Universidad Autónoma de Querétaro

Facultad de Ingeniería

Ingeniería en Automatización

Control de movimiento para servomotores sobre un perfil parabólico de velocidad implementado en un FPGA

TESIS

Que como parte de los requisitos para obtener el grado de

Ingeniero en Automatización

Presenta:

Victor Ernesto Montalvo Garfias

Dirigido por:

Dr. Juvenal Rodríguez Reséndiz

SINODALES

Dr. Juvenal Rodríguez Reséndiz Presidente	Firma
Dr. Miguel Martinez Prado Secretario	Firma
Dr. Suresh Thenozhi Vocal	Firma
Dr. Edgar Alejandro Rivas Araiza Suplente	Firma
M. en C. Carlos Miguel Torres Hernández Suplente	Firma

Dr. Manuel Toledano Ayala
Director de la facultad de ingeniería

Querétaro, México
Abril 2020

Esta tesis va dedicada a mi familia, amigos y a todos aquellos que luchan por sobreponerse tras haber caído una y otra vez...

Agradecimientos

Aquí un pequeño agradecimiento a las personas que formaron parte de este camino tan lleno de historias que representó concluir mi primer trabajo de tesis.

Comienzo por mi famila, a quiénes considero una de las partes más importantes de mi vida; te doy gracias mamá por quererme incondicionalmente y haberme enseñado a siempre creer en mí para buscar oportunidades a cada paso que doy. Papá, a ti te doy gracias por enseñarme a valorar mi contexto, que aunque no fue el mejor, es privilegiado con respecto a muchas otras historias, también por ayudarme a entender que si quiero hacer algo es mi deber hacerlo sin buscar pretextos. Hermana gracias por estar conmigo en tantos de mis momentos más difíciles y por ser la única persona que conozco que comprende y no juzga a quiénes quiere.

A mis amistades más cercanas también les digo gracias; Ariel, Renata, Julio, Daniela, Juan Luis, Miguel, Dayan, Rebeca: con ustedes he vivido bastantes historias como para conocerlos y quererlos pero no las suficientes como para querer que se detengan nuestras aventuras.

A mis profesores, no sólo de licenciatura, les agradezco mucho, ya que tener el rol de la persona que, entre otras cosas, desafía tus habilidades no siempre es fácil pues en los peores momentos puede provocar resentimiento o desagrado; aún así, sin ustedes aprender no sería la experiencia llena de vida que representan sus clases.

A los jefes que he tenido también les agradezco mucho, pues han sido un contacto con la realidad fuera del aula muy importante, y de ustedes he aprendido mucho.

Es complicado agrupar a las personas en categorías como lo son familia, amigos y trabajo, ya que hay jefes y profesores que se convierten en amigos, y amigos que a su vez se convienten en famila; pero si algo tengo muy claro es que una gran parte de mí está dada por la suma de seres queridos que han tocado mi vida y ha llegado el momento de regresarles algo.

Abstract

The aim of this thesis is to study the designing and implementation of a motion control system for a direct current (DC) servomotor controlled using a parabolic velocity profile based proportional controller. Initially, the selection of an appropriate digital platform in which the system could be implemented is investigated, and a Field-Programmable Gate Array (FPGA) is chosen due to its high processing speed and its capability to run parallel processes successfully. To facilitate the digital implementation, the overall system is divided into the communication and control system units. First, the mathematical model of the proportional controller with parabolic and trapezoidal velocity profiles are derived, which is then discretized by means of numerical methods. Secondly, an UART communication interface is developed, which allows the user to monitor and modify the angular position, velocity and acceleration of the servomotor through a Graphical User Interface (GUI) developed on MATLAB. These designs are described using the Very high-speed integrated circuit Hardware Description Language (VHDL) for final implementation. Finally, various experimental studies are performed on a DC servomotor with inbuild encoder to evaluate the proposed strategies. First, the performance of the numerical method, used to discretize the system is compared with the original mathematical model using different sampling periods. Secondly, the position measurements obtained during the experiments under each velocity profile are compared with the positions proposed by the numerical method. Then, the performance of the proportional controller with and without using velocity profiles are evaluated experimentally. Finally, an energy consumption study is performed for each velocity profile which indicates that the parabolic velocity profile consumes less energy than the trapezoidal profile.

(**Keywords**: *parabolic velocity profile, servo-mechanism, UART communication, FPGA, DC motor, Proportional controller*)

Resumen

El principal propósito de esta tesis es el estudio del diseño e implementación de un sistema de control de movimiento para un servomotor de corriente directa controloado a través de un perfil parabólico de velocidad y un controlador proporcional. Inicialmente, la selección de una apropiada plataforma digital donde el sistema podría ser implementado es investigada; un *Field-Programmable Gate Array* (por sus siglas en inglés *FPGA*) es seleccionado debido a su alta velocidad de procesamiento y capacidad para trabajar procesos en paralelo de forma existosa. Con el objetivo de facilitar la implementación digital, el sistema completo es dividido en unidades de control y comunicación. Primero, el modelo matemático del controlador y los perfiles de velocidad parabólico y trapezidal son desarrollados y discretizados haciendo uso de métodos numéricos. Después una interfaz de comunicación de tipo *Universal Asynchronous Receiver-Transmitter* (por sus siglas en inglés *UART*) es desarrollada para permitir al usuario monitorear y modificar la posición, velocidad y aceleración angular del servomotor a través de una *Graphical User Interface* (por sus siglas en inglés *GUI*) desarrollada en *MATLAB*. Los diseños para el *FPGA* son descritos haciendo uso del lenguaje *Very high-speed integrated circuit Hardware Description Language* (por sus siglas en inglés *VHDL*) para su implementación final en una plataforma digital. Finalmente, diferentes estudios experimentales son desarrollados en un servotor de corriente directa con encoder incluído para evaluar las estrategias de control propuestas. Primeramente, el rendimiento del método numérico usado para discretizar el sistema es comparado con el modelo matemático original usando diferentes tiempos de muestreo. Después, se comparan las posiciones angulares logradas por los perfiles de velocidad durante el experimento con las posiciones propuestas por el método numérico. Siguiente a ello, el rendimiento del controlador proporcional con y sin perfiles de velocidad es evaluado experimentalmente. Finalmente, se realiza un estudio del consumo de energía para ambos perfiles inicando que el perfil parabólico de velocidad consume menos que el perfil trapezoidal.

(**Palabras clave**: *Perfil parabólico de velocidad, servomotor, comunicación UART, FPGA, motor de corriente directa, controlador proporcional*).

Nomenclatura

Cuadro 1: Variables y constantes del trabajo.

Símbolo	Descripción
t	Variable de tiempo, donde $t \in \mathbb{R}$
$\theta(t)$	Posición angular con respecto al tiempo.
θ_d	Posición angular deseada que busca igualar el controlador.
θ_f	Posición angular final.
$\omega(t)$	Velocidad angular con respecto al tiempo.
$\alpha(t)$	Aceleración angular con respecto al tiempo.
e	Error en el seguimiento de la posición, $(\theta_d - \theta)$.
T_x	Señal digital para la transmisión de datos.
R_x	Señal digital para la recepción de datos.
ω_{max}	Velocidad angular máxima.
$i(t)$	Corriente con respecto al tiempo.
$P(t)$	Potencia consumida durante el tiempo.
R	Resistencia eléctrica.
E	Energía que consume el motor durante una trayectoria angular.
T_s	Periodo de muestreo.
k	Multiplicador para cada instante tiempo, donde $k \in \mathbb{Z}$.
$\alpha(kT_s)$	Aceleración angular para cada instante de tiempo.
$\omega(kT_s)$	Velocidad angular para cada instante de tiempo.
$\theta(kT_s)$	Posición angular para cada instante de tiempo.
$u(t)$	Corrección del controlador a través del tiempo.
$u(k)$	Corrección del controlador en su forma discreta.
$e(t)$	Error en la posición a través del tiempo.
$e(k)$	Error en la posición en su forma discreta.
T	Tiempo en que la posición final será alcanzada.
a	Constante de aceleración.
u_k	Salida digital que el controlador entrega al DAC.
V_1	Voltaje que el DAC entrega al servo-amplificador.
V_2	Voltaje amplificado que el motor requiere para trabajar.
θ_m	Posición angular del motor que recibe el encoder.
CH_A	Señal digital A desfasada de B que del encoder.
CH_B	Señal digital B desfasada de A que del encoder.
m	Pendiente de una recta de primer grado.
$\tau_g(t)$	Torque de un motor.
τ_f	Torque del motor debido a la fricción.
J	Incercia del motor.
K_t	Constante que relaciona el torque del motor eléctrico y la corriente.
ERR	Registro digital del error.
SEL	Selector digital de los multiplexores.

Cuadro 2: Siglas y términos anglosajones.

Término	**Descripción**
DC	*Direct current.*
FPGA	*Field Programmable Gate Array.*
VHDL	*Very High Speed Integrated Circuit Hardware Description.*
UART	*Universal Asynchronous Receiver-Transmitter.*
CNC	*Computer Numerical Control.*
DSP	*Digital Signal Processor.*
SOC	*System on Chip.*
PC	*Personal Computer.*
GUI	*Graphical User Interface.*
PWM	*Pulse Width Modulation.*
NoC	*Network on Chip.*
RPM	*Revolutions per minute.*
DAC	*Digital to Analog Converter.*
PCB	*Printed Circuit Board.*
BPS	*Bits Per Second.*
IDE	*Integrated Development Environment.*

Índice general

Índice de figuras

Índice de cuadros

CAPÍTULO 1

Introducción

El desafío del hombre en torno a su comprensión por los cuerpos en movimiento, ha sido descubrir que cuando un objeto se desplaza de un punto a otro existen diferentes e infinitas formas de ejecutar este recorrido. Las características de este recorrido están dadas por diversos parámetros físicos entre los que se encuentran: la distancia que debe recorrerse, la trayectoria que el objeto seguirá y el tiempo que el recorrido durará. Es a raíz de la relación que existe entre la distancia recorrida y el tiempo, que aparecen conceptos como la velocidad, aceleración y, no tan popular pero muy importante: el *jerk*. Podría pensarse que el conocimiento de estos conceptos es meramente útil para propósitos académicos, en el desarrollo de tecnología de punta, o solo como indicadores para el usuario cotidiano de un automóvil (solo por dar un ejemplo); sin embargo, una función de velocidad con respecto al tiempo es capaz de explicar sucesos de la vida diaria como la razón de que un automóvil no logre frenar por completo antes de una colisión.

De manera intuitiva el ser humano ha aprendido a modular sus funciones de velocidad, por ejemplo presionando más o menos el acelerador dependiendo de qué tan cerca observa al próximo vehículo, pero cuando la tarea requiere un tiempo de respuesta preciso o una posición final confiable es necesario intervenir con modelos matemáticos que nos permitan predecir el comportamiento del desplazamiento para así alcanzar aplicaciones de alta calidad, seguridad y eficiencia. En [1] argumenta que en máquinas *CNC* el uso de funciones que representen la aceleración y des-aceleración del motor garantizan una trayectoria más suave y precisa; siendo así, como se introduce el modelo del perfil trapezoidal en su trabajo.

Los perfiles de velocidad trapezoidal, parabólico y *S-curve* son los más comúnmente usados en la industria y para cada uno existen características representativas. En [2] se establece que el perfil trapezoidal cuenta con los tiempos más cortos para el movimiento punto a punto, sin embargo, produce indeseables *jerks* mecánicos, la cual es una desventaja que el perfil parabólico no presenta.

Un servo-mecanismo es un sistema de partes mecánicas, eléctricas y electrónicas que mediante mando y regulación corrigen el valor de la variable para que se mantenga en el valor deseado; el presente trabajo pretende realizar este mando y regulación mediante un perfil parabólico de velocidad y un controlador proporcional.

1.1 Planteamiento del problema

En la industria una de las primeras condiciones que debe garantizarse a los trabajadores es la seguridad. Es debido a los procesos de manufactura con herramientas pesadas que en una máquina *CNC* por ejemplo, una posición final con sobrepaso o una velocidad mal regulada, además de daño para el producto y maquinaria podría provocar una lesión grave o fatal para el operador. Es así como un perfil de velocidad programado en una máquina de ejes coordenados nos permite garantizar, en conjunto con medidas de seguridad reglamentadas por protocolo, la integridad del equipo, el producto y aún más importante del operador.

La tecnología del automovilismo nos ha llevado a contar al día de hoy con la posibilidad de realizar viajes autónomos. Sin embargo, si realmente abordamos un problema alarmante en vialidades, es necesario hablar de accidentes por colisión entre vehículos o con elementos del entorno. Es por tal motivo que el desarrollo de sistemas genéricos y adaptables a cualquier vehículo debe ser el foco de atención para la investigación y desarrollo de nuevas tecnologías. Los perfiles de velocidad son uno de los elementos más importantes para que un sistema de navegación sea capaz de reaccionar a tiempo y poner a salvo al conductor del vehículo, y en donde, por ejemplo, el *jerk* en procesos industriales repercute en el consumo de energía. En un automóvil el comportamiento del *jerk* puede representar la diferencia entre el daño o integridad del conductor.

Se necesita diseñar e implementar un sistema que logre responder a una posición deseada en el menor tiempo posible y que esté ligado a la velocidad máxima que se desea para el motor y a la trayectoria de velocidad que menor consumo energético represente para el mecanismo. Es importante recalcar que, de acuerdo a la aplicación, la velocidad máxima no solo depende de las características del motor, sino también de las propiedades de la máquina; En una fresadora *CNC*, por ejemplo, las características de los herramentales de corte limitan la velocidad de avance en los ejes así como las propiedades del material a ser cortado.

1.2 Justificación

En [3] se menciona que el control de movimiento juega el papel más importante en numerosas aplicaciones de la industria, tales como: máquinas *CNC*, automatización, robótica y manufactura de semi-conductores, entre otros; las cuales son aplicaciones para las que existe la necesidad de una posición final de alta exactitud. Debido a los requerimientos mencionados anteriormente, la investigación sobre control de movimiento gira no solo en torno a diseñar nuevos modelos matemáticos que optimicen el proceso, sino además a desarrollar la tecnología necesaria para soportar los nuevos sistemas de control, ya que sin un dispositivo que sea capaz de procesar las señales con el periodo de muestreo requerido por el lazo de control, el proyecto es solo viable en la teoría pero no en la práctica.

De acuerdo con [4] el mejor medio para alcanzar un alto rendimiento en el control de servo-mecanismos es remover el lazo de servo-control implementado en un *DSP* o en un micro-controlador y trasladarlo a un lazo de control de alta velocidad en un *FPGA*). Por otra parte, la alta velocidad de procesamiento del *FPGA* es la razón principal para que en [5] se utilice para la optimización de encoders de baja resolución.También en [6] se utiliza un *FPGA* para sistemas de visión en robots autónomos donde la velocidad de procesamiento y portabilidad son factores clave. Así mismo en

[7] se utiliza un *FPGA* para el diseño e implementación de un sistema robótico capaz de esquivar obstáculos imprevistos en tiempo real. Otra caractéristica importante del *FPGA* es el paralalismo con que se pueden implementar sus funciones permitir así replicar múltiples veces circuitos que sirven para el mismo tipo de actuadores. Un ejemplo es [8] donde se implementa un sistema de control para motores a pasos bipolares que puede ser replicado para la n cantidad de motores que contenga su máquina. Debido a las características ya mencionadas, la mayoría de los componentes del presente trabajo serán desarrollados en el lenguaje para describir circuitos digitale *VHDL* e implementados en un *FPGA*.

Es de gran importancia mencionar que la eficiencia en la producción no es la única justificación para el desarrollo de este trabajo, sino además las aplicaciones de seguridad, puesto que una correcta modulación en la velocidad máxima del actuador permite prevenir colisiones o sobre-calentamiento en partes móviles. Cabe aclarar que no solo reducir tiempos en manufactura es el medio para aumentar los ingresos en la industria, la regulación del consumo de energía afecta directamente los costos de producción y es justamente esta característica en que el perfil parabólico destaca frente al perfil trapezoidal al reducir la tasa de cambio en la aceleración con respecto al tiempo.

1.3 Hipótesis

El perfil parabólico de velocidad consume como mínimo 10% menos potencia que el perfil trapezoidal para una misma trayectoria.

1.4 Objetivo

Diseñar un sistema de control para servomotores con base en un perfil parabólico de velocidad e implementarlo en un *FPGA* para así comparar el tiempo de respuesta y consumo de energía obtenidos con un perfil trapezoidal. Además se busca comparar la respuesta de un controlador proporcional al tener como entrada la trayectoria planeada por un perfil de velocidad y al no tenerla.

1.4.1 Objetivos específicos

Los objetivos específicos para este proyecto son los siguientes:

- Modelar y discretizar un lazo de control, que incluya el perfil de velocidad y el controlador, para su implementación en *FPGA*.
- Implementar un sistema de comunicación entre la *FPGA* y la interfaz de usuario.
- Comparar las trayectorias teóricas de los perfiles de velocidad trapezoidal y parabólico con las trayectorias generadas por el sistema de control en el *FPGA*.
- Comprobar que el consumo de potencia del perfil trapezoidal con respecto al parabólico disminuye midiendo y graficando la corriente consumida.
- Comparar la respuesta del controlador con y sin perfiles de velocidad.

1.5 Estructura de la tesis

La presente tesis se encuentra organizada de la siguiente manera:

- El Capítulo 2 detalla a través de referencias a investigadores y autores las bases teóricas y proyectos pasados relacionados con el presente trabajo.
- El Capítulo 3 describe una metodología que parte de los modelos matemáticos y formulan su interpretación discreta con el objetivo de implementar la solución en dispositivos digitales. Se ilustran además los principales bloques que representan el código *VHDL* que se cargará al *FPGA*.
- El Capítulo 4 presenta y analiza los resultados obtenidos en la experimentación.
- El Capítulo 5 establece las conclusiones del trabajo y se resuelve si los objetivos e hipótesis del trabajo fueron alcanzados y comprobados.

CAPÍTULO 2

Revisión de la literatura

2.1 Antecedentes

El estudio del arte del presente trabajo ha sido construido con la revisión de la literatura de diferentes autores nacionales e internacionales. En la Tabla 2.1 se prensentan las publicaciones que más influencia han tenido en este trabajo y se describe brevemente su aportación.

Cuadro 2.1: Principales referencias bibliográficas usadas en este trabajo.

Referencia	**Aportación**
[9] 2013	De este trabajo se retoma la arquitectura y estrategia de bloques implementados en el *FPGA* para conectar un perfil de velocidad con el controlador.Además sirve como base para el diagrama de bloques propuesto para el perfil trapezoidal.
[10] 2013	Las ecuaciones analíticas que permiten calcular los coeficientes de cada perfil de velocidad fueron desarrolladas a partir de este trabajo.
[11] 2013	La teoría de control que ilustra la retroalimentación de un sistema en lazo cerrado y las ecuaciones analíticas que permitieron desarrollar el controlador proporcional se obtuvieron de este libro.
[12] 2008	Este trabajo ayuda a reforzar la importancia que tiene la planeación de trayectorias en los sistemas de control de movimiento y además permite abrir el estudio a otros perfiles de velocidad para trabajos futuros.
[13] 2007	Los diagramas de bloques y código en *VHDL* de esta tesis parten de temas fundamentales de este libro como lo son máquinas de estado, multiplexores y contadores.
[8] 2009 y [14] 2020	Estas tres publicaciones fueron de gran ayuda para seleccionar la arquitectura del sistema.
[15] 2007	Gracias a este libro se desarrolló la mayor parte del código utilizado para la Interfaz gráfica y las funciones de cálculo y conversión en *MATLAB*.

Como se establece en [12] la planeación de trayectorias en máquinas y robots con múltiples actuadores comenzó con el diseño de levas mecánicas capaces de generar movimientos deseados y que fueran sincronizadas con acoplamientos mecánicos; hoy en día se apuesta por el diseño de levas electrónicas que permiten controlar mecanismos servo-asistidos de una manera más flexible y en donde la sincronización se consigue completamente con herramientas de software. Es así como se consigue evitar el repetitivo diseño y manufactura de levas mecánicas (difíciles y costosas de fabricar de acuerdo a [16]) para cada tipo de trayectoria, y por otro lado el movimiento se simplifica al enfocarse únicamente en el control del motor eléctrico para generar diferentes perfiles de movimiento.

En el estudio de movimiento propuesto para este trabajo las trayectorias son expresadas como funciones paramétricas del tiempo y el análisis de las mismas comienza con la propuesta de diferentes curvas de velocidad. Primero se analiza el comportamiento de un perfil parabólico de velocidad, dado por una función de segundo grado, y se finaliza con el estudio del perfil trapezoidal compuesto por una función seccionada de primer grado.

2.1.1 Generador del perfil parabólico de forma analítica

En [10] se describe cómo a partir de una curva de aceleración podemos obtener las funciones de velocidad y posición angular. Una trayectoria parabólica pertenece a una función cuadrática, y a su vez la derivada de dicha función es una recta de primer grado como se observa en la Figura 2.1.

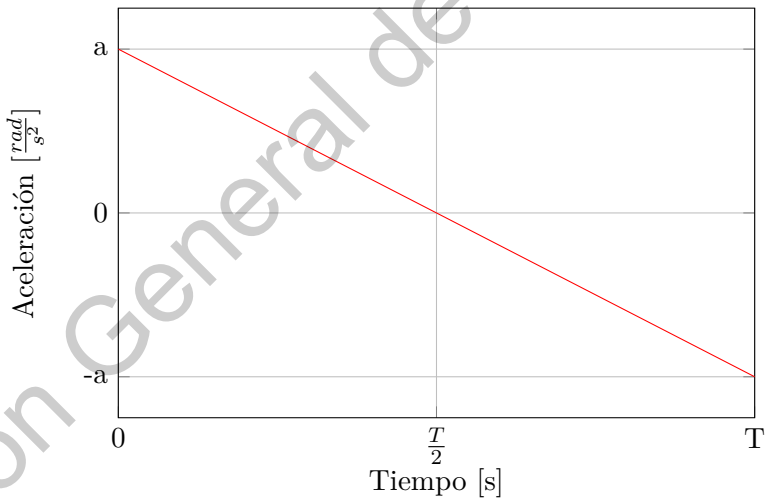


Figura 2.1: Recta de aceleración para el perfil parabólico, $\alpha(t) = \frac{-2a}{T}t + a$.

En el plano la ecuación punto-pendiente de la recta está dada por:

$$\alpha(t) = mt + b \tag{2.1}$$

donde $b = a$ y el valor de la pendiente se calcula como sigue:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{2.2}$$

al sustituir en (2.2) los valores del tiempo T y la constante de aceleración a:

$$m = \frac{-a - a}{T - 0} \tag{2.3}$$

se tiene la ecuación que describe la recta de aceleración.

$$\alpha(t) = \frac{-2a}{T}t + a \tag{2.4}$$

La ecuación de velocidad angular se obtiene al integrar la aceleración desde 0 a t y se ilustra en la Figura 2.2.

$$\omega(t) = \int_0^t \alpha(t)dt + \omega(0) \tag{2.5}$$

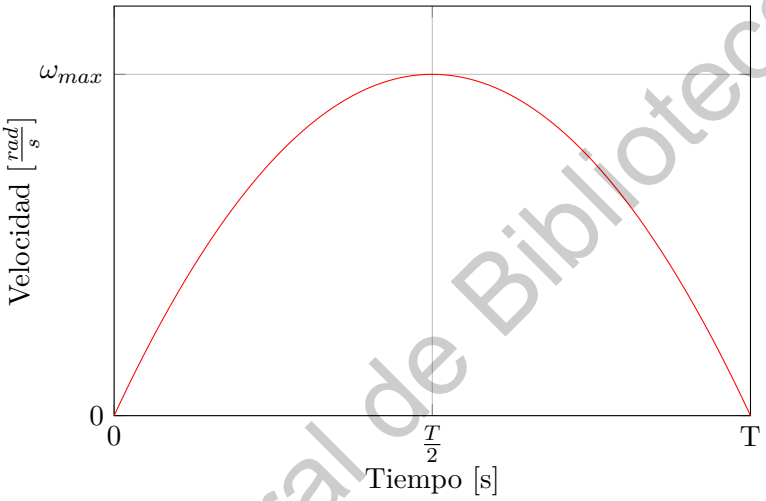


Figura 2.2: Curva de velocidad para el perfil parabólico, $\omega(t) = \frac{at^2}{T} + at$.

Resolviendo la integral de forma analítica se obtiene:

$$\omega(t) = \frac{-at^2}{T} + at, \quad \omega(0) = 0 \tag{2.6}$$

Sabemos que en el punto en el que la recta de aceleración intersecta el eje de las abscisas tenemos un valor máximo. Por tal motivo al evaluar la función de velocidad (2.6) en $\frac{T}{2}$ obtenemos la velocidad máxima:

$$\omega_{max} = \omega\left(\frac{T}{2}\right) \tag{2.7}$$

Resultando en el siguiente valor constante:

$$\omega_{max} = \frac{aT}{4} \tag{2.8}$$

Calcular la función de posición angular se logra integrando la función $\omega(t)$ desde 0 a t:

$$\theta(t) = \int_0^t \omega(t)dt + \theta(0) \tag{2.9}$$

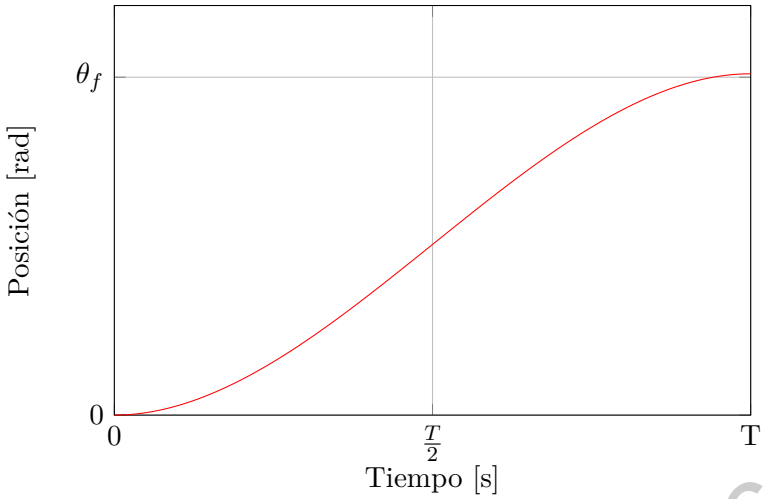


Figura 2.3: Curva de posición para el perfil parabólico, $\theta(t) = \frac{-at^3}{3T} + \frac{at^2}{2}$.

La curva de posición es de tercer grado y se encuentra dada por:

$$\theta(t) = \frac{-at^3}{3T} + \frac{at^2}{2}, \quad \theta(0) = 0 \tag{2.10}$$

La posición final se obtiene al evaluar $\theta(t)$ en T:

$$\theta_f = \theta(T) \tag{2.11}$$

y al resolver obtenemos:

$$\theta_f = \frac{2aT^2}{12} \tag{2.12}$$

Para fines de aplicación es conveniente definir la ecuación (2.12) en términos de la velocidad máxima:

$$\theta_f = \frac{2T}{3}\omega_{max} \tag{2.13}$$

La constante de aceleración puede ser despejada de (2.8):

$$a = \frac{4\omega_{max}}{T} \tag{2.14}$$

y T de la ecuación (2.13)

$$T = \frac{3\theta_f}{2\omega_{max}} \tag{2.15}$$

Con el desarrollo de estas ecuaciones se consigue que el tiempo que el perfil tarda en alcanzar la posición deseada y la aceleración máxima que el motor tendrá dependan únicamente de la posición final que el usuario asigne y la velocidad máxima que el motor pueda alcanzar. En la implementación del perfil parabólico (en su forma digital) los valores T, a y $m = \frac{-2a}{T}$ servirán como coeficientes para calcular de forma iterativa las diferentes posiciones para cada instante de tiempo a través de la integración numérica que se abordará en el siguiente capítulo.

2.1.2 Generador del perfil trapezoidal de forma analítica

Siendo que el perfil trapezoidal de velocidad es una función seccionada en 2 rectas de primer grado y un valor constante 0, la función que representa la aceleración de este perfil está dada también por una función seccionada en valores constantes como se muestra en la Figura 2.4.

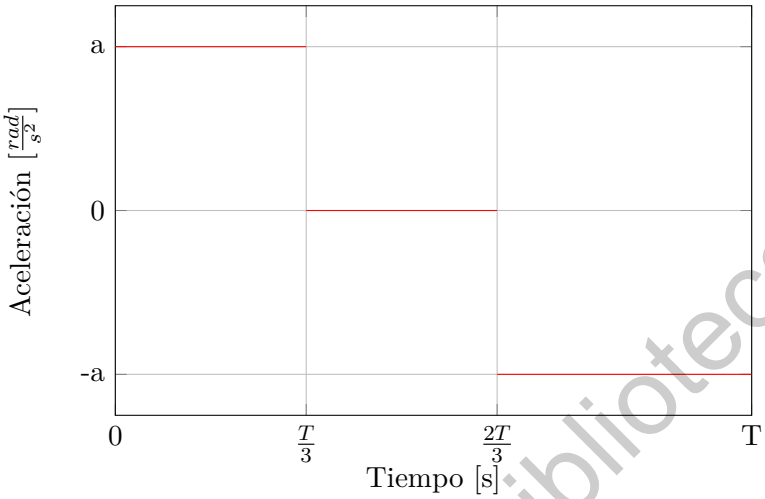


Figura 2.4: Función de aceleración $\alpha(t)$ para el perfil trapezoidal representada en (2.16).

Obtener las funciones que describen la velocidad y posición del perfil trapezoidal requiere de una integración por partes para los intervalos de tiempo establecidos en la función de aceleración, que como puede observarse en la Figura 2.4 están dados en la primera y segunda tercera parte del total del tiempo (pero podrían modificarse y experimentar con diferentes geometrías del trapecio). La función que representa la aceleración del perfil trapezoidal está dada de la siguiente manera:

$$\alpha(t) = \begin{cases} a & 0 \leq t \leq \frac{T}{3}, \\ 0 & \frac{T}{3} \leq t \leq \frac{2T}{3}, \\ -a & \frac{2T}{3} \leq t < T. \end{cases} \tag{2.16}$$

La primera recta de velocidad se obtiene integrando el valor de la aceleración para el intervalo $0 < t \leq \frac{T}{3}$:

$$\omega(t) = \int_0^t \alpha(t)dt + \omega(0) \tag{2.17}$$

El resultado es la primera recta creciente de velocidad:

$$\omega(t) = at, \quad \omega(0) = 0 \tag{2.18}$$

que al ser evaluada en $\frac{T}{3}$ representa la velocidad máxima del motor:

$$\omega_{max} = \omega\left(\frac{T}{3}\right) \tag{2.19}$$

y que da como resultado

$$\omega_{max} = \frac{aT}{3} \tag{2.20}$$

En el intervalo $\frac{T}{3} < t \leq \frac{2T}{3}$ se resuelve la integral:

$$\omega(t) = \int_{\frac{T}{3}}^{t} \alpha(t)dt + \omega\left(\frac{T}{3}\right) \tag{2.21}$$

y cuyo resultado es:

$$\omega(t) = \frac{aT}{3} \tag{2.22}$$

En el intervalo final $\frac{2T}{3} < t \leq T$ se resuelve la integral:

$$\omega(t) = \int_{\frac{2T}{3}}^{t} \alpha(t)dt + \omega\left(\frac{2T}{3}\right) \tag{2.23}$$

y se obtiene:

$$\omega(t) = -at + aT \tag{2.24}$$

Es así que la función seccionada de velocidad queda representada de la siguiente manera:

$$\omega(t) = \begin{cases} at & 0 \leq \quad t \quad \leq \frac{T}{3}, \\ \frac{aT}{3} & \frac{T}{3} \leq \quad t \quad \leq \frac{2T}{3}, \\ -at + aT & \frac{2T}{3} \leq \quad t \quad < T. \end{cases} \tag{2.25}$$

y cuya gráfica se encuentra representada de en la Figura 2.5

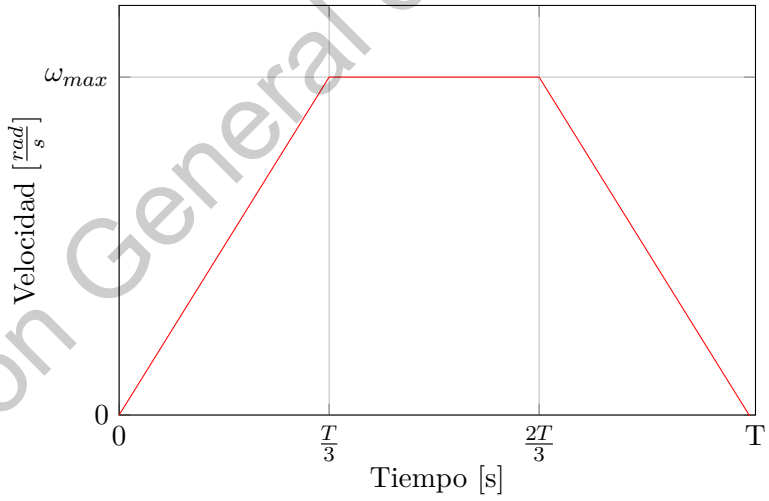


Figura 2.5: Función de velocidad $\omega(t)$ para el perfil trapezoidal representada en (2.25).

Obtener la función de posición angular requiere el mismo procedimiento de integración por partes utilizado para la función de velocidad. Para el intervalo $0 < t \leq \frac{T}{3}$:

$$\theta(t) = \int_{0}^{t} \omega(t)dt + \theta(0) \tag{2.26}$$

se resuelve:

$$\theta(t) = \frac{at^2}{2}, \quad \theta(0) = 0 \tag{2.27}$$

En el intervalo intermedio $\frac{T}{3} < t \leq \frac{2T}{3}$ se resuelve la integral:

$$\theta(t) = \int_{\frac{T}{3}}^{t} \omega(t)dt + \theta\left(\frac{T}{3}\right) \tag{2.28}$$

que da como resultado:

$$\theta(t) = \frac{aT}{3}t - \frac{aT^2}{18} \tag{2.29}$$

Para el último intervalo $\frac{2T}{3} < t \leq T$ la integral que se resuelve es:

$$\theta(t) = \int_{\frac{2T}{3}}^{t} \omega(s)dt + \theta\left(\frac{2T}{3}\right) \tag{2.30}$$

con solución:

$$\theta(t) = -\frac{at^2}{2}t + atT - \frac{5aT^2}{18} \tag{2.31}$$

Así la función seccionada de velocidad es representada de la siguiente manera:

$$\theta(t) = \begin{cases} \frac{at^2}{2} & 0 \leq t \leq \frac{T}{3}, \\ \frac{aT}{3}t - \frac{aT^2}{18} & \frac{T}{3} \leq t \leq \frac{2T}{3}, \\ -\frac{at^2}{2}t + atT - \frac{5aT^2}{18} & \frac{2T}{3} \leq t < T. \end{cases} \tag{2.32}$$

La gráfica que representa la curva de posición está dada en la Figura 2.6

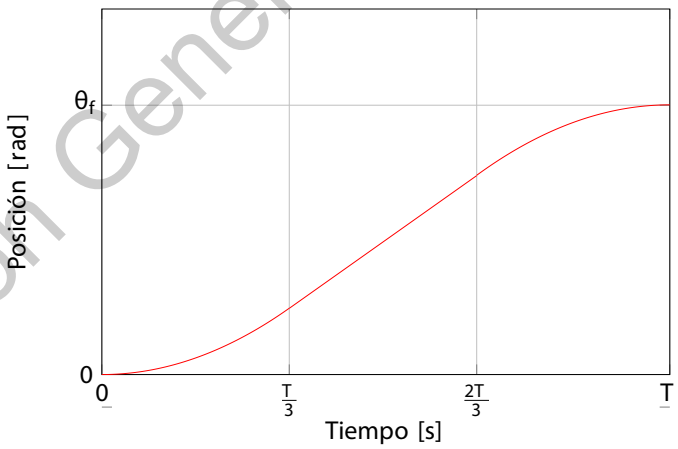


Figura 2.6: Función de posición $\theta(t)$ para el perfil trepezoidal representada en (2.32).

La posición final se obtiene al evaluar $\theta(t)$ en T:

$$\theta_f = \theta(T) \tag{2.33}$$

y al resolver obtenemos:

$$\theta_f = \frac{2aT^2}{9} \tag{2.34}$$

Es conveniente factorizar a la expresión (2.34) en el producto de $\frac{aT}{3}$ por $\frac{2T}{3}$ para así poder sustituir con la ecuación (2.20) y describir la posición final en términos de la velocidad máxima y el periodo de tiempo.

$$\theta_f = \frac{2T}{3}\omega_{max} \tag{2.35}$$

De la misma ecuación podemos despejar T:

$$T = \frac{3\theta_f}{2\omega_{max}} \tag{2.36}$$

y de la ecuación (2.20) despejamos la constante de aceleración:

$$a = \frac{3\omega_{max}}{T} \tag{2.37}$$

Así como con el perfil parabólico, los coeficientes obtenidos en función de la velocidad máxima y la posición final indicada por el usuario servirán para el cálculo iterativo en el sistema discreto del siguiente capítulo.

2.1.3 Comparación del consumo de energía entre perfiles

En [17] se describe el efecto que tiene el consumo de energía de máquinas eléctricas en el costo de producción en la industria, y establece que una correcta planeación de trayectorias representa menor desgaste al meacnismo y menor pérdida de energía. Por otro lado en [18] se propone que la conservación de la energía puede ser alacanzada encontrando mejores caminos en cortas y largas trayectorias, evitando frecuentes cambios de aceleración y sentido, y utilizando motores más eficientes. En este trabajo, las características que son usadas para comparar los perfiles de velocidad son: el tiempo en que alcanzan la posición deseada y la energía que consumen para seguir una trayectoria. En las funciones desarrolladas para cada perfil se observa que las ecuaciones (2.15) y (2.36) son iguales y representan el tiempo que cada perfil tardará en completar la trayectoria asignada; podemos concluir que bajo la misma velocidad máxima (dada por el motor) y con una misma posición final (asignada por el usuario) ambos perfiles completarán su trayectoria al mismo tiempo. Es entonces que la principal diferencia entre ambos perfiles es el consumo de energía.

Una de las principales ventajas del perfil parabólico sobre el trapezaoidal es el menor consumo de energía que el movimiento del servo-mecanismo requiere. Para demostrar esto es necesario asociar las funciones de aceleración de cada perfil con las ecuaciones de energía relativas a un servo-mecanismo. Se parte de la ley de rotación de Newton (equivalente rotacional de la segunda ley de Newton) [19]:

$$\tau_g(t) = J\frac{d\omega(t)}{dt} + \tau_f \tag{2.38}$$

donde $\tau_g(t)$ es la función que entrega el torque generado por el motor, J es la inercia y τ_f es el torque debido a la fricción. Por otra parte tenemos que el torque en un motor eléctrico está en función de la corriente $i(t)$ que circula por el mismo:

$$\tau_g(t) = K_t i(t) \tag{2.39}$$

donde K_t es la constante de proporcionalidad que relaciona la corriente con el par del motor. Podemos así despejar a la corriente como una función del torque:

$$i(t) = \frac{\tau_g(t)}{K_t} \tag{2.40}$$

De la teoría de circuitos eléctricos [20] sabemos que la potencia instantánea de un circuito está dada por:

$$P(t) = i^2(t)R \tag{2.41}$$

donde R es la resistencia eléctrica. La energía consumida en toda la trayectoría está representada por el área bajo la curva de la función de potencia, dicha área es calculada mediante una integral que va desde 0 a T.

$$E = \int_0^T P(t)dt \tag{2.42}$$

Sustituyendo la potencia con (2.41) y posteriormente la corriente con (2.40) tenemos:

$$E = \frac{R}{K_t^2}\int_0^T \tau_g^2 dt \tag{2.43}$$

Nuevamente reemplazando (2.38) en la función de torque:

$$E = \frac{R}{K_t^2}\int_0^T \left(J\frac{d\omega(t)}{dt} + \tau_f\right)^2 dt \tag{2.44}$$

Resolviendo los cuadrados:

$$E = \frac{R}{K_t^2}\int_0^T \left[J^2\left(\frac{d\omega(t)}{dt}\right)^2 + 2J\frac{d\omega(t)}{dt}\tau_f + \tau_f^2\right]dt \tag{2.45}$$

Es conveniente separar la ecuación de energía en un término para cada integral:

$$E = \frac{R}{K_t^2}\left[J^2\int_0^T\left(\frac{d\omega(t)}{dt}\right)^2 dt + 2J\tau_f\int_0^T \frac{d\omega(t)}{dt}dt + \tau_f^2\int_0^T dt\right] \tag{2.46}$$

Tenemos así a la Energía total consumida dada por E_1, E_2 y E_3 respectivamente. Encontramos que para determinar E_1 es necesario resolver la integral del cuadrado de la aceleración angular $\alpha(t)$. Es así como relacionamos el comportamiento de nuestros perfiles con el consumo energético.

$$E_1 = \frac{RJ^2}{K_t^2}\int_0^T \alpha^2(t)dt \tag{2.47}$$

Sabiendo que la velocidad angular de 0 a T es igual a 0 podemos establecer que E_2 es también 0.

$$E_2 = \frac{2RJ\tau_f}{K_t^2}\int_0^T \alpha^2(t)dt = 0 \tag{2.48}$$

El consumo E_3 es debido al torque generado por la fricción y solo será diferente en cada perfil dependiendo del tiempo transcurrido para alcanzar la posición final, pero debido a que el tiempo T es el mismo para el perfil trapezoidal y parabólico E_3 también lo será.

$$E_3 = \frac{R\tau_F^2}{K_t^2}\int_0^T dt = \frac{R\tau_f^2 T}{K_t} \tag{2.49}$$

Por lo anterior, el estudio en la diferencia de energía entre perfiles solo será abordado a través de E_1.

2.1.3.1 Consumo de energía del perfil trapezoidal.

En el caso del perfil trapezoidal sabemos que la posición final puede estar en función de la aceleración y el tiempo como se muestra en la ecuación: (2.34): Despejamos la constante de aceleración:

$$a = \frac{9\theta_f}{2T^2} \tag{2.50}$$

Sabemos que la función de la aceleración para el perfil trapezoidal es una función por partes. Sustituyendo (2.50) en (2.16) y elevando al cuadrado obtenemos:

$$\alpha^2(t) = \begin{cases} \frac{81\theta_f^2}{4T^2} & 0 \leq \quad t \quad \leq \frac{T}{3}, \\ 0 & \frac{T}{3} \leq \quad t \quad \leq \frac{2T}{3}, \\ \frac{81\theta_f^2}{4T^2} & \frac{2T}{3} \leq \quad t \quad < T. \end{cases} \tag{2.51}$$

Podemos expresar a E_1 como las integrales por partes de las constantes de aceleración:

$$E_1 = \frac{RJ^2}{K_t^2}\left(\int_0^{\frac{T}{3}} \frac{81\theta_f^2}{4T^4}dt + \int_{\frac{2T}{3}}^{T} \frac{81\theta_f^2}{4T^4}dt\right) \tag{2.52}$$

Separando las constantes de las integrales:

$$E_1 = \frac{81\theta_f^2 RJ^2}{4T^4K_t^2}\left(\int_0^{\frac{T}{3}} dt + \int_{\frac{2T}{3}}^{T} dt\right) \tag{2.53}$$

Resolviendo las integrales y simplificando:

$$E_1 = \frac{81\theta_f^2 RJ^2}{4T^4K_t^2}\left(\frac{T}{3} + \left(T - \frac{2T}{3}\right)\right) \tag{2.54}$$

Finalmente tenemos E_1 para el perfil trapezoidal.

$$E_1 = \frac{27\theta_f^2 RJ^2}{2T^3K_t^2} \tag{2.55}$$

2.1.3.2 Consumo de energía del perfil parabólico.

En el caso del perfil parabólico tenemos a la posición final dada por (2.12) y al despejar la constante de aceleración obtenemos:

$$a = \frac{6\theta_f}{T^2} \tag{2.56}$$

Al sustituir (2.56) en la aceleración angular dada por (2.4) obtenemos:

$$\alpha(t) = -\frac{12\theta_f}{T^3}t + \frac{6\theta_f}{T^2} \tag{2.57}$$

Se eleva al cuadrado $\alpha(t)$ para poder sustituirla en la ecuación de E_1

$$\alpha^2(t) = \left[\frac{-6\left(2\theta_f t - \theta_f T\right)}{T^3}\right]^2 \tag{2.58}$$

Resolviendo la potencia:

$$\alpha^2(t) = \frac{36\left(4\theta_f^2 t^2 - 4\theta_f^2 tT + \theta_f^2 T^2\right)}{T^6} \tag{2.59}$$

Sustituyendo en E_1

$$E_1 = \frac{36\theta_f^2 RJ^2}{T^6 K_t^2}\left(4\int_0^T t^2 dt - 4T\int_0^T t dt + T^2\int_0^T dt\right) \tag{2.60}$$

Simplificando las operaciones

$$E_1 = \frac{36\theta_f^2 RJ^2}{T^6 K_t^2}\left(\frac{4T^3}{3} - 2T^3 + T^3\right) \tag{2.61}$$

obtenemos E_1 para el perfil parabólico de velocidad:

$$E_1 = \frac{36\theta_f^2 RJ^2}{T^6 K_t^2}\left(\frac{T^3}{3}\right) = \frac{12\theta_f^2 RJ^2}{T^3 K_t^2} \tag{2.62}$$

Al comparar las E1 en ambos perfiles, (2.62) y (2.55), podemos observar que los mismos coeficientes están presentes en las dos ecuaciones de energía y además elevados a la misma potencia; la única diferencia es la constante numérica que los multiplica: para el perfil trapezoidal es 13.5 y para el parabólico es 12. Esta diferencia es una forma directa de demostrar que teóricamente el perfil trapezoidal consume un 12.5% más energía que el perfil parabólico.

2.1.4 Seguimiento de trayectoria con un controlador proporcional

En [21] se establece que las trayectorias dinámicas de vehículos autónomos requieren que el sistema siempre sea retroalimentado por los sensores que perciben el entorno del automóvil para así evitar colisiones. La respuesta a esta necesidad es crear un sistema en lazo cerrado y cuya retroalimentación sea recibida y procesada por un controlador.

En la introducción de [11] se explica el concepto general de un controlador usando el ejemplo de un cañón antiaéreo que debe ajustar su orientación para derribar un avión. Para conseguir esto, el cañón debe tener varios movimientos independientes que permitan ajustar su orientación (θ) dada por el movimiento de un motor eléctrico. Si se aplica un voltaje positivo al motor entonces este girará en sentido antihorario. Si el motor recibe un voltaje negativo este girará en el sentido horario y si no se le aplica un voltaje el motor eléctrico se detendrá. La posición del avión se determina mediante el uso de un radar y se usa este dato como el valor θ_d que se desea que la orientación θ del cañón alcance. Es decir, se desea conseguir que $\theta = \theta_d$ lo más rápido posible para una vez conseguido esto, disparar.

La relación del voltaje en el motor y el comportamiento antes descrito es la siguiente:

$$v = k_p(\theta_d - \theta) \tag{2.63}$$

donde k_p es una ganancia proporcional positiva. La operación es realizada con un equipo electrónico de baja potencia, por lo que es necesario utilizar un aplificador de potencia para proporcionar el voltaje necesario al motor eléctrico.

El diagrama de la Figura 2.7 ejemplifica el funcionamiento de este sistema:

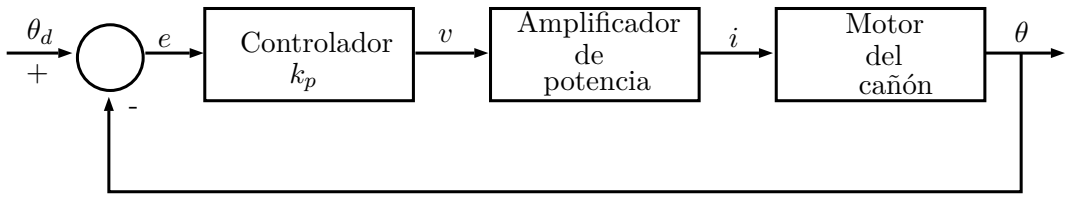


Figura 2.7: Sistema en lazo cerrado de un cañón anti-áereo.

Nótese que la construcción del controlador requiere que la posición del cañón θ (también conocida como la sálida) sea usada para generar el voltaje v (también conocido como la entrada) que se aplica al motor. La diferencia entre θ_d y θ es el error $e = \theta_d - \theta$ de la posición. Este hecho define los conceptos de retroalimentación y sistema de lazo cerrado. Una de las principales propuestas de este trabajo es que con el uso de perfiles de velocidad en un sistema digital, no es necesario implementar un controlador demasiado complejo para obtener un seguimiento de trayectoria preciso. Se realizarán tres principales experimentos para comprobar esta hipótesis: El primero consistirá en conectar la sálida del perfil parabólico a la entrada del controlador y observar el seguimiento de la trayectoria, el segundo es similar al primero pero usando el perfil trapezoidal, y finalmente el tercero consistirá en remover los perfiles de velocidad y conectar la posición deseada por el usuario directamente al controlador.

2.1.5 Implementación de la electrónica digital en un solo chip

En [14] se argumenta que para el desarrollo de sistemas de planeación y seguimiento de trayectorias que tengan buen rendimiento y alta velocidad de cálculo, es conveniente una implementación *SOC*. Dicha implementación consiste en integrar todo lo necesario para que el sistema funcione en un solo *chip* que pueda ser fácilmente replicado y que pueda trabajar en paralelo con otros módulos. Por otro lado en [8] se expone que el diseño de unidades de control *CNC* ha sido comunmente basado en microcontroladores o microporcesadores que requieren circuitos electrónicos muy sofisticados para ejecutar la función de control en la máquina *CNC*. Por otra parte, máquinas más modernas adoptan arquitecturas hibridas entre una *PC* y un *DSP*. En dichas arquitecturas la *PC* controla la operación de la *GUI* mientras que el *DSP* se encarga del computo de tareas de alta velocidad dedicadas al control de movimiento, así como de generar un *PWM* y finalmente la retroalimentación de un lazo cerrado de control. Desafortunadamente dichas arquitecturas presentan el inconveniente de no poder manipular el *hardware* fácilmente y no tener un periodo de muestreo tan rápido como el de otras tecnologías. Una arquitectura basada en un *FPGA* permite describir diferentes funcionalidades y replicarlas como bloques independientes que pueden ser ejecutados en paralelo y con una alta velocidad de procesamiento sin perder la implementación *SOC*. En [22] se exponen las principales ventajas de la arquitectura *NoC* (*Network on Chip*) entre las que destaca la re-configurabilidad de los sistemas embebidos *SOC* y que en parte puede ser lograda gracias a las características de reconfiguración que el *FPGA* tiene.

Para esta tesis se propone una arquitectura de trabajo en donde la operación de la *GUI* y el cálculo no iterativo de coeficientes será desarrollado en un *PC*; el resto de funcionalidad del sistema (perfiles de velocidad, controlador, display y comunicación UART), será implementado en un *FPGA* con una arquitectura *SOC*.

CAPÍTULO 3

Metodología

En este capítulo se documenta la descripción e implementación del sistema de control de movimiento. Se aborda la explicación del diagrama a bloques del sistema para posteriormente analizar de forma modular cada uno de los bloques que lo componen. Se añade además un listado de los recursos de *software* y *harware* necesarios para su implementación.

3.1 Especificación

Se muestra a continuación la lista de recursos de *software* y *sardware* que son requeridos para la implementación del sistema.

3.1.1 Software

- *Active HDL.*
- *Vivado Design Suite.*
- *EAGLE.*
- *Digilent Waveforms.*
- *MATLAB.*

3.1.2 Dispositivos eléctricos y electrónicos

- Tarjeta de desarrollo *Basys 3.*
- Driver de corriente *LMD18245.*
- Servomotor con reductor 131:1 y encoder de 8400 cuentas por revolución.
- Sensor de corriente *MAX471.*
- Computadora con programas instalados.

3.1.3 Equipo de laboratorio

- Osciloscopio *Analog Discovery*.
- Multímetro.
- Fuente de voltaje.
- Cautín.
- Componentes electrónicos consumibles (resistencias, capacitores, entre otros).

3.2 Diseño

3.2.1 Modelo del sistema

Como se describe en [11] modelar un sistema físico consiste en obtener y resolver ecuaciones que describan el comportamiento del sistema físico a través de sus variables. Para conseguirlo se modelan por separado los componentes del sistema y finalmente se conectan de acuerdo a su configuración y las leyes de la naturaleza que los rigen. En la Figura 3.1 se muestra el sistema propuesto para el presente trabajo:

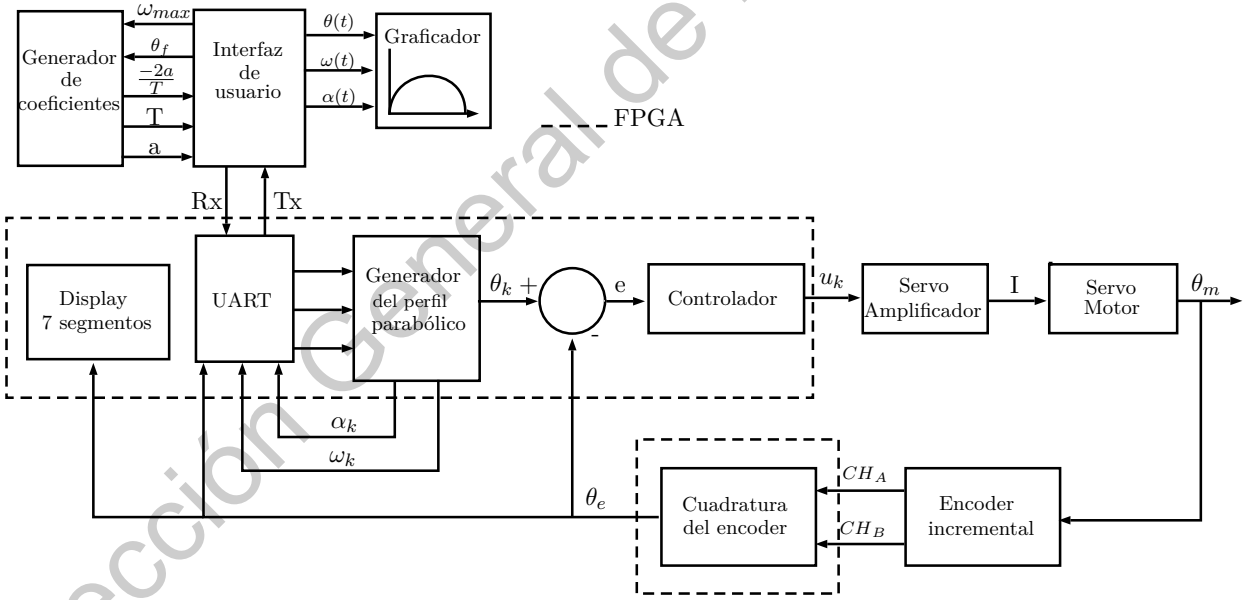


Figura 3.1: Modelo principal del sistema de control de movimiento.

El flujo de control comienza con la interfaz de usuario, en dónde se selecciona el tipo de perfil, la velocidad máxima y posición final deseada. Posteriormente el generador de coeficientes envía mediante un módulo de transmisión-recepción *UART* los coeficientes al *FPGA*. El generador del perfil calcula para cada instante de tiempo una porción de la posición final de acuerdo al perfil seleccionado. La diferencia entre la posición del generador del perfil y la posición actual del motor

es enviada al controlador quién a su vez multiplica esta diferencia por una ganancia K_p y envía este resultado saturado al driver del motor en formato digital y con un bit de signo. El driver induce en el motor una corriente proporcional al valor ingresado por el controlador y la hace fluir por el motor en el sentido que el bit de signo le especifique para seleccionar un sentido de giro. Lo posición actual del motor es medida con el encoder incremental del motor y procesada con un módulo de cuadratura descrito en el *FPGA*; el resutado del encoder es utilizado de tres formas en el sistema: como retroalimentación para el controlador, como valor decimal desplegado en un display de siete segmentos, y como vector de datos transmitido a la interfaz para graficar la trayectoria del motor.

3.3 Implementación

En esta sección se documenta el desarrollo e implementación de cada uno de los módulos que componen el sistema de control de movimiento.

3.3.1 Implementación del generador del perfil parabólico

3.3.1.1 Método numérico de integración del perfil parabólico

Para este trabajo, el cálculo de los perfiles de velocidad se ha desarrollado como el resultado de proponer una curva de aceleración que será integrada una vez para obtener la velocidad, y dos veces para obtener la posición. La implementación en el sistema digital ocurre bajo la misma lógica, pero en lugar de resolver mediante una integración analítica se utiliza una integración numérica para aproximar el valor deseado en cada instante de tiempo. Este cambio de método se debe a que un dispositivo basado en lógica digital, como lo es un *FPGA*, no está diseñado para procesar los algoritmos de la integración analítica (se puede lograr, pero requiere un desarrollo más complicado y un mayor consumo de recursos) y por otra parte, al tener un comportamiento modular y una muy alta velocidad de procesamiento (100 MHz), es preferible utilizar un método numérico para aproximar la solución; esto se debe a que los métodos numéricos, al ser de carácter iterativo, son en la mayoría de los casos operaciones muy sencillas, en dónde mientras más pequeño sea el instante de tiempo con que se opere mayor precisión se consigue en el resultado. Sabemos que la integral de una función representa el área bajo la curva de la misma. La integración rectangular se consigue dividiendo el área bajo la curva en un n número de rectángulos y sumando sus áreas:

$$\int_a^b f(x)dx \approx (b-a)f(a) \tag{3.1}$$

en donde $(b-a)$ es el tiempo de muestreo y representa a su vez la base del rectángulo; la altura está dada por $f(a)$ que es el valor de la función para ese instante de tiempo.
La ecuación discreta de aceleración se obtiene al sustituir en (2.4) el valor de t por kT_s, en donde k es un valor entero que incrementa en 1 para cada instante de tiempo y T_s es el periodo de muestreo $(b-a)$.

$$\alpha(kT_s) = \frac{-2a}{T}kT_s + a \tag{3.2}$$

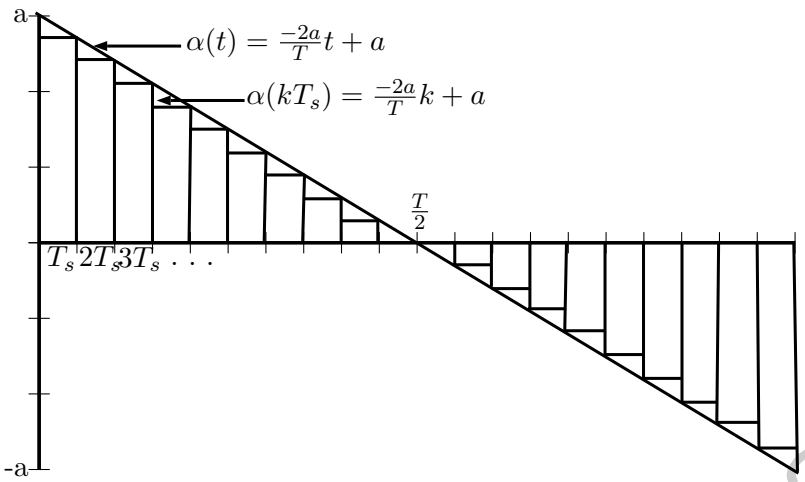


Figura 3.2: Integración rectangular de la aceleración del perfil parabólico.

Como se observa en la Figura 3.2, obtener la velocidad angular consiste en multiplicar T_s por el valor de la función en ese instante de tiempo $\alpha(kT_s)$, y para obtener la integral completa además es necesario sumar el resultado de los productos anteriores $\omega[(k-1)T_s]$:

$$\omega(kT_s) = \omega[(k-1)T_s] + \alpha(kT_s)T_s \tag{3.3}$$

La posición angular se obtiene con el mismo método:

$$\theta(kT_s) = \theta[(k-1)T_s] + \omega(kT_s)T_s \tag{3.4}$$

Con el propósito de simplificar las operaciones es conveniente seleccionar un periodo de muestreo unitario, en nuestro caso 1 ms, para que de esta forma el circuito diseñado en el *FPGA* no tenga que multiplicar la función por $(b-a)$. Las ecuaciones discretas quedarán expresadas como:

$$\alpha(kT_s) = \frac{-2a}{T}k + a \tag{3.5}$$

$$\omega(kT_s) = \omega(k-1) + \alpha(k) \tag{3.6}$$

$$\theta(kT_s) = \theta(k-1) + \omega(k) \tag{3.7}$$

Es importante mencionar que para que estas ecuaciones sean válidas la aceleración debe ser expresada en ms, es decir $\frac{rad}{ms^2}$, y la velocidad como $\frac{rad}{ms}$; la posición angular se expresa en radianes pero también podría ser expresada en grados o revoluciones.

3.3.1.2 Diseño del circuito lógico del perfil parabólico

El circuito implementado en el FPGA está descrito en la Figura 3.3. Las entradas de este componente son los coeficientes de a la curva de aceleración (obtenidos en el capítulo anterior), un reloj maestro CLK, una señal de arranque ST, y un reset maestro RST. El circuito se encarga de realizar las operaciones establecidas por la integración numérica y entregar a la sálida una posición angular para cada instante de tiempo. Los componentes que integran a este circuito son:

- Una base de tiempo: Establece la frecuencia de las operaciones en 1 ms.
- Un contador programable: Incrementa su valor cada milisegundo para establecer el instante de tiempo en que el circuito se encuentra.
- Una máquina de estados: Controla el flujo secuencial de las operaciones.
- Un multiplicador: En conjunto con un sumador realiza el cálculo de la aceleración.
- Tres sumadores: Realizan las operaciones iterativas para aceleración, velocidad y posción.
- Cuatro registros de carga: De acuerdo a la máquina de estados se habilitan para almacenar los datos para cada intante de tiempo en k y $(k-1)$.

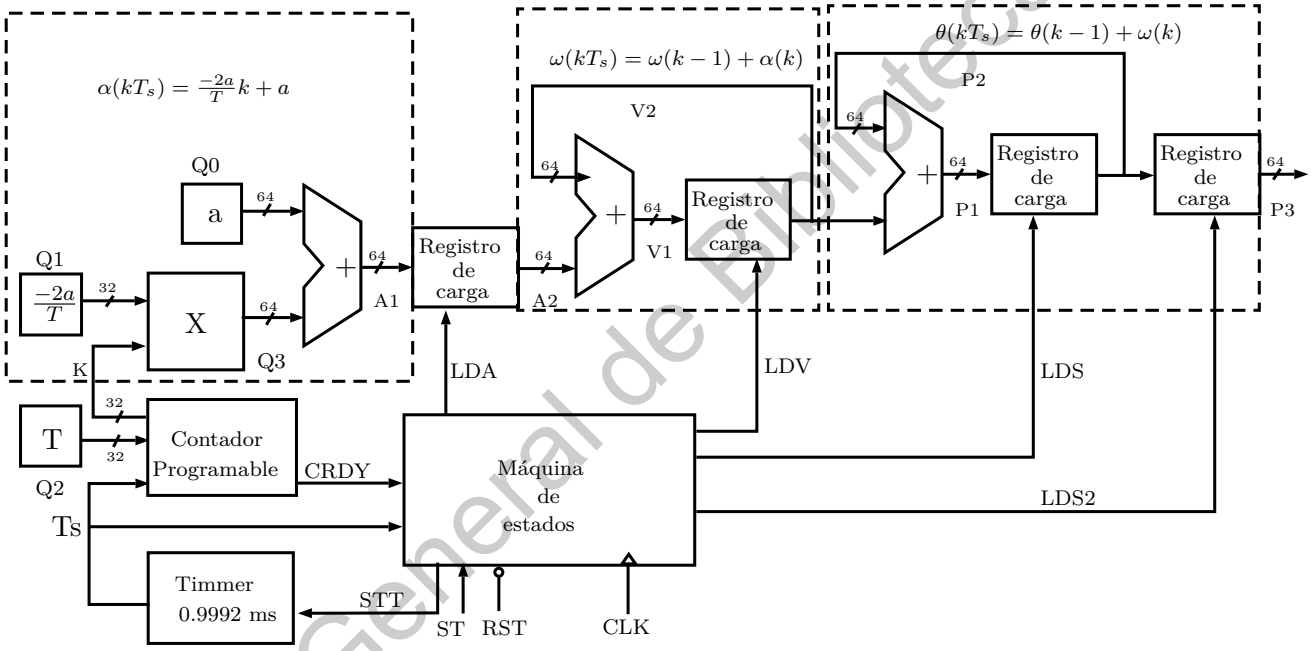


Figura 3.3: Diagrama de bloques del perfil parabólico discreto.

El flujo de las operaciones se encuentra dado por la máquina de estados como se observa en la Figura 3.4:

1. El estado S0 se encuentra a la espera de una señal de inicio ST dada por el usuario, una vez recibida se comienza el cálculo de los valores de aceleración, velocidad y posición.
2. En el estado S1 se envía la señal de inicio STT a la base de tiempo para que ésta cuente hasta 0.9992 ms y finalmente ponga la señal TS en alto para dar inicio a la carga de los registros.
3. Durante el estado S2 se habilita el primer registro de carga por medio de la señal LDA, este registro almacena el valor de la aceleración en ese instante de tiempo dado por la suma de Q0 más Q3.

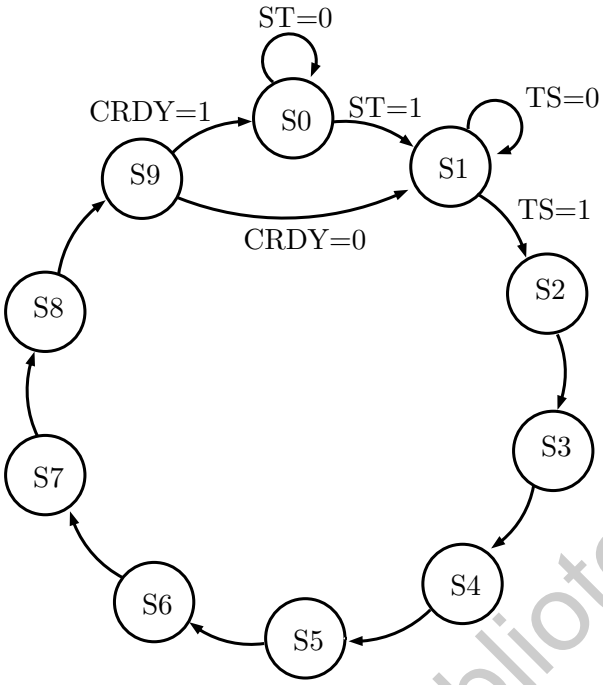


Figura 3.4: Máquina de estados del perfil parabólico discreto.

4. En los estados S3, S5, S7 y S9 las señales que habilitan los registros de carga son puestas en bajo para evitar registros secuenciados activos al mismo tiempo.

5. En el estado S4 se habilita el registro que captura la velocidad angular para ese instante de tiempo y ocurre lo mismo para la posición en el estado S6.

6. En el estado S8 habilita el registro de carga que captura la última posición angular.

7. Finalmente con el estado S9 se ponen todas las señales en bajo y en caso de que la señal de entrada CRDY esté en bajo se repite el ciclo desde el estado S1, en caso contrario se regresa al estado S0. La señal CRDY estará en alto una vez que el contador programable detecte que ya fueron calculados los valores de aceleración, velocidad y posición para todos los instantes de tiempo desde 0 hasta T.

3.3.2 Implementación del generador del perfil trapezoidal

3.3.2.1 Método numérico de integración del perfil trapezoidal

En la Figura 3.5 puede observarse la función seccionada que describe la aceleración del perfil trapezoidal y que, como en el perfil parabólico, será integrada por medio del método rectangular. Las ecuaciones (3.6) y (3.7) son válidas también para este proceso dado que se seguirá tomando 1 ms como periodo de muestreo.

Cuadro 3.1: Señales de la máquina de estados del perfil parabólico discreto.

ESTADO	**STT**	**LDA**	**LDV**	**LDS**	**LDS2**
S0	0	0	0	0	0
S1	1	0	0	0	0
S2	0	1	0	0	0
S3	0	0	0	0	0
S4	0	0	1	0	0
S5	0	0	0	0	0
S6	0	0	0	1	0
S7	0	0	0	0	0
S8	0	0	0	0	1
S9	0	0	0	0	0

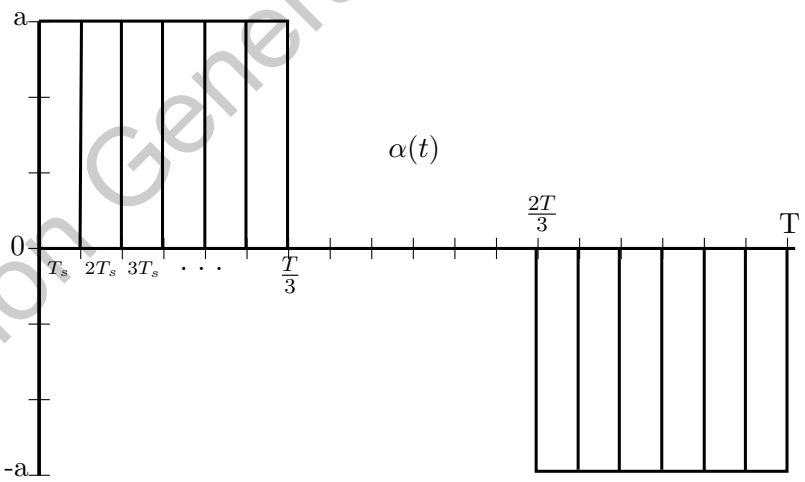


Figura 3.5: Integración rectangular de la aceleración del perfil trapezoidal.)

3.3.2.2 Diseño del circuito lógico del perfil trapezoidal

La Figura 3.6 ilustra el circuito lógico que se implementará en el *FPGA*. Las entradas del componente son los coeficientes de aceleración y periodo de tiempo, un reloj maestro CLK, una señal de arranque ST y un reset maestro RST. Con este circuito es posible realizar la integración rectangular que nos permite obtener la posición angular del perfil partiendo de su aceleración. Los componentes de circuito son:

1. Una base de tiempo: Establece el periodo de muestreo en 1 ms.
2. Dos contadores programables: El primero incrementa su valor cada milisegundo para establecer el instante de tiempo en que se encuentra el circuito; el segundo contador indica en qué tercera parte del trapecio se encuentra el periodo de tiempo y sirve como selector de las constantes de aceleración y tiempo que deberán usarse.
3. Una máquina de estados: Controla el flujo secuencial de las operaciones.
4. Dos sumadores: Realizan las operaciones iterativas para velocidad y posición.
5. Tres registros de carga: De acuerdo a la máquina de estados se habilitan para almacenar los datos para cada instante de tiempo en k y $(k-1)$.

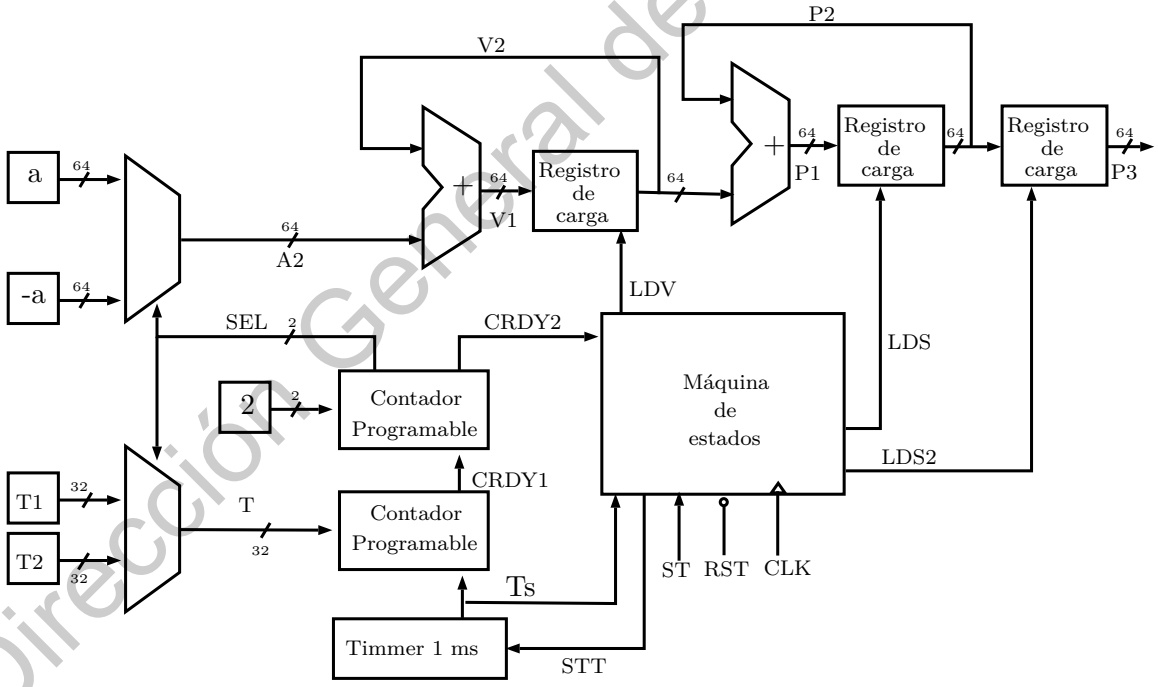


Figura 3.6: Diagrama de bloques del perfil trapezoidal discreto

El flujo de las operaciones se encuentra dado por la máquina de estados como se observa en la Figura 3.7:

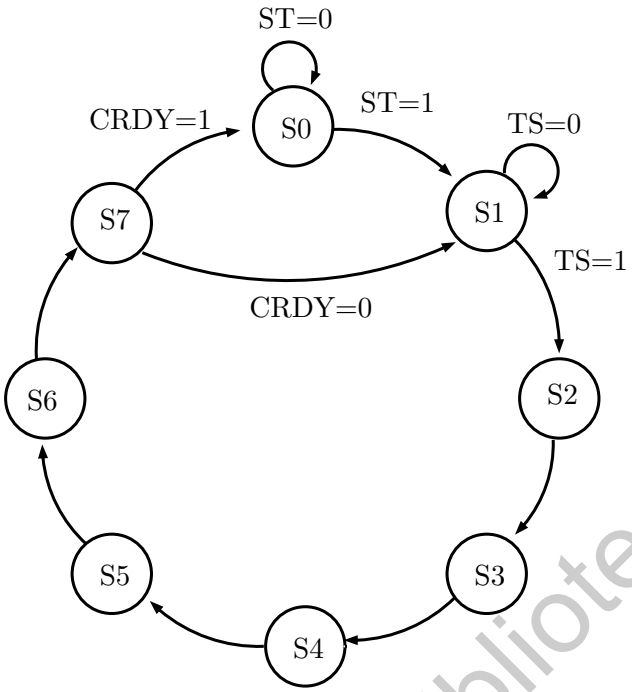


Figura 3.7: Máquina de estados del perfil trapezoidal.

Debe notarse que el comportamiento de la máquina de estados del perfil trapezoidal y el perfil parabólico es practicamente el mismo, el flujo de los estados determina de igual manera la activación y desactivación de los registros de carga que almacenan los valores de velocidad y posición. La única diferencia entre ambas máquinas es que para el perfil trapezoidal no existe un registro de aceleración que deba ser activado y en consecuencia consta de sólo siete estados.

Cuadro 3.2: Señales de la máquina de estados del perfil trapezoidal.

ESTADO	**STT**	**LDV**	**LDS**	**LDS2**
S0	0	0	0	0
S1	1	0	0	0
S2	0	1	0	0
S3	0	0	0	0
S4	0	0	1	0
S5	0	0	0	0
S6	0	0	0	1
S7	0	0	0	0

3.3.3 Implementación del controlador proporcional

Para conseguir implementar el controlador proporcional propuesto en el capítulo anterior es necesario primero discretizarlo y así añadirlo al diseño en el *FPGA*. Además es importante mencionar que así como en los perfiles de velocidad el tiempo de muestreo T_s será de 1 ms y de esta forma puede ser removido del producto por k. Partiendo de la ecuación (2.63) podríamos también expresar la diferencia $(\theta_d - \theta)$ como el error $e(t)$ reescribir la ecuación como $u(t) = k_p e(t)$ y expresar a la ecuación como una función discreta para obtener:

$$u(kT_s) = k_p e(k), \quad k \in \mathbb{Z} \tag{3.8}$$

Para calcular los valores de una función discreta es conveniente expresarla a través de operaciones recurrentes. Para ello se debe sumar el incremento en la función y el valor calculado en el instante de tiempo inmediato anterior como se muestra:

$$u(kT_s) = \Delta u(k) + u(k-1) \tag{3.9}$$

Podemos además asociar a $u[(k-1)T_s]$ con la ecuación (3.8) y escribirla como:

$$u[(k-1)T_s] = k_p e(k-1) \tag{3.10}$$

Así mismo la diferencia $\Delta u(k)$ puede despejarse de (3.9) como $u(k) - u(k-1)$ y al sustituir los valores de nuestra función asociados con el controlador obtenemos:

$$\Delta u(kT_s) = k_p e(k) - k_p e(k-1) \tag{3.11}$$

Es así como al sustituir $\Delta u(k)$ en la ecuación (3.9) expresamos el cálculo del controlador como una simple suma de términos que el *FPGA* puede realizar sin mayor complicación:

$$u(kT_s) = k_p e(k) - k_p e(k-1) + u(k-1) \tag{3.12}$$

De aquí en adelante expresaremos a k_p como el coeficiente q_0 y a $-k_p$ como q_1:

$$u(kT_s) = q_0 e(k) + q_1 e(k-1) + u(k-1) \tag{3.13}$$

La Figura 3.8 ilustra el circuito lógico del controlador. Las entradas del componente son: el error entregado por la diferencia entre la lectura del encoder y la posición deseada y los coeficientes q_0 y q_1 que representan la ganancia con que el controlador establece la corrección. Con este circuito se realizan las operaciones necesarias para recrear la ecuación (3.13) y así trazar la posición solicitada por el generador del perfil de velocidad. La sálida del circuito se encuentra saturada con la finalidad de acoplar el resultado a una acción de control de 4 bits para el *driver* del motor. Los componentes del cicruito son:

1. Seis registros de carga: Almacenan los valores de $e(k)$, $e(k-1)$, q_0, q_1, $u(k)$ y $u(k-1)$.

2. Un multiplexor: Asocia el valor del error al coeficiente correspondiente para que se multipliquen entre si de acuerdo a la ecuación (3.13).

3. Un multiplicador: Realiza el producto de los coeficientes con las funciones del error.

4. Un sumador: Suma todos los términos de la ecuación incluyendo a $u(k-1)$

5. Una máquina de estados: Controla el flujo secuencial de las operaciones.

6. Un saturador: Simplifica el resultado a una interfaz de 4 bits.

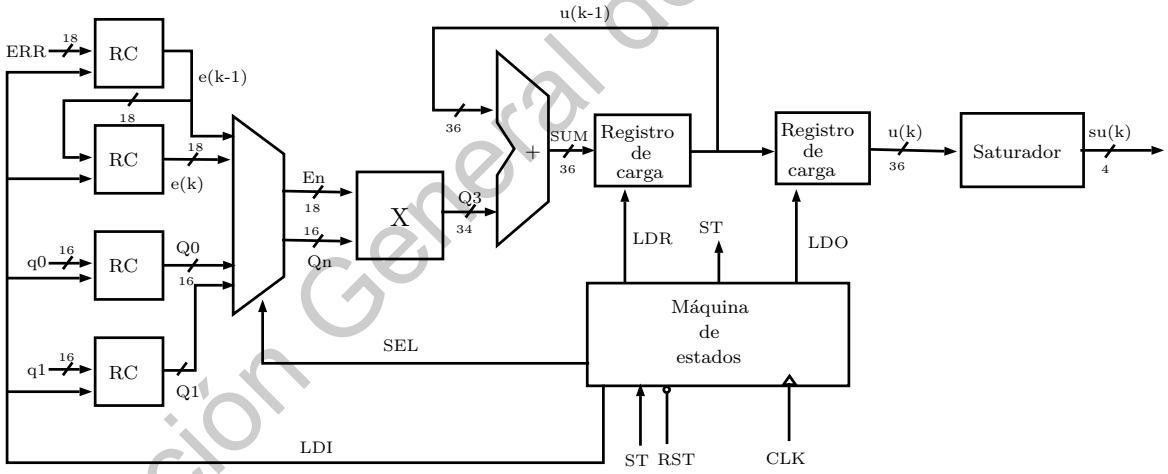


Figura 3.8: Controlador proporcional discreto.

El flujo de las operaciones se encuentra dado por la máquina de estados como se observa en la Figura 3.9:

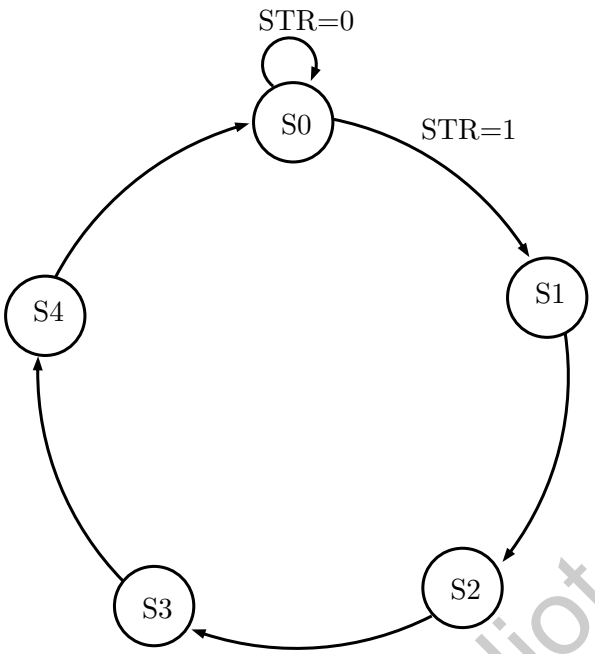


Figura 3.9: Máquina de estados del controlador proporcional.

1. El estado S0 se encuentra a la espera de una señal de inicio STR dada por el usuario, una vez recibida se comienza el cálculo de los términos que componen a la sálida $su(k)$. Además establece que el selector (SEL) del Multiplexor se encuentra en "00" lo que arroja una sálida de En = 0 y Qn = 0.

2. En el estado S1 se pone en alto la señal LDI que activa el registro del error (ERR) y de los coeficientes $e(k-1)$, q0 y q1.

3. Durante el estado S2 se habilita el registro de carga que captura el valor de $e(k)$ por q0, dado que en este estado el selector cambia a "01" y establece la sálida del Multiplexor en En = $e(k)$ y Qn = q0.

4. En el estado S3 se mantiene activo el registro de carga que captura la suma de $q0e(k)$ más la multiplicación de q1 por $e(k-1)$ debido a que el selector (SEL) ahora establece la sálida del Multiplexor en En = $e(k-1)$ y Qn = q1.

5. Con el estado S4 se pone en alto la señal LDO que activa el registro de carga que captura la suma de los términos que componen a la sálida u(k).

6. Finalmente la palabra digital u(k) es recortada por el saturador a un valor de 4 bits de resolución y 1 bit adicional para indicar el signo al driver del motor.

3.3.4 Implementación del servo-amplificador

Las principales características del servomotor utilizado para los experimentos de este trabajo se mencionan en la Tabla 3.4.

Cuadro 3.3: Señales de la máquina de estados del controlador proporcional.

ESTADO	**LDI**	**LDR**	**LDO**	**RDY**	**SEL**
S0	0	0	0	1	00
S1	1	0	0	0	00
S2	0	1	0	0	01
S3	0	1	0	0	10
S4	0	0	1	0	00

Cuadro 3.4: Principales características del motor utilizado para los experimentos.

Voltaje de operación	12V
Reducción	131:1
Máxima velocidad	80 RPM (Revoluciones Por Minuto)
Cuentas de encoder	8400 por revolución
Máxima corriente consumida	5A

De la misma forma los principales rangos de operación del *driver* LMD18245 se muestran en la Tabla 3.5.

Cuadro 3.5: Principales rangos de operación del LMD18245.

Temperatura	-40°C a +125°C
Voltaje de alimentación	+12V a +55V
Voltaje del *DAC*	0V a +5V
Máxima corriente continua	3A
Máximo pico de corriente	6A

En la Figura 3.10 se muestra la arquitectura de hardware para este trabajo. Este diagrama está principalmente enfocado en describir la funcionalidad del *driver* LMD18245 y su diagrama de conexiones con el motor, el *FPGA* y la instrumentación ncesaria para hacerlo funcionar.

El acoplamiento del motor de *DC* con el controlador digital será logrado a través del *driver* LMD18245 cuyas especificaciones se encuentran en las referencias de este documento. La función principal del *driver* consiste en controlar el suministro de corriente que el motor de *DC* recibe en cada instante de tiempo y que modifica la velocidad del mismo. En la Tabla 3.6 se muestra el diagrama de pines más relevantes para este trabajo.

La corriente que el *driver* suministra al servo-motor está dada por el voltaje de referencia V_{REF} que a su vez es limitado de forma proporcional por el valor decimal de la palabra de 4 bits formada por los pines M4 a M1, en dónde M4 es el bit más significativo. Esta referencia es sensada

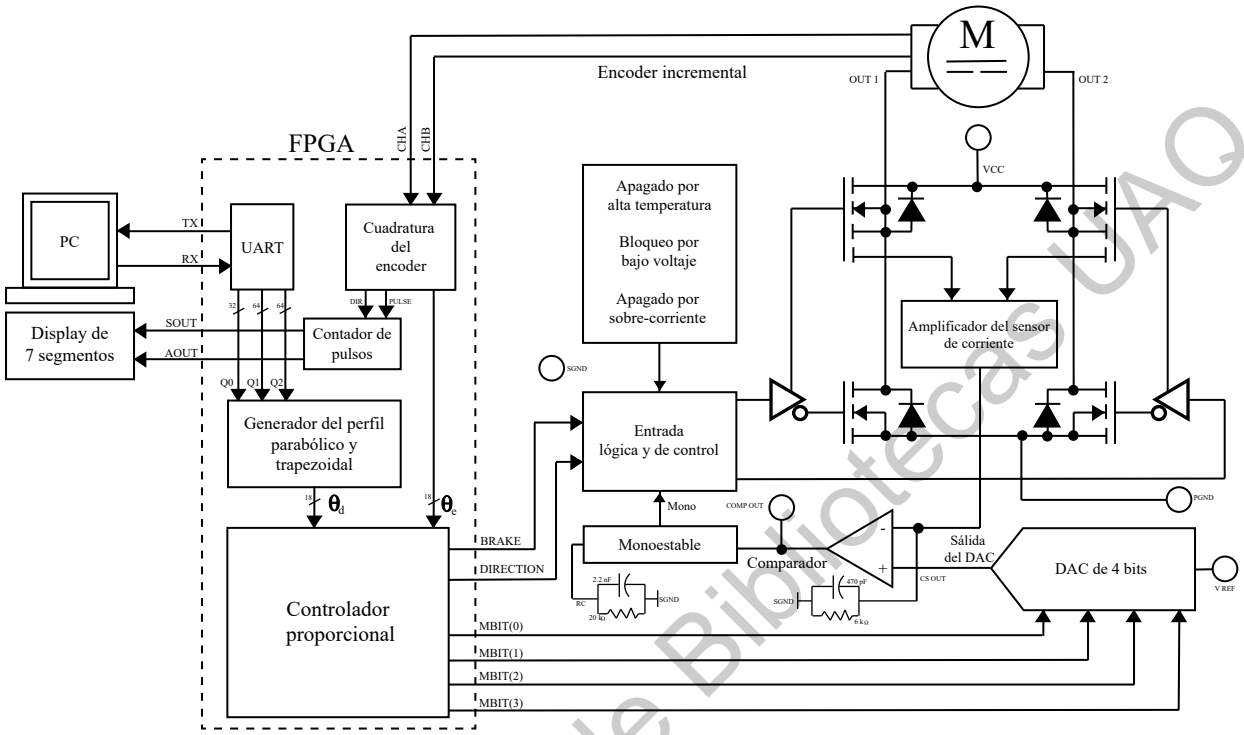


Figura 3.10: Arquitectura modular de hardware.

Cuadro 3.6: Diagrama de pines relevantes del LMD18245.

Pin	**Señal**	**Tipo**	**Descripción**
4,6,7,8	M4-M1	Entrada	Entradas digitales del *DAC*.
10	*BRAKE*	Entrada	Freno lógico del Driver
11	*DIRECTION*	Entrada	Establece el sentido de giro del motor (1 -> Horario)
14	DAC_{REF}	Entrada	Toma el voltaje de referencia del *DAC* en un rango de 0V a 5V
9	*VCC*	Entrada	Suministro de la potencia que moverá al motor
1	*OUT1*	Salida	Nodo de salida de la primera mitad del puente H
15	*OUT2*	Salida	Nodo de salida de la segunda mitad del puente H
13	*CS OUT*	Salida	Salida del aplificador del sensor de corriente

y amplificada por el pin *CS OUT* en combinación con la resistencia R_s. Los valores de corriente que se suministran al motor con $V_{REF} = 5V$ se encuentran en la Tabla 3.7 .

Cuadro 3.7: Valores de corriente suministrada al motor para cada valor decimal del *DAC*.

D	$R_s = 18.75k\omega$	$R_s = 9.375k\omega$	$R_s = 6.250k\omega$
0	0.00 A	0.00 A	0.00 A
1	0.07 A	0.13 A	0.20 A
2	0.13 A	0.27 A	0.40 A
3	0.20 A	0.40 A	0.60 A
4	0.27 A	0.53 A	0.80 A
5	0.33 A	0.67 A	1.00 A
6	0.40 A	0.80 A	1.20 A
7	0.47 A	0.93 A	1.40 A
8	0.53 A	1.07 A	1.60 A
9	0.60 A	1.20 A	1.80 A
10	0.67 A	1.33 A	2.00 A
11	0.73 A	1.47 A	2.20 A
12	0.80 A	1.60 A	2.40 A
13	0.87 A	1.73 A	2.60 A
14	0.93 A	1.87 A	2.80 A
15	1.00 A	2.00 A	3.00 A

Las referencias del LMD18245 indican 3 valores diferentes de resistencia R_s que proporcionan rangos diferentes de corriente máxima y mínima suministrada. La fórmula que relaciona estos valores es: $(V_{REF} \times D/16)/((250 \times 10^{-6}) \times R_s)$, en donde a valores más grandes de resistencia son menores los valores máximos de corriente y D es el valor decimal que regula la cantidad de corriente suministrada al motor. Con el objetivo de mantener siempre la posibilidad de alcanzar la corriente máxima del *driver* cuando cargas o motores más pesados lo demanden se decide dejar fija la resistencia de 6.250 kΩ.

Debido a que la corriente necesaria para vencer la resistencia del motor y provocar movimiento a la mínima velocidad es de 75 mA y alcanzar la velocidad nominal (sin carga) consume 150 mA, el voltaje de referencia puede ser menor a 5V. Se ajusta el voltaje a 1.8 V para aprovechar al máximo la resolución de 4 bits que el DAC proporciona y así tener cambios de velocidad más precisos. En caso de que al motor se le añadiera una carga o se reemplazara por un motor que demande más corriente solo es necesario reajustar el voltaje de referencia para aumentar el rango de corriente suministrada.

3.3.4.1 Instrumentación del *driver* y desarrollo del socket

Con la finalidad de reducir la cantidad de cables que existen entre el sistema de control embebido en el *FPGA* y el *driver* del motor, se decide fabricar un *PCB* que contenga toda la instrumentación necesaria para hacer funcionar el LMD18245 y al mismo tiempo conecte las salidas y

entradas del *FPGA* necesarias para hacer funcionar el sistema. El esquemático con los componentes de instrumentación que la tarjeta requiere para funcionar se muestra en la Figura 3.11.

Los periféricos y componentes de la tarjeta se indican en la Tabla 3.8 .

Cuadro 3.8: Valores de corriente suministrada al motor para cada valor decimal del *DAC*.

Componente	**Función**
LMD18245	Servo Amplificador
S1	Selector de bits manual M4-M1 cuando no se utilizan las señales del *FPGA*
R1	Resistencia *Pull down* para la selección del pin *BRAKE*
R2	Resistencia *Pull down* para la selección del pin *DIRECTION*
R3 a R6 A	Resistencias *Pull down* del Switch S1 para selección de bits manual M4-M1
MTR-1,2	Sálidas del LMD18245 a las terminales del motor
R7, C4	Resistencia R_s y capacitor para el sensado de corriente en *CS OUT*
R8, C5	Resistencia y capacitor para el pulso monoestable cada 1.1(RC) segundos
C1	Capacitor para suavizar *VCC* del Driver
PWR	Conector de *VCC*.
AMS1117	Regulador de 5V para las entradas y salidas digitales
JP1,2,4,5	Tiras de pines que se conectan a las IO del *FPGA*.
JP1,2,4,5	Tiras de pines que se conectan a las IO del *FPGA*.
JP8	Tira de pines destinadas para el voltaje de referencia del *driver*.
JP6	Tira de pines destinada para conectar el motor y su encoder.
JP7	Tira de pines designada al BCD de 7 segmentos que indica el valor del encoder.
JP10	Conector de alimentación del regulador AMS1117 de 5V.

Las dimensiones de la tarjeta son muy similares a las del *FPGA* con la intención de ensamblar de arriba hacia abajo y que el sistema de control no utilice demasiado espacio. La distribución de pistas y componentes se muestra en la Figura 3.12.

El resultado de la tarjeta ensamblada al *FPGA* se muestra en la Figura 3.13. Se ha decidido utilizar algunos componentes de montaje superficial así como recubrimiento de mascarilla anti-soldante con la finalidad de compactar la distribución de compomentes y aumentar su resistencia al desgaste.

3.3.5 Implementación de la Interfaz del Usuario

La interfaz del sistema de control tiene 3 funciones principales:

1. Permitir al usuario simular 4 diferentes perfiles de velocidad en donde podrá modificar: la velocidad máxima del motor (dada por sus especificaciones), la posición angular final y el tiempo de muestreo. Para cada valor ingresado por el usuario el simulador calcula un tiempo final T y una constante de aceleración para cada perfil que permiten graficar las curvas de aceleración, velocidad y posición durante el desplazamiento teórico.

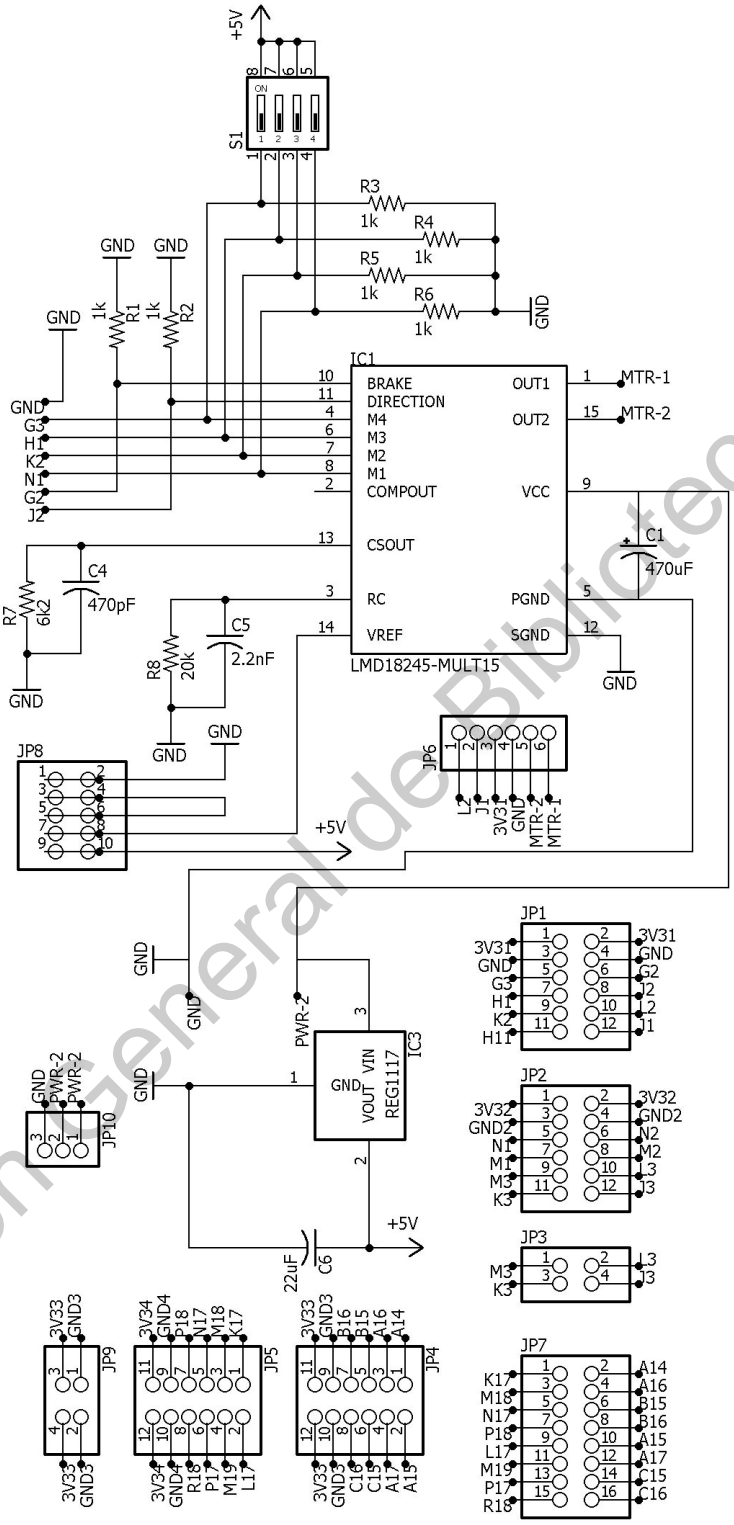


Figura 3.11: Esquemático del socket del *driver*.

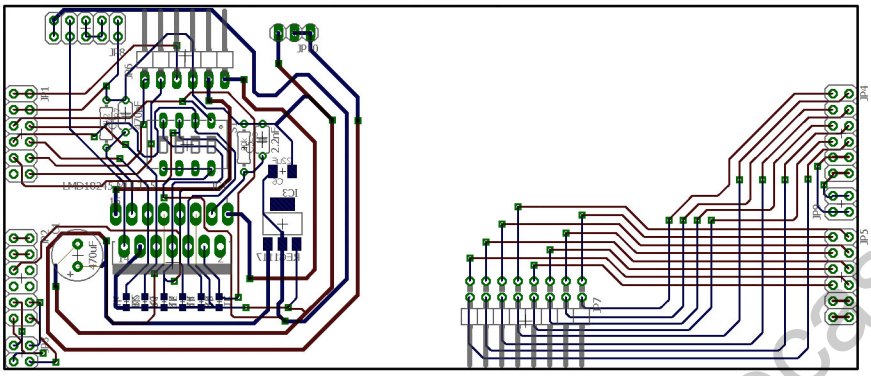


Figura 3.12: Distribución de pistas y componentes del socket del driver.

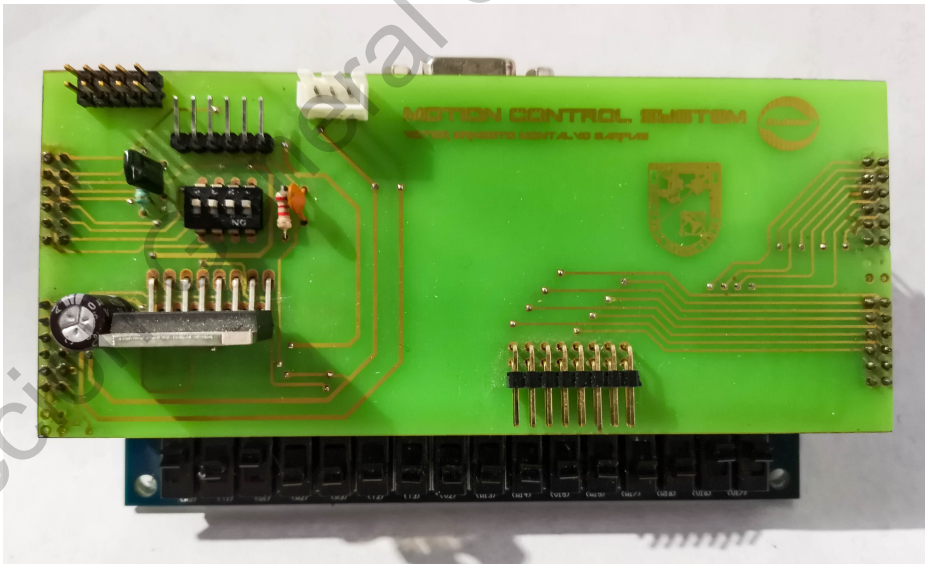


Figura 3.13: Resultado final del ensamble electrónico del *driver*.

2. Una de las desventajas del *FPGA* es lo complejo que es la descripción de hardware necesaria para realizar divisiones; por tal motivo la interfaz calcula los coeficientes iniciales que el perfil descrito en el *FPGA* utiliza para realizar las multiplicaciones y sumas iterativas para cada posición angular. A través del protocolo RS232 la interfaz envía dichos coeficientes al *FPGA* en paquetes de 8 bits que conforman parabras de hasta 32 bits en formato de punto fijo. En la Tabla 3.9 se muestran los formatos que cada perfil con sus respectivos coeficientes.

3. El propósito final de la interfaz es poder comparar los resultados reales con los calculados a través de los métodos numéricos en el simulador de la interfaz. Por tal motivo durante cada nueva posición angular ingresada, la interfaz recibe el vector de posiciones instantaneas que el encoder del motor cuenta para cada instante de tiempo; este vector puede ser comparado con el vector generado por el simulador y así probar la precisión del sistema de control.

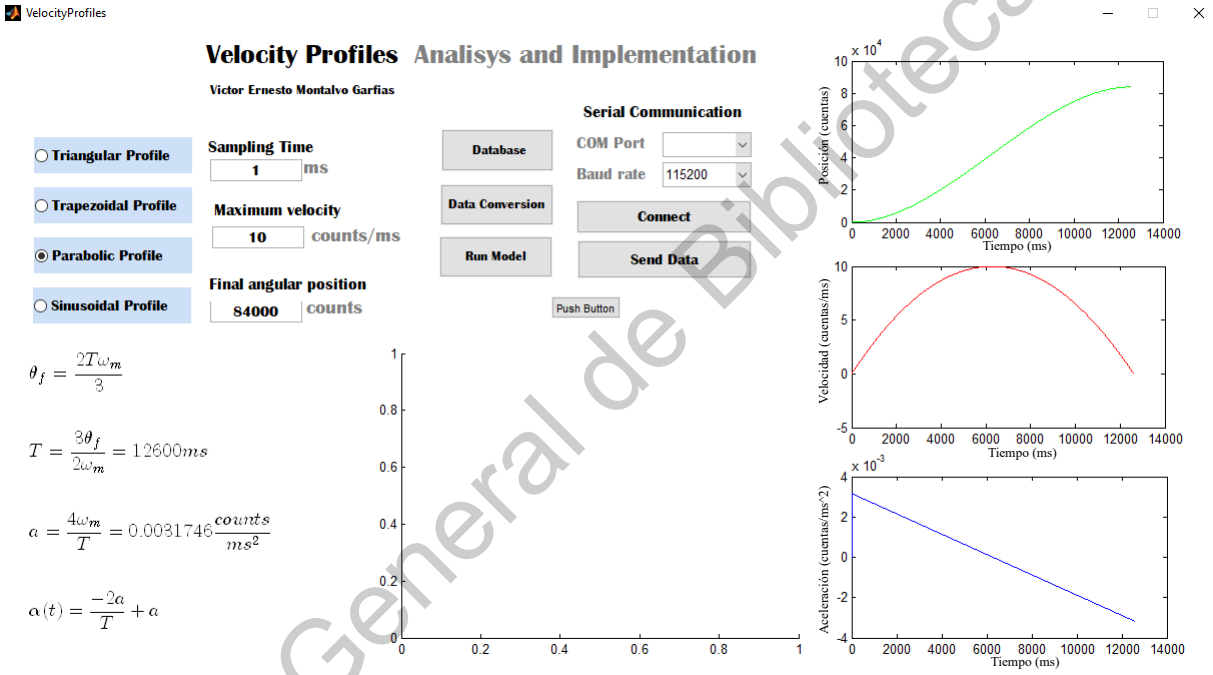


Figura 3.14: Interfaz gráfica desarrollada en *MATLAB*.

Cabe mencionar que la interfaz cuenta con un simulador para 4 perfiles, pero solo envía coeficientes al sistema de control para el perfil trapezoidal y parabólico.
Con la finalidad de facilitar la conversión de unidades para posición, velocidad y aceleración angular la interfaz cuenta con un convertidor que pemite conocer los valores para radianes, grados, vueltas y cuentas para un encoder de 8400 cuentas por vuelta. La Figura 3.15 muestra la interfaz de esta opción.

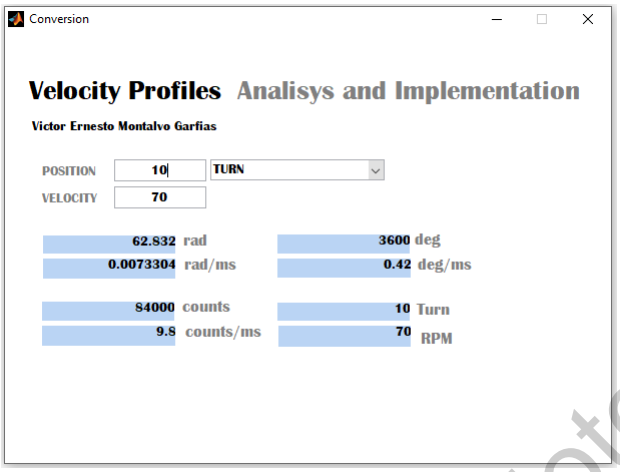


Figura 3.15: Convertidor de unidades.

Cuadro 3.9: Valor de los coeficientes en punto fijo para la interfaz y el *FPGA*.

Perfil	**Señal**	**Formato punto fijo *FPGA***	**Formato punto fijo interfaz**
Trapezoidal	$a = \frac{3\omega_{max}}{T}$	33.31	3.31
Trapezoidal	$-a = \frac{3\omega_{max}}{T}$	33.31	3.31
Trapezoidal	$\frac{T}{3}$	32.0	32.0
Parabólico	$a = \frac{4\omega_{max}}{T}$	33.31	3.31
Parabólico	$\frac{-2a}{T}$	1.31	1.31
Trapezoidal y parabólico	$T = \frac{3\theta_f}{2\omega_{max}}$	32.0	32.0
Trapezoidal y parabólico	$\theta_f = \frac{2T}{3}\omega_{max}$	33.31	NA

CAPÍTULO 4

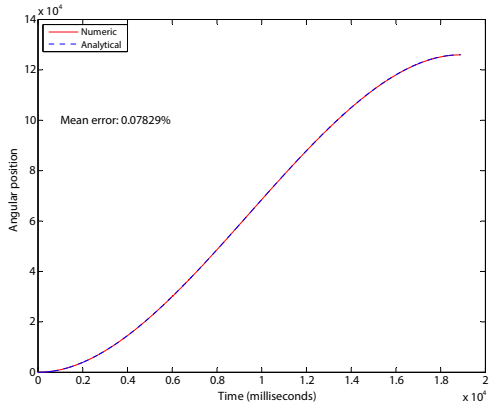
Resultados y discusión

4.1 Resultados

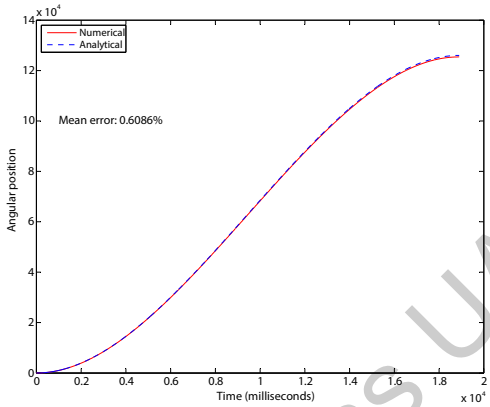
La evidencia audiovisual del proyecto funcional se encuentra en [23]. En este vídeo se muestra el funcionamiento del sistema desde la interacción del usuario con la interfaz y la forma en que se pueden realizar diversas pruebas para diferentes posiciones angulares. La finalidad de este vídeo es además servir como manual práctico para estudiantes, profesores o investigadores que decidan trabajar con el sistema para la enseñanza o la investigación en perfiles de velocidad.

4.1.1 Comparación del método analítico de integración con el método numérico

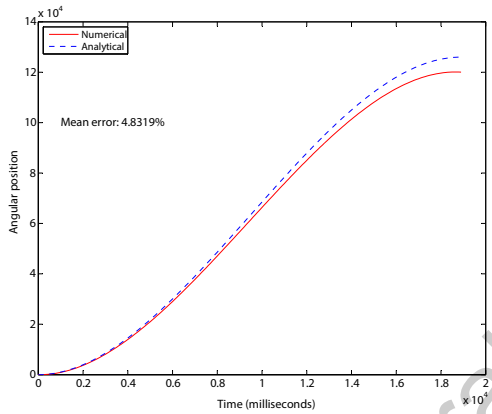
Antes de analizar los resultados logrados a través de *hardware* es conveniente estudiar el error que existe al calcular la posición angular con un método numérico y no uno analítico. Debido a que la precisión de un método numérico depende de las iteraciones que se lleven a cabo, se realiza el experimento con 15 vueltas del perfil parabólico y se compara con la posición obtenida a través de la ecuación analítica (2.10). Los resultado de esta comparación se muestran en la Figura 4.1.



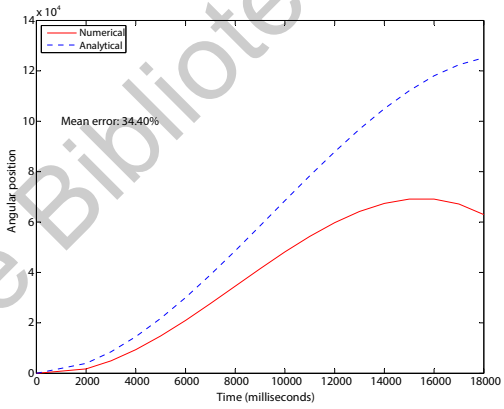
(a) Muestreo a 1 ms.



(b) Muestreo a 10 ms.



(c) Muestreo a 100 ms.



(d) Muestreo a 1000 ms.

Figura 4.1: Método analítico vs numérico para 15 vueltas en perfil parabólico.

La comparación fue realizada con muestreo cada 1, 10, 100 y 1000 ms. Es importante observar que incluso con un periodo de muestreo 10 veces más lento que el que se utilizó para este trabajo (1 ms) el error promedio se mantiene por debajo del 1%; de 100 ms hasta 1 s el error se vuelve más significativo. Es por ello que la velocidad de procesamiento en este sistema es de suma importancia para alcanzar los resultados más cercanos a los valores deseados.

4.1.2 Comparación del método numérico con la trayectoria trazada con el FPGA

Esta sección compara los resultados de movimiento angular de los perfiles parabólico y trapezoidal con las curvas teóricas que calcula la interfaz.

4.1.3 Resultados de la simulación

En esta sección se muestran los resultados de simular las señales que entran y salen del perfil parabólico descrito en el *FPGA*, por una parte en la Figura 4.2 se muestra la simulación con enfoque en observar la activación progresiva de los registros de carga y cómo esto produce un cambio en la posición final. En la Figura 4.3 se muestra la misma simulación pero con enfoque en observar los cambios en la aceleración, velocidad y posición.

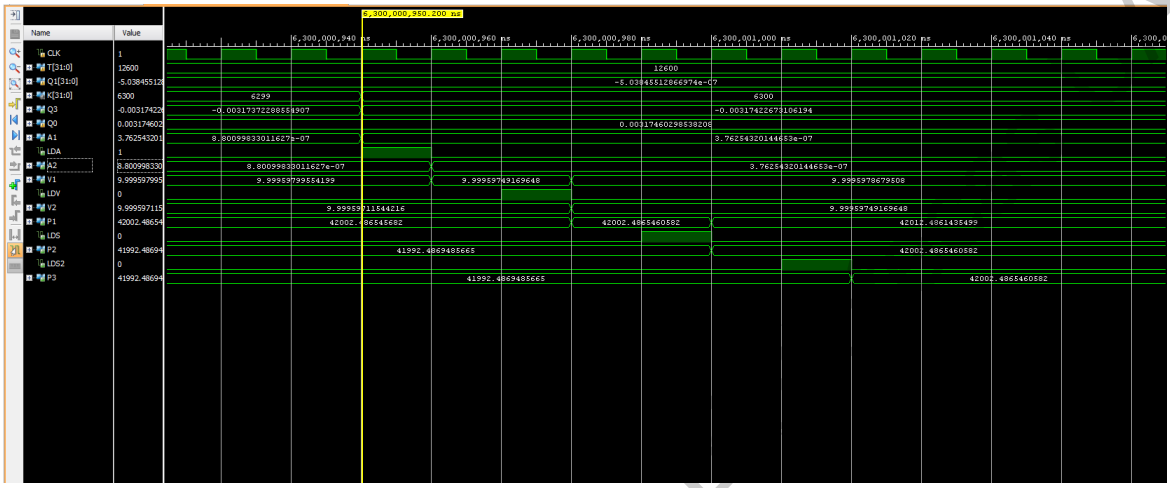


Figura 4.2: Simulación de la activación de los registros del perfil parabólico.

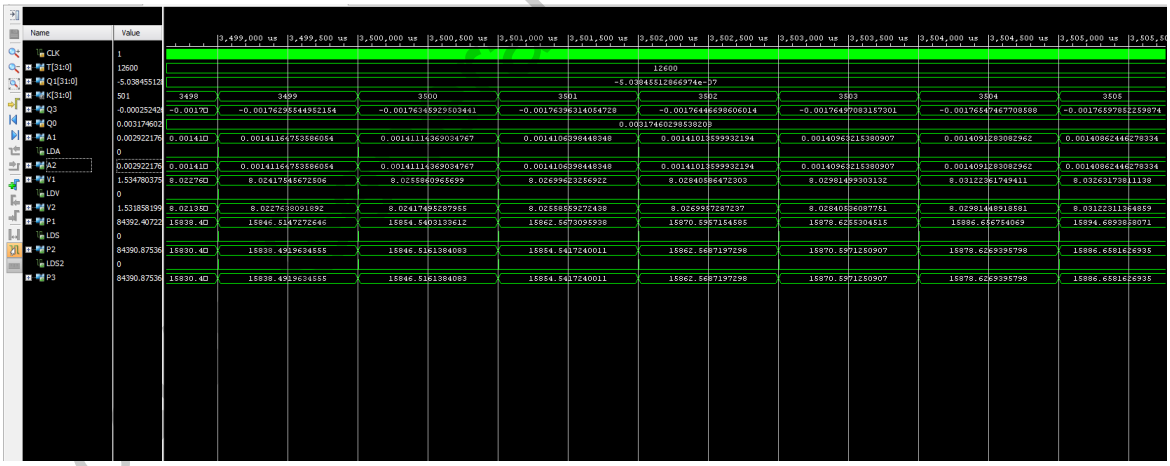


Figura 4.3: Simulación del cambio de valores de posición en el perfil parabólico.

4.1.4 Seguimiento de trayectorias

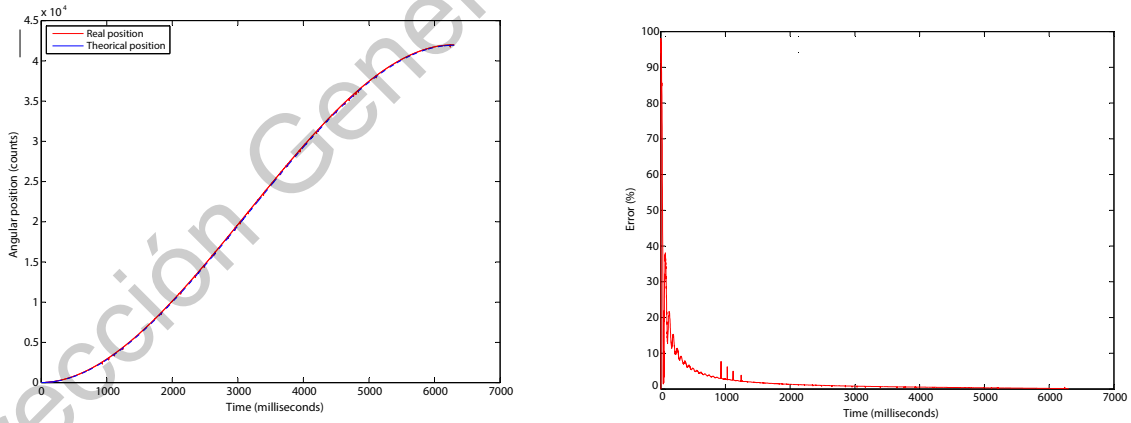
La posición angular de cada trayectoria fue trazada con un encoder incremental que se encuentra integrado al servomotor y que entrega 8400 cuentas por cada revolución. Con la finalidad de

monitorear el comportamiento del sistema de control durante diferentes distacias se realizaron las pruebas indicadas en la Tabla 4.1:

Cuadro 4.1: Posiciones angulares que se pusieron a prueba en los perfiles de velocidad.

θ_f en revoluciones	θ_f en cuentas de encoder	Tiempo requerido
5	42000	6300 ms
10	84000	12600 ms
15	126000	18900 ms

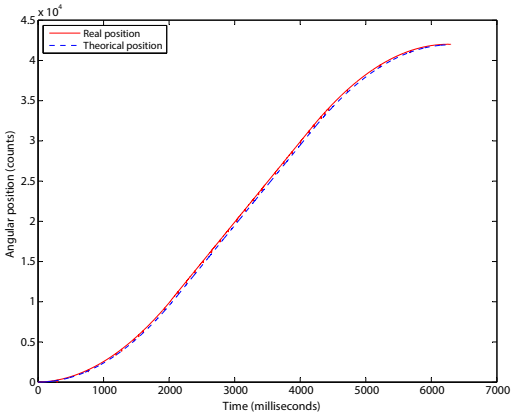
Cada prueba entregó un vector de posiciones angulares monitoreadas cada milisegundo, por lo cuál fue necesario ajustar la velocidad del protocolo RS232 a la máxima soportada por el *IDE* MATLAB: 115200 bps. Debido a la alta velocidad de recepción, la interfaz no fue capaz de graficar en tiempo real los datos recibidos por lo cual primero fueron capturados, procesados y posteriormente graficados. Es importante notar que los tiempos previstos para alcanzar cada posición angular son los mismos para ambos perfiles de acuerdo a la teoría; en las gráficas de las Figuras 4.4, 4.5, 4.6 y 4.7 analizaremos esta comparación.



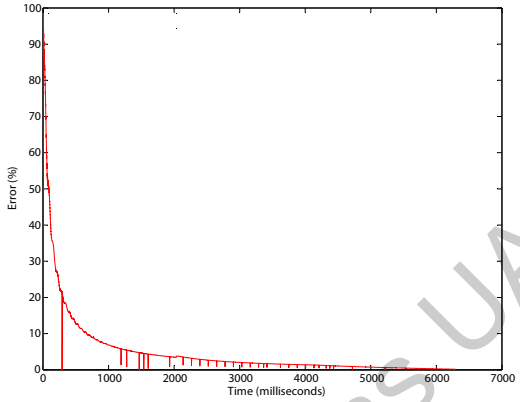
(a) Parabólico.

(b) Error parabólico.

Figura 4.4: Experimento perfil parabólico 5 vueltas.

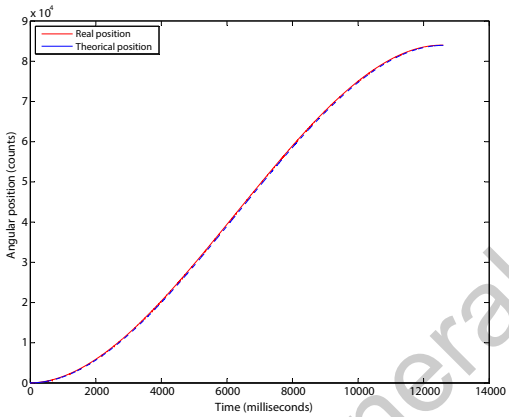


(a) Trapezoidal.

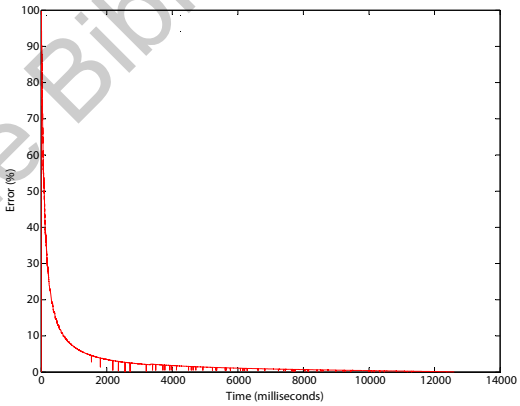


(b) Error trapezoidal.

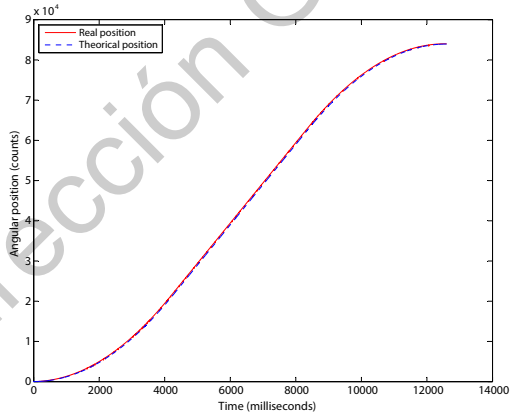
Figura 4.5: Experimento perfil trapezoidal 5 vueltas.



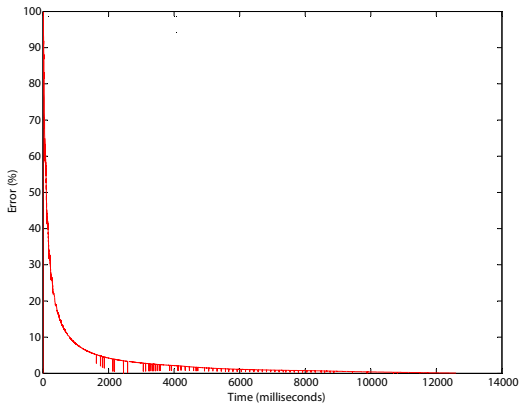
(a) Parabólico.



(b) Error parabólico.

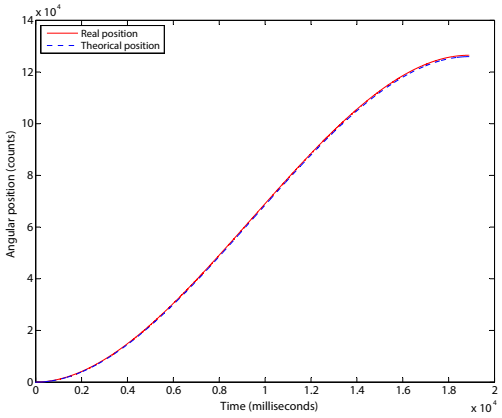


(c) Trapezoidal.

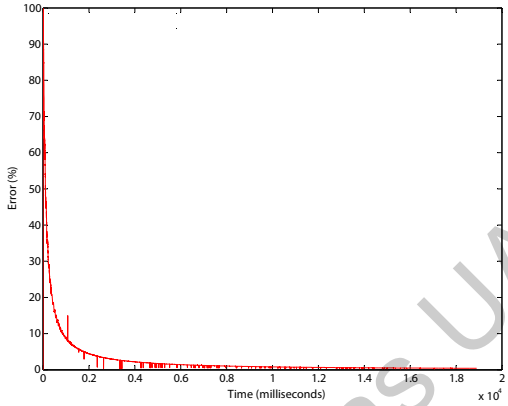


(d) Error trapezoidal.

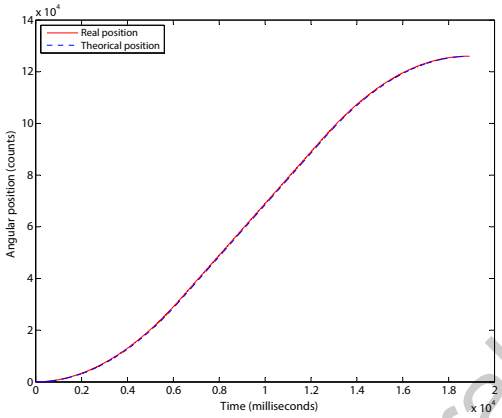
Figura 4.6: Experimento con ambos perfiles 10 vueltas.



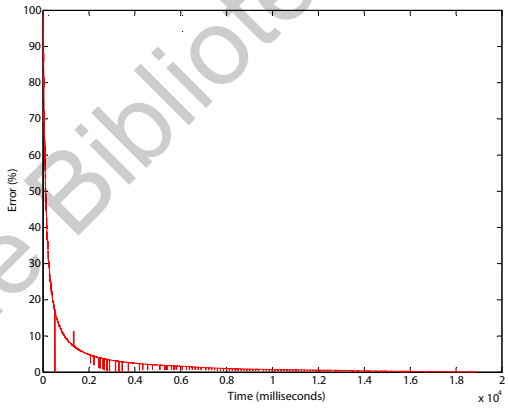
(a) Parabólico.



(b) Error parabólico.



(c) Trapezoidal.

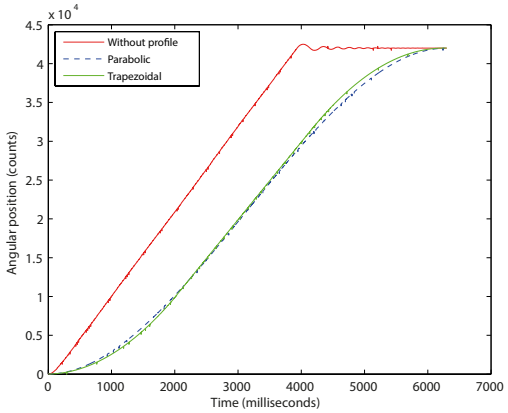


(d) Error trapezoidal.

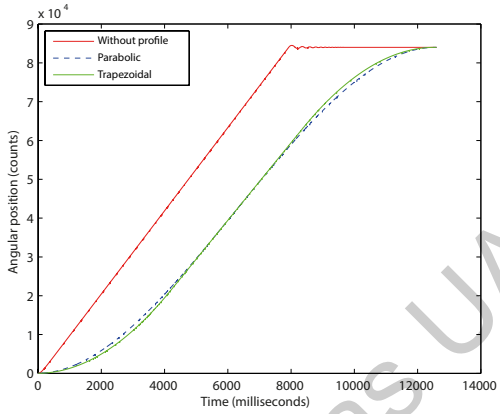
Figura 4.7: Experimento con ambos perfiles 15 vueltas.

4.1.5 Comparación en el seguimiento de trayectorias del controlador con y sin perfiles

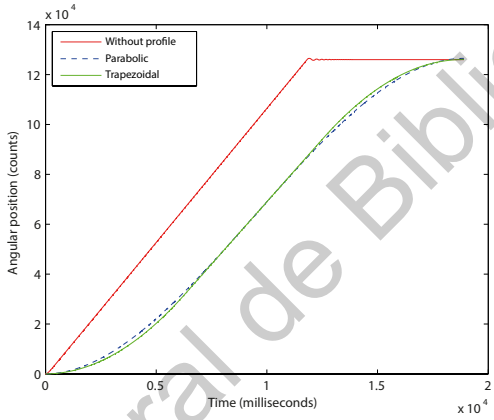
El propósito de este experimento es analizar el seguimiento del controlador cuando a su entrada se utilizan los perfiles de velocidad y la diferencia cuando se le asigna directamente la posición del usuario y una ganancia proporcional de uno. Es importante recordar que la velocidad máxima del motor es un factor que la planeación de trayectorias toma en cuenta para calcular el vector de posiciones. Para este experimento dicha velocidad fue establecida en 10 cuentas por milisegundo, equivalente a 71.4286 *RPM*,con el propósito de no llevar al límite el funcionamiento del motor. La trayectoria seguida por el controlador sin entrada de perfiles utilizará los 80 *RPM* del motor y se verá reflejado en la pendiente de la velocidad y en consecuencia el tiempo de respuesta será menor pero reflejará un aumentará el sobrepaso. Los experimentos realizados se muestran en la Figura 4.8:



(a) Controlador 5 vueltas.



(b) Controlador 10 vueltas.



(c) Controlador 15 vueltas.

Figura 4.8: Experimento de posición para comparar la respuesta del controlador con y sin perfiles.

4.1.6 Consumo de corriente para cada perfil

En esta sección se detalla el experimento realizado con la finalidad de comparar el consumo de potencia del perfil parabólico frente al perfil trapezoidal. En las Figuras 4.9 y 4.10 se muestra la medición de corriente realizada con el osciloscopio *Analog Discovery* y el sensor *MAX471* para medir corriente en una trayectoria de 10 vueltas. La línea azul representa el voltaje y la línea amarilla la corriente. Es conveniente observar que la función descrita por los voltajes de ambos experimentos es muy similar a la curva de velocidad de cada perfil.

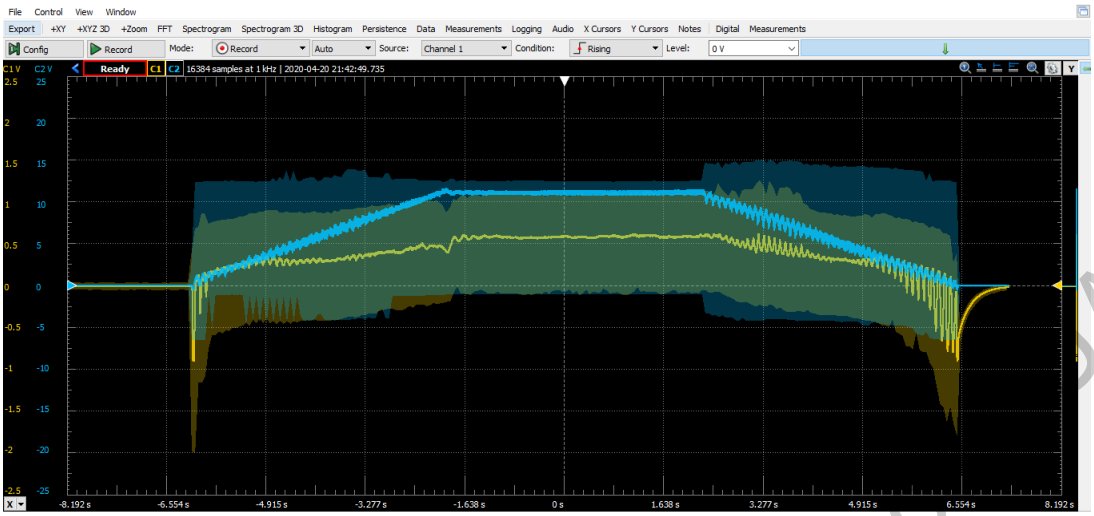


Figura 4.9: Consumo de corriente y voltaje con el perfil trapezoidal para 10 vueltas.

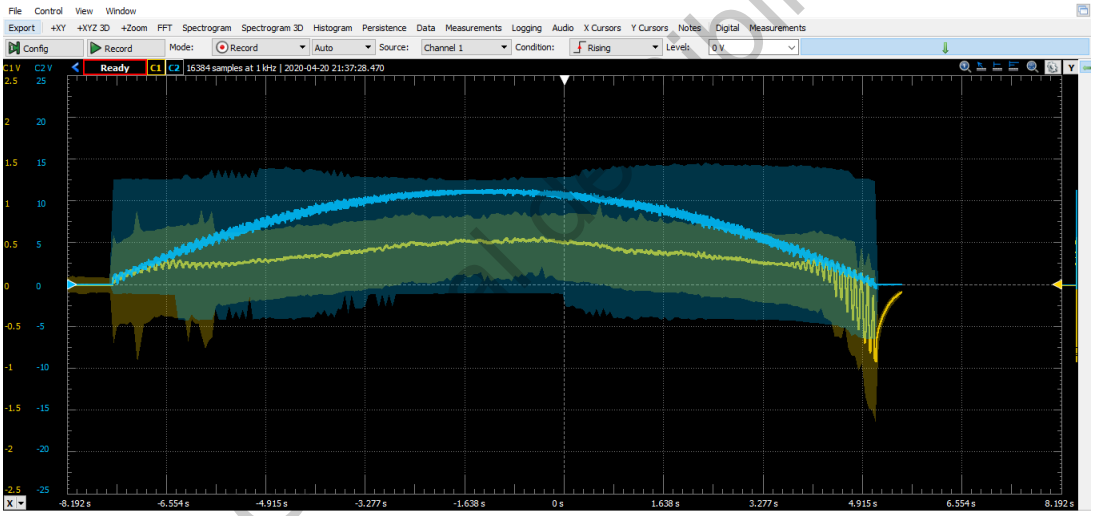
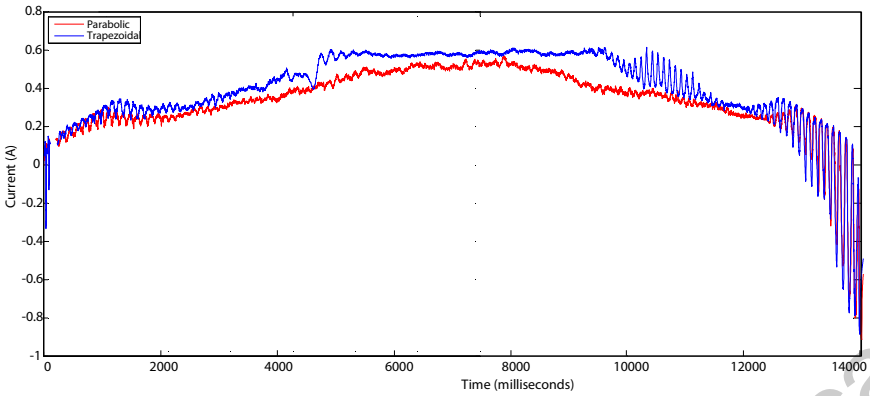


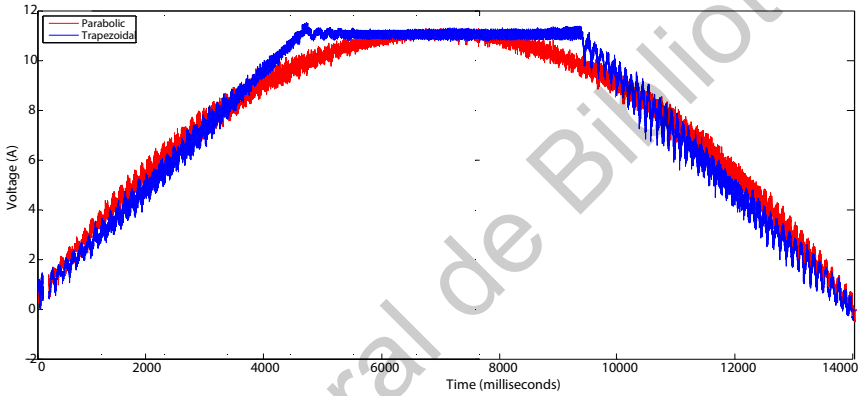
Figura 4.10: Consumo de corriente y voltaje con el perfil parabólico para 10 vueltas.

En la 4.11 se muestra la comparación en el consumo de corriente, voltaje y potencia que ambos perfiles requirieron para completar una trayectoria de 10 vueltas. La función de potencia se obtuvo como el resultado de multiplicar los valores de corriente por voltaje para cada instate de tiempo. Es claro observar que las curvas de consumo del perfil parabólico están siempre por debajo del trapezoidal en todos los casos.

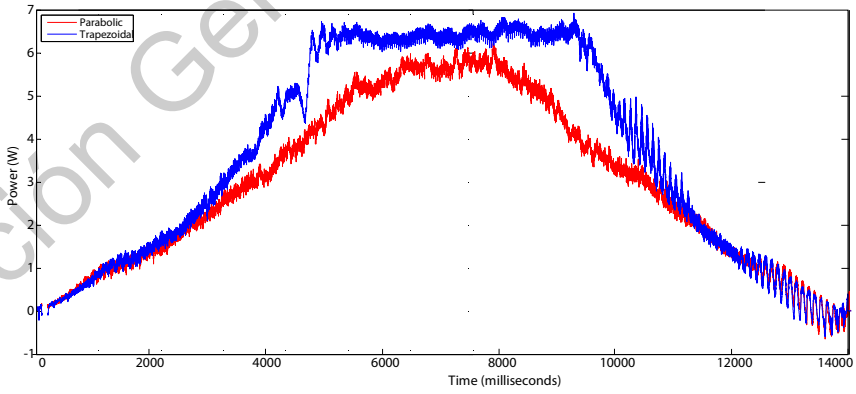
Los valores de la Tabla 4.2 se obtuvieron de la siguiente manera: Primeramente, se promediaron todos los valores de corriente y voltaje obtenidos durante la trayectoria del motor. Después, se multiplicó el voltaje y corriente promedio para obtener la potencia instantánea. Finalmente, se obtuvo la energía consumida al multiplicar la potencia instantánea por el tiempo que duró la trayectoria.



(a) Corriente



(b) Voltaje.



(c) Potencia.

Figura 4.11: Comparación de voltaje, corriente y potencia en ambos perfiles para 10 vueltas.

Cuadro 4.2: Comparación del consumo promedio en ambos perfiles para 10 vueltas.

Variable	**Perfil parabólico**	**Perfil trapezoidal**
Corriente	0.32 A	0.39 A
Voltaje	7.45 V	7.47 V
Potencia	2.38 W	2.91 W
Energía	29.98 J	36.6 J

Podemos observar que el voltaje en ambos casos fue prácticamente el mismo, sin embargo la diferencia significativa provino del consumo de corriente, al ser considerablemente menor en el perfil parabólico. Finalmente podemos concluir que el consumo de energía en el perfil parabólico fue 17.3% menor que el trapezoidal.

4.2 Discusión

Es importante observar que el sistema responde satisfactoriamente al seguimiento de la trayectoria propuesto por ambos perfiles. El error promedio de los 6 experimentos oscila entre el 2.24% y el 4.9%. Si bien el error promedio podría parecer considerable es interesante observar que el promedio del error es muy alto en el arranque del motor debido a que el vector de valores teóricos comienza con valores decimales muy pequeños menores a 1 y el encoder entrega cuentas por unidad, no por fracción; por ello los errores al inicio de la trayectoria son de hasta casi el 100%. Sin embargo después de aproximadamete la primera sexta parte del recorrido, el error se encuentra por debajo del 1%. Cabe señalar que el experimento se realizó para tres distacias diferentes (5, 10 y 15 vueltas) con el propósito de verificar que a pequeñas o grandes distancias el sistema se mantiene estable y no aumenta el valor del error.

En cuanto al consumo de energía se puede observar que, así como se muestra en la teoría, fue significativamente inferior en el caso del perfil parabólico con respecto al trapezoidal.

4.3 Impacto

A continuación se presenta el impacto que el presente trabajo tendrá en el ámbito social, ambiental y económico.

4.3.1 Impacto social

El impacto social de este trabajo proviene principalmente de la seguridad y calidad que el desplazamiento de diferentes servo-máquinas deben ofrecer a sus usuarios finales. Para puntualizar este concepto se presentan dos ejemplos; uno en el sector automotriz y otro en plantas de producción:

- En una planta de producción con máquinas de corte y desplazamiento como lo son tornos, fresadoras o rectificadoras automáticas, es de vital importancia la planeación de trayectorias

para evitar accidentes relacionados con la velocidad de corte y la precisión del mismo. El perfil parabólico representa una medida de seguridad a través de su pricipal característica que es la suavidad en la curva de posición al evitar des-aceleraciones o frenados bruscos que podrían provocar que la herramienta de corte u otra parte móvil salga disparada como proyectil al operador.

- En el sector automotriz se explora cada vez con más profundidad la navegación autónoma en vías transitadas y de alta velocidad. De acuerdo con [24] uno de los principales factores a tomar en cuenta para una navegación precisa y segura es la respuesta oportuna a trayectorias dinámicas del automóvil, para que en caso de un imprevisto el vehículo pueda desacelerar de manera oportuna pero siempre teniendo en cuenta aminorar el impacto debido al *jerk* que en casos críticos podría ocasionar daños significativos al tripulante. El perfil parabólico tiene como característica principal suavizar las trayectorias y eliminar los cambios bruscos de aceleración.

4.3.2 Impacto ambiental

La demanda eléctrica mundial hoy en día es cubierta en su mayoría por la construcción de diferentes centrales eléctricas que basan su funcionamiento en la transformación de la energía. Lamentablemente las fuentes de dicha tranformación (entre las que se encuentran la quema de combustibles, energía cinética del agua o reacciones nucleares) representan un impacto negativo para el ecosistema en el que decidan ser construidas ya que tan solo la infraestructura de una central eléctrica representa la destrucción de ambientes naturales, y por otra parte los residuos de centrales como las de combustible deterioran la atmósfera. Ante dicha problemática existen dos soluciones generales: la primera es encontrar tecnologías más eficientes para la conversión de energía. La segunda es reducir el consumo eléctrico en las actividades del hombre. Un perfil de velocidad parabólico consigue exactamente lo segundo al reducir el *jerk* del actuador y en consecuencia la potencia requerida para su funcionamiento.

4.3.3 Impacto económico

En máquinas de control numérico, como fresadoras o tornos, la velocidad máxima de avance, en la mayoría de los casos, está limitada por las características de la herramienta de corte y no por la velocidad del servomotor; sin embargo, en máquinas como las de tipo pick and place para placas electrónicas el límite para la velocidad está dado por la máxima velocidad que el motor puede entregar y en este caso una trayectoria planeada a través de un perfil garantiza el balance entre el menor tiempo necesario para conseguir la posición final y una aceleración que no dañe el motor o el mecanismo.

Por otro lado, el ahorro en la energía consumida debido a los cambios en la aceleración representa otro de los principales medios para ahorrar recursos.Reducir el *scrap* en la producción es otra medida importante para ahorrar en costos de producción. Como se menciona en [25] es importante suavizar las trayectorias en máquinas *CNC* de fresado debido a que los movimientos bruscos generan imperfecciones de corte principalmente en las esquinas y redondeados de ciertas piezas.

4.4 Publicaciones

Las pricipales publicaciones de este trabajo son:

- Documento de tesis de licenciatura.
- Vídeo de los experimentos realizados para la sección de resultados [23].
- Anexos de código libres para ser usados en trabajos relacionados con el tema.

4.5 Trabajo futuro

Uno de los objetivos de este trabajo es abrir y facilitar la investigación en los siguientes temas:

- El estudio y desarrollo de diferentes perfiles de velocidad, como el perfil triangular o S-curve.
- La implementación de cada vez más componentes para la planeación de trayectorias en el *FPGA* para la futura creación de chips modulares que sean de fácil integración a máquinas servo-asistidas.
- Hacer uso del prototipo y la descripción de hardware para prácticas y clases relacionadas con sistemas digitales y servo-mecanismos.
- Hacer uso de los principios de esta tesis para abordar temas relacionados con conducción autónoma.

Para todo lo anterior se propone como trabajo futuro la creación de un manual de usuario basado en todo el contenido de esta tesis pero especialmente en el capítulo de metodología.

CAPÍTULO 5

Conclusiones

En este trabajo se desarrolló e implemetó un sistema de control de movimiento basado en el perfil de velocidad parabólico y su comparativa con el trapezoidal. Fue así como se logró observar el comportamiento de una trayectoria planeada y su impacto al usarse como entrada de un controlador.

Durante el desarrollo de los objetivos se abordó el desafío de comprobar, a través de la experimentación, las estimaciones teóricas que abordan las características de cada perfil, como lo son la velocidad de respuesta y el consumo de energía. De manera satisfactoria se comprobó que la diferencia entre el pronóstico del modelo matemático y el ejemplo practico fue muy baja.

Gracias a la versatilidad que ofrecen las implementaciones en lógica digital fue posible reconfigurar el sistema para observar el comportamiento del controlador con y sin la asistencia de perfiles de velocidad. Se demostró que aún sin calibrar la ganancia proporcional del controlador a un valor diferente de 1, el seguimiento del vector de posiciónes fue preciso, y esto fue comprobado al observar que la curva teórica del perfil se apegó a la del experimento para diferentes posiciones angulares. El principal motivo de este comportamiento fue el tiempo de muestreo a 1 ms que permitió a los perfiles arrojar cambios de posición tan pequeños que no representaron una complicación para el controlador; para esto fue clave que el sistema pudiese responder al periodo de 1 ms. Podemos concluir que lo que facilitó el trabajo al controlador no fueron las características del perfil en particular, sino la practica de dividir la posición final en posiciones más pequeñas. Sin embargo tomar esto como una buena practica aterriza directo a la pregunta ¿Si el obetivo es segmentar la trayectoria, cómo deberíamos hacer esta división? Es justo la respuesta a esta pregunta donde nace la necesidad de seleccionar perfiles de velocidad con base en sus características.

Como siguiente paso del trabajo se abordó la comparación de estas características entre el perfil parabólico y trapezoidal con especial enfoque en el tiempo de respuesta y el consumo de energía. Finalmente, una de las mayores ventajas del perfil parabólico frente al trapezoidal, planteada en la hipótesis, fue demostrada: un menor consumo de energía necesario para completar la trayectoria deseada. En este resultado es importante remarcar que el tiempo de respuesta para ambos perfiles fue el mismo y en consecuencia ahorrar energía no representó una pérdida en la velocidad. Con esta conclusión podemos resaltar la importancia de planear trayectorias con base en modelos matemáticos que contemplen el enfoque en varibles que pueden impactar de manera significativa diferentes aplicaciones de movilidad, en este caso: la disminución de costos debido al ahorro de energía.

El presente trabajo cumple con el objetivo de abrir la investigación hacia la planeación de trayectorias en una arquitectura abierta que ofrece un gran campo de experimentación con otro tipo de trayectorias.

Bibliografía

[1] Q. Feng and L. Wang, "Fpga-based acceleration and deceleration control for cnc machine tools," in *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)*, pp. 210–214, Dec 2013.

[2] W. Bangji, L. Qingxiang, Z. Lei, Z. Yanrong, L. Xiangqiang, and Z. Jianqiong, "Velocity profile algorithm realization on fpga for stepper motor controller," in *2011 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC)*, pp. 6072–6075, Aug 2011.

[3] B. R. Mutlu, U. Yaman, M. Dolen, and A. B. Koku, "Performance evaluation of different real-time motion controller topologies implemented on a fpga," in *2009 International Conference on Electrical Machines and Systems*, pp. 1–6, Nov 2009.

[4] X. Shao and D. Sun, "Development of a new robot controller architecture with fpga-based ic design for improved high-speed performance," *IEEE Transactions on Industrial Informatics*, vol. 3, pp. 312–321, Nov 2007.

[5] J. Y. Wu, Z. Chen, A. Deguet, and P. Kazanzides, "Fpga-based velocity estimation for control of robots with low-resolution encoders," in *IEEE International Conference on Intelligent Robots and Systems*, pp. 6384–6389, 2018. Cited By :1.

[6] P. . Cardenas, W. J. P. Holgin, F. H. Fonseca Aponte, and A. D. S. Gomez, "A proposal for a soc fpga-based image processing in rgb-d sensors for robotics applications," in *2018 IEEE 2nd Colombian Conference on Robotics and Automation, CCRA 2018*, 2018. Cited By :1.

[7] M. C. Chinnaaiah, K. Anusha, B. Bharat, M. Divya, P. S. Raju, and S. Dubey, "Deliberation of curvature type obstacles: A new approach using fpga based robot," in *2018 International Conference on Control, Power, Communication and Computing Technologies, ICCPCCT 2018*, pp. 519–522, 2018. Cited By :1.

[8] Y. Chen, H. Wei, K. Sun, T. Wang, and Y. Zou, "System-on-chip (soc) design for cnc system," in *2009 IEEE International Symposium on Industrial Electronics*, pp. 690–693, July 2009.

[9] M. Martínez-Prado, A. Franco-Gasca, G. H. Ruiz, and O. Soto-Dorantes, "Multi-axis motion controller for robotic applications implemented on an fpga," *The International Journal of Advanced Manufacturing Technology*, Feb 2013.

[10] M. Martínez-Prado, J. Rodríguez-Reséndiz, R. A. Gómez, and G. H. Ruiz, "Parabolic velocity profile," in *Velocity Profiles: Analisys and Implementation*, pp. 9–12, Feb 2013.

[11] V. Henández, R. Silva, and R. Carrillo, "Modelo matemático de sistemas físicos," in *Control Automático: Teoría de diseño, Construcción de prototipos, Identificación y Pruebas Experimentales*, p. 16, Feb 2013.

[12] L. Biagiotti and C. Melchiorri, "Trayectory planning," in *Trayectory Planning for Automatic Machines and Robots*, p. 5, Feb 2008.

[13] R. Romero-Troncoso, *Electrónica Digital y lógica programable*, vol. 1 of *1.* Lascuráin de Retana No. 5. C.P. 36000: Universidad de Guanajuato, 1 ed., 2007.

[14] S. Barrios-Dv, M. Lopez-Franco, J. D. Rios, N. Arana-Daniel, C. Lopez-Franco, and A. Y. Alanis, "An autonomous path controller in a system on chip for shrimp robot," *Electronics (Switzerland)*, vol. 9, no. 3, 2020.

[15] H. Moore, *MATLAB para ingenieros.*, vol. 1 of *1.* México: Prentice Hall, 1 ed., 2007.

[16] R. Norton, *Diseño de maquinaria, Síntesis y análisis de máquinas y mecanismos*, vol. 1 of *1.* Prolongación Paseo de la Reforma 1015, Torre A Piso 17, Colonia Desarrollo Santa Fé, Delegación Alvaro Obregón, C.P. 01376, México, D.F.: Mc Graw Hill, 5 ed., 2013.

[17] S. Hosseini and I. Hahn, "Energy-efficient motion planning for electrical drives," in *2018 IEEE Electrical Power and Energy Conference, EPEC 2018*, 2018. Cited By :1.

[18] Y. Mei, Y.-H. Lu, Y. C. Hu, and C. G. Lee, "Energy-efficient motion planning for mobile robots," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 5, pp. 4344–4349, IEEE, 2004.

[19] S. Chapman, *Máquinas eléctricas*, vol. 1 of *1.* Prolongación Paseo de la Reforma 1015, Torre A Piso 17, Colonia Desarrollo Santa Fé, Delegación Alvaro Obregón, C.P. 01376, México, D.F.: Mc Graw Hill, 5 ed., 2012.

[20] C. Alexander and M. Sadiku, *Fundamentos de circuitos eléctricos*, vol. 1 of *1.* Prolongación Paseo de la Reforma 1015, Torre A Piso 17, Colonia Desarrollo Santa Fé, Delegación Alvaro Obregón, C.P. 01376, México, D.F.: Mc Graw Hill, 3 ed., 2006.

[21] V. J. Lumelsky and A. A. Stepanov, "Dynamic path planning for a mobile automaton with limited information on the environment," *IEEE Transactions on Automatic Control*, vol. 31, no. 11, pp. 1058–1063, 1986. Cited By :188.

[22] M. Oveis-Gharan and G. N. Khan, "Reconfigurable on-chip interconnection networks for high performance embedded soc design," *Journal of Systems Architecture*, vol. 106, 2020.

[23] V. E. M. Garfias, "Perfiles de velocidad." urlhttps://www.youtube.com/watch?v=SMlC-hm4IYg, 2020.

[24] M. Raineri and C. G. L. Bianco, "Jerk limited planner for real-time applications requiring variable velocity bounds," in *IEEE International Conference on Automation Science and Engineering*, vol. 2019-August, pp. 1611–1617, 2019.

[25] Q. . Xiao, M. Wan, Y. Liu, X. . Qin, and W. . Zhang, "Space corner smoothing of cnc machine tools through developing 3d general clothoid," *Robotics and Computer-Integrated Manufacturing*, vol. 64, 2020.

Apéndice

En esta última sección se añade información que será de utilidad para la comprensión y desarrollo de los temas tratados en los capítulos de la tesis. Se incluyen las siguientes secciones:

- Descripción de Hardware en VHDL del Sistema de control de movimiento.
- Código de la interfaz desarrollada en *MATLAB*.
- Código del perfil parabólico con integración rectangular.
- Código del perfil trapezoidal con integración rectangular.
- Código del convertidor decimal a punto fijo.

A.1 Descripción de Hardware en VHDL del Sistema de control de movimiento

En la siguiente sección están incluídos todos los códigos que describen los circuitos digitales detrás del sistema de control de movimiento del presente trabajo. Este código está disponible para ser útil en futuras investigaciones o trabajos del mismo campo. La siguiente descripción de Harware puede ser libremente implementada, modificado y mejorada en caso de ser requerido.

Los códigos a continuación presentados se encuentran ordenados por relevancia en el sistema de control de movimiento de la siguiente forma:

1. Bloque integrador principal.
2. Generador del perfil parabólico.
3. Generador del perfil trpezoidal.
4. Controlador proporcional.

5. Sistema de comunicación.

6. Encoder incremental.

7. Indicador digital de la posición angular.

8. Bloques en común para las funciones principales.

A.1.1 Bloque integrador principal

Aquí se presenta el código integrador en que se reunen los módulos principales de funcionalidad y comunicación, así como las salidas y entradas finales del circuito.

```
Library IEEE;
use IEEE.std_logic_1164.all;

Entity MotionControlSystem is
    port(
        CLK   : in std_logic;
        RST   : in std_logic;
        CTO   : in std_logic;
        RDD   : in std_logic;
        RX    : in std_logic;
        TX    : out std_logic;
        CHA   : in std_logic;
        CHB   : in std_logic;
        AOUT  : out std_logic_vector(7 downto 0);
        SOUT  : out std_logic_vector(6 downto 0);
        MBIT  : out std_logic_vector(3 downto 0);
        SEL   : in std_logic_vector(1 downto 0);
        DOUT  : out std_logic_vector(15 downto 0);
        CST   : in std_logic;
        MDIR  : out std_logic;
        BRAKE : out std_logic
        );
end MotionControlSystem;

Architecture Structural of MotionControlSystem is

Component UART is
    generic(n : integer :=8);
    port(
        CLK  : in std_logic;
        RST  : in std_logic;
        CTO  : in std_logic;
        DIN  : in std_logic_vector(n-1 downto 0);
        RX   : in std_logic;
        TX   : out std_logic;
        RXRDY  : out std_logic;
        IND  : out std_logic;
        DOUT : out std_logic_vector(n downto 0)
    );
end Component;

Component IncrementalEncoder is
```

```
    generic (n : integer := 16);
    port(
        CLK    : in std_logic;
        RST    : in std_logic;
        CHA    : in std_logic;
        CHB    : in std_logic;
        PULSE  : out std_logic;
        DIR    : out std_logic;
        COUNT  : out std_logic_vector(n-1 downto 0)
        );
end Component;

Component DMillionCounter is
    port(
        CLK  : in std_logic;
        RST  : in std_logic;
        ENI  : in std_logic;
        DIR  : in std_logic;
        AOUT : out std_logic_vector(7 downto 0);
        SOUT : out std_logic_vector(6 downto 0)
        );
end Component;

Component ParabolicProfileGenerator is
    port(
        CLK  : in std_logic;
        RST  : in std_logic;
        ST   : in std_logic;
        T    : in std_logic_vector(31 downto 0);
        Q1   : in std_logic_vector(31 downto 0);
        a    : in std_logic_vector(63 downto 0);
        POS  : out std_logic_vector(17 downto 0);
        ORDY : out std_logic
    );
end Component;

Component DataDivider is
    port(
        CLK  : in std_logic;
        RST  : in std_logic;
        ENI  : in std_logic;
        ENO  : out std_logic;
        DIN  : in std_logic_vector(31 downto 0);
        DOUT : out std_logic_vector(7 downto 0)
        );
end Component;

Component GenericTimeBase is
    generic( n : integer:= 10);
    port(
        CLK : in std_logic;
        RST : in std_logic;
        ENI : in std_logic;
        ENO : out std_logic
        );
end Component;
```

```
1098
Component INDMUX is
1100    port(
            SEL   : in std_logic_vector(11 downto 0);
1102        DIN1  : in std_logic_vector(7 downto 0);
            DIN2  : in std_logic_vector(7 downto 0);
1104        DIN3  : in std_logic_vector(7 downto 0);
            DIN4  : in std_logic_vector(7 downto 0);
1106        DIN5  : in std_logic_vector(7 downto 0);
            DIN6  : in std_logic_vector(7 downto 0);
1108        DIN7  : in std_logic_vector(7 downto 0);
            DIN8  : in std_logic_vector(7 downto 0);
1110        DIN9  : in std_logic_vector(7 downto 0);
            DIN10 : in std_logic_vector(7 downto 0);
1112        DIN11 : in std_logic_vector(7 downto 0);
            DIN12 : in std_logic_vector(7 downto 0);
1114        DOUT  : out std_logic_vector(7 downto 0)
    );
1116 end Component;

1118 Component QRegister is
    port(
1120        CLK   : in std_logic;
            RST   : in std_logic;
1122        ENI   : in std_logic;
            DIN   : in std_logic_vector(7 downto 0);
1124        a     : out std_logic_vector(63 downto 0);
            T     : out std_logic_vector(63 downto 0);
1126        Q1    : out std_logic_vector(31 downto 0);
            QRDY  : out std_logic
1128        );
    end Component;
1130
    Component QIND is
1132    port(
            SEL  : in std_logic_vector(4 downto 0);
1134        DIN1 : in std_logic_vector(63 downto 0);
            DIN2 : in std_logic_vector(31 downto 0);
1136        DIN3 : in std_logic_vector(31 downto 0);
            DOUT : out std_logic_vector(15 downto 0)
1138        );
    end Component;
1140
    Component SRLatch is
1142  port(
      RST  : in std_logic;
1144  CLK  : in std_logic;
      SET  : in std_logic;
1146  CLR  : in std_logic;
      QOUT : out std_logic
1148  );
    end Component;
1150
    Component PController is
1152    generic(
        n : integer :=18;
```

```
1154        m : integer :=16
            );
1156        port(
                RST  : in std_logic;
1158            CLK  : in std_logic;
                STR  : in std_logic;
1160            SP   : in std_logic_vector(n-1 downto 0);
                YT   : in std_logic_vector(n-1 downto 0);
1162            q0   : in std_logic_vector(m-1 downto 0);
                q1   : in std_logic_vector(m-1 downto 0);
1164            RDY  : out std_logic;
                UOUT : out std_logic_vector(3 downto 0);
1166            SIGN : out std_logic
                );
1168    end Component;

1170    Component TrapezoidalProfileGenerator is
            port(
1172            CLK  : in std_logic;
                RST  : in std_logic;
1174            ST   : in std_logic;
                T1   : in std_logic_vector(31 downto 0);
1176            T2   : in std_logic_vector(31 downto 0);
                A1   : in std_logic_vector(63 downto 0);
1178            NA1  : in std_logic_vector(63 downto 0);
                POS  : out std_logic_vector(17 downto 0);
1180            ORDY : out std_logic;
                RDY2: out std_logic
1182        );
    end Component;
1184
    Component POSMUX is
1186        port(
        SEL  : in std_logic_vector(1 downto 0);
1188    P1   : in std_logic_vector(17 downto 0);
        P2   : in std_logic_vector(17 downto 0);
1190    CLR1 : in std_logic;
        CLR2 : in std_logic;
1192    POUT : out std_logic_vector(17 downto 0);
        CLROUT : out std_logic
1194        );
    end Component;
1196
    signal PULSE, DIR, auxEN, auxTB, RXRDY, QRDY, STP, STPP, STPPD, CLR, QRST, CST2, CRST,
         CLR1, CLR2 : std_logic;
1198    signal COUNT       : std_logic_vector(31 downto 0);
    signal Q1, T, Q12, T2, T1, T22, TW2      : std_logic_vector(31 downto 0);
1200    signal a, a2, A1, NA1, W1, W2     : std_logic_vector(63 downto 0);
    signal POS2, W3       : std_logic_vector(31 downto 0);
1202    signal UDOUT      : std_logic_vector(8 downto 0);
    signal DDV, SDOUT : std_logic_vector(7 downto 0);
1204    signal OSEL       : std_logic_vector(3 downto 0);
    signal D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12 : std_logic_vector(7 downto
        0);
1206    signal P1, P2, CPOS : std_logic_vector(17 downto 0);
    signal CCOUNT, SP : std_logic_vector(17 downto 0);
```

```
signal PCQ0, PCQ1 : std_logic_vector(15 downto 0);
signal SBIT : std_logic_vector(3 downto 0);
signal VBR : std_logic_vector(1 downto 0);

begin
    POS2  <= "00000000000000" & CPOS;
    a2      <= "00000000000000000000000000000000000000011010000000011010000000"; --
    0.0031746
    T22      <= "00000000000000000011000100111000"; --12600
    Q12     <= "11111111111111111111101111000110"; --5.039e-07
    SDOUT <= UDOUT(8 downto 1);
    STPPD <= QRDY and CST;
    QRST <= RST xor CLR;
    SP <= "0000000000001101010";
    CCOUNT <= COUNT(17 downto 0);
    PCQ0 <= "0000000100000000"; -- 1   8.8
    PCQ1 <= "1111111100000000"; -- -1  8.8
    MBIT <= SBIT;
    DOUT(3 downto 0) <= SBIT;
    CST2 <= CST xor CLR;
    CRST <= RST;
    TW2 <= W2(31 downto 0);
    T1 <= "00000000000000000001001110001000";
    T2 <= "00000000000000000001001110001000";
    A1 <= "00000000000000000000000000000000000000010000011000100100110111";
    NA1<= "11111111111111111111111111111111111111101111100111011011001001";

    U01 : IncrementalEncoder generic map(32) port map(CLK, RST, CHA, CHB, PULSE, DIR,
    COUNT);
    U02 : UART generic map(8) port map(CLK, RST, auxTB, DDV, RX, TX, RXRDY, OPEN,
    UDOUT);
    U03 : DMillionCounter port map(CLK, RST, PULSE, DIR, AOUT, SOUT);
    U04 : ParabolicProfileGenerator port map(CLK, RST, CST, TW2, W3, W1, P1, CLR1);
    U05 : TrapezoidalProfileGenerator port map (CLK, RST, CST, W3, W3, W1, W2, P2,
    CLR2, CLR);
    U06:  POSMUX port map(SEL, P1, P2, CLR1,CLR2, CPOS, BRAKE);
    U07 : PController generic map(18, 16) port map(RST, CLK, CST, CPOS, CCOUNT, PCQ0,
    PCQ1, DOUT(4), SBIT, MDIR);
    U08 : GenericTimeBase generic map(100000)port map(CLK, RST, CTO, auxEN);
    U09 : DataDivider port map(CLK, RST, auxEN, auxTB, COUNT, DDV);
    U10 : QRegister port map(CLK, RST, RXRDY, SDOUT, W1, W2, W3, QRDY);
    U11 : GenericTimeBase generic map(2000)port map(CLK, RST, QRDY, STP);
end Structural;
```

A.1.2 Generador del perfil parabólico

En este apartado se presenta el código del bloque encargado de calcular de manera iterativa las posiciones angulares para cada instante de tiempo usando un perfil parabólico para la planeación de la trayectoria.

```
Library IEEE;
use IEEE.std_logic_1164.all;
```

```
1002
Entity ParabolicProfileGenerator is
1004    port(
            CLK  : in std_logic;
1006        RST  : in std_logic;
            ST   : in std_logic;
1008        T    : in std_logic_vector(31 downto 0);
            Q1   : in std_logic_vector(31 downto 0);
1010        a    : in std_logic_vector(63 downto 0);
            POS  : out std_logic_vector(17 downto 0);
1012        ORDY : out std_logic
    );
1014end ParabolicProfileGenerator;

1016Architecture Structural of ParabolicProfileGenerator is

1018Component GenericTimmer is
  generic(n : integer := 10);
1020  port(
  RST : in std_logic;
1022  CLK : in std_logic;
  ENI : in std_logic;
1024  ENO : out std_logic
  );
1026end Component ;

1028Component ProgrammableCounter is
  generic(n : integer := 2);
1030  port(
  RST: in std_logic;
1032  CLK: in std_logic;
  ENI: in std_logic;
1034  SP: in std_logic_vector(n-1 downto 0);
  ENO: out std_logic;
1036  COUNT: out std_logic_vector(n-1 downto 0)
  );
1038end Component;

1040Component ParabolicProfileFSM is
  port(
1042  CLK  : in std_logic;
  RST  : in std_logic;
1044  ST   : in std_logic;
  Ts   : in std_logic;
1046  CRDY : in std_logic;
  STT  : out std_logic;
1048  LDA  : out std_logic;
  LDV  : out std_logic;
1050  LDS  : out std_logic;
  LDS2 : out std_logic
1052  );
end Component;
1054
Component Multiplier is
1056  generic(n : integer := 32; m : integer := 32);
  port(
```

```
1058    OPA : in std_logic_vector(n - 1 downto 0);
        OPB : in std_logic_vector(m - 1 downto 0);
1060    RES : out std_logic_vector(n + m - 1 downto 0)
        );
1062  end Component;

1064  Component AddSubs is
        generic(n: integer:=8);
1066    port(
        OPA: in std_logic_vector(n-1 downto 0);
1068    OPB: in std_logic_vector(n-1 downto 0);
        SIGN: in std_logic;
1070    RES: out std_logic_vector(n-1 downto 0)
        );
1072  end Component;

1074  Component LoadRegister is
        generic(n : integer := 8);
1076    port(
        RST: in std_logic; --Reset asincrono
1078    CLK: in std_logic;--
        LDR: in std_logic;-- ENA
1080    DIN: in std_logic_vector(n-1 downto 0);--Dato de entrada
        DOUT: out std_logic_vector(n-1 downto 0)--dato de salida
1082    );
      end Component;
1084
      signal MRES                : std_logic_vector(63 downto 0);
1086  signal Ts, CRDY, STT        : std_logic;
      signal K                   : std_logic_vector(31 downto 0);
1088  signal LDA, LDV, LDS, LDS2 : std_logic;
      signal ACC1,VEL1,POS1      : std_logic_vector(63 downto 0);
1090  signal ACC2,VEL2,POS2,POS3 : std_logic_vector(63 downto 0);
      signal POS2S,POS3S,DIFF    : std_logic_vector(31 downto 0);
1092  signal VBR : std_logic_vector(1 downto 0);
      begin
1094      ORDY <= VBR(0);
          POS <= POS3(48 downto 31);
1096      U01 : GenericTimmer generic map(99992) port map(RST, CLK, STT, Ts);
          U02 : ProgrammableCounter generic map(32) port map(RST, CLK, Ts, T, CRDY, K);
1098      U03 : LoadRegister generic map(2) port map (RST, CLK, CRDY, "11", VBR);
          U04 : ParabolicProfileFSM port map(CLK,RST,ST,Ts,CRDY,STT,LDA,LDV,LDS,LDS2);
1100      U05 : Multiplier generic map(32, 32) port map (Q1,K,MRES);
          U06 : AddSubs generic map(64) port map(a,MRES,'0',ACC1);
1102      U07 : LoadRegister generic map(64) port map(RST,CLK,LDA,ACC1,ACC2);
          U08: AddSubs generic map(64) port map(VEL2,ACC2,'0',VEL1);
1104      U09: LoadRegister generic map(64) port map(RST,CLK,LDV,VEL1,VEL2);
          U10: AddSubs generic map(64) port map(POS2,VEL2,'0',POS1);
1106      U11: LoadRegister generic map(64) port map(RST,CLK,LDS,POS1,POS2);
          U12: LoadRegister generic map(64) port map(RST,CLK,LDS2,POS2,POS3);
1108
      end Structural;
```

A.1.3 Generador del perfil trapezoidal

En este apartado se presenta el código del bloque encargado de calcular de manera iterativa las posiciones angulares para cada instante de tiempo usando un perfil trapezoidal para la planeación de la trayectoria.

```
Library IEEE;
use IEEE.std_logic_1164.all;

Entity TrapezoidalProfileGenerator is
    port(
        CLK  : in std_logic;
        RST  : in std_logic;
        ST   : in std_logic;
        T1   : in std_logic_vector(31 downto 0);
        T2   : in std_logic_vector(31 downto 0);
        A1   : in std_logic_vector(63 downto 0);
        NA1  : in std_logic_vector(63 downto 0);
        POS  : out std_logic_vector(17 downto 0);
        ORDY : out std_logic;
        RDY2: out std_logic
    );
end TrapezoidalProfileGenerator;

Architecture Structural of TrapezoidalProfileGenerator is

Component GenericTimmer is
  generic(n : integer := 10);
  port(
  RST : in std_logic;
  CLK : in std_logic;
  ENI : in std_logic;
  ENO : out std_logic
  );
end Component ;

Component ProgrammableCounter is
  generic(n : integer := 2);
  port(
  RST: in std_logic;
  CLK: in std_logic;
  ENI: in std_logic;
  SP: in std_logic_vector(n-1 downto 0);
  ENO: out std_logic;
  COUNT: out std_logic_vector(n-1 downto 0)
  );
end Component;

Component AddSubs is
  generic(n: integer:=8);
  port(
  OPA: in std_logic_vector(n-1 downto 0);
  OPB: in std_logic_vector(n-1 downto 0);
  SIGN: in std_logic;
  RES: out std_logic_vector(n-1 downto 0)
  );
```

```
1050 end Component;

1052 Component LoadRegister is
       generic(n : integer := 8);
1054   port(
       RST: in std_logic; --Reset asincrono
1056   CLK: in std_logic;--
       LDR: in std_logic;-- ENA
1058   DIN: in std_logic_vector(n-1 downto 0);--Dato de entrada
       DOUT: out std_logic_vector(n-1 downto 0)--dato de salida
1060   );
     end Component;
1062
     Component ACCMUX is
1064     generic(n : integer :=8
                );
1066     port(
           SEL  : in std_logic_vector(1 downto 0);
1068       A1   : in std_logic_vector(n-1 downto 0);
           NA1  : in std_logic_vector(n-1 downto 0);
1070       A2   : out std_logic_vector(n-1 downto 0)
           );
1072  end Component;

1074  Component TMUX is
        generic(n : integer :=8
1076            );
        port(
1078        SEL  : in std_logic_vector(1 downto 0);
            T1   : in std_logic_vector(n-1 downto 0);
1080        T2   : in std_logic_vector(n-1 downto 0);
            T    : out std_logic_vector(n-1 downto 0)
1082        );
      end Component;
1084
     Component TrapezoidalProfileFSM is
1086       port(
           CLK  : in std_logic;
1088       RST  : in std_logic;
           ST   : in std_logic;
1090       Ts   : in std_logic;
           CRDY : in std_logic;
1092       STT  : out std_logic;
           LDA  : out std_logic;
1094       LDV  : out std_logic;
           LDS  : out std_logic;
1096       LDS2 : out std_logic
           );
1098 end Component;

1100 signal MRES                : std_logic_vector(63 downto 0);
     signal Ts, CRDY1, CRDY2, STT       : std_logic;
1102 signal K, T, TR1, TR2                  : std_logic_vector(31 downto 0);
     signal LDA, LDV, LDS, LDS2 : std_logic;
1104 signal V1,P1       : std_logic_vector(63 downto 0);
     signal A2,V2,P2,P3, AR, NAR   : std_logic_vector(63 downto 0);
```

```
signal POS2S,POS3S,DIFF     : std_logic_vector(31 downto 0);
signal SEL, SP                  : std_logic_vector(1 downto 0);
signal VBR : std_logic_vector(1 downto 0);
begin
    RDY2 <= CRDY2;
    ORDY <= VBR(0);
    POS <= P3(48 downto 31);
    SP <= "10";
    U01 : GenericTimmer generic map(100000) port map(RST, CLK, STT, Ts);
    U02 : ProgrammableCounter generic map(32) port map(RST, CLK, Ts, T, CRDY1, OPEN);
    U03 : ProgrammableCounter generic map(2) port map(RST, CLK, CRDY1, SP, CRDY2, SEL)
    ;
    U04 : TrapezoidalProfileFSM port map (CLK, RST, ST, Ts, CRDY2, STT, LDA, LDV, LDS,
     LDS2);
    U05 : LoadRegister generic map(2) port map (RST, CLK, CRDY2, "11", VBR);
    U06 : ACCMUX generic map(64) port map(SEL, A1, NA1, A2);
    U07 : TMUX generic map(32) port map(SEL, T1, T2, T);
    U08 : AddSubs generic map(64) port map(A2, V2, '0', V1);
    U09: LoadRegister generic map(64) port map(RST, CLK, LDV, V1, V2);
    U10: AddSubs generic map(64) port map(P2, V2, '0', P1);
    U11: LoadRegister generic map(64) port map(RST, CLK, LDS, P1, P2);
    U12: LoadRegister generic map(64) port map(RST, CLK, LDS2, P2, P3);

end Structural;
```

A.1.4 Controlador proporcional

A continuación de presenta la descripción del controlador encargado de seguir el vector de posiciones generadas por los perfiles de velocidad. Este bloque también puede ser utilizado para trazar directamente la posición requerida por el usuario sin usar los perfiles de velocidad; y en efecto se utiliza en el presente trabajo para comparar la precisión y estabilidad del sistema con y sin perfiles de velocidad.

```
Library IEEE;
use IEEE.std_logic_1164.all;

Entity PController is
    generic(
    n : integer :=18;
    m : integer :=16
    );
    port(
        RST  : in std_logic;
        CLK  : in std_logic;
        STR  : in std_logic;
        SP   : in std_logic_vector(n-1 downto 0);
        YT   : in std_logic_vector(n-1 downto 0);
        q0   : in std_logic_vector(m-1 downto 0);
        q1   : in std_logic_vector(m-1 downto 0);
        RDY  : out std_logic;
        UOUT : out std_logic_vector(3 downto 0);
        SIGN : out std_logic
```

```
        );
end PController;

Architecture Structural of PController is

Component Multiplier is
  generic(n : integer := 32; m : integer := 32);
  port(
  OPA : in std_logic_vector(n - 1 downto 0);
  OPB : in std_logic_vector(m - 1 downto 0);
  RES : out std_logic_vector(n + m - 1 downto 0)
  );
end Component;

Component AddSubs is
  generic(n: integer:=8);
  port(
  OPA: in std_logic_vector(n-1 downto 0);
  OPB: in std_logic_vector(n-1 downto 0);
  SIGN: in std_logic;
  RES: out std_logic_vector(n-1 downto 0)
  );
end Component;

Component LoadRegister is
  generic(n : integer := 8);
  port(
  RST: in std_logic; --Reset asincrono
  CLK: in std_logic;--
  LDR: in std_logic;-- ENA
  DIN: in std_logic_vector(n-1 downto 0);--Dato de entrada
  DOUT: out std_logic_vector(n-1 downto 0)--dato de salida
  );
end Component;

Component PMUX is
    generic(n : integer :=8;
            m : integer :=8
            );
    port(
        SEL : in std_logic_vector(1 downto 0);
        EK  : in std_logic_vector(n-1 downto 0);
        EK1 : in std_logic_vector(n-1 downto 0);
        Q0  : in std_logic_vector(m-1 downto 0);
        Q1  : in std_logic_vector(m-1 downto 0);
        QN  : out std_logic_vector(m-1 downto 0);
        EKN : out std_logic_vector(n-1 downto 0)
        );
 end Component;

Component PFSM is
     port(
         RST : in std_logic;
         CLK : in std_logic;
         STR : in std_logic;
         LDI : out std_logic;
```

```
            LDR : out std_logic;
            LDO : out std_logic;
            SEL : out std_logic_vector(1 downto 0);
            RDY : out std_logic
            );
    end Component;

    Component PSaturator is
        port(
        DIN   : in std_logic_vector(27 downto 0);
        DOUT  : out std_logic_vector(4 downto 0);
        SU    : out std_logic;
        SD    : out std_logic
        );
    end Component;

signal LDI, LDR, LDO, SU, SD : std_logic;
signal ERR, EK, EK1, EKN : std_logic_vector(17 downto 0);
signal SQ0, SQ1, QN : std_logic_vector(15 downto 0);
signal MULT : std_logic_vector(33 downto 0);
signal EKQ : std_logic_vector(35 downto 0);
signal SEL : std_logic_vector(1 downto 0);
signal UK1, UK2, SRES : std_logic_vector(35 downto 0);
signal SUK : std_logic_vector(27 downto 0);
signal SOUT: std_logic_vector(4 downto 0);
begin
SUK  <= UK2(35 downto 8);
EKQ  <= MULT(33) & MULT(33) & MULT;
SIGN <= SOUT(4);
UOUT <= SOUT(3 downto 0);
U00 : AddSubs generic map(18) port map(SP, YT, '1', ERR);
U01 : LoadRegister generic map(18) port map(RST, CLK, LDI, ERR, EK);
U02 : LoadRegister generic map(18) port map(RST, CLK, LDI, EK, EK1);
U03 : LoadRegister generic map(16) port map(RST, CLK, LDI, q0, SQ0);
U04 : LoadRegister generic map(16) port map(RST, CLK, LDI, q1, SQ1);
U05 : PMUX generic map(18, 16)port map(SEL, EK, EK1, SQ0, SQ1, QN, EKN);
U06 : Multiplier generic map(18, 16) port map(EKN, QN, MULT);
U07 : AddSubs generic map(36) port map(UK1, EKQ, '0', SRES);
U08 : LoadRegister generic map(36) port map(RST, CLK, LDR, SRES, UK1);
U09 : LoadRegister generic map(36) port map(RST, CLK, LDO, UK1, UK2);
U10 : PSaturator port map(SUK, SOUT, SU, SD);
U11 : PFSM port map(RST, CLK, STR, LDI, LDR, LDO, SEL, RDY);
end Structural;
```

A.1.5 Sistema de comunicación

El sistema de comunicación entre la computadora y el FPGA está dado por tres bloques principales: Un bloque de comunicación UART, otro encargado de dividir y organizar los datos entregados por la computadora y finalmente un bloque encargado de guardar los registros para posteriormente cargarlos en las entradas de los perfiles.

```
Library IEEE;
```

```
use IEEE.std_logic_1164.all;

Entity UART is
    generic(n : integer :=8);
    port(
        CLK    : in std_logic;
        RST    : in std_logic;
        CTO    : in std_logic;
        DIN    : in std_logic_vector(n-1 downto 0);
        RX     : in std_logic;
        TX     : out std_logic;
        RXRDY  : out std_logic;
        IND    : out std_logic;
        DOUT   : out std_logic_vector(n downto 0)
    );
end UART;

Architecture Structural of UART is

Component LoadRegister is
    generic(n : integer :=40);
    port(
    CLK  : in std_logic;
    RST  : in std_logic;
    LDR  : in std_logic;
    DIN  : in std_logic_vector(n-1 downto 0);
    DOUT : out std_logic_vector(n-1 downto 0)
    );
end Component;

Component ParityDetector is
    port(
        DIN  : in std_logic_vector(7 downto 0);
        DOUT : out std_logic
        );
end Component;

Component UARTMUX is
  port(
     DIN  : in std_logic_vector(7 downto 0);
     PAR  : in std_logic;
     SEL  : in std_logic_vector(3 downto 0);
     DOUT : out std_logic
  );
end Component;

Component SRLatch is
  port(
  RST  : in std_logic;
  CLK  : in std_logic;
  SET  : in std_logic;
  CLR  : in std_logic;
  QOUT : out std_logic
  );
end Component;

```

```
Component GenericTimeBase is
    generic( n : integer:= 10);
    port(
        CLK : in std_logic;
        RST : in std_logic;
        ENI : in std_logic;
        ENO : out std_logic
        );
end Component;

Component FreeCounter is
  generic ( n : integer := 2);
  port(
  RST  : in std_logic;
  CLK  : in std_logic;
  ENA  : in std_logic;
  COUT : out std_logic_vector(n downto 0)
  );
end Component;

Component UProgrammableCounter is
  generic ( n : integer := 10);
  port(
  RST : in std_logic;
  CLK : in std_logic;
  ENA : in std_logic;
  RDY : out std_logic
  );
end Component;

Component SPRegister is
  generic(n : integer := 40);
  port(
  RST  : in std_logic;
  CLK  : in std_logic;
  SHF  : in std_logic;
  BIN  : in std_logic;
  DOUT : out std_logic_vector(n - 1 downto 0)
  );
end Component;

Component FallingEdgeDetector is
  port(
  RST  : in std_logic;
  CLK  : in std_logic;
  XIN  : in std_logic;
  LED  : out std_logic;
  XOUT : out std_logic
  );
end Component;

Component RisingEdgeDetector is
  port(
  RST  : in std_logic;
  CLK  : in std_logic;
  XIN  : in std_logic;
```

```
  XOUT : out std_logic
  );
end Component;

signal DPE : std_logic_vector(n - 1 downto 0);
signal SEL : std_logic_vector(3 downto 0);
signal DTA : std_logic_vector(n downto 0);
signal PAR, CLL, RSS, TIC, SET, CLC, RET, CLS, RLD, TCC : std_logic;

begin
RXRDY <= CLS;
    U01 : LoadRegister generic map(8) port map(CLK, RST, CTO, DIN, DPE);
    U02 : ParityDetector port map(DPE, PAR);
    U03 : UARTMUX port map(DPE, PAR, SEL, TX);
    U04 : SRLatch port map(RST, CLK, CTO, CLL, RSS);
    U05 : GenericTimeBase generic map(868) port map(CLK, RSS, '1', TIC);
    U06 : FreeCounter generic map(3) port map(RSS, CLK, TIC, SEL);
    U07 : UProgrammableCounter generic map(11) port map(RSS, CLK, TIC, CLL);


    U08 : FallingEdgeDetector port map(RST, CLK, RX, IND, SET);
    U09 : SRLatch port map(RST, CLK, SET, CLC, RET);
    U10 : GenericTimeBase generic map(414) port map(CLK, RET, '1', CLC);
    U11 : SRLatch port map(RST, CLK, CLC, CLS, RLD);
    U12 : GenericTimeBase generic map(868) port map(CLK, RLD, '1', TCC);
    U13 : SPRegister generic map(9) port map(RST, CLK, TCC, RX, DTA);
    U14 : LoadRegister generic map(9) port map(CLK, RST, CLS, DTA, DOUT );
    U15 : UProgrammableCounter generic map(10) port map(RLD, CLK, TCC, CLS);

  --U16 : GenericTimeBase generic map(100000) port map(CLK, CTO, '1', STT);
end Structural;

Library IEEE;
use IEEE.std_logic_1164.all;

Entity DataDivider is
    port(
        CLK  : in std_logic;
        RST  : in std_logic;
        ENI  : in std_logic;
        ENO  : out std_logic;
        DIN  : in std_logic_vector(31 downto 0);
        DOUT : out std_logic_vector(7 downto 0)
        );
end DataDivider;


Architecture DataFlow of DataDivider is

Component GenericTimeBase is
    generic( n : integer:= 10);
    port(
        CLK : in std_logic;
        RST : in std_logic;
        ENI : in std_logic;
        ENO : out std_logic
```

```
        );
end Component;

Component ProgrammableCounter is
  generic(n : integer := 2);
  port(
  RST: in std_logic;
  CLK: in std_logic;
  ENI: in std_logic;
  SP: in std_logic_vector(n-1 downto 0);
  ENO: out std_logic;
  COUNT: out std_logic_vector(n-1 downto 0)
  );
end Component;

Component DividerMUX is
    port(
        SEL  : in std_logic_vector(2 downto 0);
        DIN1 : in std_logic_vector(7 downto 0);
        DIN2 : in std_logic_vector(7 downto 0);
        DIN3 : in std_logic_vector(7 downto 0);
        DIN4 : in std_logic_vector(7 downto 0);
        DOUT : out std_logic_vector(7 downto 0)
    );
end Component;

Component SRLatch is
  port(
  RST  : in std_logic;
  CLK  : in std_logic;
  SET  : in std_logic;
  CLR  : in std_logic;
  QOUT : out std_logic
  );
end Component;

signal TB, TBN, TBR  : std_logic;
signal SEL : std_logic_vector(2 downto 0);
signal DIN1, DIN2, DIN3, DIN4 : std_logic_vector(7 downto 0);
begin

DIN1 <= DIN(31 downto 24);
DIN2 <= DIN(23 downto 16);
DIN3 <= DIN(15 downto 8);
DIN4 <= DIN(7 downto 0);
ENO  <= TB;

U00 : SRLatch port map(RST, CLK, ENI, TBR, TBN);
U01 : GenericTimeBase generic map(20000)port map(CLK, RST, TBN, TB);
U02 : ProgrammableCounter generic map(3)port map(RST, CLK, TB, "011", TBR, SEL);
U03 : DividerMUX port map(SEL, DIN1, DIN2, DIN3, DIN4, DOUT);

end DataFlow;

Library IEEE;
use IEEE.std_logic_1164.all;
```

```
1226 Entity QRegister is
     port(
1228         CLK    : in std_logic;
             RST    : in std_logic;
1230         ENI    : in std_logic;
             DIN    : in std_logic_vector(7 downto 0);
1232         a      : out std_logic_vector(63 downto 0);
             T      : out std_logic_vector(63 downto 0);
1234         Q1     : out std_logic_vector(31 downto 0);
             QRDY   : out std_logic
1236         );
end QRegister;
1238
Architecture Structural of QRegister is
1240
Component LoadRegister is
1242   generic(n : integer := 8);
   port(
1244   RST: in std_logic; --Reset asincrono
   CLK: in std_logic;--
1246   LDR: in std_logic;-- ENA
   DIN: in std_logic_vector(n-1 downto 0);--Dato de entrada
1248   DOUT: out std_logic_vector(n-1 downto 0)--dato de salida
   );
1250 end Component;

1252 Component QMUX is
     port(
1254         SEL    : in std_logic_vector(3 downto 0);
             VLDR   : out std_logic_vector(11 downto 0)
1256         );
end Component;
1258
Component ProgrammableCounter is
1260   generic(n : integer := 2);
   port(
1262   RST: in std_logic;
   CLK: in std_logic;
1264   ENI: in std_logic;
   SP: in std_logic_vector(n-1 downto 0);
1266   ENO: out std_logic;
   COUNT: out std_logic_vector(n-1 downto 0)
1268   );
end Component;
1270
Component GenericTimeBase is
1272     generic( n : integer:= 10);
     port(
1274         CLK : in std_logic;
             RST : in std_logic;
1276         ENI : in std_logic;
             ENO : out std_logic
1278         );
end Component;
1280
```

```
signal RDY, FEN : std_logic;
signal VLDR : std_logic_vector(11 downto 0);
signal SEL : std_logic_vector(3 downto 0);
signal DOUTa1,DOUTa2,DOUTa3,DOUTa4,DOUTT1,DOUTT2,DOUTT3,DOUTT4,DOUTQ1,DOUTQ2,DOUTQ3,
    DOUTQ4 : std_logic_vector(7 downto 0);
--signal aAux, DOUTaAux1, DOUTT1, DOUTQ1 : std_logic_vector(31 downto 0);
signal aAux, TAux      : std_logic_vector(63 downto 0);
signal  Q1Aux      : std_logic_vector(31 downto 0);
begin

aAux  <= "00000000000000000000000000000000" & DOUTa4 & DOUTa3 & DOUTa2 & DOUTa1;
TAux  <= "11111111111111111111111111111111" & DOUTT4 & DOUTT3 & DOUTT2 & DOUTT1;
Q1Aux <= DOUTQ4 & DOUTQ3 & DOUTQ2 & DOUTQ1;
QRDY <= VLDR(11);
--aAux       <= "00000000000000000000000000000000";
--DOUTaAux2 <= aAux & DOUTaAux1;
    U01 : ProgrammableCounter generic map(4)port map(RST, CLK, ENI, "1100", OPEN, SEL)
    ;
    U02 : QMUX port map(SEL,VLDR);
    U03 : LoadRegister generic map(8) port map(RST, CLK, VLDR(0), DIN, DOUTa1);
    U04 : LoadRegister generic map(8) port map(RST, CLK, VLDR(1), DIN, DOUTa2);
    U05 : LoadRegister generic map(8) port map(RST, CLK, VLDR(2), DIN, DOUTa3);
    U06 : LoadRegister generic map(8) port map(RST, CLK, VLDR(3), DIN, DOUTa4);
    U07 : LoadRegister generic map(8) port map(RST, CLK, VLDR(4), DIN, DOUTT1);
    U08 : LoadRegister generic map(8) port map(RST, CLK, VLDR(5), DIN, DOUTT2);
    U09 : LoadRegister generic map(8) port map(RST, CLK, VLDR(6), DIN, DOUTT3);
    U10 : LoadRegister generic map(8) port map(RST, CLK, VLDR(7), DIN, DOUTT4);
    U11 : LoadRegister generic map(8) port map(RST, CLK, VLDR(8), DIN, DOUTQ1);
    U12 : LoadRegister generic map(8) port map(RST, CLK, VLDR(9), DIN, DOUTQ2);
    U13 : LoadRegister generic map(8) port map(RST, CLK, VLDR(10), DIN, DOUTQ3);
    U14 : LoadRegister generic map(8) port map(RST, CLK, VLDR(11), DIN, DOUTQ4);
    U15 : GenericTimeBase generic map(1000)port map(CLK, RST, VLDR(11), FEN);
    U16 : LoadRegister generic map(64) port map(RST, CLK, FEN,aAux, a);
    U17 : LoadRegister generic map(64) port map(RST, CLK, FEN,TAux, T);
    U18 : LoadRegister generic map(32) port map(RST, CLK, FEN,Q1Aux, Q1);

end Structural;
```

A.1.6 Encoder Incremetal

El siguiente bloque tiene la tarea de tranformar los pulsos entregados por el encoder del servomotor a una cuenta que puede ser incremental o decremental.

```
Library IEEE;
use IEEE.std_logic_1164.all;

Entity IncrementalEncoder is
    generic (n : integer := 32);
    port(
        CLK    : in std_logic;
        RST    : in std_logic;
        CHA    : in std_logic;
```

```
        CHB     : in std_logic;
        PULSE   : out std_logic;
        DIR     : out std_logic;
        COUNT   : out std_logic_vector(n-1 downto 0)
        );
end IncrementalEncoder;

Architecture Structural of IncrementalEncoder is

Component BidirectionalCounter is
    generic (n : integer:= 32);
    port(
        CLK : in std_logic;
        RST : in std_logic;
        ENI : in std_logic;
        DIR : in std_logic;
        SP  : in std_logic_vector(n-1 downto 0);
        COUNT : out std_logic_vector(n-1 downto 0)
        );
end Component;

Component QuadratureDecoder is
  generic( n : integer := 3);
  port(
  RST   : in std_logic;
  CLK   : in std_logic;
  CHA   : in std_logic;
  CHB   : in std_logic;
  ENA   : out std_logic;
  DIR   : out std_logic;
  LED   : out std_logic
  );
end Component;

signal DIRECTION,PULSEB, ENA : std_logic;
begin
    DIR <= DIRECTION;
    PULSE <= ENA;
    U01 : QuadratureDecoder generic map(3) port map(RST, CLK, CHA, CHB, ENA, DIRECTION
    , OPEN);
    U02 : BidirectionalCounter  generic map(32) port map(CLK, RST, ENA, DIRECTION, "
    11111111111111111111111111111111", COUNT);
end Structural;
```

A.1.7 Indicador digital de la posición angular

El siguiente bloque es importante para el seguimiento de la posición del motor entregada por el encoder y su despliegue en hasta 8 displays de 7 segmentos.

```
Library IEEE;
use IEEE.std_logic_1164.all;

```

```
Entity DMillionCounter is
    port(
        CLK  : in std_logic;
        RST  : in std_logic;
        ENI  : in std_logic;
        DIR  : in std_logic;
        AOUT : out std_logic_vector(7 downto 0);
        SOUT : out std_logic_vector(6 downto 0)
        );
end DMillionCounter;

Architecture Structural of DMillionCounter is

Component SevenSegDisplay is
  port(
     DIN  : in std_logic_vector(3 downto 0);
     DOUT : out std_logic_vector(6 downto 0)
      );
end Component;

Component GenericTimeBase is
    generic( n : integer:= 10);
    port(
        CLK : in std_logic;
        RST : in std_logic;
        ENI : in std_logic;
        ENO : out std_logic
        );
end Component;

Component BidirectionalCounter is
    generic (n : integer:= 16);
    port(
        CLK : in std_logic;
        RST : in std_logic;
        ENI : in std_logic;
        DIR : in std_logic;
        SP  : in std_logic_vector(n-1 downto 0);
        COUNT : out std_logic_vector(n-1 downto 0);
        RDY   : out std_logic
        );
end Component;

Component IncrementalCounter is
    generic (
        n  : integer:=3
            );
    port(
        CLK : in std_logic;
        RST : in std_logic;
        ENI : in std_logic;
        RDY : out std_logic;
        SP  : in std_logic_vector(n-1 downto 0);
        COUNT : out std_logic_vector(n-1 downto 0)
        );
end Component;
```

```
Component BCDMUX is
    port(
        SEL  : in std_logic_vector(3 downto 0);
        U    : in std_logic_vector(6 downto 0);
        D    : in std_logic_vector(6 downto 0);
        C    : in std_logic_vector(6 downto 0);
        UM   : in std_logic_vector(6 downto 0);
        DM   : in std_logic_vector(6 downto 0);
        CM   : in std_logic_vector(6 downto 0);
        UMM  : in std_logic_vector(6 downto 0);
        DMM  : in std_logic_vector(6 downto 0);
        SOUT : out std_logic_vector(6 downto 0);
        AOUT : out std_logic_vector(7 downto 0)
        );
end Component;
signal TIC,P : std_logic;
signal TD, TC, TUM, TDM, TCM, TUMM, TDMM : std_logic;
signal SEL, IU, ID, IC, IUM, IDM, ICM, IUMM, IDMM : std_logic_vector(3 downto 0);
signal SU, SD, SC, SUM, SDM, SCM, SUMM, SDMM : std_logic_vector(6 downto 0);


begin
    --AOUT <= "10000000";
    --SOUT <= "1001001";
    U01  : GenericTimeBase generic map(5000) port map(CLK, RST , '1', TIC );
    U02  : BidirectionalCounter generic map(4) port map(CLK, RST, ENI, DIR, "1001", IU
    , TD);
    U03  : SevenSegDisplay port map (IU, SU);
    U04  : BidirectionalCounter generic map(4) port map(CLK, RST, TD, DIR, "1001", ID,
     TC);
    U05  : SevenSegDisplay port map (ID, SD);
    U06  : BidirectionalCounter generic map(4) port map(CLK, RST, TC, DIR, "1001", IC,
     TUM);
    U07  : SevenSegDisplay port map (IC, SC);
    U08  : BidirectionalCounter generic map(4) port map(CLK, RST, TUM, DIR, "1001",
    IUM, TDM);
    U09  : SevenSegDisplay port map (IUM, SUM);
    U10  : BidirectionalCounter generic map(4) port map(CLK, RST, TDM, DIR, "1001",
    IDM, TCM);
    U11  : SevenSegDisplay port map (IDM, SDM);
    U12  : BidirectionalCounter generic map(4) port map(CLK, RST, TCM, DIR, "1001",
    ICM, TUMM);
    U13  : SevenSegDisplay port map (ICM, SCM);
    U14  : BidirectionalCounter generic map(4) port map(CLK, RST, TUMM, DIR, "1001",
    IUMM, TDMM);
    U15  : SevenSegDisplay port map (IUMM, SUMM);
    U16  : BidirectionalCounter generic map(4) port map(CLK, RST, TDMM, DIR, "1001",
    IDMM, OPEN);
    U17  : SevenSegDisplay port map (IDMM, SDMM);
    U18  : IncrementalCounter generic map(4) port map(CLK, RST, TIC, OPEN, "1000", SEL
    );
    U19  : BCDMUX port map(SEL, SU, SD, SC, SUM, SDM, SCM, SUMM, SDMM, SOUT, AOUT);
   --U20  : GenericTimeBase generic map(1000000) port map(CLK, RST , '1', P );

end Structural;
```

A.1.8 Bloques en común para las funciones principales

Todas las descripciones de harware anteriores basan su funcionalidad en la integración y activación de pequeños bloques elementales como lo son registros de carga, bases de tiempo, contadores genéricos y multiplexores. A continuación se muestra el código de los bloques genéricos. Aquellos bloques que sean específicos como multiplexores o o máquinas de estado especializadas no se incluyen en este trabajo pero son fácilmente deducibles a través de la comprensión y análisis de los diagramas del mismo.

A.1.8.1 Registro de carga

```
1000 Library  IEEE;
     use IEEE.std_logic_1164.all;
1002
     Entity LoadRegister is
1004   generic(n : integer := 8);
       port(
1006   RST: in std_logic; --Reset asincrono
       CLK: in std_logic;--
1008   LDR: in std_logic;-- ENA
       DIN: in std_logic_vector(n-1 downto 0);--Dato de entrada
1010   DOUT: out std_logic_vector(n-1 downto 0)--dato de salida
       );
1012 end LoadRegister;

1014 Architecture Behovial of LoadRegister is
     signal Qp, Qn: std_logic_vector(n-1 downto 0);
1016 begin
       Combinational: process(LDR, Qp, DIN)
1018   begin
         if LDR='1' then
1020       Qn<=DIN;
         else
1022       Qn<=Qp;
         end if;
1024     DOUT<=Qp;
       end process Combinational;
1026   Sequential : process(RST, CLK)
       begin
1028     if RST= '0' then
           Qp<=(others  => '0');
1030     elsif CLK'event and CLK='1' then
           Qp<=Qn;
1032     end if;
         end process Sequential;
1034   end Behovial;
```

A.1.8.2 Base de tiempo

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

Entity GenericTimeBase is
    generic( n : integer:= 10);
    port(
        CLK : in std_logic;
        RST : in std_logic;
        ENI : in std_logic;
        ENO : out std_logic
        );
end GenericTimeBase;

Architecture Behavioral of GenericTimeBase is
signal Qp, Qn : integer;
signal CMP : std_logic;
signal SEL : std_logic_vector(1 downto 0);
    begin

        Combinational: process(Qp, CMP, SEL, ENI) is
            begin
                if (Qp = (n-1)) then
                    CMP <= '1';
                else
                    CMP <= '0';
                end if;
                SEL <= ENI & CMP;

                case SEL is
                    when "10" => Qn <= Qp + 1;
                    when "11" => Qn <= 0;
                    when others => Qn <= Qp;
                end case;

                ENO <= ENI and CMP;

            end process Combinational;

        Sequential : process(CLK, RST) is
            begin
                if RST = '0' then
                    Qp <= 0;
                elsif CLK = '1' and CLK'event then
                    Qp <= Qn;
                end if;
            end process Sequential;

    end Behavioral;
```

A.1.8.3 Contador incremental

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

Entity IncrementalCounter is
    generic (
        n  : integer:=3
             );
    port(
        CLK : in std_logic;
        RST : in std_logic;
        ENI : in std_logic;
        RDY : out std_logic;
        SP  : in std_logic_vector(n-1 downto 0);
        COUNT : out std_logic_vector(n-1 downto 0)
        );
end IncrementalCounter;


Architecture Behavioral of IncrementalCounter is
    signal Qp, Qn : std_logic_vector(n-1 downto 0);
    begin
    Combinational : process(ENI,Qp, Qn, SP) is
        begin
            if Qp = SP then
                Qn <= (others => '0');
                RDY <= '1';
            else
                if ENI = '1' then
                    Qn <= Qp + 1;
                    RDY <= '0';
                else
                    Qn <= Qp;
                    RDY <= '0';
                end if;
            end if;
            COUNT <= Qp;

    end process Combinational;

    Sequential : process(CLK,RST) is
        begin
        if RST = '0' then
            Qp <= (others => '0');
        elsif CLK = '1' and CLK'event then
            Qp <= Qn;
        end if;
    end process Sequential;
end Behavioral;
```

A.1.8.4 Contador programable

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

Entity ProgrammableCounter is
  generic(n : integer := 2);
  port(
  RST: in std_logic;
  CLK: in std_logic;
  ENI: in std_logic;
  SP: in std_logic_vector(n-1 downto 0);
  ENO: out std_logic;
  COUNT: out std_logic_vector(n-1 downto 0)
  );
end ProgrammableCounter;

Architecture Behavioral of ProgrammableCounter is
signal Qn,Qp: std_logic_vector(n-1 downto 0);
begin
  Combinational: process(Qp,ENI,SP)
  begin
    if ENI = '1' then
      if Qp = SP then
        ENO <= '1';
        Qn <= (others => '0');
      else
        ENO <= '0';
        Qn <= Qp + 1;
      end if;
    else
      Qn <= Qp;
      ENO <= '0';
    end if;

    COUNT <= Qp;

  end process Combinational;

  Sequential: process(CLK,RST)
  begin
    if RST = '0' then
      Qp <= (others => '0');
    elsif CLK'event and CLK= '1' then
      Qp <= Qn;
    end if;

  end process Sequential;

end Behavioral;
```

A.1.8.5 Sumador y restador

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;


Entity AddSubs is
  generic(n: integer:=8);
  port(
  OPA: in std_logic_vector(n-1 downto 0);
  OPB: in std_logic_vector(n-1 downto 0);
  SIGN: in std_logic;
  RES: out std_logic_vector(n-1 downto 0)
  );
end AddSubs;

Architecture Behavioral of AddSubs is
begin
  Combinational: process(OPA,OPB,SIGN)
  begin
    if SIGN = '0' then
      RES <= signed(OPA) + signed(OPB);

      else
      RES <= signed(OPA) - signed(OPB);
    end if;


  end process Combinational;

end Behavioral;
```

A.1.8.6 Multiplicador

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

Entity Multiplier is
  generic(n : integer := 32; m : integer := 32);
  port(
  OPA : in std_logic_vector(n - 1 downto 0);
  OPB : in std_logic_vector(m - 1 downto 0);
  RES : out std_logic_vector(n + m - 1 downto 0)
  );
end Multiplier;

Architecture DataFlow of Multiplier is
begin
  RES <= signed(OPA) * signed(OPB);
end DataFlow;
```

A.2 Código de la interfaz desarrollada en MATLAB

La interfaz de usuario del experimento fue desarrollada con la herramienta de interfaces de usuario de MATLAB GUI que funciona a través de la programación de eventos (como presionar un botón), sin embargo, al tratarse de una gran cantidad de líneas de código se deciden no incluir en el presente trabajo. Además de que este sistema puede ser adaptado a interfaces desarrolladas en otras plataformas que soporten el protocolo de comunicación RS232. Por otro lado, sí se incluyen las funciones que permiten calcular los vectores de posición, velocidad y aceleración para cada perfil y una función que permite convertir los valores decimales a palabras binarias con formato de punto fijo necesarias para que el FPGA pueda trabajar.

A.2.1 Código del perfil parabólico con integración rectangular

```
function [time,pos,speed,accel,Tr,a] = ParabolicProfile( wmax,Thetaf,Ts )

Tr=3*Thetaf/(2*(wmax))
samples=floor(Tr*(1/Ts))
T=samples/(1/Ts)
a=4*(wmax)/T


time=zeros(samples,1);
accel=zeros(samples,1);
speed=zeros(samples,1);
pos=zeros(samples,1);

for K=2:samples
   time(K)=Ts*K;
   accel(K)=-(2*a/T)*time(K)+a;
   speed(K)=speed(K-1)+accel(K)*Ts;
   pos(K)=pos(K-1)+speed(K)*Ts;
end

end
```

A.2.2 Código del perfil trapezoidal con integración rectangular

```
function [time,pos,speed,accel,Tr,a] = TrapezoidalVelocityProfile(wmax,Thetaf,Ts)
Tr=3*Thetaf/(2*wmax);
samples=floor(Tr*(1/Ts));
T=samples/(1/Ts);
a=3*wmax/T;
```

```
time=zeros(samples,1);
accel=zeros(samples,1);
speed=zeros(samples,1);
pos=zeros(samples,1);

for K=2:samples
  time(K)=Ts*K;

  if K>0 && K<samples/3
  accel(K)=a;
  end

  if K>=samples/3 && K<(2*samples)/3
  accel(K)=0;
  end

  if K>=(2*samples)/3 && K<=samples
  accel(K)=-a;
  end
  speed(K)=speed(K-1)+accel(K)*Ts;
  pos(K)=pos(K-1)+speed(K)*Ts;
end
end
```

A.2.3 Código del convertidor decimal a punto fijo

```
function [ binaryWord, HexStr ] = dec2fixedPoint( decimalNumber,FI,FF )


    %SEPARACIÓN DE PARTE ENTERA Y FRACCIONAL
    counter = 1;
    integer       = fix(decimalNumber);
    absInteger    = abs(integer);
    fractional    = abs(decimalNumber-integer);
    binaryInteger = dec2bin(absInteger,FI);

    if FF == 0
        binaryWord = dec2bin(absInteger,FI);
    else
    %ALGORITMO PARA TRASFORMAR FRACCIÓN DECIMAL A BINARIA
    while(counter <= FF)
        fractional = fractional * 2;
        binaryFractional(counter) = fix(fractional);
        fractional = abs(fractional - fix(fractional));
        counter = counter + 1;
    end
    binaryFractional = num2str(binaryFractional);
    binaryFractional = binaryFractional(~isspace(binaryFractional));
    counter = 1;

    %CONCATENAR PARTE ENTERA Y FRACCIONAL
    binaryWord = strcat(binaryInteger,binaryFractional);
```

```
1026    FC = FI + 1;
        %COMPLEMENTO A 1 EN CASO DE SER UN DECIMAL NEGATIVO
1028    if (decimalNumber < 0)
            while (FC) <= (FI + FF)
1030            if(binaryWord(counter) == '0')
                    binaryWord(counter) = '1';
1032            else
                    binaryWord(counter) = '0';
1034            end

1036            if(binaryWord(FC) == '0')
                    binaryWord(FC) = '1';
1038            else
                    binaryWord(FC) = '0';
1040            end

1042            if counter <= FI
                    counter = counter + 1;
1044            end
                FC = FC + 1;
1046        end
        % COMPLEMENTO A 2
1048        binaryWord = binaryWord -'0';
            binaryWord = uint64(bi2de(binaryWord,'left-msb'));
1050        binaryWord = dec2bin(binaryWord + 1, FI + FF);

1052    end
        end
1054    BHWord = binaryWord;
        % CONVERSIÓN A HEXADECIMAL
1056    Bits = FI + FF;
        Words = Bits/4;
1058    WholeWords = floor(Words);
        PartWords = Words-WholeWords;
1060
        while PartWords > 0
1062        BHWord = ['0' BHWord];
            Bits = length(BHWord);
1064        Words = Bits/4;
            WholeWords = floor(Words);
1066        PartWords = Words-WholeWords;
        end
1068
        Words = length(BHWord)/4;
1070    HEX = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
        counter = 1;
1072    HexStr = [];
        while counter <= Words
1074        position = (counter * 4) - 3;
            Word = BHWord(position:position + 3);
1076        DEC = bin2dec(Word);
            HEXl = HEX{DEC + 1};
1078        HexStr = [HexStr HEXl];
            counter = counter + 1;
1080    end
```

```
1082 end
```