UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE ELECTROTECNIA Y COMPUTACIÓN

Trabajo monográfico para optar al título de

Ingeniero en Computación

Propuesta de diseño de aplicación informática para la simulación de un controlador difuso de temperatura de un horno panadero.

Autor:

Carlos de Jesús Canales López (2001-10442)

Tutor:

MSc. Ing. Humberto Francisco Zepeda Palacios

Abril 2019. Managua, Nicaragua.

DEDICATORIA

Dedico este trabajo monográfico, primeramente, a Dios, el Ser Supremo que ha creado todo lo que existe; lo visible y lo invisible, lo que es, lo que fue y lo que será. Se lo dedico en agradecimiento por darme la inteligencia y dejarme adquirir el conocimiento necesario para realizar este importante trabajo en mi vida.

Asimismo, lo dedico a mis padres que, con su educación y su esfuerzo me guiaron por buenos caminos para obtener una formación integral y de calidad. Les agradezco su paciencia, su amor y su cariño.

También, deseo dedicarlo a todos mis compañeros de la carrera de Ingeniería en Computación, con los cuales pasé momentos agradables y duros, trabajando y aprendiendo de nuestros maestros. Especialmente, lo dedico a todos mis compañeros que formaron grupos de trabajo con mi persona para dar un paso hacia delante y llegar hasta este momento, a nuestra meta.

Finalmente, dedico este trabajo a todos mis profesores, desde los que me enseñaron las primeras letras hasta los que me han enseñado la profesión de la ingeniería. Gracias por su enseñanza, por sus ejemplos, sus correcciones y sus llamados de atención cuando no hacía el trabajo correcto. Especialmente, deseo agradecer a todos mis profesores de Ingeniería en Computación, por compartir sus conocimientos y sus experiencias con nosotros sus alumnos, para formarnos en esta bella profesión.

Sea Dios bendiciéndolos a todos. Amén.

RESUMEN

En este trabajo monográfico, el autor aplica sus conocimientos adquiridos en el área de ingeniería en computación para diseñar un programa informático que simule el control de temperatura de un horno. Dicho control se lleva acabo aplicando la técnica de control difuso, la cual forma parte de una de las ramas de la Inteligencia Artificial llamada, Lógica Difusa; adicionalmente, de la Ingeniería de Control.

El trabajo completo consiste en el diseño de un control difuso de temperatura, el cual se codifica en el lenguaje de programación C#, de la plataforma .Net, y simula el controlador de temperatura de un horno real. El programa diseñado tiene la capacidad de comunicarse con una tarjeta electrónica de control de la plataforma ARDUINO, cuya función es hacer de interfaz entre el programa y el hardware necesario para llevar a cabo la simulación; es decir, el sensor de temperatura y el actuador que adecúa el nivel de voltaje para que la temperatura del horno sea la deseada.

Se hace necesario señalar que la aplicación diseñada no es para la simulación de varios controladores de temperatura; sino para, un solo controlador difuso, el cual se ha diseñado previamente al diseño de la aplicación. Por tanto, la aplicación muestra el comportamiento de ese único controlador diseñado, y no se debe ver como un simulador general de controladores difuso de temperatura o de cualquier otro tipo de controlador difuso.

Contenido

1. INTRODUCCIÓN

La ingeniería de control es un campo donde confluyen muchas áreas de la ciencia y la ingeniería; entre ellas tenemos, a la ingeniería eléctrica, la ingeniería electrónica, la ingeniería mecánica, la ingeniería en computación, la ingeniería de procesos, y otras más. Esta rama de la ingeniería, la ingeniería de control ha permitido a la humanidad ir evolucionando y avanzando en sus procesos productivos.

Existen diversas técnicas que se implementan en los procesos de control moderno. La teoría de control clásico ha permitido desarrollar una amplia gama de proyectos de alta calidad, los cuales son obtenidos a través del modelado de procesos. Hoy en día, nuevas técnicas de control han surgido para ampliar los conocimientos de la teoría de control. La Inteligencia Artificial es una rama de la ciencia y la ingeniería que, actualmente está haciendo grandes aportes al campo de la ingeniería de control, con sus tres ámbitos de estudio; las Redes Neuronales, Lógica Difusa y Los Algoritmos Genéticos.

Este trabajo monográfico se basa en la teoría del Control Difuso y en la teoría de la Ingeniería de Software. El Control Difuso es la implementación de la Lógica Difusa en el control de procesos, de los cuales es difícil o nula la obtención de un modelo matemático que los describa a la perfección. La Ingeniería de Software nos brinda varios métodos para llevar a cabo el desarrollo de una aplicación informática de manera eficiente.

Para describir un sistema es necesario desarrollar un esquema que modele, de manera visual y estructural, la interacción de sus componentes funcionales. Por tal razón, se hace necesario siempre presentar la arquitectura del sistema en general.

En este documento se presentan los objetivos de este trabajo monográfico, la justificación de este, su marco teórico; además de, los resultados obtenidos durante el desarrollo del trabajo, y las conclusiones y recomendaciones finales.

2. OBJETIVOS

Objetivo General

- Diseñar un software sencillo que se comunique con una tarjeta electrónica de la plataforma Arduino, de tal manera que, ambos en conjunto, permitan simular un controlador difuso para el control de temperatura de un horno panadero.

Objetivos Específicos

- Definir los requerimientos funcionales y no funcionales del software a diseñar, a fin de, enfrentar las necesidades de los usuarios y las funcionalidades que debe tener el sistema.

- Establecer una arquitectura que describa la estructura del software; adicionalmente que, describa la comunicación entre el software del controlador difuso y la tarjeta electrónica de la plataforma Arduino, así como, los dispositivos electrónicos necesarios para la simulación de sensores y actuadores.

- Diseñar una interfaz gráfica de usuario que permita la interacción de un usuario con el simulador del controlador difuso.

- Codificar cada uno de los módulos funcionales del controlador difuso, para posteriormente integrarlos en un solo paquete funcional.

- Realizar pruebas y correcciones al software para comprobar que el controlador difuso, que éste simula, funciona de manera adecuada.

3. JUSTIFICACIÓN

La Inteligencia Artificial es un campo de la ciencia y la ingeniería que en los últimos años ha tomado gran auge. Una de las ramas de la Inteligencia Artificial es la Lógica Difusa, por cuyos avances se han podido diseñar nuevos tipos de controladores (controladores difusos) diferentes a los diseñados con la teoría del control clásico. Los controladores difusos han demostrado ser tan buenos y eficientes como los controladores diseñados con la teoría clásica de control. Sin embargo, la bibliografía y la documentación es bastante escasa acerca de este tema; por tal razón, el autor de este protocolo monográfico ha visto la necesidad de que su Alma Mater cuente con un documento que introduzca, de manera sencilla, a los estudiantes o personas interesadas en este tema.

Por otra parte, la mayoría de los horneros de las panaderías no conocen las técnicas para medir temperatura y tampoco conocen sus unidades de medida, ya que siempre han trabajado con la sensación de calor de su piel. Por tal razón, trabajar con grados centígrados o grados Fahrenheit les es muy complicado y confuso; sin embargo, les es más fácil trabajar con términos como temperatura baja, temperatura media, temperatura alta, etc.

Otras de las razones, para los panaderos y horneros, por las que les es más fácil trabajar sin temperaturas exactas son, la variabilidad de la composición química de las materias primas que utilizan, y la cantidad de producto que se introduce a cocer en el horno. No es lo mismo hornear un solo pastel de media libra a 180 grados Celsius que hornear seis pasteles a la misma vez.

En consecuencia, el autor de este protocolo monográfico ve la oportunidad de aprovechar la necesidad de documentación en el ámbito de la lógica y controladores difusos, aunada al problema del control de temperatura de los hornos de las panaderías, para realizar su trabajo monográfico, creando así una base de conocimiento para los estudiantes de la Universidad Nacional de Ingeniería (UNI) en el área de la lógica difusa y los controladores difusos. Adicionalmente, tomando como caso de estudio el problema del control de temperatura de los hornos; ejemplificando,

así, el uso de controladores difusos, proponiendo una solución, simulada, a este problema. Todo esto, aplicando los conocimientos de desarrollo de software que ha adquirido mientras ha cursado la carrera de Ingeniería en Computación.

La lógica difusa es adecuada para realizar la simulación del control de temperatura de un horno utilizado en panadería, ya que el proceso de cocción de un horno panadero no puede ser modelado, porque existen muchos parámetros que varían de receta en receta, varían según las materias primas que se utilicen; e incluso, varían de panadería en panadería.

4. MARCO TEÓRICO

Este trabajo monográfico está fundamentado en dos pilares teóricos, los cuales son; Lógica Difusa e Ingeniería de Software. Estos dos campos de la ciencia y la ingeniería se han unido en este proyecto y, han permitido llevar a cabo todo lo necesario para poder alcanzar cada uno de los objetivos planteados previamente.

4.1. Lógica Difusa

El ser humano hace sus razonamientos y deducciones a partir de la información que le brindan sus cincos sentidos. Sin embargo, la información que obtenemos de nuestros sentidos no es exacta, es una información aproximada. Cuando nos acercamos caminando a una persona, vemos que poco a poco la distancia se va achicando, pero con cada paso, no sabemos la distancia exacta a la cual está dicha persona. De la misma manera, cuando sentimos la sensación de calor, decimos cualquiera de estas expresiones; hace un poco de calor, hace mucho calor, el calor está muy fuerte. No obstante, nunca decimos; el calor es de 35 °C, o el calor es de 32 °C. La lógica de nuestros sentidos es difusa, aproximada, no exacta. Por ende, los humanos todo el tiempo hacemos uso de la Lógica Difusa.

(Ponce, 2010) afirma lo siguiente: “La lógica difusa es una rama de la Inteligencia Artificial que le permite a una computadora analizar información del mundo real en una escala entre lo falso y verdadero. Los matemáticos dedicados a la lógica en la década de 1920 definieron un concepto clave: *todo es cuestión de grado*. La lógica difusa manipula conceptos vagos como “caliente” o “húmedo” y permite a los ingenieros construir televisores, acondicionadores de aire, lavadores y otros dispositivos que juzgan información difícil de definir”.

La lógica Crisp o tradicional, es una lógica binaria, solo tiene dos valores, ya sean; verdadero o falso, pertenece o no pertenece, 1 o 0, o cualesquiera otros dos valores que se definan. Por otro lado, la Lógica Difusa es una lógica multivaluada; lo cual significa que, tiene múltiples valores, por lo general se escogen valores entre 0 y 1, pero puede escogerse otro rango.

4.1.1. Conceptos Fundamentales

Para entender la lógica difusa, es necesario comprender algunos conceptos fundamentales que se utilizan en este campo.

En la lógica matemática se usa el concepto de ***conjunto***, el cual es una cantidad de elementos que se agrupan de acuerdo con ciertos criterios. Según (Vicent, 2014), “un conjunto es una colección de objetos bien especificados que poseen una propiedad común”.

El ***universo de discurso*** es el dominio donde existen uno o más conjuntos, incluyendo el conjunto nulo o vacío. (Vicent, 2014) lo define de la siguiente manera: “conjunto X de posibles valores que puede tomar la variable x”.

El ***grado de pertenencia*** es el valor que toma un elemento de un conjunto difuso de acuerdo con un criterio de selección.

Un ***conjunto difuso*** es aquel cuyos elementos pueden tener valores o grados de pertenencia comprendidos en el rango de 0 a 1.

Una ***Función de pertenencia*** describe el grado de pertenencia de un elemento en uno o varios conjuntos difusos. Su dominio comprende los valores que están de 0 a 1. Existen funciones matemáticas, ya definidas, que pueden ser usadas como funciones de pertenencia. En la siguiente figura se muestran las funciones más comunes.

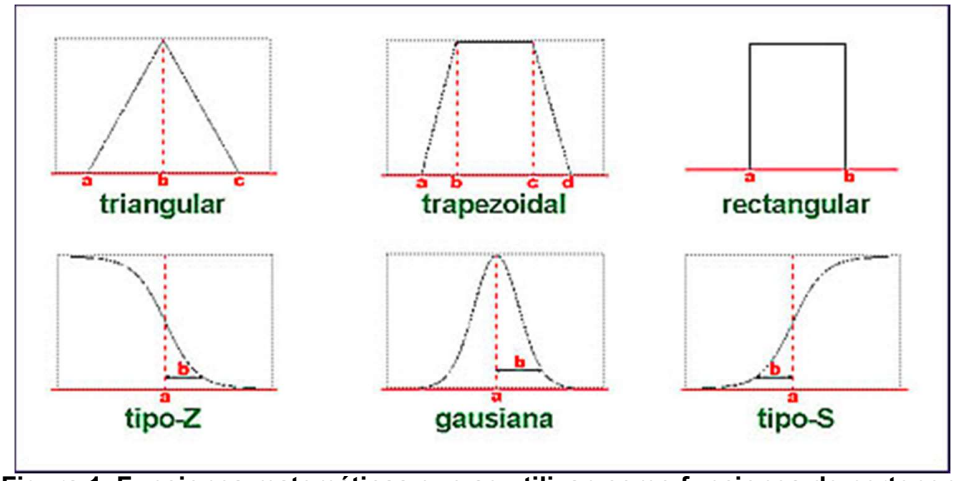


Figura 1. Funciones matemáticas que se utilizan como funciones de pertenencia.

(Reina, 2008) establece lo siguiente: "La función de pertenencia describe toda la información contenida en un conjunto difuso. Dado un conjunto difuso A, se espera que su función de pertenencia cumpla con las siguientes propiedades:

1. *Propiedad de Normalidad*:
   Debe existir $x \in A$ tal que $A(x) = 1$. Si se tiene que $A(x) < 1$ para todo $x \in A$, al conjunto difuso se le llama subnormal.
2. *Propiedad de Monotonicidad:*
   Si x1 es más próximo a x que el valor x2, entonces $A(x1) > A(x2)$
3. *Propiedad de Simetría*:
   Si x1 y x2 son equidistantes de x, entonces $A(x1) = A(x2)$".

Una ***variable lingüística*** es una variable cuyo valor no es numérico; sino que, es una palabra o una frase. Por ejemplo, supongamos que necesitamos una variable lingüística para expresar el estado de la temperatura del ambiente. Esta variable lingüística puede tomar los valores de; muy frío, frio, fresco, caliente, muy caliente, etc.; es decir, expresamos el valor de la variable con una o varias palabras. Las

variables lingüísticas son utilizadas para diseñar controladores difusos, ya que cada una de ellas define o representa a un conjunto difuso.

(Ponce, 2010) define lo siguiente: “Una variable lingüística adopta valores con palabras que permiten describir el estado de un objeto o fenómeno; estas palabras se pueden representar mediante conjuntos difusos. Una variable numérica toma valores numéricos, por ejemplo: edad = 65, mientras que una variable lingüística toma valores lingüísticos: edad es “viejo”. Todos los valores lingüísticos forman un conjunto de términos o etiquetas.”

4.1.2. Operaciones Básicas de los Conjuntos Difusos

Los conjuntos difusos tienen cuatro operaciones fundamentales. Estas operaciones son intersección (AND), unión (OR), complemento y producto cartesiano.

Sean $\mu_A(x)$ y $\mu_B(x)$ dos conjuntos difusos:

La operación de intersección u operación AND en los conjuntos difusos se define de la siguiente manera.

$$\mu_{A\cap B}(x) = Min\{\mu_A(x),\ \mu_B(x)\}$$

*La intersección de dos conjuntos difusos **A** y **B** se define como el valor mínimo de sus grados de pertenencia.*

La operación de unión u operación OR en los conjuntos difusos se define de la siguiente manera.

$$\mu_{A\cup B}(x) = Max\{\mu_A(x), \mu_B(x)\}$$

*La unión de dos conjuntos difusos **A** y **B** se define como el valor máximo de sus grados de pertenencia.*

La operación de complemento en los conjuntos difusos se define de la siguiente manera.

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x)$$

El complemento de un conjunto difuso se define como la diferencia de 1 menos el grado de pertenencia del conjunto.

El producto cartesiano de dos conjuntos difusos $\mu_A(x)$ y $\mu_B(y)$ se define de la siguiente manera.

$$\mu_R(x, y) = \mu_{AXB}(x, y) = Min(\mu_A(x), \mu_B(y))$$

El producto cartesiano es una relación difusa entre dos conjuntos difusos, la cual da como resultado el valor mínimo de sus grados de pertenencia.

Hay que señalar que, en el producto cartesiano difuso todos los elementos de $\mu_A(x)$ se combinan con todos los elementos de $\mu_B(y)$ para formar pares ordenados, al igual como se hace en el producto cartesiano de los conjuntos crisp. Sin embargo, la diferencia del resultado radica en que, de cada par ordenado difuso se saca el valor menor de los elementos del par ordenado. Ejemplo:

Sea A y B dos conjuntos difusos tales que, A = {0.5, 0.3} y B = {0.1, 0.6, 0.8}, entonces el producto cartesiano AXB queda a como sigue.

AXB = Min(A(x), B(y))

Min(A(x), B(y)) = Min {(0.5, 0.1), (0.5, 0.6), (0.5, 0.8), (0.3, 0.1), (0.3, 0.6), (0.3, 0.8)}

AXB = {0.1, 0.5, 0.5, 0.1, 0.3, 0.3}

4.1.3. Normas Triangulares

Norma T

La norma T establece que, la intersección de dos conjuntos difusos se puede representar como una función que tiene como dominio el producto cartesiano de dos

valores de membresía que van de 0 a 1, y su imagen es un valor de membresía de 0 a 1.

$$\mathbf{T: [0, 1] X [0, 1] \longrightarrow [0, 1]}.$$

Una función de norma T debe cumplir con las siguientes propiedades:

- Conmutatividad: $T\ (a, b) = T\ (b, a)$
- Monotonía: $T\ (a, b) \leq T\ (c, d)$ *si* $a \leq c$ *y* $b \leq d$
- Asociatividad: $T\ (a, T\ (b, c)) = T\ (T\ (a, b), c)$
- Elemento identidad: $T\ (a, 1) = a$

Las siguientes funciones cumplen con la norma T; y, por ende, se utilizan para realizar la intersección de dos conjuntos difusos.

- Función Mínimo: T(x,y) = Min(x,y)

El resultado de esta función es la comparación de dos valores de pertenencia ***X*** y ***Y***, devolviendo el valor más pequeño de ambos.

- Función producto: Prod(x,y) = x · y

El resultado de la función producto es la multiplicación de dos valores de pertenencia ***X*** y ***Y***.

- Función de Lukasiewicz: W(x,y) = Max (0, x+y-1)

La función de Lukasiewicz da como resultado, el valor máximo de la comparación de dos grados de pertenencias, uno de valor 0 y otro grado de pertenencia que resulta de la suma de dos grados de pertenencias ***X*** y ***Y*** menos 1. Si el resultado del segundo grado de pertenencia es menor que cero; entonces, el valor máximo será 0, pero si el resultado es positivo, entonces el valor máximo será el valor de Y.

Norma S (Co-norma T)

La norma S es la contra parte de la norma T, y establece que, la unión de dos conjuntos difusos se puede representar como una función que tiene como dominio el producto cartesiano de dos valores de membresía que van de 0 a 1, y su imagen es un valor de membresía de 0 a 1.

$$\mathbf{S: [0, 1] X [0, 1] \longrightarrow [0, 1]}.$$

Una función de norma S, al igual que las funciones de norma T, tiene que cumplir con algunas propiedades.

- Conmutatividad: S(a, b) = S(b, a)
- Monotonía: S(a, b) ≤ s(c, d) si a ≤ c y b ≤ d
- Asociatividad: S(a, S(b, c)) = S(S(a, b), c)
- Elemento identidad: S(a, 0) = a

Nótese que, las funciones de norma S cumplen con las mismas propiedades de las funciones de norma T, pero la diferencia está en la propiedad de Elemento de identidad, donde en las funciones de norma T, esta propiedad se evalúa con 1; mientras que, con las funciones de norma S, dicha propiedad se evalúa con 0.
Las funcione más utilizadas que cumplen con la norma S son las que a continuación se describen.

- Función Máximo: S(x, y) = Max(x, y)

El resultado de esta función es la comparación de dos valores de pertenencia ***X*** y ***Y***, devolviendo el valor más grande de ambos.

- Función Suma-Producto: Sum-Prod(x,y) = x + y - x · y

El resultado de esta función es la suma de dos valores de pertenencia ***X*** y ***Y***, al cual se le resta el producto de esos mismos grados de pertenencia.

- Función Operación dual de Lukasiewicz: W*(x,y) = Min(1, x+y)

El resultado de esta función depende de la suma de dos grados de pertenencia ***X*** y ***Y***, siendo éste el segundo término de la función. Esta suma se compara con el valor de 1; si el resultado de la suma es mayor que 1, el resultado final será 1, pero si el resultado es menor que 1, entonces, el resultado final será el resultado de la suma.

4.1.4. Controladores Difusos

Un controlador difuso es un tipo de controlador diferente a los controladores diseñados con la teoría clásica de control. Los controladores clásicos están diseñados en base a un modelo matemático que describe el sistema a través de una ecuación matemática. Por otra parte, los controladores difusos están diseñados para que trabajen con Lógica Difusa. Este tipo de controladores se utilizan cuando los sistemas a controlar son difíciles o imposibles de modelar a través de un modelo matemático. El diseño de los controladores difusos, generalmente, se basa en la experiencia de uno o varios operadores del sistema; los cuales han operado de forma manual o semiautomática el sistema en su conjunto, y, por ende, lo conocen en su totalidad; y adicionalmente, conocen las salidas que éste genera a determinadas entradas.

(Vélez, 2000) afirma que, "los controladores difusos son sistemas expertos especiales; cada uno emplea una base de conocimientos expresada en términos de reglas de inferencia difusas relevantes y una máquina de inferencia apropiada para resolver un problema dado de control. Los controladores difusos son capaces de utilizar el conocimiento de operación del operador humano".

En el diseño de un controlador difuso, hay que tomar en cuenta cuatro procesos esenciales. El primero es el corazón del proceso de control, y los tres últimos son los pasos que realiza un controlador difuso para controlar uno o más procesos. Tales procesos son los que a continuación se mencionan:

1. Definición de reglas, 2. Fusificación, 3. Inferencia de reglas, 3. Defusificación.

Definición de reglas

De acuerdo a (Passino & Yurkovich, 1998), en un sistema difuso, el mapeo de las entradas a las salidas se caracteriza por un conjunto de reglas de la forma:

C*ondición* → *acción*, o en forma de modus ponens (Si-Entonces).

En este proceso, el diseñador del controlador difuso define las reglas de control del proceso a manejar. Para definir estas reglas es necesario contar con el apoyo de un experto; que, por lo general, es una persona que ha operado manualmente el proceso y tiene un dominio amplio del mismo.

Las reglas definidas siguen una estructura condicional *si…entonces,* a cómo podemos apreciar a continuación.

Si A entonces B

Donde A se conoce como *antecedente* o *premisa* y B se conoce como *consecuente*. El antecedente indica la ocurrencia de uno o más eventos, y el consecuente indica los pasos a seguir cuando tales eventos ocurren. Un ejemplo de una regla para un controlador difuso de temperatura ambiente puede ser la siguiente.

Si la temperatura ambiente está muy fría entonces encender el calentador al máximo.

Fusificación

La fusificación es el proceso que realiza el controlador difuso al momento de sondear los parámetros del proceso a controlar, haciendo uso de sensores. Sin embargo, los valores que los sensores arrojan son numéricos, debido a la naturaleza no difusa de dichos parámetros. Por tanto, el controlador difuso debe traducir estos valores numéricos a valores lingüísticos, para poder realizar su trabajo de control. En resumen, la fusificación es el proceso de traducir un valor numérico a un valor lingüístico. (Chen & Pham, 2001) indican que, el propósito de la fusificación es hacer que la señal de entrada sea compatible con la base de reglas de control difuso que está en el corazón del controlador.

Inferencia de reglas

El mecanismo de inferencia tiene dos tareas básicas. La primera es, determinar el grado de relevancia de cada regla en la situación actual, según las entradas; y la segunda, sacar conclusiones haciendo uso de las entradas actuales y la información en la base de reglas (Passino & Yurkovich, 1998). Por tanto, luego de que el controlador difuso ha realizado el proceso de fusificación, éste debe tomar una o varias decisiones. Para tomar las debidas decisiones, éste debe analizar las reglas definidas por el diseñador y compararlas con los valores lingüísticos que obtuvo del proceso de fusificación. Este proceso también se conoce como evaluación de reglas.

Defusificación

Una vez que el controlador difuso toma la o las respectivas decisiones, este genera uno o varios valores lingüísticos, los cuales son la respuesta de salida del controlador a las entradas obtenidas. Pero, como sólo el controlador entiende los valores lingüísticos, éste debe traducir tales valores a valores numéricos entendibles para los dispositivos conectados en su o sus salidas. Entonces, el proceso de defusificación traduce los valores lingüísticos generados por el controlador a valores numéricos. Se puede percibir que, el proceso de defusificación es lo inverso al proceso de fusificación.

(Jimenez, 2007) sostiene que, el proceso de defusificación es un mapeo de un espacio de acciones de control difuso definido sobre un universo de discurso de salida en un espacio de acciones de control no difuso (valores precisos).

Para el proceso de defusificación se pueden hacer uso de cuatro tipos de métodos: Máximo Central, Máximo Más Pequeño, Máximo Más Grande y Centroide.

Máximo Central

Este método calcula el punto medio de todos los grados de pertenencia de máximo valor. Debido a que todos coinciden en el mismo valor de grado de pertenencia,

entonces se decide escoger el punto medio de tales valores. En otras palabras, se obtienen todas las abscisas (X) cuyas ordenadas (Y) tienen valor máximo; de ellas se suman la abscisa menor y la abscisa mayor, luego el resultado se divide entre dos.

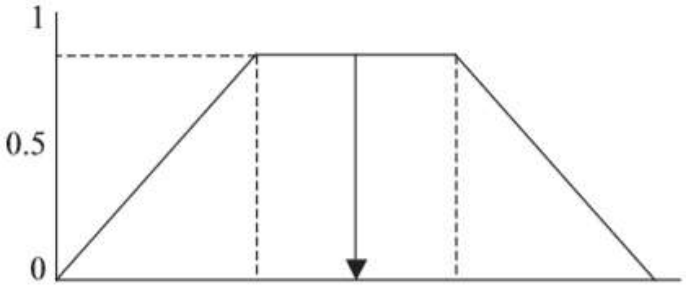


Figura 2. Máximo central.

$$Salida = \frac{x_{menor} + x_{mayor}}{2}$$

Máximo Más Pequeño

Se escogen todos los grados de pertenencia cuyos valores sean máximos, luego se escoge entre ellos el punto más pequeño. O de otra manera, se obtienen todas las abscisas (X) cuyas ordenadas (Y) son el máximo valor, luego se obtiene la abscisa más pequeña; en cuyo caso es, la posicionada más a la izquierda.

$$Salida = x_{menor}$$

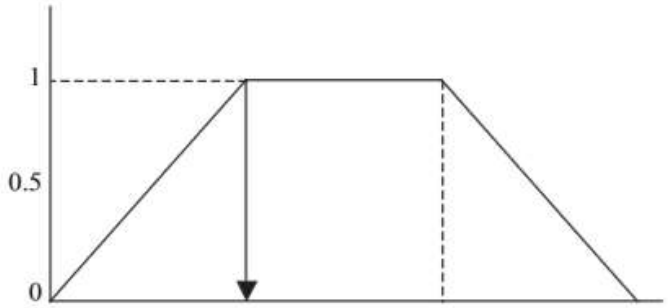


Figura 3. Máximo más pequeño.

Máximo Más Grande

Se escogen todos los grados de pertenencia cuyos valores sean máximos, luego se escoge entre ellos el punto más grande. Otra forma de expresarlo es, se obtienen todas las abscisas (X) cuyas ordenadas (Y) son el máximo valor, luego se obtiene la abscisa más grande; en cuyo caso es, la posicionada más a la derecha.

$$Salida = x_{mayor}$$

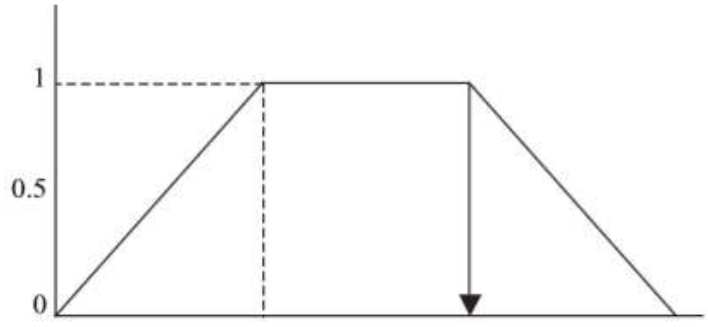


Figura 4. Máximo más grande.

Centroide

Al método del Centroide también se le conoce como Centro de gravedad, y se calcula en forma discreta de la siguiente manera:

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

De forma continua se calcula con la siguiente ecuación:

$$Salida = \frac{\int_a^b \mu(x) * x dx}{\int_a^b \mu(x) dx}$$

Donde μ(x) es el grado de pertenencia de x; o de otra manera, y(x) = μ(x).

Hay que notar una cosa, el denominador es la sumatoria o integral del producto de las ordenadas (Y) por sus respectivas abscisas (X); y el denominador es la sumatoria o integral de las ordenadas.

Existen varios tipos de controladores difusos; sin embargo, los más conocidos y utilizados son dos, los controladores de tipo Mamdani y los controladores de tipo Sugeno.

Controlador de Mamdani

El controlador de Mamdani lleva a cabo cuatro procesos: fusificación, corte, unión y defusificación.

- En el proceso de fusificación, se calcula el grado de pertenencia de un valor medido en cada función de pertenencia de los antecedentes de cada regla de

control, y haciendo uso de la operación de intersección (AND) se obtiene el de valor mínimo de pertenencia en cada regla de control.

- Luego, con los valores de pertenencia mínimos de cada regla, se hace un corte de la función de pertenencia de los consecuentes, también en cada regla de control, a través de la función mínimo, donde en el resultado estarán solo los valores de la función de pertenencia del consecuente que se ubiquen por debajo del valor de pertenencia mínimo calculado en el proceso de fusificación; mientras que, los valores de la función de pertenencia del consecuente que se ubican por encima de ese valor se desechan. Debido a que estos dos procesos, fusificación y corte, se hacen para todas las reglas de control, resulta así una función de corte para cada consecuente, y por ende para cada regla.
- El proceso de unión une a través de una operación de conjunción o unión (OR) a todas las funciones de corte de los consecuentes, resultando en una sola función, donde solo se obtienen los valores máximos de los cortes de los consecuentes.
- El proceso final de defusificación calcula un único valor de esa función de unión resultante, por medio de alguno de los métodos de defusificación descritos previamente, obteniendo así la salida final.

Controlador de Sugeno

El controlador de Sugeno es una simplificación del controlador de Mamdani. Este tipo de controlador se utiliza únicamente cuando se conoce la curva de control del proceso. Las diferencias entre el controlador de Sugeno y el de Mamdani son:

- Las reglas si-entonces en el controlador de Sugeno tienen antecedentes, pero no consecuentes. El consecuente es sustituido por una función lineal ya conocida por medio de la curva de control del proceso.
- Se hace un proceso de fusificación igual que en el controlador de Mamdani, pero no se realizan los procesos de corte, unión y defusificación.

- Con la fusificación se obtienen los valores de pertenencia de las entradas, estos se evalúan en la función que les corresponde, y el resultado de ésta se multiplica por el valor mínimo de ambos grados de pertenencia. Esto se hace en cada regla.
- Para obtener el valor de salida, se suman todos los valores obtenidos a través del ítem anterior; es decir, se suman todos los productos de la evaluación de los grados de pertenencia en la función correspondiente con el valor mínimo de dichos grados de pertenencia. El resultado de esa suma se divide con la suma de todos los grados de pertenencia mínimos obtenidos en cada regla; obteniendo así un valor escalar que será la salida del sistema.

4.2. Ingeniería de Software

La ingeniería de software nos brinda una serie de herramientas que nos ayudan y facilitan el proceso de planificación y desarrollo de una aplicación informática.

"La ingeniería de software es una actividad para la solución de problemas. Se usan los modelos para buscar una solución aceptable. Esta búsqueda es conducida por la experimentación. Los ingenieros de software no tienen recursos infinitos, y están restringidos por presupuestos y tiempos de entrega". (Bruegge y Dutoit, 2002, p. 5).

Según (Pressman, 2005), la ingeniería de software es una tecnología estratificada, cuyo enfoque debe estar comprometido con la calidad. Por tal razón, los diseñadores de aplicaciones informáticas deben de tener en cuenta estándares y normas que indican acciones a seguir para conseguir un producto que cumpla con los mínimos requisitos de calidad.

Los modelos de desarrollo de software señalan una serie de acciones que se deben llevar a cabo para obtener un producto final – el software. (Moreno, 2010) indica lo

siguiente, *"estos modelos son abstracciones que, si bien no describen detalladamente el proceso a seguir para el desarrollo de software, representan diferentes estrategias o enfoques para abordar este problema"*.

4.2.1. Metodología RUP (Rational Unified Process)

La metodología RUP tiene tres características fundamentales en el proceso de desarrollo de software. La primera son los ***Casos de Uso***, que describen los servicios que el usuario requiere del sistema; es decir, la interacción entre el usuario y el sistema (Pérez, 2011). La segunda característica es que es ***Centrada en la Arquitectura***, se muestra la visión común del sistema completo, describe los elementos del sistema que son necesarios para comprenderlo y desarrollarlo (Maida & Pacienzia, 2015). La tercera característica refiere a que esta metodología es ***Iterativa e Incremental***; (Belloso, 2009) afirma que, RUP apuesta por procesos iterativos e incrementales en donde el trabajo se divide en partes más pequeñas o mini proyectos permitiendo el equilibrio entre casos de uso y arquitectura.

El ciclo de vida de desarrollo RUP está dividido en cuatro fases de trabajo.

1. Inicio o concepción: Definición de la visión, los objetivos y el alcance del proyecto. En esta fase se obtiene una lista de casos de uso y una lista de los factores de riesgo del proyecto (Maida & Pacienzia, 2015).
2. Elaboración: Tiene como principal finalidad completar el análisis de los casos de uso y definir la arquitectura del sistema (Maida & Pacienzia, 2015).
3. Construcción: Sobre la arquitectura de base, se irá construyendo el sistema en una serie de iteraciones cortas. El objetivo de cada iteración será obtener un incremento del sistema en funcionamiento y preparar el conjunto de casos de uso que se abordarán en la siguiente (Moreno, 2010).
4. Transición: En esta fase el sistema software se entrega a los usuarios finales para sus respectivas pruebas en un entorno real y su documentación (Pérez, 2011).

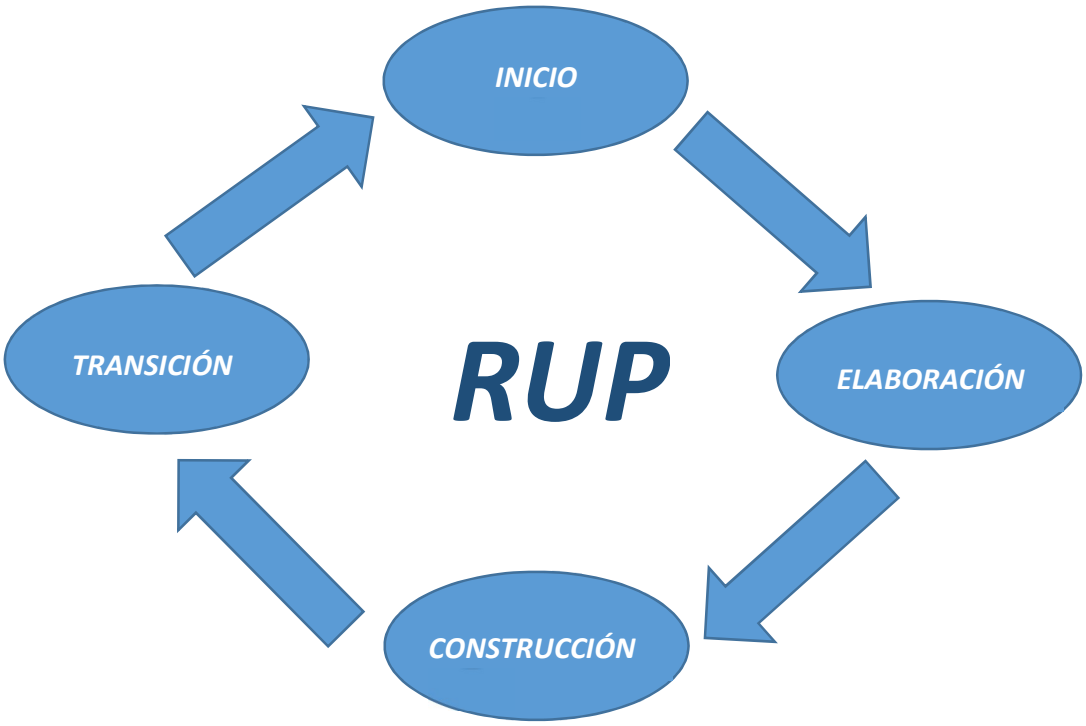


Figura 5. Fases de trabajo del ciclo de vida de desarrollo de la metodología RUP.

4.2.2. Modelado de sistemas

El modelado de un sistema nos permite visualizar la manera en que éste está estructurado, nos permite apreciar sus componentes funcionales básicos y la interacción entre los mismos, la forma en que éstos interactúan con otros componentes externos al sistema y especialmente con los usuarios.

Durante la actividad de requerimientos y diseño del sistema, éstos pueden ser modelados como un conjunto de componentes y de relaciones entre estos componentes. Esto se puede ilustrar gráficamente en un modelo arquitectónico del sistema, el cual proporciona al lector una visión general de la organización del sistema, (Sommerville, 2005).

Arquitectura del sistema

Los grandes sistemas siempre se descomponen en subsistemas que proporcionan algún conjunto de servicios relacionados. El proceso de diseño inicial que identifica estos subsistemas y establece un marco para el control y comunicación de los subsistemas se llama diseño arquitectónico. El resultado de este proceso de diseño es una descripción de la arquitectura del software, (Sommerville, 2005).

Podemos afirmar que, la arquitectura muestra la relación de todos los componentes; es decir que, es una descripción estructural y relacional del sistema en conjunto. (Pressman, 2005) afirma que, la arquitectura del software es una representación que permite que un ingeniero del software: 1) analice la efectividad del diseño para cumplir con los requisitos establecidos, 2) considere opciones arquitectónicas en una etapa en que aún resulta relativamente fácil hacer cambios al diseño, y 3) reduzca los riesgos asociados a la construcción del software.

Lenguaje Unificado de Modelado (UML)

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas (Rumbaugh, Jacobson, & Booch, 2000).

Adicionalmente, (Rumbaugh, Jacobson, & Booch, 2000) afirman que, UML capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Un sistema se modela como una colección de objetos discretos que interactúan para realizar un trabajo que finalmente beneficia a un usuario externo. La estructura estática define los tipos de objetos importantes para un sistema y para su implementación, así como las relaciones entre los objetos. El comportamiento dinámico define la historia de los objetos en el tiempo y la comunicación entre objetos para cumplir sus objetivos. Modelar un sistema desde varios puntos de vista, separados pero relacionados, permite entenderlo para diferentes propósitos.

Los componentes estáticos UML son:

- Diagrama de clases
- Diagrama de casos de uso
- Diagrama de componentes
- Diagrama de despliegue

Los componentes dinámicos UML son los siguientes:

- Diagrama de estados
- Diagrama de actividades
- Diagrama de secuencia
- Diagrama de colaboración

4.3. Plataforma ARDUINO

Arduino es una plataforma electrónica de código abierto, basada en hardware y software fáciles de usar. Está dirigida a cualquiera que realice proyectos interactivos. Una tarjeta Arduino sensa el ambiente por medio de señales de entradas recibidas de uno o varios sensores, y afecta sus alrededores por medio del control de luces, motores y otros actuadores.[1]

Las tarjetas Arduino poseen entradas y salidas con las cuales sensan y controlan un determinado ambiente. Estas entradas y salidas pueden ser analógicas o digitales, eso dependerá de la configuración que el usuario haga de cada una de ellas, de acuerdo con los sensores y actuadores[2] a utilizar.

Cuando una entrada o salida es analógica, significa que en su entrada o salida solo permite señales analógicas; o explicado de otra manera, señales continuas en el

[1] Tomado y traducido de la página oficial de Arduino https://www.arduino.cc/
[2] Actuador: Dispositivo que realiza la conversión de energía eléctrica a otro tipo de energía.

tiempo. Por otra parte, cuando una entrada o salida es digital, ésta solo permite señales digitales; las señales digitales son señales discontinuas en el tiempo.

Las tarjetas Arduino en su interior solo entienden señales digitales; es por eso por lo que, hacen uso de conversores de analógico a la digital cuando reciben a través de una entrada analógica una señal analógica. Este tipo de conversores hace que una señal analógica sea convertida a digital para poder ser procesada.

Cuando el Arduino ha procesado la señal de entrada, éste genera una señal de respuesta, la cual es digital. Sin embargo, no siempre los actuadores necesitaran de señales digitales para ser controlados, muchos necesitan de señales analógicas. Por tal razón, cuando una salida del Arduino necesita una señal analógica, previamente, éste hace uso de un conversor de digital a analógico, el cual convierte la señal digital de respuesta a una señal analógica.

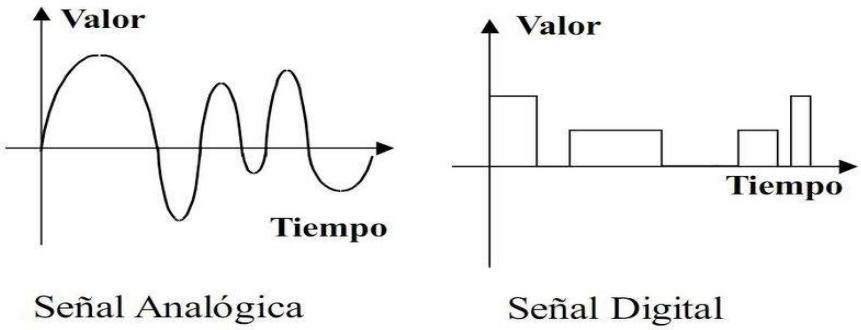


Figura 6. Diferencia entre una señal analógica y una digital.

En la siguiente imagen observamos una tarjeta Arduino YUN, la cual se utiliza para llevar a cabo este proyecto monográfico.

Figura 7. Tarjeta Arduino YUN.

Las características eléctricas principales de tarjeta Arduino Yun son las siguientes:

- Microcontrolador ATmega32u4
- Voltaje de operación 5VDC
- Voltaje de entradas y salidas 5VDC
- Pines de entrada/salida digital 20
- Canales PWM 7
- Canales de entrada analógicas 12
- Corriente DC por pin de entrada/salida 40mA
- Memoria Flash 32KB (4KB para el bootloader)
- SRAM 2.5KB
- EEPROM 1KB
- Velocidad de reloj 16MHz
- Comunicación WIFI, Ethernet y USB

5. ANÁLISIS Y PRESENTACIÓN DE RESULTADOS

5.1. Resultados de la recopilación de información

La información necesaria para el diseño del controlador difuso se obtuvo por medio de entrevistas con personas que se han dedicado al negocio de panaderías por muchos años; además de, la observación del proceso de horneado de varios tipos de masas; tanto con harina de fuerza como con harina floja o normal. *La harina de fuerza es una harina que tiene más gluten que la harina normal. A la harina normal se le denomina "harina floja" ya que, al no tener esa cantidad de gluten, da menos consistencia a las masas. La harina floja necesita levar (reposar su tamaño para que aumente al doble),*

para hacer unos donuts, pizzas, panes, etc. ya que la levadura necesita comerse ese gluten para soltar el gas y que la masa se hinche[3].

Se pudo observar que, el tiempo, al igual que la temperatura, es crucial para la cocción de las masas, pero la mayoría de las panaderías artesanales no poseen hornos con control de temperatura ni de tiempo; es por ello, que la temperatura de cocción y el tiempo depende de la experiencia de la persona encargada del horno. En otro orden, los chef y reposteros profesionales usan hornos industriales con control de temperatura y de tiempo. No obstante, el control de temperatura que se pretendía diseñar era difuso; por tal razón, se necesitaban temperaturas de referencia para ser manejados como términos lingüísticos. En la tabla 1 se muestran los términos lingüísticos y las temperaturas de referencia utilizadas.

No.	Término lingüístico	Temperatura de referencia
1	Muy Bajo	145 °C
2	Bajo	160 °C
3	Medio	180 °C
4	Alto	225 °C
5	Muy Alto	260 °C

Tabla 1. Términos lingüísticos y temperaturas de referencia

5.2. Diseño de controlador difuso

5.2.1. Consideraciones iniciales

El proceso de cocción de una masa no puede ser modelada, ya que el proceso dependerá de la materia prima utilizada, de la composición química de tales materiales, del proceso de reposo de la masa, del proceso de amasado, de la receta utilizada, etc. En consecuencia, el controlador de Mamdani es el adecuado para realizar el control de temperatura difuso para un horno panadero. Se descarta el

[3] Obtenido de: https://www.recetin.com/diferencias-harina-fuerza-y-harina-normal.html

controlador de Sugeno, porque, recordemos que, para utilizar este tipo de controlador es necesario conocer la curva de control del proceso a controlar; y en este caso, dicha curva es desconocida.

Con respecto a las variables de control, se observó que, en los trabajos de control difuso de temperatura consultados, se utilizaba el *error relativo* y la *variación del error* como variables de control de proceso; además, se descubrió que el *error relativo* era una de las características que posee el controlador de Mamdani como una de sus variables de control. Por tal motivo, se escogió el *error relativo* como una de las variables de control. Pese a que, en los trabajos consultados se utilizaba, además del *error relativo*, la variación de este como la otra variable de control; el autor de este trabajo consideró que era más conveniente utilizar la *variación de temperatura* como la segunda variable de control.

La ecuación del *error relativo* es la siguiente:

$$e = \left| \frac{V_R - V_M}{V_R} \right|$$

Donde:

e: Error relativo.

V_R: Valor real de la medida o valor deseado.

V_M: Valor obtenido de la medición o valor medido.

La *variación de temperatura* se calcula de la siguiente manera:

$$\Delta T = T_{Ac} - T_{An}$$

Donde:

ΔT: Variación de temperatura.

T_{Ac}: Temperatura actual.

T_{An}: Temperatura anterior.

En ambas operaciones, la del *error relativo* y la de la *variación de temperatura*, se ha tomado en cuenta el signo resultante. En la primera operación, se omite el valor absoluto, ya que el signo nos indica si la temperatura medida es mayor o menor que la temperatura deseada; en la segunda operación, el signo nos indica si la temperatura medida ha subido o ha bajado con respecto a la medición anterior. Esto se ha hecho con el fin de que, sea de ayuda en la definición de las reglas de control difusas.

Las dos variables de control anterior son variables de entrada al controlador difuso, las cuales se obtienen de la medición de temperatura, por medio de un sensor de temperatura LM35, y sus respectivas fórmulas. No obstante, hace falta definir la variable de control de salida.

Se necesita un dispositivo calentador para simular los dispositivos reales que dan calor en un horno real. El dispositivo escogido para tal propósito es una bombilla de 100W. Pero como se debe controlar la temperatura que ésta genera, se hace necesario controlar su potencia de salida. Esto se hace por medio de un circuito electrónico llamado dimmer. Por consiguiente, como tercera variable de control, la variable de salida, se definió la *potencia* de la bombilla eléctrica.

5.2.2. Términos lingüísticos de las variables de control

Con las variables de control previamente definidas, se procedió a la definición de los términos lingüísticos de las mismas.

En las siguientes tres tablas se presentan los términos lingüísticos de las variables de control de entrada y de salida.

ERROR RELATIVO	
Término lingüístico	Función de pertenencia
Negativo Grande	Hombro Izquierdo
Negativo Pequeño	Trapezoidal
Cero	Triangular
Positivo Pequeño	Trapezoidal
Positivo Grande	Hombro Derecho

Tabla 2. Términos lingüísticos de la variable de control error relativo.

VARIACIÓN DE TEMPERATURA	
Término lingüístico	Función de pertenencia
Negativo	Hombro Izquierdo
Cero	Triangular
Positivo	Hombro Derecho

Tabla 3. Términos lingüísticos de la variable de control variación de temperatura.

Potencia	
Término lingüístico	Función de pertenencia
Muy Bajo	Hombro Izquierdo
Bajo	Trapezoidal
Medio	Triangular
Alto	Trapezoidal
Muy Alto	Hombro Derecho

Tabla 4. Términos lingüísticos de la variable de control potencia.

En las siguientes figuras podemos ver las gráficas de los términos lingüísticos de las variables de control.

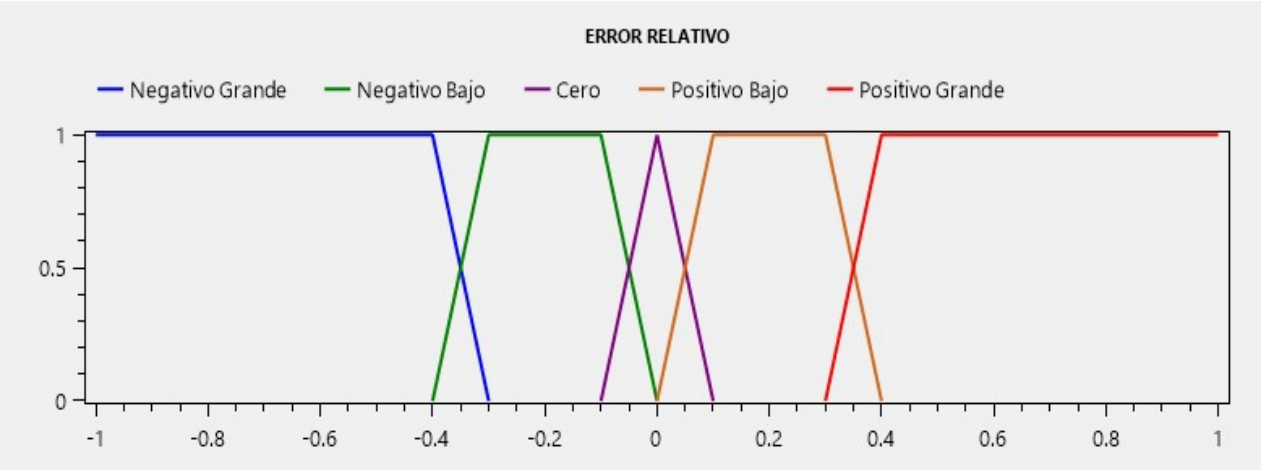


Figura 8. Términos lingüísticos del error relativo.

En la figura 8, podemos observar los límites de las funciones de pertenencia del error relativo.

La función del error Negativo Grande es una función Hombro Izquierdo, cuyo límite interno es -0.4, y su límite externo es -0.3.

Para el error Negativo Bajo se utiliza la función Trapezoidal, su límite inferior es -0.4, su límite interno izquierdo es -0.3, su límite interno derecho es -0.1, y su límite superior es 0.

El error Cero tiene como función de pertenencia a la función Triangular, donde su límite inferior es -0.1 y su límite superior es 0.1, cuyo vértice está en 0.

El error Positivo Bajo hace uso de la función Trapezoidal, su límite inferior es 0, su límite interno izquierdo es 0.1, su límite interno derecho es 0.3, y su límite superior es 0.4.

La función de pertenencia del error Positivo Grande es la función Hombro Derecho, teniendo como límite externo 0.3 y como límite interno 0.4.

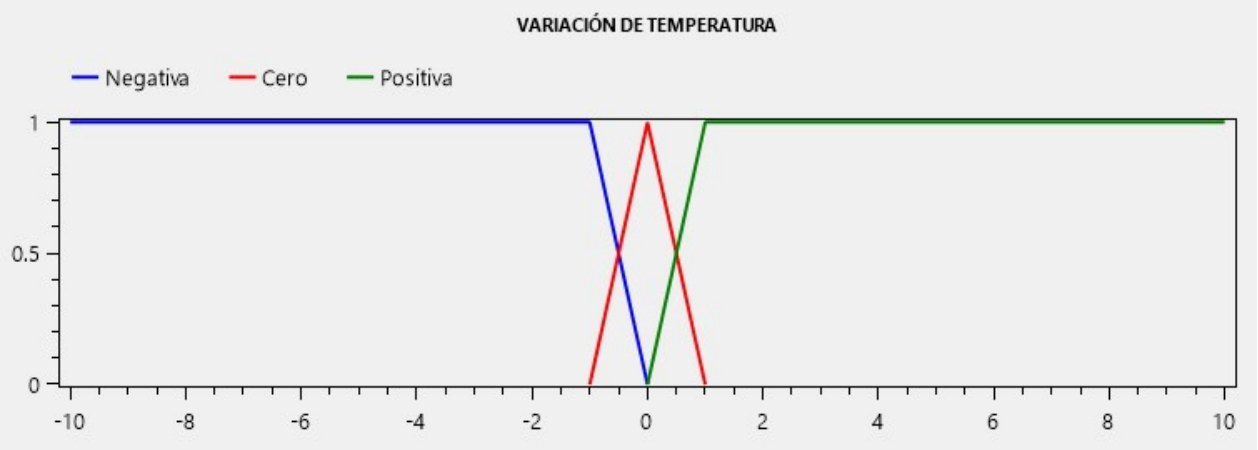


Figura 9. Términos lingüísticos de la variación de temperatura.

La figura 9 nos muestra lo siguiente:

La variación de temperatura Negativa hace uso de la función Hombro Izquierdo como función de pertenencia. Su límite interno está en -1 y su límite externo está en 0.

La función Triangular es la función de pertenencia de la variación de temperatura Cero. Su límite inferior está en -1 y su límite superior está en 1, además observamos que su vértice está en 0.

La variación de temperatura Positiva tiene como función de pertenencia a la función Hombro Derecho. El límite externo está en 0 y el límite interno está en 1.

Figura 10. Términos lingüísticos de la potencia de salida.

La figura 10 nos muestra:

La potencia de salida Muy Bajo tiene como función de pertenencia a la función Hombro Izquierdo, con límite interno en 0 y límite externo en 10.

La potencia de salida Bajo utiliza como función de pertenencia a la función Trapezoidal. Su límite inferior está en 0, su límite interno izquierdo está en 20, el límite interno derecho está en 40, y su límite superior está en 50.

La función Triangular es la función de pertenencia de la potencia de salida Medio. Tiene como límite inferior 40 y su límite superior es 60, su vértice está en 50.

La potencia de salida Alto utiliza como función de pertenencia a la función Trapezoidal. Su límite inferior está en 50, su límite interior izquierdo está 60, el límite interior derecho está en 80, y el límite superior está en 100.

La potencia de salida Muy Alto hace uso de la función Hombro Derecho, con límite externo en 80 y límite interno en 100.

5.2.3. Funciones de pertenencia

Ya se han mencionado las funciones de pertenencia elegidas de cada término lingüístico, incluso ya conocemos sus límites. Sin embargo, se hace sumamente necesario conocer sus ecuaciones para poder saber cómo se obtiene el grado de pertenencia en cada una de estas funciones matemáticas.

Función Hombro Izquierdo

Esta función tiene dos límites, uno interior ***c*** y otro exterior ***d*** a como se muestra en la figura 11. Su valor es 0 cuando el valor de ***x*** es mayor o igual a ***d***. Su valor estará entre 0 y 1 cuando el valor de ***x*** esté entre ***c*** y ***d***. Su valor es 1 si el valor de ***x*** es menor o igual a ***c***.

$$\mu(x) = \begin{cases} 0, & si\ x \geq d \\ \dfrac{d-x}{d-c}, & si\ c < x < d \\ 1, & si\ x \leq c \end{cases}$$

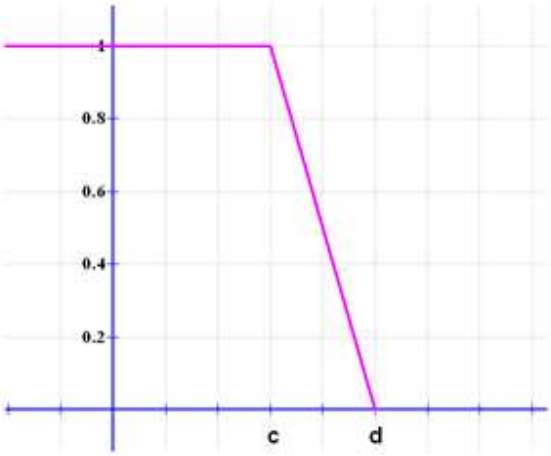


Figura 11. Función Hombro Izquierdo.

Función Hombro Derecho

Esta función posee un límite externo ***a*** y un límite interno ***b***. Su valor es 0 si el valor de ***x*** es menor o igual a ***a***. Su valor estará entre 0 y 1 si el valor de ***x*** está entre ***a*** y ***b***. Su valor es 1 si el valor de ***x*** es mayor o igual a ***b***.

$$\mu(x) = \begin{cases} 0, & si\ x \leq a \\ \dfrac{x-a}{b-a}, & si\ a < x < b \\ 1, & si\ x \geq b \end{cases}$$

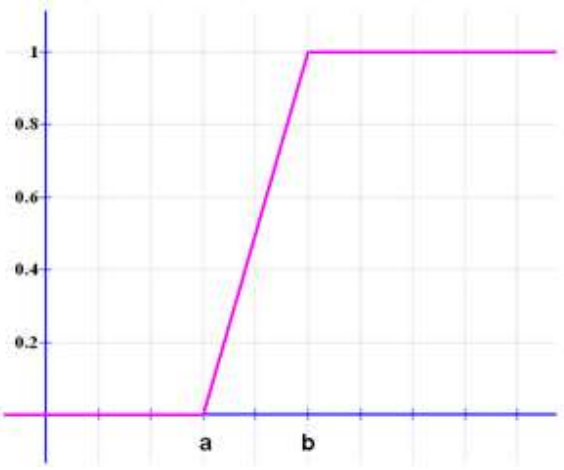


Figura 12. Función Hombro Derecho.

Función Triangular

La función triangular tiene un límite inferior ***a***, un vértice ***m***, y un límite superior ***b***. Su valor es 0 si ***x*** es menor o igual a ***a*** o si ***x*** es mayor o igual a ***b***. Su valor está entre 0 y 1 si el valor de ***x*** está entre ***a*** y ***b***. Cabe señalar que, su valor será 1, únicamente cuando ***x*** es igual a ***m***.

$$\mu(x) = \begin{cases} 0, & si\ x \leq a \\ \dfrac{x-a}{m-a}, & si\ a < x < m \\ \dfrac{b-x}{b-m}, & si\ m < x < b \\ 0, & si\ x \geq b \end{cases}$$

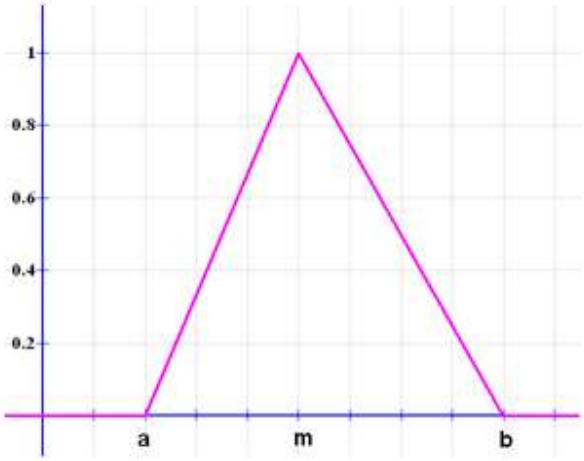


Figura 13. Función Triangular.

Función Trapezoidal

La función trapezoidal tiene dos límites externos, uno inferior ***a*** y uno superior ***d***. Además, tiene dos límites internos, uno izquierdo ***b*** y otro derecho ***c***. El valor de esta función es 0 si ***x*** es menor o igual a ***a*** o si ***x*** es mayor o igual a ***d***. Cuando ***x*** está entre ***a*** y ***b*** o entre ***c*** y ***d***, el valor de la función estará entre 0 y 1. Cuando ***x*** es mayor o igual a ***b*** y menor o igual a ***c***, el valor de la función es 1. Todos los valores de ***x*** que resultan en valor 1 son parte de lo que se llama el núcleo de la función.

$$\mu(x) = \begin{cases} 0, & si\ x \leq a\ o\ x \geq d \\ \dfrac{x-a}{b-a}, & si\ a < x < b \\ 1, & si\ b \leq x \leq c \\ \dfrac{d-x}{d-c}, & si\ c < x < d \end{cases}$$

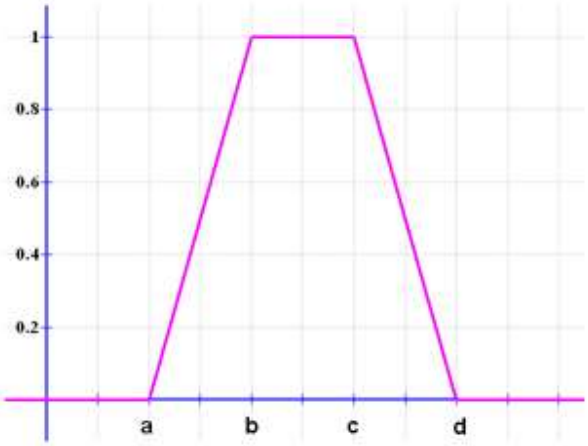


Figura 14. Función Trapezoidal.

5.2.4. Definición de reglas de control difuso

Tenemos definidas dos variables de control de entrada, por tanto, tenemos dos antecedentes. Por otro lado, tenemos una sola variable de control de salida; por tal razón, tenemos un solo consecuente.

Los antecedentes son el *error relativo* y la *variación de temperatura*. El *error relativo* tiene cinco términos lingüísticos y la *variación de temperatura* tiene tres; en

consecuencia, tendremos quince reglas de control difusas. Tales reglas se introducen a continuación.

1. *Si el error relativo es Negativo Grande y la variación de temperatura es Negativo, entonces la potencia de salida será Muy Bajo.*
2. *Si el error relativo es Negativo Grande y la variación de temperatura es Cero, entonces la potencia de salida será Muy Bajo.*
3. *Si el error relativo es Negativo Grande y la variación de temperatura es Positivo, entonces la potencia de salida será Muy Bajo.*
4. *Si el error relativo es Negativo Bajo y la variación de temperatura es Negativo, entonces la potencia de salida será Muy Bajo.*
5. *Si el error relativo es Negativo Bajo y la variación de temperatura es Cero, entonces la potencia de salida será Muy Bajo.*
6. *Si el error relativo es Negativo Bajo y la variación de temperatura es Positivo, entonces la potencia de salida será Muy Bajo.*
7. *Si el error relativo es Cero y la variación de temperatura es Negativo, entonces la potencia de salida será Muy Bajo.*
8. *Si el error relativo es Cero y la variación de temperatura es Cero, entonces la potencia de salida será Muy Bajo.*
9. *Si el error relativo es Cero y la variación de temperatura es Positivo, entonces la potencia de salida será Muy Bajo.*
10. *Si el error relativo es Positivo Bajo y la variación de temperatura es Negativo, entonces la potencia de salida será Alto.*
11. *Si el error relativo es Positivo Bajo y la variación de temperatura es Cero, entonces la potencia de salida será Alto.*
12. *Si el error relativo es Positivo Bajo y la variación de temperatura es Positivo, entonces la potencia de salida será Medio.*
13. *Si el error relativo es Positivo Grande y la variación de temperatura es Negativo, entonces la potencia de salida será Muy Alto.*
14. *Si el error relativo es Positivo Grande y la variación de temperatura es Cero, entonces la potencia de salida será Muy Alto.*

15. *Si el error relativo es Positivo Grande y la variación de temperatura es Positivo, entonces la potencia de salida será Alto.*

En la figura 15, se puede observar las quince reglas difusas ordenadas en forma de una matriz. Esta matriz es conocida como matriz difusa, y se utiliza para expresar de manera compacta todas las reglas de control de un controlador difuso.

ERROR \ Δ TEMP	NEGATIVO	CERO	POSITIVO
NEGATIVO GRANDE	*MB*	*MB*	*MB*
NEGATIVO BAJO	*MB*	*MB*	*MB*
CERO	*MB*	*MB*	*MB*
POSITIVO BAJO	*A*	*A*	*MD*
POSITIVO GRANDE	*MA*	*MA*	*A*

MB: Muy Bajo BJ: Bajo MD: Medio A: Alto MA: Muy Alto

Figura 15. Matriz difusa.

5.2.5. Fusificación

Debido a que se ha decidido diseñar un controlador de tipo Mamdani, el proceso de fusificación se hace con la operación de intersección (AND). Hay que recordar que, en las normas T, vimos algunas funciones que cumplían con tales normas para realizar la operación de intersección difusa, y que entre ellas estaba la función Mínimo. Se ha

escogido la función mínima para realizar el proceso de fusificación del controlador difuso, ya que la misma es una función fácil de aplicar y de codificar. Recordemos que, el proceso de fusificación se lleva a cabo en cada una de las reglas difusas, en donde, en cada regla se evalúa el valor medido en cada función de pertenencia de los antecedentes, obteniendo así un valor de pertenencia mínimo en cada regla.

En la figura 16, se puede apreciar el proceso de fusificación para los antecedentes de la regla 7. Nótese que, el valor de pertenencia mínimo corresponde al error relativo Cero, con un valor de 0.72, aproximadamente; mientras que, el valor de pertenencia máximo corresponde a la variación de temperatura Negativa, con un valor de 1.

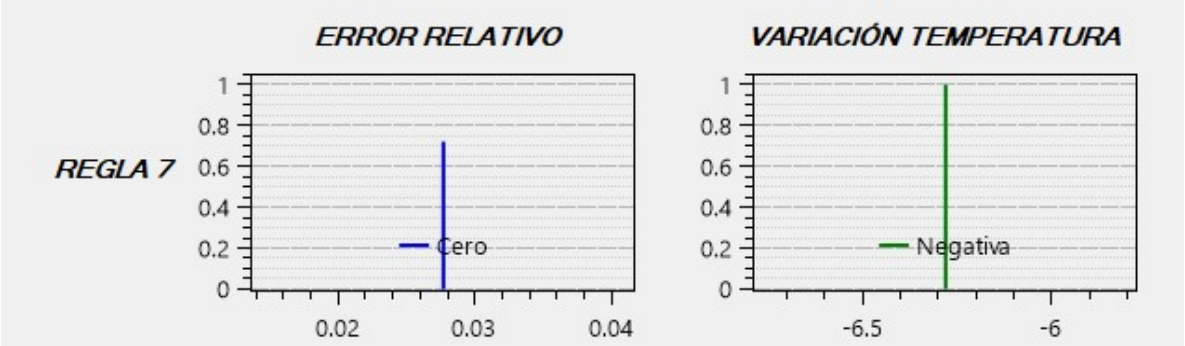


Figura 16. Proceso de fusificación.

5.2.6. Corte

En el proceso de corte, también se hace uso de la función mínimo, debido a que se corta la función de pertenencia del consecuente de acuerdo con valor del grado de pertenencia más bajo en cada regla, el cual es obtenido en el proceso de fusificación. Hay que hacer notar que, el proceso de corte obtiene los valores de la función de pertenencia del consecuente que son menores o iguales al valor de pertenencia mínimo obtenido en el proceso de fusificación.

En la figura 17, se observa cómo se hace el proceso de corte en el consecuente de la regla 7. La función de pertenencia del consecuente, potencia Muy Bajo, es cortada en el valor de la función de pertenencia del error relativo cero, el cual es el menor de ambos antecedentes. Originalmente, la función de pertenencia de la potencia de salida

Muy Bajo tiene como límite interno 0 (ver figura 10); pero obsérvese que, ahora con el corte, ese límite se ha corrido hacia la derecha, al valor 3 aproximadamente (ver figura 17).

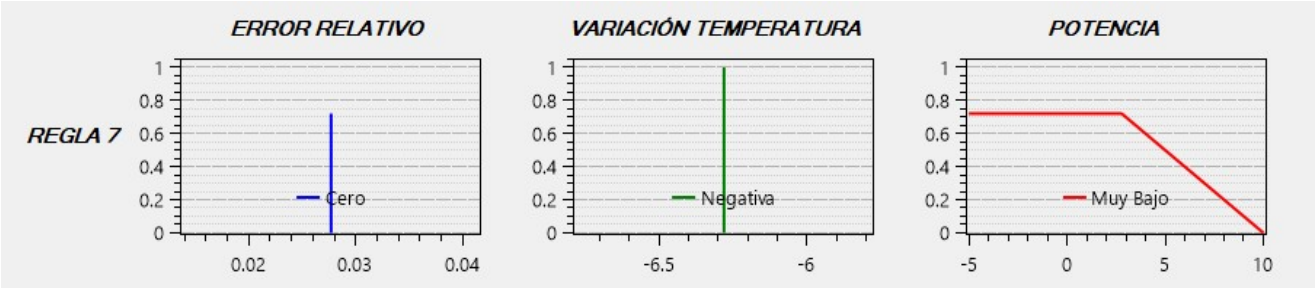


Figura 17. Proceso de Corte.

5.2.7. Unión

La unión de todos los cortes de los consecuentes se hace con la operación de conjunción difusa (OR). En la sección de las normas S o co-normas T, vimos que existían varias funciones que cumplían dichas normas, las cuales son necesarias para realizar la operación de conjunción difusa. Entre esas funciones se ha escogido la función Máximo.

En la figura 18, se observa el proceso de unión de todos los cortes de los consecuentes. Podemos apreciar que, en la gráfica hay dos cortes notables, el corte de la potencia Bajo, y el corte de la potencia Medio. Los demás cortes que no se pueden apreciar, debido a que, todos sus valores son cero.

Figura 18. Proceso de Unión de cortes de los consecuentes.

5.2.8. Defusificación

Para el proceso de defusificación se ha elegido al método del Centroide. En este proceso, el resultado obtenido es un número, el cual es el valor de la potencia de salida que se le debe aplicar al dispositivo de calentamiento, en este caso el bombillo eléctrico.

5.3. Desarrollo del software

Para el desarrollo del software de este proyecto, se utilizó la metodología RUP, la cual posee cuatro fases de trabajo en su ciclo de vida de desarrollo, a como se mencionó con anterioridad; a saber, Inicio, Elaboración, Construcción y Transición. En cada una de ellas se llevan a cabo tareas específicas para alcanzar un producto terminado, el software; además de, cumplir con los objetivos propuestos.

5.3.1. Fase de Inicio

Requerimientos No Funcionales

- El paradigma de programación a utilizar debe ser el paradigma orientado a objetos.
- El lenguaje de programación a utilizar debe ser C#.
- La librería gráfica por utilizar en el código debe ser OxyPlot.
- Las imágenes que se utilicen en el sistema deben diseñarse o editarse con el editor de imágenes GIMP.
- Las reglas de control difuso deben estar contenidas en el código; es decir, el usuario no tendrá que incluirlas en el tiempo de simulación.

* Las únicas funciones matemáticas que se utilicen para las funciones de pertenencia difusa deben ser: Función Hombro Derecho, Función Hombro Izquierdo, Función Triángulo y Función Trapecio.
* El sistema debe ser parecido al sistema de un horno eléctrico digital real. Por tanto, debe tener las funciones básicas de Inicio, paro / reset, ingreso de tiempo de cocción e ingreso de temperatura de cocción.
* La temperatura debe ser procesada y presentada únicamente en grados Celsius.
* Se debe utilizar una placa Arduino para que sea ésta la interfaz de hardware entre el software diseñado y el sensor de temperatura y los actuadores.

Requerimientos Funcionales

* El sistema no necesita de autenticación de usuario.
* El sistema debe ser monousuario.
* El sistema no necesita guardar ningún dato al finalizar la simulación.
* La comunicación entre el software y la tarjeta Arduino debe realizarse vía alámbrica.
* Al inicio del programa se debe visualizar el modelo completo del sistema; es decir, se debe visualizar la arquitectura del sistema para que el usuario comprenda de mejor manera su funcionamiento.
* El usuario debe tener disponible las opciones de ver los gráficos de las variables lingüísticas, las reglas difusas, los gráficos de fusificación, corte de consecuentes y gráfico de unión de cortes.
* En una ventana independiente de los demás gráficos se debe mostrar un gráfico temporal de las mediciones de temperatura que realiza el sensor.
* Los controles que simulan los controles reales del horno deberán presentarse en una ventana independiente de los gráficos del proceso de control.
* El sistema debe incluir un botón de encendido y apagado, el cual habilitará o deshabilitará los demás controles del horno.

* También debe contener un botón de Paro/Reset, el cual detendrá el proceso de simulación, en caso de que, el mismo haya sido iniciado, y/o reiniciará todas las variables a sus valores por defecto, cuando el proceso de simulación haya sido detenido, o cuando éste no haya iniciado, pero se hayan fijados valores de simulación previamente en el menú.
* Adicionalmente, debe contener un botón de menú. Este menú debe mostrar tres opciones, dos de ajustes de parámetros, los cuales serán, tiempo y temperatura, y otra opción de inicio de proceso.
* El tiempo de simulación puede ser de las dos maneras siguientes: libre, es decir, sin un tiempo establecido por el usuario, o a través de un tiempo previamente establecido.
* El menú de temperatura tendrá dos opciones para fijar este parámetro. La primera opción será Difusa, para cuando el usuario no desee usar una temperatura exacta en el proceso. Esa opción tendrá cinco opciones difusas adicionales, las cuales serán, muy bajo, bajo, medio, alto y muy alto. La segunda opción será cuando el usuario desee fijar una temperatura exacta.
* La opción de Iniciar Proceso se habilitará únicamente si previamente se ha hecho el respectivo ajuste de temperatura.
* El sistema debe incluir un botón de Intro para confirmar cada ajuste en el menú o confirmar el inicio del proceso.
* Asimismo, el sistema debe tener cuatro botones para desplazarse dentro de las opciones del menú. Un botón de flecha de desplazamiento hacia arriba, un botón de flecha de desplazamiento hacia abajo. Estos dos botones de desplazamientos serán para desplazarse hacia una u otra de las tres opciones del menú (Temperatura, Tiempo e Inicio); además de, aumentar o disminuir los dígitos correspondientes dentro de cada opción del menú. Adicionalmente, para completar los cuatro botones de desplazamientos, debe de haber un botón de flecha de desplazamiento hacia la derecha y otro botón de flecha de desplazamiento hacia la izquierda para desplazarse entre los dígitos correspondientes dentro de cada opción del menú.

* Todas las ventanas que se presenten en el momento que el sistema esté en ejecución, estarán contenidas dentro de una ventana MDI padre.

5.3.2. Fase de Elaboración

Arquitectura del sistema

En la figura 19, se aprecia cómo es la interacción funcional de los componentes del sistema. Vemos que estos componentes son: Controlador difuso, tarjeta electrónica (PCB), sensor y actuador.

Las flechas indican la comunicación e intercambio de datos. Las flechas unidireccionales indican que la información o ejecución de una tarea va en un solo sentido. Mientras que la flecha bidireccional indica que la comunicación es en ambos sentidos; es decir ambos componentes intercambian datos.

El funcionamiento del sistema es de la siguiente manera. En primera instancia, la PCB, que en este caso es una tarjeta Arduino, lee el dato del sensor de temperatura (LM35). Este dato, es acondicionado por la tarjeta Arduino y lo envía al controlador difuso, alojado en una computadora personal (PC). El controlador difuso obtiene el dato de temperatura y lo procesa de acuerdo con sus reglas de control difuso, en su máquina de inferencia; el dato resultante, es enviado a la tarjeta Arduino, ésta lo procesa para posteriormente ejecutar el control del actuador, el bombillo eléctrico.

Analizando el funcionamiento del sistema, podemos visualizar que la arquitectura de este se asemeja a la arquitectura Cliente-Servidor, donde el cliente es la tarjeta Arduino y, el servidor es el controlador difuso. El Arduino envía una petición de procesamiento del dato obtenido del sensor de temperatura LM35; el controlador difuso recoge el dato de temperatura y lo procesa, luego el resultado lo envía al

Arduino, donde éste, lo recoge, lo procesa y lo adecúa para controlar el nivel de potencia del bombillo eléctrico.

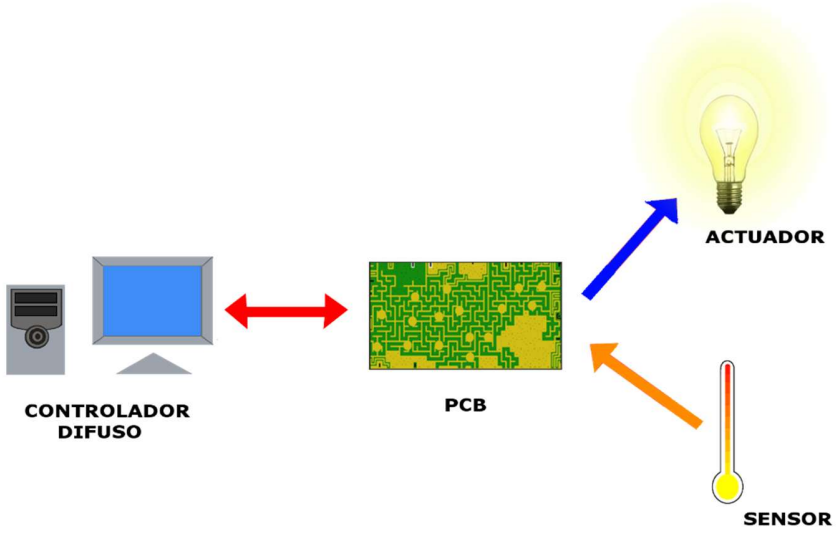


Figura 19. Arquitectura del sistema.

La figura 20, nos da otra perspectiva de la arquitectura del sistema en forma de capas, un poco más detallada. Podemos observar que, hay tres capas: la capa de Entradas y Salidas, la capa de Acondicionamiento de Señal y la capa del Controlador difuso.

La capa de Entradas y Salidas está compuesta por el Sensor de Temperatura y el Dispositivo Calentador. El Sensor de Temperatura envía el dato de la temperatura a la capa de Acondicionamiento de Señal, la cual es recibida en el bloque de Entrada Analógica; mientras que, el Dispositivo Calentador recibe el dato de accionamiento desde la capa de Acondicionamiento de Señal, a través del bloque de Salida Digital.

De la capa de Acondicionamiento de Señal, ya conocemos el funcionamiento de los bloques de Entrada Analógica y Salida Digital. Estos dos bloques envían y reciben datos del bloque del Microcontrolador. Este bloque es el encargado de procesar tanto las entradas como las salidas de la capa. El bloque del Puerto Serial recibe y envía datos desde y hacia el bloque del Microcontrolador; igualmente, desde y hacia la capa del Controlador Difuso.

En la capa del Controlador Difuso tenemos el bloque Puerto Serie USB, este bloque es el que recibe y envía los datos desde y hacia la capa de Acondicionamiento de señal; adicionalmente, recibe y envía los datos desde y hacia el bloque del Motor de Inferencia Difusa. El bloque del Motor de Inferencia Difusa procesa los datos que recibe desde el bloque del Puerto USB y el bloque de Interfaz Gráfica de Usuario – GUI; los resultados que se obtienen los envía hacia la capa de Acondicionamiento de señal por medio del bloque del Puerto USB, y hacia el bloque GUI. El bloque de Interfaz Gráfica de Usuario, o GUI, es la interfaz entre el sistema y el usuario de este. Esa interfaz, le permite al usuario introducir y recibir información del sistema.

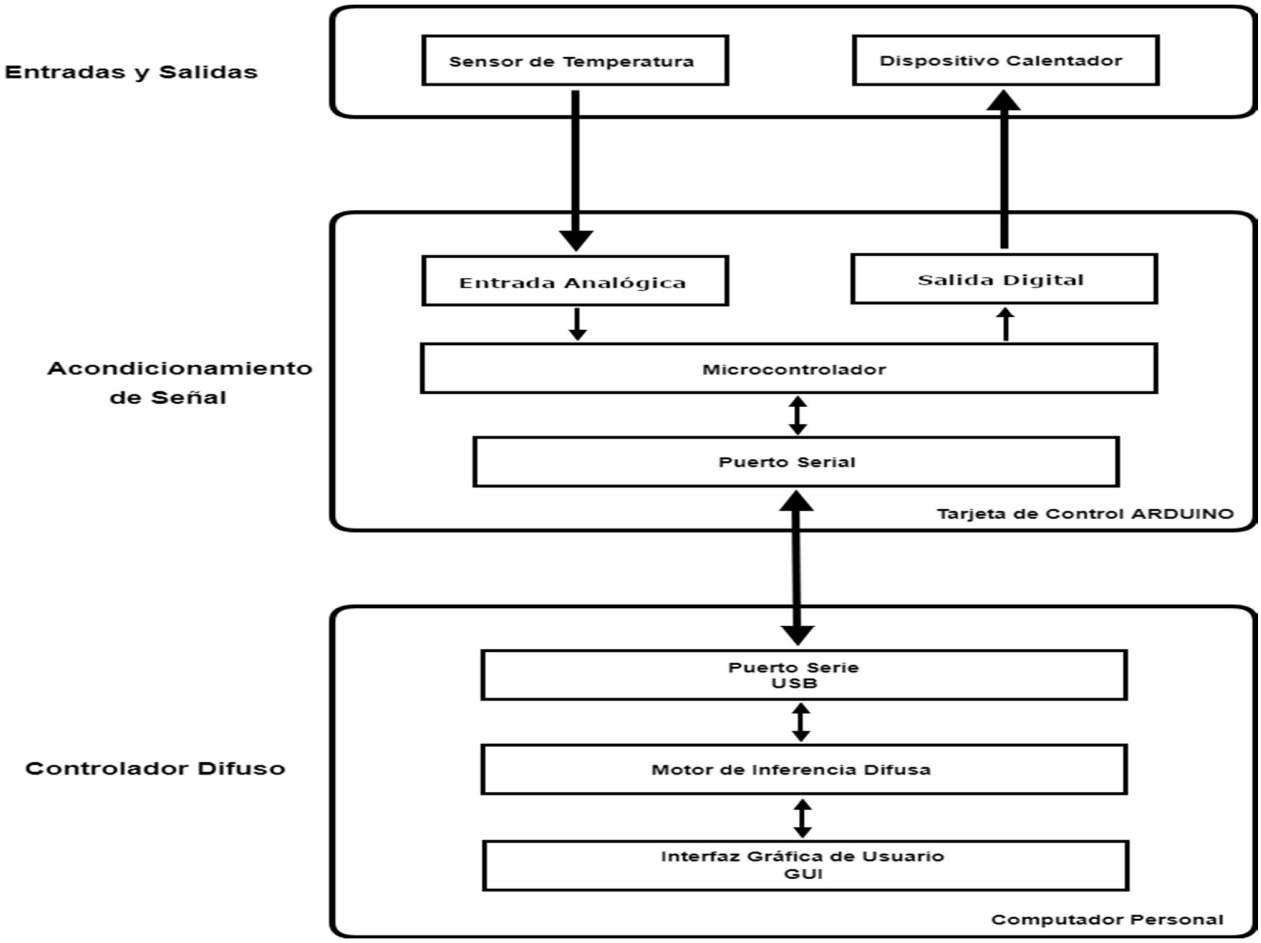


Figura 20. Arquitectura en capas del sistema.

Diagramas UML

Casos de uso

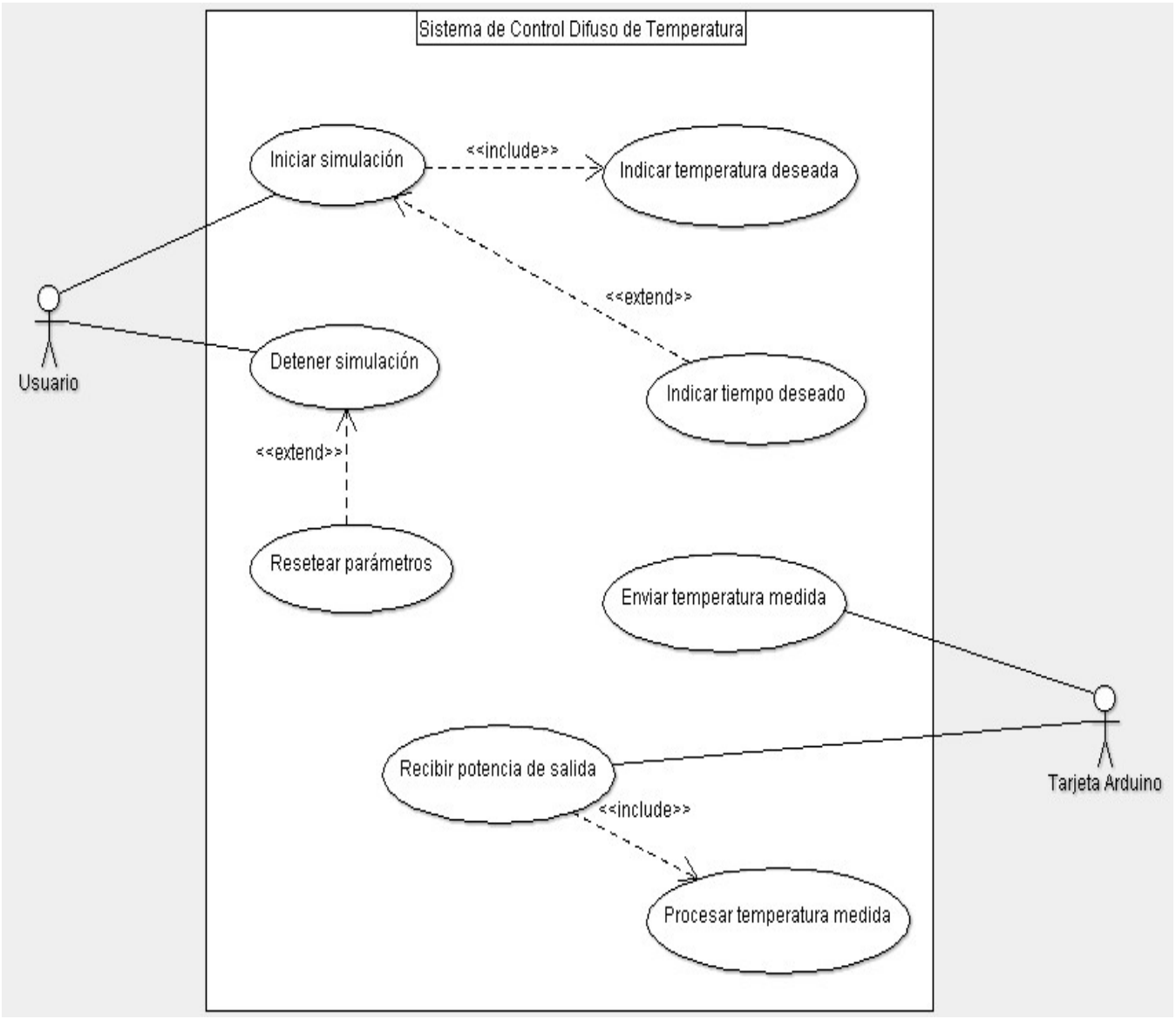


Figura 21. Diagrama de casos de uso del sistema de control difuso de temperatura.

Diagrama de clases

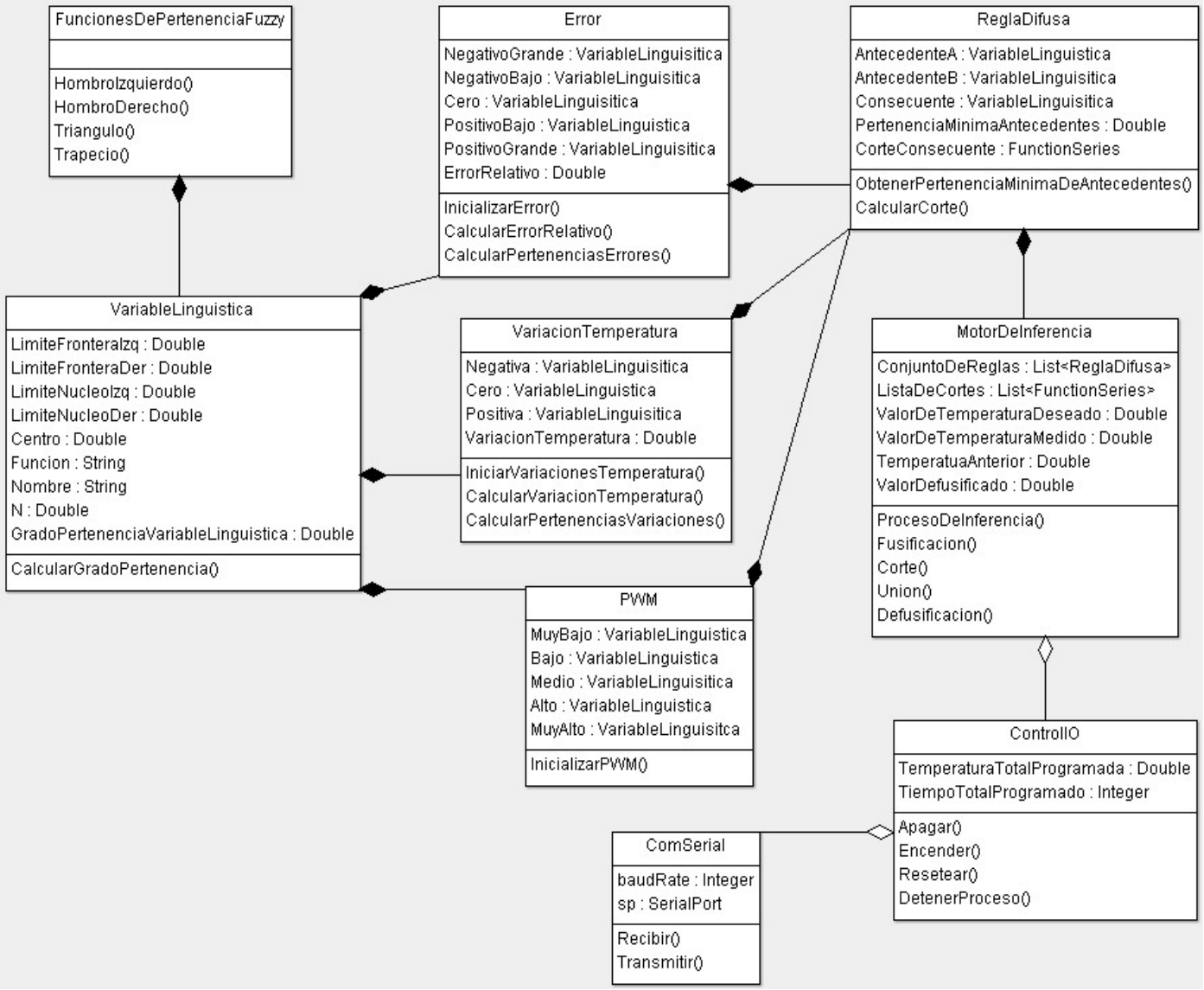


Figura 22. Diagrama de clases.

Diagrama de componentes

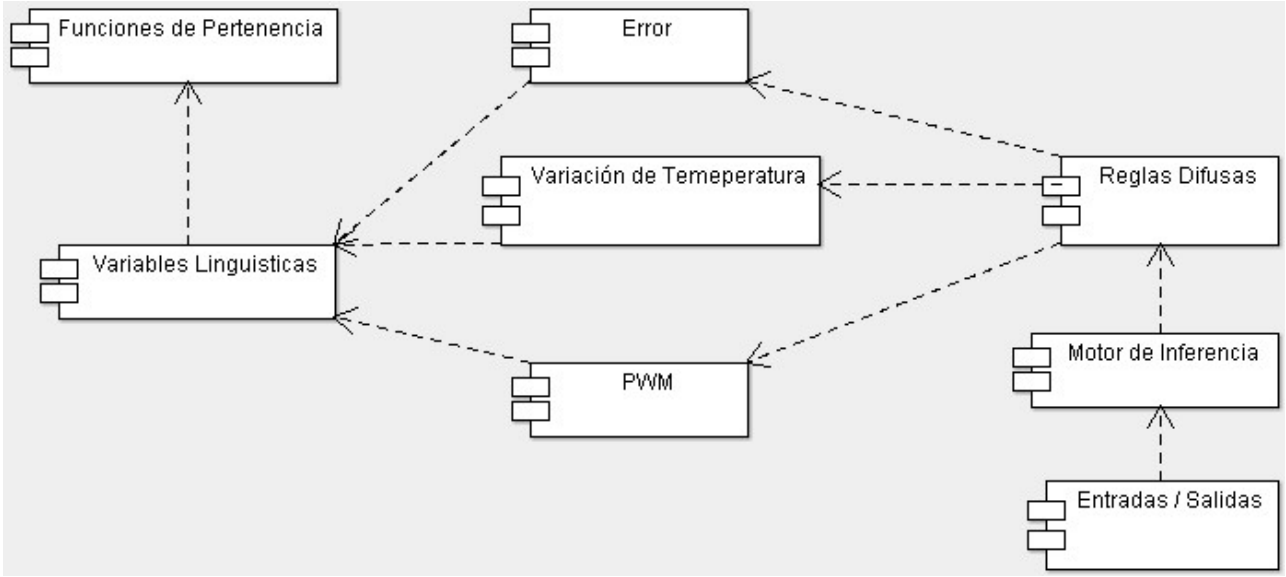


Figura 23. Diagrama de componentes.

Diagrama de despliegue

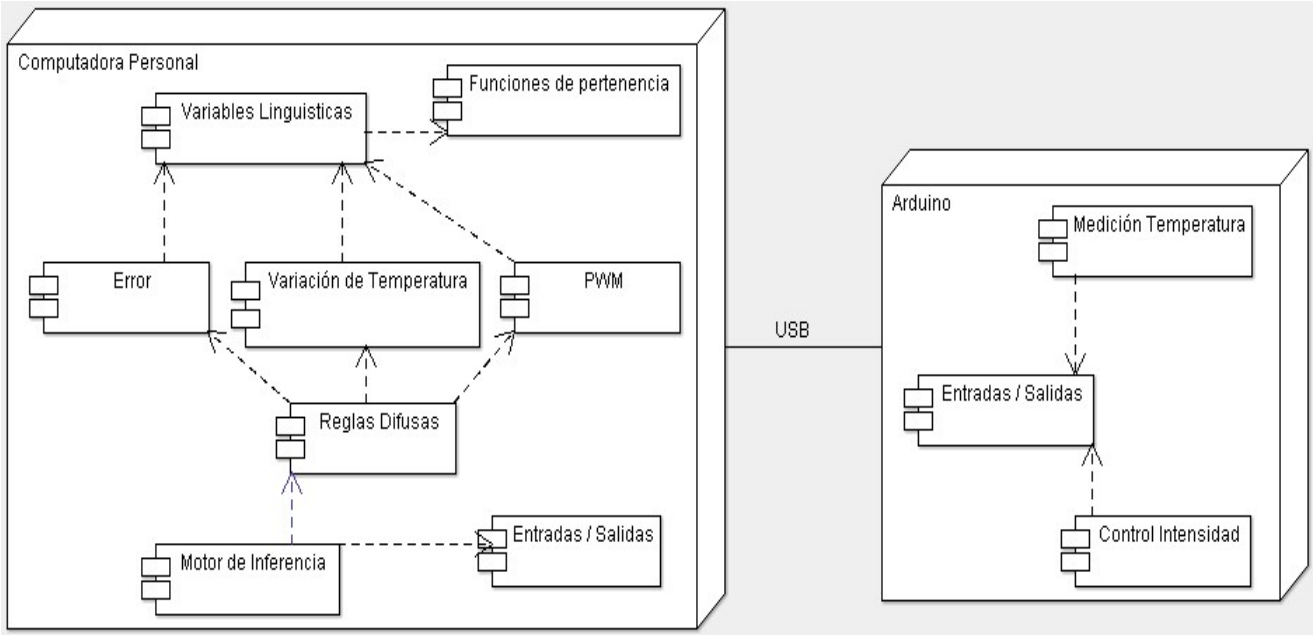


Figura 24. Diagrama de despliegue.

Diagrama de estados

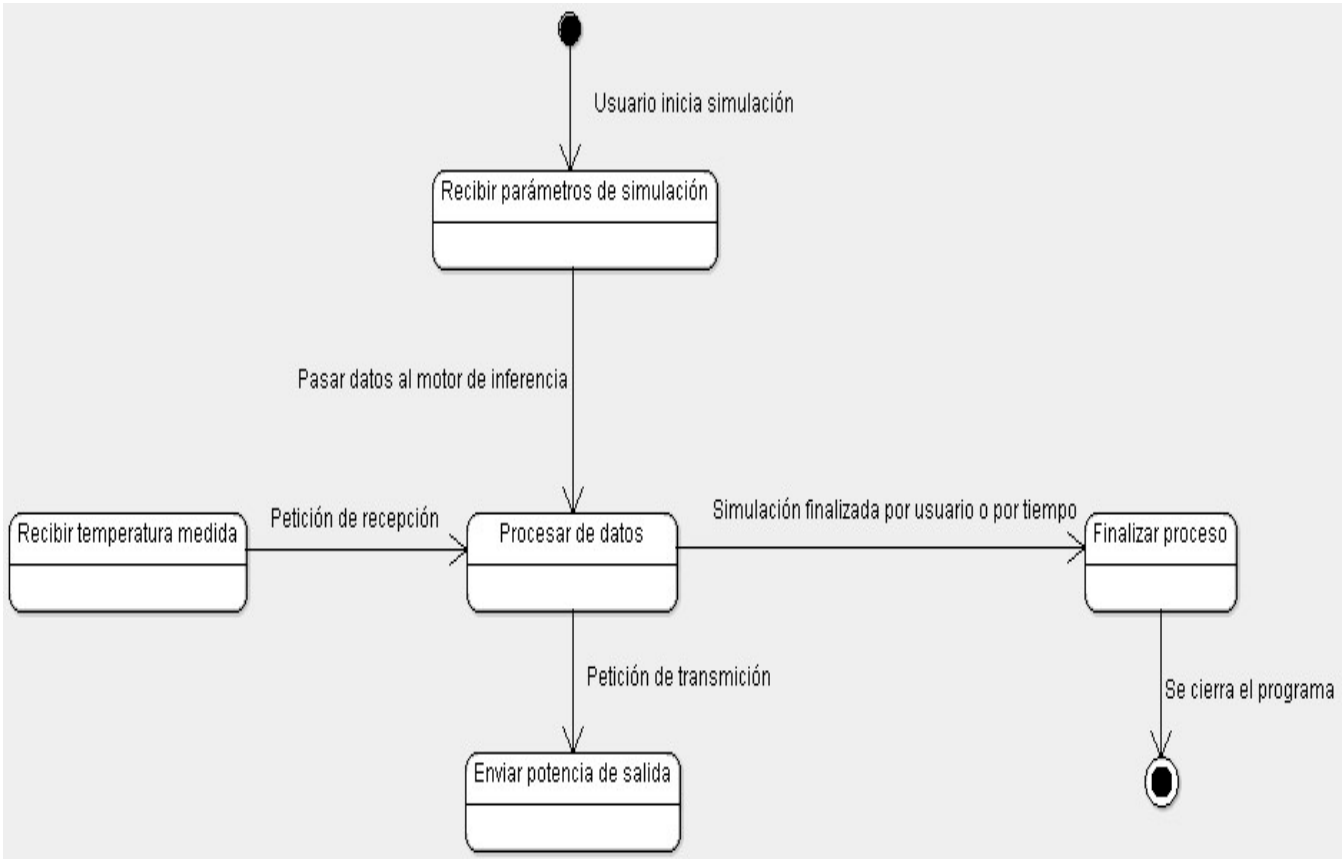


Figura 25. Diagrama de estados.

Diagrama de actividades

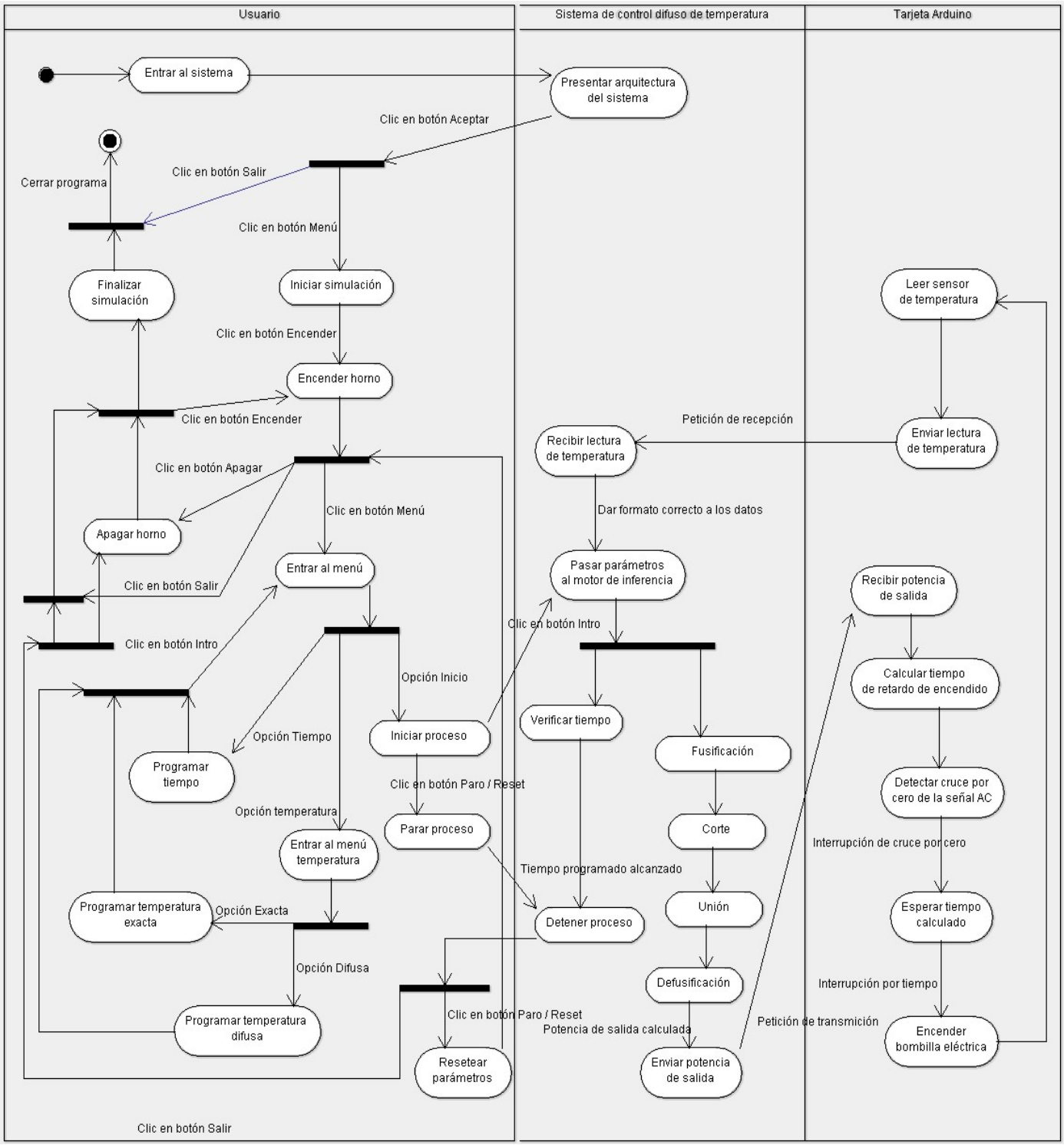


Figura 26. Diagrama de actividades.

Diagrama de secuencia

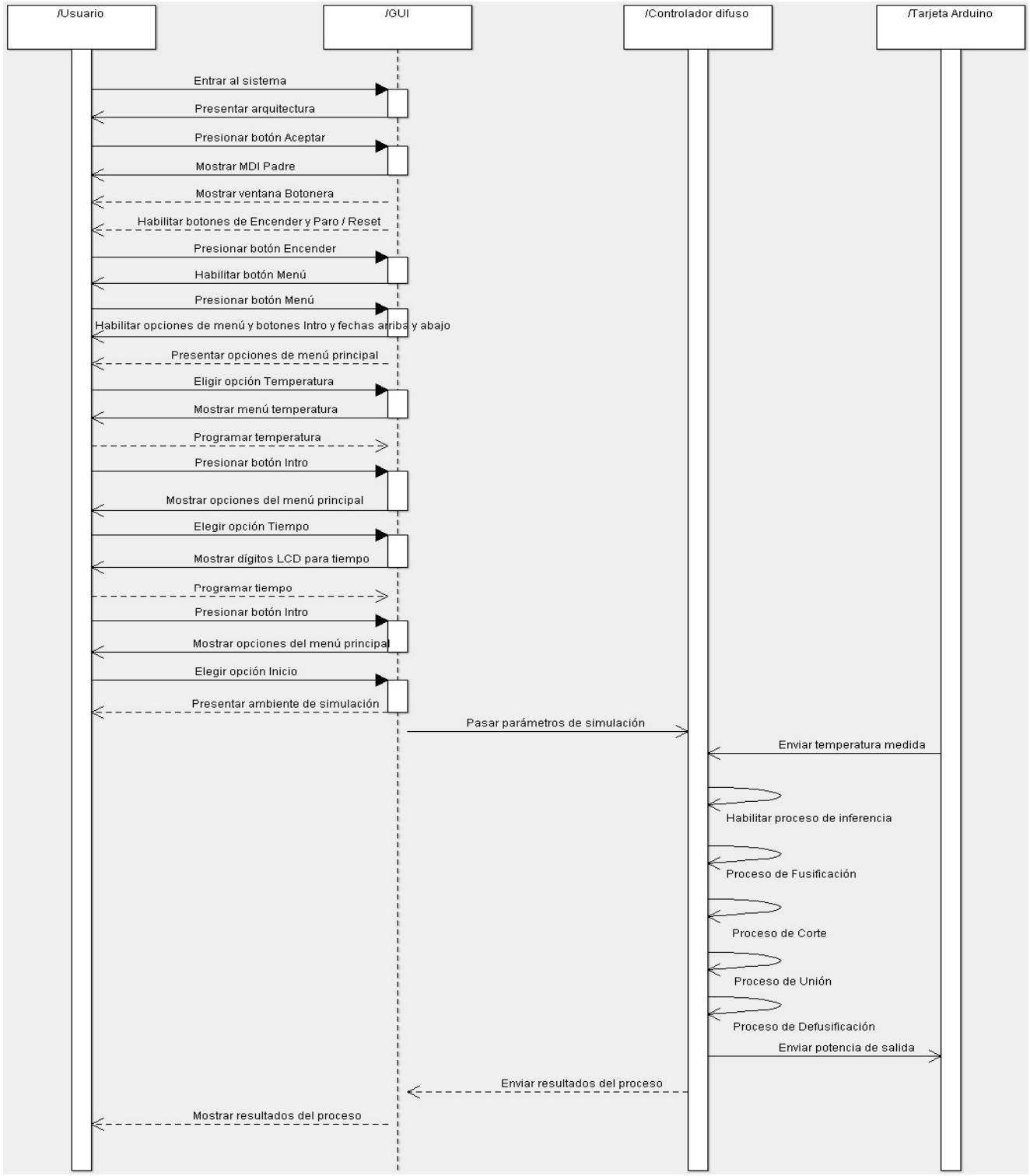


Figura 27. Diagrama de secuencia.

Diagrama de colaboración

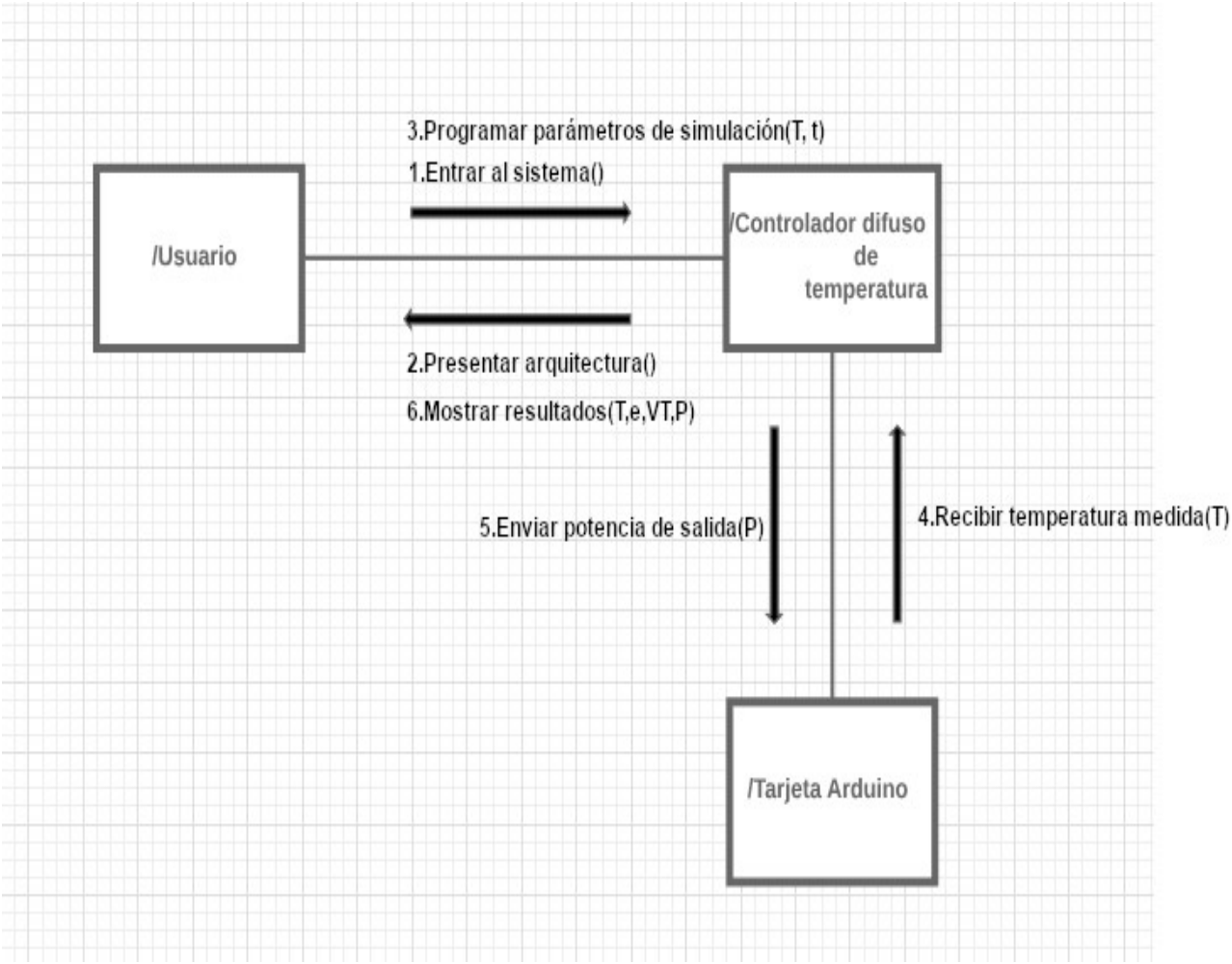


Figura 28. Diagrama de colaboración.

5.3.3. Fase de Construcción

En esta fase se cumplen dos de los objetivos planteados, el del diseño de la interfaz gráfica de usuario y el de la codificación de los módulos funcionales del controlador difuso.

Interfaz Gráfica de Usuario

Al ingresar al sistema, éste muestra primero una ventana donde presenta la arquitectura del sistema de dos formas; una a través de componentes de hardware, y la otra a través de capas, igual a las presentadas en este documento en la fase anterior.

La figura 29 muestra la ventana inicial del sistema.

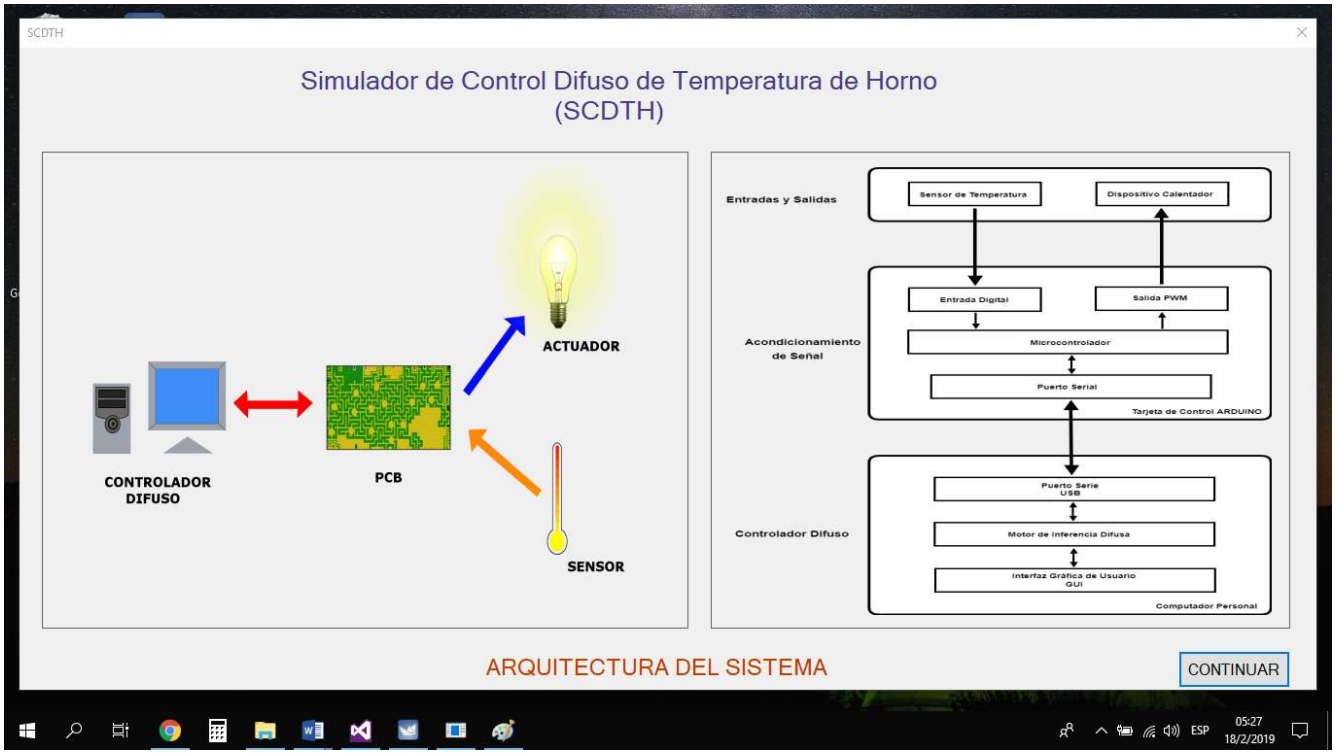


Figura 29. Ventana de inicial del Simulador de Control Difuso de Temperatura de Horno (SCDTH).

Para poder acceder al ambiente de simulación, el usuario debe de dar clic en el botón CONTINUAR, ubicado en la esquina inferior derecha.

El ambiente de simulación se muestra en la figura 30.

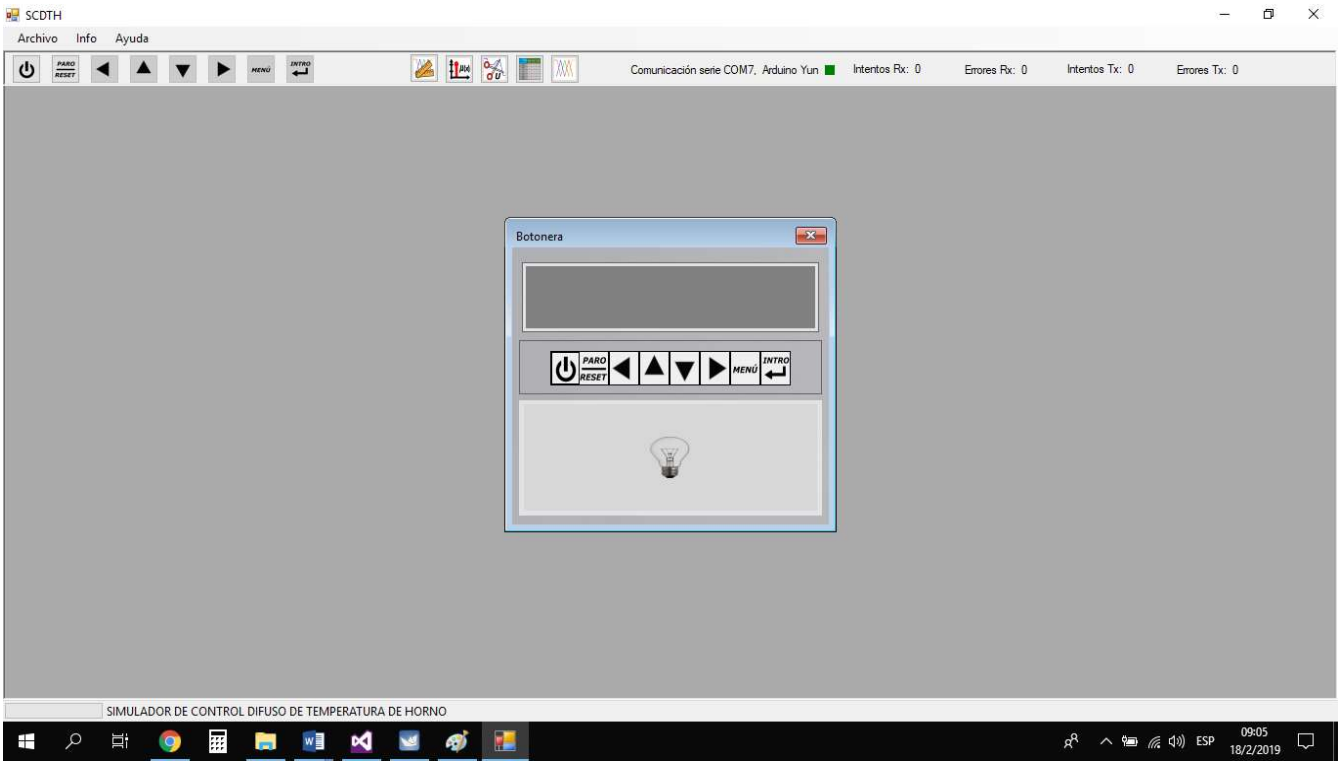


Figura 30. Ambiente de simulación.

El ambiente de simulación está compuesto de una ventana MDI padre, donde inicialmente se muestra una ventana llamada Botonera, la cual simula los botones y la pantalla de un horno real. A continuación, se explica la función de cada botón de esta ventana.

Botón de Encendido / Apagado. Enciende o apaga el horno. Para iniciar la simulación es necesario que este botón sea presionado.

Botón de Paro / Reset. Se utiliza para detener el proceso de control y/o reiniciar los parámetros de control previamente programados.

Botón de Flecha Izquierda. Se utiliza para desplazarse por los dígitos de la pantalla LCD del horno al momento de programar los parámetros de temperatura y tiempo.

Botón de Flecha Arriba. Permite el desplazamiento de manera ascendente sobre las opciones del menú.

Botón de Flecha Abajo. Permite el desplazamiento de manera descendente sobre las opciones del menú.

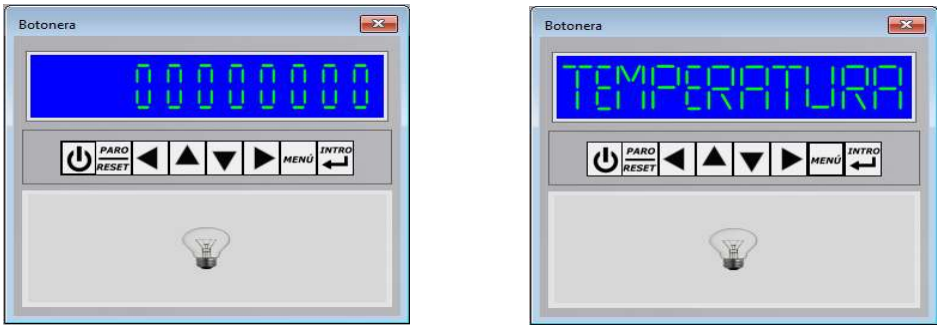
Botón de Flecha Derecha. Se utiliza para desplazarse por los dígitos de la pantalla LCD del horno al momento de programar los parámetros de temperatura y tiempo.

Botón Menú. Accede a los niveles del menú. El primer nivel es el menú principal.

Botón Intro. Permite escoger una opción del menú; además de, introducir y confirmar al controlador difuso, los parámetros de temperatura y tiempo programados. También, sirve para confirmar que el usuario desea iniciar el proceso de control.

Para poder iniciar la simulación es necesario encender el horno, haciendo clic en el botón de Encendido / Apagado. Luego se debe presionar el botón de Menú para acceder al menú principal y poder programar los parámetros de temperatura y tiempo. En caso de que el usuario no programe el parámetro de tiempo, el sistema lo considera como una simulación libre de tiempo; es decir, el usuario es el que estará pendiente del tiempo de simulación, no así el sistema. Pero, en caso de que, sí se programe el parámetro de tiempo, será el sistema que estará pendiente del tiempo de simulación.

En la siguiente figura se muestra la ventana Botonera al encender el horno, y posteriormente al acceder al menú principal.



A B

Figura 31. Apariencia de la ventana Botonera. A. Luego de presionar el botón de encendido. B. Luego de presionar el botón Menú.

Observe en la figura 32 que, los botones de la ventana botonera, también se encuentran en la parte superior izquierda del MDI padre, en el panel ubicado debajo de las tres pestañas de Archivo, Info y Ayuda. Estos botones tienen la misma funcionalidad que los botones ubicados en la ventana Botonera.

Figura 32. Parte superior de la ventana MDI padre.

Dentro de la pestaña Archivo está otra opción para cerrar el programa. La pestaña Info contiene la opción Acerca de, en la cual se da información del proyecto. La pestaña Ayuda contiene la opción de Manual de usuario.

Vea que, debajo de estas tres pestañas está un panel con varios botones e indicadores; además de, los botones que previamente se dieron a conocer. Veamos la funcionalidad de estos.

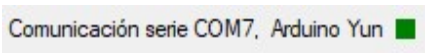
Botón de reglas difusas. Presenta la ventana donde se visualizan las reglas difusas de forma gráfica. Esta es la ventana que se muestra por defecto al iniciar el proceso de simulación, luego de que se han programado los parámetros de temperatura y tiempo.

Botón de grados de pertenencia y cortes. Presenta la ventana donde se visualizan los grados de pertenencias de cada antecedente y el corte de cada consecuente.

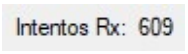
Botón de unión de cortes. Presenta la ventana donde se visualiza gráficamente el resultado de la unión de todos los cortes de los consecuentes; también, presenta la ventana que muestra de forma gráfica las mediciones hechas por los sensores de temperatura.

Botón de matriz difusa. Presenta la ventana donde se visualiza la matriz difusa que contiene las reglas difusas.

Botón de términos lingüísticos. Presenta la ventana donde se pueden visualizar gráficamente los términos lingüísticos del error relativo, la variación de temperatura y la potencia de salida.

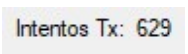


Indicador de comunicación serial. Indica el puerto serie utilizado para la comunicación con la tarjeta Arduino. En este caso, se está utilizando el puerto COM7 y la tarjeta Arduino Yun. El cuadro de color verde indica que la comunicación se está realizando de manera correcta. Si el cuadro cambia a color rojo, indica que ha habido un problema en la comunicación.



Indica el número de peticiones de recepción que ha hecho el controlador difuso para obtener la temperatura medida a través la tarjeta Arduino.

Indica el número de errores de recepción que ha habido desde el inicio del proceso de simulación.



Indica el número de peticiones de transmisión que ha hecho el controlador difuso para enviar la potencia de salida a la tarjeta Arduino.

Errores Tx: 0 Indica el número de errores de transmisión que ha habido desde el inicio del proceso de simulación.

La figura 33 muestra el proceso de simulación ya iniciado. Nótese que, dentro de la ventana MDI padre están tres ventanas, una de ellas es la ventana Botonera ubicada en parte superior izquierda, debajo del panel de botones e indicadores de comunicación serial. La pantalla de la ventana Botonera mostrará cada segundo de por medio la temperatura medida, y en el segundo intermedio mostrará el tiempo transcurrido. Debajo de la ventana Botonera, se encuentra la ventana de Indicadores. Esa ventana muestra los parámetros de simulación programados, la temperatura actual y los resultados actuales en las variables de control. En la parte izquierda, la parte más grande, se muestra la ventana de reglas difusas; recuérdese que, anteriormente se mencionó que, esta ventana es la ventana que se muestra por defecto al iniciar el proceso de simulación, pero lo que no se dijo en ese entonces que, esa ventana estaba acompañada por las ventanas Botonera e Indicadores. Esas dos últimas ventanas siempre se mantienen en esa posición; es decir, no se ocultan a menos que el usuario las cierre.

En la parte más grande es donde se muestran las demás ventanas que son habilitadas por los botones del panel. Esa parte del MDI padre, es la parte donde se muestran todas las ventanas que contienen gráficos, y cuando se habilita una de ellas, se oculta la o las que estaban visualizándose anteriormente.

La ventana de reglas difusas y la de grados de pertenencias y cortes, tienen dos botones en la parte inferior; uno en la parte izquierda, y otro en la parte derecha, los cuales son para ir visualizando las demás reglas; estas se visualizan de tres en tres, las etiquetas sobre ellas muestran las variables a las que pertenecen las gráficas de cada columna. Mientras que, las etiquetas al lado izquierdo muestran el número de la regla que se está presentando en esa fila.

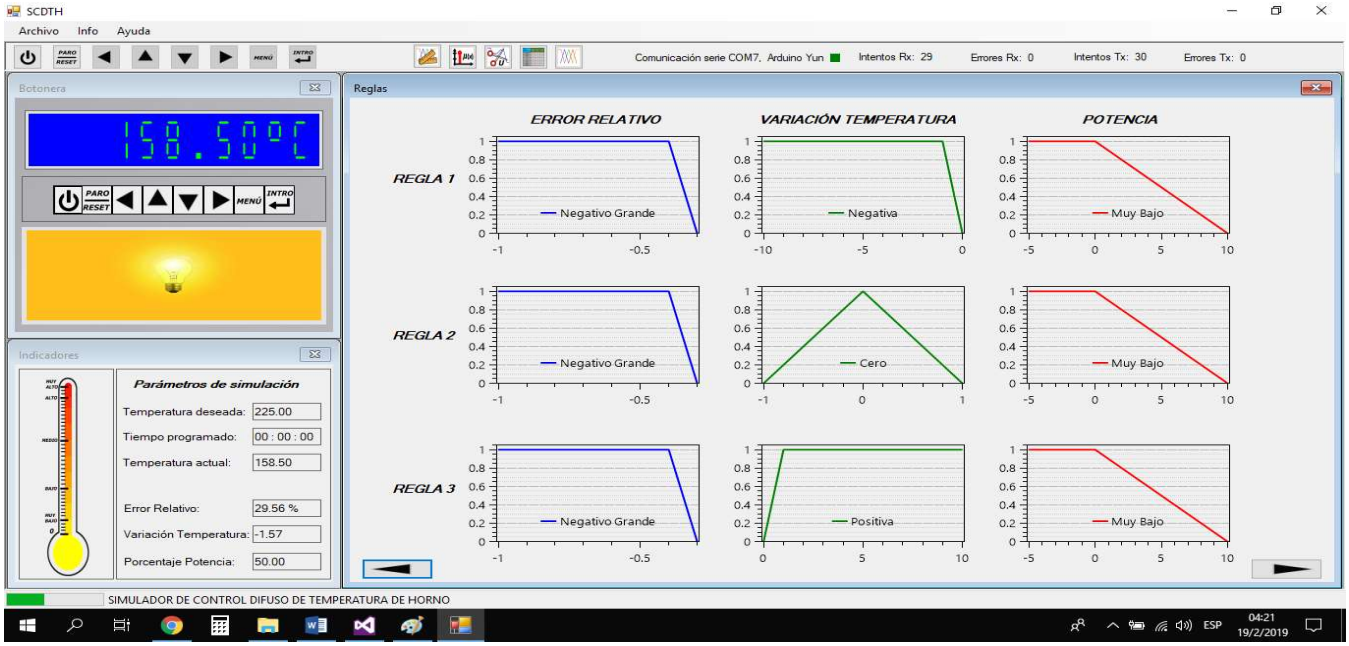


Figura 33. Proceso de simulación iniciado.

En las siguientes figuras se mostrarán las otras ventanas gráficas que se encuentran dentro del MDI padre.

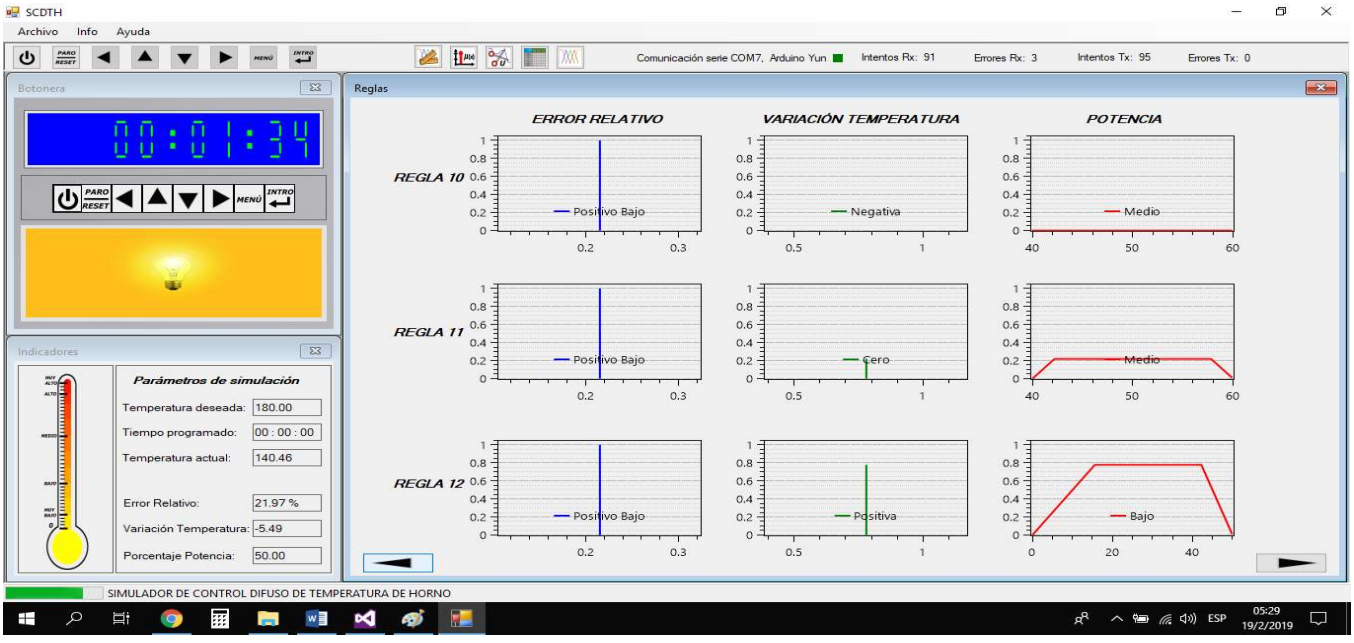


Figura 34. MDI padre mostrando ventana de grados de pertenencias y cortes.

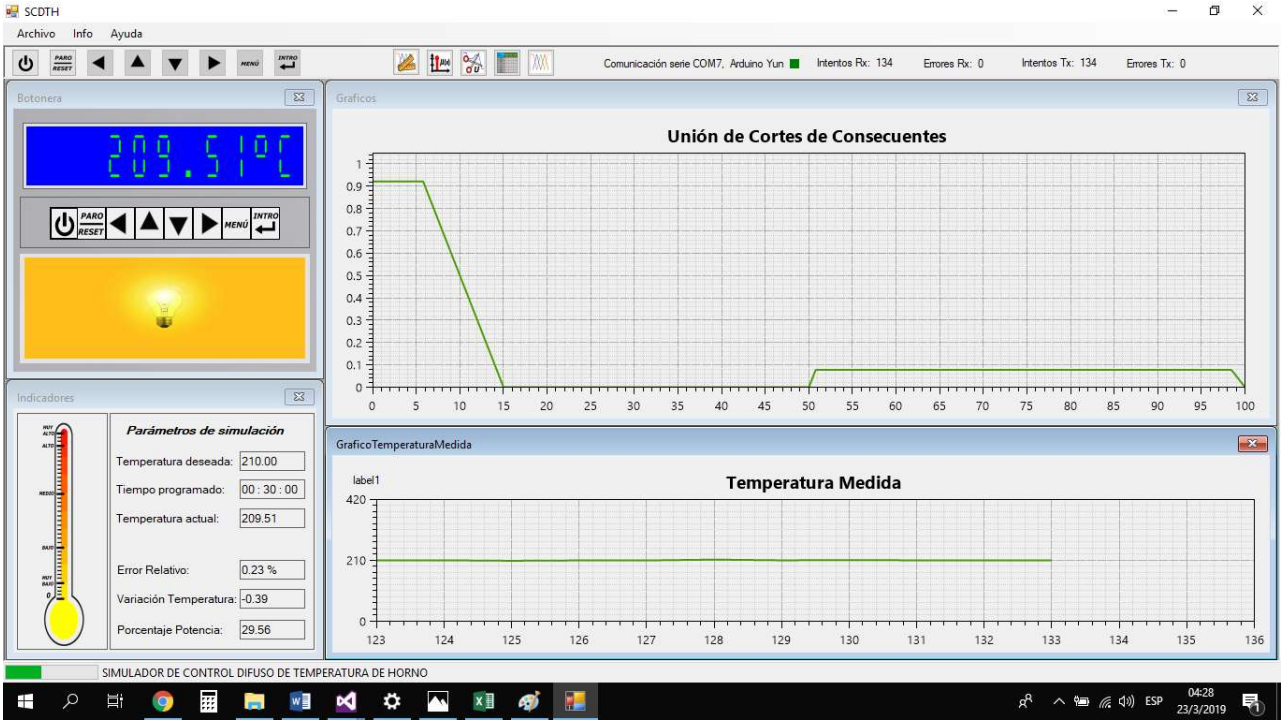


Figura 35. MDI padre mostrando ventana de unión de cortes y ventana de temperatura medida.

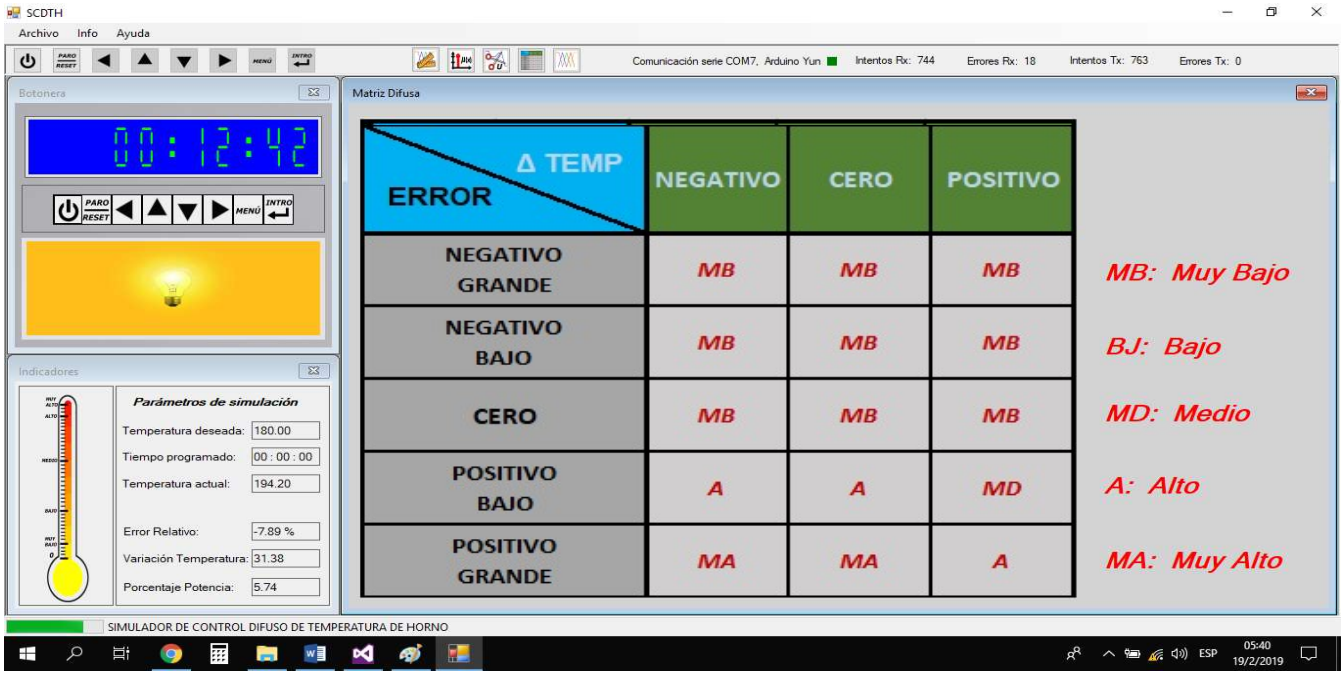


Figura 36. MDI padre mostrando ventana de matriz difusa.

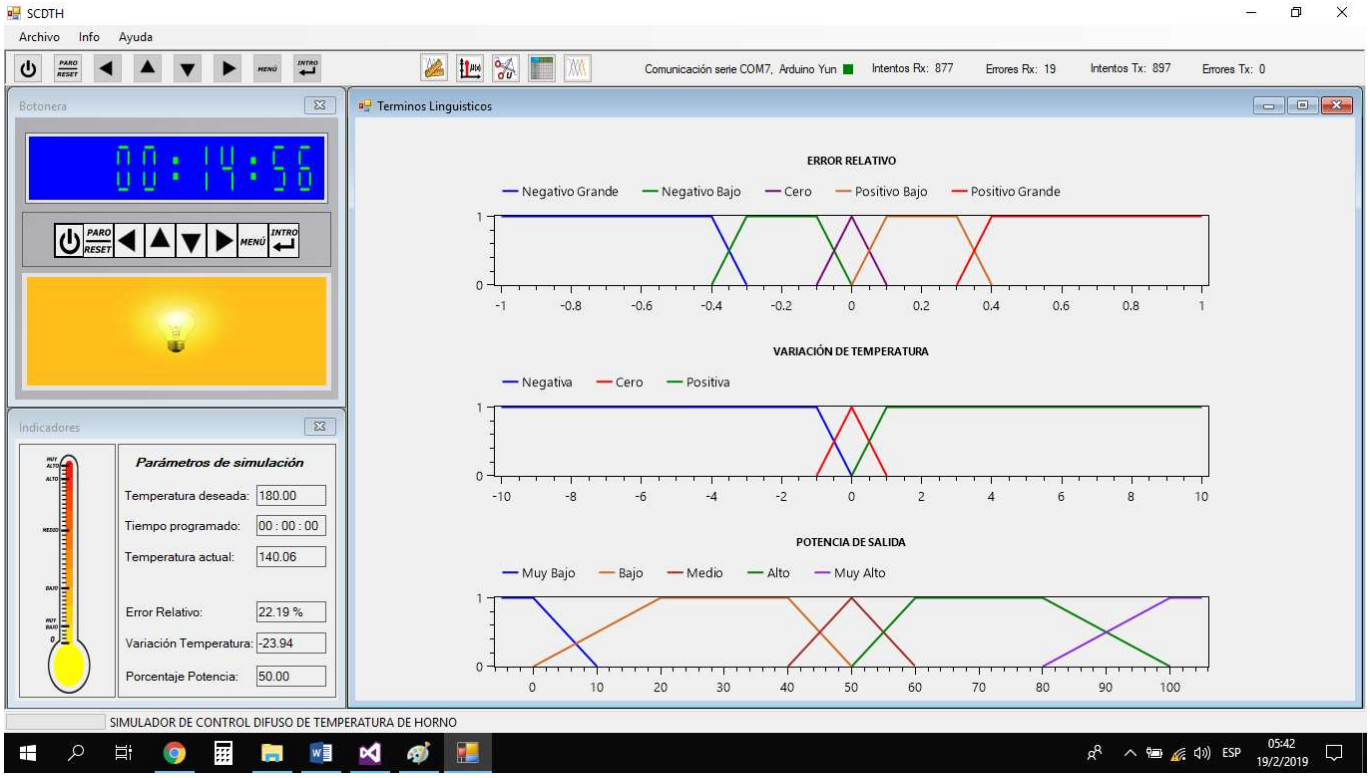


Figura 37. MDI padre mostrando ventana de términos lingüísticos.

Codificación

Para la codificación se ha hecho uso de dos paradigmas de programación. En la programación del Simulador de Control Difuso de Temperatura de Horno (SCDTH) se ha usado el paradigma de programación orientada a objetos, codificando en lenguaje de programación C# de la plataforma .Net. En otro orden, para programar la tarjeta Arduino se utilizó el paradigma imperativo, codificando en el lenguaje de programación C a través del IDE de Arduino.

Simulador de Control Difuso de Temperatura de Horno (SCDTH)

El corazón del sistema SCDTH está basado en siete de las nueve clases presentadas en el diagrama de clases de la figura 22. Estas clases son:

FuncionesDePertenenciaFuzzy, VariableLinguistica, Error, VariacionTemperatura, PWM, ReglaDifusa y MotorDeInferencia.

La interacción de estas clases es esencial para el funcionamiento del sistema SCDTH, ya que ellas son la parte difusa del sistema. Su interacción es la siguiente.

La clase MotorDeInferencia es la que hace el proceso de control difuso, a ella llegan los parámetros de simulación programados por el usuario y la temperatura medida por la tarjeta Arduino. Sin embargo, para realizar el proceso de control, el motor de inferencia necesita de las reglas difusas para ser evaluadas; es aquí donde entra en juego la clase ReglaDifusa, cada regla difusa que es evaluada por el motor de inferencia es un objeto de esta clase.

Por otro lado, recuérdese que las reglas difusas están compuestas de antecedentes y consecuentes; por tanto, la clase ReglaDifusa posee dos atributos llamados *antecedenteA* y *antecedenteB*, adicionalmente, otro llamado *consecuente*. En el proceso de diseño del controlador difuso definimos que el Error Relativo y la Variación de Temperatura serían los antecedentes de nuestras reglas; en consecuencia, se definieron las clases Error y VariacionTemperatura. También en el proceso de diseño del controlador difuso se definió que la potencia de salida sería el consecuente de las reglas difusas, por tanto, se definió la clase PWM. Con referente al nombre – PWM – al principio se había planeado hacer el control de la potencia de salida por medio de Modulación de Ancho de Pulso (PWM), pero luego se descubrió que era más fácil hacerlo controlando el porcentaje de la señal AC que le llega a la bombilla. He allí del nombre PWM.

Los antecedentes y consecuentes de las reglas difusas son variables lingüísticas que varían su tipo en cada regla; por tal razón, se necesita de la clase VariableLinguistica. Cabe recordar que, una variable lingüística necesita de una función de pertenencia para ser evaluada; he aquí la necesidad de definir la clase FuncionesDePertenenciaFuzzy. En esta última clase se definieron las funciones matemáticas que se utilizan como función de pertenencia de cada antecedente y de cada consecuente.

La clase ControlIO es la encargada del control de las entradas y salidas del sistema. Recibe los parámetros de simulación del usuario y los pasa al motor de inferencia, recibe las lecturas de temperatura del Arduino, los acondiciona a un formato específico para luego pasarlos al motor de inferencia. Adicionalmente, recibe el dato de la potencia de salida del motor de inferencia, lo acondiciona a un formato específico, para luego pasarlo a la tarjeta Arduino por medio de la clase ComSerial. La clase ComSerial está encargada de la comunicación entre el sistema SCDTH y la tarjeta Arduino, en ella están presente el método de recibir y el método de transmitir los datos entre ambos sistemas.

Existen otras clases que nos son meramente parte del corazón funcional del sistema SCDTH; es decir, son clases auxiliares para poder mostrar gráficos como la clase GraficosDifusos, o la clase ControlLCD para poder controlar lo que se debe mostrar en la pantalla LCD del horno simulado. La clase OperacionesMatematicasFuzzy contiene métodos para realizar las operaciones matemáticas básicas de la lógica difusa.

Se ha hecho uso de la librería OxyPlot para la parte gráfica del sistema; la parte que muestra los gráficos difusos del proceso de simulación. Esta librería para .Net está bajo licencia MIT y permite realizar un sin número de gráficas; pero, tiene algunos inconvenientes, los cuales son, la poca documentación de esta; aunque en la web se encuentran ejemplos de cómo realizar las gráficas, mayormente para aplicaciones WPF, y en poca cantidad para aplicaciones de WindowsForm, su documentación oficial es muy pobre e incompleta.

Programación de la tarjeta Arduino

A como se mencionó con anterioridad, la tarjeta Arduino se programó en lenguaje C a través del IDE de Arduino, el cual puede ser descargado desde la página oficial de Arduino - https://www.arduino.cc/.

Las funciones de Arduino utilizadas, esenciales para la codificación de nuestro subsistema fueron las siguientes.

analogRead(): Permite la lectura de una entrada analógica. Es la función utilizada para leer los sensores LM35.

digitalRead(): Permite la lectura de una entrada digital. Con esta función se detecta cuándo la señal AC hace el cruce por cero.

digitalWrite(): Permite activar o desactivar una salida digital. Esta función es la que permite encender, en el momento correcto, la bombilla eléctrica, para que la misma consuma exactamente la potencia de salida calculada por el sistema SCDTH.

attachInterrupt(): Permite enlazar un pin de entrada digital a una interrupción externa; es decir, a un suceso o evento. Cuando se da el cruce por cero de la señal AC, el pin digital de entrada lo detecta, pero como está enlazado a una interrupción, el programa interrumpe lo que en ese momento está ejecutando, lo deja en espera, y se dedica a ejecutar la rutina de interrupción, la cual contiene las instrucciones de lo que debe ejecutar cuando el pin detecta un cruce por cero.

En los anexos se explicará lo que es un cruce por cero, el circuito para detectarlo y pasarlo al pin de entrada del Arduino; además de, los otros circuitos necesarios para manejar el control del hardware.

5.3.4. Fase de Transición

En cada incremento del software se fueron haciendo pruebas funcionales a cada módulo que se agregaba al sistema, y se documentaron los errores más significativos, los que impedían el correcto funcionamiento del módulo. Por otro lado, los errores de sintaxis se corregían, pero no se documentaron por ser errores de programación, no de lógica ni de funcionamiento. De igual manera, sólo se documentaron los errores funcionales del sistema en general, ya cuando todo el sistema estaba integrado por todos sus módulos funcionales. Se revisó el correcto funcionamiento de la pantalla LCD del horno, el funcionamiento de cada botón, de cada indicador, de cada gráfica y hasta la conexión de la tarjeta Arduino.

Problemas de caja blanca

NO. 1	NOMBRE: PRUEBA DE GRÁFICO DE VALOR DE PERTENENCIA
Objetivo de prueba	Constatar la correcta funcionalidad del método graficarValorDePertenecia(), asegurando la visualización gráfica del valor exacto del grado de pertenencia de un valor medido en un conjunto difuso.
Módulo / Clase	GraficosFuzzy.cs
Atributo / Método / Función	`public FunctionSeries graficarValorDePertenencia(double valorMedido, double valorDePertenencia)`
Descripción del problema	La gráfica dibujada siempre se truncaba a un valor menor. Por ejemplo: si el punto a graficar era (150, 0.8333), este se truncaba a (150, 0.8). Esto no permitía visualizar de manera correcta el grado de pertenencia de un valor medido.
Solución	Dentro del método se calculaban los pares ordenados de la gráfica a dibujar, desde 0 hasta llegar al valor de pertenencia, variando solamente los valores de Y, porque el valor de X queda constante por ser el valor medido. Esto se hacía con un ciclo FOR, el cual tenía un incremento para Y de 0.0001. Para resolver el problema, se decidió eliminar el ciclo FOR que calculaba muchísimos valores innecesarios, y dejar solo dos pares ordenados ya definidos. El primero es (Valor medido, 0) y el segundo es (Valor medido, grado de pertenencia).

NO. 2	NOMBRE: PRUEBA DE FUNCIONALIDAD DE LA CLASE VARIABLELINGUISTICA
Objetivo de prueba	Verificar el funcionamiento adecuado de la clase VariableLinguistica.cs, de sus atributos y métodos.
Módulo / Clase	VariableLinguistica.cs
Atributo / Método / Función	Todos los atributos de la clase.
Descripción del problema	Los valores de los atributos siempre permanecían sin cambio, aunque en el código se trataran de cambiar.

Solución	El problema se daba debido a que, se hacía uso de propiedades autoimplementadas para tratar de cambiar los valores. Pero, al utilizar propiedades autoimplementadas el compilador crea atributos temporales ligados a esas propiedades; por tanto, no hay necesidad de declarar atributos. En consecuencia, los atributos físicos declarados eran totalmente diferentes a los atributos temporales de las propiedades autoimplementadas. Pero, como los atributos deben almacenar valores que se utilizan en todo el programa, estos no pueden ser temporales. Por tal razón, las propiedades se dejaron como propiedades normales, y no autoimplementadas.

NO. 3	**NOMBRE: PRUEBA DE FUNCIONALIDAD DEL MÉTODO CALCULARCONJUNTOCORTECONSECUENTE().**
Objetivo de prueba	Comprobar que la lista devuelta por el método calcularConjuntoCorteConsecuente() esté en concordancia con todos los cortes de los consecuente de las reglas difusas.
Módulo / Clase	ReglaDifusa.cs
Atributo / Método / Función	`public List<FunctionSeries> calcularConjuntoCorteConsecuente(double valorMedido)`
Descripción del problema	El compilador manda error: índice fuera de rango.
Solución	La lista se trataba como que si fuera un arreglo unidimensional al agregar un elemento a la misma. La solución del problema fue utilizar el método Add de las listas.

NO. 4	**NOMBRE: PRUEBA DE GRÁFICOS DE ANTECEDENTES**
Objetivo de prueba	Asegurar la correcta visualización de los gráficos de los antecedentes de las reglas difusas.
Módulo / Clase	ReglaDifusa.cs
Atributo / Método / Función	`public List<FunctionSeries> GraficarAntecedentes()`

Descripción del problema	El compilador manda error: índice fuera de rango.
Solución	Corrección de la condición de entrada del ciclo FOR que llama al método.

NO. 5	**NOMBRE: PRUEBA DE FUNCIONALIDAD DEL MÉTODO OBTENERPERTENENCIAMINIMADEANTECEDENTES()**
Objetivo de prueba	Verificar que el valor devuelto por el método ObtenerPertenenciaMinimaDeAntecedentes() sea correcto, de acuerdo a los grados de pertenencia de los valores medidos en cada uno de los antecedentes de cada regla difusa.
Módulo / Clase	ReglaDifusa.cs
Atributo / Método / Función	`public double ObtenerPertenenciaMinimaDeAntecedentes(double valorMedido)`
Descripción del problema	El compilador manda error: índice fuera de rango.
Solución	Este método tiene un ciclo FOR; en el cual, la condición de parada depende del número de elementos en una lista. Se ejecutaba un ciclo más después de haber recorrido todos los elementos de la lista, consecuentemente surgía un error fuera de rango del número de elementos de la lista. Esto se daba debido a que la condición era que i (número de ciclos) fuera <= que el número de elementos del arreglo. Eliminando el signo =, se corrigió el problema.

NO. 6	**NOMBRE: PRUEBA DE GRÁFICOS DE ANTECEDENTES**
Objetivo de prueba	Asegurar que la gráfica de los antecedentes de cada regla difusa sea exacta, de acuerdo a sus funciones de pertenencia y sus respectivos parámetros.
Módulo / Clase	ReglaDifusa.cs

Atributo / Método / Función	`public List<FunctionSeries> GraficarAntecedentes();`
Descripción del problema	El compilador lanza el siguiente error: El índice estaba fuera del intervalo. Debe ser un valor no negativo e inferior al tamaño de la colección. El problema se debía a que se pretendía guardar elementos en una variable de tipo lista, tratándola como que si fuera un arreglo.
Solución	Usar el método Add() de la clase Lista, para ir guardando cada elemento en la variable de tipo Lista.

NO. 7	**NOMBRE: PRUEBA DE GRÁFICOS DE ANTECEDENTES**
Objetivo de prueba	Asegurar que la gráfica de los antecedentes de cada regla difusa sea exacta, de acuerdo a sus funciones de pertenencia y sus respectivos parámetros.
Módulo / Clase	ReglaDifusa.cs
Atributo / Método / Función	`public List<FunctionSeries> GraficarAntecedentes()` `private FunctionSeries puntosAntecedenteA private FunctionSeries puntosAntecedenteB`
Descripción del problema	El compilador lanza el siguiente error: Referencia a objeto no establecida como instancia de un objeto. El problema era el siguiente. En el método `GraficarAntecedentes()` se obtenían los puntos de los antecedentes A y B, pero éstos no se guardaban en sus respectivos atributos, *puntosAntecedenteA* y *puntosAntecedenteB*; por tanto, sus valores correspondientes eran nulos. Al tratar de usar estos atributos,el compilador lanzaba el error.
Solución	Guardar los puntos de los antecedentes en sus respectivos atributos.

NO. 8	NOMBRE: PRUEBA DE FUNCIONALIDAD DE CLASE ERROR.CS
Objetivo de prueba	Comprobar la correcta funcionalidad de la clase Error.cs, sus atributos y sus métodos.
Módulo / Clase	Error.cs
Atributo / Método / Función	Todos los atributos de la clase.
Descripción del problema	Se hacía uso de propiedades autoimplementadas para acceder a los atributos. Por lo tanto; al hacer referencia a un atributo de la clase, el compilador enviaba un error de referencia a un objeto nulo.
Solución	Se cambiaron las propiedades autoimplementadas a propiedades normales.

NO. 9	NOMBRE: PRUEBA DE FUNCIONALIDAD DE CLASE VARIACIONTEMPERATURA.CS
Objetivo de prueba	Comprobar la correcta funcionalidad de la clase VariacionTemperatura.cs, sus atributos y sus métodos.
Módulo / Clase	VariacionTemperatura.cs
Atributo / Método / Función	`public static void InicializarVariacionesTemperatura()`
Descripción del problema	En este método se inicializan los valores de los tipos de variaciones de temperatura. Cada tipo de variación tiene un atributo N, el cual es utilizado como límite para graficarlo. El inconveniente era que, al graficar el tipo de variación y su grado de pertenencia en el mismo plotview, el gráfico del tipo de variación apenas se podía visualizar. Esto se debía a que su valor de N estaba super bajo para ser visualizado de manera satisfactoria en el plotview.
Solución	Cambiar el valor del atributo N de 2 a 600 para la variación de temperatura positiva y de -2 a -600 para la variación de temperatura negativa. Con esto la gráfica del valor de pertenencia de la variación se visualiza dentro de los rangos de la gráfica del tipo de variación de temperatura.

NO. 10	NOMBRE: PRUEBA DE FUNCIONALIDAD DEL MÉTODO AGREGARCORTE()
Objetivo de prueba	Asegurar la correcta funcionalidad del método AgregarCorte(), en el cual se agregan cada uno de los cortes de los consecuentes para posteriormente obtener la unión de todos estos cortes.
Módulo / Clase	MotorDeInferencia.cs
Atributo / Método / Función	`private static void AgregarCorte()`
Descripción del problema	El compilador lanza error: Índice fuera de rango. El método está diseñado para añadir y retornar 1000 puntos de corte. Esto se hace con el fin de acondicionar todas las funciones de corte que contengan uno o varios puntos de corte mayores a cero, para luego hacer la comparación entre todos ellos. El problema se generaba con valores de medición de temperatura negativos, los positivos funcionaban de manera correcta. Sucedía que con valores negativos se generaban el doble de puntos, es decir 2000, y el método se llamaba dentro de un ciclo For de 1000 ciclos, y allí ocurría el error fuera de índice.
Solución	La solución a este problema fue colocar una condición para generar los puntos de corte, la cual fue que, la frontera derecha de la función a evaluar fuera mayor que cero. Esto se hizo para eliminar puntos de cortes que comenzaran desde la abscisa con valor 0, y que se agregaran los verdaderos valores que comienzan desde la abscisa con valor límite derecho + 0.1.

NO. 11	NOMBRE: PRUEBA DE VERIFICACIÓN DE GRÁFICO DE UNIÓN DE LOS CONSECUENTES
Objetivo de prueba	Verificar que el gráfico resultante de la unión de los cortes de los consecuentes esté de acuerdo a los valores de los parámetros de cortes incluidos en la lista de cortes.
Módulo / Clase	MotorDeInferencia.cs
Atributo / Método / Función	`List<FunctionSeries> listaDeCortes`

	`FunctionSeries puntosDeUnionCortesConsecuentes`
Descripción del problema	El problema surgía cuando se llamaba a graficar los puntos de unión de los cortes de los consecuentes. En cada medición, los cortes son diferentes, a menos que la medición sea el mismo resultado que la anterior; eso mismo sucede con los puntos de medición. Por tanto, cada gráfica de los puntos de unión tiene que ser diferente, y ésta se presenta en pantalla. Sin embargo, todas las gráficas de los diferentes puntos de unión se presentaban a la vez. En cada medición se iba agregando una gráfica más en la pantalla. Esto era debido a que, todos los puntos de corte quedaban almacenados en el atributo `List<FunctionSeries> listaDeCortes.` Igualmente sucedía con los puntos de unión, los cuales quedaban almacenados en el atributo `FunctionSeries puntosDeUnionCortesConsecuentes.` Debido a que la clase MotorDeInferencia.cs es estática, entonces todos sus atributos y métodos son estáticos; por tanto, siempre mantienen sus valores, a menos que sean cambiados en el código. Como los dos atributos almacenan puntos de coordenadas, cada vez que se hacía una nueva medición, los nuevos puntos se añadían a los anteriores. Así, de esa manera, se dibujaban todos los puntos de la nueva medición y los puntos de las mediciones anteriores,
Solución	La solución a este problema consistió en limpiar todos los puntos de ambos atributos, haciendo uso del método clear() para las listas y para las FunctionSeries. Esto se hizo en el método ProcesoDeInferenciaDifusa(), ya que, es a este método al que se invoca en cada medición.

NO. 12	**NOMBRE: PRUEBA DE FUNCIONALIDAD DE LA CLASE CONTROLIO.CS**
Objetivo de prueba	Comprobar el correcto funcionamiento de la clase ControlIO; además de, sus atributos y métodos.
Módulo / Clase	ControlIO.cs
Atributo / Método / Función	`public static void IniciarProceso()`
Descripción del problema	El compilador enviaba un mensaje en donde se indicaba que la cadena recibida por el puerto serial no tenía el formato adecuado. Esto ocurría porque el dato recibido contenía una cadena con caracteres alfabéticos y numéricos. Por tanto, al hacer la conversión de la cadena completa a double, se producía el error.

Solución	Eliminar los caracteres alfabéticos en el código de envío de la tarjeta Arduino, para que la misma solo enviara el dato numérico de temperatura.

NO. 13	**NOMBRE: PRUEBA DE FUNCIONAMIENTO DE CÓDIGO ARDUINO**
Objetivo de prueba	Asegurar el correcto funcionamiento de las rutinas de interrupción y de la rutina principal del código de la tarjeta Arduino.
Módulo / Clase	ControlTempDimmer.ino (Código de Arduino)

Atributo / Método / Función	`void DeteccionCrucePorCero()`
	`void InterrupcionCLK()`
	`void loop()`

Descripción del problema	No se realizaba el control de potencia del dimmer.
Solución	Agregar un retardo al final en void loop().

NO. 14	**NOMBRE: PRUEBA DE FUNCIONAMIENTO DE CÓDIGO ARDUINO**
Objetivo de prueba	Asegurar el correcto funcionamiento de las rutinas de interrupción y de la rutina principal del código de la tarjeta Arduino.
Módulo / Clase	ControlTempDimmer.ino (Código de Arduino)
Atributo / Método / Función	`void loop()`
Descripción del problema	No se recibía correctamente el dato numérico enviado por la pc. Esto ocurría porque el dato recibido es de tipo string y se almacenaba en una variable float, y no se hacía la respectiva conversión de tipo de dato.
Solución	Realizar conversión de tipo de dato, de string a float.

Problemas de caja negra

NO. 1	NOMBRE: VERIFICACIÓN DEL FUNCIONAMIENTO DEL MENÚ
Objetivo de prueba	Verificar la correcta funcionalidad del menú principal y sus niveles.
Descripción del problema	El menú no regresa a su nivel anterior cuando se está en el nivel de configuración. Siempre regresa al nivel 0.
Causa	Los niveles del menú no se evaluaban según el estado de la bandera de configuración en el método de control del botón menú. Esto hacía que el flujo del programa se fuera por el camino donde se llamaba al nivel 0 del menú.
Solución	Evaluar los niveles del menú de acuerdo con el estado de la bandera de configuración. Si la bandera de configuración no está activada, el nivel del menú aumentará en una unidad, en caso contrario, al momento de estar activada el nivel del menú decrementará en una unidad.

NO. 2	NOMBRE: VERIFICACIÓN DE LA CORRECTA VISUALIZACIÓN DE LOS DÍGITOS EN LA PANTALLA LCD
Objetivo de prueba	Asegurar que los dígitos de la pantalla LCD sean visualizados de la manera correcta.
Descripción del problema	Además de la posición 0 de los dígitos LCD, aparece otra con fondo negro al momento de regresar al nivel de configuración.
Causa	Cuando se llamaba al método que presenta el dígito LCD que se estaba cambiando, el atributo del número LCD quedaba con el valor de la configuración anterior.
Solución	Crear un nuevo método, `ConfiguracionFondoNumeroLCD()`, que sea invocado antes de presentar los dígitos LCD.

NO. 3	NOMBRE: VERIFICACIÓN DE CORRECTA VISUALIZACIÓN DE LA TEMPERATURA PROGRAMADA
Objetivo de prueba	Constatar que la temperatura programada, por el usuario, sea la misma que se visualiza en la venta de indicadores.
Descripción del problema	No se muestra la temperatura programada de manera exacta. Solo se muestran correctamente los dígitos enteros, los dígitos fraccionarios se truncan a 0.
Causa	Declaración de tipo de atributo errónea. Los atributos décimas y centésimas de temperatura estaban declarados de tipo entero. Por tal razón, éstos al ser mostrados en pantalla se truncaban a 0.
Solución	Cambiar el tipo de atributo, en su declaración, de Int a double. Esto se hizo para los dos atributos, centésimas y décimas de temperatura.

NO. 4	NOMBRE: VERIFICACIÓN DE POTENCIA DE SALIDA EN EL ACTUADOR
Objetivo de prueba	Comprobar que la potencia de salida en el actuador concuerde con la potencia requerida por el controlador difuso.
Descripción del problema	El dispositivo calentador (bujía) no se mantenía en el nivel de potencia requerido por el controlador. Éste se encendía y apagaba permitiendo una variación no adecuada de temperatura.
Causa	El problema estaba localizado en el código de la tarjeta Arduino. Este código recibía el dato que le pasaba el controlador difuso a través de la función Serial.parseInt(). Esta función se encarga de convertir el dato recibido, por medio del puerto serie, a tipo entero. Pero su inconveniente es que, cuando lee los datos del buffer de recepción, hace que el buffer quede sin ningún dato; es decir, reinicia el buffer, haciendo que la lectura la mayor parte del tiempo sea 0, y por una menor parte del tiempo sea igual a la potencia deseada. Por tal razón, la bujía encendía y apagaba.
Solución	La solución fue crear un vector de tipo char de longitud 3, para poder utilizar la función Serial.readByteUntil(). Esta función permite leer el buffer de recepción, pero sin reiniciarlo. El inconveniente es que, el dato que se necesita es de tipo entero,

	pero eso se soluciona haciendo una conversión de tipo de dato, de Byte a Int.

NO. 5	NOMBRE: PRUEBA DE CIERRE DEL CONTROLADOR
Objetivo de prueba	Garantizar el correcto cierre del controlador difuso.
Descripción del problema	El messageBox que pregunta si desea cerrar la aplicación, volvía aparecer cuando se escogía la opción Sí. Es decir, no se cerraba la aplicación en el primer momento cuando se daba click en el botón sí, sino que, el messageBox volvía aparecer y se tenía que dar click en la misma opción, para que se cerrara la aplicación.
Causa	No se liberaban los recursos del formulario que llamaba al messageBox.
Solución	Colocar una instrucción Dispose(), antes de la instrucción Application.Exit(), para liberar los recursos del formulario.

NO. 6	NOMBRE: VERIFICACIÓN DE CONEXIÓN DE TARJETA ARDUINO A LA PC
Objetivo prueba	Confirmar que el controlador difuso detecte la conexión o ausencia de conexión de la tarjeta Arduino.
Descripción del problema	Se quebraba el programa después de enviar el mensaje de error de tarjeta Arduino no conectada al puerto serie.
Causa	Cuando se da este error, de que la tarjeta Arduino no está conectada al puerto serie, el flujo de programa manda a cerrar la aplicación. Sin embargo, como esta conexión de la tarjeta se verifica antes de que se inicialicen todas las demás clases de la aplicación; hay una bandera de cierre que no se inicializa, y por ende el compilador manda un error de inicialización.

Solución	Usar una bandera de inicialización en la ventana de Inicio, con el propósito de evadir, a través de un condicional IF, el segmento de código que usa la bandera no inicializada.

NO. 7	**NOMBRE: PRUEBA DE FUNCIONAMIENTO DE BOTONES DEL PANEL DEL MDI PADRE**
Objetivo de prueba	Verifcar la correcta visualización de los correspondientes gráficos que se presentan al dar click en los respectivos botones del panel del MDI padre.
Descripción del problema	Al estar abierta la ventana de la matriz difusa, y luego se daba click en cualquiera de los dos íconos que activan la ventana Reglas; esta última ventana no se abría. Ocurría también lo opuesto, si estaba abierta la ventana Reglas, la ventana de la matriz difusa no se abría.
Causa	En los métodos correspondientes que se manejan para abrir ambas ventanas, no se cerraba la ventana anteriormente mostrada. Es decir, si estaba abierta la ventana Reglas, al dar click para mostrar la ventana de la matriz difusa, en el método que abre la ventana de la matriz difusa no se cerraba la ventana Regla. También, ocurría lo contrario, si estaba abierta la ventana de la matriz difusa, al dar click para mostrar la ventana Reglas, en el método que abre la ventana Regla no se cerraba la ventana de la matriz difusa.
Solución	Cerrar ambas ventanas en los respectivos métodos, para que la ventana correspondiente pudiera ser visualizada.

5.4. Estudio de factibilidad

El estudio de factibilidad se ha realizado teniendo como referencia una estructura de acero inoxidable de 1.6m de alto por 0.8m de ancho, para un solo horno, cuyo funcionamiento es meramente eléctrico. Capacidad para 6 bandejas de 65x45cm.

5.4.1. Factibilidad operativa

NECESIDADES MÍNIMAS DE OPERACIÓN	
Cantidad de operarios	1
Habilidades mínimas del operario	Leer y escribir.
Conocimiento y habilidades especializadas del operario	Ninguna.
Capacitación acerca del sistema	Manual de usuario.
Fijación de parámetros de funcionamiento	Manual a través de botones.
Cuidados requeridos	• Mantenimiento periódico de al menos una vez al mes. • Mantener el horno en un lugar techado, ventilado, por sin exceso de corrientes de aire y libre de humedad.
En caso de incendio, tipos de extintores recomendados	• Extintor de dióxido de carbono (CO2). • Extintor de polvo químico seco ABC.

5.4.2. Factibilidad Técnica

Requerimientos de Hardware

ELEMENTO	CANTIDAD
Tarjeta Arduino Mega 2560 REV3 o compatible	1
Pantalla Touch TFT 3.2”	1
Tarjeta Mega 2560 Shield TFT LCD	1
Módulo Dimmer AC 20A.	2
Resistor tubular para horno 105 cm min.	2
Termopar tipo J con termo pozo	1
Fuente de alimentación 220VAC – 12VDC 5A.	1
Tarjeta conversor DC-DC 12VDC-5VDC	1
Contactor monofásico 220VAC 40A.	1
Tarjeta de relé 220VAC	1
Riel DIN	1
Latch On/Off SPST	1
Carcasa metálica para PCB 13x18x60 mm	1
PC 2GB RAM, 280GB HDD, procesador Core i3 2.00Hz	1

Requerimientos de software

- Ambiente de desarrollo Arduino IDE 1.8.9.
- Sistema operativo Windows 10 Pro.
- Librerías básicas de Arduino.
- Librería para manejo de dimmer.
- Librería para manejo pantalla LCD.

Requerimientos eléctricos

- Suministro de voltaje monofásico 220VAC.
- Tablero con breaker monofásico 220VAC 40A.
- Tomacorriente 220VAC 40A.

Recursos humanos

- Un programador
- Un técnico eléctrico
- Un técnico metalúrgico

5.4.3. Factibilidad Económica

Costos de Hardware

ELEMENTO	CANTIDAD	PRECIO UNITARIO ($)	PRECIO TOTAL ($)
Tarjeta Arduino Mega 2560 REV3 compatible	1	38	38.00
Pantalla Touch Shield TFT 3.2”	1	20	20.00
Módulo Dimmer AC 20A.	2	15	30.00
Resistor tubular para horno 105 cm min.	4	35	140
Termopar tipo J con termo pozo	2	15	30.00
Fuente de alimentación 220VAC – 5VDC 5A.	1	25	25.00
Contactor monofásico 220VAC 40A.	1	25	25.00
Tarjeta de relé 220VAC	1	8	08.00
Riel DIN	1	6	06.00
Latch On/Off SPST	1	5	05.00
Carcasa metálica para PCB 200x150x100 mm	1	15	15.00
Bornera cerámica de 4 polos	1	15	15.00
Borneras de plástico 12 polos	1	5	05.00
		Total $	362.00

Costo de materiales eléctricos

ELEMENTO	CANTIDAD	PRECIO UNITARIO ($)	PRECIO TOTAL ($)
Panel eléctrico Monofásico 220VAC 2 posiciones	1	20	20.00
Breaker Mononofásico 220VAC 40ª	1	30	30.00
Tomacorriente Monofásico 220VAC 40ª	1	25	25.00
Enchufe Macho Monofásico 220VAC 40ª	1	25	25.00
Cable TSJ 3x6 AWG (mts)	3.5	15	52.50
Ventilador	1	60	65.00
		Total $	217.50

Costo de estructura

ELEMENTO	CANTIDAD	PRECIO UNITARIO ($)	PRECIO TOTAL ($)
Lámina de acero inoxidable 0.8mm 3x1.22m	5	45	135.00
Tubo acero inoxidable calibre 18	3	45	225.00
Tornillos de 1/2 “ docena	2	5	10.00
Aislante térmico fibra de vidrio rollo 2x0.6m 1”	2	30	60.00
Bisagra	2	70	140.00
		Total $	570.00

Costos de mano de obra

ELEMENTO	PRECIO TOTAL ($)
Técnico metalúrgico	150.00
Técnico eléctrico	150.00
Programador	150.00
Total $	450.00

Costos totales

Costo	TOTAL ($)
Hardware	362.00
Materiales eléctricos	217.50
Estructura	570.00
Mano de obra	450.00
Costos de ingeniería	479.85
Gastos no previstos	623.81
IVA	405.50
Costo total $	**3,108.66**

Beneficios

- Cero riesgos por escape de gas licuado de petróleo (GLP).
- Permite que el hornero se dedique a otra actividad en el tiempo de cocción cuando el sistema está en control del tiempo.
- No hay necesidad de realizar limpieza periódica de tuberías con el fin de mantener una temperatura constante.
- Mantenimiento y limpieza más fácil.
- Más amigable con el medio ambiente al no haber combustión que generen gases residuales.

Desventajas

- El sistema no puede funcionar si no hay energía eléctrica.
- Se requiere de un contrato de suministro de energía de tipo industrial.
- Si un componente electrónico falla, falla todo el sistema.

6. CONCLUSIONES Y RECOMENDACIONES

6.1. Conclusiones

El software diseñado realiza, ciertamente, el trabajo de un controlador difuso de temperatura. Además, permite la comunicación vía puerto USB con una tarjeta Arduino, la cual, a su vez, hace de interfaz entre los sensores de temperatura y el actuador – el bombillo calentador. El trabajo realizado por todos los componentes de software y hardware ha permitido alcanzar, de manera satisfactoria, el objetivo general del proyecto.

La definición de los requerimientos funcionales y no funcionales permitió diseñar y codificar un sistema cuyas funcionalidades simulan, de un modo aproximado, las funcionalidades básicas de un horno real. Se han respetado todas las restricciones de los requerimientos funcionales; y adicionalmente, se han cumplido con todas las funcionalidades de los requerimientos funcionales.

Los dos tipos de arquitecturas que se presentaron inicialmente en los resultados, permiten describir la interacción de los componentes del sistema, especialmente la comunicación entre ellos. Pudimos observar que, el flujo de datos entre tales componentes no es igual; en algunos la interacción de la comunicación es unidireccional, mientras que, en otros, dicha interacción es bidireccional. Con el diagrama de clases se presenta la arquitectura del software basada en clases, donde la clase central es la clase MotorDeInferencia; ésta clase es la que evalúa las entradas de datos de temperatura y arroja el dato de potencia de salida, el cual controlará la potencia del bombillo.

La interfaz gráfica del sistema permite una fácil interacción con el usuario; en ésta, el usuario encuentra las funcionalidades básicas de un horno real, le permite programar la temperatura de control deseada. Adicionalmente, permite que el usuario programe un tiempo de simulación o que la simulación esté libre de tiempo, siendo el usuario, en este último caso, quien esté al tanto del control del tiempo. También, permite la visualización gráfica del proceso, a través de las diferentes gráficas del control difuso.

Por otra parte, se logró codificar cada uno de los módulos funcionales del controlador difuso, y del sistema en general, haciendo uso de dos paradigmas de programación, el paradigma imperativo para la programación en lenguaje C de la tarjeta Arduino, y el paradigma orientado a objeto para la programación del controlador difuso de temperatura, codificado en lenguaje C#. Unidos todos estos módulos funcionales, forman el sistema de simulación de control difuso de temperatura de un horno real.

Se realizaron pruebas de caja blanca y caja negra, lo que posibilitó corregir las líneas de código que no permitían el correcto funcionamiento de los diferentes módulos funcionales de manera individual y de manera colectiva. Esto ha permitido asegurar el funcionamiento adecuado de cada uno de los módulos funcionales y del sistema en general.

6.2. Recomendaciones

Para llevar a manufactura un horno con este sistema de control, es necesario que la computadora personal sea sustituida por un sistema empotrado. Este proyecto monográfico ha sido meramente experimental y didáctico, es un acercamiento inicial a la lógica difusa y a los controladores difusos. Utilizar una computadora personal para el control de un horno real es algo impráctico. Sin embargo, los fundamentos y conceptos utilizados en este trabajo sirven como guía para llevar a cabo este proyecto a manufactura.

Si se desea presentar otro trabajo monográfico basado en este sistema y, de igual forma, a manera de simulación, se recomienda aislar la estructura que simule al horno con fibra de vidrio o proplast, con el fin de que la temperatura ambiente no influya en el comportamiento del horno simulado. Adicionalmente, para mejorar la respuesta del controlador cuando la temperatura esté por encima de la temperatura deseada, es necesario agregar un extractor de aire, con el fin de que, éste ayude al controlador a deshacerse del calor excesivo más rápidamente, mejorando de esa manera su respuesta en el tiempo para bajar a la temperatura deseada.

Si en una aplicación real se necesita mayor exactitud de medición y control a la que posee el sistema de este trabajo monográfico, es necesario que se hagan nuevas pruebas, cambiando la cantidad de términos lingüísticos, las reglas difusas, los rangos de las variables difusas y cualquier otra cosa que el o los diseñadores estimen conveniente.

7. BIBLIOGRAFÍA

Bibliografía

Belloso, C. (2009). *Monografía sobre metodología de desarrollo de software, Rational Unified Process (RUP).* Tesis monográfica, Universidad Don Bosco.

Bruegge, B. (2002). *Ingeniería de Software Orientado a Objetos.* México: Pearson Education.

Chen, G., & Pham, T. T. (2001). *Introduction to Fuzzy Sets, Fuzzy Logic and Fuzzy Control Systems.* Houston, Texas, USA: CRC Press LLC.

Jimenez, R. (2007). *Diseño de un controlador difuso, aplicado al control de posición de un servomotor de C.D., usando un algoritmo genético.* Tésis de maestría, Universidad Veracruzana, Instituto de Ingeniería, Veracruz.

Maida, E., & Pacienza, J. (2015). *Metodologías de desarrollo de software.* Tesis de licenciatura, Universidad Católica, Buenos Aires, Argentina.

Moreno, M. (2010). *Filosofía Lean aplicada a la Ingeniería del Software.* Trabajo de fin de máster, Escuela técnica superior de Ingenieros, Sevilla, España.

Passino, K., & Yurkovich, S. (1998). *Fuzzy Control.* Addison Wesley Longman, Inc.

Pérez, O. (Junio de 2011). Cuatro enfoques metodológicos para el desarrollo de Software RUP – MSF – XP - SCRUM. *Inventum*(10), 64-78.

Ponce, P. (2010). *Inteligencia Artifical con Aplicaciones a la Ingeniería* (Primera edición ed.). México: Alfaomega Grupo Editor, S.A. de C.V.

Pressman, R. (2005). *Ingeniería del Software. Un enfoque práctico* (sexta ed.). México, México: McGraw-Hill.

Pressman, R. (2005). *Ingeniería del Software. Un enfoque práctico.* (Sexta ed.). México, México: McGraw-Hill.

Reina, D. (2008). *Fundamentos de mateática difusa.* Proyecto de investigación, Fundación Universitaria Konrad Lorenz , Facultad de Matemáticas.

Sommerville, I. (2005). *Ingeniería del Software* (Séptima ed.). Madrid, España: Pearson Addison Wesley.

Vélez, D. (2000). *Diseño de un controlador basado en lógica difusa para la regulación de flujo neutrónico.* Tésis de maestría, Instituto Tecnológico de Toluca, Departamento de sistemas y computación, Metepec, México.

Vicent, I. (2014). *Conjuntos difusos: aplicación al control de procesos.* Proyecto final de grado, Universitat Jaume I, Castellón de la Plana.

ANEXOS

Un poco de electrónica básica

La electrónica está fundamentada en un tipo de material llamado *Semiconductor*. El semiconductor es un material que tiene, en ciertas condiciones, propiedades de conductor, y en otras condiciones, propiedades de aislante. Hay dos tipos de materiales semiconductor, el material de tipo N y el material de tipo P.

El material de tipo N es un material semiconductor puro que ha sido dopado - mezclado - con otro tipo de material, el cual le permite aumentar su número de electrones libres. Por otro lado, el material de tipo P es un material semiconductor puro que ha sido dopado con otro tipo de material, el cual hace que el material semiconductor resultante quede con deficiencia de electrones; a cada deficiencia de electrón se le llama hueco, ya que, el electrón que se mueve a otra posición deja un espacio que otro electrón puede ocupar. Por tanto, los materiales de tipo N tienen exceso de electrones y los materiales de tipo P tienen exceso de huecos.

El diodo

El diodo es el dispositivo semiconductor básico. Este dispositivo está compuesto de la unión de un material de tipo P y un material de tipo N. A esta unión se le conoce como Unión PN. La siguiente figura muestra la unión PN y el símbolo del diodo.

Figura A-1. Diodo, a) unión PN, b) símbolo del diodo.

En la figura A-1 a, podemos observar la unión PN formando un diodo. La figura A-1 b, muestra el símbolo del diodo. Observe la polaridad del diodo; la parte positiva **A** es conocida como *ánodo*, la parte negativa **K** es conocida como *cátodo*.

El diodo puede ser polarizado de dos maneras, en polarización directa y en polarización inversa. En polarización directa el diodo deja pasar el flujo de electrones; mientras que, en polarización inversa bloquea el flujo de electrones. En consecuencia, el diodo solo deja pasar el flujo de electrones en una sola dirección, esto solo se presenta cuando está en polarización directa.

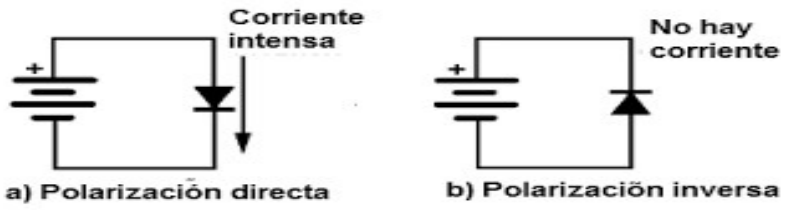


Figura A-2. Polarización del diodo.

Existen varios tipos de diodos, los cuales son utilizados para diferentes aplicaciones. Sin embargo, el que nos interesa en este momento es el diodo rectificador. Este tipo de diodo es utilizado para convertir una señal AC (alterna) a una señal DC (directa).

Circuito rectificador de onda completa

Un circuito rectificador de onda completa está compuesto de cuatro diodos rectificadores, los cuales, reciben una señal AC de entrada, a través de un transformador o de manera directa, y en su salida entregan una señal DC a una carga.

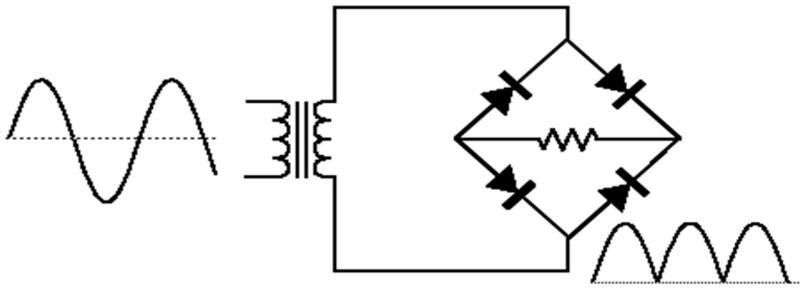
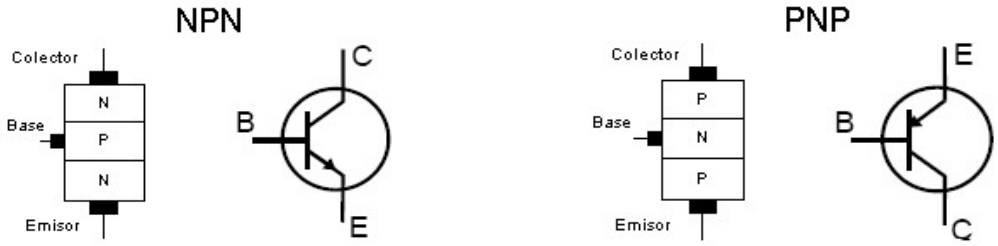


Figura A-3. Circuito rectificador de onda completa.

La figura A-3 muestra un circuito rectificador de onda completa que, recibe en su entrada una señal AC por medio de un transformador, y entrega una señal DC en una carga resistiva.

El transistor de unión bipolar (BJT)

El transistor es un dispositivo electrónico compuesto de tres capas de semiconductor dopado. Hay dos tipos de transistores de unión bipolar, uno es el tipo NPN y el otro es el tipo PNP. El transistor de tipo NPN está compuesto de dos capas semiconductoras de tipo N, separadas por una capa semiconductora de tipo P. En el transistor de tipo PNP, hay dos capas de semiconductor de tipo P, separadas por una capa de semiconductor de tipo N.



a. **b.**

Figura A-4. Transistor bipolar. a) Transistor de tipo NPN, b) Transistor de tipo PNP.

En la figura A-4 a, se puede observar en la parte izquierda las capas N y P por las que está compuesto un transistor bipolar NPN, y a la derecha el símbolo eléctrico con el que se representa en un diagrama eléctrico. En la figura A-4 b, se presenta a la izquierda las capas P y N por las que está compuesto un transistor bipolar PNP, y a la derecha el símbolo eléctrico. Nótese que, los transistores bipolares tienen tres terminales etiquetadas como Colector (C), Base (b) y Emisor (E). La flecha en el terminal Emisor en ambos símbolos eléctricos indica la dirección del flujo de electrones cuando el transistor está en conducción.

La terminal Base es la terminal de control, en ésta se recibe una señal de entrada, la cual controla la señal de salida. El transistor de unión bipolar se utiliza de dos maneras, una es como amplificador de señal, y la otra como interruptor. Esta última forma de

uso es la que nos interesa para comprender los circuitos que se explican posteriormente.

Para que un transistor de unión bipolar funcione como interruptor, se debe aplicar una tensión DC en la terminal Base del dispositivo. En el caso de que, éste sea NPN, la tensión de entrada debe ser positiva; por otro lado, si el transistor es PNP, la tensión de entrada debe ser negativa. Al aplicar la tensión en la Base, el transistor pasa a conducir electrones desde la terminal Colector a la terminal Emisor. A este estado de conducción se le conoce como estado de Saturación. Si se deja de aplicar tensión a la Base, o se aplica una tensión negativa, para el NPN, o una tensión positiva para el PNP, el transistor deja de conducir, no deja pasar el flujo de electrones. A este estado de no conducción se le llama estado de Corte. En consecuencia, si un transistor de unión bipolar es utilizado como interruptor, cuando conduce está en estado de saturación, y cuando no conduce está en estado de corte.

Circuito detector de cruce por cero

El circuito detector de cruce por cero se utiliza para detectar cuando una señal de AC pasa por el valor de 0V. Una señal AC tiene varios niveles de voltajes, comenzando desde cero, sube poco a poco hasta su valor máximo positivo, para luego poco a poco descender, de nuevo, a su valor cero; en seguida, desciende, también poco a poco, a su valor máximo negativo, y posteriormente sube a su valor cero otra vez poco a poco, para después continuar con otro ciclo. El cruce por cero de una señal, nos indica cuándo esa señal cambia de polaridad, de positivo a negativo, y de negativo a positivo; eso nos permite controlar la potencia que esa señal puede entregar a una carga.

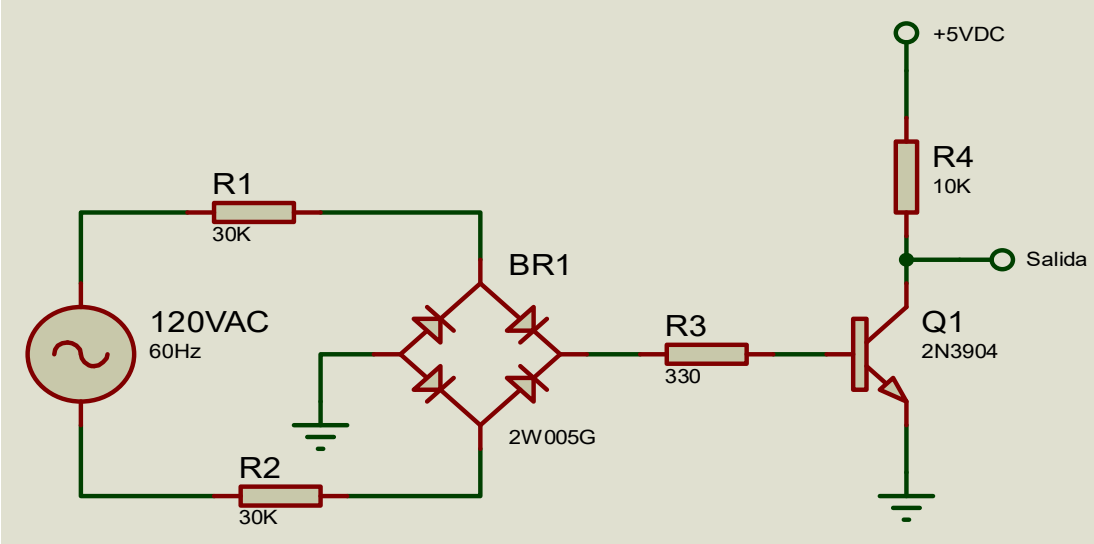


Figura A-5. Circuito detector de cruce por cero.

En la figura A-5, se muestra un circuito de cruce por cero. Nótese que, este circuito está compuesto de un rectificador de onda completa que le pasa la señal a un transistor. El funcionamiento es el siguiente. Inicialmente, el transistor está en estado de corte (interruptor abierto), por tanto, la salida será 5V. Cuando el rectificador de onda completa comienza a recibir los semiciclos de la señal AC de la fuente de 120AC, éste los rectifica y los pasa a la base del transistor a través del resistor R3, el cual limita la corriente que llega a la base. Cuando la señal llega a la base, hace que el transistor entre en estado de saturación (interruptor cerrado), haciendo que la salida sea de 0V. Ahora que el transistor está recibiendo señal en su base, la mayor parte del tiempo estará en conducción, haciendo que la salida esté la mayor parte del tiempo en 0V. Sin embargo, cuando la señal rectificada caiga en el valor de 0V, la base del transistor no recibirá señal alguna, haciendo que la salida tenga un nivel de voltaje de 5V en ese instante.

El triac

El triac es un dispositivo semiconductor que se utiliza como interruptor bidireccional. Esto significa que, este dispositivo puede manejar corriente en ambas direcciones, por lo tanto, se utiliza para el control de potencia de cargas de corriente alterna. En la figura A-6 se observa el símbolo del triac.

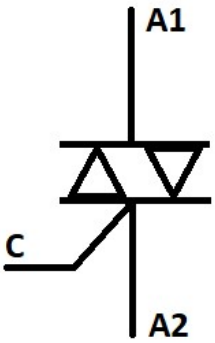


Figura A-6. Símbolo eléctrico del triac.

A las terminales A1 y A2 se les llama Ánodos, y a la terminal C se le llama Compuerta. Igual que el transistor, el triac funciona como interruptor; sin embargo, el triac puede manejar cargas de corriente alterna; mientras que, el transistor solo puede manejar cargas de corriente continua.

Para hacer que conduzca el triac, en la compuerta debe de haber una tensión mayor o igual al valor de tensión mínimo de disparo del triac. Si la tensión en la compuerta es menor que esa tensión mínima, entonces el triac no podrá conducir, quedará abierto. Cuando el triac es disparado (conduce), la corriente fluye de los terminales A1 a A2 y viceversa.

Circuito dimmer controlado por Arduino

Un circuito dimmer se utiliza para controlar la potencia que se le suministra a una carga resistiva, en corriente alterna. Este circuito está compuesto por un triac y otros componentes adicionales para realizar el disparo de la compuerta, y entregar la potencia deseada a la carga.

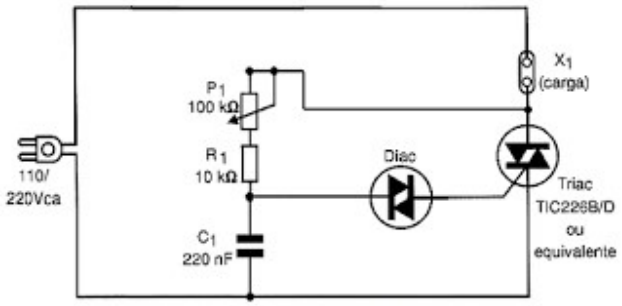


Figura A-7. Circuito dimmer.

En la figura A-7, el disparo de la compuerta del triac, se realiza por medio del potenciómetro P1, el resistor R1, el capacitor C1 y el diac. Estos componentes realizan un retardo del disparo del triac con respecto a la señal AC, haciendo que, cuando el triac sea disparado, en la carga solo llegue una parte de la señal AC; es decir, la señal no llega completa, en consecuencia, la potencia de salida en la carga X1 será menor que la potencia máxima que puede entregar la fuente 110/220Vca.

Por otra parte, para poder realizar el control de la potencia con una tarjeta Arduino, es necesario cambiar el circuito de disparo del triac por otro componente que es controlado por la tarjeta Arduino; y adicionalmente, a ese circuito se le debe agregar el circuito de cruce por cero, para que el Arduino sepa cuando la señal AC pasa por cero, y tome sus debidas decisiones para realizar el disparo en el momento correcto.

En la figura A-8, se muestra el circuito que se utilizó para el control del dispositivo calentador de este proyecto. Obsérvese que, para disparar el triac se ha utilizado un dispositivo optoacoplador entre la compuerta del triac y el Arduino. Igualmente, se utilizó un dispositivo optoacoplador entre el circuito detector de cruce por cero y el Arduino. Este opto acoplamiento se realiza con el fin de aislar a la tarjeta Arduino de las señales de la tarjeta del dimmer, con el objetivo de evitar que, por alguna razón, haya un fallo eléctrico en el dimmer que vaya a dañar a la tarjeta Arduino. El dispositivo 4N25 es un opto transistor que envía un pulso de 5V a un pin de entrada digital del Arduino, cuando la señal AC hace el cruce por cero. El circuito detector de cruce por cero detecta el cruce, haciendo que el transistor entre en estado de saturación, activando al opto transistor del 4N25.

El triac se activa una vez que el pin de salida digital del Arduino envía un nivel de voltaje de 5V a la entrada del optotriac MOC3011. Ese voltaje activa el triac interno del MOC3011, el cual le pasa la señal de disparo al triac que maneja a la carga.

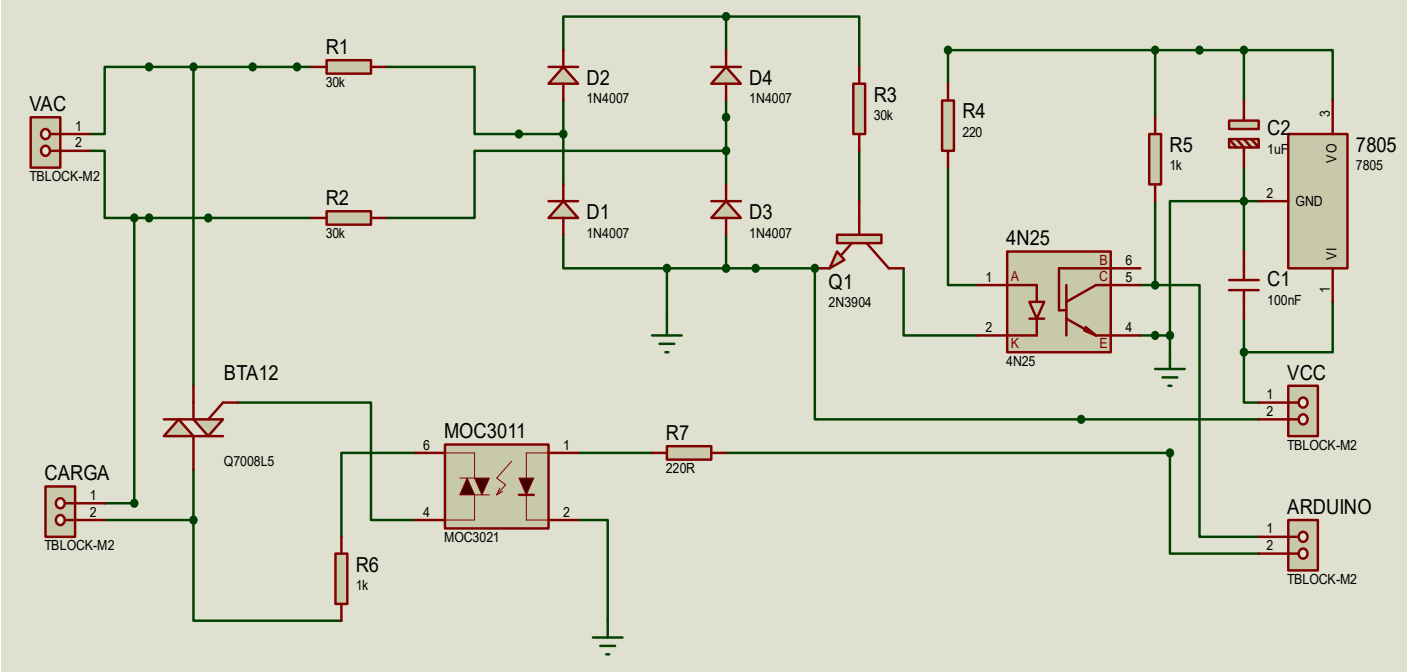


Figura A-8. Circuito de control del dispositivo calentador controlado por Arduino.

Lógica de funcionamiento del programa en Arduino

Antes de todo, es necesario que se comprenda el concepto de interrupción. Una interrupción es un paro que hace el Arduino o cualquier dispositivo controlador programable, en la ejecución del código, para ejecutar otra parte del código, llamada rutina de interrupción, que tiene mayor prioridad de ejecución. En este caso, cuando hay una llamada de interrupción, el Arduino deja de ejecutar la línea de código actual, la deja en espera, y se dedica a ejecutar toda la rutina de interrupción. Cuando finaliza de ejecutarla, regresa a ejecutar a la línea que dejó en espera y sigue el flujo del programa normal.

En el Arduino existen las interrupciones internas y externas. Las interrupciones internas son eventos que ocurren en lo interno del Arduino; ejemplo, un temporizador. Las interrupciones externas son eventos que ocurren afuera del Arduino, no dependen de él; ejemplo, pulsar un botón.

En el código Arduino que se utiliza en este proyecto para controlar la potencia del bombillo, se hace uso de ambos tipos de interrupción. Primero porque utilizamos un

temporizador para contar el tiempo de los semiciclos de la señal AC, enlazándolo con una interrupción interna; y segundo, porque se utiliza el pin digital de entrada enlazado a una interrupción externa, la cual es llamada cuando se detecte la señal del opto transistor 4N25, al hacer la señal AC un cruce por cero.

La lógica del programa es la siguiente:

1. El Arduino comienza a hacer la medición de temperatura a través de tres pines analógicos de entrada conectados a tres sensores LM35.
2. Esos valores obtenidos son niveles de voltaje, por tanto, el Arduino tiene que acondicionar esos valores para pasarlos a valores de temperatura.
3. Luego, con esos tres valores se hace un promedio, con el fin de obtener un único valor de temperatura.
4. Posteriormente, se hace otro acondicionamiento de ese valor para pasarlo a un rango de temperatura real (Recuerde que, estamos simulando valores mayores a 140°C).
5. Seguidamente, ese valor se envía al controlador difuso que está ejecutándose en el pc, esto se hace por medio del puerto serie USB.
6. El Arduino espera la respuesta del controlador difuso, la cual es el dato de potencia de salida que se debe aplicar al bombillo calentador.
7. Una vez obtenido el dato, el Arduino calcula el retardo de tiempo que debe esperar para poder activar el triac.
8. Finalmente, después de haber ejecutado los siete pasos anteriores, espera 100 milisegundos para volver al paso 1 y hacer otro ciclo de ejecución.

Esos ocho pasos realizan el Arduino continuamente cuando no se ha ejecutado la interrupción de cruce por cero, o la interrupción del temporizador. Al haber un cruce por cero, el pin de entrada digital lo detecta, hace que el Arduino deje de ejecutar la línea actual, lo envía a la rutina de interrupción de cruce por cero, para que la ejecute totalmente, y luego siga con lo demás que estaba ejecutando. Igual sucede con la rutina de interrupción del temporizador, si el tiempo total del temporizador se alcanza, esta manda a parar al Arduino, y le indica que ejecute la rutina de interrupción del temporizador.

La rutina de interrupción del cruce por cero manda apagar el triac, poniendo a cero el pin digital de salida, luego activa la bandera de cruce por cero, y reinicia el contador del tiempo de retardo.

La rutina de interrupción del temporizador verifica si la bandera de cruce por cero está activada. Si esa bandera no está activada, entonces el Arduino se sale de la rutina de interrupción del temporizador. Pero si está activada, lo primero que hace es, verificar si aún no se ha alcanzado el tiempo de retardo. Si esa condición se cumple, aumenta el contador de tiempo de retardo en una unidad. Una unidad del contador del tiempo de retardo equivale a 83us, el cual es el 1% del total del tiempo de un semiciclo tanto positivo o negativo de la señal AC. Si el tiempo de retardo de la señal AC se ha alcanzado o se ha superado, el Arduino pone a 5V el pin de salida digital, encendiendo así al triac, y pasándole al bombillo la potencia necesaria; y luego, desactiva la bandera de cruce por cero, para que esté lista para la próxima interrupción del temporizador.