SA-PID.	114

5.28. Respuesta del sistema ante presencia de fugas en el tanque 2 utilizando el algoritmo de control A*-PID.	114
--	-----

Índice de tablas

2.1. Acciones básicas de control. <i>Tomada de [13].</i>	35
2.2. Regla de sintonía de Ziegler-Nichols (primer método). <i>Elaborada con datos de [1].</i>	37
2.3. Regla de sintonía de Ziegler-Nichols (segundo método). <i>Elaborada con datos de [1].</i>	38
2.4. Reglas de sintonía por criterios integrales. <i>Elaborada con datos de [14].</i>	39
2.5. Reglas difusas de sintonía para K'_p . <i>Elaborada con datos de [4].</i>	43
2.6. Reglas difusas de sintonía para K'_d . <i>Elaborada con datos de [4].</i>	43
2.7. Reglas difusas de sintonía para α . <i>Elaborada con datos de [4].</i>	43
4.1. Reglas difusas de sintonía para K'_i	54
4.2. Constantes físicas modelo del sistema multitanque. <i>Elaborada con datos de [5].</i> .	66
4.3. Parámetros de los controladores PID.	72
4.4. Parámetros de los controladores FGS-PID.	73
5.1. Resultados ante cambios de referencias utilizando control PID.	95
5.2. Resultados ante cambios de referencias utilizando control FGS-PID.	96
5.3. Resultados ante cambios de referencias utilizando control SA-PID.	97
5.4. Resultados ante cambios de referencias utilizando control A*-PID.	98
5.5. Resultados en la simulación ante presencia de fugas en el tanque 1 utilizando un controlador PID.	99
5.6. Resultados en la simulación ante presencia de fugas en el tanque 1 utilizando un controlador FGS-PID.	99
5.7. Resultados en la simulación ante presencia de fugas en el tanque 1 utilizando un controlador SA-PID.	100
5.8. Resultados en la simulación ante presencia de fugas en el tanque 1 utilizando un controlador A*-PID.	101
5.9. Resultados en la simulación ante presencia de fugas en el tanque 2 utilizando un controlador PID.	101
5.10. Resultados en la simulación ante presencia de fugas en el tanque 2 utilizando un controlador FGS-PID.	102
5.11. Resultados en la simulación ante presencia de fugas en el tanque 2 utilizando un controlador SA-PID.	102

5.12. Resultados en la simulación ante presencia de fugas en el tanque 2 utilizando un controlador A*-PID.	103
5.13. Resultados del sistema multitanque utilizando control PID.	106
5.14. Resultados del sistema multitanque utilizando control FGS-PID.	107
5.15. Resultados del sistema multitanque utilizando control SA-PID.	108
5.16. Resultados del sistema multitanque utilizando control A*-PID.	109
5.17. Resultados ante presencia de fugas en el tanque 1 utilizando control PID.	110
5.18. Resultados ante presencia de fugas en el tanque 1 utilizando control FGS-PID.	111
5.19. Resultados ante presencia de fugas en el tanque 1 utilizando control SA-PID.	111
5.20. Resultados ante presencia de fugas en el tanque 1 utilizando control A*-PID.	112
5.21. Resultados ante presencia de fugas en el tanque 2 utilizando control PID.	113
5.22. Resultados ante presencia de fugas en el tanque 2 utilizando control FGS-PID.	113
5.23. Resultados ante presencia de fugas en el tanque 2 utilizando control SA-PID.	114
5.24. Resultados ante presencia de fugas en el tanque 2 utilizando control A*-PID.	114
6.1. Comparación de resultados del sistema multitanque simulado.	117
6.2. Comparación de resultados del sistema multitanque real.	118

Índice de extractos de código

A.1. Función PIDcontrol().	121
A.2. Función PID_Init().	121
A.3. Función PID_control().	122
B.1. Algoritmo de control FGS-PID en MATLAB.	123
B.2. Archivo fuzzy_control.fis	123
B.3. Función sfun_fuzzycontrol.	125
B.4. Función Fuzzy_Init.	127
B.5. Función gaussmf.	130
B.6. Función trimf.	130
B.7. Función trapmf.	130
B.8. Función eval_fuzzy.	130
B.9. Función fuzz.	131
B.10.Función defuzz.	132
B.11.Función Reglas_str2num.	132
B.12.Función find_index.	132
B.13.Función FGS_Init.	133
B.14.Función FGS_control.	133
C.1. Algoritmo de control SA-PID en MATLAB.	134
C.2. Función CostFunction.	134
C.3. Función CreateNeighbor.	135
C.4. Función sfun_SA.	135
C.5. Función SA_Init.	138
C.6. Función Simulated_Annealing.	139
C.7. Función calcule_cost.	141
C.8. Función SettlingTime.	141
C.9. Función Overshoot.	142
C.10.Función IAE_calc.	142
C.11.Función ITAE_calc.	142
C.12.Función ISE_calc.	143
C.13.Función ITSE_calc.	143
C.14.Función RiseTime.	143
C.15.Función CreateNeighbor.	143
D.1. Algoritmo de control A*-PID en MATLAB	145

D.2. Funcion GenerateID	146
D.3. Función sfun_Astar.	147
D.4. Función Astar_Init.	151
D.5. Función reserve_memory_Astar.	152
D.6. Función generate_ID.	154
D.7. Función Aestrella.	154
D.8. Función delimitar_coordenadas.	157
D.9. Función contador.	157

Dedicatoria

A mis padres:

Quienes me apoyan en todo momento, brindándome su amor y paciencia.

A mis hermanos Mateo y Anthony:

Con los que comparto los mejores momentos.

A Claudia P.

Quien es la compañía perfecta, brindándome su motivación.

Agradecimientos

Agradezco a mi familia por el apoyo incondicional que me han brindado, en el transcurso de mi vida y en mi formación académica.

También, agradezco a mi tutor el ingeniero Ismael Minchala, por brindar parte de su tiempo y conocimientos en todo el proceso que hizo posible la culminación de este trabajo.

Finalmente, a mis amigos, quienes me han acompañado en esta etapa de mi vida.

Michael Alexander Criollo Gordillo

Abreviaciones y Acrónimos

A* A Star. [21](#), [23](#), [32](#), [46](#), [47](#), [61–63](#), [67](#), [76](#), [82](#), [84](#), [97](#), [98](#), [100](#), [102](#), [105](#), [108](#), [111](#), [114](#), [116–119](#)

ASA Adaptive Simulated Annealing. [47](#)

BA Bat Algorithm. [46](#)

BFS Best First Search. [47](#)

DE Differential Evolution. [46](#)

FA Firefly Algorithm. [46](#)

FGS Fuzzy Gain Scheduling. [21](#), [41–43](#), [47](#), [48](#), [57](#), [60](#), [63](#), [67](#), [73](#), [74](#), [76](#), [95](#), [99–102](#), [106](#), [110](#), [113](#), [116](#), [117](#)

FIS Fuzzy Inference System. [57](#)

FLC Fuzzy Logic Control. [46](#)

GA Genetic Algorithm. [46](#), [47](#)

GCC GNU Compiler Collection. [77](#)

GS Gain Scheduling. [66](#), [68](#), [77](#), [117](#), [119](#), [120](#)

IA Artificial Intelligence. [23](#), [46](#), [64](#)

IAE Integral Absolute Error. [38](#), [39](#), [47](#), [48](#), [58](#), [59](#), [81](#)

IDE Integrated Development Environment. [77](#)

ISE Integral Square Error. [39](#), [47](#)

ITAE Integral of Time multiplied Absolute Error. [38](#), [39](#), [46](#), [47](#)

ITSE Integral of Time multiplied Square-Error. [39](#), [47](#)

LQR Linear Quadratic Regulator. [47](#)

MIMO Multiple-Input Multiple-Output. [64](#), [71](#)

MSE Mean Squared Error. [47](#)

PD Proporcional–Derivativo. [52](#)

PI Proportional–Integral. [39](#)

PID Proportional–Integral–Derivative. [20–22](#), [34](#), [36](#), [37](#), [39](#), [41–43](#), [46–52](#), [57–61](#), [63](#), [64](#), [67](#), [68](#), [70](#), [71](#), [73–76](#), [79](#), [81](#), [82](#), [84](#), [94–103](#), [105–114](#), [116–119](#)

PSO Particle Swarm Optimization. [46](#)

PWM Pulse Width Modulation. [40](#), [48–50](#), [85](#), [90](#), [103](#)

SA Simulated Annealing. [21](#), [23](#), [30](#), [46](#), [47](#), [59–61](#), [67](#), [73–76](#), [79](#), [81](#), [96](#), [100](#), [102](#), [103](#), [105](#), [107](#), [111](#), [113](#), [114](#), [116–118](#)

ZN Ziegler-Nichols. [20](#), [47](#), [57](#), [72](#)

Introducción

En este capítulo se describe la identificación del problema, justificación, alcance, objetivos y las contribuciones del proyecto.

1.1. Identificación del problema

Hoy en día el controlador **Proportional–Integral–Derivative** (PID) es el más utilizado en los procesos de control. Esto obedece a razones de simplicidad algorítmica, facilidad de uso y buen comportamiento. Actualmente, existen métodos más eficientes para el despliegue de sistemas de control. No obstante, en el caso de los procesos industriales, la mayoría de los controladores, cerca del 90 %, utilizan algoritmos PID [15].

El controlador PID ha demostrado ser el adecuado en muchas aplicaciones industriales. Sin embargo, su uso generalizado no siempre resulta en resultados eficientes, ya que no todos los procesos son lineales y/o con poco retardo en el tiempo, donde este controlador se comporta de mejor manera. En la práctica, los procesos presentan no linealidades generadas por perturbaciones y, en ocasiones, sistemas con grandes retardos o con múltiples entradas y salidas. Por tal motivo, para encontrar una solución óptima se han diseñado distintos métodos para sintonizar las ganancias: proporcional (K_p), integral (K_i) y derivativa (K_d) de un controlador PID. Entre las estrategias más primitivas, se encuentra la sintonización de estos parámetros mediante el método de **Ziegler-Nichols** (ZN). Mientras tanto, otras estrategias basadas en inteligencia artificial, corresponden a la sintonización de ganancias mediante algoritmos genéticos, redes neuronales, álgebra de lógica difusa, etc.

1.2. Justificación

La eficiencia del controlador PID depende de la sintonización de las ganancias: K_p , K_i y K_d . Por lo tanto, las técnicas de inteligencia artificial ajustan estos parámetros a los valores óptimos para obtener la respuesta del sistema de control deseada. Además, se logra que el sistema sea tolerante a fallas.

1.3. Alcance

El alcance del proyecto consiste en explorar técnicas de inteligencia artificial que permitan sintonizar las ganancias de un controlador PID para brindar, entre otras cosas, tolerancia a

fallas. Entre las estrategias que se abordarán, están los algoritmos de búsqueda A estrella (A*- A Star), recocido simulado (SA- Simulated Annealing) y el uso de la lógica difusa en un controlador PID de ganancia programada difusa (FGS-PID). El sistema de control y los algoritmos de ajuste de ganancia, en un principio, son probados en MATLAB y luego adaptados para su correcto funcionamiento sobre el sistema embebido STM32 núcleo-144, para el control de nivel de un sistema multitanque.

1.4. Objetivos

1.4.1. Objetivo general

Explorar técnicas de inteligencia artificial que permitan ajustar las ganancias de un controlador PID para mejorar su desempeño. Caso de estudio, sistema multitanque.

1.4.2. Objetivos específicos

- Revisar el estado del arte de los algoritmos de inteligencia artificial en el contexto de controladores PID con ganancia programada.
- Diseñar e implementar algoritmos de ganancia programada en MATLAB.
- Implementar los algoritmos de ganancia programada y controladores PID en la placa de desarrollo STM32 núcleo-144.
- Desplegar el sistema de control PID con ganancia programada sobre un sistema multitanque.
- Realizar pruebas de funcionamiento del controlador PID con ganancia programada y el sistema multitanque.
- Analizar el rendimiento de los algoritmos de ganancia programada ante presencia de fallas en la planta.

1.5. Contribución

Las contribuciones del proyecto son:

- Diseño e implementación de los algoritmos de ganancia programada en MATLAB.

- Diseño e implementación de un sistema de control de ganancia programada sobre el sistema embebido STM32 núcleo-144.
- Sistema de control PID con ganancias programadas integrado en un tablero de control y regulación de un sistema multitanque.

Fundamentos teóricos

En este capítulo se describen los fundamentos teóricos de los algoritmos de inteligencia artificial. Entre los que se detallan están los algoritmos de búsqueda A estrella ([A*](#)), recocido simulado ([SA](#)) y los fundamentos teóricos de la lógica difusa. Además, se describen los principales conceptos de los algoritmos de control.

2.1. Inteligencia artificial

El término inteligencia artificial ([IA](#)) se conoce desde hace décadas; sin embargo, no se ha consolidado una definición concreta del mismo. Lo antes mencionado se debe, principalmente, a que se utiliza en distintos ámbitos tecnológicos y de la ciencia. En [\[16\]](#) se reúnen varias definiciones de inteligencia artificial, de distintos autores, separadas en cuatro categorías:

- **Actuar humanamente:** Sistemas capaces de realizar funciones que requieren inteligencia como las personas [\[17\]](#).
- **Actuar racionalmente:** El estudio de los agentes inteligentes [\[18\]](#) y el comportamiento inteligente en los artefactos [\[19\]](#).
- **Pensar humanamente:** Sistemas capaces de simular el pensamiento inteligente de los humanos [\[20\]](#). La automatización de actividades como las decisiones, aprendizaje y resolución de problemas; actividades asociadas al pensamiento humano. [\[21\]](#).
- **Pensar racionalmente:** El estudio de las capacidades mentales con modelos computacionales [\[22\]](#), y cálculos que permiten razonar, percibir y actuar [\[23\]](#).

Existen varias definiciones de [IA](#); sin embargo, en este proyecto se enfatizan los aportes brindados en la informática. Dentro de este campo, se describen algunas de las técnicas clásicas de búsqueda y de razonamiento de la [IA](#), que son utilizados en la mayoría de resolución de problemas.

2.1.1. Fundamentos de la lógica difusa

La lógica difusa fue propuesta por Lotfi Zadeh en 1965, en su primera publicación sobre conjuntos difusos en [\[24\]](#). Más adelante, Zadeh continuaría el desarrollo de la teoría de lógica difusa y conjuntos borrosos. A pesar de que, en aquella época, se consideraba innecesaria; la lógica difusa era tratada como una teoría ambigua que generaba imprecisión. A pesar de todo

esto, en la actualidad, su importancia destaca en el área de la inteligencia artificial. De hecho, se utiliza principalmente en el control de sistemas complejos de la industria y, de manera general, en el tratamiento de datos y toma de decisiones en sistemas.

1. Definición de lógica difusa

La lógica difusa hace referencia a una serie de teorías que brindan una manera más fácil de llevar a cabo los procesos de razonamiento, donde la información de entrada es imprecisa, incompleta, vaga o ambigua. La lógica difusa es un mecanismo de la inteligencia computacional que permite manejar información con un elevado nivel de complejidad en comparación con la lógica clásica. Se define como una lógica multivaluada donde los valores de verdad como sí/ no, verdadero/falso, 0/1, etc. permiten valores intermedios [25].

2. Conjuntos difusos

Para el entendimiento de los conjuntos borrosos es conveniente relacionarlo con ciertos términos de la teoría de conjuntos clásicos. Sin embargo, es importante considerar que, en la teoría convencional, el grado de pertenencia de un elemento solo puede tomar los valores de 0 o 1. Mientras que, en los conjuntos difusos un elemento puede pertenecer a un conjunto de forma parcial, valores entre 0 y 1 [26].

3. Función de membresía

En la teoría clásica, si A representa un subconjunto del conjunto universo $U \rightarrow \{0, 1\}$, su función características χ_A está definida por:

$$\chi_A(x) = \begin{cases} 1 & \text{si } x \in A \\ 0 & \text{si } x \notin A \end{cases} \quad (2.1)$$

Es decir, si x está en A su función característica es 1, si no es 0.

En el caso de los conjuntos difusos, esta función se denomina función de membresía o pertenencia. De igual manera, si A representa un subconjunto del universo de discurso X , su función de membresía μ_A está definida por $\mu_A : X \rightarrow [0, 1]$ [27]. En un conjunto difuso, el valor $\mu_A(x)$, donde $x \in X$, se denomina valor o grado de membresía.

Algunas de las funciones de membresía típicas son:

- Función triangular:

$$\mu_A(x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a \leq x < b \\ \frac{c-x}{c-b}, & b \leq x \leq c \\ 0, & x > c \end{cases} \quad (2.2)$$

Siendo a el límite inferior, c el límite superior, b el punto medio (tal que $a < b < c$).

Otra forma de expresar la misma función se muestra en (2.3).

$$\mu_A(x) = \max \left(\min \left(\frac{x-a}{b-a}, \frac{c-x}{c-b} \right), 0 \right) \quad (2.3)$$

- Función trapezoidal:

$$\mu_A(x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a \leq x < b \\ 1, & b \leq x < c \\ \frac{d-x}{d-c}, & c \leq x \leq d \\ 0, & x > d \end{cases} \quad (2.4)$$

Siendo a el límite superior, d el límite inferior, b y c el límite de soporte superior e inferior respectivamente (tal que $a < b < c < d$). También expresada matemáticamente como:

$$\mu_A(x) = \max \left(\min \left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c} \right), 0 \right) \quad (2.5)$$

- Función gaussiana:

$$\mu_A(x) = e^{-\frac{(x-m)^2}{2\sigma^2}} \quad (2.6)$$

Siendo m el valor medio y σ la desviación estándar ($\sigma > 0$).

- Función bell-shaped generalizada:

$$\mu_A(x) = \frac{1}{1 + \left| \frac{x-c}{a} \right|^{2b}} \quad (2.7)$$

Siendo c el centro de la función, a la mitad del ancho y junto a b los puntos que determinan las pendientes $(-2b/a)$ sobre el punto de cruce [28].

4. Operaciones en conjuntos difusos

Algunas operaciones de la teoría de conjuntos clásicos como intersección, unión y complemento son definidas en los conjuntos difusos.

Siendo A y B subconjuntos del universo X , estas operaciones son definidas como:

Operación	Conjunto clásico	Conjunto difuso
Unión	$A \cup B = \{x x \in A \text{ ó } x \in B\}$	$\mu_{A \cup B}(x) = \text{máx}(\mu_A(x), \mu_B(x))$
Intersección	$A \cap B = \{x x \in A \text{ y } x \in B\}$	$\mu_{A \cap B}(x) = \text{mín}(\mu_A(x), \mu_B(x))$
Complemento de A	$\bar{A} = \{x x \notin A\}$	$\mu_{\bar{A}}(x) = 1 - \mu_A(x)$

5. Variable lingüística

Según Zadeh [29], una variable lingüística se puede denotar como:

$$\langle x, X, T(x), G, M \rangle \quad (2.8)$$

Donde:

- x : el nombre de la variable.
- X : es el universo de discurso.
- $T(x)$: es la colección de valores lingüísticos que x puede tomar.
- G : es una gramática que permite crear las etiquetas de $T(x)$ como alto, muy alto, bajo, muy bajo, normal, etc.
- M : es una regla semántica que asocia los valores lingüísticos de $T(x)$ con el universo de discurso X .

6. Razonamiento aproximado

En [30], se define el razonamiento aproximado como un proceso de inferencia para encontrar una solución cercana de un sistema de ecuaciones de asignación relacional.

Este último se caracteriza por su falta de claridad en las conclusiones de premisas difusas. La inferencia utiliza operaciones como unión, intersección y complemento ya antes mencionados.

El razonamiento aproximado se emplea para representar y razonar con conocimientos expresados en una forma de lenguaje natural [31]. Ejemplo: “la aceleración tiene un valor positivo pequeño”. Para convertir esta expresión en lenguaje natural a variables lingüísticas, se realiza de la siguiente manera:

- a) Definir símbolo A como la variable física de la “aceleración”.
- b) Definir símbolo PP como el valor lingüístico de “positivo pequeño” de la “aceleración”.
- c) En lenguaje natural la expresión es: A es PP .

7. Reglas difusas

Una regla difusa de la forma **IF-THEN** es expresada simbólicamente de la siguiente manera:

$$\text{IF } \langle \text{proposición difusa} \rangle \text{ THEN } \langle \text{proposición difusa} \rangle$$

La proposición difusa puede ser sencilla o compuesta, un ejemplo de proposición difusa es:

$$r : \text{IF } x \text{ es } A \text{ THEN } y \text{ es } B$$

Dado que, el antecedente es difuso, todas las reglas se usan de forma parcial, y la conclusión o consecuencia es cierto hasta cierto punto [31].

8. Identificación de sistemas difusos

Un sistema difuso está compuesto de cuatro elementos [32]:

- Interfaz de fuzzificación: es la encargada de convertir las variables de entrada a variables difusas tomando en cuenta el grado de membresía de cada variable.

- Base de conocimiento: conformada por un conjunto de reglas que determinan la relación entre las variables de entrada con las de salida e información relacionada con las funciones de membresía.
- Mecanismo de inferencia: usando el conjunto de reglas definido en la base de conocimientos, el mecanismo de inferencia obtiene la relación entre las variables de salida y variables de entrada o premisas.
- Interfaz de defuzzificación: al contrario de la fuzzificación, se encarga de convertir el valor que devuelve el mecanismo de inferencia a un valor discreto o determinista.

Existen dos enfoques básicos en la implementación de un sistema difuso:

- Mamdani
- Takagi-Sugeno-Kang

La principal diferencia entre estos dos enfoques se da en la etapa de defuzzificación.

9. Modelo difuso de Mamdani

Este modelo convierte las variables de entrada, valores numéricos generalmente provenientes de mediciones, a valores difusos para que puedan ser procesados por el mecanismo de inferencia. Dichos valores difusos son el grado de pertenencia de las entradas de los distintos conjuntos borrosos del universo de discurso. Este sistema no necesita de modelos matemáticos para controlar el sistema, el mecanismo de inferencia toma los grados de membresía y, en base a un conjunto de reglas, genera la salida [33]. El conjunto de reglas es del tipo si-entonces y está conformado por dos partes, el antecedente y el consecuente. Ambas partes son expresiones lingüísticas, denotadas como ejemplo de la siguiente manera.

$$\text{Regla } i : \underbrace{\text{Si } x \text{ es } A \text{ y } y \text{ es } B}_{\text{antecedente}} \text{ entonces } \underbrace{z \text{ es } C}_{\text{consecuente}}$$

Donde x y y son las entradas, z es la salida, A , B y C son variables lingüísticas definidas por conjuntos difusos dentro del universo de discurso X , Y y Z respectivamente.

En la etapa de inferencia del modelo de Mamdani, generalmente, utiliza los métodos de *min-max* y *producto-max* [34], dando como resultado un conjunto difuso, que aún no

puede ser entendido por el elemento de salida que usa variables numéricas. Para ello, se han establecido algunos de los métodos para convertir la salida difusa en numérica, entre los que están [35]:

- Método del centro de gravedad: consiste en encontrar el centro del área de la función de membresía del conjunto difuso implicado, matemáticamente la salida y^* está dada por:

$$y^* = \frac{\sum_{i=1}^N b_i \int_U \mu_{\hat{B}_i}(y) dy}{\sum_{i=1}^N \int_U \mu_{\hat{B}_i}(y) dy} \quad (2.9)$$

Donde:

- N : número de reglas.
 - U : universo de discurso.
 - b_i : centro del área de la función de membresía $\mu_{B_i}(y)$, que se asocia con el conjunto difuso implicado $\hat{B}_i$ para la i -ésima regla.
 - $\int_U \mu_{\hat{B}_i}(y) dy$: área bajo la función de membresía $\mu_{\hat{B}_i}(y)$.
- Método del centro promedio: la salida y^* es determinada por los centros de cada salida de las funciones de membresía. Para N reglas la salida es:

$$y^* = \frac{\sum_{i=1}^N b_i \mu_i(u_1, u_2, \dots, u_n)}{\sum_{i=1}^N \mu_i(u_1, u_2, \dots, u_n)} \quad (2.10)$$

Donde:

- b_i : es el centro del área de la función de membresía B_i , que se asocia con el conjunto difuso implicado $\hat{B}_i$ para la i -ésima regla.
 - $\mu_i(u_1, u_2, \dots, u_n) = \sup_U \{ \mu_{\hat{B}_i}(y) \}$: siendo “sup” el supremo, interpretado como el valor más alto de $\mu_{\hat{B}_i}(y)$.
- Método del valor máximo: escoge el valor máximo de la función de membresía de la salida, aunque no es un método óptimo, es muy sencillo.

10. Modelo difuso de Takagi-Sugeno-Kang

Este modelo se usa en la representación de procesos con dinámica no lineal. Para ello, reúne la información ciertos modelos locales [32]. En este sentido, está conformado por un conjunto de reglas si-entonces de la siguiente forma:

Regla i : si $x_1(t)$ es $MF_{1,i}$ y...y $x_k(t)$ es $MF_{k,i}$ entonces $y_i(x(t)) = f_i(x(t))$

Donde la regla i -ésima de un conjunto de N reglas está denotada por $x(t) = [x_1(t) \dots x_k(t)]$ que son las entradas, premisas o mediciones del sistema en el tiempo t . $MF_{k,i}$ es la función de membresía de la k -ésima entrada en la i -ésima regla, la ecuación de salida $y_i(x(t))$ es su modelo local y $f_i(x(t))$ es una función de las entradas del modelo, generalmente es lineal.

La salida del modelo difuso $y_{fuzzy}(t)$, se determina a partir de una sumatoria ponderada entre su grado de activación normalizado y cada modelo local.

$$y_{fuzzy}(t) = \frac{\sum_{i=1}^N w_i(x(t)) \cdot y_i(x(t))}{\sum_{i=1}^N w_i(x(t))} \quad (2.11)$$

El grado de activación de la i -ésima regla, $w_i(x(t))$ está definido por:

$$w_i(x(t)) = oper(\mu_i(x_1(t)), \dots, \mu_i(x_k(t))) \quad (2.12)$$

Donde la función $oper()$, puede ser el producto u operador mínimo, $\mu_i(x_k(t))$ es el grado de membresía de la entrada $x_k(t)$.

2.1.2. Algoritmo recocido simulado

El recocido simulado ([SA-Simulated Annealing](#)) es uno de los métodos metaheurísticos propuesto en 1983 por Kirkpatrick, Vecchi y Gellat como una técnica de optimización [36]. Además, se basa en el proceso metalúrgico de recocido, que consiste en calentar un metal por encima del punto de su fusión, para luego enfriarlo muy lentamente hasta llegar una temperatura muy baja. De esta manera, se forma una estructura estable y sin defectos [37].

El algoritmo recocido simulado busca la mejor solución (óptimo global) del problema propuesto. Para ello, en cada iteración de este algoritmo de búsqueda, no se elige siempre un vecino que mejore la solución; sino que, en algunas ocasiones, soluciones peores. Esto con la finali-

dad de no quedar atrapado en un óptimo local. Esta estrategia de elegir soluciones peores se hace en base a una función de probabilidad que controla los movimientos de escape. Además, si la solución va por buen camino conforme avanza la búsqueda, la función de probabilidad reduce dichos movimientos [38].

Algoritmo 1 Pseudocódigo del algoritmo de recocido simulado

Definir: Solución inicial x .

Definir: Función Objetivo $f(x)$.

Definir: Temperatura inicial $T = T_0$.

Definir: Máximo número de iteraciones N .

Definir: Iteración $i = 0$.

mientras $i < N$ **hacer**

$x_1 = \text{SeleccionAleatoria}(\text{SolucionesVecinas}(x));$

Calcular $\Delta E = f(x_1) - f(x)$

si $\Delta E < 0$ **entonces**

Se acepta la nueva solución, $x = x_1$

si no

Se acepta $x = x_1$ con probabilidad p

fin si

$i = i + 1;$

$T = g(i, T);$

fin mientras

devolver x

En el pseudocódigo del Algoritmo 1, se muestra como el algoritmo de recocido simulado selecciona de manera aleatoria una posible solución entre vecinos del estado actual, y si la solución es mejor, esta pasa a ser inmediatamente la nueva solución. Caso contrario, la solución será aceptada en base a la función de probabilidad. La función de probabilidad se muestra a continuación, esta depende de dos parámetros, el incremento de energía (ΔE) y la temperatura T .

$$p = \exp\left(-\frac{\Delta E}{T}\right) \quad (2.13)$$

La temperatura parte de un valor inicial (T_0) y decrece conforme avanza la búsqueda. En el pseudocódigo, se expresa esta función de temperatura T , misma que se actualiza en cada iteración i como $g(i, T)$. Dado que es una función decreciente, llegará a un punto donde solo se permitirán soluciones que igualen o mejoren la solución actual [39]. La manera más sencilla de elegir la función de temperatura es con un enfriamiento lineal. Pero, existen otras opciones, por ejemplo [40] presenta las siguientes.

Descenso exponencial:

$$g(i, T) = \alpha \cdot T \quad (2.14)$$

Donde, α es una constante que se aproxima a uno, generalmente se usan valores entre $[0.8 - 0.99]$.

Criterio de Boltzmann:

$$T_i = \frac{T_0}{(1 + \log(i))} \quad (2.15)$$

Criterio de Cauchy:

$$T_i = \frac{T_0}{(1 + i)} \quad (2.16)$$

Definiendo el número de iteraciones, el criterio de Cauchy (Cauchy modificado) es expresado como:

$$g(i, T) = \frac{T}{(1 + \beta T)} \quad (2.17)$$

Donde, $\beta > 0$ es un valor muy pequeño, para i iteraciones $\beta = \frac{T_0 - T_f}{iT_0T_f}$

2.1.3. Algoritmo A*

El algoritmo A estrella (**A*-A Star**) es un método de búsqueda heurística propuesto en 1968 por P. Hart, B. Raphael y N. Nilsson; lo que esperaban los autores al desarrollar el algoritmo era reducir la cantidad de nodos expandidos en la búsqueda del camino de menor costo, ruta óptima. Para ello, el algoritmo informa constantemente, cuál nodo debería expandir [41]. Por tal motivo, se encuentra clasificado dentro de los algoritmos de búsqueda en grafos de tipo informado.

El algoritmo **A*** emplea dos listas para guardar la información de los nodos. La primera se denomina lista de abiertos y contiene todos los nodos vecinos que han sido expandidos del nodo actual. La lista de cerrados que guarda los nodos que fueron el nodo actual.

Para dar con la ruta óptima, el algoritmo usa la función de evaluación $F = G + H$. El término G toma en cuenta el costo que genera al moverse del nodo origen hasta el nodo actual y el valor heurístico H estima el costo de desplazarse del nodo actual al nodo de destino [42].

A continuación, se muestra el pseudocódigo del algoritmo **A***.

Algoritmo 2 Pseudocódigo del algoritmo A^*

Definir: Nodo inicial NI y destino ND **Definir:** Lista vacía de nodos abiertos y cerrados**Definir:** Función de evaluación $F = G + H$ Colocar NI en la lista de abiertos $F(NI) = H(NI)$ **repetir** **si** Lista de abiertos está vacía **entonces** **devolver** No solución y finalizar **fin si** Seleccionar el nodo N , con F mínima, de la lista de abiertos Mover N en la lista de cerrados **si** $N = ND$ **entonces** **devolver** ruta de NI hasta ND y finalizar **fin si** Expandir(N) obteniendo un conjunto de sucesores **para** cada $S \in (Sucesores(N))$ **hacer** **si** $S \notin$ Lista abierta y $S \notin$ Lista cerrada **entonces** Colocar S en la lista de abiertos Guardar N como el predecesor de S Calcular $F(S) = G(S) + H(S)$ **si no** Recalcular $G(S)$ **si** $G(actual)$ es mejor $G(antiguo)$ **entonces** Guardar N como el nuevo predecesor de S Recalcular $F(S) = G(S) + H(S)$ **fin si** **fin si** **fin para****hasta que** el nodo destino se haya encontrado

Al empezar el algoritmo, se coloca el nodo inicial en la lista de abiertos. Luego, se realiza una estimación del coste empleando la función objetivo F . Dado que, el nodo actual es el mismo que el nodo inicial, el coste generado por desplazamiento es nulo ($G(NI) = 0$).

A continuación, el algoritmo repite una serie de instrucciones hasta encontrar el nodo destino o fallar cuando la lista de abiertos está vacía. Si este no es el caso, el algoritmo selecciona de la lista de abiertos el nodo con menor valor de F . Luego, elimina el nodo de la lista de abiertos y lo coloca en la lista de cerrados. Además, se verifica si el nodo actual corresponde al destino para finalizar el proceso iterativo, en dicho caso se devuelve la ruta recorrida (información que se almacena en sus predecesores), caso contrario se expande el nodo actual obteniendo un conjunto de nodos sucesores.

Para cada sucesor del nodo actual, se verifica si no se encuentra en la lista de nodos abiertos o en la lista de nodos que fueran expandidos. Si este es el caso, se agrega el nodo en la lista

de abiertos y se guarda la información de su predecesor. Además, se calcula el valor de F del nodo; en el caso de que el nodo sucesor se encuentre en la lista de abiertos. Seguidamente, se verifica si el costo del camino recorrido desde el nodo inicial, hasta dicho nodo, es menor. Si esto sucede, se actualiza la información de su predecesor y recalcula su función de evaluación [43].

2.2. Algoritmos de control

2.2.1. Control proporcional-integral-derivativo (PID)

Este tipo de controlador se caracteriza por su sencillez y robustez. En la actualidad, se siguen utilizando a pesar de haber cumplido un siglo desde los primeros sistemas de control basados en principios PID; siendo el trabajo de Minorsky, en el año 1922, uno de los primeros registros de su importancia teórica [44]. Es muy utilizado en la industria, gran parte sus aplicaciones de control son resueltas por los controladores PID. Especialmente, cuando la dinámica del sistema es adecuada (por ejemplo, se ha verificado su buen funcionamiento para sistemas con dinámica de segundo y primer orden) y requerimientos de control no muy exigentes (limitados por la respuesta del sistema ante cambios rápidos de referencia y comportamientos del error).

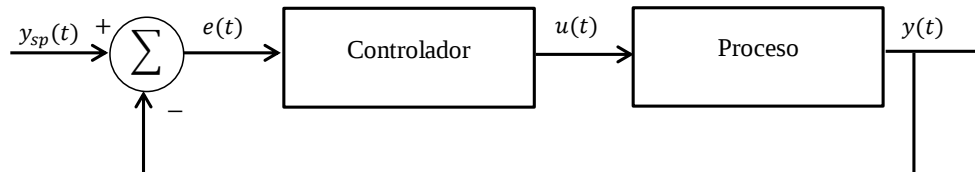


Figura 2.1: Diagrama de bloques de un sistema de control en lazo cerrado.

El controlador PID es un mecanismo de control de lazo cerrado (véase Figura 2.1), diseñado con la finalidad de hacer cero el error de estado estacionario, diferencia entre una señal deseada y la respuesta real de la salida del sistema. Para ello, hace usos de tres parámetros de control: proporcional, integral y derivativo [45].

1. Acciones de control

- **Acción de control proporcional (P):** esta acción genera una salida de control, $u(t)$, proporcional a la entrada, $e(t) = y_{sp}(t) - y(t)$; en el dominio de tiempo esta

acción de control es expresada como: $u(t) = K_p e(t)$, y su función de transferencia como: $U(s) = K_p E(s)$ [46].

El control proporcional funciona para cualquier sistema estable, al aumentar K_p la velocidad de respuesta del sistema aumenta y el error en régimen permanente (off-set) disminuye. Sin embargo, también aumenta la inestabilidad del sistema [47].

- **Acción de control integral (I):** esta acción genera una salida de control proporcional al error acumulado, garantizando la reducción del error del sistema en régimen permanente (referencia constante - error cero). En el dominio de tiempo, esta acción de control es expresada como: $u(t) = K_i \int_0^t e(\tau) d\tau$, y su función de transferencia como: $U(s) = \frac{K_i}{s} E(s)$ [46].

De manera similar que el control proporcional, al incrementar K_i , incrementa parte de la velocidad de respuesta del sistema. Pero, provoca más inestabilidad al sistema [47].

- **Acción de control derivativa (D):** esta acción genera una salida proporcional a la derivada de la señal de error; en el dominio de tiempo esta acción de control es expresada como: $u(t) = K_d \frac{de(t)}{dt}$, y su función de transferencia como: $U(s) = K_d s E(s)$ [46].

Al incrementar el valor del término derivativo se incrementa la estabilidad del sistema, pero se reduce parte de su velocidad de respuesta, conservando el mismo comportamiento del error del sistema en régimen permanente [47].

Estas acciones de control se resumen en la Tabla 2.1, donde se presenta como responde el sistema si se aumenta el valor de sus ganancias. Esto en lo que se refiere a su tiempo de subida, sobreimpulso, tiempo de establecimiento y error estacionario.

Tabla 2.1: Acciones básicas de control. Tomada de [13].

Acción de Control	Tiempo de subida	Sobreimpulso	Tiempo de establecimiento	Error estacionario
Si K_p aumenta	Disminuye	Aumenta	Cambia poco	Disminuye
Si K_i aumenta	Disminuye	Aumenta	Aumenta	Eliminado
Si K_d aumenta	Cambia poco	Disminuye	Disminuye	Cambia poco

2. Ecuación del controlador PID [48]

Al realizar la suma de los tres términos, donde la acción integral I representa el pasado, acción proporcional P el presente y la acción derivativa D el futuro. La ecuación del

control PID es:

$$u(t) = \underbrace{K_p e(t)}_P + \underbrace{\frac{K_p}{T_i} \int_0^t e(\tau) d\tau}_I + \underbrace{K_p T_d \frac{de(t)}{dt}}_D \quad (2.18)$$

y su función transferencia es :

$$C_{PID}(s) = \frac{U(s)}{E(s)} = K_p \left(1 + \frac{1}{T_i s} + T_d s \right) \quad (2.19)$$

Donde:

- K_p : ganancia proporcional.
- T_i : tiempo integral.
- T_d : tiempo derivativo.

Dominio del tiempo.

- $u(t)$: señal de control.
- $e(t)$: señal de error.

Dominio s:

- $U(s)$: transformada de Laplace de la señal de control.
- $E(s)$: transformada de Laplace de la señal de error.

3. Métodos de sintonización de controladores PID

- **Reglas de sintonización de Ziegler-Nichols (primer método)** [1]: es un método en lazo abierto que consiste en obtener la respuesta de la planta a la entrada de un escalón unitario, de tal forma que la salida sea una curva sigmoïdal o forma de S (véase Figura 2.2). Esto quiere decir que lo ideal es que la planta no tenga polos complejos conjugados ni integradores.

Para modelizar la planta como un sistema de primer orden con tiempo muerto, se traza una línea tangente en el punto de inflexión y se señalan los puntos de cruce con el eje del tiempo y el eje donde $c(t) = K$ (véase Figura 2.2). Esto permite determinar los valores de las constantes L y τ , y definir la siguiente función de transferencia.

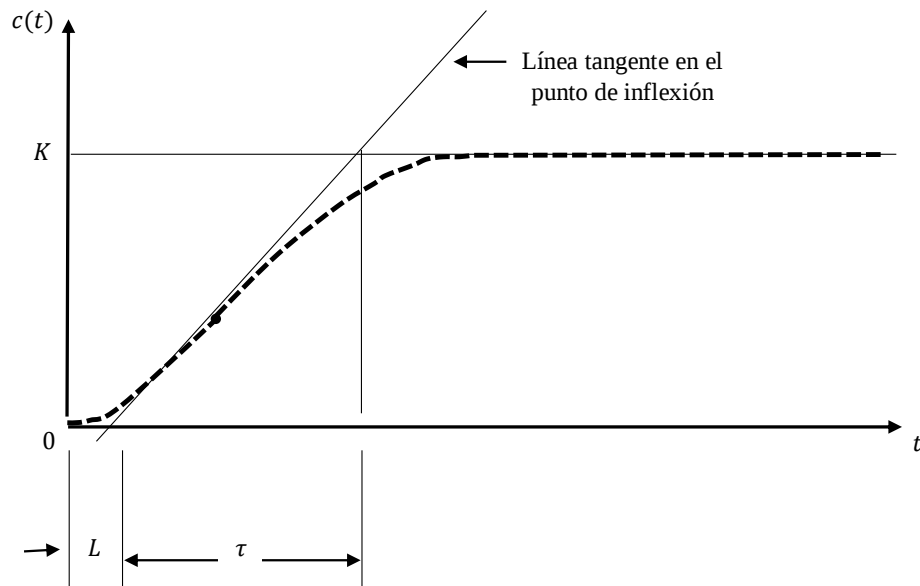


Figura 2.2: Respuesta de la planta a un escalón unitario. *Imagen adaptada de [1].*

$$\frac{C(s)}{U(s)} = \frac{Ke^{-Ls}}{\tau s + 1} \quad (2.20)$$

Donde:

- L : es el tiempo de retardo.
- τ : es una constante de tiempo.
- K : es la ganancia del sistema.

Los parámetros de L y τ son utilizados para la sintonización del controlador PID, estableciendo los valores de K_p , T_i y T_d en base a las relaciones de la Tabla 2.2.

Tabla 2.2: Regla de sintonía de Ziegler-Nichols (primer método). *Elaborada con datos de [1].*

Controlador	K_p	T_i	T_d
P	$\frac{\tau}{L}$	∞	0
PI	$0.9 \frac{\tau}{L}$	$\frac{L}{0.3}$	0
PID	$1.2 \frac{\tau}{L}$	$2L$	$0.5L$

- **Reglas de sintonización de Ziegler-Nichols (segundo método) [1]:** es un método en lazo cerrado, consiste en colocar los tiempos $T_i = \infty$ y $T_d = 0$. Posteriormente, se varía la ganancia K_p desde cero hasta un valor que genere oscilaciones sostenidas a la salida del sistema (si no genera oscilaciones no se puede aplicar este método). El valor antes mencionado, se denomina ganancia crítica o última K_u .

y al periodo de la oscilación, periodo crítico o último T_u .

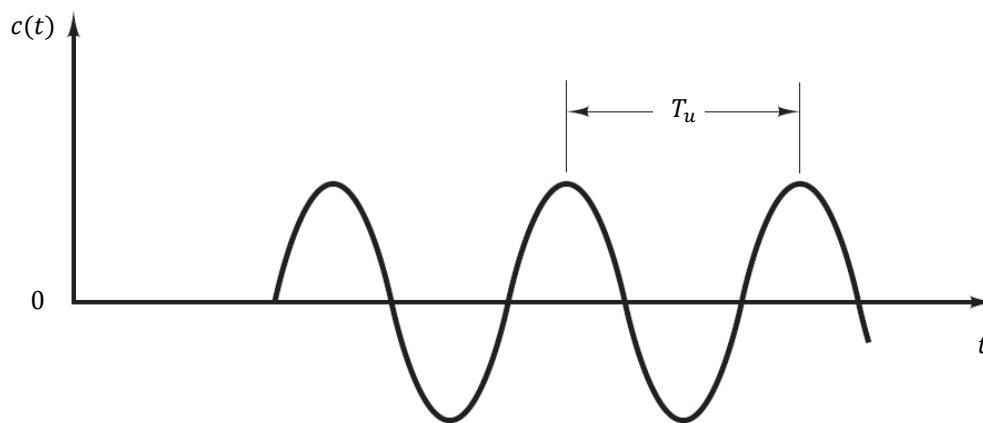


Figura 2.3: Método de la ganancia última, oscilaciones sostenidas. *Imagen adaptada de [1].*

Se obtienen los valores de K_p , T_i y T_d con las relaciones que se muestran en la Tabla 2.3.

Tabla 2.3: Regla de sintonía de Ziegler-Nichols (segundo método). *Elaborada con datos de [1].*

Controlador	K_p	T_i	T_d
P	$0.5K_u$	∞	0
PI	$0.45K_u$	$\frac{T_u}{1.2}$	0
PID	$0.6K_u$	$0.5T_u$	$0.125T_u$

4. Sintonización por criterios integrales

La necesidad de evaluar el comportamiento de los sistemas de control, condujo a la determinación de índices de desempeño a partir de la señal del error $e(t)$. Los criterios integrales o índices de desempeño más empleados en los sistemas de control son:

- **IAE** (Integral del valor absoluto del error)

$$IAE = \int_0^{\infty} |e(t)| dt \quad (2.21)$$

- **ITAE** (Integral del tiempo por el error absoluto)

$$ITAE = \int_0^{\infty} t |e(t)| dt \quad (2.22)$$

- **ISE** (Integral del error cuadrático)

$$ISE = \int_0^{\infty} e^2(t) dt \quad (2.23)$$

- **ITSE** (Integral del tiempo por el error cuadrático)

$$ITSE = \int_0^{\infty} te^2(t) dt \quad (2.24)$$

La finalidad de este método es encontrar los parámetros del controlador que minimicen el costo de cierta función objetivo, los parámetros encontrados serán óptimos según los criterios de desempeño elegidos [14].

Un claro ejemplo de este método se encuentra en el trabajo de Lopez en 1967 [49]; el autor propuso un método de sintonización por criterios integrales para rechazo de perturbaciones en un PID ideal. Al suponer el modelo del sistema de primer orden (2.20), se busca minimizar la integral de error. No obstante, el estudio se limitó para valores de $\frac{L}{\tau}$, definido como factor de incontrolabilidad, entre 0 y 1. El método determinó las siguientes ecuaciones para la sintonía:

$$K_p = \frac{1}{K} a \left(\frac{L}{T} \right)^b \quad (2.25)$$

$$T_i = \frac{\tau}{c \left(\frac{L}{\tau} \right)^d} \quad (2.26)$$

$$T_d = \tau e \left(\frac{L}{\tau} \right)^f \quad (2.27)$$

Donde los valores a , b , c , d , e y f dependen del tipo de controlador y el criterio de desempeño, por ejemplo para controladores PI y PID, con los criterios IAE, ITAE y ISE se muestra en la Tabla 2.4.

Tabla 2.4: Reglas de sintonía por criterios integrales. *Elaborada con datos de [14].*

Criterio	Controlador	a	b	c	d	e	f
IAE	PI	0.984	-0.986	0.608	-0.707	-	-
	PID	1.435	-0.921	0.878	-0.749	0.482	1.137
ITAE	PI	0.859	-0.977	0.674	-0.68	-	-
	PID	1.357	-0.947	0.842	-0.738	0.381	0.995
ISE	PI	1.305	-0.959	0.492	-0.739	-	-
	PID	1.495	-0.945	1.101	-0.771	0.560	1.006

2.2.2. Control on-off

También conocido como control todo o nada, es un algoritmo sencillo de implementar. Su funcionamiento se basa en comparar la salida del sistema con su referencia para cambiar de un estado a otro. Debido a que su salida únicamente puede tener dos estados, el controlador no es capaz de tener un valor exacto a su salida generando una constante oscilación con respecto a la señal de referencia [50]. A esta diferencia, entre un estado a otro, se conoce como histéresis. El ejemplo más común del uso de este controlador es en los sistemas de regulación de temperatura (Figura 2.4).

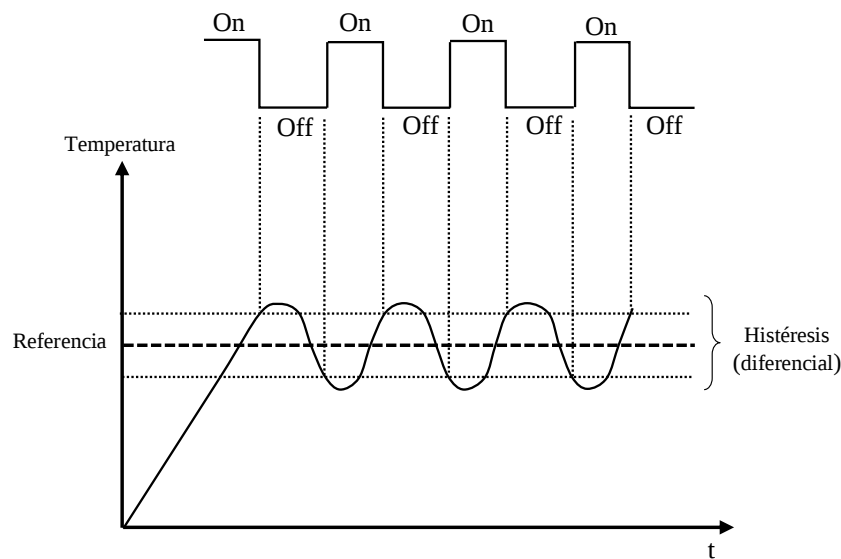


Figura 2.4: La respuesta tiempo-temperatura de un controlador On-Off. *Imagen adaptada de [2].*

2.2.3. Control proporcional de tiempo variable

Partiendo de la idea del controlador on-off, donde se tienen únicamente dos estados, desactivado 0 % y activado 100 %; el controlador proporcional de tiempo variable entrega potencia de forma porcentual entre 0 y 100 %. Este controlador se basa en el concepto de modulación de ancho de pulso (PWM). Misma que combina la relación entre el tiempo de activación t_{on} y el tiempo de un ciclo t_c para obtener una potencia promedio (Figura 2.5). A este porcentaje o relación se le denomina ciclo de trabajo y es determinado según (2.28). El control PWM a menudo se usa en controladores eléctricos [51].

$$\text{ciclo de trabajo} = \frac{t_{on}}{t_c} \cdot 100 \quad (2.28)$$

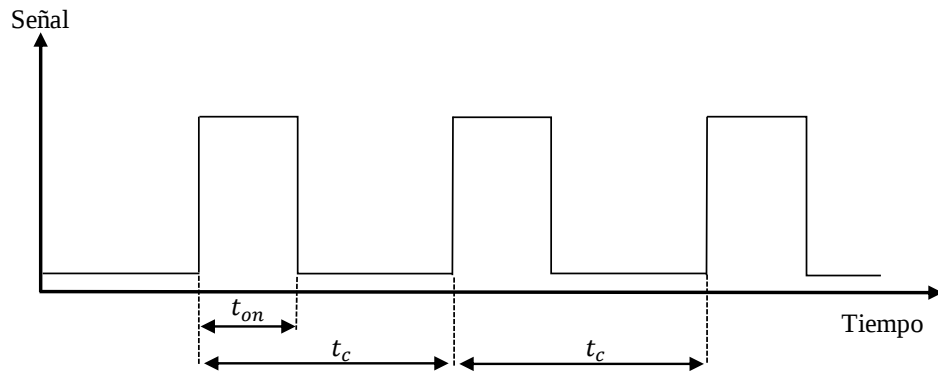


Figura 2.5: Señal PWM, (relación entre t_{on} y t_c). *Imagen adaptada de [3].*

2.2.4. Controlador PID difuso de ganancias programables (FGS-PID)

El controlador PID difuso de ganancias programables (FGS, por sus siglas en inglés **Fuzzy Gain Scheduling**) utiliza el razonamiento aproximado y un conjunto de reglas difusas para determinar las ganancias K_p , K_i y K_d del controlador PID cuando el sistema se encuentra en funcionamiento. Lo antes mencionado, fue propuesto ya hace algunas décadas en [4], donde se plantea el siguiente esquema de control (Figura 2.6).

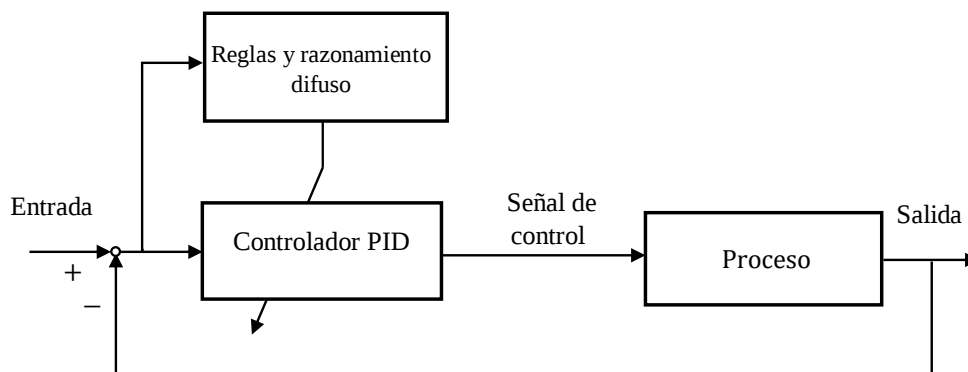


Figura 2.6: Controlador PID difuso de ganancias programables. *Imagen adaptada de [4].*

1. Algoritmo de control FGS-PID

En un principio se normaliza los valores de K_p y K_d entre 0 y 1 utilizando (2.29) y (2.30).

$$K'_p = \frac{K_p - K_{(p,min)}}{K_{(p,max)} - K_{(p,min)}} \quad (2.29)$$

$$K_d' = \frac{K_d - K_{(d,min)}}{K_{(d,max)} - K_{(d,min)}} \quad (2.30)$$

Los parámetros de control K_p y K_d se encuentra en el rango de $[K_{(p,min)}, K_{(p,max)}]$ y $[K_{(d,min)}, K_{(d,max)}]$ respectivamente. Sus valores mínimos y máximos son establecidos a partir de las siguientes relaciones.

$$K_{(p,min)} = 0.32K_u \quad (2.31)$$

$$K_{(p,max)} = 0.6K_u \quad (2.32)$$

$$K_{(d,min)} = 0.08K_u T_u \quad (2.33)$$

$$K_{(d,max)} = 0.15K_u T_u \quad (2.34)$$

Donde:

- K_u : es la ganancia límite, ganancia última o ganancia crítica.
- T_u : es el periodo crítico.

En el caso de la acción integral del controlador, el tiempo integral es proporcional al tiempo derivativo, $T_i = \alpha \cdot T_d$

El algoritmo del control **FGS-PID** utiliza como parámetro de entrada la señal de error $e(k)$ y su diferencia $\Delta e(k)$, en la etapa de fuzzificación, donde se convierte los valores de entrada a valores difusos. A continuación, el mecanismo de inferencias usando un conjunto de reglas predefinidas y razonamiento difuso devuelve como salida los valores normalizados K_p' , K_d' y el valor proporcional α ; se utiliza el método del centroide para realizar la defuzzificación. Finalmente, las ganancias del controlador **PID** son obtenidas a partir de (2.35 - 2.37).

$$K_p = (K_{(p,max)} - K_{(p,min)}) K_p' + K_{(p,min)} \quad (2.35)$$

$$K_d = (K_{(d,max)} - K_{(d,min)}) K_d' + K_{(d,min)} \quad (2.36)$$

$$K_i = \frac{K_p}{\alpha T_d} = \frac{K_p^2}{\alpha K_d} \quad (2.37)$$

2. Reglas difusas

El conjunto de reglas que utiliza el algoritmo FGS-PID se detallan en las Tablas 2.6, 2.6 y 2.7:

Tabla 2.5: Reglas difusas de sintonía para K'_p . *Elaborada con datos de [4].*

		$\Delta e(k)$						
		NB	NM	NS	ZO	PS	PM	PB
$e(k)$	NB	B	B	B	B	B	B	B
	NM	S	B	B	B	B	B	S
	NS	S	S	B	B	B	S	S
	ZO	S	S	S	B	S	S	S
	PS	S	S	B	B	B	S	S
	PM	S	B	B	B	B	B	S
	PB	B	B	B	B	B	B	B

Tabla 2.6: Reglas difusas de sintonía para K'_d . *Elaborada con datos de [4].*

		$\Delta e(k)$						
		NB	NM	NS	ZO	PS	PM	PB
$e(k)$	NB	S	S	S	S	S	S	S
	NM	B	B	S	S	S	B	B
	NS	B	B	B	S	B	B	B
	ZO	B	B	B	B	B	B	B
	PS	B	B	B	S	B	B	B
	PM	B	B	S	S	S	B	S
	PB	S	S	S	S	S	S	S

Tabla 2.7: Reglas difusas de sintonía para α . *Elaborada con datos de [4].*

		$\Delta e(k)$						
		NB	NM	NS	ZO	PS	PM	PB
$e(k)$	NB	2	2	2	2	2	2	2
	NM	3	3	2	2	2	3	3
	NS	4	3	3	2	3	3	4
	ZO	5	4	3	3	3	4	5
	PS	4	3	3	2	3	3	4
	PM	3	3	2	2	2	3	3
	PB	2	2	2	2	2	2	2

En las Tablas 2.5, 2.6 y 2.7 se encuentran denotadas siete variables lingüísticas de entrada: NB, NM, NS, ZO, PS, PM y PB donde las letras ZO hace referencia valores cercanos a cero, la letra P a valores positivos, N valores negativos junto a sus etiquetas B para valores grandes, M valores medianos, S valores pequeños.

Las variables lingüísticas de salida son: S para valores pequeños, B para valores grandes

y valores constantes entre 2 y 5.

La entrada y salida están relacionadas a partir de varias reglas difusas de la forma si-entonces. Estas últimas son obtenidas de las Tablas 2.5, 2.6 y 2.7 tomando como salida la intersección de las entradas. Por ejemplo, para la primera y última regla su expresión es:

Regla 1 : si $e(k)$ es NB, $\Delta e(k)$ es NB entonces K_p' es B, K_d' es S, α es 2

⋮

Regla 49 : si $e(k)$ es PB, $\Delta e(k)$ es PM entonces K_p' es B, K_d' es S, α es 2

3. Funciones de Membresía

Para determinar el grado de pertenencia de las variables del conjunto difuso se definen las funciones de membresía de forma triangular para la entrada (Figura 2.7), y para la salida funciones gaussianas y singletons (Figuras 2.8 y 2.9).

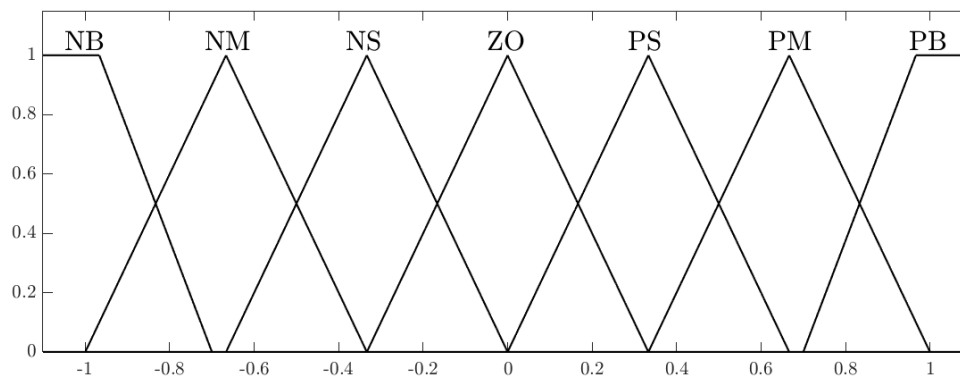


Figura 2.7: Funciones de membresía para la variable del error $e(k)$ y derivada del error $\Delta e(k)$. Elaborada con datos de [4].

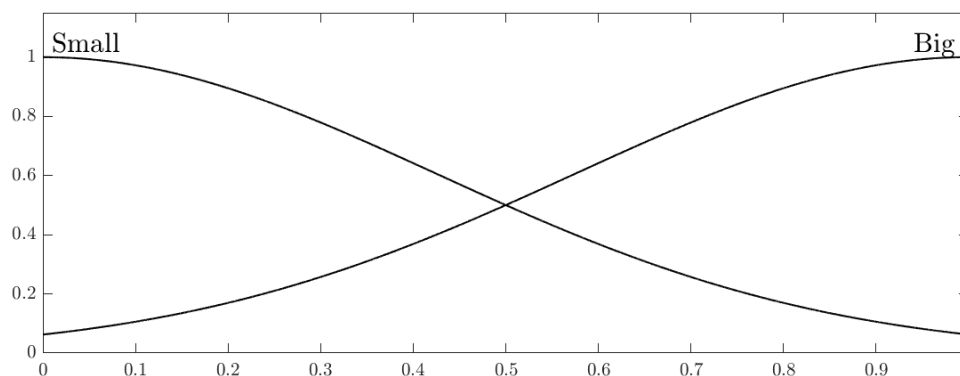


Figura 2.8: Funciones de membresía para las variables K_p' y K_d' . Elaborada con datos de [4].

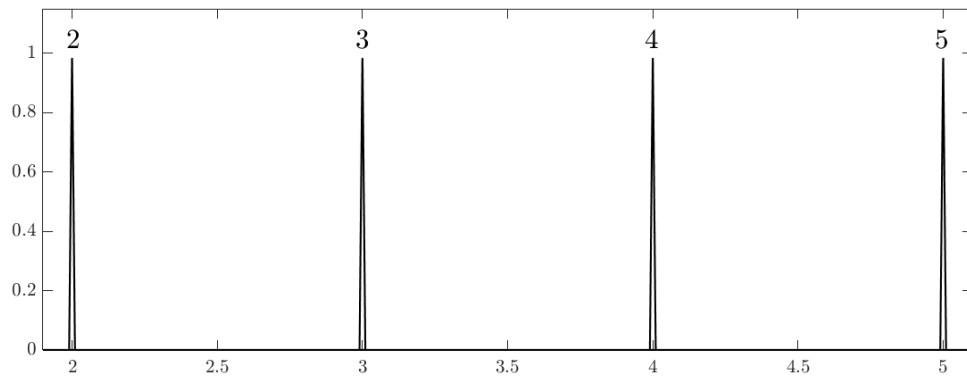


Figura 2.9: Funciones de membresía para la variable α . *Elaborada con datos de [4].*

Estado del arte

En este capítulo, se describe el estado del arte de algunos algoritmos de IA implementados para la sintonización de ganancias en controladores PID, sobre todo aquellos que son de interés en esta investigación como el algoritmo A*, recocido simulado y control basado en lógica difusa. Además, se detallan los trabajos relacionados con el control de sistemas de múltiples tanques y la implementación de sistemas de control en microcontroladores STM32. Para ello, este capítulo se divide en tres secciones, la primera detalla los algoritmos de inteligencia artificial implementados en el proceso de control, la segunda sección sobre el control de nivel en un sistema multitanque y la tercera el uso de plataformas de desarrollo STM32 para el proceso de control.

3.1. Estado del arte de los algoritmos de inteligencia artificial implementados en sistemas de control

Existen varias estrategias para la sintonización de las ganancias K_p , K_i y K_d de un controlador PID. Entre los que destaca la literatura, tenemos el uso de algoritmos meta-heurísticos, por ejemplo, la técnica de optimización de enjambre de partículas (PSO), algoritmo bat (BA) y algoritmo firefly (FA) que son algoritmos bioinspirados; es decir, se basan en los sistemas biológicos o físicos de la naturaleza [52]. Un ejemplo de la aplicación del algoritmo PSO es presentado en [53] para mejorar la eficiencia de un controlador PID basado en lógica difusa (FLC) en un sistema no lineal de suspensión de un vehículo. Otro algoritmo meta-heurístico es la evolución diferencial (DE), este método de optimización presenta un mejor comportamiento que otros enfoques de sintonización al ser empleado en un sistema interconectado de energía térmica [54]. Otra técnica muy estudiada desde hace algunas décadas, es el uso de algoritmos genéticos (GA). Este es un método heurístico muy utilizado en problemas de optimización y puede ser implementado en la sintonización de las ganancias de un controlador PID, como se presenta en [55] aplicada a una planta concentradora de cobre.

Los autores de [56] propusieron como método de ajuste para controladores PID, el algoritmo recocido simulado (SA) en un sistema de control de movimiento con perturbación constante. Este estudio se basó en minimizar la integral en el tiempo del valor absoluto del error (ITAE) y eliminar el error de estado estable del sistema. El algoritmo SA se aplicó sobre tres diferentes controladores PID mostrando que cada controlador tiene su ventaja de acuerdo a los requisitos de control. En [57], se presenta un sistema de control PID con ajuste de ganancias en línea. Utilizando el algoritmo SA, se realiza una comparación con el algoritmo genético

demostrando una disminución considerable del tiempo de cálculo. Un estudio más reciente es presentado en [58], donde se realiza una mejora de este algoritmo denominado recocido simulado adaptativo ([ASA— Adaptive Simulated Annealing](#)), el estudio muestra como el uso de un algoritmo adaptativo mejora el rendimiento en los sistemas estudiados.

Un controlador [PID](#) con ganancia programada difusa ([FGS-PID](#)) utiliza reglas difusas para cambiar las ganancias del controlador [PID](#) durante el funcionamiento [59]. Se basa en la señal de error y su primera diferencia para determinar los parámetros del controlador, fue propuesto hace unas décadas demostrando un mejor rendimiento que métodos de sintonización como [ZN](#) y Kitamori [4]. Trabajos recientes como [60], demuestran que este controlador es más robusto y tiene buen comportamiento para el seguimiento de puntos de ajuste y rechazo a las perturbaciones; entre otras características como alcanzar la posición de estabilidad, rápidamente, y menor sobreimpulso [61].

En relación con el algoritmo [A*](#), su enfoque está dirigido a la optimización de rutas más que a la sintonización de ganancias del control [PID](#). Por ejemplo, en [62] se buscaba mejorar la capacidad dinámica de un robot usando un algoritmo genético que ajusta la función de membresía de un controlador difuso e implementa el algoritmo de búsqueda [A*](#) para obtener un movimiento más suave del robot. De manera similar, en el trabajo de [63] para el control de seguimiento de trayectoria de un robot móvil, se utilizaron tres algoritmos: el algoritmo [A*](#), algoritmo de optimización de enjambre híbrido y de enjambre de partículas, demostrando un aumento de la eficiencia de la estrategia de control al comparar los resultados con otras simulaciones.

En el caso de la sintonización de parámetros del controlador [PID](#) usando el algoritmo [A*](#), no existe un método definido. Sin embargo, artículos relacionados como [64] presenta una manera de implementar el algoritmo de búsqueda Best First Search ([BFS](#)). Para la sintonización (el algoritmo [A*](#) se basa en [BFS](#)), este método plantea un nodo principal que se expande generando nodos secundarios. Así, se evalúan los nodos secundarios y se escoge el que genere buenos resultados. De esta manera, el nodo elegido pasa a ser un nodo principal para seguir expandiéndose y continuar la búsqueda.

Uno de los puntos más importantes de la implementación del algoritmo de optimización, como [GA](#), [SA](#) o [A*](#) en la sintonización de ganancias, es la determinación de la función objetivo. En efecto, de ello depende la mejoría del sistema de control. En [65], se comparan diferentes índices de desempeño usados en la función objetivo, tales como: [IAE](#), [ITAE](#), [ISE](#), [ITSE](#), [MSE](#) y [LQR](#), aplicados sobre un algoritmo rápido de [SA](#). De esta forma, se concluye que, el índice

de desempeño [IAE](#) trabaja un poco mejor que el resto. Así mismo, en [\[66\]](#) propone funciones multiobjetivo que combinan no sólo los criterios de desempeño de la integral del error, sino también parámetros característicos de la señal como el porcentaje de sobreimpulso (OS), tiempo de estabilización (t_{ss}) y tiempo de rizado (t_r).

3.2. Estado del arte del sistema multitanque y las estrategias de control aplicadas

Este proyecto parte del resultado del trabajo previo de “Diseño e implementación de los sistemas de instrumentación, comunicaciones y control de un sistema multitanque” presentado en [\[5\]](#). En dicho trabajo, los autores diseñan un sistema multitanque que consta de dos tanques de reserva, un tanque de bombeo, un sistema de instrumentación integrado en un tablero de control. Entre cada tanque, se implementa una electroválvula que regula el paso del agua, la primera electroválvula controla el flujo de entrada del sistema de forma proporcional, a través de una señal de voltaje. La segunda electroválvula, comprendida entre los dos tanques de reserva, tiene únicamente dos modos de operación, totalmente abierto y totalmente cerrado. Por lo tanto, el flujo del agua es controlado a través de una señal [PWM](#).

Finalmente, la última electroválvula controla el flujo de salida mediante su apertura. Esta última es proporcional al tiempo de alimentación de la válvula. Se realiza el control del nivel del agua de los dos tanques de reserva mediante sensores ultrasónicos. La estrategia de control en este trabajo, consta de dos esquemas. El primero consiste en la implementación de controladores [PID](#) clásicos en su totalidad y el segundo se le agrega un controlador [FGS-PID](#) en la segunda electroválvula. En las pruebas, se evalúa la respuesta de la planta ante cambios de referencia, los resultados muestran una ligera mejoría en el tiempo de estabilización y una disminución del sobreimpulso con la implementación del controlador [FGS-PID](#).

En [\[67\]](#) se presenta un sistema de tres tanques colocados en forma de cascada y un tanque de reserva, donde se almacena el agua que se suministra al sistema utilizando una bomba. El sistema está conformado por dos tipos de válvulas, válvulas manuales y electroválvulas. En referencia a las válvulas manuales, son de tipo solenoides, por lo que únicamente tiene dos posiciones (abierta o cerrado). De hecho, para ajustar de forma proporcional, a este tipo de válvulas se envía una señal [PWM](#). Esto permite que la electroválvula se active o desactive de forma progresiva, de acuerdo a los valores establecidos en el controlador. Se proponen dos estrategias de control, la primera un controlador fuzzy, sus resultados muestran un correcto seguimiento a la referencia. Sin embargo, el proceso será dependiente en gran medida de su ganancia K_p , ya que un valor alto produce recalentamiento de las válvulas y mayor suscepti-

bilidad a perturbaciones. La segunda estrategia de control se basa en redes neuronales. En este contexto, se estudian dos métodos de aprendizaje diferente, red neural inversa y modelo diferencial, obteniendo mejores resultados con el primero.

Otro estudio similar al presentado anteriormente es [68], el sistema propuesto consta de: un tanque principal, un tanque de perturbación de nivel y un tanque de reserva. Además, dos electroválvulas que trabajan en forma proporcional, una bomba sumergible y un sensor. La primera electroválvula controla un flujo de entrada proveniente del tanque de perturbación, como su nombre lo indica este flujo perturba al sistema de control de nivel. La segunda electroválvula conectada entre el tanque principal y de reserva, controla el flujo de salida. Mientras que, el flujo de entrada del sistema está determinado por el accionar de la bomba. Se compara una estrategia de control de red neuronal basado en el modelo inverso con una estrategia de control difuso. Para lograrlo, se emplearon tres parámetros de evaluación de desempeño del sistema, el sobreimpulso generado, el tiempo de estabilización y el porcentaje de error de estado estacionario. Los resultados muestran que la estrategia de control neuronal elimina el sobreimpulso y reduce en gran medida el error en estado estacionario y su tiempo de estabilización.

3.3. Estado del arte de los sistemas de control basados en STM32

STM32 es una familia de microcontroladores de 32 bits fabricados a partir de procesadores ARM Cortex-M. Estos dispositivos se utilizan en el desarrollo de proyectos que requieran el procesamiento de señales digitales, capacidades en tiempo real, operaciones de bajo consumo, conectividad y alto rendimiento junto a la sencillez de desarrollo [69]. Los microcontroladores STM32 se implementan en varios proyectos de control. Por ejemplo, en el trabajo presentado en [70], se desarrolla un sistema de control de temperatura de respuesta rápida y alta precisión utilizando el algoritmo PID. El sistema antes mencionado consta de un enfriador termoeléctrico accionado a través de una señal PWM y un sensor de temperatura. La señal PWM se ajusta de acuerdo al valor de control determinado por el algoritmo PID. Los resultados obtenidos presentan un sistema estable con respuesta rápida y un error de $\pm 0.5^{\circ}C$. Un sistema similar, presentado en [71], consiste en un sistema adaptativo de control de la temperatura del agua. Este sistema usa un algoritmo PID difuso para controlar la potencia de un calentador de agua de energía eléctrica. De hecho, los resultados muestran una mejoría en el tiempo de estabilización, disminución del sobreimpulso y rápida respuesta en comparación al PID clásico.

En [72] se muestra el diseño de un sistema de control de velocidad de un motor de CC. En este caso, el sistema usa un algoritmo PID que ajusta la velocidad del motor variando el ciclo de trabajo de la señal PWM. La implementación cumple con los objetivos de control, obteniendo un valor pequeño de estado estacionario, buena estabilidad, precisión en el control y bajo consumo de energía. Una implementación más completa es presentada en [73], consiste en un sistema automático de control de pH de la solución de nutrientes en la fábrica de plantas inteligentes. El sistema mencionado consta de una serie de sensores para la adquisición de datos como: temperatura, humedad, intensidad de luz, captura de imágenes, concentración de dióxido de carbono, pH de la solución nutritiva; y módulos para el control como: módulo de humidificación, iluminación LED, acondicionador de aire, módulo de ajuste de pH de la solución nutritiva y suministro de dióxido de carbono. En este trabajo, el estudio se enfoca en el ajuste del valor de pH en la solución nutritiva. Por tal motivo, propone dos esquemas de control, el PID clásico y PID experto. Este último es una mejora del PID clásico que agrega reglas expertas de control y un preajuste de parámetros. De forma similar, los resultados muestran que a pesar de las dificultades de ajuste del valor de pH, los dos sistemas cumplen su objetivo. Además, se establecen ventajas claras en la regulación de pH al usar el esquema de control PID experto.

Metodología

En este capítulo, se detalla el diseño e implementación de los algoritmos de inteligencia artificial para el ajuste de ganancia de un controlador PID. Simultáneamente, se presenta, de manera técnica, la metodología de diseño de estos algoritmos sobre el software de MATLAB y su implementación sobre un sistema multitanque simulado. Además, se presenta la adaptación de estos algoritmos sobre la placa STM32 núcleo-144 y su integración para el control de un sistema físico de múltiples tanques.

4.1. Diseño e implementación de los algoritmos de ganancia programada en MATLAB

En esta sección, se describen los conceptos considerados para el diseño de los distintos algoritmos de programación de ganancias en MATLAB. Lo antes mencionado se sustentará en algunas secciones de código para mayor entendimiento.

4.1.1. Controlador PID con ganancia programada basado en lógica difusa (FGS-PID)

Este esquema de control se basa en el presentado en la Sección 2.6, con la diferencia de que se busca eliminar la dependencia de los valores K_u y T_u obtenidos en la sintonización por el método de Ziegler-Nichols. Para ello, usando (2.31 - 2.34), y la reglas de sintonía de la Tabla 2.3 se obtiene que:

$$K_{(p,min)} = 0.533K_p \quad (4.1)$$

$$K_{(p,max)} = K_p \quad (4.2)$$

$$K_{(d,min)} = 1.067K_d \quad (4.3)$$

$$K_{(d,max)} = 2K_d \quad (4.4)$$

Además, se elimina la dependencia del valor de la acción integral de la derivativa (2.37). Reemplazándola por la siguiente ecuación:

$$K_i = (K_{(i,max)} - K_{(i,min)}) K_i' + K_{(i,min)} \quad (4.5)$$

Donde los valores $K_{(i,min)}$ y $K_{(i,max)}$ son determinados de acuerdo a los siguientes criterios:

- $K_{(i,min)} = 0$, implica un controlador con acción proporcional y derivativa (PD). Este reduce los transitorios como el tiempo de subida, el sobreimpulso y las oscilaciones en la salida, sin embargo, en entornos con ruido puede crear una gran inestabilidad [74].
- $K_{(i,max)} = K_i$, indica que el valor más alto que puede tomar la ganancia integral es K_i , siendo este un valor conocido y obtenido por cualquier método de sintonía PID, por tanto, garantiza la estabilidad en la salida del lazo de control.

En base a estos criterios y además de que no es preferible eliminar la acción integral del controlador, se propone el siguiente rango.

$$K_{(i,min)} = 0.01K_i \quad (4.6)$$

$$K_{(i,max)} = 0.85K_i \quad (4.7)$$

Donde los valores de 0.01 y 0.85 son obtenidos al realizar distintas pruebas. Estas consisten en ajustar el intervalo de manera que reduzcan el sobreimpulso y tiempo de estabilización en sistemas de segundo, tercer y cuarto orden, cuyas funciones de trasferencias son:

$$G_1(s) = \frac{e^{-0.5s}}{(s+1)^2} \quad (4.8)$$

$$G_2(s) = \frac{4.228}{(s+0.5)(s^2+1.64s+8.456)} \quad (4.9)$$

$$G_3(s) = \frac{27}{(s+1)(s+3)^3} \quad (4.10)$$

Para este controlador, se definen las siguientes variables lingüísticas para la entrada del error:

- **NB** (negativo grande): la señal de salida está muy por encima del valor de referencia.
- **NM** (negativo mediano): la señal de salida está medianamente encima del valor de referencia.
- **NS** (negativo pequeño): la señal de salida está por encima, pero muy cercana a la señal

de referencia.

- **ZO** (cercano a cero): la señal de salida está muy cerca del valor de referencia.
- **PS** (positivo pequeño): la señal de salida está por debajo, pero muy cercana de la señal de referencia.
- **PM** (positivo mediano): la señal de salida está medianamente por debajo de la señal de referencia.
- **PB** (positivo grande): la señal de salida está muy por debajo de la señal de referencia.

Las variables lingüísticas para la primera diferencia de la señal del error $\Delta e(k)$:

- **NB** (negativo grande): la derivada del error tiene una pendiente negativa.
- **NM** (negativo mediano): la derivada del error tiene una pendiente medianamente negativa.
- **NS** (negativo pequeño): la derivada del error tiene una pendiente negativa, casi constante.
- **ZO** (cercano a cero): la derivada del error tiene una pendiente casi o completamente constante.
- **PS** (positivo pequeño): la derivada del error tiene una pendiente positiva, casi constante.
- **PM** (positivo mediano): la derivada del error tiene una pendiente medianamente positiva.
- **PB** (positivo grande): la derivada del error tiene una pendiente positiva.

Las variables lingüísticas de salida para K_p son:

- **small** (valor pequeño): reducción de la ganancia K_p , aumenta estabilidad y disminuye la velocidad de respuesta.
- **big** (valor grande): aumento de la ganancia K_p , disminuye la estabilidad pero, aumenta la velocidad de respuesta.

Las variables lingüísticas de salida para K_d son:

- **small** (valor pequeño): aumenta la velocidad de respuesta, menor estabilidad.
- **big** (valor grande): incrementa la estabilidad del sistema, conserva el comportamiento de error en régimen permanente.

Las variables lingüísticas de salida para K_i son:

- **S** (valor pequeño): acción integral casi nula, controlador dependiente del término derivativo.
- **SM** (valor medianamente pequeño): un valor medianamente bajo de término integral genera un pequeño incremento en la velocidad respuesta del sistema, afectando un poco su estabilidad.
- **BM** (valor medianamente grande): un valor medianamente grande incrementa la velocidad de respuesta del sistema, pero a su vez incrementa su inestabilidad.
- **B** (valor grande) : acción integral fuerte, mayor velocidad de respuesta del sistema y mayor inestabilidad.

Las funciones de membresía para el caso de K_p y K_d están representadas en las Figuras 2.7 y 2.8. Los conjuntos de reglas difusas de sintonía, correspondientes a estas funciones, se presentan en las Tablas 2.5 y 2.6. La función de membresía para la ganancia K_i , se muestra en la Figura 4.1. Mientras tanto, sus respectivas reglas difusas, se presentan en la Tabla 4.1.

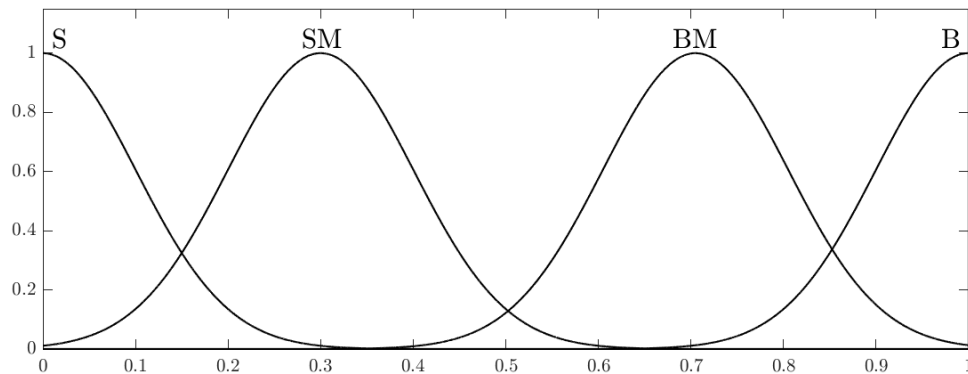


Figura 4.1: Funciones de membresía para la variable K_i' .

Tabla 4.1: Reglas difusas de sintonía para K_i' .

		$\Delta e(k)$							
		NB	NM	NS	ZO	PS	PM	PB	
$e(k)$	NB	B	B	B	B	B	B	B	
	NM	BM	BM	B	B	B	BM	BM	
	NS	SM	BM	BM	B	BM	BM	SM	
	ZO	S	SM	BM	BM	BM	SM	S	
	PS	SM	BM	BM	B	BM	BM	SM	
	PM	BM	BM	B	B	B	BM	BM	
	PB	B	B	B	B	B	B	B	

Por otra parte, en cuanto a implementación, MATLAB proporciona un conjunto de funciones para el diseño, análisis y simulación de sistemas basados en lógica difusa en el Toolbox Fuzzy

Logic. Dentro de este entorno de desarrollo, la herramienta Fuzzy Logic Designer (Figura 4.2) posibilita agregar y eliminar variables de entrada y salida. Además de especificar las funciones de membresía y detallar las reglas difusas de la forma si-entonces. Para acceder a esta herramienta se puede utilizar el comando fuzzy.

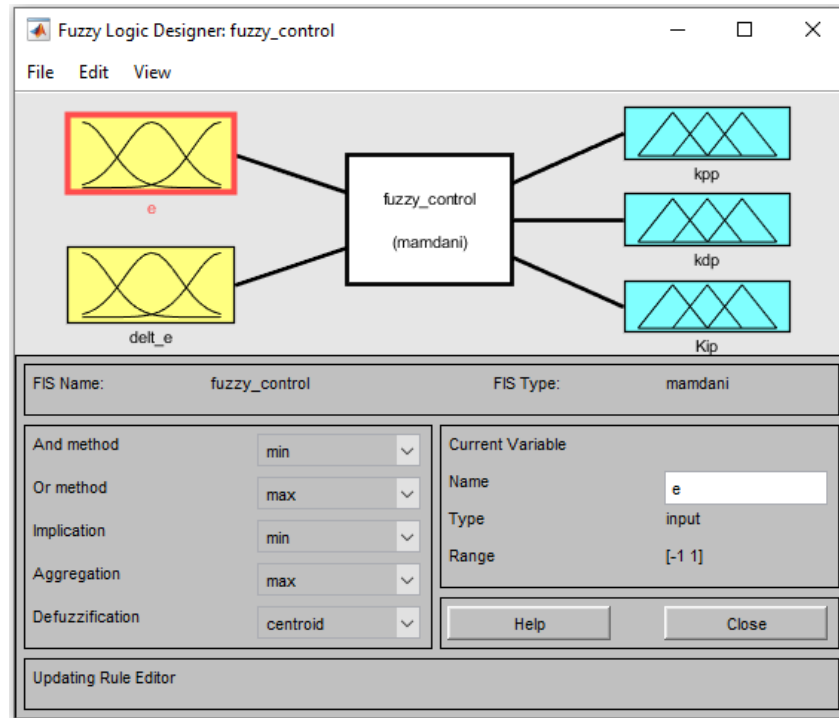


Figura 4.2: Fuzzy Logic Designer de MATLAB.

Se definen dos entradas, la señal del error y su diferencia, con funciones de membresía similares. En la Figura 4.3 se resalta la función de membresía para la variable lingüística de entrada ZO, donde los parámetros configurables son: el rango de valores del error de $[-1 \ 1]$, rango de visualización de $[-1 \ 1]$, nombre ZO, tipo triangular y valores que definen la forma de la función $([-0.3333 \ 0 \ 0.3333])$.

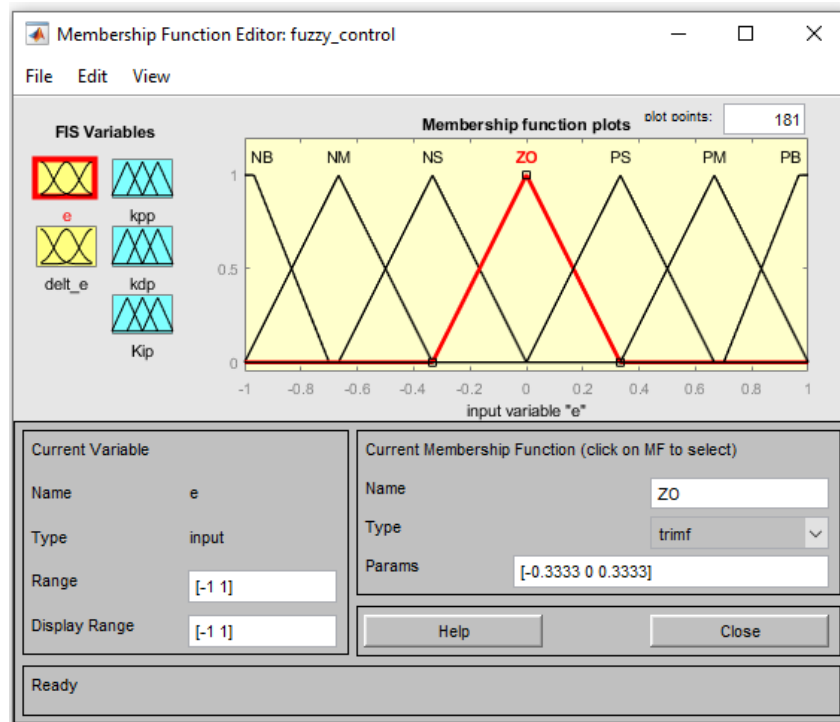


Figura 4.3: Edición de las funciones de membresía.

Luego de agregar las funciones de membresía para las dos entradas y tres salidas; se puede agregar, modificar o eliminar el conjunto de reglas con la herramienta Rule Editor mostrada en la Figura 4.4.

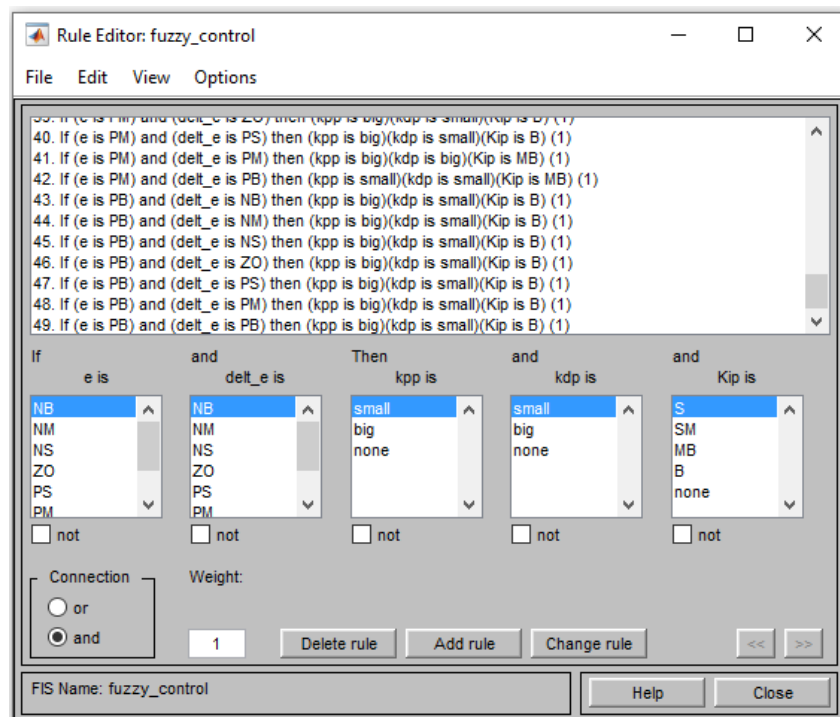


Figura 4.4: Edición de las reglas difusas.

El Toolbox Fuzzy Logic permite seleccionar entre los dos sistemas de inferencia difusa, ya sea de tipo Mamdani o Sugeno. A la vez, es posible pasar de un sistema a otro. En la configuración de este controlador, se utiliza un sistema de inferencia difuso (FIS) tipo Mamdani. Al finalizar la configuración, se genera un archivo con extensión .fis, en el Apéndice B.2 se muestra el contenido del archivo.

La sección de código, presentado en B.1, muestra un ejemplo de la implementación del controlador para un sistema representado en espacio de estados en tiempo discreto, utilizando un tiempo de muestreo de 0.01 seg. Primero, se lee el archivo para generar un sistema difuso denominado 'fis' y, a continuación, se determina la salida con la función 'evalfis'. Para ello, se utiliza la entrada del error y su diferencia. La salida corresponde a los valores normalizados. Por esto, sus ganancias K_p , K_d y K_i , se obtienen a partir de (2.35 - 2.37).

La comparación entre los dos controladores se presenta en la Figura 4.5 para las funciones de transferencia (4.8), (4.9) y (4.10), donde los parámetros del controlador PID para estas funciones son obtenidos por el método de sintonía de ZN en [4]. En la Figura 4.5(a) se muestra la respuesta del sistema $G_1(s)$ empleando el controlador PID para una entrada de escalón unitario, esta presenta un sobreimpulso de 32.98 % y tiempo de estabilización de 4.31 seg. La respuesta del sistema de tercer orden se muestra en la Figura 4.5(b), esta se caracteriza por tener un sobreimpulso de 17.04 % y tiempo de estabilización de 5.46 seg. Por último, la respuesta mostrada en la Figura 4.5(c), correspondiente a $G_3(s)$, presenta un sobreimpulso de 33.74 % y tiempo de estabilización de 3.73 seg.

Utilizando el controlador FGS-PID estos valores se reducen, obteniendo un sobreimpulso de 9.65 % y tiempo de estabilización de 3.19 seg en el sistema de segundo orden; un sobreimpulso de 6.21 % y tiempo de estabilización de 5.01 seg en el sistema de tercer y un sobreimpulso de 8.77 % y tiempo de estabilización de 2.93 seg en el sistema de cuarto orden.

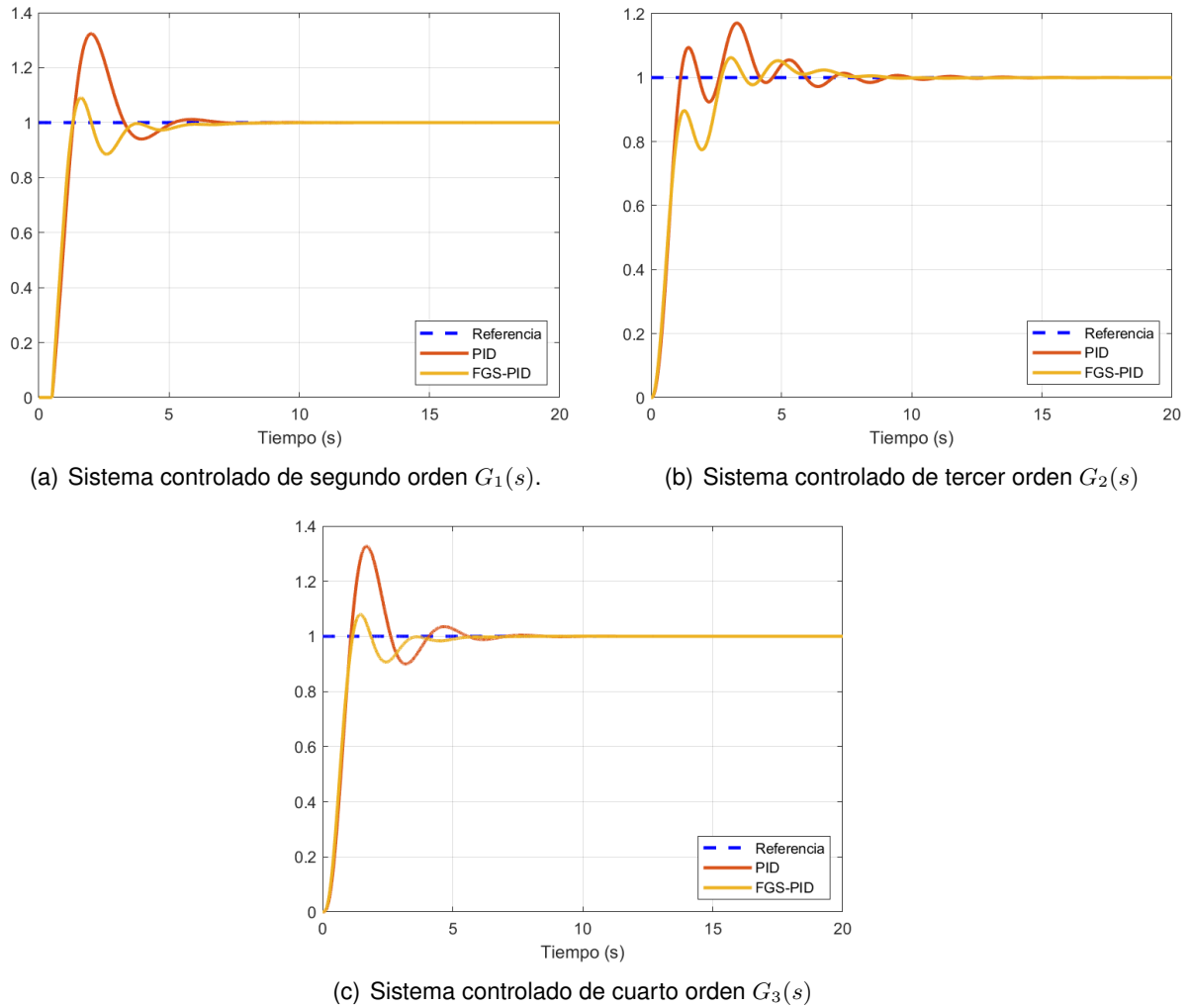


Figura 4.5: Comparación del controlador PID y FGS-PID.

4.1.2. Controlador PID con ganancia programada basado en el algoritmo recocido simulado (SA-PID)

A partir de los fundamentos del algoritmo recocido simulado, pseudocódigo presentado en el Algoritmo 1, a continuación se detalla la implementación respectiva. La solución inicial corresponde a los valores K_p , K_d y K_i . Estos últimos se calculan por cualquier método de sintonía para el controlador PID.

La función objetivo (J) utiliza múltiples variables de desempeño. Además, se basa en la reducción del sobreimpulso (OS), tiempo de estabilización (t_{ss}) y la integral del valor absoluto del error (IAE). Para unir los diferentes índices de desempeño en una sola función objetivo, se utilizan pesos (w_i) para cada variable, la función objetivo es definida en (4.11).

$$J = w_1 \cdot IAE + w_2 \cdot t_{ss} + w_3 \cdot OS \quad (4.11)$$

En las pruebas de funcionamiento se establece los siguientes valores: $w_1 = 0.05$, $w_2 = 0.05$ y $w_3 = 0.9$. Estos son considerados a partir del estudio realizado en [66], donde se analiza el rendimiento del controlador PID sintonizado para control de procesos usando un algoritmo genético multi-objetivo.

En MATLAB, para evaluar la función objetivo, se calcula la respuesta del sistema luego del lazo de control PID para una señal de referencia. Se utilizan distintos valores de ganancia del controlador y el modelo del sistema expresado en su función de transferencia (Extracto de código A.1).

Los parámetros de desempeño OS y t_{ss} son obtenidos utilizando la función 'stepinfo'. Esta última permite calcular las características del modelo de sistema o su conjunto de datos en respuesta a un escalón unitario de entrada. En efecto, el valor de IAE es obtenido mediante su aproximación de la integral por el método trapezoidal. Finalmente, en la evaluación de la función objetivo se verifica si, para un valor de ganancia, se produce inestabilidad en la salida. En el caso de que, el costo generado de la función objetivo tienda a infinito, se verifica su implementación en el Extracto de código presentado en C.2.

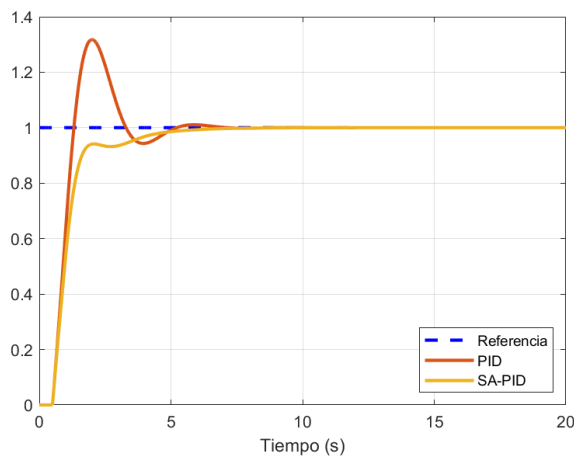
El algoritmo utiliza la función de temperatura con descenso exponencial, caracterizada en (2.14), la misma que se actualiza en cada iteración. Además del número máximo de iteraciones, se define un número de sub-iteraciones. Este proceso tiene la finalidad de que las soluciones vecinas de cada iteración se encuentren alejadas entre sí, ya que depende del cambio de las tres ganancias K_p , K_d y K_i . El algoritmo de SA-PID para la sintonización de ganancias es presentado en el Extracto de código C.1.

La función CreateNeighbor (Apéndice C.3) genera soluciones vecinas de manera aleatoria en cada sub-iteración. Por tanto, al inicio se define la función de probabilidad (4.12), donde $f(x)$ es la probabilidad de elegir una de las tres ganancias a variar en la sub-iteración. Además, la variable aleatoria x es la elección de la ganancia a variar (siendo, $1 = K_p$, $2 = K_i$, $3 = K_d$). Cabe recalcar que, la ganancia K_i tiene mayor probabilidad de ser elegida.

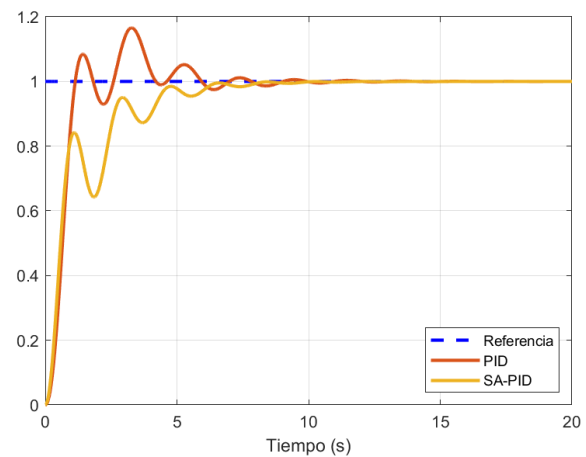
$$f(x) = \begin{cases} 1/5 & , x = 1 \\ 1/2 & , x = 2 \\ 3/10 & , x = 3 \end{cases} \quad (4.12)$$

La determinación del valor de las ganancias es aleatoria. De hecho, cada valor es equiprobable dentro de su rango. Es decir, las ganancias son variables aleatorias continuas con distribución uniforme continua, $K_p \sim U(K_{(p,min)}, K_{(p,max)})$, $K_d \sim U(K_{(d,min)}, K_{(d,max)})$ y $K_i \sim U(K_{(i,min)}, K_{(i,max)})$.

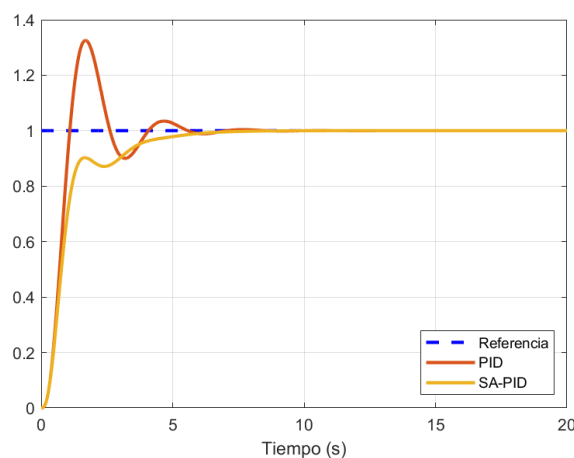
La comparación entre los dos controladores se muestra en la Figura 4.6, para esto se utiliza las funciones de transferencia (4.8), (4.9) y (4.10). Donde la respuesta empleando el controlador SA-PID en el sistema de segundo orden tiene un sobreimpulso casi nulo, de 0.0001 % y un tiempo de estabilización de 3.46 seg. Para el sistema de tercer orden se tiene un sobreimpulso de 0.0015 % y un tiempo de estabilización de 4.32 seg. Por último, el sistema de cuarto orden presenta un sobreimpulso de 0.0029 % y un tiempo de estabilización de 3.69 seg. Estos resultados demuestran una reducción del sobreimpulso y tiempo de estabilización con respecto al controlador PID, incluso un mejor desempeño en la reducción del sobreimpulso que el controlador FGS-PID.



(a) Sistema controlado de segundo orden $G_1(s)$.



(b) Sistema controlado de tercer orden $G_2(s)$.



(c) Sistema controlado de cuarto orden $G_3(s)$.

Figura 4.6: Comparación del controlador PID y SA-PID

Estas pruebas se realizaron utilizando 50 iteraciones, la elección de este valor busca una buena relación entre el número de soluciones vecinas analizadas y la carga computacional que implica. Además, se utiliza 5 sub-iteraciones, este valor considera la probabilidad de variar la ganancia K_p (menor probabilidad de ser elegida, $1/5$). Por último $T_0 = 0.025$ y $\alpha = 0.99$ son valores propuestos por los autores de [75] en la implementación de algoritmo SA estándar en MATLAB.

4.1.3. Controlador PID con ganancia programada basado en el algoritmo A estrella (A*-PID)

En base a los fundamentos teóricos del algoritmo A*, Algoritmo 2, se plantea su implementación para la programación de ganancias del controlador PID. Al iniciar el algoritmo, se define un nodo inicial y un nodo destino. El nodo inicial corresponde a los valores de ganancia K_p , K_d y K_i obtenidos con cualquier método de sintonía del controlador PID. En el caso del nodo destino, se busca el nodo que mejora las características del controlador. Sin embargo, al no conocerlo se tienen dos opciones. La primera consiste en la elección de un nodo aleatorio dentro del espacio de búsqueda y al final del algoritmo considerar como destino a uno de los nodos analizados que mejoraron las características del controlador. La segunda opción es elegir un nodo que no se encuentra en el espacio de búsqueda. En ese caso, el algoritmo analizará todos los valores de su espacio de búsqueda y, al final, seleccionará el nodo con mejor evaluación.

De acuerdo a la Figura 4.7, el espacio de búsqueda es representado en una matriz de tres dimensiones con i filas, j columnas y k capas. Es decir, existen i valores de K_p elegidos entre el rango $[K_{(p,min)}, K_{(p,max)}]$, j valores de K_p elegidos entre el rango $[K_{(d,min)}, K_{(d,max)}]$ y k valores de K_i entre el rango de $[K_{(i,min)}, K_{(i,max)}]$. Por lo tanto, cada nodo de la matriz corresponde una combinación de las ganancias.

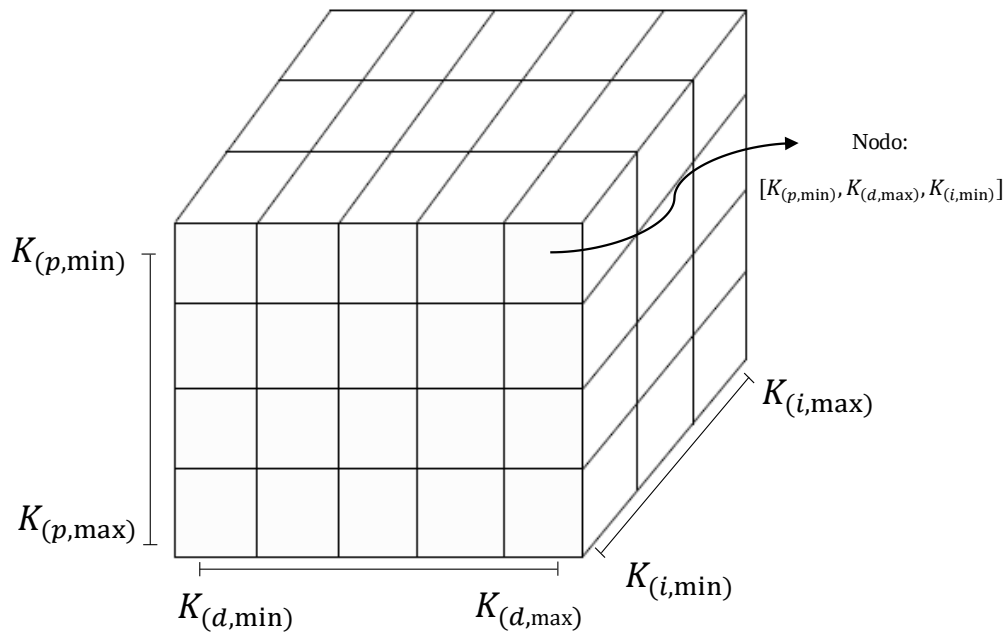


Figura 4.7: Espacio de búsqueda

En MATLAB, para mejorar el proceso de búsqueda, se asigna un identificador a cada nodo. Dicho identificador es un número entero positivo que es una transformación lineal de la fila i , columna j y la capa k , de acuerdo al Extracto de código D.2. Para la comparación realizada en la Figura 4.8 se utiliza una matriz con dimensiones $i = 2$, $j = 6$ y $k = 3$, lo que equivale a un espacio de búsqueda de 36 nodos. Es evidente que la dimensión del espacio de búsqueda está relacionado con la complejidad computacional que implica analizar los nodos. Entre más grande es el espacio de búsqueda, mejores soluciones se encontrarán, sin embargo, implica mayor carga computacional, además de requerir mayor cantidad de memoria.

El Extracto de código D.1, desarrollado en MATLAB, muestra el proceso de búsqueda del algoritmo, donde un nodo o ganancia es tratado como una coordenada en el espacio tridimensional. Por tanto, la función de expansión del algoritmo corresponde a la elección de las coordenadas adyacentes con respecto a la coordenada actual. Mientras que, los nodos sucesores son los valores de ganancias en las coordenadas adyacentes.

El algoritmo A* emplea la función de evaluación $F = G + H$ para encontrar las ganancias que mejoren el proceso de control, para este algoritmo H es descrita de manera similar por (4.11) permitiendo estimar si la ganancia actual está cerca de la solución óptima. Es importante considerar que entre más cerca esté de la solución, el valor de H disminuirá. Por otro lado, G descrita en (4.13), toma en cuenta el costo que genera el moverse de la ganancia inicial hasta

la actual. Sin mencionar que entre más se aleje de la ganancia inicial, aumenta la inestabilidad en el sistema. Por ello, el costo es mayor.

$$G(x, y, z) = G(X_1, Y_1, Z_1) + \sqrt{(x - X_1)^2 + (y - Y_1)^2 + (z - Z_1)^2} \quad (4.13)$$

La expresión (4.13) se basa en la distancia Euclidiana entre dos puntos en un espacio tridimensional. El primer punto (X_1, Y_1, Z_1) corresponde al índice del nodo de inicio y el punto (x, y, z) al índice del nodo actual.

Al evaluar una nueva ganancia, su identificador se guarda en la lista de abiertos. Esto se realiza para evitar que, en otra iteración, se calcule nuevamente su estimación H . En el caso de G , su valor se actualiza siempre que este sea menor al calculado previamente.

Luego de analizar los nodos expandidos, a partir de la lista de abiertos de los nodos, se determina el valor de F con menor costo. Posteriormente, se elimina dicho nodo de la lista de abiertos y se coloca en la lista de cerrados. Se continúa el proceso de búsqueda, tomando la coordenada de este nodo como inicio.

La comparación entre los dos controladores se muestra en la Figura 4.8, para esto se utiliza las funciones de transferencia (4.8), (4.9) y (4.10). Donde la respuesta del sistema de segundo orden empleando el controlador A*-PID tiene un tiempo de estabilización de 3.96 seg y sobreimpulso de 1.14 %. Para el sistema de tercer orden se tiene un sobreimpulso de 0.65 % y un tiempo de estabilización de 4.11 seg. Por último, el sistema de cuarto orden presenta un sobreimpulso de 1.030 % y un tiempo de estabilización de 3.25 seg. Los resultados muestran una disminución en estos dos índices de desempeño con respecto al controlador PID, además de un menor sobreimpulso que el controlador FGS-PID.

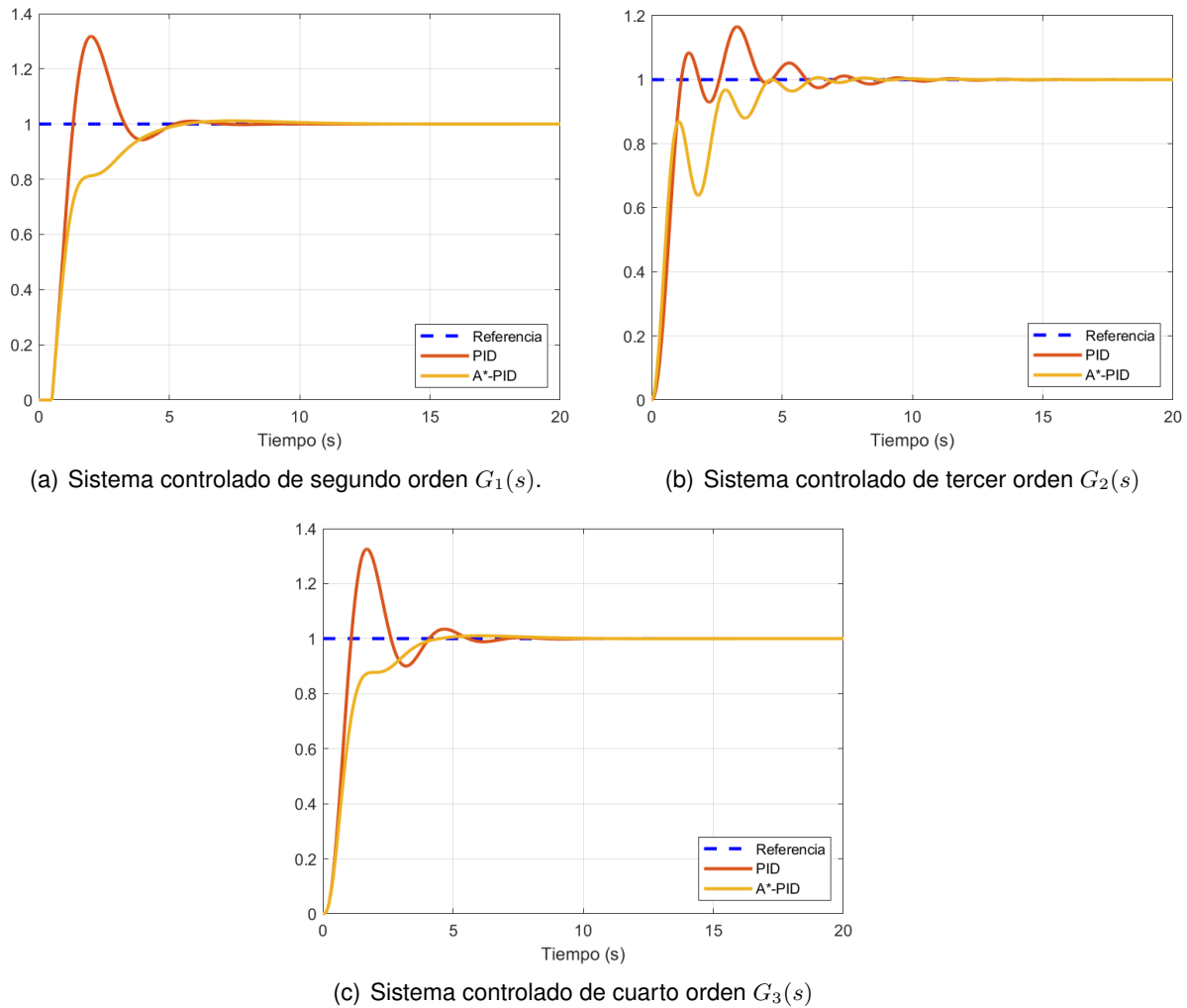


Figura 4.8: Comparación del controlador PID y A*-PID

4.2. Simulación del controlador PID con ganancia programada y sistema multitanque.

En esta sección, se describe el modelado del sistema multitanque y los criterios considerados en la simulación de los algoritmos de [IA](#) para la programación de ganancias del controlador [PID](#).

4.2.1. Modelado del sistema multitanque

El sistema multitanque es un sistema [MIMO](#) que consta de dos variables de salida y tres variables de control. Las variables de salida representan el nivel de agua del tanque 1 y tanque 2. Mientras que, las variables de entrada son el porcentaje de apertura de las válvulas; donde la primera válvula controla el flujo de entrada al sistema, la segunda válvula el flujo que sale del tanque 1 e ingresa al tanque 2 y la tercera válvula el flujo de salida del tanque 2. El diagrama

del sistema, se presenta en la Figura 4.9.

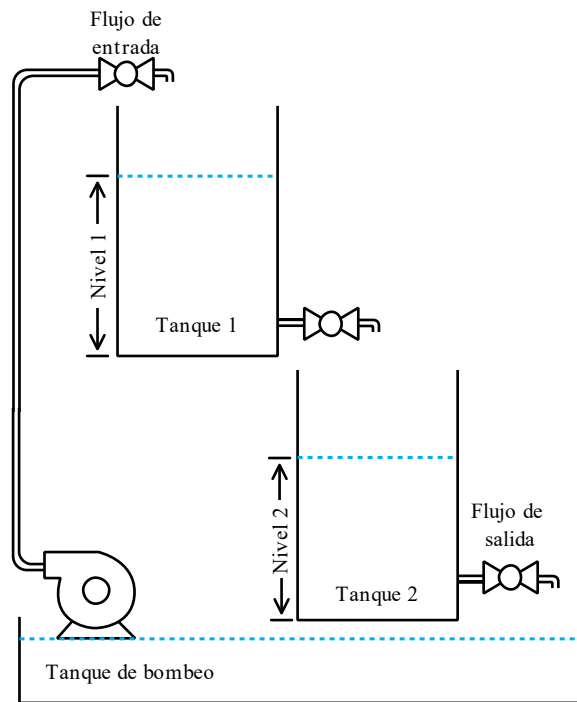


Figura 4.9: Esquema simplificado del sistema multitanque. *Imagen adaptada de [5].*

Las ecuaciones diferenciales que describen la dinámica del sistema son:

$$\begin{aligned}\frac{dh_1}{dt} &= \frac{1}{A_1} \left(u_1 q_0 - u_2 a_2 \sqrt{2gh_1} \right) \\ \frac{dh_2}{dt} &= \frac{1}{A_2} \left(u_2 a_2 \sqrt{2gh_1} - u_3 a_3 \sqrt{2gh_2} \right)\end{aligned}\tag{4.14}$$

Donde:

- A_1, A_2 : área de la sección transversal de los tanques 1 y 2.
- a_2, a_3 : área de la sección transversal de los tubos que conecta las válvulas 2 y 3.
- g : aceleración de la gravedad.
- h_1, h_2 : nivel de agua de los tanques 1 y 2.
- q_0 : caudal permanente o nominal de la bomba.
- u_1, u_2, u_3 : Porcentaje de apertura de las válvulas 1, 2 y 3, valores en el rango de 0 (válvula completamente cerrada) y 1 (válvula completamente abierta).

Se agregan fallas al sistema por medio de fugas provocadas por un agujero en el fondo de

los tanques 1 y 2. Estas fugas se modelan como un flujo de salida. Por tanto, el modelo del sistema descrito anteriormente se representa de la siguiente manera:

$$\begin{aligned}\frac{dh_1}{dt} &= \frac{1}{A_1} \left(u_1 q_0 - l_1 - u_2 a_2 \sqrt{2gh_1} \right) \\ \frac{dh_2}{dt} &= \frac{1}{A_2} \left(u_2 a_2 \sqrt{2gh_1} - l_2 - u_3 a_3 \sqrt{2gh_2} \right)\end{aligned}\quad (4.15)$$

Donde:

- a_{l_1}, a_{l_2} : área de la sección transversal de los agujeros que provocan la fuga en los tanques 1 y 2.
- l_1, l_2 : flujo de salida en las fugas de los tanques 1 y 2.

$$l_1 = k_{l_1} a_{l_1} \sqrt{2gh_1} \quad (4.16)$$

$$l_2 = k_{l_2} a_{l_2} \sqrt{2gh_2} \quad (4.17)$$

- k_{l_1}, k_{l_2} : indicador de la presencia de fugas en los tanques 1 y 2;

$$k_{1,2} = \begin{cases} 0, & \text{no existe fuga} \\ 1, & \text{existe fuga} \end{cases} \quad (4.18)$$

Las constantes físicas del sistema se presentan en la Tabla 4.2.

Tabla 4.2: Constantes físicas modelo del sistema multitanque. *Elaborada con datos de [5]*

Constante	Valor	Unidad
a_2	4.380×10^{-5}	m^2
a_3	4.601×10^{-5}	m^2
A_1	0.04	m^2
A_2	0.04	m^2
q_0	6.667×10^{-5}	m^3/s
g	9.81	m/s^2

4.2.2. Diseño del sistema de control

El sistema de control de ganancia programada ([GS-Gain Scheduling](#)), diseñado a partir de algoritmos de inteligencia artificial, es implementado en el sistema multitanque de acuerdo al esquema presentado en la Figura 4.10. Donde h_1 y h_2 son los niveles de referencia medidos de los tanques; ref_{h_1} y ref_{h_2} son los niveles de referencia deseados. Las señales de error e_1 , e_3 se obtienen a partir de la diferencia del nivel medido con el deseado. Mientras que, e_2 se

obtiene a partir de la diferencia entre los niveles medidos ($h_2 - h_1$) con la diferencia de los niveles de referencia deseados ($ref_{h_1} - ref_{h_2}$). Las señales de error ingresan al controlador PID y son empleadas por los algoritmos de ganancia programada para el ajuste en línea del controlador. Finalmente, las señales de control u_1 , u_2 y u_3 , realizan acción sobre la planta a través de la apertura o cierre de las válvulas.

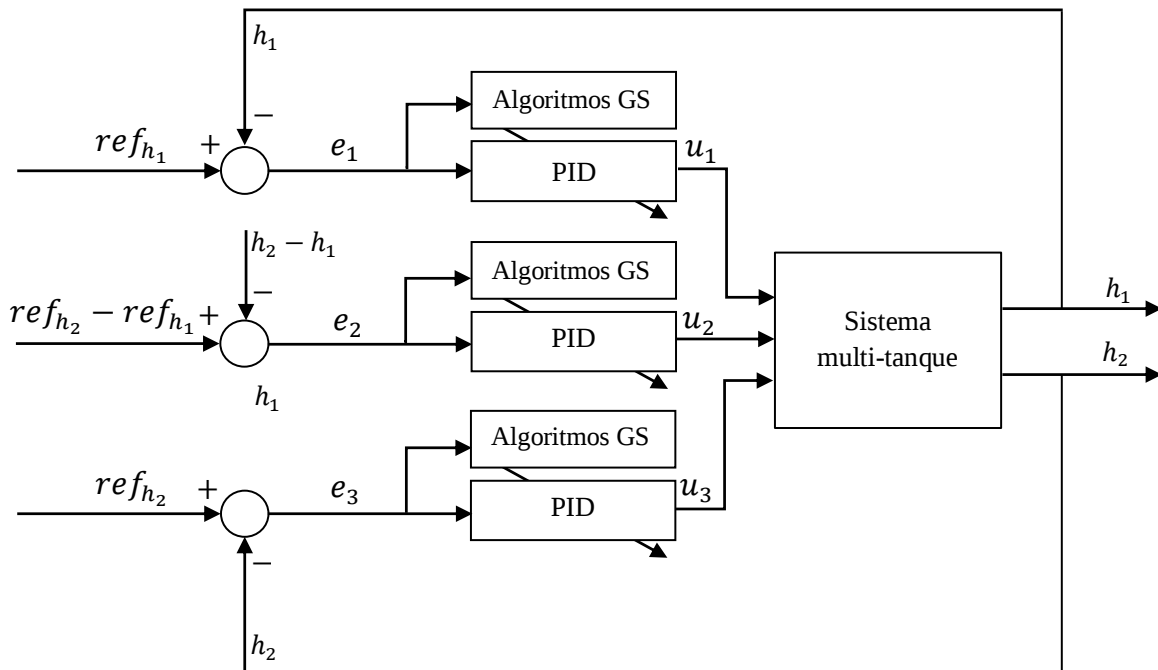


Figura 4.10: Sistema de control PID con ganancia programada y sistema multitanque.

En este proyecto, se analizan cuatro esquemas de control. El PID clásico donde sus valores de ganancia son obtenidos por cualquier método de sintonía o de manera experimental. El esquema de control FGS-PID trabaja con el error y su primera diferencia, se caracteriza por su ajuste de ganancias en tiempo real. Es decir, luego de cada muestra, se realiza el ajuste de las ganancias. Respecto al esquema de control SA-PID, trabaja en línea, pero su ajuste lo hace luego de un determinado número de muestras. La razón es que este esquema de control analiza el comportamiento del sistema ante cambios de referencia, evaluando su tiempo de estabilización y sobre-impulso. Finalmente, el esquema de control A*-PID, similar al anterior, trabaja luego de determinadas muestras.

Estos controladores son implementados, independientemente, en cada una de las válvulas para el control del nivel de agua en los tanques 1 y 2. El tiempo de muestreo que se utiliza es de un segundo, debido a que el cambio de estado de abierto a completamente cerrado de la válvula proporcional es muy lento (> 5 s).

4.2.3. Simulación del sistema en MATLAB

Se realiza la simulación de los controladores en el entorno de programación MATLAB, usando su entorno gráfico Simulink. En la Figura 4.11, se muestra el diagrama en bloques de la implementación. A partir de ello, se realiza la comparación entre el esquema de control PID y los algoritmos GS.

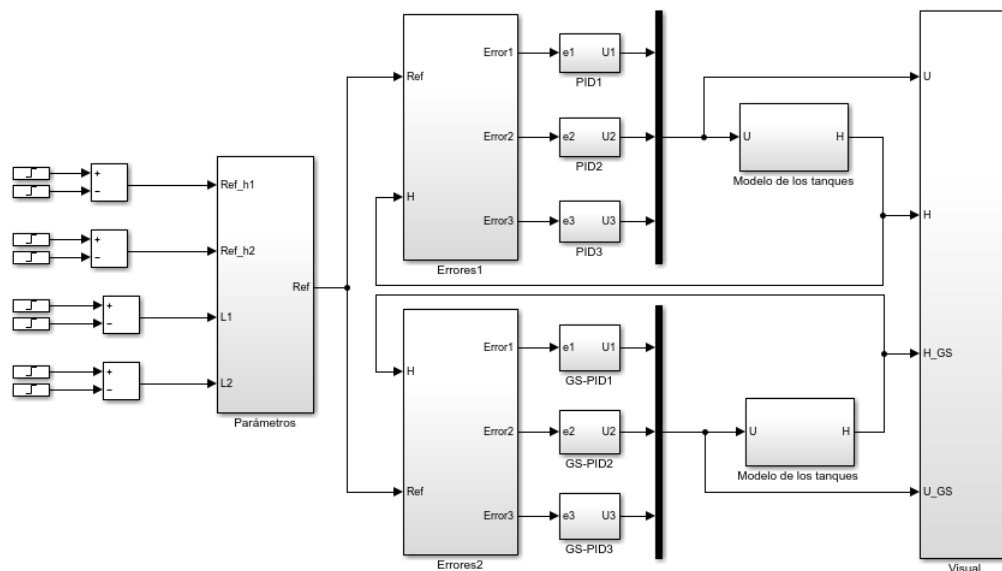


Figura 4.11: Sistema de control PID con ganancia programada y sistema multitanque en Simulink.

La simulación del sistema está compuesta por subsistemas entre los que se diferencian:

1. Parámetros

Este subsistema es único para los cuatro esquemas de control. De hecho, contiene los niveles de referencias deseados de los tanques, Ref_{h1} , Ref_{h2} y los parámetros del sistema de fugas.

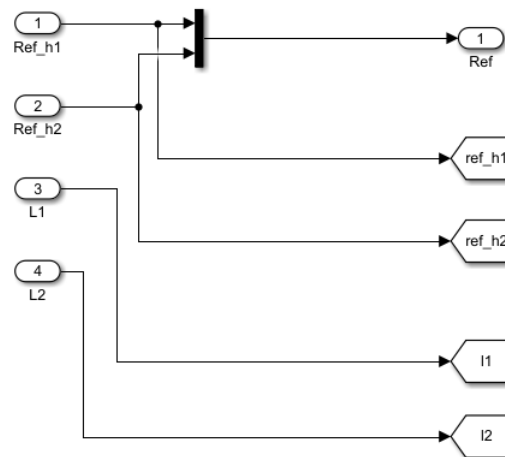


Figura 4.12: Subsistema Parámetros.

2. Errores

Este subsistema determina las señales de error a partir de los niveles de referencia deseados con los medidos a la salida del sistema. A pesar de su naturaleza de subsistema independiente para cada controlador, todos los niveles de referencia realizan la misma función.

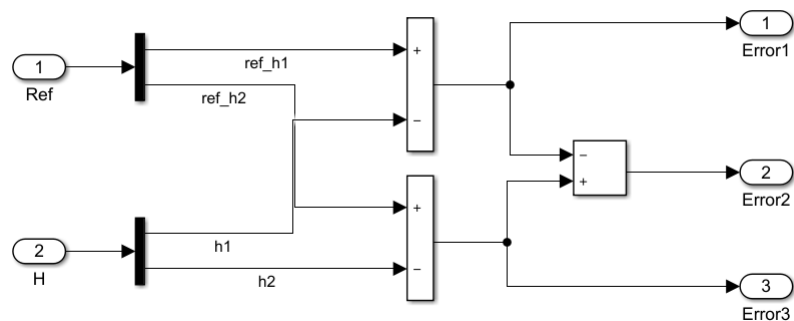


Figura 4.13: Subsistema Errores

3. Modelo de tanques

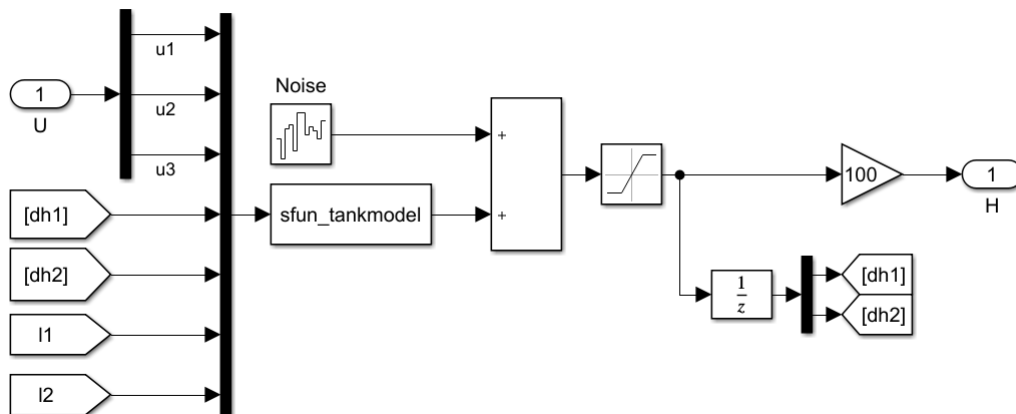


Figura 4.14: Subsistema del modelo del sistema de multitank.

Como su nombre lo indica, este subsistema modela el sistema multitank, resolviendo las ecuaciones de diferencia descritas en la Sección 4.2.1. Para ello, parte de funciones-S o funciones del sistema, estas funciones permiten ampliar las capacidades de Simulink y diseñar bloques de propósito general. Tal es el caso de modelar un sistema dinámico complejo caracterizado por su no-linealidad. El bloque de funciones-S hace uso de archivos que necesitan un código estructurado para que Simulink lo pueda ejecutar. Por esto, MATLAB proporciona plantillas de ejemplificación. En este caso, para modelar el sistema, se modifica el archivo de `sfuntmpl.m`.

4. Visualización

Este subsistema permite visualizar las gráficas resultantes obtenidas en la simulación. Estas consisten en la visualización de los valores de: nivel de fluido de los tanques H_1 y H_2 , nivel de referencia deseado Ref_{h_1} y Ref_{h_2} ; y, señales de control u_1 , u_2 y u_3 .

5. Controlador PID

La Figura 4.15 muestra la representación en bloques del controlador PID en Simulink. El esquema presentado ha sido implementado en cada una de las tres válvulas. De hecho, su señal de control, escrita en (4.19), se obtiene discretizando la función de transferencia del controlador PID continuo mostrada en (2.18).

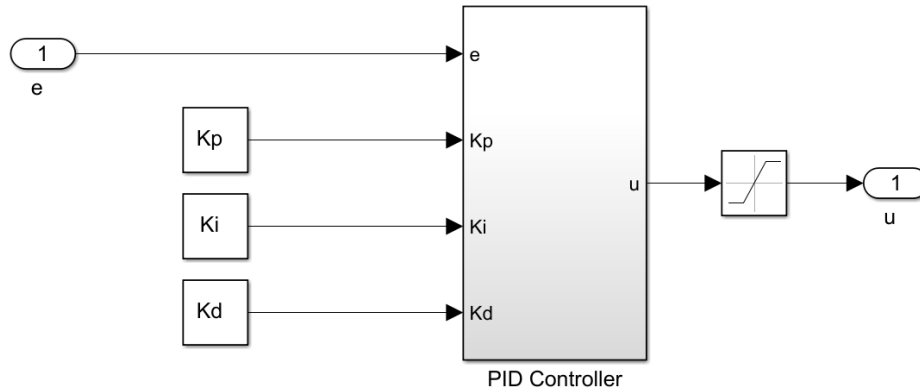


Figura 4.15: Controlador PID.

$$u[k] = K_p e[k] + K_i T_s \sum_{i=1}^n e[i] + \frac{K_d}{T_s} \Delta e[k] \quad (4.19)$$

Donde:

- $u[k]$: señal de control.
- $e[k]$: señal del error obtenido de la diferencia de la señal de salida y la señal de referencia.
- $\Delta e[k] = e[k] - e[k - 1]$
- T_s : tiempo de muestreo.
- K_p , K_d y K_i : parámetros de controlador PID ($K_d = T_d K_p$, $K_i = \frac{K_p}{T_i}$)

La señal de salida corresponde al porcentaje de apertura de las válvulas, por lo que su valor es saturado entre el rango de 0 y 1. En este contexto, 0 es completamente cerrado y 1 completamente abierto. En el caso de la válvula 3, la señal de control es negada, ya que controla el flujo de salida del tanque 2; a diferencia de las otras válvulas que controlan el flujo de entrada del tanque 1 y tanque 2. Los parámetros del controlador son obtenidos de manera experimental, es decir, sintonía mediante prueba-error [13]. Este método de sintonización de lazo cerrado sugiere fijar el valor de K_p igual o cercano a uno, reduciendo K_d y K_i al mínimo (cero). El valor de K_p se ajusta tomando en cuenta la acción de control proporcional descrita en la Sección 2.2.1. Luego de encontrar el valor se ajusta de la misma manera para K_d y K_i .

Los parámetros de los controladores mostrados en la Tabla 4.3 son elegidos de acuerdo a los siguientes criterios y consideraciones:

- 1) El sistema multitanque es un sistema MIMO, por lo que utilizar el método de sintoni-

zación ZN genera inestabilidad, esto sumado a que el sistema no es lineal [76].

- 2) Las señales de control u_1 , u_2 y u_3 representan el porcentaje de apertura de las válvulas, cuyos valores están comprendidos entre 0 y 1.
- 3) El error máximo y mínimo es conocido. Para $e_1 = ref_{h_1} - h_1$ es máximo cuando $h_1 = 0$, $ref_{h_1} = 30$ (altura del tanque 1 en cm) y mínimo cuando $h_1 = 30$, $ref_{h_1} = 0$, de igual forma $\max(e_3) = 30$ y $\min(e_3) = -30$. Por último, para $e_2 = e_3 - e_1$ se tiene que el valor $\max(e_2) = 60$ y $\min(e_2) = -60$.
- 4) El método de prueba-error sugiere, en un principio, fijar las ganancias K_d y K_i a cero (únicamente acción proporcional donde $u(t) = K_p e(t)$). Por tanto, si $\max(u_1) = 1$, entonces puede ser expresada como $K_p \max(e_1) = 1$; usando el criterio 3) da como resultando $K_p = \frac{1}{\max(e_1)} = 0.033$, este valor indica el mínimo K_p que garantiza una apertura completa de la válvula 1 cuando el error es máximo, por lo que, establece el límite inferior de K_p para el método. El mismo análisis se realiza para e_2 y e_3 , donde el mínimo K_p es 0.0166 y 0.033 respectivamente.
- 5) El límite superior de K_p para el método se hace considerando un caso crítico del sistema real, suponiendo que para todos los valores de e_1 tal que $e_1 \geq 0.1$ hace que $u_1 = 1$, es decir, la salida se satura provocando una apertura completa de la válvula en cada pequeño error. Este efecto no es deseado, ya que la respuesta del sistema se encontrará en constante oscilación (similar al control on-off). Por tanto, este criterio establece que el límite superior es $K_p = \frac{1}{e_1} = 10$, lo mismo aplica para e_2 y e_3 .
- 6) Los ajustes de K_d y K_i se realiza teniendo en cuenta las acciones básicas de control de la Tabla 2.1. Por tanto, partiendo desde cero se aumenta el valor de K_d , para disminuir el sobreimpulso y tiempo de establecimiento. Y se ajusta K_i para eliminar el error de estado estacionario. Además, se utiliza valores bajos para evitar que de la señal de control se sature constantemente.

Tabla 4.3: Parámetros de los controladores PID.

	PID 1	PID 2	PID 3
K_p	0.2	0.2	0.7
K_d	0.01	0.1	0.1
K_i	0.000575	0.0007	0.0015

Los valores de ganancias presentados en Tabla 4.3 son usados para las pruebas en la simulación. En el caso del sistema real se utiliza los mismos valores de K_p y K_d , y se ajusta levemente el término integral K_i . Para mejorar la respuesta del sistema, reduciendo

do el tiempo de estabilización, los valores de ganancias considerados para las pruebas en el sistema real son: $K_i = 0.0005$, $K_i = 0.0001$ y $K_i = 0.001$ en los controladores PID 1, 2 y 3 respectivamente.

6. Controlador FGS-PID

El esquema de control FGS-PID presentado en la Figura 4.16, es implementado sobre una función-S. Los parámetros de entrada que necesita esta función, corresponde a la señal del error e_k y su diferencia Δe_k . Debido a que estas dos señales de entrada deben encontrarse en el rango de -1 y 1 de acuerdo a sus funciones de membresía, se limita su salida sobre este rango. La función `sfun_fuzzycontrol` implementada en el Extracto de código B.3, calcula las ganancias del controlador de la manera similar que en el Extracto código B.1 para un sistema representado en espacio de estados.

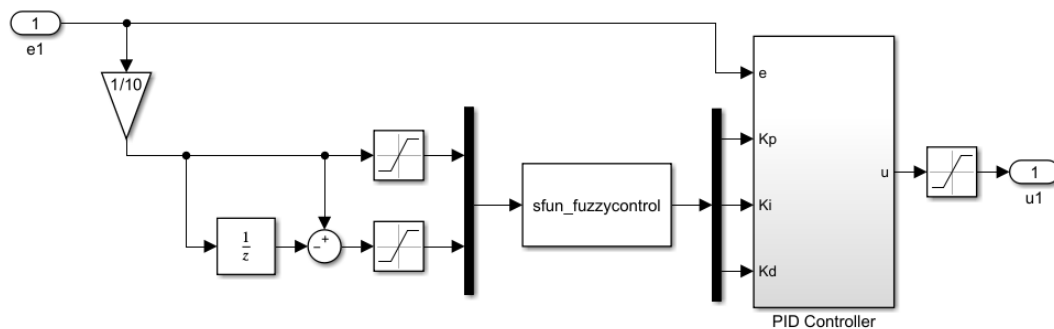


Figura 4.16: Bloque del controlador FGS-PID.

El algoritmo FGS-PID usa reglas y razonamiento difuso para determinar los parámetros del controlador PID en cada periodo de muestreo. Dichos parámetros están comprendidos en un rango mínimo y máximo, especificado en la Tabla 4.4.

Tabla 4.4: Parámetros de los controladores FGS-PID.

	PID 1	PID 2	PID 3
$[K_{(p,min)}; K_{(p,max)}]$	$[0.1066; 0.2]$	$[0.1066; 0.2]$	$[0.3731; 0.7]$
$[K_{(d,min)}; K_{(d,max)}]$	$[0.0107; 0.02]$	$[0.1067; 0.2]$	$[0.1067; 0.2]$
$[K_{(i,min)}; K_{(i,max)}]$	$[5.75 \cdot 10^{-6}; 4.9 \cdot 10^{-4}]$	$[7 \cdot 10^{-6}; 5.9 \cdot 10^{-4}]$	$[1.5 \cdot 10^{-5}; 1.3 \cdot 10^{-3}]$

7. Controlador SA-PID

La Figura 4.17, muestra la representación en bloques del controlador SA-PID. El algoritmo principal se encuentra dentro la función `sfun_SA` implementada en el Extracto de

código C.4. A diferencia del esquema FGS-PID, este controlador no actúa sobre cada muestra, sino luego de un determinado número de muestras especificadas antes de la ejecución (en la simulación del sistema multitanque se eligen 500 muestras para asegurar la estabilización en la respuesta). Así, el algoritmo se habilita en cada cambio de referencia, realizando en cada activación una actualización de las iteraciones y sub-iteraciones, determinación de las ganancias vecinas y cálculo del valor de la función objetivo. El algoritmo se detiene luego de cumplir con el número de iteraciones y sub-iteraciones que fueron especificadas. A partir de ese punto, las ganancias con mejores resultados son definidas como los nuevos parámetros del controlador. El algoritmo utiliza las condiciones iniciales descritas en la Sección 4.1.2 a excepción de las iteraciones. Recordando que el número de sub-iteraciones se eligen para asegurar al menos un cambio de la ganancia K_p en 5 activaciones del algoritmo, y el número de iteraciones se eligen en relación al número de soluciones vecinas analizadas y la carga computacional que implica. Para la simulación del sistema multitanque se reducen a 8 iteraciones, lo que equivale a 40 activaciones de algoritmo o cambios de referencia.

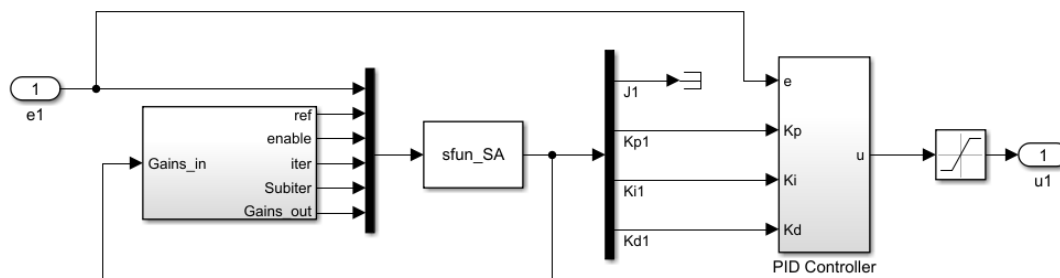


Figura 4.17: Bloque del controlador SA-PID.

8. Controlador A*-PID

El algoritmo principal de este esquema de control, que se muestra en la Figura 4.18, es implementado en la función `sfun_Astar` descrita en el Extracto de código D.3. De forma similar al controlador SA-PID, este sistema se habilita en los cambios de referencia. Además, en él se determina el costo de la función F usando muestras de la señal de salida luego del cambio de referencia.

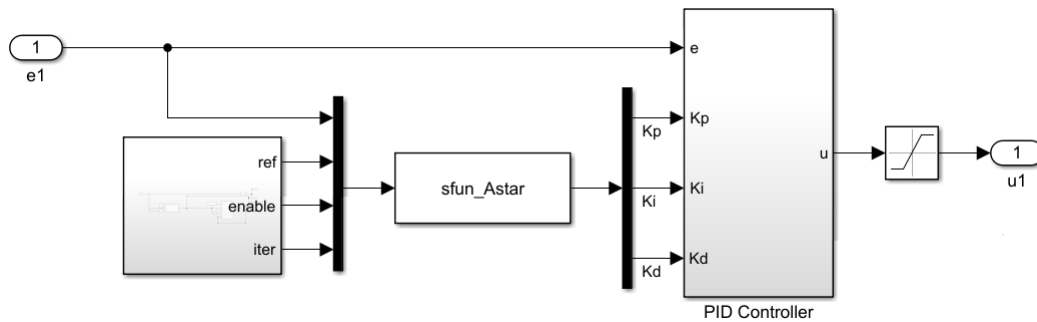


Figura 4.18: Bloque del controlador A*-PID.

Por otra parte, en la ejecución del algoritmo lo ideal es encontrar el nodo destino, sin embargo, puede darse el caso que dicho nodo no esté en el espacio de búsqueda, en esta situación todos los nodos son analizados, seleccionado aquel nodo con menor costo en la heurística H . Al finalizar, se establece los valores de ganancias asociados a este nodo como los nuevos parámetros del controlador. Si bien, para un espacio de búsqueda pequeño no es problema analizar todos los nodos, cuando crece dicho espacio resulta conveniente tener control de la cantidad de nodos que se desea analizar, es por ese motivo que para la implementación del este algoritmo se agrega el parámetro de iteraciones. El algoritmo se ejecuta hasta completar el número de iteraciones inicialmente establecidas o encontrar el nodo destino. Para la simulación del sistema multitank se estableció 40 iteraciones, que representa la misma cantidad de soluciones vecinas que se analiza en el controlador [SA-PID](#). El espacio de búsqueda está dimensionado de tal forma que cuente con el número de nodos necesarios para cumplir con las iteraciones. Además, para tener el mismo número de valores de ganancias se elige una matriz cuadrada con dimensiones $i = 4$, $j = 4$ y $k = 4$. Como ejemplo, en la Figura 4.19 se muestra el espacio de búsqueda para el controlador de la tercera válvula (**PID 3**).

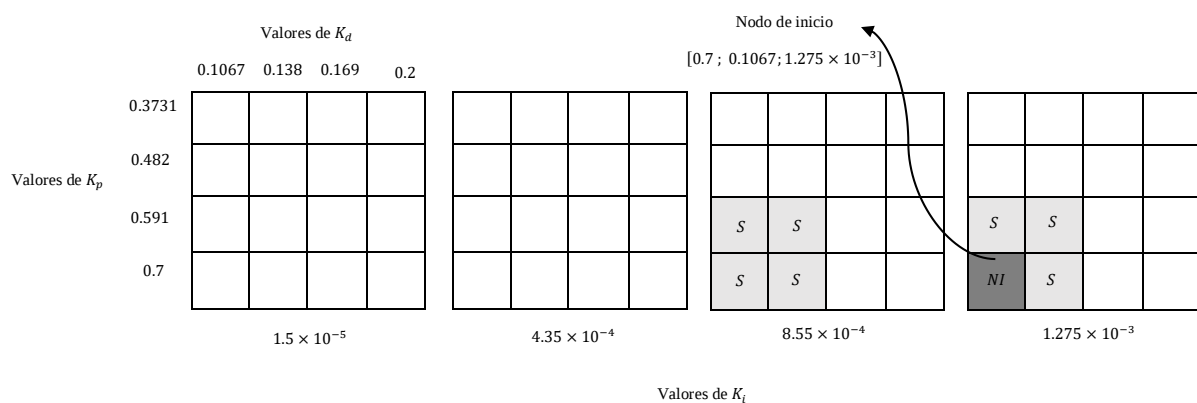


Figura 4.19: Definición del espacio de búsqueda del controlador A*-PID.

La Figura 4.19 describe de manera gráfica el funcionamiento del controlador PID-A*, donde cada recuadro representa un nodo con la combinación de las ganancias K_p , K_d y K_i . Además, al inicio del algoritmo se define el nodo de inicio (gris oscuro), este nodo tiene los valores de ganancia más cercanos al controlador PID. Los nodos sucesores (gris claro) son nodos adyacentes al nodo actual, que en este caso es el nodo de inicio. El nodo NI es colocado en la lista de cerrados y los nodos sucesores S en la lista de nodos abiertos. Para cada nodo sucesor se evalúa la función objetivo F y entre las soluciones analizadas se elige la menor, $\min(F)$. Para el ejemplo presentado, suponiendo que el nodo $N = [0.591; 0.138; 8.55 \cdot 10^{-4}]$ es el nodo como menor evaluación de F (véase la Figura 4.20), el algoritmo continúa el proceso de búsqueda generando nuevos sucesores en el orden indicado (de S_1 hasta S_{26}). En el caso de que el nodo está en la lista de cerrados (nodo rojo) su F es mínima, por tanto, no se evalúa nuevamente, por otra parte, si el nodo sucesor está en la lista de abiertos (nodo azul), se verifica que el costo de G sea menor y si lo es se actualiza. El algoritmo continúa este proceso hasta que la lista de nodos abiertos esté vacía, es decir, se haya analizado todos los nodos. También finaliza cuando el número iteraciones llega a su límite o cuando se encuentra el nodo destino. El proceso de generación de nodos, en su programación en MATLAB, se apoya de la función `delimitar_coord()` y `contador`, presentes en el Extracto de código D.3.

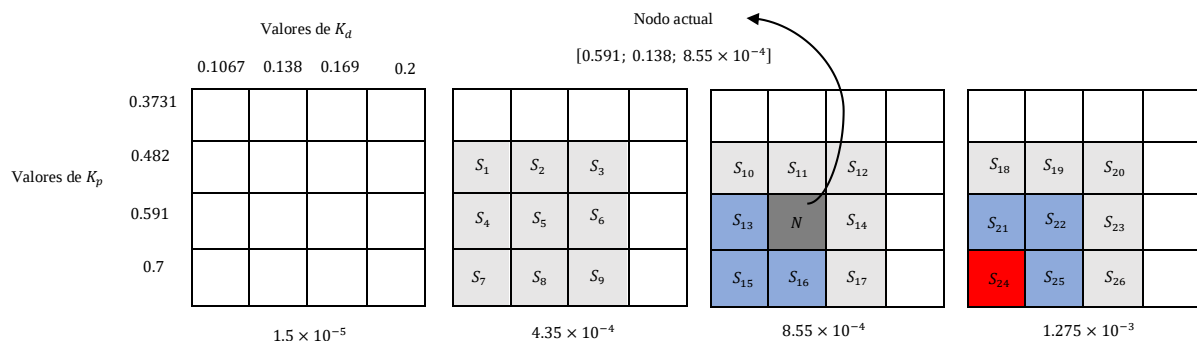


Figura 4.20: Determinación de los nodos sucesores del controlador A*-PID.

Tanto el controlador SA-PID como el A*-PID usan el mismo rango de valores que el control FGS-PID mostrando en la Tabla 4.4; sin embargo, cuando el sistema deja de funcionar correctamente es necesario un ajuste de los valores de K_i , el nuevo rango cuando el sistema se encuentra en falla está dado por (4.20) y (4.21). Debido a que la respuesta del sistema no alcanza la referencia, presencia de fugas en los tanques, es necesario aumentar el sobreimpulso, a su vez, eliminar el error de estado estacionario. Precisamente por esto es que el

nuevo rango de ganancias K_i está por encima del anteriormente considerado.

$$K_{(i,min)} = 1.2K_i \quad (4.20)$$

$$K_{(i,max)} = 2.5K_i \quad (4.21)$$

4.3. Implementación y adaptación de los algoritmos de ganancia programada y controladores PID en la placa STM32 núcleo-144

La programación de los microcontroladores STM32 se realiza en entornos de desarrollo con lenguaje de programación C/C++. Existen algunas opciones para su programación, incluido el IDE de Arduino. Sin embargo, esta opción limita las capacidades del microcontrolador. Una opción mejor es STM32Cube IDE, un entorno de desarrollo libre creado por el propio fabricante ST. El entorno antes mencionado, se basa en Eclipse y el conjunto de herramientas de GCC para la compilación. Además, este ambiente cuenta con generación de código de inicio, configuración de los periféricos de forma visual, modo debugger y otras funcionalidades que maximizan las opciones de desarrollo.

En esta sección, se explica el proceso de implementación de los algoritmos en lenguaje C. Además, se aclara cómo ciertas funciones utilizadas en MATLAB son adaptadas para el diseño del controlador sobre STM32.

4.3.1. Adaptación de las funciones básicas

Los algoritmos GS utilizan sentencias básicas como: if, for, while, switch que pueden ser traducidas directamente a su lenguaje en C. Otras funciones como max, min se definen en base a estas sentencias o directamente utilizando librerías como math.h. No obstante, la complejidad en pasar de un lenguaje a otro radica en la definición de variables y realizar operaciones con matrices/arrays. En el caso de MATLAB, estas operaciones son sencillas e intuitivas. Al emplear el lenguaje de programación C, cada variable debe definirse antes de ser utilizada. Para las matrices/arrays además de definir el tipo de dato, es necesario especificar su dimensión. Por ello, se generan ciertas complicaciones. Sin embargo, las complicaciones se solventan, fácilmente, al crear matrices/arrays de forma dinámica.

Otro de los aspectos importantes a la hora de programar en C, es el uso de punteros. Los

punteros contienen la información de la posición de memoria de una variable. Esto es útil a la hora de definir funciones; ya que, en lugar de enviar toda la matriz/array, se indica únicamente la posición de memoria donde se encuentra.

4.3.2. Adaptación de las funciones avanzadas

Todas las funciones avanzadas de MATLAB son creadas a partir de las sentencias básicas usadas en la programación. Por tanto, es posible adaptarlas en otro lenguaje de programación como C. En esta sub-sección, se describe la metodología usada en la adaptación de los algoritmos en C.

1. Funciones del algoritmo PID

- `PID_Init()`: inicializa los valores de error actual e_k , error anterior e_{k-1} , error acumulado $\sum_{i=1}^n e_i$, salida de la planta y_k , referencia r y la señal de control u_k . Además, se definen las ganancias K_p , K_d y K_i del controlador. La función se muestra en el Extracto de código [A.2](#).
- `PID_control()`: como se muestra en el Extracto de código en [A.3](#), los parámetros de entrada de esta función serán el error actual, error pasado, error acumulado, las ganancias y el tiempo de muestreo. Esto con el objetivo de obtener la señal de control u_k utilizando (2.18). Además, en cada ejecución del algoritmo, se actualizan las señales de error pasado y error acumulado.

2. Funciones del algoritmo FGS-PID

- `Fuzzy_Init()`: la función trabaja de forma parecida al diseñador de lógica difusa de MATLAB, en ella se especifica el rango de valores que puede tomar las entradas y salidas. En este caso, las entradas serán las señales de error e_k y su diferencia Δe_k . Mientras que, las salidas serán las ganancias normalizadas K'_p , K'_d y K'_i . Además, se definen NB , NM , NS , ZO , PS , PM y PB como las variables lingüísticas de las dos entradas. Adicionalmente, *small* y *big*, para las salidas K'_p y K'_d . Mientras que, las variables S , SM , BM y B , para la salida K'_i .

Las funciones de membresía son creadas a partir de las funciones `trimf()`, `trapmf()` y `guassmf()` que calculan su valor usando (2.3), (2.5) y (2.6). Luego de establecer

las funciones de membresía de entrada y salida, se especifican las reglas difusas. Estas últimas tienen la forma “1 1 , 2 1 4” donde la coma separa las entradas de las salidas y su numeración es acuerdo al orden de definición de la variable lingüística (Extracto de código B.4).

- `eval_fuzzy()`: en analogía a la función `evalfis` en MATLAB, esta función evalúa el sistema de inferencia difusa de tipo Mamdani. Así, selecciona sus dos entradas, e_k y Δe_k , y las fuzzifica. En otras palabras, al valor medido del error o diferencia del error, se le asigna su valor de pertenencia en las funciones de membresía. En este caso, al ser dos variables en el antecedente de las reglas difusas, se utiliza el operador **y/and** para que la evaluación dé como resultado un valor único. Este operador representa la operación $\min$, en los conjuntos difusos. Luego de la fuzzificación, se realiza el proceso de inferencia de Mamdani por mínimos, devolviendo la misma cantidad de conjuntos difusos inferidos como el número de reglas. Por tanto, para defuzzificar es necesario agrupar estos conjuntos, esta etapa de agrupación emplea diferentes estrategias. La estrategia más común, usa la operación $\max$. Finalmente, en la última etapa, para convertir los valores difusos en valores que entienda el controlador, se efectúa la defuzzificación, para lo cual se emplea el método de centroide.

La función antes mencionada, se muestra en el Extracto de código B.8 y es apoyada por otras funciones para la etapa de fuzzificación, defuzzificación, reglas, etc., presentadas en los Extractos de código B.9 - B.12.

- `FGS_control()`: la función ajusta la señal del error y su diferencia para que se encuentre dentro del rango de $[-1, 1]$. Además, como la salida de la función `eval_fuzzy()` devuelve los valores normalizados, esta función se encarga de determinar las ganancias K_p , K_d y K_i por medio de (2.35), (2.36) y (4.5). En adición, los valores mínimos y máximos de las ganancias, se definen previamente en la función `FGS_Init()`. Estas funciones se detallan en Extracto de código B.13 y B.14.

3. Funciones del algoritmo SA-PID

- `SA_Init()`: esta función contiene los valores iniciales de los parámetros que utiliza el algoritmo SA-PID. Entre ellos, se encuentran el número de iteraciones y sub-iteraciones, la cantidad de muestras tomadas de la señal de salida para el análisis de su estimación, la temperatura inicial T_0 , la constante α de la función de des-

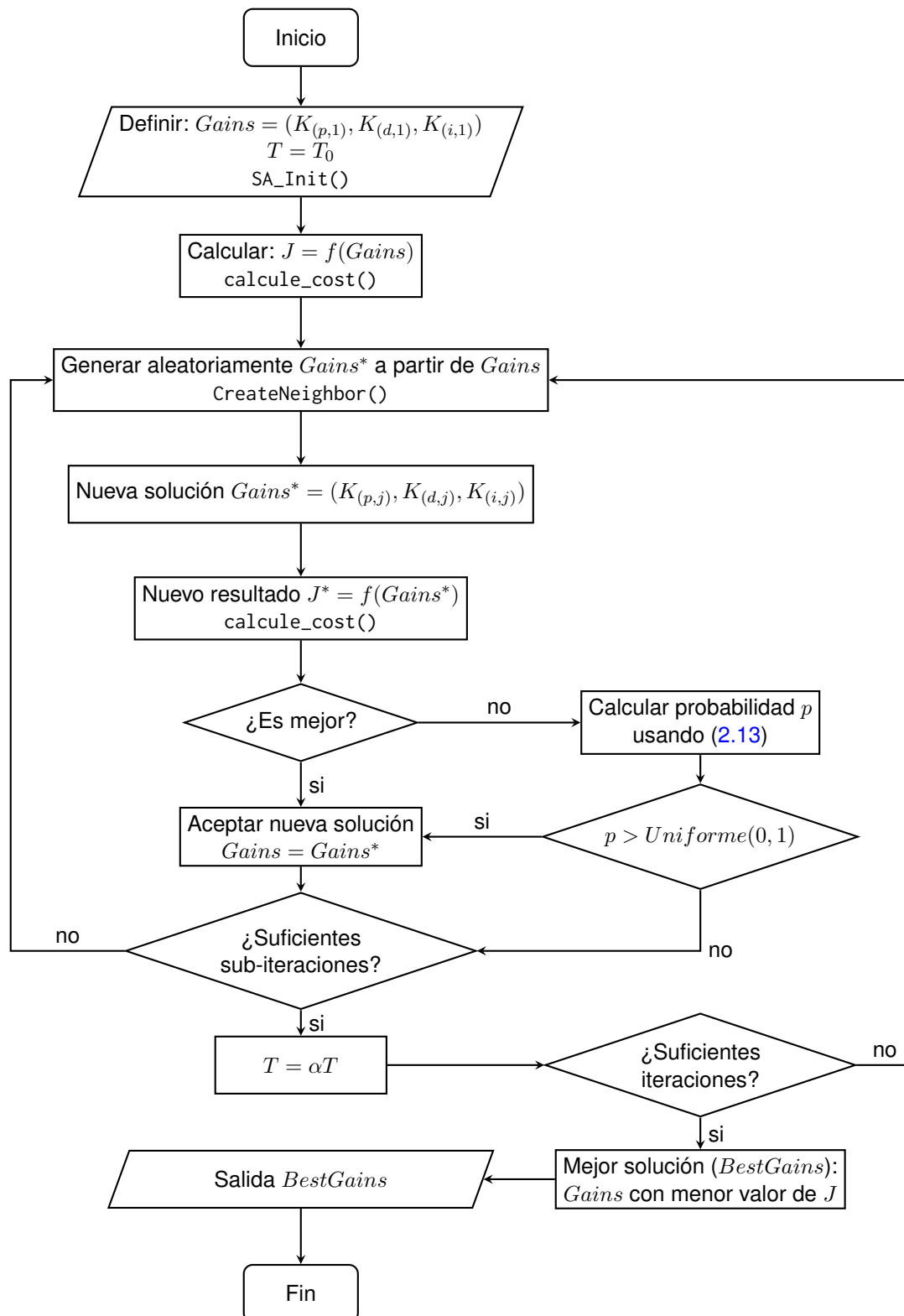


Figura 4.21: Diagrama de flujo del algoritmo SA-PID.

censo exponencial, el rango de ganancias usado para la generación de soluciones vecinas y algunas variables secundarias para conocer cuándo inicia o se habilita el algoritmo. La función se presenta en el Extracto de código C.5.

- `Simulated_Annealing()`: es el algoritmo principal para la determinación de ganancias del controlador SA-PID, trabaja de la misma forma que la implementada en MATLAB. El funcionamiento del algoritmo parte de los fundamentos conceptuales del algoritmo de recocido simulado aplicado al problema planteado, programación de ganancias del controlador PID, dando como resultado el algoritmo mostrado en la Figura 4.21. Al iniciar el algoritmo, la función `calcule_cost()` determina la solución inicial generada por las ganancias establecidas del controlador PID. A continuación, se crean ganancias vecinas por medio de la función `CreateNeighbor()`, comenzando con la primera iteración y sub-iteración. El algoritmo se habilita en cada cambio de referencia y utiliza determinadas muestras para calcular el costo. Si este nuevo valor es menor a la solución calculada, se actualiza. Caso contrario, la solución será aceptada o no basándose en la función de probabilidad descrita anteriormente en (2.13). Al completar el número de iteraciones y sub-iteraciones, el algoritmo asigna las ganancias del controlador PID aquellas con la mejor solución. El algoritmo se muestra a detalle en el Extracto de código C.6.
- `calcule_cost()`: realiza la evaluación de la función objetivo, definida por (4.11). Esta función hace uso de los parámetros de desempeño OS , t_{ss} y IAE . En MATLAB, se obtienen estos parámetros utilizando la función `stepinfo`. Sin embargo, en C es necesario determinar cada parámetro. La metodología para el cálculo de los parámetros de desempeño, se especifica a continuación:
 - `SettlingTime()`: el tiempo de establecimiento t_{ss} es el tiempo en el que la señal de salida se mantiene dentro de un margen preestablecido. En el algoritmo se especifica este margen como el 5 % de $|y_{final} - y_{init}|$. Matemáticamente, t_{ss} es el primer tiempo en que se cumple la siguiente relación.

$$|y(t) - y_{final}| \leq 0.05 \times |y_{final} - y_{init}| \quad (4.22)$$

Esta función se muestra en el Extracto de código C.8.

- `Overshoot()`: el porcentaje de sobreimpulso es determinado por (4.23).

$$OS = 100 \times \max \left(\frac{y(t) - y_{init}}{y_{final} - y_{init}} - 1 \right) \quad (4.23)$$

- `IAE_calc()`: la integral del error absoluto es aproximado por el método trapezoidal definido matemáticamente por (4.24).

$$\int_a^b f(x)dx \approx \frac{b-a}{2N} \sum_{n=1}^N (f(x_n) + f(x_{n+1})) \quad (4.24)$$

El cálculo de IAE se presenta en el Extracto de código C.10.

Existen otros índices de desempeño como ITAE, ISE, ITSE y el tiempo de subida t_r . Estos índices fueron agregados con el fin de mejorar el algoritmo. Sin embargo, no son utilizados en este trabajo, sus códigos se presentan en C.11-C.14.

- `CreateNeighbor()`: crea soluciones vecinas de las ganancias del controlador PID. En cada sub-iteración, cualquiera de las ganancias se elige para cambiar su valor de manera aleatoria. Esta función trabaja de manera idéntica a la antes explicada en su implementación en MATLAB. Su correspondiente código en C, se muestra en el Extracto de código C.15.

4. Funciones del algoritmo A*-PID

- `Astar_Init()`: esta función, detallada en el Extracto de código D.4, inicializa las variables que son definidas en el algoritmo A*-PID. Entre ellas están: número de iteraciones, lista de cerrados y abiertos, dimensión del espacio de búsqueda, espacio de búsqueda, matriz F , H y G inicializadas en cero o valores no definidos, valores de K_p , K_d y K_i que serán analizados, la coordenada del nodo de inicio tomada a partir del nodo con la ganancia más cercana o igual a la establecida en el controlador PID, coordenada del nodo destino elegida aleatoriamente, o de un nodo que no se encuentra en el espacio de búsqueda, y matriz de identificadores de los nodos creada utilizando la función mostrada en el código D.6. Esta última hace uso de otras funciones como `reserve_memory_Astar()`, la cual asigna memoria de manera dinámica (Extracto de código D.5).
- `Aestrella()`: esta función se habilita en cada cambio de referencia y determina las ganancias del controlador PID mediante el algoritmo A*. Además, la función traba-

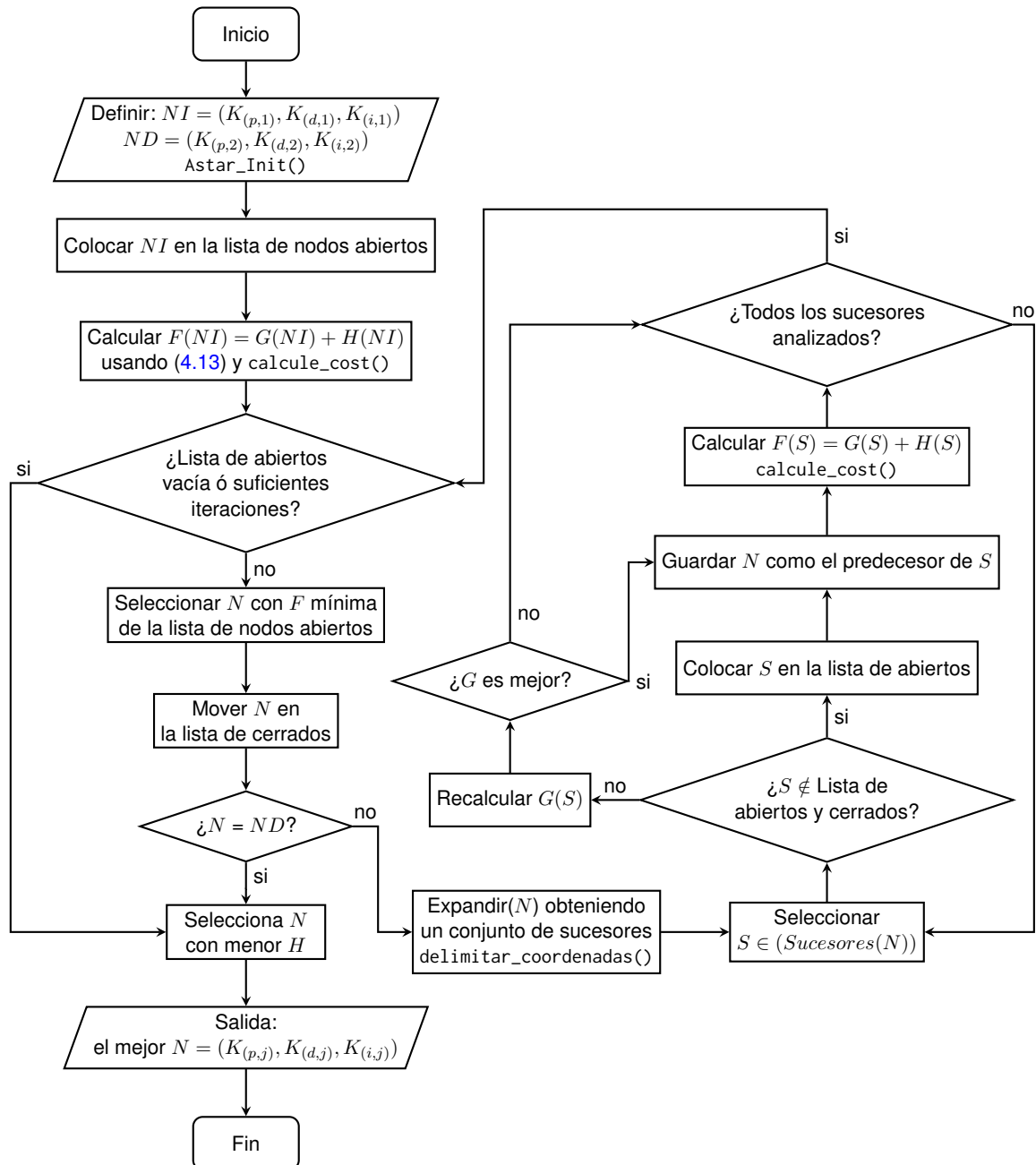


Figura 4.22: Diagrama de flujo del algoritmo A*-PID.

ja de manera similar a la implementada en MATLAB, donde se consideraba a las ganancias como una coordenada del espacio tridimensional. El funcionamiento del algoritmo parte de los fundamentos conceptuales del algoritmo de A^* aplicado al problema planteado, programación de ganancias del controlador PID, dando como resultado el algoritmo mostrado en la Figura 4.22. Al inicio de esta función, los parámetros por defecto del controlador se encuentran asignados en una coordenada, denotada como la coordenada de inicio. Esta coordenada hace referencia a la ubicación del nodo de inicio; este nodo se evalúa usando $F(NI) = G(NI) + H(NI)$, donde $G(NI) = 0$ por ser la distancia en un mismo punto y $H(NI)$ es la estimación realizada usando la función `calcule_cost()`. El nodo de inicio es el único nodo de la lista de abiertos; por ello, es elegido como el de F mínima, se mueve de la lista de nodos abiertos a la de cerrados y es expandido generando un conjunto de sucesores. Los sucesores son los nodos adyacentes a dicha coordenada; para cada sucesor, se verifica que no se encuentre en la lista de abiertos o cerrados. En el caso de que no se encuentre, se hace la evaluación del sucesor, $F(S) = G(S) + H(S)$, y se agrega el nodo a la lista de abiertos. Si se encuentra, se recalcula $G(S)$ y el algoritmo se repite, tomando el nodo con la F mínima de la lista de abiertos y finaliza cuando esta lista esté vacía o cuando se exceda el número máximo de iteraciones establecidas. Esta acción se agrega para tener un mejor control sobre el algoritmo. Al finalizar, el algoritmo usa el espacio de búsqueda analizado para establecer las nuevas ganancias del controlador PID a partir del nodo con el menor valor de la heurística H . La implementación de la función en el lenguaje de programación C, se muestra a detalle en el Extracto de código D.7.

- `delimitar_coordenadas()`: es una función secundaria, como su nombre lo indica, es utilizada para limitar el espacio de búsqueda y seleccionar los nodos que son expandidos, su código se presenta en el código D.8 y es apoyada de una función auxiliar denominada `contador`. Esta función es un contador de base 3 mostrada en el Extracto de código D.9.

4.4. Integración de la placa STM32 al tablero de control y comunicación con el sistema multitanque

En esta sección, se describe la metodología implementada en la integración de los algoritmos de ganancia programada para el control del sistema multitanque y su comunicación con el

tablero de control.

4.4.1. Descripción del sistema

El sistema multitanque es el resultado del proyecto previo “Diseño e implementación de los sistemas de instrumentación, comunicaciones y control de un sistema multitanque” disponible en [5], consta de dos tanques de reserva, un tanque de bombeo, tres electroválvulas que controlan el flujo de entrada y salida de los tanques y sensores para medir el nivel del agua.

Los sensores y electroválvulas se conectan directamente al tablero de control, donde su componente principal, la placa de desarrollo STM32 núcleo-144, se encarga de la medición de las señales, gestión de los algoritmos de control y generación de señales de control para las electroválvulas. Para esta última, la placa STM32 genera señales PWM. Por ello, trabaja de manera conjunta con un módulo convertidor de señales, así como módulos relés que acondicionan las señales de voltaje de la placa con las electroválvulas. Además de las tareas presentadas anteriormente, la placa STM32 utiliza la comunicación serial para enviar datos del nivel medido de los tanques, nivel de referencia y de las señales de control. Estos datos son interpretados en un programa en MATLAB, mismo que se encarga de presentar los resultados de manera gráfica. El esquema general del sistema se presenta en la Figura 4.23

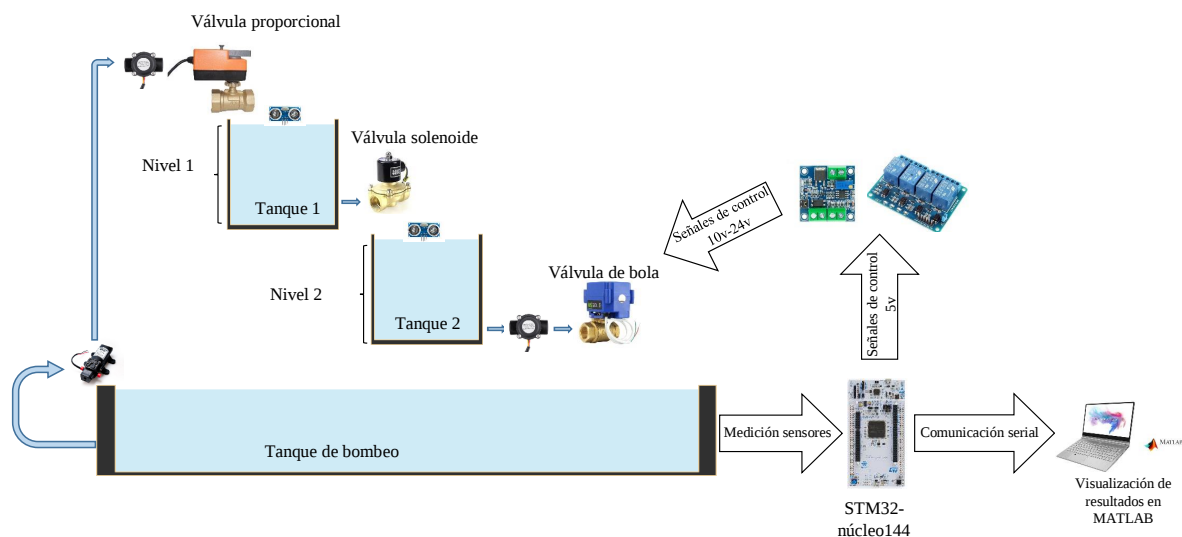


Figura 4.23: Esquema general del sistema multitanque y sistema de control.

4.4.2. Diagrama de instrumentación y procesos del sistema

En la Figura 4.24 se presenta el diagrama de procesos e instrumentación del sistema multitank y el sistema de control. Este diagrama es diseñado utilizando la norma ISA/ANSI 5.1. El sistema cuenta con una bomba de agua conectada con el tanque de bombeo y la válvula de retención FV100 encargada de proteger la bomba. FV108 es una válvula que permite introducir perturbaciones al sistema o liberar la presión en caso de ser necesario. El sistema cuenta con tres electroválvulas FCV101, FCV103 y FCV106. La primera se encarga de controlar el flujo de entrada del primer tanque de reserva, la segunda controla el flujo de salida de tanque 1 y el flujo de entrada del tanque 2; y, la tercera electroválvula controla el flujo de salida del tanque 2. Existen cuatro sensores, LT104 y LT105 son sensores de distancia que miden el nivel de agua en los tanques de reserva; FT102 y FT107 son sensores de caudal que miden el flujo de entrada y salida. Los sensores son conectados directamente al nodo FC200 que, a su vez, controla las tres electroválvulas. Además, este nodo se conecta con el nodo UG300, mismo que permite la toma de datos y su representación gráfica.

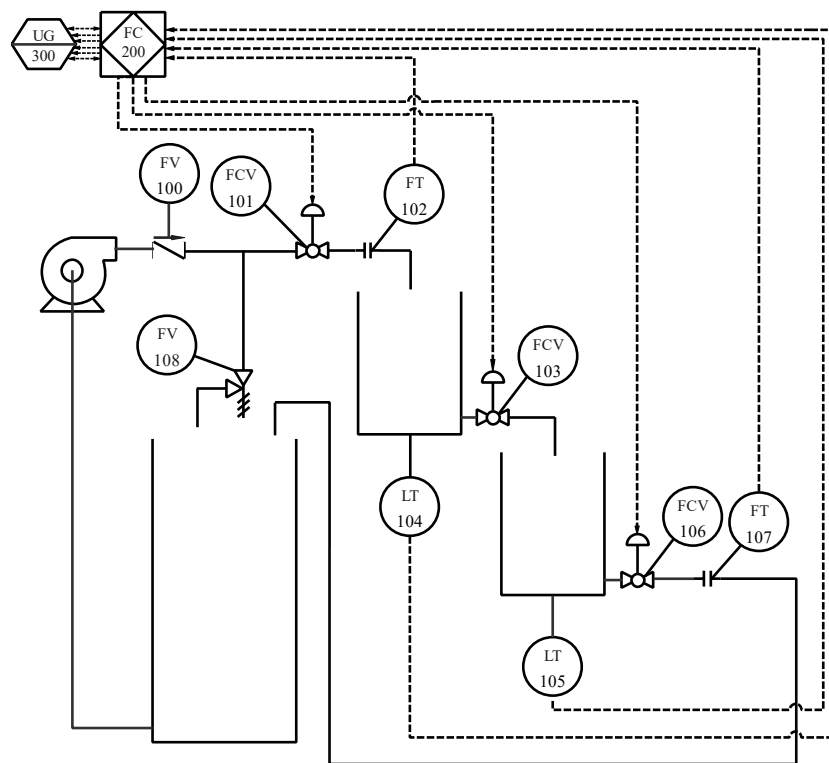


Figura 4.24: Diagrama de proceso e instrumentación del sistema multitank y sistemas de control. Imagen adaptada de [5].

4.4.3. Principales componentes del sistema multitanque y comunicación con el tablero de control

En esta sub-sección, se describen los principales componentes del sistema multitanque y el sistema de control. Además de aquellos componentes que permiten la comunicación entre ambos.

1. Bomba Favson F3012

Favson F3012 es una bomba de agua autocebante de 12 V de alimentación, misma que soporta hasta una presión de 100 PSI y proporciona un caudal de 4 L/min. Además, la bomba es adaptada al sistema usando tuberías de 3/8", y es mostrada en la Figura 4.25.

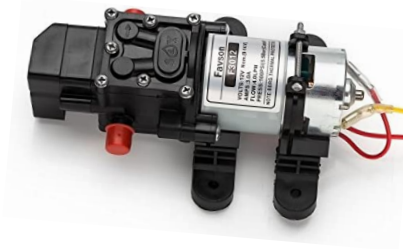


Figura 4.25: Bomba Favson F3012. Imagen tomada de [5].

2. Electroválvula Winner WVA4-3

La Electroválvula Winner WVA4-3 mostrada en la Figura 4.26 es una válvula proporcional de tipo bola. Este tipo de válvula, se alimenta con un voltaje de 24 V y su apertura es controlada por un voltaje variable de 0 a 10 V. Además, se conecta a la entrada del sistema usando tuberías de 1/2".



Figura 4.26: Electroválvula Winner WVA4-3. Imagen tomada de [5].

3. Electroválvula BACOENG 2W-15

Es una válvula de tipo solenoide, se alimenta con un voltaje de 12 V y su apertura es controlada únicamente por su alimentación. Además, posee solamente dos estados, completamente abierta y completamente cerrada. Se conecta a la salida del tanque 1 usando tuberías de 1/2", esta válvula se observa en la Figura 4.27.



Figura 4.27: Electroválvula BACOENG 2W-15. Imagen tomada de [6].

4. Electroválvula U.S. SOLID USSMSV00002

Es una válvula proporcional de tipo bola (Figura 4.28), se alimenta con un voltaje de 24 V y su apertura es controlada por el tiempo de alimentación. Se conecta a la salida del tanque 2 usando tuberías de 3/4".



Figura 4.28: Electroválvula U.S. SOLID USSMSV00002. Imagen tomada de [7].

5. Sensor ultrasónico HC-SR04

Es un sensor de distancia utilizado para medir el nivel de agua de los tanques de reserva y se alimenta con un voltaje de 5 V. Además, el sensor determina la distancia de un objeto mediante el ultrasonido (onda 40 kHz), tiene margen de error de ± 3 mm y su

rango de medición va desde 2 cm a 4.5 m. La distancia a partir de la duración de rebote de la onda ultrasónica es determinada por (4.25).

$$Distancia(m) = \frac{t_{medido} \cdot v_{sonido}}{2} \quad (4.25)$$

Donde $v_{sonido} = 343 \text{ m/s}$ y t_{medido} es el tiempo, determinado por el sensor, que tarda la onda en ir y regresar. El sensor se muestra en la Figura 4.29.

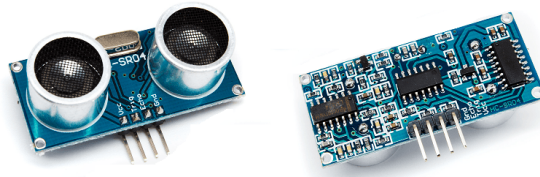


Figura 4.29: Sensor ultrasónico HC-SR04. *Imagen tomada de [8].*

6. Sensor DIGITEN

El sensor mostrado en la Figura 4.30, es un sensor de flujo de agua, se alimenta con un voltaje de 5 V, es utilizado para medir flujos entre 1 – 30 L/min y soportar presiones de hasta 254 PSI. Así, se conecta para la medición del flujo de entrada y salida del sistema usando tuberías de 3/4".



Figura 4.30: Sensor de flujo de agua DIGITEN. *Imagen tomada de [9].*

7. STM32F413ZH núcleo-144

La placa STM32F413ZH núcleo-144 presentada en la Figura 4.31 posee un microcontrolador de 32 bits basados en ARM Cortex-M4 a 100 MHz, cuenta con 1.5 Mbytes de memoria Flash y 320 Kbytes de SRAM, tiene dos tipos de conectores. El primero es el conector ST Zio que incluye soporte con ARDUINO Uno V3 y los cabezales ST morpho

que mejoran la funcionalidad y accesibilidad de la plataforma. La ventaja de esta plataforma es que tiene integrado depurador/programador ST-LINK/V2-1. La disposición de pines junto a otras características se presenta en manual del usuario [10].

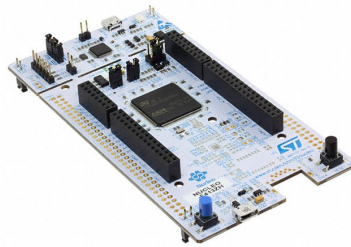


Figura 4.31: Placa STM32F413ZH núcleo-144. *Imagen tomada de [10].*

8. Módulo relé de 4 canales

El módulo relé permiten controlar las electroválvulas BACOENG 2W-15, U.S. SOLID USSMSV00002 y la bomba. Por ello, es necesario un voltaje de 5 V para alimentar las bobinas. El voltaje de operación se toma directamente del controlador o para reducir su consumo de una fuente externa. La señal de control puede ser de 3.3 V o 5 V, como se observa la Figura 4.32 posee aislamiento eléctrico por medio de optoacopladores e indicadores de activación mediante LEDs. Su tiempo de acción es de 5 ms/10 ms. A diferencia de su tiempo de muestreo igual a un segundo, el tiempo de activación es mucho menor, por lo que hace posible el uso de relés en el control.

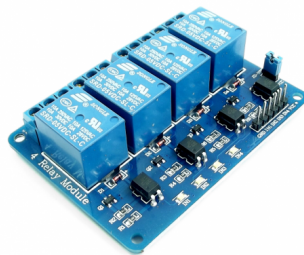


Figura 4.32: Módulo relé de 4 canales. *Imagen tomada de [11].*

9. Convertidor de PWM a voltaje

Este módulo permite convertir la señal PWM proveniente del microcontrolador a voltaje analógico 0 – 10 V, ideal para el control de la electroválvula Winner WVA4-3. Su voltaje de operación va desde 12 V hasta 30 V. El rango de frecuencia de la señal PWM es de 1 kHz – 3 kHz. Mientras que, el voltaje de la señal PWM varía de 5 V a 24 V al colocar el

puede en sus respectivos voltajes. Como se observa, la Figura 4.33 permite el ajuste del voltaje de salida mediante un potenciómetro.



Figura 4.33: Convertidor de PWM a voltaje analógico 0-10V. *Imagen tomada de [12].*

4.4.4. Descripción del hardware

El sistema multitanque presentado en la Figura 4.34 cuenta con tres tanques de color gris. El tanque ubicado en la parte inferior es el tanque de bombeo con una capacidad de 19.162 L. Además, los tanques de reserva ubicados a 61.55 cm y 19.9 cm tienen una capacidad de 12 L cada uno. Sin mencionar que, los tanques poseen una abertura de 0.95 cm que simula las fallas del sistema ante la presencia de fugas. La bomba de color naranja está ubicada por debajo del tanque 1 y en la parte posterior del tanque 2 a una altura a 19.9 cm, la bomba se encuentra fijada a la estructura por medio de 4 tornillos de 1/8". De hecho, el sistema cuenta con tres electroválvulas señaladas de color azul, la primera electroválvula está ubicada a 43 cm tomando como referencia la base del tanque 1, la válvula solenoide está ubicada a 41 cm de la base del tanque 2 y la tercera válvula se encuentra al nivel de la base del tanque de reserva 2. Los sensores son señalados con color morado. Para el caso de los sensores de nivel, son colocados a 33 cm de la base de los tanques de reserva, y para los sensores de flujo, se colocan al mismo nivel de la válvula proporcional en la entrada y la válvula de bola en la salida. El sistema se presenta con más detalle en el Apéndice D.2.3.

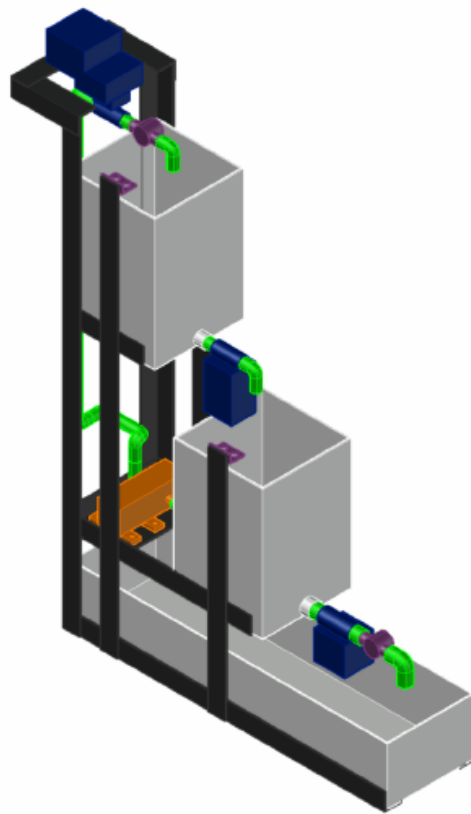


Figura 4.34: Diseño del sistema multitank. *Imagen tomada de [5].*

El tablero de control presentado en la Figura 4.35, es una caja metálica de 50 cm de largo, 40 cm de alto y 15 cm de ancho. En su interior, contiene un doble fondo de 2.5 cm para ocultar los cables. En la parte frontal presenta 24 agujeros para las borneras, 10 de ellos ubicados en la parte izquierda son utilizados para conectar las electroválvulas y la bomba, y los 14 restantes ubicados en la parte inferior derecha para conectar los sensores de nivel y de flujo (véase Figura 4.36). En la parte lateral derecha se encuentran 3 agujeros, el agujero ubicado en la parte central es destinado para colocar el interruptor principal de alimentación del tablero de control. En la parte inferior del mismo lado se ubicará los agujeros para la alimentación de 110 V; mientras que, al otro costado del tablero, se encuentran cuatro interruptores separados 8 cm entre sí. Estos últimos controlan la alimentación de las 3 válvulas y la bomba. Finalmente, en el interior del tablero de control, se ubican los módulos que ajustan el voltaje en la parte superior derecha. La placa STM32-núcleo 144 se ubica en el centro y las fuentes de alimentación de 12 V y 24 V en la parte inferior. El tablero de control se presenta con más detalle en el Apéndice D.2.3.

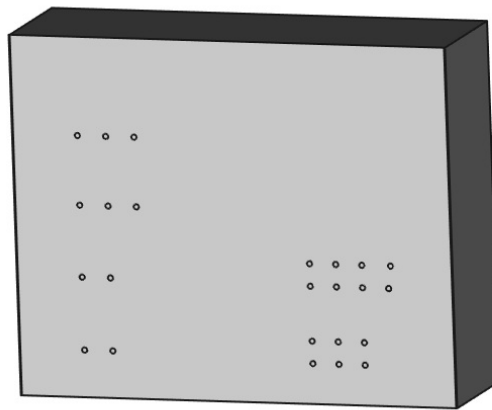


Figura 4.35: Diseño del tablero de control

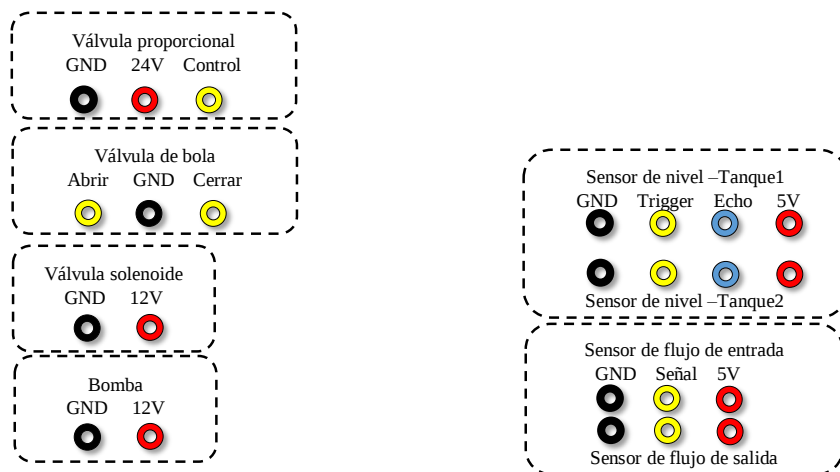


Figura 4.36: Distribución de los terminales de conexión en la parte frontal del tablero de control

Resultados

Este capítulo presenta los resultados del funcionamiento de los algoritmos de inteligencia artificial implementados en el proceso de control de un sistema multitanque. Además, se detallan los resultados obtenidos tanto en la simulación en MATLAB, como en el sistema real. Se realiza la comparación de los algoritmos analizando la respuesta de la planta ante cambios de referencia y usando como índice de desempeño el tiempo de estabilización y porcentaje de sobreimpulso. Además, se analiza el funcionamiento de los algoritmos cuando el sistema presenta fallas provocadas por fugas en los tanques.

5.1. Resultados en la simulación

Se realiza la simulación definiendo un tiempo de muestreo de un segundo. Para ello, se considera el sistema descrito en la Figura 4.9, cuyos parámetros de configuración se presentan en la Tabla 4.2. Las pruebas en la simulación se llevan a cabo para cambios de referencia de 32 % - 8 % y de 8 % - 32 % del nivel de los tanques. Estos cambios se dan en un tiempo de 500 segundos, las gráficas resultantes muestran la respuesta del sistema (azul) y su referencia (rojo). Además, se indica el rango de error del 5 % para el tiempo de establecimiento. En la Figura 5.1, se muestra la respuesta del sistema para cambios de referencia empleando control PID. En la parte superior, se aprecia la respuesta del sistema del tanque 1 y su correcto seguimiento de la señal de referencia. De la misma forma, la respuesta del tanque 2, en la parte inferior, muestra un buen comportamiento en el seguimiento de referencia, encontrándose dentro del umbral de establecimiento en un menor tiempo que el tanque 1.

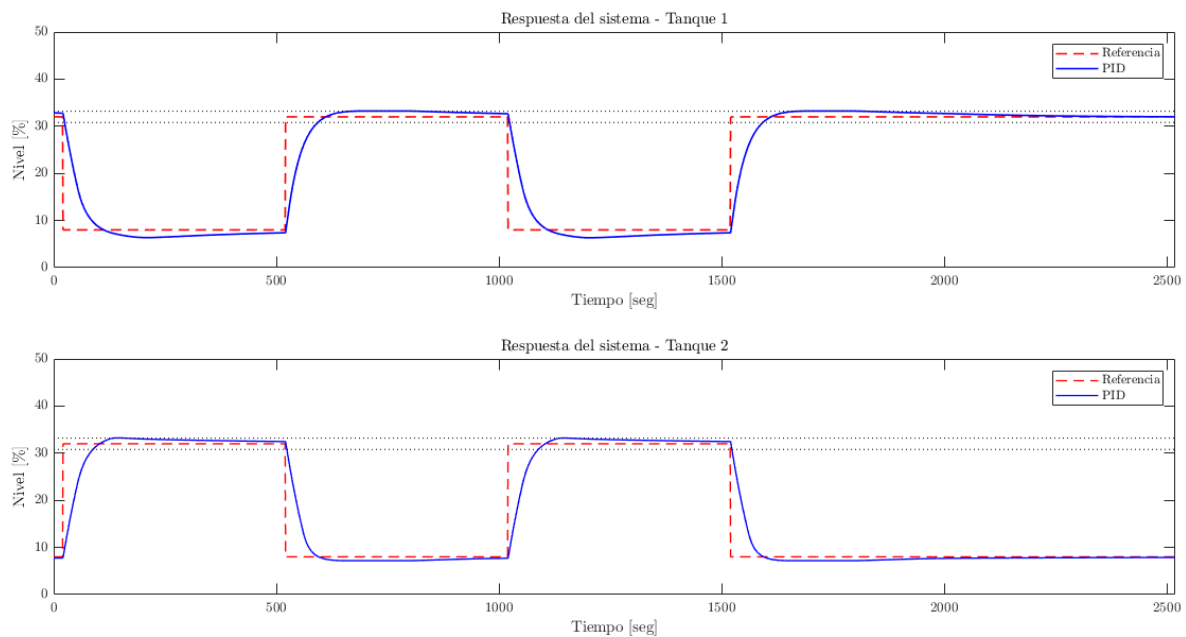


Figura 5.1: Respuesta del sistema en la simulación ante cambios de referencia utilizando el algoritmo de control PID.

La Tabla 5.1 muestra los resultados obtenidos empleando el algoritmo de control PID. Las pruebas 1 y 2 corresponden al análisis de cambios de referencia para un valor bajo a un valor alto, es decir, del 8 % - 32 %. Además, se utiliza como parámetro de evaluación el porcentaje de sobreimpulso y el tiempo de establecimiento en segundos. Los resultados de cada prueba son promediados, para el caso del tanque 1, se tiene un tiempo promedio de establecimiento de 274.67 seg y un sobreimpulso del 5.01 %. En el caso del tanque 2, el tiempo de establecimiento es menor que en el tanque 1 con un valor promedio de 132.51 seg y un sobreimpulso del 5.08 %.

Tabla 5.1: Resultados ante cambios de referencias utilizando control PID.				
		Prueba 1	Prueba 2	Promedio
Tanque1	Tiempo de estabilización [seg]	277.10	272.24	274.67
	Sobreimpulso [%]	5.01	5.00	5.01
Tanque2	Tiempo de estabilización [seg]	135.29	129.74	132.51
	Sobreimpulso [%]	5.14	5.02	5.08

Utilizando la misma configuración, los valores de los cambios de referencia e intervalo de tiempo, se analiza el algoritmo de control FGS-PID. La respuesta del sistema en la simulación para cambios de referencia se muestra en la Figura 5.2. Donde se aprecia un correcto seguimiento de la señal de referencia en el nivel de los dos tanques, además una reducción del tiempo de estabilización si se compara con la respuesta obtenida con el control PID.

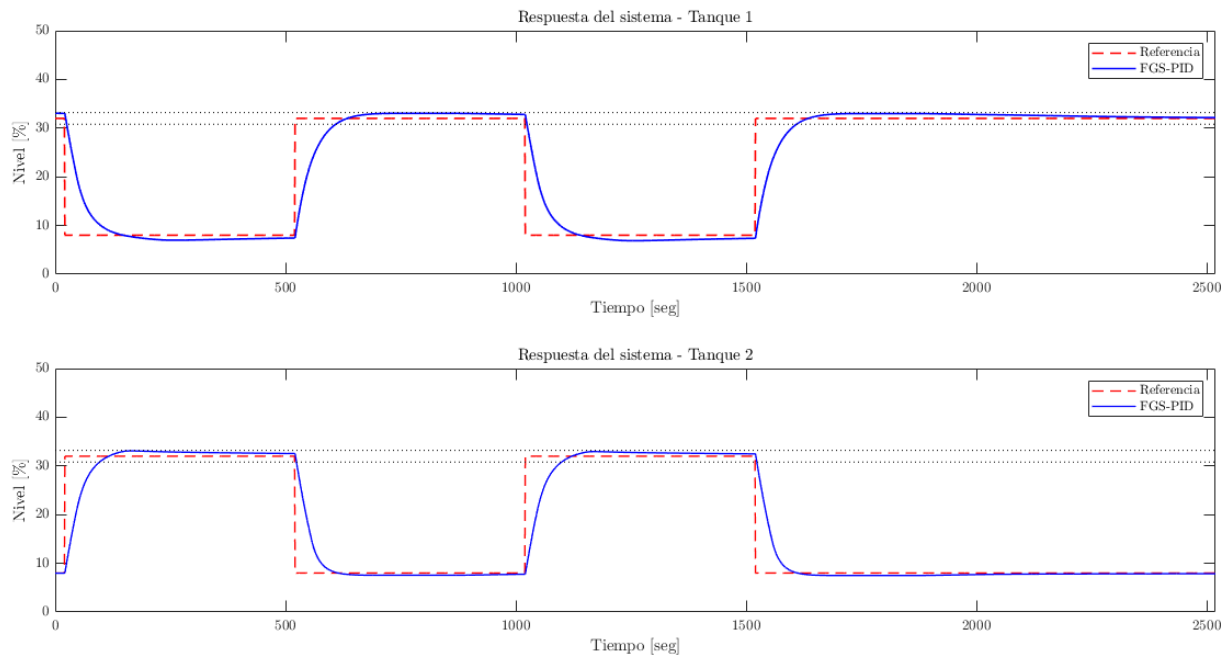


Figura 5.2: Respuesta del sistema en la simulación ante cambios de referencia utilizando el algoritmo de control FGS-PID.

Los resultados presentados en la Tabla 5.2 muestran una disminución del tiempo de establecimiento y sobreimpulso. Para el tanque 1, el tiempo de establecimiento promedio es de 88.35 seg y el sobreimpulso de 4.1 %. Mientras que, para el tanque 2, el tiempo de establecimiento de 77.01 seg y sobreimpulso de 4.21 %. Al comparar con el algoritmo PID, se tienen una disminución de 186.32 seg y 55.50 seg en el tiempo de establecimiento de los tanques 1 y 2. De igual manera, el sobreimpulso se reduce en un 0.91 % y 0.87 %.

Tabla 5.2: Resultados ante cambios de referencias utilizando control FGS-PID.

		Prueba 1	Prueba 2	Promedio
Tanque1	Tiempo de estabilización [seg]	87.95	88.75	88.35
	Sobreimpulso [%]	4.19	4.01	4.10
Tanque2	Tiempo de estabilización [seg]	75.33	78.70	77.01
	Sobreimpulso [%]	4.53	3.90	4.21

La respuesta del sistema para cambios de referencia del 8 % - 32 % usando el algoritmo **SA-PID**, se muestra en la Figura 5.3. Los resultados obtenidos se presentan en la Tabla 5.3. En el tanque 1, el tiempo de establecimiento promedio es de 82.80 seg y el sobreimpulso de 1.83 %. Esto indica una reducción de 191.87 seg y 3.17 % con respecto al algoritmo **PID**. En el tanque 2, el tiempo de establecimiento promedio es de 82.80 seg y el sobreimpulso de 3.63 %, reduciendo en 50.37 seg y 1.45 % del tiempo de establecimiento y sobreimpulso determinado

con el algoritmo **PID**.

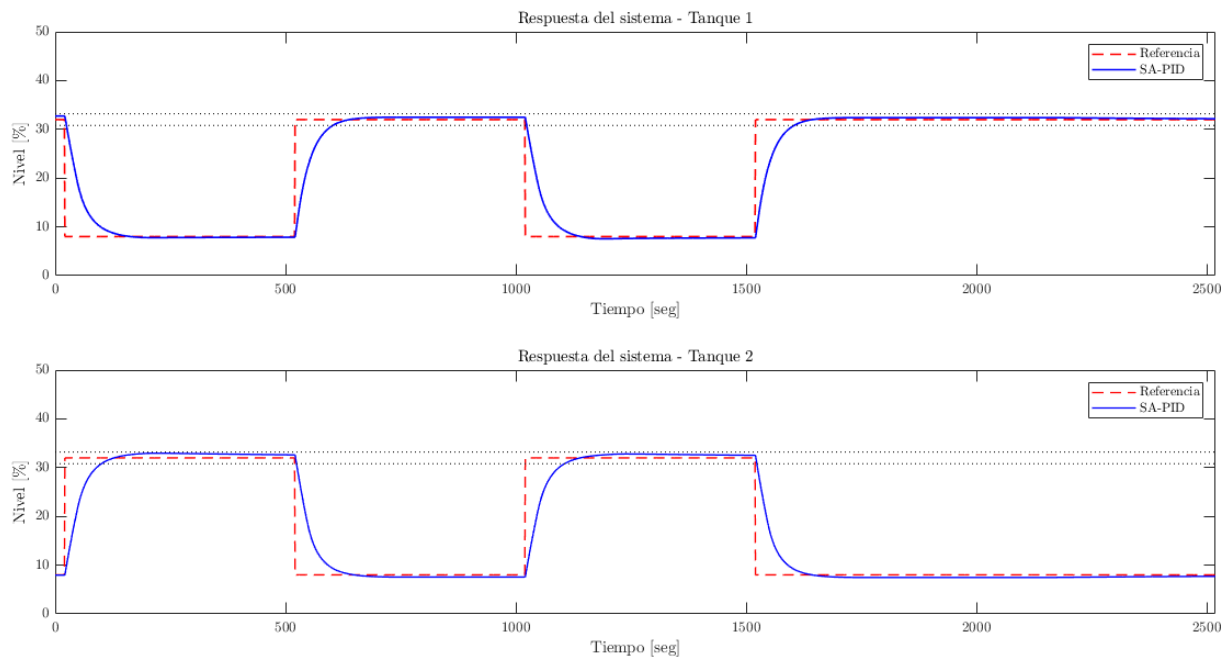


Figura 5.3: Respuesta del sistema en la simulación ante cambios de referencia utilizando el algoritmo de control SA-PID.

Tabla 5.3: Resultados ante cambios de referencias utilizando control SA-PID.

		Prueba 1	Prueba 2	Promedio
Tanque1	Tiempo de estabilización [seg]	82.01	83.59	82.80
	Sobreimpulso [%]	2.00	1.66	1.83
Tanque2	Tiempo de estabilización [seg]	79.47	84.83	82.15
	Sobreimpulso [%]	4.00	3.27	3.63

Por último, con la misma configuración de los otros algoritmos, se evalúa de la respuesta del sistema empleando el algoritmo de control **A*-PID**, obteniendo la respuesta del sistema que se muestra en la Figura 5.4. Esta respuesta muestra el correcto seguimiento de referencia junto a una reducción del sobreimpulso respecto a la respuesta obtenida con el control **PID**.

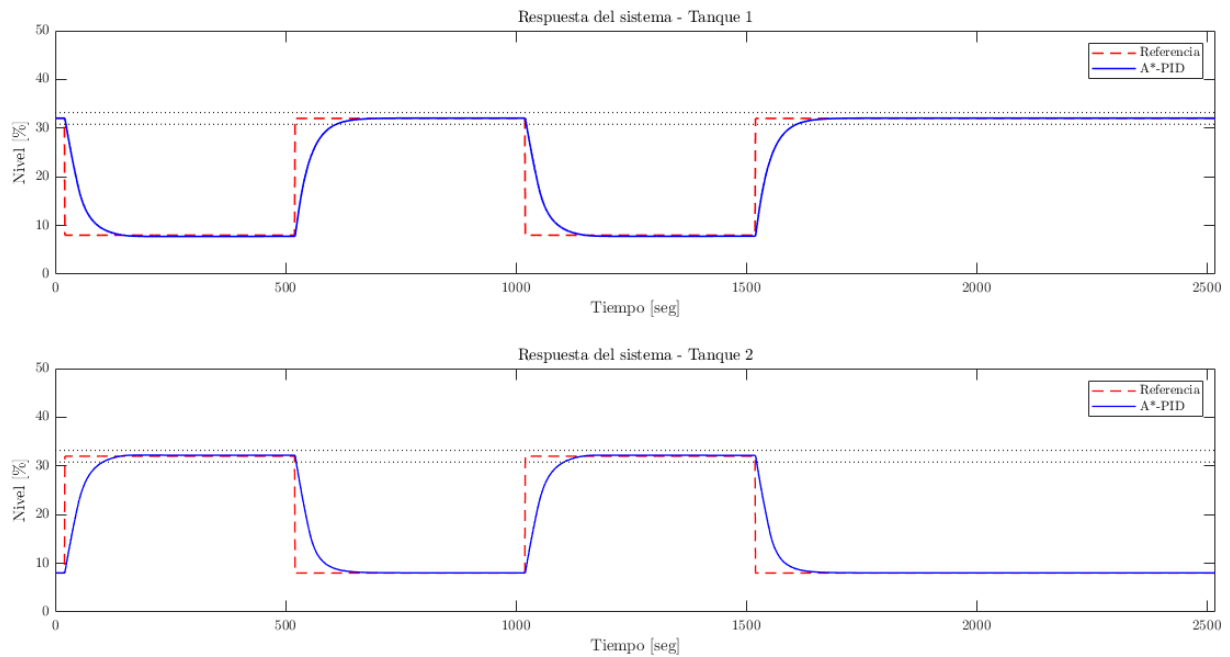


Figura 5.4: Respuesta del sistema en la simulación ante cambios de referencia utilizando el algoritmo de control A*-PID.

Los resultados de la simulación al emplear el algoritmo de control A*-PID, se muestran en la Tabla 5.4, donde el tiempo de establecimiento se reduce en 89.48 seg y el sobreimpulso en 0.22 % para el tanque 1 y en 83.35 seg y 0.86 % para el tanque 2.

Tabla 5.4: Resultados ante cambios de referencias utilizando control A*-PID.

		Prueba 1	Prueba 2	Promedio
Tanque1	Tiempo de estabilización [seg]	89.50	89.45	89.48
	Sobreimpulso [%]	0.22	0.23	0.22
Tanque2	Tiempo de estabilización [seg]	82.82	83.88	83.35
	Sobreimpulso [%]	0.90	0.82	0.86

Continuando con las pruebas en la simulación, se cambia el indicador de la presencia de fuga del tanque 1 a un valor de uno. Esto genera una nueva respuesta del sistema. La Figura 5.5 muestra la respuesta del sistema ante presencia de fuga del tanque 1 para el algoritmo de control PID, donde se realizan cambios de referencia en el tanque 1 del 8 % - 32 % y viceversa. El nivel del tanque 2 se mantiene en 0 en todo momento; por ello, el análisis se efectúa solamente sobre el tanque 1.

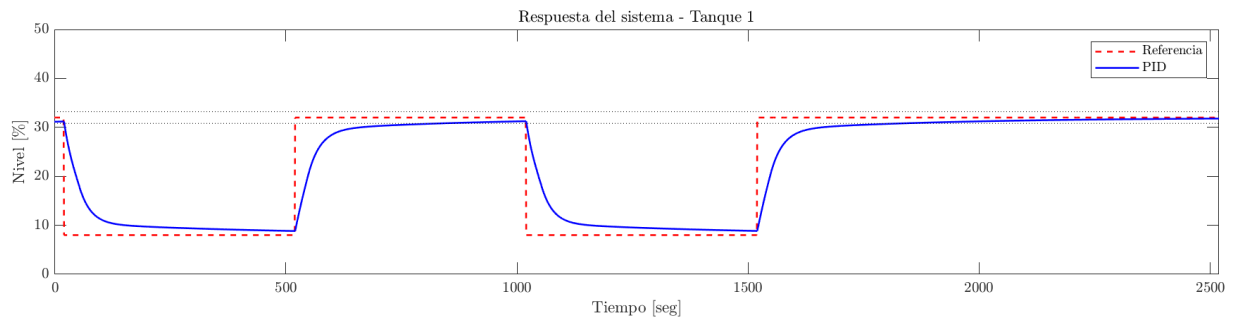


Figura 5.5: Respuesta del sistema en la simulación ante presencia de fugas en el tanque 1 utilizando un controlador PID.

En la Tabla 5.5, se muestra el valor promedio de los parámetros de evaluación, donde el tiempo de establecimiento se incrementa respecto al sistema sin fallas, a un valor de 324.04 seg. El sistema con fuga posee un flujo de salida constante, por lo que los parámetros de control PID preestablecidos inhiben la existencia de sobreimpulso.

Tabla 5.5: Resultados en la simulación ante presencia de fugas en el tanque 1 utilizando un controlador PID.

	Prueba 1	Prueba 2	Promedio
Tiempo de estabilización [seg]	325.21	322.87	324.04
Sobreimpulso [%]	0.00	0.00	0.00

Utilizando el algoritmo de control FGS-PID, se mejora la respuesta del sistema, Figura 5.6. Los resultados presentados en la Tabla 5.6, muestran una disminución del tiempo de establecimiento en 115.05 seg, sin generar sobreimpulso.

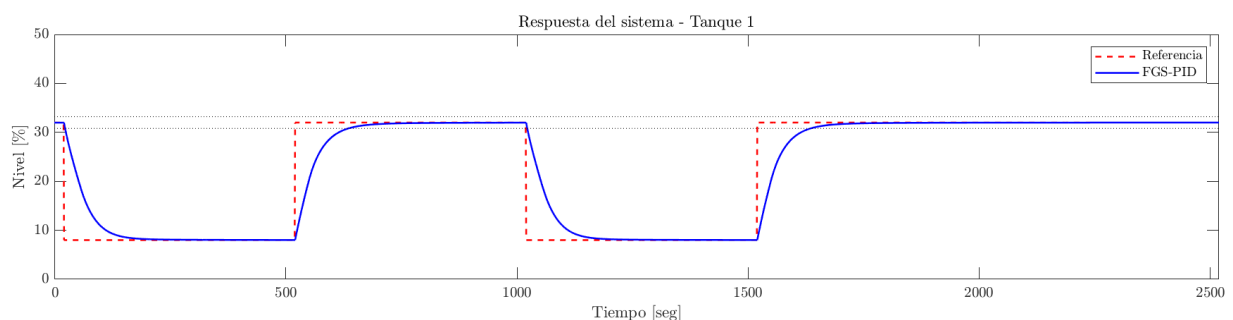


Figura 5.6: Respuesta del sistema en la simulación ante presencia de fugas en el tanque 1 utilizando un controlador FGS-PID.

Tabla 5.6: Resultados en la simulación ante presencia de fugas en el tanque 1 utilizando un controlador FGS-PID.

	Prueba 1	Prueba 2	Promedio
Tiempo de estabilización [seg]	115.03	115.07	115.05
Sobreimpulso [%]	0.00	0.00	0.00

En el caso del algoritmo de control **SA-PID**, la respuesta del sistema ante presencia de fugas en el tanque 1, se muestra en la Figura 5.7 y su análisis en la Tabla 5.7. Los resultados muestran una reducción del tiempo de establecimiento a 130.29 seg, lo cual es mucho menor al obtenido con el algoritmo **PID** y ligeramente mayor que el algoritmo **FGS-PID**. De manera similar que con los otros algoritmos, no se produce sobreimpulso.

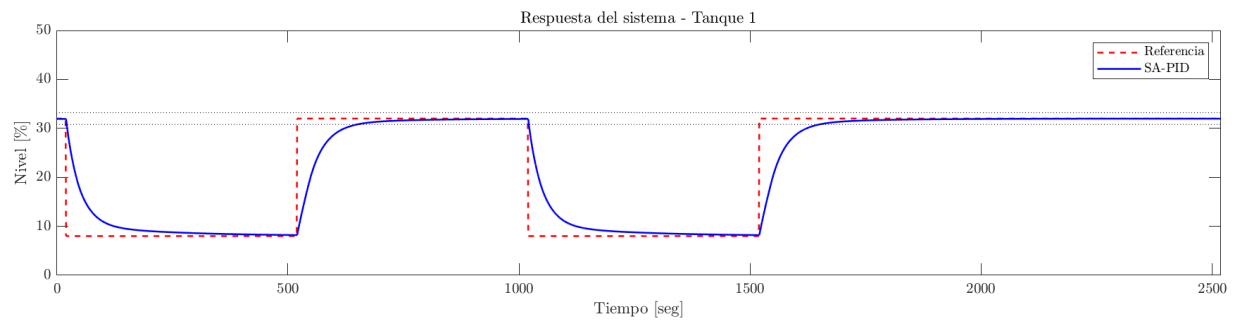


Figura 5.7: Respuesta del sistema en la simulación ante presencia de fugas en el tanque 1 utilizando un controlador SA-PID.

Tabla 5.7: Resultados en la simulación ante presencia de fugas en el tanque 1 utilizando un controlador SA-PID.

	Prueba 1	Prueba 2	Promedio
Tiempo de estabilización [seg]	130.24	130.34	130.29
Sobreimpulso [%]	0.00	0.00	0.00

Para el último caso, usando el algoritmo de control **A*-PID**, la respuesta del sistema ante presencia de fugas en el tanque 1 se muestra en la Figura 5.8. Los resultados presentados en la Tabla 5.8 demuestran que, este algoritmo consigue un menor tiempo de establecimiento que el resto, con un valor de 93.70 seg. Sin embargo, a consecuencia de un ligero incremento del sobreimpulso, en un porcentaje del 0.48 %.

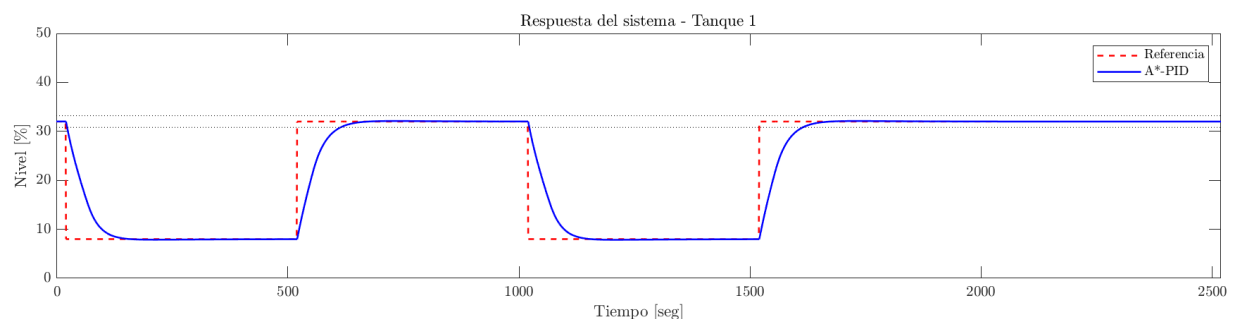


Figura 5.8: Respuesta del sistema en la simulación ante presencia de fugas en el tanque 1 utilizando un controlador A*-PID.

Tabla 5.8: Resultados en la simulación ante presencia de fugas en el tanque 1 utilizando un controlador A*-PID.

	Prueba 1	Prueba 2	Promedio
Tiempo de estabilización [seg]	93.68	93.72	93.70
Sobreimpulso [%]	0.48	0.48	0.48

En la simulación, se realiza pruebas para verificar el comportamiento de los algoritmos ante presencia de fugas del tanque 2. Por tanto, se activa el indicador de presencia de fugas sobre este tanque, se coloca el tanque 1 a un nivel de referencia del 60 % durante toda la simulación y se varía el nivel referencia del tanque 2 de 8 % - 32 % y viceversa, en intervalos de 500 seg a excepción del último cambio de referencia que se mantiene desde los 1500 seg a 2500 seg. En la Figura 5.9, se muestra la respuesta del sistema en el tanque 2 cuando se utiliza un controlador PID. Esta presenta un mayor tiempo de establecimiento que el sistema sin fallas, incluso el intervalo de tiempo de 500 seg no es suficiente para que la respuesta esté dentro del umbral de establecimiento.

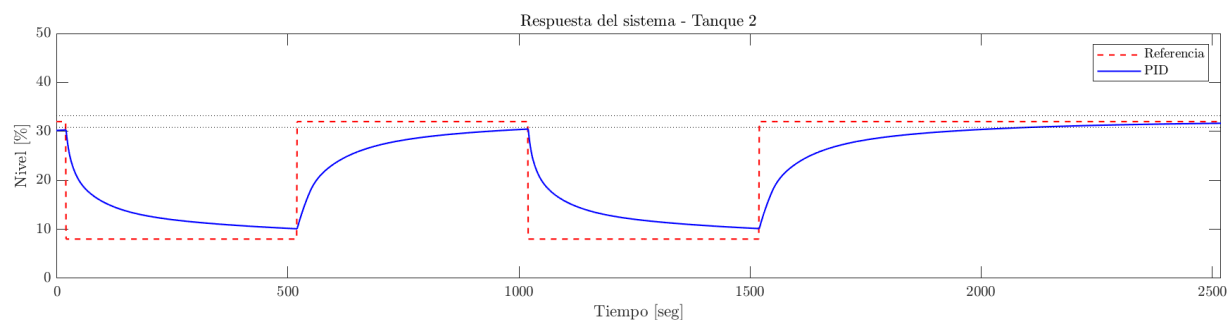


Figura 5.9: Respuesta del sistema en la simulación ante presencia de fugas en el tanque 2 utilizando un controlador PID.

Los resultados muestran que el sistema resulta afectado por la presencia de fugas en el tanque 2. El efecto es mayor que en las pruebas en el tanque 1, dando como resultado un tiempo de establecimiento de 610 seg, Tabla 5.9. La respuesta del sistema en la primera prueba no alcanza el umbral de establecimiento en el intervalo de tiempo dado, es por eso que se toma únicamente de la segunda prueba, la respuesta no posee sobreimpulso.

Tabla 5.9: Resultados en la simulación ante presencia de fugas en el tanque 2 utilizando un controlador PID.

	Prueba 1	Prueba 2	Promedio
Tiempo de estabilización [seg]	-	610.00	610.00
Sobreimpulso [%]	0.00	0.00	0.00

Al repetir la misma prueba empleando el controlador FGS-PID, Figura 5.10, se obtiene una

clara mejoría. Con esto se obtiene un tiempo de estabilización de 256.56 seg y sobreimpulso de 1.77 %, estos resultados se presentan en la Tabla 5.10.

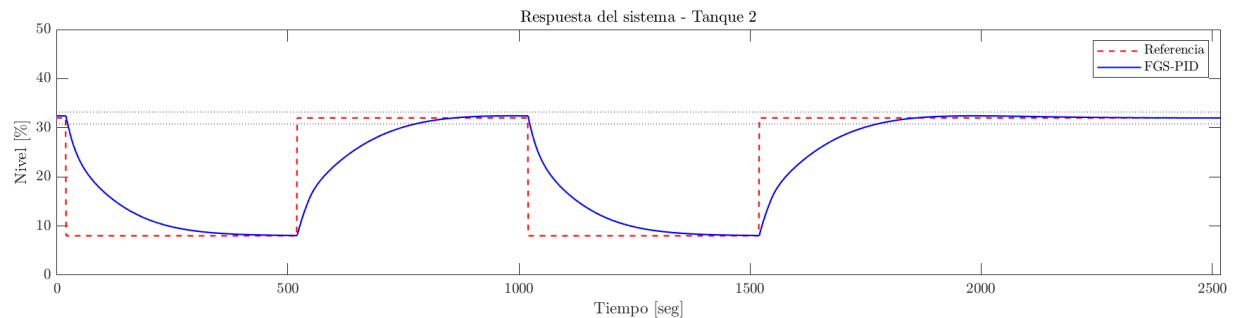


Figura 5.10: Respuesta del sistema en la simulación ante presencia de fugas en el tanque 2 utilizando un controlador FGS-PID.

Tabla 5.10: Resultados en la simulación ante presencia de fugas en el tanque 2 utilizando un controlador FGS-PID.

	Prueba 1	Prueba 2	Promedio
Tiempo de estabilización [seg]	256.56	256.56	256.56
Sobreimpulso [%]	1.77	1.77	1.77

En la Figura 5.11, se observa la respuesta del sistema empleando el algoritmo SA-PID ante presencia de fugas en el tanque 2, y su análisis de resultados en la Tabla 5.11. La respuesta no presenta sobreimpulso y tiene tiempo de establecimiento de 358.35 seg, que es menor al obtenido con el controlador PID, pero superior al valor de controlador FGS-PID.

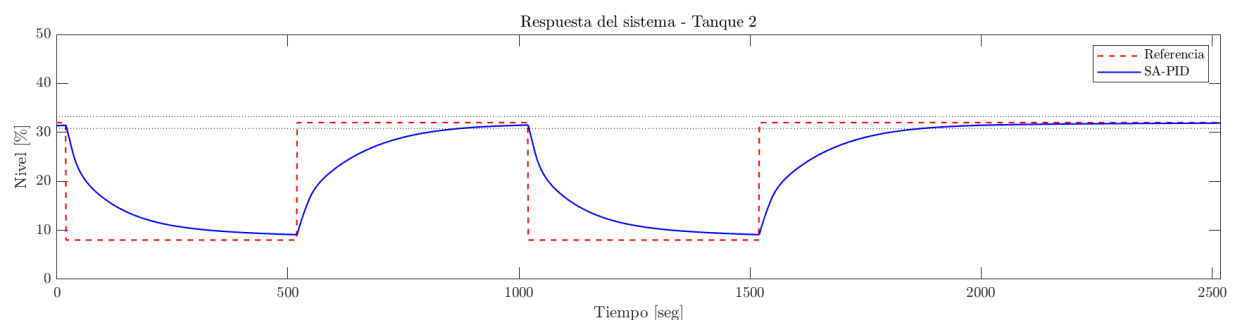


Figura 5.11: Respuesta del sistema en la simulación ante presencia de fugas en el tanque 2 utilizando un controlador SA-PID.

Tabla 5.11: Resultados en la simulación ante presencia de fugas en el tanque 2 utilizando un controlador SA-PID.

	Prueba 1	Prueba 2	Promedio
Tiempo de estabilización [seg]	358.62	358.09	358.35
Sobreimpulso [%]	0.00	0.00	0.00

La última prueba en la simulación es para el controlador A*-PID. La respuesta ante presencia

de fugas en el tanque 2, se muestra en la Figura 5.12. Mientras que, los resultados se muestran en la Tabla 5.12, donde el tiempo de estabilización del sistema empleando este algoritmo es de 323.68 seg sin la presencia de sobreimpulso en la respuesta. El algoritmo consigue llegar a la referencia en un tiempo menor que los algoritmos PID y SA-PID.

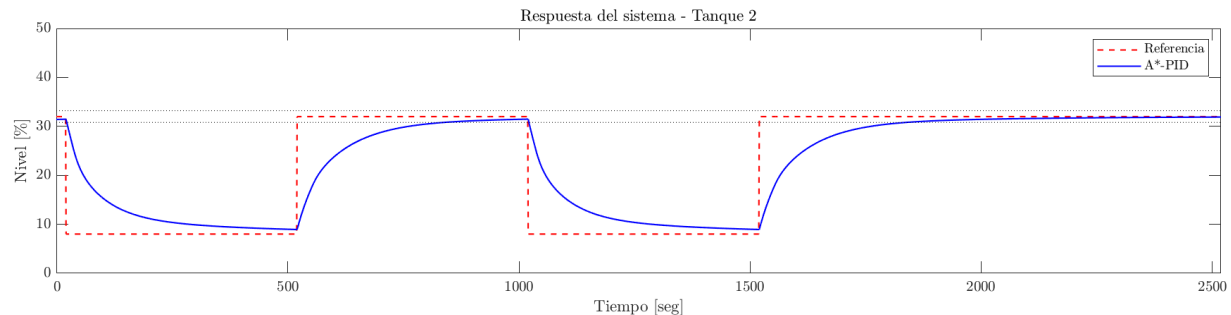


Figura 5.12: Respuesta del sistema en la simulación ante presencia de fugas en el tanque 2 utilizando un controlador A*-PID.

Tabla 5.12: Resultados en la simulación ante presencia de fugas en el tanque 2 utilizando un controlador A*-PID.

	Prueba 1	Prueba 2	Promedio
Tiempo de estabilización [seg]	323.76	323.59	323.68
Sobreimpulso [%]	0.00	0.00	0.00

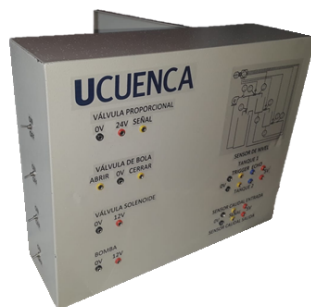
5.2. Resultados del sistema multitanque

5.2.1. Funcionamiento del sistema multitanque y sistema de control

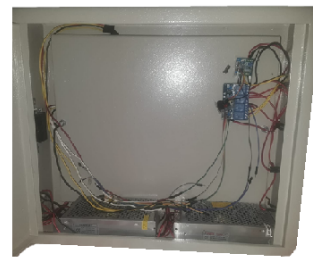
El sistema multitanque construido, se muestra en la Figura 5.13, junto a todos los elementos que lo conforman como sensores, electroválvulas, bomba de agua y tubería. El tablero de control construido se muestra en la Figura 5.14, donde se presenta su vista frontal y su interior. El componente principal del tablero de control es la placa STM32F413ZH, ya que se encarga de tomar las medidas de los sensores, generar las señales de control y enviar los datos al ordenador para su visualización. Los componentes secundarios pero fundamentales son: módulo relé de cuatro canales, módulo convertidor de señal PWM a voltaje analógico, las fuentes de alimentación de 12V/10A y 24V/5A e interruptores para apagar las electroválvulas y la bomba, de forma manual.



Figura 5.13: Sistema multitanque construido.



(a) Vista frontal



(b) Vista interior

Figura 5.14: Tablero de control construido.

Las mediciones del nivel de agua en los tanques presentan ciertas variaciones producto de las ondulaciones y salpicaduras. Para disminuir en gran medida estos errores, se utilizan tres estrategias. Como se observa en la Figura 5.15, se coloca una pared limitadora al costado del tanque cerca del sensor. La pared antes mencionada permite el paso del agua por la parte inferior, impide las salpicaduras y reduce las ondulaciones. Como segunda estrategia, se coloca un flotador negro de plástico de 19×3 cm. Mientras que, la tercera estrategia consiste en realizar múltiples mediciones durante el intervalo de tiempo de muestreo y obtener su promedio.



Figura 5.15: Estrategias para reducir el error en la medición.

Para las fugas se agrega un adaptador macho para manguera presentado en la Figura 5.16; por lo que, el área de la fuga se reduce con diámetro de 0.635 cm.



Figura 5.16: Sistema de fugas del sistema multitanque construido.

5.2.2. Resultados de las pruebas de funcionamiento sin fugas

Las pruebas de funcionamiento del sistema multitanque, se realizan para cambios de referencia del nivel de los tanques del 8 % al 32 % y de 32 % al 8 %. Y Así, se evalúa solo para cambios de referencia de un nivel bajo a un alto, dando como resultado 3 pruebas que son promediadas. La respuesta del sistema presenta ruido en la medición, lo cual afecta al proceso de análisis de la respuesta y dificulta la determinación del tiempo de estabilización y sobreimpulso. Para esto, se efectúa previamente un filtrado de la señal, utilizando un filtro de mediana de quinto orden, donde el valor de la señal filtrada y_k es la mediana de los valores x_{k-2} hasta x_{k+2} de la señal. Además, es importante mencionar que la respuesta del sistema tarda un tiempo considerable en alcanzar la referencia, por tanto, para poder llevar a cabo las pruebas se reduce el número de iteraciones respecto a la simulación. El número de iteraciones y sub-iteraciones para el control SA-PID es de 3 y el número de iteraciones para el control A*-PID es de 9.

La primera prueba se realiza utilizando el algoritmo de control PID. Como resultado, la Figura 5.17, muestra la respuesta del sistema para cambios de referencia en intervalos de 110 seg. Las señales son analizadas considerando los mismos parámetros de evaluación que en la simulación. Es decir, un rango de error de 5 % para el tiempo de estabilización.

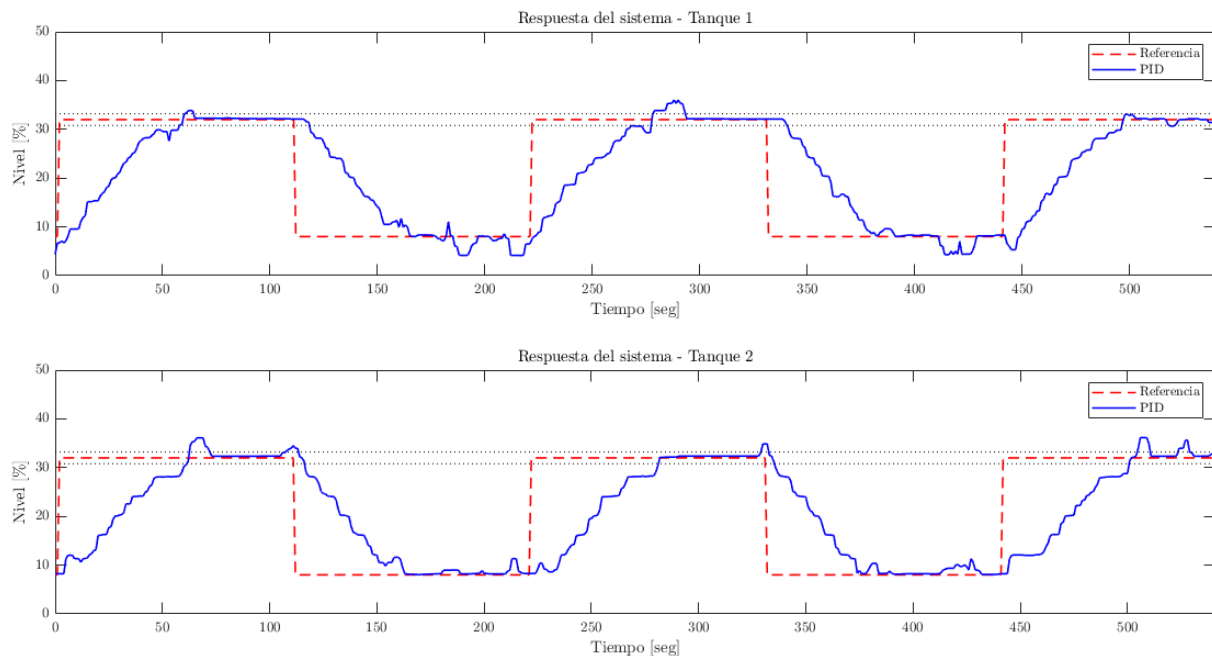


Figura 5.17: Respuesta del sistema multitank ante cambios de referencia utilizando el algoritmo de control PID.

Los resultados son presentados en la Tabla 5.13, donde se determina que el tiempo de estabilización promedio del tanque 1 es de 61.61 seg y su sobreimpulso de 9.59 %. Mientras que, para el tanque 2, el tiempo de estabilización es de 72 seg y un sobreimpulso del 12.12 %.

Tabla 5.13: Resultados del sistema multitank utilizando control PID.

		Prueba 1	Prueba 2	Prueba 3	Promedio
Tanque1	Tiempo de estabilización [seg]	62.38	71.29	51.1659	61.61
	Sobreimpulso [%]	7.28	16.58	4.8986	9.59
Tanque2	Tiempo de estabilización [seg]	70.17	59.24	86.9578	72.12
	Sobreimpulso [%]	17.34	1.52	17.5084	12.12

La Figura 5.18 muestra la respuesta del sistema para cambios de referencia de 32 % - 8 % y viceversa en intervalos de tiempo de 110 seg. La Tabla 5.14 muestra los resultados obtenidos al utilizar el controlador FGS-PID. En el caso del tanque 1, se obtiene un tiempo de establecimiento promedio de 59.77 seg y un sobreimpulso de 9.49 %. En el caso del tanque 2, se obtiene un tiempo de establecimiento promedio de 60.94 seg y un sobreimpulso de 6.72 %. Al

comparar con el algoritmo de control **PID**, se reduce en 4.17 seg y 0.10 % de valores obtenidos en el tanque 1 y 11.18 seg y 5.40 % del tanque 2.

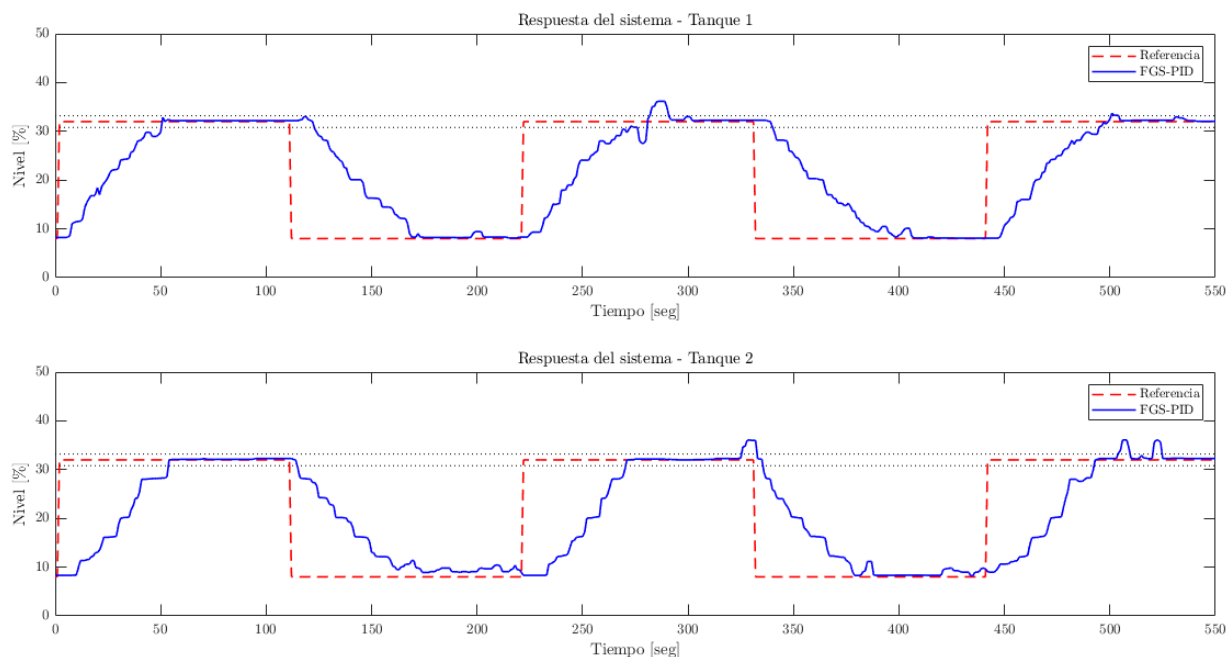


Figura 5.18: Respuesta del sistema multitank ante cambios de referencia utilizando el algoritmo de control FGS-PID.

Tabla 5.14: Resultados del sistema multitank utilizando control FGS-PID.

		Prueba 1	Prueba 2	Prueba 3	Promedio
Tanque1	Tiempo de estabilización [seg]	48.32	68.92	62.0778	59.77
	Sobreimpulso [%]	3.70	17.57	7.1906	9.49
Tanque2	Tiempo de estabilización [seg]	51.66	48.43	82.7373	60.94
	Sobreimpulso [%]	1.18	1.18	17.801	6.72

Al aplicar el algoritmo de control **SA-PID**, se genera la respuesta del sistema mostrada en la Figura 5.19. Con esto se logra una mejoría con respecto al algoritmo **PID**. Los resultados presentados en la Tabla 5.15 muestran que, el tiempo de estabilización promedio del tanque 1 es de 60.89 y su sobreimpulso de 0.98 %. Así, el tiempo de estabilización en la respuesta del tanque 2 es de 51.61 seg y 1.35 %. El algoritmo **SA-PID** reduce considerablemente el sobreimpulso, en comparación con el algoritmo **PID**. De hecho, se reduce en un 8.61 % y 10.77 % para el tanque 1 y 2, respectivamente. El tiempo de establecimiento en el tanque 1 de 3.05 seg es ligeramente menor. Sin embargo, en el tanque 2 existe una diferencia considerable de 20.51 seg.

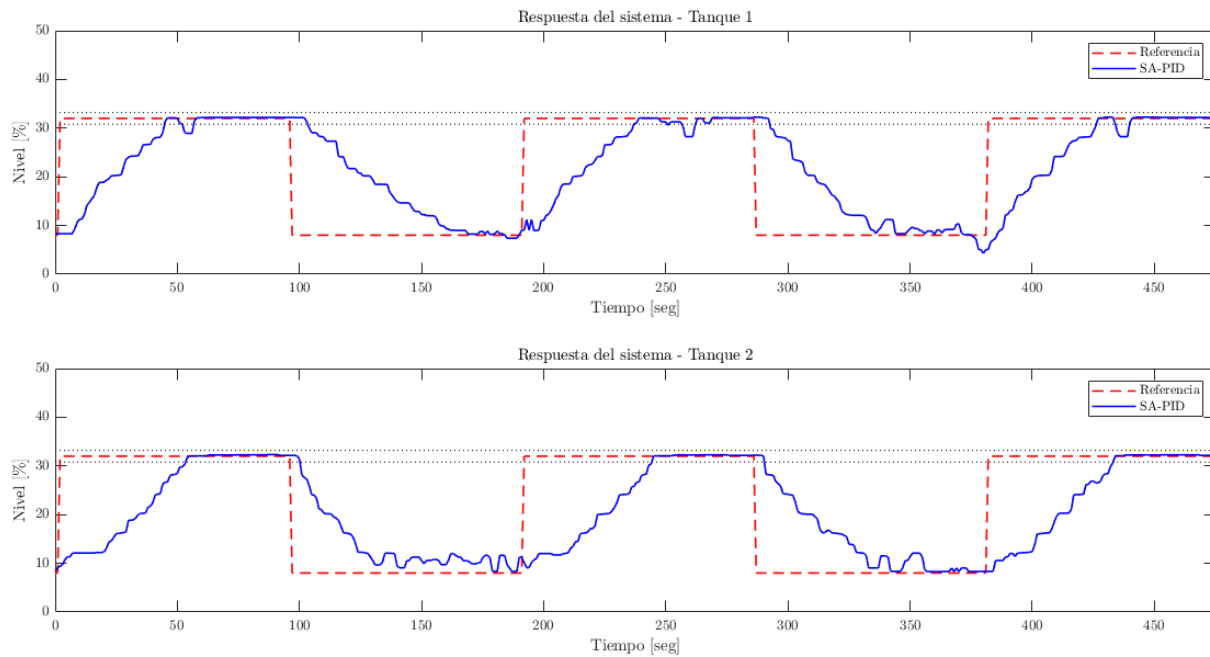


Figura 5.19: Respuesta del sistema multitank ante cambios de referencia utilizando el algoritmo de control SA-PID.

Tabla 5.15: Resultados del sistema multitank utilizando control SA-PID.

		Prueba 1	Prueba 2	Prueba 3	Promedio
Tanque1	Tiempo de estabilización [seg]	54.74	70.13	57.804	60.89
	Sobreimpulso [%]	0.84	1.05	1.0386	0.98
Tanque2	Tiempo de estabilización [seg]	51.42	52.28	51.1473	51.61
	Sobreimpulso [%]	1.42	1.28	1.3536	1.35

La Figura 5.20 muestra la respuesta del sistema ante cambios de referencias utilizando el algoritmo de control A*-PID y presentando el análisis de resultados en la Tabla 5.16. Para el tanque 1, la respuesta del sistema tiene un tiempo de estabilización promedio de 52.32 seg y sobreimpulso de 3.88 %. Mientras que, para el tanque 2, el tiempo de estabilización es de 66.81 seg y sobreimpulso de 10.45 %. Al comparar los resultados obtenidos con los anteriores algoritmos; es notorio que, este algoritmo produce el menor tiempo de estabilización para el tanque 1.

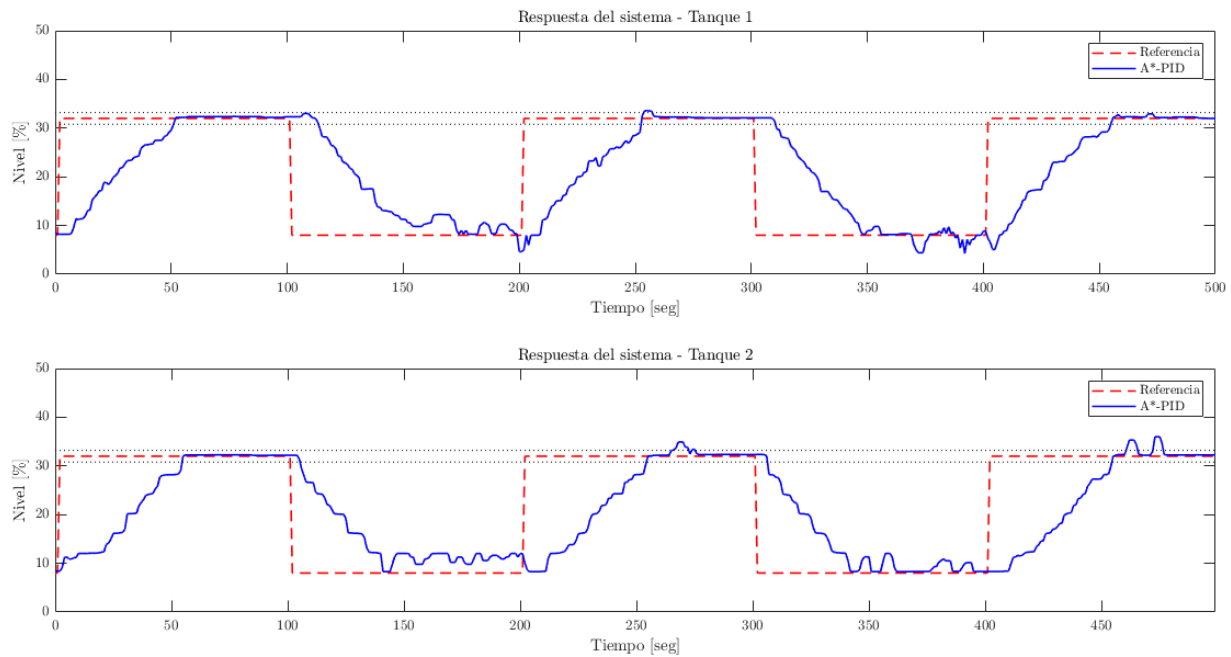


Figura 5.20: Respuesta del sistema multitank ante cambios de referencia utilizando el algoritmo de control A*-PID.

Tabla 5.16: Resultados del sistema multitank utilizando control A*-PID.

		Prueba 1	Prueba 2	Prueba 3	Promedio
Tanque1	Tiempo de estabilización [seg]	49.01	52.79	55.1518	52.32
	Sobreimpulso [%]	1.85	3.72	6.074	3.88
Tanque2	Tiempo de estabilización [seg]	52.59	74.63	73.2229	66.81
	Sobreimpulso [%]	1.19	16.89	13.2727	10.45

5.2.3. Resultados de las pruebas de funcionamiento con fugas

A diferencia con las pruebas anteriores, los cambios de referencia se dan entre el 8 % y 20 % del nivel de los tanques. Esto con la finalidad de reducir el tiempo que tarda en realizar cada prueba ante la presencia de fugas.

En el primer caso, se configura el sistema en falla activando la fuga en el tanque 1. Luego, se realizan los cambios de referencia con los valores antes indicados sobre este tanque; mientras que, la referencia del tanque 2 se mantiene en cero durante las pruebas. La Figura 5.21 muestra la respuesta del sistema ante presencia de fugas en el tanque 1 al emplear un controlador PID. Se observa que, con las condiciones por defecto del controlador PID, el sistema sigue la referencia. No obstante, para el nivel de 8 % no consigue establecerse.

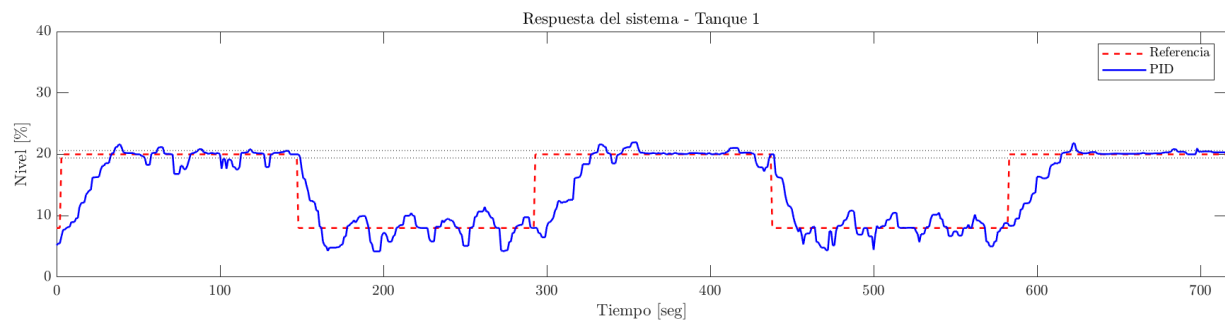


Figura 5.21: Respuesta del sistema ante presencia de fugas en el tanque 1 utilizando el algoritmo de control PID.

El tiempo de establecimiento promedio ante presencia de fugas en el tanque 1 es de 122.6 seg con un sobreimpulso de 14.61 %. Los resultados mencionados se presentan en la Tabla 5.17.

Tabla 5.17: Resultados ante presencia de fugas en el tanque 1 utilizando control PID.

	Prueba 1	Prueba 2	Prueba 3	Promedio
Tiempo de estabilización [seg]	127.68	124.46	115.65	122.60
Sobreimpulso [%]	11.98	16.39	15.46	14.61

Usando las mismas condiciones aplicadas en el controlador PID, se evalúa la respuesta del sistema utilizando el algoritmo FGS-PID. La Figura 5.22 muestra el comportamiento del sistema sobre el tanque 1. En esta ocasión, se observa un mejor seguimiento de referencia para el nivel del 8 %.

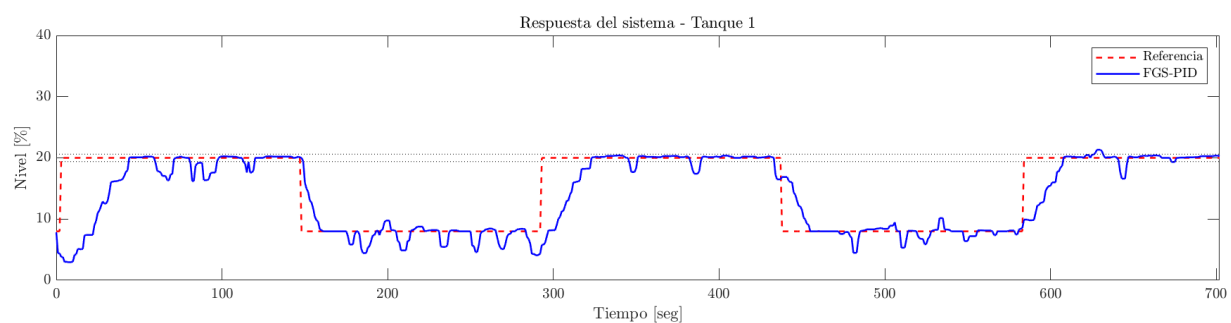


Figura 5.22: Respuesta del sistema ante presencia de fugas en el tanque 1 utilizando el algoritmo de control FGS-PID.

Los resultados del algoritmo FGS-PID, presentados en la Tabla 5.18, muestran una mejoría con respecto al algoritmo PID. En este, el tiempo de establecimiento promedio disminuye a 108.51 seg y su sobreimpulso se reduce a más de la mitad, con un valor de 5.88 %.

Tabla 5.18: Resultados ante presencia de fugas en el tanque 1 utilizando control FGS-PID.

	Prueba 1	Prueba 2	Prueba 3	Promedio
Tiempo de estabilización [seg]	95.85	116.58	113.10	108.51
Sobreimpulso [%]	2.82	1.73	13.10	5.88

La Figura 5.23 muestra la respuesta del sistema ante presencia de fugas en el tanque 1, empleando el algoritmo SA-PID. Los resultados obtenidos al aplicar este algoritmo, se resumen en la Tabla 5.19. Tal como se ha presentado, el tiempo de estabilización promedio obtenido es de 105.99 seg y el sobreimpulso de 9.63 %.

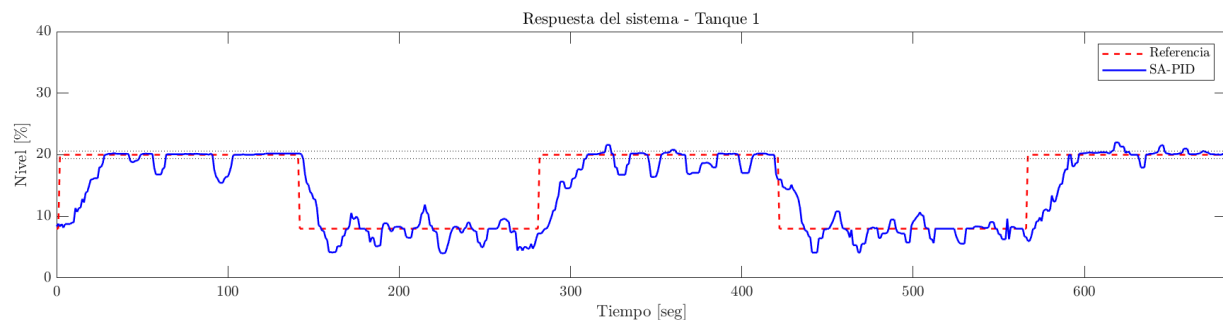


Figura 5.23: Respuesta del sistema ante presencia de fugas en el tanque 1 utilizando el algoritmo de control SA-PID.

Tabla 5.19: Resultados ante presencia de fugas en el tanque 1 utilizando control SA-PID.

	Prueba 1	Prueba 2	Prueba 3	Promedio
Tiempo de estabilización [seg]	100.72	93.33	123.94	105.99
Sobreimpulso [%]	2.06	14.29	12.54	9.63

La última prueba ha sido realizada con la configuración del sistema en falla. El desarrollo de esta prueba consiste en el análisis del algoritmo A*-PID. En la Figura 5.24, se observa la respuesta del sistema en el tanque 1 y la Tabla 5.20 tabula sus resultados. Al comparar los resultados obtenidos; de manera similar al algoritmo de control SA-PID, se observa la mejora del algoritmo A*-PID en el tiempo de estabilización del sistema, mismo que es de 105.98 seg. Además, se identifica una reducción considerable del sobreimpulso a un valor de 6.79 %.

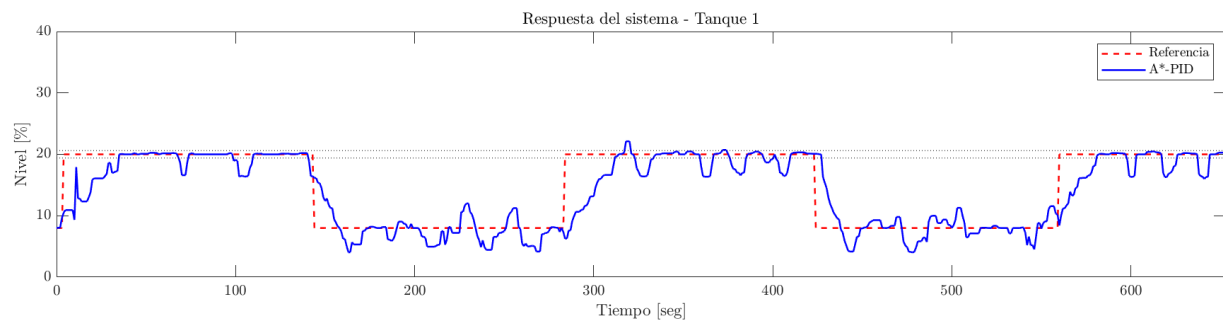


Figura 5.24: Respuesta del sistema ante presencia de fugas en el tanque 1 utilizando el algoritmo de control A*-PID.

Tabla 5.20: Resultados ante presencia de fugas en el tanque 1 utilizando control A*-PID.

	Prueba 1	Prueba 2	Prueba 3	Promedio
Tiempo de estabilización [seg]	105.65	125.57	86.73	105.98
Sobreimpulso [%]	2.5316	15.45	2.40	6.79

En el segundo caso, se configura el sistema en falla. Para ello, se activa la fuga en el tanque 2. Similarmente, se realizan cambios de referencias del 8 % al 20 % del nivel del tanque 2. Mientras tanto, para el tanque 1, se mantiene una referencia constante sobre el 32 %. El análisis de resultados correspondientes a este caso, se efectúa cuando la referencia pasa de un valor alto a un valor bajo, es decir, de 20 % al 8 %. La Figura 5.25 muestra las primeras pruebas para esta configuración utilizando el algoritmo PID. Se examina que, en este caso, los parámetros por defecto del controlador no permiten el seguimiento de la referencia.

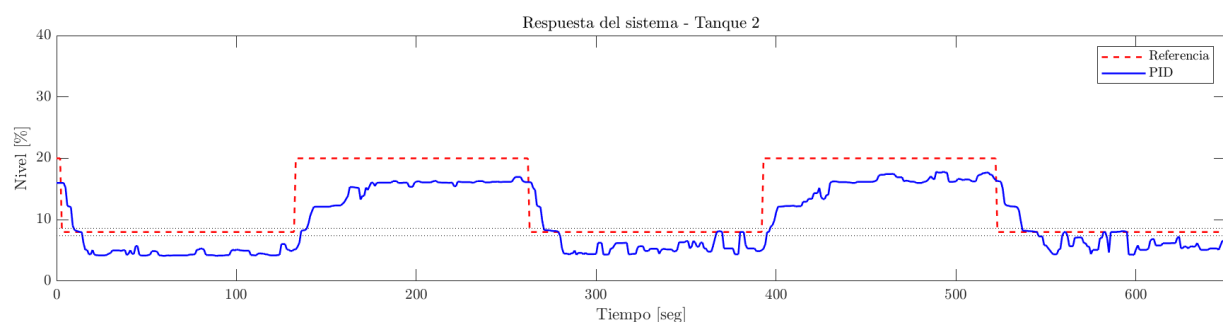


Figura 5.25: Respuesta del sistema ante presencia de fugas en el tanque 2 utilizando el algoritmo de control PID.

Debido a que el sistema no alcanza la referencia, los resultados presentados en la Tabla 5.21 para el algoritmo de control PID, no cuentan con un tiempo de estabilización. Sin embargo, el valor promedio del sobreimpulso es de 46.41 %.

Tabla 5.21: Resultados ante presencia de fugas en el tanque 2 utilizando control PID.

	Prueba 1	Prueba 2	Prueba 3	Promedio
Tiempo de estabilización [seg]	-	-	-	-
Sobreimpulso [%]	48.50	45.81	44.928	46.41

La respuesta del sistema presentado en la Figura 5.26, muestra el seguimiento de referencia empleando el algoritmo FGS-PID. En esta ocasión, el sistema sigue la referencia en el nivel de 2 %, lo cual permite su análisis. El resultado del valor promedio del tiempo de establecimiento, presentado en la Tabla 5.22, es de 110.81 seg y su sobreimpulso del 34 %.

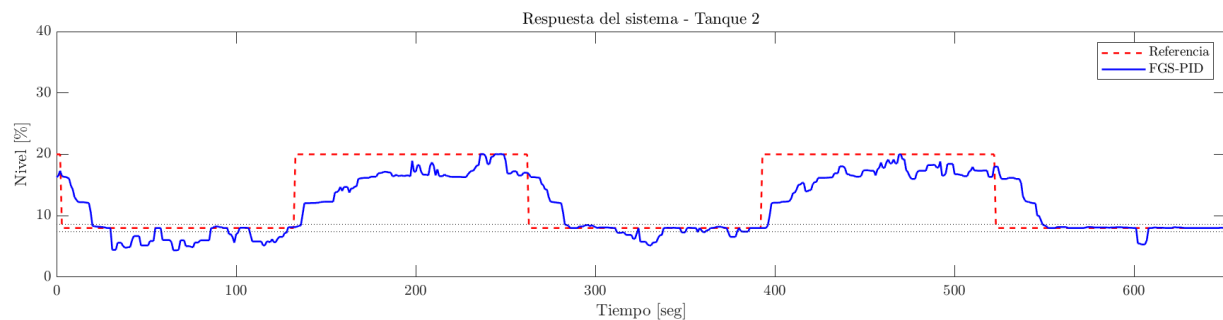


Figura 5.26: Respuesta del sistema ante presencia de fugas en el tanque 2 utilizando el algoritmo de control FGS-PID.

Tabla 5.22: Resultados ante presencia de fugas en el tanque 2 utilizando control FGS-PID.

	Prueba 1	Prueba 2	Prueba 3	Promedio
Tiempo de estabilización [seg]	125.41	122.27	84.74	110.81
Sobreimpulso [%]	42.92	32.27	26.80	34.00

De los resultados presentados, se deduce que, la respuesta del sistema en presencia de fallas mejora considerablemente al usar el algoritmo de control SA-PID, véase Figura 5.27. Así también, los resultados presentados en la Tabla 5.23 muestran que, en el mejor de los casos, el sistema se establece en 24.35 seg, sin presentar sobreimpulso. Este resultado, junto con los obtenidos en las demás pruebas, generan un promedio de tiempo de establecimiento de 80.90 y sobreimpulso de 10.58 %.

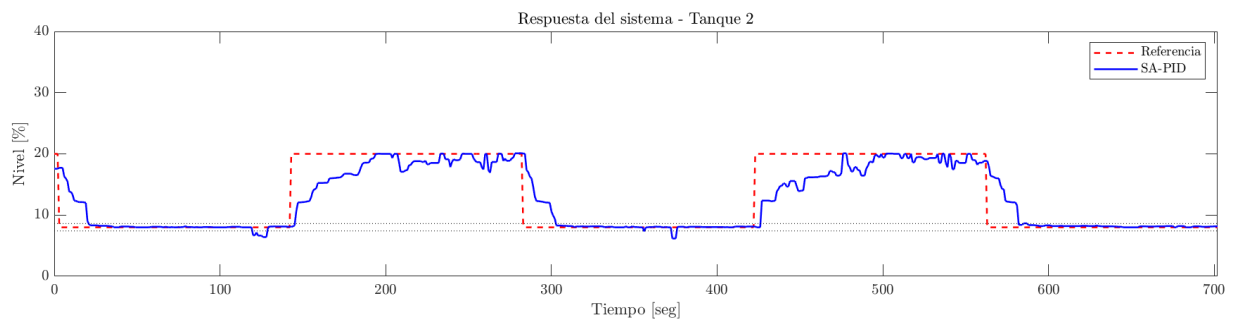


Figura 5.27: Respuesta del sistema ante presencia de fugas en el tanque 2 utilizando el algoritmo de control SA-PID.

Tabla 5.23: Resultados ante presencia de fugas en el tanque 2 utilizando control SA-PID.

	Prueba 1	Prueba 2	Prueba 3	Promedio
Tiempo de estabilización [seg]	125.70	92.66	24.35	80.90
Sobreimpulso [%]	16.46	15.28	0.00	10.58

Por último, se realizan pruebas para el algoritmo **A*-PID**. La respuesta del sistema ante presencia de fugas en el tanque 2, se muestra en la Figura 5.28. Mientras que, el análisis de sus resultados se han tabulado en la Tabla 5.24. El sistema tiene un tiempo de establecimiento promedio de 62.21 seg y sobreimpulso de 11.26 %. Es importante clarificar que, en el mejor de los casos, el sistema se establece en 18.95 seg, y no genera sobreimpulso; comparando con el resto de algoritmos, este controlador presenta el menor tiempo de estabilización del sistema ante fallas. Adicionalmente, en el caso del valor de sobreimpulso, el algoritmo **SA-PID** tiene mejor evaluación.

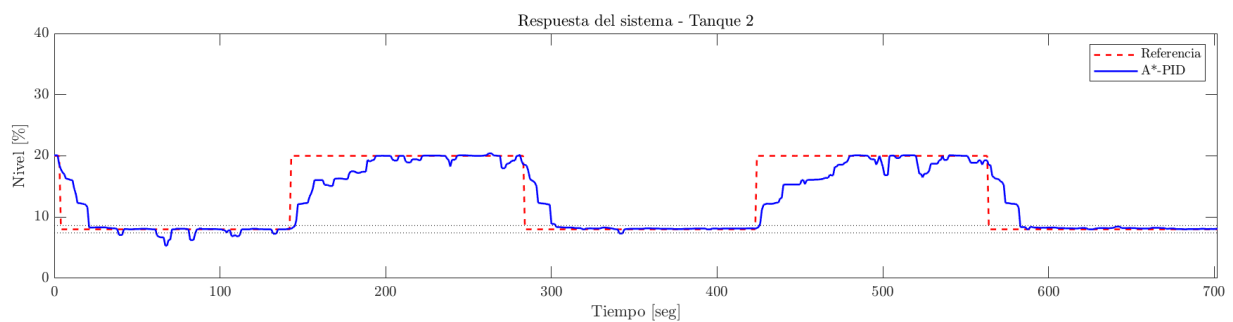


Figura 5.28: Respuesta del sistema ante presencia de fugas en el tanque 2 utilizando el algoritmo de control A*-PID.

Tabla 5.24: Resultados ante presencia de fugas en el tanque 2 utilizando control A*-PID.

	Prueba 1	Prueba 2	Prueba 3	Promedio
Tiempo de estabilización [seg]	108.47	59.22	18.95	62.21
Sobreimpulso [%]	27.35	6.44	0.00	11.26

Los resultados obtenidos, tanto en la simulación como las pruebas en el sistema físico del multitank, demuestran que los algoritmos posibilitan la reducción del tiempo de establecimiento y sobreimpulso del sistema.

Conclusiones y trabajos futuros

En este capítulo, se describen las conclusiones obtenidas al finalizar el trabajo de investigación. Así, se resaltan los resultados obtenidos en los diferentes esquemas de control y sus limitaciones. Además, se presentan propuestas para trabajos de investigación futuros.

6.1. Conclusiones

Las estrategias de control **FGS-PID**, **SA-PID** y **A*-PID**, se basan en los principios de los algoritmos de inteligencia artificial. Entre ellos, se encuentran la lógica difusa, recocido simulado y el método de búsqueda **A***. El control **FGS-PID** ha sido arduamente usado en diversos sistemas de control. Sin embargo, en este trabajo se define un nuevo conjunto de reglas que eliminan la dependencia existente entre la ganancia derivativa y la integral. La ventaja de este sistema de control es que trabaja utilizando el error de la señal y su primera diferencia. Por ello, el sistema actúa sobre cada muestra; a diferencia de los esquemas de control **SA-PID** y **A*-PID** que, trabajan en cada cambio de referencia. Consecuentemente, la respuesta de la señal es mejorada luego de cierto lapso. Ambos algoritmos determinan el valor de las ganancias, tomando un determinado número de muestras de la señal luego del cambio de referencia. Las ganancias son seleccionadas basándose en una función de costo u objetivo. Pese al uso de la misma función, estos algoritmos trabajan de forma diferente. Tal es el caso del algoritmo **SA-PID**, mismo que usa una función de temperatura exponencial y una función de probabilidad encargada de seleccionar mejores soluciones y, en algunos casos peores, para no estancarse en una solución local. Cabe recalcar que, la selección de vecinos para este algoritmo se realiza de forma aleatoria, comprendiendo un número infinito de valores dentro de un rango mínimo y máximo de ganancias. Esta última característica representa una clara diferencia con el algoritmo **A*-PID**, cuyo análisis se desarrolla en un espacio finito de valores. Por ende, las ganancias vecinas son los nodos expandidos en el proceso de búsqueda.

El modelamiento del sistema multitanque y la implementación de los sistemas de control en la simulación, son una premisa del comportamiento de los algoritmos de inteligencia artificial previo a la verificación sobre el sistema real. Debido a que el sistema físico de multitanque presenta ciertas variaciones que no se pueden controlar; tales como la no linealidad de las válvulas, pérdidas provocadas en las conexiones de las tuberías, ruido en las mediciones, etc., los resultados difieren con la simulación. A pesar de ello, se deduce la misma conclusión con respecto al funcionamiento de los algoritmos, observando una disminución del tiempo de establecimiento y sobreimpulso al implementar los algoritmos de inteligencia artificial.

La placa de desarrollo STM32F413 núcleo-144 es la encargada de controlar el sistema multitanque. En efecto, su velocidad de procesamiento, capacidad de memoria y disponibilidad de pines de entradas y salidas; son suficientes para llevar a cabo el proceso de control utilizando los algoritmos de ganancia programada. Esta plataforma trabaja con el lenguaje de programación C/C++. Por lo tanto, todas las funciones implementadas en la simulación en MATLAB son adaptadas en C. Efectivamente, los resultados muestran que, las funciones son implementadas correctamente permitiendo el control del sistema multitanque.

Con la finalidad de analizar el comportamiento de los algoritmos de control **GS** en el sistema multitanque, se construye un tablero de control. Este último no solo cuenta con el componente principal de control, la placa STM32, sino también elementos necesarios para el funcionamiento de electroválvulas y módulos conversores. Los módulos, por su parte, adecuan las señales de control, e interruptores que manipulan la alimentación tanto del sistema, en general, como para cada válvula y bomba.

Los controladores **PID**, **FGS-PID**, **SA-PID** y **A*-PID**, son implementados en cada válvula del sistema multitanque. De esta manera, se cumple con el objetivo de seguimiento de referencia. Tanto en la simulación como el sistema real, se muestra que la respuesta ante cambios de referencia se mejora al utilizar los controladores **GS**. La Tabla 6.1 sintetiza los resultados obtenidos en la simulación, donde se muestra que el controlador **SA-PID** obtiene un menor tiempo de estabilización en la respuesta del tanque 1, y el controlador **FGS-PID** en la respuesta del tanque 2. El algoritmo **A*-PID** no supera a los otros algoritmos en el tiempo de estabilización; sin embargo, su mejora se ve reflejada en el sobreimpulso. Además, en presencia de fallas del sistema, el algoritmo **A*-PID** posee el menor tiempo de estabilización cuando la fuga está en el tanque 1. Y el algoritmo **FGS-PID** posee el menor tiempo de estabilización cuando la fuga está en el tanque 2.

Tabla 6.1: Comparación de resultados del sistema multitanque simulado.		PID	FGS-PID	SA-PID	A*-PID
Tanque 1	Tiempo de estabilización [seg]	274.67	88.35	82.80	89.48
	Sobreimpulso [%]	5.01	4.10	1.83	0.22
Tanque 2	Tiempo de estabilización [seg]	132.51	77.01	82.15	83.35
	Sobreimpulso [%]	5.08	4.21	3.63	0.86
Falla Tanque 1	Tiempo de estabilización [seg]	324.04	115.05	130.29	93.70
	Sobreimpulso [%]	0.00	0.00	0.00	0.48
Falla Tanque 2	Tiempo de estabilización [seg]	610.00	256.56	358.35	323.68
	Sobreimpulso [%]	0.00	1.77	0.00	0.00

Los resultados del sistema multitanque real se sintetizan en la Tabla 6.2, donde el algoritmo **A*-PID** presenta el menor tiempo de estabilización para el tanque 1 y el algoritmo **SA-PID** para el tanque 2. Del mismo modo, al utilizar el algoritmo **SA-PID**, se obtiene la mejor evaluación en el sobreimpulso. Para el sistema con fugas, en los dos casos, el algoritmo **A*-PID** presenta el menor tiempo de estabilización. Además, se evidencia que en el tanque 2, el sistema de control **PID** no logra cumplir con los objetivos de seguimiento de referencia. Sin embargo, los algoritmos de ganancia programada si lo hacen, demostrando que los algoritmos son tolerantes a fallas.

Tabla 6.2: Comparación de resultados del sistema multitanque real.

		PID	FGS-PID	SA-PID	A*-PID
Tanque 1	Tiempo de estabilización [seg]	61.61	59.77	60.89	52.32
	Sobreimpulso [%]	9.59	9.49	0.98	3.88
Tanque 2	Tiempo de estabilización [seg]	72.12	60.94	51.61	66.81
	Sobreimpulso [%]	12.12	6.72	1.35	10.45
Falla Tanque 1	Tiempo de estabilización [seg]	122.60	108.51	105.99	105.98
	Sobreimpulso [%]	14.61	5.88	9.63	6.79
Falla Tanque 2	Tiempo de estabilización [seg]	-	110.81	80.90	62.21
	Sobreimpulso [%]	46.41	34.00	10.58	11.26

Los algoritmos de ganancia programada mejoran notablemente la respuesta del sistema en relación a su tiempo de estabilización y sobreimpulso, pese a esto establecer cuál es el mejor algoritmo resulta un desafío, ya que cada uno actúa de forma diferente. Además, la aleatoriedad que presentan los algoritmos **SA-PID** y **A*-PID** hace que se obtengan diferentes resultados para una misma prueba. Finalmente se debe considerar que los algoritmos son sub-óptimos, la razón es que los parámetros del controlador son escogidos a partir de un rango mínimo y máximo de ganancias, lo que no quita el hecho que fuera de este rango exista mejores soluciones.

6.2. Trabajos futuros

- Evaluar otros algoritmos de inteligencia artificial para la programación de ganancias del controlador **PID**.
- Evaluar nuevos índices de desempeño en la definición de la función objetivo de los algoritmos **SA-PID** y **A*-PID**.
- Aumentar el número de interacciones de los algoritmos **SA-PID** y el espacio de búsqueda

del algoritmo A^* -PID. Para esto, se recomienda realizar las pruebas sobre un sistema con una respuesta rápida.

- Buscar un método analítico para la optimización. Entre los principales parámetros de los algoritmos de control GS está el rango de ganancias, obtenido de forma experimental. Por tanto, definir un método analítico capaz de determinar este rango permitiría a los algoritmos ser óptimos.

Apéndice

Los apéndices [A](#), [B](#), [C](#) y [D](#) presentan los extractos de código de los algoritmos [GS](#) implementados en MATLAB para la simulación del sistema multitanque y los extractos de código implementados en STM32CubeIDE para el control de la placa STM32-núcleo144. Los códigos junto a sus archivos de apoyo se encuentran en los repositorios de GitHub, en [\[77\]](#) para MATLAB y en [\[78\]](#) para STM32CubeIDE. Por último, el apéndice [D.2.3](#) muestra el diseño del sistema multitanque y tablero de control.

Controlador PID

A.1. MATLAB

A.1.1. Determinación de la respuesta del sistema en un lazo de control PID

```
1 function [y,t] = PIDcontrol(G,ref,Gains,dt)
2     Kp = Gains(1);
3     Ki = Gains(2);
4     Kd = Gains(3);
5     s = tf('s');
6     Gc = Kp + Ki/s + Kd*s/(1+.001*s);
7     Loop = series(Gc,G);
8     ClosedLoop = feedback(Loop,1);
9     t = 0:dt:20;
10    [y,t] = step(ClosedLoop,t);
11    y = ref*y;
12 end
```

Extracto de código A.1: Función PIDcontrol().

A.2. STM32CubeIDE

A.2.1. Inicialización de variables y determinación de la señal de control del algoritmo de control PID

```
1 void PID_Init(pid_s *pid, int select)
2 {
3     (*pid).r = 1.0;           // referencia
4     (*pid).e[0] = 0.0;        // error
5     (*pid).e[1] = 0.0;        // error anterior e(k-1)
6     (*pid).e[2] = 0.0;        // error acumulado
7     (*pid).y = 0;             // salida de la planta
8     (*pid).u = 0;             // senal de control
9
10    switch (select)
11    {
12        // ganancias del controlador PID para la valvula 1
13        case 1:
14            (*pid).Kp = 0.2;
15            (*pid).Kd = 0.01;
16            (*pid).Ki = 0.0005 ;
17            break;
18        // ganancias del controlador PID para la valvula 2
19        case 2:
20            (*pid).Kp = 0.2;
21            (*pid).Kd = 0.1;
22            (*pid).Ki = 0.0001;
23            break;
24        // ganancias del controlador PID para la valvula 3
25        case 3:
26            (*pid).Kp = 0.7;
27            (*pid).Kd = 0.1;
```

```
28         (*pid).Ki = 0.001;
29         break;
30     // ganancias del controlador PID por defecto
31     default:
32         (*pid).Kp = 1;
33         (*pid).Kd = 0;
34         (*pid).Ki = 0;
35     }
36     (*pid).Gains[0] = (*pid).Kp; // ganancia proporcional
37     (*pid).Gains[1] = (*pid).Ki; // ganancia integral
38     (*pid).Gains[2] = (*pid).Kd; // ganancia derivativa
39 }
```

Extracto de código A.2: Función PID_Init().

```
1 float PID_control( float *e, float Gains[3], float Ts)
2 {
3     float u; // señal de control
4     float Kp = Gains[0], Ki = Gains[1], Kd = Gains[2]; // ganancias del
5     // controlador PID
6     float error = e[0]; // error actual
7     float error_ant = e[1]; // error anterior
8     float error_sum = e[2]; // sumatoria de los errores pasados
9     float delt_error = error - error_ant; // delta error
10    float error_acum = error_sum + error; // error acumulado
11    // ley de control
12    u = Kp * error + Ki * (Ts * error_acum) + Kd * (delt_error / Ts);
13    // actualizacion de valores del error anterior y el error acumulado
14    *(e+1) = error;
15    *(e+2) = error_acum;
16    // retorna el valor de control
17    return u;
18 }
```

Extracto de código A.3: Función PID_control().

Controlador FGS-PID

B.1. MATLAB

B.1.1. Implementación del controlador FGS-PID para un sistema representado en espacio de estados

```

1  fis = readfis('fuzzy_control')
2  for k = N+2 : Ns
3      y(k) = C*x(:,k) ;           % vector de salida
4      e(k) = r(k) - y(k);         % error
5      e_acum = e_acum + e(k);     % error acumulado
6      delt_e = (e(k) - e(k-1));  % diferencia del error
7      % -----Controlador FGS-PID -----
8      output= evalfis(fis,[e(k) delt_e]);
9      Kpp = output(1);
10     Kdp = output(2);
11     Kip = output(3);
12     Kp = (Kpmax - Kpmin) * Kpp + Kpmin;
13     Kd = (Kdmax - Kdmin) * Kdp + Kdmin;
14     Ki = (Kimax - Kimin) * Kip + Kimin;
15     u(k) = Kp * e(k) + Ki * T * e_acum + Kd * delt_e / T;
16     % -----
17     x(:,k+1) = A*x(:,k) + B*u(k); % vector de estados
18 end

```

Extracto de código B.1: Algoritmo de control FGS-PID en MATLAB.

B.1.2. Archivo fuzzy_control.fis

```

[System]
Name='fuzzy_control'
Type='mamdani'
Version=2.0
NumInputs=2
NumOutputs=3
NumRules=49
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='centroid'

[Input1]
Name='e'
Range=[-1 1]
NumMFs=7
MF1='NB': 'trapmf', [-1e+100 -1e+100 -0.9667 -0.7]
MF2='NM': 'trimf', [-1 -0.6667 -0.3333]
MF3='NS': 'trimf', [-0.6667 -0.3333 -5.551e-17]
MF4='ZO': 'trimf', [-0.3333 0 0.3333]
MF5='PS': 'trimf', [-5.551e-17 0.3333 0.6667]
MF6='PM': 'trimf', [0.3333 0.6667 1]
MF7='PB': 'trapmf', [0.7 0.9667 1e+100 1e+100]

```

```
[Input2]
Name='delt_e'
Range=[-1 1]
NumMFs=7
MF1='NB': 'trapmf', [-1e+100 -1e+100 -0.9667 -0.7]
MF2='NM': 'trimf', [-1 -0.6666 -0.3334]
MF3='NS': 'trimf', [-0.6666 -0.3334 0]
MF4='ZO': 'trimf', [-0.3334 0 0.3334]
MF5='PS': 'trimf', [0 0.3334 0.6666]
MF6='PM': 'trimf', [0.3334 0.6666 1]
MF7='PB': 'trapmf', [0.7 0.9666 1.034 1e+100]
```

```
[Output1]
Name='kpp'
Range=[0 1]
NumMFs=2
MF1='small': 'gaussmf', [0.4247 0]
MF2='big': 'gaussmf', [0.4247 1]
```

```
[Output2]
Name='kdp'
Range=[0 1]
NumMFs=2
MF1='small': 'gaussmf', [0.4247 0]
MF2='big': 'gaussmf', [0.4247 1]
```

```
[Output3]
Name='Kip'
Range=[0 1]
NumMFs=4
MF1='S': 'gaussmf', [0.1 0]
MF2='SM': 'gaussmf', [0.1 0.3]
MF3='BM': 'gaussmf', [0.1 0.705]
MF4='B': 'gaussmf', [0.1 1]
```

```
[Rules]
1 1, 2 1 4 (1) : 1
1 2, 2 1 4 (1) : 1
1 3, 2 1 4 (1) : 1
1 4, 2 1 4 (1) : 1
1 5, 2 1 4 (1) : 1
1 6, 2 1 4 (1) : 1
1 7, 2 1 4 (1) : 1
2 1, 1 2 3 (1) : 1
2 2, 2 2 3 (1) : 1
2 3, 2 1 4 (1) : 1
2 4, 2 1 4 (1) : 1
2 5, 2 1 4 (1) : 1
2 6, 2 2 3 (1) : 1
2 7, 1 2 3 (1) : 1
3 1, 1 2 2 (1) : 1
3 2, 1 2 3 (1) : 1
3 3, 2 2 3 (1) : 1
3 4, 2 1 4 (1) : 1
3 5, 2 2 3 (1) : 1
3 6, 1 2 3 (1) : 1
3 7, 1 2 2 (1) : 1
```

```

4 1, 1 2 1 (1) : 1
4 2, 1 2 2 (1) : 1
4 3, 1 2 3 (1) : 1
4 4, 2 2 3 (1) : 1
4 5, 1 2 3 (1) : 1
4 6, 1 2 2 (1) : 1
4 7, 1 2 1 (1) : 1
5 1, 1 2 2 (1) : 1
5 2, 1 2 3 (1) : 1
5 3, 2 2 3 (1) : 1
5 4, 2 1 4 (1) : 1
5 5, 2 2 3 (1) : 1
5 6, 1 2 3 (1) : 1
5 7, 1 2 2 (1) : 1
6 1, 1 2 3 (1) : 1
6 2, 2 2 3 (1) : 1
6 3, 2 1 4 (1) : 1
6 4, 2 1 4 (1) : 1
6 5, 2 1 4 (1) : 1
6 6, 2 2 3 (1) : 1
6 7, 1 1 3 (1) : 1
7 1, 2 1 4 (1) : 1
7 2, 2 1 4 (1) : 1
7 3, 2 1 4 (1) : 1
7 4, 2 1 4 (1) : 1
7 5, 2 1 4 (1) : 1
7 6, 2 1 4 (1) : 1
7 7, 2 1 4 (1) : 1

```

Extracto de código B.2: Archivo fuzzy_control.fis

B.1.3. Implementación del controlador FGS-PID para el control del sistema multitank en Simulink

```

1 function [sys,x0,str,ts,simStateCompliance] = sfun_fuzzycontrol(t,x,u,flag,
   fuzzy_vars)
2 switch flag,
3 % Initialization %
4 case 0,
5     [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes;
6 % Derivatives %
7 case 1,
8     sys=mdlDerivatives(t,x,u);
9 % Update %
10 case 2,
11     sys=mdlUpdate(t,x,u);
12 % Outputs %
13 case 3,
14     sys=mdlOutputs(t,x,u,fuzzy_vars);
15 % GetTimeOfNextVarHit %
16 case 4,
17     sys=mdlGetTimeOfNextVarHit(t,x,u);
18 % Terminate %
19 case 9,
20     sys=mdlTerminate(t,x,u);
21 % Unexpected flags %

```

```

22     otherwise
23         DAStudio.error('Simulink:blocks:unhandledFlag', num2str(flag));
24
25 end
26
27 function [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes
28 % defined by the S-function parameters.
29 sizes = simsizes;
30
31 sizes.NumContStates = 0;
32 sizes.NumDiscStates = 0;
33 sizes.NumOutputs = 3;
34 sizes.NumInputs = 2;
35 sizes.DirFeedthrough = 1;
36 sizes.NumSampleTimes = 1; % at least one sample time is needed
37
38 sys = simsizes(sizes);
39
40 % initialize the initial conditions
41 x0 = [];
42 % str is always an empty matrix
43 str = [];
44 % initialize the array of sample times
45 ts = [0 0];
46
47 % Specify the block simStateCompliance. The allowed values are:
48 % 'UnknownSimState', < The default setting; warn and assume
49 %   DefaultSimState
50 % 'DefaultSimState', < Same sim state as a built-in block
51 % 'HasNoSimState', < No sim state
52 % 'DisallowSimState' < Error out when saving or restoring the model sim
53 %   state
54 simStateCompliance = 'UnknownSimState';
55
56 function sys=mdlDerivatives(t,x,u)
57 sys = [];
58
59 function sys=mdlUpdate(t,x,u)
60 sys = [];
61
62 function sys=mdlOutputs(t,x,u,fuzzy_vars)
63     Kpmin = fuzzy_vars(1);
64     Kpmax = fuzzy_vars(2);
65     Kdmin = fuzzy_vars(3);
66     Kdmax = fuzzy_vars(4);
67     Kimin = fuzzy_vars(5);
68     Kimax = fuzzy_vars(6);
69     e = u(1);
70     delt_e = u(2);
71     global fis
72     %***** FSG-PID *****
73     output= evalfis(fis,[e delt_e]);
74     Kpp = output(1);
75     Kdp = output(2);
76     Kip = output(3);
77     Kp = (Kpmax - Kpmin) * Kpp + Kpmin;
78     Kd = (Kdmax - Kdmin) * Kdp + Kdmin;
79     Ki = (Kimax - Kimin) * Kip + Kimin;

```

```

78
79 sys = [Kp Ki Kd];
80
81 function sys=mdlGetTimeOfNextVarHit(t,x,u)
82 sampleTime = 1;    % Example, set the next hit to be one second later.
83 sys = t + sampleTime;
84
85 function sys=mdlTerminate(t,x,u)
86 sys = [];

```

Extracto de código B.3: Función sfun_fuzzycontrol.

B.2. STM32CubeIDE

B.2.1. Generación de las funciones de membresía y definición de las reglas difusas

```

1 void Fuzzy_Init(){
2     // Definicion de rangos de las funciones de error,delta error, Kp',
3     // Kd' y Ki'
4     fuzzy.range_e[0] = -1;          fuzzy.range_e[1] = 1;          //
5     // rango del error
6     fuzzy.range_de[0] = -1;         fuzzy.range_de[1] = 1;         //
7     // rango de la diferencia del error (delta error)
8     fuzzy.range_kpp[0] = 0;          fuzzy.range_kpp[1] = 1;        //
9     // rango de la ganancia proporcional Kp'
10    fuzzy.range_kdp[0] = 0;           fuzzy.range_kdp[1] = 1;        //
11    // rango de la ganancia derivativa Kd'
12    fuzzy.range_kip[0] = 0;           fuzzy.range_kip[1] = 1;        //
13    // rango de la ganancia integral Ki'
14
15    // definicion de logitud maxima de los vectores para las funciones
16    // de membresia
17    int len_e = n_max_e;
18    int len_de = n_max_e;
19    int len_kpp = n_max_kpp;
20    int len_kdp = n_max_kpp;
21    int len_kip = n_max_kip;
22    fuzzy.len_MFout[0] = len_kpp;
23    fuzzy.len_MFout[1] = len_kdp;
24    fuzzy.len_MFout[2] = len_kip;
25
26    /* generacion de los vectores de error, delta error, Kp',Kd',Ki'
27    con valores dentro del rango y espacio entres valores (paso)
28    presetablecido */
29    float paso = PASO;
30    generate_vector(fuzzy.error, fuzzy.range_e[0], fuzzy.range_e[1], paso
31    );
32    generate_vector(fuzzy.delt_error, fuzzy.range_de[0], fuzzy.range_de
33    [1], paso);
34    generate_vector(fuzzy.kpp, fuzzy.range_kpp[0], fuzzy.range_kpp[1],
35    paso);
36    generate_vector(fuzzy.kdp, fuzzy.range_kdp[0], fuzzy.range_kdp[1],
37    paso);
38    generate_vector(fuzzy.kip, fuzzy.range_kip[0], fuzzy.range_kip[1],
39    paso);

```

```

28 // definicion de las variables de entrada de entrada y salida
29 fuzzy.fun_in_out[0] = fuzzy.error;
30 fuzzy.fun_in_out[1] = fuzzy.delt_error;
31 fuzzy.fun_in_out[2] = fuzzy.kpp;
32 fuzzy.fun_in_out[3] = fuzzy.kdp;
33 fuzzy.fun_in_out[4] = fuzzy.kip;
34 // funciones de membresia para la variable de entrada del error
35 for (int i = 0 ; i < len_e; ++i){
36     fuzzy.eNB[i] = trapmf(fuzzy.error[i], -1E100, -1E100,
37         -0.9667, -0.7);
38     fuzzy.eNM[i] = trimf(fuzzy.error[i], -1, -0.6667, -0.3333)
39         ;
40     fuzzy.eNS[i] = trimf(fuzzy.error[i], -0.6667, -0.3333,
41         -5.551E-17);
42     fuzzy.eZO[i] = trimf(fuzzy.error[i], -0.3333, 0, 0.3333);
43     fuzzy.ePS[i] = trimf(fuzzy.error[i], -5.551e-17, 0.3333,
44         0.6667);
45     fuzzy.ePM[i] = trimf(fuzzy.error[i], 0.3333, 0.6667, 1);
46     fuzzy.ePB[i] = trapmf(fuzzy.error[i], 0.7, 0.9667, 1E10,
47         1E10);
48 }
49 fuzzy.eMF[0] = fuzzy.eNB;
50 fuzzy.eMF[1] = fuzzy.eNM;
51 fuzzy.eMF[2] = fuzzy.eNS;
52 fuzzy.eMF[3] = fuzzy.eZO;
53 fuzzy.eMF[4] = fuzzy.ePS;
54 fuzzy.eMF[5] = fuzzy.ePM;
55 fuzzy.eMF[6] = fuzzy.ePB;
56 // funciones de membresia para la variable de entrada delta error
57 for (int i = 0 ; i < len_de; ++i){
58     fuzzy.deNB[i] = trapmf(fuzzy.delt_error[i], -1E100, -1E100,
59         -0.9667, -0.7);
60     fuzzy.deNM[i] = trimf(fuzzy.delt_error[i], -1, -0.6667,
61         -0.3334);
62     fuzzy.deNS[i] = trimf(fuzzy.delt_error[i], -0.6666,
63         -0.3333, 0);
64     fuzzy.deZO[i] = trimf(fuzzy.delt_error[i], -0.3334, 0,
65         0.3334);
66     fuzzy.dePS[i] = trimf(fuzzy.delt_error[i], 0, 0.3334,
67         0.6667);
68     fuzzy.dePM[i] = trimf(fuzzy.delt_error[i], 0.3334, 0.6667,
69         1);
70     fuzzy.dePB[i] = trapmf(fuzzy.delt_error[i], 0.7, 0.9667,
71         1E10, 1E10);
72 }
73 fuzzy.deMF[0] = fuzzy.deNB;
74 fuzzy.deMF[1] = fuzzy.deNM;
75 fuzzy.deMF[2] = fuzzy.deNS;
76 fuzzy.deMF[3] = fuzzy.deZO;
77 fuzzy.deMF[4] = fuzzy.dePS;
78 fuzzy.deMF[5] = fuzzy.dePM;
79 fuzzy.deMF[6] = fuzzy.dePB;
80 // funciones de membresia para las varaibles de salida Kp'
81 for (int i = 0 ; i < len_kpp; ++i){
82     fuzzy.small1[i] = gaussmf(fuzzy.kpp[i], 0.4247, 0);
83     fuzzy.big1[i] = gaussmf(fuzzy.kpp[i], 0.4247, 1);
84 }
85 fuzzy.kppMF[0] = fuzzy.small1; fuzzy.kppMF[1] = fuzzy.big1;

```

```

74 // funciones de membresia para las varaibles de salida Kd'
75 for (int i = 0 ; i < len_kdp; ++i){
76     fuzzy.small2[i] = gaussmf(fuzzy.kdp[i], 0.4247, 0);
77     fuzzy.big2[i] = gaussmf(fuzzy.kdp[i], 0.4247, 1);
78 }
79 fuzzy.kdpMF[0] = fuzzy.small2; fuzzy.kdpMF[1] = fuzzy.big2;
80 // funciones de membresia para las varaibles de salida Ki'
81 for (int i = 0 ; i < len_kip; ++i){
82     fuzzy.S[i] = gaussmf(fuzzy.kip[i], 0.1,0);
83     fuzzy.MS[i] = gaussmf(fuzzy.kip[i], 0.1, 0.295);
84     fuzzy.MB[i] = gaussmf(fuzzy.kip[i], 0.1, 0.705);
85     fuzzy.B[i] = gaussmf(fuzzy.kip[i], 0.1, 1);
86 }
87 fuzzy.kipMF[0] = fuzzy.S; fuzzy.kipMF[1] = fuzzy.MS;
88 fuzzy.kipMF[2] = fuzzy.MB; fuzzy.kipMF[3] = fuzzy.B;
89 // funciones de membresia de entrada
90 fuzzy.MFin[0]=fuzzy.eMF; fuzzy.MFin[1]=fuzzy.deMF;
91 // funciones de membresia de salida
92 fuzzy.MFout[0]=fuzzy.kppMF; fuzzy.MFout[1]=fuzzy.kdpMF; fuzzy.MFout
    [2]=fuzzy.kipMF;
93 // definicion de las reglas difusas
94 char *valores[N_RULES]= {
95     "1 1, 2 1 4",
96     "1 2, 2 1 4",
97     "1 3, 2 1 4",
98     "1 4, 2 1 4",
99     "1 5, 2 1 4",
100    "1 6, 2 1 4",
101    "1 7, 2 1 4",
102    "2 1, 1 2 3",
103    "2 2, 2 2 3",
104    "2 3, 2 1 4",
105    "2 4, 2 1 4",
106    "2 5, 2 1 4",
107    "2 6, 2 2 3",
108    "2 7, 1 2 3",
109    "3 1, 1 2 2",
110    "3 2, 1 2 3",
111    "3 3, 2 2 3",
112    "3 4, 2 1 4",
113    "3 5, 2 2 3",
114    "3 6, 1 2 3",
115    "3 7, 1 2 2",
116    "4 1, 1 2 1",
117    "4 2, 1 2 2",
118    "4 3, 1 2 3",
119    "4 4, 2 2 3",
120    "4 5, 1 2 3",
121    "4 6, 1 2 2",
122    "4 7, 1 2 1",
123    "5 1, 1 2 2",
124    "5 2, 1 2 3",
125    "5 3, 2 2 3",
126    "5 4, 2 1 4",
127    "5 5, 2 2 3",
128    "5 6, 1 2 3",
129    "5 7, 1 2 2",
130    "6 1, 1 2 3",

```

```

131         "6 2, 2 2 3",
132         "6 3, 2 1 4",
133         "6 4, 2 1 4",
134         "6 5, 2 1 4",
135         "6 6, 2 2 3",
136         "6 7, 1 1 3",
137         "7 1, 2 1 4",
138         "7 2, 2 1 4",
139         "7 3, 2 1 4",
140         "7 4, 2 1 4",
141         "7 5, 2 1 4",
142         "7 6, 2 1 4",
143         "7 7, 2 1 4"};
144     for (int i = 0 ; i < N_RULES; ++i) {
145         fuzzy.reglas[i] = valores[i];
146     }
147 }

```

Extracto de código B.4: Función Fuzzy_Init.

```

1 // standard deviation: sigma, and mean: m,
2 float gaussmf(float x, float sigma, float media){
3     float y , value;
4     value = -pow(x-media,2)/(2*pow(sigma,2));
5     y = exp(value);
6     return y ;
7 }

```

Extracto de código B.5: Función gaussmf.

```

1 float trimf(float x, float a, float b, float c){
2     float y ;
3     y = max(min((x-a)/(b-a),(c-x)/(c-b)),0);
4     return y ;
5 }

```

Extracto de código B.6: Función trimf.

```

1 float trapmf(float x, float a, float b, float c, float d){
2     float y ;
3     y = max(min((x-a)/(b-a),min(1,(d-x)/(d-c))),0);
4     return y ;
5 }

```

Extracto de código B.7: Función trapmf.

B.2.2. Evaluación del sistema de inferencia difuso

```

1 void eval_fuzzy(float *entradas, float *salidas){
2     float paso = PASO;
3     int n_entradas = num_MFin;
4     int n_salidas = num_MFout;
5     int n_rules = N_RULES;
6     int index_in[n_entradas];
7     float aux_var;
8     float **out;

```

```

9      float fuzzy_value;
10     int selec_MFin[n_entradas], selec_MFout[n_salidas];
11
12     // reservamos espacio de memoria para out
13     out = (float **) calloc(n_salidas, sizeof(float *));
14     for(int k = 0; k < n_salidas; ++k)
15         out[k] = (float *) calloc(fuzzy.len_MFout[k], sizeof(float)
16                                     );
17
18     //buscamos el indice correspondiente para cada variable de entrada;
19     for (int i = 0; i < n_entradas; ++i)
20         index_in[i] = find_index(*(entradas + i), fuzzy.error[0],
21                                   paso);
22
23     for(int i = 0 ; i < n_rules; ++i ){
24         /* Las salidas son seleccionadas de acuerdo a las reglas
25            difusas. Ejem: IF (e is NB) and (de is NB) THEN (kpp is
26            big ) and (kdp is small) and (Kpi is B)*/
27         Reglas_str2num(selec_MFin,selec_MFout,i);
28         //fusificacion
29         fuzzy_value = fuzz(n_entradas,selec_MFin,index_in);
30         for (int k = 0; k < n_salidas; ++k){
31             for (int j = 0; j < fuzzy.len_MFout[k]; ++j) {
32                 // inferencia difusa de mandani (and =>
33                 min)
34                 aux_var = min(*(fuzzy.MFout[k]+
35                                selec_MFout[k])+ j),fuzzy_value);
36                 // agrupacion
37                 if (i == 0)
38                     (*(out +k)+j)) = aux_var;
39                 else
40
41                     (*(out +k)+j) = max(*(out +k)+j)
42                                     ,aux_var);
43             }
44         }
45     }
46     //Defuzzificacion
47     for (int k = 0 ; k < n_salidas; ++k)
48         *(salidas+k) = defuzz(fuzzy.fun_in_out[n_entradas + k], out[k],
49                               fuzzy.len_MFout[k]);
50
51     //liberamos espacio de memoria de out
52     for (int k = 0; k < n_salidas; ++k)
53         free(out[k]);
54     free(out);
55 }

```

Extracto de código B.8: Función eval_fuzzy.

```

1 //fuzzificacion
2 float fuzz(int n_entradas, int selec_MFin[n_entradas], int index_in[
3     n_entradas]){
4     float fuzzy_value;
5     for ( int k = 0 ;k <n_entradas; ++k){
6         if (k == 0){
7             fuzzy_value = (*(fuzzy.MFin[k]+selec_MFin[k])+ index_in[k
8                                 ]);
9         }
10    }

```

```

7         }
8         else{
9             fuzzy_value = min(fuzzy_value ,*(*(fuzzy.MFin[k]+(selec_MFin
              [k]))+ index_in[k]));
10        }
11    }
12    return fuzzy_value;
13 }

```

Extracto de código B.9: Función fuzz.

```

1 //defuzzifiacion, metodo del centroide
2 float defuzz(float *x, float *mf,int len){
3     float out;
4     float suma = 0;
5     float Area = 0;
6     for (int i = 0;i<len;++i){
7         suma = suma + (*(x+i))*(*(mf+i));
8         Area = Area + (*(mf+i));
9     }
10    out = suma/Area;
11    return out;
12 }

```

Extracto de código B.10: Función defuzz.

```

1 // convierte la expresion de las reglas en forma string a valores numericos
2 void Reglas_str2num(int *v1,int *v2, int i){
3     int n = 0, cont1 = 0, cont2 = 0;
4     while (fuzzy.reglas[i][n] != ','){
5         if (fuzzy.reglas[i][n] != ' '){
6             *(v1+cont1) = fuzzy.reglas[i][n] - '0'-1;
7             cont1++;
8         }
9         n++;
10    }
11    while (fuzzy.reglas[i][n] != '\\0'){
12        if (fuzzy.reglas[i][n] != ' ' && fuzzy.reglas[i][n] != ','){
13            *(v2+cont2) = fuzzy.reglas[i][n] - '0'-1;
14            cont2++;
15        }
16        n++;
17    }
18 }

```

Extracto de código B.11: Función Reglas_str2num.

```

1 // encuentra la posicion de un elemento en un vector ordenado;
2 int find_index(float x, float inicio, float paso){
3     int index;
4     index = (x-inicio)/paso + 0.5;
5     if (index > n_max_e-1)
6         index = n_max_e-1;
7     else if (index < 0)
8         index = 0;
9     return index;

```

10 }

Extracto de código B.12: Función find_index.

B.2.3. Determinación de las ganancias mediante el algoritmo FGS-PID

```
1 void FGS_Init(fgs_s *fgs, int select){
2     pid_s pid;
3     PID_Init(&pid, select);
4     (*fgs).Kpmin = 0.533 * pid.Kp;
5     (*fgs).Kpmax = 1 * pid.Kp;
6     (*fgs).Kdmin = 1.067 * pid.Kd;
7     (*fgs).Kdmax = 2 * pid.Kd;
8     (*fgs).Kimin = 0.01 * pid.Ki;
9     (*fgs).Kimax = 0.85 * pid.Ki;
10    Fuzzy_Init();
11 }
```

Extracto de código B.13: Función FGS_Init.

```
1 void FGS_control(fgs_s *FGS, pid_s *pid){
2     float delt_e = pid->e[0] - pid->e[1];
3     float in[2], out[3];
4     in[0] = pid->e[0] ; in[1] = delt_e;
5     // ajuste del error y diferencia del error a valores en el rango de
6     // [-1,1]
7     if (pid->r != 0)
8     {
9         in[0] = in[0]/pid->r;
10        in[1] = in[1]/pid->r;
11    }
12    // evaluacion del sistema difuso
13    eval_fuzzy(in, out);
14    float kpp = out[0], kdp = out[1], kip = out[2];
15    // desnormalizacion de las ganancias
16    out[0] = (FGS->Kpmax - FGS->Kpmin) * kpp + FGS->Kpmin;
17    out[2] = (FGS->Kdmax - FGS->Kdmin) * kdp + FGS->Kdmin;
18    out[1] = (FGS->Kimax - FGS->Kimin) * kip + FGS->Kimin;
19    for (int i =0; i<3; ++i)
20        pid->Gains[i]=out[i];
21 }
```

Extracto de código B.14: Función FGS_control.

Controlador SA-PID

C.1. MATLAB

C.1.1. Implementación del algoritmo de control SA-PID

```

1 ObjectiveFunction = @(Gains)CalculeCost(G,ref,Gains,T);      % Funcion
  Objetivo
2 % ***** Parametros SA
  *****
3 iter = 50;          % Nnmero de iteraciones
4 subiter = 5;        % Numero of Sub-iteraciones
5 T0 = 0.025;         % Temperatura inicial
6 alpha = 0.99;       % Temp. Reduction Rate
7
8 % ***** Inicializacion *****
9 sol.Gain = [Kp Ki Kd];
10 sol.Cost = ObjectiveFunction(Gains);
11 BestSol = sol;      % mejor solucion inicial encontrada
12 BestCost = zeros(iter, 1); % vector de los mejores valores de costo
13 T = T0;             % temperatura inicial
14 for i = 1:iter
15     for subit = 1:subiter
16         % Creacion de una nueva solucion
17         newsol.Gain = CreateNeighbor(sol.Gain,range_gains);
18         newsol.Cost = ObjectiveFunction(newsol.Gain);
19         if newsol.Cost <= sol.Cost % verificamos si la nueva solicion es
mejor
20             sol = newsol;
21         else % si no es mejor solucion
22             DELTA = (newsol.Cost-sol.Cost)/sol.Cost;
23             P = exp(-DELTA/T);
24             if rand <= P
25                 sol = newsol;
26             end
27         end
28         % actualiza la mejor solucion encontrada
29         if sol.Cost <= BestSol.Cost
30             BestSol = sol;
31         end
32     end
33     BestCost(i) = BestSol.Cost;
34     T = alpha*T; % actualiza el valor de temperatura
35 end
36

```

Extracto de código C.1: Algoritmo de control SA-PID en MATLAB.

C.1.2. Determinación del costo de la función objetivo

```

1 function J = CostFunction(y,ref,select_costF,dt)
2     muestras = length(y);
3     yfinal = ref;
4     t = linspace(0,(muestras-1)*dt,muestras);
5     e = ref-y;

```

```

6   S1 = stepinfo(y,t,yfinal,'SettlingTimeThreshold',0.05);
7   ts = S1.SettlingTime;
8   OS = S1.Overshoot;
9   IAE = trapz(t,abs(e));
10  J = 0.05*ITAE + 0.05*ts + 0.90*OS;
11  if (isnan(J) || J == 0)
12      J = inf;
13  end
14 end

```

Extracto de código C.2: Función CostFunction.

C.1.3. Determinación de valores vecinos de ganancia

```

1 function new_gain = CreateNeighbor(gain,range_gains)
2     Kpmin = range_gains(1);
3     Kpmax = range_gains(2);
4     Kdmin = range_gains(3);
5     Kdmax = range_gains(4);
6     Kimin = range_gains(5);
7     Kimax = range_gains(6);
8     new_gain = gain;
9     pKp = 0.2;
10    pKi = 0.5;
11    pKd = 1-pKp-pKi;
12    r = rand;
13    c = cumsum([pKp pKi pKd]);
14    variacion = find(r <= c, 1, 'first');
15    switch variacion
16        case 1
17            Kp = Kpmin + (Kpmax-Kpmin)*rand;
18            new_gain(1) = Kp;
19        case 2
20            Ki = Kimin + (Kimax-Kimin)*rand;
21            new_gain(2) = Ki;
22        case 3
23            Kd = Kdmin + (Kdmax-Kdmin)*rand;
24            new_gain(3) = Kd;
25    end
26 end

```

Extracto de código C.3: Función CreateNeighbor.

C.1.4. Implementación del controlador SA-PID para el control del sistema multitanque en Simulink

```

1 function [sys,x0,str,ts,simStateCompliance] = sfun_SA(t,x,u,flag,SA_vars)
2 switch flag,
3     % Initialization %
4     case 0,
5         [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes;
6     % Derivatives %
7     case 1,

```

```

8     sys=mdlDerivatives(t,x,u);
9     % Update %
10    case 2,
11        sys=mdlUpdate(t,x,u);
12    % Outputs %
13    case 3,
14        sys=mdlOutputs(t,x,u,SA_vars);
15    % GetTimeOfNextVarHit %
16    case 4,
17        sys=mdlGetTimeOfNextVarHit(t,x,u);
18    % Terminate %
19    case 9,
20        sys=mdlTerminate(t,x,u);
21    % Unexpected flags %
22    otherwise
23        DASTudio.error('Simulink:blocks:unhandledFlag', num2str(flag));
24
25    end
26
27    function [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes
28    % defined by the S-function parameters.
29    sizes = simsizes;
30
31    sizes.NumContStates = 0;
32    sizes.NumDiscStates = 0;
33    sizes.NumOutputs = 4;
34    sizes.NumInputs = 9;
35    sizes.DirFeedthrough = 1;
36    sizes.NumSampleTimes = 1;    % at least one sample time is needed
37
38    sys = simsizes(sizes);
39
40    % initialize the initial conditions
41    x0 = [];
42    % str is always an empty matrix
43    str = [];
44    % initialize the array of sample times
45    ts = [0 0];
46    % Specify the block simStateCompliance. The allowed values are:
47    %     'UnknownSimState', < The default setting; warn and assume
48    %         DefaultSimState
49    %     'DefaultSimState', < Same sim state as a built-in block
50    %     'HasNoSimState', < No sim state
51    %     'DisallowSimState' < Error out when saving or restoring the model sim
52    %         state
53    simStateCompliance = 'UnknownSimState';
54    function sys=mdlDerivatives(t,x,u)
55    sys = [];
56    function sys=mdlUpdate(t,x,u)
57    sys = [];
58    function sys=mdlOutputs(t,x,u,SA_vars)
59    global SA
60    alpha = SA_vars(1);    % constante alpha
61    Temp = SA_vars(2);    % temperatura
62    select = SA_vars(3);    % seleccion de la funcion de costo
63    Ts = SA_vars(4);    % tiempo de muestreo
64    r_gains = SA_vars(5:end);    % rango de ganancias (Kpmin,Kpmax...)
65    e = u(1);    % error

```

```

64     ref = u(2); % referencia
65     enable = u(3); % habilitar algoritmo
66     iter = u(4); % numero de iteracion
67     subiter = u(5); % numero de subiteracion
68     sol.Cost = u(6); % costo de la solucion
69     sol.Gain = u(7:9); % ganancias de la solucion
70     y = ref - e; % respuesta del sistema
71
72 if enable == 1 && length(SA(select).y) > 1 % si SA-PID esta habilitado
73     Temp = Temp * (alpha)^(iter-1); % actualiza la temperatura
74     if subiter == 0 && iter == 0 % si es la primera vez que usa
75         el algoritmo
76         % calcula el costo inicial
77         sol.Cost = CostFunction(SA(select).y,ref,Ts,SA(select).selectCostF)
78         ;
79         % define las ganancia inicial
80         sol.Gain = [SA(select).Kp SA(select).Ki SA(select).Kd];
81         SA(select).BestSol = sol; % define la mejor solicion inicial
82         newsol.Gain = CreateNeighbor(sol.Gain,r_gains); % determina
83         soluciones vecinas
84         SA(select).Kp = newsol.Gain(1); % nueva ganancia Kp
85         SA(select).Ki = newsol.Gain(2); % nueva ganancia Ki
86         SA(select).Kd = newsol.Gain(3); % nueva ganancia Kd
87         SA(select).y = []; % vector de muestras
88         SA(select).subiter = 1; % primera sub-iteracion
89         SA(select).iter = 1; % primera iteracion
90         SA(select).sol = sol; % guarda la solucion
91         % salida del funcion sfun_SA
92         sys = [sol.Cost newsol.Gain(1) newsol.Gain(2) newsol.Gain(3) ];
93     else
94         if iter <= SA(select).iter_max % si no supera el numero de
95         iteraciones max
96             % determina el nuevo costo
97             newsol.Cost = CostFunction(SA(select).y,ref,Ts,SA(select).
98             selectCostF);
99             % define las nuevas ganancias
100             newsol.Gain = [SA(select).Kp SA(select).Ki SA(select).Kd];
101             if newsol.Cost <= sol.Cost % verificamos si la nueva solucion
102             es mejor
103                 sol = newsol; % si lo es actualizamos
104             else % si no es mejor solucion
105                 % la solucion se acepta con probablidad P
106                 DELTA = (newsol.Cost-sol.Cost)/sol.Cost;
107                 P = exp(-DELTA/Temp);
108                 if rand <= P
109                     sol = newsol;
110                 end
111             end
112             % actualizamos la mejor solucion encontrada
113             if sol.Cost <= SA(select).BestSol.Cost
114                 SA(select).BestSol = sol;
115             end
116             SA(select).y = []; % reinicia el vector de muestras
117             % determina soluciones vecinas
118             newsol.Gain = CreateNeighbor(sol.Gain,r_gains);
119             % guarda la nueva soluciones
120             SA(select).Kp = newsol.Gain(1);
121             SA(select).Ki = newsol.Gain(2);

```

```

116         SA(select).Kd = newsol.Gain(3);
117         SA(select).sol = sol;
118         % salida de la funcion sfun_SA
119         sys = [SA(select).sol.Cost newsol.Gain(1) newsol.Gain(2) newsol
120             .Gain(3) ];
121         % si termina el algoritmo la salida de la funcion sfun_SA es
122         if iter == SA(select).iter_max
123             % la mejor solucion
124             sys = [SA(select).BestSol.Cost SA(select).BestSol.Gain(1)
125                 ...
126                 SA(select).BestSol.Gain(2) SA(select).BestSol.Gain
127                 (3)];
128         end
129     else
130         % si no se habilita el algoritmo
131         SA(select).subiter = subiter; % guarda el numero de sub-iteracion
132         SA(select).iter = iter; % guarda el numero de iteracion
133         SA(select).y = [SA(select).y y ]; % guarda la muestra
134         sys = [SA(select).sol.Cost SA(select).Kp SA(select).Ki SA(select).Kd ];
135         % salida de la funcion sfun_SA
136     end
137     % si termina el algoritmo la salida de la funcion sfun_SA es
138     if iter > SA(select).iter_max
139         % la mejor solucion
140         sys = [SA(select).BestSol.Cost SA(select).BestSol.Gain(1) SA(select).
141             BestSol.Gain(2) SA(select).BestSol.Gain(3)];
142     end
143
144 function sys=mdlGetTimeOfNextVarHit(t,x,u)
145 sampleTime = 1; % Example, set the next hit to be one second later.
146 sys = t + sampleTime;
147
148 function sys=mdlTerminate(t,x,u)
149 sys = [];

```

Extracto de código C.4: Función sfun_SA.

C.2. STM32CubeIDE

C.2.1. Inicialización del algoritmo SA-PID

```

1 void SA_Init(sa_s *SA, int select){
2     SA->iniciar = 0;
3     SA->enable = 0; // habilita el algoritmo SA
4
5     SA->iter[0] = 0; // Inicio
6     SA->iter[1] = 3; // Fin
7
8     SA->subiter[0] = 0; // Inicio
9     SA->subiter[1] = 3; // Fin
10
11     SA->muestra_i = 0; // Inicio
12     SA->muestras = 70; // Fin

```

```

13     SA->selectCostF = 2;    // Selecion de la funcion objetivo
14     SA->Temp = 0.025;      // Temperatura Inicial
15     SA->alpha = 0.99;
16     SA->y = (float *)malloc((SA->muestras+1)*sizeof(float));
17     SA->BestCost = (float *)malloc((SA->iter[1])*sizeof(float));
18
19     pid_s pid;
20     PID_Init(&pid, select);
21     SA->rangeGains[0] = 0.533 * pid.Kp; // Kp minimo
22     SA->rangeGains[1] = 1 * pid.Kp;    // Kp maximo
23     SA->rangeGains[2] = 1.067 * pid.Kd; // Kd minimo
24     SA->rangeGains[3] = 2 * pid.Kd;    // Kd maximo
25     SA->rangeGains[4] = 0.01 * pid.Ki; // Ki minimo
26     SA->rangeGains[5] = 0.85 * pid.Ki; // Kd maximo
27     srand(time(NULL));                // establece la semilla para
                                     generar numeros aleatorios.
28 }

```

Extracto de código C.5: Función SA_Init.

C.2.2. Determinación de las ganancias mediante el algoritmo SA-PID

```

1 void Simulated_Annealing(sa_s *SA, pid_s *pid, float dt)
2 {
3     if (SA->iniciar == 1) // se inicializa en el primer cambio de
4         referencia
5     {
6         if (SA->enable == 1) // se habilita en cada cambio de referencia
7         {
8             solSA_s newsol;
9             // se determina la solucion inicial
10            if (SA->iter[0] == 0 && SA->subiter[0] == 0)
11            {
12                for (int i =0;i<3;++i){
13                    SA->sol.Gain[i] = (*pid).Gains[i];}
14                    SA->sol.Cost = calcule_cost((SA->muestras)-1, SA->y, SA->r,
15                        dt, SA->selectCostF);
16                    SA->BestSol = SA->sol;
17                    // crea nuevos valores de Kp , Kd o Ki
18                    CreateNeighbor(pid->Gains,SA->rangeGains);
19                    (SA->iter[0])++; // primera
20                        iteracion
21                    (SA->subiter[0])++; // primera
22                        subiteracion
23                }
24            else
25            {
26                if (SA->iter[0] <= SA->iter[1]) // realiza si aun no se
27                    cumple con numero de iteraciones
28                {
29                    if (SA->subiter[0]<=SA->subiter[1]) // realiza si aun
30                        no se cumple con numero de sub-iteraciones
31                    {
32                        for (int i =0;i<3;++i) {
33                            newsol.Gain[i] = (*pid).Gains[i];} //nuevos valores
34                            de ganancias

```

```

28         newsol.Cost = calcule_cost((SA->muestras)-1, SA->y,
29             SA->r, dt, SA->selectCostF); //evaluacion de la
30             funcion objetivo
31         // la nueva solucion pasa ha ser la solucion actual
32             si tiene un costo menor
33         if (newsol.Cost <=SA->sol.Cost){SA->sol = newsol;}
34         else
35         { // caso contrario la nueva solucion puede ser
36             aceptada o no con probabilidad P
37             float DELTA = (newsol.Cost-SA->sol.Cost)/SA->
38                 sol.Cost;
39             float P = exp(-DELTA/SA->Temp);
40             if (randf() <= P){SA->sol = newsol;}
41         }
42         // actualiza la mejor solucion encontrada
43         if (SA->sol.Cost <= SA->BestSol.Cost)
44         {
45             SA->BestSol = SA->sol;
46         }
47         CreateNeighbor(pid->Gains,SA->rangeGains);// crea
48             sol. vecinas
49     }
50     int cont = 0;
51     // actualiza el valor de las sub-iteraones
52     if (SA->subiter[0] == SA->subiter[1]){(SA->subiter[0])
53         =1;cont = 1;}
54     else {(SA->subiter[0])++;}
55     if (cont == 1)
56     { // actualiza el valor de temperatura
57         SA->Temp = SA->alpha*SA->Temp;
58         // guarda la mejor solucion
59         SA->BestCost[(SA->iter[0])-1] = SA->BestSol.Cost;
60         // actualiza el valor de las iteraciones
61         (SA->iter[0])++;
62         cont = 0;
63     }
64 }
65 else // al finalizar el numero de iteraciones
66 { // se fija las ganancias del controlador PID con la
67     mejor solucion
68     for(int i = 0;i<3;i++)
69         pid->Gains[i] = SA->BestSol.Gain[i];
70 }
71 }
72 SA->muestra_i = 0; SA->enable = 0;
73 }
74 else // se toma determinadas muestras
75 {
76     if (SA->muestra_i < SA->muestras-1) {SA->muestra_i ++};
77 }
78 }
79 }

```

Extracto de código C.6: Función Simulated_Annealing.

C.2.3. Evaluación de la función objetivo

```

1  float calcule_cost(int muestras, float y[muestras], float ref, float dt,
2      int select){
3      float J,t[muestras] , e[muestras],OS,tr,tss,ISE ,ITSE , IAE, ITAE,
4          y_aux[muestras];
5      float yinit = y[0],yfinal = ref;
6      linspace(t, 0,(muestras-1)*dt+0,muestras);
7      prod_vector_escalar(muestras, y, -1, y_aux);
8      sum_vector_escalar(muestras, y_aux, ref, e);
9      switch (select) {
10         case 1:
11             IAE = IAE_calc(muestras, e, dt);
12             OS = Overshoot(muestras, y, yinit, yfinal);
13             tss = SettlingTime(muestras, t, y, yinit,
14                 yfinal, 0.05);
15             J = 0.10*IAE + 0.60*OS + 0.20*tss;
16             break;
17         case 2:
18             IAE = IAE_calc(muestras, e, t, dt);
19             tss = SettlingTime(muestras, t, y, yinit,
20                 yfinal, 0.05);
21             OS = Overshoot(muestras, y, yinit, yfinal);
22             J = 0.05*IAE + 0.05*tss + 0.90*OS;
23             break;
24         default:
25             ISE = ISE_calc(muestras, e, dt);
26             ITSE = ITSE_calc(muestras, e, t, dt);
27             IAE = IAE_calc(muestras, e, dt);
28             ITAE = ITAE_calc(muestras, e,t, dt);
29             OS = Overshoot(muestras, y, yinit, yfinal);
30             tss = SettlingTime(muestras, t, y, yinit,
31                 yfinal, 0.05);
32             tr = RiseTime(muestras, t, y, yinit, yfinal
33                 );
34             J = tr + tss + OS + ITSE + IAE + IAE +
35                 ITAE;
36             break;
37     }
38     if (J != J){
39         J = INFINITY;
40     }
41     return J;
42 }

```

Extracto de código C.7: Función calcule_cost.

```

1  float SettlingTime(int muestras,float *t, float *y,float yinit, float
2      yfinal,float porcentaje){
3      float tss = NAN, yaux = yfinal;
4      float ajuste=0;
5      int i = muestras-1;
6
7      while (fabsf((y[i]-yfinal)/(yfinal-yinit))<=porcentaje && i>=0){
8          i--;
9      }
10     //ajuste lineal para un valor mas aproximado del tiempo;

```

```

11
12     if (i != muestras-1)
13     {
14         if ((y[i]-yfinal)/(yfinal-yinit)>0)
15             yaux = yfinal*(1+porcentaje);
16         else if ((y[i]-yfinal)/(yfinal-yinit)<0)
17             yaux = yfinal*(1-porcentaje);
18
19         ajuste = (t[i+1]-t[i])/(y[i+1]-y[i])*(yaux-y[i]);
20
21         tss = t[i]+ajuste;
22     }
23
24     return tss;
25 }

```

Extracto de código C.8: Función SettlingTime.

```

1 float Overshoot(int muestras, float *y, float yinit, float yfinal){
2
3     float OS = INFINITY;
4     float max_num = 0;
5     for (int i=0 ; i<muestras;++i)
6     {
7         if ((y[i]-yinit)/(yfinal-yinit)-1 > max_num)
8             max_num = ((y[i]-yinit)/(yfinal-yinit)-1);
9     }
10    if (max_num>0)
11        OS = max_num*100;
12    return OS;
13 }

```

Extracto de código C.9: Función Overshoot.

```

1 float IAE_calc(int muestras, float e[muestras], float dt)
2 {
3     float IAE = 0;
4     for (int i=1; i<muestras;++i)
5     {
6         IAE += ((fabsf(e[i])+fabsf(e[i-1]))/2)*dt;
7     }
8
9     return IAE;
10 }

```

Extracto de código C.10: Función IAE_calc.

```

1 float ITAE_calc(int muestras, float e[muestras], float t[muestras], float dt)
2 {
3     float ITAE = 0;
4
5     for (int i=1; i<muestras;++i){
6         ITAE += ((t[i]*fabsf(e[i])+t[i-1]*fabsf(e[i-1]))/2)
7             *dt;
8     }
9
10    return ITAE;

```

Extracto de código C.11: Función ITAE_calc.

```
1 float ISE_calc(int muestras, float e[muestras], float dt)
2 {
3     float ISE = 0;
4     for (int i=1; i < muestras; ++i){
5         ISE += ((e[i]*e[i]+e[i-1]*e[i-1])/2)*dt;
6     }
7     return ISE;
8 }
```

Extracto de código C.12: Función ISE_calc.

```
1 float ITSE_calc(int muestras, float e[muestras], float t[muestras], float dt)
2 {
3     float ITSE = 0;
4     for (int i=1; i<muestras; ++i)
5     {
6         ITSE += ((t[i]*e[i]*e[i]+t[i-1]*e[i-1]*e[i-1])/2)*dt;
7     }
8
9     return ITSE;
10 }
```

Extracto de código C.13: Función ITSE_calc.

```
1 float RiseTime(int muestras, float *t, float *y, float yinit, float yfinal){
2     float tr = 0;
3     int i1 = 0, i2 = 0;
4     int i = 0;
5
6     while((y[i]-yinit)/(yfinal-yinit) <= 0.1 && i<muestras){
7         i++;
8     }
9     i1 = i;
10    while((y[i]-yinit)/(yfinal-yinit) <= 0.9 && i<muestras){
11        i++;
12    }
13    i2 = i;
14
15    if ((i1 !=0 || i1 != muestras) && (i2!=0 || i2!=muestras) )
16        tr = t[i2]-t[i1];
17
18    return tr;
19 }
```

Extracto de código C.14: Función RiseTime.

C.2.4. Determinación de valores vecinos de ganancia

```
1 void CreateNeighbor(float *Gains, float range_gains[6]){
2
3     float Kpmin = range_gains[0],           Kpmax = range_gains
4         [1];
5     float Kdmin = range_gains[2],           Kdmax = range_gains[3];
```

```
5      float Kimin = range_gains[4],           Kimax = range_gains[5];
6
7      float pKp = 0.2,  pKi = 0.5,  pKd = 1-pKp-pKi;
8
9      float r = randf();
10     int variacion = 0;
11     if (r <=pKp)
12         variacion = 1;
13     if (r>pKp && r<=pKp+pKi)
14         variacion = 2;
15     if (r>pKp+pKi && r<=pKp+pKi+pKd)
16         variacion = 3;
17
18     switch (variacion) {
19         case 1:
20             Gains[0] = Kpmin + (Kpmax-Kpmin)*randf();
21             break;
22         case 2:
23             Gains[1] = Kimin + (Kimax-Kimin)*randf();
24             break;
25         case 3:
26             Gains[2] = Kdmin + (Kdmax-Kdmin)*randf();
27             break;
28         default:
29             break;
30     }
31
32 }
```

Extracto de código C.15: Función CreateNeighbor.

Controlador A*-PID

D.1. MATLAB

D.1.1. Determinación de las ganancias mediante el algoritmo A*-PID

```

1 function [Kp, Ki, Kd] = Aestrella(space, X1, Y1, Z1, X2, Y2, Z2, ID,
2   Kp_values, Kd_values, Ki_values, funtion_H)
3   [dim1, dim2, dim3] = size(space);           % espacio de busqueda
4   F = nan*ones(dim1, dim2, dim3);             % F = G + H
5   G = zeros(dim1, dim2, dim3);                % Costo
6   H = nan*ones(dim1, dim2, dim3);             % Heuristica
7   y_cifras = 10^length(num2str(dim1));
8   z_cifras = 10^(length(num2str(dim1)) + length(num2str(dim2)));
9   padre = zeros(dim1, dim2, dim3);            % nodos padres
10  Lista_a = ID(X1, Y1, Z1);                    % lista abierta
11  Lista_aF = nan*ones(dim1*dim2*dim3, 1);      % lista F (costos totales)
12  Lista_aF(1) = ID(X1, Y1, Z1);
13  Lista_c = [];                                % lista cerrada
14  fin = 0;                                     % bandera de salida
15  G_X1 = X1; G_Y1 = Y1; G_Z1 = Z1;            % guarda el valor del punto
16  inicial
17  while fin == 0 % mientras no encuentre el punto final
18      for z = Z1-1:Z1+1
19          for y = Y1-1 : Y1+1 % busca en los pixeles vecinos el valor F
20              con menor costo
21                  for x = X1-1 : X1+1
22                      if x <= dim1 && y <= dim2 && z <= dim3 && x >= 1 && y
23                          >= 1 && z >= 1
24                              if space(x,y,z) == 0 && isempty(find(Lista_c == ID(
25                                  x,y,z), 1)) == 1
26                                  pos_a = find(Lista_a == ID(x,y,z), 1);
27                                  if isempty(pos_a) == 1 % si se tiene nuevos
28                                      vecinos
29                                          % calculaa el costo G y la heuristica H
30                                          Gains = [Kp_values(x) Ki_values(z)
31                                              Kd_values(y)];
32                                          H(x,y,z) = funtion_H(Gains);
33                                          G(x,y,z) = G(X1, Y1, Z1) + sqrt((x-X1)^2+(y-Y1
34                                              )^2+(z-Z1)^2);
35                                          F(x,y,z) = G(x,y,z) + H(x,y,z);
36                                          % nodo de donde provino
37                                          padre(x,y,z) = ID(X1, Y1, Z1);
38                                          % se agrega a la lista de pixel de vecinos
39                                          conocidos
40                                          Lista_a = cat(1, Lista_a, ID(x,y,z)) ;
41                                          Lista_aF(length(Lista_a)) = F(x,y,z);
42                                  else
43                                      G_a = G(X1, Y1, Z1) + sqrt((x-X1)^2+(y-Y1)^2)+(z
44                                          -Z1)^2;
45                                      if G_a < G(x,y,z)
46                                          G(x,y,z) = G_a;
47                                          padre(x,y,z) = ID(X1, Y1, Z1); % si el
48                                          gasto es menor
49                                          F(x,y,z) = G(x,y,z) + H(x,y,z); % coloca
50                                          su nuevo padre
51                                          Lista_aF(pos_a) = F(x,y,z); % y costo

```

```

40                                     end
41                                 end
42                            end
43                        end
44                    end
45                end
46            end
47            ID_XYZ = find(Lista_aF == min(Lista_aF),1); % busca su
identificador
48            if isempty(ID_XYZ) ~= 1 % si se encontro la
posicion del valor minimo
49                ZYX = Lista_a(ID_XYZ); % determina las
cordenadas
50                Z1 = fix(ZYX / z_cifras);
51                YX = ZYX - Z1 * z_cifras;
52                Y1 = fix( YX / y_cifras);
53                X1 = YX - Y1 * y_cifras;
54                % quita el elemento encontrado de la lista de vecinos
55                % abiertos y lo coloca en los cerrados
56                Lista_c = cat(1,Lista_c,ID(X1,Y1,Z1));
57                Lista_a = [Lista_a(1:ID_XYZ-1,1); Lista_a(ID_XYZ+1:end,1)];
58                Lista_aF = [Lista_aF(1:ID_XYZ-1,1); Lista_aF(ID_XYZ+1:end,1)];
59            end
60            % si llega al destino o si se ha analziado todos los valores de
espacio de busqueda
61            % termina el bucle
62            if ((X1 == X2 && Y1 == Y2 && Z1 == Z2) || isempty(ID_XYZ) == 1)
63                fin = 1 ;
64                [x_min, yz_min] = find( H == min(H,[],'all'),1);
65                y_min = (mod(yz_min-1,dim2)+1);
66                z_min = fix((yz_min-1)/ dim2)+1;
67                Kp = Kp_values(x_min) ;
68                Kd = Kd_values(y_min);
69                Ki = Ki_values(z_min);
70            end
71        end
72    end

```

Extracto de código D.1: Algoritmo de control A*-PID en MATLAB

D.1.2. Generación de un identificador único a cada nodo

```

1 function ID = GenerateID(dim1,dim2,dim3)
2     ID = zeros(dim1,dim2,dim3); % matriz de ceros
3     x_cifras = 1;
4     y_cifras = 10^length(num2str(dim1));
5     z_cifras = 10^(length(num2str(dim1)) + length(num2str(dim2)));
6
7     for i = 1:dim1 % para cada fila de la matriz
8         for j = 1:dim2 % para cada columna de la matriz
9             for k = 1:dim3
10                 % el numero de identificacion es una transformacion lineal
11                 % de a fila i columna j y capa k
12                 ID(i,j,k) = i*x_cifras + j * y_cifras + k * z_cifras
13             ;
14         end
15     end

```

```

14         end
15     end
16 end

```

Extracto de código D.2: Funcion GenerateID

D.1.3. Implementación del controlador A*-PID para el control del sistema multitank en Simulink

```

1 function [sys,x0,str,ts,simStateCompliance] = sfun_Astar(t,x,u,flag,AS_vars
2 )
3 switch flag,
4     % Initialization %
5     case 0,
6         [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes;
7     % Derivatives %
8     case 1,
9         sys=mdlDerivatives(t,x,u);
10    % Update %
11    case 2,
12        sys=mdlUpdate(t,x,u);
13    % Outputs %
14    case 3,
15        sys=mdlOutputs(t,x,u,AS_vars);
16    % GetTimeOfNextVarHit %
17    case 4,
18        sys=mdlGetTimeOfNextVarHit(t,x,u);
19    % Terminate %
20    case 9,
21        sys=mdlTerminate(t,x,u);
22    % Unexpected flags %
23    otherwise
24        DASTudio.error('Simulink:blocks:unhandledFlag', num2str(flag));
25 end
26
27 function [sys,x0,str,ts,simStateCompliance]=mdlInitializeSizes
28 % defined by the S-function parameters.
29 sizes = simsizes;
30
31 sizes.NumContStates = 0;
32 sizes.NumDiscStates = 0;
33 sizes.NumOutputs = 3;
34 sizes.NumInputs = 4;
35 sizes.DirFeedthrough = 1;
36 sizes.NumSampleTimes = 1;    % at least one sample time is needed
37
38 sys = simsizes(sizes);
39
40 % initialize the initial conditions
41 x0 = [];
42 % str is always an empty matrix
43 str = [];
44 % initialize the array of sample times
45 ts = [0 0];

```

```

46
47 % Specify the block simStateCompliance. The allowed values are:
48 %   'UnknownSimState', < The default setting; warn and assume
   DefaultSimState
49 %   'DefaultSimState', < Same sim state as a built-in block
50 %   'HasNoSimState',   < No sim state
51 %   'DisallowSimState' < Error out when saving or restoring the model sim
   state
52 simStateCompliance = 'UnknownSimState';
53
54 function sys=mdlDerivatives(t,x,u)
55 sys = [];
56
57 function sys=mdlUpdate(t,x,u)
58 sys = [];
59
60 function sys=mdlOutputs(t,x,u,AS_vars)
61 global AS
62 select = AS_vars(1); % seleccion de la funcion H
63 Ts = AS_vars(2); % tiempo de muestreo
64 e = u(1); % error
65 ref = u(2); % referencia
66 enable = u(3); % bandera habilita el algoritmo
67 iter = u(4); % numero de iteracion
68 y = ref - e; % respuesta del sistema
69 if AS(select).fin == 0 % si el algoritmo aun no finaliza
70     if enable == 1 && length(AS(select).y) > 1 % si esta habilitado
71         dim = size(AS(select).space); % define la dimension
72         y_cifras = 10^length(num2str(dim(1))); % j de columnas
73         z_cifras = 10^(length(num2str(dim(1))) + length(num2str(dim(2))));
74         % k capas
75         salir = 0; % bandera de salida
76         while salir == 0 % si no esta habilitada la bandera de salida
77             posActual = [AS(select).X1 AS(select).Y1 AS(select).Z1]; %
78             posicion actual
79             % si no es el nodo destino
80             if ~( (AS(select).X1 == AS(select).X2 && AS(select).Y1 == AS(
81                 select).Y2 ...
82                 && AS(select).Z1 == AS(select).Z2) || AS(select).iter > AS(
83                 select).iter_max)
84                 % es la primera vez que inicia el algoritmo
85                 if (AS(select).cont == 0)
86                     cont_next = 1; % siguiente numero de conteo
87                     AS(select).pause = 1; % bandera de pausar
88                     coord = posActual; % coordenana actual
89                     else % si no lo es
90                         % delimita las coordenadas para que esten dentro del
91                         % espacio de busqueda
92                         AS(select).cont = delimitar_coord(dim,posActual,AS(
93                 select).cont );
94                 cont_next = delimitar_coord(dim,posActual,AS(select).
95                 cont +1);
96                 subiter = contador(AS(select).cont);
97                 coord = posActual + subiter -2 ;
98             end
99             % cambio de variable a (x,y,z)
100             x = coord(1); y = coord(2); z = coord(3);
101             if AS(select).space(x,y,z) == 0 && ...

```

```

96         isempty(find(AS(select).ListaC == AS(select).ID(x,y,z),
97             1)) == 1
98         pos_a = find(AS(select).ListaA == AS(select).ID(x,y,z),
99             1);
100         if isempty(pos_a) == 1 % si tiene nuevos vecino
101             if AS(select).pause == 1
102                 % calculamos el costo G y la heuristica H
103                 AS(select).H(x,y,z) = CostFunction(AS(select).
104                     y,...
105                     ref,Ts,AS(select).selectH );
106                 AS(select).G(x,y,z) = AS(select).G(AS(select).
107                     X1,...
108                     AS(select).Y1,
109                     AS(select).Z1)+ sqrt((x-AS(select).X1)^2+...
110                     (y-AS(select).Y1)^2+(z-AS(select).Z1)^2);
111                 AS(select).F(x,y,z) = AS(select).G(x,y,z) +...
112                 AS(select).H(x,y,z);
113                 % idicamos su predecesor
114                 AS(select).padre(x,y,z) = AS(select).ID(AS(
115                     select).X1,...
116                     AS(select).Y1,AS(select).Z1);
117                 % agreeamos a la lista de pixel de vecinos
118                 AS(select).ListaA = cat(1,AS(select).ListaA,...
119                     AS(select).ID(x,y,z)) ;
120                 AS(select).ListaAF =cat(1,AS(select).ListaAF
121                     ,...
122                     AS(select).F(x,y,z));
123                 AS(select).pause = 0; % no detiene el algoritmo
124                 AS(select).y = []; % reinicia el vector de
125                 muestras
126             else
127                 AS(select).pause = 1; % detiene el algoritmo
128                 salir = 1; % acitva la bandera de salida
129             end
130         else
131             % si no actulizamos G si es menor
132             G = AS(select).G(AS(select).X1,AS(select).Y1,AS(
133                 select).Z1)+...
134             sqrt((x-AS(select).X1)^2+(y-AS(select).Y1)^2)+(z-AS
135                 (select).Z1)^2;
136             if G < AS(select).G(x,y,z) % si es mejor solucion
137                 AS(select).G(x,y,z) = G;% guarda G
138                 % actualiza el predecesor, F y su costo
139                 AS(select).padre(x,y,z) = AS(select).ID(AS(
140                     select).X1,...
141                     AS(select).Y1,AS(select).Z1);
142                 AS(select).F(x,y,z) = AS(select).G(x,y,z) +...
143                 AS(select).H(x,y,z);
144                 AS(select).ListaAF(pos_a) = AS(select).F(x,y,z)
145             ;
146         end
147     end
148 end
149 if (((mod(AS(select).cont-1,27)>mod(cont_next-1,27)) && ...
150     AS(select).pause == 0 )||AS(select).cont == 0)
151     % buscamos su indentificador

```

```

141 ID_XYZ = find(AS(select).ListaAF == min(AS(select).
ListaAF),1);
142 if isempty(ID_XYZ) ~= 1% si se encontro la posicion
del valor minimo
143 ZYX = AS(select).ListaA(ID_XYZ);% determinamos las
coordenadas
144 AS(select).Z1 = fix(ZYX / z_cifras);
145 YX = ZYX - AS(select).Z1 * z_cifras;
146 AS(select).Y1 = fix( YX / y_cifras);
147 AS(select).X1 = YX - AS(select).Y1 * y_cifras;
148 % quitamos el elemento encontrado de la lista de
vecinos
149 % abiertos y colocamos en los cerrados
150 AS(select).ListaC = cat(1,AS(select).ListaC,...
151 AS(select).ID(AS(select).X1,AS(select).Y1,AS(select
).Z1));
152 AS(select).ListaA = [AS(select).ListaA(1:ID_XYZ
-1,1);...
153 AS(select).ListaA(ID_XYZ+1:end,1)];
154 AS(select).ListaAF = [AS(select).ListaAF(1:ID_XYZ
-1,1);...
155 AS(select).ListaAF(ID_XYZ+1:end,1)];
156 if (length(AS(select).ListaC) == dim(1)*dim(2)*dim
(3))
157 AS(select).fin =1;
158 salir = 1;
159 end
160 end
161 end
162 % si la bandera de pause se activa no aumenta el contador
163 if (AS(select).pause == 0)
164 AS(select).cont = AS(select).cont+1;
165 end
166 % si encuentra el destino termina.
167 else
168 AS(select).fin =1;
169 salir = 1;
170 end
171 end
172 if AS(select).fin == 1 % Si finaliza el algoritmo
173 % busca el min H determinado en la busqueda
174 [x_min, yz_min] = find( AS(select).H == min(AS(select).H,[],'
all'),1);
175 y_min = (mod(yz_min-1,dim(2))+1);
176 z_min = fix((yz_min-1)/ dim(2))+1;
177 % define las ganancias con mejor evaluacion
178 AS(select).Kp = AS(select).Kp_values(x_min) ;
179 AS(select).Kd = AS(select).Kd_values(y_min);
180 AS(select).Ki = AS(select).Ki_values(z_min);
181 else
182 % si no finaliza las ganancias vecinas son de la coordenada
actual
183 AS(select).Kp = AS(select).Kp_values(x) ;
184 AS(select).Kd = AS(select).Kd_values(y);
185 AS(select).Ki = AS(select).Ki_values(z);
186 end
187 % salida de la funcoion sfun_Astar
188 sys = [ AS(select).Kp AS(select).Ki AS(select).Kd ];

```

```

189     else
190         AS(select).iter = iter; % guarda el numero de iteracion
191         AS(select).y = [AS(select).y y ]; % guarda una muestra
192         % salida de la funcoion sfun_Astar
193         sys = [ AS(select).Kp AS(select).Ki AS(select).Kd ];
194     end
195 else
196     % salida de la funcion sfun_Astar
197     sys = [ AS(select).Kp AS(select).Ki AS(select).Kd ];
198 end
199
200 % delimita el espacio de busqueda
201 function iter = delimitar_coord(dim,posActual,iter)
202     while true
203         subiter = contador(iter);
204         coord = posActual + subiter -2;
205         for i = 1:3
206             if (coord(i)<=0 || coord(i)>dim(i))
207                 iter = iter + 3^(i-1);
208             end
209         end
210         if ~(coord(1)<=0 || coord(1)>dim(1) || coord(2)<=0 ...
211             || coord(2)>dim(2) || coord(3)<=0 || coord(3)>dim(3) )
212             break;
213         end
214     end
215
216 % contador de 3 cifra (111, 112 ... 333,111)
217 function subiter = contador(iter)
218     base = 3; cifras = 3;
219     cont = zeros(1,cifras);
220     subiter = zeros(1,cifras);
221     for i = 1:cifras
222         cont(i) = mod((iter -1),base^i)+1;
223         subiter(i) = fix((cont(i)-1)/(base^(i-1)))+1;
224     end
225
226 function sys=mdlGetTimeOfNextVarHit(t,x,u)
227 sampleTime = 1; % Example, set the next hit to be one second later.
228 sys = t + sampleTime;
229
230 function sys=mdlTerminate(t,x,u)
231 sys = [];

```

Extracto de código D.3: Función sfun_Astar.

D.2. STM32CubeIDE

D.2.1. Inicialización del algoritmo A*-PID

```

1 void Astar_Init(Astar_s *Astar,int select){
2     Astar->muestra_i = 0; // contador de muestras
3     Astar->muestras = 70; // numero de muestras
4     Astar->iter[0] = 1; // contador de iteraciones
5     Astar->iter[1] = 6; // numero maximo de iteraciones

```

```

6      Astar-> enable = 0;          // bandera que habilita el algoritmo
7      Astar-> fin = 0;             // bandera que finaliza el algoritmo
8      Astar->pausar = 0;           // bandera que pausa el algoritmo
9      Astar-> lenC = 0;            // longitud inicial de la lista de
    cerrados
10     Astar-> lenA = 0;            // longitud inicial de la lista de
    abiertos
11     Astar-> cont = 0;
12     Astar->dim_x = 3;            // numero de valores analizados de Kp
13     Astar->dim_y = 3;            // numero de valores analizados de Kd
14     Astar->dim_z = 3;            // numero de valores analizados de Ki
15
16     // rango de valores que puede tocar cada una de las ganancias
17     pid_s pid; PID_Init(&pid, select);
18     Astar->Kpmin = 0.533 * pid.Kp; // valor minimo de Kp
19     Astar->Kpmax = 1 * pid.Kp;    // valor maximo de Kp
20     Astar->Kdmin = 1.067 * pid.Kd; // valor minimo de Kd
21     Astar->Kdmax = 2 * pid.Kd;    // valor maximo de Kd
22     Astar->Kimin = 0.01 * pid.Ki; // valor minimo de Ki
23     Astar->Kimax = 0.85 * pid.Ki; // valor maximo de Ki
24     // reservamos espacio de memoria dinamica
25     reserve_memory_Astar(Astar);
26     // funcion generadora de identificador de los nodos
27     generate_ID(Astar->dim_x, Astar->dim_y, Astar->dim_z, Astar->ID,
    Astar->cifras);
28     zerosR3(Astar->dim_x, Astar->dim_y, Astar->dim_z, Astar->space);
29     zerosR3(Astar->dim_x, Astar->dim_y, Astar->dim_z, Astar->padre);
30     zerosR3f(Astar->dim_x, Astar->dim_y, Astar->dim_z, Astar->G);
31     onesR3f(Astar->dim_x, Astar->dim_y, Astar->dim_z, Astar->F);
32     prod_escalarMatrizR3(Astar->dim_x, Astar->dim_y, Astar->dim_z,
    Astar->F, NAN);
33     prod_escalarMatrizR3(Astar->dim_x, Astar->dim_y, Astar->dim_z,
    Astar->H, NAN);
34     linspace(Astar->Kp_values, Astar->Kpmin, Astar->Kpmax, Astar->dim_x)
    ;
35     linspace(Astar->Kd_values, Astar->Kdmin, Astar->Kdmax, Astar->dim_y)
    ;
36     linspace(Astar->Ki_values, Astar->Kimin, Astar->Kimax, Astar->dim_z)
    ;
37
38
39     // nodo de inicio, se encuentra en el vector el indice mas cercano
    al valor dado
40     Astar->X1 = find(pid.Gains[0], Astar->dim_x, Astar->Kp_values, -1);
41     Astar->Y1 = find(pid.Gains[2], Astar->dim_y, Astar->Kd_values, -1);
42     Astar->Z1 = find(pid.Gains[1], Astar->dim_z, Astar->Ki_values, -1);
43     // nodo destino, se asigna un nodo que no es encuentra en el espacio de
    busqueda
44     Astar->X2 = -1;
45     Astar->Y2 = -1;
46     Astar->Z2 = -1;
47 }

```

Extracto de código D.4: Función Astar_Init.

```

1 void reserve_memory_Astar(Astar_s *Astar){
2     //Asignacion dinamica de memoria para F
3     Astar->F =(float***) malloc(Astar->dim_x*sizeof(float**));

```

```

4      for (int i = 0 ; i<Astar->dim_x;++i){
5          Astar->F[i] = (float**) malloc(Astar->dim_y*sizeof(float*));
6          ;
7          for(int j = 0 ; j<Astar->dim_y;++j){
8              Astar->F[i][j] = (float*) malloc(Astar->dim_z*
9                  sizeof(float));
10         }
11     }
12     //Asignacion dinamica de memoria para H
13     Astar->H =(float**) malloc(Astar->dim_x*sizeof(float*));
14     for (int i = 0 ; i<Astar->dim_x;++i){
15         Astar->H[i] = (float**) malloc(Astar->dim_y*sizeof(float*));
16         ;
17         for(int j = 0 ; j<Astar->dim_y;++j){
18             Astar->H[i][j] = (float*) malloc(Astar->dim_z*
19                 sizeof(float));
20         }
21     }
22     //Asignacion dinamica de memoria para G
23     Astar->G =(float**) malloc(Astar->dim_x*sizeof(float*));
24     for (int i = 0 ; i<Astar->dim_x;++i){
25         Astar->G[i] = (float**) malloc(Astar->dim_y*sizeof(float*));
26         ;
27         for(int j = 0 ; j<Astar->dim_y;++j){
28             Astar->G[i][j] = (float*) malloc(Astar->dim_z*
29                 sizeof(float));
30         }
31     }
32     //Asignacion dinamica de memoria para el espacio de busqueda
33     Astar->space=(int**) malloc(Astar->dim_x*sizeof(int*));
34     for (int i = 0 ; i<Astar->dim_x;++i){
35         Astar->space[i] = (int**) malloc(Astar->dim_y*
36             sizeof(int*));
37         for(int j = 0 ; j<Astar->dim_y;++j){
38             Astar->space[i][j] = (int*) malloc(Astar->
39                 dim_z*sizeof(int));
40         }
41     }
42     //Asignacion dinamica de memoria para matriz de padres
43     Astar->padre=(int**) malloc(Astar->dim_x*sizeof(int*));
44     for (int i = 0 ; i<Astar->dim_x;++i){
45         Astar->padre[i] = (int**) malloc(Astar->dim_y*
46             sizeof(int*));
47         for(int j = 0 ; j<Astar->dim_y;++j){
48             Astar->padre[i][j] = (int*) malloc(Astar->
49                 dim_z*sizeof(int));
50         }
51     }
52     //Asignacion dinamica de memoria para matriz ID
53     Astar->ID=(int**) malloc(Astar->dim_x*sizeof(int*));
54     for (int i = 0 ; i<Astar->dim_x;++i){
55         Astar->ID[i] = (int**) malloc(Astar->dim_y*sizeof(
56             int*));
57         for(int j = 0 ; j<Astar->dim_y;++j){
58             Astar->ID[i][j] = (int*) malloc(Astar->
59                 dim_z*sizeof(int));
60         }
61     }

```

```

50 //Asignacion dinamica de memoria para los valores de Kp, Kd y Ki
51 Astar->Kp_values = (float *) malloc(Astar->dim_x*sizeof(float));
52 Astar->Kd_values = (float *) malloc(Astar->dim_y*sizeof(float));
53 Astar->Ki_values = (float *) malloc(Astar->dim_z*sizeof(float));
54 //Asignacion dinamica de memoria para listas de abiertos, cerrados y
    costos de abiertos
55 int max_valores = (Astar->iter[1]<(Astar->dim_x*Astar->dim_y*
    Astar->dim_z)?Astar->iter[1]:(Astar->dim_x*Astar->dim_y*Astar->
    dim_z));
56 Astar->ListaA = (int *) malloc(max_valores*sizeof(int));
57 Astar->ListaC = (int *) malloc(max_valores*sizeof(int));
58 Astar->ListaAF = (float *) malloc(max_valores*sizeof(float));
59 //Asignacion dinamica de memoria la senal de salida "y"
60 Astar->y= (float *)malloc((Astar->muestras+1)*sizeof(float));
61 }

```

Extracto de código D.5: Función reserve_memory_Astar.

D.2.2. Generación de un identificador único a cada nodo

```

1 void generate_ID(int filas,int columnas,int capas, int ***ID,int *cifras){
2
3     int x_cifras = 1,y_cifras, z_cifras;
4     char text1[10],text2[10];
5
6     sprintf(text1,"%d",filas-1);
7     y_cifras = (pow( 10,strlen( text1)) + 0.000001);
8
9     sprintf(text2,"%d",columnas-1);
10    z_cifras = (pow(10,strlen(text1)+strlen(text2))+ 0.000001);
11
12    cifras[0] = x_cifras;
13    cifras[1] = y_cifras;
14    cifras[2] = z_cifras;
15    for (int k = 0;k<capas;++k){
16        for (int i = 0;i<filas;++i){
17            for (int j = 0;j<columnas;++j){
18                ID[i][j][k] = i*x_cifras+j*y_cifras+k*
                    z_cifras;
19            }
20        }
21    }
22 }

```

Extracto de código D.6: Función generate_ID.

D.2.3. Determinación de las ganancias mediante el algoritmo A*-PID

```

1 void Aestrella(Astar_s *Astar,pid_s *pid,float dt)
2 {
3     if (Astar->fin == 0){ // si no ha finalizado
4         if (Astar->enable == 1) // se habilita en cambios de referencia
5         {
6             int x,y,z; // coordenada actual

```

```

7      int *coord[] = {&x,&y,&z}; // vector de la coordenada actual
8      // dimension del espacio de busqueda
9      int dim[] = {Astar->dim_x,Astar->dim_y,Astar->dim_z};
10     int posActual[3]; // definicion del nodo de inicio
11     int subiter[3];    // sub-iteraciones para la expacion de nodos
12     int salir = 0;     // bandera de salida
13     // variables auxiliares
14     int lenAF,pos,cont_next, find_indexC, find_indexA,ID_XYZ,ZYX,
        YX ;
15     float G;
16     do{ // establece el nuevo nodo de inicio
17         posActual[0] = Astar->X1;posActual[1] = Astar->Y1;posActual
            [2] = Astar->Z1;
18         // si no es el nodo destino hacer
19         if (!((Astar->X1 == Astar->X2 && Astar->Y1 == Astar->Y2 &&
            Astar->Z1 == Astar->Z2)|| Astar->iter[0] > Astar->iter
            [1]))
20         {
21             // si es la primera vez que se activa el algoritmo
22             if (Astar->cont == 0)
23             { // nodo actual es igual al nodo de inicio
24                 for (int i = 0;i<3;++i){(*coord[i]) = (posActual[i]
                    );}
25                 cont_next = 1; // actualiza el contador
26                 Astar->pausar = 1; // se detiene
27             }
28             else
29             {// Delimita las coordenadas que se encuentren fuera
                del rango 0<(x,y,z)<dim(x,y,z)
30                 Astar->cont = delimitar_coordenadas(dim, posActual,
                    Astar->cont );
31                 cont_next = delimitar_coordenadas(dim, posActual,
                    Astar->cont+1);
32                 //Determina la coordenada (x,y,z) actual
33                 contador(Astar->cont, subiter);
34                 for (int i = 0;i<3;++i){(*coord[i]) = (posActual[i]
                    ) + subiter[i]-2;}
35             }
36             // busca el nodo esta en la lista de cerrados
37             find_indexC=find_int(Astar->ID[x][y][z],Astar->lenC,
                Astar->ListaC, 0);
38             // si no esta en la lista de cerrados
39             if (Astar->space[x][y][z] == 0 && find_indexC ==-1)
40             { // busca en la lista de abiertos
41                 find_indexA = find_int(Astar->ID[x][y][z], Astar->
                    lenA, Astar->ListaA, 0);
42                 if (find_indexA == -1)
43                 {
44                     if (Astar->pausar == 1)
45                     {
46                         Astar->H[x][y][z] =
47                         calcule_cost((Astar->muestras)-1,Astar->y,
                            Astar->r, dt,2);
48                         Astar->G[x][y][z] = Astar->G[Astar->X1][
                            Astar->Y1][Astar->Z1] + sqrtf(powf(x-
                            Astar->X1,2)+powf(y-Astar->Y1,2)+powf(z-
                            Astar->Z1,2));

```

```

49         Astar->F[x][y][z] = Astar->G[x][y][z] +
50             Astar->H[x][y][z];
51         Astar->padre[x][y][z] = Astar->ID[Astar->X1
52             ][Astar->Y1][Astar->Z1];
53         Astar->lenA++;
54         Astar->ListaA[Astar->lenA-1] = Astar->ID[x
55             ][y][z];
56         Astar->ListaAF[Astar->lenA-1] = Astar->F[x
57             ][y][z];
58         Astar->pausar = 0;
59         Astar-> iter[0] ++;
60     }
61     else
62     {
63         Astar->pausar = 1;
64         salir = 1 ;
65     }
66 }
67 else
68 {
69     G = Astar->G[Astar->X1][Astar->Y1][Astar->Z1] +
70         sqrtf(powf(x-Astar->X1,2)+powf(y-Astar->Y1
71             ,2)+powf(z-Astar->Z1,2));
72     if (G < Astar->G[x][y][z] )
73     {
74         Astar->G[x][y][z] = G;
75         Astar->padre[x][y][z] = Astar->ID[Astar->X1
76             ][Astar->Y1][Astar->Z1];
77         Astar->F[x][y][z] =Astar->G[x][y][z] +
78             Astar->H[x][y][z];
79         Astar->ListaAF[find_indexA] = Astar->F[x][y
80             ][z];
81     }
82 }
83 }
84 if (((Astar->cont-1)%(27))>((cont_next-1)%(27)) &&
85     Astar->pausar == 0)||Astar->cont == 0 )
86 {
87     ID_XYZ = find(min_v(Astar->lenA, Astar->ListaAF),
88         Astar->lenA, Astar->ListaAF, 0);
89     if (ID_XYZ !=-1)
90     {
91         ZYX = Astar->ListaA[ID_XYZ];
92         Astar->Z1 = (int) (ZYX / Astar->cifras[2]);
93         YX = ZYX - Astar->Z1 * Astar->cifras[2];
94         Astar->Y1 = (int) YX / Astar->cifras[1];
95         Astar->X1 = YX - Astar->Y1 * Astar->cifras[1];
96         Astar->lenC++;
97         Astar->ListaC[Astar->lenC-1] = Astar->ID[Astar
98             ->X1][Astar->Y1][Astar->Z1];
99         lenAF = Astar->lenA;
100         pos = setdiff_int(&Astar->lenA, Astar->ListaA,
101             Astar->ID[Astar->X1][Astar->Y1][Astar->Z1]);
102         remove_value_position(&lenAF, Astar->ListaAF,
103             pos);
104         if (Astar->lenC ==(Astar->dim_x*Astar->dim_y*
105             Astar->dim_z)){Astar->fin = 1;salir= 1;}
106     }
107 }

```

```

92         }
93         if (Astar->pausar == 0){Astar-> cont ++;}
94     }
95     else
96     {
97         Astar->fin = 1;salir= 1;
98     }
99     }while (salir == 0);
100     //Selecciona los nuevos valores de ganacia para la siguiente
        iteracion
101     // la funcion menor Costo
102     if (Astar->fin == 1)
103     {
104         int out[3];
105         findminMatrizR3(Astar->dim_x, Astar->dim_y, Astar->dim_z,
            Astar->H, out);
106         if (out[0]!=-1)
107         {
108             pid->Gains[0] = Astar->Kp_values[out[0]];
109             pid->Gains[1] = Astar->Ki_values[out[2]];
110             pid->Gains[2] = Astar->Kd_values[out[1]];
111         }
112     }
113     else
114     {
115         pid->Gains[0] = Astar->Kp_values[x];
116         pid->Gains[1] = Astar->Ki_values[z];
117         pid->Gains[2] = Astar->Kd_values[y];
118     }
119     Astar->muestra_i = 0; Astar->enable = 0;
120 }
121 else
122 {
123     if (Astar->muestra_i < Astar->muestras-1){Astar->muestra_i ++;}
124 }
125 }
126 }

```

Extracto de código D.7: Función Aestrella.

```

1  int delimitar_coordenadas(int dim[3], int posActual[3],int iter){
2      int subiter[3];
3      int coord[3];
4      do {
5          contador(iter, subiter);
6          for (int i = 0;i<3;++i){
7              coord[i] = (posActual[i]) + subiter[i]-2;
8              if (coord[i] <0 || coord[i] >= dim[i]){iter += (int
                ) powf(3,i);}}
9
10         } while (coord[0]<0 || coord[0]>=dim[0] || coord[1]<0 || coord[1]>=
            dim[1] || coord[2]<0 || coord[2]>=dim[2]);
11
12         return iter;
13 }

```

Extracto de código D.8: Función delimitar_coordenadas.

```
1 // contador de 3 cifras
2 void contador(int iter, int *out){
3     int base = 3; int cifras = 3;
4     int cont[cifras];
5     for(int i = 0; i < cifras; ++i){
6         cont[i] = ((iter-1)%((int)powf(base, i+1)))+1;
7         out[i] = ((cont[i]-1)/powf(base, i)) + 1;}
8
9 }
```

Extracto de código D.9: Función contador.

Bibliografía

- [1] K. Ogata, *Ingeniería de Control Moderna*. Pearson Educación, 2013.
- [2] West Instruments, "Controladores de procesos industriales." 2016. [Online]. Available: http://www.sapiensman.com/tecnoficio/electricidad/instrumentacion_industrial4.php
- [3] C. Veloso, "Que es el Duty Cycle o ciclo de trabajo de una señal PWM," 2016. [Online]. Available: <https://www.electrontools.com/Home/WP/que-es-el-duty-cycle-o-ciclo-de-trabajo/>
- [4] Z.-Y. Zhao, M. Tomizuka, and S. Isaka, "Fuzzy gain scheduling of pid controllers," *IEEE transactions on systems, man, and cybernetics*, vol. 23, no. 5, pp. 1392–1398, 1993.
- [5] R. S. E. Bustos and F. E. L. Alvarado, "Diseño e implementación de los sistemas de instrumentación, comunicaciones y control de un sistema multi-tanque," B.S. Thesis, Universidad de Cuenca, 2018. [Online]. Available: <http://dspace.ucuenca.edu.ec/handle/123456789/31493>
- [6] BACOENG, "VITON Brass Solenoid Valve," 2022. [Online]. Available: <https://bacoeng.com/products/bacoeng-1-4-dc12v-solenoid-valve1-4-3-8-1-2-3-4-dc12v-ac100v-220v-available>
- [7] U.S. Solid, "Motorized Ball Valve in Stainless Steel- 3/4 inch Full port Electrical Ball Valve with a 3 Wire, 9-24V Setup," 2022. [Online]. Available: <https://ussolid.com/u-s-solid-motORIZED-ball-valve-3-4-stainless-steel-electrical-ball-valve-with-full-port-9-24-v-ac-dc-3-wire-setup.html>
- [8] L. Llamas, "Medir Distancia Con Arduino Y Sensor De Ultrasonidos Hc-Sr04," p. 1, 2017. [Online]. Available: <https://www.luisllamas.es/medir-distancia-con-arduino-y-sensor-de-ultrasonidos-hc-sr04/>
- [9] AliExpress, "Interruptor de Sensor de flujo de agua G1/2 ", medidor de flujo, contador de 1 30L/min|water flow|flow meterwater flow hall sensor," 2010-2022. [Online]. Available: <https://es.aliexpress.com/item/2052758715.html>
- [10] ST, "User manual STM32 Nucleo-144 boards," 2022, [Online]. Available: <https://www.st.com/en/evaluation-tools/nucleo-f413zh.html> and <https://www.digikey.com/en/products/detail/stmicroelectronics/NUCLEO-F413ZH/6559189>.

- [11] Mecatronium, "Módulo Relay, 4 canales 5VDC," p. 1, 2017. [Online]. Available: <https://naylorpmechatronics.com/drivers/152-modulo-relay-4-canales-5vdc.html>
- [12] UNIT Electronics, "Convertidor de PWM a Voltaje Analógico 0-10V PLC," 2016. [Online]. Available: <https://uelectronics.com/producto/convertidor-de-pwm-a-voltaje-analogico-0-10v-plc/>
- [13] A. Rosillo Meseguer, "Capítulo 6: Sintonía experimental de controladores." *Universidad Politécnica de Cartagena*, 2008. [Online]. Available: <https://repositorio.upct.es/handle/10317/174>
- [14] O. A. Orozco and V. M. A. Ruiz, "Sintonización de controladores pi y pid utilizando los criterios integrales iae e itae," *Ingeniería. Revista de la Universidad de Costa Rica*, vol. 13, no. 1-2, pp. 31–39, 2003.
- [15] E. P. Bazzetti de los Santos, "Diseño de un controlador PID con autosintonía basado en un modelo de redes neuronales dinámicas y control adaptativo." *Universidad Ricardo Palma*, p. 149, 2019. [Online]. Available: <https://repositorio.urp.edu.pe/handle/20.500.14138/1516>
- [16] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.
- [17] R. Kurzweil, R. Richter, R. Kurzweil, and M. L. Schneider, *The age of intelligent machines*. MIT press Cambridge, 1990, vol. 580.
- [18] N. J. Nilsson and N. J. Nilsson, *Artificial intelligence: a new synthesis*. Morgan Kaufmann, 1998.
- [19] D. Poole, A. Mackworth, and R. Goebel, "Computational intelligence: a logical approach. 1998," *Digital Library*, 1998.
- [20] J. Haugeland, *Artificial intelligence: The very idea*. MIT press, 1989.
- [21] R. Bellman, *Artificial Intelligence: Can Computers Think?* Course Technology, 1978.
- [22] E. Charniak and D. McDermott, "Introduction to artificial," *Intelligence (Addison Wesley, Reading, MA, 1984)*, 1985.
- [23] P. H. Winston, *Artificial intelligence*. Addison-Wesley Longman Publishing Co., Inc., 1992.
- [24] L. A. Zadeh, "Fuzzy sets, department of electrical engineering and electrics research laboratory," *University of California, Berkeley, California*, 1965.

- [25] C. Eduardo, E. L. De Vito *et al.*, “Introducción al razonamiento aproximado: lógica difusa,” *Revista Americana de Medicina Respiratoria*, vol. 6, no. 3, pp. 126–136, 2006.
- [26] O. G. Duarte, “Sistemas de lógica difusa: fundamentos,” *Ingeniería e Investigación*, pp. 22–30, 1999.
- [27] H. T. Nguyen, C. Walker, and E. A. Walker, *A first course in fuzzy logic*. Chapman and Hall/CRC, 2018.
- [28] F. Vidal-Verdu, A. Rodríguez-Vázquez, and B. Linares-Barranco, “Circuits and algorithms for adaptive neuro-fuzzy analog chips,” in *Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*. IEEE, 1994, pp. 331–338.
- [29] L. A. Zadeh, “The concept of a linguistic variable and its application to approximate reasoning—i,” *Information sciences*, vol. 8, no. 3, pp. 199–249, 1975.
- [30] —, “Fuzzy logic and approximate reasoning,” *Synthese*, vol. 30, no. 3, pp. 407–428, 1975.
- [31] C. G. Morcillo, “Lógica difusa una introducción práctica: Técnicas de soft computing,” p. 21, 2015. [Online]. Available: https://www.esi.uclm.es/www/cglez/downloads/docencia/2011_Softcomputing/LogicaDifusa.pdf
- [32] L. A. Gutiérrez Álvarez, “Metodología de identificación difusa basada en el estudio de controlabilidad de sistemas dinámicos,” *Facultad de Ciencias Físicas y Matemáticas*, 2016.
- [33] Jesus Alfonso Lopez, “Lógica Difusa : Introduccion y conceptos basicos,” p. 12, 2003. [Online]. Available: https://members.tripod.com/jesus_alfonso_lopez/FuzzyIntro2.html
- [34] F. O. Tellez, “Control con Lógica Difusa: Control difuso,” *Facultad de Ingeniería Eléctrica*, pp. 1–27, 2018.
- [35] D. A. Almeida Viteri, “Simulación digital de un control difuso para el problema de la bola suspendida,” B.S. thesis, Quito: EPN, 2001.
- [36] M. V. Espí, “Un nuevo algoritmo para la optimación de estructuras: el recocido simulado,” *Informes de la Construcción*, vol. 46, no. 436, pp. 49–69, 1995.
- [37] J. H. Wilches-Visbal and A. Martins Da Costa, “Algoritmo de recocido simulado generalizado para matlab,” *Ingeniería y Ciencia*, vol. 15, no. 30, pp. 117–140, 2019.

- [38] K. A. Dowsland and B. A. Díaz, “Diseño de heurística y fundamentos del recocido simulado,” *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, vol. 7, no. 19, p. 0, 2003.
- [39] R. Marin and P. Jose, *Inteligencia artificial. Técnicas, métodos y aplicaciones*. McGraw-Hill Interamericana de España S.L., 2008. [Online]. Available: <https://books.google.com.ec/books?id=cB8PPwAACAAJ>
- [40] J. d. C. P. Abarca, J. E. Urbano, and B. M. Bahena, “Esquema de enfriamiento en la metaheurística de recocido simulado,” *Inventio. La génesis de la cultura universitaria en Morelos*, 2019.
- [41] P. E. Hart, N. J. Nilsson, and B. Raphael, “the Heuristic Determination,” *IEEE Transactions of systems science and cybernetics*, pp. 100–107, 1968.
- [42] W. J. Seo, S. H. Ok, J. H. Ahn, S. Kang, and B. Moon, “An efficient hardware architecture of the A-star algorithm for the shortest path search engine,” *NCM 2009 - 5th International Joint Conference on INC, IMS, and IDC*, pp. 1499–1502, 2009.
- [43] C. Malagón, “Búsqueda heurística,” *Universidad de Nebrija*, pp. 1–25, 2017.
- [44] N. Minorsky, “Directional stability of automatically steered bodies,” *Journal of the American Society for Naval Engineers*, vol. 34, no. 2, pp. 280–309, 1922.
- [45] M. A. Moreno, “Apuntes de control pid,” *La Paz Enero*, vol. 8, pp. 6–7, 2001.
- [46] M. A. Johnson and M. H. Moradi, *PID control*. Springer, 2005.
- [47] Carlos Pardo Martín, “Controlador PID - Control Automático,” 2018. [Online]. Available: https://www.picuino.com/_sources/es/control-pid.rst.txt
- [48] V. Mazzone, “Controladores pid,” *Quilmes: Universidad Nacional de Quilmes*, 2002.
- [49] A. Lopez, “Tuning controllers with error-integral criteria,” *Instrumentation Technology*, vol. 14, pp. 57–62, 1967.
- [50] J. Villajulca, “Control ON/OFF o Todo/Nada - Instrumentacion y Automatizacion Industrial,” 2019. [Online]. Available: <https://instrumentacionycontrol.net/control-on-off-o-todo-nada/>
- [51] E. M. A. Perona, “Sistemas - Conceptos de control,” *Instituto Politécnico Nacional*, p. 54, 2016. [Online]. Available: <http://dea.unsj.edu.ar/cea1/Sistemas.pdf>

- [52] O. Serrano-Pérez, M. G. Villarreal-Cervantes, J. C. González-Robles, and A. Rodríguez-Molina, "Meta-heuristic algorithms for the control tuning of omnidirectional mobile robots," *Engineering Optimization*, 2019.
- [53] H. Taghavifar and B. Li, "Pso-fuzzy gain scheduling of pid controllers for a nonlinear half-vehicle suspension system," *SAE International Journal of Passenger Cars-Mechanical Systems*, vol. 12, no. 1, pp. 5–21, 2019.
- [54] A. Mewara and G. Parmar, "Comparison of de optimized pid controllers for agc of interconnected power system," in *2017 International Conference on Computer, Communications and Electronics (Comptelix)*. IEEE, 2017, pp. 178–182.
- [55] R. J. M. Ordoñez, "Sintonización de un controlador pid utilizando algoritmos genéticos aplicada a una planta concentradora de cobre," Thesis, Pontificia Universidad Católica del Perú-CENTRUM Católica (Peru), 2021.
- [56] Z. Yachen and H. Yueming, "On pid controllers based on simulated annealing algorithm," in *2008 27th Chinese control conference*. IEEE, 2008, pp. 225–228.
- [57] J. G. Hoyos, J. E. Cardona, and R. Arango, "Control en línea con algoritmos genéticos y recocido simulado," *Scientia et technica*, vol. 1, no. 35, 2007.
- [58] L. F. Fraga-Gonzalez, R. Q. Fuentes-Aguilar, A. Garcia-Gonzalez, and G. Sanchez-Ante, "Adaptive simulated annealing for tuning pid controllers," *AI Communications*, vol. 30, no. 5, pp. 347–362, 2017.
- [59] J. Jantzen, *Foundations of Fuzzy Control: A Practical Approach*. Wiley, 2013. [Online]. Available: <https://books.google.com.ec/books?id=nIEuBjPvjxQC>
- [60] M. Emarah, "Design of gain scheduled fuzzy PID controller for multi-link robot manipulators," *IOP Conference Series: Materials Science and Engineering*, vol. 973, no. 1, 2020.
- [61] I. M. Mehedi, H. S. Shah, U. M. Al-Saggaf, R. Mansouri, and M. Bettayeb, "Fuzzy pid control for respiratory systems," *Journal of Healthcare Engineering*, vol. 2021, 2021.
- [62] R. Wen and M. Tong, "Mecanum wheels with astar algorithm and fuzzy pid algorithm based on genetic algorithm," in *2017 International Conference on Robotics and Automation Sciences (ICRAS)*. IEEE, 2017, pp. 114–118.
- [63] O. A. R. A. Wahhab and A. S. Al-Araji, "Path planning and control strategy design for mobile robot based on hybrid swarm optimization algorithm," *International Journal of Intelligent Engineering and Systems*, vol. 14, no. 3, pp. 565–579, 2021.

- [64] A. Yufka and A. YAZICI, “An intelligent pid tuning method for an autonomous mobile robot,” in *International workshop on Unmanned vehicles*, 2010, pp. 130–133.
- [65] Z. Asani and S. Rexhepi, “Designing a controller via Simulated Annealing,” in *UBT International Conference*, University of Tetova. Macedonia: Faculty of Applied Sciences, 2018, p. 8. [Online]. Available: <https://knowledgecenter.ubt-uni.net/conference/2018/all-events/337/>
- [66] M. Jain, V. Singh, and A. Rani, “Performance Analysis of Multi-objective Genetic Algorithm Tuned PID Controller for Process Control,” *International Journal on Advanced Computer Theory and Engineering*, vol. 1, no. 2, pp. 2319–2526, 2013.
- [67] L. A. Armijos and D. R. Chicaiza, “Diseño e implementación de controladores inteligentes en el control de nivel de un sistema multi-tanque,” Thesis, Escuela Politécnica del Ejército, Sangolquí, 2011. [Online]. Available: <http://repositorio.espe.edu.ec/bitstream/21000/4945/1/T-ESPE-033010.pdf>
- [68] F. I. Robayo, A. M. Barrera, and L. C. Polanco, “Desarrollo de un controlador basado en redes neuronales para un sistema multivariable de nivel y caudal,” *Ingeniería y Región*, pp. 43–54, 2015.
- [69] STMicroelectronics, “STM32 32-bit ARM Cortex MCUs - STMicroelectronics,” 2014. [Online]. Available: <https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html>
- [70] J. Zhang, H. Li, K. Ma, L. Xue, B. Han, Y. Dong, Y. Tan, and C. Gu, “Design of pid temperature control system based on stm32,” *IOP Conference Series: Materials Science and Engineering*, vol. 322, no. 7, p. 072020, 2018.
- [71] J. Yan, W. Chen, and X. Hou, “Adaptive water temperature control system based on stm32,” in *International Conference on Electronic Information Technology (EIT 2022)*, vol. 12254. SPIE, 2022, pp. 175–181.
- [72] B. Zhou and J. Zhang, “Design of dc motor pid control system based on stm32 single chip microcomputer,” *International Core Journal of Engineering*, vol. 6, no. 7, pp. 62–67, 2020.
- [73] X. Cai, S. Cui, L. He, Y. Zhang, Y. Wang, and X. Lv, “Research on automatic ph control system of nutrient solution in smart plant factory based on stm32,” in *2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, vol. 9. IEEE, 2020, pp. 1837–1840.

- [74] V. M. Sandeep, "When and why to use P, PI, PD and PID Controller?" 2021. [Online]. Available: <https://medium.com/@svm161265/when-and-why-to-use-p-pi-pd-and-pid-controller-73729a708bb5>
- [75] M. K. Heris, "Simulated Annealing in MATLAB - Yarpiz," 2015. [Online]. Available: <https://yarpiz.com/223/ypea105-simulated-annealing>
- [76] S. A. Castaño Giraldo, "Todo sobre Ziegler Nichols - Sintonia de Control PID," 2019. [Online]. Available: <https://controlautomaticoeducacion.com/control-realimentado/ziegler-nichols-sintonia-de-control-pid/>
- [77] M. Criollo, "michaelcriollo98/PID-GS-MATLAB: Algoritmos GS-PID en Matlab," Sep. 2022, uRL <https://doi.org/10.5281/zenodo.7084111> and <https://github.com/michaelcriollo98/PID-GS-MATLAB.git>.
- [78] —, "michaelcriollo98/PID-GS-STM32CubeIDE: Algoritmos GS-PID en STM32CubeIDE," Sep. 2022, uRL <https://doi.org/10.5281/zenodo.7084121> and <https://github.com/michaelcriollo98/PID-GS-STM32CubeIDE.git>.

Anexos

Anexo A: Diseño de los tanques, almacén de la planta del sistema multitank y tablero de control

En este anexo, se presentan las láminas técnicas del diseño de la planta del sistema multitank (tanques de reserva y tanque de bombeo); así como el montaje completo del sistema (almacén, tuberías, sensores y actuadores) elaborado por los autores de [5]. Además, se presentan las láminas técnicas del tablero de control construido.

A6
105x148

A5
148x210

A4
210x297

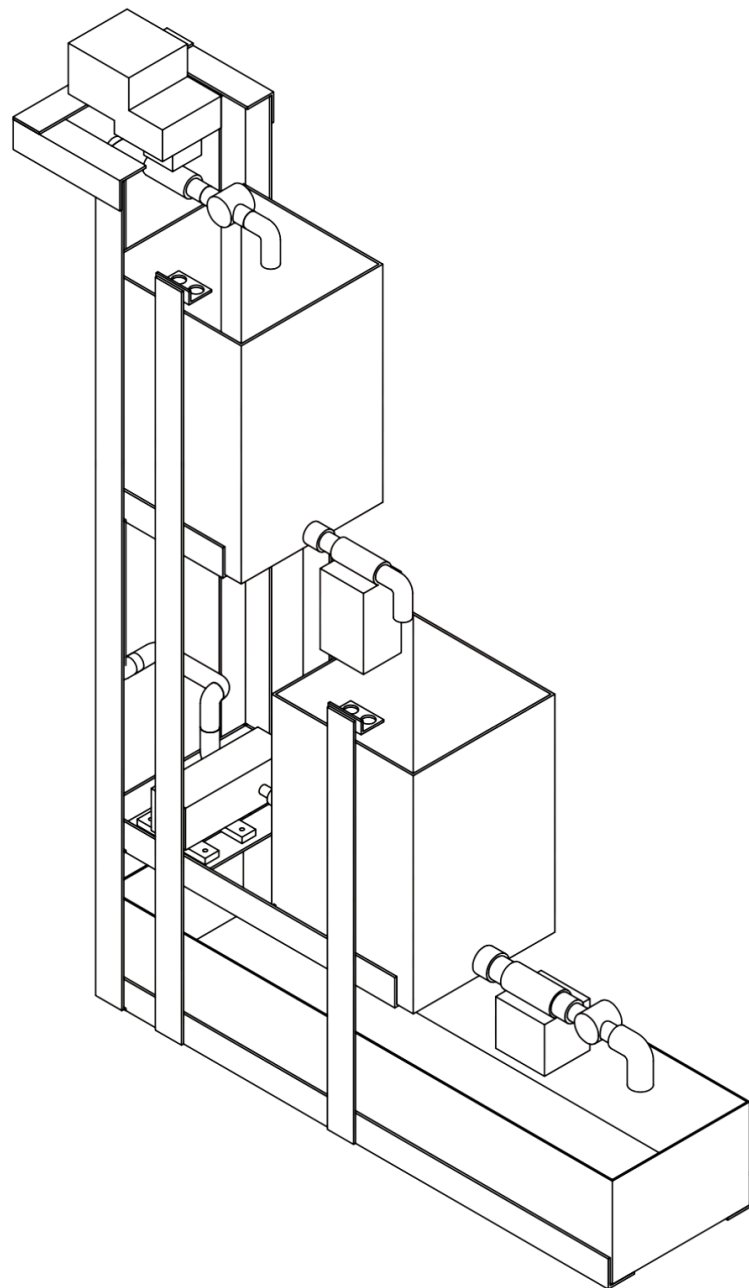
A3
297x420

A2
420x594

A1
594x841

A0
841x1189

Hoja de enseñanza Técnica



	Nombre	Fecha	Escuela de Ingeniería Electrónica y Telecomunicaciones	 UNIVERSIDAD DE CUENCA
Dibujado por:	Ricardo Enderica Fabricio López	25/06/2018		
Proyección	PLANTA DEL SISTEMA MULTITANQUE			Escala: 1:6
				Lámina N° 1

A6
105x148

A5
148x210

A4
210x297

A3
297x420

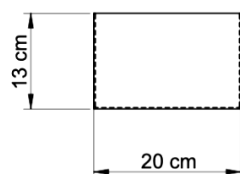
A2
420x594

A1
594x841

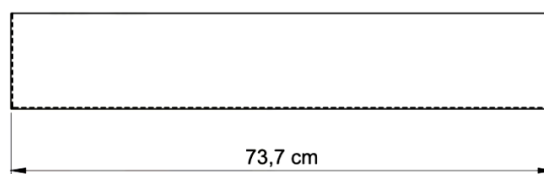
A0
841x1189

Hoja de enseñanza Técnica

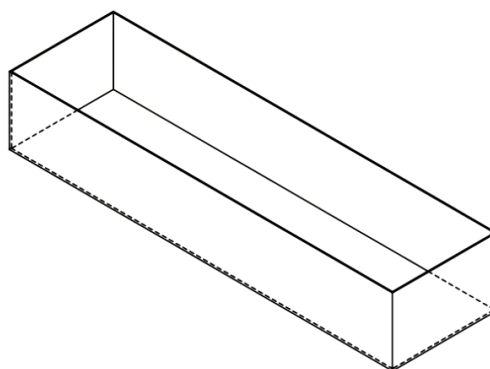
Vista frontal



Vista lateral



Vista isométrica



	Nombre	Fecha	Escuela de Ingeniería Electrónica y Telecomunicaciones	 UNIVERSIDAD DE CUENCA
Dibujado por:	Ricardo Enderica Fabricio López	25/06/2018		
Proyección	TANQUE DE BOMBEO			Escala: 1:8 Lámina N° 4
				

A6
105x148

A5
148x210

A4
210x297

A3
297x420

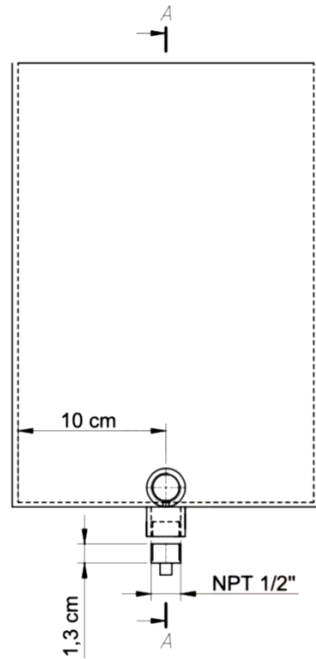
A2
420x594

A1
594x841

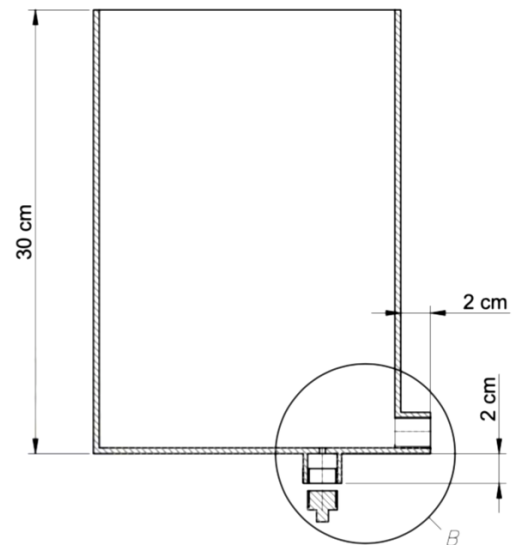
A0
841x1189

Hoja de enseñanza Técnica

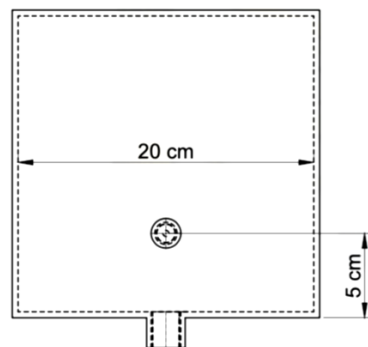
Vista frontal



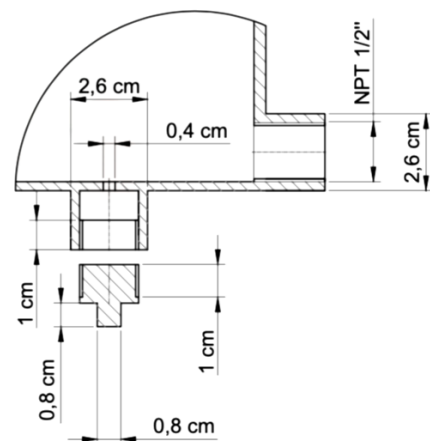
Corte A-A



Vista inferior



B (1:2)



	Nombre	Fecha	Escuela de Ingeniería Electrónica y Telecomunicaciones	 UNIVERSIDAD DE CUENCA
Dibujado por:	Ricardo Enderica Fabricio López	25/06/2018		
Proyección	TANQUE DE RESERVA 1			Escala: 1:4
				Lámina N° 2

A6
105x148

A5
148x210

A4
210x297

A3
297x420

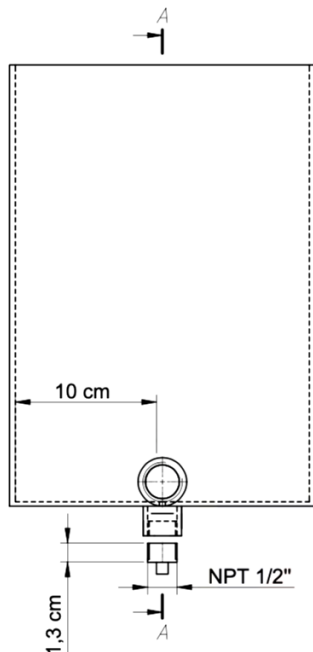
A2
420x594

A1
594x841

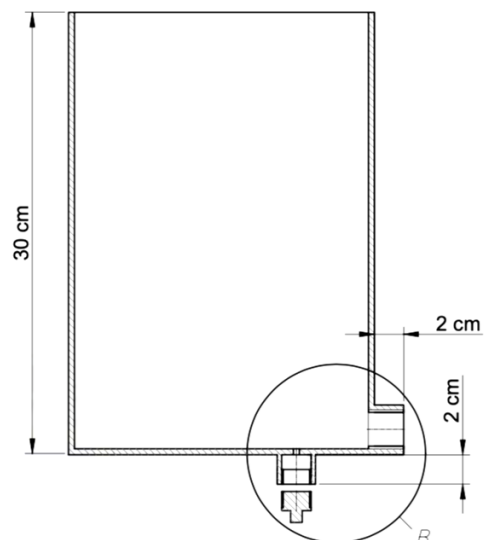
A0
841x1189

Hoja de enseñanza Técnica

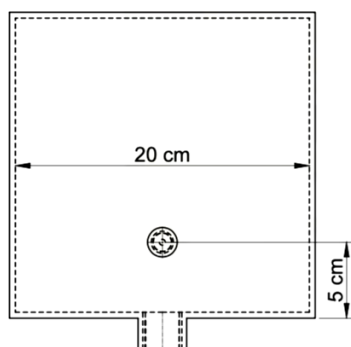
Vista frontal



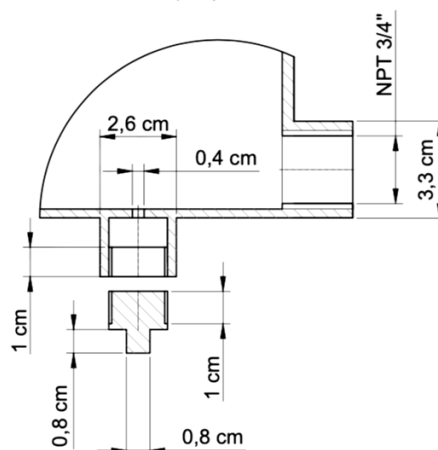
Corte A-A



Vista inferior



B (1:2)



	Nombre	Fecha	Escuela de Ingeniería Electrónica y Telecomunicaciones	 UNIVERSIDAD DE CUENCA
Dibujado por:	Ricardo Enderica Fabricio López	25/06/2018		
Proyección	TANQUE DE RESERVA 2			Escala: 1:4 Lámina N° 3

A6
105x148

A5
148x210

A4
210x297

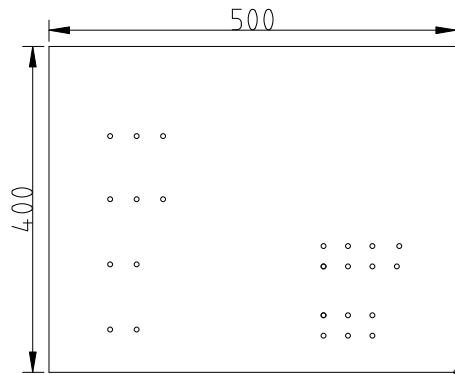
A3
297x420

A2
420x594

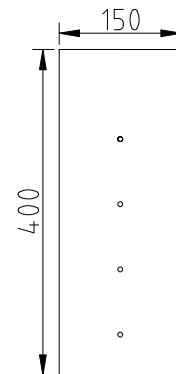
A1
594x841

A0
841x1189

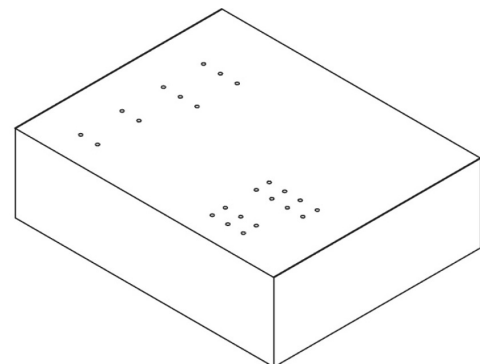
Vista Frontal



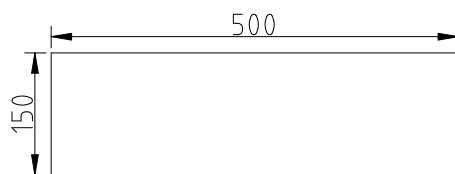
Vista Lateral Izquierda



Vista Isométrica



Vista Inferior y superior



	Nombre:	Fecha:	Escuela de Ingeniería Electrónica y Telecomunicaciones	UCUENCA UNIVERSIDAD DE CUENCA
Dibujado por:	Michael Criollo	17/08/2022		
	TABLERO DE CONTROL			ESCALA: 1:8
				LÁMINA N 1

A6
105x148

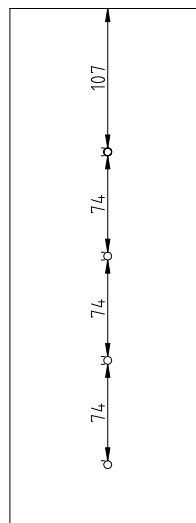
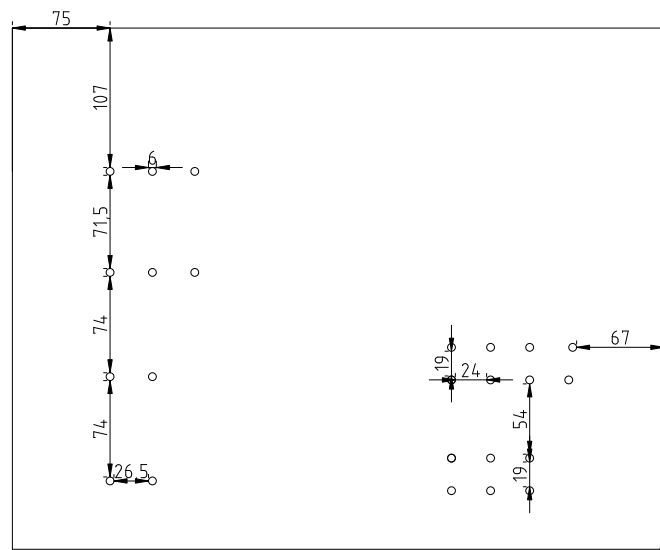
A5
148x210

A3
297x420

A2
420x594

A1
594x841

A0
841x1189



	Nombre:	Fecha:	Escuela de Ingeniería Electrónica y Telecomunicaciones	 UCUENCA UNIVERSIDAD DE CUENCA
Dibujado por:	Michael Criollo	17/08/2022		
	DETALLES DEL TABLERO DE CONTROL			ESCALA: 1:5
				LÁMINA N 2