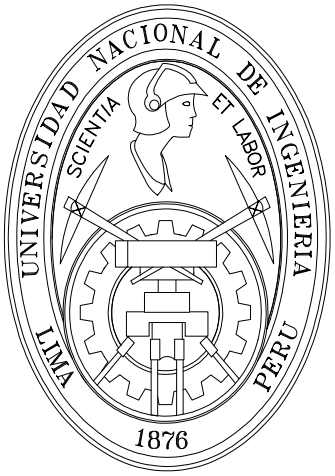
UNIVERSIDAD NACIONAL INGENIERIA

FACULTAD INGENIERÍA ELÉCTRICA Y ELECTRÓNICA



DISEÑO E IMPLEMENTACIÓN DE SISTEMA DE CONTROL DE TENSADO Y ALINEAMIENTO PARA UNA MÁQUINA PROCESADORA DE PAPEL BASADO EN REDES NEURONALES ARTIFICIALES Y CONTROL PID.

TESIS

PARA OPTAR EL GRADO DE MAESTRO EN CIENCIAS

MENCIÓN EN AUTOMÁTICA E INSTRUMENTACIÓN

PRESENTADA POR:

FERNANDO VÁSQUEZ CHOQUEHUANCA

LIMA-PERÚ

2007

UNIVERSIDAD NACIONAL INGENIERIA
Facultad Ingeniería Eléctrica y Electrónica
"Diseño e implementación de sistema de control de tensado y alineamiento para una máquina procesadora de papel basado en redes neuronales artificiales y control PID"
Tesis
Para optar el grado de Maestro en Ciencias
Mención: Automática e Instrumentación
Presentada por:
Fernando Vásquez Choquehuanca
Lima-Perú

Resumen

Se propone un nuevo método de control de tensado para máquinas de procesamiento de papel. Este método se basa en un algoritmo de control PID mejorado por un controlador basado en redes neuronales artificiales feedforward el cual compensa cambios dinámicos y perturbaciones externas. Este método será denominado control PID-RNA en adelante. La unión de ambos métodos de control se realiza sumando las señales de control generadas por ambos algoritmos de control.

La tensión es medida por un sensor el cual provee una señal eléctrica proporcional a la tensión mecánica a un procesador digital de señales (dsPIC30F4011), este procesador determina la señal de control, la cual es enviada a un adaptador de potencia que controlara un determinado actuador pudiendo ser este un motor de corriente alterna, un motor de corriente continua o un freno.

En el caso del control de alineamiento un sensor óptico detecta si el material, en este caso papel, esta alineado correctamente, el actuador que alinea el papel puede ser un motor de corriente continua o un motor paso a paso. El actuador moverá un sistema mecánico que guía el papel.

Se realiza una simulación de la aplicación industrial para obtener mayor realismo además de haberse implementado en una máquina de procesamiento de papel. La notable capacidad de rechazo a perturbaciones del controlador PID-RNA es demostrada y los resultados muestran la efectividad del controlador propuesto.

UNIVERSIDAD NACIONAL INGENIERIA
Facultad Ingeniería Eléctrica y Electrónica
"Diseño e implementación de sistema de control de tensado y alineamiento para una máquina procesadora de papel basado en redes neuronales artificiales y control PID"
Tesis
Para optar el grado de Maestro en Ciencias
Mención: Automática e Instrumentación
Presentada por:
Fernando Vásquez Choquehuanca
Lima-Perú

Abstract

A new control method is proposed for tension control in a paper processing machine. This method is based on an algorithm of PID control improved by a feedforward neural network which compensates the dynamic changes and extern perturbations. This method will be called PID-RNA controller in advance. The union of both methods of control was realized adding the control signals generated by both algorithms of control.

The tension is measured by a tension sensor which provides a electric signal proportional to the mechanical tension to a digital signal processor (dsPIC30F4011), this processor determines the control signal and send it to a power driver that control an actuator that could be a alternating current motor, a direct current motor or a brake.

In the case of the alignment control an optical sensor detects if the material, in this case paper, is aligned correctly, the actuator that aligns the paper could be a direct current motor or a steeper motor. The actuator operates a mechanical system that guides the paper.

A simulation is of the industrial application is realized for get a better realism, in addition it was proved in a paper processing machine. The remarkable disturbance rejection capability of a PID-RNA controller is also demonstrated and the results show the effectiveness of the controller proposed.

Índice general

Resumen ..... **ii**

Abstract ..... **iii**

Agradecimientos ..... **iv**

Índice general ..... **v**

Índice de figuras ..... **viii**

Índice de tablas ..... **xi**

Capitulo 1 ..... **1**

Introducción **1**

1.1. Planteamiento del problema 2

1.2. Antecedentes 2

1.3. Objetivos 6

1.4. Organización de la tesis 7

Capitulo 2 ..... **9**

Modelado de la planta **9**

2.1. Introducción 9

2.2. Descripción del sistema 9

2.3. Modelado dinámico 10

Capitulo 3 ..... **13**

Sistema de Control **13**

3.1. Introducción 13

3.2. Control PID 16

3.2.1. Componentes de controlador PID 17

3.2.2. Métodos de sintonización PID. 21

3.3. Redes Neuronales Artificiales 28

3.3.1. El neurón artificial 31

3.3.2. Arquitectura de las redes neuronales artificiales. 33
3.3.3. Entrenamiento de redes neuronales artificiales. 35
3.3.4. El perceptrón 36
3.3.5. Algoritmo de Retropropagación 38
3.3.6. Controladores Neuronales. 43
3.4. Algoritmo de control PID-RNA 45
Capitulo 4 47
Implementación en tiempo real 47
4.1. Introducción 47
4.2. Actuadores y adaptadores de potencia 49
4.2.1. Motores de corriente alterna AC 49
4.2.2. Variadores de velocidad 51
4.2.3. Motor de corriente continua DC 53
4.2.4. Etapa de potencia de motores DC 55
4.3. Sensores y acondicionamiento de señal 56
4.3.1. Sensor de tensión 56
4.3.2. Sensor de alineamiento. 60
4.4. Programación del controlador 62
4.5. Entrenamiento de la red neuronal 66
4.6. Circuito eléctrico del controlador 68
Capitulo 5 70
Simulaciones 70
5.1. Simulación de la planta 70
5.2. Simulación del controlador PID 71
5.3. Simulación del controlador PID-RNA 72
Capitulo 6 76
Resultados experimentales 76
6.1. Resultados del controlador PID 76

6.2. Resultados del controlador PID-RNA 77

Capitulo 7..... 79

Conclusiones 79

6.1. Conclusiones 79

6.2. Recomendaciones 80

Anexos..... 81

A.1. Código fuente del controlador 81

A.2. Código fuente para el entrenamiento 94

A.3. Programa en Java para la adquisición de datos 95

A.4. Sistema implementado. 97

A.5. Valores de tensión típicos para papel. 99

Bibliografía 100

Índice de figuras

Figura 1.1: Ejemplo de planta de procesamiento de papel. 1
Figura 1.2: Controlador manual. 3
Figura 1.3: Control con retroalimentación. 3
Figura 1.4: Métodos de control de lazo cerrado. 4
Figura 1.5: Controladores de tensado industriales. 5
Figura 1.6: Diagrama de bloques del controlador MCS2000. 5
Figura 1.7: Enrollamientos sin control de alineamiento. 6
Figura 1.8: Problemas causados por exceso o falta de tensión. 7
Figura 2.1: Diagrama de la línea de procesamiento. 10
Figura 2.2: Diagrama simplificado de la línea de procesamiento. 10
Figura 2.3: Diagrama de bloques de una zona de tensión de cinta. 11
Figura 2.4: Diagrama de bloques linealizado de una zona de tensión. 12
Figura 3.1: Diagrama de un sistema de control. 13
Figura 3.2: Diagrama de un sistema de control en el dominio de la frecuencia. 14
Figura 3.3: a) Controlador on-off ideal, b) con zona muerta, c) con histéresis. 15
Figura 3.4: Estructura del lazo de retroalimentación con controlador PID. 17
Figura 3.5: Curva de respuesta del proceso al escalón. 22
Figura 3.6: Respuesta a razón de decaimiento de 1/4. 23
Figura 3.7: Determinación del punto ultimo. 25
Figura 3.9: Salida y entrada de un proceso con un control con relé ideal 26
Figura 3.10: Desarrollo en series de Fourier de señal de control del relé. 27
Figura 3.11: Distintos tipos de relé y sus funciones N(a). 28
Figura 3.12: Modelo de neuronas biológicas. 30
Figura 3.13. Cálculo de la salida de un neurón artificial. 31
Figura 3.14: Funciones de activación mas comunes para redes neuronales. 32
Figura 3.15: Red de propagación multicapa. 33
Figura 3.16: Red de propagación de una capa. 34
Figura 3.17: Red de propagación de Hopfield. 34
Figura 3.18: Entrenamiento Supervisado. 35
Figura 3.19 Entrenamiento no Supervisado 36
Figura 3.20: Modelo de red neuronal del perceptrón. 36

Figura 3.21: Un perceptrón multicapa de tres capas. 38
Figura 3.22: MLP para entrenamiento por retropropagación de neurón simple. 39
Figura 3.23: Notación para red neuronal MLP de varias capas. 41
Figura 3.24: Calculo del error con algoritmo de retropropagación. 42
Figura 3.25: Control inverso directo para sistema de primer orden 44
Figura 3.26: Entrenamiento generalizado del modelo inverso. 44
Figura 3.27: Entrenamiento especializado del modelo inverso. 45
Figura 3.28: Controlador PID-RNA 46
Figura 4.1: Sistema de control de tensado y alineamiento. 47
Figura 4.2: Sistema de control de tensado 48
Figura 4.3: Funcionamiento del control de alineamiento 48
Figura 4.4: Clasificación de motores de corriente alterna 49
Figura 4.5: Características de operación del motor de inducción 50
Figura 4.6: Motores por clasificación NEMA. 51
Figura 4.7: Diagrama de bloques de un variador de velocidad. 52
Figura 4.9: Clasificación de motores de corriente continua. 54
Figura 4.10: Característica torque-velocidad de motores DC. 54
Figura 4.11: Configuración básica de puente H. 55
Figura 4.12: Puente H en circuito integrado L293. 56
Figura 4.13: Sensores de tensión para diferentes materiales. 57
Figura 4.14: Métodos para medición de tensión 58
Figura 4.15: Sensor tipo brazo danzarín. 59
Figura 4.16: Tipos de sensores de alineamiento. 60
Figura 4.17: Sensores de alineamiento. 61
Figura 4.18: Ubicación correcta del sensor de alineamiento. 61
Figura 4.19: Diagrama de tiempos de procesamiento multitarea en el dsPIC. 62
Figura 4.20: Controlador digital. 63
Figura 4.21: Regla de integración rectangular hacia delante 64
Figura 4.22: Diagrama de bloques del algoritmo PID. 65
Figura 4.24: Algoritmo de control PID-RNA implementado. 66
Figura 4.25: Respuesta de la red neuronal inversa sin entrenar. 67
Figura 4.26: Rendimiento de la red neuronal versus épocas. 67
Figura 4.27: Respuesta de la red neuronal inversa entrenada. 68
Figura 4.28: Circuito eléctrico del controlador digital 69

Figura 5.1: Diagrama de bloques en Simulink. 70
Figura 5.2: Respuesta de la planta a un escalón. 71
Figura 5.3: Diagrama de bloques en Simulink del controlador PID..... 71
Figura 5.4: Sub-bloque PID Digital o algoritmo de control..... 72
Figura 5.5: Resultados para el controlador PID. 72
Figura 5.6: Diagrama de bloques Simulink del controlador PID-RNA..... 73
Figura 5.7: Red neuronal de dos capas. 73
Figura 5.8: Primera capa de la red neuronal..... 74
Figura 5.9: Segunda capa de la red neuronal..... 74
Figura 5.10: Resultados para el controlador PID-RNA. 75
Figura 6.1: Respuesta del control PID. 76
Figura 6.2: Respuesta del control PID con filtro. 77
Figura 6.3: Respuesta al impulso del controlador PID-RNA. 78
Figura A.3: Sistema total implementado..... 98

Índice de tablas

Tabla 2.1: Nomenclatura utilizada para el modelo matemático. 11
Tabla 3.1: Parámetros pre-especificados para controladores PID. 22
Tabla 3.2: Efectos de las constantes K_p, K_i y K_d en un proceso. 22
Tabla 4.1: Ventajas y desventajas de una celda de carga..... 57
Tabla 4.2: Ventajas y desventajas de un sensor de brazo danzarín. 58
Tabla 4.3: Ventajas y desventajas de sensores en lazo abierto..... 59

Capitulo 1

Introducción

El guiado y transporte de materiales en forma de bandas continuas ha sido estudiado por muchos años, un material con forma de banda continua se refiere a cualquier material continuo y flexible delgado con forma de banda que es muy largo comparado con su ancho y muy ancho comparado con su espesor. Muchos tipos de materiales son fabricados o procesados en forma de banda continua. El papel, el plástico, la tela y el acero por ejemplo son procesados en forma de banda continua muy frecuentemente, en la presente tesis el material con el cual se desarrollan las pruebas es el papel.

Los requerimientos del control de tensado del papel se han incrementado debido a la demanda de alta velocidad en el equipo de procesamiento. Las plantas de producción actuales tienen bandas continuas de material que se mantienen en constante movimiento por un gran número de maquinas eléctricas que son controladas por diferentes lazos de control. Dichas bandas comúnmente tienen que pasar diferentes etapas de procesamiento y generalmente existen bobinadoras instaladas al inicio y al final de la proceso, como se observa en la figura 1.1.

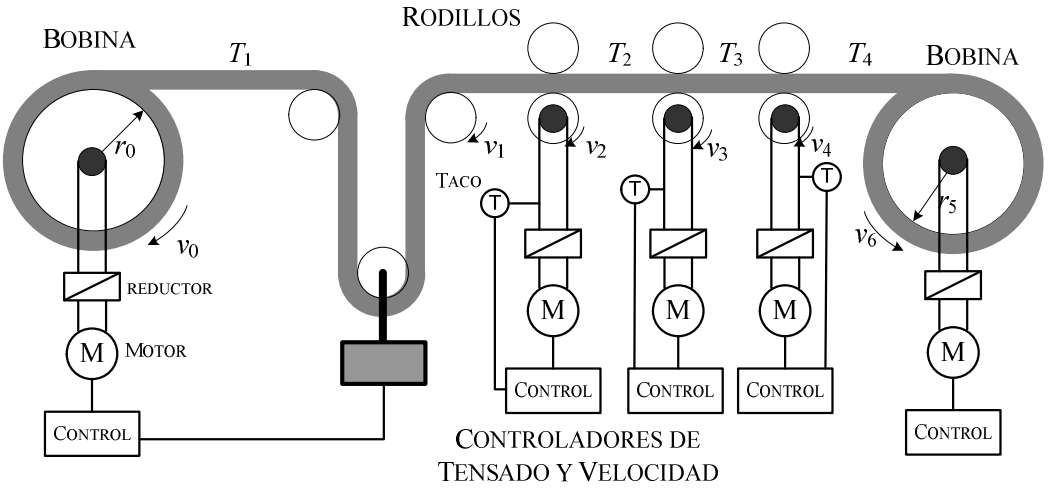


Figura 1.1: Ejemplo de planta de procesamiento de papel

Las plantas de producción con movimiento continuo de material tienen una compleja estructura donde se involucran diferentes problemas eléctricos y mecánicos. En el sistema mostrado en la figura 1.1, el papel es procesado en diferentes etapas. La tensión del papel debe mantenerse en un valor deseado para asegurar la calidad del

producto. Los rodillos son impulsador por motores eléctricos que controlan diferentes variables incluyendo corriente, velocidad y en algunos casos torque.

1.1. Planteamiento del problema

En la presente tesis se abordan dos problemas básicamente, primeramente tenemos el problema de controlar la tensión mecánica en un material determinado, este material en nuestro caso es papel. Este problema es resuelto mediante la utilización de un controlador PID apoyado por un controlador basado en redes neuronales artificiales feedforward, este ultimo tipo de control dedicado a compensar cambios dinámicos y perturbaciones externas.

El segundo problema abordado es controlar el alineamiento para la etapa de rebobinado o enrollado de papel, para de esta manera obtener un rollo de papel totalmente parejo, sin deformaciones en el enrollado ni tampoco arrugas. Para este segundo problema se utiliza un control tipo On-Off. En este caso la magnitud a medir es sencillamente la distancia del papel con respecto de alguna referencia, esta magnitud obviamente se debe mantener constante para mantener alineado el papel. Y en el caso del actuador se implementara un dispositivo mecánico controlado por un motor paso a paso para mover la banda de papel lateralmente, obteniendo así el alineamiento deseado.

1.2. Antecedentes

Para el control de tensión mecánica en la industria actualmente se utilizan controladores manuales, PI (Proporcional Integral) y PID (Proporcional Integral Derivativo), estos tipos de control no solucionan de manera eficiente el control de la tensión en ambientes con donde se genera mucho ruido y disturbios.

Los controles de tensado manuales, donde un operario tiene que estar ajustando la potencia de un determinado actuador (freno o motor) constantemente para obtener un tensado uniforme, es muy complicado alcanzar un tensado uniforme, el operario debe de estar atento todo el tiempo, este operario debe tener mucha experiencia y aun así no alcanzara un tensado uniforme del material. En la figura 1.2 podemos observar el diagrama de bloques de un controlador de este tipo, donde mediante un potenciómetro se ajusta la potencia del actuador.

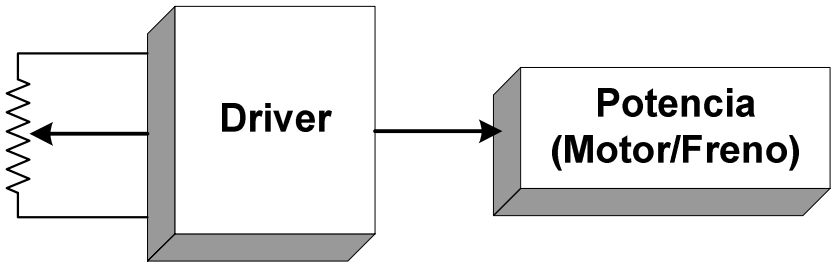


Figura 1.2: Controlador manual.

Como mencionamos anteriormente existen también controladores de tensión industriales con retroalimentación de tipo PI y PID. Estos controladores han demostrado una muy buena eficiencia en aplicaciones industriales, pero cuando existen muchas perturbaciones no logran mantener un funcionamiento óptimo. La ventaja de estos controladores es el bajo costo que tienen. En la figura 1.3 se muestra el diagrama de bloques de un sistema de control de lazo cerrado en este caso Proporcional Integral.

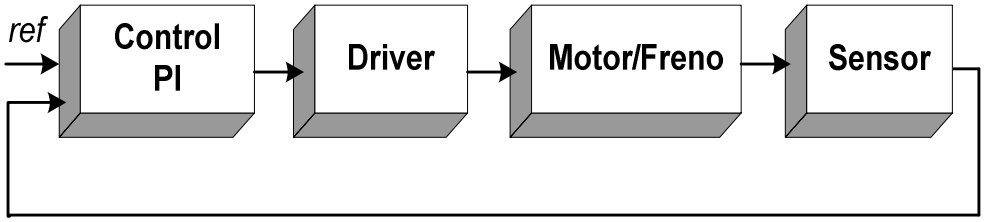


Figura 1.3: Control con retroalimentación.

Así como se tiene varios métodos para controlar el tensado, tenemos también muchas formas de medir la tensión mecánica, esto se puede hacer directamente o indirectamente. Una medición indirecta de tensión por ejemplo se realiza midiendo la dimensión del rollo de donde se desenrolla o enrolla el material, figuras 1.4a y 1.4b, de este valor se calcula mediante fórmulas la fuerza necesaria para mantener constante la tensión, este método no es muy exacto por que no toma en cuenta la tensión real del material. Otra forma de es la medición de la tensión es mediante celdas de carga, figura 1.4c, las cuales miden la tensión directamente. Otra forma de medir tensión de forma directa es mediante la utilización de sensores tipo brazo danzarín (arm dancer) como se muestra en la figura 1.4d.

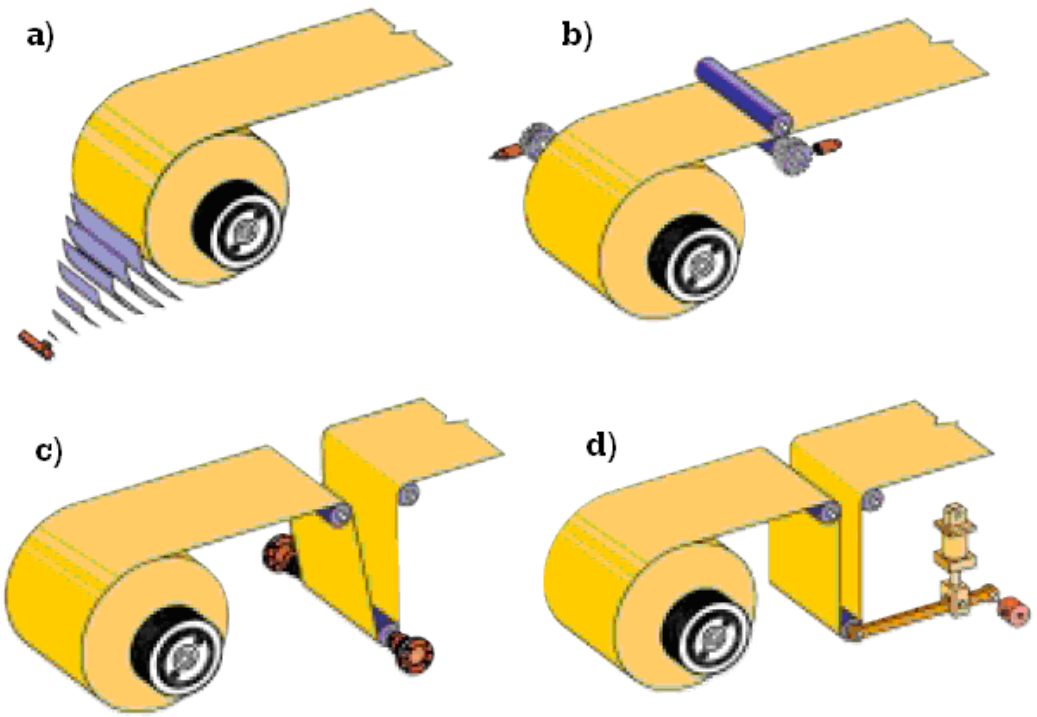


Figura 1.4: Métodos de control de lazo cerrado.

En el mercado existen muchos controladores de tensión por ejemplo la marca Montalvo ofrece diferentes productos para este fin, así podemos observar en la figura 1.5 un controlador de tensión modelo D-3000ce-UW, este controlador utiliza un algoritmo PID el cual se compensa de manera automática el diámetro del rollo. Las salidas del controlador son de 4 a 20 mA y 0 a 10 voltios DC lo que le permite interactuar con una gran variedad de dispositivos para controlar la tensión como frenos y motores.

Existen también otras marcas fabricantes de controladores como Wichita Clutch que tienen una gran variedad de controladores, por ejemplo tenemos el controlador MCS202-E que es un controlador analógico que acepta como entrada sensores tipo dancer únicamente y actúa sobre frenos, su algoritmo de control esta basado en un lazo PID, así también tiene un controlador de tensado por diámetro es el MCS-2000 cuyo diagrama de bloques se muestra en la figura 1.6, donde se puede observar que tiene también un algoritmo PID.

Figura 1.5: Controladores de tensado industriales.

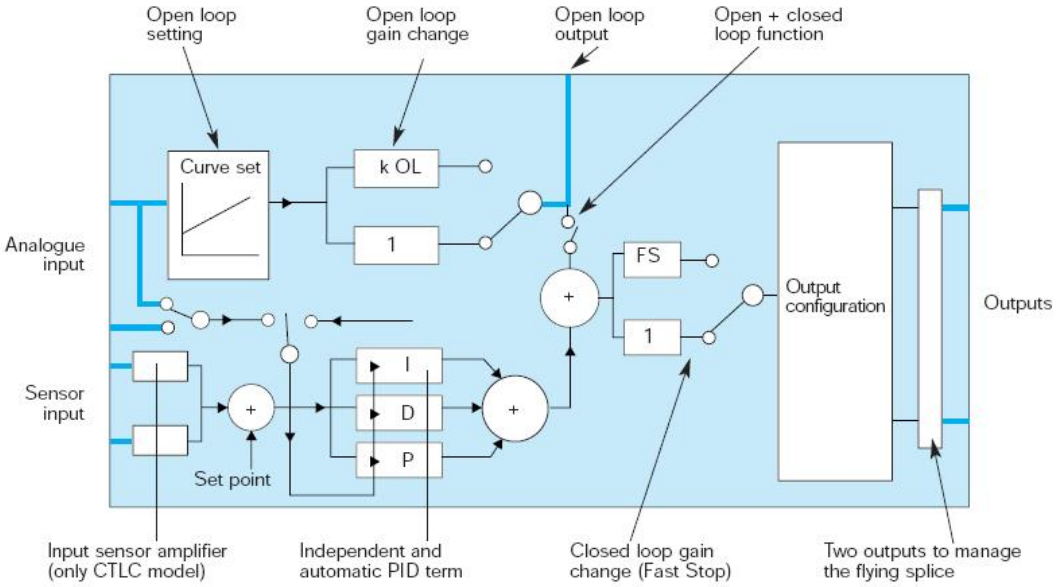


Figura 1.6: Diagrama de bloques del controlador MCS2000.

En la figura 1.6 se puede observar el diagrama de bloques de un controlador industrial PID, la tensión mecánica es adquirida por una entrada de tipo analógica, la cual es convertida a digital, luego es introducida a la entrada de un controlador PID digital, donde se compara esta señal con la referencia (set point). Finalmente es calculado el valor para que se envía a los actuadotes.

1.3. Objetivos

Objetivos Generales.

Diseñar e implementar un controlador de tensión basado en el control PID apoyado en otro controlador basado en redes neuronales artificiales para mantener una buena respuesta bajo un amplio rango de perturbaciones externas. El sistema de control debe proveer la tensión apropiada en una variedad de condiciones, incluyendo, altos niveles de vibración, inconsistencias del producto en el proceso, diámetros variables de rollos y también diferentes velocidades.

Diseñar e implementar un controlador de alineamiento on-off, donde el papel deberá mantenerse alineado en la etapa de enrollado, para lograr un mejor control del tensado y así obtener un producto final (rollo) óptimo es decir consistente y sin defectos como son arrugas o rollos deformes.

Objetivos Específicos.

Mejorar la calidad de los controladores de tensión industriales que actualmente se ofrecen en el mercado, que solamente proporcionan algoritmos P, PI y PID, los cuales no son suficientemente robustos especialmente plantas con gran cantidad de ruido.

Programación de los algoritmos de control PID y control feedforward basado en redes neuronales en el procesador digital de señales dsPIC30F4011.

Eliminar o reducir al mínimo todos los efectos negativos de la falta de un control de tensión y alineamiento como son las roturas, generación de arrugas, variaciones en el espesor, deformaciones en el rollo como las que se muestra en las figura 1.7 y 1.8.

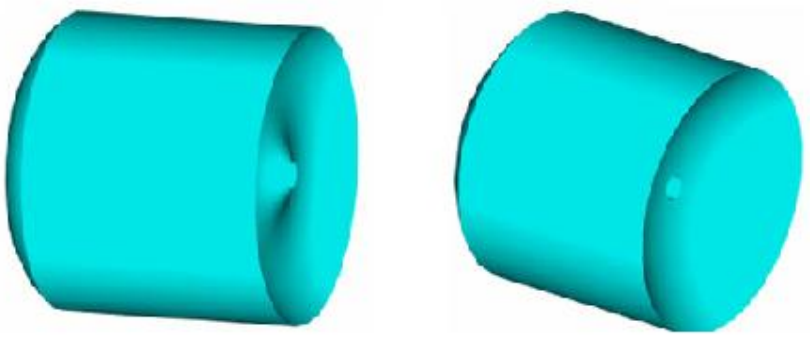


Figura 1.7: Enrollamientos sin control de alineamiento.

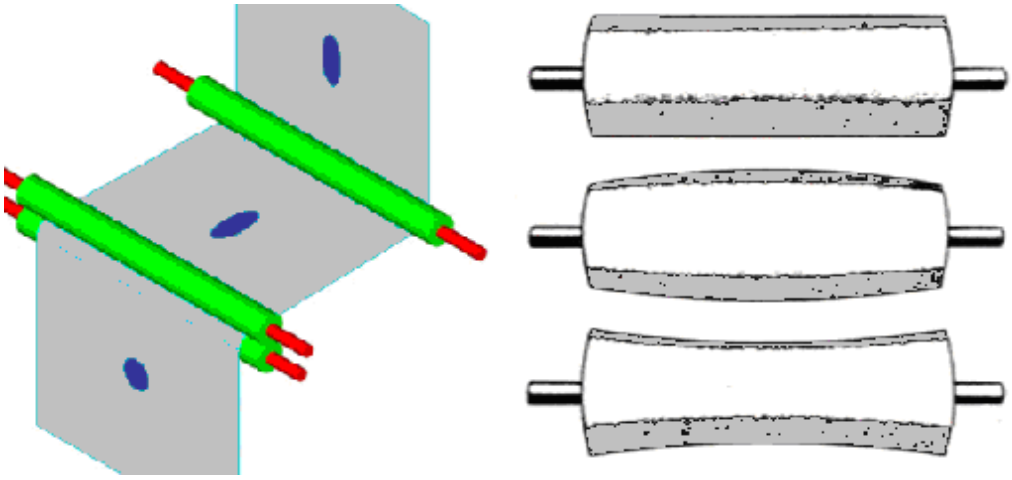


Figura 1.8: Problemas causados por exceso o falta de tensión.

Eliminar o reducir al mínimo todos los efectos negativos de la falta de un control de tensión y alineamiento como son las roturas, generación de arrugas, variaciones en el espesor, deformaciones en el rollo como las que se muestra en las figura 1.7 y 1.8.

1.4. Organización de la tesis

La presente tesis esta distribuida en 7 capítulos y 2 apéndices. Se describe a continuación el contenido de cada uno de ellos.

En el capítulo 2 se explica detalladamente el funcionamiento de una máquina de procesamiento de papel, para esto se presenta una breve introducción dichas máquinas y seguidamente se estudia su modelo dinámico.

En el capitulo 3 se presenta los detalles concernientes a los controladores, primeramente se da una introducción a los diferentes algoritmos de control clásicos como control on-off, P, PI y PID, luego hace una introducción a las redes neuronales artificiales y finalmente se analiza su uso en el control de procesos.

En el capitulo 4 se explica el funcionamiento de los sistemas de control su diseño e implementación en tiempo real. Se definen los actuadores y sensores que se utilizan junto con sus etapas adaptadoras de potencia. Se explican el circuito eléctrico del sistema de control y los programas de los controladores.

En el capitulo 5 se muestran y explican los programas de la simulación del procesos y los controladores, se ilustran los resultados de dichas simulaciones realizadas con ayuda de algunas herramientas de Matlab tales como el Toolbox de Redes Neuronales y Simulink.

En el capitulo 6 se presenta los resultados de las pruebas experimentales realizadas en un maqueta de una máquina de procesamiento de papel. Se analiza las respuestas obtenidas por en control PID, el control neuronal y el control PID-RNA.

En el capitulo 7 se presenta las conclusiones derivadas de este trabajo, también se presentan recomendaciones a tomar en cuenta en investigaciones posteriores relacionadas con la presente tesis.

Los apéndices contienen el programa grabado en el procesador digital de señales y el programa de Matlab que se utilizó para el entrenamiento de la red neuronal, también se han incluido fotografías de la implementación de la tarjeta de circuito impreso del controlador digital y de la maqueta elaborada.

Capitulo 2

Modelado de la planta

2.1. Introducción

El modelo se basa en cualquier material cuya longitud sea mucho mayor que su ancho, y su anchura se muy grande comparada con su espesor. Dicho material debe pasar a través de varias secciones de procesamiento en el proceso de manufactura de un producto intermedio a uno final. Todas las secciones del proceso se asocian al material.

Usualmente se requiere diferentes magnitudes de tensión en cada sección de proceso. Grandes variaciones de tensión en estas secciones puede conducir a la degradación del producto final y hasta incluso la ruptura del material procesado en nuestro caso papel. El significado económico son pérdidas e impacto negativo en la línea de producción. Para disminuir la perdida potencial, se necesita incrementar adecuadamente el control de la tensión dentro de un rango predefinido.

2.2. Descripción del sistema

El complejo problema del modelado en la línea de procesamiento de una cinta se puede observar en la figura 2.1 donde tenemos una primera etapa de desenrollado del material, luego es transportado por rodillos para pasar por un sensor, seguidamente por unas bridas donde es procesado y finalmente es enrollado o bobinado obteniéndose el producto final.

El proceso puede ser simplificado dividiéndolo en partes donde las tensiones son diferentes por ejemplo entre el desenrollado y la brida 1, tendremos una tensión determinada diferente a la tensión entre la brida 1 y la brida 2 así como también entre la brida 2 y el enrollado, esto es obvio ya que los mecanismos que ejecutan fuerzas están solamente en las bridas y los motores de desenrollado y enrollado. Entonces nos queda el modelo simplificado que se muestra en la figura 2.2 donde podemos observar solamente el desenrollado, procesamiento y enrollado.

Entonces, solo es necesario estudiar una de estas partes y obtener un modelo para la tensión general que se podrá aplicar a cualquiera de las partes del sistema, también cabe

recalcar que la parte más importante es el enrollado, que es donde se obtiene el producto final, es por ello que se recomienda controlar la tensión principalmente en esta sección.

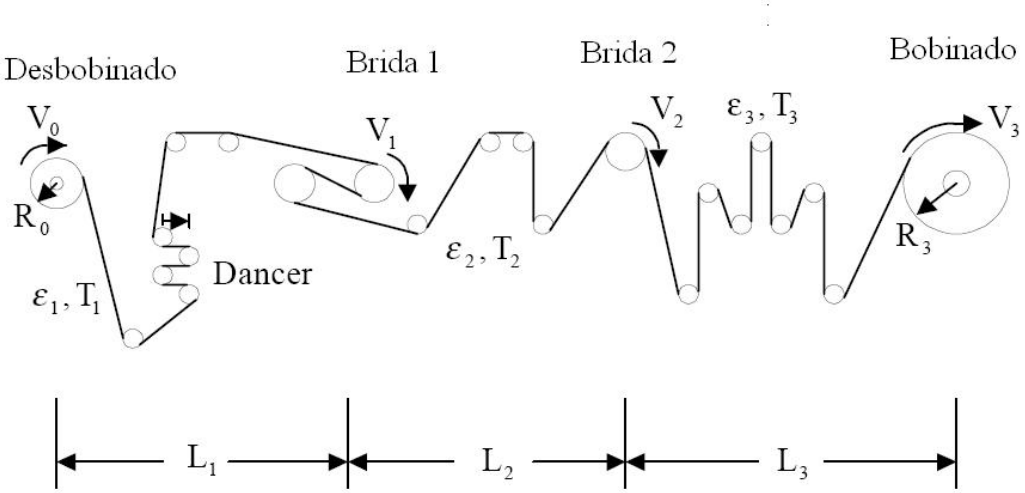


Figura 2.1: Diagrama de la línea de procesamiento.

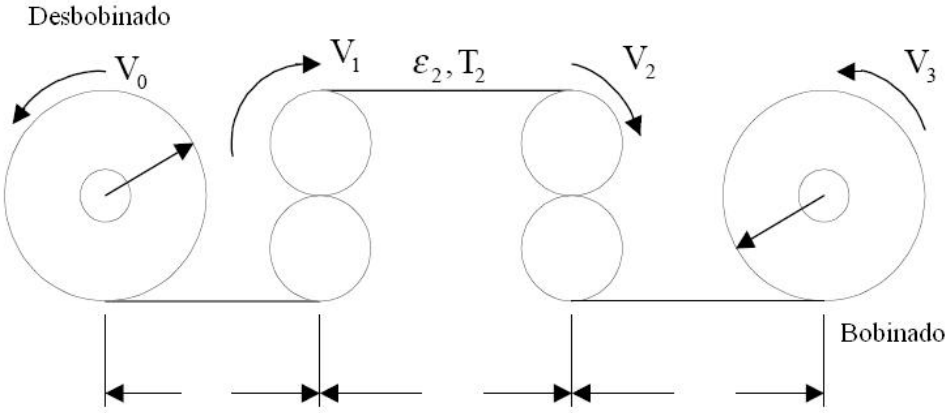


Figura 2.2: Diagrama simplificado de la línea de procesamiento.

2.3. Modelado dinámico

La dinámica de la tensión asociada con el transporte de la cinta a través de la zona de tensión es descrita en la ecuación (2.1). Esto está basado en el principio de conservación de la masa en un sistema de flujo de masa (**Referencia [6]** Hou Yi. Novel Control Approaches for Web Tension Regulation).

$$L\frac{dT_i}{dt} = E_i \cdot A_i \cdot (V_i - V_{i-1}) + V_{i-1} \cdot T_{i-1} - V_i \cdot T_i \qquad (2.1)$$

Donde:

$$V_i = \frac{R_i}{GR_i} \cdot \frac{2\pi}{60} \cdot \omega_i \qquad (2.2)$$

La ecuación del torque motor/carga es:

$$J_i \frac{d\omega_i}{dt} = \tau_i + \frac{R_i}{GR_i} \cdot \left(T_{i+1} - T_i\right) \tag{2.3}$$

Tabla 2.1: Nomenclatura utilizada para el modelo matemático.

Nomenclatura	Significado y unidades utilizadas
J_{motor}	Inercia del motor [kg·m^2]
J_{load}	Inercia del rollo reflejada (carga)
J_i	J_{motor}+J_{load} [kg·m^2]
V_i	Velocidad superficial del rodillo i-ésimo [m/min]
ω_i	Velocidad rotacional del rodillo i-ésimo [rpm]
R_i	Radio del rodillo i-ésimo [m]
GR_i	Proporción de engranajes del rodillo i-ésimo
L_i	Longitud de la zona de tensión i-ésima [m]
T_i	Tensión en la zona de tensión i-ésima [Kgf]
τ_i	Torque reflejado del eje del rodillo [Kgf·m]
E	Modulo de elasticidad [kgf·mm^2]
A	Área de sección de cruce [mm^2]
LS_i	Velocidad de operación de línea [m/min]

El motor y la caja de engranajes han sido omitidas para este modelo, y las suposiciones detalladas son como siguen:

- Las bandas del material en este caso papel no se estiran
- Los actuadores, bridas o motores son reguladores de velocidad ideales.

Las ecuaciones anteriores pueden ser presentadas en un diagrama de bloques como se muestra en la figura 2.3. Esta presentación modular incluye los enlaces que permiten acoplar las múltiples secciones. Hay que notar también que para una representación rigurosa de la ecuación (2.3) se requiere que a los integradores en la figura 2.3 se les represente con su respectivo valor inicial.

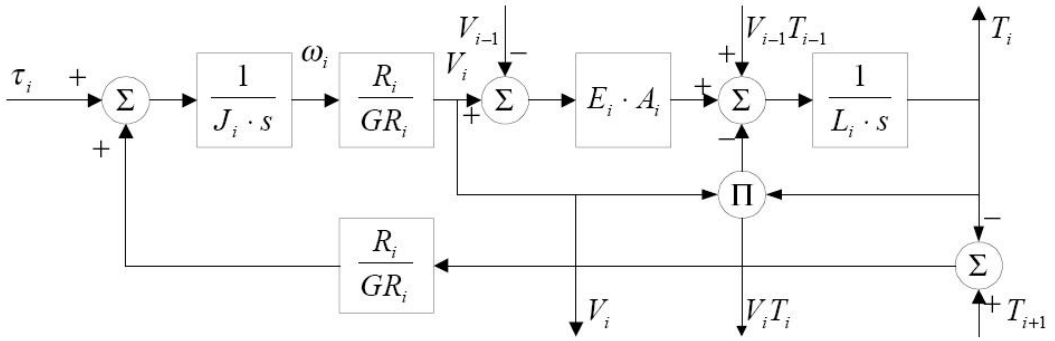


Figura 2.3: Diagrama de bloques de una zona de tensión de cinta.

Para una velocidad de operación de línea LS_i, se puede obtener una representación lineal aproximada de la ecuación (2.3). El diagrama de bloques del modelo linealizado se muestra en la figura 2.4, **con las suposiciones de que la tensión de entrada de la cinta es cero y que la referencia de velocidad del sistema V_{i-1} es constante.**

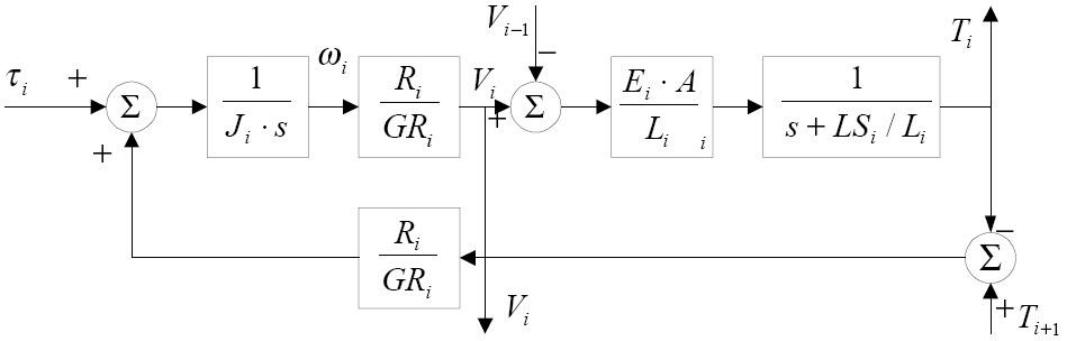


Figura 2.4: Diagrama de bloques linealizado de una zona de tensión.

El comportamiento del lazo de tensión en estado estable (esto es cuando la cinta deja de moverse). En este punto, con algunas simplificaciones, la función de transferencia puede ser aproximada como un sistema típico de segundo orden como se muestra en la ecuación 2.4.

$$\frac{T}{\tau} = \frac{k}{\frac{s^2}{\omega^2} + \frac{2 \cdot \zeta}{\omega} s + 1} \tag{2.4}$$

Donde $\omega \in [\omega_{\min}, \omega_{\max}]$, $\zeta \in [\zeta_{\min}, \zeta_{\max}]$ y $k \in [k_{\min}, k_{\max}]$.

La ecuación diferencial correspondiente es:

$$\ddot{T} = -2\zeta\omega\dot{T} - \omega^2 T + k\omega^2\tau \tag{2.5}$$

Considerando la tensión de acoplamiento, la zona muerta y otras no linealidades en el sistema como disturbios externos, una ecuación diferencial más realista para la dinámica de la tensión es:

$$\ddot{y} = f(t, y, \dot{y}, w) + bu \tag{2.6}$$

Donde *y* es la tensión, *u* es el torque del motor y *w* es la perturbación. Se entiende que $\ddot{y} = f(t, y, \dot{y}, w)$ es una función no lineal y variante en el tiempo que representa la dinámica real del sistema. La dificultad en el control de tensión radica principalmente en que la función cambia significativamente durante la operación. Por lo tanto las estrategias tienen que resolver este problema práctico.

Capitulo 3

Sistema de Control

3.1. Introducción

El objetivo del sistema de control consiste en que a una determinada señal de referencia, la respuesta del sistema siga la trayectoria indicada por dicha referencia. Este seguimiento debe ser tan cercano como sea posible y además también de realizarse en el menor tiempo posible.

Entonces el problema esencialmente es determinar una señal de control *u* dentro de un objetivo prescrito para que todos los objetivos de diseño sean satisfechos. Esta actividad no es sencilla debido a que el sistema analizado es del tipo no lineal como lo muestran las ecuaciones del proceso obtenidas en el capitulo anterior.

El sistema de control esta configurado para operar en lazo cerrado, es decir la señal de salida tendrá efecto sobre la acción de control, donde a esta acción de control se le denomina retroalimentación. El controlador es el encargado de procesar la información de la señal de error y generar una señal encargada de disminuir el valor de este error.

La figura 3.1 muestra el diagrama de un sistema de control en lazo cerrado, donde se muestran sus principales componentes: referencia o valor deseado, salida o variable controlada y el sistema de control; donde este último involucra el controlador, los actuadores, los sensores, los transductores y los detector de error.

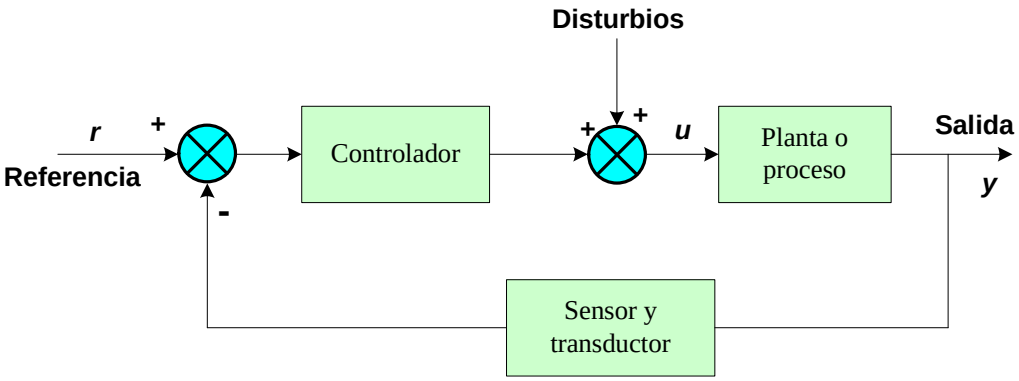


Figura 3.1: Diagrama de un sistema de control.

Entonces tenemos dos entradas: el valor deseado *r(t)* y la perturbación *v(t)*, y una salida: la señal realimentada y(t), si se considera las variables en el dominio de la frecuencia (variable compleja 's') y se define $G_c(s)$ y $G_p(s)$ como las funciones de transferencia del controlador y de la planta respectivamente a partir del diagrama de bloques del sistema de control de lazo cerrado de la figura 3.2, se obtiene que la señal realimentada, representación de la variable controlada, está dada por:

$$y(s) = \frac{G_c(s)G_p(s)}{1+G_c(s)G_p(s)}r(s) + \frac{G_p(s)}{1+G_c(s)G_p(s)}v(s) \tag{3.1}$$

Se debe considerar, entonces, dos posibles condiciones de operación del sistema de control, en primer lugar un servomecanismo es decir para: $v(s) = 0$

$$y(s) = \frac{G_c(s)G_p(s)}{1+G_c(s)G_p(s)}r(s) \tag{3.2}$$

Esta condición requiere un buen seguimiento del valor deseado. En segundo como un regulador, es decir para $r(s) = 0$:

$$y(s) = \frac{G_p(s)}{1+G_c(s)G_p(s)}v(s) \tag{3.3}$$

En donde lo importante es la insensibilidad a las perturbaciones.

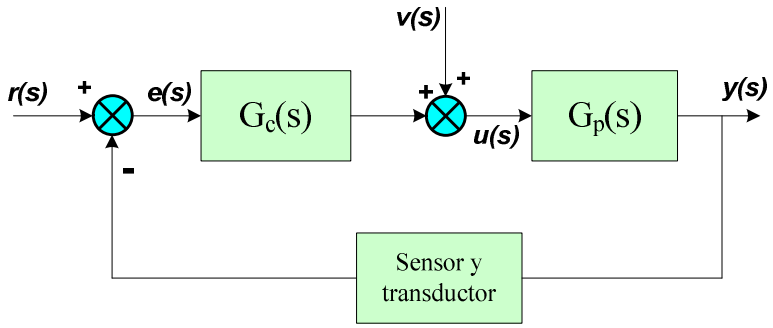


Figura 3.2: Diagrama de un sistema de control en el dominio de la frecuencia.

Dado que los numeradores de las funciones de transferencia de lazo cerrado (3.2) y (3.3) son diferentes, no necesariamente un buen ajuste del controlador para operar como servomecanismo, proveerá un buen funcionamiento como regulador, razón por la cual se han desarrollado procedimientos de sintonización para ambos tipos de operación.

En la industria de procesos, la mayor parte de los controladores se utilizan para responder a un cambio en la perturbación, y se requiere de una buena regulación para seguir el cambio en el valor deseado, con excepción de los controladores esclavos en los

sistemas de control en cascada. Si el controlador se ha sintonizado para lograr una buena respuesta en un cambio en el valor deseado, no eliminará las perturbaciones en forma efectiva, si el sistema contiene un integrador o si su constante de tiempo es grande.

Si el controlador se ha sintonizado para lograr una buena respuesta en un cambio en el valor deseado, no eliminará las perturbaciones en forma efectiva, si el sistema contiene un integrador o si su constante de tiempo es grande. Es importante, entonces, determinar los requisitos de funcionamiento del lazo de control para seleccionar el procedimiento de sintonización adecuado.

Por ejemplo un algoritmo de control muy utilizado actualmente en la industria es el control on-off, este mecanismo de realimentación simple se puede describir matemáticamente como sigue:

$$u = \begin{cases} u_{\max}, & e(t) > 0 \\ u_{\min}, & e(t) < 0 \end{cases} \tag{3.4}$$

Donde $e(t) = y(t) - r(t)$ (diferencia entre la referencia especificada por el operador y la salida medida del proceso) es el denominado error de control, u es la ley de control, u_{max} es el valor máximo que puede tomar u y u_{min} es el valor mínimo que puede tomar u.

Esta ley de control implica que siempre se usa la acción correctiva máxima. De esta manera, la variable manipulada tiene su valor más grande cuando el error es positivo y su valor más pequeño cuando el error es negativo. Este algoritmo es simple y no tiene parámetros que configurar, aparte de las acciones mínima y máxima que se ejecutan en el cálculo de la señal de control. El control on-off muchas veces es apropiado para mantener la variable controlada del proceso cerca del valor de la referencia que fue especificada, pero típicamente resulta en un sistema donde las variables oscilan. Es común tener algunas modificaciones ya sea introduciendo histéresis o una zona muerta como se muestra en la figura 3.3.

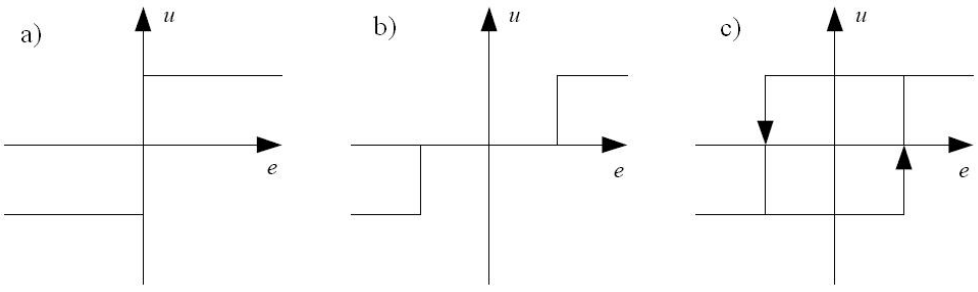


Figura 3.3: a) Controlador on-off ideal, b) con zona muerta, c) con histéresis.

3.2. Control PID

En la actualidad, cuando se habla de sistemas de control en la industria, se está haciendo referencia en la mayoría de casos a controladores diseñados con técnicas clásicas de control, a menudo lineal y casi en todos los casos se utiliza el control proporcional integral y derivativo o control PID. La utilización de estos controladores está actualmente generalizada y su potencia sobradamente demostrada en los casos en los que es aplicable. Debido a ello es que se implementó inicialmente un sistema de control clásico PID.

La acción de control PID genera una señal resultado de la acción proporcional, integral y la derivativa conjuntamente. El control PID consiste en multiplicar el error por una constante K_p llamada constante proporcional o ganancia, agregando además la derivada del error multiplicada por una constante T_d, llamada constante de tiempo derivativa, y la integral del error también multiplicada por otra constante T_i, llamada constante de tiempo integral. Entonces un controlador PID genera la siguiente señal de control:

$$u = K_p\left(e + \frac{1}{T_i}\int e dt + T_d\frac{de}{dt}\right) \tag{3.5}$$

$$donde: \quad e = r - y \tag{3.6}$$

O también:

$$u = K_p e + K_i\int e dt + K_d\frac{de}{dt} \tag{3.7}$$

$$donde: \quad K_i = \frac{K_p}{T_i}, \quad K_d = K_p T_d \tag{3.8}$$

La función de transferencia del controlador tendrá entonces la siguiente forma:

$$G_c(s) = K_p\left(1 + \frac{1}{T_i s} + T_d s\right) \tag{3.9}$$

Esta ecuación representada en la figura 3.4 es la forma mas conocida de un controlador PID. Es posible tener el mismo controlador, con otras funciones de transferencia, que varían en la forma de presentación pero no en el funcionamiento final, ya que contienen parte proporcional, integral y derivativa.

Forma cascada:

$$G_c(s) = K'\left(1 + \frac{1}{T_i' s}\right)\left(1 + T_d' s\right) \tag{3.10}$$

Forma paralela:

$$G_c(s) = \left(k + \frac{k_i}{s} + k_d s \right) \tag{3.11}$$

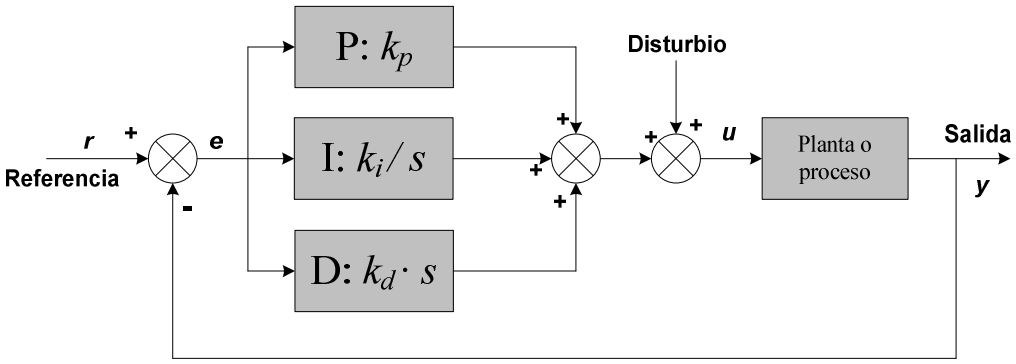


Figura 3.4: Estructura del lazo de retroalimentación con controlador PID.

3.2.1. Componentes de controlador PID

Como se ha explicado anteriormente el control PID tiene tres componentes, una proporcional, otra integral y finalmente una derivativa, cada una de estos componentes cumplen funciones específicas las cuales procederemos a explicar una por una, además daremos una breve explicación de otras variantes de control PID como son el control proporcional integral PI y el control proporcional derivativo PD para poder entender mejor el comportamiento del controlador PID.

Componente Proporcional

En este caso la acción de control es simplemente proporcional al error de control, el cual es definido como la diferencia entre la señal de medida de la variable o señal de salida y la referencia o valor deseado. Esto se consigue multiplicando el error de control por una constante *Kp*, y para polarizar la señal de control *u* se le suma un valor constante p_0 comúnmente a este valor se le conoce como “offset”.

$$u(t) = K_p e(t) + p_0 \tag{3.12}$$

Cuando el error de control es cero, la variable de control toma el valor $u(t) = p_0$. Esta variable es ajustada manualmente o precalculada de forma que el error de control en estado estacionario sea cero en una referencia dada. Generalmente, la polarización p_0 a menudo se la fija en:

$$p_0 = \frac{u_{\max} + u_{\min}}{2} \tag{3.13}$$

Esta componente PID toma un papel importante cuando la señal de error es grande, pero su acción se ve mermada con la disminución de dicha señal. Este efecto tiene como consecuencia la aparición de un error permanente, que hace que la parte proporcional nunca llegue a solucionar por completo el error del sistema.

La constante proporcional determinará el error permanente, siendo éste menor cuanto mayor sea el valor de la constante proporcional. Se pueden establecer valores suficientemente altos en la constante proporcional como para que hagan que el error permanente sea casi nulo pero, en la mayoría de los casos, estos valores solo serán óptimos en una determinada porción del rango total de control, siendo distintos los valores óptimos para cada porción del rango. Sin embargo, existe también un valor límite en la constante proporcional a partir del cual, en algunos casos, el sistema alcanza valores superiores a los deseados. Este fenómeno se llama sobre-oscilación y, por razones de seguridad, no debe sobrepasar el 30%, aunque es conveniente que la parte proporcional ni siquiera produzca sobre-oscilación.

La parte proporcional no considera el tiempo, por tanto la mejor manera de solucionar el error permanente y hacer que el sistema contenga alguna componente que tenga en cuenta la variación con respecto al tiempo es incluyendo y configurando las acciones integral y derivativa.

Componente integral

En la acción integral, la señal de control varía en razón proporcional a la integral de la señal de error, es decir el error es integrado, lo cual tiene la función de promediarlo o sumarlo por un periodo de tiempo determinado, luego es multiplicado por una constante K_I que representa la constante de integración. Esta componente tiene la siguiente señal de control:

$$u(t) = K_i \int e(t)dt \tag{3.14}$$

La componente integral tiene como propósito disminuir y eliminar el error en estado estacionario, provocado por el modo proporcional. Sin embargo, cuando cualquier error exista entre la medición y el valor de consigna, la acción integral hace que la salida comience a cambiar y continúe cambiando en tanto el error exista. Esta función, entonces, actúa sobre la salida para que cambie hasta un valor correcto necesario para mantener la medición en el valor de consigna a varias cargas sea alcanzado.

La acción de control integral recibe también el nombre de control de reset o restablecimiento, esto generalmente para controladores digitales donde este controlador se implementa con sumatorias.

Cuanta más acción integral exista en el controlador, mas rápido cambia la salida en función del tiempo. Entre las varias marcas de controladores, la salida de acción integral es medida de una o dos maneras, tanto en minutos por repetición, o en número de repeticiones por minuto.

Para aquellos controladores que miden en minutos por repetición, el tiempo de integración es la cantidad de tiempo necesaria para que dicho modo repita la respuesta del lazo abierto causada por el modo proporcional para un paso de cambio de error. Así, para estos controladores, cuanto menor sea el número de integraciones, mayor será la acción del modo integrador. En aquellos controladores que miden la acción integral en repeticiones por minuto, el ajuste indica cuantas repeticiones de la acción proporcional son generados por el modo de reset en un minuto. Así, para dichos controladores cuanto mayor sea el número reset, mayor será la acción integral. La correcta cantidad de acción reset depende de cuan rápido la medición puede responder al recorrido adicional de válvula que la misma causa.

Componente Derivativa

Así como la respuesta proporcional responde al tamaño del error y la integral responde al tamaño y duración del error, el modo derivativo responde a cuan rápido cambia el error. La función de la acción derivativa es mantener el error al mínimo corrigiéndolo proporcionalmente con la velocidad misma que se produce, de esta manera evita que el error se incremente. Esta componente tiene la siguiente señal de control:

$$u(t) = K_d \frac{de(t)}{dt} \tag{3.15}$$

Una mayor derivativa corresponde a un cambio más rápido y el controlador puede responder acordemente. La acción derivativa se manifiesta cuando hay un cambio en el valor absoluto del error, si el error es constante, solamente actúan los modos proporcional e integral.

Para evitar un gran pico causado por las escalones de cambio en el valor de consigna, la mayoría de los controladores modernos aplican la acción derivativo sólo a cambios en la medición. La acción derivativa en los controladores ayuda a controlar

procesos con constantes de tiempo grandes y tiempo muerto significativo, la acción derivativa es innecesaria en aquellos procesos que responden y no puede ser usado en absoluto en procesos con ruido en la señal de medición, ya que la acción derivativa en el controlador responderá a los cambios bruscos en la medición que el mismo observa en el ruido. Esto causará variaciones rápidas y grandes en la salida del controlador, lo que hará que la señal de control esté varíe bruscamente de manera continua produciendo mucho desgaste en los actuadores.

Un controlador solamente derivativo nunca se usa ya que este tipo de control solo es efectivo en periodos transitorios, como se verá mas adelante en el control PD para tener un funcionamiento mas adecuado debe aumentársele un control P.

Control proporcional derivativo (PD)

El controlador proporcional derivativo tiene como señal de control y función de transferencia, en ese orden, las siguientes ecuaciones:

$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt} \tag{3.16}$$

$$G_C(s) = K_p + K_d s \tag{3.17}$$

Donde *Kp* y *Kd* son las constantes proporcional y derivativa respectivamente.

Podemos resumir el comportamiento del controlador PD diseñado adecuadamente en las siguientes características: mejora el amortiguamiento y reduce el sobre-pico máximo, reduce el tiempo de subida y el tiempo de decaimiento, incrementa el ancho de banda, mejora el margen de ganancia y el margen de fase, puede acentuar el ruido en altas frecuencias, no es efectivo para sistemas ligeramente amortiguados o inicialmente inestables.

Control proporcional integral (PI)

Un control integral solo es usado muy pocas veces debido a que tiene velocidad de respuesta excesivamente lenta, y presenta una respuesta pobre ante cambios rápidos. Por este motivo se le agrega un control P el cual hace la respuesta más rápida, su señal de control y función de transferencia en ese orden son:

$$u(t) = K_p e(t) + K_i \int e(t)dt \tag{3.18}$$

$$G_c(s) = K_p + \frac{K_i}{s} \tag{3.19}$$

En resumen el controlador PI, mejora el amortiguamiento y reduce el sobrepaso máximo, incrementa el tiempo de subida, disminuye el ancho de banda, mejora el margen de ganancia y el margen de fase y finalmente filtra el ruido de alta frecuencia.

3.2.2. Métodos de sintonización PID.

La sintonización de los controladores PID, consiste en la determinación del ajuste de sus parámetros *Kc*, *Ti* y *Td*, para lograr un comportamiento del sistema de control aceptable y robusto de conformidad con algún criterio de desempeño establecido. Para poder realizar la sintonización de los controladores, primero debe identificarse la dinámica del proceso, y a partir de ésta determinar los parámetros del controlador utilizando el método de sintonización seleccionado.

Se dividen en métodos de lazo abierto y métodos de lazo cerrado:

En los **métodos de lazo cerrado** el controlador cuando está instalado operará manualmente. Produciendo un cambio escalón a la salida del controlador se obtiene la curva de reacción del proceso, a partir de la cual se identifica un modelo para el mismo, usualmente de primer orden más tiempo muerto. Este modelo es la base para la determinación de los parámetros del controlador.

En los **métodos de lazo abierto** el controlador opera automáticamente produciendo un cambio en el valor deseado se obtiene información del comportamiento dinámico del sistema para identificar un modelo de orden reducido para el proceso, o de las características de la oscilación sostenida del mismo, para utilizarla en el cálculo de los parámetros del controlador.

Cabe mencionar que también se puede calibrar un controlador PID utilizando valores pre-especificados para determinados procesos, como podemos ver en la tabla 3.1, estos valores están dados en rangos por lo que no son exactos, pero son una buena aproximación, el procedimiento es simplemente escoger el tipo de lazo o proceso que se tiene y colocar en el controlador PID los parámetros dentro del rango especificado y luego variarlos en un proceso de pruebe y error, teniendo en cuenta la tabla 3.2.

Los procedimientos de sintonización de lazo abierto utilizan un modelo de la planta que se obtiene, generalmente, a partir de la curva de reacción o respuesta del proceso, el modelo mas utilizado es el de primer orden más tiempo muerto y su función de transferencia es:

$$G_p(s) = \frac{k_p e^{-t_m s}}{\tau s + 1} \tag{3.20}$$

Tabla 3.1: Parámetros pre-especificados para controladores PID.

Tipo de Lazo	Kp	Ti (min)	Td (min)
Flujo	0.2 - 2	0.005 - 0.05	-
Presión (Líquido)	0.2 - 2	0.005 - 0.05	-
Presión (Gas)	2 - 100	1-50	0.02 - 0.1
Nivel (Líquido)	2 - 100	1 - 100	0.01 - 0.05
Temperatura	1-50	2-50	0.1 - 20

Tabla 3.2: Efectos de las constantes K_p, K_i y K_d en un proceso.

Efecto de incrementar parámetro:	**Tiempo de subida**	**Sobre-pico**	**Tiempo de establecimiento**	**Error en estado estacionario**	**Estabilidad**
K_p	Decrece	Incrementa	Pequeño incremento	Decrece	Degrada
K_i	Pequeño decremento	Incrementa	Incrementa	Elimina	Degrada
K_d	Pequeño decremento	Decrece	Decrece	Pequeño cambio	Mejora

Donde k_p es la ganancia del proceso, τ es la constante de tiempo del proceso y t_m es el tiempo muerto o retardo. Estas variables se pueden observar en la figura 3.5, donde se muestra una curva de respuesta un determinado proceso.

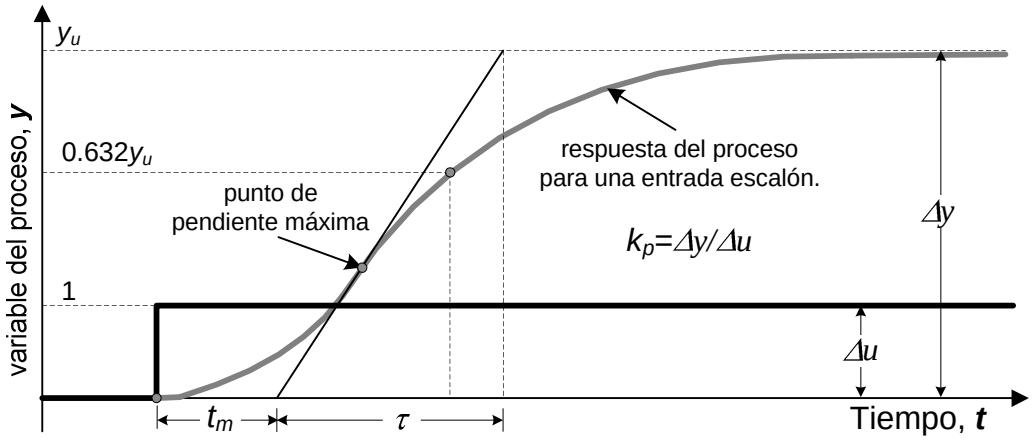


Figura 3.5: Curva de respuesta del proceso al escalón.

También se tienen otros modelos para procesos, menos utilizados pero igualmente importantes. Estos modelos son más difíciles de obtener al tener más variables para calcular o identificar. Por ejemplo tenemos:

Modelo de polo doble más tiempo muerto:

$$G_p(s) = \frac{k_p e^{-t_m s}}{(\tau s + 1)^2} \tag{3.21}$$

Modelo de segundo orden más tiempo muerto:

$$G_p(s) = \frac{k_p e^{-t_m s}}{(\tau_1 s + 1)(\tau_2 s + 1)} \tag{3.22a}$$

$$G_p(s) = \frac{k_p e^{-t_m s}}{\tau^2 s^2 + 2\zeta\tau s + 1} \tag{3.22b}$$

Sintonización Ziegler-Nichols de lazo abierto

El primer procedimiento sistematizado para el cálculo de los parámetros de un controlador PID fue desarrollado por Ziegler y Nichols. El criterio de desempeño que seleccionaron fue el de un decaimiento de ¼, o sea que el error decae en la cuarta parte de un periodo de oscilación, como se observa en la figura 3.6. Las ecuaciones fueron determinadas de forma empírica a partir de pruebas realizadas en el laboratorio con diferentes procesos, y están basadas en un modelo de primer orden más tiempo muerto identificado por el método de la tangente, para un funcionamiento del lazo de control como regulador con un controlador PID Ideal.

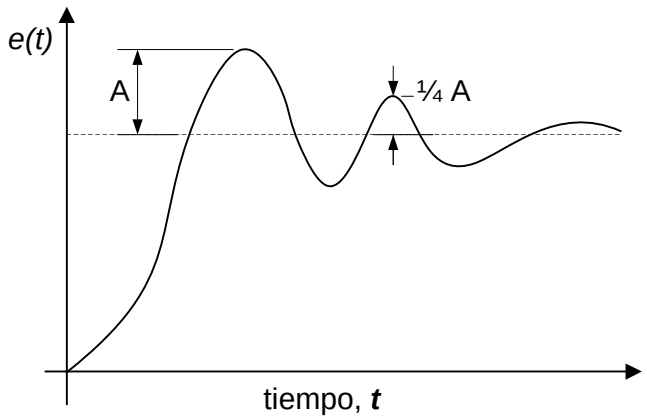


Figura 3.6: Respuesta a razón de decaimiento de 1/4.

Las ecuaciones de sintonización de este método para el controlador PID son:

$$K_c = 1.2\frac{\tau}{k_p t_m} \quad hasta \quad 2.0\frac{\tau}{k_p t_m} \tag{3.23a}$$

$$T_i = 2 \cdot t_m \tag{3.23b}$$

$$T_d = 0.5 \cdot t_m \tag{3.23c}$$

Este tipo de sintonización aun es ampliamente utilizado en la industria, también cabe mencionar que los controladores sintonizados con este método tienen un sobre-pico considerable.

Método de Ziegler y Nichols lazo cerrado

Al igual que sucedió con los procedimientos de sintonización basados en la curva de reacción del proceso, el primer procedimiento de sintonización basado en una prueba de lazo cerrado fue propuesto por Ziegler y Nichols, quienes presentaron ambos procedimientos en la misma publicación.

La principal dificultad del método de la respuesta transitoria que planteamos anteriormente como método Ziegler y Nichols de lazo abierto es que es muy sensible a las perturbaciones, ya que el experimento está basado en la respuesta sin retroalimentación. Los métodos basados en lazo cerrado evitan esta dificultad.

Utilizando un controlador puramente proporcional y mediante un proceso iterativo, el procedimiento requiere aumentar paulatinamente la ganancia del mismo hasta lograr que el sistema entre en una oscilación sostenida ante un cambio del escalón en el valor deseado. La ganancia en este punto es la ganancia última *Kcu* y el periodo de la oscilación, el periodo último *Tu*, a este punto se le conoce también como punto ultimo, el cual se puede observar en la figura 3.7.

Para el ajuste proporcional seleccionaron, como se indicó, el decaimiento de 1/4 como un compromiso entre el error permanente y el decaimiento, y encontraron que la ganancia proporcional para un controlador P debería ser la mitad de la ganancia última. Las ecuaciones de sintonización del controlador PID son:

$$K_c = 0.6K_{cu} \quad hasta \quad 1.0K_{cu} \tag{3.24a}$$

$$T_i = 0.5 \cdot T_u \tag{3.24b}$$

$$T_d = 0.125 \cdot t_u \tag{3.24c}$$

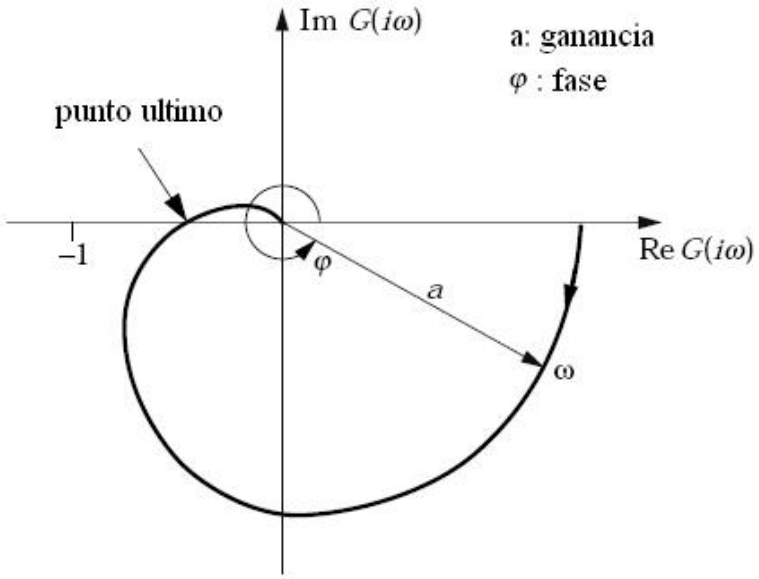


Figura 3.7: Determinación del punto ultimo.

La información última (*Kcu, Tu*) utilizada en las ecuaciones anteriores, también puede ser obtenida mediante una prueba con realimentación con relé como se mostrara a continuación.

Método de Realimentación con relé

La realimentación con relé, fue propuesta por K. J. Åström y T. Hägglund en 1984. En los últimos 20 años, la realimentación con relé se ha utilizado en numerosos sectores de la industria y es el método más importante comercialmente usado, ya que ahorra mucho tiempo en la sintonización debido a que es un proceso de auto-ajuste o auto-sintonización, es decir no se tiene la necesidad supervisar el proceso de sintonización.

La figura 3.8 muestra el diagrama de bloques de un ajuste automático para el control PID. Aström y Hägglund se basaron en el hecho de que un proceso que tenga un retraso de fase de por lo menos 180º a altas frecuencias, oscilará con un periodo de oscilación igual al periodo crítico bajo el control de un relé. El esquema está basado en un selector que permite al operador seleccionar entre el modo de control PID (selector en posición A) y el modo de ajuste automático de parámetros o auto-ajuste (selector en posición B).

Cuando se demanda la función de ajuste, se pone el selector a B, lo que significa que se activa la realimentación con relé y se desconecta el regulador PID. Cuando se obtiene

un ciclo límite estable, se calculan los parámetros del PID y luego se conecta el controlador PID al proceso con los parámetros calculados.

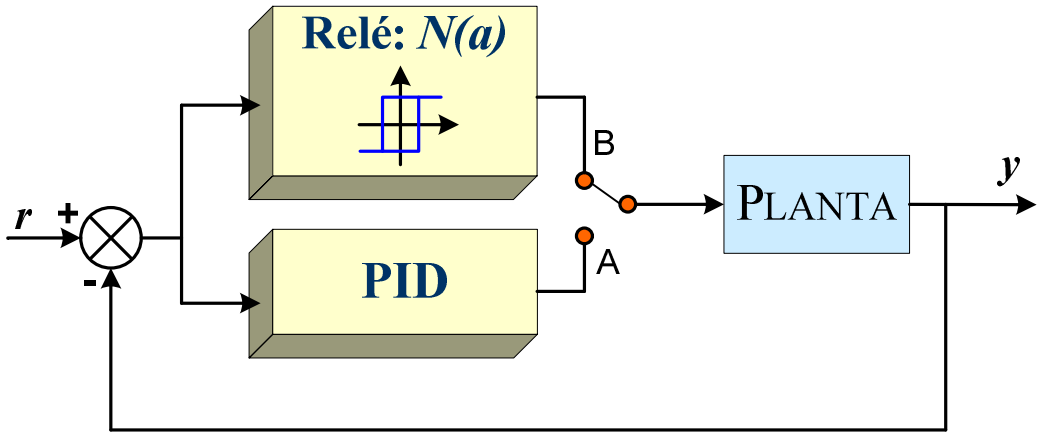


Figura 3.8: Diagrama de sintonizador con relé.

La realimentación con relé consiste en provocar una oscilación pequeña pero mantenida en un proceso por lo demás estable. La frecuencia y la ganancia límites del proceso se determinan según el período de las oscilaciones y los cambios de amplitud observados en la variable del proceso.

Una condición aproximada para la oscilación se puede determinar asumiendo que existe un ciclo límite con periodo T_u y frecuencia $\omega_u = 2\pi / T_u$ tal que la salida del relé es una onda periódica, cuadrada y simétrica como en la figura 3.9.

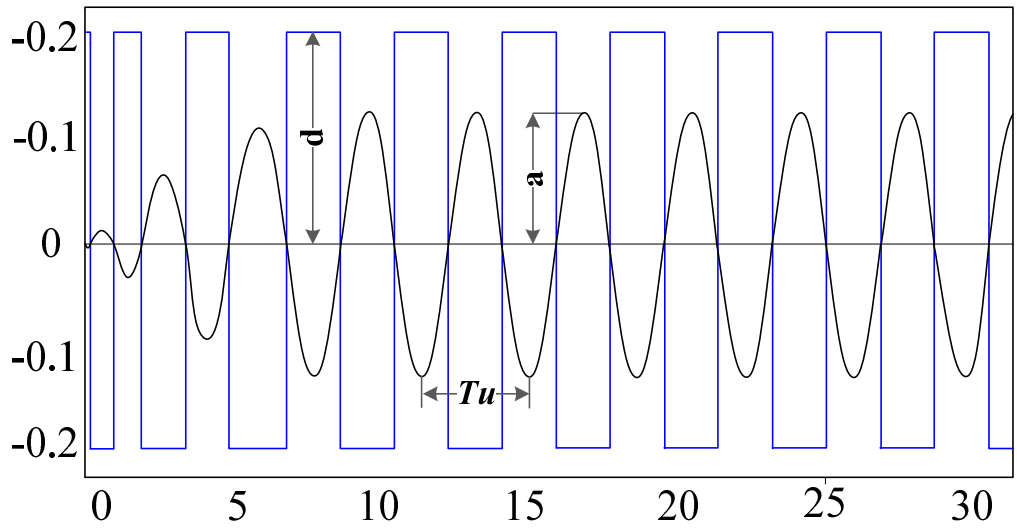


Figura 3.9: Salida y entrada de un proceso con un control con relé ideal

Si la amplitud del relé es *d* como se muestra en la figura 3.10 una simple expansión en series de Fourier de la salida del relé (ver figura 3.10) muestra que la primera componente harmónica (harmónica fundamental) tiene una amplitud de *4d /π.* La respuesta de la planta tendrá que ser entonces una forma senoidal de amplitud *a* (amplitud de la salida), de lo que resulta una ganancia ultima *Ku* dada por:

$$K_u = \frac{\frac{4d}{\pi}}{a} = \frac{4d}{\pi \cdot a} \tag{3.25}$$

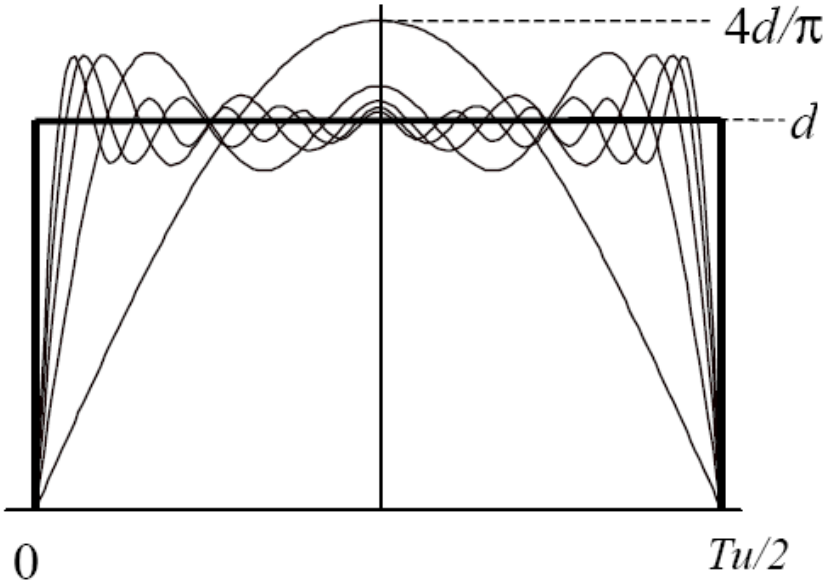


Figura 3.10: Desarrollo en series de Fourier de señal de control del relé.

También es posible llegar a la misma conclusión analizando la estabilidad con las funciones de transferencia del relé y del proceso, siendo estas *N(a,ω)* y *G(jω)* respectivamente. Un estado de oscilación es determinado por:

$$1 + N(a,\omega)G(j\omega) = 0 \quad \Rightarrow \qquad G(j\omega) = -\frac{1}{N(a,\omega)} \tag{3.26}$$

$$\mathrm{Re}\{G(j\omega)\} = -\frac{1}{N(a,\omega)} \quad y \quad \mathrm{Im}\{G(j\omega)\} = 0 \tag{3.27}$$

$$-\frac{1}{Ku} = -\frac{1}{N(a)} \quad \Rightarrow \quad Ku = N(a) = \frac{4d}{\pi \cdot a} \tag{3.28}$$

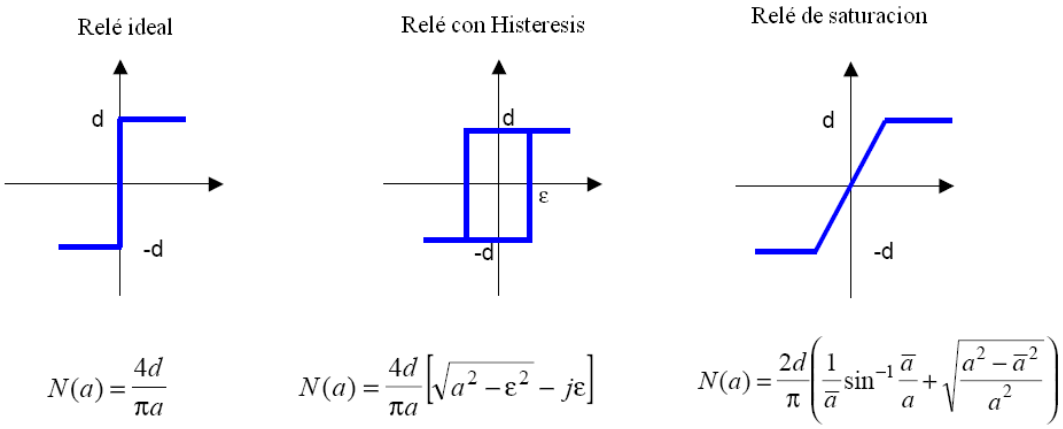


Figura 3.11: Distintos tipos de relé y sus funciones N(a).

El análisis realizado para un relé ideal se puede utilizar cuando el proceso es lento, pero cuando tenemos un proceso mucho mas rápido el relé debe considerarse el modelo de un relé real, es decir con histéresis o saturación como se muestra, las respuestas de tales relees son como se muestran en la figura 3.11.

3.3. Redes Neuronales Artificiales

Las redes neuronales artificiales son modelos matemáticos inspirados en sistemas biológicos, adaptados y simulados en computadoras convencionales. Las redes neuronales artificiales están compuestas por elementos que se comportan de una manera parecida a las funciones más elementales de una neurona biológica. Estos elementos se organizan de una manera que puede o no estar relacionada a la manera en que está organizado el cerebro.

No obstante su burdo parecido a los sistemas biológicos, las redes neuronales artificiales presentan algunas características propias del cerebro como son: aprendizaje, generalización y abstracción.

Aprendizaje

Pueden aprender de la experiencia. Las redes neuronales artificiales pueden modificar su comportamiento en respuesta al medio ambiente. Esto es muy importante. Las redes neuronales artificiales se auto-ajustan produciendo respuestas consistentes con el medio ambiente. Existen muchos algoritmos de aprendizaje que pueden aplicarse a las redes neuronales artificiales.

Generalización

Pueden generalizar a partir de ejemplos previos. Una vez entrenada, una red neuronal artificial es, hasta cierto grado, insensible a variaciones pequeñas en sus entradas. Esto es, las redes neuronales artificiales producen sistemas capaces de manejar el mundo "imperfecto" en que vivimos.

Abstracción

Pueden abstraer características esenciales de entradas que contienen datos irrelevantes. Algunas redes neuronales artificiales son capaces de abstraer la esencia de una serie de entradas. Se pueden abstraer patrones perfectos de modelos distorsionados.

Entre algunas de las desventajas de las redes neuronales artificiales es que son hasta cierto grado impredecibles (como los humanos). No hay manera de garantizar la salida de una red para una entrada a menos que se prevean todas las posibilidades en las entradas, y se entrene la red suficientemente bien. Sin embargo esto puede resultar impráctico en muchos casos y verdaderamente imposible en otros, lo que ha originado muchas críticas a las redes neuronales artificiales. Parte de esta crítica se debe al hecho de que esperamos que las computadoras sean perfectas, pero los humanos no son perfectos. Otro problema relacionado y criticado en las redes neuronales artificiales es su inhabilidad de "explicar" como resuelven el problema. Esta inhabilidad se debe a que la representación interna generada en la red puede ser demasiado compleja aún en los casos mas sencillos.

La base y fuente de inspiración de las redes neuronales artificiales es la célula del sistema nervioso de los animales, conocida como neurona, y es por tanto importante observar su fisiología para comprender como los investigadores en ingeniería y matemática tratan de imitar los mecanismos de almacenamiento y procesamiento de la información en el cerebro. En una neurona biológica, figura 3.12, se puede distinguir cuatro partes fundamentales: el soma, el axón y las sinapsis.

El soma

El soma o núcleo de la célula es la parte central redonda donde se realizan casi todas las funciones lógicas de la neurona. La capa externa del soma tiene la capacidad única de generar impulsos nerviosos, la cual es una función vital.

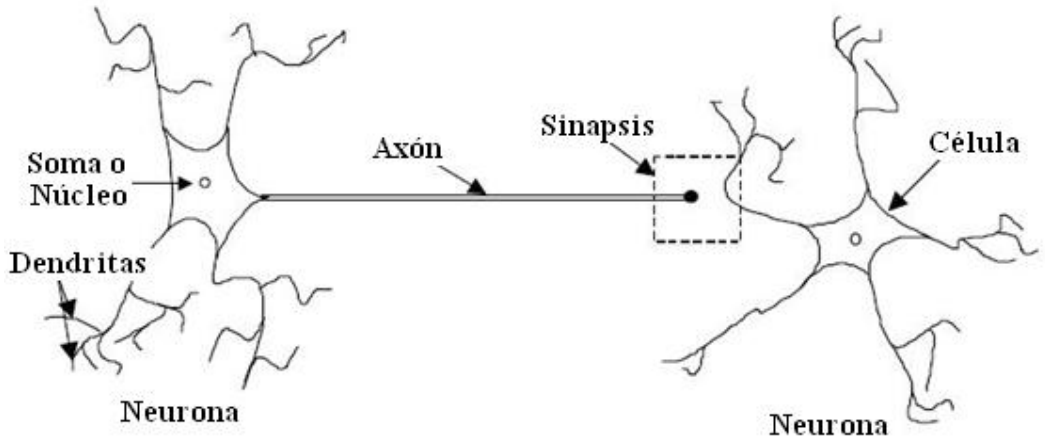


Figura 3.12: Modelo de neuronas biológicas.

El axón

La conexión sináptica es un pequeño bulbo que contiene estructuras esféricas llamadas vesículas sinápticas, las cuales contienen un gran número de moléculas neurotransmisoras. Además de este proceso de activación neuronal, algunas neuronas que han sido activadas muy débilmente transmiten ciertos signos electromagnéticos a través de su interior con cierta respuesta. Podemos decir que son "locales" pues el impulso muere después de cierta distancia.

El axón es una fibra nerviosa conectada directamente con el soma y que sirve como canal de salida. El axón usualmente esta muy ramificado para permitir su conexión a un gran número de neuronas. En estos sistemas biológicos las señales son secuencias de impulsos que se propagan por el axón sin atenuación.

Las dendritas

Las dendritas son ramas que salen del cuerpo. Las dendritas son las entradas de información a la neurona. Son un grupo de fibras muy ramificadas y de forma irregular que se conectan directamente al soma. Se calcula entre 103 y 104 el número de dendritas en una neurona, permitiendo que esta reciba información de un gran grupo de otras neuronas.

Las sinapsis

Sinapsis son contactos especializados entre los axones y las dendritas de diferentes neuronas. Estas sinapsis pueden cambiar la polaridad de los potenciales provenientes de otras neuronas y en estos casos se suele hablar de naturaleza excitadora o inhibidora según sea su función para la excitación o bloques de la neurona. Se considera que el

almacenamiento de la información esta concentrado en estas conexiones sinápticas. Se conoce que en el sistema nervioso de los seres humanos estas conexiones sinápticas son de naturaleza química muy compleja a diferencia de los insectos que tienen conexiones de transmisión eléctrica simples.

3.3.1. El neurón artificial

El funcionamiento de un neurón artificial es simple. Básicamente consiste en aplicar un conjunto de entradas, cada una representando la salida de otro neurón, o una entrada del medio externo, realizar una suma ponderada con estos valores, y "filtrar" este valor con una función.

McCulloch y Pitts construyeron un modelo básico de neurona artificial, con una neurona muy simple a base de un sumador y una función de activación. Las conexiones (sinapsis) de una neurona se consideran como se muestra en la figura 3.13. En ellas, las activaciones o entradas x_j con unas determinadas intensidades w_j de otras neuronas son sumadas, y se permite que en la salida de la neurona (axón) se origine una actividad siempre que la suma $x_j \cdot w_j$ supere un valor umbral θ. Como podemos observar, esta es una suma algebraica de las entradas ponderadas.

La expresión matemática de esta neurona es:

$$y = f(h) = f\left(\sum_{j=1}^{N} w_j x_j + \theta\right) \tag{3.29}$$

Donde w_j son los pesos sinápticos que ponderan las entradas x_j. La función de activación de la neurona es f y n es el número total de pesos sinápticos conectados a la entrada de la neurona, y finalmente y es la salida de la neurona.

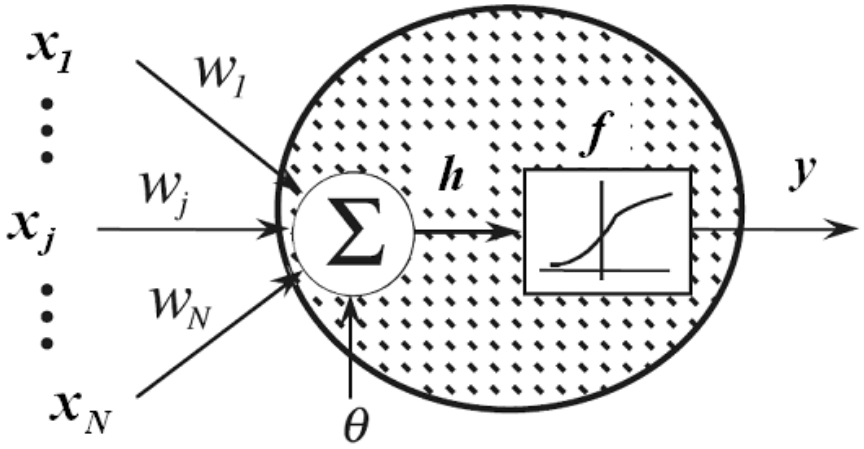


Figura 3.13. Cálculo de la salida de un neurón artificial.

Existen muchas funciones de activación, a continuación se presentan las principales:

a) Función escalón de conexión o desconexión: Esta función define dos posiciones, si o no. La salida de esta función es, o bien una constante positiva, una constante negativa o cero (ver figura 3.14a). Esta función posee una discontinuidad en un punto que imposibilita la evaluación de la derivada en dicho punto.

Función escalón asimétrica:

$$f(h) = \begin{cases} 0, & h < 0 \\ 1, & h \geq 0 \end{cases} \tag{3.30}$$

Función escalón simétrica:

$$f(h) = \begin{cases} -1, & h < 0 \\ 1, & h \geq 0 \end{cases} \tag{3.31}$$

b) Función lineal: Es muy sencilla, tiene característica lineal y no tiene límites en su rango (ver figura 3.14b), esta dada por la ecuación:

$$y = f(h) = ah + b \tag{3.32}$$

c) Función sigmoide: Es una función continua y diferenciable en todo su dominio, monótonamente creciente y posee una característica bipolar (ver figura 3.14c). Matemáticamente se representa en la siguiente ecuación:

$$f(h) = \frac{1}{1 + e^{-h/T}} \tag{3.33}$$

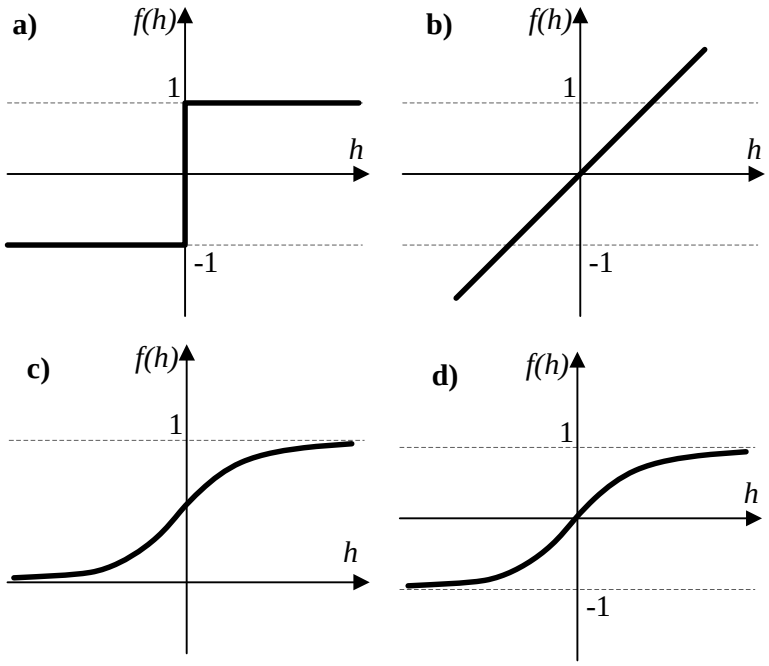


Figura 3.14: Funciones de activación mas comunes para redes neuronales.

d) Función tangente hiperbólica: Esta función es muy común, su forma es similar a la función sigmoide pero es simétrica. Es usada por biólogos como un modelo matemático de activación nervioso (ver figura 3.14d). Está dada por:

$$f(h) = \tanh(h/T) \tag{3.34}$$

3.3.2. Arquitectura de las redes neuronales artificiales.

Si bien la red neuronal artificial es la unidad principal, no menos importante es como se interconectan entre ellas para formar una red que procese la información. Nos referimos a la arquitectura como las diferentes formas de interconectar esas unidades básicas creando redes de topología diferentes. Pueden existir un sinnúmero de posibles combinaciones de las conexiones entre ellas.

Redes Neuronales de propagación hacia adelante

Este tipo de estructuras se organiza en un grupo de neuronas que procesan la información de las entradas paralelamente y luego las salidas de las neuronas pueden combinarse para obtener unas salidas de la red o alimentar otro grupo de neuronas. La figura 3.16 muestra una estructura de red neuronal de una sola capa de salida. La figura 3.15 muestra una estructura de red neuronal multicapa que contiene una o más capas ocultas. Normalmente, las entradas se conectan a la primera capa oculta. La salida de cada neurona se conecta a las entradas de las neuronas de la siguiente capa hasta alcanzar a la capa de salida.

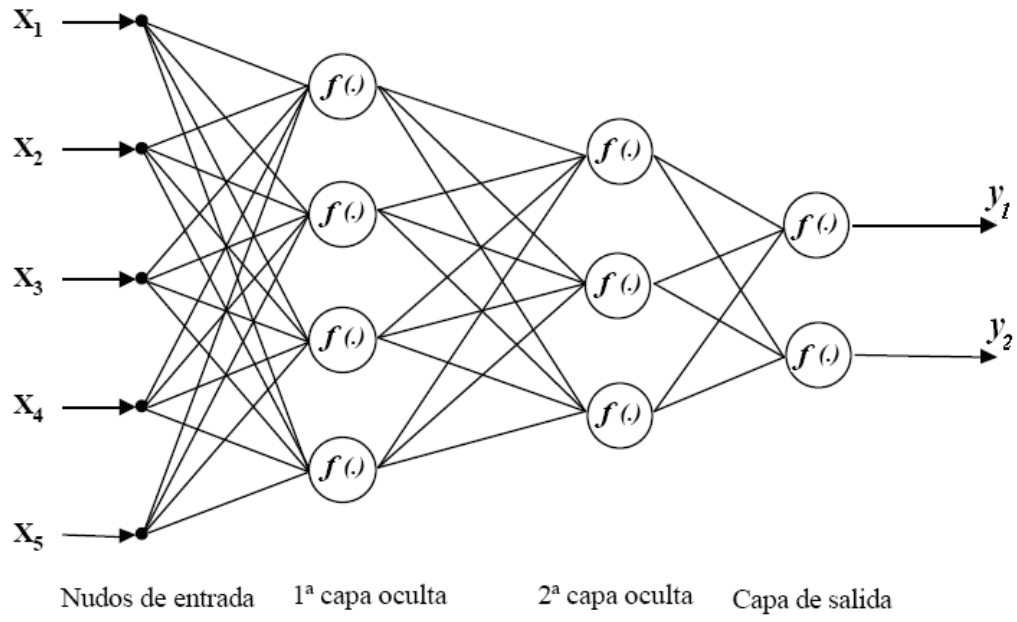


Figura 3.15: Red de propagación multicapa.

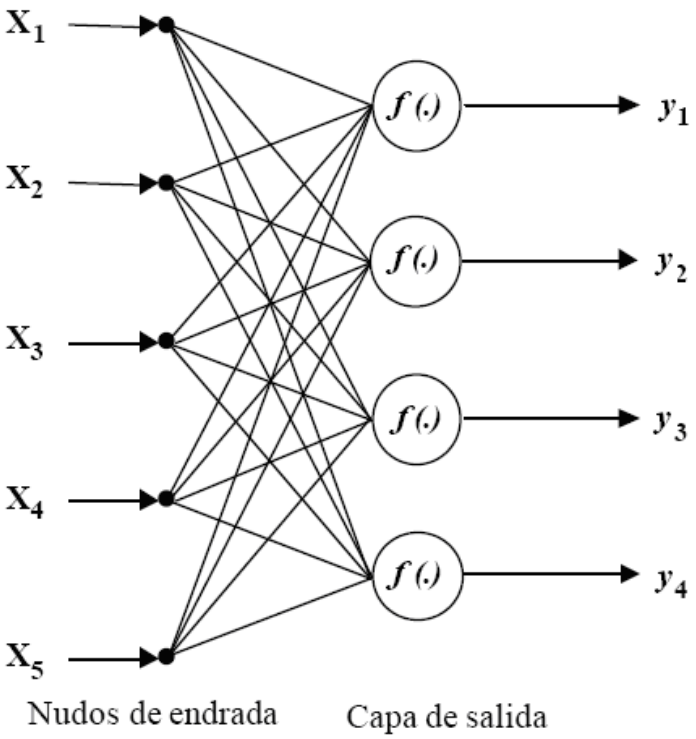


Figura 3.16: Red de propagación de una capa.

Red Hopfield

Algunas de las arquitecturas de redes neuronales se caracterizan por el hecho de que algunas de sus entradas están realimentadas por salidas de la propia red. La red propuesta por Hopfield (ver figura 3.17) es un ejemplo de estas arquitecturas incluyendo su propio modelo de neurona, como se presenta en la sección anterior.

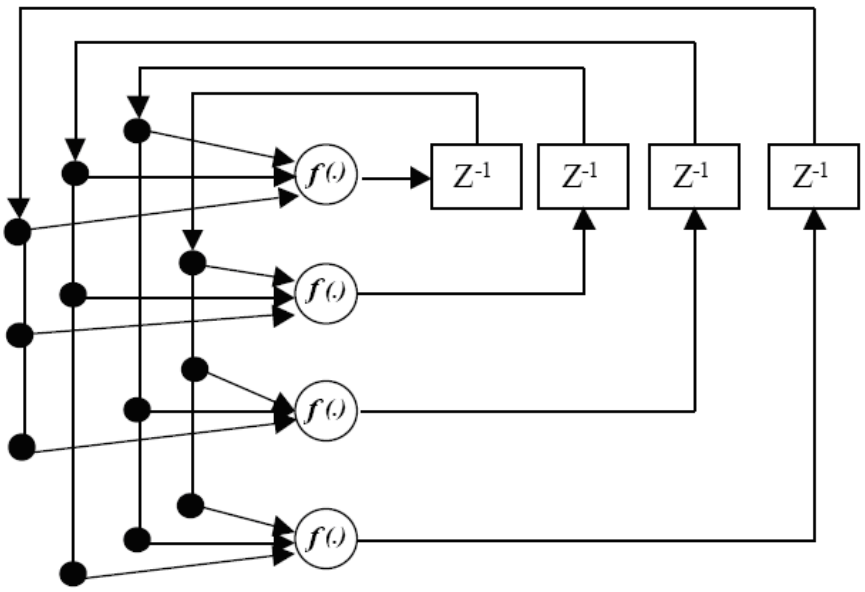


Figura 3.17: Red de propagación de Hopfield.

3.3.3. Entrenamiento de redes neuronales artificiales.

El entrenamiento de una red consiste en obtener los pesos w_j de manera que un conjunto de valores de entrada produce la salida deseada. Durante el entrenamiento, los pesos w_j se van ajustando gradualmente, hasta encontrar aquellos que generen las salidas deseadas.

Hay dos tipos de entrenamiento de redes neuronales: supervisado y no supervisado.

En el **entrenamiento supervisado**, se requiere que cada vector de entrada esté relacionado con una salida deseada representando cada vector. Al par de vectores representando los valores de entrada y salida deseada se le llama el par de entrenamiento. Cada vector de entrada se aplica a la red, la correspondiente una salida que se compara con la salida deseada. La diferencia o error se da a la red y los pasos se cambian de acuerdo a algoritmos que minimizan este error (ver figura 3.18). Diferentes vectores de entrenamiento se aplican a la red secuencialmente hasta que el error en el conjunto de aprendizaje es suficientemente pequeño.

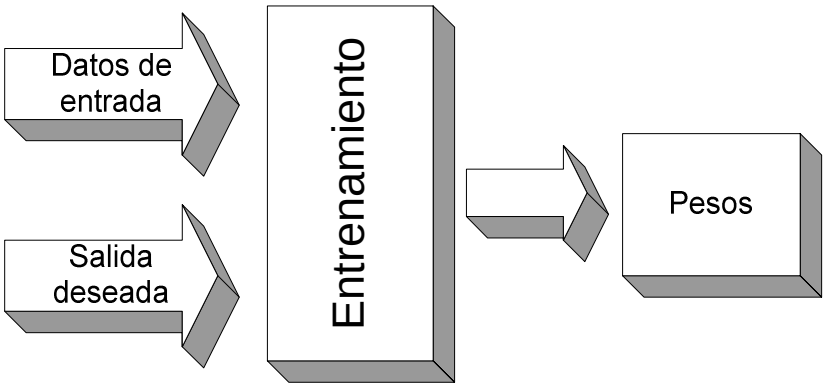


Figura 3.18: Entrenamiento Supervisado.

El **entrenamiento no supervisado** se caracteriza porque no requiere de un vector de salida como se muestra en la figura 3.19. Los pesos de la redes se van modificando de manera que produzcan vectores de salida que sean consistentes, esto es, se espera que entradas similares produzcan las mismas salidas. Matemáticamente hablando, el proceso de entrenamiento no supervisado extrae propiedades estadísticas del conjunto de entrenamiento.

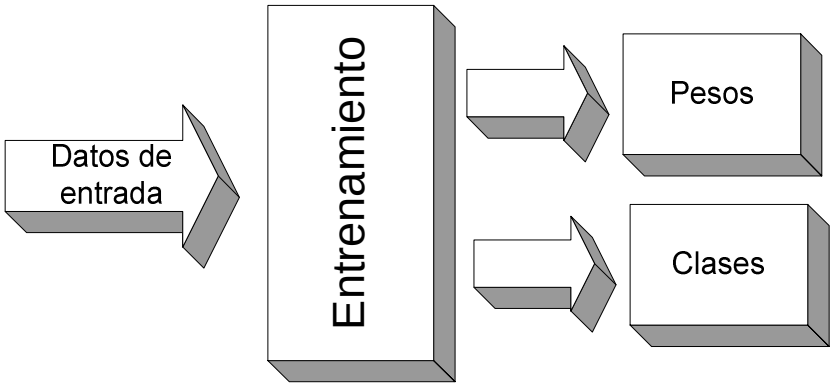


Figura 3.19 Entrenamiento no Supervisado

3.3.4. El perceptrón

En el modelo del perceptrón mostrado en la figura 3.20, se emplea un simple neurón con una red ponderada con función de activación escalón. La entrada del neurón es $\underline{x} = (x_1, x_2,\ldots, x_n)$. La red $h(\underline{x})$ es la suma ponderada de las entradas:

$$h(\underline{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \theta \tag{3.35}$$

Y la salida $y(\underline{x})$ es obtenida de $h(\underline{x})$ vía la función de activación:

$$y(\underline{x}) = \begin{cases} 0, & h(\underline{x}) < 0 \\ 1, & h(\underline{x}) \geq 0 \end{cases} \tag{3.36}$$

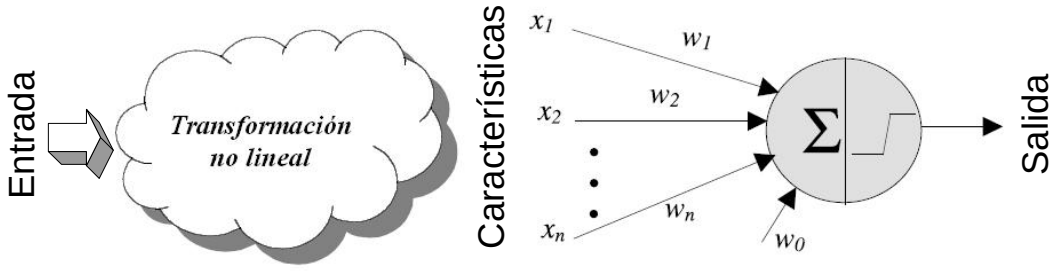


Figura 3.20: Modelo de red neuronal del perceptrón

El vector de ponderación $\underline{w} = (w_1, w_2,\ldots, w_n)$ debe ser determinado para aplicar el modelo del perceptrón. Usualmente, se dan un conjunto de muestras de entrenamiento $\{(\underline{x}(i), \underline{d}(i)); i \in I_r\}$ y muestras de prueba $\{(\underline{x}(i), d(i)); i \in I_t\}$. Donde $d(i)(\in[0,1])$ es el valor de la salida deseada de $y(\underline{x}(i))$ si el vector de ponderación $\underline{w}$ es escogido correctamente y además I_r e I_t son conjuntos índices diferentes.

Se puede aplicar un algoritmo de aprendizaje secuencial en línea para estimar de forma iterativa el valor correcto de $\underline{w}$ presentando las muestras de entrenamiento al

neurón perceptrón en una secuencia de orden al azar. El algoritmo de enterramiento tiene la siguiente formula:

$$\underline{w}(k+1) = \underline{w}(k) + \eta\big(d(k) - y(k)\big)\underline{x}(k) \qquad (3.37)$$

Donde *y(k)* es calculado usando las ecuaciones 3.43 y 3.44. En la ecuación 3.44 la taza de aprendizaje $\eta(0 < \eta < 1/|\underline{x}(k)|_{\max})$ es un parámetro escogido por el usuario, donde $|\underline{x}(k)|_{\max}$ es la máxima magnitud de las muestras de entrenamiento. El índice *k* es usado para indicar que las muestras de entrenamiento son aplicadas secuencialmente al perceptrón en un orden al azar. Para cada tiempo una muestra de entrenamiento es aplicada, la correspondiente salida del perceptrón *y(k)* es para ser comparada con la salida deseada *d(k)*. Si son las mismas, significa que el vector de ponderación $\underline{w}$ es correcto para esta muestra de entrenamiento. Por otro lado si son diferentes, entonces $\underline{w}$ será actualizado con un pequeño paso en la dirección del vector de entrada $\underline{x}(k)$.

Esta demostrado que si las muestras de entrenamiento son linealmente separables, el algoritmo de aprendizaje del perceptrón convergerá a una solución factible para el vector de ponderación con un numero finito de iteraciones. Por otro lado, si las muestras de entrenamiento no son linealmente separables, el algoritmo no convergerá a un valor de η, diferente de cero y constante.

El perceptrón multicapa

Un modelo de red neuronal de perceptrón multicapa MLP (multilayer perceptron) consiste en una red de varias capas y feedforward de neuronas de McCulloch y Pitts. Cada neurón. Cada MLP tiene una función de activación no lineal que es usualmente continuamente diferenciable.

Una típica configuración de MLP se muestra en la figura 3.21 donde cada círculo representa un neurón individual. Estos neurones están organizados en capas. La última capa se denomina capa de salida y las otras capas se denominan capas ocultas.

Una MLP provee un mapeo no lineal entre las entradas y salidas. Está demostrado que con una cantidad suficiente de neurones ocultos, una MLP con tan solo dos capas de neurones ocultas es capaz de aproximar estructuras complejas con un soporte limitado.

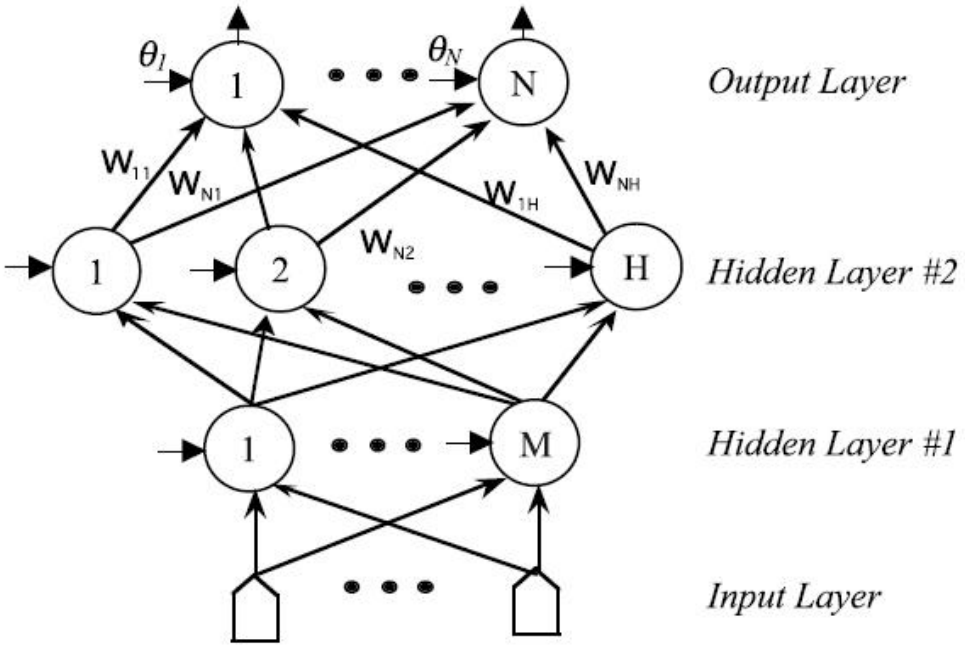


Figura 3.21: Un perceptrón multicapa de tres capas.

3.3.5. Algoritmo de Retropropagación

El algoritmo de retropropagación del error (Backpropagation) es el más utilizado para el entrenamiento de los perceptrones multicapa. Se trata de un algoritmo de gradiente descendiente, normalmente de aprendizaje supervisado, que modifica los pesos de cada conexión.

El proceso de actualización de los pesos se puede llevar a cabo tras la presentación de cada patrón de entrenamiento o bien una vez que todos los patrones de entrenamiento han sido presentados a la red. La clave en la aplicación de un modelo MLP es la determinación de la matriz de pesos o matriz de ponderación, estos valores se calculan mediante el algoritmo de retropropagación.

Para ilustrar este procedimiento consideremos inicialmente un simple neurón, representado en la figura 3.22 en dos partes: una unidad de suma para calcular las funciones de la red *h*, y una función de activación *z=f(h)*, la salida *z* es comparada con el valor deseado *d*, y su diferencia, el error *e = d - z*, será calculado. Tenemos dos entradas $[x_1\ x_2]$ con sus correspondientes pesos w_1 y w_2. La entrada etiquetada con 1 representa el término del umbral θ visto en las figuras 3.15 y 3.22. Aquí el término umbral se etiqueta como w_0, entonces la función de la red es calculada por:

$$h = \sum_{i=0}^{2} w_i x_i = \mathbf{W}\mathbf{x} \tag{3.38}$$

Donde $x_0=1$, $\mathbf{W} = [w_0\ w_1\ w_2]$ es el vector de ponderación y $\mathbf{x} = [1\ x_1\ x_2]^T$ es el vector de entrada.

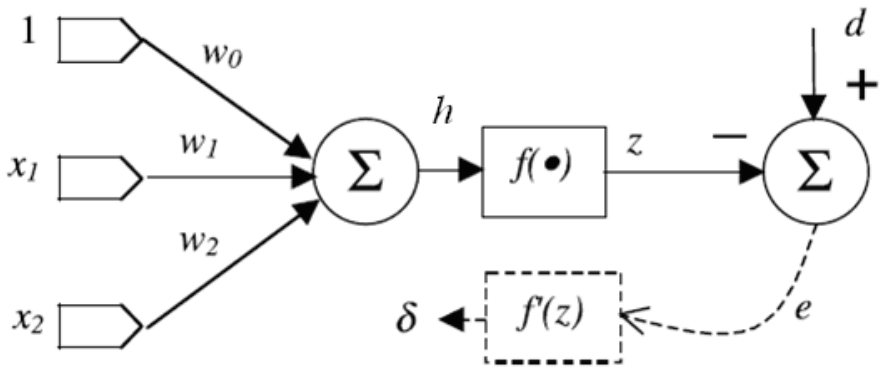


Figura 3.22: MLP para entrenamiento por retropropagación de neurón simple.

Dado un conjunto de muestras de entrenamiento $\{(x(k), d(k)); 1 \le k \le K\}$, el entrenamiento de retropropagación de error empieza al alimentar todas las entradas K a través de la red MLP y calculando la salida $\{z(k); 1 \le k \le K\}$ correspondiente. Inicialmente la matriz de ponderación **W** se tantea. Entonces la suma los errores elevados al cuadrado se calcula de la siguiente manera:

$$E = \sum_{k=1}^{K} [e(k)]^2 = \sum_{k=1}^{K} [d(k) - z(k)]^2 = \sum_{k=1}^{K} [d(k) - f(\mathbf{W}\mathbf{x}(k)]^2 \qquad (3.39)$$

El objetivo es ajustar la matriz de pesos W para minimizar el error E. Esto conduce al problema de optimización cuadrática no lineal mínima. Existen numerosos métodos de optimización disponibles para resolver este problema. Básicamente, estos algoritmos adoptan una formulación similar para la iteración:

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \Delta\mathbf{W}(t) \qquad (3.40)$$

Donde $\Delta\mathbf{W}(t)$ es la corrección hecha para los pesos actuales de **W**(t). Los algoritmos básicamente difieren en la forma de $\Delta\mathbf{W}(t)$. Algunos algoritmos importantes se pueden ver a continuación.

a) Método de gradiente descendiente:

$$\Delta\mathbf{W}(t) = \eta g(t) = -\eta \frac{dE}{d\mathbf{W}} \qquad (3.41)$$

Donde g es conocida como el vector gradiente, η es el tamaño de paso o taza de aprendizaje. Este algoritmo es también conocido como aprendizaje de retropropagación de error.

b) Método de newton:

$$\Delta \mathbf{W}(t) = -\mathbf{H}^{-1} g(t) \tag{3.42}$$

$$\Delta \mathbf{W}(t) = -\left[\frac{d^2 E}{d\mathbf{W}^2}\right]^{-1} \frac{dE}{d\mathbf{W}} \tag{3.43}$$

H es conocida como la matriz Hessiana. Existen muchos métodos para calcularla.

c) Método de Gradiente-Conjugada:

$$\Delta \mathbf{W}(t) = \eta \cdot p(t) \quad donde: \quad p(t+1) = -g(t+1) + \beta \cdot p(t) \tag{3.44}$$

Nos enfocaremos en el método de gradiente descendiente que es también la base del algoritmo de aprendizaje de retropropagación de error. La derivada de E con respecto a cada peso individual se calcula como sigue:

$$\frac{\partial E}{\partial w_i} = \sum_{k=1}^{K} \frac{\partial [e(k)]^2}{\partial w_i} = \sum_{k=1}^{K} 2[d(k) - z(k)]\left(-\frac{\partial z(k)}{\partial w_i}\right) \qquad para \quad i = 0,1,2 \tag{3.45}$$

Donde:

$$\frac{\partial z(k)}{\partial w_i} = \frac{\partial f(h)}{\partial h}\frac{\partial h}{\partial w_i} = f'(h)\frac{\partial}{\partial w_i}\sum_{j=0}^{2}\left(w_j x_j\right) = f'(h)x_i \tag{3.46}$$

Entonces:

$$\frac{\partial E}{\partial w_i} = -2\sum_{k=1}^{K}[d(k) - z(k)]\ f'(h(k))\, x_i(k) \tag{3.47}$$

Con $\delta(k)=[d(k)-z(k)]\ f'(h(k))$, la ecuación anterior puede ser expresada como:

$$\frac{\partial E}{\partial w_i} = -2\sum_{k=1}^{K}\delta(k) x_i(k) \tag{3.48}$$

$\delta(k)$ es la señal de error $e(k)=d(k)-z(k)$ modulada por la derivada de la función de activación $f'(h(k))$ y por lo tanto representa cantidad de corrección necesaria a aplicarse al peso w_i para la entrada dada $x_i(k)$. El cambio total Δw_i es así la suma de cada contribución sobre todas las muestras de entrenamiento K. En consecuencia, la formula de actualización para los pesos es de la siguiente forma:

$$w_i(t+1) = w_i(t) + \eta \sum_{k=1}^{K} \delta(k) x_i(k) \tag{3.49}$$

En el caso de utilizar la función de activación sigmoide, definida en la ecuación 3.40, entonces *δ(k)* se calcula como:

$$\delta(k) = \frac{\partial E}{\partial h} = \left[d(k) - z(k)\right] \cdot z(k) \cdot \left[1 - z(k)\right] \tag{3.50}$$

En el caso de tener un perceptrón de múltiples capas, se procede de forma similar, en la figura 3.23 se observa la función de red y salida corresponden a la *k*-ésima muestra de entrenamiento de la neurona *j*-ésima de la capa *(L-1)*-ésima, denotados por $u_j^{L-1}(k)$ y $z_j^{L-1}(k)$, respectivamente. La capa de entrada es la capa *0*-ésima. En particular podemos decir que $z_j^0(k) = x_j(k)$. La salida es alimentada dentro del neurón *i*-ésimo de la capa *L*-ésima vía un peso sináptico denotado por $w_{ij}^L(t)$ o por simplemente w_{ij}^L.

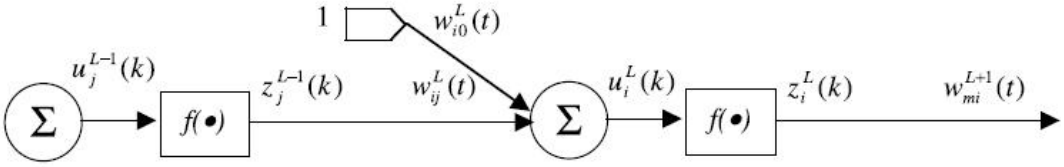


Figura 3.23: Notación para red neuronal MLP de varias capas.

Entonces debemos calcular:

$$\frac{\partial E}{\partial w_{ij}^L} = -2\sum_{k=1}^{K} \frac{\partial E}{\partial u_i^L(k)} \frac{\partial u_i^L(k)}{\partial w_{ij}^L} = -2\sum_{k=1}^{K} \left[\delta_i^L(k) \cdot \frac{\partial}{\partial w_{ij}^L} \sum_m w_{im}^L z_m^{L-1}(k) \right] \tag{3.51a}$$

$$\frac{\partial E}{\partial w_{ij}^L} = -2\sum_{k=1}^{K} \delta_i^L(k) \cdot z_j^{L-1}(k) \tag{3.51b}$$

En la ecuación 3.58, la salida $z_j^{L-1}(k)$ puede ser evaluada aplicando la *k*-ésima muestra de entrenamiento **x***(k)* para el MLP con los pesos fijos $w_{ij}^L(t)$. Sin embargo, el término de error delta $\delta_j^L(k)$ no esta fácilmente disponible y tiene que ser calculado.

Hay que recordar que el error delta $\delta_i^L(k) = \partial E / \partial u_i^L(k)$. En la figura 3.24 podemos ver como la se calcula iterativamente $\delta_i^L(k)$ a partir de δ_i^{L+1} y los pesos de la capa *(L+1)*-ésima.

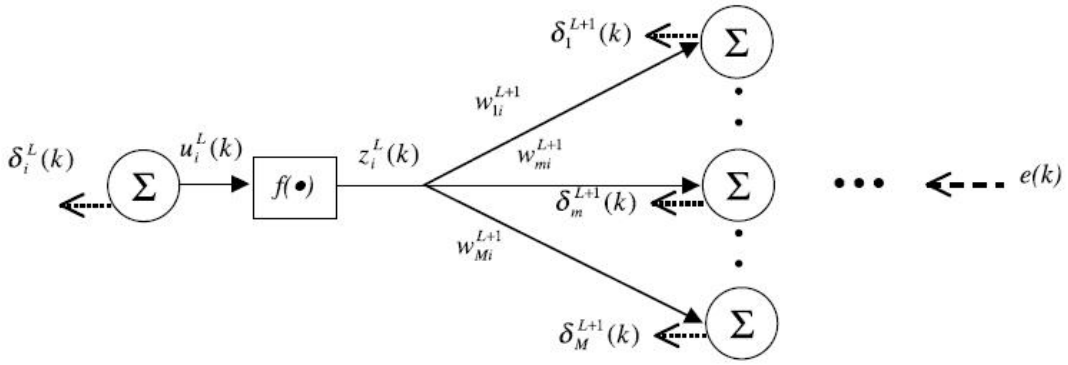


Figura 3.24: Calculo del error con algoritmo de retropropagación.

Notar que $z_i^L(k)$ esta alimentado en todas las neuronas M en la *(L+1)*-ésima capa. Por lo tanto:

$$\delta_i^L(k) = \frac{\partial E}{\partial u_i^L(k)} = \sum_{m=1}^{M} \frac{\partial E}{\partial u_m^{L+1}(k)} \cdot \frac{\partial u_m^{L+1}(k)}{\partial u_i^L} = \sum_{m=1}^{M} \left[\delta_m^{L+1}(k) \cdot \frac{\partial}{\partial u_i^L} \sum_{j=1}^{J} w_{mj}^L f(u_j^{L-1}(k)) \right]$$

$$= f'(u_m^L(k)) \cdot \sum_{m=1}^{M} \delta_m^{L+1}(k) \cdot w_{mi}^L \qquad (3.52)$$

Note that $z_i^L(k)$ is fed into all M neurons in the $(L+1)$th layer. Hence:

$$\delta_i^L(k) = \frac{\partial E}{\partial u_i^L(k)} = \sum_{m=1}^{M} \frac{\partial E}{\partial u_m^{L+1}(k)} \cdot \frac{\partial u_m^{L+1}(k)}{\partial u_i^L(k)} = \sum_{m=1}^{M} \left[\delta_m^{L+1}(k) \cdot \frac{\partial}{\partial u_i^L(k)} \sum_{j=1}^{J} w_{mj}^L f\left(u_j^L(k)\right) \right]$$

$$= f'\left(u_i^L(k)\right) \cdot \sum_{m=1}^{M} \delta_m^{L+1}(k) \cdot w_{mi}^L \,. \qquad (1.16)$$

La ecuación (3.60) es la formula de error de retropropagación que calcula el error delta a partir de la capa salida hacia la capa de entrada capa por capa. Dado el error delta los pesos serán actualizados de acuerdo a la ecuación (3.56) modificada:

$$w_{ij}^L(t+1) = w_{ij}^L(t) + \eta \cdot \sum_{k=1}^{K} \delta_i^L(k) z_j^{L-1}(k) \qquad (3.53)$$

El perceptrón multicapa presenta muy buenas propiedades de generalización y extrapolación una vez entrenado, y presenta una gran consistencia. Sin embargo el proceso de entrenamiento es muy lento, y debido a la utilización de funciones de activación globales, la incorporación de nuevo conocimiento tiende a influir negativamente sobre lo ya aprendido (olvido) si no se hace con mucha precaución.

Esto lo hace muy adecuado para diseñar controladores mediante métodos que trabajen off-line, pero poco interesante para métodos que requieran aprendizaje o perfeccionamiento on-line.

3.3.6. Controladores Neuronales.

Como ya se menciono en secciones anteriores controlar un sistema es hacer que este se comporte de una forma deseada y este "comportamiento deseado" depende de la dinámica del sistema, los actuadores, sensores y otros componentes involucrados. Sin embargo expresar el comportamiento de todos estos componentes en formulaciones matemáticas, es muy complicado y se encuentra además con limitaciones prácticas. Como se mostrará a continuación muchos controladores basados en redes neuronales no necesitan tener los modelos matemáticos para controlar un sistema. Para empezar las redes neuronales tienen dos formas de controlar un sistema:

Diseño de un sistema de control directo: "Directo" significa que el controlador es una red neuronal. Un controlador basado en redes neuronales es frecuentemente ventajoso cuando la plataforma en tiempo real disponible no permite soluciones complicadas. La implementación es simple mientras el diseño y calibración son complicados. Los tipos de control directo son: control directo inverso, control por modelo interno, linealización de retroalimentación, feedforward con modelo inverso, control óptimo.

Diseño de un sistema de control indirecto: Este tipo de diseño esta siempre basado en un modelo. La idea es usar una red neuronal para modelar el sistema a ser controlado. El modelo es entonces empleado en un diseño de controlador más "convencional". El modelo es usualmente entrenado anticipadamente, pero el controlador es diseñado en línea. Los métodos de control indirecto pueden ser por ejemplo: control predictivo, control predictivo no lineal, colocación de polos aproximado, varianza mínima.

3.3.6.1. Control directo con modelo inverso:

Asumiendo que el sistema a ser controlado puede ser descrito por:

$$y(t+1) = g\left[y(t),...,y(t-n+1),u(t),...,u(t-m)\right] \tag{3.54}$$

La red neuronal deseada es entonces aquella que calcula la nueva entrada de control:

$$\hat{u}(t) = g^{-1}\left[y(t+1),y(t),...,y(t-n+1),u(t-1),...,u(t-m)\right] \tag{3.55}$$

Asumiendo que la red neuronal ha sido obtenida, entonces podemos usar para controlar el sistema sustituyendo la salida para el tiempo *t+1* por la referencia *r(t+1).* Si la red neuronal representa la inversa exacta, la entrada de control producida por esta

conducirá entonces la salida del sistema del tiempo *t+1* a *r(t+1)*. Este principio se ilustra en la figura 3.25 para un sistema de primer orden.

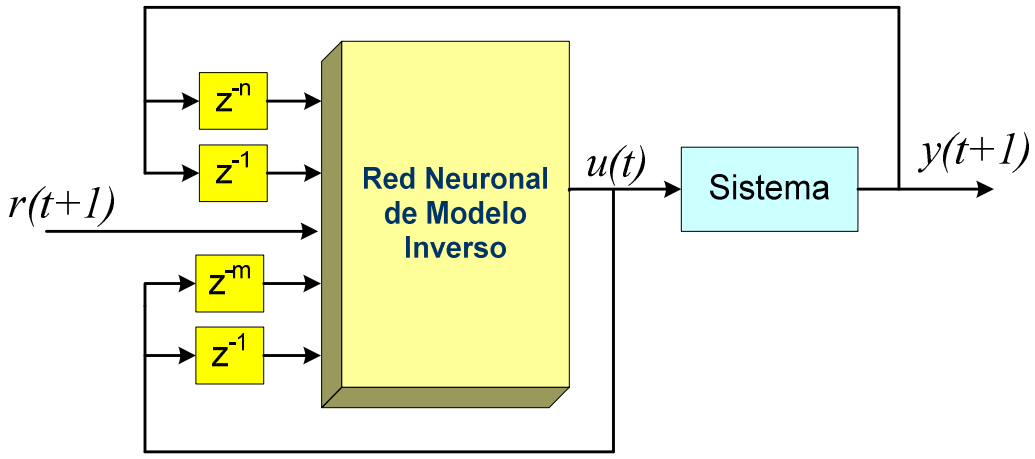


Figura 3.25: Control inverso directo para sistema de primer orden

3.3.6.2. Entrenamiento general:

En este tipo de entrenamiento la red neuronal es entrenada off-line para minimizar el error medio cuadrático entre la señal de control aplicada al sistema en una prueba inicial y la señal de control producida por la red neuronal como se observa en la figura 3.26. Entonces la red neuronal entrenada para minimizar el criterio es la siguiente:

$$J(\theta, Z^N) = \frac{1}{2N}\sum_{t=1}^{N}\left[u(t) - \hat{u}(t\,|\,\theta)\right]^2 \qquad (3.56)$$

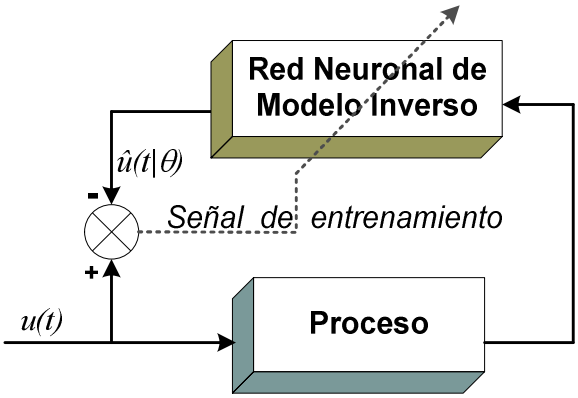


Figura 3.26: Entrenamiento generalizado del modelo inverso.

Esta red neuronal puede ser entrenada utilizando cualquier método de entrenamiento para redes neuronales, en el presente trabajo se utilizara el método de retropropagación, presentado anteriormente.

La red neuronal entrena para aprender el comportamiento inverso del sistema. Para este aprendizaje se tendrá como patrones de entrada la salida la del proceso y como patrones de salida la señal de entrada del proceso.

3.3.6.3. Entrenamiento especializado:

El objetivo en este tipo de entrenamiento es minimizar el error medio cuadrático entre la señal de referencia y la salida del sistema, como se muestra en la ecuación 3.64. Esto se realiza on-line con un algoritmo de entrenamiento recursivo mostrado en la figura 3.27.

$$J(\theta, Z^N) = \frac{1}{2N}\sum_{t=1}^{N}[r(t) - y(t)]^2 \qquad (3.57)$$

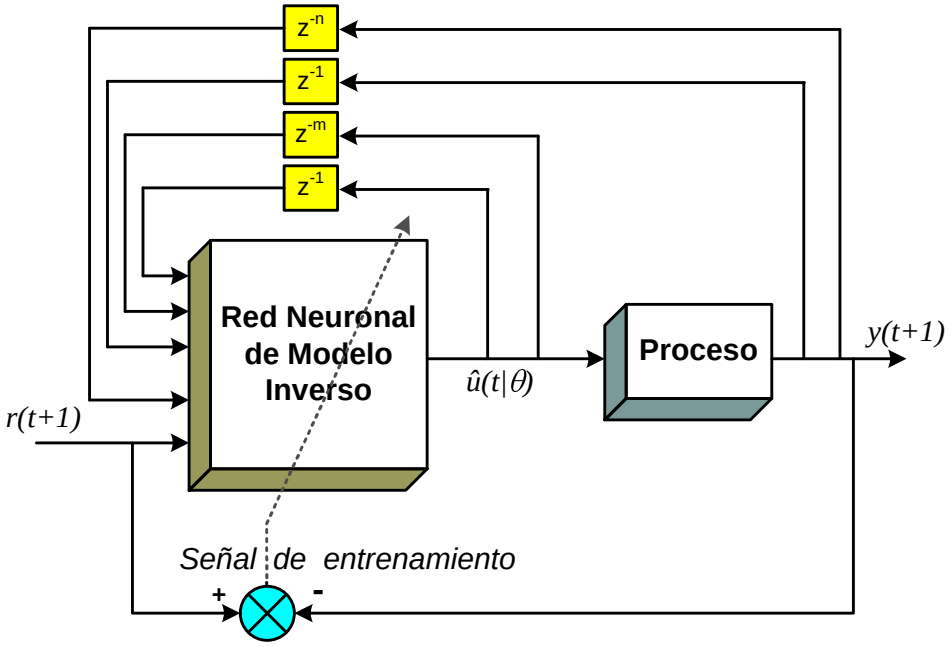


Figura 3.27: Entrenamiento especializado del modelo inverso.

3.4. Algoritmo de control PID-RNA

El algoritmo de control usado se presenta en la figura 3.28, donde se observa un sistema de control clásico PID al cual se le ha agregado un controlador anticipativo basado en una red neuronal artificial, este controlador (al cual llamaremos controlador PID-RNA).

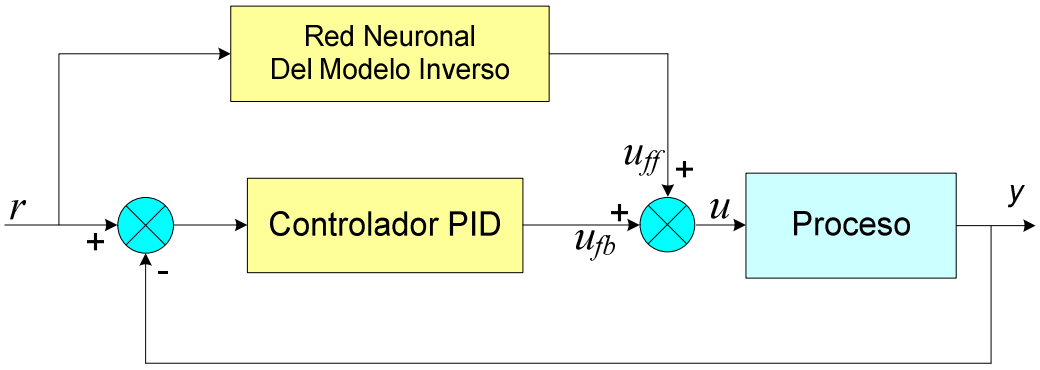


Figura 3.28: Controlador PID-RNA

$$u = u_{ff} + u_{fb} \tag{3.58}$$

Donde u_{fb} es la señal de control del controlador PID y u_{ff} esta dada por:

$$u_{ff}(t) = \hat{g}\left\lfloor r(t+d),...,r(t+d-n),u_{ff}(t-1),...,u_{ff}(t-m+1)\right\rfloor \tag{3.59}$$

Si el modelo inverso es estable, la introducción de un controlador anticipativo no cambiara las propiedades de estabilidad del sistema de control de lazo cerrado. Sin embargo, es difícil saber si el modelo inverso es estable o no.

Capitulo 4

Implementación en tiempo real

4.1. Introducción

El sistema del presente trabajo consta de dos partes: una primera parte de mayor importancia es la de de control de tensión mecánica del papel y una segunda parte es el control de alineamiento del papel en el momento del enrollado. En la figura 4.1 podemos observar el sistema completo y la forma en que interactúan sus diferentes partes. En los próximos incisos se explica detalladamente cada una de las partes del sistema.

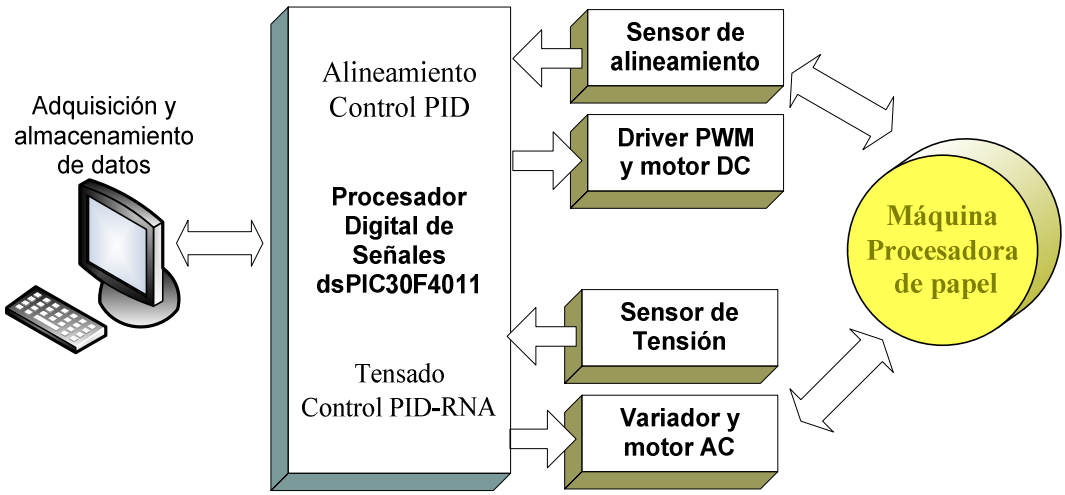


Figura 4.1: Sistema de control de tensado y alineamiento.

Como podemos observar el dsPIC30F4011 se comunica opcionalmente con una computadora por el puerto serial, desde la cual se puede adquirir y almacenar datos del sistema control El dsPIC30F4011 controla ambos tareas de control, el control del tensado y el control del alineamiento, esto se hace a través de una técnica de programación multitareas.

Se diseño e implemento un sistema de control PID-RNA en un procesador digital de señal dsPIC30F4011 de microchip, el cual recibe una referencia o punto de consigna que no es otra cosa que la tensión deseada, la compara con la señal del sensor de tensión y finalmente envía una señal de control al variador de velocidad que controla directamente al motor inducción, como se muestra en la figura 4.2.

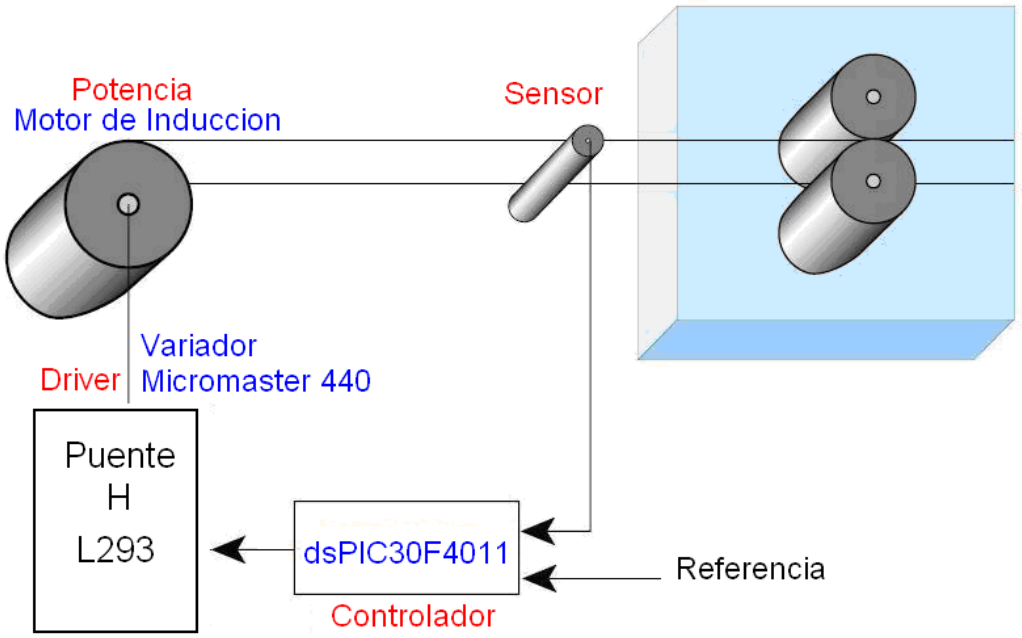


Figura 4.2: Sistema de control de tensado.

En el caso del control de alineamiento se implementa como se vio en la figura 4.1 con el mismo dsPIC30F4011 valiéndonos del procesamiento multitarea, este dsPIC30F4011 recibe la señal del sensor y verifica o compara con el valor preestablecido de posición para así mandar una señal de control al actuador que en este caso es un motor DC con un adaptador de potencia basado en puente H, como se muestra en la figura 4.3.

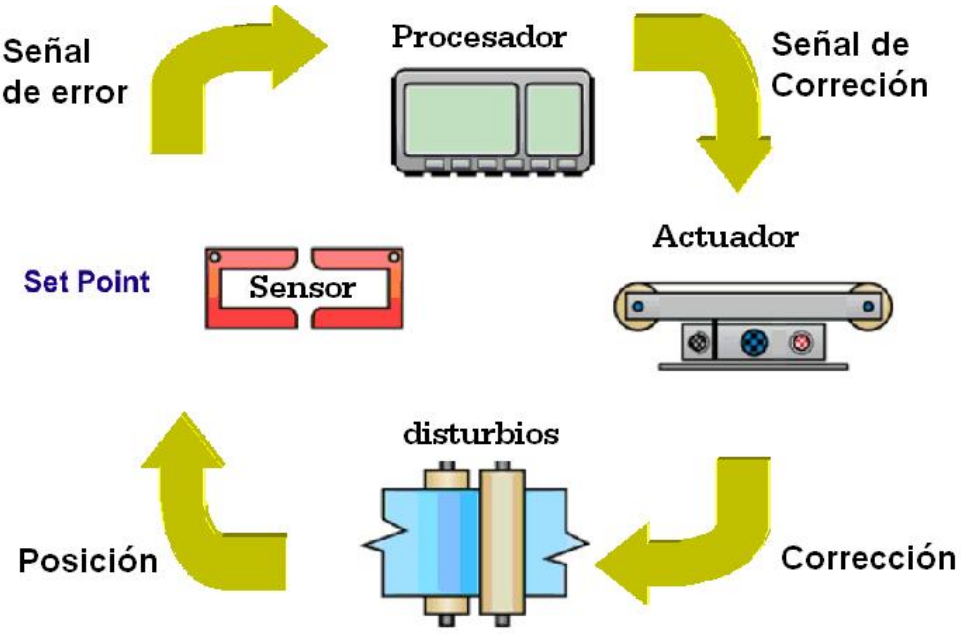


Figura 4.3: Funcionamiento del control de alineamiento.

El motor DC controla un sistema mecánico que guía el papel, este sistema mecánico gira conforme se la señal de control del microcontrolador que controla el alineamiento. El algoritmo de control es simplemente un control proporcional.

4.2. Actuadores y adaptadores de potencia

El actuador utilizado en el caso del control de tensado es un motor de inducción, se presentará una introducción a los motores de corriente alterna para luego describir el motor de inducción utilizado y finalmente se presenta una descripción del adaptador de potencia de este tipo de motor.

4.2.1. Motores de corriente alterna AC

Se denomina motor de corriente alterna a aquellas máquinas eléctricas que funcionan con corriente alterna. Un motor eléctrico de corriente alterna convierte la electricidad en fuerzas de giro por medio de la acción mutua de los campos magnéticos. La mayoría de los motores modernos utilizados en la industria trabajan con fuentes de corriente alterna ya sea polifásica o monofásica.

Existen muchos tipos de motores de AC, en la figura 4.4 se muestra su clasificación. Los motores de corriente alterna de fase simple son usados mayormente en aplicaciones de baja potencia y comercialmente existen desde 5 HP. Los motores de fase simple síncronos son solamente usados por debajo de 1/10 HP. Las aplicaciones usuales son temporización y control de movimiento, donde pequeños valores de torque son requeridos a velocidades específicas.

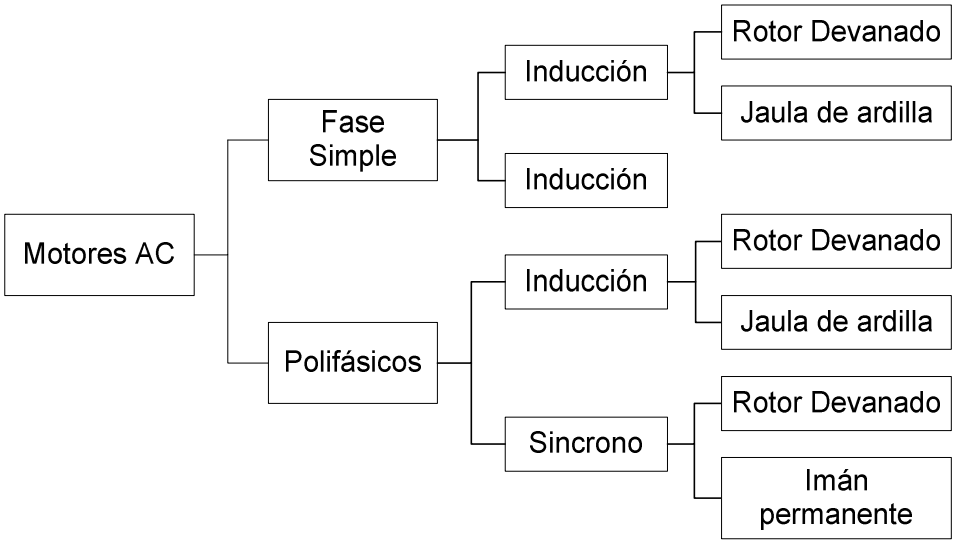


Figura 4.4: Clasificación de motores de corriente alterna.

Los motores de inducción de fase simple son usados en artefactos electrodomésticos y máquina de 1/3 a 5 HP. Por otra parte los motores de corriente alterna polifásicos son principalmente trifásicos y son por mucho los más usados en todo tipo de industria. En el mercado se encuentran desde 5 hasta 50000 HP. El motor trifásico jaula de ardilla de inducción es el más común. Esta comercialmente disponible desde 1 HP hasta varios cientos de HP. Sus aplicaciones más comunes son ventiladores, bombas y manipulación de materiales. En la figura 4.5 se muestra el comportamiento de un motor de inducción.

Cuando la característica torque-velocidad de un motor AC de inducción necesita ser modificada se usa el motor de inducción de rotor devanado. Estos motores reemplazan a los rotores de jaula ardilla por un rotor devanado y anillos deslizantes. Se utilizan resistencias externas para ajustar la característica torque-velocidad para controlar la velocidad en aplicaciones como grúas, montacargas y elevadores.

Los motores trifásicos síncronos pueden ser adquiridos con campo de imán permanente desde arriba de 5 HP y son usados para aplicaciones como líneas de procesamiento y fajas transportadoras a velocidades muy precisas. En el rango de los 10000 HP los motores trifásicos síncronos de campo devanado son usados de preferencia a los motores de inducción de jaula ardilla. La corriente de arranque y otras características pueden ser controladas por un excitador de campo externo. Los motores trifásicos síncronos con campo devanado están disponibles alrededor de los 50000 HP.

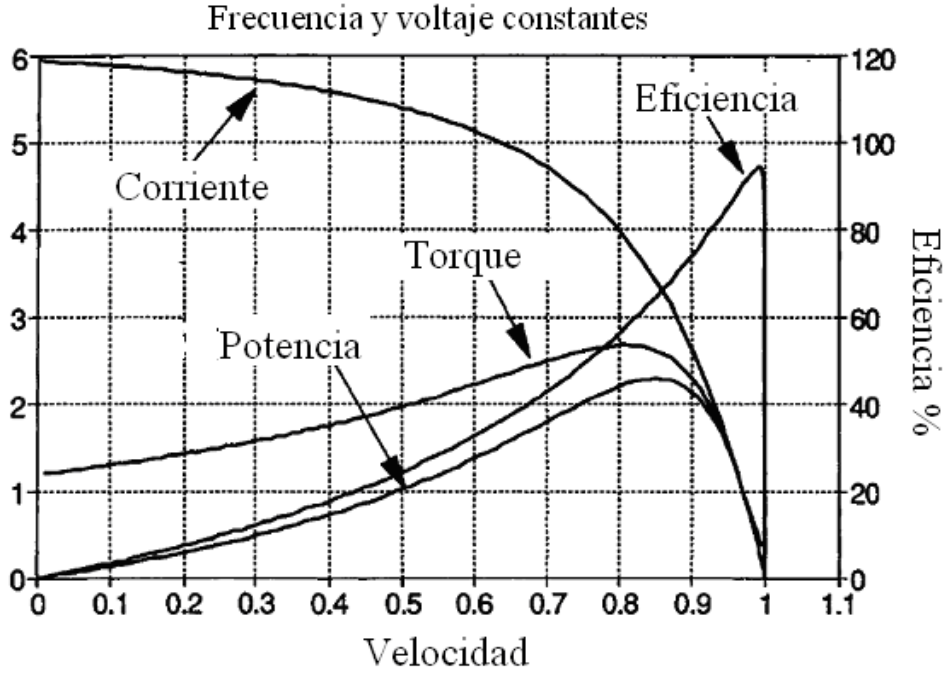


Figura 4.5: Características de operación del motor de inducción.

Otro tipo de clasificación para los motores de corriente alterna es la clasificación NEMA, que es un estándar que designa a los motores de acuerdo a su característica torque-velocidad en A, B, C y D como se muestra en la figura 4.6, el diseño B es el más comúnmente usado.

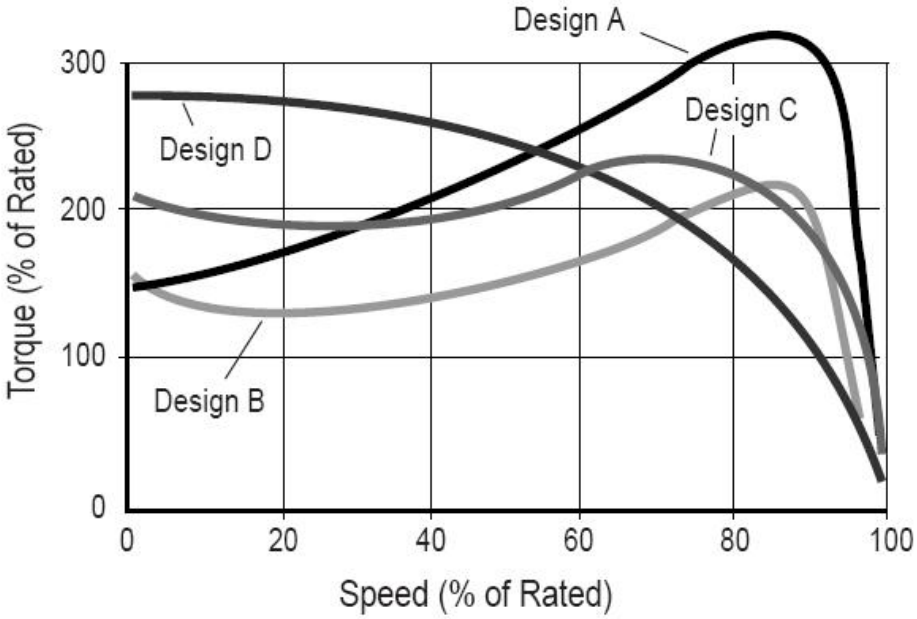


Figura 4.6: Motores por clasificación NEMA.

4.2.2. Variadores de velocidad

Un motor de corriente alterna, a pesar de ser un motor robusto, de poco mantenimiento, liviano e ideal para la mayoría de las aplicaciones industriales, tiene el inconveniente de ser un motor rígido en cuanto a su velocidad. La velocidad del motor asincrónico depende de la forma constructiva del motor y de la frecuencia de alimentación. Como la frecuencia de alimentación que entregan las compañías de electricidad es constante, la velocidad de los motores asincrónicos es constante, salvo que se varíe el número de polos, el resbalamiento o la frecuencia.

El método más eficiente de controlar la velocidad de un motor eléctrico es por medio de un variador electrónico de velocidad frecuencia (también conocidos como variadores de frecuencia). No se requieren motores especiales, son mucho más eficientes y tienen precios cada vez más competitivos. Los variadores de velocidad para motores asincrónicos de corriente alterna son equipos indispensables, ya que los mismos no solo permiten controlar, limitar o aumentar la velocidad de los motores, sino

que además mejoran el rendimiento y protegen al motor tanto eléctrica como mecánicamente, incrementando así su vida útil.

El variador de frecuencia regula la frecuencia del voltaje aplicado al motor, logrando modificar su velocidad. Sin embargo, simultáneamente con el cambio de frecuencia, debe variarse el voltaje aplicado al motor para evitar la saturación del flujo magnético con una elevación de la corriente que dañaría el motor.

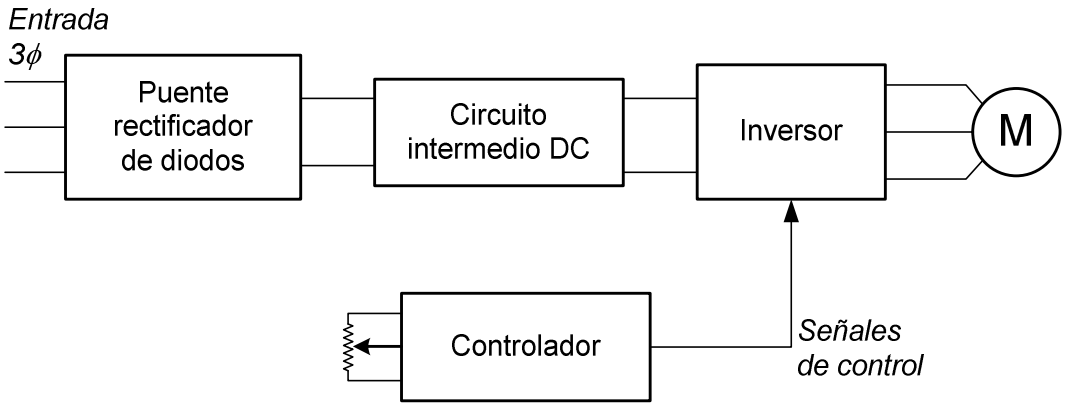


Figura 4.7: Diagrama de bloques de un variador de velocidad.

Un variador de velocidad cumple las funciones antes mencionadas realizando en varias etapas las cuales se muestran en la figura 4.7, así los variadores de frecuencia están compuestos por las siguientes etapas:

Etapa Rectificadora. Convierte la tensión alterna en continua mediante rectificadores de diodos, tiristores, IGBTs y muchos otros componentes electrónicos de potencia.

Etapa intermedia o Filtro. En la etapa intermedia se usan condensadores y bobinas para disminuir las armónicas y mejorar el factor de potencia. Los fabricantes que utilizan bobinas en la línea en lugar del circuito intermedio, pero tienen la desventaja de ocupar más espacio y disminuir la eficiencia del variador.

Inversor. Convierte la tensión continua en otra de tensión y frecuencia variable mediante la generación de pulsos. Actualmente se emplean mucho los IGBTs (Isolated Gate Bipolar Transistors) para generar los pulsos controlados de tensión. Los equipos más modernos utilizan IGBTs inteligentes que incorporan un microprocesador con

todas las protecciones por sobre-corriente, sobre-tensión, baja tensión, cortocircuitos, puesta a masa del motor y hasta sobre-temperaturas.

Etapa de control. Esta etapa controla los IGBT para generar los pulsos variables de tensión y frecuencia. Y además controla los parámetros externos en general. Los variadores mas utilizados utilizan modulación PWM (Modulación de Ancho de Pulsos).

4.2.3. Motor de corriente continua DC

El motor de corriente continua es una máquina que convierte la energía eléctrica en, este caso de corriente continua, en mecánica, en la mayoría de casos es en movimiento rotativo. El motor DC es una de las maquinas eléctricas más versátiles en la industria. Su fácil control de posición, par y velocidad la han convertido en una de las mejores opciones en aplicaciones de control y automatización de procesos. Pero con la llegada de la electrónica han caído en desuso (sobretodo en maquina eléctricas de alta potencia), pues los motores de corriente alterna del tipo asíncrono, pueden ser controlados de igual forma a precios más accesibles para el consumidor medio de la industria. A pesar de esto el uso de motores de corriente continua sigue y se usan en aplicaciones de trenes o tranvías y en aplicaciones donde la potencia requerida no es muy grande.

Una máquina de corriente continua (generador o motor) se compone de dos partes, un estator que da soporte mecánico al aparato y tiene un hueco en el centro generalmente de forma cilíndrica. En el estator además se encuentran los polos, los cuales pueden estar devanados sobre la periferia del estator, o pueden estar de forma saliente. El rotor es generalmente de forma cilíndrica, también devanado.

Existen diferentes tipos de motores DC, en la figura 4.18 se muestra su clasificación. Los motores de imán permanecen ocupan el rango comercial de motores de baja potencia, disponibles en valores aproximados de 10 HP. Por debajo de 1 HP son usados en servo aplicaciones, como en maquinas de herramientas para robótica y periféricos de alta rendimiento para computadoras.

Los motores de campo devanado son usados para aplicaciones encima de 10 HP, están comercialmente disponibles hasta en potencias de varios cientos de HP y son comúnmente usados en tracción o fuerza motriz y otras aplicaciones con necesidad de control de velocidades en un amplio rango.

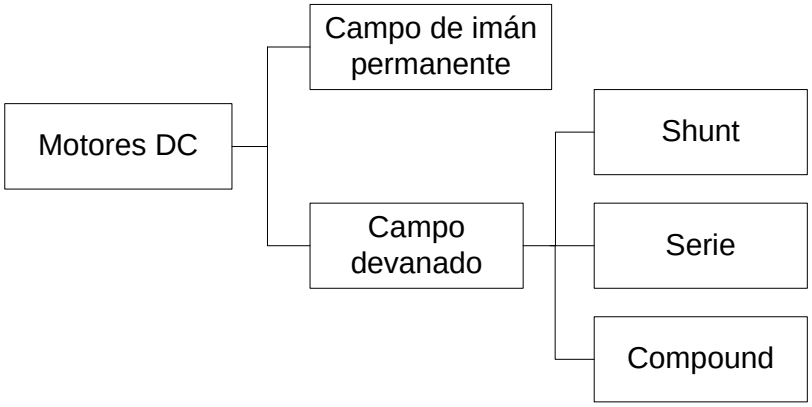


Figura 4.9: Clasificación de motores de corriente continua.

El motor DC shunt es comúnmente usado en aplicaciones industriales como molienda, maquinas herramienta, elevadores y grúas. Los motores de devanado compuesto tienen ambas componentes campo serie y shunt para proveer características específicas de torque velocidad. Motores de vehículos de transito son usualmente motores DC compound.

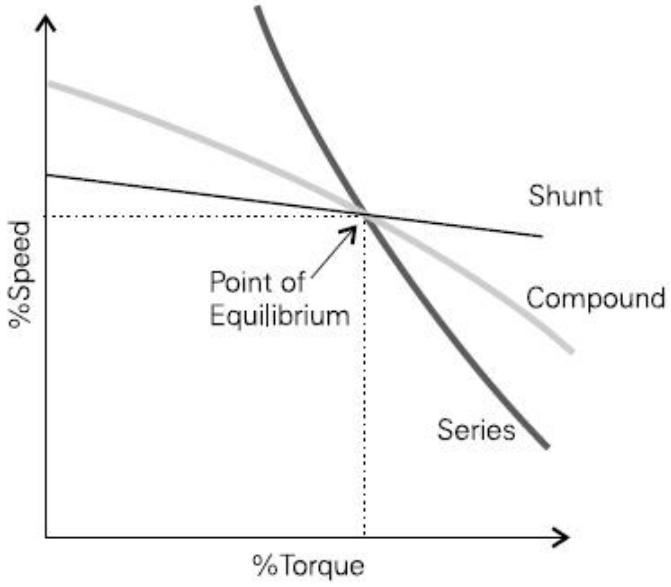


Figura 4.10: Característica torque-velocidad de motores DC.

En la figura 4.10 podemos observar el comportamiento torque-velocidad de los diferentes tipos de motores DC. La principal característica y ventaja del motor de corriente continua es la posibilidad de regular la velocidad desde vacío a plena carga.

4.2.4. Etapa de potencia de motores DC

Los motores de corriente continua son usualmente controlados por modulación de ancho de pulso (PWM), la señal PWM es generada usualmente por un procesador digital o computadora. Sin embargo no se puede conectar directamente el procesador digital a un motor DC, es necesaria una etapa de potencia.

El circuito electrónico de potencia para PWM mas difundido y a la vez utilizado en la industria es el puente H, que es el adaptador de potencia que utilizaremos en el presente trabajo. En la figura 4.11 se muestra la configuración básica de un puente H, donde se observa que esta formado básicamente por cuatro interruptores (usualmente transistores), como se puede observar el puente H además del control PWM permite el cambio de sentido de giro, cerrando los interruptores A y B en un sentido y cerrando C y D para el otro sentido.

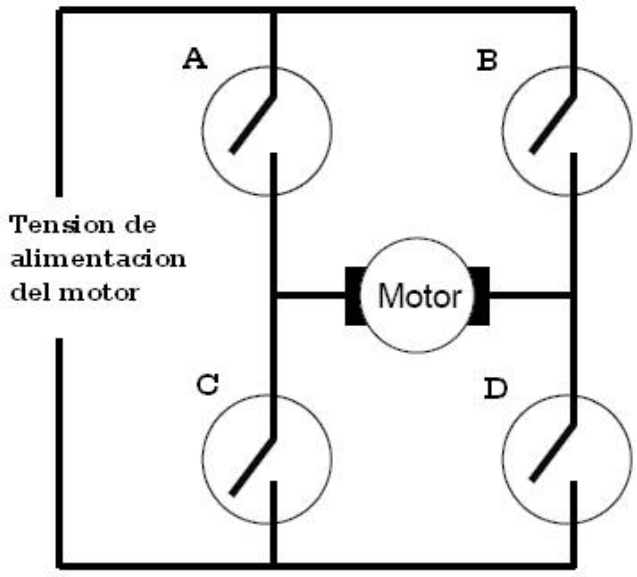


Figura 4.11: Configuración básica de puente H.

En el presente trabajo se utilizó el puente H en circuitos integrados L293 con el cuales facilitan la implementación de esta etapa. Un L293 nos provee dos mitades de puente H separados, que al configurar adecuadamente podemos obtener un puente H completo como se muestra en la figura 4.12.

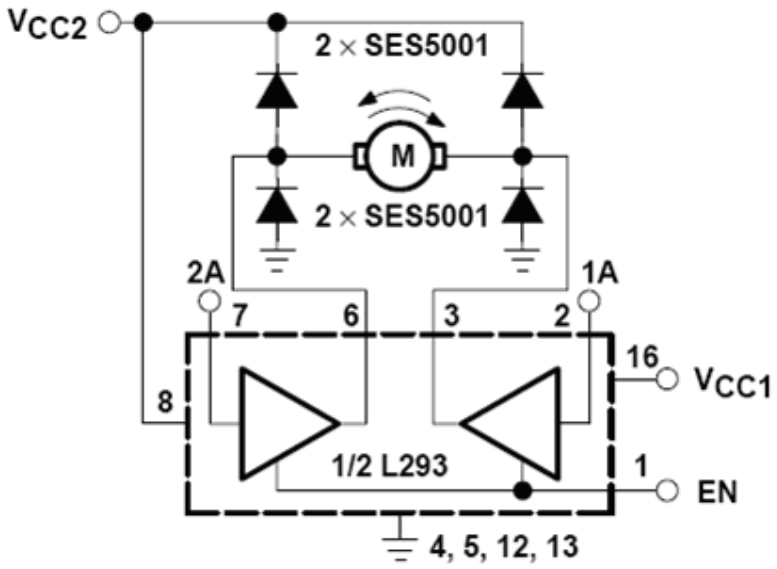


Figura 4.12: Puente H en circuito integrado L293.

4.3. Sensores y acondicionamiento de señal

4.3.1. Sensor de tensión

Relacionado con el control de tensión es el de conocer la magnitud que debemos controlar, es decir la medición de tensión mecánica, en este trabajo tomaremos como definición de tensión a la fuerza aplicada a una cinta continua en la dirección de procesamiento de la maquina.

Típicamente la tensión mecánica se mide en unidades de fuerza por unidad de área, como se muestra la ecuación 4.1, es medida generalmente en newtons por metro cuadrado (pascal) o en libras por pulgada cuadrada (psi, pounds per square inch). Pero como trabajaremos con materiales donde generalmente el espesor es constante y además muy pequeño comparado con el ancho del material, entonces podemos considerar solamente la longitud, definiendo así la tensión lineal T_L de un material como cantidad de fuerza por unidad de longitud, es muy común medir la longitud lineal en PLI (pounds per linear inch), como se muestra en le ecuación 4.2.

$$Tension = \frac{Fuerza}{Area} \tag{4.1}$$

$$T_L = \frac{Fuerza}{Longitud} \tag{4.2}$$

Como se observa en la figura 4.13 tenemos muchas variedades de sensores de tensión, varían de acuerdo a la aplicación. Comercialmente existen básicamente dos formas de obtener la tensión de un proceso: directa e indirectamente. Directamente se logra mediante sensores con celdas de carga. Indirectamente puede lograrse mediante sensores de posición tipo brazo danzarín, otra forma de medir la tensión de forma indirecta pero esta vez en lazo abierto se logra con sensores que miden el diámetro del rollo de papel. A continuación describiremos cada uno de estos tipos de medición de tensión.

Figura 4.13: Sensores de tensión para diferentes materiales.

Medición de tensión directa con celda de carga:

Un primer tipo de sensor mide la tensión mediante una celda de carga (load cell) que mide la fuerza aplicada a la celda directamente como se muestra en la figura 4.9a). En la tabla 4.1 se mencionan las ventajas y desventajas de este tipo de sensor.

Tabla 4.1: Ventajas y desventajas de una celda de carga.

Aplicaciones	Ventajas	Desventajas
Maquinas de cortar y ahuecar	Medición de tensión directa	No tiene absorción de picos de tensión
Para material pesado	Mecánicamente bien integrado	Aceleración/deceleración no es fácil de manejar
Espacio limitado	No es móvil.	Difícil empalme aéreo
Picos de tensión aceptados		No tiene rápida aceleración/deceleración

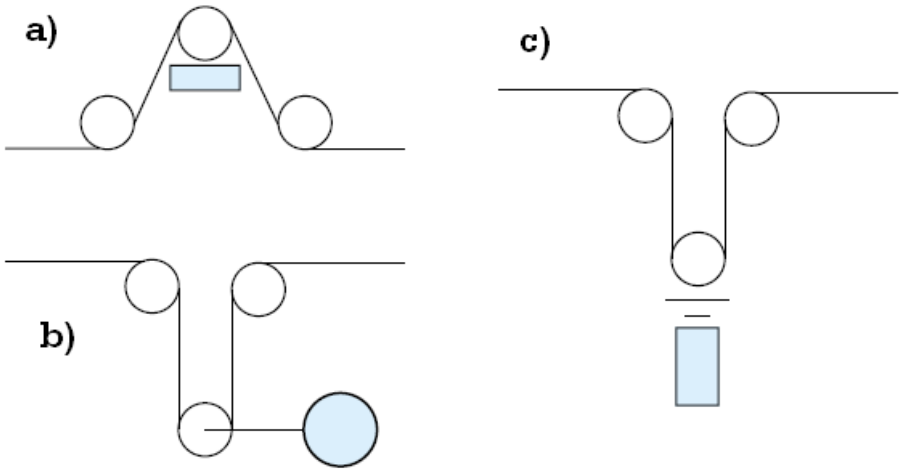


Figura 4.14: Métodos para medición de tensión.

Medición de tensión indirecta con brazo danzarín:

Un segundo método es median un sensor de brazo danzarín o (arm dancer), este sensor mide la posición de un rodillo por donde pasa la cinta, como se muestra en las figuras 4.9a) y 4.9b), este sensor se mueve de acuerdo a la fuerza con la que tira la cinta, obteniendo así de una manera indirecta la tensión de la cinta.

Tabla 4.2: Ventajas y desventajas de un sensor de brazo danzarín.

Aplicaciones	Ventajas	Desventajas
Impresión	Absorbe picos de tensión	Necesita mas espacio
Función intermitente	Puede actuar como acumulador	Partes móviles
Empalme aéreo necesario	Fácil montaje aéreo	
	Fase de aceleración / deceleración bien absorbida	
	Flexibilidad	

En la figura 4.15 se muestra el montaje de un sensor tipo de brazo danzarín fabricado con un potenciómetro cuyo eje esta conectado al brazo móvil o danzarín el cual se mueve proporcionalmente con la tensión del papel.

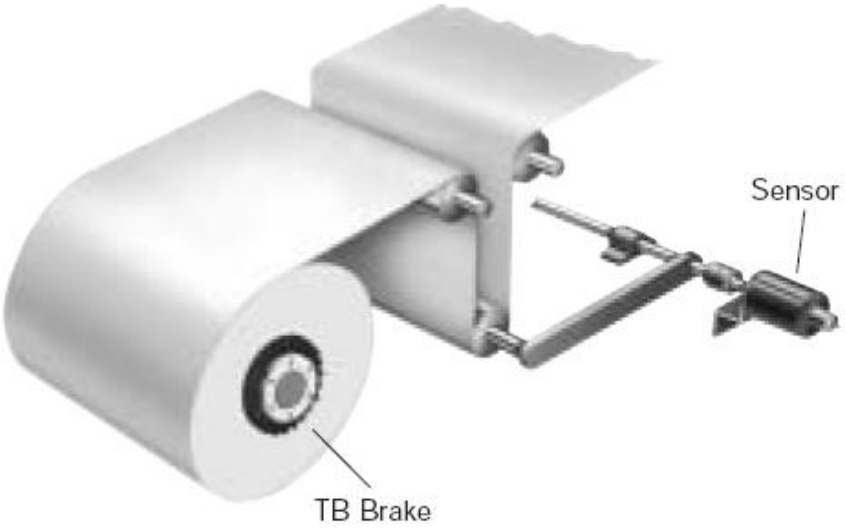


Figura 4.15: Sensor tipo brazo danzarín.

Medición de tensión indirecta en lazo abierto.

Finalmente un tercer método que no es más que una variante del anterior, es medir la posición con un sensor donde no haya contacto con el rodillo, como un sensor de distancia óptico o ultrasónico como se muestra en la figura 4.14c.

Tabla 4.3: Ventajas y desventajas de sensores en lazo abierto.

Aplicaciones	Ventajas	Desventajas
Máquinas textiles	Absorbe picos de tensión	Difícil posición de lectura confiable
Muy baja tensión	Puede actuar como acumulador	Necesita mas espacio
	Fácil montaje aéreo	Partes móviles
	Fase de aceleración/deceleración bien absorbida	
	Flexibilidad	

En la presente tesis, la tensión se mide mediante un sensor de brazo danzarín fabricado con un potenciómetro, lo cual permite un coste bajo y un funcionamiento aceptable. El potenciómetro es alimentado con 5 Voltios y se conecta directamente a una de las entradas analógicas del dsPIC30F4011.

4.3.2. Sensor de alineamiento.

Para controlar el alineamiento en una maquina de procesamiento de papel, usualmente se usan sensores que detectan los bordes del papel, los sensores mas comunes en estos casos son los sensores ópticos, pero también se utilizan sensores de ultrasonido. Se puede colocar uno o dos sensores fijos o móviles como se muestra en la figura 4.15. También se puede determinar el alineamiento de mediante marcas (usualmente una línea de referencia) en la propia cinta que determinen su alineamiento, en este último caso solo se utilizan solamente sensores ópticos.

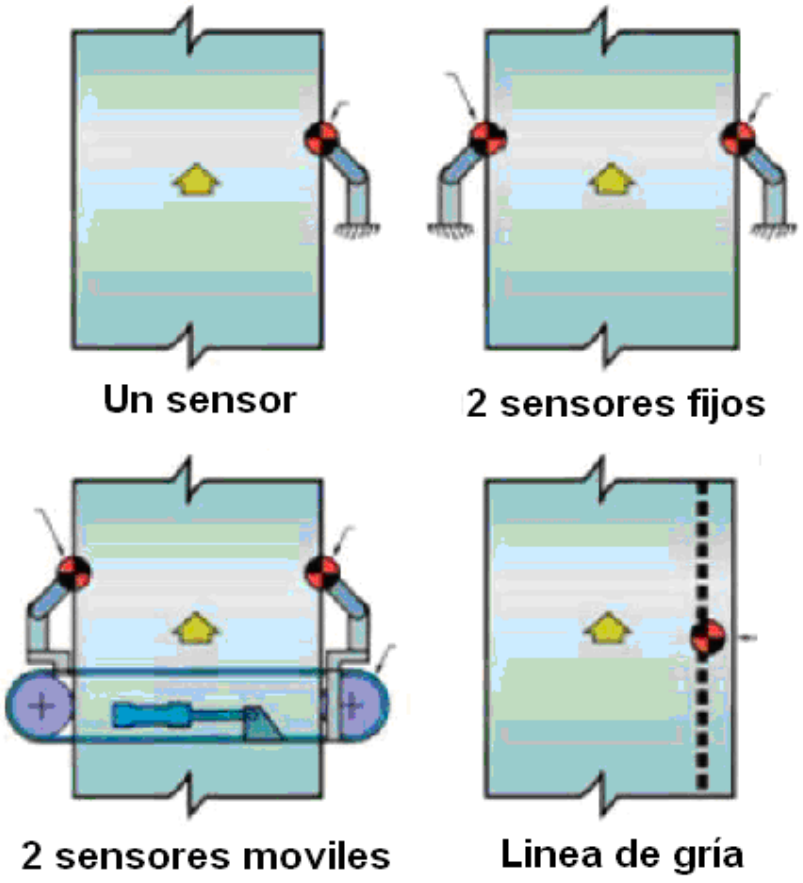


Figura 4.16: Tipos de sensores de alineamiento.

Un Sensor fotoeléctrico de borde utiliza generalmente un LED que indica la posición del material. Cuando el material esta en el centro del sensor, el LED Y el material mantendrá su posición por el control. Existen muchos tipos de sensores de alineamiento como se puede observar en la figura 4.17, cada uno de ellos es aplicable a diferentes tipos de procesos, en su mayoría tienen salida analógica de 4 a 20 mA que nos indica el nivel de alineamiento.

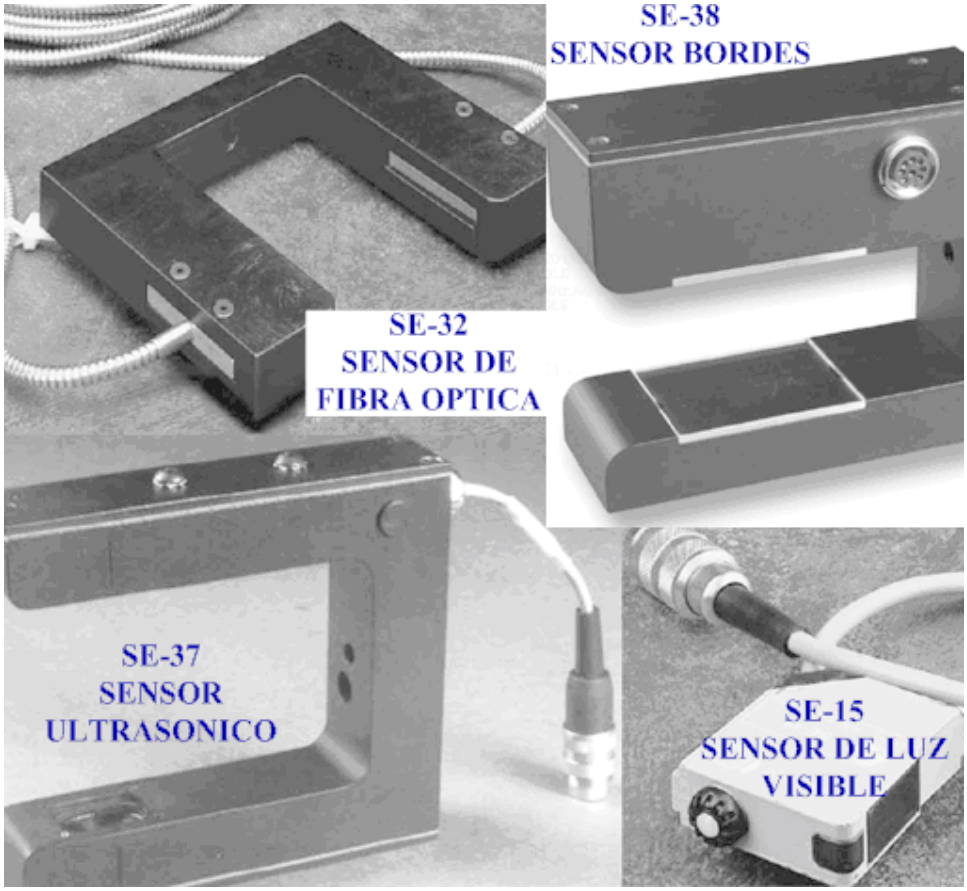


Figura 4.17: Sensores de alineamiento.

El sensor debe ser instalado lo más cerca posible de la salida del rodillo del actuador utilizado para el alineamiento, sin importar el tipo del sistema de guiado que se utilice, en la figura 4.18 se observan las formas correcta e incorrecta de la ubicación del sensor de alineamiento, en dos tipos de actuadores.

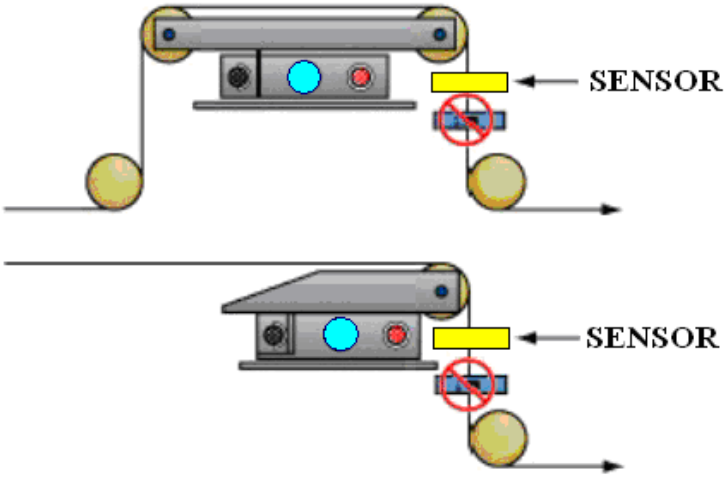


Figura 4.18: Ubicación correcta del sensor de alineamiento.

En la presente tesis, el alineamiento se mide mediante un sensor infrarrojo extraído de una impresora en desuso, el sensor consta de un transmisor y un receptor infrarrojo, el transmisor es alimentado con 5 Voltios y el receptor se conecta directamente una de las entradas analógicas del dsPIC30F4011.

4.4. Programación del controlador

El dsPIC30F4011 se programó mayoritariamente en lenguaje ANSI C, sin embargo también se utilizó lenguaje ensamblador pero en menor escala y solo para las subrutinas donde el tiempo fue crítico. El lenguaje ANSI C y el lenguaje ensamblador para programar el procesador digital de señales son especificados por Microchip para cada una de sus familias de circuitos integrados, proveyendo así manuales y hojas de datos en su página web.

El programa del dsPIC esta basado en un esquema de procesamiento multitarea que se presenta en la figura 4.19, donde cada tarea se ejecuta cada cierto tiempo y con una duración de un milisegundo como máximo, el tiempo está determinado por el temporizador 1 del dsPIC. Se han implementado cuatro tareas, las cuales se ejecutan con una determinada frecuencia, como se explica a continuación:

La tarea 0 se realiza cada 4 milisegundos 250 veces por segundo

La tarea 1 se realiza cada 8 milisegundos 125 veces por segundo.

La tarea 2 se realiza cada 16 milisegundos 62.5 veces por segundo.

La tarea 3 se realiza cada 32 milisegundos 21.25 veces por segundo

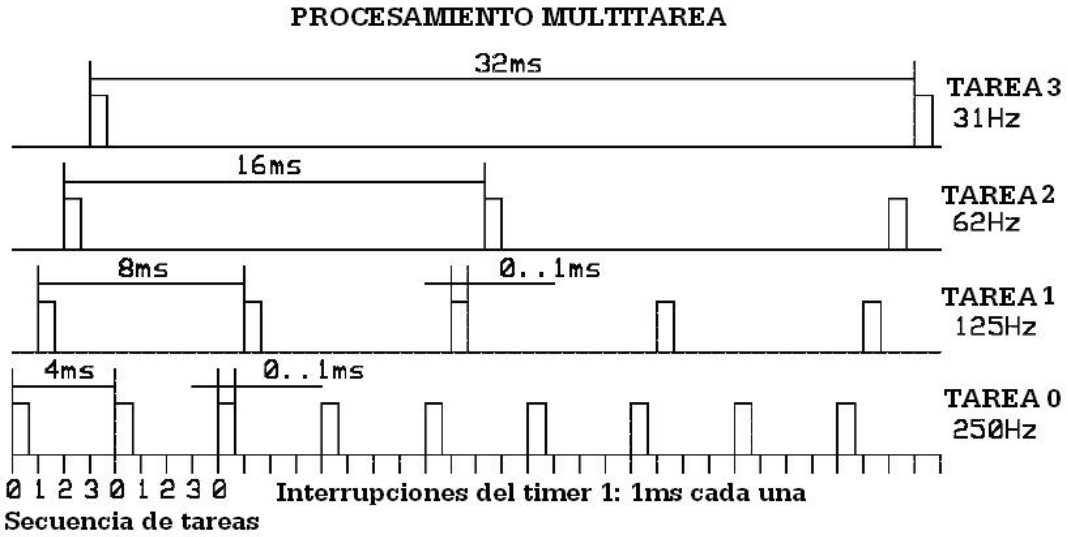


Figura 4.19: Diagrama de tiempos de procesamiento multitarea en el dsPIC.

Los procesos del procesador digital de señales entonces se colocaran en una de estas tareas, dependiendo de cuan frecuentemente se desea que se ejecuten determinados procesos. En la tarea 0 está el programa del controlador de alineamiento, en la tarea 1 el control de tensado, las comunicaciones en la tares 2 y en la tarea 3 se ha puesto rutinas de verificaciones del funcionamiento correcto del dsPIC.

Actualmente todos los controladores industriales son digitales, es por esto que el controlador PID-RNA se implementa digitalmente y se diseña en tiempo discreto. El diagrama de bloques de la figura 4.20 muestra el diseño del controlador digital, donde se observa que la señal de referencia ya es digital, y la señal de control digital se pasa a tiempo continuo para que actué sobre el proceso y la salida del proceso es adquirida por el sensor y digitalizada por un conversor analógico a digital para que el procesador compare la referencia con esta salida.

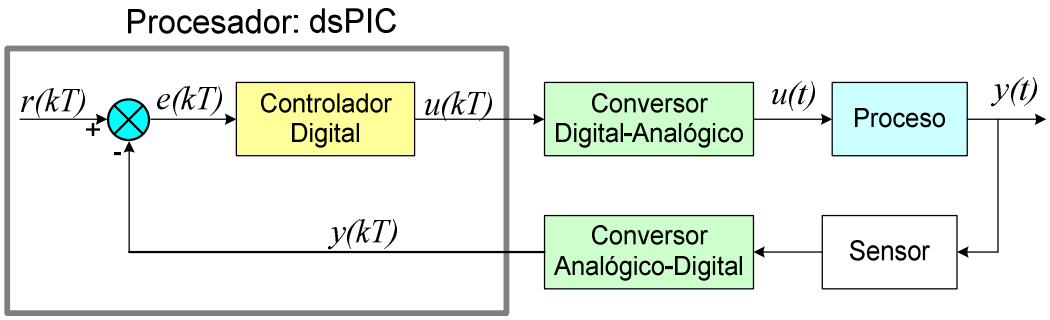


Figura 4.20: Controlador digital.

El controlador digital genera una señal de control de la forma:

$$u(kT) = u(k) = f(u(k-1), u(k-2), ..., e(k), e(k-1), e(k-2), ...) \quad (4.3)$$

Ahora debemos hacer que el controlador PID tenga esta forma, es decir este en función del error y señales de control pasadas, para esto utilizaremos la transformada Z, y la ecuación (3.7) vista en el capitulo anterior, de la cual obtenemos la señal de control:

$$u = K_p e + K_d \frac{de}{dt} + K_i \int e dt \quad (4.4)$$

Donde el componente proporcional *Kp* se implementa en forma digital mediante una multiplicación digital de la función e*(t)* en *t=kT* por la constante *Kp*, esto se realiza con un procesador preferentemente que tenga capacidad de operación de punto flotante.

Mientras que la derivada con respecto al tiempo, se puede implementar en un procesador digital con la ayuda de la regla de diferenciación hacia atrás, donde se aproxima la derivada de e*(t)* en *t=kT* de la siguiente forma:

$$\left.\frac{de(t)}{dt}\right|_{t=kT} = \frac{1}{T}\left[e(kT) - e((k-1)/T)\right] \tag{4.5}$$

$$G_d(z) = K_d\,\frac{z-1}{Tz} \tag{4.6}$$

Para aproximar la forma digital del controlador integral, existen un numero de reglas de integración numérica las cuales pueden ser usadas, como son: integración trapezoidal, integración rectangular hacia delante e integración rectangular hacia atrás.

En nuestro caso escogeremos por simplicidad de implementación la integración rectangular hacia adelante, donde se aproxima el área bajo de *e(t)*, mediante rectángulos como muestra la figura 4.21. Entonces la integral de *e(t)* en $t = kT$ se aproxima por:

$$u(kT) = u\left[(k-1)T\right] + Te(kT) \tag{4.7}$$

Al tomar la transformada z en ambos miembros de la ecuación (4.7), la función de transferencia del integrador digital empleando la regla rectangular hacia delante queda de la siguiente forma:

$$G_i(z) = K_i\,\frac{U(z)}{e(z)} = \frac{K_i Tz}{z-1} \tag{4.8}$$

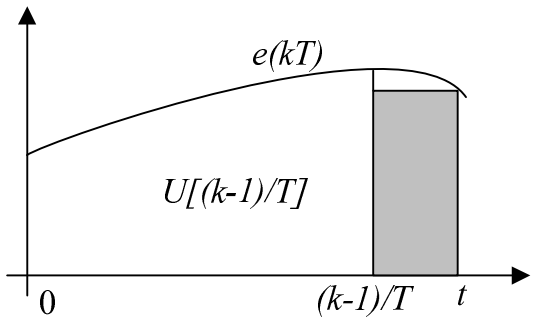


Figura 4.21: Regla de integración rectangular hacia delante

Finalmente la función de transferencia del controlador PID, queda como se muestra en la ecuación 4.9, esta función de transferencia será la que finalmente se implementará en el procesador digital de señales.

$$G_c(z) = \frac{\left(K_p + K_d/T + TK_i\right)z^2 - \left(K_p + K_d/T\right)z + K_d/T}{z(z-1)} \tag{4.9}$$

Una vez que la función de transferencia esta ya determinada es posible implementarla en un procesador digital, microcontrolador o computadora. En la figura

4.22 se muestra el diagrama de bloques del controlador PID donde el componente z^{-1} se interpreta como un tiempo de retardo de T segundos. En la práctica el retardo se implementa mediante el almacenamiento de la variable en la memoria del procesador y se la toma después de T segundos.

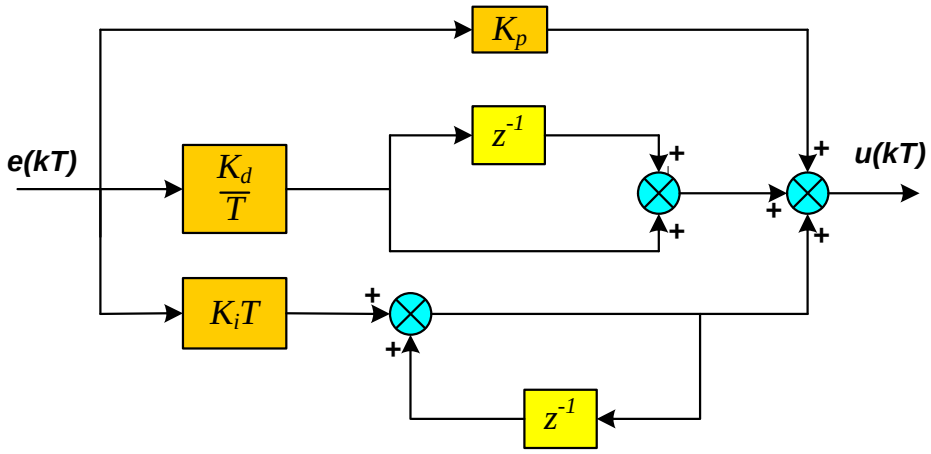


Figura 4.22: Diagrama de bloques del algoritmo PID.

Sin embargo esta no es la señal de control final, la señal de control final es como se especificó en la ecuación 3.58 en el capitulo anterior, donde la señal de control PID es sumada con la señal de control de la red neuronal feedforward.

La estructura de la red neuronal se escogió heurísticamente, dicha estructura consta de siete neuronas de entrada con función de activación *tanh* y una neurona de salida con función de activación lineal tal como se muestra en la figura 4.23. Se utilizó un algoritmo de entrenamiento generalizado por retropropagación.

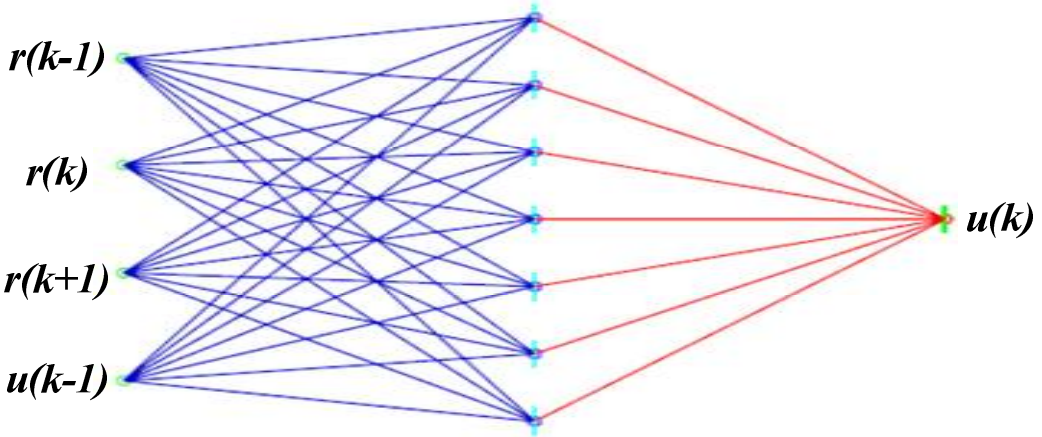


Figura 4.23: Red Neuronal implementada.

El programa para el controlador PID-RNA final diseñado e implementado se muestra en la figura 4.24, donde se observa las todas las partes funciones en forma resumida del controlador PID-RNA. Notemos que la Red Neuronal de modelo inverso no tiene ninguna retroalimentación, es decir es un controlador anticipativo o feedforward (*ff*), en cambio el controlador PID retroalimentado o feedback (*fb*).

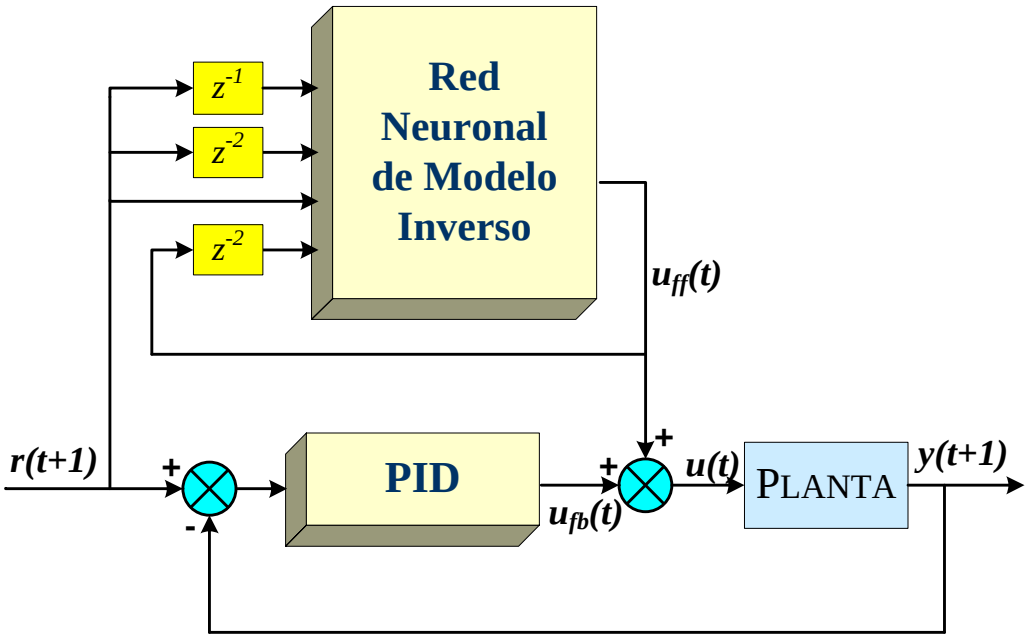


Figura 4.24: Algoritmo de control PID-RNA implementado.

4.5. Entrenamiento de la red neuronal

La red neuronal ha sido entrenada utilizando el software de computadora Matlab, donde se desarrollo un programa para satisfacer las necesidades de la maquina procesadora de papel, los datos inicialmente fueron tomados del modelo de la planta, sin embargo al tener los datos de la planta real disponibles, se utilizó estos datos para el entrenamiento. El programa del entrenamiento se puede apreciar en el anexo A.2.

En la figura 4.25 se observa el comportamiento de la red neuronal sin entrenar, notamos que no tiene un buen desempeño, presentando picos muy elevados y un tiempo de establecimiento considerable, además de no tiene ni exactitud ni precisión, en su respuesta.

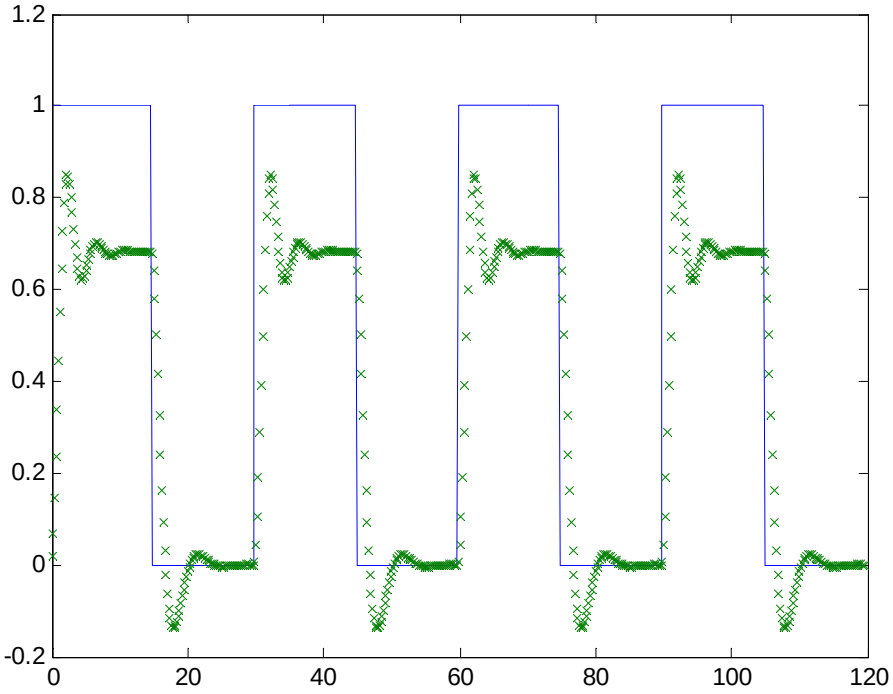


Figura 4.25: Respuesta de la red neuronal inversa sin entrenar.

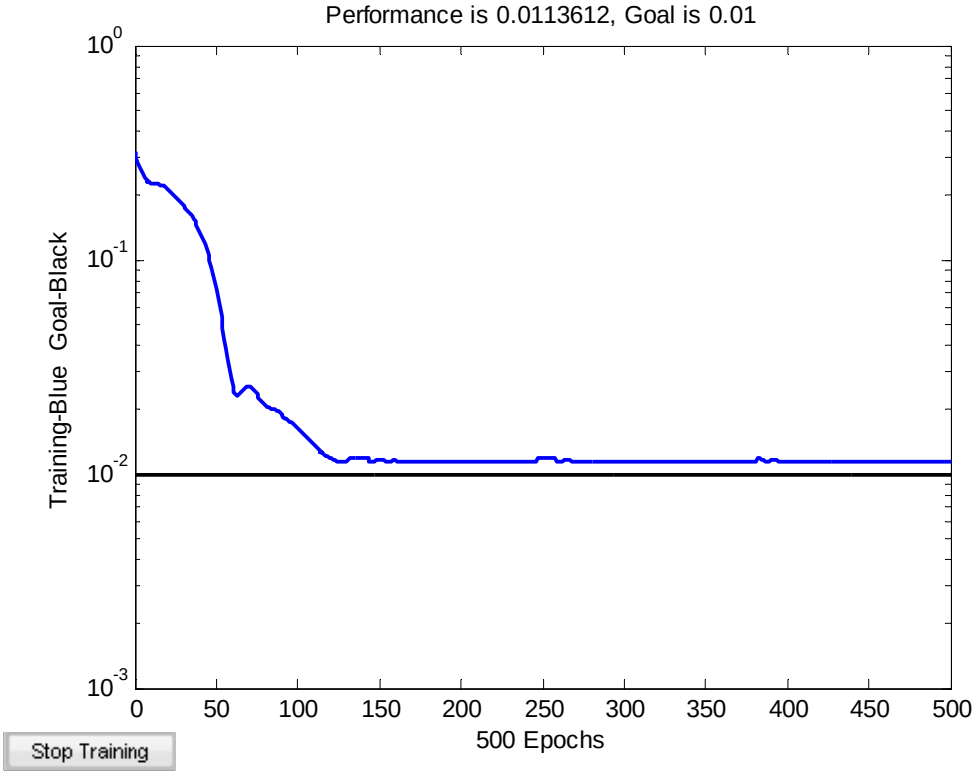


Figura 4.26: Rendimiento de la red neuronal versus épocas.

En la figura 4.26 se observa el rendimiento de la red neuronal en cada época, en cada época mejora el comportamiento de la red neuronal sin embargo a partir de la época 200, la mejora ya no es significativa. Adicionalmente se muestra los resultados numericos obtenidos de Matlab del entrenamiento:

```
TRAINGDX, Epoch 0/2000, MSE 7.46437/0.01, Gradient 14.7023/1e-006
TRAINGDX, Epoch 25/2000, MSE 0.0573882/0.01, Gradient 0.272606/1e-006
TRAINGDX, Epoch 50/2000, MSE 0.0277394/0.01, Gradient 0.102687/1e-006
TRAINGDX, Epoch 75/2000, MSE 0.0209544/0.01, Gradient 0.0371186/1e-006
...
TRAINGDX, Epoch 450/2000, MSE 0.0113368/0.01, Gradient 0.00532752/1e-006
TRAINGDX, Epoch 475/2000, MSE 0.0113338/0.01, Gradient 0.00221793/1e-006
TRAINGDX, Epoch 500/2000, MSE 0.0113281/0.01, Gradient 0.00084995/1e-006
```

Finalmente en la figura 4.27 se muestra el comportamiento de la red neuronal ya entrenada, donde la respuesta a una función escalón es muy cercana a la deseada, esta el la red neuronal que utilizaremos para el desarrollo del controlador neuronal.

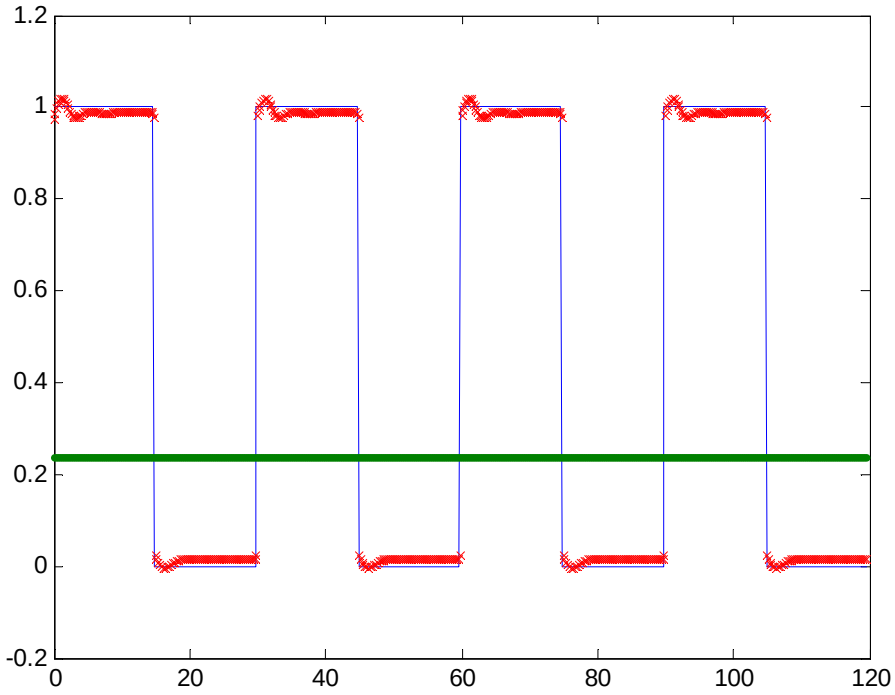


Figura 4.27: Respuesta de la red neuronal inversa entrenada.

4.6. Circuito eléctrico del controlador

El circuito eléctrico fue diseñado y desarrollado con la ayuda del software EAGLE versión 4.28, con el cual se diseño también las placas de circuito impreso. Esta placa consta de un MAX232, para efectos de comunicar el dsPIC30F4011 con otros procesadores, también tiene un buffer 74HC245 para tener salidas digitales protegidas,

un latch D 74HC573 para las entradas de señales digitales y finalmente las entradas analógicas están directamente conectadas al dsPIC30F4011.

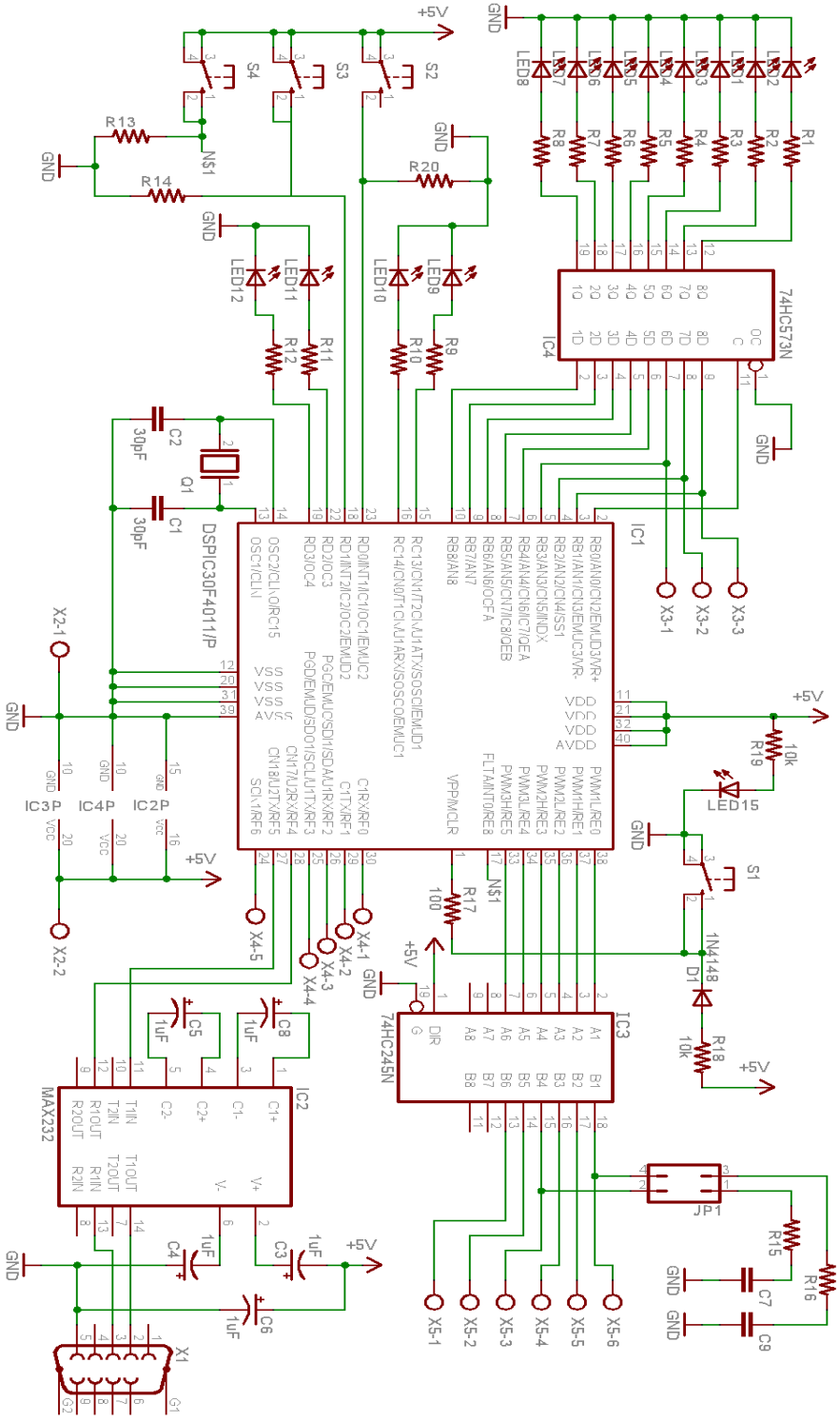


Figura 4.28: Circuito eléctrico del controlador digital.

Capitulo 5

Simulaciones

En este capitulo se muestra las simulaciones del controlador PID-RNA y a la vez se le compara con las simulaciones del controlador PID sólo. Los resultados de las simulaciones son mostrados de una sola forma, en la parte superior se muestra la señal de control y en la parte inferior la referencia o punto de consigna y la respuesta de la planta.

Para ambos controladores PID solo y PID-RNA se han utilizado los mismos parámetros para el controlador PID, planta y ruido, para que la comparación sea mas exacta y podamos observar claramente las diferencias.

5.1. Simulación de la planta

Primeramente se ha simulado la respuesta de la planta sin ningún controlador, para esto se ha colocado un switch que aísla en controlador PID en el diagrama de bloques de Simulink como se muestra en la figura 5.1, así podemos obtener lo que se desea.

La planta simulada esta implementada con las ecuaciones deducidas en el capítulo 2, de modelado matemático del proceso. La respuesta se puede observar en parte inferior de la figura 5.2 donde se notamos que la planta no responde bien a la referencia y el ruido causa muchas deformaciones en la salida de la planta.

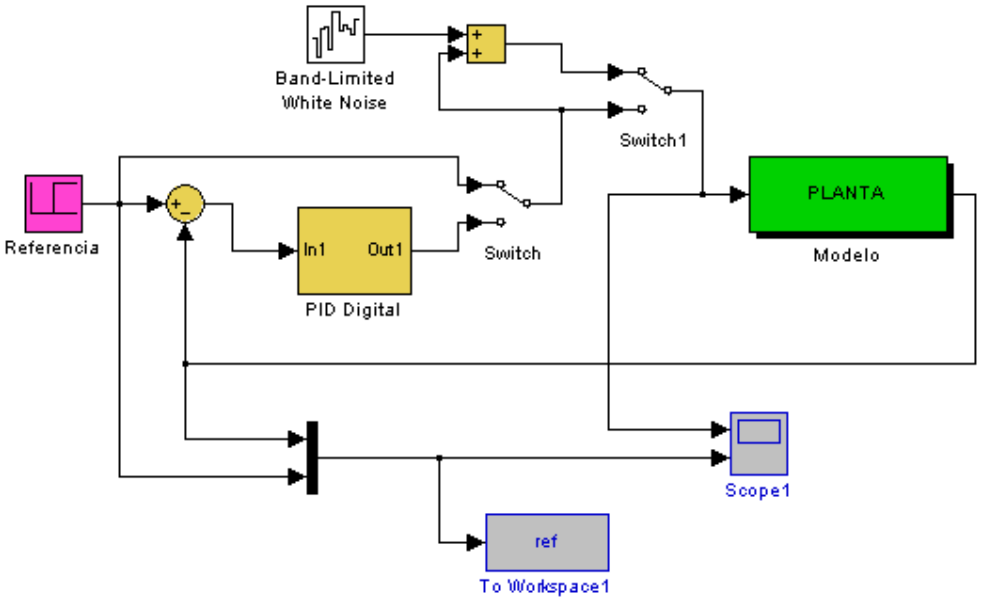


Figura 5.1: Diagrama de bloques en Simulink.

Figura 5.2: Respuesta de la planta a un escalón.

5.2. Simulación del controlador PID

En la figura 5.3 se observa el diagrama de bloques simulado en Simulink, donde se observa un diagrama de bloques de un controlador convencional, el diagrama de bloques del algoritmo de control PID se muestra en la figura 5.4.

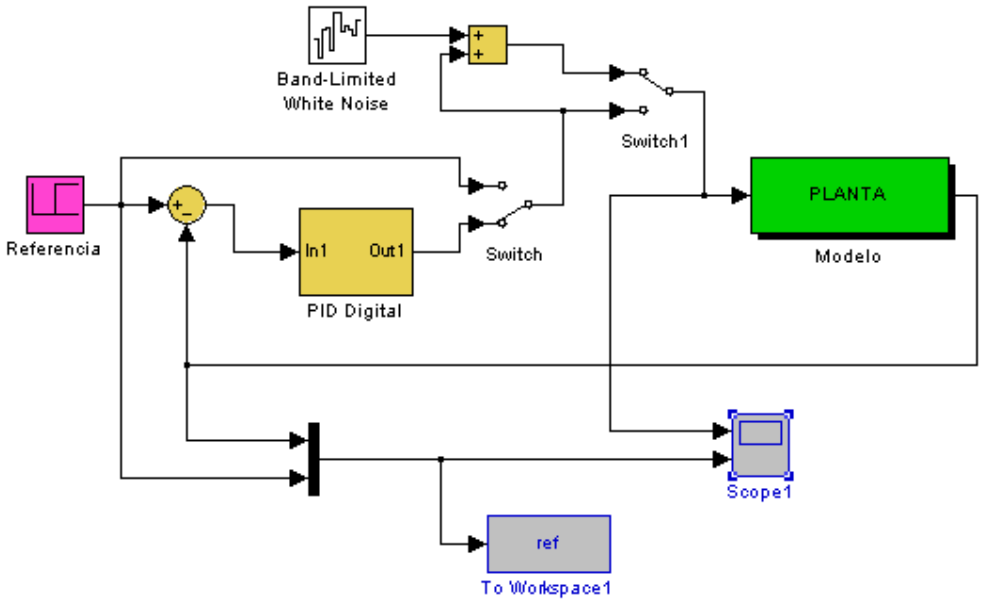


Figura 5.3: Diagrama de bloques en Simulink del controlador PID.

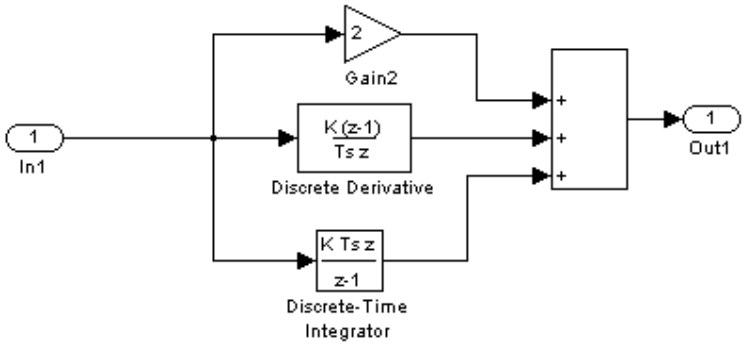


Figura 5.4: Sub-bloque PID Digital o algoritmo de control.

Los resultados de simulación del controlador PID se muestran en la figura 5.5, donde observamos que la respuesta a la referencia, en este caso un escalón unitario, notamos que es buena, es decir es rápida y tiene un sobre-pico aceptable, sin embargo el rechazo a ruido del sistema no es muy bueno se observa que la salida del sistema presenta gran cantidad de ruido y de gran amplitud.

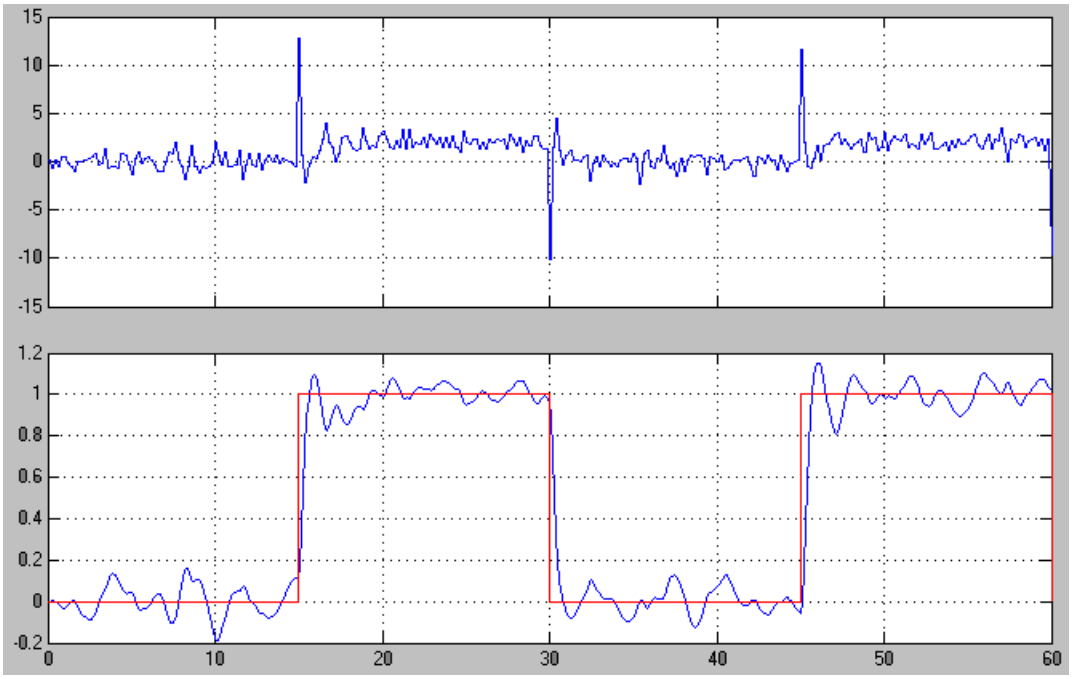


Figura 5.5: Resultados para el controlador PID.

5.3. Simulación del controlador PID-RNA

Ahora veremos los resultados de la simulación del control PID-RNA, El diagrama de bloques de este controlador se muestra en la figura 5.6, donde observamos la que la red neuronal es alimentada por la referencia, la misma referencia retardada y su propia

salida retardada también. Como se mencionó anteriormente al controlador PID se le agrega la señal del controlador neuronal. También hemos incluido un generador de ruido con los mismos parámetros que el que se utilizó en el controlador PID solo.

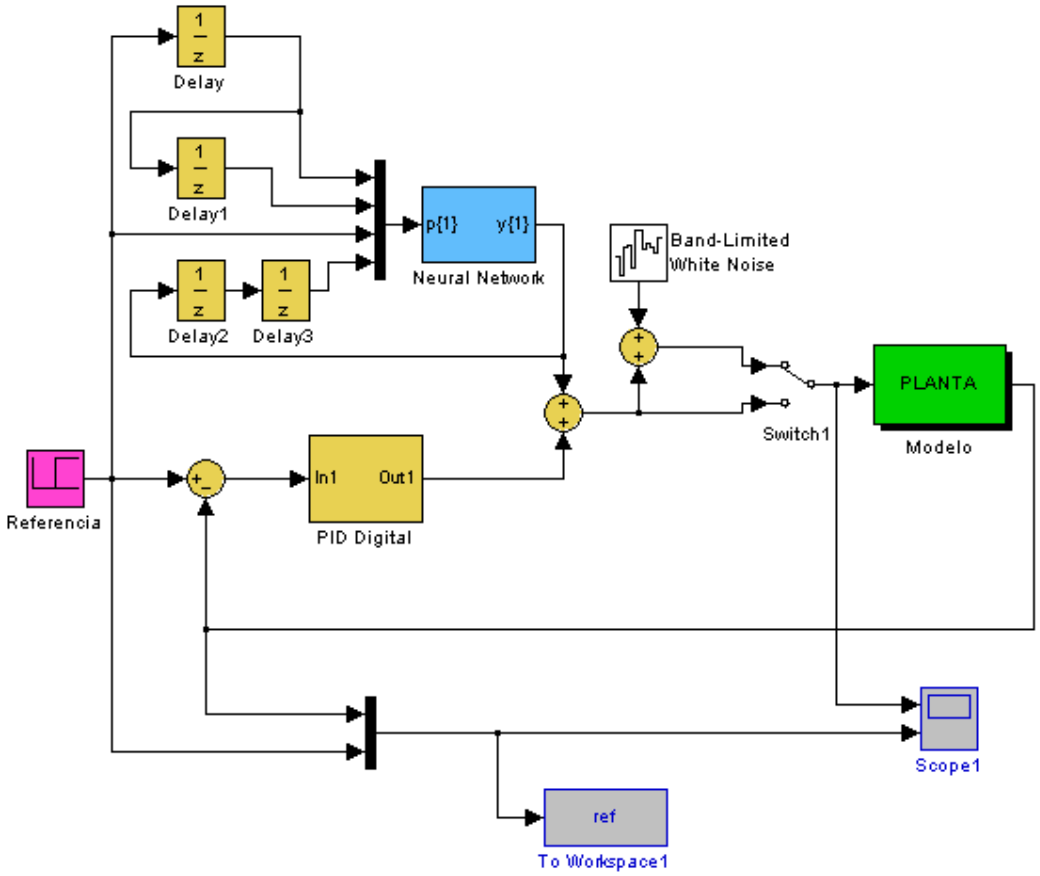


Figura 5.6: Diagrama de bloques Simulink del controlador PID-RNA.

La red neuronal utilizada para la simulación fue creada con la ayuda de comando gensim del Matlab, el cual luego de entrenar la red neuronal, crea su respectivo modelo para Simulink, el cual podemos observar en las figuras 5.7, 5.8 y 5.9. En las figuras 5.8 y 5.9 en ambos casos se muestran en la parte superior la red neuronal en forma resumida, y en la parte inferior se desarrolla con mayor detalle el bloque de los pesos.

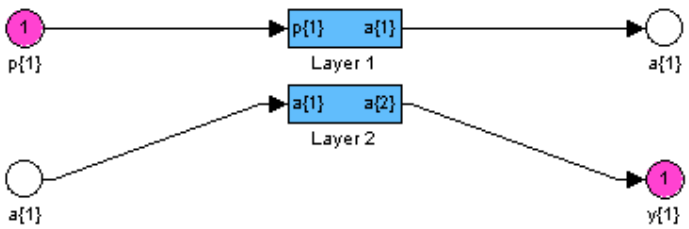


Figura 5.7: Red neuronal de dos capas.

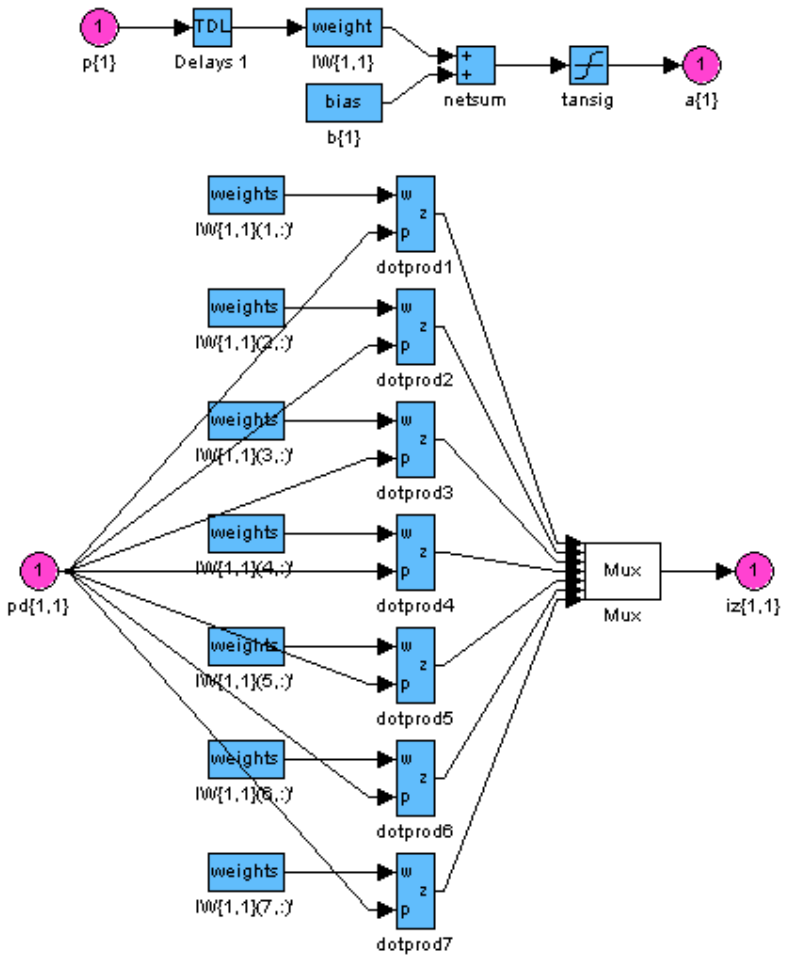


Figura 5.8: Primera capa de la red neuronal.

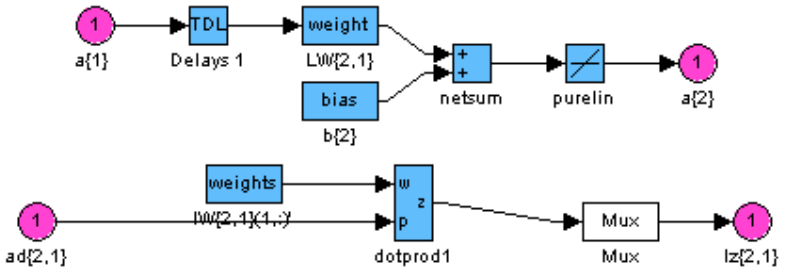


Figura 5.9: Segunda capa de la red neuronal.

Los resultados del controlador PID-RNA se muestran en la figura 5.7, donde se observa que la respuesta al escalón, es mucho mejor que la del controlador PID solo, el rechazo a ruido también mejora notablemente.

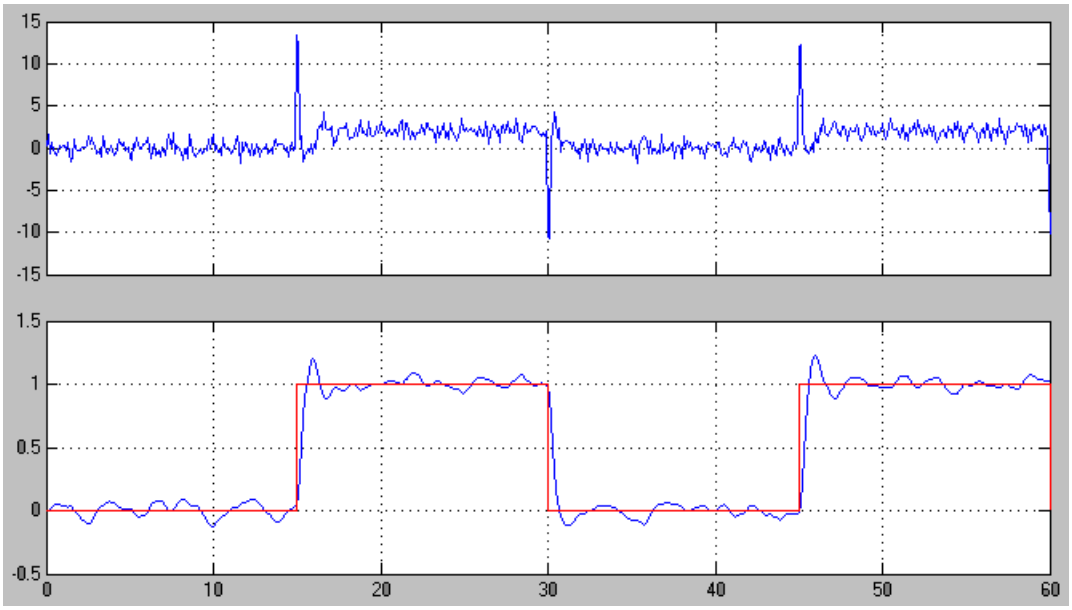


Figura 5.10: Resultados para el controlador PID-RNA.

Capitulo 6

Resultados experimentales

Los resultados experimentales para el sistema de control propuesto se muestran en las figuras 6.1 a 6.5. Inicialmente mostraremos los resultados del controlador PID, luego los resultados del controlador PID-RNA, para posteriormente compararlos.

Para la adquisición de datos del dsPIC30F4011 se utilizo una computadora personal, con software para controlar las comunicaciones con el dsPIC30F4011 a través del puerto serie RS-232. Los datos fueron adquiridos y almacenados, con un programa hecho en Visual Basic, en archivos de formato de Microsoft Excel, para poder ser graficados como se muestra en este capitulo.

6.1. Resultados del controlador PID

Los resultados se muestran

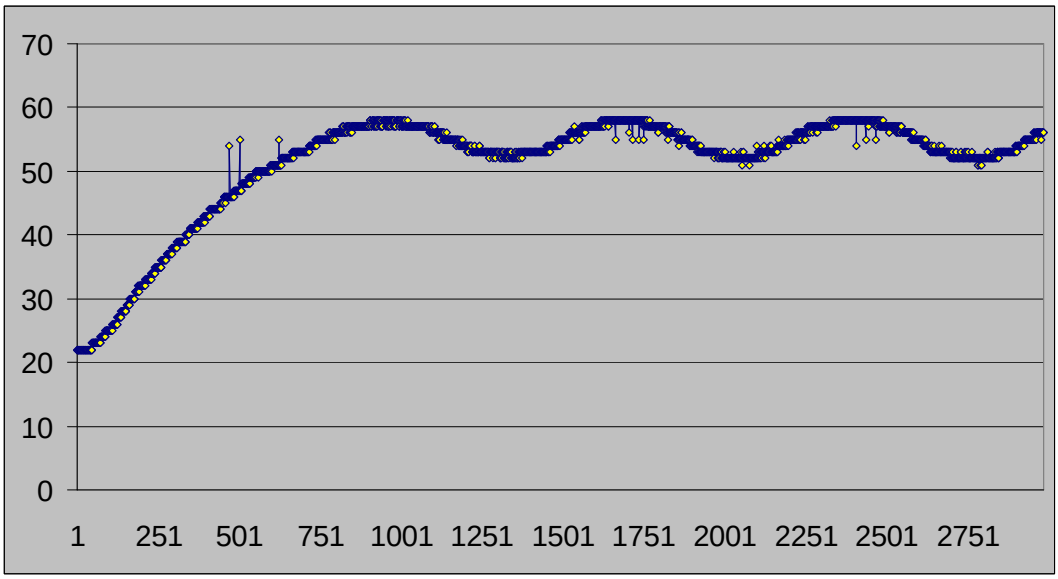


Figura 6.1: Respuesta del control PID.

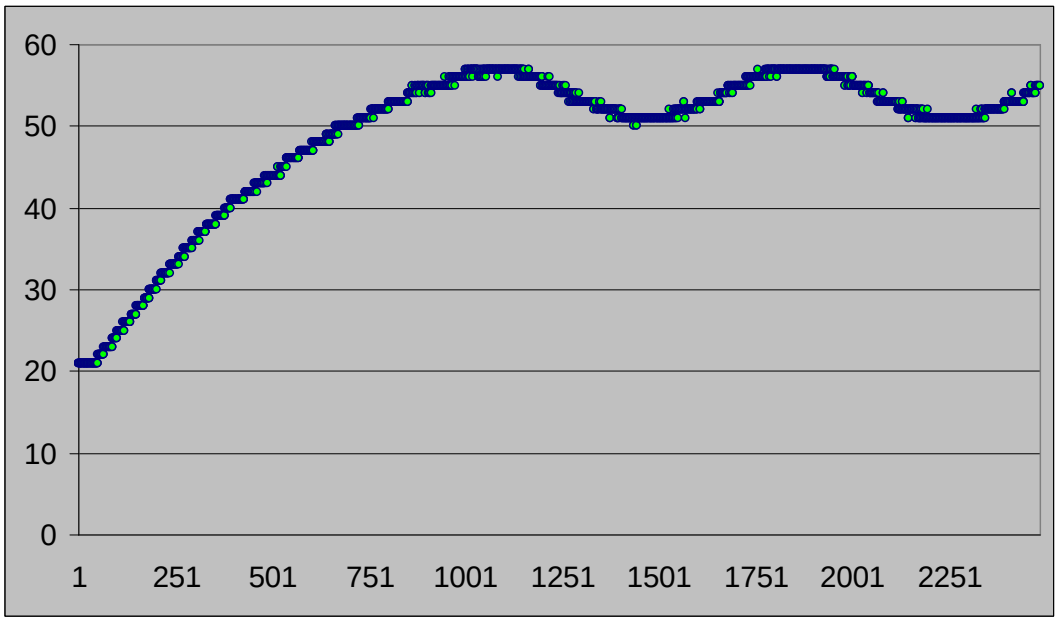


Figura 6.2: Respuesta del control PID con filtro.

6.2. Resultados del controlador PID-RNA

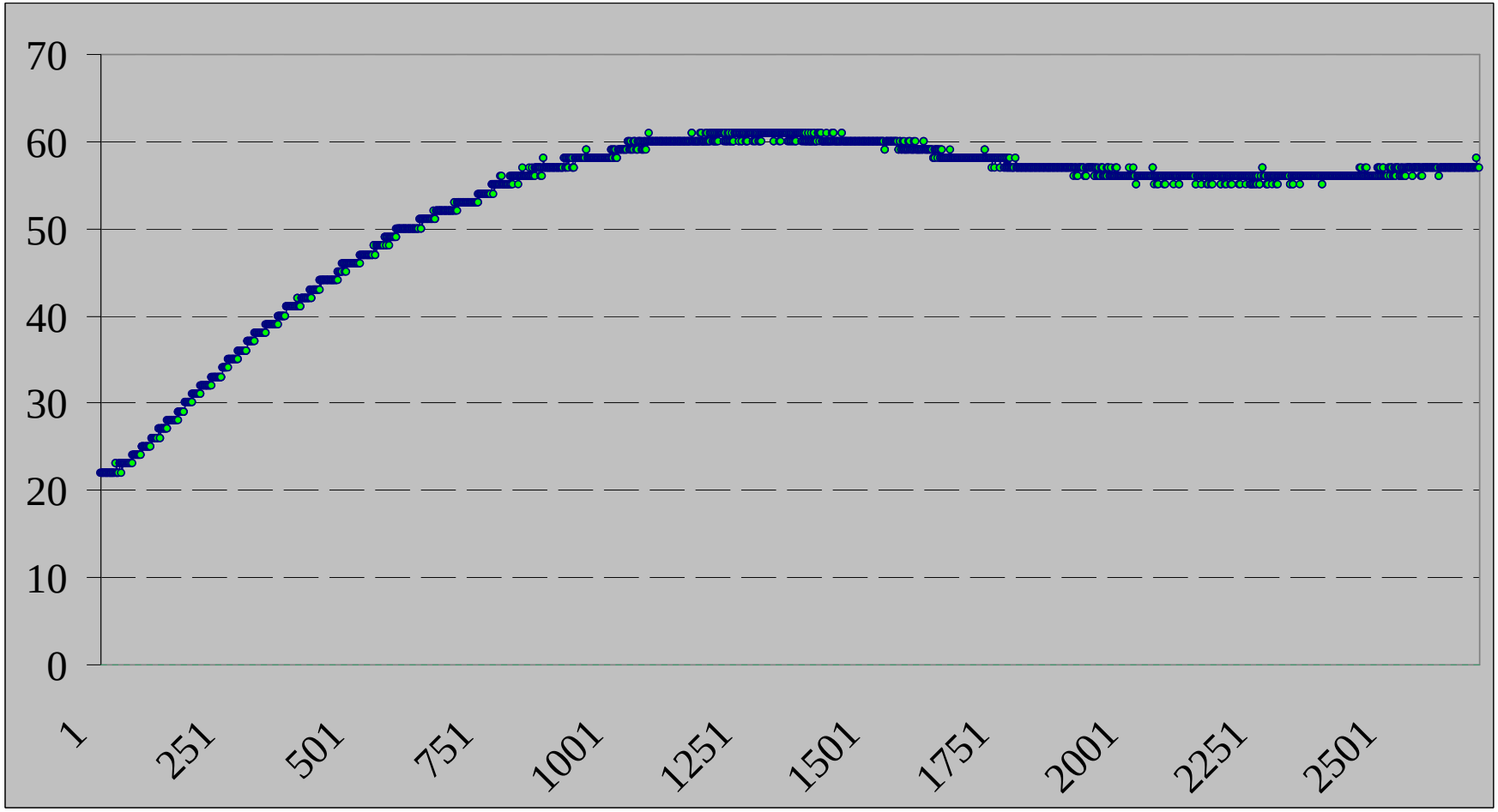


Figura 6.3: Respuesta al impulso del controlador PID-RNA.

Capitulo 7

Conclusiones

En este capitulo se presenta un resumen de las principales conclusiones derivadas de este trabajo, así como también recomendaciones para investigaciones posteriores relacionadas con el tema de automatización u otras áreas afines.

6.1. Conclusiones

[1] Se mostró la forma de aplicación de conceptos de redes neuronales artificiales a sistemas de control, además también se probó que se pueden utilizar en combinación con algoritmos de control clásico como el control PID.

[2] La solución obtenida se puede aplicar a cualquier tipo de proceso donde el ruido sea considerable, adicionalmente si ya se tiene un controlador PID sintonizado actuando sobre algún determinado proceso, se puede adicionar a su señal de control la señal de control basada en RNA, con lo cual se ahorrará tiempo y dinero en desarrollar un controlador más inmune al ruido.

[3] El controlador PID controla la de forma aceptable el tensado del papel sin embargo en un ambiente con mucho ruido no lo logra un funcionamiento óptimo, esto es debido a que el control PID es diseñado generalmente para obtener una mejor respuesta a la referencia y no a las perturbaciones.

[4] El controlador PID-RNA presenta un mejor desempeño, especialmente cuando se presenta ruido en el sistema, esto se debe a que el controlador neuronal esta diseñado y entrenado especialmente para reducir el efecto de dicho ruido.

[5] La red neuronal artificial es entrenada más rápidamente cuando se utilizan mas neuronas de entrada, sin embargo el comportamiento final ya no mejora notablemente a partir de la utilización de siete neuronas, es decir al utilizar ocho o nueve neuronas el comportamiento del controlador se mantiene similar que al utilizar siete neuronas.

[6] Finalmente se concluye que esta es una tecnología prometedora para aplicaciones control de tensado, debido a que es intuitiva, no requiere modelos matemáticos explícitos de la planta a controlar.

6.2. Recomendaciones

[1] Es posible mejorar el control PID, proporcionando un sistema de auto-sintonía, para acelerar calibración para el proceso y su puesta en marcha. Con esta mejora adicionalmente el sistema se haría mucho más comercial, ya que en el mercado son muy comunes los controladores PID este tipo de herramienta.

[2] Debido a que el procesador digital de señales fue programado en lenguaje ANSI C y lenguaje ensamblador, tomó un tiempo considerable programarlos y ponerlos a punto, por lo que se recomendaría utilizar para casos en que el tiempo es relevante controladores lógico programables, los cuales son más fáciles de programar sin embargo hay que recordar pero son mas caros que un microcontrolador o procesador digital de señales.

[3] Para la elección de la arquitectura de la red neuronal es recomendable hacer otras pruebas con arquitecturas diferentes, ya que se tiene una gran cantidad de arquitecturas para redes neuronales artificiales, existe siempre la posibilidad de encontrar una con un mejor comportamiento para el caso estudiado, además de que cada vez se crean nuevas arquitecturas.

[4] Se recomienda incluir el programa del entrenamiento de la red neuronal dentro del procesador digital de señales, para tener mayor independencia del controlador, y así no necesitar de un software adicional para su entrenamiento como es el caso del Matlab.

[5] El procesador digital de señales dsPIC30F4011 respondió correctamente a todos los programas implementados, sin embargo se debería reconsiderar su utilización en el caso de que la complejidad de la arquitectura de la red neuronal sea mayor, de ser así se debe cambiar por otro procesador digital.

Anexos

A.1. Código fuente del controlador

```
/* ========================================================================
Proyecto/Archivo              Multitask6/main.c
Procesador:                   30F4011
Frecuencia:                   XT w/PLL*8; 80MHz; 20MIPS (4*16=64Mhz)
Descripcion:                  Controladores PID y PID-RNA
Programacion:                 F Vasquez Ch
======================================================================== */

#include "p30f4011.h"         //30F4011 Definiciones del procesador
#include "dsp.h"
#include "UTIL.C"
#include "DATA.C"
#include "IO.C"               //Inicializaciones input/output
#include "TIMERS.C"           //Temporizadores
#include "AD.C"               //Conversor analogico a digital
#include "RS232.C"            //Manupulación de Puerto UART
#include "PID.C"              //Controlador PID
#include "NET.C"              //Controlador Neuronal
#include "MOTORS.C"           //Controlador de motores PWM y PAP
#include "INTERR.C"           //Interrupciones; Siempre al final!

      _FWDT(WDT_OFF);
      _FOSC(CSW_FSCM_OFF & XT_PLL16);
      _FBORPOR(PBOR_ON & BORV_20 & PWRT_64 & MCLR_EN);
      _FGS(CODE_PROT_OFF);

int main(void)
{ ADPCFG=0xFFFF;              //deshabilitar conversión A/D
  _ADON=0;                    //apagar conversor A/D
  delay20();                  //retardo para voltajes
  initIO();                   //definicion de direccion de puertos
  initad();                   //inicia conversores A/D
  inittimer1();               //inicia timer1
  initRS232();                //inicia UART2
  initTenPID();               //inicia PID 1: Tensado
  initFerPID();               //inicia PID 2: Alineamiento
  initNeuralNetwork();        //inicia la red neuronal
  initMotors();

  while(1)                    //ejecutar siempre
  { if (stmr1.istask0)        //task0; cada 4ms; 250 veces por segundo
    { if(rx.cambio==1)        // ¿ se recibio ordenes de la PC ?
      { rxSCADA();            //Decodificación de datos recibidos
        rx.cambio=0;
      }
      stmr1.istask0=0;
    }
    if (stmr1.istask1)        //task1; cada 8ms; 125 veces por segundo
    { switch (subtask1) {     //subtareas para algoritmos PID-RNA
      case 0: subtask1++;     //actualizar para sub-tarea siguiente
        checkad();            //leer datos del ADC_1
        if (vi1<10)           //promediado de datos del ADC_1
        { suma1=suma1+ad.val; //actualizar valor ADC_1
          vi1=vi1+1;          //siguiente dato
        }
        else
        { val1=(int)(0.1*suma1);          //actualizar valor ADC_1
          vi1=0; suma1=0;                 //re-inicializar promediador
          ref1 = Q15(-0.05);
          tenPID.controlReference = ref1; //establecer set point PID_1
```

```
          tenPID.measuredOutput = val1;   //actualizar valor de tensado
          net.r0=net.r1; net.r1=net.r2;   //establecer entradas RNA:
          net.r2=ref1; net.uff0=net.uff1; //  r(k-1) r(k) r(k+1) u(k-1)
          if (hot1==1)                    //¿activado controlador_1?
          { PID(&tenPID);                 //ejecutar algoritmo PID
            //exeNeuralNetwork();           //ejecutar algoritmo RNA
            //u1_NET=net.uff1;              //señal de control RNA
            u1_PID=tenPID.controlOutput;    //señal de control PID
            u1=u1_PID;
            //u1=u1_PID+u1_NET;             //señal de control final
            PDC3=(int)(0.0304*u1)+999;    //1023/(15*32767/500)
          }
          else PDC3=0;
        }
      break;
      case 1: subtask1++;     //actualizar para sub-tarea siguiente
        checkad();            //leer datos del ADC_2
        if (vi2<10)           //promediado de datos del ADC_2
        { suma2=suma2+ad.val; //actualizar valor ADC_2
          vi2=vi2+1;          //siguiente dato
        }
        else
        { val2=(int)(0.1*suma2);        //actualizar valor ADC_2
          vi2=0; suma2=0;                //re-inicializar promediador
          if (hot2==1)                   //¿activado controlador?
          { if (val2>=100)               //algoritmo control on-off
              hora();                    //movimientos horario
            else anti();                 //movimientos antihorario
            pasotx=paso[step];           //señal de control final
          }
          else LATE=0;
        }
      break;
      default: subtask1=0; break; }
      stmr1.istask1=0;
    }
    if (stmr1.istask2)        //task2; cada 16ms; 62 veces por segundo
    { txSCADA();              //transmitir datos del dsPIC a la PC
      stmr1.istask2=0;
    }
    if (stmr1.istask3)        //task3; cada 32ms; 31 veces por segundo
    { switch (subtask3) {     //verificación de funcionamiento del dsPIC
      case 0: subtask3++; toggle(_LATB8); break;
      case 1: subtask3++; break;
      case 2: subtask3++; break;
      case 3: subtask3=0; break;
      default: subtask3=0; break; }
      stmr1.istask3=0;
    }
    nop();
  }
  return 0;
}

/* ================================================================================
Proyecto/Archivo:                Multitask6/util.c
Descripcion:                     Macros utiles
================================================================================ */

//definiciones generales
#define INTMAX 65535 //límite máximo de unsigned integer
#define CHARMAX 255 //límite máximo de unsigned char
#define nop() {Nop();}
#define nop2() {nop();nop();}
#define nop3() {nop2();nop();}
#define nop4() {nop2();nop2();}
#define nop5() {nop2();nop3();}
#define nop10() {nop5();nop5();}
```

```
#define nop20() {nop10();nop10();}
#define nop40() {nop20();nop20();}
#define toggle(a) {a=a^1;} //intercambiar valor bit

//macros de acceso a bits
#define setbit(bit,a) {a=a|(xglue2(MASK,bit));} //sets bit en a
#define clearbit(bit,a) {a=a&(~(xglue2(MASK,bit)));}
#define isbit(bit,a) ((a&(xglue2(MASK,bit)))==(xglue2(MASK,bit)))

//macros de retardos
unsigned int x;
#define delay10() {for(x=0;x<INTMAX;x++) nop10();}
#define delay20() {delay10(); delay10();}

//macros de acceso a byte
#define TESTLED _LATD3 //TESTLED is on port RD3
#define TESTLED2 _LATD2 //TESTLED2 is on port RD2
#define Lbyteptr(var16) ((unsigned char*)(&var16))
#define Hbyteptr(var16) ((unsigned char*)(&var16)+1)
#define setLbyte(var16,var8) {(unsigned char)(*Lbyteptr(var16)=var8);}
#define setHbyte(var16, var8) {(unsigned char)(*Hbyteptr(var16)=var8);}
#define getLbyte(var16) ((unsigned char)(*Lbyteptr(var16)))
#define getHbyte(var16) ((unsigned char)(*Hbyteptr(var16)))

/* ================================================================================
Proyecto/Archivo:                Multitask6/data.c
Descripcion:                     Variables globales
================================================================================*/

//Definiciones de bits de FLAGS

static int subtask1=0;
static int subtask3=0;
static unsigned char step=0;
static unsigned char paso[5]={0,10,9,5,6};
static unsigned char pasotx;

static int val1=0;
static int vi1=0;
static long int suma1=0;

static int val2=0;
static int vi2=0;
static long int suma2=0;

static int ref1=0;
static int ref2=0;
static int u1_PID=0;
static int u1_NET=0;
static int u1=0;
static int u2=0;
static char hot1=1;
static char hot2=1;

tPID tenPID;
fractional abcCoefficient[3] __attribute__ ((section (".xbss, bss,
xmemory")));
fractional controlHistory[3] __attribute__ ((section (".ybss, bss,
ymemory")));
fractional kCoeffs[] = {0,0,0};

tPID ferPID;
fractional abcCoefficient2[3] __attribute__ ((section (".xbss, bss,
xmemory")));
fractional controlHistory2[3] __attribute__ ((section (".ybss, bss,
ymemory")));
fractional kCoeffs2[] = {0,0,0};
```

```
/* ================================================================================
Proyecto/Archivo:                Multitask6/io.c
Descripcion:                     Manejo de entradas/salidas
================================================================================*/

//variable declaration
struct
{
  unsigned int time;
  unsigned char roll;
  unsigned position :1;

}int0edge;

//TRIS registro que determina la dirección de los puertos: 0:Salida; 1:Entrada
void initIO()                    //configurar direccion de E/S
{
  //-------------------------------------------------------------PORTB I/O setup
  _TRISB0=0;                     //RB0=Salida Latch H:On L:Off=Qo
  _TRISB1=1;                     //RB1=Entrada Analógica 1
  _TRISB2=1;                     //RB2=Entrada Analógica 2
  _TRISB3=1;                     //RB3=Entrada Analógica 3
  _TRISB4=1;                     //RB4=Entrada Analógica 4
  _TRISB5=1;                     //RB5=Entrada Analógica 5
  _TRISB6=0;                     //RB6=Salida LED6
  _TRISB7=0;                     //RB7=Salida LED7
  _TRISB8=0;                     //RB8=Salida LED8
  LATB=0x00;

  //-------------------------------------------------------------PORTC I/O setup
  _TRISC13=0;                     //RC13=Salida LED9
  _TRISC14=0;                     //RC14=Salida LED10
  LATC=0x00;

  //-------------------------------------------------------------PORTD I/O setup
  _TRISD0=1;                     //RD0=Entrada INT1/Botón 2
  _TRISD1=1;                     //RD1=Entrada INT2/Botón 3
  _TRISD2=0;                     //RD2=Salida LED11
  _TRISD3=0;                     //RD3=Salida LED12

  //-------------------------------------------------------------PORTE I/O setup
  _TRISE0=0;                     //RE0=Salida bornera 5-6
  _TRISE1=0;                     //RE1=Salida bornera 5-5
  _TRISE2=0;                     //RE2=Salida bornera 5-4
  _TRISE3=0;                     //RE3=Salida bornera 5-3
  _TRISE4=0;                     //RE4=Salida bornera 5-2
  _TRISE5=0;                     //RE5=Salida bornera 5-1
  _TRISE8=1;                     //RE8=Entrada INT0/Botón 4
  LATE=0x00;

  //-------------------------------------------------------------PORTF I/O setup
  _TRISF0=0;                     //RF0=Salida bornera 4-1
  _TRISF1=0;                     //RF1=Salida bornera 4-2
  _TRISF2=1;                     //RF2=SerialPort1 Entrada/bornera 4-3
  _TRISF3=0;                     //RF3=SerialPort1 Salida/bornera 4-4
  _TRISF4=1;                     //RF4=SerialPort2 Entrada
  _TRISF5=0;                     //RF5=SerialPort2 Salida
  _TRISF6=0;                     //RF5=Salida bornera 4-5
}

//Inicualizar Interrupcion Externa INT0
void initINT0()
{
  int0edge.position=0;           //0 positivo ; 1 negativo
  int0edge.time=0;               //0 positivo ; 1 negativo
  int0edge.roll=0;               //0 positivo ; 1 negativo
  INTCON2=0x0000;                //configura flanco positivo
  IPC0bits.INT0IP=4;             //prioridad 4
```

```
  IFS0bits.INT0IF=0;             //Reset Flag INT0
  IEC0bits.INT0IE=1;             //habilitar INT0
}

//set External Interrupt edge
void setedgeINT0(unsigned char edge)
{
   IFS0bits.INT0IF=0;            //resetear Flag INT0
   IEC0bits.INT0IE=0;            //deshabilitar INT0
   INTCON2=edge;                 //nuevo flanco
   IEC0bits.INT0IE=1;            //habilitar INT0
}

/* ================================================================================
Proyecto/Archivo:                Multitask6/timers.c
Descripción:                     Temporizadores
================================================================================*/

//constant definitions
#define TMR1PERIOD 16000  //16000 ciclos de instrucción para generar interrup.
                          //62.5 ns*16000=1 ms; ajusta este valor para el
                          //periodo de timer1 deseado
//timer1 variables
struct tmrdata                    //multitareas
{
  unsigned mtmcount :2;           //contador principal; valores: 0,1,2,3
  unsigned istask0  :1;           //task 0 flag de habilitación
  unsigned istask1  :1;           //task 1 flag de habilitación
  unsigned istask2  :1;           //task 2 flag de habilitación
  unsigned istask3  :1;           //task 3 flag de habilitación
  unsigned ctask0   :1;           //task 3 flag de habilitación
  unsigned ctask1   :1;           //contador task1; valores 0,1
  unsigned ctask2   :2;           //contador task2; valores 0,1,2,3
  unsigned ctask3   :3;           //contador task3; valores 0,1,2,3,4,5,6,7
} stmr1;

//inicializa timer1
void inittimer1()
{
  stmr1.mtmcount=0;
  stmr1.istask0=0;
  stmr1.istask1=0;
  stmr1.istask2=0;
  stmr1.istask3=0;
  stmr1.ctask1=0;
  stmr1.ctask2=0;
  stmr1.ctask3=0;

  //initialize timer1
  T1CON=0;
  TMR1=0;
  PR1=TMR1PERIOD;                 //cargar registro de periodo de timer1
  IPC0bits.T1IP=5;                //configurar prioridad 5
  IFS0bits.T1IF=0;                //resetear flag de interupción timer1
  IEC0bits.T1IE=1;                //habilitar interrupción de timer1
  T1CONbits.TON=1;                //iniciar timer1
}

//Timer4 ----------------------------------------------------------------------
#define TMR4PERIOD 0xFFFF //65535 ciclos de instrucción
                          //50 ns*65535*256=0.838s; periodo máximo
                          //16 bits, 256 pre-escalador

struct                            //holds data needed to control timer4
{ unsigned char rcount;           //this will hold timer4 rollover counts
}stmr4;                           //timer4 structure

void inittimer4()
```

```
{ stmr4.rcount=0;                //borrar variable de contador
  T4CON=0x0030;                  //pre-escalador 1:256
  TMR4=0;                        //borrar registro de timer4
  PR4=TMR4PERIOD;                //cargar registro de periodo de timer4
  IPC5bits.T4IP=2;               //configurar prioridad 2
  IFS1bits.T4IF=0;               //resetear flag de interrupción timer1
  IEC1bits.T4IE=0;               //no habilitar timer4 interrupción, ahora
  T4CONbits.TON=0;               //no iniciar timer4, ahora
}

//Timer4 control: turn ON/OFF-------------------------------------------------
void turntimer4(unsigned char state)
{
  _T2IF=0;                       //resetear flag de interrupción
  TMR4=0;                        //resetear registro timer4
  IEC1bits.T4IE=state;           //enable/disable interrupciones timer4
  T4CONbits.TON=state;           //stop/start timer4
}

/* ========================================================================
Proyecto/Archivo:                Multitask6/ad.c
Descripción:                     Conversión analógico a digital
======================================================================== */

struct
{ int val;
  unsigned index :1;
} ad;

void initad()                    //configurar canales de conversión A/D
{ ad.val=0;
  ad.index=0;
  ADPCFG=0xFFC1;                 //configurar RB1,2,3,4,5 analógicas
                                 // C1=1100 0001 RB0=pin2 RB1=pin3 RB2=pin4
  _ADON=0;                       //deshabilitar A/D
}

void checkad()                   //Inicio de conversión
{ ADCON1=0x03E0;                 //HIGH:02,LOW:00,SH:03, auto-conversión
  ADCHS=subtask1;                //escoger ADC subtask=canal [1,2,3,4,5]
  ADCSSL=0;                      //saltar escaneo de entrada
  ADCON3=0x0107;                 //tiempo de muestreo=1*TAD; TAD=4*TCY clk
  ADCON1bits.ADON=1;             //habilitar el ADC
  ADCON1bits.SAMP=1;             //comenzar conversión de ADC
  while(!ADCON1bits.DONE);       //esperar conversión 12*4*62.5ns=3us
  ad.val=ADCBUF0;                //leer el dato convertido ADCx=subtask1
}

/* ========================================================================
Proyecto/Archivo:             Multitask6/RS232.c
Descripción:                  Controla el puerto serial RS232
========================================================================*/

struct
{
  unsigned char rx;          //contiene el byte RX
  unsigned char tx;          //contiene el byte TX
  unsigned isrx :1;          //flag indica que hay nuevo valor en RX
} rs232;                     //estructura RS232

struct
{
  unsigned int j;            //subíndice de dato a enviar
  unsigned char val1;        //subindice de dato a enviar
  unsigned char val2;        //subindice de dato a enviar
  unsigned char u1;          //subindice de dato a enviar
  unsigned char u2;          //subindice de dato a enviar
} tx;
```

```
struct
{
  unsigned int dir;            //subindice de dato a recibir
  unsigned char ref1;          //subindice de dato a recibir
  unsigned char ref2;          //subindice de dato a recibir
  unsigned char dato;          //subindice de dato a recibir
  unsigned cambio :1;          //si cambia entrar a decodificar
  unsigned tipo :1;            //flag de tipo de datos
} rx;

void initRS232()                  //inicializa UART2
{ rs232.rx=0;                     //borrar variable RX
  rs232.tx=0;                     //borrar variable TX
  rs232.isrx=0;                   //borrar flag RX
  U2BRG=103;                      //BRG=129 at 9600 Baud;  %err=¿?
  IPC6bits.U2TXIP=3;              //TX priority 3
  IPC6bits.U2RXIP=5;              //RX prioridad: 5
  U2STA=0;                        //borrar registro de estado
  U2MODE=0x8000;                  //modo: UART Enable,8bit,NoParity,StopBit
  U2STAbits.UTXEN=1;              //habilitar TX
  IEC1bits.U2RXIE=1;              //habilitar RX
}

void testTX()                     //probar transmisión
{ U2TXREG=0x41;                   // transmitir: 0x41 caracter "A"
}

void txSCADA()                    //enviar datos
{ switch (tx.j) {                 //subtareas para PID
  case 0: U2TXREG=0x41;           //send 01000001 ASCII(65)="A"=0x41
          tx.j++;                 //siguiente tarea
          break;
  case 1: U2TXREG=0xFA;           //enviar 11111010=ASCII(250)="û"=0xFA
          tx.j++;                 //siguiente tarea
          break;
  case 2: tx.val1=getHbyte(val1); //tomar el byte alto
          U2TXREG=tx.val1;        //enviar 8 bits de ADC 1
          tx.j++;                 //siguiente tarea
          break;
  case 3: tx.val2=getHbyte(val2); //tomar el byte alto
          U2TXREG=tx.val2;        //enviar 8 bits de ADC 2
          tx.j++;                 //siguiente tarea
          break;
  case 4: tx.u1=getHbyte(u1);     //tomar el byte alto
          U2TXREG=tx.u1;          //enviar señal de control 1 high
          tx.j++;                 //siguiente tarea
          break;
  case 5: tx.u2=pasotx;           //tomar el byte bajo
          U2TXREG=tx.u2;          //enviar señal de control 1 low
          tx.j++;                 //siguiente tarea
          break;
  case 6: U2TXREG=0x42;           //send 01000010=ASCII(66)="B"=0x42
          tx.j++;                 //siguiente tarea
          break;
  case 7: U2TXREG=0x0D;           //send 00001101=ASCII(13)="CR"=0x0D
          tx.j++;                 //siguiente tarea
          break;
  case 8: tx.j++;   break;        // retardo
  case 9: tx.j++;   break;        // retardo
  case 10: tx.j++;  break;        // retardo
  case 11: tx.j++;  break;        // retardo
  case 12: tx.j++;  break;        // retardo
  case 13: tx.j++;  break;        // retardo
  case 14: tx.j++;  break;        // retardo
  case 15: tx.j++;  break;        // retardo
  default:tx.j=0;   break;
  }
```

```
}

void rxSCADA()              //decodificador de dato recibido
{ switch (rx.dir) {         //dirección del dato
  case 48:                  //dato de prueba "0"=0x30=48
    toggle(_LATC13);        //test: cambiar de estado LED
    break;
  case 49:                  //"1"=0x31=49
    toggle(_LATC14);        //test: cambiar de estado LED
    ref1=256*(int)(2.56*rx.dato-128); //nuevo Set Point 1
    break;
  case 50:                  //"2"=0x32=50
    toggle(_LATC14);        //test: cambiar de estado LED
    ref2=256*(int)(2.56*rx.dato-128); //nuevo Set Point 2
    break;
  case 51:                  //"3"=0x33=51
    if(rx.dato==97)         // "a"=0x61=97
      hot1=1;              // PID_1: ON
    if(rx.dato==98)         // "b"=0x62=98
      hot1=0;               // PID_1: OFF
    break;
  case 52:                  //"4"=0x34=52
    if(rx.dato==97)         // "a"=0x61=97
      hot2=1;              // PID_2: ON
    if(rx.dato==98)         // "b"=0x62=98
      hot2=0;               // PID_2: OFF
    break;
  default:nop();    break;
  }
}

/* ======================================================================
Proyecto/Archivo:                Multitask6/pid.c
Descripcion:                     Inicializacion de los PID
====================================================================== */

void initTenPID()
{ tenPID.abcCoefficients = &abcCoefficient[0];
  tenPID.controlHistory = &controlHistory[0];
  PIDInit(&tenPID); //Borrar historia del controlador
       kCoeffs[0] = Q15(0.9);
       kCoeffs[1] = Q15(0.04);
       kCoeffs[2] = Q15(0.05);
  //Calcular coeficientes a,b, & c apartir de Kp, Ki & Kd */
  PIDCoeffCalc(&kCoeffs[0], &tenPID);
}

void initFerPID()
{ ferPID.abcCoefficients = &abcCoefficient2[0];
  ferPID.controlHistory = &controlHistory2[0];
  PIDInit(&ferPID); //Borrar historia del controlador
       kCoeffs2[0] = Q15(0.7);
       kCoeffs2[1] = Q15(0.04);
       kCoeffs2[2] = Q15(0.05);
  //Calcular coeficientes a,b, & c apartir de Kp, Ki & Kd */
  PIDCoeffCalc(&kCoeffs[0], &ferPID);
}

/* ======================================================================
Proyecto/Archivo:              Multitask6/net.c
Descripción:                   Controlador de Red Neuronal
====================================================================== */

struct                         //w(fila,col)
{ float w1[7][4];              //pesos de la primera capa
  float in[4];                 //entradas de la RNA
  float b1[7];                 //offset de la primera capa
  float hi[7];                 //resultado capa oculta
```

```
  float w2[7];                  //pesos de la segunda capa
  float b2;                     //offset de la segunda capa
  float out;                    //salida de la segunda capa
  float r0;                     // r(k+l) referencias para
  float r1;                     // r(k)    la red neuronal
  float r2;                     // r(k-l)
  float uff1;                   // u(k)   salida de control
  float uff0;                   // u(k-l)    feedforward
} net;

void initNeuralNetwork()        //definir valores iniciales
{ net.w1[0][0]=0.00046603329040204028; net.w1[0][1]=3.5769628629137056e-005;
  net.w1[0][2]=-0.008798785085077044 3; net.w1[0][3]=-0.01576461566947273;
  net.w1[1][0]=0.022285890756671076; net.w1[1][1]=0.019989722374275716;
  net.w1[1][2]=-0.1055974292819525; net.w1[1][3]=-0.19843191341430735;
  net.w1[2][0]=0.12910755876644145; net.w1[2][1]=0.10728480106272513;
  net.w1[2][2]=-0.30374582847429182; net.w1[2][3]=-0.48568841758851211;
  net.w1[3][0]=-0.16485000832453106; net.w1[3][1]=-0.1472383001770996;
  net.w1[3][2]=0.34327927999510088; net.w1[3][3]=0.61521724730216665;
  net.w1[4][0]=0.033028795196384783; net.w1[4][1]=0.029523667957737539;
  net.w1[4][2]=-0.098199875314555304; net.w1[4][3]=-0.18544747277216186;
  net.w1[5][0]=-0.0055055064374616765; net.w1[5][1]=-0.0041755942305803147;
  net.w1[5][2]=0.078960365806839211; net.w1[5][3]=0.14211767195893715;
  net.w1[6][0]=-0.00072757545292944335; net.w1[6][1]=-0.00062749074255978154;
  net.w1[6][2]=0.0016613948061406797; net.w1[6][3]=0.0041559101111832177;

  net.b1[0]=-2.2739362296672678;  net.b1[1]=1.506425933437771;
  net.b1[2]=-0.43637865299220802;  net.b1[3]=-0.17309020270641337;
  net.b1[4]=0.74957290017723277;  net.b1[5]=1.476550169563863;
  net.b1[6]=-2.2765233302811478;


  net.w2[0]=-0.30985799217144488; net.w2[1]=-0.75778662075170045;
  net.w2[2]=-1.0553784792752672; net.w2[3]=0.84822114857616782;
  net.w2[4]=-0.26709278392276603; net.w2[5]=0.74153895704701578;
  net.w2[6]=0.078195233092717573;

  net.b2=-0.31193076915963075;
}

void exeNeuralNetwork()         //ejecuta algoritmo de control RNA
{ net.in[0]=net.r0;
  net.in[1]=net.r1;
  net.in[2]=net.r2;
  net.in[3]=net.uff0;
  //net.hi=tansig((net.w1*net.in)+net.b1);
  //net.out=(net.w2*net.hi)+net.b2;
  net.uff0=net.out;
}

float tansig(float x)           //funcion sigmoidal: 2/(1+exp(-2*x))-1
{ float t;
  t=2/(1+exp(-2*x))-1;
  return t;
}

/*
float multip(float)             //multiplicacion de matrices
{ int i,j,k;
  int A[n][n],B[n][n],C[n][n];
  for(i=0;i<n;i++)
    for(i=0;i<n;i++)
    {
      C[i][j]=0;
      for(i=0;i<n;i++)
        C[i][j]+=A[i][k]*B[k][j];
    }
}
```

```
*/

/* ================================================================================
Proyecto/Archivo:            Multitask6/motors.c
Descripción:                 Configuiracion del puerto de PWM
================================================================================*/

// FCY = 20 000 000
// Tpwm= TCY*(PTPER+1)/PTMR escalado   PTPER<14:0>
//  PTMR=OSC/4 Tiempo Base                    PTMR <14:0>
//  Modo de Operacion:00:free                 PTMOD<1:0>=PTCON<1:0>
//  PTMR: prescaler:(1,4,16,64):0=min  PTCKS<1:0>=PTCON<3:2>
//        postscaler: (1,...,16):0=min PTOPS<3:0>=PTCON<7:4>
//
// PWMCON1: 0x0FFF: Activar pines PWMH y PWML
// PDC1<15:0>: Ciclo de duracion del PWM 1
//             * PDCx tiene doble resolucion que PTMR
//  PWMPIN: 0:controlado por modulo y 1:controlado por E/S
//  HPOL: polaridad para PWMH 1:positiva 0:negativa
//  LPOL: polaridad para PWML 1:positiva 0:negativa
/* pag.352 deseamos una frecuencia de PWM de 20 KHz
                FCY                                      20,000'000
      PTPER = -------------------- - 1 = ------------ - 1 = 999
         (Fpwm*PTMR escalado)         20,000*1

  Fpwm=16,000'000/(999+1)=16,000=16 KHz
*/

void initMotors()
{ PWMCON1=0x0444;     // 0000 0100 0100 0100
  PDC1 = 0;           // PDC de 0 a 999*2
  PDC2 = 0;
  PDC3 = 0;
  PTPER = 999;
  PTCONbits.PTEN = 1; // Habilitar PWM
}

void hora()             // mueve motor en sentido horario
{ LATE=paso[step];
  if(step<4)
    step=step+1;
  else
    step=1;
}

void anti()             // mueve motor en sentido antihorario
{ LATE=paso[step];
  if(step>1)
    step=step-1;
  else
    step=4;
}

/* ================================================================================
Project/File:                   Multitask6/interr.c
Description:                    Rutinas para interrupciones
================================================================================*/

//ISR timer1 ------------------------------------------------------------------
void _ISR _T1Interrupt()          //Interrupciones de timer1
{ if(stmr1.mtmcount==0)           //-------------------------------------task0
  { stmr1.istask0=1;              //set task0 flag
  }
  else if(stmr1.mtmcount==1)      //-------------------------------------task1
  { if(stmr1.ctask1==0)           //verificar tiempo para task1
    { stmr1.istask1=1;            //set task1 flag
    }
    stmr1.ctask1++;               //incrementar contador task1
```

```
  }
  else if(stmr1.mtmcount==2)      //-------------------------------------task2
  { if(stmr1.ctask2==0)           //verificar tiempo para task2
    { stmr1.istask2=1;            //set task2 flag
    }
    stmr1.ctask2++;               //incrementar contador task2
  }
  else if(stmr1.mtmcount==3)      //-------------------------------------task3
  { if(stmr1.ctask3==0)           //verificar tiempo para task3
    { stmr1.istask3=1;            //set task3 flag
    }
    stmr1.ctask3++;               //incrementar contador task3
  }
  else                            //-------------------------------errors trap
    ;
  stmr1.mtmcount++;               //incrementar contador maestro
  IFS0bits.T1IF=0;                //resetear flag de interrupción de timer1
}

//ISR RX UART2 ----------------------------------------------------------------
void _ISR _U2RXInterrupt()        //interrupción RS-232 RX
{ rs232.rx=U2RXREG;               //leer buffer RX
  rs232.isrx=1;                   //setear flag RX
  if(rx.tipo==0)
  { rx.dir=rs232.rx;              //copiar RX a dirección
    rx.tipo=1;                    //indica el siguiente es dato
  }
  else
  { rx.dato=rs232.rx;             //copiar RX a dato
    rx.tipo=0;                    //indica el sgte. es dir
    rx.cambio=1;                  //flag de entrada de datos
  }
  IFS1bits.U2RXIF=0;              //resetear flag de interrupción de RX
}


/* ========================================================================
Project/File:                     Multitask6/pid.s
Description:                      Algoritmo PID
======================================================================== */

        ; Local inclusions.
        .nolist
        .include        "dspcommon.inc"         ; fractsetup
        .list

        .equ    offsetabcCoefficients, 0
        .equ    offsetcontrolHistory, 2
        .equ    offsetcontrolOutput, 4
        .equ    offsetmeasuredOutput, 6
        .equ    offsetcontrolReference, 8

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        .section .libdsp, code
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; _PID:
; Prototipo:
;              tPID PID ( tPID *fooPIDStruct )
;
;   controlOutput[n] = controlOutput[n-1]
;                    + controlHistory[n] * abcCoefficients[0]
;                    + controlHistory[n-1] * abcCoefficients[1]
;                    + controlHistory[n-2] * abcCoefficients[2]
;
;   abcCoefficients[0] = Kp + Ki + Kd
;   abcCoefficients[1] = -(Kp + 2*Kd)
;   abcCoefficients[2] = Kd
```

```
;   controlHistory[n] = measuredOutput[n] - referenceInput[n]
;
; Entrada:
;       w0 = Dirección de la estructura de datos tPID
; Return:
;       w0 = Dirección de la estructura de datos tPID
;.....................................................................

        .global _PID
_PID:

        push    w8
        push    w10
        push    CORCON

        fractsetup      w8

        mov [w0 + #offsetabcCoefficients], w8
        mov [w0 + #offsetcontrolHistory], w10

        mov [w0 + #offsetcontrolOutput], w1
        mov [w0 + #offsetmeasuredOutput], w2
        mov [w0 + #offsetcontrolReference], w3

        lac     w3, a
        lac     w2, b
        sub     a
        sac.r   a, [w10]

        ; Calcular la salida de control PID
        clr     a, [w8]+=2, w4, [w10]+=2, w5
        lac     w1, a
        mac     w4*w5, a, [w8]+=2, w4, [w10]+=2, w5
        mac     w4*w5, a, [w8], w4, [w10]-=2, w5
        mac     w4*w5, a, [w10]+=2, w5
        sac.r   a, w1
        mov     w1, [w0 + #offsetcontrolOutput]

        ;Actualizar histórico de errores en la línea de retardo
        mov     w5, [w10]
        mov     [w10 + #-4], w5
        mov     w5, [--w10]

        pop     CORCON                  ; restaurar CORCON.
        pop     w10                     ; restaurar registros de trabajo.
        pop     w8
        return

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; _PIDInit:
;
; Prototipo:
; void PIDInit ( tPID *fooPIDStruct )
;.....................................................................

        .global _PIDInit
_PIDInit:
        push    w0
        mov     [w0 + #offsetcontrolOutput], w0
        clr     [w0]
             pop    w0
             push w0

        mov     [w0 + #offsetcontrolHistory], w0
        clr     [w0++]                  ; ControlHistory[n]=0
        clr     [w0++]                  ; ControlHistory[n-1] = 0
        clr     [w0]                    ;ControlHistory[n-2] = 0
        pop     w0                      ;Restaurar puntero a la base tPID
```

```
        return

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; _PIDCoeffCalc:
;
; Prototipo:
; void PIDCoeffCalc ( tPIDGainCoeff *fooPIDGainCoeff, tPID *fooPIDStruct )
;
; Operación: Esta rutina calcula los coeficientes del PID por medio de
;            Kp, Ki y Kd provistos. Los coeficientes calculados son:
;             A = Kp + Ki + Kd
;             B = -(Kp + 2*Kd)
;             C = Kd
; Entradas:
;       w0 = Dirección de los arrays de los coeficientes Kp, Ki and Kd
;       w1 = Dirección de la estructura tPID definido en dsp.h
;
;.....................................................................

        .global _PIDCoeffCalc
_PIDCoeffCalc:
        push    CORCON
        fractsetup      w2

        mov     [w1], w1
        lac     [w0++], a               ; ACCA = Kp
        lac     [w0++], b               ; ACCB = Ki
        add     a                       ; ACCA = Kp + Ki
        lac     [w0--], b               ; ACCB = Kd
        add     a                       ; ACCA = Kp + Ki + Kd
        sac.r   a, [w1++]
        lac     [--w0], a               ; ACCA = Kp
        add     a                       ; ACCA = Kp + Kd
        add     a                       ; ACCA = Kp + 2Kd
        neg     a                       ; ACCA = -(Kp + 2Kd)
        sac.r   a, [w1++]
        sac     b, [w1]                 ; _abcCoefficients[2] =  Kd

        pop     CORCON
        return

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        .end
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; OEF
```

A.2. Código fuente para el entrenamiento

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%          Programa de entrenamiento: fernet4.m           %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all;
clc;
sim('trainsig2');
T=tout; in=trainsig;
n=length(T);

[T,X,Y]= sim('spm2',T,simset(),[T in]);   % y(t-1)
Y=Y';
for i=1:n-2,
    y0(i)=Y(i);      % y(t-1) truncada
end
for i=1:n-2,
    y1(i)=Y(i+1);   % y(t)
end
for i=1:n-2,
    y2(i)=Y(i+2);   % y(t+1), r(t+1)
end
for i=1:n-2,
    u0(i)=in(i);     % u(t-1) truncado
end
for i=1:n-2,
    u1(i)=in(i+1);   % û(t) salida deseada de RNA
end
for i=1:n-2,
    t(i)=T(i);       % tiempo truncado
end

%% CORTAR TAMAÑO DE y1 y2 y3
figure(1); plot(t,u1,'-',t,y1,'x');

p = [y1;y0;y2;u0]; % Entradas de la RNA
%PR = [-1e3 1e3;-1e3 1e3;-1e3 1e3;-1e3 1e3]; % debe ser de R x 2
PR = [-10000 10000;-10000 10000;-10000 10000;-10000 10000]; % debe ser de R x
2

net = newff(PR,[7 1],{'tansig' 'purelin'},'traingdx'); %traingdx
out1 = sim(net,p); % Resultado de la Red Neuronal SIN entrenar

%%%%%%%%%%%% Entrenamiento de la Red Neuronal %%%%%%%%%%%%%%%%

net.trainParam.epochs = 2000; % Paramatros de la RNA
net.trainParam.goal = 1e-8;
net = train(net,p,u1);  % Entrenando para señal de control
out2 = sim(net,p); % Resultado de la RNA ENTRENADA
figure(2); plot(t,u1,'-',t,out1,'.',t,out2,'x');

%net.IW{1,1} % containing 1 input weight matrix
%net.LW{2,1} % containing 1 layer weight matrix
%net.b

gensim(net,0.2) % genera modelo de RNA en simulink
```

A.3. Programa en Java para la adquisición de datos

```
/* ========================================================================
Project/File:                      logger/Logger0.java
Description:                       Adquisicion y almacenamiento de datos
======================================================================== */

package logger;

import java.io.*;
import java.util.*;
import javax.comm.*;

public class Logger0 implements Runnable,SerialPortEventListener {
  static CommPortIdentifier idPuerto;
  static Enumeration listaPuertos;
  InputStream entrada;
  OutputStream salida;
  SerialPort puertoSerie;
  Thread tLectura;
  FileWriter fs = null;  // crea el flujo de salida: fs
  int fin=0;
  int inicio=0;
  int index=0;
  int k=0;
  int valorantes,norepeat=0,valorStore;
  Object value1,value0;
  public int ten,fer,u1,u2;

  public Logger0() { // Thread que esta en espera de datos
    listaPuertos = CommPortIdentifier.getPortIdentifiers();

    while( listaPuertos.hasMoreElements() ) {
      idPuerto = (CommPortIdentifier)listaPuertos.nextElement();
      if( idPuerto.getPortType() == CommPortIdentifier.PORT_SERIAL ) {
        if( idPuerto.getName().equals("COM1") ) {
         try {    // Si el puerto no está en uso, se intenta abrir
           puertoSerie = (SerialPort)idPuerto.open( "AplLectura",2000 );
         } catch( PortInUseException e ) {}
          try {    // Se obtiene un canal de entrada
            entrada = puertoSerie.getInputStream();
          } catch( IOException e ) {}
          try {     // Se obtiene un canal de salida
            salida = puertoSerie.getOutputStream();
          } catch( IOException e ) {}
          try {    // Añadimos un oyente de eventos al puerto serial
            puertoSerie.addEventListener( this );
          } catch( TooManyListenersException e ) {}
          puertoSerie.notifyOnDataAvailable( true ); // Notifica si hay datos
          try {    // Se fijan los parámetros de comunicación del puerto
            puertoSerie.setSerialPortParams( 9600,
                                             SerialPort.DATABITS_8,
                                             SerialPort.STOPBITS_1,
                                             SerialPort.PARITY_NONE );
          } catch( UnsupportedCommOperationException e ) {}

          tLectura = new Thread( this ); // Crea y lanza el thread
          tLectura.start();
        }
      }
    }
  }

  public void run() {
    try {
```

```
      Thread.sleep( 20000 );
    } catch( InterruptedException e ) {}
  }

  public void serialEvent( SerialPortEvent _ev ) {
    switch( _ev.getEventType() ) {
      case SerialPortEvent.BI:
      case SerialPortEvent.OE:
      case SerialPortEvent.FE:
      case SerialPortEvent.PE:
      case SerialPortEvent.CD:
      case SerialPortEvent.CTS:
      case SerialPortEvent.DSR:
      case SerialPortEvent.RI:
      case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
        break;
      case SerialPortEvent.DATA_AVAILABLE: // datos disponibles
        byte[] bufferLectura = new byte[1];
        try {
          while( entrada.available() > 0 ) {
            int nBytes = entrada.read( bufferLectura );
          }
/*************************************************************************/
          try {   // convertir y guargar en archivo
            int valor=(int)bufferLectura[0];  // convertir a int
            valorStore=(valor+128)*100/256;
            fs=new FileWriter("data.csv",true);
            if(index>=1) {
              fs.write(valorStore+";");
              System.out.print(valor+";");
            }
            else {                          // DETECCION DE INICIO
              if (valor==65) { inicio=1; } // puede ser el inicio
              else {
                if(valor == -6 & inicio == 1) { // si es el inicio
                  index=1;
                }
                else { index=0; inicio=0;} // no es el inicio
              }
            }
/*************************************************************************/
          }
          catch (IOException e) {
            System.out.println("Error: "+e.toString() );
          }
          try {     //cerrar archivo
            fs.close();
          }
          catch (IOException e) {
            System.out.println("Error: "+e.toString() );
          }
        } catch( IOException e ) {}
        break;
      }
    }

  void envia(String mensaje){  // metodo que envia mensaje
    try {
      salida.write( mensaje.getBytes() );
    } catch( IOException e ) {}
  }

  public static void main( String[] args ) {
    Logger0 lector = new Logger0();
  }
}
```

A.4. Sistema implementado.

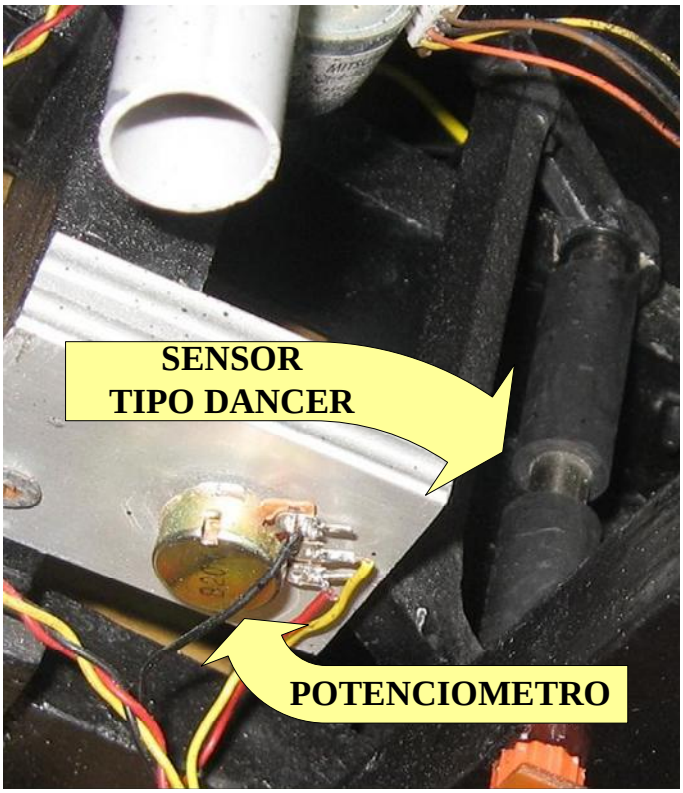


Figura A.1: Sensor de tensado.

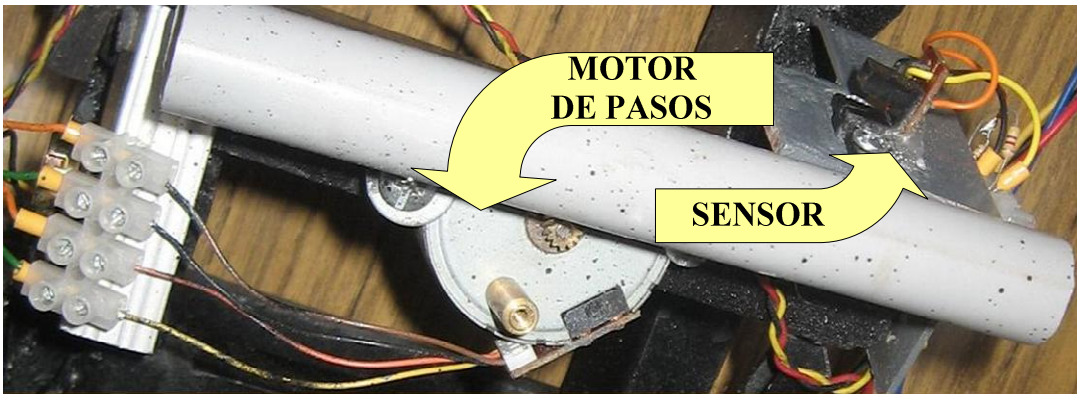


Figura A.2: Sensor y actuador de alineamiento.

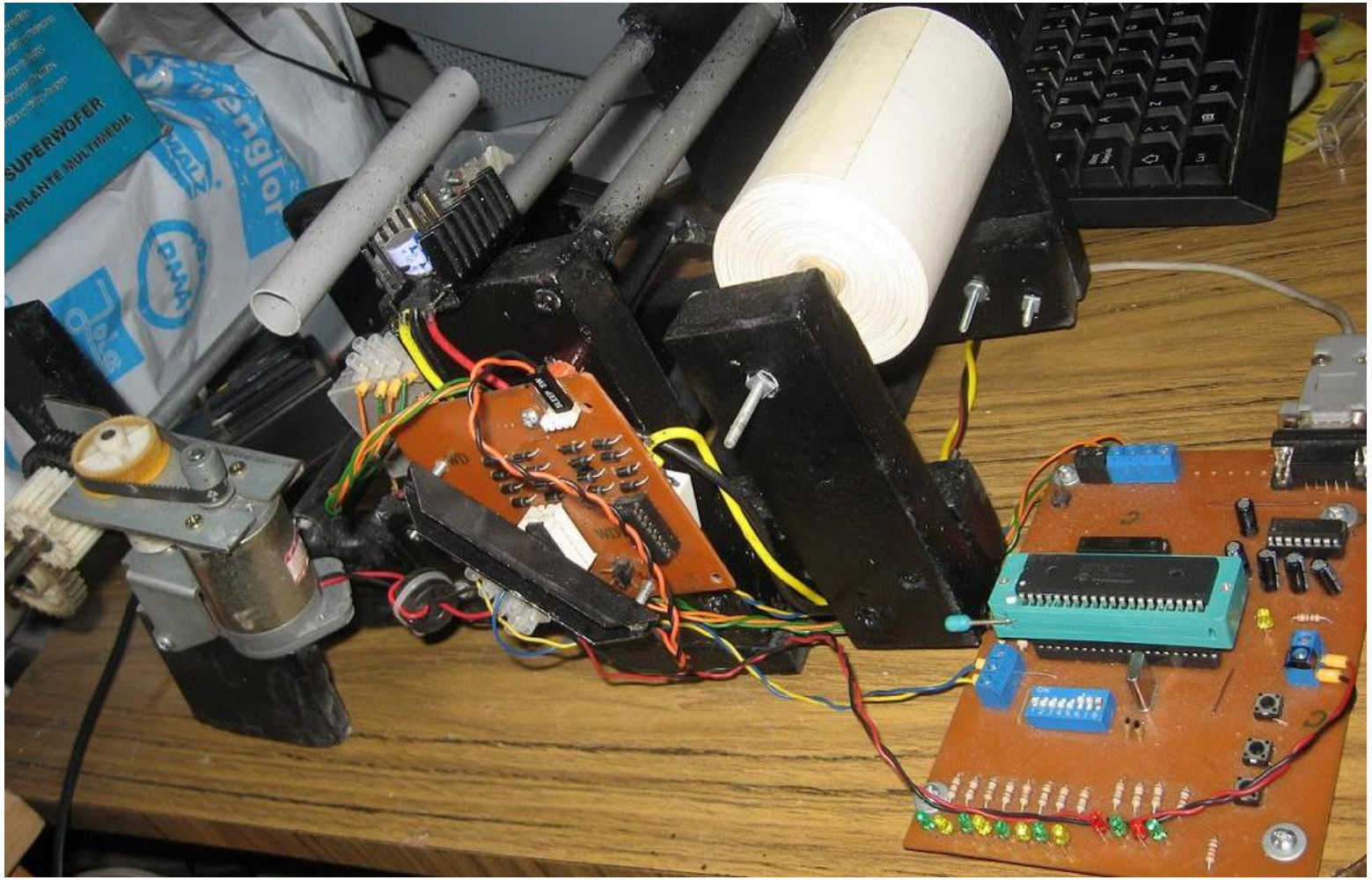


Figura A.3: Sistema total implementado.

A.5. Valores de tensión típicos para papel.

Para bobinado.

Producto		Tensión	
Papel oscuro	33gsm	0.75	lb.pli
	39gsm	0.875	lb.pli
	49gsm	1.0	lb.pli
	57gsm	1.25	lb.pli
Papel Blanco	60gsm	1.4	lb.pli
	70gsm	1.75	lb.pli
	80gsm	2.0	lb.pli
	90gsm	2.25	lb.pli
	95gsm	2.4	lb.pli
	100gsm	2.5	lb.pli

Para des-bobinado

Producto		Tensión	
Papel oscuro	33gsm	0.5	lb.pli
	39gsm	0.6	lb.pli
	49gsm	0.75	lb.pli
	57gsm	0.9	lb.pli
Papel Blanco	60gsm	0.9	lb.pli
	70gsm	1.0	lb.pli
	80gsm	1.3	lb.pli
	90gsm	1.4	lb.pli

95gsm	1.5	lb.pli
100gsm	1.75	lb.pli

gsm: gramos por metro cuadrado
pli: libra por pulgada lineal

Bibliografía

[1] Angulo Usategui José María, García Zapirain Begoña, Angulo Martínez Ignacio, Vicente Sáez Javier. Microcontroladores Avanzados dsPIC. Paraninfo, 2006.

[2] Aström K.J., T. Hägglund. Automatic Tuning of PID Controllers, Research Triangle Park, NC. EUA, Instrument Society of America, 1988.

[3] Boulter Brian Thomas, Y. Hou, Z. Gao, F. Jiang. Active Disturbance Rejection Control for Web Tension Regulation and Control. IEEE Conference on Decision and Control, 2001.

[4] Boulter Brian T. and Gao Zhiqiang. A Novel Approach for On-Line Self-Tuning Web Tension Regulation. IEEE Conference on Control Applications – Proceedings, 1995.

[5] Freeman A. James, Skapura M. David. Redes Neuronales. Addison-Wesley Iberoamericana, 1993.

[6] Hou Yi. Novel Control Approaches for Web Tension Regulation. Doctoral Dissertation, Dept. of Electrical and Computer Engineering, Cleveland State University, 2001.

[7] Issermann Rolf. Digital Control Systems Volume 1: Fundamentals, Deterministic Control. Springer-Verlag Berlin Heidelberg, 1989.

[8] Issermann Rolf. Digital Control Systems Volume 2: Sthocastic Control, Multivariable Control, Adaptive Control, Applications. Springer-Verlag Berlin Heidelberg, 1989.

[9] Kernighan Brian W., Ritchie Dennis M. The ANSI C Programming Language. Second Edition, 1983.

[10] King, Robert E. Computational Intelligence in Control Engineering. Control Engineering Series, 2004.

[11] Konar Amit. Artificial Intelligence and Soft Computing Behavioral and Cognitive Modelling of the Human Brain. CRC Press LLC, 2000.

[12] Kuo Benjamín. Sistemas de Control Automático. Prentice-Hall, 1996.

[13] Krebs S. Multiple tension control using a new approach in signal processing. Fourth International Conference on Web Handling Oklahoma State University, 2004.

[14] Nørgaard Magnus. Neural Network Based System Identification TOOLBOX. Technical University of Denmark, 1995.

[15] Nørgaard M., Ravn O., Poulsen N. K. and Hanken L.K. Neural Network for Modelling and Control of Dynamic Systems: A Practitioner's Handbook. Springer-Verlag London Limited 2000.

[16] Microchip PIC16F87X Data Sheet. Microchip Technology Inc., 1999.

[17] Microchip 16-Bit Language Tools Libraries High-Performance Digital Signal Controllers. Microchip Technology Inc., 2006.

[18] Microchip 16-bit Microcontrollers and Digital Signal Controllers. Microchip Technology Inc., 2006.

[19] Microchip dsPIC30F Family Reference Manual High-Performance Digital Signal Controllers. Microchip Technology Inc., 2006.

[20] Microchip dsPIC30F4011/4012 Data Sheet. Microchip Technology Inc., 2006.

[21] Mohamed Azlan Hussaina, Paisan Kittisupakornb and Wachira Daosud. Implementation of Neural-Network-Based Inverse-Model Control Strategies on an Exothermic Reactor. Science Asia 27, 2001.

[22] Mplab® C30 C Compiler User's Guide. Microchip Technology Inc., 2006.

[23] Ogata K. Discrete-time Control Systems, Prentice - Hall D.L., 1995.

[24] Ogata Katsuhiko. Ingeniería de Control Moderna, Editorial Pearson, 2004.

[25] Pope O. G. Learn Hardware Firmware and Software Design. Corollary Theorems, 2006.

[26] Paanasalo Jari. Modelling and Control of Printing Paper Surface Winding, Helsinki University of Technology Control Engineering Laboratory, 2005.

[27] Prabhakar R. Pagilla, Inderpal Singh, Ramamurthy V. Dwivedula. A Study on Control of Accumulators in Web Processing Lines. Journal of Dynamic Systems, Measurement and Control, Vol. 146, 2004.

[28] Rojas Moreno, Arturo. Control Avanzado Diseño y Aplicaciones en Tiempo Real. Publicación independiente, 2001.

[29] Smith Carlos A., Corripio Armando B. Control Automático de Procesos Teoría y Práctica. Limusa, 1991.

[30] Spooner Jeffrey T., Maggiore Manfredi, Ordoñez Raúl, Passino Kevin M. Stable Adaptive Control and Estimation for Nonlinear Systems: Neural and Fuzzy Approximator Techniques. 2002 John Wiley & Sons, 2002.

[31] Timothy J. Maloney. Electrónica Industrial Moderna, 3ª Ed. Prentice-Hall Hispanoamérica, 1997.

[32] What Is PID—Tutorial Overview. http://www.expertune.com/tutor.html, Expertune, Inc.

[33] Yu Hen Hu, Jenq-Neng Hwang. Handbook of Neural Network Signal Processing. CRC Press LLC, 2002.

[34] Zilouchian Ali, Jamshidi Mo. Intelligent Control Systems Using Soft Computing Methodologies. CRC Press LLC, 2001.