Universidad Autónoma de Querétaro
Facultad de Ingeniería
Maestría en Ciencias (Instrumentación y Control)

Estrategias de control automático para la optimización del consumo energético en un deshidratador de frutas y hortalizas.

Opción de titulación
Tesis

Que como parte de los requisitos para obtener el Grado de
Maestría en Ciencias (Instrumentación y Control)

Presenta:
Ing. Juan Carlos Tapia Cisneros

Dirigido por:
Dr. Genaro Martin Soto Zarazúa

Dr. Genaro Martin Soto Zarazúa
Presidente — Firma

Dr. Manuel Toledano Ayala
Secretario — Firma

Dr. Juan Fernando García Trejo
Vocal — Firma

Dr. Gonzalo Macías Bobadilla
Suplente — Firma

Dr. Edgar Alejandro Rivas Araiza
Suplente — Firma

Nombre y Firma
Director de la Facultad

Dra. Ma. Guadalupe Flavia Loarca Piña
Director de Investigación y Posgrado

Centro Universitario
Querétaro, Qro.
Junio 2015

Resumen

La presente tesis documenta la instrumentación y las estrategias de control implementadas en un deshidratador de frutas y hortalizas. El objetivo principal de la tesis consistió en determinar la estrategia de control que permite alcanzar las condiciones de deshidratado al menor costo energético, por lo que se planteó el comparar un controlador difuso y un control neuronal. Ambos controles fueron implementados en Python para ser ejecutados en un sistema embebido con sistema operativo basado en el kernel de Linux el cual realiza las acciones de sincronización con las terminales remotas con capacidad de monitoreo y control. El sistema cuenta con una interfaz web para consultar el estado actual y pasado del sistema así como para modificar los parámetros de control.

(**Palabras clave**: Control, redes neuronales, control difuso, sistemas embebidos)

Summary

This thesis documents the instrumentation and control strategies implemented in dehydrator of fruits and vegetables. The main aim of the thesis was determine the better control strategy that achieves the dehydrated conditions to lower energy cost, so that was raised comparing a fuzzy controller and neural control. Both controls were implemented in Python to run on an embedded device with operating system based on Linux kernel that performs the actions of sync with the remote terminal capable of monitoring and control. The system has a web interface to query the current state.

(**key words**: Control, neural networks, fuzzy logic, embedded systems)

Dedicatoria:

A Dios por la oportunidad.

A mis padres por su constante apoyo.

A mis hermanas por su comprensión.

A mis amigos por sus consejos.

A mis profesores por los conocimientos compartidos.

Agradecimientos

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACyT) y a la Universidad Autónoma de Querétaro por su apoyo y patrocinio para la realización de este proyecto.

Tabla de contenidos

Índice de figuras

Índice de tablas

1. Introducción

El proceso de deshidratado es esencial para la preservación de los productos en la agricultura, consiste en remover la humedad del producto con la finalidad de reducir la actividad microbiana y así lograr preservarlos productos sin necesidad de métodos de refrigeración. En el caso de la deshidratación de productos alimenticios como las frutas y los vegetales se recomiendan realizar el deshidratado manteniendo una temperatura constante en un rango de 45 – 60 grados centígrados para un deshidratado seguro, aumentar la temperatura reduce el tiempo empleado para la deshidratación pero compromete la integridad de sus nutrientes. El deshidratado bajo condiciones controladas de temperatura y humedad permiten disminuir el contenido de humedad de forma razonablemente rápida sin comprometer la calidad de los productos (Sharma, 1994), es por lo anterior que surge la necesidad de implementar alguna estrategia de control para controlar las condiciones climatológicas en las que se desarrolla la deshidratación.

El uso de sistemas forzados de deshidratación mejoran significativamente la calidad de los productos esto debido a que el tiempo de deshidratación se reduce significativamente, y con ello se reduce el tiempo de exposición al medio ambiente al mismo tiempo que se reduce la actividad microbiana. Lo que se traduce en un aumento de la calidad de los productos.

En el mercado existen diversos deshidratadores caseros y de uso industrial que emplean diferentes fuentes de energía como es el caso de los deshidratadores solares (empleados en deshidratadores caseros), deshidratadores eléctricos (empleado en industria química) entre otros. Los sistemas híbridos que involucran resistencias eléctricas y colectores solares son la solución ideal para aplicaciones industriales en las que se pretenda optimizar el consumo energético.

La presente tesis trata de la instrumentación y control de un deshidratador hibrido (solar eléctrico) en el que se buscó mediante la aplicación de algoritmos de control avanzado la optimización del consumo energético, el documento incluye

detalles en relación a la instrumentación (sensores y actuadores), manejo de información (base de datos, sistema embebido) así como los resultados de pruebas para el cálculo del consumo energético.

1.1 Antecedentes

Durante la etapa de pos-cosecha el deshidratado es esencial para la preservación de los productos agrícolas. El deshidratado bajo condiciones controladas de temperatura y humedad permiten disminuir el contenido de humedad de forma razonablemente rápida sin comprometer la calidad de los productos (Sharma, 1994). Los productos alimenticios, especialmente las frutas y vegetales pueden ser deshidratados mediante distintos métodos, como son: convección, conducción, radiación, dieléctrico o una combinación de métodos. Sin embargo el método que se emplea con mayor frecuencia en la industria es el deshidratado por convección, el cual, esencialmente consiste en mantener un flujo de aire caliente sobre el alimento a deshidratar. El aire caliente debe estar en un rango de 45 – 60 grados centígrados para un deshidratado que no altere nutrientes.

Los estudios que se mencionan a continuación se encuentran estrechamente relacionados con nuestro tema de estudio, abarcan temas como son simulación, turbulencia, eficiencia térmica entre otros, se exponen brevemente para sentar las bases de nuestra tesis:

- El interés en sistemas de control de flujo en lazo cerrado está aumentando en los últimos años. Muchos de los estudios son basados en simulaciones (Becker, 2005). Los estudios en simulación representan una base para establecer límites operativos de un sistema de control robusto, es por ello que se requiere conocer tanto los resultados de simulaciones como los resultados experimentales de sistemas similares a los deshidratadores.

- Los estudios basados en simulación permiten establecer modelos que describen el comportamiento de los deshidratadores. Curcio (2008), propuso un modelo que no depende de la resistencia térmica, con ello obtiene una herramienta general capaz de describir el comportamiento del deshidratador sobre una gran cantidad de rangos de operación y condiciones generadas por la dinámica de fluidos. Estudios experimentales realizados por Curcio muestran una buena relación entre las predicciones de su modelo y los resultados experimentales, con ello Curcio valida su modelo matemático.
- Uno de los aspectos que son ampliamente investigados debido a su importancia en los procesos de deshidratado son los relacionados con el rendimiento termodinámico. Bhushan (2012) realizó estudios concernientes a la eficiencia térmica en calentadores solares de aire, relacionó la geometría y la rugosidad para obtener las características geométricas que maximiza la eficiencia de sistemas térmicos basados en aire.
- Las turbulencias generadas por el flujo de aire son un factor que debe ser suprimido, las oscilaciones causadas por el flujo sobre cavidades han generado un interés especial (Cattafesta, 2008). Un correcto estudio de las posibles turbulencias generadas por la geometría de los actuadores, sensores y otros componentes del sistema físico aumenta la eficiencia y calidad del flujo, que finalmente impacta en la calidad de los productos.
- Para mejorar el desempeño termodinámico y disminuir las turbulencias, Inaoka (2004) propone el uso de actuadores electrónicos miniatura dentro del ducto para mantener una frecuencia determinada y así contrarrestar la turbulencia generada por las cavidades o los bordes presentados dentro de los ductos del aire.
- Naghib (2010) propone el uso de una serie de puertos de inyección distribuidos en el cuerpo principal de ducto para contrarrestar los vórtices generados en el flujo. Simulaciones numéricas demostraron una atenuación en las fluctuaciones aerodinámicas.

- Durante el proceso de deshidratado se presentan cambios en el color y la textura del producto deshidratado, esto debido a la temperatura a la que se realiza la deshidratación. Nagaya (2005) propone un deshidratador a baja temperatura que permite obtener productos deshidratados con daños mínimos y sin afectar la velocidad de deshidratado. Mediante experimentación Nagaya demostró que sus productos deshidratados conservaron su color fresco y un alto contenido vitamínico.

1.2 Justificación

Los sistemas de control que actualmente son empleados por los deshidratadores industriales son basados en termostatos o en sistemas digitales tradicionales (PID), en cualquier caso el sistema de control mantiene las condiciones necesarias pero no se presta atención al consumo energético que generan los diferentes esfuerzos de control. Es por esto que se requiere el diseño de un sistema de control automatizado basado en estrategias inteligentes que aproveche las condiciones externas para minimizar los requerimientos energéticos del proceso de deshidratado.

Es común encontrar deshidratadores híbridos que unen lo mejor de las tecnologías disponibles para calentar el aire que posteriormente será utilizado durante la convección, la combinación más usual con los deshidratadores solares en la unión con resistencias eléctricas, este tipo de dispositivo permite utilizar la energía solar durante el día y las resistencias eléctricas en horas en las que la radiación no es suficiente. En esta configuración y con el apoyo de un algoritmo de control que le permita manipular los distintos mecanismos para recirculación de aire y de circulación de agua se obtiene un sistema eficiente el cual responde a las variaciones climatológicas y con ello se reduce el consumo energético.

1.3 Planteamiento del problema

Un calentador eléctrico con ventilador axial de 45cm consume 15kwatts para mantener 80 grados centígrados con un flujo a 2m/s, un panel solar genera 150W/m2, lo que implica una superficie de 100m2 para generar la potencia eléctrica requerida por el calentador. Es necesaria una estrategia para optimizar el consumo de energía considerando la etapa de potencia y el sistema de control, el cual debe contar con las funciones de monitoreo, registro y control manteniendo un bajo consumo energético.

Los sistemas de control clásicos, como son los PIDs resultan eficientes para la mayoría de los procesos en los que se mantienen las condiciones de operación constantes, para un sistema portátil, las condiciones climatológicas pueden variar complicando la obtención de uniformidad en la calidad del producto (Kosuke et al. 2005), además, las condiciones de deshidratado requeridas dependen del fruto que se quiera deshidratar. Es por esto que se requiere un sistema de control que se adapte a las condiciones ambientales y que pueda operar ante múltiples entradas.

Las operaciones relacionadas con el monitoreo y control, requieren un sistema capaz de atender simultáneamente las peticiones para almacenar y consultar la información generada por el sistema, debido a la gran cantidad de información generada por los sensores y requerida por los actuadores, es necesario un protocolo de comunicación robusto que prevenga colisiones y fallas. Además, es necesaria una interfaz de comunicación con el sistema que deberá seguir la tendencia hacia los dispositivos táctiles y conexiones remotas.

1.4 Hipótesis y objetivos

1.4.1 Hipótesis

Un sistema de control automático para un deshidratador solar-eléctrico basado en redes neuronales permite obtener una mayor eficiencia energética en comparación con un sistema de control basado en lógica difusa.

1.4.2 Objetivo general

Instrumentar y diseñar un sistema automático de control y monitoreo basado en FPGA con soft-procesador Microblaze y SO uClinux que permita comparar el consumo energético requerido por un control neuronal y un controlador difuso implementado en un deshidratador solar-eléctrico.

1.4.3 Objetivos particulares

- Diseñar e integrar un sistema automático para la adquisición y control del proceso de deshidratado implementando el kernel de linux.
- Codificar e implementar los sistemas de control por redes neuronales y por lógica difusa.

2. Marco teórico

El sistema empleado durante la tesis consta de diversos subsistemas que involucran diferentes temas, cada uno de los temas será expuesto a continuación y su aplicación será detallada en la sección de resultados en la que se expondrá el método en que fue aplicado cada uno de los temas que mencionaremos.

2.1 Proceso de deshidratación

El termino deshidratación se refiere a remover la humedad de un material con el objetivo primario de reducir la actividad microbiana y la degradación (Ratti, 2001). Durante la deshidratación se producen dos procesos de forma simultánea:

- Trasferencia de la energía del medio circundante para evaporar la humedad superficial.
- Trasferencia de la humedad interna a la superficie del sólido para la subsecuente evaporación debido al primer proceso.

La transferencia de energía del medio al sólido puede ocurrir como resultado de la convección, conducción o radiación y en algunos casos como resultado de la combinación de estos efectos (Mujumdar, 2006).

La Tabla 2.1 expone el principio de funcionamiento de cada método así como notas de su aplicación.

Tabla 2.1 Funcionamiento por método

Método	Descripción	Notas
Calentamiento por convección	El calor es suministrado mediante aire caliente que es soplado sobre la superficie del sólido.	Común en el deshidratado de partículas y pastas sólidas. Son llamados secadores directos
Calentamiento por conducción	El calor es suministrado por superficies calientes que contienen o trasportan el sólido. La humedad evaporada es trasportada utilizando corrientes de gas.	Empleados en procesos que involucran productos finos o muy húmedos. Mayor eficiencia en comparación a los deshidratadores directos.
Calentamiento por radiación	Permiten el calentamiento desde el interior del sólido, reduciendo su resistencia a la trasferencia térmica	Bajo consumo de energía para el secado. Alto consto de operación (Mujumdar, 2006).

2.1.1 Generalidades del proceso de deshidratación

Las propiedades de la deshidratación se conocen desde la antigüedad en la que se empleaba el método natural para deshidratar alimentos y así poder consumirlos fuera de temporada o durante sequias, dicho método consiste en exponer el alimento a los rayos solares, este proceso genera alimentos de baja calidad debido a que no se controla la temperatura y el tiempo empleado en el proceso favorece la descomposición por actividad microbiana así como la contaminación del alimento. En la actualidad los procesos de deshidratación forzados reducen el tiempo requerido para la deshidratación de días a horas, y con ello se mejora la calidad del producto.

Los métodos para realizar la deshidratación de alimentos fueron perfeccionados durante el período de las guerras mundiales, en donde se requería el manejo de grandes cantidades de alimento pero debido a cuestiones de logística los alimentos debían ser de poco volumen y gran contenido nutritivo, por lo que el proceso de deshidratación para su futura rehidratación en el lugar de consumo resultó la solución ideal.

Durante el proceso de deshidratación se presentan cambios significativos en las propiedades físicas del alimento. Estas propiedades dependen de varios factores como son el pre tratamiento, el contenido de humedad, el método utilizado, y las condiciones de deshidratación. (Krokida et al., 2000).

La calidad de los productos alimenticios es realmente importante y lamentablemente el proceso de deshidratación afecta significativamente la calidad de los mismos. En la actualidad se trabaja en el desarrollo de procesos que permitan aumentar la calidad del producto conservando las ventajas del proceso de deshidratación como son el aumento de vida útil, disminución de volumen y peso que facilita el trasporte y manejo del producto. Algunas de las características que se pretende mejorar son: aroma, sabor, color, textura y el valor nutricional. Sin embargo, la calidad de la comida puede ser afectada por otros parámetros ajenos como son: el pH, la composición de la comida, los pre tratamientos y la presencia

de sal, aceites o solventes (Mujumar, 1997), lo que impide el desarrollo de un producto que satisfaga completamente las características de calidad.

2.1.2 Clasificación de los deshidratadores

Araya-Farias et al. en el 2009 proponen una clasificación en base a los siguientes criterios:

- Tipo de operación: Pasivo o continuo.
- Presión de operación: vacío, presión atmosférica o alta presión
- Modo de trasferencia de energía: conducción, convección, radiación, por radio frecuencia o una combinación de lo anterior.
- Estado del producto antes de la deshidratación: estacionario, en movimiento, agitado, fluido o atomizado
- Tiempo requerido: poco <1min, medio 1-60min, largo >60min.

Los deshidratadores directos pueden ser agrupados de acuerdo a la fuente de energía:

- Solar: Los deshidratadores soleares de uso industrial requieren de una zona de recolección de radiación solar en la que se calienta aire que es impulsado sobre el sólido a deshidratar.
- Eléctrico: Este tipo de deshidratadores utilizan resistencias eléctricas para calentar el aire que es impulsado sobre el sólido a deshidratar.
- De Gas: Los deshidratadores de gas emplean quemadores para calentar el aire que es impulsado sobre el sólido a deshidratar.

Una tercera clasificación puede ser vista en la Figura 2.1, la cual clasifica los deshidratadores en base al método de trasferencia de calor utilizado, la Figura 2.1 incluye ejemplos de deshidratadores que emplean cada una de los distintos métodos de calentamiento.

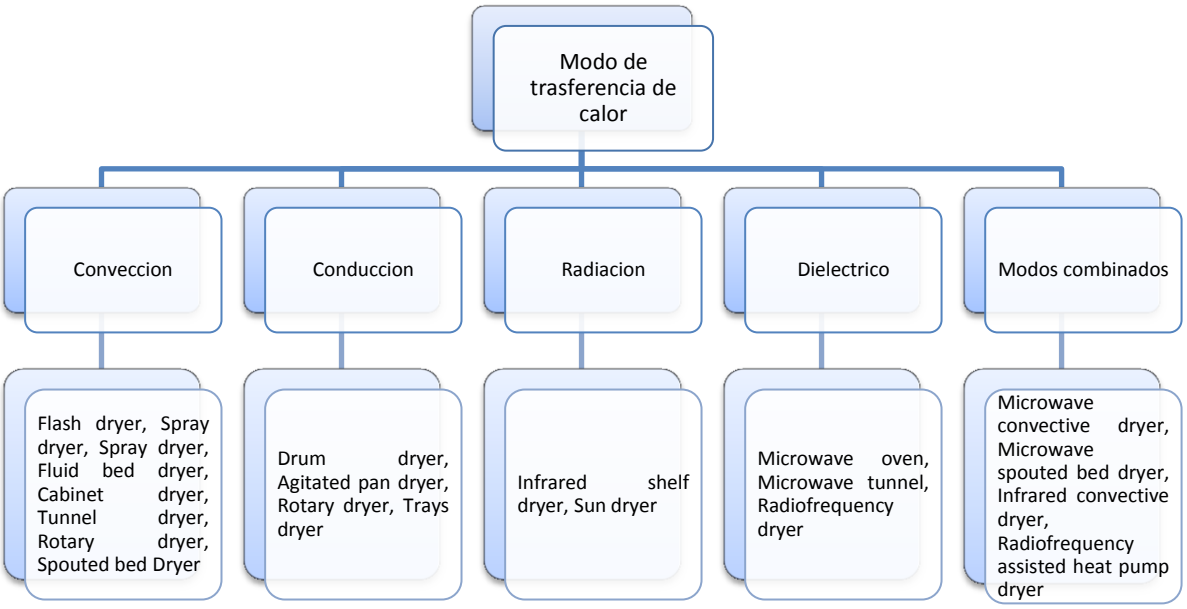

Figura 2.1 Calcificación de acuerdo al método de trasferencia de calor

2.1.3 Funcionamiento

El funcionamiento de los distintos tipos de deshidratadores varía únicamente en la fuente de calor empleada para su funcionamiento, a continuación se describe de forma general el deshidratador empleado durante la tesis.

2.1.3.1 Componentes

La Figura 2.2 muestra los componentes que forman el sistema hidráulico, eléctrico y de control. El deshidratador utiliza un calentador solar equipado con una resistencia eléctrica de 0.5kw para calentar el fluido que circula por un serpentín dentro de la cámara de deshidratado en una zona en que utilizando aire se genera el intercambio de calor. La energía eléctrica para alimentar el motor del soplador, la bomba de recirculación de agua, las resistencias eléctricas y el sistema de monitoreo se obtienen de la línea eléctrica comercial.

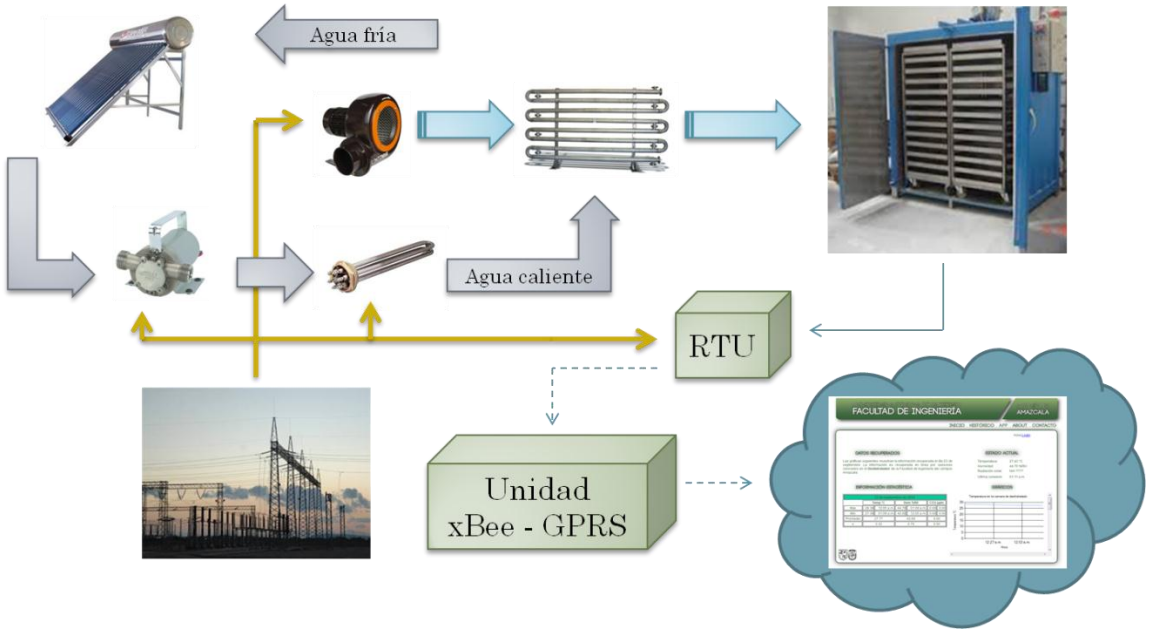

Figura 2.2 Componentes del deshidratador solar-eléctrico

2.1.3.2 Sistema Hidráulico

El sistema hidráulico está formado por un circuito cerrado en el que se calienta agua utilizando radiación solar durante el día y una resistencia eléctrica por la noche o en días nublados (Figura 2.3). El fluido es enviado por tubería de CPVC 1in desde el calentador (Figura 2.4) hacia el interior del edificio (Figura 2.5) en donde circula por un serpentín colocado dentro del deshidratador y finalmente utilizando una bomba se garantiza el retorno del fluido por el tubo desde el interior del edificio hasta el calentador solar para iniciar nuevamente el ciclo.

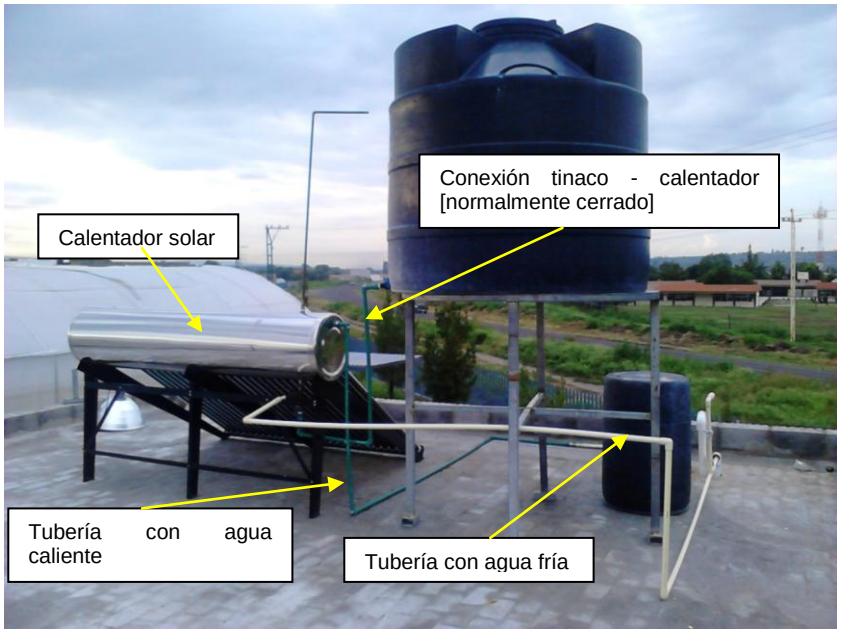

Figura 2.3 Contenedores de agua

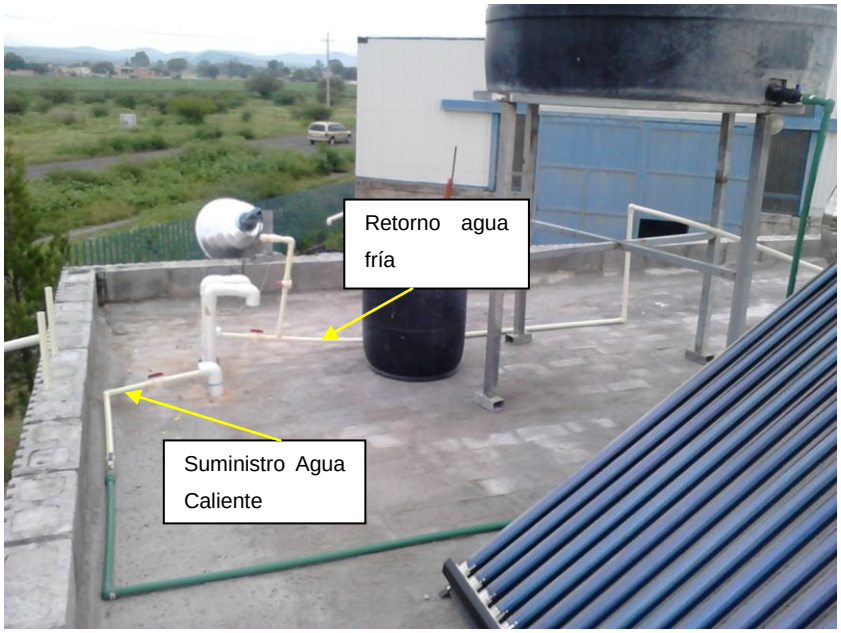

Figura 2.4 Detalle de la conexión en el exterior del edificio

Figura 2.5 Detalle de conexión en el interior del edificio

2.1.3.3 Sistema Eléctrico

El deshidratador se recibió con un motor monofásico 126VCA 60Hz 0.5HP, durante la instalación se colocó una bomba de recirculación de agua trifásica 220VCA 60Hz 1HP, el calentador de agua cuenta con una resistencia electica de 0.5kw. Las modificaciones que se realizaron serán presentadas en la sección de resultados.

2.1.3.4 Sistema de control y monitoreo

El sistema no fue suministrado con sistemas de monitoreo, el control es únicamente On/Off en todos sus componentes. Las modificaciones que se realizaron serán presentadas en la sección de resultados.

2.2 Estrategias de control

2.2.1 Control difuso

El concepto de logica difusa surgió en 1965, en la Universidad de California, Berkeley por Lofti A. Zadeh, quien combinó los conceptos de la lógica y de los conjuntos de Lukasiewicz mediante la definición de grados de pertenencia.

La Lógica Difusa ha cobrado una gran fama por la variedad de sus aplicaciones las cuales van desde el control de procesos industriales complejos hasta el diseño de dispositivos artificiales de deducción automática, pasando por la construcción de artefactos electrónicos de uso doméstico y de entretenimiento así como también de sistemas de diagnóstico (Montufar, 2012).

El control difuso es una herramienta que permite emular el razonamiento humano. Los seres humanos pensamos y razonamos por medio de palabras y en grados entre dos estados por ejemplo blanco y negro ó frío y caliente, etc. Estos sistemas de lógica difusa son una mejora a los sistemas experto tradicionales, en el sentido de que permiten utilizar lenguaje humano como nosotros razonamos.

2.2.1.1 Diseño de controladores difusos

El diseño del controlador difuso requiere la construcción de reglas de control, las cuales pueden ser obtenidas en base a las acciones del operador o en base a la respuesta del sistema en el formato Si-Entonces, de esta forma el procedimiento de diseño es el mostrado en la Figura 2.6.

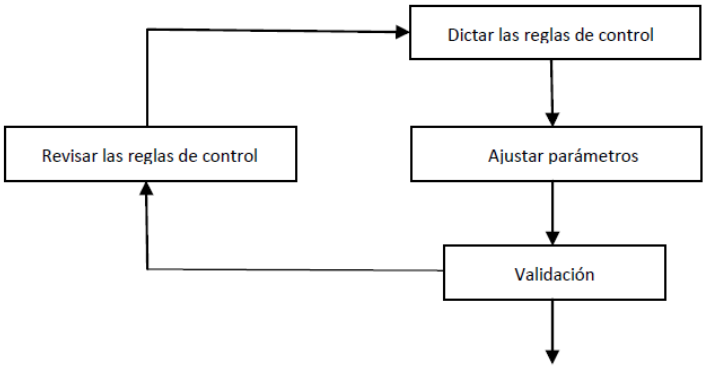

Figura 2.6 Procedimiento de diseño de controladores difusos (Noriega, 2014)

En general, para la mayoría de las aplicaciones de control, la información requerida para el análisis y diseño de los sistemas se encuentran divididas en dos conjuntos: la información numérica y la información lingüística obtenida de los operadores. Para combinar ambas fuentes en un conjunto de información que dé lugar a las reglas del control difuso se puede emplear el método propuesto por Wang y Mendel (1992) denominado FAM (Fuzzy Associative Memory).

2.2.1.2 Generación de reglas en base a datos numéricos

Para la mejor explicación del procedimiento se asumirá que se tiene un conjunto deseado de pares entrada-salida:

$$\left(x_1^{(1)}, x_2^{(1)}; y^1\right), \left(x_1^{(2)}, x_2^{(2)}; y^2\right), \ldots \qquad (1)$$

Donde x_1 y x_2 son entradas y y la salida. Del conjunto (1) se generaran las reglas y se mapearan las reglas para definir f: $(x_1, x_2) \rightarrow y$.

2.2.1.2.1 Paso 1: Dividir las entradas y salidas en regiones difusas

Asumiendo que el dominio de los intervalos para x_1, x_2 y y es $[\mathrm{x}_1^-, \mathrm{x}_1^+]$, $[\mathrm{x}_2^-, \mathrm{x}_2^+]$ y $[\mathrm{y}^-, \mathrm{y}^+]$ respectivamente, donde el dominio indica los valores mas probables en que la variable se encuentre. El dominio es dividido en $2N + 1$ regiones, en general las regiones se identifican con la siguiente nomenclatura: SN (Small N), …, S1 (Small 1), (CE (Center), B1 (Big 1), …, BN (Big N). Cada región es asignada a una función de pertenencia, la Figura 2.7 muestra un ejemplo para cinco regiones para x_1 y y y siete regiones para x_2, las funciones de pertenencia son triangulares con vértice en el centro de cada región.

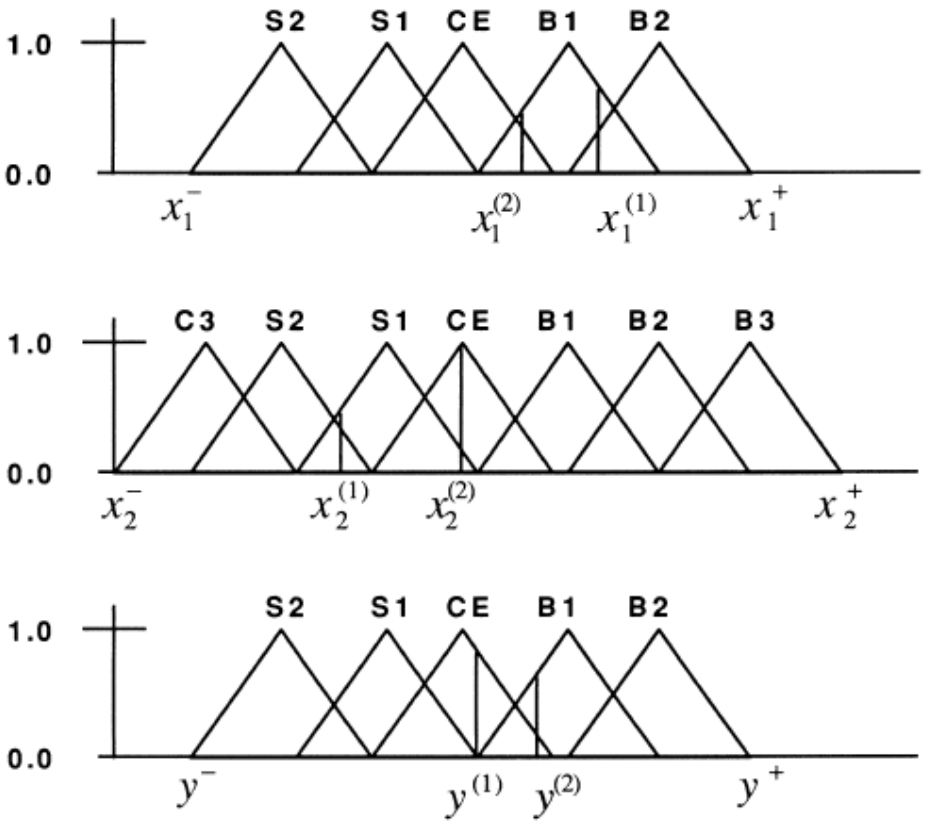

Figura 2.7 Divisiones de las entradas y salidas en regiones difusas y su correspondiente función de membrecía

2.2.1.2.2 Paso 2: Generar reglas en base a los pares dados

Es necesario determinar los grados de pertenecía de las entradas y las salidas que corresponden a cada región. Por ejemplo $x_1^{(1)}$ en la Figura 2.7 tiene un grado 0.8 en B1, 0.2 en B2 y cero en todas las otras regiones. De igual forma, $x_2^{(2)}$ en la Figura 2.7 tiene un grado 1 en CE, y cero en todas las otras regiones. El grado de pertenencia asignado es el máximo en caso de tener valores distintos de cero. Por ejemplo, $x_1^{(1)}$ es asignado en B1 y $x_2^{(2)}$ es asignado a CE. Finalmente, una regla se obtiene por un par de datos, ejemplo:

Regla 1: Si x_1 en B1 y x_2 en S1, Entonces y es CE

Regla 2: Si x_1 en B1 y x_2 en CE, Entonces y es B1

2.2.1.2.3 Paso 3: Asignación del grado para cada regla

Existe la posibilidad de tener dos reglas que entren en conflicto por lo que es necesario asignar un grado a cada regla y de esta forma conservar únicamente la que tiene mayor peso.

El grado asignado para cada regla es el producto de sus valores, ejemplo, la regla 1 tiene un valor de 0.504 ($0.8 * 0.7 * 0.9 = 0.504$) y la regla 2 tiene un valor de 0.42 ($0.6 * 1.0 * 0.7 = 0.42$). Un cuarto valor puede ser considerado durante el cálculo del grado, el cuarto valor corresponde al asignado por un experto el cual puede tener información a priori y considerar que alguna de las reglas puede ocasionar problemas por lo que asignaría un valor bajo ocasionando que la regla sea despreciada.

2.2.1.2.4 Paso 4: Creación de la base FAM

La forma de la base FAM es mostrada en la Figura 2.8, la figura representa una regla (Si x_1 en S1 o x_2 en CE, Entonces y es B2) en la que se emplea el operador “o” por lo que se llenan las casillas mencionadas, en caso de ser el operador “y” solo se llenaría la casilla de la intersección.

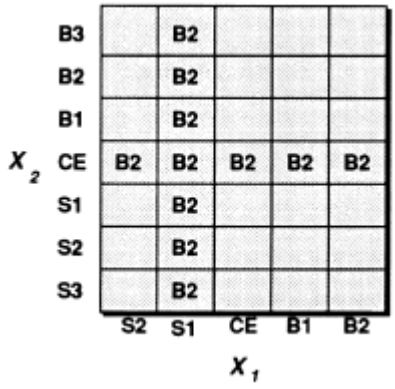

Figura 2.8 Representación grafica de una FAM

2.2.1.2.5 Paso 5: Determinación del mapeo en base al banco FAM

La siguiente estrategia determina la salida de control y para las entradas (x_1, x_2):

1. Todas las reglas difusas en el banco FAM son representadas por funciones de pertenencia de la forma mostrada en el paso 2.
2. Para un par de entradas, las reglas son combinadas usando productos para determinar el grado de la salida de control correspondiente

$$m_{Oi}^{i} = m_{I_1^i}(x_1) m_{I_2^i}(x_2) \qquad (2)$$

Ejemplo: La regla 1: $m_{CE}^{1} = m_{B1}(x_1) m_{S1}(x_2)$

3. La ecuación (defucificacion-centroide) (3) determina la salida de control

$$y_j = \frac{\sum_{i=0}^{K} m_{O_j^i}^{i} y_j^{-i}}{\sum_{i=1}^{K} m_{O^i}^{i}} \qquad (3)$$

Donde y_j^{-i} indica el centro de la región O_j^i.

Para problemas de control, las entradas son las variables que serán controladas y las salidas serán las señales de control que se envían a la planta. Este método esencialmente “aprende” de las “muestras” y de expertos para obtener el mapeo, suponiendo tener una muestra adecuada el resultado del procedimiento permitirá generar salidas esperadas y acertadas. El método puede ser considerado como un modelo de auto sintonización difusa debido a que no se requiere de un modelo matemático y que el sistema aprende de una serie de muestras (Quiroga, 1994).

2.2.2 Redes neuronales artificiales

Las redes neuronales artificiales son redes de elementos simples usualmente adaptativas y organizadas de forma paralela, que pretenden interactuar con los objetos del mundo real de la misma manera que los seres vivos. Las redes neuronales pueden ser vistas como algoritmos computacionales capaces de realizar tareas de clasificación de información, memoria asociativa y

comprensión de información. La programación de los algoritmos se realiza mediante el aprendizaje, es decir, a través del ensayo de un conjunto de ejemplos definidos, en base a los cuales el sistema es capaz de generalizar el conocimiento adquirido.

2.2.2.1 Arquitectura de las redes neuronales artificiales adaptativas

Una red neuronal (Figura 2.9) es una estructura de red cuyo comportamiento global de entrada/salida se determina por los valores de una colección de parámetros modificables. Más específicamente, la configuración de estas redes está compuesta de un conjunto de nodos interconectados por conexiones directas, donde cada nodo es una unidad de proceso que produce una única señal de salida para una entrada dada. Usualmente, cada nodo es una función de parámetros modificables; cambiando estos parámetros cambiamos el comportamiento general de la red.

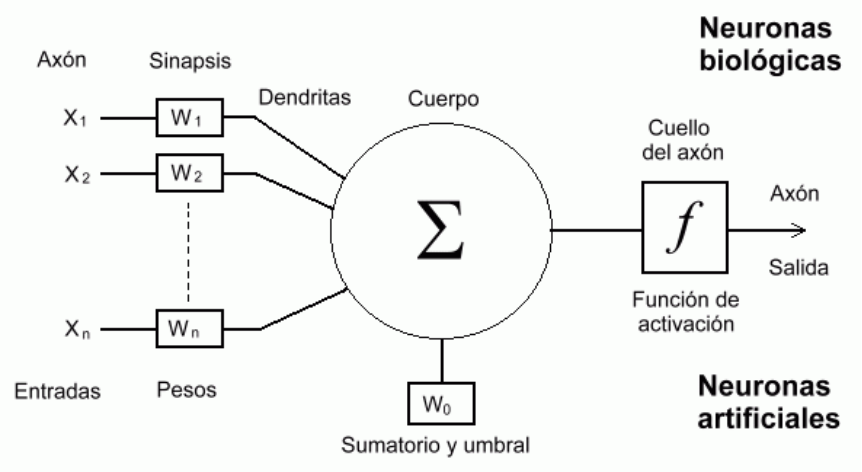

Figura 2.9 Partes principales de una neurona artificial.

2.2.2.2 Controlador multi variable directo auto ajustable basado en una red neuronal recurrente.

Debido a la existencia de sistemas complejos en los que la obtención del modelo matemático resulta complicado se desarrollaron nuevas alternativas de

control como son las redes neuronales que tiene como característica el aprendizaje, el paralelismo, la tolerancia a errores, el no requerir modelo, además se destacan por su adaptabilidad a los diferentes sistemas. Es por lo anterior que las redes neuronales son un buen candidato para controlar sistemas dinámicos no lineales en entornos complejos.

Las redes neuronales diagonales auto ajustables tiene la capacidad de controlar complejos sistemas dinámicos debido a su característica única la cual es responder regulando el error y no solo respondiendo directamente al error en la salida de la neurona, con lo que se mejora la convergencia. La Figura 2.10 muestra una red neuronal diagonal recurrente autoajustable empleada como controlador.

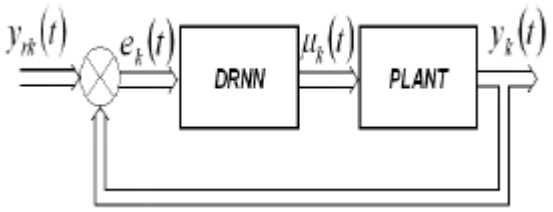

Figura 2.10 Esquema de un controlador por DRNN

2.2.2.3 Algoritmo de control recurrente

La Figura 2.11 muestra un arreglo de redes neuronales diagonales recurrentes con n entradas, l neuronas sigmoideas en la capa oculta y m neuronas lineales en la capa de salida.

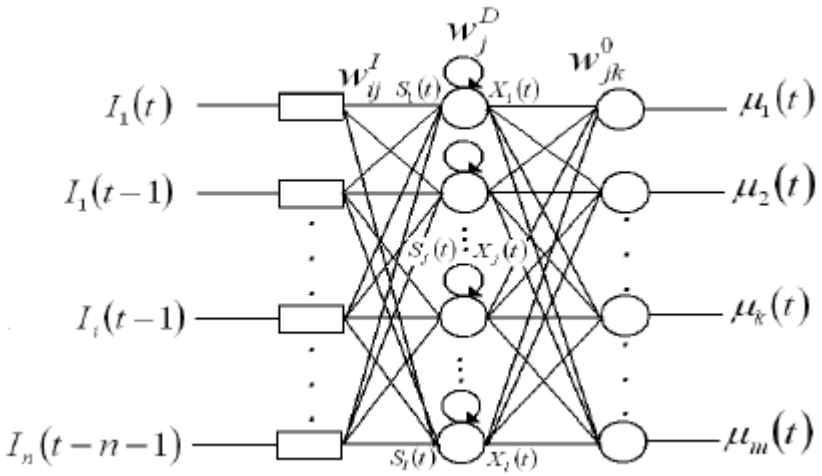

Figura 2.11 Diagrama de una red neuronal recurrente multi-variable (Pérez, 2011)

El criterio de minimización es el siguiente:

$$E(t)=\frac{1}{2}\sum_{k=1}^{m}e_k^2(t)=\frac{1}{2}\sum_{k=1}^{m}\left[y_{rk}(t)-y_k(t)\right]^2 \qquad (4)$$

El error del control está definido por:

$$e_k(t)=y_{rk}(t)-y_k(t) \qquad (5)$$

Las entradas para el entrenamiento de la red son los errores del control:

$$I_l(t)=\left[e_k(t) \quad e_k(t-1) \quad \ldots \quad e_k(t-n-1)\right] \qquad (6)$$

La salida de la neurona j de la capa oculta por función sigmoide es:

$$X_j(t)=\frac{1}{1+e^{-S_j(t)}},\ j=1,2,3,\ldots,l \qquad (7)$$

Donde

$$S_j(t)=W_j^D X_j(t-1)+\sum_{i=1}^{n}W_{ij}^I I_i(t) \qquad (8)$$

La salida para la neurona lineal k en la capa de salida es:

$$\mu_k(t) = \sum_{k=1}^{n} W_{jk}^{O} X_j \qquad (9)$$

El criterio de minimización o función de error para el entrenamiento consiste en mover en dirección negativa del gradiente de la función $E(t)$ con respecto a los pesos. El proceso de entrenamiento de una red neuronal consiste en: presentando las entradas I_l, calculando la salida de la neurona $y_k(t)$, los errores $e_k(t)$ y aplicando el proceso de minimización, los pesos se acerca ran a valores que garanticen un error mínimo entre la salida de la panta $y_k(t)$ y el set point $yr_k(t)$. En este sentido, $E(t)$ debe ser minimizado usando el meto del gradiente sobre los el vector de pesos W. El gradiente $E(t)$ es un vector multidimensional cuyos componentes son:

$$\nabla E(t) = \begin{bmatrix} \dfrac{\partial E}{\partial W_{jk}^{O}} \\ \dfrac{\partial E}{\partial W_{j}^{D}} \\ \dfrac{\partial E}{\partial W_{ij}^{I}} \end{bmatrix} \qquad (10)$$

Los componentes diferenciales del vector son resueltos usando la regla de la cadena, así, los pesos en la capa de salida son:

$$W_j^D(t) = W_j^D(t-1) + \eta \sum_{k=1}^{n} \delta_k e_k W_{jk}^{O}(t) \qquad (11)$$

Donde δ_k es:

$$\delta_k(t) = X_j(t)(1 - X_j(t))\left[X_j(t-1) + W_j^D(t)\delta_k(t-1)\right] \qquad (12)$$

Finalmente, los pesos de la capa de entrada son:

$$W_{ij}^{I}(t) = W_{ij}^{I}(t-1) + \eta \rho_{ij}(t) \sum_{k=1}^{m} e_k W_{jk}^{O}(t) \qquad (13)$$

Donde ρ_{lj} es:

$$\rho_{ij}(t) = X_j(t)(1 - X_j(t))\left[I_i(t) + W_j^D(t)\rho_{ij}(t-1)\right] \qquad (14)$$

Es importante mencionar que las ecuaciones 12 y 14 son ecuaciones dinámicas no lineales recursivas y pueden ser resueltas usando condiciones iniciales, ellas son equivalentes a ecuaciones diferenciales. Para la rede neuronal clásica (feedforward), los pesos W_j^D son cero y las ecuaciones mencionadas arriba se convierten en funciones algebraicas. Debido a este factor, las redes neuronales clásicas (feedforward) son conocidas como estáticas, mientras que las redes neuronales recurrentes son dinámicas permitiendo un mejor desempeño cuando se aplican en sistemas no lineales.

2.3 Sistemas embebidos

La demanda de FPGAs (Field Programmable Gate Array) para el diseño de sistemas embebidos aumento en los últimos años debido al bajo costo, flexibilidad para desarrollar diversas actividades en hardware y la facilidad con la que se pueden describir sistemas complejos (Leiva et al., 2013). Usando lenguajes como son VHDL (Very High Speed Integrated Circuit Hardware) o Verilog se pueden desarrollar desde pequeños circuitos combinacionales hasta complejos sistema que incluyen micro controladores, memorias embebidas y periféricos. Los desarrolladores de FPGAs como son Xilinx, Altera y Lattice proveen procesadores soft/core como MicroBlaze, Nios II y Micro32, estos procesadores tienen la ventaja de poder agregar módulos y expandir sus características para mejorar su desempeño.

2.3.1 Kernel de Linux

Los sistemas operativos basados en el kernel de Linux soportan múltiples arquitecturas de 32 y 64 bits y una de sus características es el poder trabajar en distintas configuraciones del procesador. Por lo tanto, el kernel de Linux fue

seleccionado para ser implementado en una tarjeta de desarrollo Genesys que cuenta con una FPGA Virtex5 LX50T en la que se describió un procesador con OPBs (On-Chip Peripheral Bus) y LMBs (Local Memory Bus) para instrucciones y datos por separado como lo presento Yu Du en 2005.

2.3.2 Etapas y requisitos para el diseño del sistema

El desarrollo del sistema involucra una serie de pasos que deben ser seguidos de forma ordenada debido a que los componentes depende unos de otros, por ejemplo, la configuración del kernel depende de los periféricos descritos en la FPGA por lo que se debe iniciar con la descripción para continuar con la configuración y posterior compilación.

La Figura 2.12 muestra un diagrama de bloques de las etapas para el desarrollo del sistema. El diagrama inicia con la creación del procesador MicroBlaze, menciona las etapas que se deben seguir para configurar y crear el archivo bitstream que contiene la descripción del procesador, este archivo se utiliza como base para configurar el kernel el cual, al ser compilado genera una archivo que es la imagen del sistema operativo, ambos archivos, como lo muestra la Figura 2.12, se deben combinar y descargar en la FPGA para correr el sistema operativo.

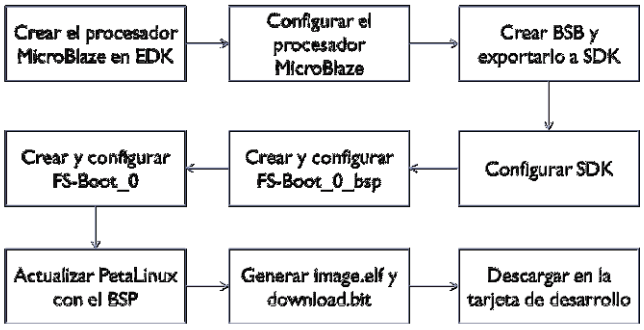

Figura 2.12 Diagrama de las etapas de desarrollo

La lista del software y herramientas utilizadas durante el desarrollo se muestra en la Tabla 2.2.

Tabla 2.2 Software y Herramientas requeridas

Software	Versión
Xilinx Integrated Software Environment (ISE)	14.7
Petalinux	2013.10
CentOS	6
Oracle VM VirtualBox	4.3.8 r92456
minicom	2.3

La Figura 2.12 menciona las herramientas Embedded Development Kit (EDK) y Software Development Kit (SDK) que son parte del ISE proporcionado por Xilinx, no se menciona el Xilinx Platform Studio (XPS) debido a que la herramienta no fue empleada a profundidad durante la tesis.

2.3.3 Procesador MicroBlaze

El procesador Microblaze tiene una arquitectura Harvard con buses separados para instrucciones y datos. Los distintos buses son descritos en la Tabla 2.3.

Tabla 2.3 Buses de comunicación

Bus	Descripción
On-Chip Peripheral Bus (OPB)	OPB en un bus síncrono de propósito general diseñado para su fácil conexión de periféricos de la clase on-chip. El OPB provee una conexión a ambos on- y off-chip periféricos y memorias
Local Memory Bus (LMB)	LMB es un bus síncrono usado principalmente para acceder al bloque de memoria RAM on-chip. Usa una cantidad mínima de señales de control y un protocolo simple para asegurar el acceso

	al bloque de memoria RAM en un único ciclo de reloj.

MicroBlaze puede ser configurado en seis formas distintas. Estas configuraciones difieren en la forma en que se encuentran interconectados los buses y las interfaces a los periféricos. La Figura 2.13 muestra la configuración usada para nuestra implementación.

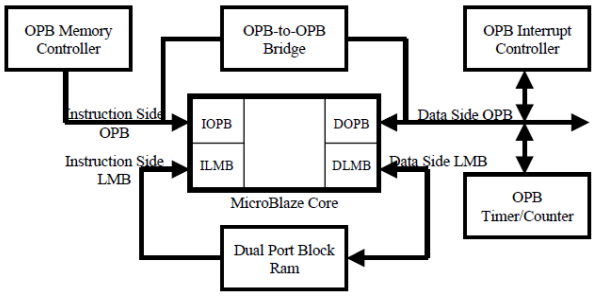

Figura 2.13 MicroBlaze configuración IOPB+ILMB+DOPB+DLMB (YU, 2005)

2.4 Actuadores y sensores

La siguiente sección expone los actuadores y sensores que integran el sistema, se mencionan sus características principales así como el circuito mínimo para su implementación.

2.4.1 MAX31855: Convertidor termopar a digital (Unión fría compensada)

2.4.1.1 Descripción

El integrado MAX31855 digitaliza la señal de un termopar además compensa la unión fría. El dato de salida (14bits), es enviado mediante el protocolo SPI en modo solo lectura. El integrado tienen una exactitud de +/-2ºC para termopares tipo K.

2.4.1.2 Características

- Unión fría compensada.

- 14-Bit, resolución 0.25ºC
- Versiones disponibles para termopares tipo K, J, N, T, R y E.
- Comunicación SPI (solo lectura)
- Detección de corto a GND o Vcc
- Detección de falla en termopar.

2.4.1.3 Circuito mínimo

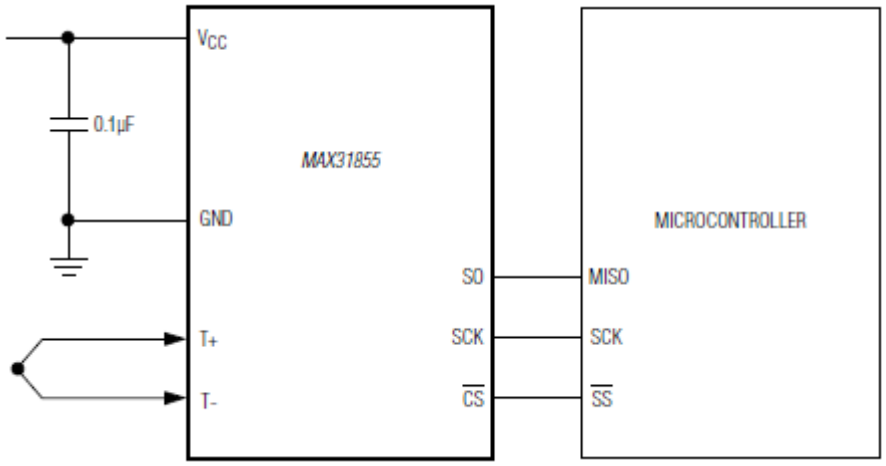

Figura 2.14 Circuito mínimo para MAX31855 (maxim integrated, 2014)

2.4.2 DAC7612: Convertidor dual digital-analógico (12bit entrada serial)

2.4.2.1 Descripción

El DAC7612 es un convertidor digital-analógico de 12 bits, requiere una alimentación de +5V, contiene un registro de recorrimiento, un latch, un voltaje de referencia de 2.435V, un DAC dual y una alta velocidad de respuesta.

La interfaz serial es compatible con la mayoría de los DSPs y micro controladores. Cuenta con las señales necesarias para una interfaz serial síncrona además de contar con una señal de “carga”.

2.4.2.2 Características

- Bajo consumo: 3.7mW
- Configuración rápida: 7us apara 1 LSB

- 1mV por bit con una escala completa a 4.095V
- Señal de referencia
- 12 bit lineales y monótonos bajo temperaturas industriales.
- Interfaz de comunicación de 3 cables con reloj de hasta 20MHz
- Encapsulado: SOIC 8 pines

2.4.2.3 Circuito mínimo

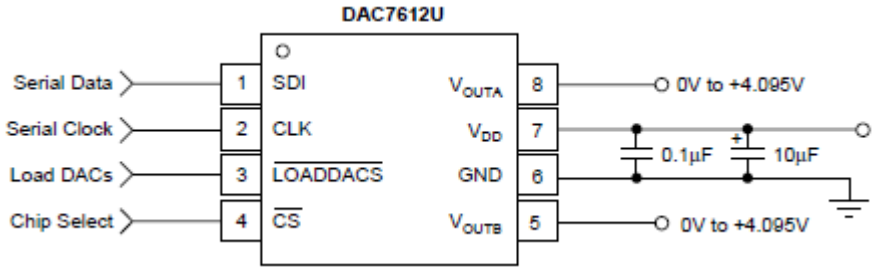

Figura 2.15 Circuito mínimo para DAC7612U (Burr-Brown, 1999)

2.4.3 Pirómetro de radiación

2.4.3.1 Descripción

Sensor para radiación solar en el rango 300 a 1100 nanómetros. La salida del sensor es lineal: $W/m^2 = V * 500$

2.4.3.2 Especificaciones

- Rango: 0-1500 W/m2, +/-5%
- Voltaje de alimentación: 3-5VDC
- Salida del sensor: 0-3VDC

2.4.4 CS5480: Circuito integrado para la medición de energía con tres canales

2.4.4.1 Descripción

El integrado CS5480 es un medidor de energía con tres canales de alta precisión, incorpora convertidores analógicos a digitales por cada canal, así como un circuito de referencia y salida de mediciones de energía activa, pasiva y aparente, además factor de potencia y valor RMS, la salida es mediante un puerto serial.

Los valores de corriente, voltaje y potencia están disponibles mediante el puerto serial. El integrado puede ser configurado para enviar señales de interrupción al detectar cruce por cero. El integrado soporta múltiples sensores como son: resistencias shunt, trasformadores de corriente y bobinas de rogowski.

2.4.4.2 Características

- Desempeño superior con bajo ruido y alta Signal-to-noise ratio (SNR)
- Precisión en la medición de energía del %0.1 bajo un rango dinámico 4000:1
- Precisión en la medición RMS del %0.1 bajo un rango dinámico de 1000:1
- 3 convertidores independientes analógicos a digital.
- 3 salidas digitales configurables.
- Soportes para sensores: resistencias shunt, Trasformadores de corriente y bobinas de Rogowski.
- Calculo y medición en el chip
    - Potencia activa, reactiva y aparente
    - Corriente y voltaje RMS
    - Factor de potencia y frecuencia en la línea
    - Voltaje, corriente y potencia instantánea
- Detección de sobre corriente, baja tención.
- Comunicación serial
- Sensor de temperatura
- Voltaje de alimentación: +3.3V

2.4.4.3 Circuito mínimo

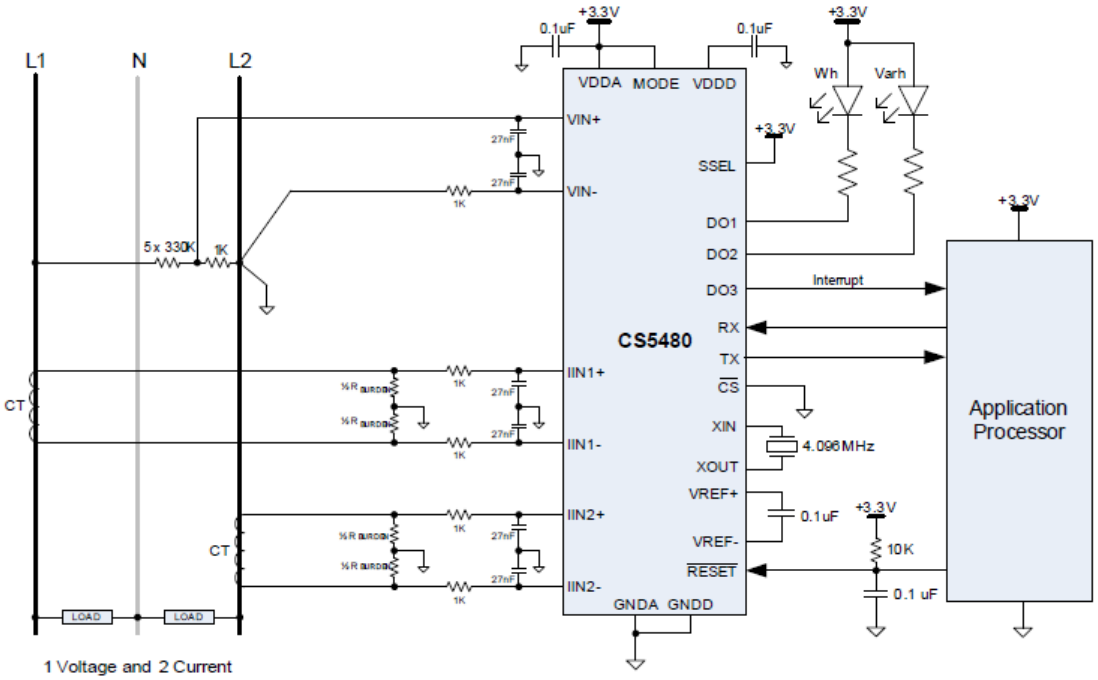

Figura 2.16 Configuración típica para conexión 1~fase 3-cables (Cirrus Logic, 2011)

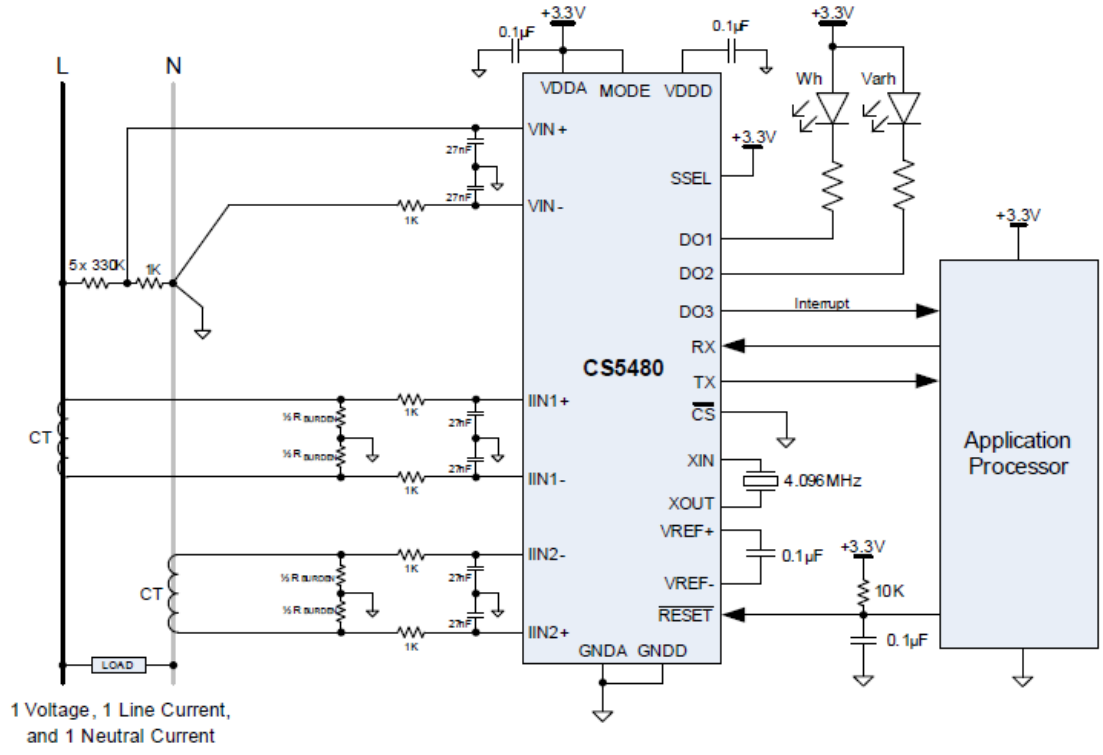

Figura 2.17 Configuración típica para conexión 1~fase 2-cables (Cirrus Logic, 2011)

2.4.5 D6F-W: Sensor de velocidad de aire

2.4.5.1 Descripción

El integrado D6F-W es un sensor de velocidad de aire unidireccional, con escalas de medición de 0 a 1 m/s, 0 a 4m/s y 0 a 10 m/s, con precisiones +/-5% a escala completa. Su salida de voltaje no es lineal, un ejemplo de su comportamiento se presenta en la Tabla 2.4 y en la Figura 2.18:

Tabla 2.4 D6F-W10A1 voltaje vs flujo

Velocidad del flujo (m/s)	**0**	**2**	**4**	**6**	**8**	**10**
Voltaje de salida (V)	1 +/-0.24	1.94 +/-0.24	3.23 +/-0.24	4.25 +/-0.24	4.73 +/-0.24	5 +/-0.24

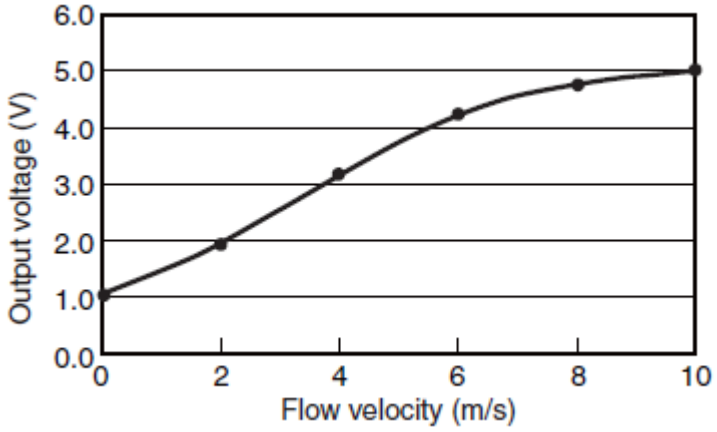

Figura 2.18 Grafica flujo (m/s) vs Salida de voltaje (V) (OMRON)

2.4.5.2 Conexión eléctrica

La salida del sensor es analógica en un rango de 0 a 5V, su diagrama de conexión muestra en la Figura 2.19.

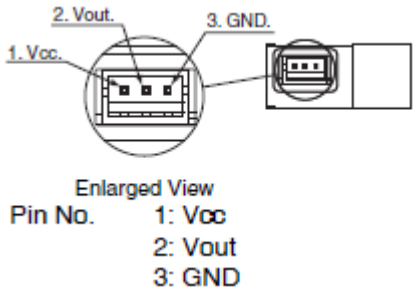

Figura 2.19 Diagrama de conexión D6F-W10A1 (OMRON)

2.4.6 DRV8812: Circuito integrado con puente dual para el control de motores

2.4.6.1 Descripción

El integrado DRV8812 cuenta con dos puentes H lo que permite controlar un motor a pasos bipolar o dos motores de corriente directa. El controlador es compatible con motores de 1.6 A (con conexión a 24V)

Su comunicación es mediante una interfaz paralela compatible con estándares industriales. Cuenta con protecciones contra cortocircuito, bajo voltaje, bloqueo y sobrecalentamiento.

2.4.6.2 Características

- Puente H dual
    - Compatible con un motor a pasos bipolar o dos motores de corriente directa.
    - Sensor de corriente y configuración de niveles de corriente.
- Corriente de operación 1.6A a 24V 25º C
- Salida de voltaje de referencia de 3.3V
- Interfaz de control paralelo
- Voltaje de alimentación 8-45V

2.4.6.3 Circuito mínimo

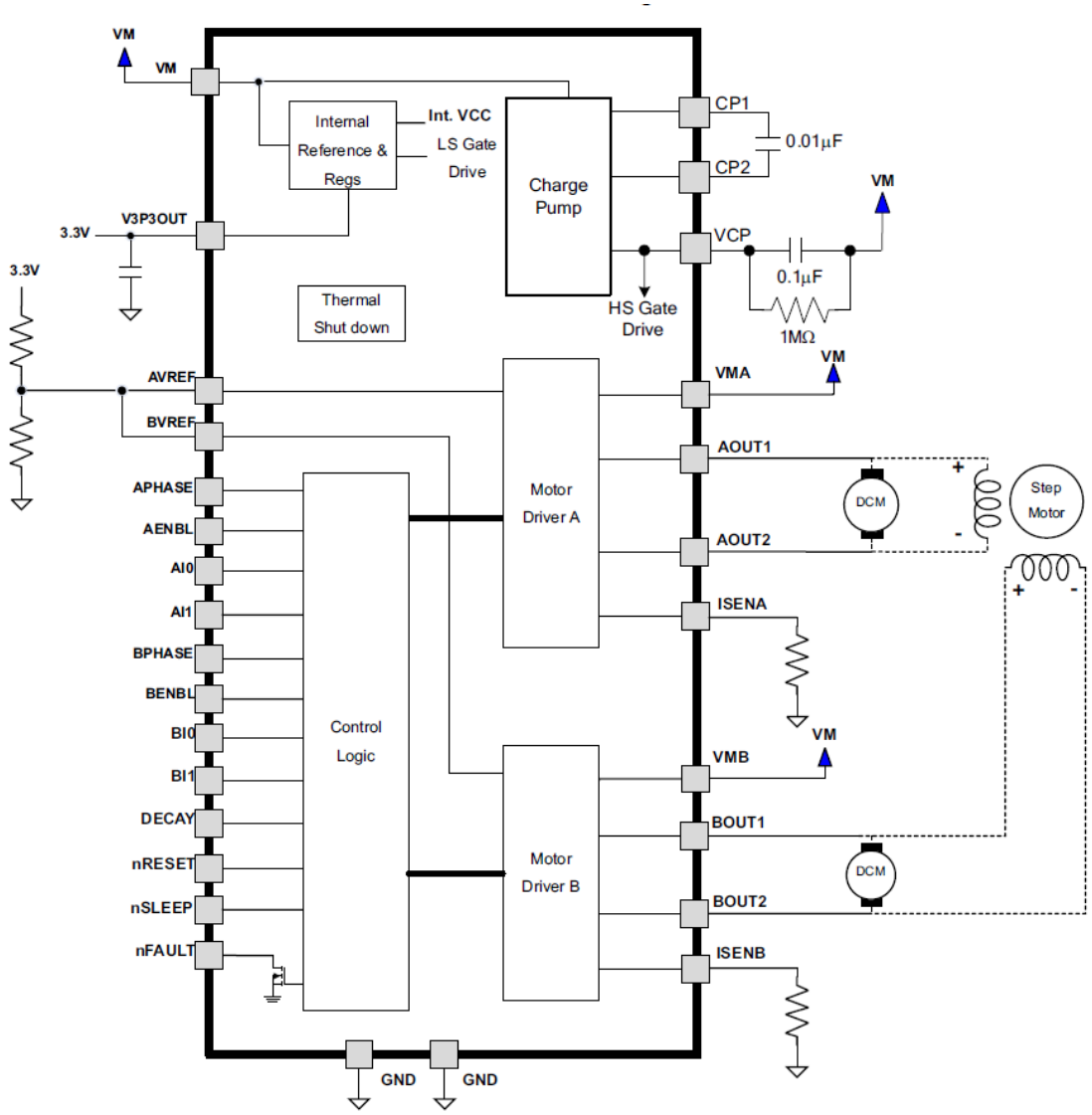

Figura 2.20 Configuración típica para conexión de motor a pasos (Texas Instruments, 2013)

3. Metodología

3.1 Sistema automático

El deshidratador en el que se realizó el experimento se recibió sin ningún tipo de control, por lo que se realizó una selección de instrumentos de medición y actuadores para posteriormente alimentar un sistema de control en lazo cerrado.

3.1.1 Adquisición de datos

El control propuesto requería de información no solo de variables internas sino que se requerían valores climatológicos para su consideración dentro de los algoritmos, por lo que fue necesaria una selección de instrumentos que permitieran recuperar datos como son: radiación solar, temperatura y humedad exterior, flujo de agua y aire, temperatura del agua y temperatura y humedad en diferentes puntos del deshidratador, así como sensores de corriente y voltaje para el cálculo de potencia.

3.1.2 Etapa de potencia

Los actuadores de mayor potencia son: el motor que recircula el agua, el soplador y las resistencias eléctricas. Para controlarlos se requirió la selección de variadores de frecuencia y relevadores de estado sólido acorde a la potencia de cada actuador.

3.1.3 Interfaz gráfica de usuario

Para la interacción hombre maquina fue necesaria la creación de una aplicación web que almacena y presenta la información mediante gráficos, además mediante control de acceso permitiera la modificación de los parámetros como son los setpoints y el estado de algunos actuadores.

3.2 Procesador: MicroBlaze

Una de las piezas fundamentales en el desarrollo de un sistema mínimo es el procesador, por lo que fue necesaria la implementación de un procesador de la clase soft en una tarjeta Genesys con FPGA Virtex 5.

3.2.1 Documentación

Se requirió una fase para la revisión de bibliografía en la que se encontraron manuales, y tesis que permitieron el entendimiento de las diferentes partes de un procesador así como mostrar el procedimiento estándar para describirlo.

3.2.2 Requerimientos de Software y Hardware

Etapa destinada para la adquisición de los diferentes componentes de software y hardware necesarios para el montaje y prueba del procesador.

3.2.3 Implementación

Etapa en la que se cargó y probo mediante códigos en c la funcionalidad del procesador descrito siguiendo la documentación encontrada.

3.3 OS: uCLinux

La distribución seleccionada para ser implementada fue la llamada PetaLinux que es una sub distribución de uCLinux. Petalinux es una versión optimizada para las FPGAs de Xilinx lo cual fue el factor determinante en la elección del sistema operativo.

3.3.1 Documentación

Etapa destinada a la consulta de diferentes fuentes de información para conocer el procedimiento para el montaje del sistema operativo en una tarjeta Virtex 5 con procesador MicroBlaze

3.3.2 Requerimientos de Software

Etapa en la cual se obtuvo el software que se requiere para el montaje del sistema operativo.

3.3.3 Implementación

Etapa destinada a la implementación y documentación del proceso para la puesta a punto del sistema así como de las pruebas para confirmar su funcionamiento.

3.3.4 Dispositivo alterno

El sistema montado en la tarjeta Genesys fue presentado en congreso internacional con la intención de demostrar las capacidades de una FPGA Virtex 5 y motivar su desarrollo, por desgracia las tarjetas de desarrollo no deben ser empleadas en entornos industriales, por lo que se destinó una etapa para la selección de un dispositivo alterno que cuanta con una distribución de Linux en la que se puede ejecutar los programas de control de igual forma.

3.4 Control Neuronal

Uno de los algoritmos de control que se implemento fue una red neuronal diagonal recurrente auto ajustable, la cual se programó en Python para su implementación en Linux.

3.4.1 Pruebas al control neuronal auto-ajustable

El controlador neuronal fue probado para registrar su comportamiento, se buscaba registrar las acciones de control que tomaba para posteriormente configurar el control difuso considerando el control neuronal como un agente inteligente, además, se registró el consumo energético que requiere el controlador para alcanzar el setpoint en diferentes condiciones climatológicas.

3.5 Control difuso

El controlador difuso fue implementado en Python para ser ejecutado en la plataforma con sistema operativo PetaLinux. Se configuro de tal forma que recibiera la misma información que el controlador neuronal y se generaran las mismas señales de control, con la intención de comparar ambos controladores bajo las mismas condiciones.

3.5.1 Reglas del control difuso

Las reglas fueron obtenidas mediante una FAM utilizando los datos obtenidos del controlador neuronal como agente experto y utilizando el conocimiento empírico de los trabajadores se realizaron las modificaciones a las reglas de tal forma que se obtuviese el mismo comportamiento en relación al controlador neuronal.

3.6 Consumo energético

La tesis presentada propone una hipótesis en la que se establece que el controlador neuronal puede realizar el control de un deshidratador industrial utilizando menores recursos en cuanto a energía eléctrica se refiere, por lo que se realizaron pruebas en las que se utilizó un controlador difuso y uno neuronal.

3.6.1 Banco de datos

El banco de datos fue formado por el reporte del comportamiento del deshidratador durante pruebas que consistieron en enfriar el deshidratador y posteriormente iniciar el procedimiento para que el deshidratador alcanzara una temperatura de 50º C con un flujo de 1.5m/s. En todas las pruebas se buscó que el clima exterior fuera similar con la intención de estandarizar la prueba y que el factor diferente fuera el controlador.

3.6.2 Resultados

Los resultados se obtuvieron mediante estadísticas de correlación entre el clima y el consumo energético por cada controlador. Posteriormente se realizó una comparación entre el consumo entre controladores.

4. Resultados y discusión

4.1 Acondicionamiento del deshidratador

Como se describió en la sección 2.1.3, el deshidratador fue recibido sin instrumentación y con actuadores básicos con controles on/off, por lo que fue necesaria la adquisición de sensores y el diseño para el mecanismo de los actuadores.

Los sensores y actuadores que fueron seleccionados se describen en la sección 2.4. En general, el acomodo de los sensores fue el siguiente:

- Tres sensores de velocidad del viento fueron colocados de la siguiente forma: uno en la entrada del aire fresco, uno en la salida del aire caliente húmedo y uno en el tubo de retroalimentación de aire caliente.
- Cuatro sensores de temperatura y humedad, uno en la cámara de deshidratado, otro en la salida de aire caliente húmedo, uno en el exterior del deshidratador, y un último en el exterior del edificio.
- Un sensor de radiación solar colocado en el exterior de edificio.

- Un termopar colocado en un pozo dentro del tanque de agua.
- Un medidor de flujo de agua colocado en la tubería de retorno de agua fría.
- Tres censores para la medición de potencia: uno colocado en el soplador, uno en la bomba de recirculación de agua y uno en la alimentación principal del sistema de calefacción y control.
- Interruptores de contacto para determinar el cierre de las compuertas.

Los actuadores fueron colocados en los soportes diseñados utilizando el software CAD NX7, los diagramas se muestran en el apéndice A y los renders se observan en las siguientes figuras.

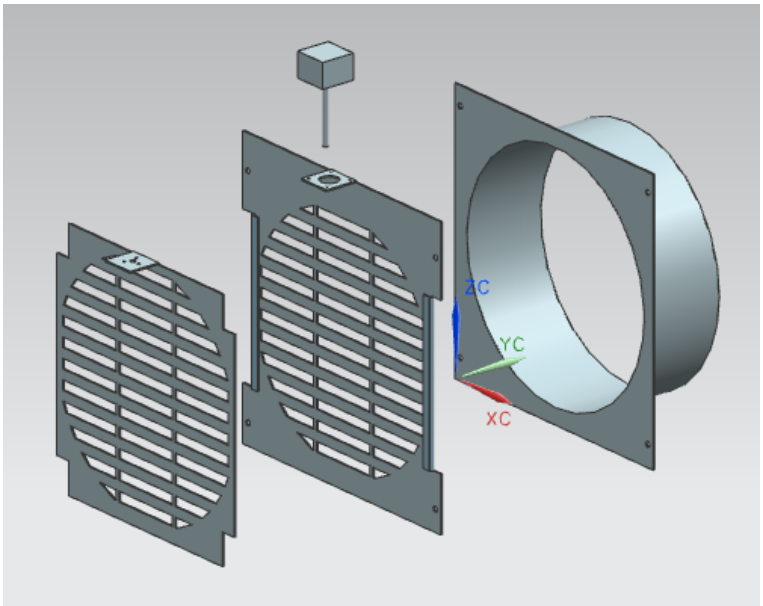

Figura 4.1 Render de la compuerta de entrada con motor a pasos

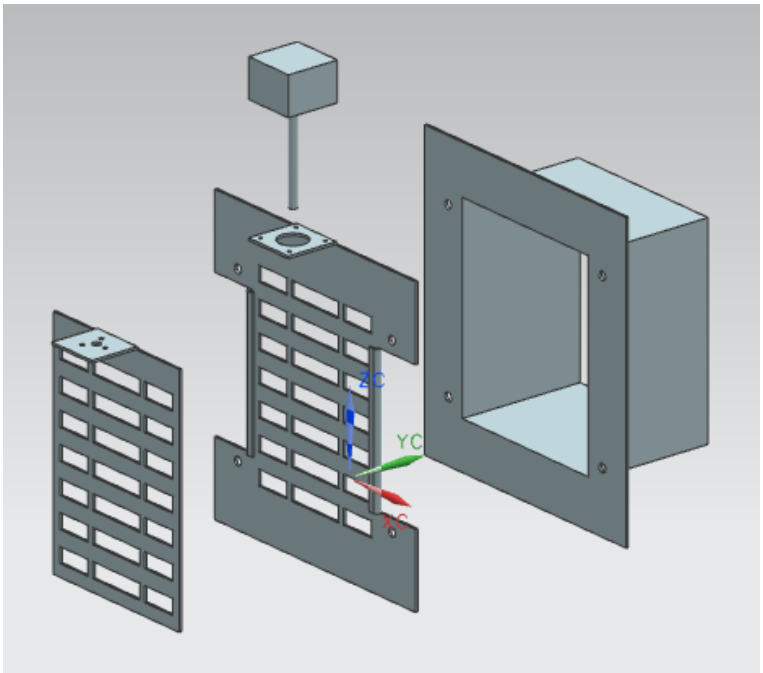

Figura 4.2 Render de la compuerta de salida con motor a pasos

4.2 Implementación del kernel de Linux

Para automatizar el proceso de desarrollo del sistema se recurrió a los archivos Base System Builder (BSB) que contienen las configuraciones y los diseños optimizados para cada tarjeta de desarrollo, el BSB utilizado es proporcionado por el fabricante y es cargado al ISE para facilitar la configuración del hardware.

La Tabla 4.1 resume las modificaciones que se realizaron al BSB utilizando el asistente de diseño del ISE.

Tabla 4.1 Configuración del Hardware

Componente	Característica	Configuración
Board	Vendor	Digilent
	Name	Digilent Genesys System

		Board
	Revision	C
System	Type	Single-processor system
Procesador	Reference clock frequency	100MHz
	Processor type	MicroBlaze
	System clock frequency	125MHz
	Local Memory	64KB
	Debug interface	On-Chip HW Debug Module
	Enable Floating Point unit	NO
Hard_Ethernet_MAC	DMA	YES
	Interrupt	YES
RS232_Uart_0	RS232_Uart_0	xps_uartlite
	Baud rate	115200
	Data Bits	8
	Parity	None
	use interrupt	YES
RS232_Uart_1	Eliminado	
xps_timer	Agregado	
	Core	xps_timer

	Count width	32
	Configure mode	Two timers are present
	Interrupt	YES

Es posible eliminar el segundo puerto UART, no se empleara durante el desarrollo. El timer agregado es requisito del sistema operativo, al ser un BSB genérico no cuenta con el modulo precargado pero si es posible agregarlo.

El asistente utiliza la información del archivo BSB para pre configurar con valores predeterminados los distintos componentes, al concluir el asistente considerando las modificaciones de la Tabla 4.1, se puede acceder a los archivos system.mhs y system.mss, que contiene la descripción del hardware y del software.

4.2.1 Modificación del archivo system.mhs

Se debe modificar el procesador identificado con el nombre microblaze_0 para que soporte “barrel shifter”. Lo anterior mejora el desempeño de la unidad lógica aritmética al permitir realizar corrimientos de forma pre optimizada.

4.2.2 Modificación del archivo system.mss

Es necesario modificar el archivo de la siguiente forma. En la sección “OS & Library settings” cambiar el OS de “standalone” por “petalinux”. En la sección “OS and lib configuration” se debe localizar los siguientes componentes y asociarlos a su respectivo periférico:

- stdin: RD232_Uart_0
- stdout: RD232_Uart_0
- main_memory: DDR2_SDRAM
- flash_memory: FLASH

• lmb_memory: dlmb_cntlr

• gpio: LEDs_8bit

• Timer: xps_timer_0

Al finalizar las configuraciones no queda más que generar el archivo bitstram.bit al mismo tiempo que se empaqueta en una estructura de archivos denominada Board Support Package (BSP) que contiene todos los componentes de software requeridos para formar el sistema operativo en su respectivo entorno.

4.2.3 Configuración y compilación del kernel

La herramienta Petalinux depende del BSP para configurar el kernel. Para dar de alta el BSP generado por ISE es necesario ejecutar la siguiente instrucción: "$ petalinux-create -t project -- template microblaze - n PetaLinux-MicroBlaze_v01", con ello se agrega el software, para crear una copia del hardware que será utilizado por el compilador se utiliza la instrucción "$ petalinux-config --get-hw-description -p ~ /Desktop /Xilinx /Projects /XUP - MicroBlaze-project_v01 /PetaLinux-MicroBlaze_v01/".

Para acceder al entorno grafico (Figura 4.3) que permite configurar el kernel se utiliza la instrucción "$ petalinux-config", ya en ella se selecciona el vendor "Xilinx" y el product "XUP-MicroBlaze-project".

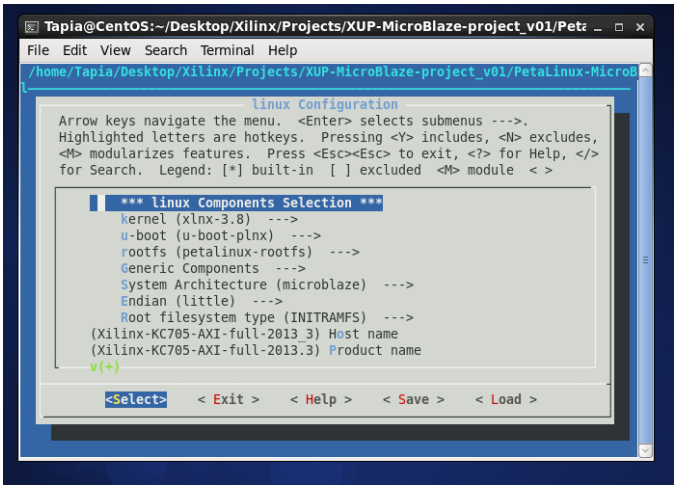

Figura 4.3 Entorno de configuración Linux

El BSP contiene una configuración predeterminada que permite operar el sistema sin necesidad de realizar mayores cambios en el kernel, por lo tanto solamente se modifican los valores de vendor y product y se ejecuta la instrucción “petalinux-build” para compilar el kernel. La Tabla 4.2 describe los archivos resultantes, siendo el de mayor importancia, el archivo image.elf.

Tabla 4.2 Configuración del Hardware

Nombre	Descripción
Kernel de Linux e Imágenes del sistema	
image.elf	Kernel de linux e imagen del sistema de archivos root en format ELF
image.ub	Kernel de linux e imagen del sistema de archivos root en format U-Boot
U-Boot	
u-boot.bin	Imagen U-Boot en formato binario
u-boot.srec	Imagen U-Boot en formato ELF
u-boot.elf	Imagen U-Boot en formato SREC
u-boot-s.bin	Imagen U-Boot reubicable en formato binario
u-boot-s.elf	Imagen U-Boot reubicable en formato ELF
u-boot-s.srec	Imagen U-Boot reubicable en formato SREC

Para genera el archivo download.bit se utiliza la herramienta data2men, la cual combina: el archivo con la descripción de la memoria (system_db.mm generado por ISE/XPS), el código asociado al hardware, el archivo con la descripción del hardware (system.bit generador por ISE/XPS) y el arrancador del

sistema operativo (fs-boot_0.elf creado por SDK). El resultado es un archivo que cuenta con un arrancador actualizado. La Figura 4.4 muestra el código que se utiliza para la creación del .bit con la herramienta data2mem.

```
#!/bin/sh
data2mem \
   -bm system_bd.mm \
   -p xc5vlx50tff1136-1 \
   -bt system.bit \
   -bd fs-boot_0.elf \
   tag microblaze_0 \
   -o b download.bit
```

Figura 4.4 Código en bash para utilizar data2mem

4.2.4 Botear y pruebas al sistema

4.2.4.1 Botear la imagen

El proceso de arranque del sistema inicia con descargar el archivo "download.bit", para ello se utiliza la instrucción "$ petalinux-boot --jtag --fpga --bitstream tempConfig /download.bit -v", como resultado en la hiperterminal se muestra el mensaje de la Figura 4.5, en la que se advierte de la necesidad de descargar la imagen que contiene el SO.

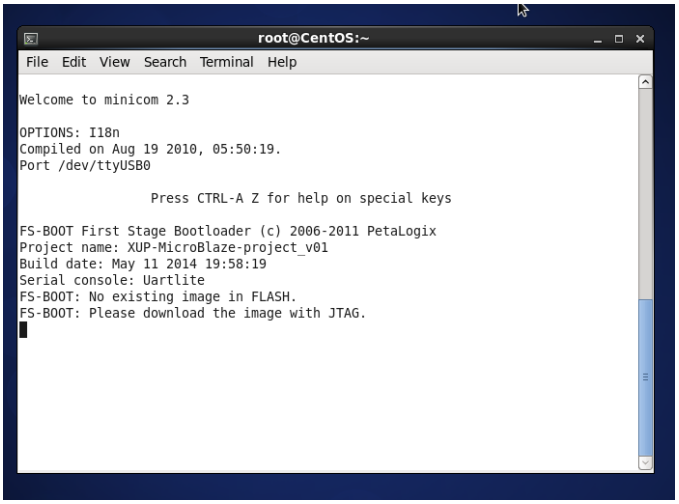

Figura 4.5 Salida por el serial al descargar el archivo download.bit

Para descargar la imagen se utiliza la siguiente instrucción “$ petlinux-boot --jtag --image images/linux/image.elf -v”. Al terminar la descarga se inicia automáticamente el sistema operativo, al concluir con la inicialización se muestra un mensaje similar al de la Figura 4.6.

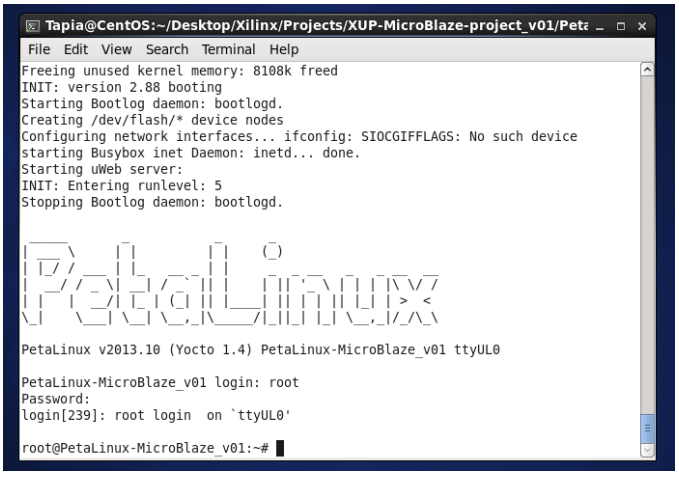

Figura 4.6 Salida por el serial al descargar la imagen de SO y acceder como usuario

4.2.4.2 Prueba del sistema

La prueba se realizó desde la cuenta “root” que tiene como contraseña la palabra “root”. Para realizar la prueba se implementó el siguiente script que imprime en cuatro ocasiones el mensaje “Hola mundo” precedido del número de renglón.

```
for i in 1 2 3 4
do
        echo “$i Hola mundo“
done
```

El siguiente script contiene los comandos completos para acceder a la carpeta temporal, crear un archivo .sh, modificar los atributos, agregar el código al archivo .sh y ejecutar el programa.

```
# cd /tmp/
# touch script.sh && chmod +x script.sh
# echo ‘for i in 1 2 3 4; do echo “$i Hola mundo”;  done’ > script.sh
# ./script.sh
```

La Figura 4.7 muestra la salida al ejecutar el script.

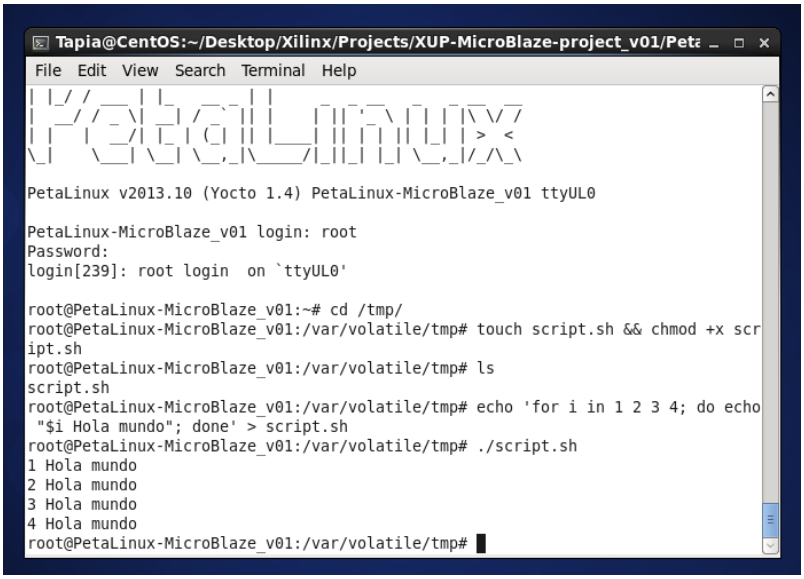

Figura 4.7 Salida por el serial al ejecutar el script

El procedimiento detallado puede ser consultado en el apéndice B.

4.3 Integración de componentes en la tarjeta electrónica

El diseño de la tarjeta electrónica se realizó utilizando el software de diseño Altium Designer, en ella se integraron los diferentes componentes mencionados en la sección 2.4. La Figura 4.8 muestra el render de la tarjeta, en ella se aprecian los componentes que forman parte de la tarjeta, recordando que los sensores de flujo de aire y agua así como el termopar y los sensores de temperatura y humedad son conectados mediante cables a la tarjeta, es por ello que se requirió colocar seis bloques de conectores de 12 posiciones cada uno para poder conectar a una tarjeta todos los periféricos necesarios. Además cuenta con comunicación serial mediante un trasmisor ZigBee.

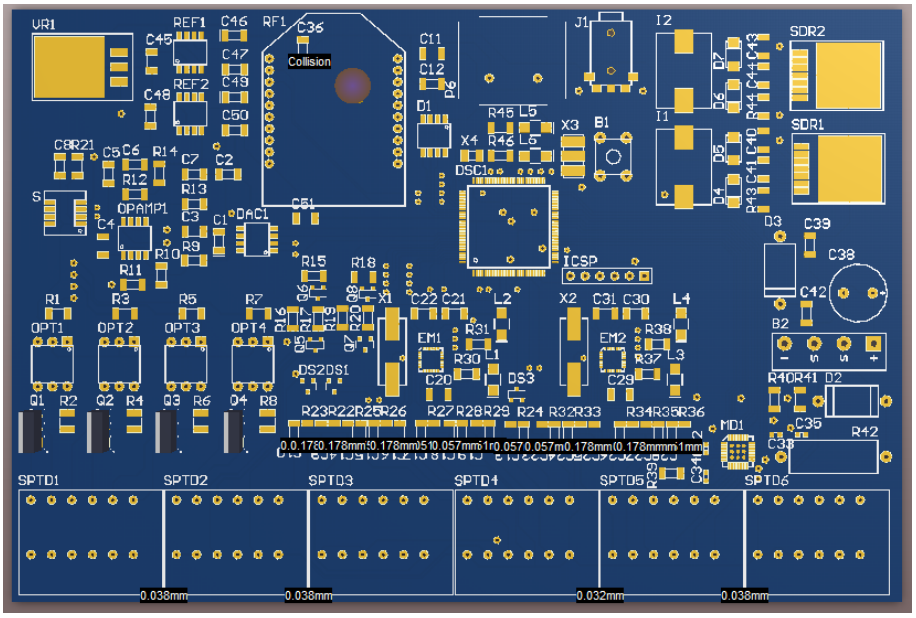

Figura 4.8 Render de la RTU

La tarjeta fue diseñada en dos capas (Figura 4.9), se implementaron fuentes de precisión para ser utilizadas como voltajes de referencia y así disminuir el ruido que se pudiera presentar en la lectura de los sensores analógicos.

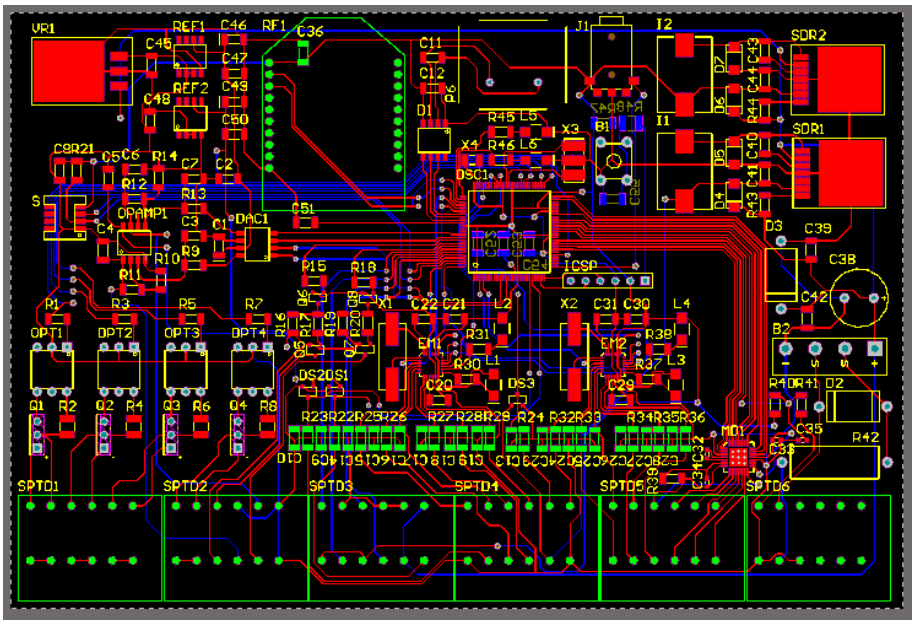

Figura 4.9 Vista de las dos caras del PCB

4.4 Programación

La programación del sistema requirió implementar diversos lenguajes de programación como son: “c” para la programación del dsPIC, “Python” para el desarrollo de los algoritmos de control y “C#” para el desarrollo la lógica implementada en el servidor. Algunos lenguajes de apoyo como son “HTML” o código para “bash” fueron implementados en menor medida, a continuación se presenta la descripción de las clases principales empleadas en cada uno de los códigos.

4.4.1 Red neuronal

La red neuronal fue programada en Python utilizando como guía la tesis presentada por Estela Pérez en el 2011 dirigida por el M. en C. Alfonso Noriega Ponce. En la cual implementan una red neuronal diagonal recurrente (DRNN) para el control de un proceso.

El fragmento de código mostrado en la Figura 4.10 es un algoritmo diseñado para inicializar y ejecutar una prueba a la DRNN.

```
6
7  from DRNN import myDRNN
8
9  def program():
10     SetPoint = [10]
11     epocas = 100
12     epoca = 0
13
14     #Crear red [SP,entrenamiento,entradas,salidas,entrada_Zs,capa_oculta]
15     RN = myDRNN(SetPoint, 0.8, 1, 1, 1, 2)
16
17     #red
18     uk_T = [15]
19     while epoca < epocas:
20         epoca += 1
21         uk_T[0] -= 0.2
22         RN.goDRNN(uk_T)
23         print('---- %i ----' % epoca)
24         RN.getInfo()
25
26 if __name__ == '__main__':
27     program()
28
```

Figura 4.10 Fragmento del código para la inicialización de la DRNN

El siguiente fragmento del código es la implementación del algoritmo para una red neuronal diagonal recurrente. El código completo puede ser consultado en la sección Python DRNN del apéndice C,

```
    def goDRNN(self,UK_T):

        global n,i,k,i_z,j,Y_r,I_i,W_I_ij,W_I_ij_z1,W_D_j,W_D_j_z1,W_O_jk,W_O_jk_z1,X_j,X_j_z1,sg_k,sg_k_z1,p_ij,p_ij_z1,e,uk,uk_T
        self.uk_T = UK_T

        #etapa 1
        for _j in range(0,self.j):            #capa oculta
            self.S_j = self.W_D_j[_j]* self.X_j_z1[_j]
            for _i in range(len(self.I_i)):
                self.S_j += self.W_I_ij[_i][_j] * self.I_i[_i]
            self.X_j[_j] = 1 / (1 + exp(-1 * self.S_j))
        for _k in range(0,self.k):             #capa salida
            self.uk[_k] = 0
            for _j in range(0,self.j):
                self.uk[_k] += self.W_O_jk[_j][_k] * self.X_j[_j]

        #etapa 2
        for _j in range(0,self.j):
            for _k in range(0,self.k):
                self.e[_k] = self.Y_r[_k] - self.uk_T[_k]    #errores
                self.W_O_jk[_j][_k] = self.W_O_jk_z1[_j][_k] + self.n * self.e[_k] * self.X_j[_j]
        for _j in range(0,self.j):
            sumAEWo = 0
            for _k in range(0,self.k):
                self.sg_k[_k] = self.X_j[_j] * (1 - self.X_j[_j]) * (self.X_j_z1[_j] + self.W_D_j[_j] * self.sg_k_z1[_k])
                sumAEWo += self.sg_k[_k] * self.e[_k] * self.W_O_jk[_j][_k]
            self.W_D_j[_j] = self.W_D_j_z1[_j] + self.n * sumAEWo
        for _i in range(0,self.i):
            for _j in range(0,self.j):
                self.p_ij[_i][_j] = self.X_j[_j] * (1 - self.X_j[_j]) * (self.I_i[_i] + self.W_D_j[_j] * self.p_ij_z1[_i][_j])
                sumEWo = 0
                for _k in range(0,self.k):
                    sumEWo += self.e[_k] * self.W_O_jk[_j][_k]
                self.W_I_ij[_i][_j] = self.W_I_ij_z1[_i][_j] + self.n * self.p_ij[_i][_j] * sumEWo

        #etapa 3
        for _i in range(0,len(self.I_i),1+self.i_z):
            for _i_t in range(0,self.i_z):
                self.I_i[_i+1+_i_t] = self.I_i[_i+_i_t]
            self.I_i[_i] = self.e[_i]
        for _j in range(0,self.j):
            self.X_j_z1[_j] = self.X_j[_j]
            self.W_D_j_z1[_j] = self.W_D_j[_j]
        for _k in range(0,self.k):
            self.sg_k_z1[_k] = self.sg_k[_k]
        for _j in range(0,self.j):
            for _k in range(0,self.k):
                self.W_O_jk_z1[_j][_k] = self.W_O_jk[_j][_k]
        for _i in range(0,self.i):
            for _j in range(0,self.j):
                self.p_ij_z1[_i][_j] = self.p_ij[_i][_j]
                self.W_I_ij_z1[_i][_j] = self.W_I_ij[_i][_j]
```

Figura 4.11 Fragmento del código para DRNN

El algoritmo mostrado en la Figura 4.11 se encuentra dentro de una clase que permite ser inicializada con cualquier cantidad de neuronas y parámetros de entrada y salida. Fue diseñada de forma que pudiera ser implementada con facilidad por lo que sus funciones son estándar para adaptarse a los datos proporcionados por el despachador.

4.4.2 Lógica difusa

El controlador difuso fue programado de tal forma que sea fácilmente adaptable a las entradas que el despachador proporciona, con ello se garantiza

que las entradas que la red neuronal utilizo serán las mismas que el controlador neuronal utiliza para realizar el control.

El código mostrado en la Figura 4.12 es el utilizado para realizar pruebas al algoritmo de lógica difusa.

```
from FUZZY import myFUZZY

def programa():
    epocas = 10
    epoca = 0

    min=-1
    max=1
    #SetPoint
    SP = [5]
    #Entrada de datos
    I = [0]
    #Transformación
    I[0] = I[0] * 0.1
    SP[0] = SP[0] * 0.1
    e = [0,0]


    ep=5    #numero de funciones de membresia de la variable linguistica error de posicion
    ev=5    #numero de funciones de membresia de la variable linguistica error de velocidad
    #centros de las funciones de membresia de entrada y salida
    cep = [-1,-0.1,0,0.1,1] #vector de centros de las funciones de membresia ep1
    cev = [-1,-0.1,0,0.1,1] #vector de centros de las funciones de membresia ev1

    #reglas
    reglas = [[1,1,1,1,0],[1,0.1,0.1,0,-1],[1,0.1,0,-0.1,-1],[1,0,-0.1,-0.1,-1],[0,-1,-1,-1,-1]]

    #distancia de los centros de las funciones de membresia
    MD = [0.9, 0.1, 0.1, 0.9, 0.1]
    MI = [0.1, 0.9, 0.9, 0.1, 0.9]
    MDe = [0.9, 0.1, 0.1, 0.9, 0.1]
    MIe = [0.1, 0.9, 0.9, 0.1, 0.9]

    FZ = myFUZZY(ep, ev, cep, cev, reglas, MD, MI, MDe, MIe)

    while epoca < epocas:
        epoca += 1
        for dato in I:
            if (dato > min and dato < max):
                e[1] = e[0]
                e[0] = I[0] - SP[0]
                de = e[1]
                res = FZ.goFUZZY(e[0], de)
                print('Epoca: %s' % epoca)
                print('Señal: %s' % res)
            else:
                print("Error en los limites")

if __name__ == '__main__':
    programa()
```

Figura 4.12 Programa de prueba para el control difuso

El siguiente fragmento pertenece a la clase FUZZY.

```
    def goFUZZY(self, qtil, qptil):
        global cep, cev

        #registros para almacenar las FM activas
        e_cont = 0
        e_int = 0
        de_cont = 0
        de_int = 0
        #numerador y denominador del defuzificador
        num = 0.0
        den = 0.0
        #variables para almacenar grado de membresia
        mfe = [0.0 for y in range(self.ep)]
        mfc = [0.0 for y in range(self.ep)]
        #guarda el resultado de la inferencia
        prem = 0.0
        #salida del defuzificador
        ui = 0.0

        #-----------------------fusificacion---------------------------

        #para la variale linguistica error de posicion (ep)
        if qtil <= self.cep[1]: #asegura maximo a la izquierda de la variable error
            for j in range(2, len(mfe)):
                mfe[j] = 0.0
            mfe[1] = 1.0
            e_cont += 1
            e_int = 1;
        else:
            if qtil >= self.cep[len(self.cep) - 1]:
                for j in range(0, len(self.cep) - 1):
                    mfe[j] = 0.0
                mfe[len(mfe)] = 1.0
                e_cont += 1
                e_int = self.ep
            else:
                for i in range(1, self.ep):
                    if qtil <= self.cep[1]:
                        mfe[i] = max(0, 1 + (qtil - self.cep[i]) * self.MI[i])    #calcu
                        if mfe[i] != 0:
                            e_cont += 1
                            e_int = i
                    else:
                        mfe[i] = max(0, self.cep[i] - qtil) * self.MD[i]    #calcula el
                        if mfe[i] != 0:
                            e_cont += 1
                            e_int = i

```

Figura 4.13 Fragmento de la clase FUZZY

El algoritmo completo y el programa de prueba pueden ser consultados en la sección Python FYZZY del apéndice C.

4.4.3 Controlador

El controlador también denominado despachador, es el encargado de coordinar las actividades del sistema, su hardware lo conforma un Connect Port X4 conectado a cuatro RTUs que envían su información al controlador el cual

ejecuta el algoritmo de control pertinente y responde con las acciones de control que debe implementar la RTU en turno.

El programa que ejecuta el controlador debe enviar las respuestas con las señales de control y debe enviar al servidor mediante una petición GET la información reportada por los sensores de las RTUs. El siguiente fragmento muestra las funciones de inicialización del programa.

```
184
185 def programa():
186     global run
187     print "Inicio"
188     sd.bind(("", 0xe8, 0, 0))
189     minuto = datetime.datetime.now().strftime("%M")
190     Seg = datetime.datetime.now().strftime("%S")
191     run = True
192     t = threading.Thread(target=fromXbee)
193     t.start()
194     var = 1
195     while var < 4 :
196         try:
197             nMinuto = datetime.datetime.now().strftime("%M")
198             nSeg = datetime.datetime.now().strftime("%S")
199 
200             if minuto != nMinuto:
201                 minuto = nMinuto
202                 var+=1
203                 #IO servidor
204                 setData()
205 
206             if Seg != nSeg:
207                 #control
208                 continue
209         except:
210             print "Error [main]:", sys.exc_info()[0]
211             pass
212     run = False
213     t.join()
214     print "Fin"
215 
216 if __name__ == '__main__':
217     programa()
```

Figura 4.14 Fragmento del código implementado en el ConnectPort X4

Una de las actividades que debe cumplir el controlador es el registro de las variables en un servidor, por lo que se emplea las funciones mostradas en la Figura 4.15 para generar las peticiones GET que serán enviadas al servidor en las que se incluyen los valores que se pretenden almacenar.

```
def toXbee (destino,trama):
    DESTINATION = (destino,0xe8,0xc105,0x11)
    print DESTINATION
    print trama
    print destino
    sd.sendto(trama,0,DESTINATION)

def fromXbee():
    tipo = {RTU_1: from_RTU_1, RTU_2: from_RTU_2, GPRS: from_GPRS}
    while run:
        try:
            res = ()
            res = sd.recvfrom(255,MSG_DONTWAIT)
            if len(res) == 2:
                trama = res[0]
                info = res[1]
                origen = info[0]
                origen = origen.replace("[", "");
                origen = origen.replace("]", "");
                tipo[origen](trama)
        except:
            time.sleep(1)
            #print "Error:", sys.exc_info()[0]
            pass

def setData():
    try:
        DESTINATION = (GPRS,0xe8,0xc105,0x11)
        datos = "setData?A=DESHIDRATADOR&"
        sd.sendto(datos,0,DESTINATION)
        #datos = "B=sp_FRA,sp_FAR,sp_TAR2,sp_CA,sp_CB,sp_TAG,&"
        datos = "B=%s,%s,%s,%s,%s,%s,&" %(sp_FRA,sp_FAR,sp_TAR2,sp_CA,sp_CB,sp_TAG)
        sd.sendto(datos,0,DESTINATION)
        #datos = "C1=CLA,CLB,FAR1,FAR2,FAR3,TAR1,TAR2,TAR3,HAR1,HAR2,HAR3,&"
        datos = "C1=%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,&" %(CLA,CLB,FAR1,FAR2,FAR3,TAR1,TAR2,TAR3,HAR1,HAR2,HAR3)
        sd.sendto(datos,0,DESTINATION)
        #datos = "C2=VA,VF,VC,IA,IF,IC,PA,PF,PC,FRA,HA,TA,RSO,TAG,&"
        datos = "C2=%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,&" %(VA,VF,VC,IA,IF,IC,PA,PF,PC,FRA,HA,TA,RSO,TAG)
        sd.sendto(datos,0,DESTINATION)
        #datos = "D=CA,CB,FA,FF,CtrlA,CtrlB,RST,B1,B2,B3,"
        datos = "D=%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s," %(CA,CB,FA,FF,CtrlA,CtrlB,RST,B1,B2,B3)
        sd.sendto(datos,0,DESTINATION)
    except:
        print "Error [setData]:", sys.exc_info()[0]
```

Figura 4.15 Fragmento de código: Peticiones GET

4.4.4 Interfaz Asp.Net

Para la visualización del estado del deshidratador así como la programación de los diferentes setpoint fue necesario realizar una interfaz Web (Figura 4.16) que permite consultar el estado actual y mediante control de usuarios, el acceso al panel de control que permite activar o desactivar los diferentes actuadores.

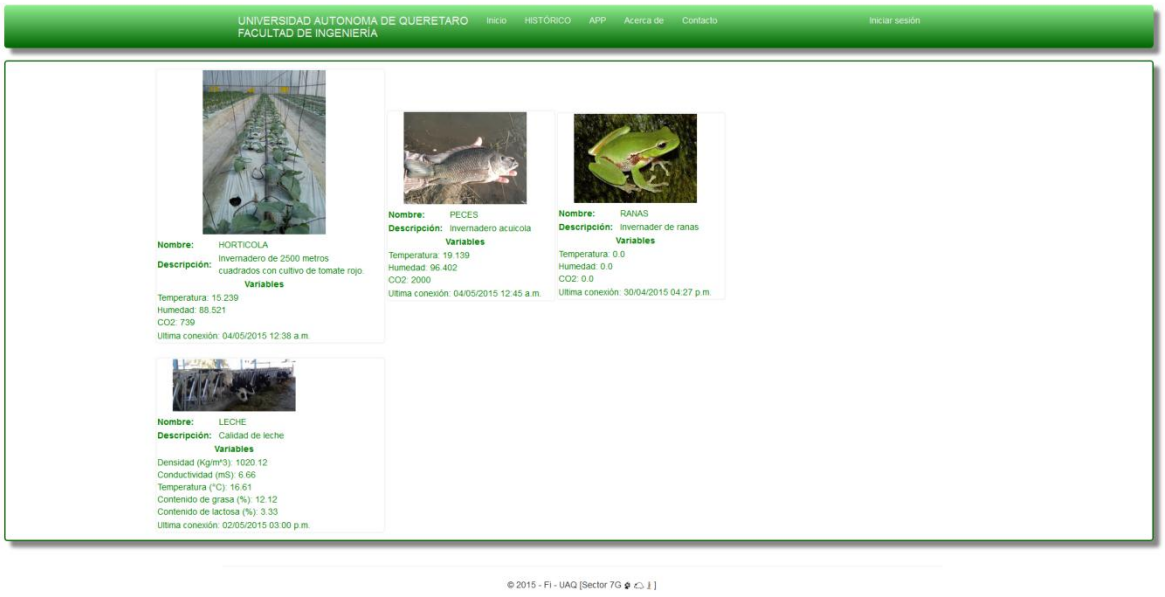

Figura 4.16 Interfaz grafica

La interfaz permite consultar la información de todos los sistemas instalados en el campus Amazcala de la facultad de ingeniería de la UAQ, incluye gráficas y estadísticas básicas relacionadas con los mínimos y máximos presentados por día, además cuenta con un panel para realizar acciones de control (Figura 4.17).

La interfaz cuenta con acceso a la base de datos para realizar consultas de días anteriores (Figura 4.18), la información es presentada en forma de tabla y puede ser descargada en formato .CSV para su futuro análisis.

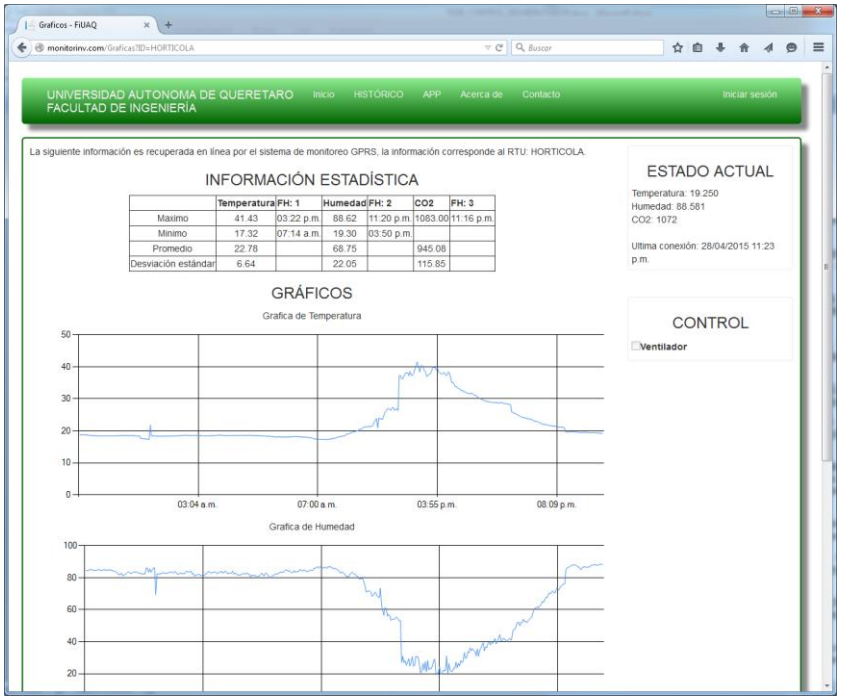

Figura 4.17 Interfaz Gráfica: Información detallada

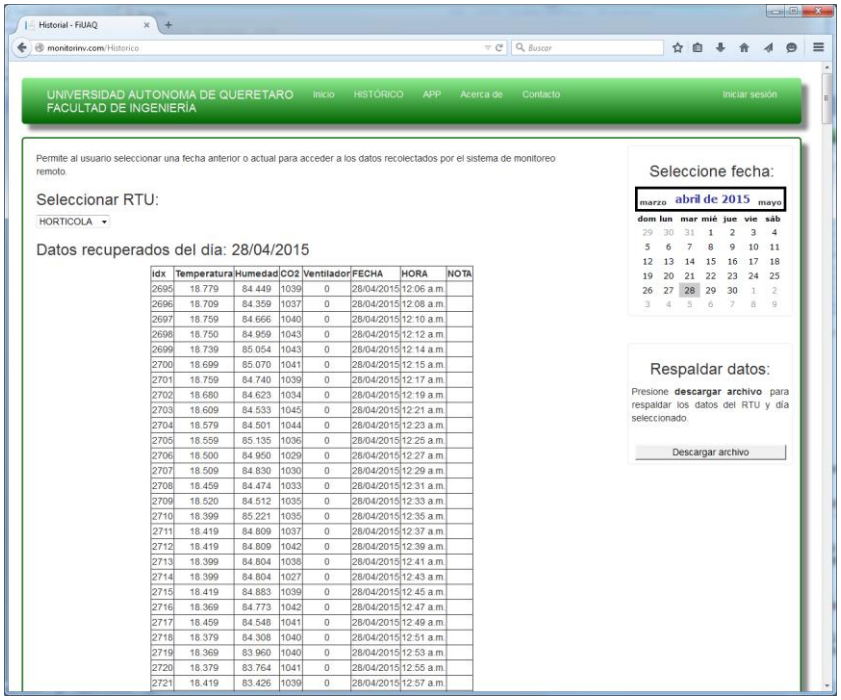

Figura 4.18 Interfaz gráfica: Historial

4.4.5 Pruebas del sistema

El sistema fue simulado utilizando como origen de datos los publicados por la Comisión Nacional del Agua (CONAGUA) para el día 3 de Junio de 2015, los datos fueron enviados al ConnectPort X4 utilizando la interfaz x-ctu y con ello se comprobó la capacidad del ConnectPort para realizar la gestión de la información.

Queda pendiente el utilizar modelos matemáticos de los diferentes componentes para realizar una simulación que permita conocer fuera de lineal el comportamiento del deshidratador. Es necesaria la etapa de simulación previa para no generar pérdidas económicas ocasionadas por posibles fallas durante la deshidratación de 500kg de producto fresco.

A la fecha de impresión, se realizan pruebas de deshidratado para obtener las cinética de deshidratado, con ello se establecerá la respuesta del producto al sistema trabajando a plena capacidad. Terminadas las pruebas se podrá agregar la cinética al modelo de la planta y con ello completar la simulación que permitirá validar las diferentes estrategias para finalmente realizar la prueba que permita comparar ambas estrategias de control aplicadas en un deshidratador industrial.

4.4.6 Pruebas para obtención de cinética de deshidratado

Se realizaron diversas pruebas con la intención de obtener curvas de deshidratado empleando un deshidratador de laboratorio (Figura 4.19) que cuenta con instrumentación para realizar la medición de las variables: peso, temperatura y humedad, además cuenta con control de temperatura y flujo de aire.

Figura 4.19 Deshidratador de Laboratorio

La Figura 4.20 a la Figura 4.24 muestran el proceso que se siguió durante una prueba realizada el sábado 14 de marzo de 2015, en el que se deshidrato 302.2gr de producto fresco (jitomate bola) el cual se sometió a 80 grados centígrados con un flujo de 1 m/s durante 2.5 horas.

Figura 4.20 Rodajas colocadas en charolas

Figura 4.21 Charolas con producto en el deshidratador

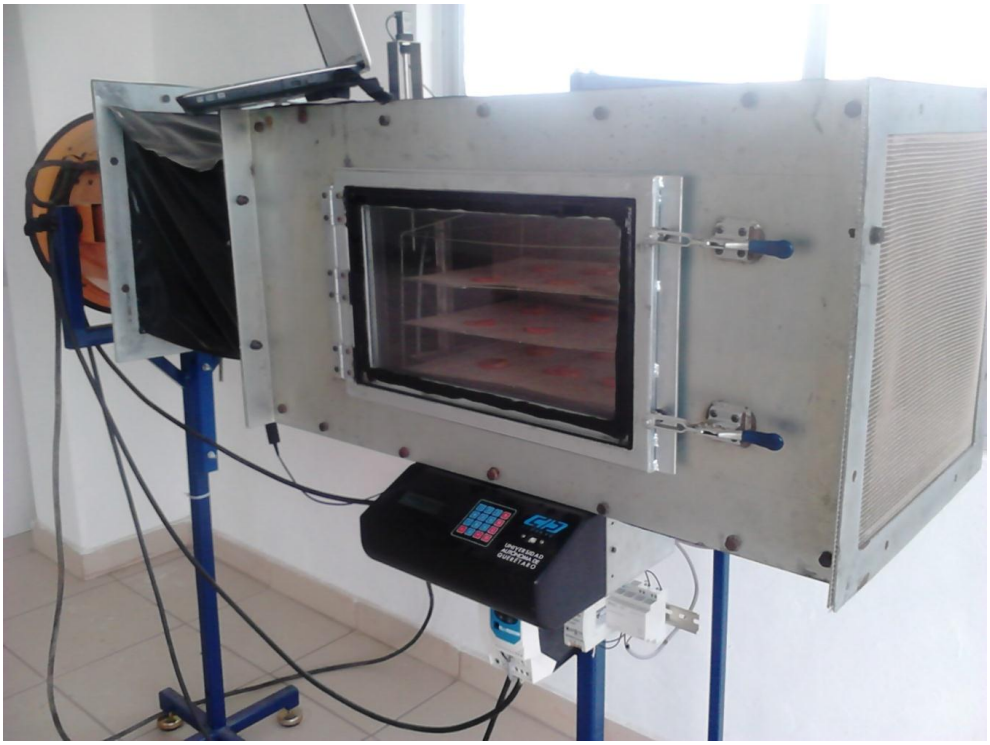

Figura 4.22 Deshidratador operando

Figura 4.23 Muestras deshidratadas

Figura 4.24 Muestra deshidratada (dentro del deshidratador)

La Tabla 4.3 muestra un resumen de los datos obtenidos durante una prueba realizada utilizando el deshidratador de laboratorio, en total se realizaron 7458 registros, la tabla corresponden a los datos muestreados cada 20 minutos.

Tabla 4.3 Resumen de los registros (prueba realizada el 14/3/2015)

# Muestra	Hora	Peso (gr)
0	11:00	302.2
833	11:20	292.6
1837	11:40	291.3
2842	12:00	264.0
3846	12:20	239.5

4850	12:40	221.5
5854	13:00	207.1
6859	13:20	198.1
7458	13:33	198.6

La Figura 4.25 muestra el comportamiento del peso contra tiempo para una muestra de 302.2gr de producto fresco sometida a 90º C y un flujo constante de 1m/s durante 2.5hrs.

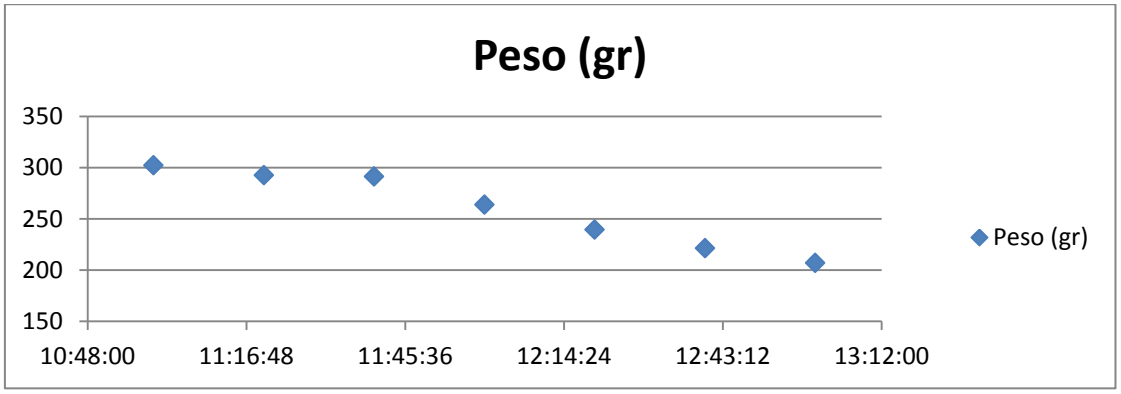

Figura 4.25 Cinemática de deshidratado (90ºC 1m/s 2.5hrs)

5. Referencias

Noriega. P. A. 2014. Diseño de controladores difusos, Facultad de Ingenieria, Universidad Autonoma de Queretaro.

Becker R., Garwon M., Gutknecht C., Barwolff G. y King R. 2005. Robust control of separated shear flows in simulation and experiment, Journal of Process Control.

Bhushan B. and Singh R. 2012. Thermal and thermohydraulic performance of roughened solar air heater having protruded absorber plate. Journal of Solar Energy.

Cattafesta L. N., QiSong, Williams D. R., Rowley C. W. and , Alvi F. S. 2008. Activecontrolofflow-inducedcavityoscillations. Journal Progressin Aerospace Sciences

Curcio S., Aversa M., Calabro V and Iorio G. 2008. Simulation of food drying: FEM analysis and experimental validation. Journal of Food Engineering.

Inaoka K., Nakamura K., Senda M. 2004. Heat transfer control of a backward-facing step flow in a duct by means of miniature electromagnetic actuators. Journal of Heat and Fluid Flow

Leiva-Cochachin Andres M. and Chalco-Mendoza Fredy, 2013. Embedding Linux with Ability to Analyze Network Traffic on a Development Board based on FPGA. International Journal of Computer Applications (0975 – 8887) Volume 77 – No.17, September 2013

Montufar, V. F. 2012. Diseño de un controlador por lógica difusa aplicando a un robot manipulador. Universidad Autónoma de Querétaro.

Nagaya K, Li Y., Jin Z., Fukumuro M., Ando Y and Akaishi A. 2006. Low-temperature desiccant-based food drying system with airflow and temperature control. Journal of Food Engineering

Naghib A. L. and Hangan H. 2010. Active flow control for reduction of fluctuating aerodynamic forces of a blunt trailing edge profiled body. Journal of Heat and Fluid Flow

Perez R. E. 2011. Recurrent Neural Controller for a Continuous Ohmic Heather. Instituto politécnico Nacional, Centro de investigación en ciencia aplicada y tecnología avanzada.

Quiroga A. L. A. 1994. Learning Fuzzy Logic From Examples. Ohio University, Faculty of the College of Engineering and Technology

Sharma VK, Colangelo A, Spagna G. Experimental investigation of different solar dryers suitable for fruit and vegetable drying. Department Enerzia, Divisione Ingegneria Sperimentale, Italy, Drying 1994;94;879-86.

Yu Du, 2005, A SOC Implementation of Ogg Audio Player using MicroBlaze

6. Apéndice

6.1 Apéndice A

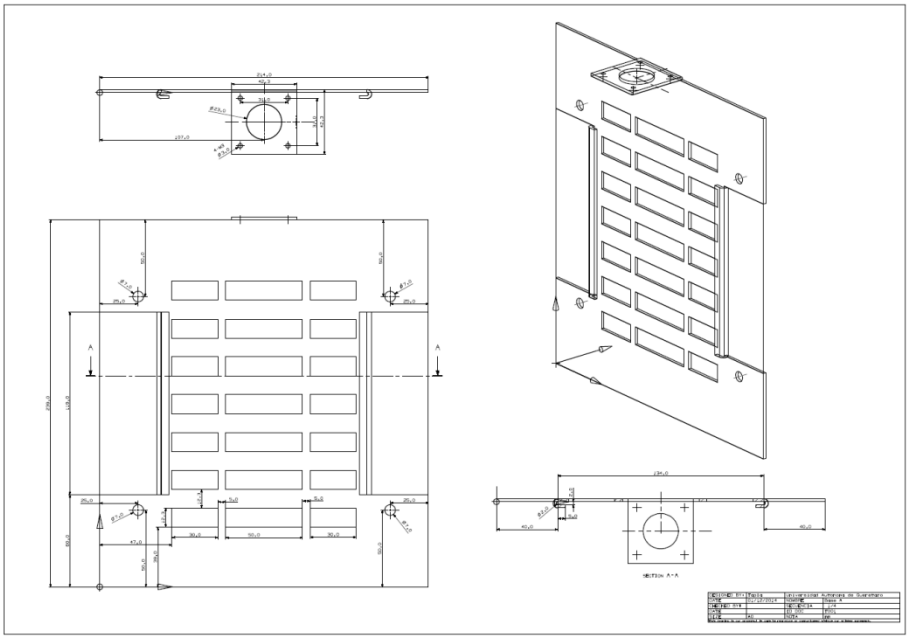

Figura 6.1 Base A

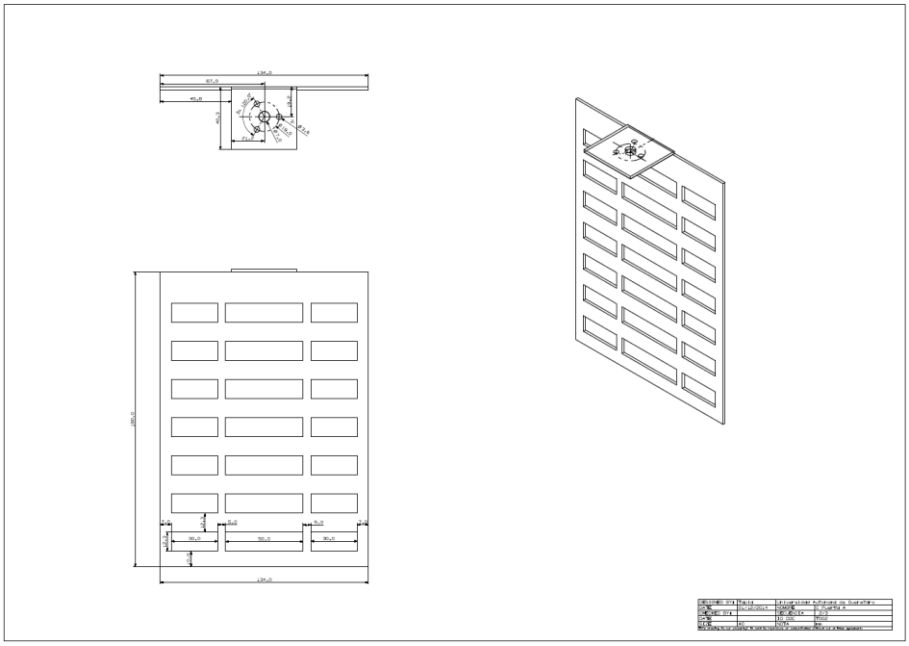

Figura 6.2 Compuerta A

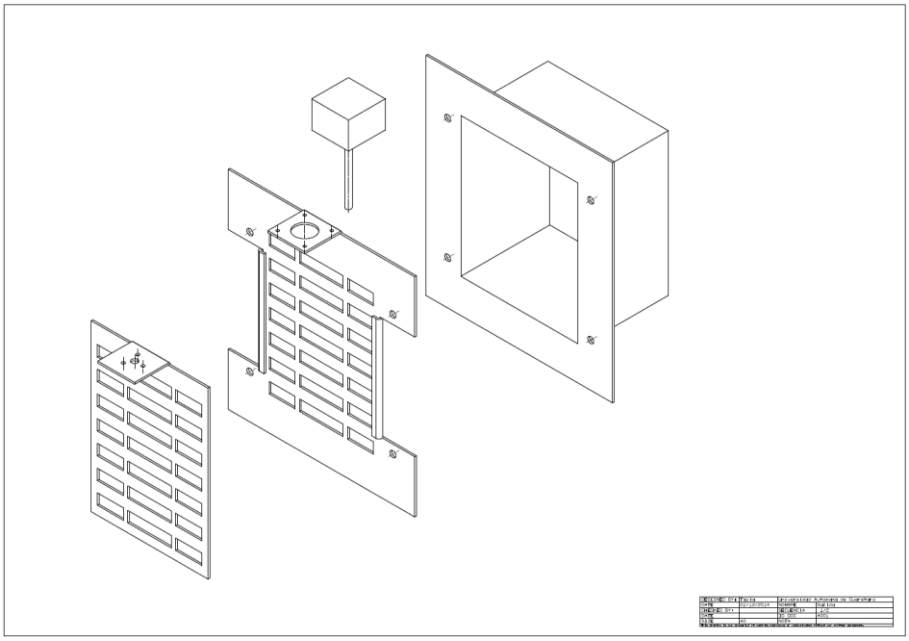

Figura 6.3 Unión Base A + Compuerta A + Motor

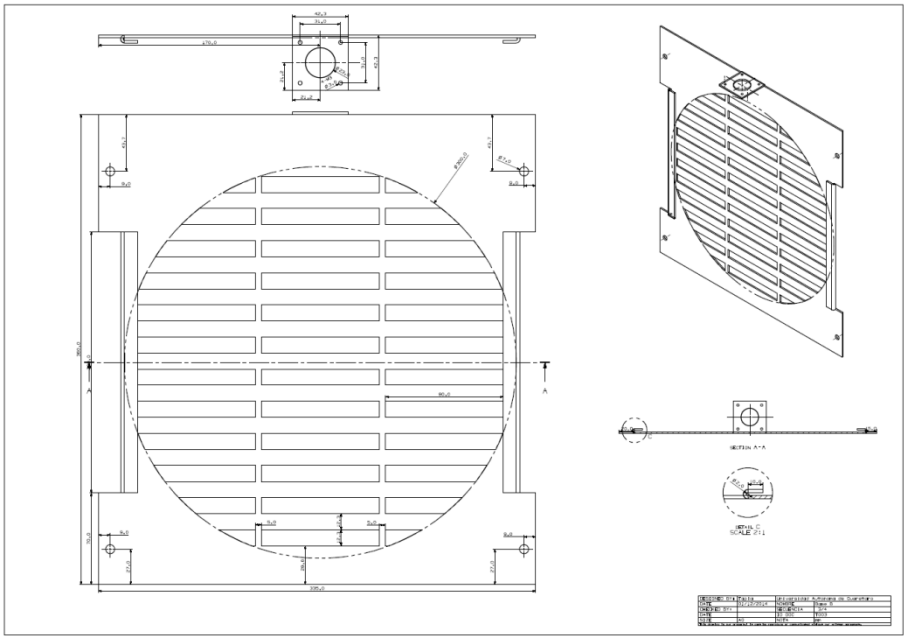

Figura 6.4 Base B

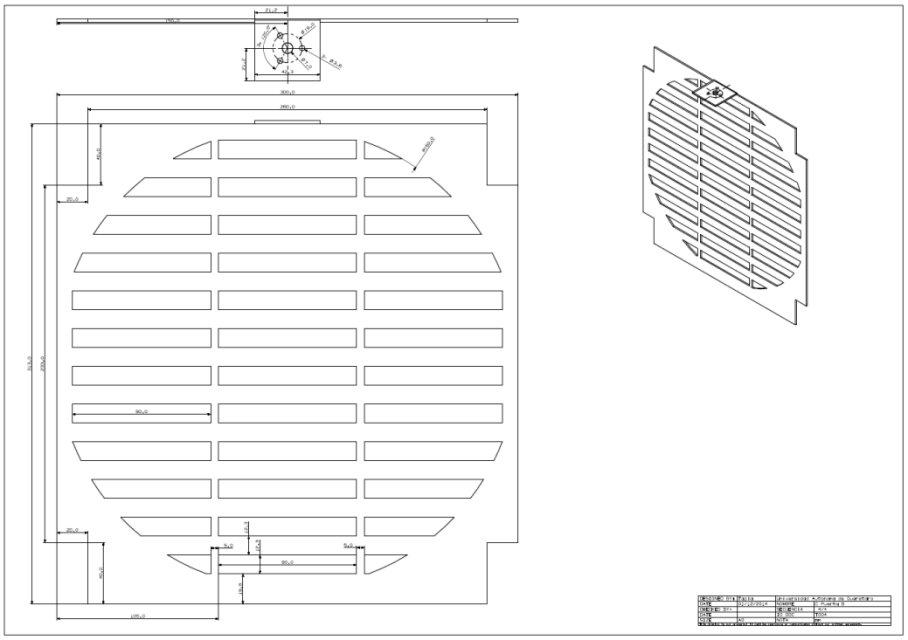

Figura 6.5 Compuerta B

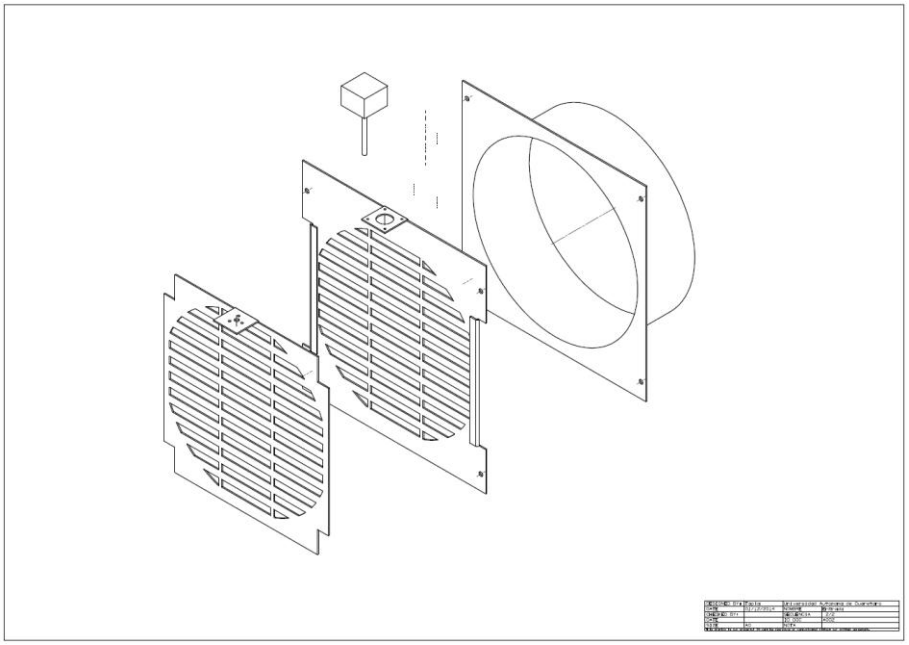

Figura 6.6 Unión Base B + Compuerta B + Motor

6.2 Apéndice B

Instrucciones y notas para uClinux en Virtex 5

1. Instalar Oracle VM VirtualBox 4.3.10 for Windows x86/x64
    a. http://download.virtualbox.org/virtualbox/4.3.10/VirtualBox-4.3.10-92957-Win.exe
2. Montar el sistema CentOS 6.5
    a. Presionar el botón nueva para abrir el asistente “Crear maquina virtual”
        i. Definir el nombre y sistema operativo
            1. Nombre: CentOS
            2. Tipo: Linux
            3. Versión: Red Hat (64 bit)
        ii. Tamaño de la memoria
            1. Memoria RAM mínima recomendada: 1GB
        iii. Unidad de disco duro
            1. Seleccionar “Crear un disco duro virtual ahora”
        iv. Tipo de archivo de unidad de disco duro
            1. Seleccionar “VDI (VirtualBox Disk Image)”
        v. Almacenamiento en unidad de disco duro físico
            1. Seleccionar Tamaño fijo
        vi. Ubicación del archivo y tamaño
            1. Nombre del archivo recomendado: CentOS
            2. Tamaño mínimo recomendado: 60GB
        vii. Presionar el botón aceptar para iniciar la creación del disco
    b. Crear un segundo disco duro
        i. Clic en configuración
        ii. En la opción de almacenamiento
        iii. En el árbol de almacenamiento, seleccionar “Agregar disco duro” del controlador SATA
        iv. En la ventana emergente seleccionar “Crear nuevo disco”
        v. En el asistente “Crear unidad de disco”
            1. Tipo de archivo: VDI (VirtualBox Disk Image)
            2. Almacenamiento en unidad de disco duro físico: Tamaño fijo
            3. Ubicación del archivo y tamaño
                a. Nombre recomendado: CentOS_2
                b. Capacidad: 1GB
        vi. Presionar el botón crear para iniciar la creación del disco
    c. Cargar la imagen iso del sistema CentOS
        i. Clic en configuración
        ii. En la opción de almacenamiento
        iii. En el árbol de almacenamiento, seleccionar del controlador IDE el dispositivo de la unidad de CD
            1. En la sección de atributos, clic sobre seleccionar un archivo
            2. Indicar la ruta del archivo CentOS_6.5_Final.iso
    d. Iniciar maquina
    e. En la pantalla inicial del instalador CentOS 6 seleccionar la opción Install or upgrade an existing system
    f. Seleccionar la opción “skip” para evitar la validación de medios
    g. Seleccionar idioma y teclado (recomendado ingles)
    h. Seleccionar “Basic Storage devices” como respuesta a la pregunta sobre los tipos de dispositivos
    i. En la advertencia seleccionar “Yes, discard any data”

j. Configurar: Nombre del equipo, Zona y contraseña del root
k. Tipo de partición
    i. Create custom layout
l. Crear la estructura
    i. Sda1 /boot 200MB Partición estándar tipo ext4 de tamaño fijo y obligar a ser una partición primaria
    ii. Sda2 swap 2000MB(doble de la RAM) Partición estándar tipo swap con tamaño fijo
    iii. Sda3 / ¿?MB(free space) Partición estándar tipo ext4 de tamaño máximo permitido y obligar a ser una partición primaria
    iv. Sdb1 500MB Partición estándar de tamaño fijo
m. Verificar la tabla de resumen y dar clic en siguiente
n. Formatear disco
o. Escribir cambios en el disco
p. Clic en siguiente (no realizar cambios en la pantalla)
q. Seleccionar el tipo de instalación: Desktop
r. Clic en siguiente, al finalizar la instalación debe reiniciar la maquina
s. Al reiniciar, aceptar la licencia, crear nuevo usuario, definir fecha y hora

3. Instalar ISE 14.7
    a. Descomprimir la carpeta
    b. Cambiar a súper usuario
        i. $ su - root
    c. Ejecutar el archivo xsetup para iniciar la instalación del ISE
        i. # sudo sh xsetup
        ii. No instalar los drivers para cable
        iii. No actualizar
    ~~d. Configura la licencia~~
        ~~i. # xlcm~~
    e. Configura las variables de entorno, para ello entra en la carpeta en la que se instalo (/opt/Xilinx/11.1/) y ejecuta la siguiente línea
        i. # source settings32.sh
4. Instalar PetaLinux v12.12
    a. Descomprimir la carpeta
    b. Cambiar a súper usuario
        i. $ su – root
    c. Crear la carpeta contenedora del petalinux
        i. # mkdir /opt/pkg/
    d. Verificar que se cuente con las herramientas necesarias para la instalación
        i. # yum install –y dos2unix
        ii. # yum install –y iproute
        iii. # yum install –y gawk
        iv. # yum install –y gcc
        v. # yum install –y git
        vi. # yum install –y gnutls-devel
        vii. # yum install –y net-tools
        viii. # yum install -y ncurses-libs.i686
        ix. # yum install –y ncurses-devel
        x. # yum install –y tftp-server
        xi. # yum install –y zlib
        xii. # yum install –y zlib-devel
        xiii. # yum install –y flex
        xiv. # yum install –y bison
        xv. # yum install –y libstdc++.i686
        xvi. # yum install -y glibc.i686

xvii. # yum install -y libselinux.i686
xviii. # yum install -y gcc-c++

e. Para activar el servicio TFTP, ejecutar la siguiente línea
    i. # gedit /etc/xinetd.d/tftp
f. Modificar el archivo
    i. Line 13: server_args = -s /tftpboot (/var/lib/tftpboot)
    ii. Line 14: disable = no
g. Crear la carpeta y modificar los privilegios /tftpboot
    i. # mkdir /tftpboot
    ii. # chmod 777 /tftpboot
h. Reiniciar el servicio para aplicar los cambios
    i. # service xinetd restart
i. Para instalar usar el siguiente comando dentro de la carpeta descomprimida
    i. 3 chmod 777 petalinux-v2013.10-final-intall.run
    ii. # ./petalinux-v2013.10-final-intall.run /opt/pkg

5. Configurar las variables de entorno
    a. Ejecutar los comandos
        i. $ source /opt/Xilinx/14.7/ISE_DS/settings64.sh
        ii. $ source /opt/pkg/petalinux-v2013.10-final/settings.sh
    b. Utilizar el comando “set” para conocer todas las variables de entorno. Confirmar que es correcto y se encuentran las variables XILINX y XILINX_EDK
        i. XILINX=<Path to the installed Xilinx ISE>
        ii. XILINX_EDK=<Path to the installed Xilinx EDK>
        iii. PETALINUX=<Path to the PetaLinux root directory>
        iv. PETALINUX_VER=<PetaLinux Version>
6. Descargar el BSB de la tarjeta Genesys de Digilent
    a. El documento está disponible desde la pagina del fabricante
        i. http://www.digilentinc.com/Products/Detail.cfm?Prod=GENESYS
    b. Utilizar el doc# DSD-0000275
        i. http://www.digilentinc.com/Data/Products/GENESYS/Digilent_Genesys_BSB_Support_V_1_03.zip
    c. Cambiar a súper usuario
        i. $ su - root
    d. Descomprimir el archivo Diligen_Genesys BSB Support
    e. Localizar la carpeta “/opt/Xilinx/14.7/ISE_DS/EDK/board/”
    f. Cambiar los privilegios de la carpeta board para que sea editable
        i. # chmod 777 board
    g. La carpeta que tiene que ser colocada en la carpeta board es la nombrada “Digilent” que está en el zip dentro de la carpeta lib.
    h. No continuar como súper usuario
7. Instalar cKermit
    a. Cambiar a súper usuario
        i. $ su - root
    b. Descargar ckermit-9.0.301-1.el6.x86_64.rpm desde la dirección
        i. http://dl.fedoraproject.org/pub/epel/6/x86_64/
    c. Registrar el rpm
        i. # rpm –Uvh ckermit-9.0.301-1.el6.x86_64.rpm
    d. Instalar ckermit
        i. # yum install ckermit
8. Configurar Kermit
    a. Cambiar a super usuario
        i. $ su - root
    b. Abrir o crear el archive kermrc

i. # gedit ~/.kermrc

c. En el editor agregar el siguiente texto (no colocar el index)

```
 i. set line /dev/ttyUSB0
 ii. set speed 115200
 iii. set carrier-watch off
 iv. set handshake none
 v. set flow-control none
 vi. robust
 vii. set file type bin
 viii. set file name lit
 ix. set rec pack 1000
 x. set send pack 1000
 xi. set key \127 \8
 xii. set key \8 \127
 xiii. set window 5
```

d. la primera línea corresponde al serial que fue conectado para determinar el tty usado recurrir a la dirección /dev/serial/ para identificar por el id al dispositivo el cual debe estar vinculado al tty correspondiente (para ver los detalles usar la iusntruccion ll)
   i. $ cd /dev/serial/by-id
   ii. $ ll

9. Configurar minicom
   a. Cambiar a super usuario
      i. $ su - root
   b. Instalar minicom
      i. # yum install minicom
   c. Configurar minicom
      i. # minicom –s
      ii. Seleccionar “serial port setup”
      iii. Modificar serial device: /dev/ttyUSB0
      iv. salir y seleccionar “Modem and dialing”, colocar “A –init string” y “B – reset string” como vacios, para ello presiona enter
      v. Salir y guardar seleccionando la opción “sabe setup as dfl”
   d. Conectar a minicom
      i. # minicom
   e. Desconectar
      i. “Control+A” y después “Q”
10. Instalar el cable de Xilinx
   a. Conectar por el Puerto “Xilinx USB” y encender la tarjeta de desarrollo
      i. El foco debe estar en la máquina virtual
   b. Cambiar a súper usuario
      i. $ su - root
   c. Descargar libusb-devel
      i. # yum install libusb-devel
   d. Descargar fxload
      i. Crear el archivo de configuración para el repositorio
         1. # gedit /etc/yum.repos.d/linuxtech.repo
      ii. Modificar el contenido agregando las siguientes líneas (sin los números de línea)

```
1.[linuxtech]
2.name=LinuxTECH
3.baseurl=http://pkgrepo.linuxtech.net/el6/release/
4.enabled=1
5.gpgcheck=1
6.gpgkey=http://pkgrepo.linuxtech.net/el6/release/RPM-GPG-
   KEY-LinuxTECH.NET
```

iii. ejecutar el comando
1. # yum install -y fxload

~~e. Descargar el driver usb-driver-HEAD-2d19c7c.tar.gz~~
~~i. http://git.zerfleddert.de/cgi-bin/gitweb.cgi/usb-driver?a=snapshot;h=HEAD;sf=tgz~~

f. Instalar el driver
i. Verificar que se cuenta con el paquete de desarrollo del usb
1. # rpm –qa | grep libusb-devel
~~ii. Descomprimir y crear el driver~~
~~1. # tar –xzvf usb-driver-HEAD-2d19c7c.tar.gz~~
~~2. # cd usb-driver-HEAD-2d19c7c~~
~~3. # make~~
~~iii. Verificar la creacion del archivo "libusb-driver.so"~~
iv. Instalar el driver
1. # cd /opt/Xilinx/14.7/ISE_DS/common/bin/lin64/install_script/install_drives/linux_drivers/pcusb
2. # ./setup_pcusb
v. Verificar la existencia del dispositivo
1. # lsusb
2. Bus 001 Device 003: ID 03fd:000d Xilinx , Inx.
vi. Copiar el archivo xusbdfwu.rules al directorio /etc/udev/rules.d
1. # cp xusbdfwu.rules /etc/udev/rules.d/
2. Sobrescribir el existente
vii. Abrir el archivo
1. # gedit /etc/udev/rules.d/xusbfwu.rules
viii. Agregar la siguiente linea al archivo
1. SUBSYSTEMS==”usb”, ACTION==”add”, ATTRS{idVendor}=”03fd”, ATTRS{idProduct}=”0008″, MODE=”666″
ix. Ejecutar el siguiente comando
1. # sed -i -e ‘s/TEMPNODE/tempnode/’ -e ‘s/SYSFS/ATTRS/g’ -e ‘s/BUS/SUBSYSTEMS/’ /etc/udev/rules.d/xusbdfwu.rules
x. Reiniciar la máquina virtual
xi. Cambiar a súper usuario
1. $ su – root
xii. Actualizar los archivos .hex
1. # /sbin/fxload –v –t fx2 –I /opt/Xilinx/11.1/ISE_DS/common/bin/lin64/install_script/install_drives/linux_drivers/pcusb/xusbdfwu.hex –D /proc/bus/usb/002/002
2. NOTA: los dos últimos valores son la dirección, para obtenerla usar lsusb
xiii. Desconectar y reconectar la tarjeta
xiv. No continuar como super ususario

~~g. Prepara la vaiables de entrono considerando las variables nuevas variables~~
~~i. $ source /opt/Xilinx/11.1/ISE/settings64.sh~~
~~ii. $ source /opt/Xilinx/11.1/EDK/settings64.sh~~
~~iii. $ source /opt/Xilinx/11.1/ChipScope/settings64.sh~~
~~iv. $ export LD_PRELOAD="/path/libusb-driver.so"~~
~~v. $ export LD_LIBRARY_PATH="/usr/local/bin:$LD_LIBRARY_PATH"~~
~~vi. $ export XIL_IMPACT_USE_LIBUSB=1~~
~~vii. $ export DISPLAY=:0~~

11. Crear la plataforma para el sistema
a. Verificar las variables de entorno
i. $ set
b. Iniciar el Xilinx Plataform Studio (XPS),para iniciar la GUI utilizar el comando xps

i. $ xps

c. Seleccionar “Create New Project Using Base System Builder” en la ventana de inicio.
d. Crear y seleccionar como carpeta de destino del proyecto la carpeta ~/Xilinx/Projects/XUP-MicroBlaze-project_v00
e. Selecciona como “Interconnect Type”
    i. PLB System
f. Direccionar “Set Project peripheral repositories” a la carpeta “/opt/pkg/petalinux-v2013.10-final/components/edk-user-repository/”
g. En la pantalla de bienvenida seleccionar “I would like to create a new design”
h. En la pantalla board selccionar “I would like to create a system for the dollowing development board”
    i. Board Vendor: Digilent
    ii. Board Name: “Digilent Genesys System Board”
    iii. Board revision: C
i. En la pantalla System seleccionar “single-processor system”
j. En la pantalla Processor configurar como sigue
    i. Reference clock frequency: 100 MHz
    ii. Processor type: MicroBlaze
    iii. Syste, clock frequency: 125 MHz
    iv. Local Memory: 8KB
    v. Debug interface: On-Chip HW Debug Module
    vi. Sin “Enable Floating Point unit”
k. En la pantalla Peripheral realizar los siguientes cambios
    i. Seleccionar “Hard_Ethernet_MAC”, activar “Use DMA” y “Use Interrupt”
    ii. Seleccionar “RS232_Uart_0”, configuración:
        1. RS232_Uart_0: xps_uartlite
        2. Baud rate 115200
        3. Data Bits: 8
        4. Parity: None
        5. con “use interrupt”
    iii. Remover “RS232_Uart_1”
    iv. Agregar un timer (requerido por el SO), para ello se selecciona de la columna izquierda el periférico “xps_timer”, usando el botón “add” se agrega a los periféricos del procesador, la configuración es:
        1. Core: xps_timer
        2. Count width: 32
        3. Configure mode: “two timers are present”
        4. Use interrupt: checked
l. En la pantalla Cache
    i. Activar Instruction cache
        1. Instruction cache size: 2 KB
        2. Instruction cache memory: DDR2_SDRAM
    ii. Activar Data cache
        1. Data cache size: 2 KB
        2. Data cache memory: DDR2_SDRAM
m. Verificar la información en la ventana “Summary”, para finalmente presionar el botón “Finish”
n. “The next step”: “Start using plataform studio”

12. Crear el sistema
    a. Modificar el procesador
        i. Dentro de la pestaña “System assembly view”, localizar el procesador “microblaze_0” y seleccionar “Configure IP…” del menú emergente.
        ii. En la ventana, seleccionar la configuración “Linux with MMU”
    b. Modificar el Software

i. Localizar el botón “Export Hardware Design to SDK”
ii. En la ventana emergente, seleccionar “Export & launch SDK” (incluir bitstream y BMM files)
    1. Nota: al terminar de sintetizar el proyecto se abrirá el SDK, las siguientes instrucciones son respecto a la SDK
iii. Seleccionar el workspace en el directorio ~/Xilinx/Projects/XUP-MicroBlaze-project_v00
iv. Agregar el repositorio de petalinux
    1. XilinxTools -> Repositories
    2. Local repsoitories – New..
    3. Localizar la carpeta /opt/pkg/petalinux-v2013.10-final/components/edk_user_repository
    4. Reescan repositories
v. Ejecutar el asistente para crear un nuevo FS-boot
    1. File -> New -> Project -> Xilinx -> Aplication Project
    2. Project name: fs-boot_0
    3. Use default location
    4. Confirmar que se trata de nuestra paltaforma y procesador
    5. OS Plataform: petalinux
    6. Lenguaje: C
    7. Board support package: create new (fs-boot_0_bsp)
    8. Next
    9. Available templates: FS-BOOT (FS-BOOT by PetaLogix)
    10. Finish
    11. Nota: Marca error no importa
vi. Configurar el BSP
    1. Clic derecho sobre fs-boot_0_bsp
    2. Seleccionar “Board support package settings”
    3. Overview
        a. OS version: 2.00.a
    4. Petalinux
        a. stdin: RD232_Uart_0
        b. stdout: RD232_Uart_0
        c. main_memory: DDR2_SDRAM
        d. flash_memory: FLASH
        e. lmb_memory: dlmb_cntlr
        f. gpio: LEDs_8bit
        g. Ethernet: Hard_Ethernet_MAC
        h. Timer: xps_timer_0
vii. Configurar el fs-boot
    1. Clic derecho sobre fs-boot_0
    2. Seleccionar c/C++ Build settings
    3. Selccionar “Settings” de “C/C++ Build”
    4. Colocar [All configurtion] en la opción configuración
    5. Seleccionar “Symbols” de “MicroBlaze gcc compiler”
    6. Presionar el botón ADD del cuadro “Define symbols (-D)”
    7. Como valor colocar la cadena “CONFIG_FS_BOOT_OFFSET=0xB00000” sin las comillas
    8. Seleccionar “Miscellaneous” de “MicroBlaze gcc linker”
    9. En el cuadro de texto Linker flags colocar la sigueonte cadena sin comillas “-Zxl-mode-novectors” (deshabilita las interrupciones dentro del bootloader, microblaxe no las requiere)
    10. Apply

11. OK

viii. Activar el fs-boot_0
    1. Clic derecho sobre fs-boot_0
    2. Seleccionar “Build configurations”
    3. Seleccionar “Relase” dentro de “Set Active”

ix. Finalizar
    1. Clic derecho sobre fs_boot_0
    2. Seleccionar “Build Project”

13. Agregar la nueva plataforma a Petalinux
    a. Ir al directorio en el que se guardara nuestro proyecto
        i. $ cd ~/Xilinx/Projects/XUP-MicroBlaze-project_v00
    b. Utilizar el siguiente comando para crear el proyecto
        i. $ petalinux-create -t project -- template microblaze - n PetaLinux-MicroBlaze_v00
    c. Desde la carpeta /fs-boot_0_bsp ejecutar la siguiente instruccion
        i. $ petalinux-config --get-hw-description -p ~/Xilinx/Projects/XUP-MicroBlaze-project_v00/PetaLinux-MicroBlaze_v00/
14. Configurar el sistema MicroBlaze Linux
    a. Desde el directorio del proyecto (~/Xilinx/Projects/XUP-MicroBlaze-project_v00/PetaLinux-MicroBlaze_v00/) ejecutar las siguientes instrucciones
        i. $ petalinux-config
        ii. Realizar las siguientes modificaciones
        iii. “Host name”y “Product name”
            1. “Digilent-Genesys-XC5VLX50T”
        iv. System Fash type
            1. Parallel flash
        v. Flash Partition table
            1. Partition 1
                a. Name: fpga
                b. Size: 0xB00000
            2. Partition 2
                a. Name: boot
                b. Size: 0x40000
            3. Partition 3
                a. Name: bootenv
                b. Size 0x20000
            4. Partition 4
                a. Name: image
                b. Size: 0xC00000
            5. Partition 5
                a. Name: spare
                b. Size: 0x0
        vi. Salir y guardar
        vii. $ petalinux-config -c kernel
        viii. Salir
        ix. $ petalinux-config -c rootfs
        x. Salir
    b. Crear la imagen
        i. $ petalinux-build
15. Crear el bitstream con BRAM inicializada por “fs-boot”
    a. Crear una carpeta temporal en el proyecto (PetaLinux-MicroBlaze_v00) para realizar las operaciones
        i. $ mkdir config
    b. Localizar y copiar los archivos a la carpta config

i. Archivo: system_bd.bmm
    1. Creador por: ISE/XPS
    2. Carpeta: XUP-MicroBlaze-project_v00/ XUP-MicroBlaze-project_v00_hw_plataform
ii. Archivo: system.bit
    1. Creado por: ISE/XPS
    2. Carpeta: XUP-MicroBlaze-project_v00/XUP-MicroBlaze-project_v00_hw_plataform
iii. Archivo: fs-boot_0.elf
    1. Creado por: SDK
    2. Carpeta: /XUP-MicroBlaze-project_v00/fs-boot_0/Release
iv. Archivo: u-boot.elf
    1. Creado por: Petalinux SDK
    2. Carpeta: /XUP-MicroBlaze-project_v00/PetaLinux-MicroBlaze_v00/images/linux/
v. Archivo: u-boot-s.bin
    1. Creado por: Petalinux SDK
    2. Carpeta: /XUP-MicroBlaze-project_v00/PetaLinux-MicroBlaze_v00/images/linux/
vi. Archivo: image.ub
    1. Creado por: Petalinux SDK
    2. Carpeta: /XUP-MicroBlaze-project_v00/PetaLinux-MicroBlaze_v00/images/linux/

c. Crear el archivo d2m (para configurar el data2mem)
    i. $ gedit d2m
    ii. Agregar el siguiente texto (sin el índice, cuida la indentacion)
        1. #!/bin/sh
        2. data2mem \
        3. -bm system_bd.mm \
        4. -p xc5vlx50t \
        5. -bt system.bit \
        6. -bd fs-boot_0.elf \
        7. tag microblaze_0 \
        8. -o b download.bit
d. Ejecutar el d2m para genera el .bit
    i. $ chmod 777 d2m
    ii. $ ./d2m
    iii.

16. Descargar los archivos al la tarjeta
    a. Descargar el hardware
        i. $ petalinux-boot --jtag --fpga --bitstream config/download.bit -v
    b. Descargar el kernel
        i. $ petlinux-boot --jtag --image images/linux/image.elf -v

6.3 Apéndice C

6.3.1 Python DRNN

Test_DRNN.py

```
'''
Created on 16/12/2014

@author: Tapia
'''

from DRNN import myDRNN

def program():
    SetPoint = [10]
    epocas = 100
    epoca = 0

    #Crear red [SP,entrenamiento,entradas,salidas,entrada_Zs,capa_oculta]
    RN = myDRNN(SetPoint, 0.8, 1, 1, 1, 2)

    #red
    uk_T = [15]
    while epoca < epocas:
        epoca += 1
        uk_T[0] -= 0.2
        RN.goDRNN(uk_T)
        print('---- %i ----' % epoca)
        RN.getInfo()

if __name__ == '__main__':
    program()
```

DRNN.py

```
'''
Created on 17/12/2014

@author: Tapia
'''
from math import exp

class myDRNN(object):
    n = 0.5                         #entrenamiento
    i = 1                           #numero de entradas
    k = i                           #numero de salidas
    i_z = 1                         #numero de retrasos en las entradas
    j = 2                           #numero de neuronas capa oculta
```

```
    Y_r = []                        #lista SetPoint
    I_i = []                        #lista de entradas con retraso
    W_I_ij = []                     #lista de pesos I
    W_I_ij_z1 = []                  #lista de pesos I con retraso
    W_D_j = []                      #lista neuronas capa oculta
    W_D_j_z1 = []                   #lista retraso capa oculta
    W_O_jk = []                     #lista de pesos JK
    W_O_jk_z1 = []                  #lista de pesos JK
    X_j = []                        #lista sigmoide capa oculta
    X_j_z1 = []                     #lista sigmoide capa oculta con retraso
    sg_k = []
    sg_k_z1 = []
    p_ij = []
    p_ij_z1 = []

    e = []                          #lista con errores
    uk = []                         #Salida
    uk_T = []                       #Salida despues del transducer

    def __init__(self,SetPoint,N,I,K,I_Z,J):
        global
n,i,k,i_z,j,Y_r,I_i,W_I_ij,W_I_ij_z1,W_D_j,W_D_j_z1,W_O_jk,W_O_jk_z1,X_j,X_j_z1,
sg_k,sg_k_z1,p_ij,p_ij_z1,e,uk,uk_T
        self.n = N
        self.i = I
        self.k = K
        self.i_z = I_Z
        self.j = J

        #inicializacion
        for _r in range(0,len(SetPoint)):
            self.Y_r.append(SetPoint[_r])        #SP vector entradas
            self.uk.append(0)                    #señal control vector salida
            self.uk_T.append(0)                  #señal despues del transductor
vector salida
            self.e.append((self.Y_r[_r]-self.uk[_r]))      #inicializacion de la
lista de errores
        for x in range(0,self.i):                #inicializacion capa entrada
I_i
            self.I_i.append(0)
            for y in range(0,self.i_z):
                self.I_i.append(0)
        #inicializacion pesos W_I_ij
        self.W_I_ij = [[0 for y in range(self.j)] for y in range(len(self.I_i))]
        self.W_I_ij_z1 = [[0 for y in range(self.j)] for y in
range(len(self.I_i))]
        self.p_ij = [[0 for y in range(self.j)] for y in range(len(self.I_i))]
        self.p_ij_z1 = [[0 for y in range(self.j)] for y in
range(len(self.I_i))]
        #inicializacion capa oculta
        self.W_D_j = [0 for x in range(self.j)]
        self.W_D_j_z1 = [0 for x in range(self.j)]
        self.X_j = [0 for x in range(self.j)]
        self.X_j_z1 = [0 for x in range(self.j)]
```

```
        #inicializacion pesos W_O_jk
        self.W_O_jk = [[0 for y in range(self.k)] for y in range(self.j)]
        self.W_O_jk_z1 = [[0 for y in range(self.k)] for y in range(self.j)]
        self.sg_k = [0 for y in range(self.k)]
        self.sg_k_z1 = [0 for y in range(self.k)]

    def goDRNN(self,UK_T):

        global
n,i,k,i_z,j,Y_r,I_i,W_I_ij,W_I_ij_z1,W_D_j,W_D_j_z1,W_O_jk,W_O_jk_z1,X_j,X_j_z1,
sg_k,sg_k_z1,p_ij,p_ij_z1,e,uk,uk_T
        self.uk_T = UK_T

        #etapa 1
        for _j in range(0,self.j):            #capa oculta
            self.S_j = self.W_D_j[_j]* self.X_j_z1[_j]
            for _i in range(len(self.I_i)):
                self.S_j += self.W_I_ij[_i][_j] * self.I_i[_i]
            self.X_j[_j] = 1 / (1 + exp(-1 * self.S_j))
        for _k in range(0,self.k):            #capa salida
            self.uk[_k] = 0
            for _j in range(0,self.j):
                self.uk[_k] += self.W_O_jk[_j][_k] * self.X_j[_j]

        #etapa 2
        for _j in range(0,self.j):
            for _k in range(0,self.k):
                self.e[_k] = self.Y_r[_k] - self.uk_T[_k]    #errores
                self.W_O_jk[_j][_k] = self.W_O_jk_z1[_j][_k] + self.n *
self.e[_k] * self.X_j[_j]
        for _j in range(0,self.j):
            sumAEWo = 0
            for _k in range(0,self.k):
                self.sg_k[_k] = self.X_j[_j] * (1 - self.X_j[_j]) *
(self.X_j_z1[_j] + self.W_D_j[_j] * self.sg_k_z1[_k])
                sumAEWo += self.sg_k[_k] * self.e[_k] * self.W_O_jk[_j][_k]
            self.W_D_j[_j] = self.W_D_j_z1[_j] + self.n * sumAEWo
        for _i in range(0,self.i):
            for _j in range(0,self.j):
                self.p_ij[_i][_j] = self.X_j[_j] * (1 - self.X_j[_j]) *
(self.I_i[_i] + self.W_D_j[_j] * self.p_ij_z1[_i][_j])
                sumEWo = 0
                for _k in range(0,self.k):
                    sumEWo += self.e[_k] * self.W_O_jk[_j][_k]
                self.W_I_ij[_i][_j] = self.W_I_ij_z1[_i][_j] + self.n *
self.p_ij[_i][_j] * sumEWo

        #etapa 3
        for _i in range(0,len(self.I_i),1+self.i_z):
            for _i_t in range(0,self.i_z):
                self.I_i[_i+1+_i_t] = self.I_i[_i+_i_t]
            self.I_i[_i] = self.e[_i]
        for _j in range(0,self.j):
            self.X_j_z1[_j] = self.X_j[_j]
```

```
            self.W_D_j_z1[_j] = self.W_D_j[_j]
        for _k in range(0,self.k):
            self.sg_k_z1[_k] = self.sg_k[_k]
        for _j in range(0,self.j):
            for _k in range(0,self.k):
                self.W_O_jk_z1[_j][_k] = self.W_O_jk[_j][_k]
        for _i in range(0,self.i):
            for _j in range(0,self.j):
                self.p_ij_z1[_i][_j] = self.p_ij[_i][_j]
                self.W_I_ij_z1[_i][_j] = self.W_I_ij[_i][_j]

    def getInfo(self):
        print('error %s ' %(self.e))
        print('I_i %s' %(self.I_i))
        print('W_D_j %s' %(self.W_D_j))
        print('W_O_jk %s' %(self.W_O_jk))
        print('uk %s' %(self.uk))
```

6.3.2 Python FUZZY

Test_FUZZY.py

```
'''
Created on 17/12/2014

@author: Tapia
'''

from FUZZY import myFUZZY

def programa():
    epocas = 10
    epoca = 0

    min=-1
    max=1
    #SetPoint
    SP = [5]
    #Entrada de datos
    I = [0]
    #Transformación
    I[0] = I[0] * 0.1
    SP[0] = SP[0] * 0.1
    e = [0,0]


    ep=5    #numero de funciones de membresia de la variable linguistica error
de posicion
    ev=5    #numero de funciones de membresia de la variable linguistica error
de velocidad
    #centros de las funciones de membresia de entrada y salida
    cep = [-1,-0.1,0,0.1,1] #vector de centros de las funciones de membresia ep1
```

```
    cev = [-1,-0.1,0,0.1,1] #vector de centros de las funciones de membresia ev1

    #reglas
    reglas = [[1,1,1,1,0],[1,0.1,0.1,0,-1],[1,0.1,0,-0.1,-1],[1,0,-0.1,-0.1,-
1],[0,-1,-1,-1,-1]]

    #distancia de los centros de las funciones de membresia
    MD = [0.9, 0.1, 0.1, 0.9, 0.1]
    MI = [0.1, 0.9, 0.9, 0.1, 0.9]
    MDe = [0.9, 0.1, 0.1, 0.9, 0.1]
    MIe = [0.1, 0.9, 0.9, 0.1, 0.9]

    FZ = myFUZZY(ep, ev, cep, cev, reglas, MD, MI, MDe, MIe)

    while epoca < epocas:
        epoca += 1
        for dato in I:
            if (dato > min and dato < max):
                e[1] = e[0]
                e[0] = I[0] - SP[0]
                de = e[1]
                res = FZ.goFUZZY(e[0], de)
                print('Epoca: %s' % epoca)
                print('Señal: %s' % res)
            else:
                print("Error en los limites")

if __name__ == '__main__':
    programa()
```

FUZZY.py

```
'''
Created on 17/12/2014

@author: Tapia
'''

class myFUZZY(object):

    ep=0  #numero de funciones de membresia de la variable linguistica error de
posicion
    ev=0  #numero de funciones de membresia de la variable linguistica error de
velocidad

    #centros de las funciones de membresia de entrada y salida
    cep = []    #vector de centros de las funciones de membresia ep1
    cev = []    #vector de centros de las funciones de membresia ev1
```

```
    reglas=[[0],[0]]

    #distancia de los centros de las funciones de membresia
    MD = []
    MI = []
    MDe = []
    MIe = []

    def __init__(self,_ep,_ev,_cep,_cev,_reglas,_MD,_MI,_MDe,_MIe):
        global     ep, ev, cep, cev, reglas, MD, MI, MDe, MIe
        self.ep = _ep
        self.ev = _ev
        self.cep = _cep
        self.cev = _cev
        self.reglas = _reglas
        self.MD = _MD
        self.MI = _MI
        self.MDe = _MDe
        self.MIe = _MIe

    def goFUZZY(self, qtil, qptil):
        global cep, cev

        #registros para almacenar las FM activas
        e_cont = 0
        e_int = 0
        de_cont = 0
        de_int = 0
        #numerador y denominador del defuzificador
        num = 0.0
        den = 0.0
        #variables para almacenar grado de membresia
        mfe = [0.0 for y in range(self.ep)]
        mfc = [0.0 for y in range(self.ep)]
        #guarda el resultado de la inferencia
        prem = 0.0
        #salida del defuzificador
        ui = 0.0

        #-----------------------fusificacion----------------------------

        #para la variale linguistica error de posicion (ep)
        if qtil <= self.cep[1]: #asegura maximo a la izquierda de la variable
error
            for j in range(2, len(mfe)):
                mfe[j] = 0.0
            mfe[1] = 1.0
            e_cont += 1
            e_int = 1;
        else:
            if qtil >= self.cep[len(self.cep) - 1]:
                for j in range(0, len(self.cep) - 1):
                    mfe[j] = 0.0
                mfe[len(mfe)] = 1.0
```

```
                e_cont += 1
                e_int = self.ep
            else:
                for i in range(1, self.ep):
                    if qtil <= self.cep[1]:
                        mfe[i] = max(0, 1 + (qtil - self.cep[i]) * self.MI[i])
#calcula el grado de membresia a la izquierda de la FM
                        if mfe[i] != 0:
                            e_cont += 1
                            e_int = i
                    else:
                        mfe[i] = max(0, self.cep[i] - qtil) * self.MD[i]
#calcula el grado de membresia a la izquierda de la FM
                        if mfe[i] != 0:
                            e_cont += 1
                            e_int = i


        #para la variable linguistica error de velocidad (ev)
        if qptil <= self.cev[1]:   #asegura maximo a la izquierda de la variable
derivada del error
            for j in range(0,self.ev):
                mfc[j] = 0.0
            mfc[1]=1
            de_cont+=1
            de_int=1
        else:
            if qptil >= self.cev[len(self.cev)-1]: #asegura maximo a la derecha
de la variable derivada del error
                for j in range(len(self.cev) - 1):
                    mfc[j]=0.0
                mfc[len(mfc)]=1
                de_cont+=1
                de_int=ev
            else:
                for i in range(1,self.ev):
                    if qptil < self.cev[i]:
                        mfc[i] = max(0.0, 1 + (qptil - self.cev[i]) *
self.MIe[i])    #calcula grado de membresia a la izquierda de la FM
                        if mfc[i] != 0:
                            de_cont+=1
                            de_int=i
                    else:
                        mfc[i] = max(0.0, 1 + (self.cev[i] - qptil) *
self.MDe[i])    #calcula grado de membresia a la derecha de la FM
                        if mfc[i] != 0:
                            de_cont+=1
                            de_int=i

        #inferencia del minimo y defuzificacion
        fun = e_int - e_cont + 1
        fund = de_int - de_cont + 1
        for m in range(fun,e_int+1):
            for n in range(fund,de_int+1):
```

```
            prem = min(mfe[m], mfc[n])

            #metodo de defuzificacion por promedio de centros
            num += (self.reglas[n][m] * prem)
            den += prem

    if den != 0:
        ui=num/den
    else:
        print("Error en FUZZY")

    return ui
```