UNIVERSIDAD INDUSTRIAL DE SANTANDER
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA
Y DE TELECOMUNICACIONES
Perfecta Combinación entre Energía e Intelecto

DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PID CON CAPACIDAD PARA COMUNICARSE EN LÍNEA MEDIANTE TCP/IP

IRINA ALEXANDRA AMAYA CÁCERES
DIANA CAROLINA AVENDAÑO VILLAMIZAR

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
Bucaramanga
2008

DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PID CON CAPACIDAD PARA COMUNICARSE EN LÍNEA MEDIANTE TCP/IP

IRINA ALEXANDRA AMAYA CÁCERES
DIANA CAROLINA AVENDAÑO VILLAMIZAR

Trabajo de Grado para optar al título de Ingeniera Electrónica

Director:
Ing. PhD. Rodolfo Villamizar Mejía,

Codirector:
Ing. MsC. Jorge Hernando Ramón Suárez

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
Bucaramanga
2008

AGRADECIMIENTOS

A los profesores Rodolfo Villamizar Mejía y Jorge Hernando Ramón, por su apoyo y respaldo durante la realización de este proyecto, al ingeniero William Salamanca por su colaboración incondicional en los momentos más difíciles, a todos los ingenieros integrantes del grupo SITELRED por soportar nuestras constantes visitas.

De manera muy especial a nuestro padres y hermanos, quienes con su constante acompañamiento, vivieron junto a nosotras el desarrollo de este proyecto y su culminación.

A nuestros amigos de la Universidad por el tiempo compartido y los buenos recuerdos que este dejó, a nuestras amigas de toda una vida Erika y Ana María.

Dedicado a Dios
A mis padres
A mi hermana
A mi amor ,Mario,
Sin cuya asistencia
No lo hubiera logrado

Irina A.

“*Si algún momento se pudiera llamar felicidad, sería este***”**

A Dios y a esos cinco Ángeles que me regaló en la tierra,
mis padres, Hernán y Eddy,
mis hermanos, Luis y Luz Dary
y a mi pequeño, TONY…
Diana C.

CONTENIDO

pág.

LISTA DE FIGURAS

pag.

LISTA DE ANEXOS

pág.

RESUMEN

TÍTULO:
DISEÑO E IMPLEMENTACIÓN DE UN CONTROLADOR PID CON CAPACIDAD PARA COMUNICARSE EN LÍNEA MEDIANTE TCP/IP [1]

AUTORAS:
IRINA ALEXANDRA AMAYA CÁCERES [2]
DIANA CAROLINA AVENDAÑO VILLAMIZAR [2]

PALABRAS CLAVES:
CONTROL, SOCKET, TCP/IP,

DESCRIPCIÓN:

El desarrollo de este proyecto se enfoca en la implementación de un modulo de comunicación, que permite variar en línea los parámetros de un controlador PID, que hace parte de un esquema de control adaptativo. Basado en el objetivo general, el esquema de comunicación está soportado en la pila de protocolos TCP/IP y haciendo uso del API de sockets disponible para el sistema operativo Linux, presente en la tarjeta de desarrollo utilizada para la implementación del controlador.

El módulo de comunicación, utiliza el esquema cliente-servidor, lo cual permite establecer una estación de trabajo de alto rendimiento, donde está implementado el mecanismo de adaptación, como servidor y como cliente la tarjeta de desarrollo ECB_AT91 en cuyo procesador se encuentra el algoritmo de control, de esta forma es posible desacoplar la adaptación de parámetros del dispositivo de control, por lo cual la adición de un canal de comunicación sobre TCP/IP a los sistemas de control clásico, logra integrar la fiabilidad de los esquemas básicos, con la flexibilidad y robustez de los algoritmos de control avanzado.

La verificación de los objetivos planteados se basa en la implementación de un prototipo de prueba, que consta tres estaciones de trabajo, el PC1 donde está programado un mecanismo de adaptación básico y el programa servidor, la tarjeta de desarrollo ECB_AT91 programación del controlador y el cliente y un PC2 para emular el proceso que será controlado. Integrando un sistema de control realimentado, basado en control PID con ajuste de parámetros.

[1] Trabajo de Grado
[2] Facultad de Ingenierías Físico-Mecánicas, Escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones, Director: PhD. Rodolfo Villamizar Mejía

SUMMARY

TITLE:
DESIGN AND IMPLEMENTATION OF A PID CONTROLLER WITH CAPABILITY TO COMMUNICATE ON LINE BY TCP / IP [3]

AUTHORS:
IRINA ALEXANDRA AMAYA CÁCERES [4]
DIANA CAROLINA AVENDAÑO VILLAMIZAR [**]

KEY WORDS:
CONTROL, SOCKET, TCP/IP,

DESCRIPTION:

The development of this project focuses on the implementation of a communication module, which allows online vary the parameters of a PID controller, which is part of an adaptive control scheme. Based on the overall objective, the scheme of communication is supported in the protocol stack TCP/IP and using the sockets API available for the Linux operating system, present in the development of card used for the implementation of the controller.

The communication module uses the client-server scheme, which allows a workstation for high performance, which is implementing the mechanism for adaptation, such as server and client development ECB_AT91 card processor in which we find the algorithm control, so that we can decouple the adjustment of parameters of the control, so the addition of a channel of communication over TCP / IP control systems classic succeeds integrate the reliability of the basic, with the flexibility and robustness of the advanced control algorithms.

The verification of the objectives is based on the implementation of a prototype test, which consists of three stations work, which is scheduled PC1 a mechanism for adaptation and the core server program, the card's driver development ECB_AT91 programming and the customer and a PC2 to emulate the process that will be controlled. Integrating a feedback control system based on PID control with setting parameters.

[3] Degree work.
[4] Faculty of Physics-Mechanical Engineering. Electrical and Electronic Engineering and of Telecommunication School. Director: PhD. Rodolfo Villamizar Mejía.

INTRODUCCIÓN

El uso generalizado de los controladores de la familia PID en procesos industriales, se debe en parte a la sencillez y fiabilidad de su algoritmo, que permite ser aplicado en el control de variables de distinta naturaleza (temperatura, posición, velocidad, etc.). Con ellos se logran márgenes de operación aceptables cuando el sistema opera en condiciones normales. Sin embargo su uso se ve limitado cuando las ganancias de sus acciones de control (proporcional, integral y/o derivativa), son insuficientes para garantizar una operación deseada del proceso. Dicha limitación se debe principalmente a variaciones en las condiciones dinámicas del proceso, perturbaciones, entre otras.

A pesar de las limitaciones mencionadas, el uso los controladores de la familia PID no se excluye totalmente del control de plantas y procesos cuya dinámica, no hace apropiada su aplicación en un esquema de control clásico. Cuando el sistema requiere una mayor exigencia en la etapa de control, el controlador PID se puede integrar a un esquema de control avanzado tal como el adaptativo. En tal esquema se toma el controlador clásico como el controlador principal y este interactúa con los bloques adicionales del esquema. Específicamente, el esquema de control adaptativo varía en línea las ganancias del controlador, usando medidas en línea de las variables controladas del proceso y teniendo en cuenta la condición de operación deseada del mismo. Tal esquema acerca más al proceso a operar a las condiciones deseadas y le da una mayor robustez ante variaciones y/o perturbaciones presentes.

Sin embargo, como resultado de la investigación y observación de diversos esquemas de control se deduce que la principal restricción para implementar un controlador adaptativo que contenga tanto la ley de control, como los adaptadores, es la capacidad de cómputo requerido, velocidad de procesamiento y el consumo de memoria[5,6]. Un esquema de implementación alternativo que puede resolver esta limitación es la de tener un dispositivo dedicado exclusivamente a calcular en línea la ley de control (controlador clásico), además de ser capaz de recibir en línea instrucciones para cambiar las ganancias de dicha ley de control (algoritmo de control).

[5] BARAJAS, Leandro. BARÓN, María del Pilar. Sistemas de control Integrado Neurofuzzy. 1998.
[6] GARRIDO, Santiago. Identificación, Estimación y control de sistemas no lineales mediante RGO. 1999.

Bajo la motivación de proponer dicho tipo de soluciones, el presente proyecto de grado se enmarca dentro de un planteamiento realizado al interior del grupo CEMOS de buscar la solución teórica e implementación de esquemas de control avanzado que puedan ser aplicados a nivel industrial. El proyecto consiste en implementar la comunicación entre un controlador PID clásico, cuyas ganancias puedan ser ajustadas en línea desde una estación donde se encuentra el mecanismo de adaptación. Para validar experimentalmente el esquema, se utilizaron dos equipos de cómputo y una tarjeta de desarrollo (ECB_AT91), en donde se implementaron respectivamente el mecanismo de adaptación, la planta simulada y el controlador.

El equipo donde se implementó el mecanismo de adaptación (PC1) se conecta con el controlador a través un canal implementado sobre la pila de protocolos TCP/IP y haciendo uso de la interfaz de *sockets*. Por otra parte, la señal de control es enviada al proceso, el cual es emulado computacionalmente mediante el segundo equipo (PC2) usando la herramienta Simulink de MATLAB.

El desarrollo de este tipo de esquemas de control permiten producir mejoras sobre controladores industriales existentes, en los que solo debe adicionarse un canal de comunicación que acceda al algoritmo de control implementado y en el que se permitan cambiar en línea sus parámetros, sin interrumpir en ningún momento las acciones de control. Por otra parte, con este tipo de esquemas es posible evitar altos costos computacionales en los controladores de campo, dado que los algoritmos del adaptador de parámetros se pueden implementar en centrales de cómputo ubicadas por ejemplo en los centros de control.

El presente informe de proyecto se divide en 5 capítulos, de la siguiente manera: El capítulo 1 presenta los Conceptos Básicos de las principales temáticas abordadas para el desarrollo de este proyecto, tales como Control PID, Control Adaptativo y comunicación sobre TCP/IP. El Capítulo 2 presenta el protocolo de comunicación implementado para la conexión mediante la interfaz de *sockets*, así como los pasos a seguir en cada punto de enlace (servidor, cliente), para iniciar, llevar a cabo y finalizar la comunicación. Mientras tanto, el Capítulo 3 presenta el diseño e Implementación del controlador, en donde se indican las consideraciones matemáticas y planteamientos de programación utilizados, así como la adición de mejoras en la implementación del mismo. En el capítulo 4 se describe el prototipo de prueba, en donde se describen los módulos implementados y la programación adicional utilizada para su desarrollo. Finalmente, el capítulo 5 especifica las pruebas realizadas y los resultados obtenidos de ellas, además de su respectivo análisis.

1. CONCEPTOS BÁSICOS

El desarrollo de este proyecto tiene su base conceptual en sistemas de control clásico y avanzado, sin embargo requiere del conocimiento de programación de unidades de procesamiento, tal que se puedan aplicar reglas, esquemas y algoritmos para realizar diseños e implementaciones de los diferentes módulos que hacen parte del prototipo de prueba final. El presente capítulo tiene como fin recordar los conceptos básicos del controlador clásico PID (Sección 1.1) y del controlador adaptativo basado en modelo de referencia (Sección 1.2), así como la comunicación sobre TCP/IP mediante *sockets* (Sección 1.3). Sin embargo si se requiere mayor profundidad en las temáticas, se recomienda revisar la bibliografía relacionada en el presente informe.

1.1 CONTROLADOR PID

El controlador PID es uno de los más utilizados a nivel industrial, su implementación en cerca del 95% de los lazos de control[7], muestra la preferencia del uso de leyes de control simple en gran variedad de plantas y procesos, especialmente cuando su dinámica es la apropiada y los requerimientos de las variables a controlar pueden ser alcanzados. Actualmente, los principios sobre los cuales se basan los controladores PID, son complementados por una serie de prestaciones que mejoran las características en cuanto a su desempeño, tales como las técnicas *anti-windup* y las técnicas de conmutación de modos de control.

El controlador PID es una ley de control basada en el error obtenido entre la señal de referencia (y_{sp}) y la salida del proceso (y). Esto teniendo en cuenta que dicho controlador hace parte de un sistema realimentado, en el que se tiende a mantener una relación preestablecida entre la salida y la entrada de referencia, como se muestra en la figura 1.

[7] A.S., Erbay. An Overview on PID Control, 2000

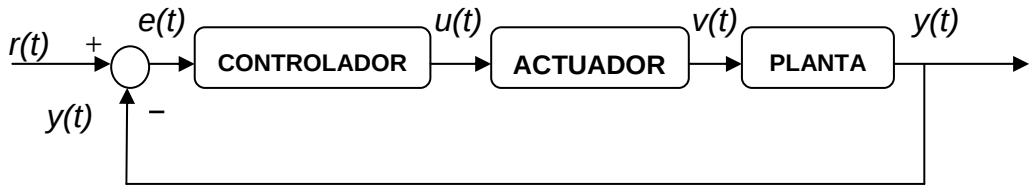


Figura 1. Diagrama de bloques para un sistema de control con realimentación negativa

Un controlador PID tiene como entrada el error definido por *e(t)* = *r(t)-y(t)* y como salida la señal de control *u(t),* dada por la ecuación 1:

$$u(t) = K_P\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\,\frac{de(t)}{dt} \right) \qquad (1)$$

El controlador PID en su implementación práctica, puede presentar fallos debido a condiciones externas o propias del lazo de control, como pueden ser las condiciones de operación, perturbaciones externas, fallas sobre los equipos o valores de consigna muy exigentes, entre otras. A continuación, se tratarán dos fenómenos presentados en la implementación del control PID y una alternativa de solución: Amplificación de ruido debido a la acción derivativa y efecto *Wind-up* del integrador.

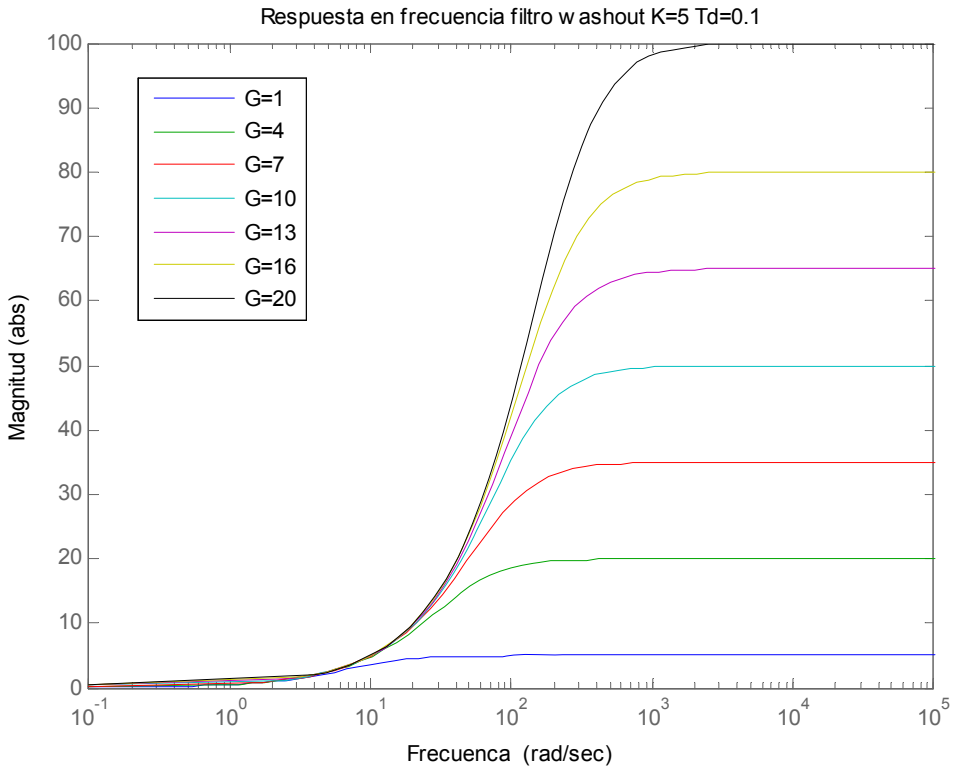
1.1.1 Limitación de la ganancia derivativa.

La limitación de la ganancia derivativa es necesaria en la implementación de los controladores PID, debido al efecto negativo de amplificación de ruido no deseado en la señal de error. Esto puede provocar grandes cambios en la señal de control que incide sobre los actuadores y puede generar daños en ellos. Para ello se limita la ganancia, mediante un filtro *wash-out* implementado dentro del término derivativo, tal como se formula en la ecuación 2 [8].

[8] AMÉSTEGUI, Mauricio. Apuntes de Control PID, 2001. p.18

$$D = -\frac{sKT_d}{1 + s\frac{T_d}{G}}\, y \qquad (2)$$

En donde; G es la ganancia del filtro y es denominado término *wash-out* y K*G es la máxima ganancia del bloque diferencial para altas frecuencias de las señales de entrada. En la figura 2 se muestra la respuesta en frecuencia del término diferencial con limitación de ganancia.



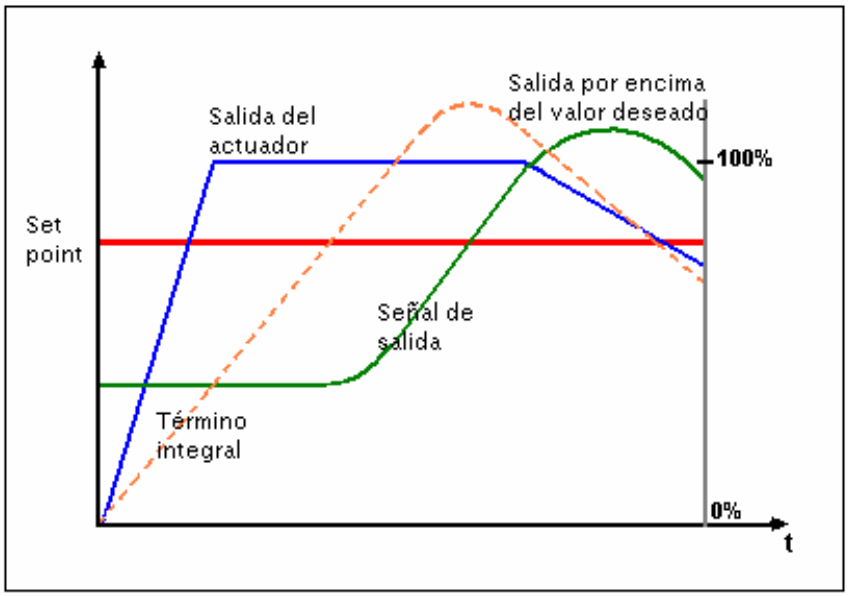
Fuente: Generada en Simulink

Figura 2. Respuesta en frecuencia del término diferencial

1.1.2 *Wind-up* del Integrador.

Otro efecto adverso en la implementación práctica del controlador PID, es el efecto denominado saturación del término integral o *windup*. Son varias las causas de este efecto, tales como perturbaciones, mal funcionamiento de los equipos o valores muy altos de la consigna. Todos ellos contribuyen en el incremento de la

salida del integrador, lo que puede causar daños en los actuadores, dado que la señal de entrada sobrepasa los límites de diseño. En la figura 3 se muestran las señales que intervienen en el efecto *windup*, debido a un alto valor de consigna.

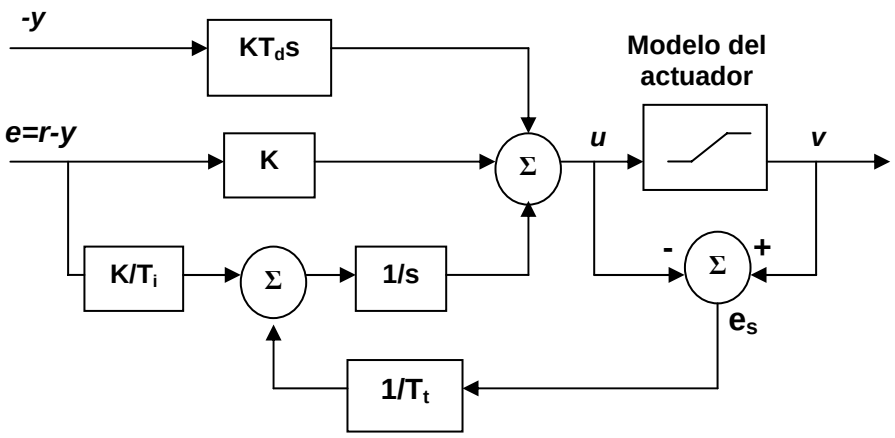


Fuente: Basado en Piedrahita[9]

Figura 3. Efecto *wind-up* debido a un alto valor de consigna

Con el fin de evitar este efecto, se han desarrollado varios esquemas anti-windup entre los que se encuentran: la limitación del término integral, la integración condicional, la limitación de la referencia, y el recálculo y seguimiento integral. Este último esquema tiene la ventaja de no reiniciar el integrador de manera instantánea, pero si de manera dinámica con una constante de tiempo T_t. La figura 4 muestra el diagrama de bloques de un controlador PID con *anti-windup*, implementado mediante la técnica de recalculo y seguimiento[10].

[9] PIEDRAHITA M., Ramón. Implementación programada de reguladores.
[10] SALAZAR. M. Control Adaptativo: Una alternativa de control automático, 1994.



Fuente: Basado en Améstegui[11]

Figura 4. Implementación controlador PID con anti-*wind-up* de recalculo y seguimiento

Este modelo tiene un lazo de realimentación extra, generado por la medición de la salida del actuador (modelo) y la formación de una señal de error e_s, que es la diferencia entre la salida del actuador y el controlador. La señal e_s es alimentada a la entrada del integrador, a través de la ganancia $1/T_t$- Dicha señal es cero (0) cuando no existe saturación, por tanto no tiene ningún efecto sobre la operación normal del controlador y es diferente de cero cuando existe saturación en el actuador.

El parámetro T_t es llamado constante de seguimiento y determina que tan rápido la integral es puesta en reset. Aparentemente sería óptimo utilizar valores bajos de T_t para evitar rápidamente la saturación del actuador, pero se debe tener cuidado cuando existe acción derivativa en el controlador, ya que pueden generarse falsos errores en la señal de medición que pueden poner en reset la integral aún sin ser requerido- Es por ello que la constante T_t debe ser mas alta que T_d y más pequeña que T_i. Una manera práctica de calcular este parámetro es mediante la ecuación 3[12].

$$T_t = \sqrt{T_d T_i} \qquad (3)$$

[11] AMÉSTEGUI, Mauricio. Apuntes de Control PID, 2001.
[12] Ibid., p.28.

1.2 CONTROL ADAPTATIVO

Usando la definición de Salazar[13], un sistema de control adaptativo *"es aquel que continua y automáticamente mide las características dinámicas de la planta, las compara con las características deseadas y usa la diferencia para variar parámetros ajustables del sistema o para generar una señal de accionamiento, de modo que se pueda mantener el funcionamiento óptimo con independencia de las variaciones externas*".

Según sea el diseño de los bloques que conforman el esquema de control adaptativo, se pueden establecer tres clases de controladores adaptativos:

- Controladores adaptativos basados en modelo de referencia
- Reguladores adaptativos auto-ajustables
- Control supervisorio

Cada una de estas clases presenta ventajas y desventajas respecto a la otra, por tanto no resulta conveniente señalar el predominio de una en especial. Es decir, su buen o mal desempeño va ligado a la naturaleza de la planta y a las condiciones a las cuales está sometida. En el desarrollo del proyecto se adoptará específicamente como estrategia de control, el control adaptativo basado en modelo de referencia.

1.2.1 Control adaptativo basado en modelo de referencia (MRAC).

El esquema de control adaptativo basado en modelo de referencia, está conformado por tres elementos principales[14]:

- Controlador Primario: este controlador puede ser alguna de las configuraciones de controladores lineales conocidas, con la condición de que el sistema completo de lazo cerrado, pueda replicar al modelo de referencia, lo que supone restricciones sobre el orden y la estructura del controlador.

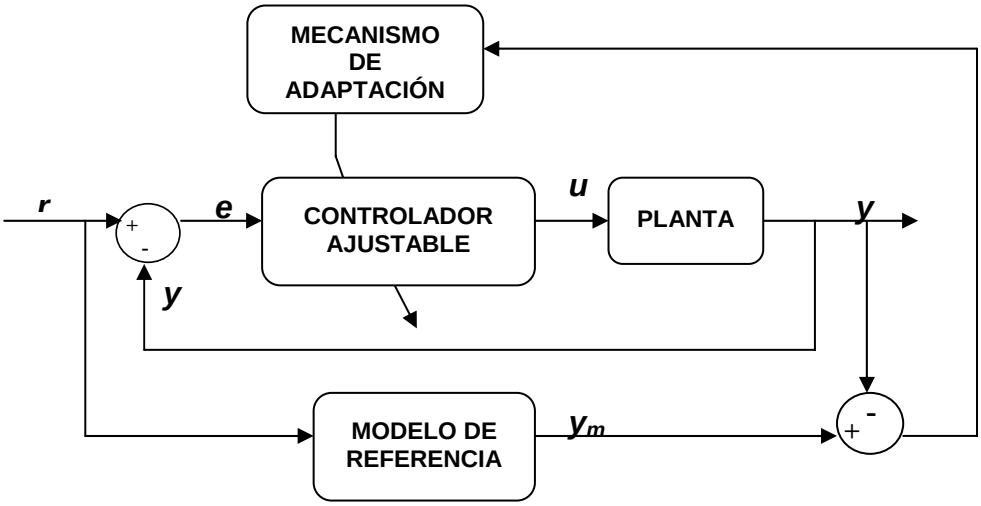
[13] SALAZAR. M. Control Adaptativo: Una alternativa de control automático, 1994. p.4.
[14] RODRIGUEZ, Francisco. LÓPEZ, Manuel. Control adaptativo y robusto 1996 Universidad de Sevilla, p. 48-49

- Modelo de referencia: especifica el comportamiento deseado en lazo cerrado del sistema.

- Mecanismo de Adaptación: Es el bloque encargado de calcular los parámetros del controlador, tal que la salida del proceso se aproxime a la respuesta deseada, a partir de la información obtenida del modelo de referencia y del proceso.

En la dinámica de este esquema, el controlador primario es usado para obtener el comportamiento en lazo cerrado (esquema de control convencional), la respuesta deseada a una señal entrada de entrada del proceso se especifica como un modelo de referencia, mientras que el mecanismo de adaptación obtiene el error entre la salida del modelo de referencia (y_m) y la señal de salida del proceso (y), y a partir de esta señal de error calcula lo nuevos parámetros del controlador, de forma que la diferencia tienda a cero.

El esquema de control adaptativo puede encontrarse representado de diversas formas, de acuerdo a la disposición del modelo de referencia. La más usada es la disposición en paralelo, aunque es posible encontrarlo en serie con el proceso. La estructura paralelo puede verse en la figura 5.



Fuente: Basado en Rodríguez[15]

Figura 5. Esquema de control adaptativo con modelo de referencia (MRAC)

[15] RODRÍGUEZ, Francisco. LÓPEZ, Manuel. Control adaptativo y robusto, 1996. Universidad de Sevilla

El diseño del mecanismo de adaptación puede realizarse mediante técnicas como la Regla de MIT, el Método de Lyapunov o el Método de Hipersensibilidad. Debido a que el proyecto no va enfocado al diseño del mecanismo de adaptación, en el presente documento no se hace énfasis en la fundamentación teórica de estás técnicas.

1.3 COMUNICACIÓN MEDIANTE *SOCKETS* SOBRE TPC/IP

La interconexión de cientos de computadoras por medio de redes internas o sobre Internet ha sido desarrollada en su mayoría sobre la pila de protocolos TCP/IP. Esta permite un manejo seguro en el envío y transmisión de datos y genera un sistema de comunicación fiable. La comunicación a través del modelo TCP/IP se lleva a cabo mediante el intercambio de datos entre capas. A medida que los datos descienden a través de las capas en el emisor, es agregada una cabecera que permite que los datos sean reconocidos en la capa equivalente del receptor. Cuando el dato es procesado por una capa (en el receptor), la información correspondiente a la cabecera es suprimida para permitir avanzar a una capa superior. En el esquema de la figura 6 se muestra este proceso.

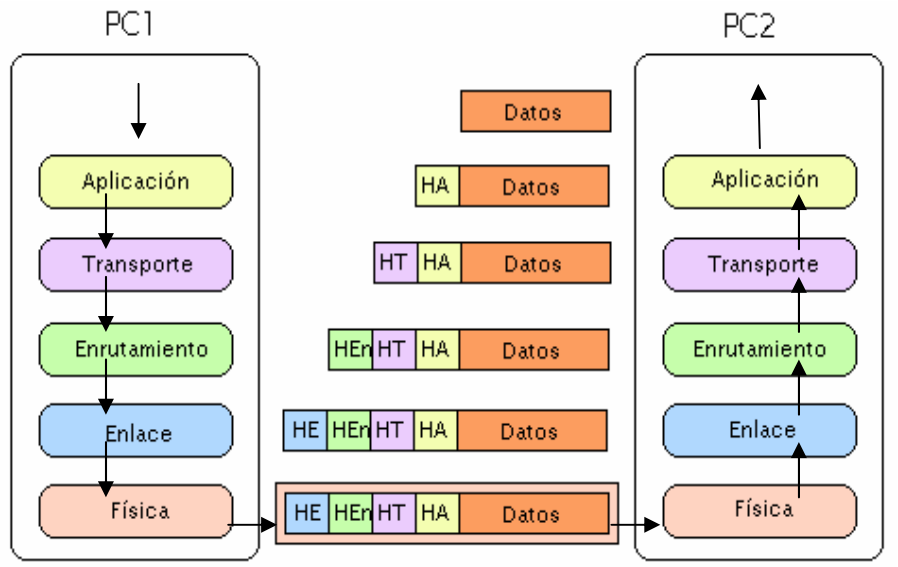


Figura 6. Esquema de comunicación entre capas (TCP/IP)

1.3.1 API de *Sockets*.

El API de *sockets* es una interfaz de programación de aplicaciones basada en librerías que proporciona comunicación entre procesos, usando los servicios de la capa de transporte sobre la pila de protocolos TCP/IP. Esta aplicación fue desarrollada para sistemas UNIX (Linux), pero existen equivalentes para otros sistemas operativos como *Winsock* (Microsoft), *MacTCP* (Mac OS) entre otras.

Los *sockets* son mecanismos de comunicación bidireccional que permiten que un proceso intercambie información con otro, estando o no sobre la misma máquina. Un *socket* se define completamente a partir de la dirección IP y del puerto utilizado para la conexión. Para identificar un *socket* se utiliza un descriptor, generado al momento de crearlo, llamado por los procesos que lo requieran. Es decir, todas las operaciones que se realizan sobre el *socket* deben hacer referencia al descriptor generado. Las operaciones más comunes en *sockets* son: la creación del *socket*, la asignación de un nombre, la llamada *listen*, escritura-lectura y cierre del *socket*, entre otras.[16]

La comunicación mediante *sockets* se basa en el esquema Cliente-Servidor: un proceso actuará como servidor creando el *socket* cuyo nombre es conocido por el cliente, quién se podrá comunicar con el servidor una vez haya establecido la conexión. Básicamente los *sockets* están definidos por dos características fundamentales:

a. El tipo de *socket*, que define la naturaleza del mismo y el tipo de comunicación que puede generarse entre ellos. Los tipos de *sockets* disponibles son:

- SOCK_DGRAM: *sockets* para comunicaciones en modo no conectado, con envío de datagramas de tamaño limitado. El nivel de transporte sobre el que se basa es el UDP.

[16] DONAHOO, Michael. CALVERT, Kennet. TCP/IP *Socket*s en C, Practical Guide for Programmers. 2001

- SOCK_STREAM: para comunicaciones fiables en modo conectado, de dos vías y con tamaño variable de los mensajes de datos. Uso del protocolo TCP.

- SOCK_RAW: permite el acceso a protocolos de más bajo nivel como el IP.

- SOCK_SEQPACKET: tiene las características del SOCK_STREAM pero el tamaño de los mensajes es fijo.

b. El dominio del *socket*, que especifica el conjunto de *sockets* que pueden establecer una comunicación con el mismo y donde se encuentran los procesos que se van a comunicar. Algunos de los dominios disponibles son:

- AF_UNIX: los procesos están en una misma máquina.

- AF_INET: los procesos están en maquinas distintas y se hallan unidos mediante una red TCP/IP.

2. MODULO DE COMUNICACIÓN

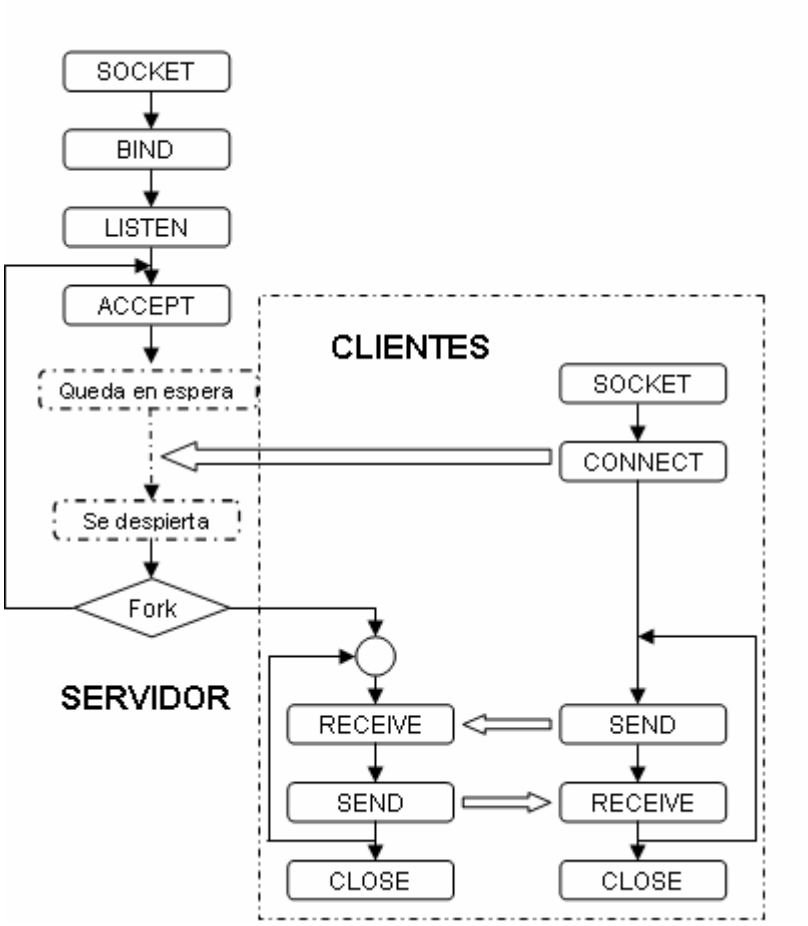
La implementación del modulo de comunicación se realiza con el API de *sockets* mediante programación en lenguaje C sobre Linux, sistema operativo con el que cuenta la Tarjeta ECB_AT91 V1 que es utilizada para el desarrollo de este proyecto. Antes de iniciar la programación se verifica la existencia de las librerías requeridas para la creación y manejo de *sockets* sobre los terminales donde se encuentran implementados los procesos a comunicar (ECB_AT91 – PC1).

2.1 ESQUEMA BÁSICO DE COMUNICACIÓN

La interfaz de *sockets* a utilizar trabaja sobre el modelo de capas de TCP/IP, por tanto la comunicación establecida será orientada a conexión, lo que garantiza que los datos llegan correctamente de un proceso a otro.

En el esquema convencional de comunicación con *sockets*, el servidor crea solo un descriptor de *socket* y este se encarga tanto del manejo de conexiones, envío y transmisión. Una vez concluido el intercambio de datos se cierra el *socket* y es necesario ejecutar de nuevo la aplicación para poder establecer conexión con el cliente.

El esquema a implementar difiere respecto al esquema convencional, puesto que se crea un proceso hijo que se encargará del manejo de datos (transmisión-recepción), mientras que el proceso padre estará alerta a la espera de nuevas llamadas de clientes. El uso de procesos hijos está ligado al concepto de multitarea, puesto que el servidor podrá aceptar múltiples conexiones aún cuando se está transmitiendo o recibiendo datos de un cliente específico. El esquema de comunicación implementado se muestra en la figura 7. En este se declaran las diferentes llamadas hechas para la creación de un *socket*, así como las necesarias para llevar a cabo la conexión y posterior recepción y envío de datos.



Fuente: Basado en Donahoo[17]

Figura 7. Esquema de comunicación concurrente mediante *sockets*

Basados en los requerimientos de la comunicación, establecidos para el proyecto se plantea la siguiente descripción general del esquema:

1. Crear *socket* en cada máquina a comunicar.
2. El servidor asigna un nombre al *socket* para recibir conexiones (paso opcional para el cliente).
3. Habilitar el servidor para recibir conexiones, quedando el *socket* principal en estado de alerta.

[17] DONAHOO, Michael. CALVERT, Kennet. TCP/IP *Socket*s en C, Practical Guide for Programmers. 2001

4. Realizar petición desde el cliente para establecer un canal de comunicación.
5. Aceptar petición desde el servidor, mediante la creación de un proceso hijo que atenderá al cliente enlazado.
6. Liberar al *socket* principal del servidor para establecer nuevas conexiones.
7. Realizar intercambio de datos entre el *socket* cliente y el *socket* hijo creado en el servidor.
8. Cerrar todos los *sockets* creados, al finalizar la comunicación.

2.2 IMPLEMENTACIÓN CLIENTE-SERVIDOR

Para iniciar la implementación de este módulo, se debe identificar que proceso va a operar como servidor y cual como cliente. Basados en el esquema de control planteado, los procesos que deben comunicarse son el mecanismo de adaptación y el controlador. Es apropiado asignar como servidor, la estación donde está el mecanismo de adaptación y como cliente el dispositivo donde se programa el controlador, puesto que si fuera necesario tener varios sistemas de control trabajando paralelamente y ajustando sus parámetros, cada controlador actuaría como cliente en el esquema de comunicación establecido.

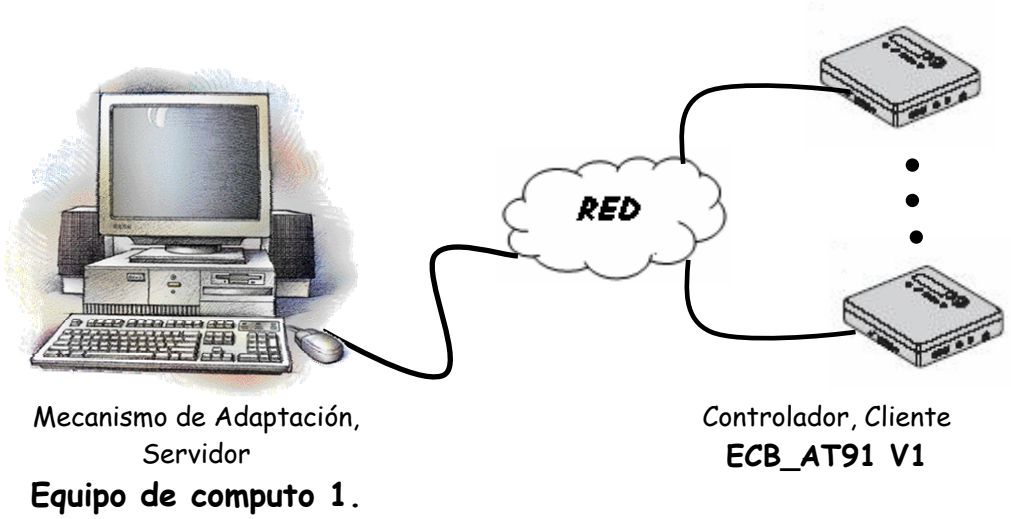


Figura 8. Esquema de asignación Cliente-Servidor

En la programación del servidor y el cliente, se utiliza la estructura de dirección del protocolo de Internet, llamada `struct sockadrr_in`.. Con ella

se asigna un dominio específico para la aplicación, el número del puerto y la dirección IP.

2.2.1 Servidor.

El trabajo del servidor es configurar un punto de comunicación y esperar pasivamente la conexión de un cliente. Los pasos básicos para llevar a cabo la comunicación desde el servidor, son:

1. Crear un *socket* TCP usando la operación `socket()`.
2. Asignar un número de puerto al *socket* mediante `bind()`.
3. Decirle al sistema que permita conexiones asociadas con el puerto asignado usando `listen()`.
4. Repetidamente hacer lo siguiente:

- Llama `accept()` para obtener un *socket* auxiliar para cada nuevo cliente.
- Se crea un proceso hijo con `fork`(), para iniciar el intercambio de datos.
- Comunica cada cliente mediante el nuevo *socket* usando `send`() y `recv`().
- Cerrar la conexión del *socket* auxiliar (`close`()).

5. Cerrar la conexión del *socket* principal mediante `close()`.

Para que el cliente pueda realizar conexión con el servidor, el servidor debe tener asignada una dirección local y un puerto. Es decir, mientras que el cliente debe especificar la dirección de la maquina a la cual se va a conectar, el servidor únicamente debe especificar su propia dirección.

La secuencia de comandos descrita para el servidor, se programa en lenguaje C y su código (SERVIDOR.c) se encuentra adjunto en el Anexo D. Códigos de Programación

2.2.2 Cliente.

La distinción entre cliente y servidor es muy importante, puesto que cada uno utiliza una interfaz diferente para realizar ciertos pasos de comunicación. El trabajo del cliente es contactar a un servidor que está pasivamente esperando peticiones de conexión.

Los pasos que se llevan acabo en la implementación del cliente TCP son:

1. Crea un *socket* mediante la llamada `socket`().
2. Establece la comunicación utilizando `connect`().
3. Se comunica utilizando `send`() y `recv`().
4. Cierra la conexión con `close`().

El código que representa esta secuencia (CLIENTE.c), se programa sobre lenguaje C y está adjunto en el Anexo D. Códigos de Programación.

2.2.3 Consideraciones generales de la programación.

Además de las llamadas básicas para configuración, conexión y comunicación de *sockets*, existen funciones de control que permiten acceder a opciones avanzadas. Una de estas opciones fue utilizada ya que el servidor implementado es un proceso repetitivo, por tanto es necesario permitir la reutilización de la dirección IP de la máquina, mediante la función `setsockopt`().

Otra característica de los *sockets* que es posible modificar, es el modo de operación, para definir si el *socket* es bloqueante o no. El modo de operación de los *sockets* programados es bloqueante, es decir, siempre esperan respuesta a cualquier tipo de petición (lectura, escritura, etc.), quedando bloqueado indefinidamente hasta que la llamada realizada se lleve a cabo. A diferencia del modo bloqueante, cuando un *socket* trabaja como no bloqueante, envía una petición y si no recibe una respuesta en un lapso de tiempo determinado, simplemente finaliza la petición.

2.3 MANEJO DE VARIABLES

Además de la transmisión de datos por medio del API de *sockets*, se realiza el intercambio de variables entre programas sobre un mismo equipo (estación de trabajo o tarjeta de desarrollo). Esto se implementa mediante el manejo de archivos de texto, sobre los que es posible escribir y leer las variables que se envían sobre TCP/IP.

Para dar a entender la forma en que se manejan estas variables, se presenta un esquema en la Figura 8 y a continuación una explicación de la ruta de cada una de las variables.

En el esquema se muestran los dos equipos conectados mediante sockets y las acciones de escritura-lectura representadas mediante flechas. Las flechas rojas llegan a los programas que leen los archivos de texto, las verdes salen de los programas que escriben sobre los archivos, las azules hacen referencia a la comunicación mediante sockets (TCP/IP) y las amarillas representan las señales de salida y entrada, hacia la estación donde está emulado el proceso.

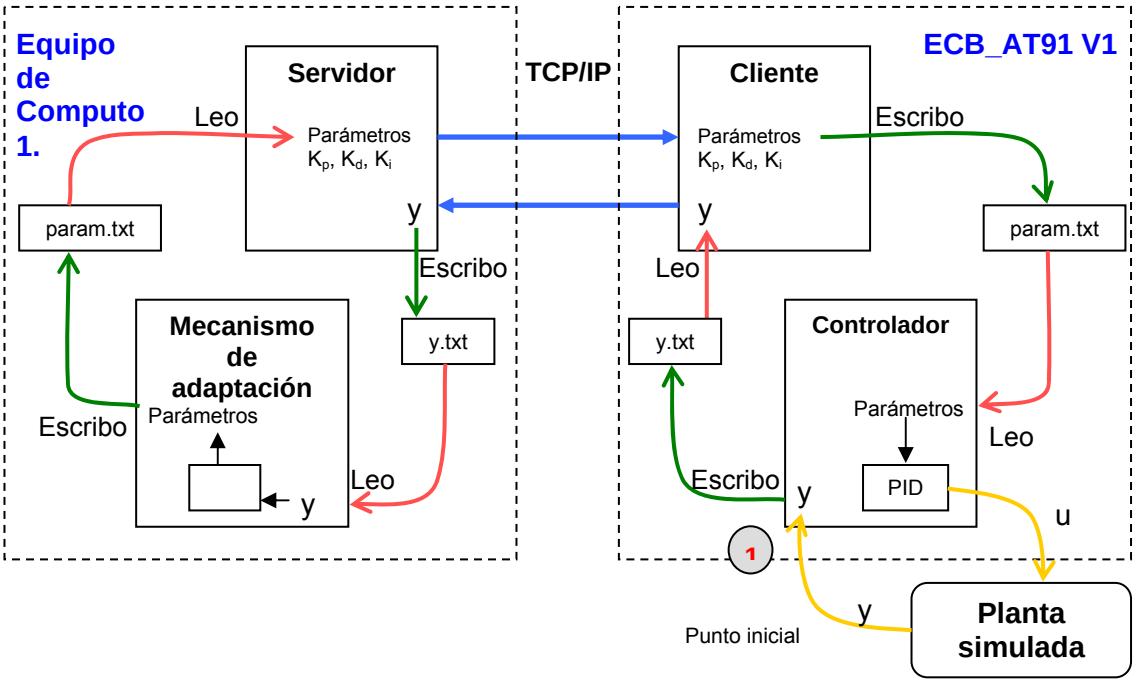


Figura 9. Esquema de manejo de variables

Para iniciar la explicación, se supone como punto de partida el retorno de la señal *y*, que resulta de una señal de control inicial (ver punto 1 en el esquema figura 8).

Señal *y*, salida del proceso a controlar. El valor de esta señal es almacenado en un archivo de texto (y.txt), que es creado sobre de la tarjeta de desarrollo, dentro de la ejecución del controlador. Posteriormente es leído del archivo por el programa cliente, que requiere este dato para enviarlo al servidor. En el servidor se hace un procedimiento similar para llevar la variable hasta el mecanismo de adaptación y realizar su posterior lectura y uso. Allí se crea un

archivo homónimo al de la tarjeta de desarrollo, pero generado sobre la estación de trabajo.

K_p, K_d, K_i, Parámetros del controlador. Estos datos son calculados en el mecanismo de adaptación y almacenados en el archivo param.txt, en seguida se leen por el programa servidor para enviarlos al cliente. Cuando estos valores llegan al programa cliente son escritos en un el archivo de texto (param.txt sobre la tarjeta de desarrollo). Archivo que es leído al inicio de la ejecución del controlador, para hacer uso de dichos parámetros dentro del esquema de control.

3. DISEÑO E IMPLEMENTACIÓN DEL CONTROLADOR

En el presente proyecto de grado se diseñó e implementó un controlador PID digital, utilizando criterios matemáticos para su diseño y planteando algoritmos de programación, que en conjunto generan la ley de control.

3.1 CRITERIOS MATEMÁTICOS

El diseño parte de la consideración del controlador PID analógico como prototipo de referencia. Sin embargo este controlador en su forma estándar pude presentar efectos negativos de implementación sobre la señal de control, por lo que se incluyen mejoras para su implementación como son, la limitación de la ganancia de alta frecuencia del término derivativo (Filtro Wash-out) y el anti-windup. La ley control resultante se representa mediante la ecuación 5.

$$U(s) = Ke(s) - \frac{KsT_d}{1 + sT_d/N}Y(s) + \frac{K}{T_i s}e(s) + \frac{1}{Tt + s}(V(s) - U(s)) \quad (5)$$

Donde U(s),Y(s),e(s),V(s) corresponden a las transformadas de Laplace de u(t),y(t) señal de salida del proceso, e(t) y v(t) señal de salida del modelo del actuador dado por la implementación del *anti-wind* up (Véase figura 4).

A partir de la ecuación (5) se realiza la aproximación numérica sobre cada uno de los términos de la señal de control. A continuación se muestra la función de transferencia discreta obtenida para cada uno ellos.

- **Acción proporcional**

El término proporcional no requiere de una aproximación ya que este presenta solo parte estática[18], debido a esto su función discreta estada como:

$$P(z) = Ke(z) \quad (6)$$

[18] ÅSTRÖM, Karl J. WITTERNMARK, Björn. Computer-Controlled Systems, Theory and Desing. p. 308

- **Acción Derivativa**

Para la implementación del término derivativo se utilizó la aproximación de diferencias en atraso o conocida también como método de Euler. Esta presenta un comportamiento estable para la mayoría de valores de tiempo derivativo (Td), ya que el polo tiende a cero cuando Td tiende a cero.

El término obtenido mediante la aproximación esta dado como:

$$D(z) = \frac{KT_d N(1 - z^{-1})}{TN + T_d(1 - z^{-1})} Y(z) \tag{7}$$

- **Acción Integral**

La aproximación utilizada para la discretización del término integral es la dada por la aproximación de Tustin. Esta aproximación tiene un comportamiento más cercano a la función de transferencia de tiempo continuo.

La función discreta esta dada como:

$$I(z) = \frac{KT}{2T_i} \frac{(1 + z^{-1})}{(1 - z^{-1})} e(z) + \frac{T}{2T_t} \frac{(1 + z^{-1})}{(1 - z^{-1})} (V(z) - U(z)) \tag{8}$$

Utilizando cada una de las aproximaciones anteriores, la señal de control se describe mediante,

$$U(z) = P(z) - D(z) + I(z) \tag{9}$$

Esta función discreta obtenida en el dominio de la frecuencia es transformada al dominio del tiempo discreto. De esta forma se encuentra la ecuación en diferencias que se solucionará a través de cálculos iterativos realizados por medio de programación.

El diseño se desarrolló para un controlador PID, PI y PD. A continuación se presentan las ecuaciones en diferencias obtenidas para cada uno de ellos.

Control PID

$$u(k)=\frac{a_1e(k)+a_2e(k-1)+a_3e(k-2)-b_1y(k)+b_2y(k-1)-b_3y(k-2)+c_1v(k)+c_2v(k-1)-c_3v(k-2)-d_1u(k-1)-d_2u(k-2)}{(TN+T_d)\left(1+\frac{T}{2T_t}\right)}$$

$$a_1=K(TN+T_d)\left(1+\frac{T}{2t_i}\right)$$

$$a_2=K\left(\frac{T^2N}{2T_i}-2T_d-TN\right)$$

$$a_3=KT_d\left(1+\frac{T}{2T_i}\right)$$

$$b_1=b_3=KT_dN$$

$$b_2=2KT_dN$$

$$c_1=\frac{T}{2T_t}(TN+T_d)$$

$$c_2=\frac{T^2N}{2T_t}$$

$$c_3=\frac{T_dT}{2T_t}$$

$$d_1=\left(\frac{T^2N}{2T_t}-2T_d-TN\right)$$

$$d_2=T_d\left(1-\frac{T}{2T_t}\right)$$

Control PI

$$u(k) = \frac{a_1 e(k) + a_2 e(k-1) + b_1 v(k) + b_2 v(k-1) + c_1 u(k-1)}{1 + \frac{T}{2T_t}}$$

$$a_1 = K\left(1 + \frac{T}{2T_i}\right)$$

$$a_2 = K\left(\frac{T}{2T_i} - 1\right)$$

$$b_1 = b_2 = \frac{T}{2T_t}$$

$$c_1 = 1 - \frac{T}{2T_t}$$

Control PD

$$u(k) = \frac{a_1 e(k) - a_2 e(k-1) - b_1 y(k) + b_2 y(k-1) - T_d u(k-1)}{TN + T_d}$$

$$a_1 = K\left(TN + T_d\right)$$

$$a_2 = KT_d$$

$$b_1 = b_2 = KT_d N$$

3.2 ESPECIFICACIONES DE PROGRAMACIÓN

El desarrollo del código de programación realizado para calcular la acción de control, tuvo en cuenta las siguientes especificaciones:

- El programa presenta como entradas:
    - Las condiciones iniciales de los parámetros del controlador, con el objetivo de utilizar el sistema de control para cualquier aplicación requerida. Este argumento es utilizado solo durante el primer lazo de control.
    - La elección del controlador requerido (PID, PI o PD). Se determina esta condición debido a que algunas aplicaciones solo requieren de un control PD o PI para la implementación de la acción de control.
    - El periodo de muestreo. Utilizado para los cálculos realizados en el programa, puesto que este tiempo depende de la dinámica del sistema que se desea controlar.
- Como salidas presenta:
    - La acción de control. Esta será procesada para ser enviada al sistema simulado en la plataforma Matlab/Simulink
- Procesos internos:
    - Después de haber realizado el primer lazo de control, las condiciones de los parámetros del controlador se obtiene de la lectura de un archivo de texto escrito en el programa Cliente. Para esto previamente se obtienen los parámetros a través de la interfaz de comunicación establecida con el punto de conexión del Terminal, donde se encuentra el sistema adaptador.
    - Cálculo de la señal de control mediante el desarrollo recursivo de la ecuación en diferencias.
    - El programa cliente se llama a ejecución con el fin de realizar la transmisión de la señal de salida dada por el sistema simulado en la plataforma Matlab/Simulink

4. PROTOTIPO DE PRUEBA

La implementación del prototipo de prueba se lleva cabo mediante la interacción de tres sistemas, a saber:

i. Equipo de cómputo 1 (PC1), En este se programa el algoritmo de adaptación que determina los parámetros del controlador acorde a las variaciones existentes en la planta (el diseño y desarrollo de este algoritmo no hace parte de este proyecto de grado). Además en esta Terminal se encuentra el servidor que transfiere las diferentes señales de adaptación. Esta estación de trabajo esta bajo el sistema operativo Linux Debian 4.0 Etch

ii. ECB_AT91 V1. En esta tarjeta de desarrollo se encuentra programado el algoritmo de control y el programa del cliente, que se ejecutan mediante acceso remoto a la tarjeta utilizando el protocolo SSH (Secure SHell). Este protocolo se ejecutará en el equipo PC1.

iii. Equipo de cómputo 2 (PC2). En esta Terminal se encuentra el proceso a emular implementado bajo la plataforma de simulación Matlab/Simulink.

La comunicación entre PC1 y la ECB_AT91 V1 se realiza mediante el uso del API de *sockets*, (explicada en el capitulo Modulo de Comunicación) mientras que la interfaz de comunicación entre la tarjeta de desarrollo y PC2 se realiza utilizando la interfaz serial SPI. A continuación se muestra el esquema general del prototipo de prueba.

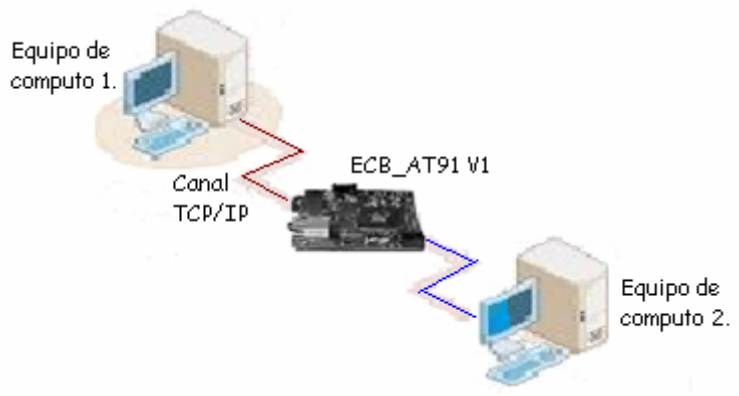


Figura 10. Esquema del prototipo de prueba

4.1 EQUIPO DE CÓMPUTO 1.

Este equipo trabaja sobre el sistema operativo Linux. Allí se encuentran y ejecutan los programas del *socket* servidor y del mecanismo de adaptación, que se comunican con la tarjeta de desarrollo mediante la interfaz de *sockets*, soportada en TCP/IP. La programación del punto de conexión para *sockets* en este equipo fue desarrollada en capítulos anteriores.

El mecanismo de adaptación se desarrolla con técnicas de adaptación básicas, puesto que su implementación no hace parte de los objetivos de este proyecto, pero es necesario un algoritmo básico para la realización de las pruebas finales.

4.2 TARJETA DE DESARROLLO ECB_AT91 Versión 1

La ECB_AT91 fue desarrollada en marzo de 2006, por el Profesor de la Universidad Nacional de Colombia, Carlos Iván Camargo como parte de sus tesis Doctoral. Esta tarjeta de desarrollo cuenta con las siguientes características:

Especificaciones de Hardware

- Procesador ARM 180MHz (ATMEL AT91RM9200)
- Flash Serial de 2Mbytes
- Hasta 64Mbytes de SDRAM (soporta 8M/16M/32M/64M)
- 1 Ranura SD/MMC
- 1 Interfaz Ethernet Intel 10/100
- 1 Interfaz USB de alta velocidad
- 4 Interfaces SPI
- 2 Interfaces serials (RS232)
- Soporte JTAG

Sobre este procesador se encuentra el sistema operativo Linux distribución Debian Etch 4.0, además de todas las herramientas necesarias para utilizar las diferentes interfaces con la que cuenta la ECB_AT91.

La principal razón para justificar el uso de esta tarjeta de desarrollo es la ventaja de contar con un soporte fiable para llevar a cabo los diferentes esquemas planteados, además de la disponibilidad de esta tarjeta para el desarrollo de proyectos de grado en la Universidad.

Las características de este dispositivo permiten realizar la programación del controlador y el programa del *socket* cliente, como se explicó en capítulos anteriores.

La salida del controlador digital se transmite mediante el puerto RS232 desde la ECB_AT91 hasta el equipo de computo 2, en donde se encuentra emulado el proceso a controlar, el cual a su vez envía el valor de salida mediante el mismo protocolo para completar el lazo de realimentación planteado en el esquema de control.

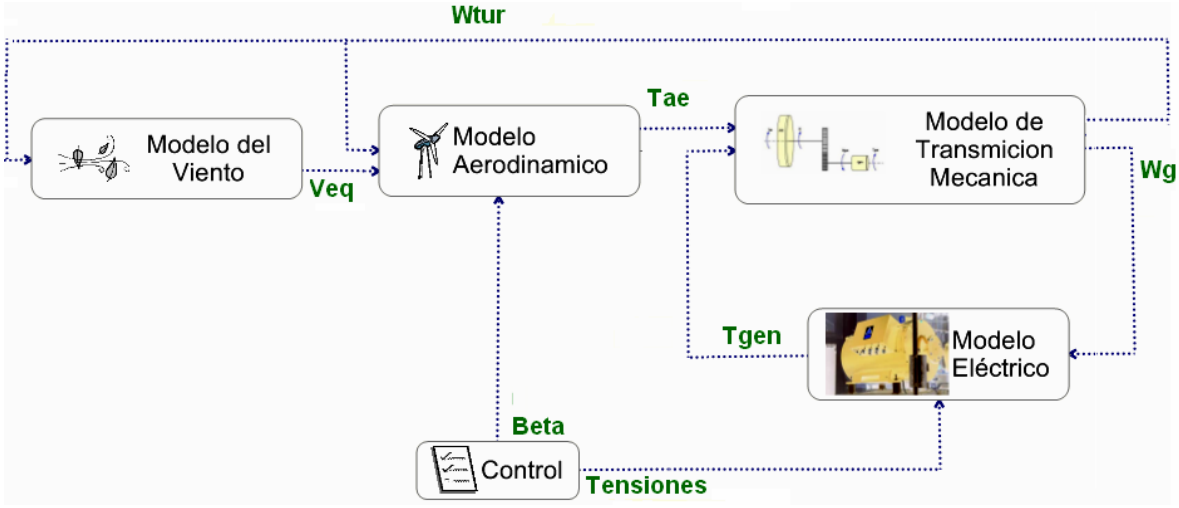
Para intercambiar estas señales entre el proceso simulado en Simulink/MATLAB y el controlador, implementado en la ECB_AT91, se hizo uso de la herramienta "COMMSTR7.02.0.5" la cual permite transmisión y recepción de datos, mediante el puerto serial.

4.3 EQUIPO DE CÓMPUTO 2.

En este Terminal se encuentra la emulación del proceso del prototipo de prueba. Se eligió como proceso el control de generación de una turbina eólica, puesto que es un proceso no lineal y su comportamiento dinámico sufre variaciones ocasionadas en gran parte por las turbulencias presentes en la velocidad del viento. El modo de operación de este sistema se simula en la plataforma MATLAB/Simulink, utilizando la Toolbox para aplicaciones de turbinas eólicas desarrollado en el proyecto ""*A Simulation Platform to Model, Optimize and Design Wind Turbines*" realizado en cooperación entre "*Aalborg University*" y "*RISØ National Laboratory*". Su uso esta restringido a trabajos de investigación y de docencia[19].

El esquema general del sistema de la turbina eólica a simular se ilustra en la figura 11.

[19] SØRENSEN, P., HANSEN, A.D., ROSAS, P.A.C. Wind models for simulation of power fluctuations from wind farms, Journal of Wind Engineering, 90, 2002, p.1381-1402



Fuente: Basado en Uscátegui & Prada[20]

Figura 11. Esquema de la turbina eólica

A continuación se realiza una breve descripción de cada uno de los bloques que conforman el sistema.

4.3.1 Modelo del viento.

La combinación del los efectos estocásticos (aleatorios) causados por la turbulencia y efectos determinísticos (causas precisas y conocidas) representados por un perfil promedio de la velocidad del viento son la característica fundamental de este bloque. El modelo se basa en el espectro de Kaimal, en donde la velocidad equivalente del viento (Veq) es calculada como el promedio de la velocidad en un punto fijo sobre todo el rotor teniendo en cuenta la turbulencia rotacional[21].

El modelo tiene como salida la velocidad equivalente del viento (Veq) y como entrada la velocidad angular del rotor (Wtur), obtenida del modelo de transmisión mecánica.

[20] USCÁTEGUI, Omar. PRADA, Darío. Modelado y simulación del sistema mecánico de un generador eólico de eje horizontal con tres palas, UIS. 2007 p.101.
[21] SØRENSEN, P., HANSEN, A.D., ROSAS, P.A.C. Wind models for simulation of power fluctuations from wind farms, Journal of Wind Engineering, 90, 2002, p.1381-1402;

4.3.2 Modelo Aerodinámico.

El modelo se fundamenta en el coeficiente de torque (Cq) que teóricamente es el coeficiente de potencia (Cp) dividido por la velocidad específica (λ). El coeficiente de potencia (Cp) expresa que cantidad de la potencia total que posee el viento incidente es realmente capturada por el rotor de dicho aerogenerador. Este coeficiente a su vez depende de la velocidad específica y del ángulo de paso (β), la aproximación numérica utilizada para el desarrollo del modelo es la propuesta por Siegfried Heier la cual hace referencia a turbinas con ángulo de paso variable[22].

En la simulación, este coeficiente de torque esta tabulado con base en pruebas experimentales desarrolladas por los organismos encargados del proyecto de desarrollo de la *toolbox* en el año 2003 y aceptado por el Código Danés para la construcción de turbinas de viento. La tabla implementada en el modelo se muestra en el Anexo B.

El modelo tiene como salida el torque aerodinámico (Tae) y como entradas la velocidad equivalente del viento (Veq) proveniente del modelo del viento ,la velocidad angular del rotor de la turbina (Wtur), tomada del modelo de transmisión mecánico y la tercera entrada es el ángulo de paso (β), que se obtiene sistema de control.

4.3.3 Modelo de transmisión mecánica.

En la simplificación del sistema de transmisión mecánica para la implementación del modelo, se considera solo las partes de la turbina que puedan transmitir grandes oscilaciones a la potencia eléctrica de salida. El modelo con el que se trabajo en la simulación fue el de dos masas rotando con un coeficiente de rigidez y amortiguamiento en el eje de baja velocidad junto con relación de la caja de multiplicadora .El motivo de considerar este modelo como un abstracción del sistema de transmisión mecánica es que el eje principal soporta el torque aerodinámico producido por la fuerza del viento y la inercia del rotor de la turbina que en comparación con la inercia del generador eléctrico y la reducción del torque debido a la acción de la caja multiplicadora hace que las deformaciones del eje de alta velocidad sean inapreciables.

[22] HEIER Siegfried, Grid Integration of Wind Energy Conversion Systems, 1998.

El modelo tiene como salidas el la velocidad angular del rotor (Wtur) y la velocidad angular del generador (Wgen); como entradas tiene el torque aerodinámico (Tae) obtenido del modelo aerodinámico, y el torque del generador (Tgen) proveniente de del modelo eléctrico.

4.3.4 Modelo Eléctrico.

El modelo utilizado en la simulación maneja dos especificaciones de los generadores de inducción. La primera de ellas consiste el desarrollo sobre el marco de referencia del sistema dq0 que permite simplificar el modelo matemático basado en la teoría propuesta por RH Park, y la segunda es la consideración de los efectos causados en el rotor debido al deslizamiento cuando se esta modelando una turbina de gran potencia, que para el caso del prototipo a simular es generador de 2MW. Estos modelos fueron implementados por el Laboratorio Nacional de RISØ.[23]

[23] HANSEN Anca Daniela, SØRENSEN Poul, BLAABJERG Frede. Wind Turbine Blockset in Matlab/Simulink ,General Overview and Description of the Models, Florin Iov, , Aalborg University March 2004, p. 33, 41

5. PRUEBAS Y ANÁLISIS DE RESULTADOS

Los módulos implementados se validaron mediante los siguientes tipos de pruebas: i) Pruebas del módulo de comunicación con *sockets,* ii) pruebas del módulo de comunicación - controlador y iii) pruebas del prototipo final

5.1 PRUEBAS DEL MÓDULO DE COMUNICACIÓN (*SOCKETS*)

Al realizar la programación de los *sockets*, se configuran las interfaces de red de cada uno de los equipos. Dentro de esta configuración se establece la dirección IP para identificar el equipo donde se encuentra el servidor junto con uno de sus puertos libres.

La dirección IP del equipo PC1 (Servidor) es 192.168.46.62, mientras que la asignada a la ECB_AT91 es 192.168.46.102. El puerto elegido para conectar el servidor es el 6290. Este puerto puede escogerse dentro de los puertos disponibles en la máquina, que van desde el 0 hasta el 65535, teniendo en cuenta que los puertos del 0 al 1024 están reservados para el sistema.

La ejecución del servidor se realiza mediante la siguiente instrucción,

```
[usuario]$ ./servidor 6290 mensaje
```

Archivo creado durante la compilación — Puerto — Mensaje

Mientras que el cliente se ejecuta así:

```
[usuario]$ ./cliente 192.168.46.62 6290 mensaje
```

Archivo creado durante la compilación — Dirección IP Servidor — Puerto Servidor — Mensaje

Esta prueba retorna a la pantalla de cada equipo el mensaje recibido y enviado.

5.1.1 Prueba 1.

La prueba inicial se realizó sobre dos equipos, en los que se ejecutan el programa básico del servidor y el del cliente. Estos programas realizan el envío y transmisión de un mensaje desde cada punto de conexión. Como argumentos de entrada para el servidor se tiene el número del puerto y una palabra que será el mensaje a enviar, mientras que sobre el cliente son la dirección IP del servidor, el puerto y un mensaje.

Tras varias ejecuciones, se comprobó que el módulo implementado, realiza a cabalidad las operaciones de intercambio de información, puesto que se observa que los datos de llegada en cada punto de conexión son replicas de los datos enviados por el emisor.

5.1.2 Prueba 2.

Posteriormente, se procede a modificar el código básico implementado para el cliente, agregando un lazo *for*. Dicha modificación permite realizar la operación de envío y recepción de datos de manera repetitiva y ver el efecto de saturar al servidor con múltiples conexiones consecutivas. Se encontró que el servidor tarda un tiempo en cerrar su conexión y liberar al *socket*, por ello al recibir una nueva petición, se presenta el error "*Address already in use*", indicando que el par de datos (Dirección IP, puerto) que se está intentando asociar a este *socket*, ya se está utilizando.

Este inconveniente se soluciona mediante la configuración avanzada de *sockets,* realizando una petición al sistema operativo, para liberar al *socket* inmediatamente mediante la opción SO_REUSEADDR.

5.2 PRUEBAS MODULO DE COMUNICACIÓN – CONTROLADOR

En esta prueba los valores para la identificación del servidor (Dirección IP y puerto) al cual se conecta el cliente, se fijan dentro del código de implementación. Además, los datos intercambiados por los socket son extraídos de archivos de texto, creados por el controlador y el mecanismo de adaptación.

Para esta prueba el programa del socket cliente es llamado dentro de la ejecución del controlador, para realizar el envío de la señal de salida del proceso y la recepción de los nuevos parámetros. Estos programas se encuentran implementados en la tarjeta de desarrollo ECB_AT91. De la misma forma, el mecanismo de adaptación es llamado dentro de la ejecución del servidor, para obtener los nuevos parámetros que serán enviados al cliente.

Al igual que en las pruebas del modulo de comunicación, la condición inicial indispensable para garantizar el correcto envío y recepción de los datos, debe ser el estado alerta del socket servidor. Una vez logrado este estado se realiza la ejecución del controlador sobre la ECB_AT91 desde el PC1, mediante el protocolo SSH, que permite acceder remotamente al sistema de la tarjeta de desarrollo.

El controlador al iniciar su ejecución requiere el ingreso de algunos datos como son los parámetros iniciales, el tiempo de muestreo en microsegundos, el valor de la señal de referencia y el tipo de controlador a utilizar. Esta ejecución se encuentra enmarcada dentro de un lazo infinito, ya que debe ser un control permanente y puede ser detenido desde la estación remota donde está el mecanismo de adaptación (PC1), en caso de requerirlo.

Durante el desarrollo de esta prueba, se obtuvieron resultados satisfactorios, dado que los valores enviados por el socket servidor son recibidos de manera correcta en el cliente y usados en el controlador para calcular la señal que irá al proceso en el PC2.

Otro aspecto relevante al realizar esta prueba, fue la continuidad de la ejecución del controlador, aún cuando el servidor sale de trabajo y a su vez el mecanismo de adaptación.

CONCLUSIONES

- La implementación de la comunicación mediante el API sockets sobre la pila de protocolos TCP/IP, se enfoca en el manejo de la capa de transporte, lo que facilita la programación del esquema cliente-servidor y garantiza al usuario una conexión bidireccional fiable. Esta característica permite que los datos intercambiados, en el sistema de control adaptativo donde se implementa este tipo de comunicación, son replicas exactas del emisor.

- Implementar una conexión bajo el esquema cliente-servidor, sobre la estación de trabajo donde se realizan los cálculos de adaptación, y el dispositivo sobre el cual se programa el controlador, logra un desacople entre ellos que facilita la configuración del algoritmo de control de manera remota.

- El ajuste de ganancias del controlador PID digital diseñado, se realiza mediante cambios en línea, logrando de esta forma un sistema de control flexible y libre de interrupciones.

- Con el propósito de realizar un esquema de control, que sea útil para diversas plantas y procesos, el programa del controlador presenta argumentos de entrada, que deben ser introducidos por el usuario como: parámetros iniciales, tiempo de muestreo y la posibilidad de elegir entre un controlador PID, PI o PD.

BIBLIOGRAFÍA

[1] BARAJAS, Leandro. BARÓN, María del Pilar. Sistemas de control Integrado Neurofuzzy. 1998. Santafé de Bogotá.

[2] GARRIDO, Santiago. Identificación, estimación y control de sistemas no lineales mediante RGO. 1999.

[3] A.S, Erbay, An Overview on PID Control, RTP Corp., 2000

[4] AMÉSTEGUI M. Mauricio. Apuntes de control PID, La Paz 2001. Universidad Mayor de San Andrés.

[5] PIEDRAHITA M., Ramón. Implementación programada de reguladores, Zaragoza. Escuela Universitaria de Ingeniería Técnica Industrial, Universidad de Zaragoza.

[6] SALAZAR. M. Control Adaptativo: Una alternativa de control automático, 1994. p.4.

[7] RODRÍGUEZ, Francisco. LÓPEZ, Manuel. Control adaptativo y robusto, 1996. Universidad de Sevilla, Pág. 64

[8] DONAHOO, Michael. CALVERT, Kennet. TCP/IP Sockets en C, Practical Guide for Programmers. 2001

[9] ÅSTRÖM, Karl J. WITTERNMARK, Björn. Computer-Controlled Systems, Theory and Desing. p. 306-308

[10] USCÁTEGUI, Omar. PRADA, Darío. Modelado y simulación del sistema mecánico de un generador eólico de eje horizontal con tres palas, UIS. 2007 p.101.

[11] SØRENSEN, P., HANSEN, A.D., ROSAS, P.A.C. Wind models for simulation of power fluctuations from wind farms, Journal of Wind Engineering, 90, 2002, p.1381-1402

[12] HEIER Siegfried, Grid Integration of Wind Energy Conversion Systems, 1998.

[13] HANSEN Anca Daniela, SØRENSEN Poul, BLAABJERG Frede. Wind Turbine Blockset in Matlab/Simulink ,General Overview and Description of the Models, Florin Iov, , Aalborg University March 2004, p 33, 41

ANEXOS

Anexo A

REFERENCIA API DE *SOCKETS*

Estructura de datos

- **sockaddr:** Estructura genérica de direcciones (No específica familia de direcciones

```
#include <sys/socket.h>          /*Archivo genérico de cabecera de
                                 sockets*/

struct sockaddr
{
unsigned short sa_family;        /*Familia de
direcciones(ej.AF_INET)/
char sa_data[14];                /* Protocolo específico de dirección
*/
}
```

- **sockaddr_in:** Estructura de direcciones del protocolo de Internet.

```
#include <netinet/in.h>          /* Especificaciones de sockets del
                                 protocolo de Internet*/

struct in_addr
{
unsigned long s_addr;             /* Dirección IP (32 bits) */
}

struct sockaddr_in
{
unsigned short sin_family;       /* Protocolo de Internet (AF_INET)
*/
unsigned short sin_port;         /* Puerto (16 bits) */
struct iruaddr sin_addr;         /* Dirección IP (32 bits) */
}
```

Todos los valores de esta estructura deben estar en formato “network byte order”

La mayoría de las llamadas mencionadas a continuación retornan 0 si no ocurre ningún error en la ejecución de lo contrario retorna -1. Si el valor de retorno es diferente, se menciona al final de la explicación.

Configuración de *Sockets*

- ***socket*():** Crea un *socket* TCP o UDP, el cual puede ser usado como un punto final de conexión para enviar y recibir datos usando el protocolo espeficado. Se especifica TCP o UDP con tipo de *socket*/protocolo así SOCK_STREAM/IPPROTO_TCP y SOCK_DGRAM/IPPROTO_UDP, respectivamente.

```
#include <sys/types.h>
#include <sys/socket.h>

int socket(int protocolFamily, int type, int protocol)
```

protocolFamily	Dominio de comunicación (Familia de protocolos)
type	Tipo de *socket* (SOCK_STREAM o SOCK_DGRAM)
protocol	Protocolo del *socket* (IPPROTO_TCP o IPPROTO_UDP) por lo general el dominio y tipo de *socket* solo admite un protocolo, por lo geneneral este campo se pone en 0

socket () retorna el descriptor del nuevo *socket* si no ocurre ningún error de lo contrario retorna -1.

- **bind():** Asocia el *socket* a la dirección local de Internet y al puerto. El número del puerto debe ser especificado. La llamada falla (error: EADDRINUSE) si el número de puerto especificado es el puerto local de otro *socket* y la opción de *socket* SO_REUSEADDR no ha sido establecida.

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int socket, struct sockaddr *localAddress, unsigned int
addressLength)
```

socket	Descriptor del *socket* creado (retornado por *socket*())

localAddress	Estructura de datos donde se especifica la dirección y puerto al que se asocia el *socket*.
addressLength	Longitud de la estructura de datos anterior.

- **getsockname():** Retorna la información local para un *socket* en la estructura sockaddr. Todos los campos de la estructura están en formato network byte order.

```
#include <sys/socket.h>

int getsockname(int socket, struct sockaddr *localAddress,unsigned int
*addressLength )
```

socket	Descriptor del *socket* creado (retornado por *socket*())
localAddress	Estructura de datos donde se especifica la dirección y puerto al que se asocia el *socket*.
addressLength	Variable de entrada o salida que contiene el número de bytes de la estructura sockaddr

Conexión de *sockets*

- **connect():** Establece una conexión entre el *socket* dado y el *socket* remoto asociado con la dirección externa. Para retornar con éxito, los campos de la IP y el puerto del *socket* local y remoto deben haber sido llenados. Si el *socket* no ha sido ligado previamente a un puerto local, se asigna uno aleatoriamente. Para *sockets* TPC, connect() retorna satisfactoriamente solo después de completar on éxito el lazo con la implementación remota TCP; esto implica la existencia de un canal confiable para este *socket*.

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int socket, sockaddr *foreignAddress, int addresstength)
```

socket Descriptor del *socket* creado (retornado por *socket*())
foreignAddress Estructura de datos donde se especifica la dirección y puerto con el que deseamos establecer la conexión.
addressLength Longitud de la estructura de datos anterior.

- **l i s t e n ( )** (Sólo *sockets* Stream/TCP): Indica que el *socket* está listo para aceptar conexiones entrantes. El *socket* debe estar ya asociado a un puerto local. Después de esta llamada, la conexión TCP entrante solicitada al puerto local determinado (y la dirección IP, si está establecida) se completará y pasará a cola, hasta que es pasada al programa mediante accept().

```
#include <sys/types.h>
#include <sys/socket.h>

int listen(int socket, int backlog)
```

socket Descriptor del *socket* creado (retornado por *socket*())
backlog Longitud máxima de la cola de conexiones pendientes, el valor máximo que s valido es 128 en Linux, aunque implementaciones como BSD, por lo general, lo limitan a 5.

- **accept()** (Sólo *sockets* Stream/TCP): Bloques de espera para las conexiones dirigidas a la dirección IP y el puerto al que el *socket* está ligado. (listen() debe haber sido dado en el *socket*) Cuando llega una conexión TCP y se completa el lazo de comunicación con éxito, un nuevo *socket* se devuelve. La dirección y puerto local y remoto del nuevo *socket* se han llenado con el número de puerto local del nuevo *socket*, y la dirección remota ha sido devuelto en la estructura sockaddr_in (en formato network byte order).

```
#include <sys/types.h>
#include <sys/socket.h>

int accept(int socket, struct sockaddr *clientAddress, int
*addressLength )
```

socket Descriptor del *socket* creado (retornado por *socket*())

clientAddress	Estructura de datos que contendrá la dirección y puerto del descriptor que se ha conectado a este *socket* después de la llamada.
AddressLength	Puntero al entero que contiene la longitud de la estructura de datos anterior.

accept () retorna el nuevo descriptor del *socket* conectado si no hay errores o -1 de lo contrario.

- **getpeername():** retorna la información remota para un *socket* en la estructuras sockaddr. Todos los campos deben estar en formato network byte order.

```
#include <sys/socket.h>

int getpeername(int socket, struct sockaddr *localAddress, unsigned int
*addressLength)
```

socket	Descriptor del *socket* creado (retornado por *socket*())
localAddress	Estructura sockaddr retorna la dirección remota
addressLength	Longitud de la estructura de datos anterior.

Comunicación

- **send():** Envía los bytes contenidos en el buffer sobre el *socket*. El *socket* debe estar en estado conectado. Cuando la llamada retorna, los datos pasan a estar en cola para la transmisión sobre la conexión. La semántica depende del tipo de *socket*. Para una *socket* stream, cuando los datos se transmiten, la conexión se cierra normalmente. Para un *socket* datagram, no hay garantías de entrega. Sin embargo, los datos de un solo send() nunca serán divididos en múltiples llamadas recv(). El valor de retorno indica que el número de bytes transmitidos en realidad. El argumento permite banderas de características de protocolos especiales.

```
#include <sys/types,h>
#include <sys/socket.h>

int send(int socket, const void *msg, unsigned int msgLength, int
flags, struct sockaddr *destAddr, int destAddrLen)
```

socket	Descriptor de *Socket* (debe estar en estado conectado)
msg	Puntero a los datos que serán transmitidos
msgLength	Número de datos que serán enviados
flags	Banderas de control (0 en la mayoría de los casos)

- **sendto():** Envía los bytes contenidos en el buffer sobre el *socket*. Usar sendto() con un *socket* TCP, requiere que esté en estado conectado. La semántica es la misma que para send().

```
#include <sys/types,h>
#include <sys/socket.h>

int sendto(int socket, const void *msg, unsigned int msgLength, int flags)
```

socket	Descriptor de *Socket* (debe estar en estado conectado)
msg	Puntero a los datos que serán transmitidos
msgLength	Número de datos que serán enviados
flags	Banderas de control (0 en la mayoría de los casos)
destAddr	Dirección de destino de los datos (Network byte order)
destAddrLen	Longitud de la estructura de datos destino

- **recv():** Copia un número específico de bytes, recibidos en el *socket*, en una posición específica. El *socket* correspondiente debe estar en estado conectado. Normalmente, la llamada bloquea hasta que por lo menos un byte sea devuelto o hasta que la conexión sea cerrada. El valor retornado indica el número de bytes copiados en el buffer iniciando desde el primer valor de la posición. La semántica depende del tipo de *socket*. Par un *socket* stream, lo bytes se entregan en el mismo orden como fueron transmitidos, sin omisiones. Para un *socket* datagram, cada recv() devuelve los datos de al menos un send(), y el orden no es necesariamente preservado. Si el buffer para recv()no es lo suficientemente grande para el próximo datagrama disponible, el datagrama se trunca en silencio con el tamaño del buffer.

```
#include <sys/types.h>
#include <sys/socket.h>

int recv(int socket, void *rcvBuffer, int bufferLength, int flags)
```

```
socket          Descriptor de socket (debe estar en estado
                conectado)
rcvBuffer       Buffer que contendrá los datos leídos
bufferLength    Longitud del buffer en bytes
flags           Opciones de recepción, por lo general es 0
```

- **recvfrom():** Copia un número específico de bytes, recibidos en el *socket*, en una posición específica. Usar recvfrom() con un *socket* TCP, requiere que esté en estado conectado. La semántica es la misma que para recv().

```
#include <sys/types.h>
#include <sys/socket.h>

int recvfrom(int socket, void *rcvBuffer, int bufferLength, int
flags, struct sockaddr *fromAddr, unsigned int *fromAddrLen))
```

```
socket          Descriptor de socket (debe estar en estado
                conectado)
rcvBuffer       Buffer que contendrá los datos leídos
bufferLength    Longitud del buffer en bytes
flags           Opciones de recepción, por lo general es 0
fromAddr        Dirección de datos del remitente
fromAddrLen     Tamaño de la estructura de la dirección del
                remitente
```

- **close():** Termina la comunicación de un *socket*. El *socket* es etiquetado para inhabilitar tanto el envío como la recepción.

```
#include <unistd. h>

int close(int socket)
```

```
socket Descriptor del socket  (debe estar en estado conectado)
```

- **shutdown():** Termina la comunicación de un *socket*. El *socket* es etiquetado para inhabilitar el envío, la recepción o los dos, de acuerdo al segundo parámetro. Si este es 0, la recepción será deshabilitada, si es 1 no se puede enviar o si es 2, no se puede enviar ni recibir. El *socket* debe esta en estado conectado.

```
#include <sys/socket. h>
```

```
int shutdown(int socket, int how)

     socket          Descriptor de socket
     how             0 = no recibe, 1 = no envía, 2 = no envía ni
                     recibe
```

Control de *sockets*

- **getsockopt():** Recupera una opción de un *socket*. Las opciones de *socket* son usadas para modificar el comportamiento predeterminado de un *socket*.

```
#include <sys/typesh>
#include <sys/socketh>

int getsockopt(int socket, int level, int optName, void *optVal,
unsigned int * optLen)

     socket          Descriptor de socket
     level           Nivel de la opción
     optName         Nombre de la opción
     optVal          Puntero a un buffer para grabar el valor de
                     la opción
     optLen          Longitud del buffer (bytes)del valor de la
                     opción
```

- **setsockopt():** Ajusta las opciones de un *socket*. Las opciones de *socket* son usadas para modificar el comportamiento predeterminado de un *socket*.

```
#include <sys/types.h>
#include <sys/socket.h>

int setsockopt(int socket, int level, int optName, constvoid
*optVal, unsigned int optLen)

     socket          Descriptor de socket
     level           Nivel de la opción
     optName         Nombre de la opción
     optVal          Puntero a un buffer para grabar el valor de
                     la opción
     optLen          Longitud del buffer (bytes) del valor de la
                     opción
```

Conversión Binario/Cadena

- **inet_ntoa():** Convierte una dirección IP en notación binaria (network byte order) a su correspondiente formato en notación de puntos (p.e. “192.168.46.1)

```
#include <sys/socket.h>
#include <netinet/in,h>
#include <arpa/inet.h>

char *inet_ntoa(struct in_addr address)

     address          Estructura que contiene la dirección IP en
                      representación de 32 bits
```

inet_ntoa() retorna un puntero a una cadena. La cadena es asignada estáticamente a un buffer que cambia con cada llamada; el buffer debe ser copiado antes de las próximas llamadas.

- **inet_addr ():** convierte una dirección IP en notación de puntos a su correspondiente notación binaria (network byte order)

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

char *inet_addr(const char *address)

     address          Puntero a la cadena de caracteres que
                      contiene la dirección IP en notación de
                      puntos.
```

inet_addr ( ) retorna el “unsigned long”, representación binaria de la dirección si no hay errores o -1 de otra forma

- **htons( ), htonl( ), ntohs( ), n t o h l( ):** Convierte del formato *host byte order* a *network byte order* y viceversa

```
#include <netinet/in.h>

short int htons(short int hostShort)
```

```
long int htonl(long int hostLong)
short int ntohs(short int netShort)
long int ntohl(long int netLong)

     hostShort       Entero corto en formato host byte order
     hostLong        Entero largo en formato host byte order
     netShort        Entero corto en formato network byte order
     netLong         Entero largo en formato network byte order
```

htons(), htonl(), ntohs(), and ntohl() retornan el valor convertido. Estas funciones no tienen valores de fallo.

Información de Host y Servicios

- **gethostname():** retorna el nombre del *host* local en un buffer específico

```
int gethostname(char *hostName, unsigned int length)

     hostName        Buffer donde se almacena el nombre del host
     length          Longitud del buffer hostName

gethostname() retorna - 1 en caso de fallos; 0 de otra forma
```

- **gethostbyname():** dado el nombre de un host, retorna la estructura que contiene una descripción del host nombrado.

```
#include <netdb.h>

struct hostent *gethostbyname (const char * hostName)

     hostName        Nombre del host del cual se desea obtener
                     información
```

gethostbyname() retorna NULL si hay errores o la estructura si retorna con éxito.

```
struct hostent {
char *h_name;         /* Nombre oficial del host */
char **h_aliases;     /* Lista de los alias (cadenas) */
int h_addrtype;       /* Tipo de dirección de host (AF_INET) */
int h_length;         /* Tamaño de la dirección */
```

```
char **h_addr_list;  /* Lista de direcciones (binaria en formato
                     network byte order) */
}
```

- **gethostbyaddr():** Dada un a dirección IP, retorna la estructura que contiene la descripción del host de la dirección dada.

```
#include <netdb.h>
#include <sys/socket.h>

struct hostent *gethostbyaddr(const char *address, int
addressLength, int addressFamily)

     address         Dirección (binaria en formato network byte
                     order) del host del que se quiere obtener
                     información
     addressLength   Longitud de la dirección dada (bytes)
     addressFamily   Familia de la dirección dada (AF_INET)
```

Retorno igual a gethostbyname().

- **getservbyname ():** Da el nombre de un servicio (p.e, echo) y el protocolo implementado por este servicio. gtservbyname() retorna la estructura del servicio que contiene la estructura del servicio nombrado.

```
#include <netdb.h>

struct servent *getservbyname(const char *serviceName, const char
*protocol)

     serviceName     Nombre del servicio a obtener información
     protocol        Nombre del protocolo que utiliza el servicio
```

getservbyname() retorna NULL si hay error y una estructura de servicio si retorna con éxito;

```
struct servent {
char * s_name ;      /* Official service name */
char **s_aliases;    /* lista de nombres alternativos (cadenas) */
int s_port;          /* Número del servicio de puerto */
char *s_proto;       /* Protocolo de implementación (tcp o udp) */

};
```

- **g e t s e r v b y p o r t ( ):** Dado el puerto y el nombre del protocolo de servicio, retorna una estructura de servicio que contiene la descripción del servicio.

```
#include <netdb. h>
#include <sys/socket.h>

struct servent *getservbyport(int port, const char *protocol)

     port       Puerto (binario, formato network byte order) del
                el servicio a obtener información
     protocol   Nombre de la aplicación del protocolo de servicio
```

getservbyport() retorna NULL si hay errores y la estructura de servicio en caso de éxito.

Anexo B

COEFICIENTE DE TORQUE Cq (Velocidad específica vs. ángulo de paso)

	-90	-85	-80	-75	-70	-65	-60	-55	-50	-45	-40	-35	-30
0	-0.0066	-0.0139	-0.0179	-0.0179	-0.0179	-0.0177	-0.0171	-0.0159	-0.0148	-0.0136	-0.0124	-0.0111	-0.0094
1	-0.0305	-0.0307	-0.0307	-0.0307	-0.0305	-0.0298	-0.0287	-0.0273	-0.0256	-0.0235	-0.0211	-0.0182	-0.0151
2	-0.064	-0.0647	-0.0647	-0.0643	-0.0633	-0.0615	-0.0588	-0.0552	-0.0508	-0.0458	-0.0404	-0.0346	-0.0284
3	-0.1198	-0.1208	-0.1213	-0.12	-0.1172	-0.1124	-0.1062	-0.0988	-0.0905	-0.0814	-0.0716	-0.061	-0.0498
4	-0.1984	-0.201	-0.2006	-0.1975	-0.1908	-0.1821	-0.1713	-0.1593	-0.1456	-0.1309	-0.1145	-0.0973	-0.0782
5	-0.3011	-0.3045	-0.3021	-0.2958	-0.2847	-0.2709	-0.2549	-0.2366	-0.2163	-0.1937	-0.1694	-0.1424	-0.1136
6	-0.4285	-0.43	-0.4265	-0.4147	-0.3993	-0.3791	-0.3572	-0.3307	-0.3025	-0.2699	-0.2355	-0.1961	-0.1559
7	-0.5793	-0.5784	-0.5727	-0.555	-0.534	-0.507	-0.4771	-0.4418	-0.403	-0.3598	-0.3111	-0.2584	-0.2043
8	-0.7518	-0.7499	-0.74	-0.7171	-0.6888	-0.6551	-0.6145	-0.5704	-0.5176	-0.4628	-0.3955	-0.3279	-0.2581
9	-0.9466	-0.9452	-0.9285	-0.9014	-0.8639	-0.8234	-0.7699	-0.7137	-0.6462	-0.5716	-0.4884	-0.4031	-0.3178
10	-11.641	-11.641	-11.393	-11.078	-10.598	-10.084	-0.9428	-0.8688	-0.7878	-0.6913	-0.5909	-0.488	-0.3815
11	-14.051	-14.067	-13.726	-13.336	-12.762	-12.093	-11.316	-10.399	-0.9421	-0.8255	-0.7057	-0.5837	-0.4529
12	-16.693	-1.671	-16.283	-15.792	-1.512	-14.278	-13.371	-12.289	-11.122	-0.974	-0.8328	-0.6896	-0.5315
13	-19.569	-19.545	-19.064	-18.445	-17.652	-16.668	-1.561	-14.357	-12.976	-11.365	-0.9717	-0.8049	-0.6176
14	-22.681	-22.607	-22.065	-21.288	-20.387	-1.926	-18.041	-16.599	-14.983	-13.123	-11.223	-0.9298	-0.7113
15	-26.027	-25.896	-25.273	-24.347	-23.331	-22.058	-20.653	-19.008	-17.143	-15.019	-12.847	-10.645	-0.8122
16	-29.608	-2.941	-28.674	-27.632	-26.484	-25.055	-2.345	-21.591	-19.458	-1.705	-14.587	-12.087	-0.9206
17	-33.422	-33.137	-32.297	-3.114	-29.848	-28.248	-26.436	-24.346	-21.927	-19.216	-16.442	-1.362	-10.365
18	-3.747	-37.072	-36.144	-34.868	-3.342	-31.641	-29.606	-27.272	-24.549	-21.516	-18.413	-1.525	-1.159
19	-41.753	-4.123	-40.206	-38.814	-37.198	-35.231	-32.961	-30.368	-27.325	-23.953	-20.497	-16.971	-12.895
20	-46,263	-45,616	-44,503	-42,979	-41.186	-3.902	-3.65	-33.633	-30.258	-26.515	-22.698	-18.792	-14.266

	-25	-20	-15	-10	-5	0	5	10	15	20	25	30	35
0	-0.0073	-0.0052	-0.003	-0.0006	0.0019	0.0044	0.0069	0.0094	0.0118	0.0141	0.0163	0.0184	0.0203
1	-0.0117	-0.0083	-0.0047	-0.0011	0.0026	0.0063	0.0099	0.0135	0.0172	0.021	0.0248	0.0286	0.0311
2	-0.022	-0.0153	-0.0083	-0.0011	0.0062	0.014	0.0223	0.0307	0.0372	0.0383	0.0334	0.0238	0.0108
3	-0.0378	-0.0253	-0.0124	0.0011	0.0163	0.0319	0.0443	0.0485	0.0426	0.0298	0.0121	-0.0086	-0.0302
4	-0.0584	-0.038	-0.0165	0.0082	0.0333	0.0524	0.058	0.0498	0.0323	0.009	-0.0175	-0.0467	-0.0746
5	-0.084	-0.0536	-0.0179	0.0199	0.0507	0.0654	0.0601	0.0426	0.0164	0.0146	-0.0505	-0.0897	-0.1244
6	-0.1145	-0.0691	-0.0165	0.0314	0.0624	0.0685	0.0567	0.0323	-0.0006	-0.0402	-0.0882	-0.1377	-0.183
7	-0.1495	-0.0819	-0.0134	0.0378	0.062	0.0653	0.0509	0.0216	-0.0185	-0.0681	-0.1309	-0.1928	-0.2507
8	-0.1812	-0.0935	-0.0171	0.032	0.0521	0.0589	0.0449	0.0106	-0.0376	-0.099	-0.1793	-0.2551	-0.3287
9	-0.2144	-0.1089	-0.0283	0.0181	0.0405	0.0523	0.0388	-0.0006	-0.0577	-0.1341	-0.2332	-0.3248	-0.4172
10	-0.2547	-0.132	-0.0422	0.0025	0.0289	0.0457	0.0331	-0.0119	-0.0794	-0.1733	-0.2925	-0.4029	-0.5159
11	-0.301	-0.1602	-0.0598	-0.0126	0.0189	0.0395	0.0276	-0.0246	-0.1027	-0.2169	-0.3575	-0.4892	-0.6251
12	-0.3524	-0.1924	-0.0798	-0.0279	0.0096	0.0338	0.0222	-0.0372	-0.1274	-0.2649	-0.4284	-0.5839	-0.7448
13	-0.4099	-0.2283	-0.1024	-0.0432	0.0004	0.0284	0.0163	-0.0505	-0.1538	-0.3172	-0.5046	-0.6868	-0.875
14	-0.4724	-0.2671	-0.1263	-0.0583	-0.0084	0.0228	0.011	-0.0645	-0.1819	-0.3733	-0.5867	-0.7978	-10.157
15	-0.5399	-0.3088	-0.1515	-0.075	-0.0165	0.0178	0.0055	-0.0794	-0.2116	-0.4333	-0.6748	-0.9171	-11.669
16	-0.6128	-0.3535	-0.1792	-0.0919	-0.025	0.0127	0	-0.095	-0.2431	-0.4969	-0.7694	-10.448	-13.287
17	-0.6909	-0.4012	-0.2075	-0.1094	-0.0336	0.0079	-0.0059	-0.1109	-0.2766	-0.5655	-0.8696	-1.181	-15.012
18	-0.7736	-0.4519	-0.2378	-0.1274	-0.0424	0.0028	-0.0117	-0.128	-0.312	-0.6382	-0.9756	-13.254	-16.844
19	-0.8613	-0.5052	-0.2696	-0.1467	-0.0518	-0.0018	-0.0189	-0.1461	-0.3492	-0.715	-10.881	-14.776	-18.775
20	-0.9538	-0.5615	-0.303	-0.1669	-0.0608	-0.0067	-0.0257	-0.1648	-0.3882	-0.7959	-12.067	-1.639	-20.809

	40	45	50	55	60	60	65	70	75	80	85	90
0	0.022	0.0233	0.0245	0.0257	0.0267	0.0267	0.0275	0.0281	0.0273	0.0217	0.0104	-0.0018
1	0.0305	0.0258	0.0179	0.0072	-0.0047	-0.0047	-0.0157	-0.0245	-0.0308	-0.0353	-0.0376	-0.0386
2	-0.0046	-0.0198	-0.0331	-0.0438	-0.0525	-0.0525	-0.059	-0.0634	-0.0668	-0.0695	-0.0717	-0.0736
3	-0.0502	-0.0667	-0.081	-0.0926	-0.1011	-0.1011	-0.1085	-0.1148	-0.1208	-0.1267	-0.1299	-0.1305
4	-0.0986	-0.1205	-0.1386	-0.1526	-0.1649	-0.1649	-0.1764	-0.1878	-0.199	-0.206	-0.208	-0.2085
5	-0.1567	-0.1861	-0.2088	-0.2282	-0.2466	-0.2466	-0.2651	-0.2834	-0.2994	-0.3049	-0.3074	-0.309
6	-0.2257	-0.2638	-0.2931	-0.3203	-0.3475	-0.3475	-0.3746	-0.4016	-0.419	-0.4249	-0.4288	-0.433
7	-0.3073	-0.3538	-0.3922	-0.4298	-0.4674	-0.4674	-0.5049	-0.542	-0.5587	-0.566	-0.573	-0.5802
8	-0.4012	-0.4572	-0.5069	-0.5566	-0.6062	-0.6062	-0.656	-0.703	-0.7188	-0.7291	-0.7397	-0.7507
9	-0.5065	-0.5737	-0.6371	-0.7006	-0.7641	-0.7641	-0.828	-0.8828	-0.8996	-0.9143	-0.929	-0.9447
10	-0.623	-0.704	-0.783	-0.8617	-0.941	-0.941	-10.207	-10.822	-11.019	-11.211	-1.141	-1.162
11	-0.751	-0.8481	-0.9441	-10.402	-11.369	-11.369	-12.342	-13.014	-13.255	-1.35	-13.756	-14.027
12	-0.8909	-1.006	-11.207	-12.359	-13.519	-13.519	-14.684	-15.404	-15.703	-16.008	-16.329	-16.668
13	-10.426	-11.778	-13.129	-14.488	-15.858	-15.858	-17.233	-1.8	-18.363	-18.736	-19.128	-19.542
14	-12.058	-13.631	-15.205	-1.679	-18.386	-18.386	-19.989	-20.804	-21.236	-21.683	-22.153	-22.649
15	-13.812	-15.623	-17.437	-19.264	-21.103	-21.103	-22.949	-23.814	-24.321	-24.849	-25.405	-2.599
16	-15.687	-17.751	-19.823	-2.191	-24.011	-24.011	-26.115	-27.032	-2.762	-28.235	-28.882	-29.566
17	-17.682	-20.018	-22.364	-24.728	-27.108	-27.108	-29.487	-30.456	-31.131	-31.839	-32.584	-33.374
18	-19.798	-22.422	-25.061	-27.717	-30.394	-30.394	-33.067	-34.085	-34.855	-35.663	-36.513	-37.416
19	-22.036	-24.966	-27.912	-30.879	-33.869	-33.869	-36.848	-37.921	-38.792	-39.707	-40.669	-41.692
20	-24.393	-27.647	-30.921	-34.213	-37.534	-37.534	-40.834	-41.963	-42.941	-43.968	-45.049	-46.201

Anexo C

APROXIMACIÓN DIGITAL DE CONTROLADORES CONTINUOS

Las funciones de transferencia en tiempo continuo pueden ser representadas mediante aproximaciones numéricas que hacen posible la implementación de controladores digitales. Dentro de estas aproximación se encuentran algunos métodos tales como, Diferencia en adelanto y Atraso, método de Tustin también conocido como trapezoidal o transformación bilineal, equivalente rampa entre otros.

Las variables z y s están relacionadas por $z = e^{sh}$, por lo tanto las aproximaciones correspondientes a la expansión en series de los métodos más utilizados están dadas por:

$z = e^{sh} \approx 1 + sh$ (Método de Euler)

$z = e^{sh} \approx \frac{1}{1 - sh}$ (Método de Diferencias hacia Atrás)

$z = e^{sh} \approx \frac{1 + sh/2}{1 - sh/2}$ (Método Trapezoidal)

Donde h corresponde al periodo de Muestreo

Usando los métodos de aproximación anteriores, la función de transferencia H(z) se obtiene simplemente reemplazando el argumento s en G(s) por s', donde

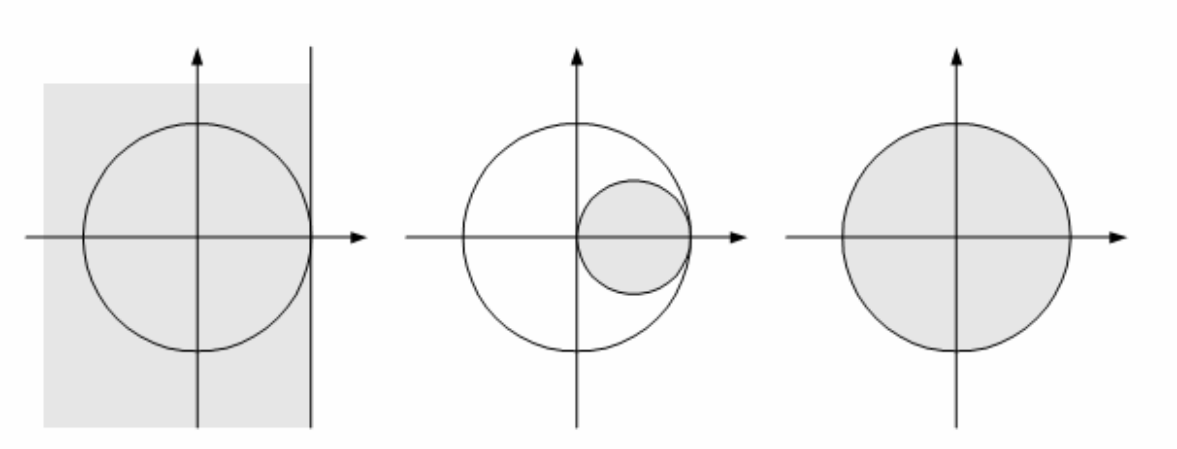
$s' = \frac{z - 1}{h}$ (Método de Euler)

$s' = \frac{z - 1}{zh}$ (Método de Diferencias hacia Atrás)

$s' = \frac{2}{h} \frac{z - 1}{z + 1}$ (Aproximación de Tustin)

De aquí: $H(z) = G(s')$

Los métodos son muy fáciles de aplicar aún en cálculos a mano. La Figura C1. muestra cómo se mapea la región de estabilidad $\mathrm{Re}\, s < 0$ del plano s al plano z de cada una de las aproximaciones descritas anteriormente.



Diferencias hacia delante | Diferencias hacia Atrás | Tustin

Fuente: Autoras, basadas de la referencia [6]

Figura C1. Mapeo de la región de estabilidad del plano s sobre el plano z

Anexo D

CÓDIGOS DE PROGRAMACIÓN

CLIENTE

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <errno.h>
/*********************************************************/
/* funcion MAIN */
/*********************************************************/
main ()
{
int s,t;
struct sockaddr_in bs,des;
char param[255];
char ysal[255];
extern int errno;

// Creamos el socket
s = socket(AF_INET,SOCK_STREAM,0);
      if (s!=-1) {
      bs.sin_family = AF_INET;
      bs.sin_port = htons(6290);
      bs.sin_addr.s_addr = INADDR_ANY; //Asigna una IP de la mÎ±quina

            //Asigna un nombre local al socket
            if(bind(s,(struct sockaddr*)&bs, sizeof(bs))!= -1)
            {
                  des.sin_family = AF_INET;
                  des.sin_addr.s_addr =htonl (0xC0A82E3E);
                  des.sin_port = htons(6290);

                  //Establece la conexion con la maquina remota
                  connect(s,(struct sockaddr*)&des,sizeof(des));

                  //Envia mensaje
                  FILE *y1;
                  y1=fopen("/home/proyectodi/y.txt","r");
                  fscanf(y1,"%s",ysal);
                  fclose(y1);
                  if(send(s,ysal,strlen(ysal),0)==-1){
                  perror("Error en el envio");}
                  else{
```

```
                printf("->Enviando: %s\n",ysal);}

                //Recibe la respuesta
                if(recv(s,param, sizeof(param) ,0) == -1){
                      perror("Error en recvfromresp");}
                else {
                      printf("<-Recibiendo: %s\n",param);
                      FILE *p1;
                      p1=fopen("/home/proyectodi/param.txt","w");
                      fclose(p1);}

                //Se cierra la conexion (socket)
                close(s);
        }
            else {
            printf("ERROR al nombrar el socket: %d\n",errno);
            close(s);
            }
      }
      else {
        printf("ERROR: El socket no se ha creado correctamente!\n");
    }
}
```

SERVIDOR

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <fcntl.h>
#define MAX_CONN 10 //N° maximo de conexiones en espera

/*******************************************************/
/* función MAIN */
/* Parametro = Puerto local
/* */
/*******************************************************/
main(int argc, char *argv[])
{
    int s,s_aux;
    struct sockaddr_in bs,in,out_s;
    char ysal[255];
    char param[255];
    int on=1;
    if (argc == 2) {
        // Creamos el socket
        s = socket(AF_INET,SOCK_STREAM,0);
        if (s != -1) {
```

```
    bs.sin_family = AF_INET;
    bs.sin_port = htons(atoi(argv[1]));
    bs.sin_addr.s_addr = htonl(INADDR_ANY);
    setsockopt(s,SOL_SOCKET,SO_REUSEADDR,&on,sizeof(on));

    //Asigna un nombre local al socket
    if( bind(s,(struct sockaddr*)&bs, siparameof(bs)) != -1) {
    printf("\n\aServidor ACTIVO escuchando en el puerto:
    %s\n",argv[1]);

        //Espera al establecimiento de alguna conexión
        listen(s,MAX_CONN);

        //Permite atender a múltiples usuarios
        while (1)
        {
            //Establece una conexión
            if((s_aux=accept(s,(struct sockaddr*)&in,&sd))==-1){
                perror("Error en accept");
                //continue;
            }
            printf("\nConexion aceptada\n");
            if(fork()==0){//Proceso hijo

                //Recibe variable de salida del clietne
                if (recv(s_aux,ysal,sizeof(ysal),0)==-1){
                    perror("Error en recv");}
                else{
                    printf("\n<-Recibido:%s\n",ysal);
                    FILE *y;
                    y=fopen("/home/proyecto/y.txt","w");
                    fprintf(y,"%s",ysal);
                    //fwrite(ysal,sizeof(char),strlen(ysal),y);
                    fclose(y);}

                //Envia parametros al cliente
                FILE *p;
                p=fopen("/home/proyecto/param.txt","r");
                fscanf(p,"%s",param);
                //fread(param,sizeof(char),slrlen(param),p);
                fclose(p);
                if (send(s_aux,param,strlen(param),0) == -1){
                    perror("Error en sendto");}
                else {
                    printf("->Enviando: %s\n",param);}
            close(s_aux);
            }
            exit(0);
        }
        close(s);
    } else {
        printf("ERROR al nombrar el socket\n");
    }
}
```

```
        else {
            printf("ERROR: El socket no se ha creado correctamente!\n");
        }
    } else {
        printf("\n\n\aEl número de parámetros es incorrecto\n\n");
    }
}
```