Análisis del desempeño de un sistema microcontrolado de 32 bits para la implementación de algoritmos de inteligencia artificial para procesamiento de datos

Lizarazo Vesga David Enrique
Ramírez Jiménez Carlos Antonio

Jiménez Moreno Robinson

UNIVERSIDAD MILITAR NUEVA GRANADA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA MECATRONICA
BOGOTÁ D.C.30, AGOSTO DE 2019

Análisis del desempeño de un sistema microcontrolado de 32 bits para la implementación de algoritmos de inteligencia artificial para procesamiento de datos.

Lizarazo Vesga David Enrique

Ramírez Jiménez Carlos Antonio

Jiménez Moreno Robinson

UNIVERSIDAD MILITAR NUEVA GRANADA
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA MECATRONICA
BOGOTÁ D.C.30, AGOSTO DE 2019

Nota aceptación

_____
_____
_____
_____

_____
Firma de tutor

_____
Firma de jurado 1

_____
Firma de jurado 2

Bogotá D. C., agosto de 2019

DEDICATORIA

Este trabajo lo dedicamos a nuestros padres, los principales promotores de nuestros éxitos, quienes con su trabajo y esfuerzo nos han permitido llegar a este punto de nuestras vidas.

A cada uno de nuestros familiares, hermanos, abuelos, quienes en medio de la desesperanza nos dieron las palabras indicadas para seguir y crecer. Por sus consejos y apoyo.

A todos aquellos que nos tendieron su mano, que nos deleitaron con su conocimiento, con su experiencia. A docentes, amigos y compañeros. A esos que conformaron parte de nuestro proceso. A aquellas que creyeron en nosotros más que nosotros mismos, y motivaron cada mala pasada por esta etapa de nuestras vidas.

AGRADECIMIENTOS

Agradecemos a Dios, por ser un punto de apoyo en los momentos en que nadie más estaba alrededor para apoyarnos, y darnos fortaleza en los momentos de debilidad y duda.

Un eterno agradecimiento a nuestros padres: Lizarazo Víctor, Vesga Elvia, Ramírez Miguel y Patricia Jiménez. A nuestra universidad, por brindarnos momentos que nos formaron como futuros profesionales. A aquellos docentes que con las puertas abiertas nos recibieron, e instruyeron.

Contenido

Pág.

Lista de figuras

Pág.

Lista de ecuaciones

Pág.

Lista de tablas

Pág.

1. INTRODUCCIÓN

El presente documento tiene como finalidad exponer la implementación de algoritmos de inteligencia artificial en un microcontrolador de 32 bits y analizar el desempeño de este durante la ejecución de cada algoritmo por separado, para tal caso se empleó el microcontrolador STM32F746ZGT. Los algoritmos escogidos son los siguientes: una red neuronal para la linealización de sensores, un controlador difuso para planta de primer y segundo orden, un clasificador bayesiano para control de un cultivo agrícola y un algoritmo genético para realizar un control PID a una planta de segundo orden.

Para tal fin se desarrolló un compendio de librerías en C, mediante el compilador Keil uVision5, las cuales facilitaron tanto el desarrollo como el análisis de cada uno de los algoritmos ya mencionados. Basando su creación en la posibilidad de ser usado por personas ajenas al proyecto, fueron creadas estas librerías de tal manera que sirvan como guía para otras aplicaciones.

El desempeño de cada algoritmo se determinó por medio de los tiempos que le tomó al microcontrolador realizar una iteración. Añadido a esto se tomaron medidas de consumo energético del microcontrolador con el fin de corroborar si en la ejecución de un algoritmo se presentaba un aumento en la potencia consumida al realizar procesos más complejos o dispendiosos. Proceso mediante el cual fue necesario y beneficioso el crear una estructura reutilizable para la elaboración de circuitos impresos a fin de tener un fácil acceso a los periféricos del microcontrolador propuesto.

El análisis de desempeño del microcontrolador tiene una gran utilidad ya que permite dar una idea respecto a la capacidad en el procesamiento de datos, esto según la implementación de cada algoritmo de inteligencia artificial propuesto. En este caso el enfoque se orientó específicamente en el área de monitoreo y control de sistemas realimentados, ya que es el área en donde se ha logrado desarrollar aplicaciones de inteligencia artificial de corte mecatrónico.

La metodología empleada evalúa el diseño de cada algoritmo en dos etapas, la primera etapa consiste en el diseño del algoritmo en un software especializado como lo es MATLAB, mientras que la segunda etapa expone la programación en la plataforma de programación del microcontrolador, de esta manera se define una metodología generalizada para el diseño, la implementación y evaluación de cada algoritmo.

Se propone el microcontrolador STM32F746ZGT debido a que tiene una buena relación costo beneficio, una alta capacidad de procesamiento (32 bits) y posee una gran cantidad de periféricos que le permiten ser muy versátil a la hora que tener la necesidad de implementar una solución de ingeniería microcontrolada.

1.1 ANTECEDENTES DEL PROBLEMA

Si bien las diferencias entre un microprocesador y un microcontrolador son visibles en su misma arquitectura, se requiere un análisis específico al momento de referirse al desempeño que tiene un algoritmo que funcione en alguno de estos dos dispositivos. Y es que múltiples ventajas y desventajas se hacen notar en precios, tamaños, consumos, entre otros.

Afortunadamente, se han hecho ya comparaciones entre estas dos arquitecturas de desarrollo, (Santos, 2018) pudiendo observar que, si bien el tiempo de ejecución de un microcontrolador es mayor al de el microprocesador, son los microcontroladores los que poseen ventajas al momento de ser implementados en aplicaciones autónomas, o sobre dispositivos de poca capacidad de carga. Esto se evidencia en su tamaño, costo, peso, y versatilidad de conexiones por los periféricos que los microcontroladores presentan.

Las características que presentan los microcontroladores para dar solución a problemas de implementación representan una barrera; ya sea por las diferencias en el entorno de programación para realizar una misma tarea, o por las limitaciones de memoria tanto de programa como de almacenamiento, que poseen. Si bien esta limitante es un caso que puede no llegar a generarse a lo largo de un periodo de aprendizaje con estos dispositivos, si representan una diferencia notable al momento de desarrollar código en aplicaciones de alto nivel o de corte comercial, que requiera de bases de datos de gran magnitud o un proceso de sobre escritura de variables durante un ciclo repetitivo.

Esta limitante se aborda en la metodología de la implementación de los algoritmos que se proponen en este proyecto, debido al desafío que representa implementar algoritmos de inteligencia artificial propios de programación en microprocesador.

Por lo explicado anteriormente es necesario tener en cuenta el espacio de memoria que podrían llegar a ocupar los algoritmos. No obstante, por la naturaleza de cada uno de estos, es de esperar que difieran en gran medida en la asignación de memoria debido a las variables que requieran para su funcionamiento. En general se suelen tener 2 clases diferentes de asignación de espacio de memoria al momento de crear una variable, la definición de variables estática y dinámica.

Con la finalidad de cumplir con el análisis de desempeño de los algoritmos, se realizará una implementación estática de las variables, evitando de esta manera un mayor consumo en tiempos de ejecución de cada uno de los algoritmos al estar establecidas todas las variables desde el primer instante en que se ejecuta el código. Esta estrategia descarta el espacio de memoria de cada algoritmo como parte del análisis, lo cual implica tener una mejor definición de las variables a emplear, evitando el uso y tipos innecesarias de estas.

Otra de las interrogantes al momento de analizar el desempeño de los algoritmos en el microcontrolador, es la variación de consumo energético que puede llegar a representar la puesta en marcha de alguno de ellos, y la manera en que deben ser obtenidos estos valores con el fin de evitar caer en un consumo no necesario debido al procesamiento de algoritmos de alto nivel, como los propuestos. Si bien el uso de sensores, elementos de comunicación serial y visualización como interfaz de usuario (LCD), son elementos que aportan a un mayor consumo, no deben ser tenidos en cuenta en el aporte de estos para el análisis de desempeño de los algoritmos propuestos.

Por último, y a sabiendas de que cada algoritmo se realiza en un proceso cíclico, y que una vez estos alcancen su objetivo, el funcionamiento no se interrumpe a menos que el usuario así lo desee, se plantea el uso de una función propia para la evaluación del consumo del microcontrolador en tiempos, y el desarrollo de un circuito para facilitar el proceso de medición de consumo de este.

Una de las limitantes que presume el uso de un microcontrolador es el del uso de periféricos o elementos externos en el momento de desarrollar cualquier proyecto. Esto dependerá directamente de la documentación del microcontrolador elegido o el acceso a código que ejemplifique el uso de este tipo de elementos. Esta problemática se encuentra presente en algunos de los algoritmos propuestos, ya que, a diferencia de otros microcontroladores, este no presenta librerías o código abierto que pueda ser utilizado para la lectura de sensores, generación de números aleatorios (RNG) o multiplicaciones entre matrices.

Dichos procesos representan un aspecto de vital importancia en el desarrollo de la mayoría de los algoritmos. Razón por la cual se plantea crear librerías propias, e implementarlas de manera intuitiva para facilitar los cambios en los paradigmas de cada algoritmo.

1.2 PLANTEAMIENTO

La ausencia de una estructura funcional e intuitiva, para la creación de proyectos y aplicaciones en microcontroladores de alta capacidad como el que se presentan en este proyecto, es una de las necesidades que se desean suplir a partir de los resultados obtenidos en el proceso de programación y conexiones eléctricas del mismo. Así como la posibilidad de mitigar la preferencia existente en estudiantes de la carrera por microcontroladores genéricos, con menores capacidades y que limitan el desarrollo de nuevas aplicaciones y tecnologías.

Debido al escaso análisis en la implementación de sistemas microcontrolados orientados al desarrollo de proyectos de inteligencia artificial, se presenta una oportunidad para aportar al estado del arte en dicho estudio. Ya que, si bien los microcontroladores son usados comúnmente para aplicaciones de ingeniería y en sistemas mecatrónicos, se convierten en herramientas subestimadas frente a su capacidad de responder de una manera rápida, versátil y eficaz en la solución de múltiples problemas.

1.3 JUSTIFICACIÓN

Los sistemas mecatrónicos requieren de la implementación de arquitecturas microcontroladas de cada vez mayor capacidad, donde la inteligencia artificial cobra mayor relevancia en el desarrollo de soluciones a nivel industrial y de hogar. Derivado de esto y el estado del arte consultado, se expone la implementación de algoritmos de inteligencia artificial en microcontroladores ARM de 32 bits, evaluando su capacidad para soportar este tipo de algoritmos, como una guía al estudiante y al ingeniero en mecatrónica para el desarrollo de dichos sistemas.

A su vez, en el caso de la inteligencia artificial, se ha determinado que al desarrollar algoritmos que sean capaces de aprender como los humanos y determinar un resultado probable (como lo hacen las redes neuronales), se logra aumentar la eficiencia en el desarrollo de aplicaciones mecatrónicas embebidas, como es el caso del reconocimiento de imágenes, control de procesos o analítica de datos. (Mackinnon, 2000)

1.4 DELIMITACIÓN CONCEPTUAL

Para el desarrollo del proyecto se propone el uso de un microcontrolador STM32F746ZGT en el cual se implementarán 4 algoritmos de Inteligencia Artificial siendo estos, una red neuronal, un controlador difuso, un clasificador Bayesiano y un Algoritmo Genético. La programación de cada algoritmo de inteligencia artificial se realiza por medio de la interfaz de programación Keil Uvision5 de libre distribución y uso, debido a que es un software orientado a microcontroladores de la marca ARM (desarrolladora del microcontrolador propuesto).

En la sección de análisis de desempeño se analizará la toma de tiempos que demora cada algoritmo en su ejecución y la potencia consumida respectivamente. Para realizar los procesos de medición de rendimiento de cada uno de los algoritmos, se implementa un timer perteneciente al microcontrolador. Esto se logra mediante 2 funciones propias que funcionan como una marca de inicio y final para obtener el tiempo transcurrido entre el llamado de cada una de estas funciones. Es por esta razón que se desarrollan los algoritmos basados en funciones que pueden ser invocadas independientemente, con el fin de situar el proceso iterativo de cada una de estas entre las marcas de tiempo. Estos datos serán utilizados posteriormente para realizar un análisis estadístico de cada algoritmo, con el fin de determinar la media y varianza de los tiempos según la ecuación 1.

$$S^2 = \frac{\sum(x_i - \bar{x})^2}{n-1}$$

Ecuación 1. Calculo de la varianza

La medición de consumo del microcontrolador se realizará mediante el valor de potencia (w) del microcontrolador. Para tal objetivo se diseña un circuito e interfaz física que facilite el uso de 2 multímetros para medir los valores de corriente y voltaje que emplea el embebido en el proceso de ejecución de cada uno de los algoritmos. Estos datos son tomados en intervalos de dos segundos, correspondiente a las medidas posteriores a una iteración de cada algoritmo. Dicho proceso se aplica tanto para los algoritmos como para la interfaz de espera, donde se realiza una prueba de las funciones o marcas de tiempos, mencionadas anteriormente.

Así mismo se implementó una interfaz de usuario mediante el uso de una LCD con el fin de facilitar la interacción del usuario con cada uno de los algoritmos, de esta manera es posible verificar el funcionamiento, variables de entrada, valores de salida, puesta en marcha y la información respecto al desempeño del microcontrolador durante la ejecución de cada algoritmo.

1.5 OBJETIVOS

1.5.1 OBJETIVO GENERAL

Analizar el desempeño de un sistema microcontrolado de 32 bits para la implementación de al menos 3 algoritmos de inteligencia artificial, destacando el consumo energético y procesamiento como análisis para cada algoritmo.

1.5.2 OBJETIVOS ESPECÍFICOS

- Diseñar e implementar una red Bayesiana para la supervisión de un ambiente controlado. Con el fin de calcular la probabilidad de afectación según las condiciones internas del mismo.

- Diseñar una red neuronal para linealizar sensores no lineales por medio de la obtención de valores censados usando el microcontrolador.

- Diseñar un algoritmo genético para realizar un control de una planta lineal de tipo eléctrico a partir del cálculo del error.

- Analizar costo computacional y análisis estadísticos que permitan evaluar el desempeño del sistema microcontrolado según la implementación de cada algoritmo.

1.6 METODOLOGÍA

La base de programación de los algoritmos de IA se plantea estructurada de tal manera que puedan ser trabajados en paralelo en un código principal, el cual realizara el compendio del desarrollo de códigos independientes en forma de clases que serán invocados y ejecutados por el código principal. Sin embargo, el apartado significativo no es simplemente el modo en que será realizado y ejecutado el código, sino también la manera en que es puesto a prueba.

Es importante destacar el plan de resultados o “tests” los cuales proporcionan información valiosa no solo de avance y resultados sino de posibles errores, y representa la mayor parte de los procesos de verificación y validación en el software a desarrollar. Este aspecto del proyecto es cubierto mediante la metodología de “Pruebas unitarias” y “Pruebas de sistema” (o Unit testings y System testings por su traducción al inglés).

Estas pruebas unitarias son una pieza de código que permite comprobar el funcionamiento de una función o clase, para luego ser implementadas en la totalidad de un proyecto, evitando de esta manera bugs o errores al momento de ser utilizadas. Otras de las ventajas de esta técnica es el ahorro en tiempo de desarrollo al evitar los errores en pruebas al código, tiempo de compilación, y eludir la revalidación de funciones ya comprobadas. También permite que el código pueda ser ejecutado en diferentes momentos con diversas entradas, así como encontrar y enfrentarse a problemas puntuales y finalmente que al momento de desarrollar las pruebas estas sirvan como evidencia y documentación. Siendo esta metodología una forma ya probada de avanzar eficazmente en el desarrollo de un proyecto. (Segue Technologies, 2014)

En la figura 2 se presenta entonces un ejemplo de validación de una clase. Estas clases e incluso sus funciones suelen implementarse como cajas negras al momento de ser invocadas y utilizadas, por lo que encontrar un error en un código o alterar sus salidas al momento de ser testeada fuera de una prueba unitaria, puede resultar difícil y tomar más tiempo del realmente requerido por un programador externo e incluso el mismo desarrollador del código. Siendo el caso de las pruebas unitarias (Diagrama inferior de la figura 2) en donde se realizan pruebas a cada función con sus posibles entradas, esta técnica permite identificar fácilmente los bugs o errores, e incluso probar sus entradas específicas para hacer que fallen, generando de esta manera un código que funcione a prueba de errores en su posterior implementación del proyecto.

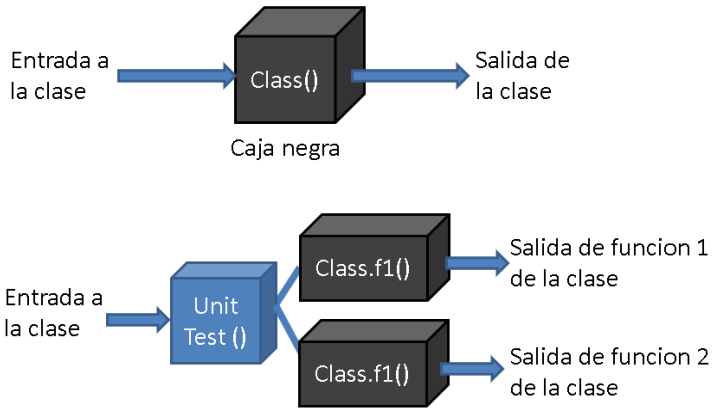


Figura 1.Funcionamiento de pruebas unitarias en la validación de una clase.

La arquitectura de pruebas de un proyecto no solo se concentra en el desarrollo de pruebas unitarias, siendo el caso de la Figura 3 la forma más conocida y eficaz de realizar estas pruebas. En esta se muestra como primer escalón las pruebas unitarias que permiten validar el funcionamiento de una función en solitario o en conjunto con otras funciones que ya hayan sido corroboradas en pruebas anteriores.

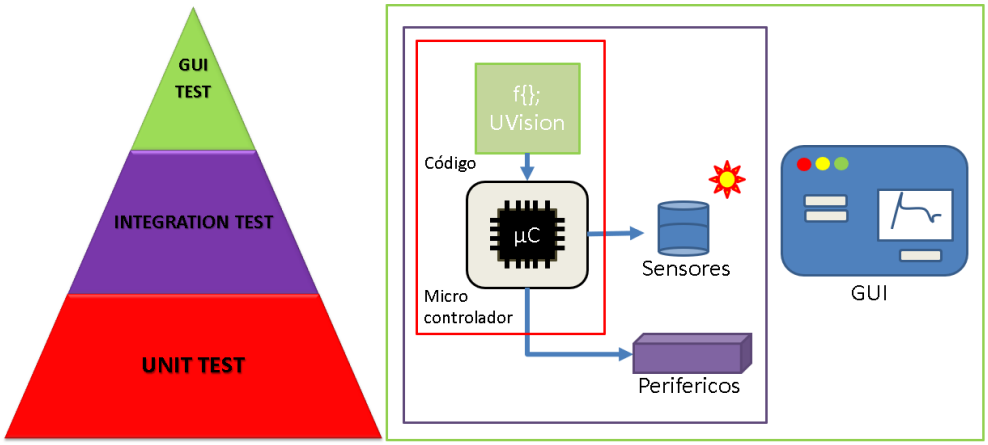


Figura 2. Arquitectura de pruebas de sistema para el desarrollo de un proyecto.

Las pruebas de integración (Integration Tests) componen la siguiente etapa en la arquitectura de pruebas, en donde se realizan pruebas del código implementando posibles agentes externos que responden a las pruebas unitarias. Esta etapa puede llegarse a tratar en ocasiones como una prueba unitaria, ya que algunas condiciones del código no pueden ser ejecutadas por separado de los agentes externos como son sensores o periféricos, sin embargo, sí representa un gran impacto la prueba individual de estos elementos del proyecto, ya que enfocan los esfuerzos en la solución de posibles errores.

Finalmente se establece una prueba de sistema o GUI TEST, donde por medio de una comunicación directa con el usuario se ponen a prueba todos los elementos del

sistema. Estas últimas pruebas al igual que todas las pertenecientes a estas arquitecturas, pueden ser validadas no solo por un desarrollador perteneciente al proyecto, sino que en caso de ser desarrolladas de manera correcta puede realizarlo una persona externa siguiendo los documentos de validación (Anexo ejemplos de pruebas unitarias y pruebas de integración del presente proyecto, para el proceso de comprobación de este). (Runeson, 2006)

Además de las ventajas que tiene la creación de código para la realización de pruebas unitarias en el proyecto, es necesario conocer los problemas que este puede generar para de esta manera no incurrir en ellos en el proceso de creación. Un aspecto importante a tener en cuenta es el caso del código que no se puede replicar sin información de bases de datos o código ajeno al que se desea evaluar (Fowler, 2014), así como otras limitaciones al escribir código que permita comunicar de manera sencilla sus salidas o respuestas en caso de algún error (Mackinnon, 2000); como fue el caso en la visualización y modificaciones de vectores y matrices dentro de una función en el microcontrolador, siendo esto solucionado a partir de variables volátiles, para su monitoreo por medio del debugger del compilador de Keil.

Por esta misma razón se aplica una metodología de Unit testing basada en la comprobación de código en conjunto con otras unidades que componen el proyecto final, combinando así la metodología de pruebas unitarias con las pruebas de integración, a fin de no separar las dos instancias de desarrollo y evitar posibles errores en el salto de una etapa a otra.

Es común que algunas pruebas se realicen en solitario, aislando el código de prueba de unidades pertenecientes al proyecto como se muestra en la figura 4, sin embargo el desarrollo de estas pruebas en conjunto con unidades funcionales y reales del código principal , permite evitar errores en etapas superiores de la prueba de sistema, lo que facilita la reparación de los mismos (Mackinnon, 2000), esto también es implementado de esta manera, debido a la necesidad de probar cada periférico con una misma configuración del microcontrolador, evitando posibles redundancias que puedan afectar el comportamiento de los periféricos , o de las mismas funciones ya establecidas.

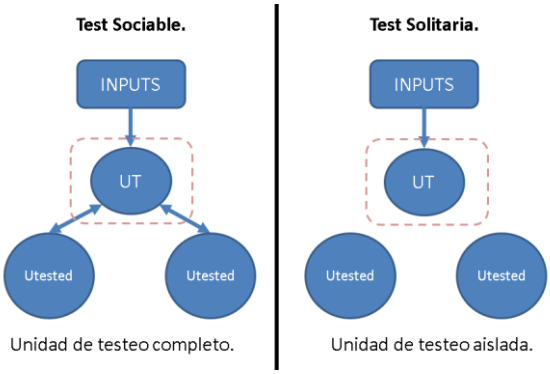


Figura 3. Metodologías de desarrollo de pruebas unitarias.

Esta estructuración del proyecto ha sido probada ya previamente aportando un gran beneficio en el desarrollo de proyectos de implementación de software en arquitecturas de hardware especializadas. (Spanish Small Satellites International Forum, 2019)

Por más de 4 décadas ha sido tema de investigación las diferentes técnicas de testeo para el desarrollo de software (Zhu, 1997).Para el progreso de estas pruebas se utilizará el formato anexo, en el cual se realizan las respectivas pruebas que componen el proyecto; tales como pruebas específicas de conexión e inicialización del proyecto, como pruebas unitarias y de integración de cada uno de los algoritmos de inteligencia artificial, para finalmente obtener una única prueba de sistema que componga todo lo requerido en el presente trabajo y de esta manera entregar un mejor desarrollo práctico en la documentación y ejecución que sirva como bases para proyectos futuros.

Una vez entendida la metodología de pruebas que se implementó, se debe profundizar en el funcionamiento del microcontrolador, y las capacidades que estos poseen. La potencialización de las estructuras tradicionales de los microcontroladores se basa en la integración de diferentes módulos conocidos como periféricos, estos módulos tienen diferentes funciones como convertir señales análogas a digitales, capacidad de realizar conteos o producir señales PWM, también existen módulos que permiten llevar a cabo comunicación serial por medio de diferentes protocolos de comunicación como lo son el I2C, SPI, UART, USART o CAN.

Normalmente las arquitecturas de los microcontroladores varían según la aplicación, por lo que existe una gran variedad de estos. Por esta razón es necesario tener claro que la aplicación determina el microcontrolador y no al contrario. Por consiguiente el microcontrolador fue seleccionado con el fin de aprovechar las velocidades, capacidad de memoria y periféricos con los cuales contaba (RNG,ADC,UART,TIM entre otros). Previo al proceso de programación fue necesario establecer el procedimiento que llevaría a cabo el microcontrolador una vez fueran implementados y probados los algoritmos. A continuación, en la figura 5, se muestra el diagrama de flujo que seguirá la programación en el embebido para ser ejecutado cualquiera de los algoritmos.

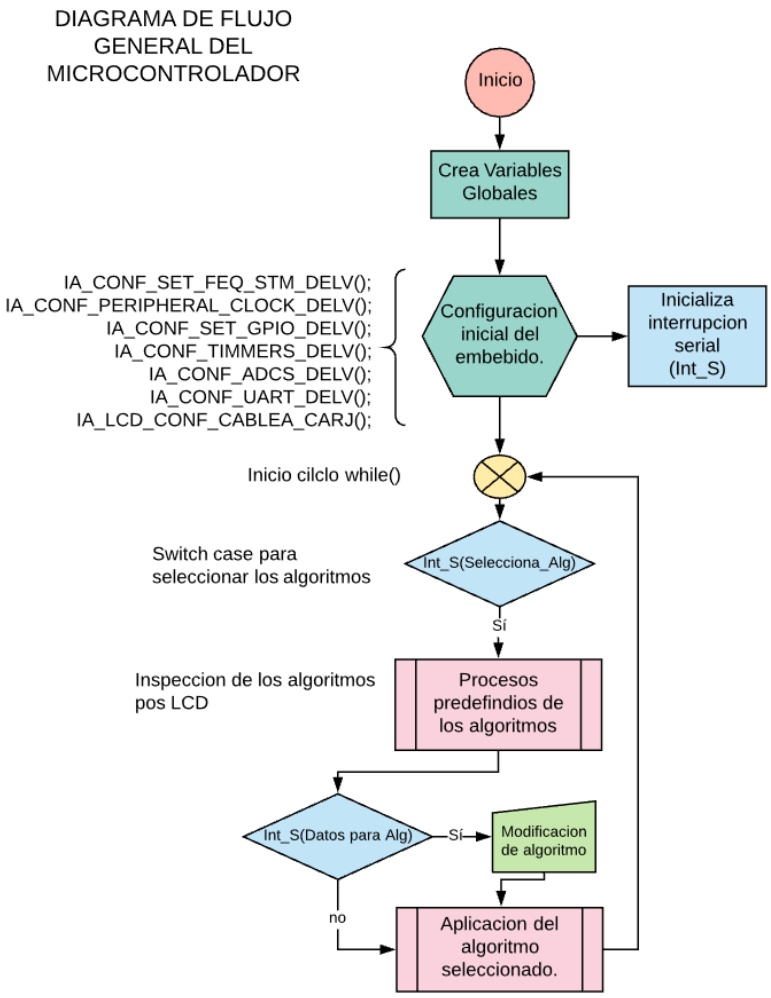


Figura 4. Diagrama de flujo del código principal del microcontrolador.

Siguiendo el diagrama de flujo, se encontraría entonces en la sección de Configuración inicial del embebido, la clave para el desarrollo de los códigos por separado. Dejando allí las funciones estrictamente necesarias y la inicialización de los periféricos del algoritmo que se desea poner a prueba. Igualmente se emplearon periféricos fijos para la prueba de los 4 algoritmos, como es la recepción de información por puerto serial, y la visualización dinámica en una pantalla LCD. Funcionando esta como un debugger de variables de salida de los algoritmos, y tiempos de estos.

Para el funcionamiento de los algoritmos se hace el envío al microcontrolador de la selección del algoritmo, para que de esta manera se puedan ejecutar los primeros comandos para establecer variables, periféricos y demás aspectos que se requieran para el algoritmo seleccionado. Esto se implementó con el fin de permitir un control de los mismo desde una interfaz de usuario externa al embebido, logrado esto a partir de la comunicación serial.

Si bien este microcontrolador cuenta con una amplia gama de puertos multipropósitos de entrada y salida (GPIO), es importante acotar que la distribución de los mismos no es óptima para su implementación. Es por esta razón que se decide realizar una placa de baquelita que permita empalmar de manera sencilla el microcontrolador a un circuito preestablecido con una organización más intuitiva de los pines y periféricos que se desean implementar.

Para el desarrollo de la placa o baquelita, se realizó un “Project Clip” que permitiera realizar las conexiones del microcontrolador de una manera más fácil e intuitiva, dicha creación se realizó mediante el software de diseño y simulación de dispositivos electrónicos, Proteus. Este proyecto cuenta con la respectiva enumeración de los pines, y su distribución física, permitiendo que una vez sean soldados los componentes se pueda insertar un microcontrolador STM32F7 en sus respectivos sockets, sin la necesidad de utilizar cables sobre el microcontrolador.

Sin embargo, y con el fin de implementar un sistema fácil de programación en este microcontrolador, se hizo la adición de cuatro pines conectados por cables, que pertenecen a la selección de alimentación de este, ya que dichas conexiones se encuentran en la parte superior del microcontrolador, y debido a su versatilidad no es estrictamente necesario la conexión de dichos cables, pudiendo reemplazarlos mediante conectores de superficie. A continuación, se encuentra en la figura 6 el “project Clip” mencionado anteriormente.

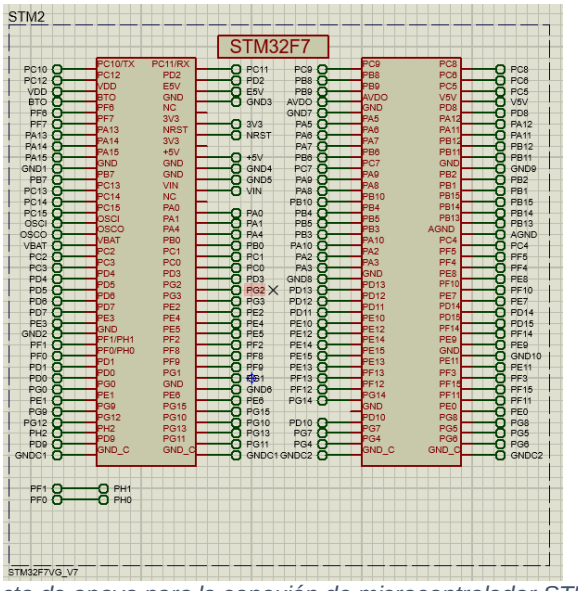


Figura 5. Diseño de un proyecto de apoyo para la conexión de microcontrolador STM32F7 en software Proteus.

Una vez son creados esta clase de proyectos, es posible asignar un diseño PCB que debe ser creado con anterioridad. En este caso y con ayuda de un calibrador se logró realizar esta distribución de pines en un PCB Layout, con la medida óptima entre los mismos para la inserción del microcontrolador una vez soldados los componentes.

Este diseño se puede observar en la figura 7, donde se han agregado las conexiones a los periféricos utilizados para el desarrollo del proyecto. Para establecer las posibles conexiones de periféricos como el UART, los ADC o los pines de alimentación y multipropósito, se recurrió a los manuales con los que cuenta el microcontrolador; al manual general (RM0385 Reference manual) con el fin de comprobar el funcionamiento de registros, tiempos, y demás información relevante al momento de realizar la programación y toma de decisiones con los periféricos, y al manual de las funciones alternantes del microcontrolador (STM32F745xx STM32F746xx) para establecer los pines que se utilizarían.

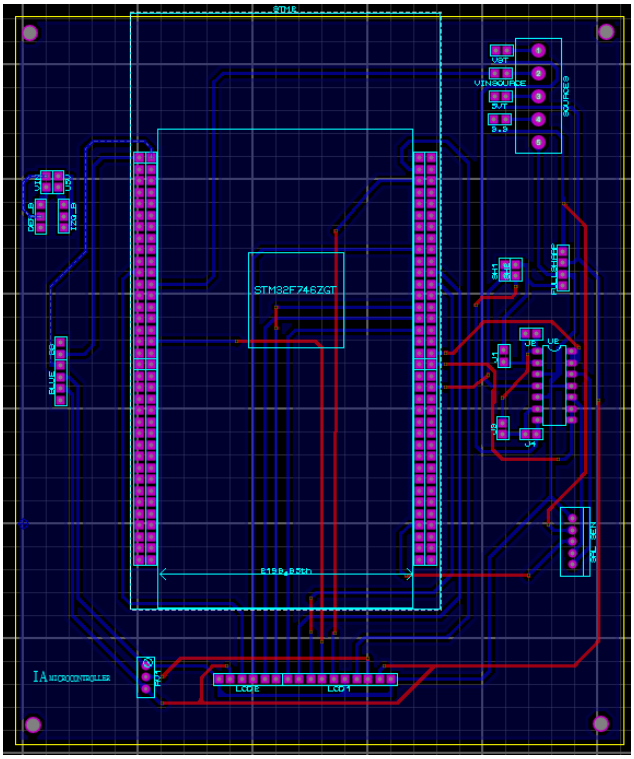


Figura 6. Diseño final de conexiones del PCB para microcontrolador.

Adicional a la creación del PCB (figura 7), Proteus tiene una función extra para visualizar los componentes y las conexiones del mismo. Esto permite dimensionar los componentes y los espacios que estos pueden utilizar como se muestra en la figura 8.

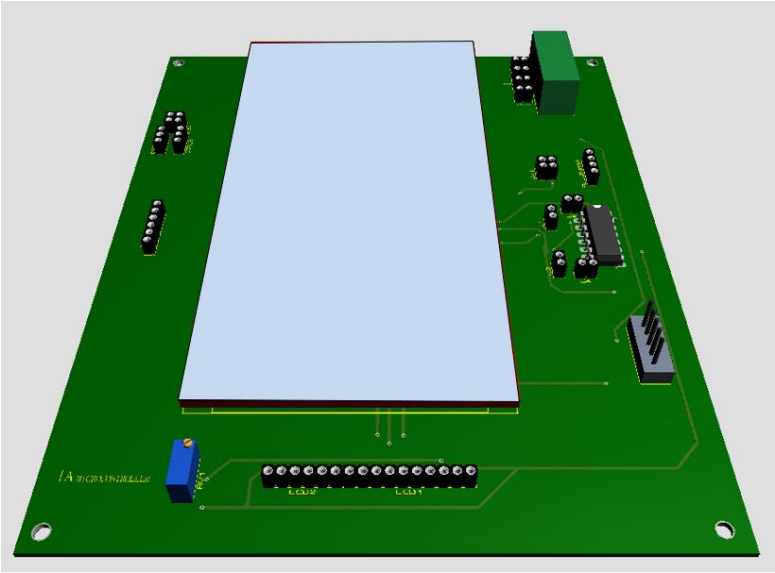


Figura 7. Vista 3D del diseño de la baquela para integración de STM con sus periféricos.

Una vez se realizaron las correcciones de los periféricos que se requerían, se procedió con la elaboración del código. Esto se realiza conociendo las limitaciones y necesidades de cada uno de los algoritmos, las cuales se habían estudiado previamente para el uso de pines del microcontrolador. Estas limitaciones para implementar los algoritmos se enumeran a continuación:

1.Redes Neuronales: Entrada de señal de ADC, etapa de acondicionamiento (atenuación) de una posible señal de 0 a 5v y recepción de datos por serial.
2.Control Difuso: Timmer independiente para procesos iterativos.
3.Clasificador Bayesiano: Acondicionamiento de señales de entrada de sensores, pin multipropósito para ser cambiado de entrada a salida y un timmer independiente para realizar el protocolo de adquisición de datos de un sensor DHT11.
4.Algoritmos Genéticos: Timmer independiente para procesos iterativos.

Dentro de todas las limitaciones que presenta cada algoritmo en términos del microcontrolador, se deben añadir la implementación de múltiples pines GPIO para la visualización dinámica en una LCD, y la configuración de los puertos de comunicación serial UART. Periféricos presentes en cada uno de los algoritmos. Para la visualización dinámica de las variables se empleó una LCD de 20 columnas por 4 filas (20x4), como se muestra en la figura 9.

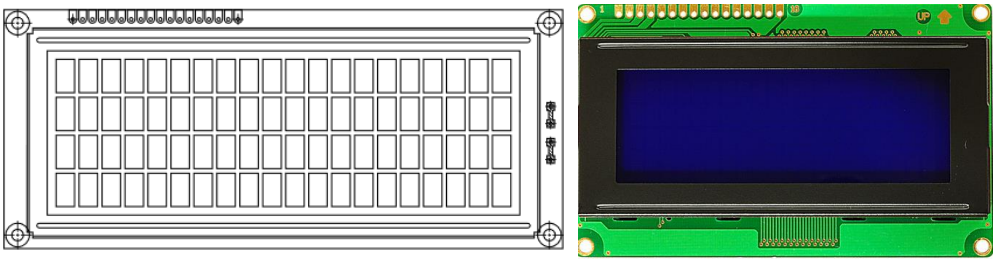


Figura 8. Pantalla LCD 20x4 [24]

La programación para la visualización al igual que la de los diferentes algoritmos y protocolos de lectura de datos, impresión, guardado de información, operaciones matemáticas, y demás, fue implementado mediante la creación de clases. Esto permitió realizar un llamado más fácil a las funciones según el algoritmo que lo requiriera, y siguiendo una estructura de asignación de nombres a los mismos; IA_NOMBRE_PROGRAMADOR.

Las clases creadas paulatinamente, iban siendo agregadas en el encabezado del código principal (figura 10) que seguiría la estructura mostrada en el diagrama de flujo de la figura 6. Si se desea obtener más información del proceso de creación de las mismas y su estructuración, remitirse al anexo de pruebas unitarias (IA_KEIL_CLASES_Unit_Test_Procedure_v5).

```
1  #include "stm32f7xx.h"
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include "ALGORITMOS/CONFIGURACION/CONF.h"
5  #include "ALGORITMOS/ALGORITMO_1/ALG1.h"
6  #include "ALGORITMOS/ALGORITMO_2/ALG2.h"
7  #include "ALGORITMOS/ALGORITMO_3/ALG3.h"
8  #include "ALGORITMOS/ALGORITMO_4/ALG4.h"
9  #include "ALGORITMOS/OPERACIONES/OPE.h"
10 #include "ALGORITMOS/LCD/LCDFUN.h"
11 #include "ALGORITMOS/DHT/DHT11.h"
```

Librerías propias del compilador

Figura 9. Clases creadas para la implementación de los algoritmos de inteligencia artificial en el microcontrolador.

Finalmente, para el proceso de comunicación serial entre el microcontrolador y usuario, se realizó una interfaz utilizando Matlab, que permitiera establecer dicha comunicación. Esta interfaz permite al usuario empezar o detener los procesos llevados a cabo por los algoritmos, así como seleccionarlos o modificarlos según sea posible en cada uno de ellos. Esta versatilidad es mostrada en la figura 11, donde se muestran las diferentes ventanas de la interfaz de usuario.

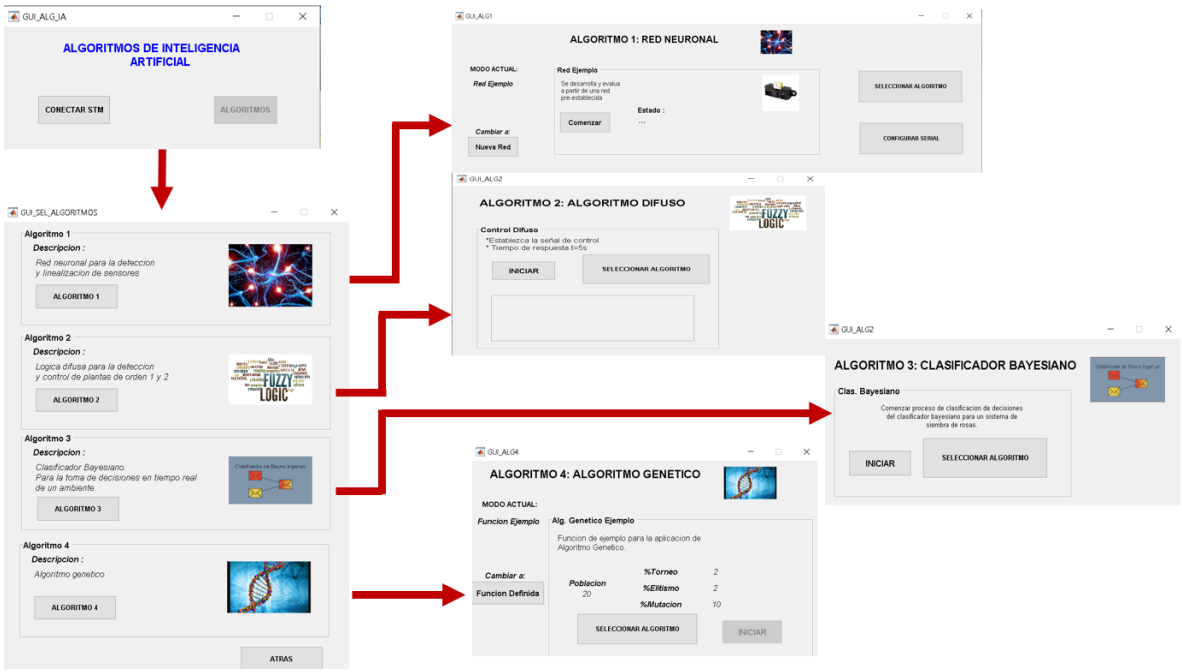


Figura 10. Ventanas de selección de algoritmos de inteligencia artificial para ejecución en el microcontrolador.

Una vez son implementados los algoritmos de inteligencia artificial comandados desde la interfaz gráfica de Matlab, se encuentran listos los algoritmos para realizar su correspondiente análisis de desempeño. Como se puede observar en la figura 11. Por otro lado, como se mostrará más adelante en el primer algoritmo, es posible ejecutar los algoritmos a partir de aplicaciones externas o portátiles de envió serial por bluetooth, siguiendo la estructura de los datos que se utilizó para codificar la información y ordenes en el microcontrolador.

2. MARCO TEÓRICO

La evaluación de desempeño de un microprocesador muchas veces se realiza mediante la comparación de un programa que se implementa en diferentes dispositivos, esto con el fin de obtener resultados de cada uno y luego a partir de la información obtenida determinar que dispositivo es más recomendable para su uso. Un caso particular se realizó desarrollando un microprocesador softcore de 8-bit con doble acumulador que se implementó en varias FPGA's orientado hacia un fácil uso mediante la selección del dispositivo y los pines de entrada y salida. Como resultado se obtienen medidas de una tabla de consulta (LUT's por sus siglas en inglés), Slices y Delays con el fin de determinar que microprocesador es mejor, una vez que se tienen los datos se concluye que el microprocesador evaluado tiene características que representan una ventaja respecto al resto, como lo son una menor cantidad de LUTs. De esta manera se argumenta que el desempeño se mide según ciertos criterios como los LUT's, Slices y delays. (Sáenz, 2018)

A continuación, se presenta el desarrollo de un detector de bordes de imágenes implementada en un microcontrolador STM32FI-DISCO. Se aclaró que el microcontrolador no se encuentra adaptado para el procesamiento digital de imágenes a diferencia de los microcontroladores Texas Instruments o Analog Devices. Con el fin de lograr esta finalidad se integró una cámara para la captura de las fotos, una vez que se aplicaba el algoritmo de detección de bordes era posible observar su funcionamiento mediante una interfaz gráfica de tipo display de película delgada de transistor. El desarrollo dio como resultado una comparación entre los resultados obtenidos desde el microcontrolador y del software Matlab logrando así un buen desempeño del mismo. (Ordoñez, 2016)

Es importante realizar una comparación entre los sistemas embebidos y los computadores debido a que esto permite reconocer las diferencias entre ambos dispositivos para lograr determinar cuándo se usa uno o el otro. En general el uso de microcontroladores se encuentra implementado para sistemas embebidos en donde estos logran realizar las tareas de un PC con la facilidad de incluir diversos periféricos, lo cual reduce el tamaño del sistema a implementar. El uso de un microcontrolador ARM se desarrolla bajo la premisa de que estos poseen una arquitectura de conjunto de instrucciones (ISA, por sus siglas en inglés) lo cual para efectos de implementación permite un mejor trasplante de sistema operativo. (Chen, 2017)

El uso de controles implementados en sistemas embebidos ha tenido un gran impacto en muchas áreas del conocimiento, es el caso de la síntesis de materiales en la creación de revestimientos y películas delgadas, y en la fabricación de semiconductores y dispositivos electrónicos. Para lograr este objetivo se propone la implementación de un control usando un PIC16F877a. Este se encuentra conectado

a un inyector electromecánico y de esta manera logra mejorar la precisión a la hora de iniciar el proceso de depositar el material. (Serrano, 2017)

La figura 1 muestra un diagrama de bloques en donde se indica el funcionamiento del controlador con el inyector para la fabricación de los semiconductores, ya que permite mejorar la precisión a la hora de calibrar el inyector. En conclusión, se determinó que este control sirvió como una alternativa de bajo costo para mejorar la precisión del inyector. (Serrano, 2017)

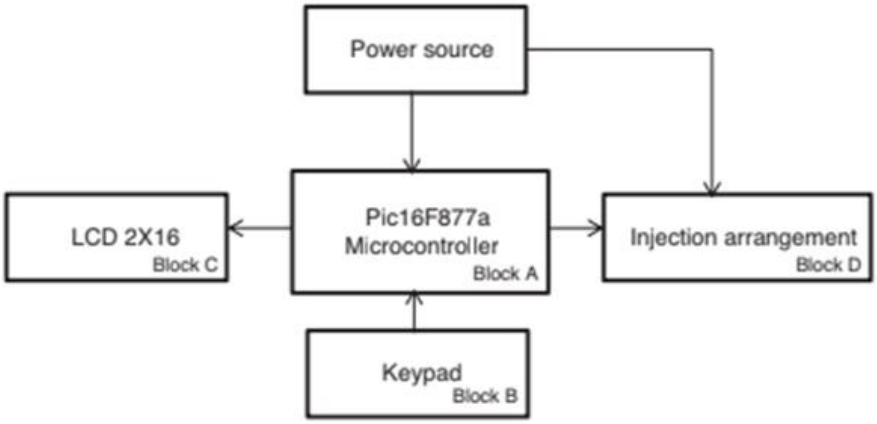


Figura 11. Diagrama de bloques del sistema propuesto

Los microcontroladores tienen una gran diversidad de usos ya que al ser dispositivos programables permite que estos emulan sistemas o en este caso circuitos electrónicos que puedan desempeñarse en tareas complejas. Tal es el caso del uso de un microcontrolador para el control de un motor de paso de un monocromador para llevar la cuenta de los fotones. El uso de un microcontrolador en este sistema permitió el reemplazo de piezas obsoletas debido a que el motor pertenece a un espectrómetro análogo en donde el componente electrónico se encontraba en mal estado. (Giménez, 2014)

Se han tenido avances significativos en el sector agrícola, uno de estos es el monitoreo de un ambiente de aves de corral las cuales requieren un estricto control respecto a las variaciones de temperatura y luminosidad, ya que de no ser monitoreado se afecta severamente la tasa de mortalidad de las aves. Este control requirió de la conexión de varios sensores para el monitoreo del ambiente, de esta manera se lograban obtener los datos, para luego ser enviados a un servidor de Matlab en donde son procesados con el fin de evitar un alto consumo de energía. Finalmente se obtuvo un buen desarrollo del funcionamiento del control ya que esto permitió un aumento en la producción. (Kalezhi, 2018)

Sin embargo, se debe tener presente que existe una gran variedad de microcontroladores soportados en diferentes plataformas como lo son la Raspberry o la Beagle. No obstante, estas plataformas requieren de un conocimiento profundo

o del uso de diferentes drivers cuyo funcionamiento no es sencillo de entender. Por esta razón se decidió usar un microcontrolador ARM, ya que estos se caracterizan por su protocolo de almacenamiento masivo y de esta manera no es necesario el uso de drivers adicionales. Esto argumenta que estos dispositivos permiten una mayor facilidad de uso sin olvidar que poseen características de carácter avanzado para todo tipo de aplicaciones. (Vechet, 2014)

En el área del monitoreo es común ver la implementación de protocolos de comunicación para la transmisión de datos entre servidores inalámbricos, para cumplir con esta tarea se suelen implementar microcontroladores en sistemas embebidos para monitorear y a su vez, enviar los datos recopilados (temperatura, velocidad y batería restante entre otros) a otro servidor con el fin de que este sea capaz de determinar si el sistema se encuentra funcionando dentro de los rangos permitidos. La transmisión de los datos se logra usando el módulo CAN a una velocidad de 9600 baudios por segundo. El uso de microcontroladores para el cumplimiento de dicha labor se ve soportado por su bajo consumo de potencia y precio. (Hock, 2011)

3. REDES NEURONALES.

3.1 Problemática.

A lo largo del tiempo se han definido los algoritmos de redes neuronales de múltiples manera (Aldabas, 2002), recibiendo estas su nombre directamente de su similitud con la estructura que compone el cerebro humano. Las redes neuronales son algoritmos fácilmente implementados en diversos problemas de clasificación, identificación y operación de datos, convirtiéndose hoy día en una herramienta no solo aplicable a dispositivos de cómputo complejos, sino también a dispositivos embebidos y de uso cotidiano. Para el desarrollo del presente trabajo, se implementa en un sistema embebido microcontrolado de 32 bits, que permita evidenciar el funcionamiento de este poderoso algoritmo.

Las similitudes de una red neuronal informática y una biológica radican no solamente en la forma en que estas reciben la información y la envían hacia otras neuronas en forma progresiva hasta alcanzar la salida del sistema, sino en su modo de aprendizaje bajo iteraciones (experiencias vividas por la red), generando así un proceso de aprendizaje y auto mejoramiento de la salida. Este intercambio de información es empleado generalmente para la solución de problemas que no son muy precisos en sistemas computacionales, o en los cuales resulta dispendioso y complejo realizar dicho proceso. Un ejemplo de esto es su aplicación en el reconocimiento de patrones, reconocimiento de imágenes y tendencias en el tiempo entre otras aplicaciones como es el caso del control de sistemas, regresiones no lineales. (Matich, 2001) (Guanuchi, 2017) (Egmont, 2002)

Pese a el poder que esté algoritmos adquiere por sus semejanzas con las estructuras biológicas existen todavía ciertas diferencias que limitan este algoritmo, y es que mientras una neurona biológica realiza procesos mucho más lento que una artificial, se puede tener un número inmenso de neuronas en un cerebro convencional con un número de interconexiones o sinapsis descomunal, las cuales se reorganizan, mueren y nacen a lo largo de nuestra vida, procesos que al sumarse con su gran eficiencia energética superan en funcionamiento a los computadores modernos . (Ocaña, 2009)

El manejo de gran cantidad de información proporciona un esquema completo de las posibles soluciones a las que se enfrenta una red neuronal, además de adquirir la capacidad de predecir y utilizar esta gran cantidad de datos de manera satisfactoria (Wen, 2009). Posibilidad que en caso de otros algoritmos se ve estrictamente limitada a bases de datos más reducidas. Sin embargo, estas características que definen el algoritmo lo hacen poco intuitivo a la hora de proponer la estructura correcta de la red, donde no siempre un mayor número de neuronas significa una mejor respuesta de la misma (Lin, 2010). No obstante, esta gran capacidad de cómputo y manejo de datos de este algoritmo no siempre fue así, fue luego de años de investigación que se lograron dichas características. Ya que

inicialmente carecía de habilidades para el aprendizaje de tareas que otros métodos desarrollaban con normalidad.

Es por todas las limitaciones y la evolución que ha tenido este algoritmo a lo largo del tiempo que se pretende mediante este trabajo el demostrar su implementación en sistemas embebidos de una manera organizada y generalizada, permitiendo al microcontrolador hacer uso de la estructura de una red preestablecida o una nueva definida por un usuario, pudiendo ser tan sencilla o compleja como este lo desee, para establecer una salida ante una entrada de un sensor análogo.

3.2 Características del algoritmo

Al igual que sus predecesoras, las redes neuronales artificiales son creadas a partir de una estructura sencilla, la neurona. Estas células cerebrales por sí solas pueden no tener una aplicabilidad relevante para procesar información, sin embargo, la conexión o sinapsis entre estas adquieren la capacidad de resolver problemas cada vez más complejos. Solo hace falta observar los billones de neuronas que poseemos, las cuales se encargan del tratamiento de toda la información, estímulos y decisiones que se realizan en el día a día.

Es basada en esa similitud que se plantean también las mismas neuronas artificiales, tal como se observa en la figura 12, estas comparten los rasgos más importantes que necesitan para su funcionamiento, una sección de entrada por la cual ingresa la información a la neurona, un centro de procesamiento de esta información o soma, y las salidas.

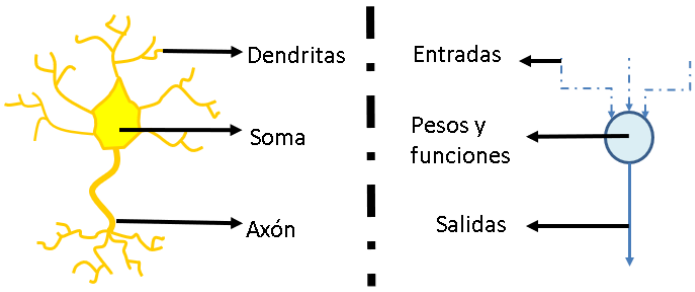


Figura 12. Comparación de las partes principales entre una neurona biológica) (izquierda) y una artificial (derecha).

Como se puede observar en la figura 12, cada neurona puede contar con múltiples entradas. Estas entradas en la neurona artificial siguen un debido proceso para entregar un valor de salida.

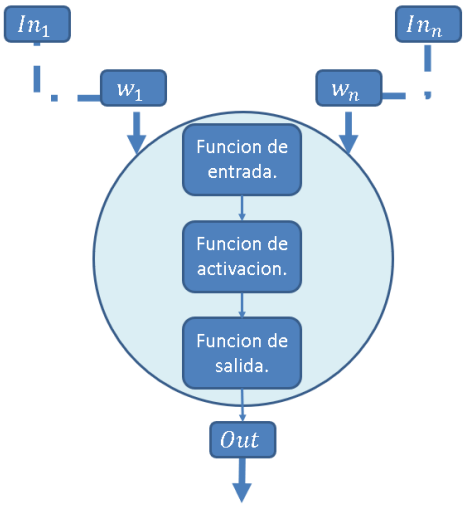


Figura 13. Estructura estándar de una neurona artificial.

Como se observa en la figura 13 y como se introdujo anteriormente, la entrada de una neurona pasa por un proceso secuencial para obtener una salida, este proceso se enumera a continuación.

1. Se obtiene el producto del valor de entrada (in_n) por el respectivo peso de la dendrita que lo transportaba (w_n).
2. Se hace uso de la función de entrada con los valores recibidos. Existen diferentes tipos de funciones de entrada, pero para nuestro caso en particular se utilizará la sumatoria del producto de las entradas por su respectivo peso. Representado por la siguiente ecuación.

$$\sum_{i=0}^{n} (\boldsymbol{n_{ij} w_{ij}}), \boldsymbol{para\ j = 1, 2, \ldots n\ = g_{in}}$$

Ecuación 2. Sumatoria de productos de entrada a una neurona.

3. La salida de la función de entrada pasa entonces a la función de activación. Esta determina el estado de activación de una neurona (si se encuentra activa o inactiva) sin embargo, estas pueden llegar a tomar un valor dentro de un rango dado según la función de activación. Esta funciona calculando la entrada global (g_{in}) menos un umbral de activación(θ_1).

   Si bien también existen varios tipos de funciones de activación (Función lineal, función sigmoidea, y función tangente hiperbólica) se ha decidido en utilizar la función tangente hiperbólica por el rango de valores que este puede tomar (-1 a 1), y por su comportamiento asintótico en estos puntos. La ecuación que describe esta función de activación es la que se muestra en la ecuación 3.

$$f(x) = \frac{e^{\alpha x} - e^{-\alpha x}}{e^{\alpha x} + e^{-\alpha x}}$$

Ecuación 3. Función de activación de las neuronas.

La ecuación 3 es el equivalente entonces a la función de activación tangente hiperbólica, donde x representa la entrada a esta función (g_{in} - θ_1). Y α representa la pendiente de la función de tal forma que la neurona sea más sensible a ser activada o no, o que por el contrario pueda tomar más valores intermedios. Para aclarar esta distinción entre las variables, se graficaron 3 diferentes funciones de activación con diferente valor de α , y por último una gráfica de la función tangente hiperbólica que tiene por defecto el programa Matlab, como se observa en la figura 14.

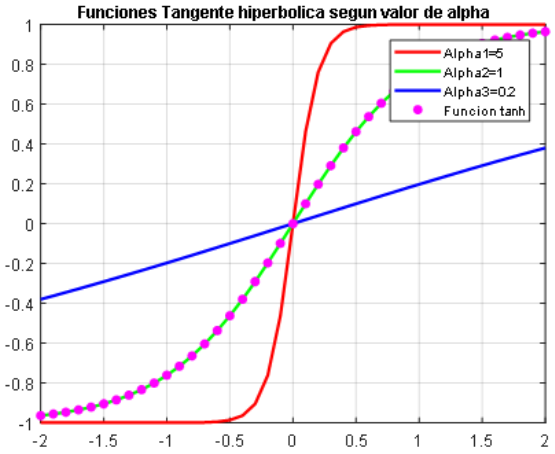


Figura 14. Funciones de activación tangente hiperbólica para diferentes valores de coeficiente Alpha.

4. Por último, la salida de la función de activación se convierte en la entrada a la función de salida la cual entregará el valor final de la neurona (out_i) que será transmitido a la siguiente. Por ende, la salida no puede tomar cualquier valor, pues representa la entrada a una nueva neurona, para las cuales se tiene un rango de valores de entrada permitido. Esta función puede ser binaria o no ser ninguna función.

   En el caso de la **binaria** es utilizada para la activación o desactivación completa de la neurona, estableciendo así un rango de valores en los cuales se activará o no la misma (0 o 1, -1 o 1, según el rango de entrada establecido). Sin embargo y con el fin de aprovechar la función de activación elegida, no se realizará ninguna función de salida, de tal manera que la salida de la función de activación se convierte en la salida de la neurona.

Una vez entendidas las características de una neurona, es posible adentrarse en la estructura general de una red neuronal, la cual suele estar constituida por 3 capas generales: Una primera capa de entrada de datos, la segunda siendo una capa

oculta (Donde se pueden tener múltiples capas), y la tercer capa o capa de salida en donde se tienen la respuesta final de la red. Dicha configuración se observa en la figura 15, donde se puede ver la estructura de la misma. (Matich, 2001)

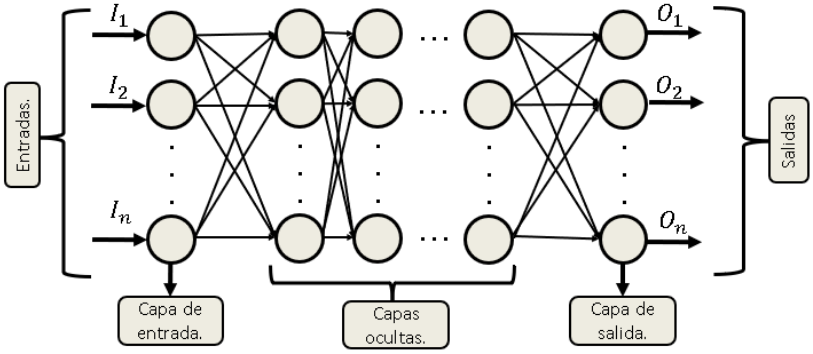


Figura 15. Configuración de una red neuronal.

Como se observa en la figura 15, una vez se obtiene la entrada en la primera capa la información se propaga paulatinamente hacia adelante. Este medio de conexión y propagación de la información es conocido como MLP o "MultiLayer Perceptron". (Aldabas, 2002) (Sandoval, 2016)

Sin embargo, esta no es la única arquitectura que existe, ya que si bien es la más común es pertenece a la siguiente clasificación, las redes de pre-alimentación, y las redes recurrentes (RNN). Las redes de pre-alimentación si bien fueron las primeras en ser estudiadas no pudieron abarcar problemas que requerían un conocimiento del estado temporal por parte de la red, razón por la cual surgen las redes recurrentes; en este tipo de redes la salida de una neurona se puede convertir en la entrada de sí misma o de otras neuronas de su propia capa o nivel, lo que hace posible tratar secuencias temporales o secuencias lógicas (como es el caso de los predictores de texto) (Perez, 2002) (Cruz, 2007) (Martin, 2003).

Las redes de pre-alimentación por otro lado adoptaron el nombre de perceptrón con el paso de los años, y que por su comportamiento también fueron conocidas como redes unidireccionales (RNU) o Feedforward neural networks en inglés (FFNN). Este tipo de red neuronal también puede dividirse en redes neuronales profundas (DNN) y las redes convolucionales (CNN), las cuales comparten su arquitectura, siendo las convolucionales más usadas para el tratamiento de imágenes por su poder de clasificación de características de las entradas, y menor costo computacional al no conectar cada neurona de una capa con las de sus capas adyacentes. (Martin, 2003) (Sánchez, 2016)

Este último tipo de red neuronal será el utilizado finalmente para el seguimiento de sensores y funciones no lineales, teniendo igualmente un gran potencial en la detección de sistemas. Esta red en particular y como se mencionó anteriormente, podrá ser modificada a gusto del usuario para definir una salida ante una entrada análoga. (Pacheco, 2017)

3.3 Antecedentes del algoritmo

El primer acercamiento a las redes neuronales artificiales como las conocemos hoy día nace en 1943 cuando se modela por primera vez una red neuronal a partir de circuitos eléctricos. 3 años después Donald Hebb sería el encargado de explicar psicológicamente como se da el aprendizaje, generando cambios en las neuronas. (Matich, 2001)

Uno de los eventos más significativos es el Congreso de Dartmouth en donde se propone establecer como nombre de inteligencia artificial, a todo el desarrollo e investigaciones que se realizarán para replicar la mente humana artificialmente (Chávez, 2012). Un año después es cuando Frank Rosenblatt comienza a desarrollar los primeros ápices en redes neuronales (el perceptrón), que, si bien presentaba un avance novedoso en esta nueva ciencia, también poseía sus inconvenientes, más específicamente con las no linealidades de los problemas planteados. A Pesar de esto, en 1960 se aplica a un problema real el algoritmo de una red neuronal, los filtros para la eliminación de eco en líneas telefónicas, utilizado por décadas hasta ahora. (Matich, 2001)

La muerte de las redes neuronales fue casi producida en 1969 cuando Marvin Minsy y Seymour Papert demostraron matemáticamente las limitaciones que presentaba el perceptrón frente a problemas reales solucionados ya por otras herramientas. Tuvieron que pasar 5 años para el desarrollo de la idea que impulsó nuevamente este algoritmo a la luz, y un año más para madurar el cambio que haría de esta técnica un avance suficiente para lo que es hoy día, con el redescubrimiento de la técnica de entrenamiento por propagación hacia atrás o back propagation en inglés. (Matich, 2001)

Si bien esos fueron los cambios más relevantes que permitieron que estos algoritmos prosperaran, avances en áreas de implementación en sistemas embebidos, como sistemas de alimentación híbridos, reconocimientos de camino y visión de máquina, son los precedentes que marcaron relevancia para el desarrollo del presente proyecto, siendo de gran inspiración para este y los demás algoritmos que lo componen. (Ramos, 2016)

3.4 Entrenamiento

Para el desarrollo del entrenamiento de este tipo de algoritmos es necesario un nivel de computo elevado, esto claramente afectado por la forma en que se desarrolle el entrenamiento, y todos los parámetros que se establezcan. Sin embargo y con el fin de realizar una comparación de los algoritmos presentados en el presente trabajo y no de su entrenamiento, se optó por realizar el entrenamiento de la red neuronal fuera del microcontrolador propuesto, esto mediante la herramienta Matlab. De esta manera y como ha sido probado, se puede realizar el entrenamiento y cálculo de

los pesos que caracterizan a la red neuronal con su respectivo error, para luego ser enviados al microcontrolador.

Para el entrenamiento de la red neuronal que se aplicará en el sistema embebido se realizó un entrenamiento bajo supervisión. En este tipo de entrenamiento es necesario definir los valores de salida, lo que permite el cambio de los pesos de la red para cumplir con la función objetivo. El algoritmo que se usará para lograr esto, es el de retro propagación o backpropagation. Este se encarga de revisar el error a la salida de la red, y a partir de este reajustar los pesos de esta hasta alcanzar un error que se encuentre dentro de los rangos permitidos. Esto explicado a grandes rasgos (Sandoval, 2016).

El proceso que se llevará a cabo para implementar las redes neuronales será:

1. Entrenar una red bajo de ejemplo con parámetros preestablecidos utilizando Matlab.
2. Crear una red neuronal de ejemplo basado en el punto anterior, que funcionara bajo ciertas especificaciones en el embebido.
3. Mediante la interfaz gráfica permitir al usuario utilizar la red de ejemplo creada o crear y entrenar una nueva red, de una entrada y una salida (SISO) para ser ejecutada en el embebido.

Entrenamiento de red de ejemplo:

Para crear la red es necesario primero establecer los parámetros de entrada y salida de esta. Para esto se realizó un proceso de caracterización de un sensor, ya que como se explicó anteriormente el objetivo de la misma será el de realizar un proceso de seguimiento a una función no lineal. Para esto se decide utilizar un sensor de infrarrojo Sharp, el cual tiene un comportamiento no lineal como se muestra en figura 16.

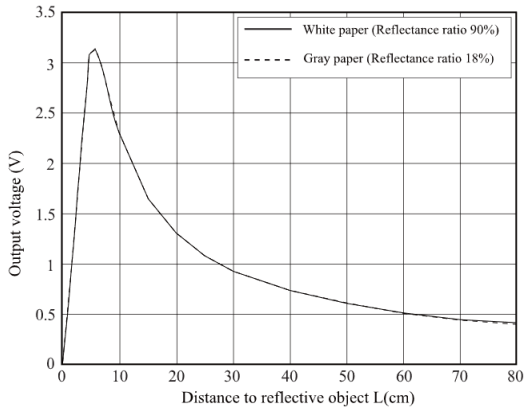


Figura 16. Función de distancia contra voltaje de un sensor infrarrojo SHARP. [Tomado de Datasheet]

El proceso de caracterización se guarda en una hoja de datos de Excel, con el fin de poder ser utilizados con facilidad por un script de Matlab. Este proceso de caracterización fue realizado en 6 momentos, a lo largo de un día, con el fin de tener condiciones ambientales diferentes para cada muestra. Se establece entonces una tabla de distancia contra voltaje, donde para efectos de la red neuronal, la entrada será el voltaje y la salida deseada la distancia. Una vez son guardados los valores se realiza la adquisición de datos del sensor como se muestra en la figura 17, donde se ilustran las líneas de código empleadas para importar los datos a Matlab.

```
%% Declaración de la entrada en valores reales.
Axls_1=xlsread('Caracterizacion_SHARP','Muestra_2');
Axls_2=xlsread('Caracterizacion_SHARP','Muestra_3');
Axls_3=xlsread('Caracterizacion_SHARP','Muestra_4');
Axls_4=xlsread('Caracterizacion_SHARP','Muestra_5');
Axls_5=xlsread('Caracterizacion_SHARP','Muestra_6');
X_Total = [Axls_1(:,2),Axls_2(:,2),Axls_3(:,2),Axls_4(:,2),Axls_5(:,2)];
```

Figura 17. Código para definir valores de entrada y salida de la red.

Posterior a la asignación de valores según el proceso de caracterización, se crean los pesos arbitrarios de la red. Siendo para este caso necesario solo la creación de cuatro vectores, en los cuales se tendrán los pesos que corresponden a 2 capas dentro de la capa oculta.

Como se observa en la figura 18, existen dos vectores que contienen los pesos propios de cada neurona (W_neurona) según las capas ocultas establecidas y dos vectores que contienen los pesos de la conexión con la siguiente capa de neuronas (W_dendritas). Este proceso se debe aplicar a cada capa de neuronas que se tenga en la red, puesto que las neuronas les corresponden sus pesos propios y los pesos con que conectan a la siguiente capa.

```
%% Asignacion de Valores de Pesos
W_Dendrita_1 = (-2.4/Num_Pri_Neu)    +((2.4+2.4)/Num_Pri_Neu)     *rand(Num_Neuronas,    1);
W_Dendrita_2 = (-2.4/Num_Neuronas)   +((2.4+2.4)/Num_Neuronas)    *rand(1,   Num_Neuronas);
W_Neurona_1  = (-2.4/2)              +((2.4+2.4)/2)               *rand(Num_Neuronas,     1);
W_Neurona_2  = (-2.4/2)              +((2.4+2.4)/2)               *rand(Num_Sal_Seg_Neu,  1);
```

Figura 18. Código de creación de pesos aleatorios de la red neuronal.

Posterior a esto se crean las variables previas al entrenamiento como es el error deseado, dicho valor definirá el momento en que el entrenamiento sea finalizado. También se crea la variable que se usará para comparar si la red a alcanzado la meta, la variable Alpha para los cambios de los pesos de la red y un contador de la cantidad de iteraciones realizadas, esto se muestra en la figura 19.

```
%% Declaracion de variables para el aprendizaje
Error_Deseado   = 0.005;
Error_Calculado = 1;
alfa = 0.01;
m = 1; %Contador de iteraciones
```

Figura 19. Creación de variables de aprendizaje.

Seguido de esto se comienza con el algoritmo de aprendizaje, el cual se encuentra dentro de un ciclo que garantiza que el entrenamiento no termine hasta alcanzar el error objetivo (figura 20). Utilizando un comando extra en Matlab (tic y toc) para medir el tiempo de ejecución.

```
%% Aprendizaje
tic; %Funcion para visualizar tiempo de aprendizaje
while Error_Calculado>Error_Deseado
```

Figura 20. Condición para el ciclo de aprendizaje de la red neuronal.

Dentro del bucle de entrenamiento, se realiza primero la propagación hacia adelante de la información. Posteriormente se calcula el error respecto a la capa de salida. Una vez realizado este proceso es posible comenzar con la modificación de la red a través de los valores de sus pesos. Esto se logra calculando la salida de las neuronas de la capa de salida con el valor deseado que deben tener cada una de ellas, luego se calcula la sensibilidad de la capa de salida. Esta sensibilidad permite realizar la modificación de los pesos que conectan con la capa mencionada.

Para realizar el proceso de retro propagación, se realizó el cálculo de la sensibilidad de cada una de las capas que compone la red. Esto permite hacer la respectiva modificación de los pesos según la estructura de la red. La sensibilidad de la primera capa se calcula a través de la ecuación 4**.**

$$Sens_{Primercapa} = diag(1 - Out^2) * Peso_{Dend} * Sens_{CapaSalida}$$

Ecuación 4. Sensibilidad de salida de la primera capa.

Este proceso debe entonces realizarse también para la salida de la siguiente capa de la red. Por lo que la ecuación que la define está dada por la ecuación 5**.**

$$Senss_{CapaSalida} = diag(1 - Out^2) * Err$$

Ecuación 5. Sensibilidad de salida de la capa de salida.

Una vez que se han modificado los pesos que conectan con la capa de salida se calcula la sensibilidad de la capa anterior y así se puede calcular la sensibilidad de las neuronas de esta misma capa. Este proceso se realiza de manera repetitiva hasta modificar todos los pesos de la red, una vez que se hace este proceso se vuelve a ingresar la entrada y la red vuelve a calcular la salida para comparar con

el valor deseado hasta que el error se encuentre dentro de los rangos permitidos como se muestra en la figura 21.

```
%Salida de la primera capa
y1 = tanh((W_Dendrita_1*X_Total(k,q))+W_Neurona_1);
%Salida segunda capa
y2 = tanh((W_Dendrita_2*y1)+W_Neurona_2);

%Error
e = Y(k,q)-y2;

%Sensibilidad de la capa de salida
sig2 = diag(1-y2.^2).*e;
sig  = diag(1-y1.^2)*W_Dendrita_2'*sig2;

%Cambio en los pesos, back propagation
W_Dendrita_2 = W_Dendrita_2 + (alfa.*(sig2.*y1'));
W_Dendrita_1 = W_Dendrita_1 + (alfa.*(sig*X_Total(k,q)));
W_Neurona_2  = W_Neurona_2  + (alfa*sig2);
W_Neurona_1  = W_Neurona_1  + (alfa*sig);
ei(k,q) = (e'*e);
```

Figura 21. Algoritmo de entrenamiento de la red.

Una vez se alcanza el error deseado, se procede con la generación de 4 archivos. Estos se encargarán de contener los valores de los pesos generados para el entrenamiento realizado (figura 22). De esta manera se puede ejecutar el script y tener guardados los valores para su posterior uso.

```
%% Genera archivos de excell para ser leidos posteriormente por el codigo.
xlswrite('W_Dendritas_1.csv',W_Dendrita_1);
xlswrite('W_Dendritas_2.csv',W_Dendrita_2);
xlswrite('W_Neuronas_1.csv',W_Neurona_1);
xlswrite('W_Neuronas_2.csv',W_Neurona_2);
```

Figura 22. Código para la generación de archivos CSV a partir de los pesos calculados.

También se realiza una prueba de la red, a partir de un valor arbitrario de entrada. Como se muestra en la figura 23, se hace con el penúltimo dato, el cual correspondía a un valor en distancia de 45 cm.

```
%%  Prueba o Test
x =X_Total(10,1);
%Salida de la primera capa

y12 = tanh((W_Dendrita_1*x)+W_Neurona_1);
%Salida segunda capa
y22 = tanh((W_Dendrita_2*y12)+W_Neurona_2);
y22 = interp1([-1,1],[0,50],y22)
```

```
Error_Calculado =

    0.0050

Elapsed time is 1031.879286 seconds.

y22 =

   44.8926
```

Figura 23. Código para realizar prueba de los pesos calculados.

Como se puede observar en la figura 23, una vez el código alcanza el error deseado se muestra el tiempo transcurrido en el proceso de entrenamiento. Este tiempo puede variar según el ordenador que se emplee para dicha ejecución. Siendo en este caso un aproximado de 15 minutos, entregando finalmente el valor de salida de la red, 44.8926 correspondiendo a una salida deseada de 45 cm. Debido a que el error de la red no se calcula a partir del valor individual de la misma, sino de la totalidad de datos, se observa un valor de error mayor al deseado, sin embargo, satisface las condiciones para la implementación del sensor.

Implementación de la de red de ejemplo en el microcontrolador:

Para el caso de los algoritmos de redes neuronales se tiene entonces el siguiente diagrama de flujo para los “Procesos predefinidos” del algoritmo (sección de la figura 6):

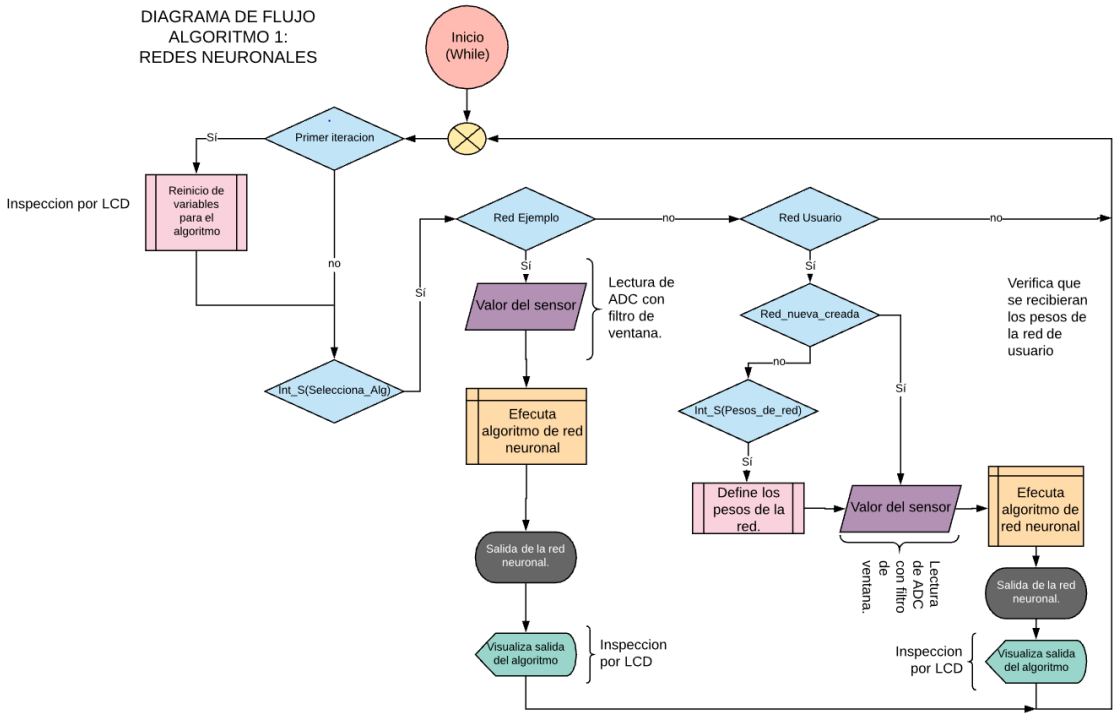


Figura 24. Diagrama de flujo del algoritmo de redes neuronales en un microcontrolador.

Una vez se ha definido el diagrama de flujo como se observa en la figura 24, es posible entender el proceso que se realiza en el código del microcontrolador.

En la figura 25, se tienen 3 condiciones posibles para la ejecución del código de redes neuronales. Esto se da por la forma en que está estructurada la interfaz gráfica. Ya que una vez se selecciona el algoritmo de redes neuronales, envía por serial la selección del algoritmo 1, esto generará que solo se ejecute la sección mostrada anteriormente. Por otro lado, una vez se selecciona que tipo de red se va

a utilizar, se envía la selección del algoritmo, pero esta vez con el valor de 10 o 11. 10 indicando que la red preestablecida será la que se utilice, o un 11 para indicar que la red que se desea ejecutar es una red que el usuario definirá. A continuación, se puede observar cómo funciona el envío de esta información desde la interfaz gráfica, siendo cada valor multiplicado por un factor de 10000 (1 corresponde a 010000, 10 a 100000, y 11 a 110000).

```
// *** WHILE ***
while(1){

  switch (SELECTOR_DE_ALGORITMO) {
    case 1 :    /*  *** ALGORITMO 1. REDES NEURONALES    *** */
    case 10:    /*  *** RED PRE ESTABLECIDA (:)          *** */
    case 11:    /*  *** NUEVA RED           (;)          *** */

      if (CODIGO_CORRIDO_ONCE==1){
      /*ESTA PORCION DE ODIGO SE CORRE UNA UNICA VEZ, PARA
          CONFIGURAR INFORMACION PREESTABLECIDA*/
          IA_LCD_CLEAN_ALG_CABLE_DELV(FILA2|FILA3|FILA4);
          memset( MENSAJE_ALGORITMOS  ,0, sizeof(MENSAJE_ALGORITMOS)  );
          strcat(MENSAJE_ALGORITMOS,"ALG1.RED NEURONAL");
          IA_LCD_WRITE_CABLE_DELV(MENSAJE_ALGORITMOS,2,0);

          memset( MENSAJE_ALGORITMOS  ,0, sizeof(MENSAJE_ALGORITMOS)  );
          strcat(MENSAJE_ALGORITMOS,"VAL SAL: ... ");
          IA_LCD_WRITE_CABLE_DELV(MENSAJE_ALGORITMOS,3,0);

          IA_OPE_WINDOW_INICIAL_DELV(WINDOW_VECTOR);
          SIZE_N_RED=0;
          CODIGO_CORRIDO_ONCE=0;
      }
```

Limpia filas de LCD

Imprime en LCD

Inicializa filtro para lectura ADC

Reinicia las variables

Figura 25. Código de selección del algoritmo para una red neuronal.

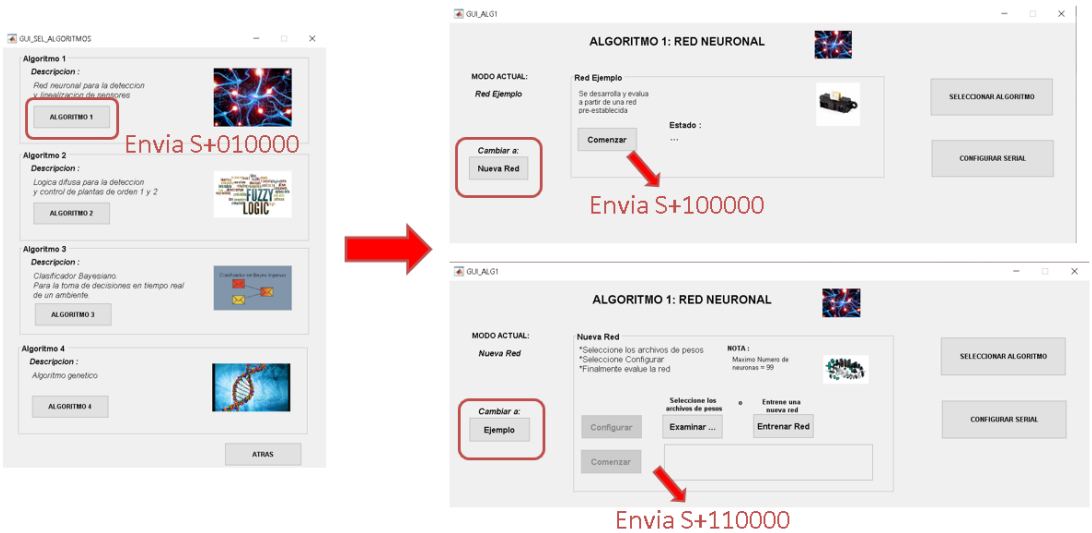


Figura 26. Proceso de selección del algoritmo de redes neuronales a través de interfaz de usuario creada.

En la figura 26 podemos observar la ventana de selección de algoritmo que permite elegir qué tipo de red se desea ejecutar. Y muestra los datos enviados por la interfaz según cada una de sus variantes. Dicha estructura de datos será empleada para los demás comandos de activación de los algoritmos. Esta estructura esta formada de la siguiente manera:
X1 SIGNO V1 VF

- X1=Valor char para determinar que hacer con los datos, dicha variable puede tomar los siguientes valores: C (Conectado, para verificar conexión), D (Envío de datos para ser utilizados por algún algoritmos), S (Se hace la selección de alguno de los algoritmos), T (Se enviaran valores de configuración)
- SIGNO= Determina si el dato a enviar será positivo o negativo.
- V1= Corresponde a un valor numérico entre 0 y 99 multiplicado por un factor de 10000, correspondiendo siempre a un valor de 6 dígitos.
- VF= Fin de valor enviado ('F').

En caso de que se desee hacer una nueva red neuronal, se muestra a continuación las opciones que presenta la interfaz. Ya que, si bien es posible seleccionar los archivos de los pesos para ser enviados directamente al microcontrolador, también es posible abrir una nueva ventana para establecer algunos parámetros y entrenarla según los criterios que se deseen. Con el fin de no extender las posibilidades del algoritmo, se limitó como variables de cambio el valor del número de neuronas que se desean en la capa oculta, y el error que se desea alcanzar, tal como se observa a continuación en la figura 27.

Figura 27. Selección de creación de red neuronal a partir de datos de caracterización.

Como se observa en la figura 27. La primera vez que se desee examinar el archivo de caracterización para realizar el proceso de entrenamiento de la red, un “Pop up” aparecerá, enumerando los pasos a seguir para evitar errores en el proceso. Dichos errores serán mostrados en el command Window de la ventana de entrenamiento, advirtiendo la razón del error.

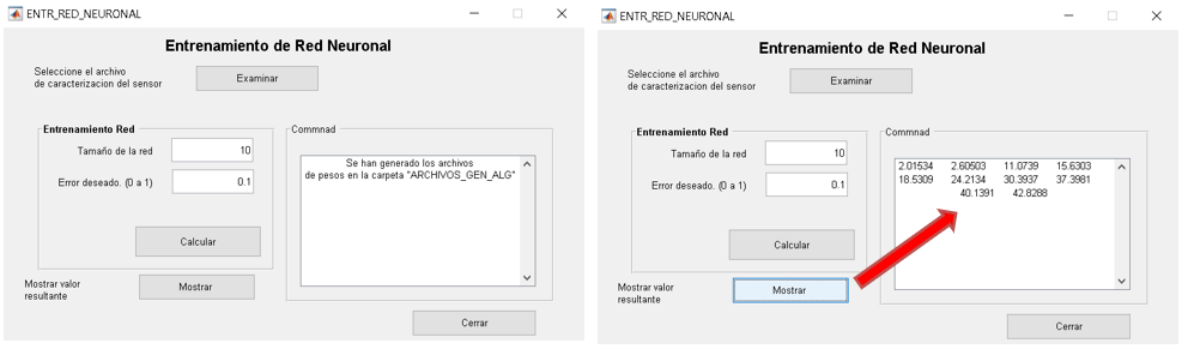


Figura 28. Resultados de entrenamiento de red neuronal.

Una vez se introducen de manera satisfactoria los valores de caracterización y de entrenamiento para la red, se procede a calcular los pesos de la misma, siguiendo el procedimiento mostrado en las figuras 17 a la 26. Estos serán luego enviados en una trama de datos por el botón Configurar de la ventana de la nueva red neuronal, mostrado en la figura 28. Una vez se define el tipo de red que se desea ejecutar en el microcontrolador, se procede a ejecutar la red según corresponda.

En la figura 29 podemos observar la sección de código que se requiere para el funcionamiento de una red de ejemplo. En esta se obtienen los valores del sensor por medio del ADC, luego se realiza una interpolación de los datos del sensor mediante una función creada para dicho fin, para posteriormente insertar el valor interpolado en la función de evaluación para la red neuronal.

```
//!ALGORITMO DE RED NEURONAL ( EJEMPLO )
if (SELECTOR_DE_ALGORITMO==10){
    IA_OPE_WINDOW_MOVE_DELV   (WINDOW_VECTOR);
    VALOR_SENSOR    =WINDOW_VECTOR[WINDOWS_SIZE];   } Obtener ADC con filtrado
    VALOR_SENSOR    =IA_OPE_INTERP_GEN_DELV (-1,1,0,3.3,VALOR_SENSOR);
    VALOR_VAR_MEDIDA=IA_ALG1_EVAL_RED_DELV  (0,0,VALOR_SENSOR);
```

Figura 29. Código de evaluación de red neuronal de ejemplo en microcontrolador.

Por otro lado, si se observa la figura 30, se encuentra el código que se encarga de evaluar y adaptar la red neuronal establecida por el usuario. Tal como se observa entre el código de evaluación para la red de ejemplo y la nueva, la diferencia radica en el establecimiento de los pesos en el caso de la red de usuario, y en los parámetros de entrada de la función “IA_ALG1_EVAL_RED_DELV” el cual cuenta con 3 valores de entrada como se muestra a continuación, donde el primero establece el tipo de evaluación que se hará (Red ejemplo o red nueva), el segundo valor de la función , es el número de neuronas, y por último el tercer valor la interpolación del valor del sensor (entrada de la red) en el momento de la iteración.

```
//!ALGORITMO DE RED NEURONAL ( NUEVA )
if (SELECTOR_DE_ALGORITMO==11){
    if (SIZE_N_RED == 0 ){ → Establece los valores de los nuevos pesos
      SIZE_N_RED=CONFIG_ALGORITMO[0];
      IA_ALG1_SET_PESOS_DELV(VALOR_VECTOR_SERIAL_NUM , SIZE_N_RED);
    }
    else {
      IA_OPE_WINDOW_MOVE_DELV   (WINDOW_VECTOR);      } Obtener ADC con filtrado
      VALOR_SENSOR    =WINDOW_VECTOR[WINDOWS_SIZE];
      VALOR_SENSOR_DR =IA_OPE_INTERP_GEN_DELV(-1,1,0,3.3,VALOR_SENSOR);
      VALOR_VAR_MEDIDA=IA_ALG1_EVAL_RED_DELV(1,SIZE_N_RED,VALOR_SENSOR_DR);
```

Figura 30. Código de evaluación de una red neuronal creada por usuario.

En la figura 31 se puede observar la declaración de los valores de pesos tanto de la red de ejemplo, como la definición de los vectores de pesos para la red creada por el usuario, estando limitadas por la constante "MAX_TAM_RED".

```
28  //!  Variables de los pesos (PRE-DEFINIDOS)
29 volatile float W Dendrita 1 [20]={0.569742847,6.232753058,-2.099165344,0.90498193,-0.451257865,
34 volatile float W Dendrita 2 [20]={-0.376501873, -1.703092009, 2.379459521 ,-0.390041655 ,
40 volatile float W Neurona 1  [20]={  0.024396616,0.139452807,1.846470156,0.327359543,
46  volatile float W Neurona 2       =1.202672117;
52 volatile float WD_1[20]={0.1 ,  0.1  , 0.1 , 0.1  , -0.1 ,
53                         -0.1 ,  0.1  , 0.1 , 0.1  ,  0.1 ,
54                         -0.1 , -0.1  , 0.1 , 0.1  , -0.1 ,
55                         -0.1 , -0.1  , 0.1 , 0.1  ,  0.1 }; // Vector para operaciones
56
57  //!
58  volatile float W_D_1[MAX_TAM_RED];  // Pesos de primer  capa de nuevas dendritas
59  volatile float W_D_2[MAX_TAM_RED];  // Pesos de segunda capa de nuevas dendritas
60  volatile float W_N_1[MAX_TAM_RED];  // Pesos de primer capa de nuevas  neuronas
61  volatile float W_N_2;               // Pesos de Ultima neurona
62
63  volatile float W_Ops[MAX_TAM_RED];  // Vector para las operaciones realizadas.
64  volatile float NUMERO_ALEATORIO;
```

Figura 31. Código de creación de las variables de pesos para algoritmo de red neuronal.

Posterior a esto se crea la función que evaluará la red. La cual sigue el mismo procedimiento que se realiza en Matlab para la evaluación de la misma. Se obtienen los valores del número de neuronas dependiendo estas de la recepción de datos por serial. Posteriormente se obtiene la salida de la primera capa. En donde la línea de Matlab en los comentarios se extiende a 3 en Keil.

Finalmente, y como se observa en la figura 32 en el código se obtiene la salida del algoritmo a partir de una función de interpolación. De esta manera retorna el valor de salida de la red al código principal donde será utilizado para la visualización en la LCD.

```
float IA_ALG1_EVAL_RED_DELV       (int TIPO_EVAL ,int NUM_N ,  float Sensor_Value){
  // TIPO_EVAL:
  //   * 0 = RED DE EJEMPLO
  //   * 1 = RED NUEVO CON PESOS RECIBIDOS POR SERIAL
//! Declaracion de variables
int Num_Neuronas=sizeof(W_Dendrita_1)/(sizeof(W_Dendrita_1[0]));

//Salida de la primera capa // y12 = tanh((W_Dendrita_1*x)+W_Neurona_1)
IA_OPE_MULT_VEC_SCA_DELV (W_Dendrita_1,Sensor_Value,Num_Neuronas,WD_1);
for (int i=0;i<(int) Num_Neuronas;i++){ WD_1[i]               =WD_1[i]+W_Neurona_1[i];  }
for (int j=0;j<(int) Num_Neuronas;j++){ Salida_Primer_Capa[j] =tanh(WD_1[j]);           }

//Salida segunda capa : Salida_Segunda_Capa = tanh((W_Dendrita_2*Salida_Primer_Capa)+W_Neurona_2);
//y22 = tanh((W_Dendrita_2*y12)+W_Neurona_2);
Salida_Segunda_Capa = IA_OPE_MULT_VEC_DELV (W_Dendrita_2,Salida_Primer_Capa,Num_Neuronas);
Salida_Segunda_Capa = tanh(Salida_Segunda_Capa+W_Neurona_2);

// Interpolacion del Valor de salida para la escala correspondiente del sensor
Salida_Algoritmo = IA_OPE_INTERP_DELV(0,Salida_Segunda_Capa);

return Salida_Algoritmo;
```

Figura 32. Evaluación del algoritmo de red neuronal.

3.5 Análisis de desempeño.

Una vez se ejecuta el algoritmo de ejemplo, se puede observar una comparación entre la caracterización del sensor y la respuesta del embebido mediante la visualización en la LCD, esto se observa en la figura 33. Los datos que se observan en las imágenes tomadas corresponden al valor de salida de la red (la distancia), el voltaje de lectura por ADC y finalmente el tiempo de ejecución del algoritmo.

Figura 33. Distribución de información en la LCD.

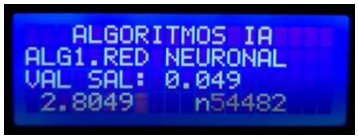
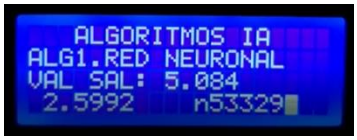
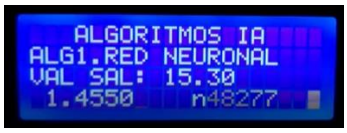
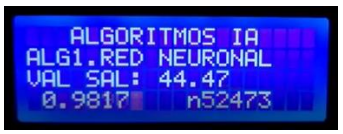
Voltaje del sensor (v)	Distancia (cm)	Respuesta de la red.	Voltaje del sensor (v)	Distancia (cm)	Respuesta de la red.
2.8	0		1.17	25	
2.6	5		1.08	30	
1.68	10		1.02	35	
1.46	15		1	40	
1.34	20		0.98	45	

Tabla 1. Comparación de datos de salida del algoritmo de red neuronal.

Como se puede observar en la tabla 1, los valores de distancias corresponden a los valores de caracterización medidos al momento de ejecutar el algoritmo. No obstante, es necesario resaltar que el sensor tiene una respuesta bastante rápida para las medidas más cercanas al mismo, lo que dificulto la toma de valores de voltaje y distancia. Esto se debe a la respuesta que este tipo de sensores tienen, tal como se mostró en la figura 16.

Una vez se ha ejecutado el algoritmo, y se comprobó su correcto funcionamiento respecto a las salidas esperadas en posición del sensor SHARP, se procede a realizar el análisis estadístico de los tiempos que tomo ejecutar la red. Para dicha finalidad se hizo funcionar el algoritmo por un periodo de tiempo continuo mientras se recorría el espectro completo del sensor.

Los valores de tiempos tomados durante esta medición permitieron obtener un valor promedio de $50988.78\ ns$, equivalente a $50.98\ \mu s$ aproximadamente, lo cual supone un tiempo de respuesta bueno para diferentes aplicaciones en que se requiera

utilizar con precisión dispositivos no lineales. De la misma manera, se obtuvo una varianza de los valores de $21.777\ \mu s^2$, los cuales representan un posible aumento en el tiempo de ejecución del algoritmo bajo ciertos valores de entrada en la función no lineal, que puede ser despreciado.

4. CONTROL DIFUSO.

4.1 Problemática

La lógica difusa ha tenido un gran impacto tecnológico desde su desarrollo. Debido a esto se ha logrado su aplicación en el ámbito computacional ya que es el área en donde mayor viabilidad ha tenido y es debido a que esta tiene una gran adaptabilidad en su funcionamiento ya que no depende exclusivamente de un procesamiento numérico de la información presente en el problema en que se desea aplicar el algoritmo, sino en una cualificación de la misma para así determinar por medio de una inferencia el comportamiento que debe tener el sistema según el criterio del programador.

Por esta razón resulta de gran utilidad evaluar el desempeño de cualquier hardware orientado a sistemas portables, en donde sea posible implementar un algoritmo difuso, con el fin de determinar la viabilidad del mismo a la hora de ser usado para desarrollar una aplicación que requiera del uso de la lógica difusa. Esto comúnmente se conoce como un análisis de desempeño y brinda información respecto a si un hardware es viable para la implementación de un algoritmo de lógica difusa o no.

En la actualidad esto resulta útil debido a que en algunas ocasiones las fallas en la implementación no corresponden a una programación incorrecta ya que existe la posibilidad de que el hardware no cumpla con los requisitos necesarios.

4.2 Características del algoritmo

El control difuso es una herramienta que permite realizar aplicaciones como el control de sistemas no lineales, debido a que no evalúa funciones matemáticas con el fin de determinar la señal de control, sino que cualifica el comportamiento del sistema en diferentes funciones de pertenencia. Estas funciones permiten determinar que tanto pertenece la señal de entrada a cada una de las mismas y una vez que se tiene el grado de pertenencia sobre las funciones se procede a establecer una serie de reglas que determinen el valor de la señal de control que se deberá usar para regular el sistema propuesto. (Ponce, 2010)

Es importante mencionar que el algoritmo que se usará es en base al de Mamdani, el cual por medio de la diferencia entre el valor deseado y el real (error) se seleccionan las funciones de pertenencia, además de esto se establecen las reglas por medio de condicionales como se mencionó anteriormente. En la ecuación 6 se muestra el cálculo del error para el método Mamdani, en esta se obtiene el error, a partir del valor deseado y real del sistema.

$$\varepsilon = V_{deseado} - V_{real}$$

Ecuación 6. Calculo del error para el método Mamdani.

Así mismo en la ecuación 7, se define la ecuación que determina las reglas según el método propuesto. (Ponce, 2010)

$$X_t(x,z) = \vee_{y \in Y} \left(X_R(x,y) \wedge X_S(y,z)\right)$$

Ecuación 7. Calculo de las reglas según el método Mamdani.

Como se mostró en la ecuación 7, la inferencia se realiza por medio de una composición máx-min con el fin de determinar así la regla. (Ponce, 2010)

Una vez que se determinan las reglas se procede a calcular el valor de la señal de control a partir del área resultante formada por la intersección de las funciones de pertenencia mediante el cálculo del centroide de esta área (ecuación 8). El valor resultante es equivalente a la salida o señal del controlador difuso. (Ponce, 2010)

$$\frac{\sum \mu(x) * x}{\sum \mu(x)}$$

Ecuación 8. Calculo del centroide del área resultante.

Estas funciones de membresía se definen por medio de ecuaciones matemáticas básicas como la ecuación de la recta, ecuaciones cuadráticas y/o trigonométricas. Por lo que es necesario determinar la ecuación que describa el comportamiento de cada una de estas funciones para establecer la pertenencia de la señal de entrada a cada una de ellas. (Ponce, 2010)

4.3 Antecedentes del algoritmo

La lógica difusa se desarrolló en los años 60's por el ingeniero eléctrico Lofty Zadeh en la Universidad de California y a partir de su descubrimiento se perfeccionó la técnica en Europa y Asia en donde se aplicó principalmente en Japón. (Rodríguez, 2009)

El control difuso, se ha caracterizado por ser un método que cualifica cualquier tipo de medición física puesto que pretende interpretar el sentido común que posee el ser humano. Es un proceso de simulación del razonamiento humano, que para el área de control permite interpretar el comportamiento del sistema por medio de palabras en lugar de números, esto supone una solución cuando no es posible determinar su comportamiento matemático modelamiento mediante una función de transferencia. (Kouro, 2002)

Este control ha sido usado para un helicóptero a escala ya que este sistema es de tipo MIMO (Multiple Input, Multiple Output, por sus siglas en inglés) pero para simplificación del problema se fijó uno de sus ejes de rotación, dejando la posibilidad de controlar el ángulo vertical del mismo. Es importante para cualquier sistema

realizar las pruebas suficientes para determinar el funcionamiento cualitativo y así generar el conocimiento experto del mismo. Luego de determinar el funcionamiento del sistema se definen dos entradas para el control difuso que son el error en el ángulo y la derivada del error del mismo. (Kouro, 2002)

Del conocimiento experto se procedió a realizar el controlador difuso usando el Toolbox de MATLAB, para las entradas se establecieron 3 funciones de membresía y 3 funciones de membresía para la salida. Con las funciones de membresía establecidas se procede a diseñar una matriz que permite establecer las reglas del control difuso, una vez que se establecen las reglas se procede a incluir el control en el sistema para mejorar su comportamiento. (Kouro, 2002)

Otra aplicación muy usada para el control difuso es enfocada a sistemas no lineales, en donde es difícil aplicar el control clásico debido a que el comportamiento de un control de este tipo debería ser entonces, dinámico con el fin de controlar el sistema en cualquier punto de operación. Por esta razón resulta más conveniente cualificar la entrada en diferentes rangos para que de esta manera se pueda controlar el sistema sin la necesidad de una representación numérica del comportamiento de este. (Morcillo, 2011)

Un ejemplo específico es el control de un péndulo invertido, el cual se estimula mediante el movimiento del soporte al cual se encuentra anclado el péndulo. En este problema lo más importante es establecer un conocimiento experto que le dé la capacidad al desarrollador de definir las reglas pertinentes para asegurar el buen funcionamiento del sistema, ya que para este caso se establece un orden de relevancia entre la oscilación del péndulo y el movimiento del carro. (Morcillo, 2011)

A su vez la lógica se ha usado en el control de otra planta no lineal en el sector de las energías renovables ya que permite mejorar el funcionamiento de un aerogenerador. Este dispositivo posee en su comportamiento zonas donde se encuentran no linealidades, suponiendo que un controlador convencional no logrará adecuar el comportamiento del sistema a las condiciones deseadas cuando se presenten estas no linealidades. (Peñas, 2013)

Es importante resaltar que la inteligencia computacional en los últimos tiempos aporta soluciones a los problemas reales que no habían sido abordados por su complejidad, pero con la aplicación del soft computing se ha logrado con éxito realizar controles inteligentes y mejorar considerablemente su funcionamiento en las aplicaciones desarrolladas. (Peñas, 2013)

Sin embargo, su funcionamiento no ha presentado muy buenos resultados ya que la complejidad del sistema representó un problema al no existir un método sistemático para la sintonía del controlador difuso. A pesar de este inconveniente se logró aplicar el control y mantener la salida en torno a la referencia con la presencia de una variabilidad de las entradas. (Peñas, 2013)

El uso de la lógica difusa se presenta también en la segmentación de imágenes cuando se usa el método de la Transformada de Watershed. Su enfoque se encuentra en mostrar la robustez y simplicidad con la que se desarrolla el algoritmo y su fácil adaptabilidad a las características de cada imagen. (Gonzáles, 2008)

La segmentación se realiza según la información que el observador desee obtener de la imagen y para lograr este objetivo se usó la transformada de Watershed que como se mencionó anteriormente, se adapta fácilmente a los diferentes tipos de imágenes y distinguir objetos complejos que no pueden ser analizados correctamente mediante algoritmos convencionales. (Gonzáles, 2008)

El sistema de inferencia difusa tuvo como finalidad el reconocimiento de objetos de interés con la viabilidad de aplicarse a otras imágenes realizando el mismo procedimiento. Esto demostró que el sistema es adaptable a cada imagen en particular. (Gonzáles, 2008)

Un caso adicional del uso de la lógica difusa es aplicado a la protección financiera y sus componentes (crédito, operacional y liquidez) usando un clasificador que permita determinar y clasificar el riesgo mencionado. Se diseñó un modelo usando la técnica de clasificación difusa llamada “Subtractive Clustering”. (Mártinez, 2007)

El diseño de este modelo usando el algoritmo presentado demostró una buena efectividad y una alta capacidad de adaptación al tipo de problema que se modeló. En la mayoría de los resultados presentados por el algoritmo indican errores bastante bajos tanto en el entrenamiento como en las pruebas realizadas, esto traduce en que se obtuvo un alto porcentaje de acierto en la clasificación. (Mártinez, 2007)

4.4 Funcionamiento

Inicialmente fue necesario un repaso para recordar los conceptos importantes de la lógica difusa ya que para esta se tiene una matemática diferente con el fin de establecer una guía universal para la aplicación de la misma.

Como se explicó anteriormente, es necesario adquirir un conocimiento experto para el diseño del controlador difuso, por esta razón se propuso una planta de primer orden cuya entrada es de tipo escalón unitario, teniendo en cuenta que si se usa una entrada de mayor orden (rampa o cuadrática) el error de estado estacionario tiende a infinito y este comportamiento no es el deseado.

Luego se evalúa el comportamiento del sistema en lazo abierto y en lazo cerrado con el fin de determinar la existencia de error en estado estacionario usando la herramienta de Simulink de MATLAB como se muestra a continuación en la figura 34.

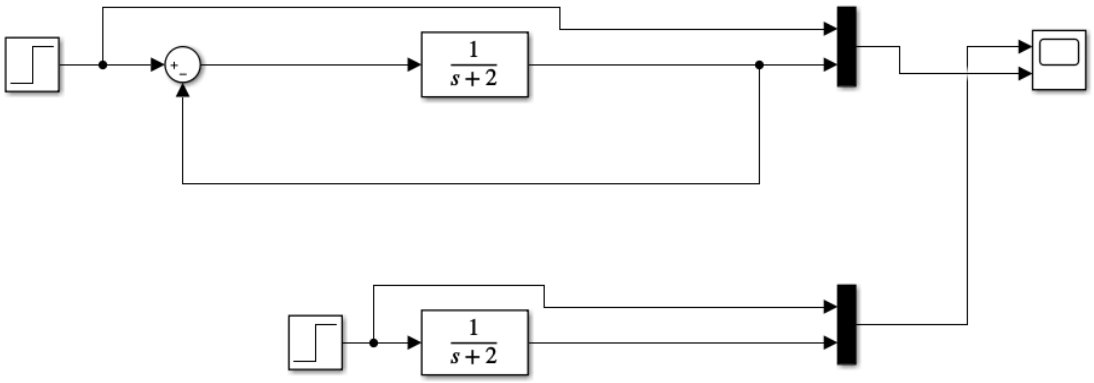
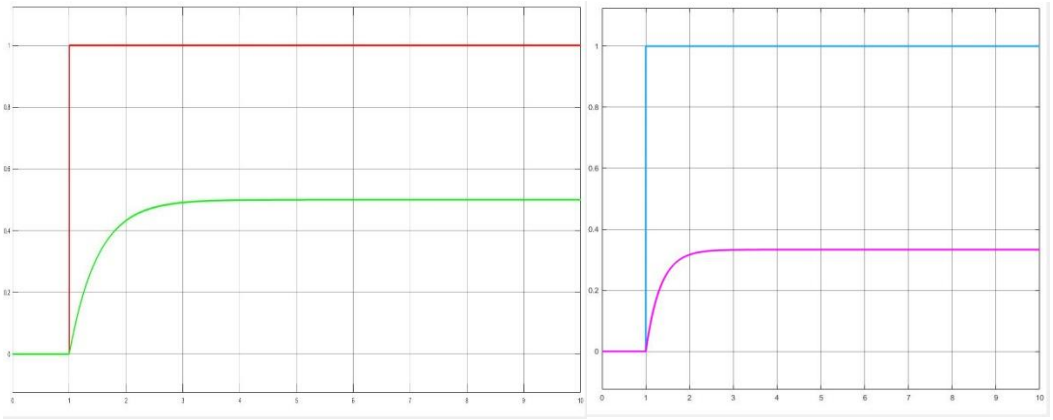


Figura 34. Simulación del sistema en lazo abierto y en lazo cerrado.

Luego de obtener la respuesta del sistema (figura 35) se procede a diseñar el controlador difuso. El diseño inicialmente es para la planta propuesta con el fin de realizar un primer acercamiento al funcionamiento de este.



a. b.

Figura 35. Respuesta del sistema ante una entrada escalón en lazo abierto (a) y lazo cerrado (b).

De lo que se realizó en el Toolbox fue posible obtener una información inicial para determinar el “Conocimiento Experto” ya que es necesario tener mucha claridad sobre el funcionamiento del sistema que se quiere controlar, por lo que este conocimiento experto es necesario antes de realizar el control. Implica además la capacidad de poder determinar las variaciones en la entrada y como afectan al sistema y salida del mismo.

A partir de este conocimiento experto se determinó que las máximas variaciones en la señal de entrada eran de -1 a 1. Este rango se determinó en función de la referencia que como se observó en la figura anterior es una entrada escalón unitario. Luego entonces, con esta información se determinaron 3 funciones de membresía para la entrada y 3 para la salida.

Desde la interfaz “Fuzzy” en MATLAB se obtuvo la siguiente superficie difusa mostrada en la figura 36.

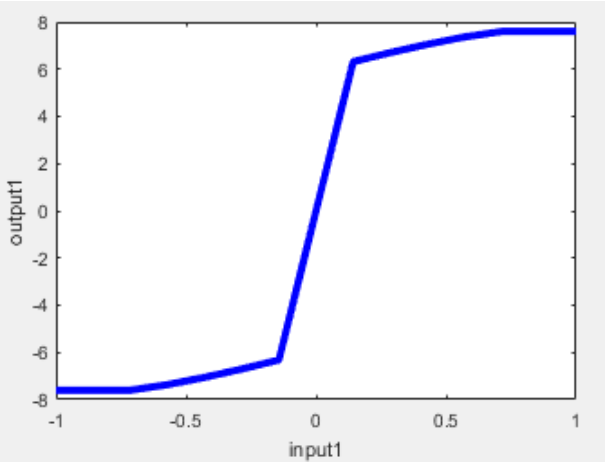


Figura 36. Superficie difusa del control para planta de 1er orden.

En la figura 36 se muestra cómo se establecieron un total de 3 reglas para realizar el control de la planta, se analizó el comportamiento de manera simulada y se verificó que el diseño hubiera sido correcto.

Como se observa, debido a la baja cantidad de reglas establecidas para el control difuso se dio como resultado una superficie que no considera muchos casos posibles que relacionen la señal de entrada con la señal de salida, sin embargo, esto no afecta el funcionamiento del controlador.

Para la simulación del sistema con el controlador implementado fue necesario usar el bloque “Fuzzy Controller” en la interfaz de Simulink, este bloque nos permite cargar el archivo correspondiente al control difuso ya diseñado en el ToolBox de MATLAB. Al integrar el bloque mencionado se procedió a realizar la simulación, una vez realizada la simulación del sistema presentado en la figura 37 se obtiene una respuesta deseada para el control de la planta.

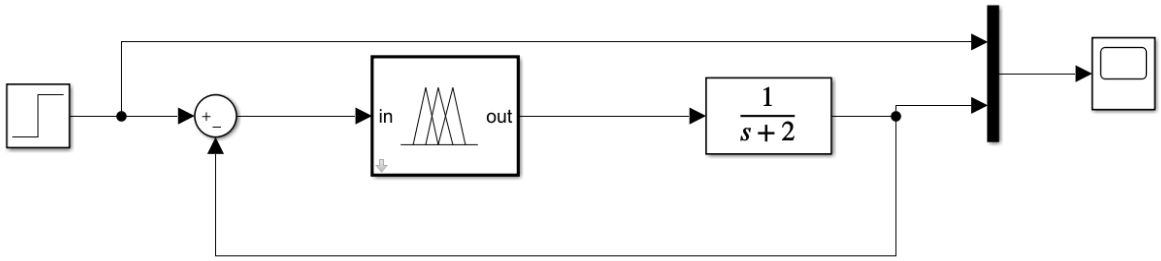


Figura 37. Simulación del sistema controlado en MATLAB usando el ToolBox.

De la figura 38 se puede observar que el sistema controlado da como resultado una salida cuyo valor es de 0.983 por lo que se tiene un error de aproximadamente 1.7% con respecto al valor deseado. Al usar el Toolbox de MATLAB “Fuzzy” se pudo observar que el control presentaba un funcionamiento adecuado a las exigencias requeridas por lo que la siguiente tarea era replicar el funcionamiento del mismo con la excepción de que se desarrollaría un algoritmo de programación propio con el fin de implementarlo en el microcontrolador.

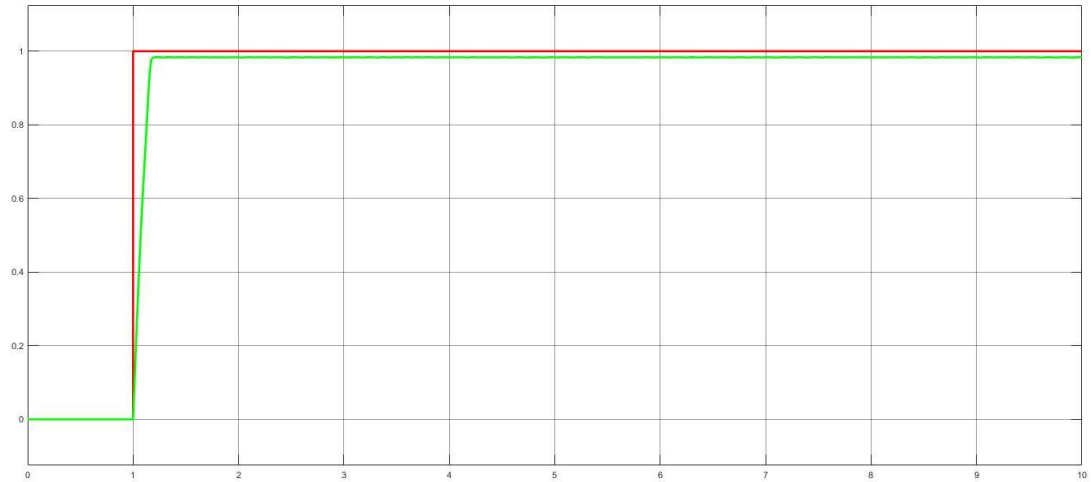


Figura 38. Respuesta del sistema usando control difuso del Toolbox de MATLAB.

Para este algoritmo fue necesario tener claros los conceptos de la matemática difusa ya que es la forma de cuantificar la señal de entrada para su procesamiento dentro del controlador.

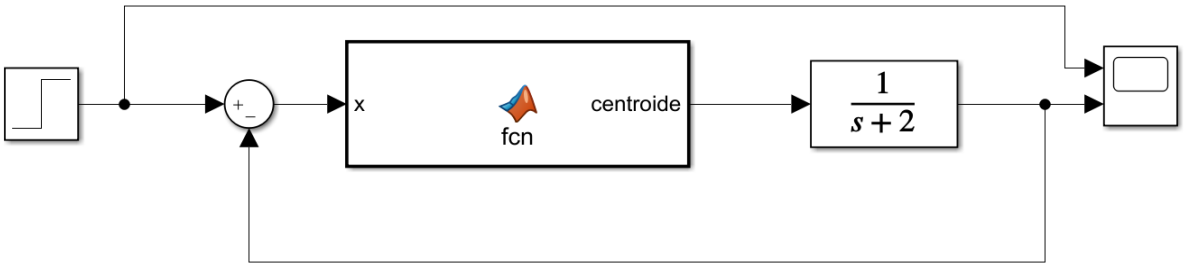


Figura 39. Sistema controlado por lógica difusa, usando código propio.

Es necesario entonces, conocer las ecuaciones matemáticas que describen el comportamiento de las funciones de membresía tanto para las de entrada como para las de salida, estas ecuaciones suelen ser funciones a trozos ya que describen polígonos o triángulos en la mayoría de los casos y al obtener sus ecuaciones correspondientes es posible calcular la pertenencia de la señal de entrada a cada una de estas.

Una vez que se evalúa la pertenencia de la señal de entrada a las funciones de membresía se debe calcular la salida según las reglas que se hayan establecido, en este caso como solamente se tienen tres reglas es relativamente sencillo estimar la región resultante de la intersección de las funciones membresía por medio de la matemática propia de la lógica difusa.

Se debe entonces, evaluar el grado de cumplimiento de cada regla (ß), el cual se calcula realizando una comparación correspondiente a cada regla establecida. Como se mencionó anteriormente, se deben calcular tres grados de cumplimiento puesto que se tienen 3 reglas, según cada regla se evalúan ciertas funciones de membresía definidas por la regla misma.

La comparación permite determinar el mayor valor entre las funciones de membresía de la entrada y las de la salida, teniendo en cuenta que la señal de entrada tiene una pertenencia determinada en las funciones de membresía de entrada por lo que es un valor constante al momento de realizar la comparación. De esta manera se obtiene una región compuesta por intersecciones entre un valor constante y las funciones que incluya la regla por lo que tendrán tres regiones diferentes debido a que se tienen 3 reglas.

Una vez que se determinan los grados de cumplimiento para cada regla se realiza una última intersección entre estas regiones con el fin de obtener una región conformada por los grados de cumplimiento. A esta región final se le debe calcular el centroide y el valor resultante es equivalente a la señal de control, sin embargo, para realizar este cálculo se debe realizar una división entre la sumatoria de la multiplicación de la región final por su imagen en el eje coordenado X y el valor de la región evaluada en ese punto.

Una vez que se desarrolla el algoritmo de programación para el control difuso, se implementa usando el bloque MATLAB function en la interfaz de Simulink. Es importante resaltar que este control es discreto por lo que se debe su funcionamiento es mejor si el muestreo es mayor, esto resulta en una consideración importante a la hora de implementar el algoritmo ya que se tendrá mayor precisión a medida que el muestreo sea mayor.

Una vez que se integra el algoritmo de programación del control difuso se procede a simular y evaluar la respuesta del sistema controlado. Al implementar el algoritmo de programación se simulan dos respuestas del sistema para comparar según el

control implementado, como se observa en la figura 40, la curva roja representa la entrada escalón, la curva amarilla representa el control mediante el algoritmo de programación desarrollado y la curva azul representa el control del sistema usando el Toolbox Fuzzy de matlab.

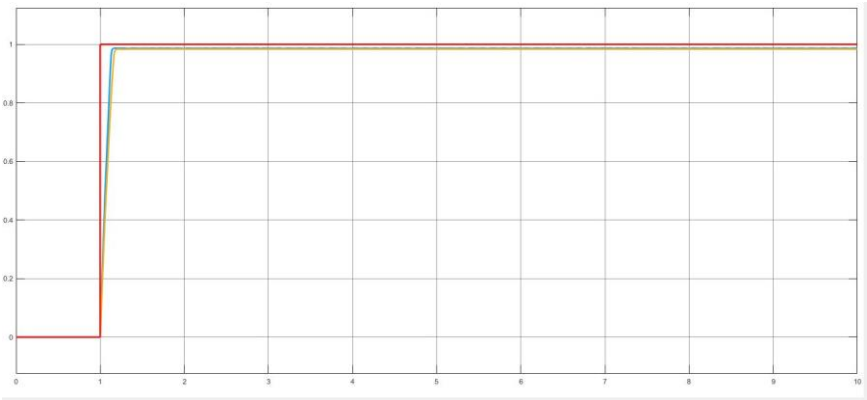


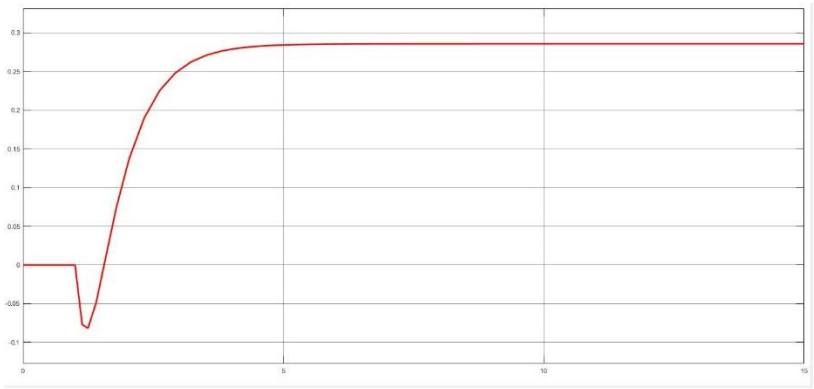
Figura 40. Respuesta del sistema controlado.

Una vez que se realizó el control de la planta de primer orden se propuso una segunda planta de un orden mayor para aumentar la complejidad en el diseño del control difuso. El control se aplicará a la planta, basado en dos entradas que son el error y la derivada del error respectivamente. De nuevo es necesario desarrollar el conocimiento experto del sistema por lo que se debe analizar el comportamiento de la planta. La función de transferencia se presenta en la siguiente ecuación y de esta manera se presentará la respuesta de la misma en lazo cerrado con el fin de determinar el comportamiento de las dos entradas del control difuso.

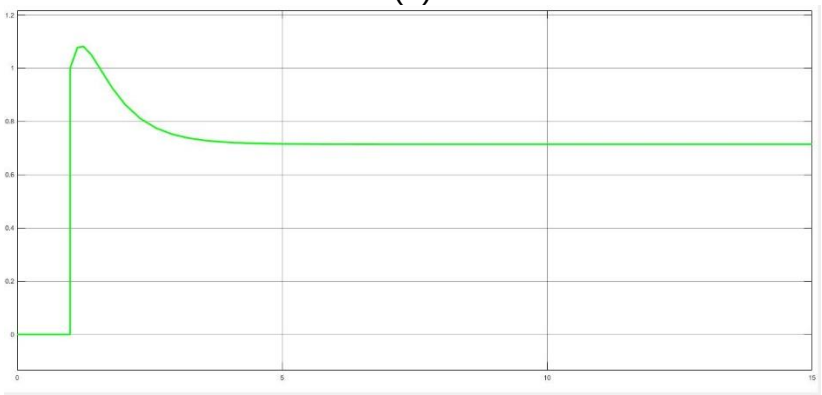
$$\boldsymbol{F(s)} = \frac{\boldsymbol{-S+2}}{\boldsymbol{S^2+7S+5}}$$

Ecuación 9. Función de transferencia para control difuso.

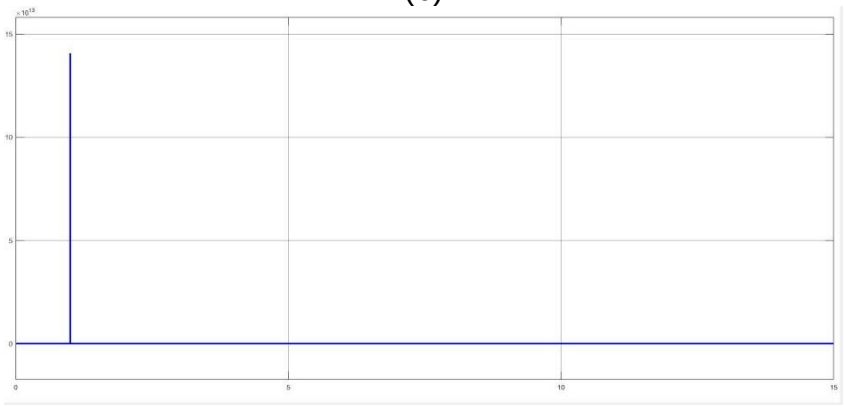
Sin embargo, la función de transferencia no es necesaria para realizar el control, aunque nos permite entender el funcionamiento de esta y así determinar el rango de variación del error y de su derivada.



(a)



(b)



(c)

Figura 41. Respuesta del sistema en lazo cerrado (a), señal de error (b), derivada de la señal de error (c).

A diferencia de la planta de 1er orden que se usó inicialmente, esta planta posee un cero inestable y su efecto es la curva negativa que se muestra en la figura 41 (a) cuando se estimula con la entrada paso. Este comportamiento se debe tener en cuenta para el desarrollo del control difuso ya que al analizar la figura la parte “a” muestra que es un pico que supera la entrada escalón, sin embargo, esto permite contemplar el rango de la primera entrada del control difuso mientras que en la parte b de la figura se observa que la derivada del error solo muestra un pico con

tendencia a un valor infinito como consecuencia de ser la derivada de una constante.

El diseño con control difuso implica dos entradas que son la señal de error junto con su derivada respectivamente, para ambas entradas se especifica el rango de variación de las señales y así mismo se establecen las funciones de membresía. A diferencia del primer control, para esta planta se definen 5 funciones de membresía que abarcan una región de variación de -3 a 3 para la primera entrada.

Debido al pico tan elevado que presenta la derivada del error no es posible estimar la variación de la señal en los tiempos posteriores por lo que se debe hacer un zoom para poder determinar el rango que se usará para el control difuso respecto a esta entrada.

Al observar la figura 42 de manera detallada, la derivada del error se estima que la variación es de -0.4 a 0.4, considerando una variación negativa, se toma este rango para el diseño del control difuso respecto a la segunda entrada del mismo.

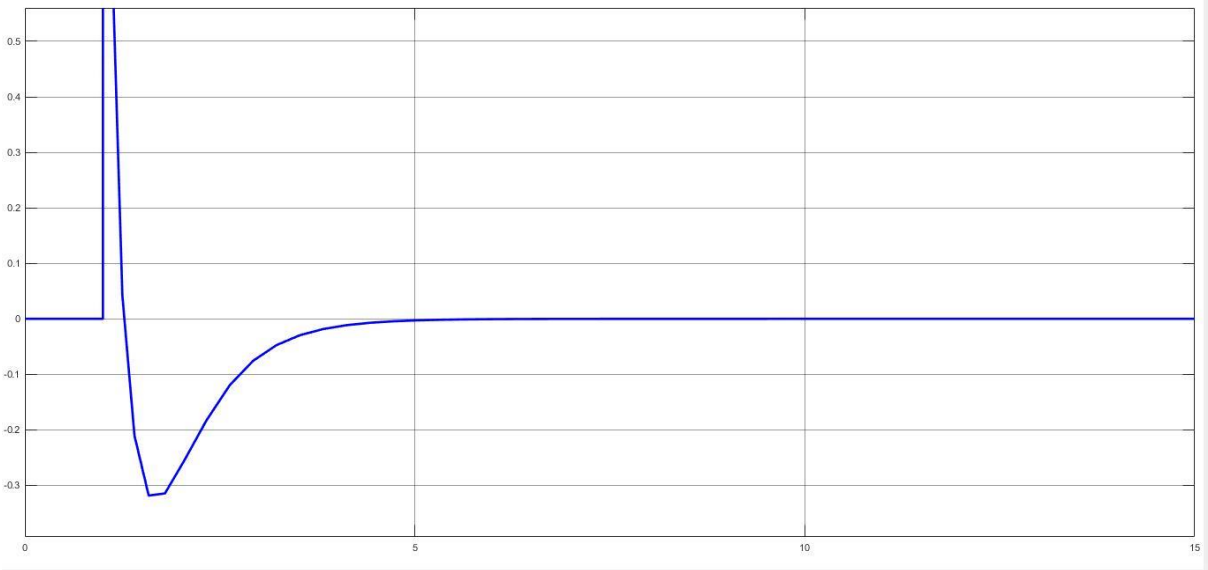


Figura 42. Variación señal de error en tiempo posterior al pico inicial.

Una vez que se estableció el rango se procedió a agregar las funciones de membresía para la derivada del error, con el fin de lograr un control más preciso se agregaron dos funciones más en comparación con la primera planta implementada. Completando así, 5 funciones de membresía para la primera entrada y 5 para la segunda.

Durante el análisis del sistema se observó también que la derivada del error se aproxima considerablemente a cero, esta fue una razón para establecer 5 funciones de membresía que correspondiera al comportamiento mencionado de la derivada del error.

Al tener una gran cantidad de funciones de membresía se pueden evaluar muchas posibilidades que se presenten en la señal de error y su derivada. Es importante tenerlo presente ya que esto dio paso a desarrollar 25 reglas para evaluar todos los posibles comportamientos del sistema. Al desarrollar las reglas se obtuvo la siguiente superficie difusa que se muestra en la figura 43.

Como se observa en la figura 43, la superficie difusa se representa de manera 3D ya que se tienen dos entradas y una sola salida. Es importante recordar que las reglas establecidas relacionan las dos entradas por lo que siempre la salida se encuentra en función de estas.

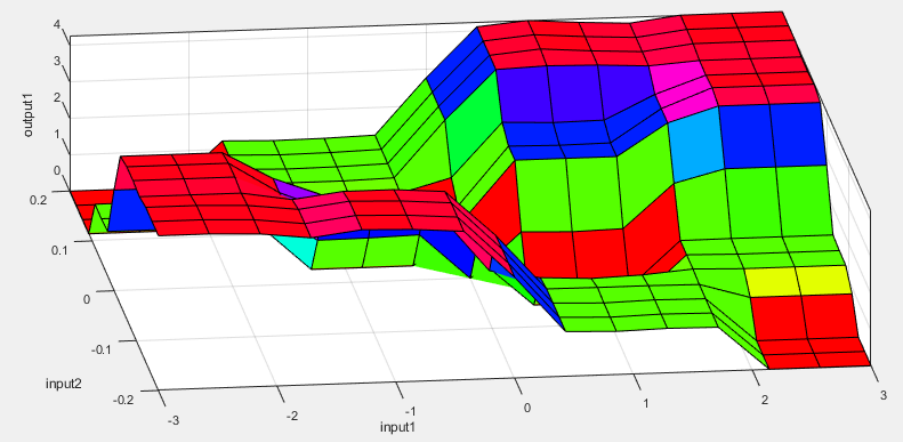


Figura 43. Superficie difusa para la planta de 2do orden.

En la implementación del controlador usando el toolbox de matlab se obtuvo el siguiente resultado mostrado en la figura 44.

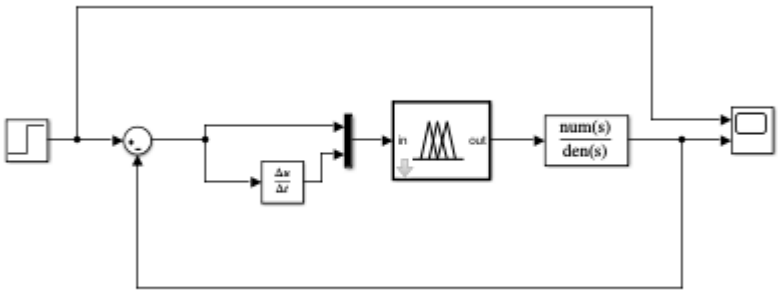


Figura 44. Sistema simulado de segundo orden con controlador difuso.

Una vez que se simuló el sistema usando el Toolbox de Matlab se obtuvo la siguiente respuesta que se observa en la figura 45.

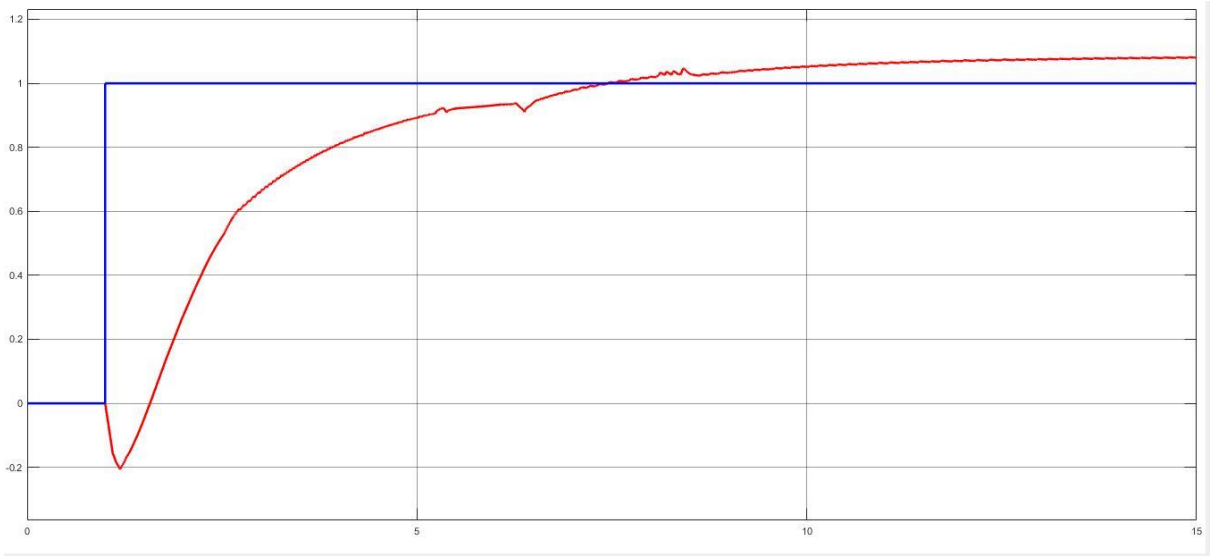


Figura 45. Respuesta del sistema controlado.

Como se observa en la figura 45, el sistema se controla con un error del 7.75%. Esto indica que el control diseñado es adecuado para el sistema por lo que se puede diseñar el algoritmo de programación que replique el funcionamiento del Toolbox Fuzzy.

Para el desarrollo del algoritmo se usó la estructura del que ya se había diseñado previamente por lo que el paso a seguir era agregar las reglas faltantes, sin embargo, a la hora de programar los grados de pertenencia de cada regla se tuvo un contratiempo debido a que cada regla evalúa de manera simultánea los valores de las dos entradas por lo que fue necesario recurrir a la matemática difusa con el fin de evaluar los dos conjuntos difusos generados por ambas entradas.

Una vez que se estableció el grado de pertenencia de cada regla, se programa la comparación entre los grados de pertenencia para obtener la región resultante y así calcular la señal de salida del controlador.

Para finalizar el diseño, se simula el sistema nuevamente con la excepción de que se usa el algoritmo diseñado reemplazando al Toolbox de Matlab y se verifica su funcionamiento.

En comparación con la respuesta anterior donde se usó el toolbox de matlab se observa en la figura 46 un comportamiento diferente debido a la exactitud en el cálculo de la señal de control por lo que esto afecta el comportamiento del sistema, sin embargo, el error que presenta esta respuesta es del 3.65%. Esto supone un mejor comportamiento del sistema y así mismo nos permitió demostrar que el controlador era funcional y mejoró el funcionamiento de la planta propuesta.

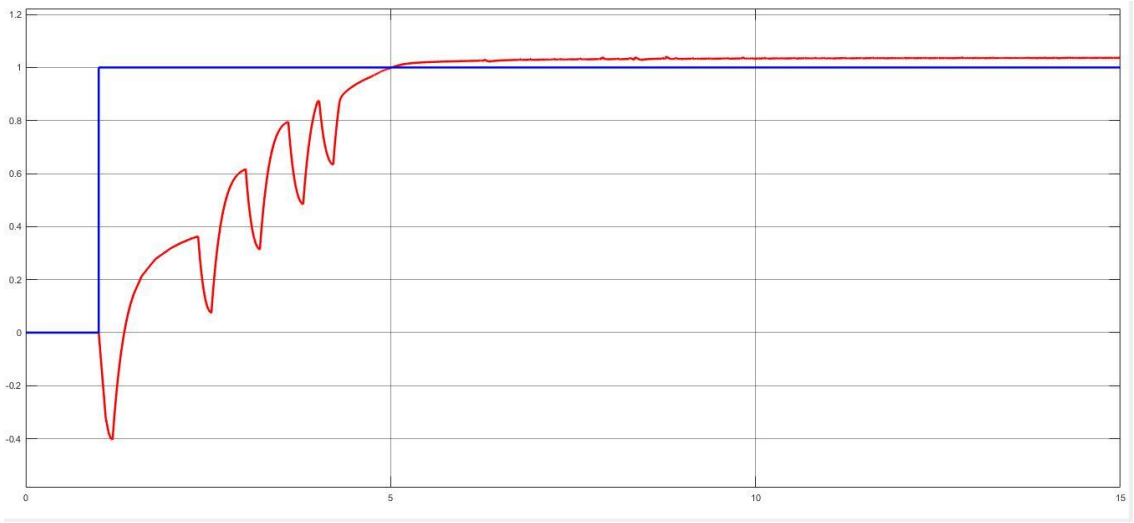


Figura 46. Respuesta del sistema controlado con el algoritmo programado.

En la figura 47. Se expone el diagrama de flujo del funcionamiento lógico del controlador difuso que se desea implementar en el microcontrolador, este diagrama permite tener una idea clara de lo que se desea programar y a su vez es una guía para entender el funcionamiento de este.

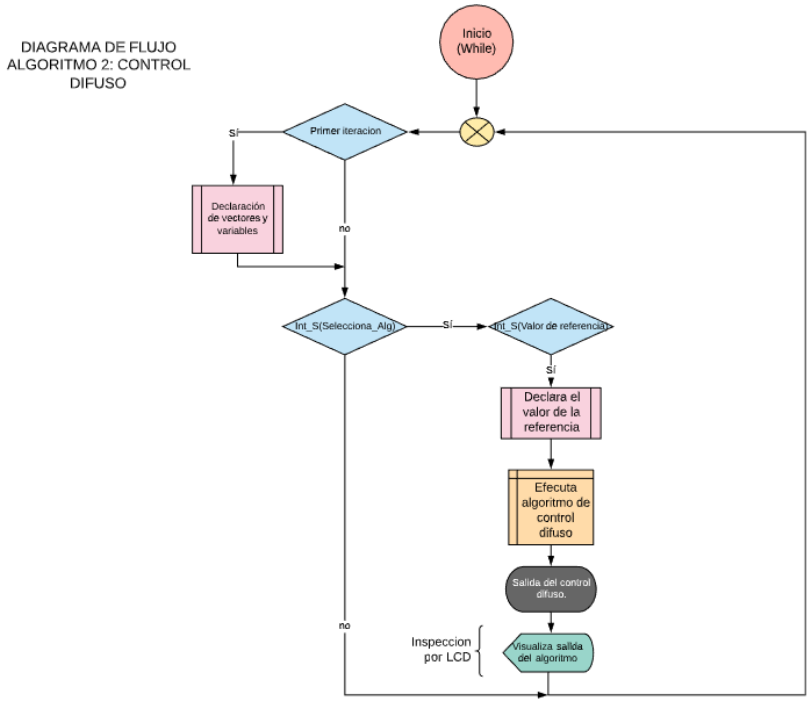


Figura 47. Diagrama de flujo del controlador difuso.

Debido a que ambas respuestas son aceptables se procede a implementar el control en el microcontrolador STM. Para la implementación en el microcontrolador fue necesario discretizar la planta con el fin de poderla implementar en el mismo y realizar todas las simulaciones en el embebido.

Esto se logra por medio del uso de la transformada Z con el fin de obtener el comportamiento en frecuencia o plano S, ya que la función de transferencia se encuentra funcionando de manera continua o en tiempo real por lo que se debe calcular su equivalente en discreto que nos permita representar su comportamiento exacto y así poderla programar en el microcontrolador.

Para lograr esto se hace uso de las técnicas de discretización aprendidas para luego obtener su función de transferencia discreta y luego calcular la ecuación en diferencias para ser ingresada en el código del microcontrolador como se observa en la ecuación 10.

```
Uk = -0.0096*Ck + 0.0098*Ck1 + 1.9319*Uk1  - 0.9324*Uk2;
```

Ecuación 10. Ecuación en diferencias de la planta discreta.

Cabe resaltar que para la obtención de la ecuación en diferencias se debe realizar un muestreo de la salida de la planta, este muestreo se determina según criterios de precisión ya que entre más rápido sea (mayor frecuencia) se obtendrá una mayor precisión en la ecuación en diferencias que describe el comportamiento de la planta, sin embargo, se debe tener en cuenta que esto también se regula según la capacidad que posea el microcontrolador para realizar dicha acción de muestreo.

Una vez que se define la ecuación en diferencias de la planta se procede a realizar los atrasos de las variables que ingresan y salen de la misma. Esto debido a que son atrasos necesarios para describir el comportamiento de la señal de salida de la planta y también para establecer la derivada del error, ya que la derivada del error es una razón de cambio en un tiempo específico (figura 48).

```
Uk2 = Uk1;
Uk1 = Uk;
Ck1 = Ck;

Ek = 1 - Uk;
dEk = (Ek -Ek1)/0.01;
Ek1 = Ek;
```

Figura 48. Atraso de las entradas y salidas de la planta y cálculo de la derivada del error.

Luego de realizar la programación de la planta se procede a agregar el controlador difuso. Para esto es necesario declarar varios vectores que contendrán las funciones de pertenencia de las entradas y de la salida según se muestra en la figura 49.

```
double c[701];
double c1[701],c2[701],c3[701],c4[701];

double B11 [701],B12 [701],B13 [701],B14 [701],B15 [701],B16 [701],B17 [701],B18 [701],B19 [701],B110[701];
double B111[701],B112[701],B113[701],B114[701],B115[701],B116[701],B117[701],B118[701],B119[701],B120[701];
double B121[701],B122[701],B123[701],B124[701],B125[701];

double BB1 [701],BB2 [701],BB3 [701],BB4 [701],BB5 [701],BB6 [701],BB7 [701],BB8 [701],BB9 [701],BB10[701];
double BB11[701],BB12[701],BB13[701],BB14[701],BB15[701],BB16[701],BB17[701],BB18[701],BB19[701],BB20[701];
double BB21[701],BB22[701],BB23[701],BB24[701];
//double BB25[701];

double cenNUM, cenDEN, centroide;
```

Figura 49. Inicialización de los vectores necesarios para el control difuso.

Se inicializan los vectores que contendrán la evaluación de las reglas (beta). A pesar de que el control es discreto puesto que se debe ejecutar en un microcontrolador, el muestreo no tiene relevancia debido a que su comportamiento no se basa en la velocidad de muestreo. Sin embargo, durante la programación del microcontrolador se ejecutó el algoritmo en un ciclo repetitivo (while). Dentro del ciclo while se definieron las funciones de membresía de las dos entradas y de la salida, estas funciones como se mencionó antes se establecen a trozos ya que su comportamiento varía como una figura geométrica.

Como se observa en la figura 50, la primera función de membresía es también a su vez una función a trozos, por esta razón es necesario definirla por rangos usando condicionales. De esta manera se definen las funciones de membresía faltantes para las dos entradas y la salida.

```
while(1){
  /************ FUNCIONES DE MEMBRESÍA PARA LA ENTRADA ************/
  //Negativo
  if(e <= -2.3){
    e1 = 1;
  } else {
    if((-2.3 < e) && (e < -1.6)){
      e1 = -1.4286 * e - 2.2857;
    }else{
      if(e >= 1.6){
        e1 = 0;
      }
    }
  }
```

Figura 50. Declaración de la primera función de membresía de la primera entrada.

Una vez que se establecen las funciones de membresía se procede a evaluar el grado de cumplimiento de cada regla, este proceso se realiza para las reglas que se hayan declarado y permite evaluar que tanto se cumple una según los valores de las dos entradas. Estos valores luego de pasar por las funciones de membresía

toman una pertenencia a cada una de ellas y así mismo los valores de pertenencia permiten calcular que tanto se cumple cada una de las reglas establecidas.

En la figura 51 se tiene la definición de las reglas y se hace por medio de una comparación de los dos conjuntos difusos que se encuentran representados por las dos entradas del controlador. Por lo que se evalúa cuál de las dos entradas es mayor, luego se procede a comparar la entrada con la función de pertenencia de la salida correspondiente a la regla que se está definiendo. Esto se ejecuta de manera repetitiva para obtener el vector correspondiente a cada regla (B11 hasta B125).

```
/************ DEFINICIÓN DE CADA REGLA ************/
for(int n = 0; n < 701; n++){
  //Primera regla
  if(el < del){
    if(c4[n] < el){
      B11[n] = c4[n];
    }else{
      B11[n] = el;
    }
  }else{
    if(c4[n] < del){
      B11[n] = c4[n];
    }else{
      B11[n] = del;
    }
  }
```

Figura 51. Definición de la primera regla.

Una vez que se establecen las 25 reglas que permiten el funcionamiento del controlador difuso, se procede a evaluar que tanto se cumple cada regla según las entradas actuales. Esto permite obtener una región que describe la intersección de la cual se puede obtener el centroide, este es equivalente a la señal del controlador difuso.

Una vez que se han evaluado las reglas y se obtiene la región resultante que se muestra en la figura 52, se almacena en un vector para luego realizar el cálculo del centroide de la siguiente manera.

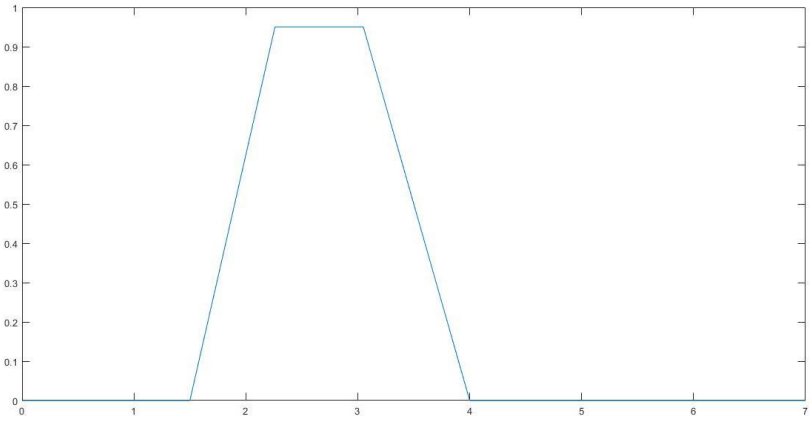


Figura 52. Región resultante de la intersección de las reglas establecidas.

El resultado del cálculo de la señal de control se observa en la figura 53, se ingresa a la planta, de esta se obtiene un valor que se resta con la referencia establecida y así se cierra el lazo de control para lograr que la planta se establezca en el valor deseado.

```
for(int n = 0; n < 701; n++){
  cenNUM = cenNUM + (BB24[n]*c[n]);
  cenDEN = cenDEN + BB24[n];
}

centroide = cenNUM/cenDEN;
Ck = centroide;
```

Figura 53. Cálculo del centroide de la señal de control.

En un primer acercamiento se observa (figura 54) que después de 8000 iteraciones el comportamiento del sistema se aproxima a la referencia. Esto evidencia que el controlador con el objetivo de eliminar el error de estado estacionario. Sin embargo, es importante tener en cuenta el tiempo que esta toma en alcanzar dicho valor, ya que cada iteración dura aproximadamente 22 ms.

Uk	1.08137596	float
i	8000	int

Figura 54. Respuesta del sistema luego de 8000 iteraciones.

4.5 Análisis de desempeño.

Al finalizar el desarrollo del algoritmo se obtuvo el siguiente comportamiento:

ITERACIONES	RESPUESTA DEL SISTEMA		RESPUESTA DEL SISTEMA
0		2500	
500		3000	
1500		3500	
2000		4000	
4500		7000	
5000		7500	
5500		8000	
6000		23000	

6500		57500	

Tabla 2. Comportamiento del sistema según número de iteraciones.

Como se observa en la tabla 2. A medida que las muestras aumentan el sistema se estabiliza por lo que se logra verificar que el controlador difuso funciona correctamente, sin embargo, es necesario que se procesen aproximadamente unas 23000 muestras para que el sistema se logre estabilizar y luego la salida del sistema se mantiene constante en tiempos posteriores.

Para el análisis estadístico se realizaron 50 iteraciones del algoritmo con el fin de determinar las variaciones en los tiempos que le tomaba al microcontrolador realizar cada una de estas. En donde se obtuvo un promedio de las iteraciones de 22.2010765 ms y una varianza de 0.077640569 ms^2 lo cual nos permitió confirmar que al ejecutar el algoritmo 50 veces, los tiempos de iteración tuvieron una variación casi despreciable puesto que era de centésimas de milisegundo.

5. CLASIFICADOR BAYESIANO.

5.1 Problemática

El cultivo de rosas en Colombia en los últimos años se ha incrementado de una manera considerable, llegando a convertirse en el segundo país que más exporta rosas en el mundo. Colombia cuenta con zonas aptas para el cultivo de las mismas, sin embargo, se debe considerar que existen ciertas condiciones ambientales que se deben mantener con el fin de asegurar que el cultivo mantenga la calidad que le ha permitido a Colombia posicionarse en el segundo puesto de exportaciones. (Procolombia, s.f.)

Es por esta razón que en el área agrícola es posible realizar desarrollos en pro de generar una mejora en la producción usando técnicas de supervisión mediante el uso de sensores integrados a un microcontrolador ya que de esta manera es posible asegurar que se mantengan las condiciones necesarias para que el cultivo se pueda cosechar sin poner en riesgo la calidad del mismo.

Una forma de implementar un sistema de supervisión puede ser mediante el uso de un clasificador bayesiano ya que es un algoritmo que permite el procesamiento de los datos y a su vez es capaz de realizar predicciones por medio de la probabilidad de condiciones individuales que al cumplirse permiten que se lleve a cabo el cultivo del producto. Estas condiciones individuales son la temperatura, humedad relativa y luminosidad. Así es posible determinar según la medida de estas tres variables si el cultivo se encuentra en condiciones aptas o no aptas.

Para lograr la implementación de este algoritmo se propone el desarrollo de un ambiente controlado en donde se efectuaron variaciones de temperatura, humedad y luminosidad, estas variaciones se realizaron usando 3 actuadores diferentes como lo son un bombillo halógeno, una resistencia eléctrica para calentar agua y dos ventiladores. Estos se integraron en un vivero cuyo diseño fue propio y de esta manera se recreó un ambiente similar para que el clasificador fuese capaz de determinar según las condiciones actuales si era apropiado cultivar rosas o no.

Se investigó acerca del cultivo de rosas en Colombia, obteniendo las siguientes condiciones climáticas. Las rosas se deben cultivar en temperaturas templadas de aproximadamente 17-25°C, la humedad relativa se debe mantener elevada por encima del 60% y finalmente se deben asegurar niveles de luminosidad elevados ya que la rosa es una planta de día largo por lo que su mayor época de producción es en verano debido a que se tiene una mayor exposición a la luz solar mientras que en invierno la producción baja en comparación. (Infoagro, s.f.)

5.2 Características del algoritmo

El clasificador de Naïve Bayes es un método estadístico que evalúa la probabilidad de que ocurra un evento determinado según eventos anteriores. Modelan un fenómeno mediante un conjunto de variables que se encuentran relacionadas entre ellas. Tomando como base sucesos individuales que componen al evento en su totalidad y de cada evento que ocurre, se debe extraer la información pertinente para que sea comparada con una base de datos que le permite al clasificador inferir la probabilidad de que el suceso sea exitoso o no. (Sucar, 2006)

Por esta razón se debe obtener la información respectiva de cada requisito que se debe cumplir para que el suceso ocurra de manera correcta, esta información suele separarse en rangos con el fin de que estos le brinden al clasificador la capacidad de que pueda evaluar todos los casos posibles que ocurran cada vez que un evento nuevo ingrese. (Sucar, 2006)

Esto se logra por medio de la organización de la información mediante el diseño de unas tablas de frecuencia. Estas tablas permiten determinar la probabilidad total a partir de la información individual y sus rangos correspondientes para que de esta manera sea como el clasificador logre evaluar los casos nuevos ya que toma de la base de datos la información necesaria para determinar la probabilidad total de éxito. (Sucar, 2006)

De la ecuación 11 se logra determinar las dos probabilidades que son cuando el evento se puede cumplir y cuando no es probable que se cumpla.

$$P(c|x) = \frac{P(x|c) * P(c)}{P(x)}$$

Ecuación 11. Calculo de la probabilidad de éxito.

Esto nos permite determinar la probabilidad de la clase al sumar ambos valores tal como se muestra en la ecuación 12.

$$P(x) = P(si|x) * P(si) + P(no|x) * P(no)$$

Ecuación 12. Calculo de la probabilidad total.

Luego de tener ambos valores se calcula la probabilidad real de éxito teniendo en cuenta la probabilidad total usando los valores hallados usando la ecuación 11.

5.3 Antecedentes del algoritmo

Las redes bayesianas permiten la predicción de la ocurrencia de algún suceso específico tomando en cuenta variables o causas que lo pueden llegar a provocar. Un avance significativo fue el diseño de una red que permitiera la predicción para el diagnóstico de diabetes. Esto debido a que las contribuciones que han aportado los sistemas inteligentes no han sido exploradas por completo, adicionalmente se obtuvieron mejores resultados al agregar la variable de la insulina a la red permitiendo mejorar la capacidad de predicción de la misma. (Castrillón, 2017)

Otra aplicación de las redes bayesianas es para la minería de datos puesto que sirve para la clasificación ya que es su mayor distintivo. Su aplicación fue para la simulación del razonamiento para la identificación botánica en donde se evalúan soluciones diferenciales, más específicamente se comparan dos métodos logrando muy buenos resultados ya que por medio de las simulaciones se logró verificar que el comportamiento era el esperado. (Mariño, 2016)

En general las redes bayesianas permiten la clasificación de la información o también predecir acontecimientos, esto tiene un gran uso en el área de la medicina. Se desarrolló una red bayesiana que permite la clasificación a partir del modelado de un sistema experto de traje en los servicios de urgencias. Todo bajo la necesidad de mejorar el servicio médico, mostrando que las redes bayesianas tienen un gran potencial para el uso de sistemas expertos, incluyendo la posibilidad de mejorar el funcionamiento usando datos reales, así mismo también se propone agregar otras categorías de síntomas que se relacionen con otras enfermedades. (Abad, 2007)

Así mismo, se logró la implementación de un clasificador de Naïve Bayes en una tarjeta programable FPGA con el fin de que sea capaz de realizar un control de spam según el contenido del mensaje mediante un sistema numérico logarítmico para el procesamiento de los datos. De la implementación del clasificador se obtuvieron muy buenos resultados ya que la FPGA fue capaz de procesar 117 millones de características por segundo incluyendo la posibilidad de extender el estudio para que el clasificador sea capaz de reconocer el manejo del spam en la bandeja de entrada a la hora de recibir un correo nuevo. (Marsono, 2008)

Actualmente el uso de los clasificadores bayesianos se ha enfocado en el procesamiento de la información de las bases de datos digitales. Este método se ha vuelto muy útil y resulta ser una herramienta ya que de esta manera se puede ordenar la información evitando la pérdida de la misma que se considere importante debido a la cantidad considerable de información que se maneja. La aplicación de un clasificador de Naïve Bayes se realiza a un caso relacionado con el diagnóstico de uso de lente de contacto. (Samuel, 2005)

El proceso consiste en su primera fase en la adquisición de la información de cada caso, luego se diseña una tabla de conteo (tablas de frecuencia) y de esta manera se obtiene la información respectiva a las probabilidades condicionales de los casos posibles. Una vez que se calculan las probabilidades el clasificador les permite ordenar la información de nuevos posibles casos a medida que se ingresen en la base de datos. (Samuel, 2005)

5.4 Funcionamiento

Para el desarrollo de este algoritmo se diseñó un ambiente controlado con la capacidad de variar la temperatura, luminosidad y humedad por medio de diferentes actuadores con el fin de emular el comportamiento del ambiente propuesto que se muestra en la figura 55.

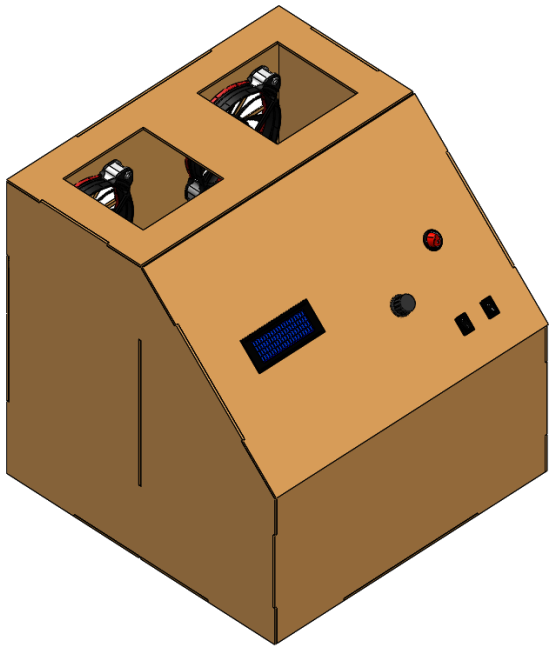


Figura 55. Recreación del ambiente controlado.

Así mismo se usaron 2 sensores diferentes para medir las variaciones en las 3 lecturas de las variables mencionadas y poder aplicar el algoritmo de Clasificador Bayesiano para determinar según una base de datos si la entrada actual cumple con los requisitos para ejecutar una acción determinada.

El ambiente controlado o “planta”, se diseñó por medio del Software SolidWorks ya que con esta herramienta se pueden diseñar las partes de la planta para luego ensamblarlas y según su correcto ensamblaje se procedió a desarrollar los planos

(figura 56) de cada parte para mandar a cortar las piezas y así obtener la base de datos del algoritmo.

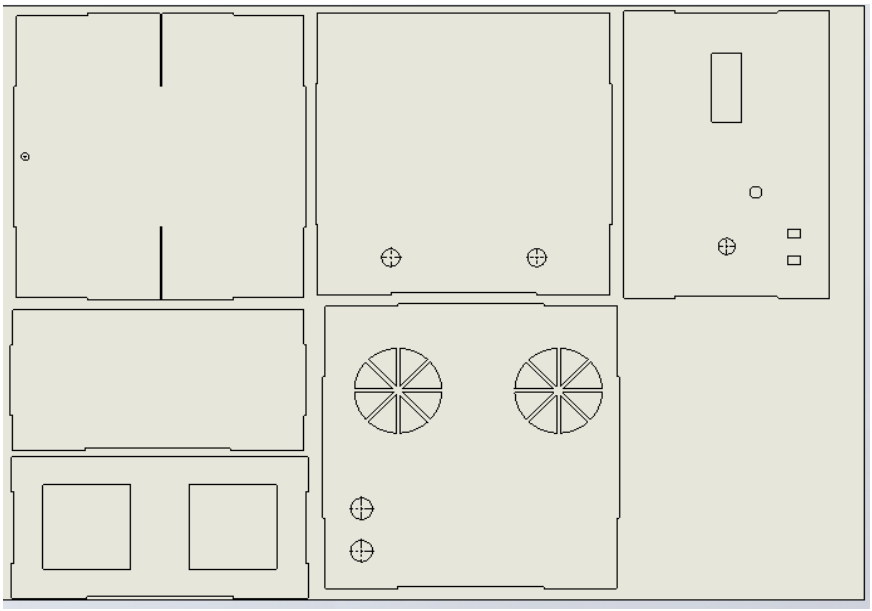


Figura 56. Planos de cada pieza para su corte.

Así mismo era necesario tener la facilidad de observar las medidas de los sensores por lo que se usó una LCD para la visualización de los valores del sensor junto con la programación de una función de escritura para mayor facilidad a la hora de imprimir estos valores.

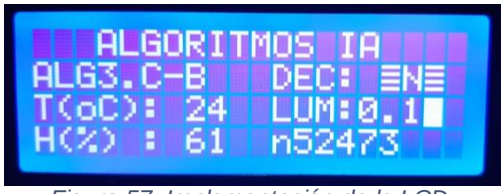


Figura 57. Implementación de la LCD.

El sensor usado para la medición de la humedad y de la temperatura fue el DHT11, (figura 58) este sensor tiene un protocolo propio de comunicación ya que solo cuenta con 3 pines en donde dos de ellos son para alimentación y el tercero sirve para la comunicación. Esta comunicación es simplex debido a que no hay posibilidad de que la información fluya en ambos sentidos entre dispositivos (microcontrolador y sensor). Por esta razón, los desarrolladores del sensor diseñaron su protocolo propio.

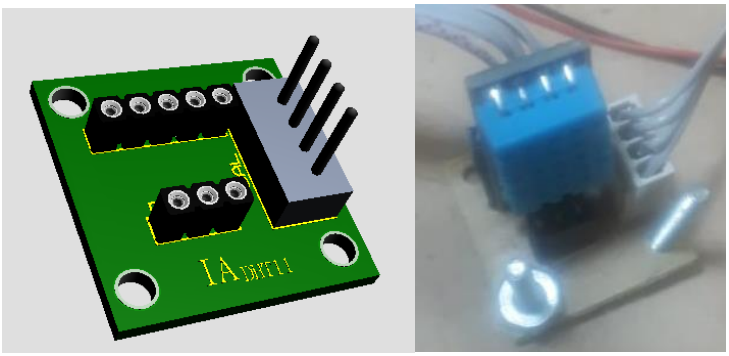


Figura 58. Sensor de humedad y temperatura DHT11 y su baquelita diseñada.

La luminosidad se midió usando una fotorresistencia que se muestra en la figura 59, la cual se conectó en serie a otra resistencia y de esta manera producir una división en el voltaje a medida que la fotorresistencia variaba. Este voltaje variable se acondiciona con el fin de que mantenga un rango específico para que de esta manera no se ponga en riesgo el funcionamiento del ADC del microcontrolador. Una vez el voltaje ingresa al microcontrolador se realiza una conversión de unidades para obtener la luminiscencia en función del voltaje.

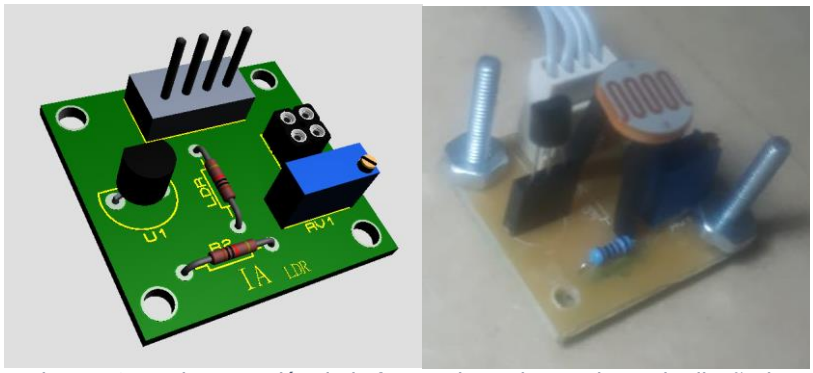


Figura 59. Implementación de la fotorresistencia y su baquela diseñada.

En la figura 60 se presenta el diagrama de flujo del clasificador bayesiano. Inicialmente se realiza la inicialización de las variables y se ejecuta la comunicación con el sensor DHT11 para la medición de la temperatura y de la humedad. Una vez que se hace la inicialización se procede a ejecutar el algoritmo.

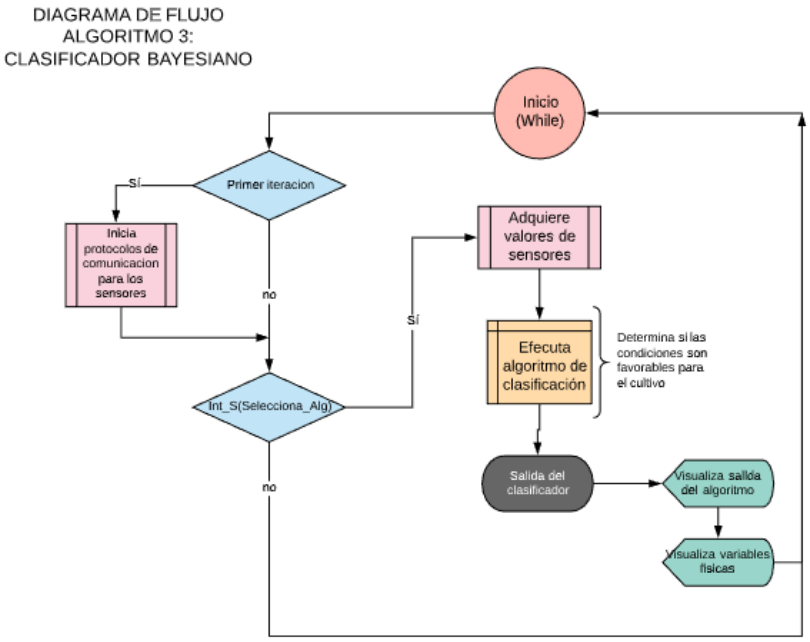


Figura 60. Diagrama de flujo del clasificador bayesiano.

En la figura 61 se observa la inicialización de las variables correspondientes a los rangos de las mediciones realizadas que se encuentran en las tablas de frecuencia

```
float T1S = 5/93.0,T1N = 10/125.0,T2S = 80/93.0,T2N = 25/125.0,T3S = 8/93.0,T3N = 90/125.0;
float H1S = 41/93.0,H1N = 110/125.0,H2S = 33/93.0,H2N = 7/125.0,H3S = 19/93.0,H3N = 8/125.0;
float L1S = 12/93.0,L1N = 85/125.0,L2S = 11/93.0,L2N = 18/125.0,L3S = 70/93.0,L3N = 22/125.0;
```

Figura 61. Definición de las variables con las probabilidades.

Una vez que se capturaron los datos que en total fueron 218 muestras en donde cada una representa un día de cultivo, se procedió a programar el clasificador usando las tablas de frecuencia, las cuales contienen la información de cada evento individual y a su vez se ordenan los eventos por el tipo de variable (figura 62).

TEMP		
	SI(93)	NO(125)
FRÍO	5	10
TEMPLADO	80	25
CALUROSO	8	90

HUM		
	SI(93)	NO(125)
SECO	41	110
PAR. HUM	33	7
HUMEDO	19	8

LUM		
	SI(93)	NO(125)
OSCURO	12	85
NUBLADO	11	18
SOLEADO	70	22

Figura 62. Tablas de frecuencia de cada variable.

Adicionalmente se definieron tres intervalos para cada variable con el fin de obtener más combinaciones y así poder clasificar los posibles sucesos que ocurran. Estas tablas de frecuencia a su vez permiten determinar la probabilidad de que ocurra un evento en específico ya que al ingresar un dato que no se encuentre registrado, el clasificador calcula la probabilidad según la base de datos y la entrada actual.

TEMPERATURA °C	
FRIO	24-30
TEMPLADO	31-36
CALUROSO	37-42

HUMEDAD %HR	
SECO	23-42
PAR. HUM	43-61
HUMEDO	62-80

LUMINOSIDAD V	
OSCURO	1.2-1.6
NUBLADO	1.7-2.1
SOLEADO	2.2-2.6

Figura 63. Definición de los rangos de las variables de medida para el ambiente controlado.

Como se comentó antes y se observa en la figura 63, es importante definir los rangos de las variables debido a que de esta manera se establecen las tablas de frecuencia para el funcionamiento de este y estos rangos son los que se programan en el microcontrolador con el fin de efectuar el funcionamiento del clasificador una vez que ingrese un nuevo dato.

Como se observa en la figura 64, por medio de condicionales se calcula la probabilidad del suceso teniendo en cuenta el rango al cual pertenece la entrada actual dentro del clasificador.

```
if(T <= 31){
  PS = T1S;
  PN = T1N;
}else{
  if(31 < T <= 36){
    PS = T2S;
    PN = T2N;
  }else{
    if(T >= 36){
      PS = T3S;
      PN = T3N;
    }
  }
}
```

Figura 64. Programación del algoritmo del clasificador Bayesiano.

5.5 Análisis de desempeño.

Como se observa en la figura 65. El clasificador determina la decisión de cultivar o no según las entradas actuales que son la temperatura, humedad y luminosidad. Adicional es posible observar el tiempo que se demora en ejecutar el algoritmo, sin embargo, este tiempo es tenido en cuenta a partir del momento en que se ingresa a la función que ejecuta el algoritmo de clasificación. Esta aclaración es necesario realizarla debido a que la captura de datos de humedad y temperatura realizadas por el sensor DHT11 requiere de un tiempo de espera entre muestras de 2 segundos, el cual no es tomado como parte de procesamiento del algoritmo.

Figura 65. Funcionamiento del clasificador bayesiano.

El análisis estadístico realizado para este algoritmo nos permitió determinar los valores promedio de tiempo de ejecución de las diferentes condiciones que se podían recrear dentro de la planta, por medio de los actuadores. Como era de esperarse al modificar estas condiciones, los tiempos de iteración del algoritmo variaban, ya que, según las tablas de frecuencia, las condiciones individuales son diferentes entre sí según la entrada al sistema, por lo que ciertos factores podrían ser obtenidas más rápido que otros. Una vez que se evaluaron diferentes condiciones del sistema y se obtuvieron los tiempos diferentes, fue posible calcular la varianza entre estos, obteniendo un valor de $0.89020895\ \mu s^2$. Y un promedio de tiempo de ejecución de $7.29992\ \mu s$.

6. ALGORITMO GENETICO.

6.1 Problemática

Los algoritmos genéticos han sido usados con múltiples objetivos a lo largo de la historia, recreando en un modelo matemático lo que la naturaleza nos ha mostrado por millones de años. Esto se ha logrado al adoptar funciones para búsqueda y optimización de funciones procurando minimizar o maximizar un proceso, según corresponda. (Guanuchi, 2017)

Normalmente estos algoritmos son ejecutados en computadores de propósito general, donde pueden ser ejecutados junto con muchos otros procesos, siendo esta su ventaja a la hora de implementarlos. Sin embargo, esta metodología no siempre es beneficiosa para el desarrollo de aplicaciones de bajo presupuesto, o donde no se requiera ejecutar más procesos que el mismo algoritmo genético. Es bajo estas premisas que el microcontrolador toma ventaja. Pues al poseer características de un microprocesador, y disminuir en gran medida su costo y consumo energético permite ser implementado en aplicaciones portables que requieran del algoritmo (Mary, 2015). Bajo esta premisa se han implementado múltiples algoritmos pertenecientes a los algoritmos evolutivos (Guanuchi, 2017).

El presente proyecto tiene como objetivo, el de implementar el algoritmo genético en el microcontrolador de la familia STM32xxx, para el control de una planta de segundo orden, obteniendo así un control que pueda supervisar las constantes de control de una planta y adaptarse para obtener una respuesta óptima. Durante el proceso de iteración del algoritmo, el controlador del sistema es modificado en cada prueba con el fin de seguir la dinámica de la planta. Siendo en este caso necesaria la implementación del sistema de control completo dentro del controlador.

La implementación de un sistema de control en un microcontrolador es una de las ventajas que se ha obtenido con la llegada de estos elementos. Esto trajo consigo variantes para la prueba de un controlador en una planta determinada, tal es el caso de la configuración mostrada en la figura 66 donde dos microcontroladores separados cubren la función de simular la planta y el control para la misma; esto es realizado generalmente para identificar el funcionamiento del controlador antes de ser implementado en la planta real.

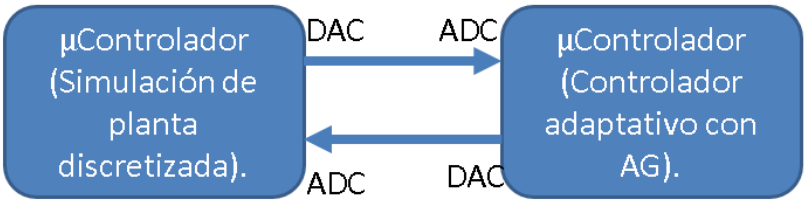


Figura 66. Implementación de un sistema de control discreto para una planta conocida y discretizada.

Sin embargo y con el fin de obtener una respuesta certera, y un mejor análisis de desempeño por parte del algoritmo, se realizó el proceso de evaluación de la planta dentro de un mismo microcontrolador. Con esta estructura se planea obtener la respuesta en tiempos de procesamiento del microcontrolador evaluando el comportamiento de la planta y el control a partir de una misma función dentro del microcontrolador.

Con el fin de dar una mayor versatilidad al algoritmo, se implementaron las constantes del control como variables que pueden ser modificadas por el usuario, permitiendo cambiar la dinámica de la planta con estos valores, y generando un cambio en la salida del algoritmo. Bajo este esquema también se implementaron como variables algunas de las características que definen el proceso del algoritmo, como la población que se generará, y los porcentajes de elitismo, torneo y mutación.

6.2 Características del algoritmo.

Los algoritmos genéticos pertenecen a la una serie de técnicas que emulan el desarrollo de una población de seres vivos a lo largo del tiempo. Se caracterizan por establecer una población aleatoria dentro de rangos definidos por el desarrollador, a medida que la población se reproduce las nuevas generaciones van mejorando según el criterio que haya establecido el programador. Sin embargo, es bueno acotar que un rango de valores comúnmente utilizados para las poblaciones, varía desde los 40 a 100 individuos (Cole, 1998).

A esto se suma el hecho de que, al igual que en las redes neuronales; más no significa mejor. Ya que un número elevado en la población genera una disminución del error más lenta. Esto no solo debe ser tenido en cuenta por la lentitud que toma el algoritmo en encontrar o satisfacer la condición establecidas, sino que también supone un espacio de memoria más grande al momento de crear el programa.

La razón por la cual es un algoritmo sumamente usado y aplicado en la búsqueda de máximos o mínimos globales en un estudio es por su capacidad de tomar a los mejores individuos dentro de una población para convertirlos en miembros de la próxima, y generar así una nueva generación basado en la información de sus predecesoras. Esto extiende las posibilidades de salir de puntos óptimos locales para la solución del problema. La cantidad de individuos de la población que pasan

a la siguiente generación está definida por un porcentaje de la misma, definido como porcentaje de elitismo.

Los mejores elementos se escogen usando una función de idoneidad la cual es establecida por el programador para definir qué tan idóneo es un individuo. Generalmente lo que se busca con estas funciones es encontrar un valor máximo o mínimo al evaluar los elementos de la población. Siendo el caso del presente proyecto, el de minimizar el valor de error en el lazo cerrado de control de una planta de segundo orden.

Una vez que se tienen los mejores individuos, se debe completar la población haciendo un torneo en donde de manera aleatoria se elige el mejor individuo de entre un pequeño grupo, esto se hace dos veces con el fin de obtener dos individuos para que de ellos se genere un nuevo elemento para la siguiente generación. El torneo se realiza la cantidad de veces necesarias para completar la población establecida. Este proceso puede ser observado en la siguiente figura, donde la selección aleatoria para el torneo se realiza tantas veces como población exista.

De la figura 67 se puede observar como de una población total N, se toman aleatoriamente n(%t) individuos, que corresponde al número total de individuos multiplicado por el porcentaje de torneo. Esto se realiza para que del torneo se tenga la oportunidad de escoger a los mejores individuos de una subpoblación, siendo posible que estos elementos no representan una solución inmediata para el problema, pero si permitan redirigir la búsqueda a un mejor hallazgo. Posteriormente se realiza el cruce entre estos individuos para generar un miembro nuevo.

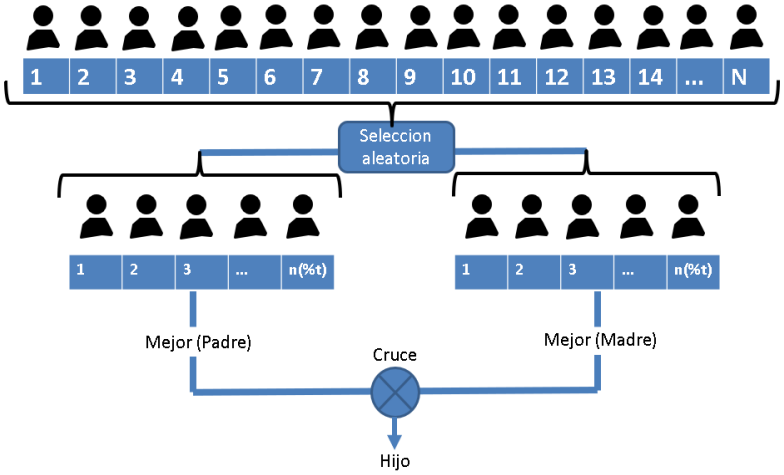


Figura 67. Representación de torneo para la generación de una nueva generación.

Las metodologías de cruce varían según los requerimientos o gustos de quien está realizando el algoritmo. Sin embargo, una buena práctica, es la de darle la

capacidad al mismo de elegir aleatoriamente que cruce realizar entre los individuos clasificados del torneo.

Las técnicas observadas en la figura 68 corresponden a cruce de doble punto (a.), cruce de codificación binaria (b), crossover aritmético genérico (c.) y crossover realizado por un operador AND, no obstante, estas no son todas las técnicas de cruce que se pueden aplicar en un proceso como estos. Siendo utilizado en el presente proyecto la técnica de crossover aritmético; utilizando como salida del cruce un promedio entre los padres.

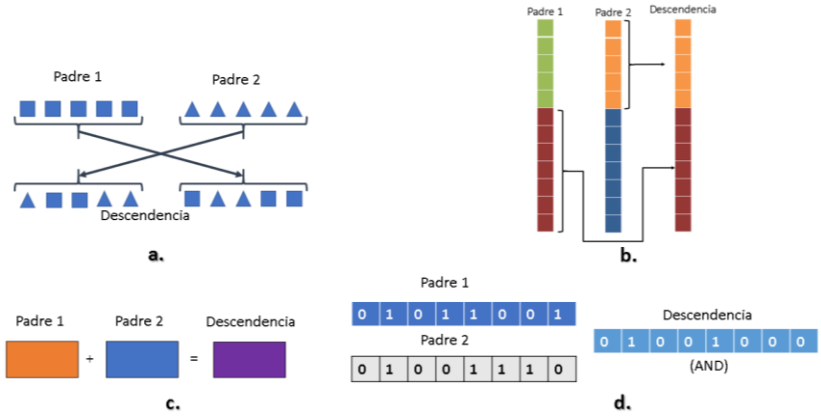


Figura 68. Técnicas de cruce para creación de una nueva generación en un algoritmo genético.

Por último, se realiza una mutación de los hijos obtenidos en el proceso de cruce. Esta mutación también puede variar según se desee, llegando a generar cambios drásticos en el resultado de la generación. Y pueden corresponder al cambio del valor de una posición en el vector del hijo, generando así una diferencia entre el resultado del cruce y la nueva generación. Cabe resaltar que este proceso es solo aplicable para los elementos de la población que se generaron a partir del cruce. Ya que los mejores elementos encontrados al inicio del algoritmo (la élite), pasan intactos a la siguiente generación.

A medida que la población se sigue reproduciendo, llegará el punto en que la mayoría de los elementos adquieran características similares, esto significa que el algoritmo ha completado o se ha acercado lo suficiente a su objetivo. De esta manera se logra tener un barrido completo de todas las posibles soluciones a un problema específico.

Por último, es recomendado siempre trabajar con funciones de idoneidad que sean fácilmente implementadas, ya que el proceso de evaluación se realizará para cada uno de los individuos, y se llevará a cabo el número de generaciones que sean necesarias para que el algoritmo alcance su objetivo. Una mala práctica de este

procedimiento puede generar un aumento significativo en el costo computacional del algoritmo.

6.3 Antecedentes del algoritmo

Si bien estos algoritmos han sido implementados naturalmente por la naturaleza durante millones de años, fue hasta los años 60 que Jhon Holland , un estudiante de la Universidad de Michigan, desarrolló los primeros avances basados en la imitación de la selección natural. Inicialmente este trabajo fue conocido como "planes reproductivos", y no sería sino hasta 1973 cuando Rechemberg definiría las bases de este grupo de estrategias de evolución. (Arranz de la Peña, 2007) (Rivero, 2010). La idea original de Rechemberg nacería realmente desde 1963, cuando él y su equipo de estudio se enfrentaban a problemas de optimización. Un problema aparentemente sencillo de hidrodinámica representaba un reto para los métodos utilizados convencionalmente, lo cual representó un reto para todo el equipo. Es bajo estas circunstancias que Rechemberg intentaría imitar procesos de mutación evolutivos vistos en la naturaleza para encontrar la mejor solución a este problema, mostrando grandes avances al ser implementado (Aguirre, 1998).

Al igual que otros descubrimientos de este carácter, que presentaban mejoras formidables en planteamientos sencillos, no fueron bien recibidos inicialmente por la comunidad científica. Por fortuna sería en 1975 cuando Holland publicaría un libro sobre sistemas adaptativos, el cual servirá de enlace para lo que son las estrategias evolutivas y algoritmos genéticos, tal como los conocemos hoy día.

Luego de haber dado inicio al auge de los algoritmos genéticos y las estrategias evolutivas, se realiza en 1985 la primera conferencia internacional sobre algoritmos genéticos, la cual marcaría la entrada de múltiples investigaciones en este campo de estudio. De esta manera el algoritmo llega a ser implementado para solución de problemas estructurados, como el desarrollo de sistemas de control robusto, redes de comunicación, procesos de manufactura flexible entre otros campos más de la ciencia e ingeniería. (García, 2000) (Yolis, 2003)

6.4 Entrenamiento

Como parte del proceso de entrenamiento del algoritmo genético, es necesario establecer una función que desee ser maximizada o minimizada. Uno de los entornos más comunes en sistemas mecatrónicos, es el control de plantas, y la necesidad de poder hacerlo de manera intuitiva y eficiente. Por esta razón y con el fin de ser implementado en el microcontrolador, se escogió una planta de segundo orden. El modelo que se desea controlar mediante este algoritmo es el de un sistema masa, resorte, amortiguador, la cual se da de la forma:

$$Ft = \frac{1/m}{s^2 + s(b/m) + k/m}$$

Ecuación 13. Función de transferencia que modela un sistema masa resorte amortiguador.

Esta función posee una respuesta con un error en estado estacionario bastante grande. Razón por la que se desea alcanzar la función objetivo a partir del cálculo de las constantes de control mediante el algoritmo genético.

La comparación entre el comportamiento de la planta en tiempo continuo y discreto de la figura 69, fue necesario para comprobar que la respuesta de la planta en tiempo discreto se comportara igual, debido a la necesidad de implementarla de esta forma en el microcontrolador.

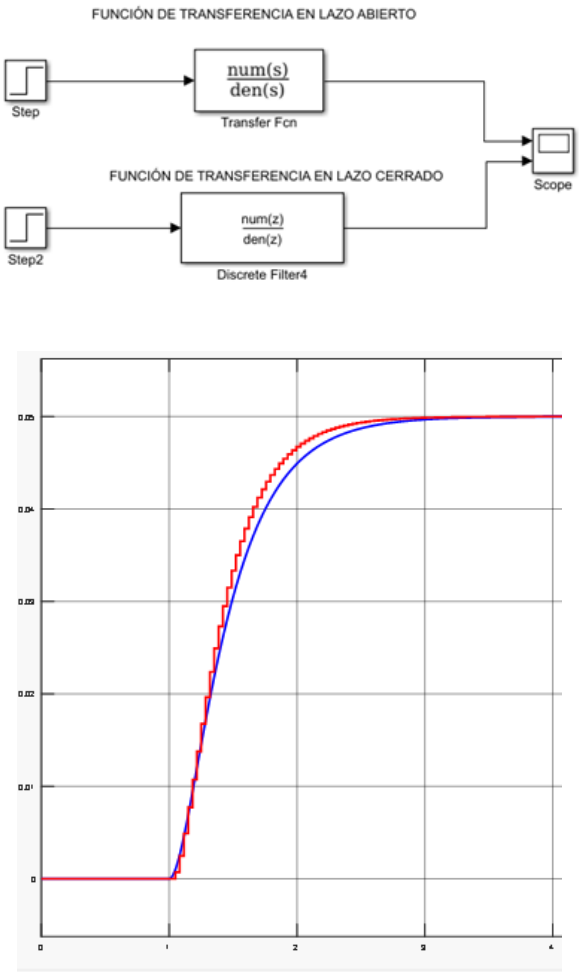


Figura 69. Respuesta de la planta propuesta en tiempo continuo y discreto ante entrada de escalón unitario.

Una vez fue comprobado que las respuestas fueran la misma, se procedió con el proceso de programación del algoritmo en el microcontrolador. Dicho proceso se siguió según el siguiente diagrama de flujo presentado en la figura 70, donde se

expone de manera sencilla, el proceso de iteración del algoritmo genético para el cálculo e impresión de la salida.

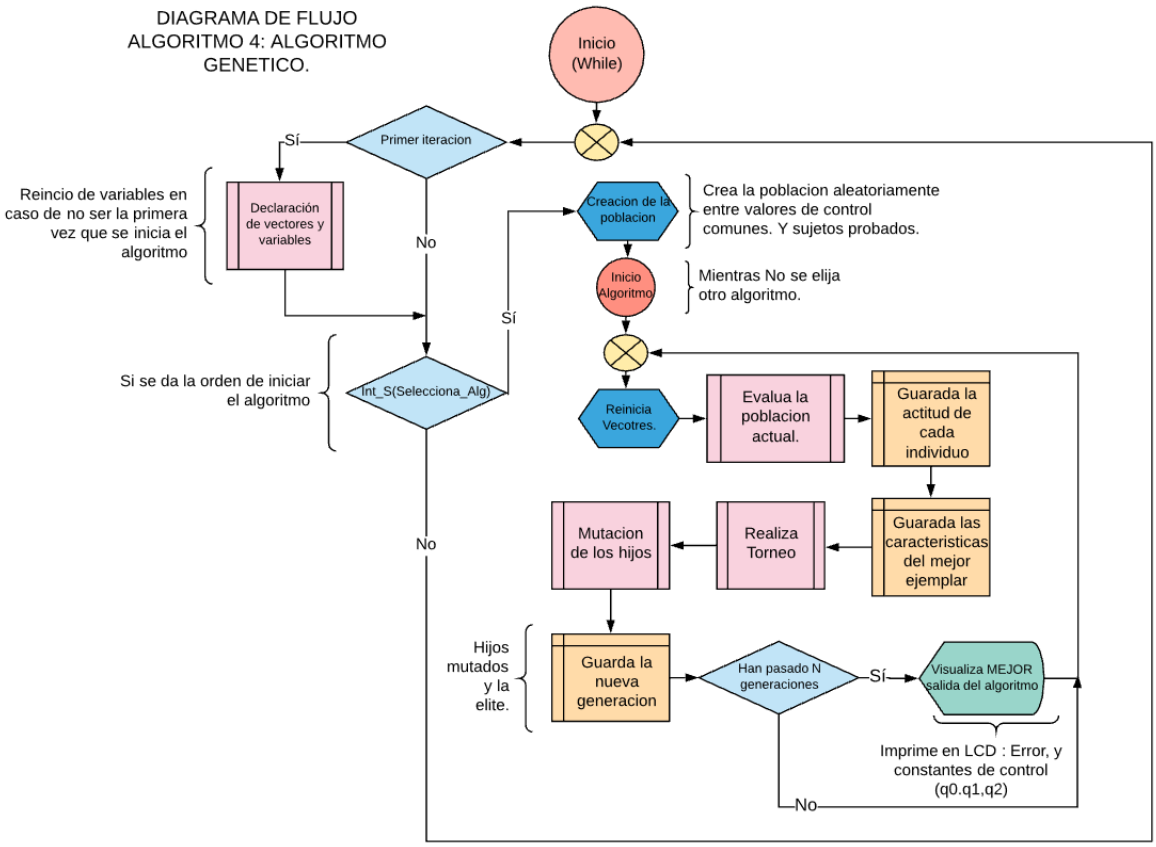


Figura 70. Diagrama de flujo para la programación del algoritmo genético en el microcontrolador.

A continuación, se presenta el proceso de programación en el microcontrolador en donde es necesario conocer cada uno de los pasos que se debe seguir y qué limitaciones se pueden encontrar en el mismo:

Como primera instancia se debe crear la población aleatoria, manteniendo un rango de valores en el cual se debe de ubicar los valores de control. Y a su vez definir el tamaño de la variable que contendrá esta población. Esto se hace de suma importancia en este algoritmo, ya que, si se asigna la creación de las variables dentro del ciclo repetitivo, ocupará la memoria del microcontrolador tras ejecutar cierto número de iteraciones, a continuación, se presenta dicha asignación de variables en la figura 71.

```
#include "ALG4.h"
//!DECLARACION DE VARIABLES ***
float Poblacion[100][3]; // POBLACION MAXIMA CON COLUMNAS PARA KP KI KD
float Nueva_Gen[100][3];

float Madre         [3];
float Padre         [3];
float Hijo          [3];

float Auxiliar    [100];
float VAL_ACTITUD [100];
int   Pos_Elite   [100];
```

Figura 71. Declaración de variables principales para el algoritmo genético.

Una vez definidos los vectores se pueden crear los elementos de la población de manera aleatoria, utilizando una función creada con el propósito de hacer uso de definir números aleatorios. Este paso es posible realizarlo gracias al periférico con el cual cuenta el microcontrolador, el cual fue aprovechado para ser implementado en una función como se observa en la figura 72, siendo la función "IA_ALG1_NUM_ALEATORIO_DELV()", la encargada para dicho propósito..

```
// CREAR MATRIZ DE POBLACION Y LLENARLA CON VALORES ALEATORIOS.
for (int i=0;i<POBLACION_EJ-1;i++){
  for (int j=0;j<3;j++){
    VAL_ALEATORIO_GEN=IA_ALG1_NUM_ALEATORIO_DELV();
    VAL_ALEATORIO_GEN=IA_OPE_INTERP_GEN_DELV(-1500,1500,0,65535,VAL_ALEATORIO_GEN);
    Poblacion[i][j]=VAL_ALEATORIO_GEN;
  }
}
```

Figura 72. Creación de la población en el microcontrolador.

Como se observa, se ejecuta un ciclo repetitivo del tamaño de la población con el fin de crear la población aleatoria, la cual oscila entre valores de -1500 y 1500 (valores que rondan constantes de control comunes para plantas de segundo orden). Otorgando así un espectro amplio del cual generar la población. La interpolación para la obtención del rango de valores ya mencionado es realizado teniendo en cuenta el valor máximo que puede llegar a tomar el registro de creación de números aleatorios.

Posterior a este proceso se evalúa la red, lo cual se logra mediante una función, como se observa a continuación. En la figura 73, la función de evaluación de la población retorna el valor de aptitud del individuo que es guardado en 2 vectores diferentes, uno que servirá para almacenar el total, y el segundo (Auxiliar) que permitirá obtener la Élite de la población. Este valor, pertenece al valor de salida de la función de idoneidad la cual sigue la misma estructura de la Ecuación 8, ya que al igual que en el algoritmo de control difuso, se está realizando un proceso de control para una planta de segundo orden.

```
for (int i=0;i<POBLACION_EJ;i++){ // EVALUAR LA POBLACION (EL CONTROL.)
  q0=Poblacion[i][0]; /* ********************************** */
  q1=Poblacion[i][1]; /* OBTIENE LOS VALORES DE LA POBLACION*/
  q2=Poblacion[i][2]; /* ********************************** */
  VAL_ACTITUD [i]=IA_ALG4_EVAL_POBL_DELV(false);// Obtengo y Guardo factor de actitud
  Auxiliar    [i]= abs(VAL_ACTITUD[i]);
}
```

Figura 73. Evaluación de la población.

Tal como se explicó en la figura 66, se implementó entonces un control discreto para una planta ya discretizada, funcionando esto como la función a evaluar. Y siendo el error (la salida de la planta - el valor del escalón unitario) el valor de actitud a minimizar. Bajo esta premisa, se realiza la evaluación de la población como se muestra en la siguiente figura, donde los valores de la población y su respuesta, modifican el control, que entrega su señal a la planta, y este a su vez la realimenta para cerrar el lazo.

Este proceso se realiza durante 100 ciclos (El equivalente a 4 segundos), valor para el cual la planta tuvo que haber alcanzado la referencia (Un error cercano a 0). El valor de tiempo al cual pertenece el número de iteraciones está relacionado con la discretización del sistema, ya que se empleó un tiempo de muestreo de 0.04 segundos para tal fin, el cual es tomado en cuenta al suponer que cada ciclo representa un cambio de 0.04 segundos.

Una vez evaluada la población (figura 74) se prosiguió con el proceso de elitismo, para el cual también se creó una función que se encarga de comparar el valor de actitud en la variable auxiliar, y de esta manera obtener un número determinado de posiciones en los cuales la respuesta fue cercana a 0. Esta función se logra implementar con ayuda del vector auxiliar creado en la figura 71, ya que la forma de su funcionamiento, es comparando el valor de aptitud de cada individuo de la población, estableciendo al mejor de la población como punto de comparación, y guardando su posición al evaluar toda la población. Siendo esto realizado las veces que sea necesario. Posterior a esto y como se observa en la figura 75, se establecen los valores de las variables que serán visualizadas en la LCD para esa generación.

```
while(cont<=100){ // 100 iteraciones equivale a 4 segundos de la funcion de transferencia
    Ek=REF-Ck;
    if (FUNCION COMPLETA==true){
    else{
      Uk=(q0*Ek)+(q1*Ek1)+(q2*Ek2)+Uk1;                    // Control
      if(Uk>VAL_SATURADOR){Uk=VAL_SATURADOR;}              /***********SATURADOR**************/
      else { if (Uk<-VAL_SATURADOR){Uk=-VAL_SATURADOR;} }  /***********SATURADOR**************/
      Ck=(Uk*NUM_K)+(Uk1*NUM_N)+(DEN_P*Ck1)-(DEN_Q*Ck2);   // PLanta
    }
    Uk1=Uk;
    Ek3=Ek2;Ek2=Ek1;Ek1=Ek;
    Ck3=Ck2;Ck2=Ck1;Ck1=Ck;
    cont++;
```

Figura 74. Evaluación de la población.

```
IA_ALG4_SEL_ELITE_DELV(POBLACION_EJ,NUM_ELITE_EJ);// Sacara los NUM_ELITE_EJ (numero ) mejores (los 4 mejores)abort
ERR_VIS= VAL_ACTITUD[Pos_Elite[0]];
Q0_VIS = Poblacion  [Pos_Elite[0]][0];    /****************    CAPTURA Y OBTIENE  ******************/
Q1_VIS = Poblacion  [Pos_Elite[0]][1];    /************LOS MEJORES COEFICIENTES Y ERROR *************/
Q2_VIS = Poblacion  [Pos_Elite[0]][2];    /**********************DE LA GENERACION*******************/
ERR=ERR_VIS;
if(ERR<0){ERR=ERR*-1;}
```

Figura 75. Obtención de la población más apta.

Por último, se realiza el cruce entre la población torneada. Este proceso de cruce también se realizó mediante 2 funciones diferentes. Una encargada de modificar el vector de la variable global de madre y padre, para luego realizar el cruce y obtener el hijo de este proceso. Posteriormente se muta mediante otra función, que permite finalmente guardar los valores obtenidos en los vectores de la nueva generación. Dicho proceso es implementado en el microcontrolador como se observa a continuación en la figura 76.

```
for (int i=0;i<POBLACION_EJ-NUM_ELITE_EJ;i++){
  IA_GET_MAPADRE_TORNEADO_DELV(1,NUM_TORNEADOS_EJ,POBLACION_EJ);  // Modifica el vector de Madre
  IA_GET_MAPADRE_TORNEADO_DELV(0,NUM_TORNEADOS_EJ,POBLACION_EJ);  // Modifica el vector de Padre
  IA_ALG4_CROSS_PADRES_DELV();                                    // CRUCE (Creacion de los hijos)
  IA_ALG4_MUTACION_DELV();                                        // MUTACION
  Nueva_Gen[i][0]=Hijo[0];
  Nueva_Gen[i][1]=Hijo[1];
  Nueva_Gen[i][2]=Hijo[2];
}
```

Figura 76. Cruce y mutación para la creación de la nueva generación.

Finalmente se complementa la nueva generación a partir de las mejores “n” candidatos, determinados en el proceso de elitismo. Estos individuos se convertirán finalmente en la población para una repetir el proceso de nuevo, mostrando en cada iteración la mejor respuesta del algoritmo, tal como se muestra en la figura 77.

```
int j=0;
for(int i=POBLACION_EJ-NUM_ELITE_EJ;i<POBLACION_EJ;i++){
    Nueva_Gen[i][0]=Poblacion  [Pos_Elite[j]][0];
    Nueva_Gen[i][1]=Poblacion  [Pos_Elite[j]][1];
    Nueva_Gen[i][2]=Poblacion  [Pos_Elite[j]][2];
    j++;
}
memset(Poblacion,0, sizeof(Poblacion));// Elimina valores de poblacion
memcpy(Poblacion, Nueva_Gen, sizeof(Nueva_Gen)+1);//Poblacion=Nueva_Gen
```

Figura 77. Actualización de la población a partir de la nueva generación creada por el algoritmo.

De esta manera el algoritmo se realimenta en cada iteración con la información anterior, para tender finalmente a una población que minimice el error de la planta.

6.5 Análisis de desempeño.

Al momento de ejecutar el algoritmo en el microcontrolador, se obtuvieron una primera serie de resultados, estos compartían un mismo tipo de comportamiento al

momento de ser implementados en el lazo de control de Matlab. Y que al ser probados mostraron un comportamiento como el que se observa en la figura 80.

Como se observa en la figura 78, el comportamiento del control generaba un gran número de oscilaciones alrededor de la referencia. En la figura 80 se puede observar de la misma manera los valores de q0, q1 y q2 que calculo el algoritmo genético para obtener dicha respuesta.

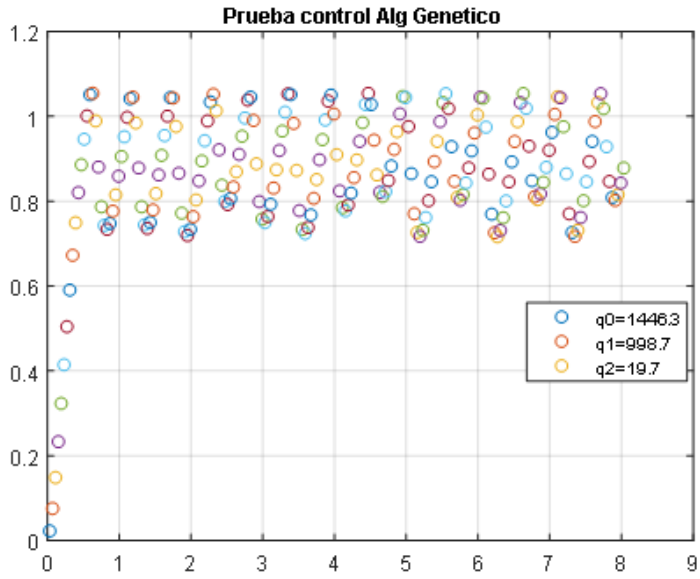


Figura 78. Respuesta de la planta, ante primeras variables de control calculadas con el algoritmo genético.

Fue gracias a esta respuesta que se pudo hacer una modificación en la función de aptitud que se estaba implementando en el microcontrolador, donde el valor que tomaba como respuesta de la planta con el control, estaba siendo siempre el último al terminar los 4 segundos de prueba por individuo de la población. Por esta razón no era posible que el algoritmo determinará cuando la respuesta estaba oscilando o no. Como modificación a la función de aptitud, se realizó la sumatoria de los últimos 20 valores absolutos del error de la planta, para posteriormente dividirlo en el número de muestras (20) y obtener un error que permitiera discernir entre las respuestas oscilatorias o las respuestas no oscilatorias.

Mediante esta última modificación en la función de aptitud del algoritmo genético, se lograron obtener las siguientes constantes que se muestran en la figura 79.

Figura 79. Respuesta del algoritmo genético.

Una vez se obtuvieron estas constantes, y con la posibilidad de que la respuesta presentara posibles inconsistencias, se decidió realizar la implementación de estas constantes en el lazo de control implementado en Matlab. Al implementarla se obtuvo la respuesta que se observa en la figura 80.

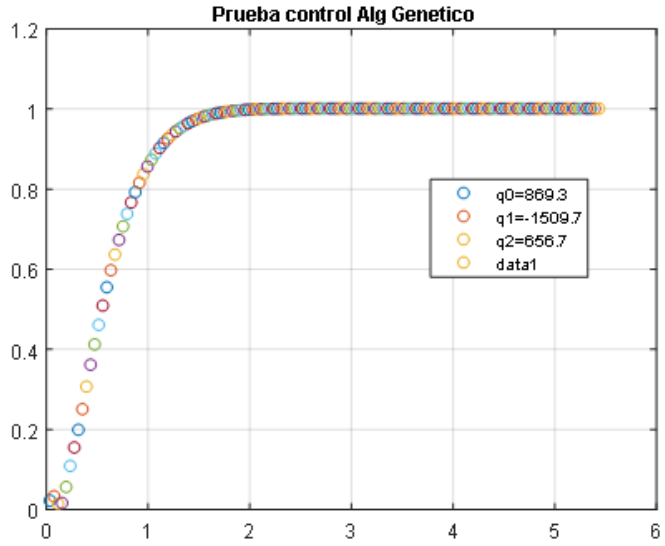


Figura 80. Respuesta de la planta ante control calculado por algoritmo genético.

Como se observa, el control que logró calcular el algoritmo responde de manera consistente y cumpliendo con parámetros de control como es error cercano a 0 en estado estacionario (como muestra la figura 79), y un tiempo de establecimiento entre 1 y 2 segundos. Es necesario acotar que el algoritmo demoró un poco más de un segundo para completar el objetivo. Lo cual puede variar según los parámetros que se modifiquen del algoritmo en sí mismo.

Para el desarrollo del análisis estadístico fue necesario forzar las condiciones del sistema a realizar múltiples intentos de iteraciones, cambiando los porcentajes de mutación y de creación de la población inicial. Ya que una vez el algoritmo alcanzaba un valor ideal de control, este realizaba un proceso sin cambios, lo cual se evidenciaba no solo en los valores de control mostrados, sino también en el tiempo de ejecución, ya que estos se mantenían constantes, siendo de esta manera un promedio de 1083 milisegundos durante la ejecución, y una varianza nula por la repetitividad de sus valores. Sin embargo, y con el fin de simular condiciones para una planta con variables menos conocidas se variaron los parámetros de aleatoriedad .Una vez hecho esto se forzó al algoritmo a trabajar bajo condiciones diferentes en cada iteración, por lo que se obtuvo valores mas relevantes, con un tiempo promedio de $1072.6\,ms$, y una varianza entre estos tiempos de $195.84\,ms$, tiempo que si bien no es significativo puede depender de la planta al ser o no funcional.

7. CONCLUSIONES.

La implementación de metodologías agiles para el desarrollo y repartición de los algoritmos que se programarían en el microcontrolador, fueron de gran ayuda en la detección y solución de errores, así como en la confluencia de cada parte del código final. Sumado a esto, fue también gracias a la implementación de nombres intuitivos y seccionados de funciones, variables y constantes, lo que agilizo el desarrollo del proyecto.

La implementación de los algoritmos de inteligencia artificial propuestos y desarrollados, representaron una serie de obstáculos como lo fueron simples proceso de operaciones matriciales, iteraciones, entre otras. Al ser procedimientos indispensables en el desarrollo de los algoritmos, fue necesaria la creación de funciones que realizaran estos procesos, para facilitar y hacer más entendible los pasos a seguir en cada uno de los algoritmos.

Dentro de las necesidades que satisfacen estas funciones, está el realizar de manera intuitiva el proceso de retro propagación en el algoritmo de redes neuronales, facilitando la multiplicación tanto de matrices como de vectores y procesos de interpolación de estos. Otra de las funciones implementadas bajo esta premisa fue la generación de los números aleatorios, la cual fue acompañada de una función de interpolación lineal, con el fin de establecer los límites del valor aleatorio generado, aplicado en mayor medida en la creación de las poblaciones iniciales para los algoritmos genéticos.

El desarrollo del segundo algoritmo supuso un problema en cuanto se desarrolló en MATLAB debido a que primero debe evaluar funciones a trozos que en programación significan condicionales que separan en rangos las ecuaciones que representan el comportamiento de las funciones de membresía. Adicionalmente el número de condicionales usados para la programación del control difuso es bastante considerable por lo que la mayoría de los vectores no debían exceder un número superior a 1000 espacios de datos ya que se hacerlo se afectaba directamente el funcionamiento del microcontrolador.

Como se observa en la tabla 3. Los tiempos que le toma al microcontrolador para la ejecución de los algoritmos varía entre ellos, esto debido a que de entre los 4 algoritmos programados el que mayor tiempo marcó es el controlador difuso ya que realiza un proceso de evaluación extenso para determinar la salida. Y al ser comparado con el resto de algoritmos, en su funcionamiento se puede determinar que entre mayor cantidad de reglas se necesiten pues será mayor el tiempo que requiera el microcontrolador para ejecutar el control difuso.

A la hora de implementar el clasificador bayesiano en comparación con los demás algoritmos se tuvo un buen desempeño ya que este algoritmo tiene como finalidad

procesar información y (valga la redundancia) clasificarla y de esta manera determinar cuándo se debe realizar el cultivo según las condiciones presentes. Por esta razón es que el clasificador no toma mucho tiempo durante su ejecución lo cual además se corrobora mediante su funcionamiento cuando se ingresa un dato que se espera que la respuesta sea la deseada.

Debido a que los algoritmos genéticos requieren de varias iteraciones para mejorar la población, se debe procurar programar el algoritmo de tal manera que en cada iteración se requiera de la menor cantidad de código ya que esto puede llegar a comprometer el funcionamiento del algoritmo y a su vez generar un mal funcionamiento del microcontrolador.

El desarrollo de la investigación nos permitió determinar la importancia que tiene el programador durante el diseño de cada algoritmo ya que el microcontrolador posee muchos módulos que de no ser bien configurados afectarán el funcionamiento en general del mismo, por esta razón se recomienda una buena documentación a la hora de realizar aplicaciones en el microcontrolador STM32F746ZGT.

Algoritmo	Tiempos		Potencia
Algoritmo vacío.	$39653\ ns$		$12mW$
Redes neuronales.	$50988.78\ ns$		$13mW$
Control difuso.	$22.20108\ \text{ms}$		$15mW$
Clasificador bayesiano.	$7.29992\ \mu s$		$14.5mW$
Algoritmo genético.	$1072.6\ ms$		$14.5mW$

Tabla 3. Mediciones de los tiempos que demora el procesamiento de cada algoritmo implementado.

Bibliografía

Abad, M. (2007). Aplicación de redes bayesianas en el modelado de un sistema experto de triaje en servicios de urgencias médicas.

Aguirre, A. (1998). Estrategias evolutivas: La versión Alemana del algoritmo genético (Parte 1). 38-45.

Aldabas, E. (2002). Introducción al reconocimiento de patrones mediante redes neuronales.

Arranz de la Peña, J. (2007). Algoritmos genéticos.

Castrillón, O. (2017). Sistema bayesiano para la predicción de la diabetes. 161-168.

Chávez, J. (2012). Utilización de la inteligencia artificial en el diagnóstico patológico de edificaciones de valor patrimonial . 297-305.

Chen, J. (2017). Systematic development of an optimized real-time embeddes control platform., (pp. 1075-1080).

Cole, R. (1998). *Clustering with geretic algorithms.*

Cruz, I. (2007). Redes neuronales recurrentes para el análisis de secuencias. *Revista Cubana de ciencias informáticas* , 48-57.

Egmont, M. (2002). Image processing with neural networks. 2279-2301.

Fowler, M. (2014). *Martin Fowler*. Retrieved from https://martinfowler.com/bliki/UnitTest.htm

García, M. (2000). Automatic loop-shaping of QFT robust controllers via genetic algorithms. 603-608.

Giménez, A. (2014). Coontrol system development for a Raman Spectrometer using microcontroller technology. *Journal of applied research and technology*, 139-144.

Gonzáles, M. (2008). Segmentación de imágenes utilizando la transformada Watershed. 223-228.

Guanuchi, O. (2017). *Diseño de un sistema inteligente que permita estimar y simular la autonomía de un vehículo eléctrico en rutas urbanas de la ciudad de Cuenca.*

Hock, G. (2011). Development of wireless controller area network using low cost and low power consumption ARM microcontroller for solar car application., (pp. 244-248).

Infoagro. (n.d.). *El cultivo de la rosa*. Retrieved from http://www.infoagro.com/documentos/el_cultivo_rosa.asp

Kalezhi, J. (2018). Microcontroller-Based Monitoring and Controlling of Environmental Conditions in Farming. 284-288.

Kouro, S. (2002). Control mediante lógica difusa.

Lin, C. (2010). Comparison of artificial neural network and logistic regression models for predicting mortality in elderly patients with hip fracture. 869-873.

Mackinnon, T. (2000). endo-testing: Unit testing with mock objects. Extreme programming examined.

Mariño, S. (2016). Simulación del razonamiento en el proceso de identificación botánica basado en redes bayesianas. *Revista de la escuela de perfeccionamiento en investigación operativa*.

Marsono, M. (2008). Binary LNS-based naïve Bayes inference engine for spam control: Noise and fpga implementation. 56-62.
Martin, I. (2003). Análisis y predicción de la serie de tiempo del precio externo del café colombiano utilizando redes neuronales artificales. *Universitas Scientiarum*, 45-50.
Mártinez, C. (2007). *Uso de las técnicas de preprocesamiento de datos e inteligencia artificial en la clasificación del riesgo bancario.* Bogotá.
Mary, R. (2015). *Microcontrollers*. Retrieved from http://www.engineersgarage.com/microcontroller.
Matich, D. (2001). Redes Neuronales: Conceptos básicos y aplicaciones. In M. D.
Morcillo, C. (2011). Lógica difusa, una introducción práctica.
Ocaña, A. (2009). *Aprendizaje y comportamiento basados en el funcionamiento del cerebro humano.*
Ordoñez, J. (2016). Detector de bordes de imágenes usando un microcontrolador ARM. *Revista de ciencia y tecnología*, 30-35.
Pacheco, M. (2017). Identificación de sistemas no lineales con redes neuronales convolucionales.
Peñas, M. (2013). Aplicación de la lógica difusa en el ámbito de las energías renovables.
Perez, J. (2002). Modelos predictivos basados en redes neuronales recurrentes de tiempo discreto.
Ponce. (2010). *Inteligencia artificial con aplicaciones a la ingeniería.*
Procolombia. (n.d.). *Procolombia*. Retrieved from http://www.procolombia.co/node/1255
Ramos, O. (2016). Reconocimiento de patrones vocálicos mediante la implementación de una red neuronal artificial utilizando sistemas embebidos. *Información tecnológica*, 133-142.
Rivero, M. (2010). Introducción a los algoritmos genéticos y la programación genética.
Rodríguez, M. (2009). Diffused logic as a tool for interpreting clean production data in the agriculture area. 101-105.
Runeson. (2006). A survey of unit testing practices. *IEE Software, vol 23*, 22-29.
Sáenz, R. (2018). Dual-accumulator softcore 8-bit microprocessor designed to be used on FPGAs. *Tecnura*, 40-50.
Samuel, P. (2005). *El clasificador Naïve Byes en la extracción del conocimiento de bases de datos.*
Sánchez, N. (2016). Diseño de un sistema de reconocimiento automático de matrículas de vehículos mediante una red neuronal convolucional.
Sandoval, T. (2016). Uso de redes neuronales artificales en predicción de morfología mandibular a través de variables craneomaxilares en una vista posteroaterior. 21-28.
Santos, P. (2018). Multi-Objetive Genetic Algorithm Implmentedon a STM32F Microcontroller.
Segue Technologies. (2014). *Seguetech*. Retrieved from https://www.seguetech.com/the-benefits-of-unit-testing/

Serrano, E. (2017). Sequential microcontroller-based control for a chemical vapor deposition process. *Journal of applied research and technology*, 593-598.
Spanish Small Satellites International Forum. (2019).
Sucar, L. (2006). Redes bayesianas. 77-100.
Vechet, S. (2014). Building a scaled model of autonomous convoy powered by ARM mbed microcontrollers., (pp. 711-714).
Wen, U. (2009). A review of Hopfield neural networks for solving mathematical programming problems. *European Journal of Operational Research*, 675-687.
Yolis, E. (2003). Algotimos genéticos aplicados a la categorización automática de documentos.
Zhu, H. (1997). Software unit test coverage and adequacy. 366-427.