DISEÑO E IMPLEMENTACION DE UN CONTROLADOR PID DIGITAL EN UN MICROCONTROLADOR UTILIZANDOTECNICASDE PROTOTIPADO RAPIDO

JAIRO EDUARDO NORIEGA QUINTERO

UNIVERSIDAD PONTIFICIA BOLIVARIANA
ESCUELA DE INGENIERIAS
FACULTAD DE INGENIERIA ELECTRONICA
SECCIONAL BUCARAMANGA
2011

DISEÑO E IMPLEMENTACION DE UN CONTROLADOR PID DIGITAL EN UN MICROCONTROLADOR UTILIZANDOTECNICASDE PROTOTIPADO RAPIDO

JAIRO EDUARDO NORIEGA QUINTERO

PROYECTO DE GRADO

OMAR PINZON ARDILA
DIRECTOR DEL PROYECTO

UNIVERSIDAD PONTIFICIA BOLIVARIANA
ESCUELA DE INGENIERIAS
FACULTAD DE INGENIERIA ELECTRONICA
SECCIONAL BUCARAMANGA
2011

Nota de Aceptación

_____
_____
_____
_____
_____
_____

_____

Firma de Presidente del Jurado

_____

Firma del Jurado

_____

Firma del Jurado

AGRADECIMIENTOS

Al doctor Omar Pinzón por su apoyo, guía y supervisión en el desarrollo de este proyecto.

A los profesores de electrónica por sus conocimientos impartidos a lo largo del pregrado.

TABLA DE CONTENIDO

TABLA DE FIGURAS

TABLA DE ECUACIONES

RESUMEN GENERAL DE TRABAJO DE GRADO

TITULO: **DISEÑO E IMPLEMENTACION DE UN CONTROLADOR PID DIGITAL EN UN MICROCONTROLADOR UTILIZANDO TECNICAS DE PROTOTIPADO RÁPIDO**

AUTOR: **JAIRO NORIEGA QUINTERO**

FACULTAD: **FACULTAD DE INGENIERIA ELECTRONICA**

DIRECTOR(A): **OMAR PINZON ARDILA**

RESUMEN

Se diseñó e implementó un controlador PID para controlar la posición y velocidad de un motor de corriente directa utilizando técnicas de prototipado rápido en un microcontrolador. La identificación del modelo matemático del motor fue realizada con la ayuda de la herramienta System Identification Tool de MATLAB. El diseño del controlador fue realizado en la herramienta sisotool de MATLAB. El diseño del controlador se implementó en el sistema de desarrollo modular Tower System de Freescale, con los módulos serial (TWR-SER) y MCF51CN (TWR-MCF51CN), este último módulo cuenta con un microcontrolador Coldfire V1. La programación del microcontrolador se realizó en lenguaje C y se implementó también el sistema operativo en tiempo real MQX 3.5. El prototipado rápido se realizo gracias a la herramienta Real-Time Workshop Embedded Coder de Simulink que ajusta la simulación de Simulink y genera código en lenguaje C de manera embebida para diferentes tipos de hardware entre los cuales se encuentra la referencia Coldfire de la empresa Freescale. Las señales obtenidas tanto como por el taco generador del cual se obtenía la velocidad, como la señal obtenida del potenciómetro lineal del cual se obtiene la posición fueron acondicionadas mediante un arreglo de amplificadores operacionales para que estuvieran entre el rango permitido por el conversor analógico digital del microcontrolador. Se realizó una interfaz gráfica de usuario en MATLAB en la cual se pueden modificar tanto el valor de referencia del sistema como las diferentes constantes del controlador y visualizar la respuesta del sistema de control, para esto se implementó una comunicación serial con protocolo RS232.

PALABRAS CLAVE: Controlador PID digital, Prototipado rápido, sistemas operativos para microcontroladores.

ABSTRACT OF THE THESIS PROJECT

TITTLE: **DESIGN AND IMPLEMENTATION OF A DIGITAL PID CONTROLLER ON A MICROCONTROLLER USING FAST PROTOTYPING TECHNIQUES**

AUTHOR: **JAIRO NORIEGA QUINTERO**

FACULTY: **FACULTAD DE INGENIERIA ELECTRONICA**

DIRECTOR: **OMAR PINZÓN ARDILA**

ABSTRACT

A PID controller was designed and implemented on a microcontroller for controlling the position and speed of a direct current engine using fast prototyping techniques. The identification of the mathematic model was realized with the System Identification Tool of MATLAB. The design of the controller was made in the sisotool tool of MATLAB. The controller was implemented on the modular development system Tower System of Freescale, with the Serial, and the MCF51CN modules, the MCF51CN module has a Coldfire V1 microcontroller. The programming of the microcontroller was in C language and the MQX Real Time Operating System 3.5 was also implemented. The fast prototyping was possible thanks to the Real-Time Workshop Embedded Coder Tool of Simulink, that adjust the simulation and generates de code in C language for some embedded hardware that includes the Coldfire device among many others. The speed signal generated by the tacho generator and the position signal generated by a lineal potentiometer where fit up by an array of operational amplifiers so they fitted the voltage range of the analog to digital converter of the microcontroller. A graphical user interface was made in MATLAB to allow the user change the set point, the constants of the PID controller and visualize the response of the control system; to be able of doing this a serial communication was implemented with the rs232 protocol.

KEYWORDS: Digital PID controller, Fast Prototyping, microcontrollers, Real Time operating system for microcontrollers.

1. INTRODUCCION.

El controlador PID es un algoritmo de control por realimentación que tiene la capacidad de eliminar los errores en régimen permanente través de una acción integral y puede anticipar el futuro mediante una acción derivativa[1].

Aunque en un inicio el control PID se realizó de forma analógica, con la aparición de las computadores en los setentas se pudo llegar a implementar este tipo de control en forma digital. Hoy en día es posible implementar un control digital en diferentes dispositivos electrónicos, donde su principal característica consiste en realizar en un corto tiempo de muestreo su acción de control.

Actualmente en los diferentes trabajos de grado encontrados encontrados en biblioteca en la Universidad son muy pocos los que estan relacionados con el control digital, y especificamente en el campo de los controladores PID digitales. La implementacion de este controlador para su verificacion experimental se realizara con un microcontrolador comercial, lo cual reduce el costo del proyecto adicionalmente se utilizaran tecnicas de prototipado rapido.

El prototipado rápido se asocia fundamentalmente a la idea de desarrollar diferentes sistemas de control mediante la creación automática de prototipos de software o de hardware para su posterior evaluación. El desarrollo de la simulación o prototipado del sistema es de gran ayuda porque permite visualizar en línea el comportamiento del sistema y obtener información que facilita la identificación de problemas previos a una producción masiva del producto[2].

En este proyecto se describe el proceso empleado para la implementación del controlador PID digital y el algoritmo de programación del microcontrolador. Para la comprobación experimental de este controlador se realizara un control de posición y velocidad sobre un motor de corriente directa, finalmente se contrastan los resultados obtenidos experimentalmente con los resultados de simulación.

[1] ASTROM, K. y HAGGLUND, T. Advanced PID Control. Pittsburgh: ISA, 2006.
[2] FLORIA, A. Recuperado el 22 de Noviembre de 2009 de: http://www.sidar.org/recur/desdi/traduc/es/visitable/nuevos/Rápido.htm.

2. OBJETIVOS.

2.1 OBJETIVO GENERAL.

Diseñar e implementar un controlador PID digital en un microcontrolador utilizando técnicas de prototipado rápido.

2.2 OBJETIVOS ESPECIFICOS.

- Estudiar los diferentes algoritmos y técnicas para implementar un control PID digital.

- Instalar y comprender el modo de funcionamiento de la herramienta *Real-Time Workshop Embedded Coder de Mathworks*.

- Analizar las diferentes alternativas comerciales que mejor se adapten para la implementación del algoritmo de control en un microcontrolador

- Implementar un algoritmo de control PID digital en un microcontrolador utilizando el código generado por la herramienta *Real-Time Workshop Embedded Coder de Mathworks.*

- Realizar simulaciones de los diferentes algoritmos de control PID digital en *Simulink* y verificar su funcionamiento experimentalmente mediante el control de velocidad y posición de un motor de corriente directa.

3. MARCO TEORICO

3.1 SISTEMAS EMBEBIDOS:

Un sistema embebido, también conocido como sistema empotrado o incrustado, es un sistema en el cual se combina hardware y software diseñado para ejecutar una función o un grupo de funciones especificas, las cuales generalmente no pueden ser programadas por el usuario final, este concepto difiere radicalmente del de un computador personal que está diseñado y programado para realizar un gran número de tareas sobre las cuales el usuario final tiene un absoluto control[3]. Algunas características importantes de los sistemas embebidos son: un bajo consumo, un tamaño reducido y la maximización de los recursos (memorias internas o externas, módulos de entrada y salida, módulos de comunicación, etc.).

La parte principal de un sistema embebido se encarga de dar la capacidad de cómputo al sistema, la cual se denomina CPU (Unidad Central de Procesamiento). Esta CPU es programable y se programa en diferentes lenguajes tales como lenguaje ensamblador o lenguajes de alto nivel como por ejemplo el lenguaje C, C++ y en algunos casos BASIC dependiendo del tipo de núcleo. Esta CPU se implementa en un microcontrolador, microprocesador, DSP o FPGA, por nombrar algunos ejemplos.

Un sistema embebido se encarga de realizar tareas tan simples como visualizar una variable de velocidad, como es el caso del tacómetro en los vehículos automotor, hasta tareas más complejas que incluyan el control de diferentes variables, como es el caso de la temperatura en un horno o la velocidad de reproducción de un reproductor MP3. La palabra control se asocia con aplicaciones para sistemas embebidos, ya que generalmente el objetivo de estas aplicaciones es controlar una o varias variables del sistema bajo control.

La CPU en diferentes equipos electrónicos y electrodomésticos apareció antes de que lo hicieran los computadores personales y actualmente este tipo de sistemas están más integrados a la vida diaria más que cualquier otro tipo de circuito electrónico, de hecho un automóvil actualmente cuenta con más de 50 CPU y las lavadoras también cuentan con CPU que se encargan de supervisar los diferentes ciclos de lavado[4].

Hoy en día con la migración de las tecnologías analógicas por tecnologías digitales se ha extendido el uso de dispositivos electrónicos que integran

[3] HEATH, S. Embedded Systems Design. Oxford: Newnes, 1997.
[4] HEATH, S. Embedded Systems Design. Oxford: Newnes, 1997.

sistemas embebidos, como es el caso de las cámaras digitales, los DVD y los reproductores MP3 por mencionar algunos ejemplos.

Muchos sistemas embebidos requieren de sistemas de tiempo real. Un sistema de tiempo real debe responder, dentro de un intervalo restringido de tiempo, a eventos externos mediante la ejecución de la tarea asociada con cada evento. Los sistemas de tiempo real se pueden caracterizar como críticos y no-críticos. Si un sistema de tiempo real no-crítico incumple con las restricciones previamente establecidas de tiempo, simplemente se degrada el rendimiento del sistema. En contraste, si el sistema de tiempo real crítico no cumple con las restricciones de tiempo previamente establecidas, el sistema falla y este fallo puede traducirse en consecuencias catastróficas para el sistema de control.

Un sistema embebido complejo puede utilizar un sistema operativo como apoyo para la ejecución de sus programas, sobre todo cuando se requiere la ejecución simultánea de los mismos. Cuando se requiere un sistema operativo para un sistema embebido complejo, lo más probable es que se requiera de un sistema operativo de tiempo real (RTOS). Los RTOS se diseñan y optimizan para administrar restricciones críticas de tiempo, las cuales se asocian con eventos en aplicaciones de tiempo real. Por lo tanto, en una aplicación compleja de tiempo real es necesario utilizar un RTOS que soporte múltiples tareas, simplificando el desarrollo del software.

3.2 SISTEMA OPERATIVO DE TIEMPO REAL MQX 3.5.

El MQX es un sistema operativo de la casa *Freescale* Semiconductor que se adapta a sus plataformas para el desarrollo de sistemas embebidos *Coldfire* y *PowerPC.*

3.2.1 Sistema operativo de tiempo real: según la real academia de la lengua española un sistema operativo es el programa o conjunto de programas que efectúan la gestión de los procesos básicos de un sistema informático, el cual permite la ejecución normal del resto de las operaciones[5].

Las principales características de un sistema operativo son la gestión de los recursos de la maquina, la interfaz entre el hardware y el software y una biblioteca de funciones para las diferentes aplicaciones que este sistema operativo deba ejecutar.

[5] DICCIONARIO DE LA REAL ACADEMIA DE LA LENGUA ESPAÑOLA

Un sistema de tiempo real es un sistema digital que debe actualizar un planificador constantemente y asegurarse que las tareas se ejecuten en el mismo orden en el que se solicitaron, además la medida de sensores, ley y acción de control sobre mecanismos de regulación deben cumplir con los tiempos críticos de actuación[6].

Integrando los dos conceptos anteriores, los RTOS son sistemas operativos que gestionan los recursos de un dispositivo para que éste pueda cumplir con los tiempos críticos de actuación de las diferentes tareas para las que hayan sido programados, las principales características de un buen RTOS son: soporte de multitareas, soporte de prioridades entre sus diferentes tareas, comunicación entre tareas, cumplimiento de la restricciones, gestión de recursos y de interrupciones. Estos sistemas operativos en su mayoría se diseñan para microcontroladores o microprocesadores.

3.2.2 Planificador MQX. Los RTOS se utilizan para dar solución a proyectos relativamente grandes dividiéndolos en tareas, están tareas compiten por el acceso al procesador y por esto es necesario un planificador que se encarga de gestionar los diferentes recursos tales como: convertidor analógico-digital y módulos de comunicación por mencionar un par de ellos.

Al momento de crear una tarea es muy importante tener en cuenta su estado inicial y su pila de memoria, si el tamaño de la pila de memoria no es suficiente en el momento que ocurra un cambio de contexto se empezaran a perder datos. Las tareas pueden encontrarse en cuatro estados diferentes: activo, bloqueado, preparado, y terminado. Dependiendo del estado en el que se encuentre la tarea, el planificador decide que sucede con esta, aunque las tareas sólo tendrán uno de dos estados iníciales: preparado o bloqueado.

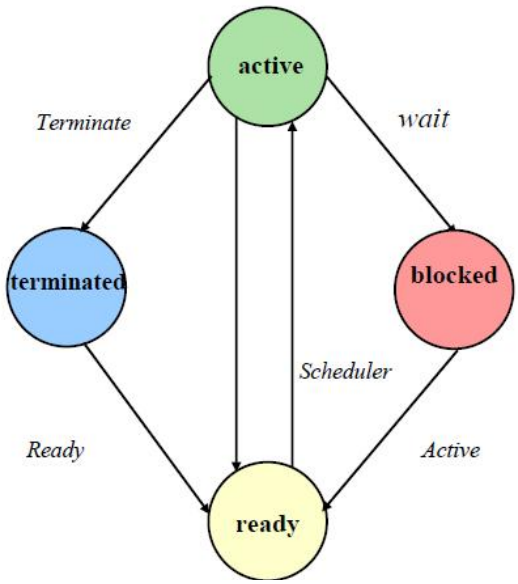
- *Estado activo:* La tarea se está ejecutando en el procesador.
- *Estado preparado:* La tarea se encuentra en cola de espera para ser ejecutada.
- *Estado bloqueado:* La tarea se encuentra bloqueada por el usuario, o porque esta tarea trata de acceder a un recurso que actualmente está usando otra tarea.
- *Estado terminado:* La tarea fue terminada por el usuario liberando los recursos asignados a esta.

Cualquier tarea en estado activo puede pasar a cualquiera de los otros tres estados, mientras que del estado preparado sólo puede pasar al estado

[6] PEREZ-CARPINTERO, J. MORERA, J. Conceptos de sistemas operativos. Madrid: Universidad Pontificia de Comillas, 2002.

activo; del estado bloqueado sólo puede pasar al estado preparado y del estado terminado sólo puede pasar al estado preparado si esta tarea se vuelve a crear. En la *figura 1* se observa el diagrama del flujo donde muestra los estados de las tareas existentes en el MQX.

Figura 1. Flujo estados de las tareas[7].



El planificador puede programarse en modo FIFO, *round robin* o realizar una planificación especifica. El planificador FIFO del MQX es preferente (del inglés *preemptive*), por lo tanto, el planificador ejecuta la tarea de mayor prioridad y con mayor tiempo de espera en la cola.

El planificador *round robin* asigna una ranura de tiempo para la ejecución de cada una de las tareas, cada vez que una tarea activa consume una ranura de tiempo, el planificador ejecuta la tarea de mayor prioridad y con mayor tiempo en la cola de espera. Cuando una tarea haya consumido su ranura de tiempo no entra en la cola de espera hasta que las demás tareas también lo hagan. De esta forma el planificador ejecuta las tareas de manera "simultanea".

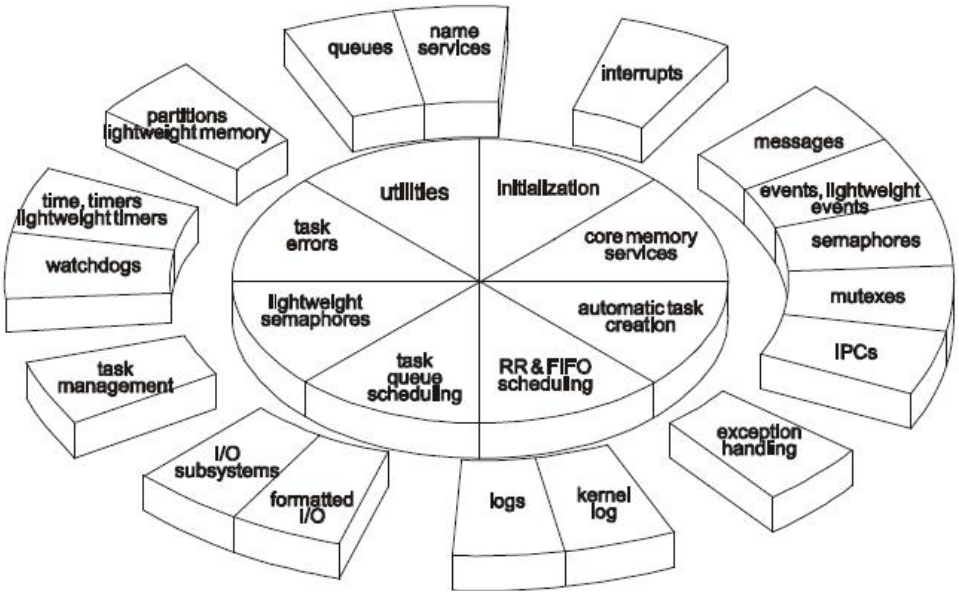
El planificador programado de manera específica utiliza mecanismos de sincronización más complejos, los cuales permiten utilizar varias colas de espera donde explícitamente se pueden planificar cada una de las tareas.

[7] RINCON, E. Programación de sistemas embebidos usando sistemas operativos de tiempo real [Diapositivas]. Bucaramanga, 2010.

Otro concepto muy importante del planificador es el cambio de contexto, cada vez que una tarea pasa de activa a otro estado excepto el terminado, ocurre un cambio de contexto, exigiendo guardar los registros del procesador, las variables y direcciones de la tarea en la pila memoria de la misma. El tamaño de esta pila se asigna en la configuración de cada tarea y el espacio lo guarda el MQX hasta que la tarea se termine, liberando todos los recursos asignados a esta tarea.

3.2.3 El núcleo del MQX 3.5. El MQX configura diferentes componentes tanto opcionales como no que son fundamentales para el correcto funcionamiento del sistema operativo. En la figura 2 se observan en el círculo central los componentes no opcionales de MQX y en el círculo exterior los componentes opcionales[8].

Figura 2. Componentes del sistema operativo MQX[9].



Además de la configuración del MQX, el sistema operativo cuenta con una característica que facilita la programación de algunos de los periféricos de los microcontroladores o microprocesadores como lo son el convertidor analógico digital o la comunicación serial. Sin embargo, hay otros periféricos que no cuentan con esta característica, tal es el caso del módulo de PWM. El añadir componentes opcionales al MQX, al igual que periféricos conlleva

[8]FREESCALE SEMICONDUCTOR. Freescale MQX Real-Time Operating system user's Guide. Denver: Freescale Semiconductor, 2010.

[9]FREESCALE SEMICONDUCTOR. Freescale MQX Real-Time Operating system user's Guide. Denver: Freescale Semiconductor, 2010.

a que el núcleo que genera el MQX ocupe más memoria interna del dispositivo.

3.3 PROTOTIPADO RAPIDO.

Se entiende por prototipo una representación no terminada de un producto final, sobre la cual se pueden ejercer diferentes juicios y determinar los errores o aciertos que se encuentran en las diferentes etapas del proceso de diseño. Un prototipo es útil desde la determinación de los requerimientos del sistema hasta el momento de la validación del mismo.

En la mayoría de los casos los prototipos se desechan y para la entrega del producto final se desarrolla uno nuevo, a lo anterior se le conoce como prototipado desechable, realizar un nuevo producto después de tener un prototipo bastante avanzado puede parecer ilógico sin embargo esto dio pie al concepto de prototipado rápido. Este tipo de prototipado permite completar el conjunto de requerimientos o especificaciones del sistema, modificando el prototipo hasta obtener el producto final[10].

Segun *Gordon y Bieman*[11] en 1995 destacaron que algunos de los beneficios de utilizar prototipado rápido son:

- Mejora la utilización del sistema.
- Mayor concordancia entre el sistema y las necesidades del usuario.
- Mejor calidad de diseño.
- Reducción en el esfuerzo de desarrollo.

3.4 REAL-TIME WORKSHOP EMBEDDED CODER RTWEC.

En la figura 3 un diagrama de secuencia para el diseño de un proyecto usando el RTWEC. La herramienta RTWEC permite la generación de un código en lenguaje C de los algoritmos que han sido modelados gráficamente en *Simulink* junto con las funciones de *Matlab* que son útiles para aplicaciones embebidas en tiempo real.

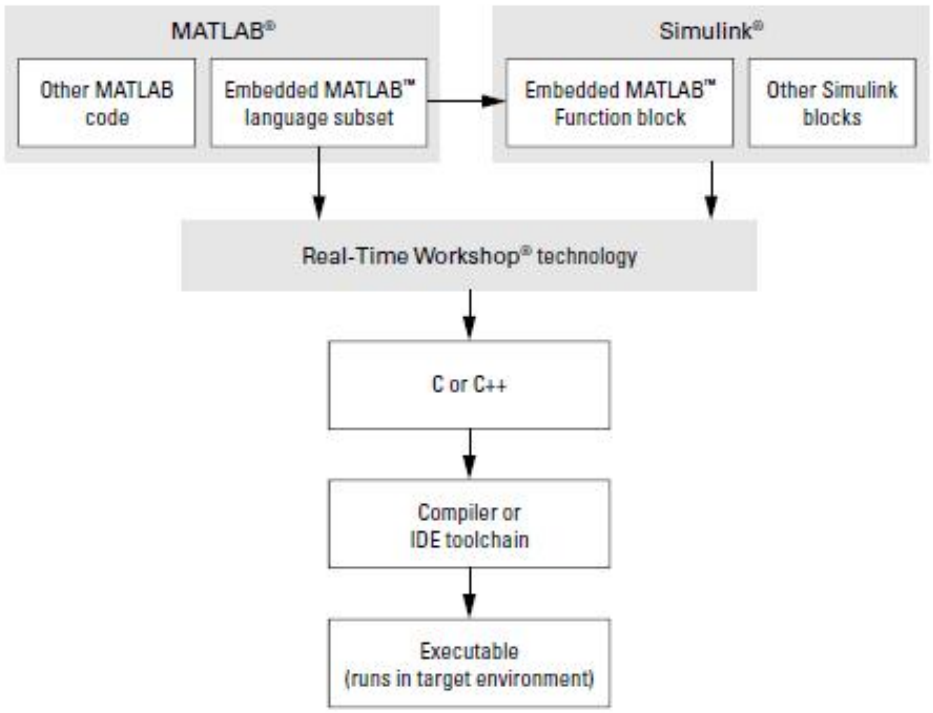
Simulink es un software que permite simular y analizar sistemas dinámicos, este software abre la posibilidad de observar lo que sucede con el sistema modelado si se cambian algunos de sus parámetros, esta característica es importante al tratar de realizar un prototipado rápido ya el código generado

[10] SOMMERVILLE, I. Ingeniería del Software. Madrid: Pearson Educación S.A, 2005.
[11] GORDON, V.S. Bieman, J.M. Rapid prototyping: Lessons learned. En: IEEE Software. 1995.
.

por la RTWEC puede ser evaluado facilmente en el mismo ambiente de simulación.

Con *Simulink* facilmente se pueden construir modelos desde cero o modificar modelos existentes para resolver las necesidades que se tengan. *Simulink* soporta el diseño de sistemas lineales y no lineales modelados en tiempo continuo, discreto, o una mezcla de los dos[12].

Figura 3. Diagrama de secuencia para el diseño de un proyecto usando el RTWEC[13].

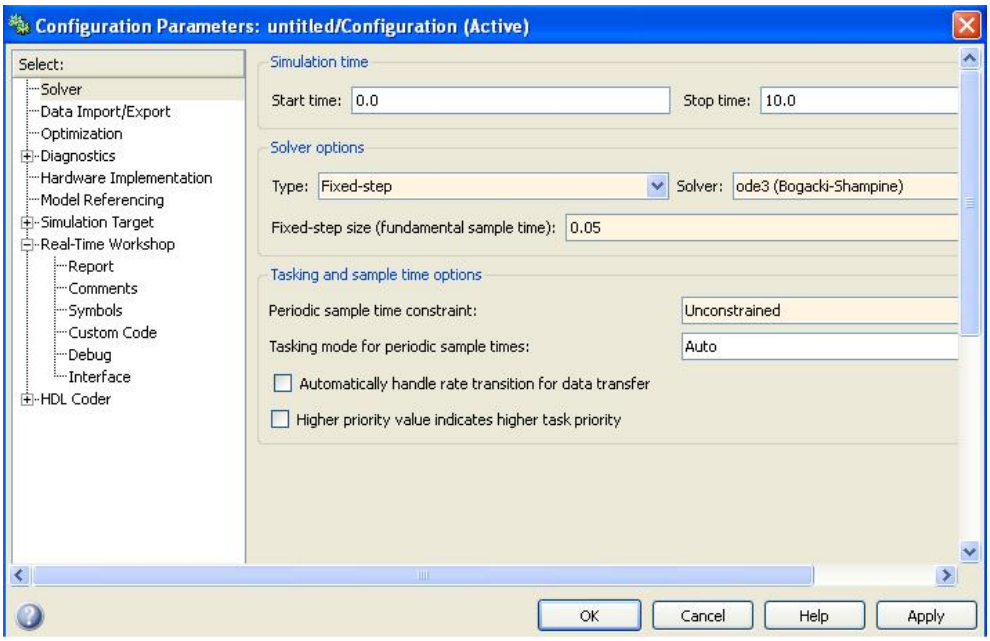


La herramienta *Real-Time Workshop* esta integrada a la ejecucion de *Matlab* y *Simulink*, ya que de hecho las simulaciones obtenidas en *Simulink* utilizan la tecnologia *Real-Time Workshop*. La RTWEC Permite ademas modificar la apariencia del codigo generado, optimizarlo para una tarea o ambiente específico e integrarlo a funciones o aplicaciones existentes[14].

[12] THE MATHWORKS INC. Real-Time Workshop Embedded Coder 5 Getting Started Guide. Natick: The Mathworks Inc, 2009.
[13]THE MATHWORKS INC. Real-Time Workshop Embedded Coder 5 Getting Started Guide. Natick: The Mathworks Inc, 2009.
[14] THE MATHWORKS INC. Real-Time Workshop Embedded Coder 5 Getting Started Guide. Natick: The Mathworks Inc, 2009.

3.4.1 Generación de código utilizando el RTWEC. En la figura 4 se muestra una ventana ejemplo de configuración de parámetros. Para generar el código es necesario configurar algunos parámetros para que el código generado se pueda implementar en el dispositivo electrónico deseado.

Figura 4. Ventana de Configuración de parámetros.



En la figura 4 se observa la ventana de configuración de parámetros en la cual se selecciona la opción *Solver*, el tipo de esta característica se cambia a un tipo de Paso fijo y se selecciona un tiempo de muestreo fundamental igual el tiempo de muestro que vaya se implemente en el microcontrolador.

En la opción *Hardware Implementation* se selecciona el fabricante del hardware y el tipo de dispositivo, en este proyecto se requiere seleccionar como fabricante a *Freescale* y como tipo de dispositivo el *Coldfire*. Adicionalmente, se debe cambiar el archivo *System-Target* a ert.tlc en la opción *Real-Time Workshop*.

3.4.2 Análisis código generado por el RTWEC. La herramienta RTWEC entrega una carpeta con varios archivos, la cantidad de archivos con extensión .C puede cambiar dependiendo las características del modelo, entre los archivos que genera se encuentra uno con el nombre del modelo y otro con el nombre ert_main.

El archivo ert_main contiene un código muy sencillo que llama una función de inicialización y *Mathworks* recomienda que la función que se ejecuta cada periodo de muestreo del sistema sea asociada a una interrupción.

4. HARDWARE UTILIZADO

4.1 SISTEMA DE DESARROLLO TOWER.

Para la implementación del controlador PID se utiliza el sistema de desarrollo TOWER de la familia Freescale Semiconductor. Este sistema de desarrollo es un sistema modular donde se utlizan fundamentalmente dos módulos. El primero de estos es el módulo MCF51CN128, el cual cuenta con un microcontrolador Colfire V1 de 32 bits embebido, que funciona como CPU del sistema de desarrollo. El segundo de estos es el módulo TWR SER. Para la comunicación entre módulos se utilizan los elevadores de Freescale, el módulo elevador primario se encarga de realizar la interconexion entre los diferentes módulos y el módulo elevador secundario no realiza ninguna función.

En la figura 5 se muestra sistema de desarrollo TWR-MCF51CN128. El módulo MCF51CN128 cuenta con diferentes periféricos, como lo son 3 acelerómetros, 4 LEDs entre otros. El módulo tambien cuenta con conexión USB que facilita la programación del microcontrolador embebido

Figura 5. Sistema de desarrollo TWR-MCF51CN128[15].

El segundo módulo TWR-SER es un módulo dedicado a las comunicaciones que cuenta con los diferentes conectores para implementar los siguientes protocolos de comunicación:

- RS232 y RS485.

[15] Recuperado el 10 de diciembre de 2010 de: http://sc.leadix.com/DEVmonkey/files/06-24-09%20Freescale%20Tower%20Fig1.jpg

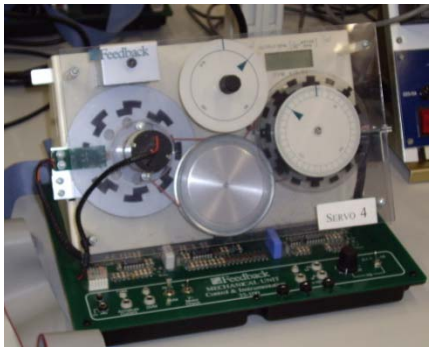
- Ethernet.
- CAN.
- USB.

El sistema de desarrollo TOWER cumple con los requisitos necesarios para la implementación de este proyecto porque el microcontrolador MCF51CN128 embebido cuenta con convertidor-analógico digital de resolución de 12 bits, soporta la comunicación ya sea Ethernet o con el protocolo RS232 y tiene un módulo que facilita la generación de señales PWM.

4.2 UNIDAD MECANICA DE SERVOMECANISMOS.

En la figura 6 se muestra la unidad mecánica de servomecanismos. La unidad mecánica de servomecanismos de *Feedback* es una unidad que permite realizar el control didáctico de un motor de corriente continua. Esta unidad tiene tanto sensores analógicos como sensores digitales para medir la velocidad y posición del motor.

Figura 6. Unidad mecánica de servomecanismos[16]



La unidad mecánica tiene un potenciómetro lineal que al estar alimentado entrega un voltaje en un rango de -10V a +10V dependiendo de la posición en la cual se encuentre el motor, esta señal se utiliza en este proyecto para realizar el control de posición.

Además del potenciómetro lineal, la unidad mecánica cuenta con un taco-generador que produce un voltaje proporcional a la velocidad del motor, si el motor gira en sentido de las manecillas del reloj este voltaje será positivo y si lo hace en sentido contrario será negativo. Después de realizar las pruebas de funcionamiento del módulo se obtiene un voltaje de salida que se encuentra en un rango entre -5.3V a +5.3V.

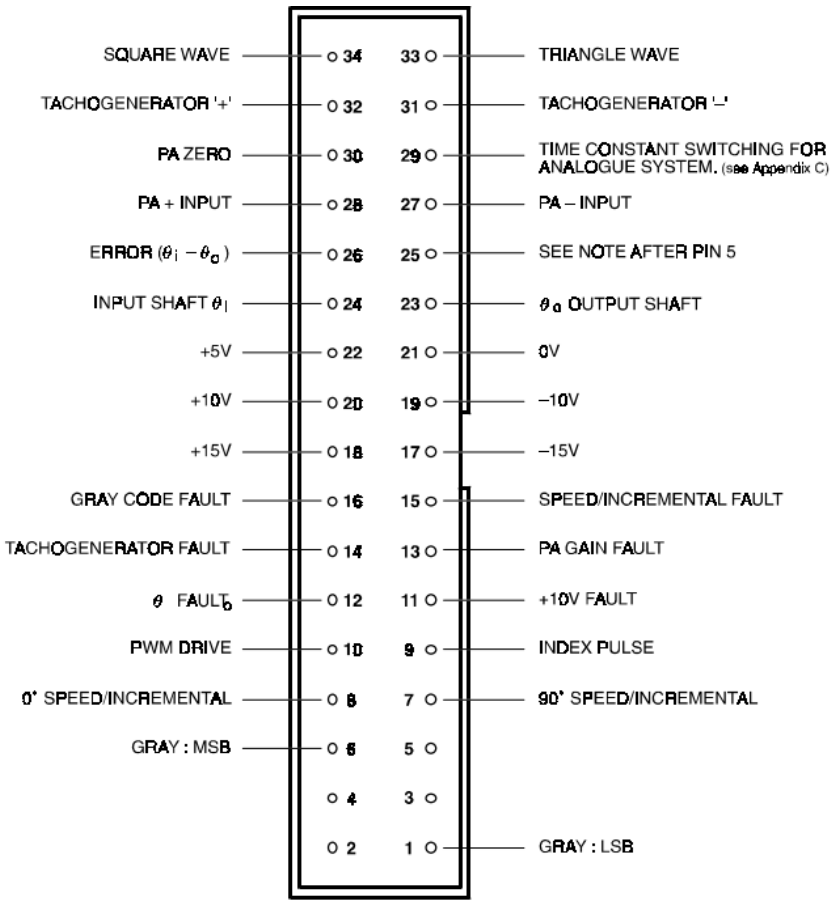
[16] Recuperado el 10 de diciembre de 2010 de: http://arvc.umh.es/ceaRsc/doc/feedback1.jpg

Para realizar el control de la velocidad y sentido de giro del motor el módulo cuenta con una entrada a un driver PWM que dependiendo del ancho de pulso que este detecte hará que el motor gire a una velocidad determinada ya sea en un sentido u otro. El módulo posee las siguientes características:

- 50% de ciclo de trabajo. Motor detenido.
- 0% de ciclo de trabajo. Velocidad máxima en sentido de las manecillas del reloj.
- 100% de ciclo de trabajo. Velocidad máxima en sentido contrario de las manecillas del reloj.

En la figura 7 se muestra el conector de 34 pines que dispone la unidad mecánica de servomecanismos para realizar la interconexión con el módulo de servomecanismos.

Figura 7. Conector de 34 pines del módulo de servomecanismos[17].



[17] FEEDBACK INSTRUMENTS LTD. Analogue Servo-Fundamentals Trainer. Sussex: Feedback instruments Ltd.

4.3 CIRCUITO ACONDICIONADOR DE SEÑAL.

El módulo de servomecanismos entrega señales que se encuentra por fuera del rango del convertidor analógico-digital del microcontrolador MCF51CN128, este convertidor recibe señales entre 0V y 3.3V. Por esto se diseña un circuito acondicionador de señal que adecua los voltajes tanto de la salida del taco-generador como del potenciómetro lineal para que estas pudieran ser medidas por el microcontrolador.

En la figura 8 se observa el esquema eléctrico del circuito acondicionador de señal. El circuito acondicionador de señal (CAS) se realizó mediante la división y posterior suma de voltajes utilizando amplificadores operacionales en las configuraciones inversor y sumador inversor. Para la alimentación de estos operacionales el conector de 34 pines provee voltajes de +15V, +10V, +5V, 0V, -10V y -15V.

La ecuación del voltaje de salida para la entrada del potenciómetro es:

$$Vout = \frac{1}{6}Vin + \frac{5}{3}V$$

Ecuación 1.

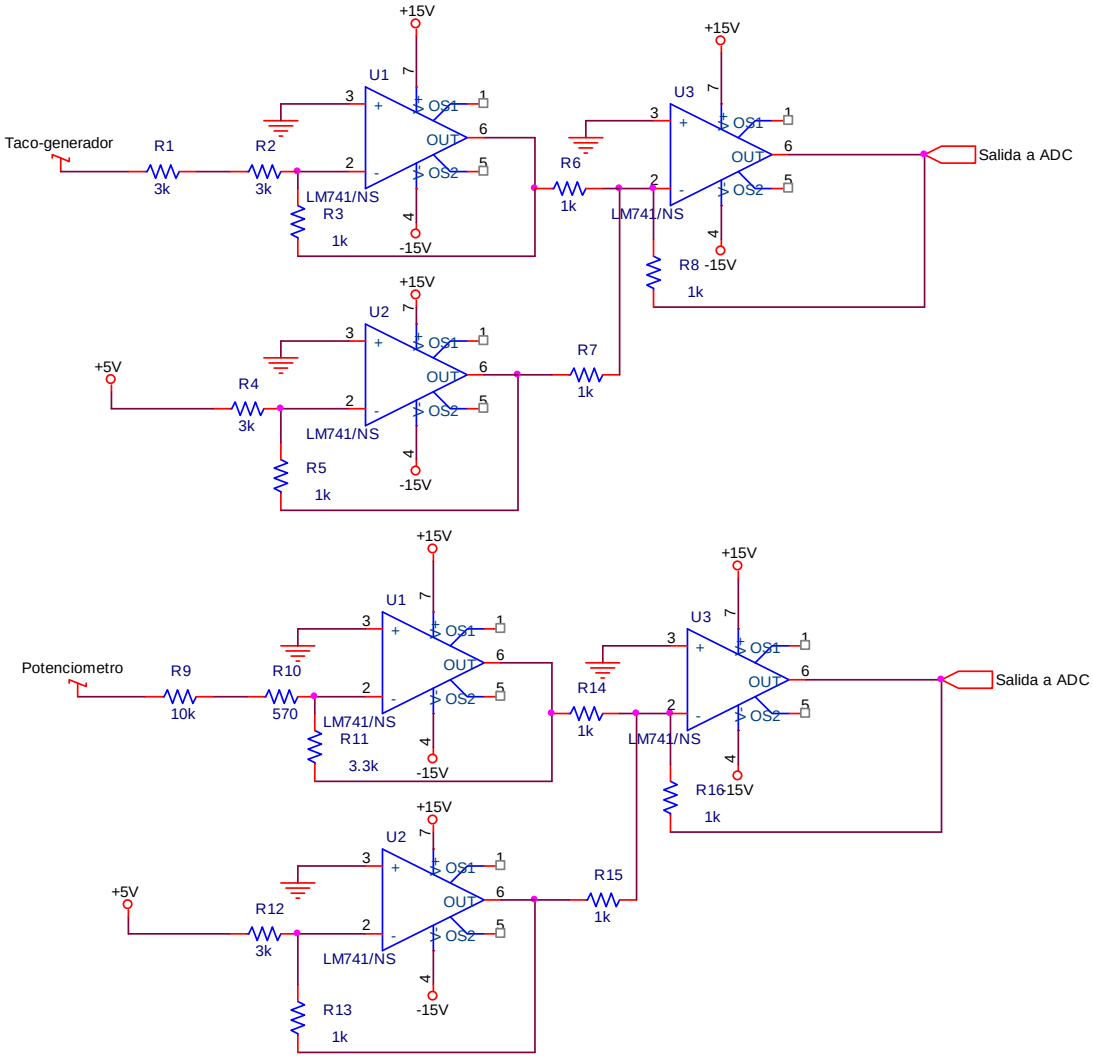
Al reemplazar en la ecuación anterior los valores de -10V y +10V se obtendrá una salida aproximada de 0V y +3.3V respectivamente, por esto se asegura que la señal que entra al microcontrolador se encuentra en el rango anterior. La ecuación del voltaje de salida para la entrada del taco generador es:

$$Vout = \frac{3}{10.57}Vin + \frac{5}{3}V$$

Ecuación 2.

Al reemplazar en la ecuación anterior los valores de -5.3V y +5.3V se obtendrá una salida aproximada de 0V y +3.3V respectivamente, por esto se asegura que la señal que entra al microcontrolador se encuentra en el rango anterior.

Figura 8. Circuito acondicionador de señal.



5. PROGRAMACION DEL MICROCONTROLADOR MCF51CN128.

Para la programación de este microcontrolador se utiliza el Software Codewarrior versión 6.3 junto con el RTOS MQX versión 3.5. El algoritmo de control se genera con la herramienta de RTWEC de *Simulink*.

El proyecto se divide en "pequeñas" tareas. Para realizar esta división se revisaron los requerimientos del proyecto. Los requerimientos del proyecto son:

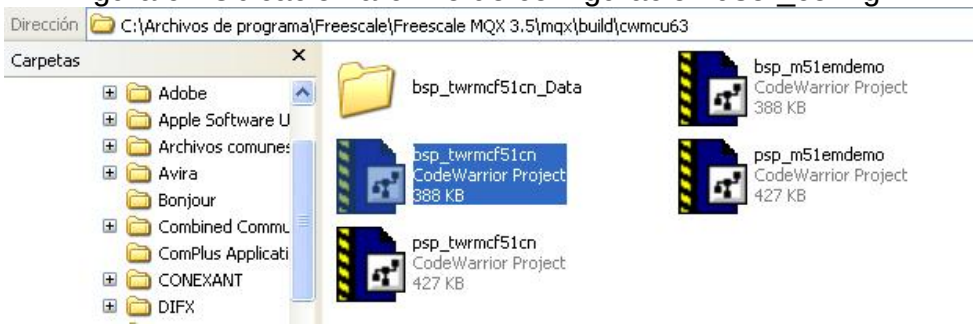
- Lectura de una entrada analógica.
- Salida PWM.
- Comunicación bidireccional RS232.

Adicionalmente se realiza un archivo de configuración donde se seleccionan los periféricos que se van a utilizar en el proyecto, facilitando el enlace con el código generado con la RTWEC.

5.1 CONFIGURACION DEL RTOS MQX 3.5.

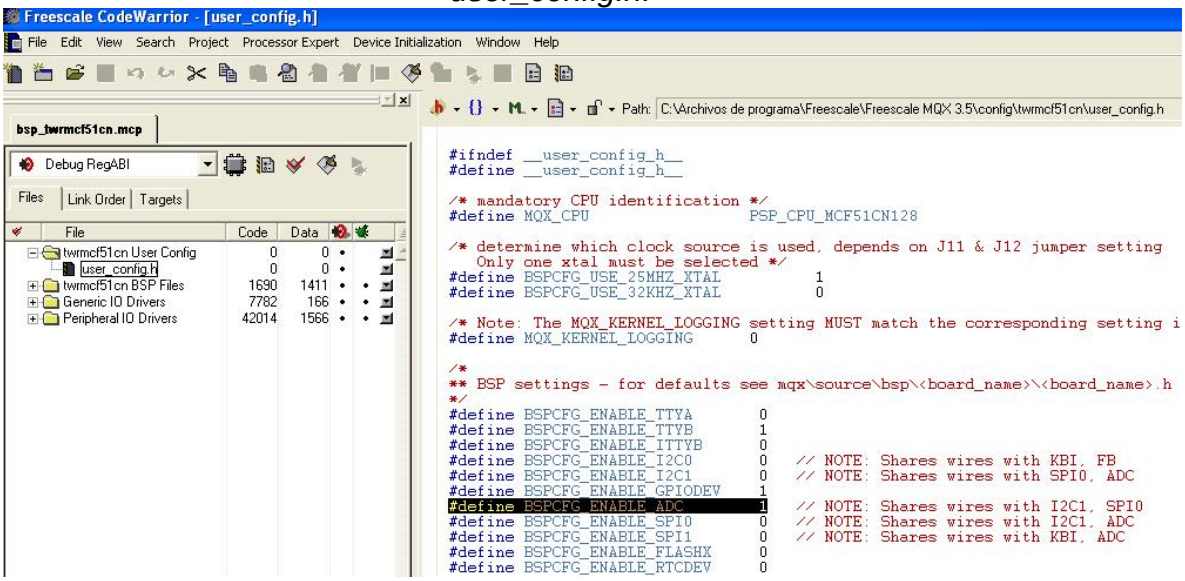
Para programar utilizando MQX éste se debe configurar el MQX dependiendo de las necesidades del proyecto, para hacer esto es necesario hacer los cambios respectivos en el archivo de configuración de usuario *bsp_twrmcf51cn* que se encuentra en la carpeta *\mqx\build\cwmcu63* de la instalación del MQX, tal como se muestra en la figura 9. Para que los cambios surtan efecto es necesario compilar el archivo con el *Codewarrior*.

Figura 9. Ubicación archivo de configuración user_config.h



En el archivo *user_config.h*, se debe verificar que esté habilitado el convertidor analógico-digital, asignando un valor de 1, tal como se muestra en la figura 10. Es importante anotar que la configuración por defecto del planificador es FIFO preferente.

Figura 10. Configuración convertidor analógico-digital en el archivo *user_config.h.*



5.2 PROGRAMACION DE LAS TAREAS.

En la figura 11 se muestra la sección del código donde se configuran las tareas del RTOS. Para crear las tareas es necesario configurar sus características en el TASK_TEMPLATE_STRUCT del archivo main.c que genera el MQX.

Figura 11. Sección Task_Template_Struct.

```
TASK_TEMPLATE_STRUCT MQX_template_list[] =
{
/*  Task number, Entry point, Stack, Pri, String, Auto? */
   {INIT_TASK,   Init_task,     400,  10,   "init", MQX_AUTO_START_TASK},
   {ACTUA_TASK,   Actua_task,   600,  11,   "adc",       0,             },
   {COM_TASK,    Com_task,      800,  8,    "com",       0,             },
   {PID_TASK,   Pid_task,       800,  9,   "pid",       0,             },
   {0,           0,           0,     0,    0,       0}
};
```

Una tarea se configura con los siguientes valores:

- *Task Number:* Corresponde a un número que identifica la tarea. En la figura 10 se hace referencia con un nombre que previamente se definió con una directiva de preprocesamiento.
- *Entry point:* es el nombre alfanumérico con el que el usuario puede identificar la tarea.

- *Stack:* Es el valor reservado de memoria donde el MQX guarda los registros y variables de la tarea en el momento de los cambios de contexto.
- *Prioridad*: Es el número que identifica el grado de importancia de la tarea. Es importante anotar que a mayor número de prioridad menor es su grado de importancia.
- *String*: Es el puntero que indica la posición de memoria asignada a cada una de las tareas.
- *Auto*: Corresponde a una directiva del MQX donde indica el estado de la tarea en el momento de su creación. Si la directiva corresponde a MQX_AUTO_START_TASK, el MQX le asigna a la tarea el estado de preparada, si por el contrario la directiva corresponde a 0, el MQX reserva los recursos de la tarea pero no inicia la misma.

5.2.1 Tarea de inicialización *(Init_task*). Esta tarea se ha configurado para iniciar automáticamente en el planificador, se encargada de inicializar algunos de los periféricos mediante las siguientes funciones:

- *declaraciones*: esta función configura los LEDs como pines de salida y los pulsadores como pines de entrada, dependiendo de los requerimientos del usuario estos se activan o se desactivan.
- *PWM*: esta función se encarga de configurar el módulo PWM del microcontrolador.

El módulo PWM se configura de tal forma que la señal de salida tenga una frecuencia de 300Hz. Después de configurar el módulo PWM, éste actualiza su valor por medio de la interrupción denominada *Vtpm2ovf_isr*.

Una vez terminado el proceso de inicialización de esta tarea se crea la tarea de actualización.

5.2.2 Tarea de Actualización (*Actua_task*). Esta tarea se encarga de terminar la tarea de inicialización para liberar sus recursos, luego se encarga de inicializar el convertidor analógico-digital mediante la función *SEC_Initialize* con lo cual empieza la adquisición de valores. Es importante resaltar que función el canal del convertidor analógico-digital es necesario configurar la directiva ADC_INIT_CHANNEL_STRUCT que se encuentra en el archivo adc.c creado por el autor, tal y como se muestra en la figura 12.

Figura 12. Configuración del ADC_INIT_CHANNEL_STRUCT

```
const ADC_INIT_CHANNEL_STRUCT adc_ch_param[ANALOGAS_TOT] = {
    #if USE_POT
    {

            BSP_ADC_CH_POT,
            ADC_CHANNEL_MEASURE_LOOP | ADC_CHANNEL_START_NOW,
            10,             /* number of samples in one run sequence */
            0,              /* time offset from trigger point in ns */
            50000,      /* period in us (50ms) */
            0x10000,        /* scale range of result (not used now) */
            1,          /* circular buffer size (sample count) */
            ADC_TRIGGER_1  /* logical trigger ID that starts this ADC channel */

    },
    #endif
    #if USE_ACCEX
    {

            BSP_ADC_CH_ACCEL_X,
            ADC_CHANNEL_MEASURE_LOOP | ADC_CHANNEL_START_NOW,
            1,              /* number of samples in one run sequence */
            0,              /* time offset from trigger point in ns */
            25000,      /* period in us (25ms) */
            0x10000,        /* scale range of result (not used now) */
            1,          /* circular buffer size (sample count) */
            ADC_TRIGGER_1  /* logical trigger ID that starts this ADC channel */

    },
    #endif
```

Como se puede observar en la figura 12 en esta estructura es necesario configurar el canal que se va a leer, el tipo de lectura, el periodo de la interrupción de la adquisición de datos y la directiva para indicar el tipo de disparo (*trigger*) del canal analógico. La actualización del valor leído por el convertidor analógico-digital tiene lugar en la interrupción *ReadADC.*

Esta tarea a diferencia de la tarea de inicialización queda en un ciclo infinito en el cual se almacena los valores leídos por el convertidor analógico-digital y actualiza el valor del ciclo de trabajo del PWM.

Una vez inicializado el convertidor analógico-digital se crean las tareas de control y la tarea de comunicación.

5.2.3 Tarea de control PID (*Pid_task*). Esta tarea se encarga de llamar las funciones creadas con la herramienta RTWEC. La tarea ejecuta la función de inicialización y después entra en un ciclo infinito en el cual espera una bandera que se activa en la tarea de comunicación para empezar a ejecutar el algoritmo de control. De igual forma cuando la comunicación con el computador se detiene la bandera cambia y detiene la ejecución del algoritmo de control. El algoritmo de control implementado se explica en la sección 8.

5.2.4 Tarea de comunicación (*Com_task*). Esta tarea se encarga de realizar la comunicación serial entre el microcontrolador y el computador con el fin de visualizar la respuesta del sistema de control y sintonizar las

ganancias del controlador PID y el Set Point del sistema. Esta tarea se podría dividir en las siguientes etapas.

- Etapa 1. El microcontrolador espera la señal del computador indicando que está preparado para iniciar la conexión.
- Etapa 2. El microcontrolador recibe la información del computador con los datos correspondientes al valor de referencia del sistema de control y el valor de las diferentes ganancias del microcontrolador.
- Etapa 3. Transfiere los datos del valor de referencia y las ganancias del controlador al algoritmo de control.
- Etapa 4. Este es un ciclo que solo se interrumpe cuando el computador solicita romper la comunicación con el microcontrolador, llegado este caso esta tarea volverá a la etapa 1. Si no se ha roto este ciclo el microcontrolador envía al computador los datos requeridos para que este dibuje la respuesta del sistema de control, estos datos son el valor de referencia del sistema de control, el valor medido en el convertidor analógico-digital y un contador que lleva el registro del número de muestra.

En esta tarea de comunicación se envían y reciben datos de doble precisión, pero el sistema operativo MQX no reconoce la orden de lectura ni escritura de este tipo de datos. Por esto teniendo en cuenta que el microcontrolador utiliza el estándar IEEE745 y que utiliza un ordenamiento de bytes de tipo *Big Endian* para almacenar los datos en su memoria.

El estándar IEEE745 especifica que para almacenar cualquier valor de punto flotante se debe realizar una serie de operaciones para pasarlo a binario y así poder ser procesado en un dispositivo electrónico determinado. En este proyecto se enviaron y se recibieron este tipo de datos en segmentos hexadecimal aplicando las siguientes definiciones:

- Envío de datos de precisión doble: Para enviar datos hacia el computador de precisión doble fue necesario utilizar un puntero de memoria a la variable que va a ser transmitida, gracias a este puntero, se pueden transmitir los bytes uno a uno especificando que se están transmitiendo en formato hexadecimal.

- Recepción de datos de precisión doble: Para recibir datos de precisión doble es necesario crear dos variables debido a que mientras un dato de precisión doble cuenta como ocho bytes, el tamaño más grande de una variable sin punto flotante que se puede crear en el microcontrolador es de cuatro bytes, por esto se crearon dos variables en las cuales se van a almacenar los cuatro bytes más significativos y los cuatro bytes menos significativos del número de precisión doble respectivamente, cada una de estas variables lleva asignado un

puntero a su memoria, después de haber almacenado los ocho bytes en las dos variables diferentes de tipo *unsigned long*, se pasa a ordenar estos bytes gracias a los punteros en la variable de precisión doble hacia la cual venia dirigido el dato.

5.2.5 Archivo de enlace RTWEC y RTOS MQX. La creación de este archivo surgió por la necesidad de agilizar la compilación del código generado por el RTWEC evitando hacer cambios en el código donde se programa la tarea. En este archivo se seleccionan los periféricos del microcontrolador que se utilizan en el proyecto.

Para el enlace de las variables generadas en el código mediante el RTWEC con las variables del ambiente de desarrollo Codewarrior, ambas variables deben ser del mismo tipo. Se hizo un análisis sobre los diferentes tipos de datos que se pueden utilizar, estos tipos son:

- Entradas analógicas. Estas entradas corresponden a valores que se deben tomar del convertidor analógico-digital del microcontrolador, éste entrega una variable entera sin signo de 16 bits. En el diagrama de bloques de *Simulink* debe ser un bloque de *IN*.

- Entradas digitales. Estas entradas corresponden a valores booleanos que deben ser leídos de cualquier pin del microcontrolador, este tipo de variable corresponde a un valor booleano. En el diagrama de bloques de *Simulink* este debe ser un bloque de *IN*.

- Salidas digitales. Estas salidas corresponden a valores booleanos. En el diagrama de bloques de *Simulink* este debe ser un bloque de OUT.

- Salida PWM. El valor de esta variable se configura para que reciba un valor entre 0 y 100, éste valor es equivalente al porcentaje del ciclo de trabajo de la salida PWM. En el diagrama de bloques de *Simulink* este debe ser un bloque de OUT.

- Ganancias del controlador PID. En este proyecto se ha implementado una comunicación serie que permite modificar los parámetros P, I y D del controlador. En el diagrama de bloques de *Simulink* este debe ser un bloque de Constant.

- Referencia del sistema de control. Al igual que para los parámetros P, I y D del controlador se ha programado la posibilidad de cambiar el la referencia mediante una comunicación serial, el tipo de esta variable es double. En el diagrama de bloques de *Simulink* este debe ser un bloque Constant.

Para que el archivo de configuración reconozca las variables creadas en el modelo de *Simulink* y las enlace correctamente con el periférico del módulo TWR-MCF51Cn128 estas deben cumplir con los nombres que se observan en la tabla 1.

Tabla 1. Nombres de las variables para funcionamiento del archivo de configuración.

Nombre modelo de *Simulink*	**Bloque de *Simulink***	**Nombre otorgado por usuario al bloque**	**Periferico que se lee o escribe.**	**Tipo Variable RTWEC**
FAST_P	IN	POTE	Canal 3 CAD	Uint_16
FAST_P	IN	ACEX	Canal 0 CAD	Uint_16
FAST_P	IN	ACEY	Canal 1 CAD	Uint_16
FAST_P	IN	ACEZ	Canal 2 CAD	Uint_16
FAST_P	IN	SW2	Pin G6	Booleana
FAST_P	IN	SW3	Pin G7	Booleana
FAST_P	OUT	LED1	Pin E3	Booleana
FAST_P	OUT	LED2	Pin G5	Booleana
FAST_P	OUT	LED3	Pin E5	Booleana
FAST_P	OUT	LED4	Pin H3	Booleana
FAST_P	OUT	DUTY	Módulo PWM.	Uint_16
FAST_P	Constant	Kp	Módulo Serial	Double.
FAST_P	Constant	Ki	Módulo Serial	Double.
FAST_P	Constant	Kd	Módulo Serial	Double.
FAST_P	Constant	SET_POINT	Módulo Serial	Double.

En la figura 13 se muestra una sección del archivo de configuración *core.h.* Para realizar este archivo se tomó como base el archivo de configuración de MQX en el cual se selecciona mediante la asignación de un 1 ó un 0 si se van a enlazar los diferentes periféricos, o se va a hacer uso de la modificación de los valores del controlador mediante la comunicación serial.

Figura 13. Sección del archivo de configuración core.h

```
//CORE.h
#define ENTRADAS_ANALOGAS       0
#define ENTRADAS_DIGITALES      0
#define NUMERO_SALIDAS_ANA      0
#define NUMERO_GAIN             0
#define NUMERO_CONST            0
#define NUMERO_SALIDAS_DIG      0
#define COM_SERIAL              0
#define  USE_SET_POINT          0
#define  SET_POINT              0
#define  CONTROL_P              0
#define  CONTROL_I              0
#define  CONTROL_D              0
#define SALIDAS_PWM             1
#define DUTY_CICLE              50
/*PERIFERICOS ESPECIFICOS DE TWR MCF51CN128*/
#define USE_POT             0
#define USE_ACCEX           1
#define USE_ACCEY           0
#define USE_ACCEZ           0
#define USE_LED1            0
#define USE_LED2            0
#define USE_LED3            0
#define USE_LED4            0
#define USE_SW2             0
#define USE_SW3             0
```

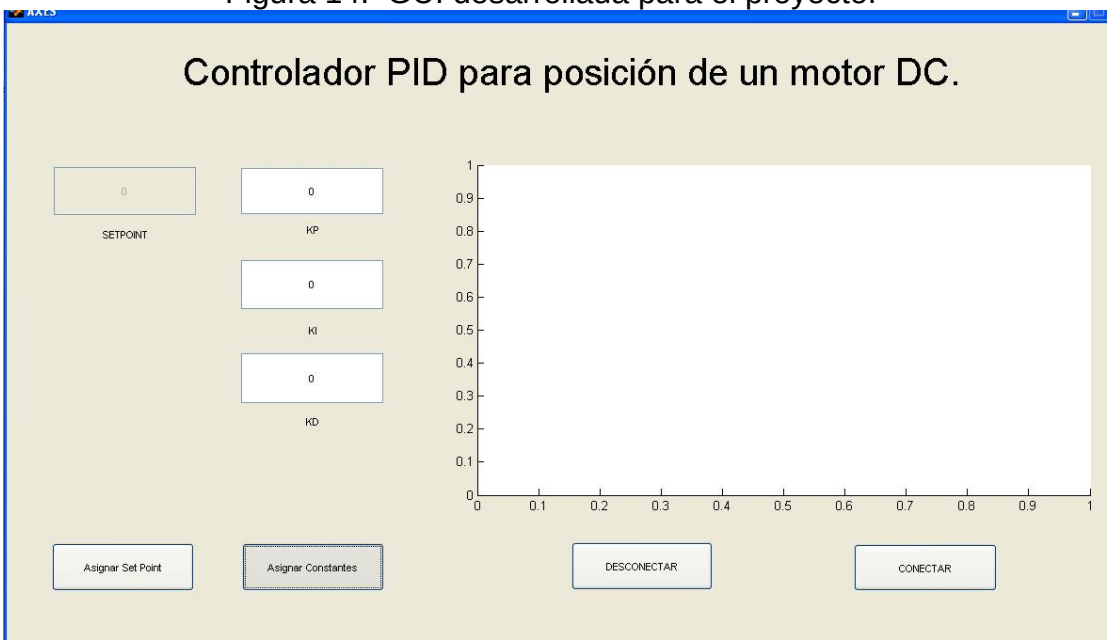
Dependiendo de la selección en el archivo de configuración, al compilar el código se omitirán las instrucciones dirigidas hacia los periféricos que no están seleccionados, de esta forma el código será menos robusto.

5.3 INTERFAZ DE USUARIO

El proyecto cuenta con una comunicación entre el microcontrolador y el computador. Por lo tanto, se necesita contar con una interfaz que permita cambiar el valor de las ganancias del controlador PID así como el valor de referencia del sistema, además de tener la posibilidad de graficar la respuesta del sistema de control. La interfaz se ha desarrollado utilizando una GUI del software *Matlab*.

5.3.1 Diseño gráfico de la GUI. En la figura 14 se muestra la GUI desarrollada para éste proyecto. La GUI cuenta con dos botones en la parte izquierda, uno de estos habilita el cuadro de texto para a ajustar la referencia del sistema de control y el otro habilita los cuadros de texto para ajustar las ganancias de el controlador PID. En la derecha de la GUI se observa el espacio (*axes*) donde se grafican los datos recibidos y debajo de este, dos botones, uno para la conexión con el microcontrolador y otro para la desconexión del microcontrolador.

Figura 14. GUI desarrollada para el proyecto.



5.3.2 Comunicación GUI-microcontrolador. Para iniciar una comunicación entre la GUI y el microcontrolador se configura un objeto de *Matlab* que habilita la comunicación serial, este objeto se configura con los valores que MQX entrega para la comunicación. El algoritmo para la comunicación desde el microcontrolador cuenta con las siguientes etapas:

- Etapa 1. En el momento de dar clic en conectar la GUI crea un temporizador, la función de inicio del temporizador se ejecuta una vez. En esta función el computador envía una señal al microcontrolador para iniciar la comunicación, después de enviar esta señal de inicio el computador envía los valores de la referencia y de las ganancias del controlador PID, luego espera que el microcontrolador envié estos mismos valores de vuelta.
- Etapa 2. En esta etapa el computador espera que la función de temporización se ejecute periódicamente, al ejecutarse ésta se reciben los datos del CAD del microcontrolador, un contador para indicar el número de muestra y la referencia del sistema de control, con estos datos se realiza la gráfica de la respuesta del sistema de control.

Como el MQX no soporta la transmisión de datos con punto flotante, el computador debe enviar y recibir los datos de acuerdo a lo explicado en la sección 5.3.5.

- Envío de datos precisión doble: Para enviar estos datos que se toman los cuadros de texto de la GUI. *Matlab* cuenta con la instrucción *num2hex*, que convierte un número de doble precisión a un número hexadecimal, este valor es almacenado en dos variables, cada una de cuatro bytes para luego ser enviada.

- Recepción de datos de doble precisión: Para recibir los datos de doble precisión que se envían desde el microcontrolador se utiliza la instrucción *str2num*, con esta instrucción se convierte el dato recibido a un número de doble precisión.

6. DISEÑO DEL CONTROLADOR.

Para el diseño del controlador se utilizó Matalb con su herramienta de identificación de sistemas (*system identification tool*) para determinar el modelo de la planta y la herramienta de control (*sisotool),* para obtener las constantes de un controlador PID.

6.1 IDENTIFICACION DEL SISTEMA DE VELOCIDAD.

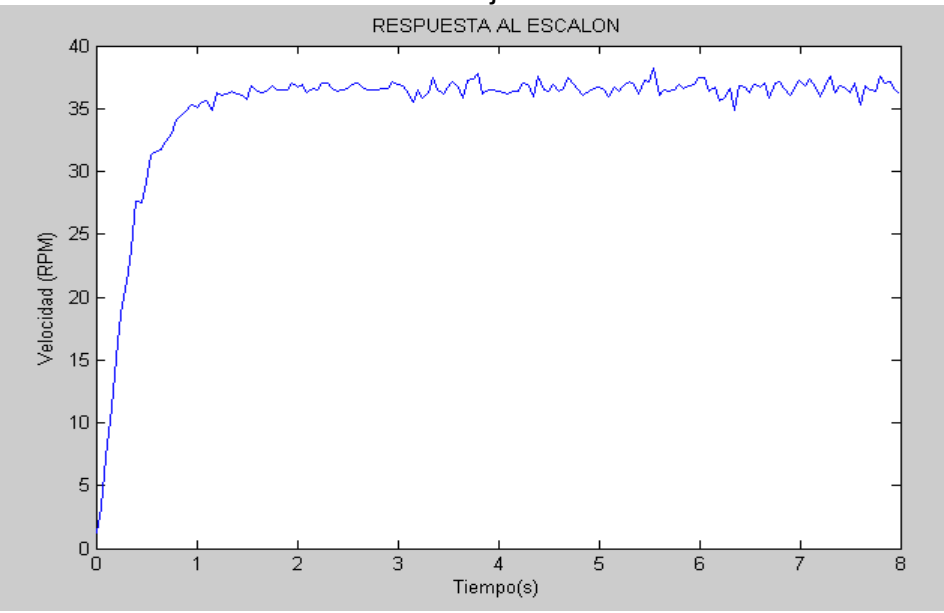
Para identificar el sistema de velocidad del motor, primero se realiza la conversión del valor medido por el convertidor analógico-digital a un su valor equivalente en revoluciones por minuto, utilizando la ecuación 1.

$$X_{(RPM)=}(A - 1870)/37$$

Ecuación 3.

Después se registran los datos de la respuesta del motor en lazo abierto a una entrada de escalón. Para la caracterización del sistema se aplica una señal PWM con un ciclo de trabajo de 20%. Para capturar los datos se implementa un código muy sencillo en el microcontrolador que se encarga de enviar por el puerto serial el valor del convertidor analógico-digital, utilizando un tiempo de muestreo de 50 milisegundos. Para la captura de los datos en el computador se utiliza un programa en *Matlab* que recibe los datos y los almacena en un vector.

Figura 15. Respuesta en lazo abierto ante un PWM con un 20% de ciclo de trabajo.



Este vector obtenido se utiliza para hallar el modelo del sistema, este modelo es hallado con la herramienta de identificación de *Matlab* que entrega la función de transferencia de un sistema al analizar la salida de este frente a una entrada, en este caso particular la entrada es un escalón. La función de transferencia obtenida de esta herramienta se muestra en la ecuación 4.

$$\Omega(s) = \frac{-3.698}{s + 3.025}$$

Ecuación 4.

Dado que el controlador se implementa en un microcontrolador es necesario convertir el sistema en tiempo continuo a un sistema en tiempo discreto. Para hacerlo se utiliza la orden c2d de *Matlab* con un tiempo de muestreo de 50 milisegundos, dando como resultado la ecuación 5.

$$\Omega(z) = \frac{-0.1716}{z - 0.8597}$$

Ecuación 5.

6.2 IDENTIFICACION DEL SISTEMA DE POSICION.

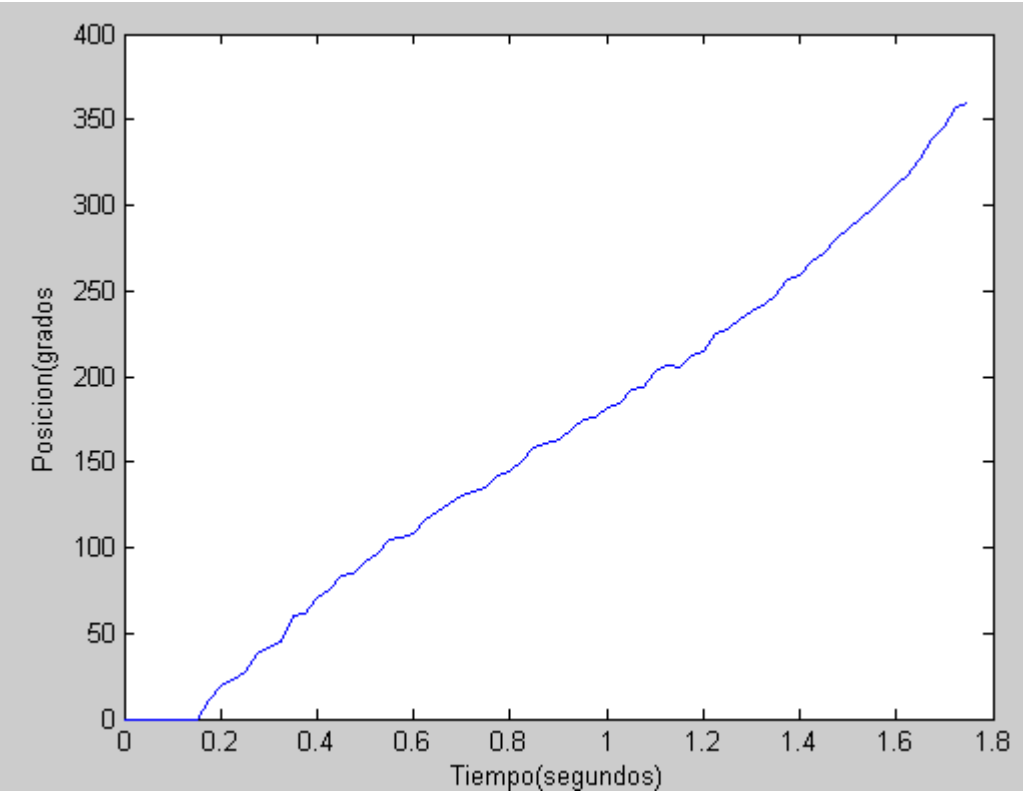
Como se están utilizando diferentes sensores para medir la velocidad y la posición no se puede integrar la función de transferencia de velocidad para hallar la función de transferencia de posición. Al igual que en la identificación del sistema de velocidad se realiza la conversión del valor medido utilizando la ecuación 6.

$$X_{grados} = \frac{A}{11.406}$$

Ecuación 6.

Utilizando los mismos programas se obtiene la respuesta de la planta ante un escalón PWM con un 20% de ciclo de trabajo, tal y como se muestra en la figura 16.

Figura 16. Respuesta en lazo abierto del sistema de posición a un escalón de 20% de ciclo de trabajo de PWM.



Utilizando la herramienta *ident* e y la orden c2d de igual forma como fueron utilizadas para la identificación del sistema de velocidad se obtienen las funciones de transferencia correspondientes al sistema de posición. Estas ecuaciones se muestran a continuación:

$$P(s) = \frac{-7.619}{s + 0.09509}$$

Ecuación 7.

$$P(z) = \frac{-0.38}{z - 0.9953}$$

Ecuación 8.

6.3 LEY DE CONTROL.

Para poder implementar el algoritmo de control PID se lleva este algoritmo a una ecuación de diferencias, la ecuación de un controlador PID en tiempo continuo es:

$$Y_{(t)} = k_p e_{(t)} + k_i \int e_{(t)} dt + k_d \frac{de_{(t)}}{dt}$$

Ecuación 9.

Para transformar esta ecuación a una ecuación de diferencia, se sustituyen los términos integral y derivativo por su equivalente en el dominio del tiempo discreto, aplicando esto se obtiene la siguiente ecuación:

$$\frac{Y_{(k)}}{e_{(k)}} = k_p + k_i \frac{T_s}{z-1} + k_d \frac{(z-1)}{T_s}$$

Ecuación 10.

De (9) la ecuación de diferencias que resulta es:

$$y_{(k)} = k_p\left(e_{(k)} - e_{(k-1)}\right) + k_i T_s\left(e_{(k-1)}\right) + \frac{k_d}{T_s}\left(e_{(k-2)} - 2e_{(k-1)} + e_{(k)}\right) + y_{(k-1)}$$

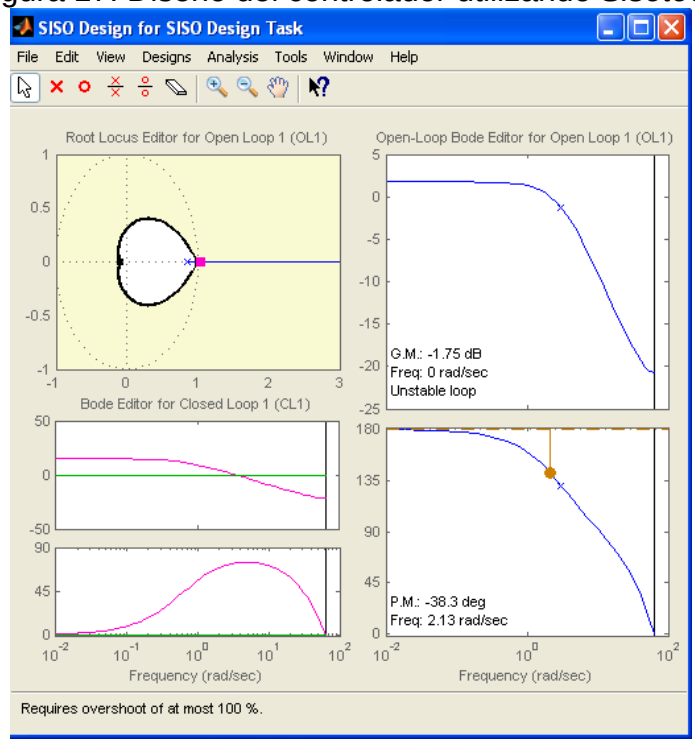
Ecuación 11.

En este proyecto esta ecuación en diferencia se modela en *Simulink* y con la ayuda de la herramienta RTWEC se genera el código correspondiente que se implementa en el microcontrolador.

6.4 SINTONIZACIÓN DE LAS CONSTANTES DEL CONTROLADOR DE VELOCIDAD.

Para halla el valor de las diferentes ganancias del controlador se utiliza la herramienta *sisotool* de *Matlab*. En la ventana de diseño se pueden modificar los requerimientos de diseño del sistema de control. En este caso se restringe el controlador para evitar un sobrepaso mayor al 10% en el sistema. En la figura 17 se muestra que el programa delimita una zona en la cual se deben ubicar las raíces del sistema para que cumpla el requerimiento deseado.

Figura 17. Diseño del controlador utilizando *Sisotool*.



En la ecuacion 12 se muestra el compensador que resulta del diseño utilizando sisotool.

$$\frac{y_{(k)}}{e_{(k)}} = -0.65144x\frac{(z-0.675)}{(z-1)}$$

Ecuación 12.

Igualando la ecuación 12 y la ecuación 10 se obtienen las siguientes constantes:

- Kp= -0.65144
- Ki= -4.23436.

6.5 SINTONIZACIÓN DE LAS CONSTANTES DEL CONTROLADOR DE POSICION.

Para sintonizar las constantes del controlador de posición se utiliza la herramienta *sisotool* de *Matlab* de forma igual que en el de velocidad, asumiendo un requerimiento de sobrepaso menor al 10%, En este caso a diferencia del caso anterior se utiliza un controlador proporcional. La constante proporcional que se obtiene es:

- Kp= -0.3.

7. SIMULACIONES

Se realizan las simulaciones de los controladores para observar su posible comportamiento. Esta simulación se realiza en *Simulink*, el diagrama de bloques que se diseña para la simulación del controlador se adapta fácilmente al microcontrolador con la ayuda del RTWEC. Para ello es necesario utilizar un bloque de saturación para limitar las salidas de controlador entre -50 y +50, ya que esta señal representa la variación de PWM alrededor de su valor de offset de 50%.

7. 1 SIMULACION DEL CONTROLADOR DE VELOCIDAD.

Para simular este controlador es necesario simular todo el lazo de control. Primero se necesita abrir un nuevo modelo en *Simulink* y dibujar el diagrama de bloques equivalente a la ecuación 11, teniendo en cuenta el valor de las constantes calculadas en la sección 6.4 y el valor del tiempo de muestreo de 50 milisegundos. La simulación se realiza para una entrada de referencia que sigue un tren de escalones entre -40 y +40 revoluciones por minuto con un periodo de 8 segundos. En la figura 18 se muestra el diagrama de bloques de simulación del controlador de velocidad y en la figura 19 se muestra el detalle del controlador de velocidad. En la Figura 20 también se muestra el resultado de la simulación, donde la referencia se dibuja en color azul y la respuesta del sistema de control se dibuja en color verde.

Figura 18. Diagrama de bloques simulación controlador de velocidad.

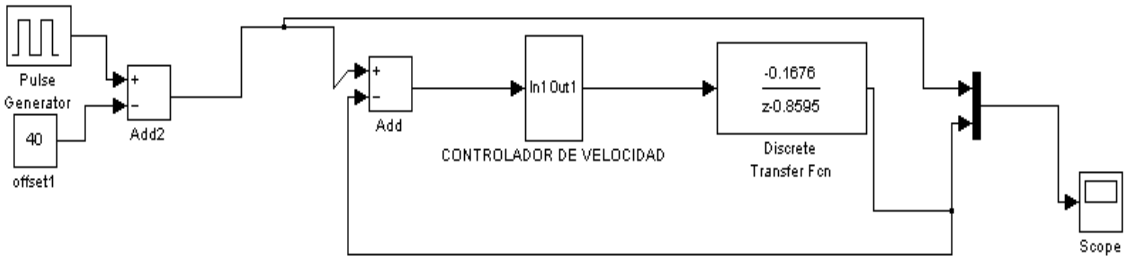


Figura 19. Bloque CONTROLADOR DE VELOCIDAD.

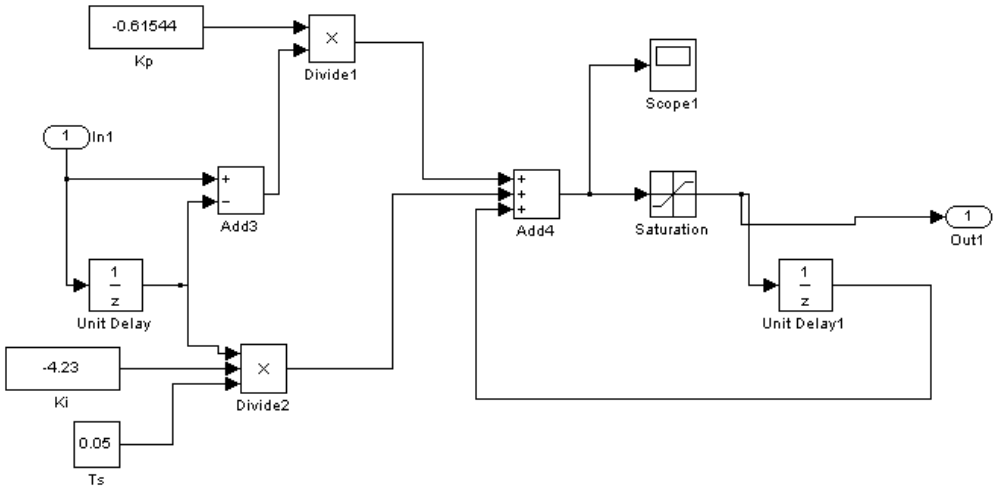
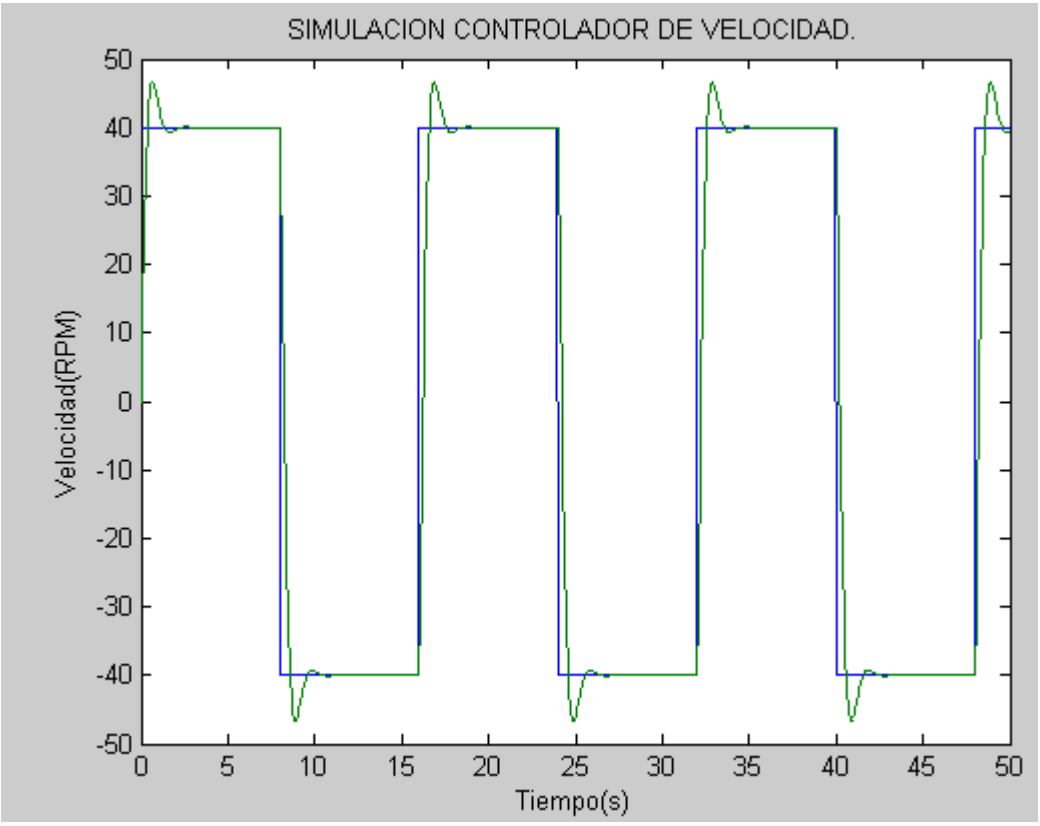


Figura 20. Simulación controlador de velocidad.



Los resultados obtenidos por la simulación se ajustan a lo que se esperaba de acuerdo con lo descrito en el diseño del controlador, ya que en el

momento de diseño del mismo se especifico un sobresalto menor al 10%, y a pesar de que en la simulación el sobresalto es un poco mayor a éste porcentaje, se decide generar el código con el RTWEC para el bloque controlador de velocidad cuyo diagrama se observa en la figura 19.

7.2 SIMULACION CONTROLADOR POSICION.

Para la simulación del control de posición se realizó exactamente el mismo procedimiento que para la simulación del controlador de velocidad cambiando lógicamente la ecuación en diferencia de acuerdo a la constante hallada en la sección 6.5. Para este caso como la posición oscila entre 0 y 359 grados la referencia de posición se simula para que oscile entre 100 y 200 grados.

Como se observa en la figura 16 El sistema de posición tiene un retardo de 0.15 segundos, por lo anterior se incluyó un bloque de retardo en la simulación. En la figura 21 se muestra el diagrama de bloques de la simulación del controlador de posición, en la figura 22 se muestra el detalle del controlador de posición. En la Figura 23 también se muestra el resultado de la simulación, donde la referencia se dibuja en color azul y la respuesta del sistema de control se dibuja en color verde.

Figura 21. Diagrama de bloques simulación control de posición.

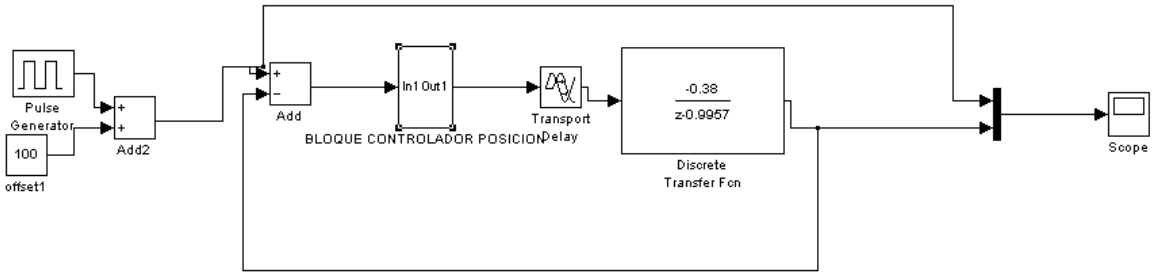


Figura 22. Diagrama BLOQUE CONTROLADOR DE POSICION.

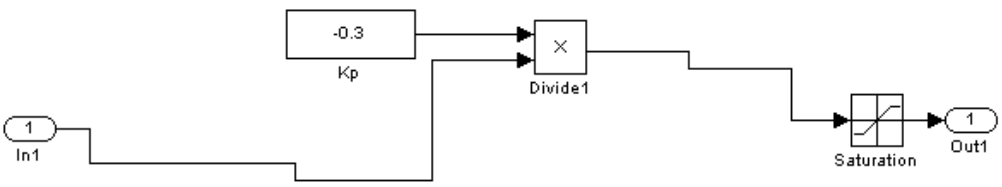
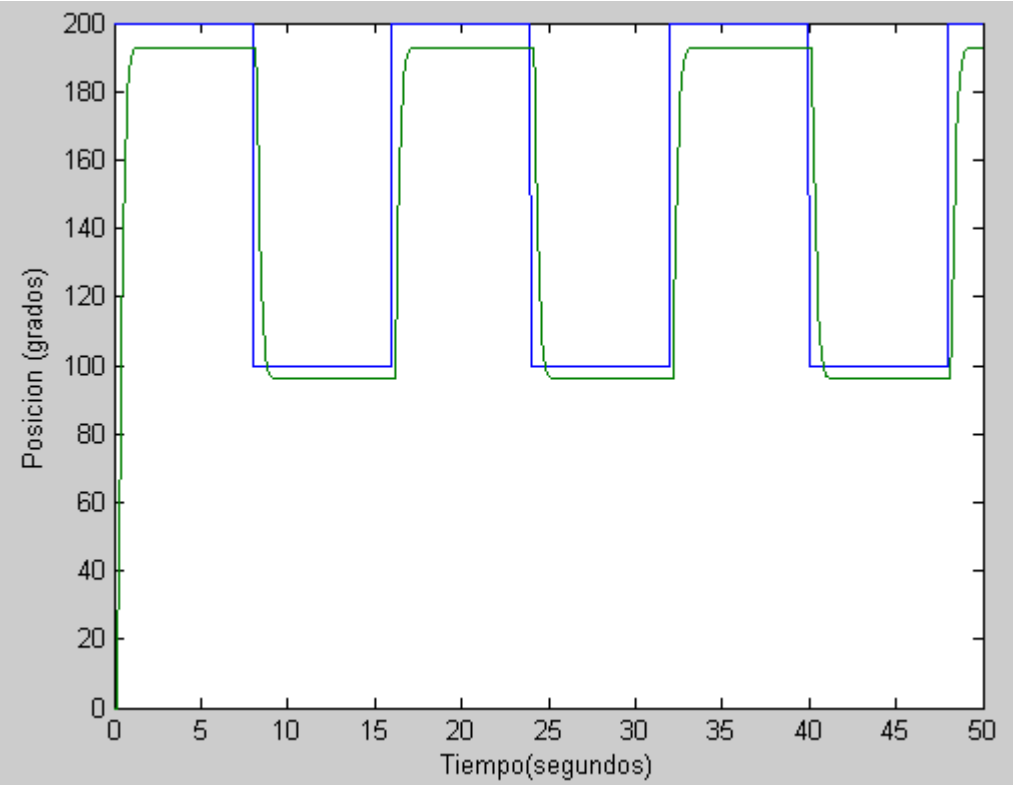


Figura 23. Simulación controlador de posición.



Al observar la grafica obtenida se observa que no hay un error de estado estable, esto debido a que el controlador netamente proporcional no elimina este error, de igual manera se procede a generar el código para este bloque teniendo en cuenta que se tiene un buen punto de partida para la sintonización del controlador.

8. GENERACION DE CODIGO PARA LA TAREA PID.

Para empezar a crear el proyecto es necesario recordar que el diagrama de bloques debe llevar el nombre FAST_P al momento de guardar el proyecto, y para que las variables que se generen coincidan se debe utilizar la tabla 1 para nombrar las diferentes variables, esta tabla ya había sido tenida en cuenta en el momento de realizar las simulaciones. En las figuras 24 y 25 se observa respectivamente el diagrama de bloques del cual se generó el código con la RTWEC para implementar el control de velocidad y de posición para el motor de corriente directa.

Figura 24. Diagrama del cual se genera el código para implementar un control de velocidad en el microcontrolador.

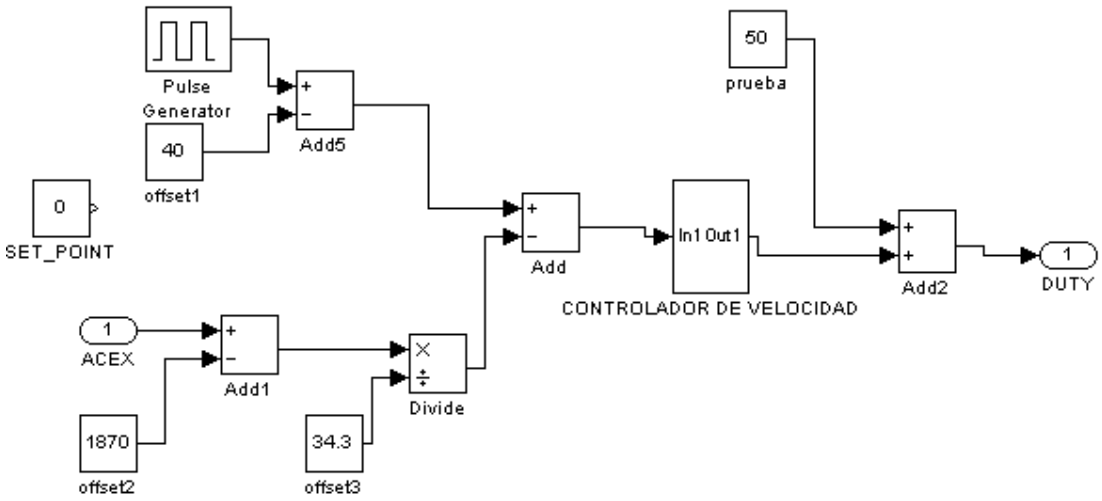
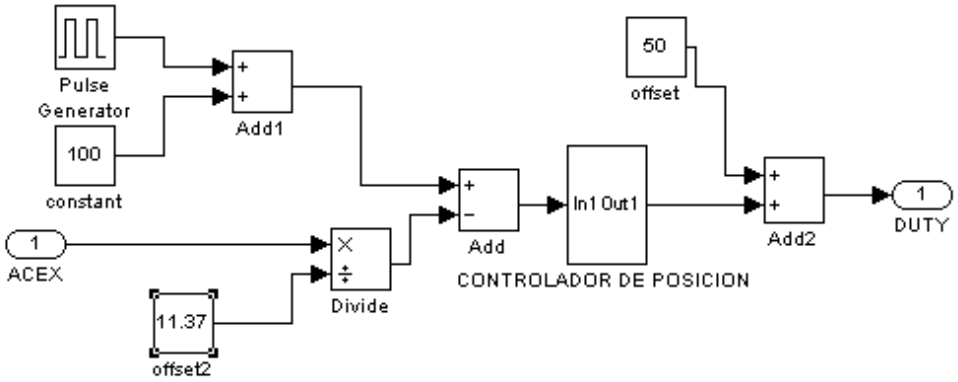


Figura 25. Diagrama del cual se genera el código para implementar un control de velocidad en el microcontrolador.



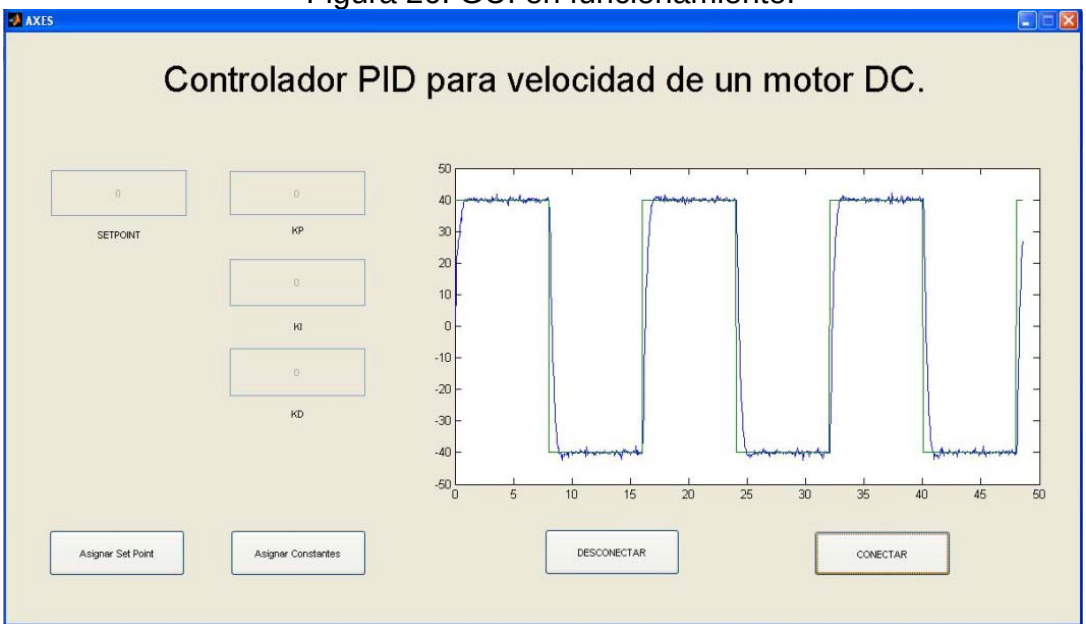
Como se puede observar a la entrada del convertidor analógico digital se le debe aplicar las operaciones especificadas en las ecuaciones 1 y 4 para velocidad y posición respectivamente. Se debe recordar que el controlador diseñado satura sus salidas en -50 y +50, por esto es necesario sumarle un offset a su salida de 50. Si se observa el valor de referencia del sistema lo va a dar un generador de pulsos idéntico al utilizado en las simulaciones, esto para que los datos que se obtengan experimentalmente puedan ser comparados como los obtenidos en estas.

Después de utilizar la herramienta RTWEC de *Simulink* para generar el código, se toman los archivos .c que se hayan generados y se copian en el proyecto para ser compilados e implementados en el microcontrolador, teniendo en cuenta que la tarea de control (*Pid_task)*, se programó para que llame las funciones de inicializar y la función de paso, esta última es la encargada de realizar el control del sistema, es decir ejecuta lo creado en el diagrama de bloques una vez cada tiempo de muestreo.

9. RESULTADOS.

Después de programar en el microcontrolador código generado por el RTWEC se realizó la conexión con el motor de corriente continua por medio del circuito acondicionador de señal y se realizó una toma de datos con el fin de comparar los resultados obtenidos experimentalmente con los obtenidos en las simulaciones. En la figura 26 se observa una imagen de la GUI en funcionamiento.

Figura 26. GUI en funcionamiento.



Como se puede notar en la figura para esta verificación no se requiere comunicar un valor de referencia, ni el valor de las variables, por esto los botones de activación no están pulsados y por ende los cuadros de texto están deshabilitados.

Para facilitar la visualización de los resultados se graficaron en una misma figura los datos tanto de la simulación

9.1RESULTADOS DEL CONTROLADOR DE VELOCIDAD.

En la figura 27 se observan los datos obtenidos para las constantes sintonizadas mediante el proceso explicado en la sección 6.4. Aparte de la obtención de los datos que se muestran en la figura 27, se obtuvieron unos datos tanto en simulación como experimentalmente en los cuales la constante integrar fue reducida a -2.4080, para disminuir el sobre salto del sistema, estos resultados se observan en la figura 28.

Figura 27. Resultados controlador de velocidad.

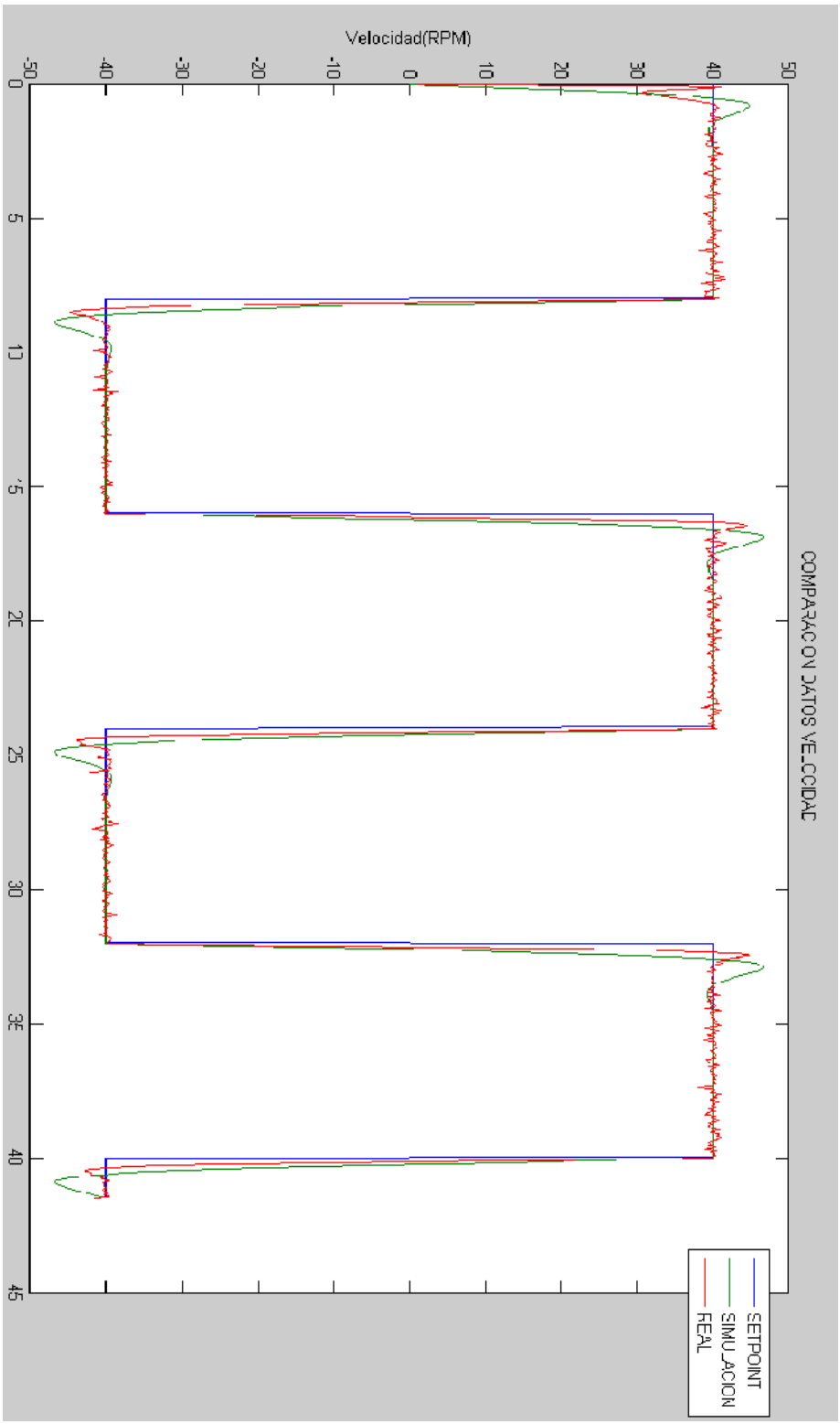
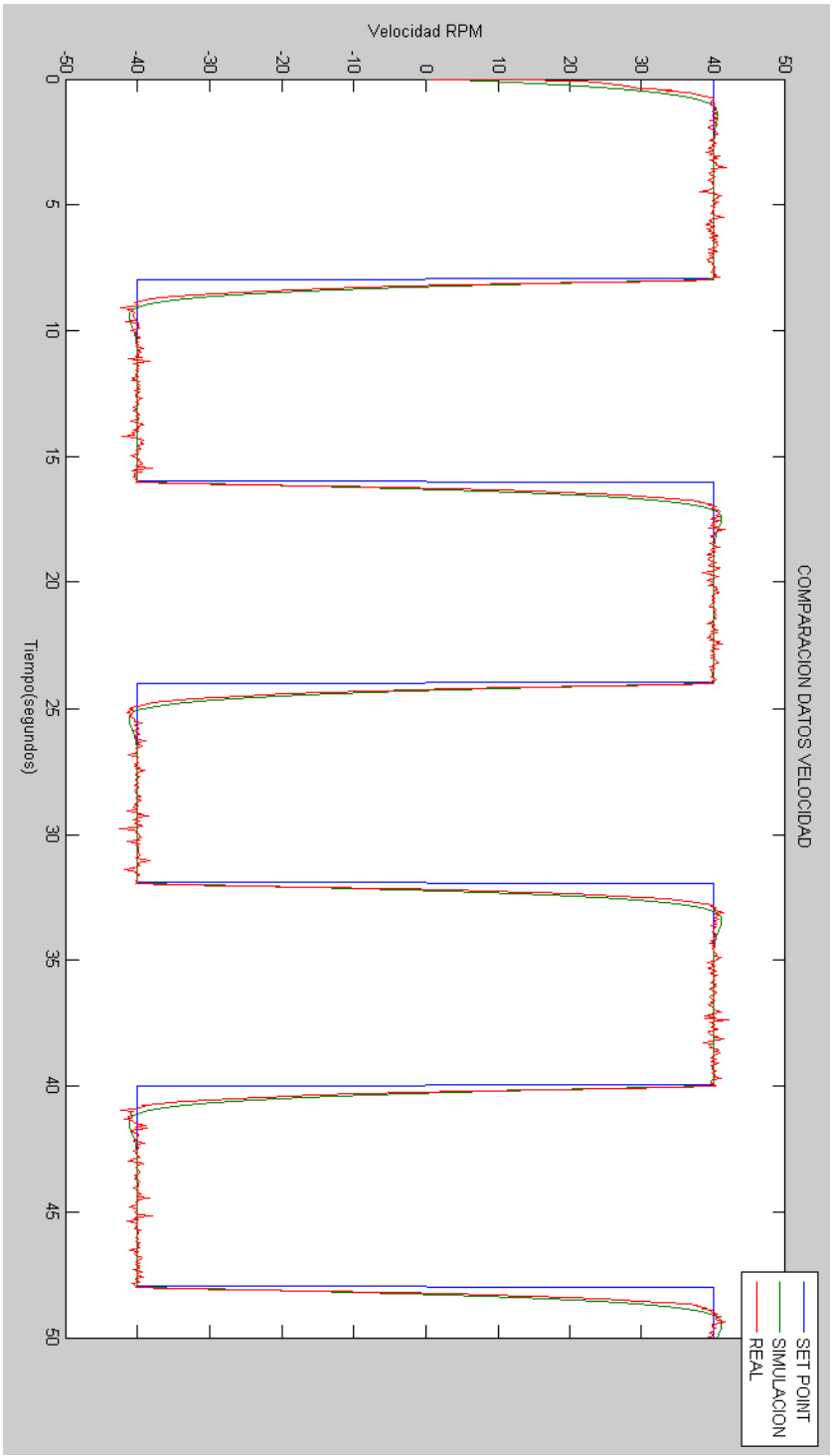


Figura 28. Resultados controlador velocidad con sobresalto reducido.



Como se muestra en la figura 27, el sistema real también tiene un sobresalto, pero en este caso es menor que el sobresalto de la simulación al igual que el tiempo de establecimiento, sin embargo la diferencia no es muy significativa. Al comparar los datos obtenidos en la figura 27 con los obtenidos en la figura 28, se confirma un principio básico de control, al disminuir la ganancia integral y mantener la ganancia proporcional constante se disminuye el sobresalto pero aumenta el tiempo de establecimiento.

9.2 RESULTADOS CONTROLADOR POSICIÓN

Al igual que con los resultados presentados en velocidad, para la posicion también se tomaron datos con 2 valores de constante diferente, uno -0.3 que fue el obtenido en la sección 6.5, y otro -0.6 este último solo para observar el comportamiento y validar la simulación.

En la figura 29 se muestra la comparación de los resultados experimentales con los de simulación para una constante proporcional de -0.3, se observa la similitud del sistema real con el sistema simulado, pero se nota una diferencia considerable cuando el motor gira en sentido contrario de las manecillas del reloj, esto indica que el motor de corriente continua tiene una dinámica diferente cuando gira en este sentido, y no se notó anteriormente porque la caracterización del mismo se realizo haciendo girar el motor en sentido de las manecillas del reloj.

En la figura 30 se muestra la comparación de los resultados experimentales con los de simulación para una constante proporcional de -0.6, se observa la similitud del sistema real con el sistema simulado, al igual que en la figura 29 además de la diferencia en el comportamiento del motor cuando gira en sentido contrario a las manecillas del reloj, se observa un sobresalto. El retardo del sistema que se observa en la figura 16, junto con un valor elevado para la constante proporcional ocasiona el sobresalto.

Figura 29. Resultados posición Kp=-0.3

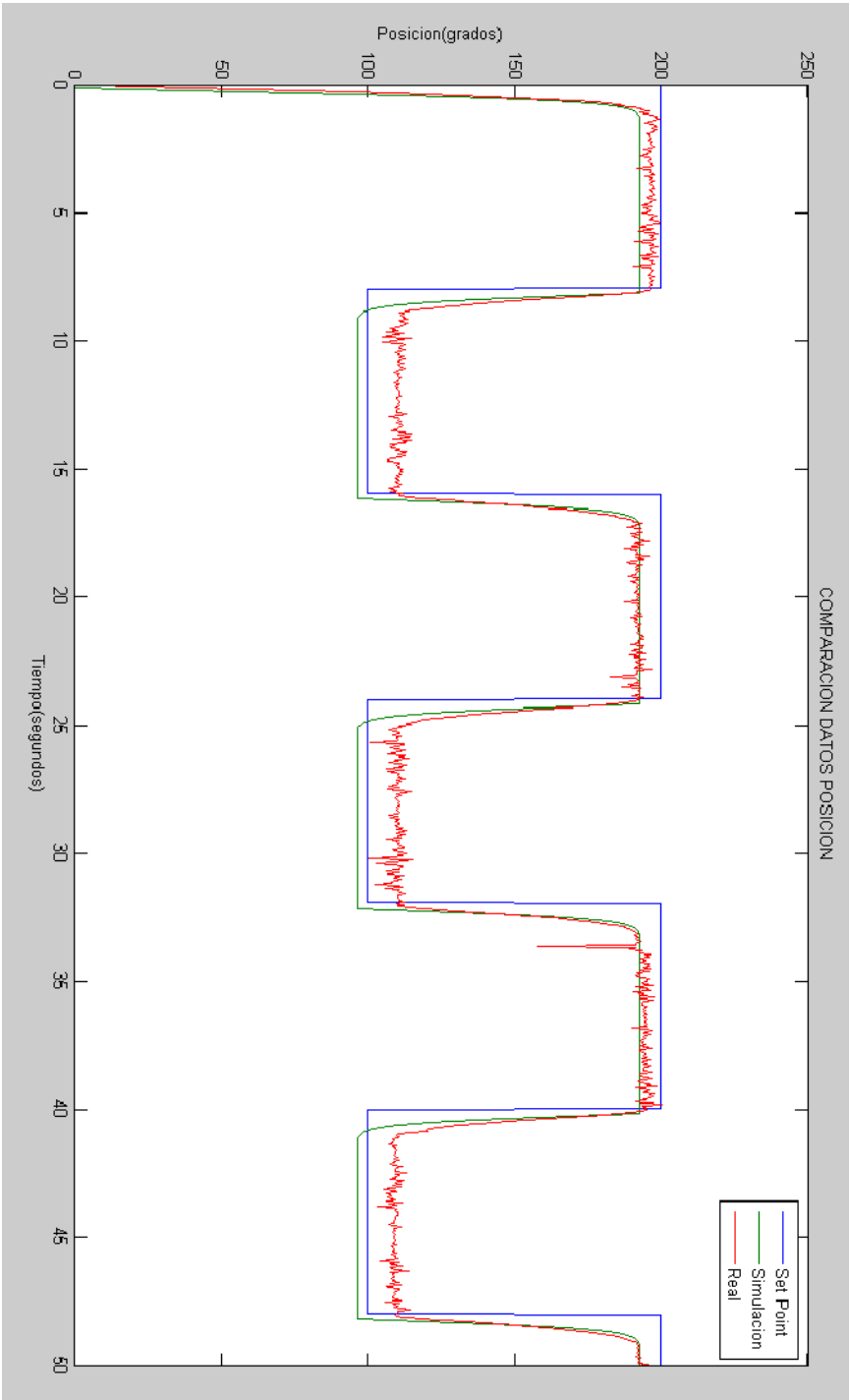
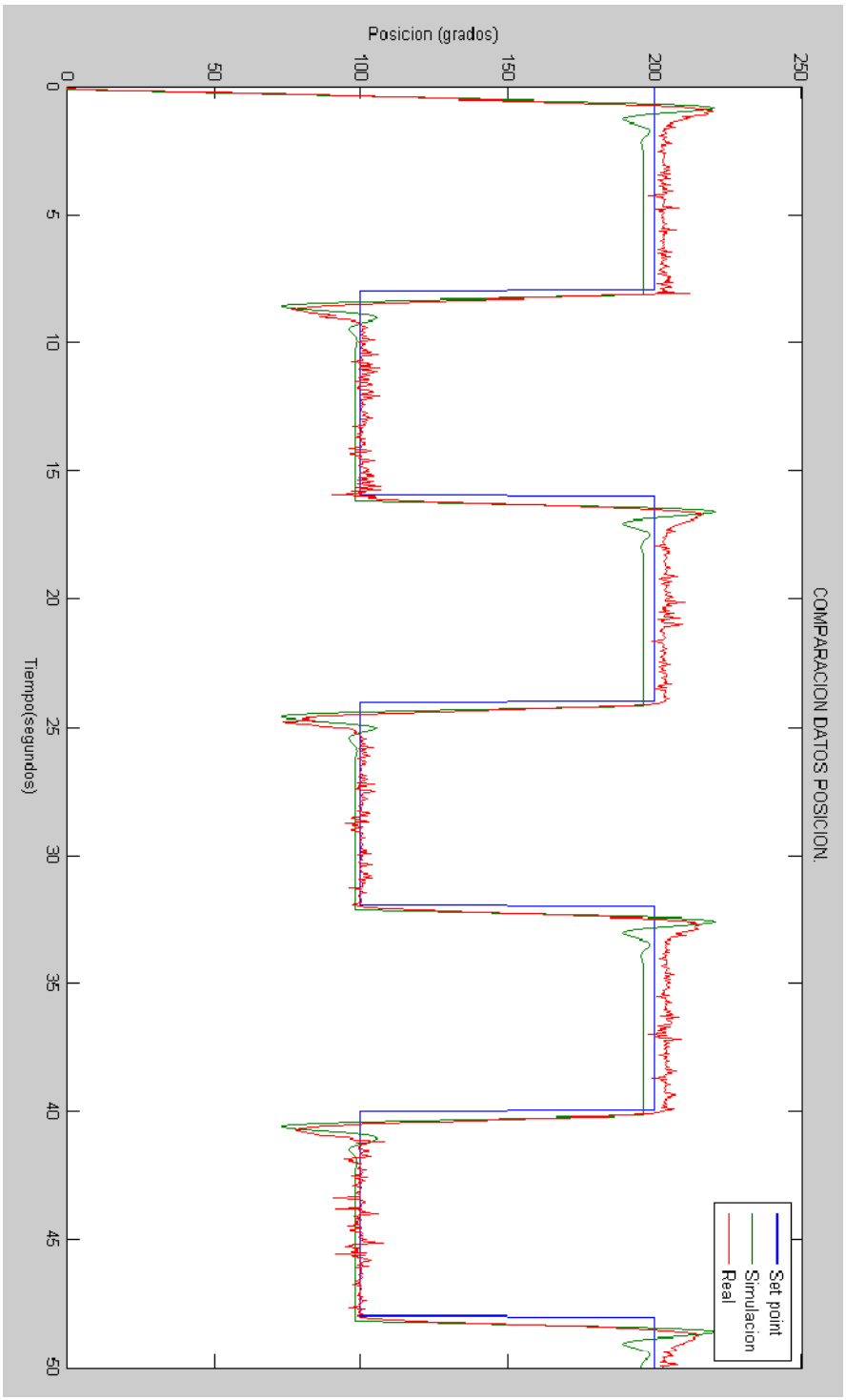


Figura 30. Resultados controlador posición con Kp=-0.6



.

10. RECOMENDACIONES.

Implementar una comunicación TCP/IP que permita trabajar en el modo externo de *Simulink,* agilizando la compilación e implementación del código generado con la RTWEC.

Realizar la automatización de una planta utilizando sistemas embebidos de control.

Utilizar el módulo *TOWER* de *Freescale* para implementar las prácticas de laboratorio que utilicen el módulo de servomecanismos de *Feedback.*

Utilizar las técnicas de prototipado rápido para proyectos diferentes al control de una variable o un sistema.

11. CONCLUSIONES.

Se diseñó e implementó controlador PID digital a un motor de corriente continua, el cual cuenta con comunicación con un computador permitiendo observar la respuesta del sistema. Adicionalmente el valor de referencia del sistema puede ser cambiado al igual que el valor de las diferentes variables.

Se utilizaron técnicas de prototipado rápido para el diseño del controlador digital PID con la simulación de este algoritmo y la posterior generación de código con la herramienta RTWEC.

Las herramientas de *Matlab* de identificación y control fueron fundamentales para el rápido diseño del algoritmo de control PID digital debido a su fácil utilización y a la eficacia de sus procesos.

Es importante anotar la implementación de un sistema operativo de tiempo real en el desarrollo en el proyecto asegurando con este una ejecución que cumpla con las restricciones de tiempo propias del sistema de control.

Se verifica la efectividad de las técnicas de prototipado rápido y la claridad y eficiencia del código generado por el RTWEC al comparar los valores obtenidos en simulación con los valores medidos experimentalmente del sistema de control.

12. BILBIOGRAFIA

[1] ASTROM, K. y HAGGLUND, T. Advanced PID Control. Pittsburgh: ISA, 2006.

[2] FLORIA, A. Recuperado el 22 de Noviembre de 2009 de: http://www.sidar.org/recur/desdi/traduc/es/visitable/nuevos/Rápido.htm.

[3] HEATH, S. Embedded Systems Design. Oxford: Newnes, 1997.

[4] HEATH, S. Embedded Systems Design. Oxford: Newnes, 1997.

[5] DICCIONARIO DE LA REAL ACADEMIA DE LA LENGUA ESPAÑOLA.

[6] PEREZ-CARPINTERO, J. MORERA, J. Conceptos de sistemas operativos. Madrid: Universidad Pontificia de Comillas, 2002

[7] RINCON, E. Programación de sistemas embebidos usando sistemas operativos de tiempo real [Diapositivas]. Bucaramanga, 2010.

[8] FREESCALE SEMICONDUCTOR. Freescale MQX Real-Time Operating system user's Guide. Denver: Freescale Semiconductor, 2010.

[9] FREESACALE SEMICONDUCTOR. Freescale MQX Real-Time Operating system user's Guide. Denver: Freescale Semiconductor, 2010.

[10] SOMMERVILLE, I. Ingeniería del Software. Madrid: Pearson Educación S.A, 2005

[11] GORDON, V.S. Bieman, J.M. Rapid prototyping: Lessons learned. En: IEEE Software. 1995.

[12] THE MATHWORKS INC. Real-Time Workshop Embedded Coder 5 Getting Started Guide. Natick: The Mathworks Inc, 2009.

[13] THE MATHWORKS INC. Real-Time Workshop Embedded Coder 5 Getting Started Guide. Natick: The Mathworks Inc, 2009.

[14] THE MATHWORKS INC. Real-Time Workshop Embedded Coder 5 Getting Started Guide. Natick: The Mathworks Inc, 2009.

[15] Recuperado el 10 de diciembre de 2010 de:

http://sc.leadix.com/DEVmonkey/files/06-24-09%20Freescale%20Tower%20Fig1.jpg

[16] Recuperado el 10 de enero de 2010 de: http://arvc.umh.es/ceaRsc/doc/feedback1.jpg

[17] FEEDBACK INSTRUMENTS LTD. Analogue Servo-Fundamentals Trainer. Sussex: Feedback instruments Ltd.