UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

INSTITUTO DE INGENIERÍA

MAESTRÍA Y DOCTORADO EN CIENCIAS E INGENIERÍA

«MÉTODO TEÓRICO PARA DISMINUIR EL ERROR DE POSICIONAMIENTO EN UN SISTEMA DE BARRIDO LÁSER USANDO UN CONTROLADOR DIGITAL EMBEBIDO Y MOTORES DE CORRIENTE DIRECTA»

TESIS PARA OBTENER EL GRADO DE:

MAESTRÍA EN CIENCIAS

PRESENTA

MIGUEL REYES GARCÍA

Dr. OLEG SERGIYENKO

Dr. LARS LINDNER

Mexicali, B.C. *Agosto de 2019*

Contenido

Agradecimientos

Les agradezco a todo el Cuerpo Académico de Optoelectrónica y Mediciones Automáticas de la Universidad Autónoma de Baja California, especialmente al Dr. Oleg Sergiyenko y al Dr. Lars Lindner por su gran apoyo en mi formación académica durante mi estancia dentro del Instituto de Ingeniería. Para mis padres Leticia y Miguel quienes me brindaron su apoyo moral para concluir mis estudios de posgrado. También, agradezco a mis hermanos Ricardo y Roxana Jocelin por creer en mi proyecto de vida. De igual manera, para mis amigos Jesús Alfredo y Alejandro quienes me recordaban eventualmente que las metas son logradas a través del esfuerzo, compromiso y la constancia.

Resumen

De la tesis de Miguel Reyes García, presentada como requisito parcial para la obtención del grado de MAESTRIA EN CIENCIAS, EN EL ÁREA DE FÍSICA APLICADA, Mexicali, Baja California, México. Agosto del 2019.

MÉTODO TEÓRICO PARA DISMINUIR EL ERROR DE POSICIONAMIENTO EN UN SISTEMA DE BARRIDO LÁSER, USANDO UN CONTROLADOR DIGITAL EMBEBIDO Y MOTORES DE CORRIENTE DIRECTA

Resumen aprobado por:

_____

Dr. Oleg Sergiyenko

Dentro del laboratorio de optoelectrónica y mediciones automáticas, el cual forma parte del área de Ingeniería Física en el departamento de Física Aplicada del Instituto de Ingeniería de la Universidad Autónoma de Baja California, se encuentra en fase de desarrollo en su versión núm. 3, un sistema de barrido láser nombrado como «Technical Vision System» (TVS). Este prototipo tiene la condición de obtener coordenadas tridimensionales de objetos físicos que interfieren en su campo de visión, aprovechando un método fundamentado matemáticamente e innovador llamado «Triangulación Dinámica», propuesto por los mismos integrantes que conforman el Cuerpo Académico de Optoelectrónica y Mediciones Automáticas. Conforme a investigaciones previas del TVS prototipo núm. 3, usa motores de corriente continua (CC) con escobillas, para sus dos principales partes que en conjunto definen coordenadas espaciales; las dos partes son: Posicionador Láser (PL) y Apertura de Escaneo (AE). El enfoque de esta tesis está en el PL, debido a estar usando una configuración en lazo cerrado de una implementación previa para el sistema de control del PL con un controlador digital proporcional, demostrando inestabilidad y poca precisión, operando a desplazamientos angulares cortos menores a los cinco grados, demostrando una reducción significativa de tiempo de posicionamiento y un error de posicionamiento relativo menor al 1 por ciento. En consecuencia el trabajo de esta tesis consistió en disminuir el error promedio de posicionamiento angular relativo para el PL. Para lo anterior, metodologías teóricas innovadoras son presentadas, experimentadas para el PL del TVS prototipo núm. 3, llevando

a cabo una implementación de un sistema de control en lazo cerrado usando el controlador LM629N-8, el cual es idóneo para ejercer movimientos precisos a sistemas articulados que integran servomecanismos industriales. Alcanzando una reducción de error de posicionamiento relativo promedio menor al once por ciento, evaluando estabilidad, exactitud, alta precisión y tiempos de posicionamiento satisfactorios menores a los 500 *ms*. Además de lo anterior, se desarrolló una Interfaz Gráfica de Usuario (GUI por Graphic User Interface), con el beneficio de presentar gráficamente las posiciones angulares finales configuradas arbitrariamente por quien opere la GUI, de igual forma visualizar la trayectoria de la velocidad de referencia en forma trapezoidal, generada por el LM629N-8. Esta velocidad es asignada a un motor Maxon DCX22S con un sensor de posición incremental, el motor Maxon DCX22S es aplicado en alguna de las dos principales partes ya mencionadas del TVS núm. 3. Resaltando los resultados experimentales hechos del PL, usando un motor Maxon DCX22S conforme a un diseño de experimentación por la metodología de Taguchi para determinar los factores significativos involucrados para observar estabilidad y calidad al sistema de control en lazo cerrado propuesto para el PL acorde a este trabajo de tesis.

Abstract

Of the thesis presented by Miguel Reyes García, in order to obtain the degree of MASTER SCIENCIES, ENGINEERING PHYSIC AREA, Mexicali, Baja California, Mexico. August, 2019.

THEORETICAL METHOD TO REDUCE THE POSITIONING ERROR IN A SCANNING LASER SYSTEM, USING AN EMBEDDED DIGITAL CONTROLLER AND DIRECT CURRENT MOTORS

Abstract approved by:

_____

Dr. Oleg Sergiyenko

In the laboratory of optoelectronics and automatic measurements, which is part of the Engineering Physic area in the Department of Applied Physics at the Engineering Institute of UABC, there is a Scanning Laser System is developed, named "Technical Vision System" (TVS) prototype No. 3, which complies the condition to obtain 3D coordinates of physic objects that interfere inside the TVS Field of View (FOV). About the above mentioned, it is taken the advantage of a fundamental mathematical method, called "Dynamic Triangulation", and proposed by the members of Optoelectronics and Automatic Measurements. According of previous research of the TVS No. 3, which used brushed DC electric motors for its two main parts to determine spatial coordinates, those are: Laser Positioner (LP) and Scanning Aperture (SA). The principal approach of this thesis is applied for the Laser Positioner, whose previous configuration used a Closed-Loop control system, using a Proportional Algorithm (P-Algorithm), evidencing instability and low precision of the LP and operating to small angular displacements < 5 degrees, setting a significant reduction of the positioning time, and relative positioning error lower than 1 percent. Thus, the main contribution of this thesis is the reduction of the relative positioning error average. Novel experimental and theoretical methodologies are demonstrated for the TVS prototype No.3, conducting a implementation of the Closed-loop Control System, using the LM629N-8 controller, which is ideal to perform precise motions to articulated systems that integrate industrial servomechanisms. Achieving a reduction of the relative positioning error average

lower than eleven percent, evaluating stability, accuracy, and high precision and satisfactory positioning times lower than 500 milliseconds. Additionally about the above, a Graphic Interface User (GUI) is developed, whose purpose is to present the arbitrary and configured final angular positions graphically. In the same way the profile of trapezoidal velocity trajectory is displayed, which is generated by the LM629N-8, whose velocity is proved to Maxon brushed DCX22S motor applied to some of the two main parts mentioned of the TVS No. 3. The experimental results are evaluated with experimental design using the Taguchi methods to determine the significant factors to adjust stability and quality in the proposed control system.

Lista de Figuras

Lista de Tablas

Lista de Ecuaciones

Objetivos

El objetivo general de este proyecto de maestría en ciencias, es disminuir el error de posicionamiento del Posicionador Láser (PL) del Technical Vision System (TVS) prototipo núm.3, el cual es controlado en lazo cerrado, usando un perfil de velocidad en forma trapezoidal generado por el controlador LM629N-8. El principal actuador es el motor Maxon DCX22S, además el error de posicionamiento será relativo promediado y evaluado conforme a un diseño de experimentación definido por la metodología de Taguchi. Para lograr y comprender este objetivo general, a continuación se enumeran los objetivos específicos a obtener:

1. Proponer un método teórico del controlador digital LM629N-8 en lazo cerrado. Este método fija los parámetros que constituyen el perfil de la trayectoria de velocidad trapezoidal de referencia. Calculando la posición final del eje del DC motor Maxon DCX22S al integrar la velocidad.

2. Modelar y simular el funcionamiento del controlador digital LM629N-8 usando Simulink de Matlab. Evaluando los parámetros del método teórico para obtener una respuesta satisfactoria de la trayectoria de velocidad trapezoidal para el motor Maxon DCX22S.

3. Diseñar la programación orientada al software para la Interfaz Gráfica de Usuario (GUI), cual forma parte de la implementación del sistema de control usando el controlador embebido LM629N-8.

4. Diseñar la programación orientada al software para la Interfaz Hardware (Puerta de enlace), el cual actúa como intercomunicador entre la GUI y el controlador LM629N-8.

5. Desarrollar integración de tarjetas electrónicas para la implementación de hardware definido al diseño del controlador LM629N-8, para eliminar falsos contactos con el uso de tarjetas de prueba.

6. Diseñar la experimentación usando la metodología de Taguchi, para obtener factores significativos del desempeño de la implementación del sistema de control para el Posicionador Láser del TVS No. 3.

1. Introducción

En relación a investigaciones previas a sistemas de visión artificial, [1], [2], [3], [4] se presentan trabajos que dan difusión a fundamentos teóricos que constituyen a los mismos sistemas, además se describen a detalle las experimentaciones hechas y los resultados obtenidos. Demostrando el alcance y limitaciones de diferentes sistemas de visión usando métodos y técnicas de medición particulares [5]. Diversas aplicaciones pueden ser realizables tales como, por ejemplo: inspección automática de un objeto físico, control de calidad a procesos industriales, reconstrucción de objetos para ser modelados virtualmente, monitoreo de salud estructural y de parcelas de productos orgánicos, soporte en el estudio de biometría, solo por nombrar unas pocas. Estas aplicaciones funcionales por un sistema de visión artificial llamado «Technical Vision System» (TVS). Este TVS es capaz de obtener mediciones de coordenadas espaciales [6] y también hacer mediciones del centro energético de la señal emitida por una fuente de luz reflejada a objetos físicos [7], [8] que interfieren dentro del campo de visión del mismo TVS, usando Triangulación Dinámica, el cual representa un método matemático formalizado en [9], [10]. Actualmente, este TVS se presenta en su versión prototipo núm. 3 (TVS No.3) [11]. El TVS No.3 se dispone, a ser parte principal de los robots autónomos móviles [12], un ejemplo de lo anterior, es que se pueden ver en los vehículos aéreos no tripulados [13].

Un reto ambicioso es obtener métodos innovadores para llevar a cabo mediciones automáticas de objetos de estudio con alta calidad y simplicidad, usando implementaciones basadas a visión artificial. Midiendo a detalle las distancias entre los relieves geométricos que forman la superficie de un objetos físico. Ya que, los trabajos de trascendencia científica son los que aportan nuevo conocimiento presentando el método teórico y práctico. En el caso del TVS No, 3, es obtener exactitud, alta precisión a las coordenadas espaciales definidas en relación a un tiempo de posicionamiento óptimo.

Por lo tanto, en las investigaciones hechas en [2], [14], [11], [13], [15] el TVS No.3 ha demostrado resultados significativos, usando motores de Corriente Continua (motores CC), para sus dos principales partes que determinan las coordenadas espaciales. Esas partes son: Posicionador Láser (PL) y Apertura de Escaneo (AE). El TVS No.3 ha demostrado tener poca precisión e inestabilidad en su componente activo (PL), el cual emite un rayo láser, con proyección dentro del campo de visión del TVS No.3. Particularmente en la sección de «Experimentation Analysis» de la tesis «Theoretical method to increase the speed of continuous mapping in a three-dimensional laser scanning system using servomotors control» da a conocer una mejora significativa del error promedio angular relativo, así como

también el tiempo de subida promedio a la señal escalón referencia para definir la posición del rayo láser. Esta mejora proviene del promedio de 5 pruebas hechas usando 6 factores experimentales, cuales se colocaron en 15 arreglos hechos para el PL del TVS No.3 conforme a un diseño de experimentación [11]. La Tabla 1.1, muestra la mejora significativa como antecedente a esta tesis. Donde, el PL fue controlado por un controlador proporcional digital en lazo cerrado con el factor de amplificación a 500 unidades, usando la tarjeta Arduino Uno basado en el circuito integrado ATmega328P, el módulo L298 puente H y el motor Maxon RE-max29 [16]. Este motor tiene dos ventajas, al ser comparado con otros motores de baja calidad. Debido a que este motor no contiene hierro en el núcleo del rotor, eliminando el par de torsión (cogging torque), es decir eliminando la interacción entre los imanes del estator y el rotor, y la reducción del par de arranque del eje del rotor usando rodamientos de bolas [16].

Tabla 1.1. Respuestas de la mejora significativa del Posicionador Láser

Motor RE-Max 29 Maxon					**Unidad**
Posición angular de referencia φ_r	1	2	3	90	*deg*
Error promedio de posicionamiento angular relativo $\bar{\varphi}'_e$	11.60	11.11	4.16	0.22	*%*
Tiempo de subida promedio en respuesta al escalón referencia	21	45	87	108	*ms*

Los resultados mostrados en la Tabla 1.1, fueron a consecuencia de ajustar los siguientes factores: datos enteros al algoritmo proporcional; debido a mayor número de dígitos para representar un entero frente a los datos flotantes, usando aritmética de punto fijo para la conversión de números reales a enteros a la magnitud del Pulse-Width Modulation (PWM); modo de operación del Timer 1 a un PWM en modo rápido; al ajustar una resolución máxima de 16,000 cuentas a la amplitud de la magnitud de la señal PWM, balanceando la condición mutuamente excluyente, al mantener una alta frecuencia y alta resolución de ciclo de trabajo de la señal del PWM; una frecuencia de 500 Hz en la ejecución del algoritmo proporcional, permitiendo obtener la señal de error de posición actual cada 2 *ms*, y finalmente; una resolución de algoritmo proporcional a tres dígitos [11] para cada número real convertido a entero, por cada número real almacenado.

Las posiciones angulares de referencia son 1, 2, 3 y 90 grados como ángulos pequeños y uno mediano respectivamente, para el eje del motor. Observando los errores de posicionamiento angular relativo fueron decreciendo a partir de $\bar{\varphi}'_e = 11.60\ \%$, conforme a la amplitud del ángulo de referencia crece, notando efectos de la fricción que causan movimientos no lineales a desplazamientos angulares cortos menores a los 5 grados. El tiempo de subida de respuesta al escalón referencia fue creciendo conforme al incremento del desplazamiento angular como ejemplo, 108 *ms* para una posición de referencia a 90 grados. Adicionalmente, las pruebas anteriores presentan un comportamiento inestable del motor Maxon RE max 29, debido a que

existen respuestas oscilatorias a la señal escalón, como resultado poca precisión en la posición angular final comparada con la posición de referencia.

En consecuencia, las observaciones anteriores surge la idea de usar un controlador LM629N-8 el cual está dedicado para el control de movimiento. Para implementar un sistema de control y usarse en el Posicionador Láser (PL) del TVS prototipo 3. El controlador LM629N-8 tiene las siguientes características notables: Período de muestro del sistema: (256 a 65.536) μs con un reloj de sistema a 8.0 MHz para determinar muestras de posición en un instante de tiempo dado a la posición angular actual, además provee un módulo de decodificación de señales de un sensor de posición incremental para sus canales A, B e Índice; una frecuencia máxima de captura de estados de posicionamiento a 1 MHz. El controlador LM629N-8 genera una trayectoria de velocidad en forma trapezoidal para un motor aplicado.

Tomando como ventaja de esta generación de trayectoria, el rendimiento de la trayectoria de movimiento provee estabilidad al motor aplicado, y las posiciones por desplazamientos angulares cortos son precisos en comparación al trabajo previo consultado en [11]. Además, se desarrolló una Interfaz Gráfica de Usuario (GUI), usando la plataforma de programación visual LabVIEW 2015, para hacer configuraciones a la trayectoria de la velocidad trapezoidal. La GUI permite ajustar parámetros para obtener resultados favorables al ser comparados con la implementación del algoritmo proporcional consultado en [11]. Así mismo, la GUI ofrece todas las funcionalidades del controlador digital LM629N-8, incluyendo la ventaja de visualizar la trayectoria real de la posición angular final obtenida y la trayectoria de la velocidad deseada del motor aplicado.

Esta tesis, está organizada con el siguiente orden: Después de esta introducción, se presenta el capítulo 2 para dar a conocer los antecedentes involucrados a esta tesis explicando, como eran los primeros dos Posicionadores Láser de los TVS prototipos 1 y 2. Luego, el Capítulo 3 es presentando como Marco Teórico, este introduce los conceptos básicos matemáticos para la justificación de las metodologías posteriores y usadas para reducir el error de posicionamiento angular. En seguida, se presenta el Capítulo 4, este trata la metodología usada para el funcionamiento del Posicionador Láser del TVS prototipo 3. Posteriormente llega el capítulo 5, titulado como «Experimentación de la Reducción de Error a la Posición Angular» aquí se explica la integración de Hardware y Software, la ejecución del PL desde la GUI desarrollada y el diseño de experimentación usando la metodología de Taguchi. Luego, el capítulo 6, presenta los Resultados de la Experimentación. El capítulo 7 presenta una comparación del LM629N-8 y del controlador industrial Maxon EPOS 24/1 [16], observando las respuestas de ambos controladores. Finalmente, se presentan, las conclusiones del trabajo total de esta tesis en relación a los objetivos alcanzados, la Bibliografía y las Publicaciones hechas.

2. Antecedentes

A continuación, se presenta una revisión de investigaciones previas a favor del trabajo desarrollado y concentrado en esta tesis. Este trabajo fue totalmente en relación con el Posicionador Láser (PL) del sistema optoelectrónico llamado «Technical Vision System prototype No. 3 (TVS No.3)» mostrado en la Figura 2.1. Por lo anterior, son consultados los artículos: [6], [11], [17], [18] [19] y el libro Machine Vision: Approaches and Limitations [5]. Estos informan características físicas del sistema y características de los sistemas de control para los Posicionadores Láser desarrollados con anterioridad a los prototipos 1 y 2 del TVS. En este capítulo se hace una revisión breve a las características físicas de los Posicionadores Láser previos y también una comparación del comportamiento de los mismos, usando la Tabla 2.1.

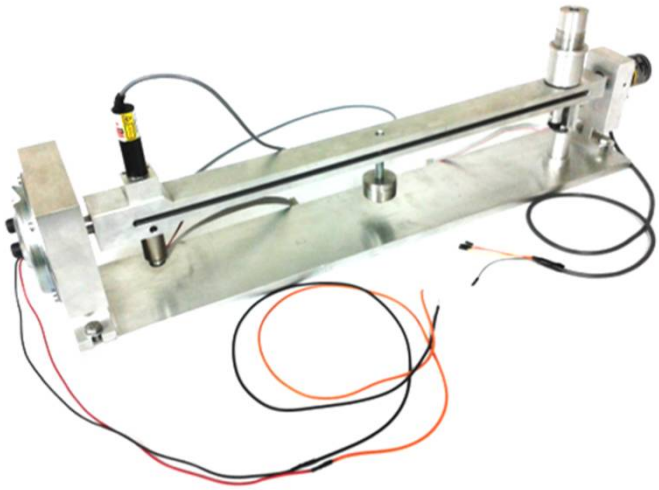

Figura 2.1. Technical Vision System prototype No. 3 (TVS No. 3) [11]

2.1 Investigaciones Previas del Posicionador Láser

La funcionalidad del TVS No. 3 es brevemente visto en la Figura 2.2, donde se determina un triángulo imaginario usando un rayo láser, este triángulo es dinámico al mover la orientación del rayo láser por el PL. Una Apertura de escaneo recibe el rayo láser reflejado por el objeto de estudio, obteniendo información del entorno circundante como por ejemplo coordenadas tridimensionales del mismo objeto. El ángulo γ es para definir la orientación del rayo láser emitido por el PL y el ángulo β es de orientación del rayo láser reflejado y recibido por la AE. La importancia de un Posicionador Láser (PL) es trascendental en los «Laser Scanning

Systems (LSS)», debido a que un PL representa el elemento activo del sistema completo TVS por la acción de explorar un entorno circundante dentro de un campo de visión «Field Of View (FOV)» mostrado en la Figura 2.3. Este es limitado y definido por el mismo TVS debido a que el PL emite un rayo láser controlando su barrido hacia el FOV, inmediatamente surgen los rayos láser reflejados por superficies reflectoras de cuerpos físicos que interfieren dentro del FOV. Estos rayos reflejados son usados por la Apertura de Escaneo (AE), la cual es el elemento pasivo del TVS y sirve para obtener información de la geometría, e identifica inclusive el tipo de color de esos objetos de estudio [20], derivando nuevos métodos de medición por parte del TVS [5], tales como: Inspección automática de un objeto físico, control de calidad a procesos industriales, reconstrucción de objetos para ser modelados virtualmente, monitoreo de salud estructural y cultivos de productos orgánicos, soporte en el estudio de biometría, solo por nombrar unas pocos. Sin embargo, para hacer la exploración continua hacia el FOV visto en la Figura 2.3 donde el ángulo β describe el desplazamiento del láser reflejado en la apertura de escaneo y el ángulo γ representa la posición angular del rayo láser emitido, el cual requiere estabilidad, precisión, exactitud y tiempos cortos de posicionamiento para el sistema de control del PL. Las cualidades anteriores son discutidas en el capítulo 3 «Marco Teórico».

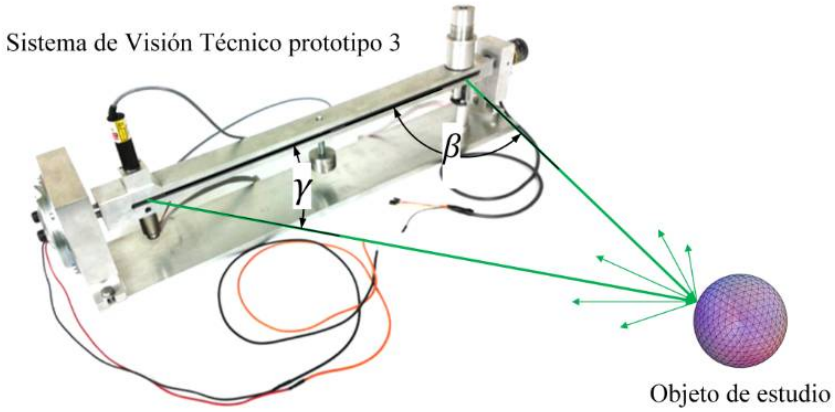

Figura 2.2. TVS No. 3 con su funcionalidad

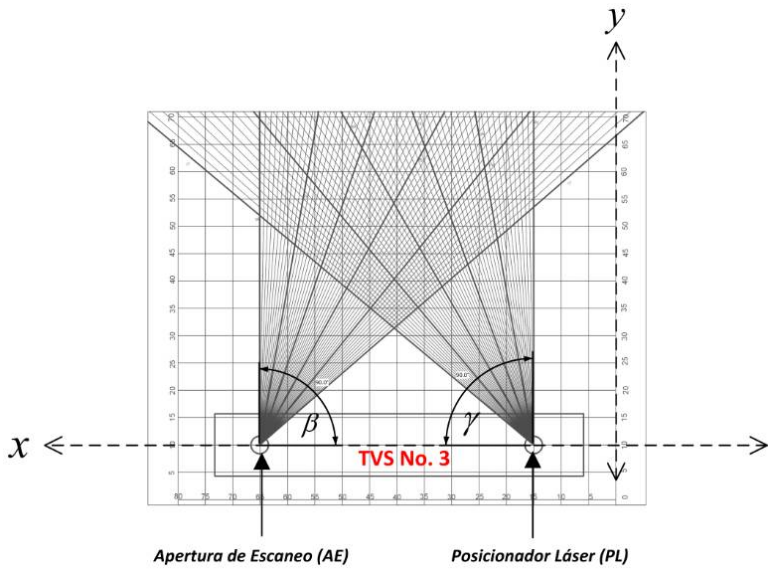

Figura 2.3. Campo de visión para el TVS No. 3

Los Posicionadores Láser desarrollados para los prototipos 1 y 2 del TVS son mostrados en la Figura 2.4. Las imágenes en A y B muestran el PL del TVS prototipo 1, observando un bloque cilíndrico acostado horizontalmente de color metálico, este es el cabezal de un módulo láser (en círculo superior rojo en la imagen) que proyecta un rayo láser con orientación perpendicular en relación con el plano horizontal del cabezal. De esa forma es posible proyectar un rayo láser hacia un componente con superficie reflectora y para ello, se usa un espejo inclinado a 45 grados en relación con el plano horizontal del sistema con la finalidad de cambiar la dirección de la proyección del rayo láser emitiéndolo a 90 grados respecto al sentido vertical original del rayo láser y ser proyectado finalmente al frente del TVS en su campo de visión FOV al entorno circundante. Para lo anterior, el espejo es de un material sólido y acoplado directamente al eje de un motor de pasos, este espejo está montado a una base de material solido color blanco. Para la rotación del TVS se usó una polea acoplada al eje de un motor de pasos (círculo rojo inferior en la imagen B para fijar una banda entre la polea del motor y un anillo fijo exterior acoplado concéntricamente a la barra principal del TVS sincronizando el movimiento del eje del motor de pasos a la barra principal del TVS. En la imagen C se muestra un aspecto general de lo que consiste el PL del TVS prototipo 2, en este se utilizan transmisiones con engranes (en círculos rojos) ampliando la resolución del posicionamiento angular además para dar redirección del rayo láser emitido por un módulo láser acostado de la misma forma que el primer prototipo. También se utiliza un motor de pasos para la rotación del TVS prototipo 2.

Los sistemas de posicionamiento láser desarrollados en los prototipos 1 y 2 en la Figura 2.4, han presentado comportamientos estables y con tiempos de posicionamiento largos

detectados en los experimentos hechos y demostrados en [11]. Como una evidencia de esos comportamientos observados, la Tabla 2.1 muestra una comparación de algunas ventajas y desventajas entre los prototipos construidos del PL para el TVS.

Tabla 2.1. Cuadro comparativo de los prototipos para el PL del TVS

Componentes principales para el Posicionador Láser	Desventajas	Ventajas
Usando motores de pasos con un acoplamiento directo. (PL de TVS prototipo No. 1)	Control en lazo abierto	Sistema simple a bajo costo
	Exploración discontinua	Precisión y repetibilidad de la posición angular deseada
	Condición mutuamente excluyente entre alta velocidad y un tiempo corto de posicionamiento	Estabilidad
Usando transmisión de engranes y motores de pasos con un acoplamiento directo. (PL de TVS prototipo No. 2)	Juego de tornillo «sinfín»	Par de motor alto
	Exploración discontinua	Alta resolución de posicionamiento
	Condición mutuamente excluyente entre alta velocidad y un tiempo corto de posicionamiento	Alta resolución con movimiento suaves usando la técnica «Microstepping»
Usando motores de CC sin par de torsión y bajo par de arranque, acoplamiento directo. (PL de TVS prototipo No. 3)	Sensor de posición digital	Exploración continua
	Sistema complejo	Movimiento suaves
	Sobreoscilaciones en la variable de control	Balance en la condición mutuamente excluyente

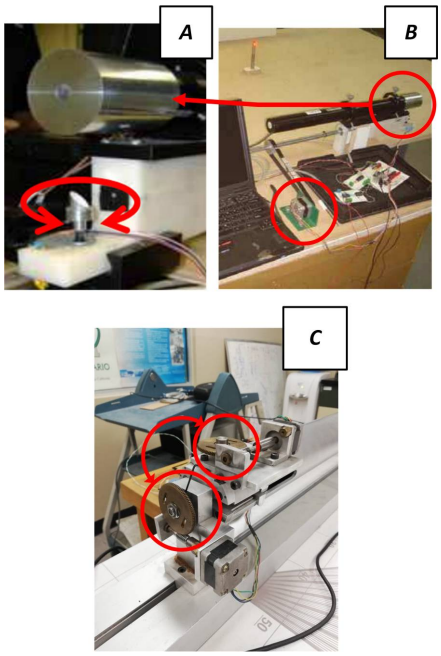

Figura 2.4. A y B pertenecen al PL del TVS No. 1; C es el PL en TVS No. 2 [5] y [35].

2.2 Posicionador Láser del TVS Prototipo 3

La Fig.2.5 presenta las dos partes principales que componen al TVS No. 3. Las partes son: a) Posicionador Láser (PL) y b) Apertura de Escaneo (AE). Esta sección se enfoca en las características del Posicionador Láser. Este es capaz de definir posiciones angulares del eje del motor Maxon DCX22S con un sensor de posición rotativo, el cual es un encoder incremental Maxon ENX 16 con una resolución de 1024 pulsos por revolución (*ppr*), para proporcionar información de la posición angular actual.

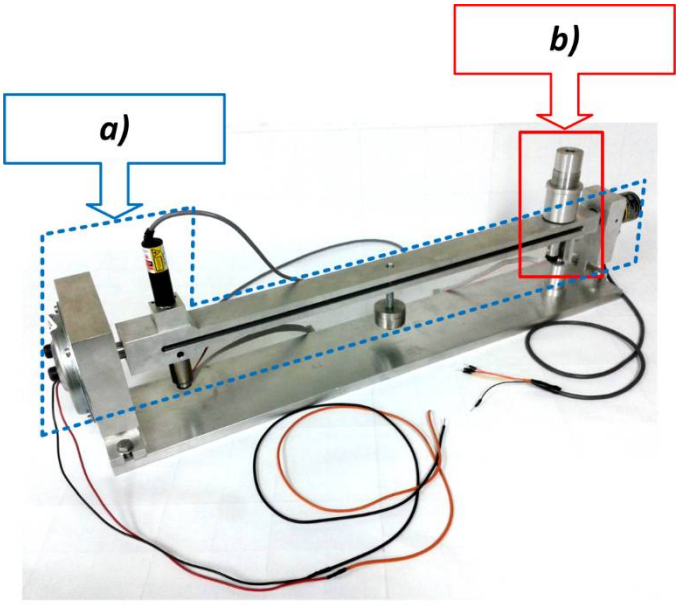

Figura 2.5. TVS prototipo No. 3; a) Posicionador Láser, b) Apertura de Escaneo.

El PL hace un barrido láser hacia su campo de visión (FOV) visto en la Figura 2.3 en la sección anterior 2.1. En trabajos hechos anteriormente del Sistema de Control de Movimiento (SCM) desarrollado para el PL del TVS No.3, se usó una implementación conforme al diagrama de bloques de la Figura 2.6 [11]. Donde se define un SCM en configuración en lazo cerrado. Donde un usuario es capaz de ingresar los valores de posición angular de referencia φ_r, y el valor de la ganancia proporcional para el controlador de posición, usando una GUI (Graphical User Interface) desarrollada, mostrada en la Figura 2.8. El control digital representa una tarjeta Arduino Uno, la cual contiene un microcontrolador Atmega328. En ese microcontrolador se procesa un algoritmo de control proporcional. El algoritmo cuenta con un sumador (el bloque circular) donde se ingresa la señal de la posición de referencia φ_r para ser sumada con la señal negativa de retroalimentación $\varphi_m(k)$ del sensor de posición (Convertidor Analógico/Digital), la cual representa una señal digital en un instante de tiempo

bajo la muestra *k*. La variable $\varphi_m(k)$ representa la información digital de la posición angular actual del eje del motor $\varphi_o(t)$. Del sumador, hay una señal de error $\varphi_e(k)$ de salida, esta alimenta al controlador de posición para obtener la variable de control $y(k)$, el cual representa el porcentaje del ciclo de trabajo del PWM. La $y(k)$ es de baja potencia con niveles de voltaje de 0 a 5 v_{cc} y un máximo de corriente 40 mA. Debido a lo anterior, se usa un amplificador de potencia obteniendo un voltaje nominal de armadura para el motor CC (Convertidor D/A) para finalmente obtener la posición angular deseada $\varphi_o(t)$.

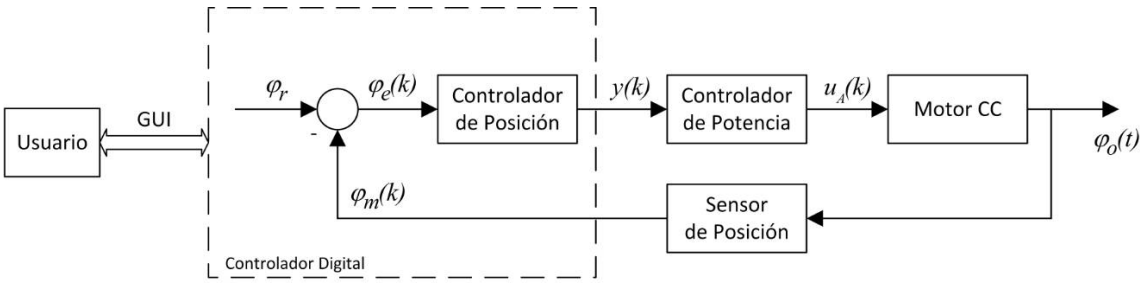

Figura 2.6. Sistema de control digital en lazo cerrado implementado en el PL [11]

La Fig. 2.7 muestra los componentes que componen al PL; A) Módulo láser: Coherent StingRay-514 de 10mW con una longitud de onda de 518 nm, B) Motor plano modelo GPM9 para la rotación síncrona de los sistemas PL y AE, C) Brazo principal orientado en plano horizontal, para el montaje de cada una de las bases que soportan al Posicionador Láser y la Apertura de Escaneo, a la orilla de cada extremo del brazo respectivamente, D) Encoder Incremental Omron E6B2-CWZ6C con 2000 *ppr* para conocer la posición de giro del sistema TVS No.3, E) Espejo con 45 grados de inclinación que esta acoplado concéntricamente al eje del F) motor Maxon DCX22S del PL. El rayo láser saliente del módulo láser es emitido al espejo y proyectado al entorno circundante en sincronía con eje del motor Maxon DCX22S.

En la Figura 2.8 muestra la GUI desarrollada y es usada para comunicarse entre el usuario y el controlador proporcional de posición. El usuario envía variables de referencia y comandos para la tarjeta Arduino Uno y recibe las variables de estado del mismo. La implementación del diagrama presentado en la Figura 2.6 junto con la GUI desarrollada de la Fig. 2.8 fueron parte de la metodología usada para las experimentaciones hechas en los trabajos anteriores concentrados en [11], [18] y los resultados de esas investigaciones reportaron poca precisión con desplazamientos angulares menores a los 5 grados, con un error de posicionamiento menor al 1 %, el sistema de control presentó inestabilidad.

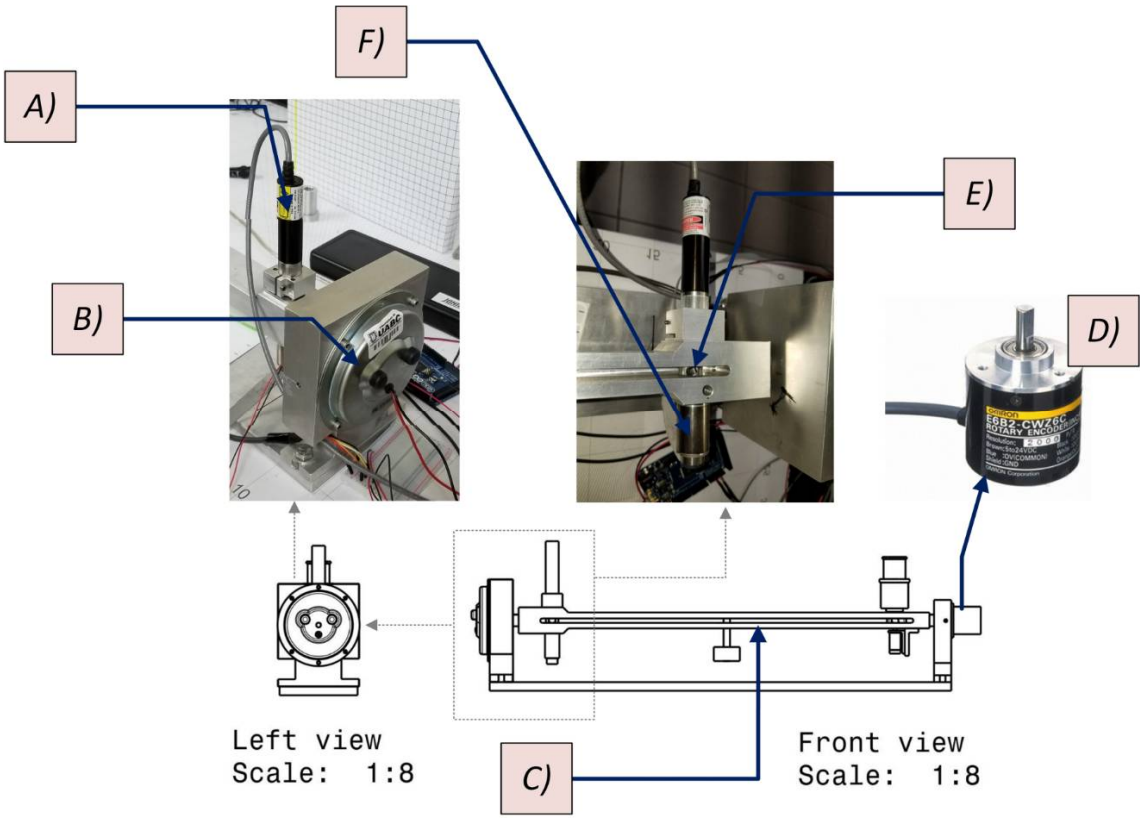

Figura 2.7. Descripción de partes principales del PL del TVS No. 3

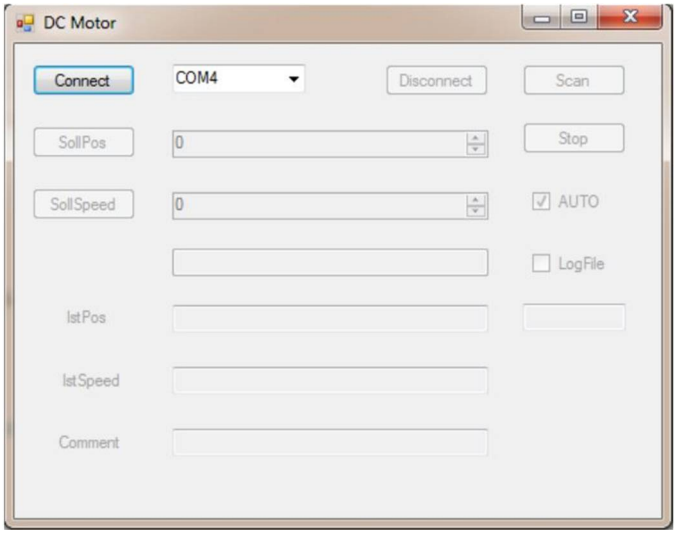

Figura 2.8. GUI desarrollado usando el SCM anterior para el PL del TVS No. 3

3. Marco Teórico

En este capítulo, se presentan los conocimientos teóricos que definen a detalle los conceptos básicos usados para la elaboración de esta tesis. Se estudiará la trayectoria cinemática de la velocidad con forma trapezoidal de un motor con escobillas de corriente continua (CC) para conocer la posibilidad de obtener la posición final del eje del motor CC. Luego, se dan a conocer las ventajas de usar un controlador embebido llamado LM629N-8 para los Sistemas de Control de Movimiento (SCM) debido a que este controlador genera un perfil de velocidad trapezoidal como variable de control usando servomotores. Posteriormente, se analiza un sistema de control para aplicarlo al TVS. Además, se presenta un modelo teórico y simulado en la plataforma Matlab usando su paquetería Simulink para mostrar la ventaja de usar una velocidad con forma trapezoidal para el SCM como parte del propósito de trabajo de esta tesis. Finalmente, se define un diseño de experimentación usando la metodología de Taguchi para conocer los factores significativos que determinan la reducción del error de posicionamiento del SCM para el PL del TVS No.3.

3.1. Trayectoria de Velocidad Trapezoidal

En la ciencia de la física, existe la «Cinemática» que describe los movimientos de los objetos tangibles omitiendo las causas que los originan [21]. La Cinemática analiza la trayectoria generada por el movimiento del objeto de estudio en función del tiempo. Por lo anterior, se analizó una trayectoria generada por el desplazamiento angular del eje de un motor CC, el cual presenta una velocidad con forma trapezoidal que es posible usarse como variable controlada, creando desplazamientos angulares arbitrarios del eje de un motor de CC con escobillas. Esta variable es usada en los SCM de máquinas eléctricas. En estos últimos 50 años, los SCM se requieren para hacer mediciones automáticas [22], [23], [24] se utiliza una variedad de técnicas para controlar el movimiento de los ejes de motores de corriente continua dentro de equipos dinámicos e industriales, como ejemplo en máquinas de control numérico CNC (Computer Numerical Control) y robots autómatas. Usar un perfil de velocidad trapezoidal como variable de control es ventajosa debido a que las rutinas automatizadas de operaciones industriales son estables bajo condiciones de precisión y alta velocidad usando motores eléctricos de corriente continua. Por ejemplo los motores Maxon [16] de imanes permanentes con rotores devanados son de alta calidad debido a que estos motores no contienen hierro en el núcleo del rotor, eliminando el par de torsión (cogging torque) es decir eliminando la interacción entre los imanes del estátor y el núcleo del rotor, y

la reducción de par de arranque del eje del rotor por usar rodamientos de bolas y usando escobillas de material precioso [16].

El perfil idealizado es mostrado en la Figura 3.1 para la velocidad en forma trapezoidal usada en el motor Maxon DCX22S presentándose tres intervalos de tiempo a considerar; el primer intervalo está limitado hasta el tiempo t_a alcanzando la trayectoria de la aceleración inicial usando una razón de cambio constante desde la velocidad cero hasta llegar a la velocidad máxima ω_{max} a partir de ahí hasta el tiempo t_b requerido es para la velocidad máxima constante lograda, finalmente la última tercera parte de la trayectoria representa una pendiente decreciente con la misma razón de cambio usado en la aceleración inicial. Por lo tanto se genera una desaceleración del eje del motor, llegando al tiempo t_φ el cual es el tiempo requerido para obtener la posición deseada y medible al integrar el perfil trapezoidal de la velocidad. La trayectoria puede ser modificada en relación con el tiempo de posicionamiento deseado. Un controlador de movimiento puede alterar la variable de control, en esta sección se estudia el controlador LM629. La variable de control es el perfil de velocidad para el motor aplicado.

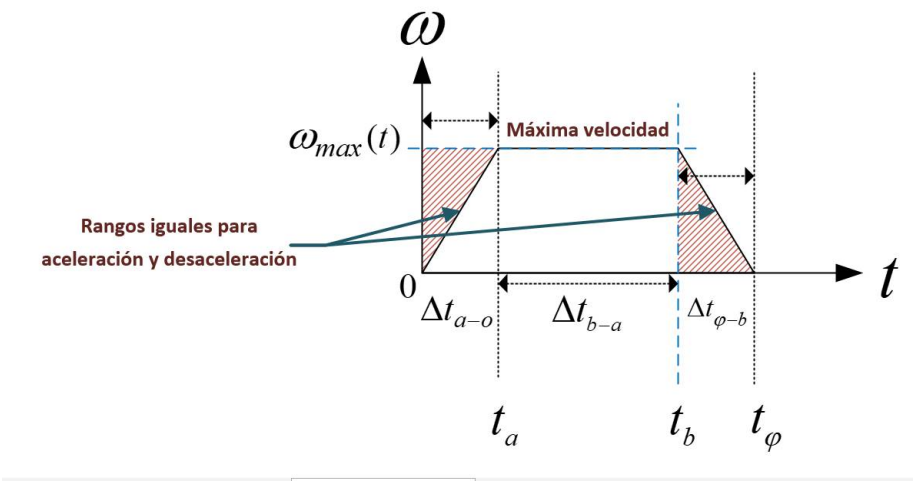

Figura 3.1. Trayectoria de velocidad en forma trapezoidal

Reconociendo que hay diferentes tipos de perfiles de movimiento para los motores eléctricos debido a garantizar que las aceleraciones sometidas por motores con carga requieran un par motor menor que el par eléctrico máximo del motor. Por otra parte, los perfiles de curvas-S son también implementados en los SCM, sin embargo estos tienen desventajas, una de ellas es la difícil implementación [25].

$$\omega_o(t)\begin{cases}\dfrac{\omega_{max}(t)}{t_a} & ,0 \le t \le t_a \\ \omega_{max} & ,t_a \le t \le t_b \\ \omega_{max} - \dfrac{\omega_{max}}{(t_\varphi - t_b)} & ,t_b \le t \le t_\varphi\end{cases} \quad \textbf{(3.1.1)}$$

Donde:

- Velocidad máxima: ω_{max}
- Tiempo de aceleración: t_a
- Tiempo de velocidad máxima: $(t_b - t_a)$
- Tiempo de desaceleración: $(t_\varphi - t_b)$
- Tiempo total:t_φ

El desplazamiento recorrido completo por el eje del motor es la posición final desde la posición cero y representada por:

$$\varphi_o(t_\varphi) = \omega_{max} \cdot t_b = \int_0^{t_b} \omega(t)dt \quad \textbf{(3.1.2)}$$

Donde, la aceleración es la derivada de la velocidad respecto al tiempo

$$a = \frac{d\omega}{dt} \quad \textbf{(3.1.3)}$$

Siendo, la velocidad como la derivada de la posición respecto al tiempo

$$\omega(t) = \frac{d\varphi}{dt} \quad \textbf{(3.1.4)}$$

Por lo tanto, es posible integrar la velocidad angular para obtener la posición angular

$$\omega(t) = \frac{d\varphi}{dt} \rightarrow \int \omega(t)dt = \int d\varphi \quad \textbf{(3.1.5)}$$

La Figura 3.1, se indican dos líneas discontinuas de color azul. Comenzando con la línea azul horizontal, para representar el valor de la velocidad máxima constante usando la variable $\omega_o(t)$, y la línea azul vertical indica el tiempo de inicio a la desaceleración del eje del motor aplicado en t_b. Por lo tanto, se puede observar un rectángulo simétrico definido donde es notable ver un triángulo invertido de color rojo a la izquierda el cual es el área que compensa la tercera parte del trapezoide en su lado derecho el cual es también un triángulo de color rojo. De tal manera, que es posible obtener la integral de la ecuación 3.1.2 para obtener la posición angular final dada por $\varphi_o(t_\varphi)$. En la Figura 3.2, es notable como la trayectoria trapezoidal se deforma en forma de triángulo, debido a adquirir tiempos cortos de posición

usando un alto grado de velocidad angular limitada por la velocidad nominal del motor aplicado evitando comportamientos no óptimos.

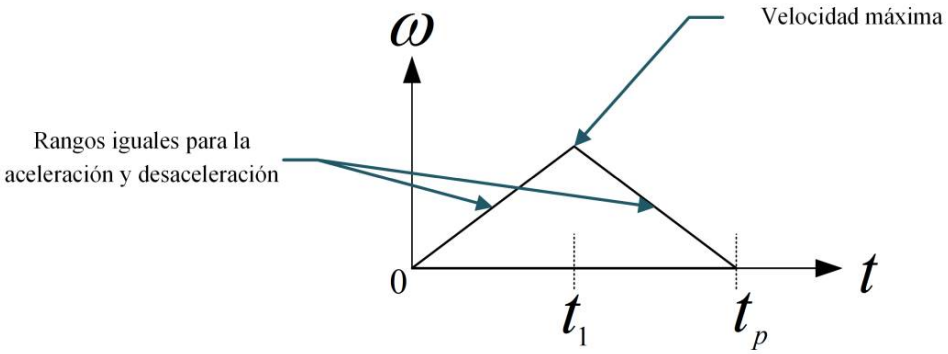

Figura 3.2. Trayectoria de velocidad con perfil triangular

3.2. Sistema de Control para el Posicionador Láser

En la sección 3.1 anterior se validó la importancia de usar un perfil de velocidad trapezoidal como una variable de control para los SCM. Los controladores embebidos han sido parte de un gran número de sistemas de control para el mundo [26]. Estos siguen teniendo importancia en la tecnología de ahora debido a su facilidad de programarlos para las rutinas requeridas, además estos son tan potentes como para implementar SCM complejos ya que son pequeños a escalas micrométricas y con grandes capacidades de memoria, procesamiento y con un gran número de puertos de entradas y salidas. Para la propuesta de reducir el error de posición del sistema de barrido láser (TVS No. 3) se decide usar el controlador embebido LM629 por sus capacidades de operación requeridas en máquinas de control numérico (CNC). Particularmente, el LM629N-8 es el controlador para realizar la tarea principal, la cual es posicionar el rayo láser del PL dentro del FOV del TVS. A continuación, en la Tabla 3.1 se presentan las características principales de este controlador.

Tabla 3.1. Datos del controlador embebido LM629N-8

Características principales	Ventajas
Sistema Embebido	Maximizar espacio, usando sus módulos internos: Interfaz de comunicación manipulada por pines de control, Generador de trayectoria, Decodificador de sensor de posición, Controlador PID y una señal PWM de salida de 8 bits
Señal PWM (8-bit) de salida	Señal de voltaje promediado 0 a 5 V_{CC}(Ciclo de trabajo programable) con signo ($\mp$) conforme a la información procesada por configuración en lazo cerrado

Generador de Perfil de velocidad trapezoidal	Secuenciador de posición comandado por un procesador de 32 bits usando un tiempo mínimo a 256 μs de muestreo
Controlador PID	Controlador digital programable a 16 bits con la posibilidad de elegir el tipo de controlador (P, PI, PD o PID)
Decodificador	Procesador de retroalimentación de posición de 32 bits para decodificar las señales de los canales A, B, Índex con una máxima frecuencia de 1 MHz, usando un reloj de sistema a 8 MHz
Interfaz anfitrión	Puerto de comunicación paralela de 8 bits para operar el controlador embebido, bajo a los estados lógicos de los pines físicos de control para que habilitar la comunicación con un procesador externo
Modos de operación	Programable para operar el controlador en modo velocidad o posición

Figura 3.3, muestra un diagrama de bloques; en este diagrama se presentan los componentes principales tanto internos como externos necesarios para el funcionamiento correcto del controlador LM629N-8. Comenzando con los cuatro bloques con líneas discontinuas que representan los componentes externos, que a continuación se presentan:

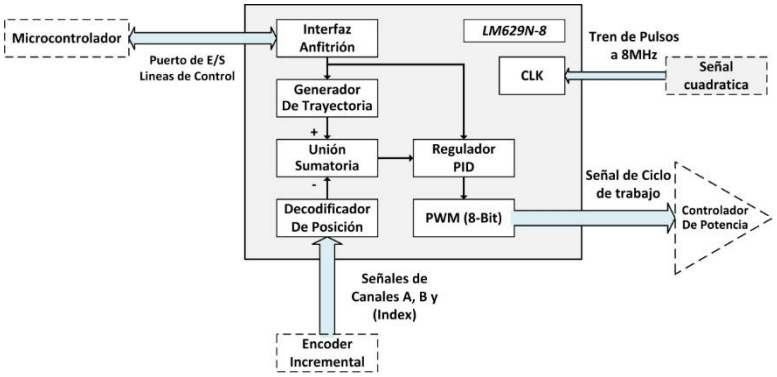

Figura 3.3. Esquema del Controlador Embebido LM629N-8

Microcontrolador

A principios de la década de los setentas sale al mercado el primer microcontrolador, llamado TMS 1000, inventado por Texas Instruments [27], actualmente se siguen comercializando diversos microcontroladores, debido al potencial que presentan en las aplicaciones tecnológicas demandantes por la sociedad en la que vivimos actualmente. Los microcontroladores son circuitos eléctricos embebidos programables, estos generalmente están encapsulados en físico. Los hay en las escalas; nano y micro, estos poseen en general tres unidades: Unidad Central de Procesamiento «Central Processing Unit (CPU)» Memorias: «Read-Only Memory (ROM), Static Reading Access Memory (SRAM), Electrically Erasable Programmable Read-Only Memory (EEPROM) y FLASH» y puertos de entradas y salidas «Entradas/Salidas (E/S)» para la comunicación entre los dispositivos

que interactúan con el microcontrolador. La Figura 3.4, muestra la tarjeta Arduino Mega que usa un microcontrolador Atmega 2560 de la compañía Atmel adquirida por Microchip Technology Inc actualmente. Esta tarjeta es la encargada de mantener una comunicación paralela con el Host Interface del controlador LM629N-8, el cual usa su bus de datos, instrucción y de control internos usando 8-bits de resolución, manipulando la lógica de los pines de control del Host Interface para validarlo como puerto de escritura o lectura [27].

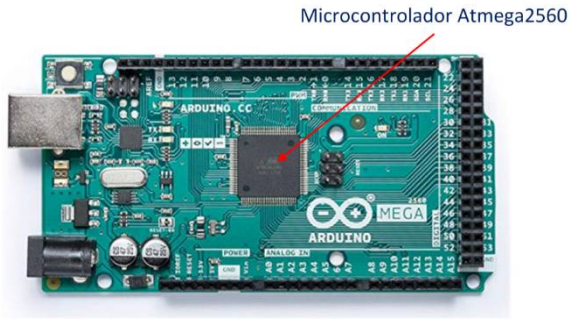

Figura 3.4. Tarjeta Arduino Mega 2560 usada como puerta de enlace

Encoder Incremental

Los sensores de posición rotativos, también se les conocen como codificadores rotatorios o encoders incrementales, usados para convertir la información de la posición angular actual del eje de un actuador, mediante un sistema de medición sin contacto y un disco dentado de alta resolución como un ejemplo visto en la Figura 3.5. El Encoder Incremental (EI) provee una salida incremental en información digital, mediante señales de tren de pulsos usando 3 canales de salida (Canal A y B, cada uno envía un tren de pulsos desfasado a 90 grados entre ellos mostrado en la Figura 3.5) y un indexador para validar un giro de 360 grados mediante un pulso eléctrico. El *EI* se puede presentar en diferentes formatos, los hay magnéticos, ópticos y electromecánicos.

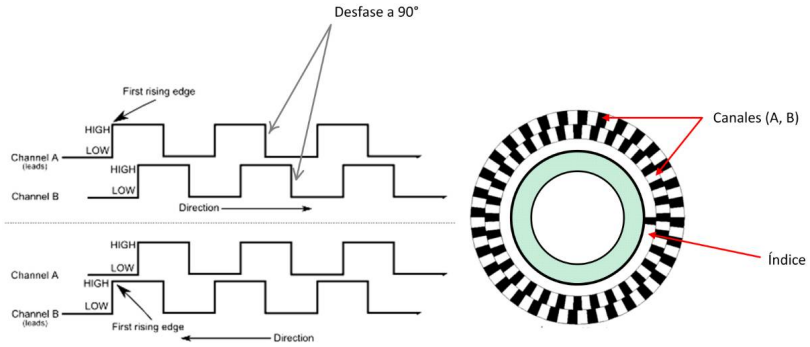

Figura 3.5. Señales y configuración de un Encoder Incremental

Señal Cuadrática

La electrónica digital trata señales discretas y las señales cuadráticas son imprescindibles. En relación con los microcontroladores existen los generadores de pulsos como parte de sus módulos embebidos, estos son los responsables de proveer el reloj a los sistemas electrónicos por ejemplo al controlador LM629N-8, mediante un generador externo que es alimentado por un tren de pulsos con frecuencia fija: f_{CLK} definida como 1 MHz $\leq f_{CLK} \leq$ 8 MHz, dependiendo de la frecuencia del tren de pulsos, será la rapidez para procesar las instrucciones del controlador en cada iteración de sus rutinas de su algoritmo de control, en la Figura 3.6 se puede observar una señal cuadrática real a 8 MHz para generar el reloj externo del controlador LM629N-8.

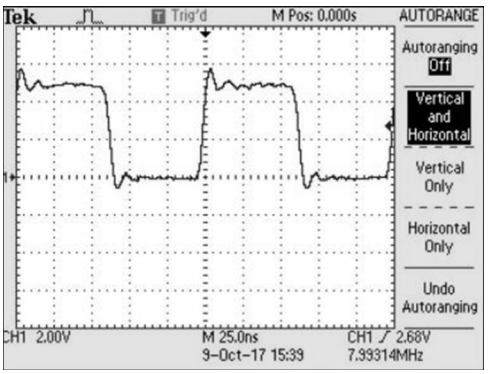

Figura 3.6. Tren de Pulsos a 8 MHz usado como el reloj del LM629N-8

Controlador de Potencia

Este es un circuito electrónico y es un amplificador de potencia de modo conmutado, debido a que tiene un arreglo de transistores MOSFETS. El controlador de potencia es el encargado de amplificar la variable controlable (Voltaje y Corriente eléctrica), proveniente del controlador LM629N-8. Este controlador de potencia proporciona el voltaje de armadura controlable para un motor aplicado al sistema de control. Para la propuesta de esta tesis, el controlador de potencia XY-160D 7A es usado y su aplicación es suministrar el voltaje optimo y la corriente nominal para los motores CC usados para el trabajo de esta tesis. El controlador de potencia es presentado en la Figura 3.7.

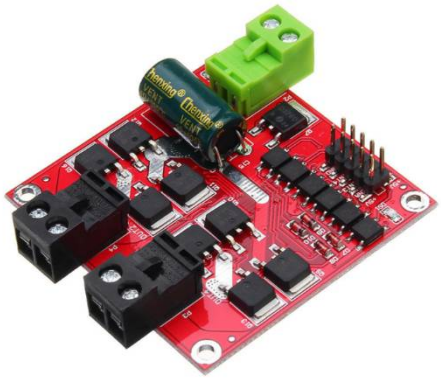

Figura 3.7. Controlador de Potencia XY-160D 7A

Figura 3.8 muestra un diagrama de bloques que representa un Sistema de Control de Movimiento (SCM) para el Posicionador Láser (PL) del TVS No.3, el flujo de información inicia por un «Usuario» que interactúa con una «GUI desarrollada» esta es ejecutada y procesa configuraciones de comandos y datos que solicita el usuario.

Usando un puerto de escritura o lectura a través de una comunicación serial transfiere las configuraciones entre la computadora y el «Microcontrolador». Posteriormente, los datos y comandos procesados en el microcontrolador son escritos o leídos usando una comunicación paralela con un puerto de 8-bits entre el microcontrolador y el controlador LM629N-8 este procesa los datos y comandos con su algoritmo de control soportándose de 6 módulos internos.

Un paquete de datos provenientes de las configuraciones creadas en la GUI, los cuales son:φ_r, ω_{max}, a_r, ppr, las ganancias ajustables del «Controlador PID»:k_p, k_i, k_d y comandos de programación son descargados a la «Interface anfitrión» esta gestiona los datos conforme a los comandos recibidos. Por lo tanto, los datos φ_r, ω_{max}, a_r, ppr son enviados al «Generador de trayectoria» este módulo crea un perfil de velocidad en forma trapezoidal arbitraria conforme a la cantidad ingresada de los pulsos por revolución (*ppr)* convirtiéndolos en cuentas por revolución (*cpr*). Los valores destinados a las ganancias de control son descargados para el Controlador PID interno. La «Unión sumatoria» es la encargada de obtener el error de posicionamiento mediante una resta de los valores homologados, estos son los *cpr* y las cuantas por revolución medidos por el «Decodificador de posición» conforme a las señales cuadráticas (*ppr*) de las salidas de los canales A y B del Encoder Incremental este provee la información de la poción actual del rayo láser emitido por el «Módulo Láser» y orientado al espejo a 45 grados para ser proyectado al entorno circundante

en el campo de visión del TVS No.3. El Controlador PID es alimentado por la señal de error de posición, para obtener la señal de control para ser enviada al bloque del «PWM (8-Bit)» este es el encargado de enviar la señal de control $y(t)$ voltaje promediado por el ciclo de trabajo con signo $\pm$, para dar la dirección de giro del eje del motor. El controlador de potencia es un amplificador de modo conmutado de la señal $y(t)$ (Voltaje y corriente eléctrica) para obtener el voltaje de armadura del «Motor» CC aplicado bajo el parámetro $u_A(t)$.

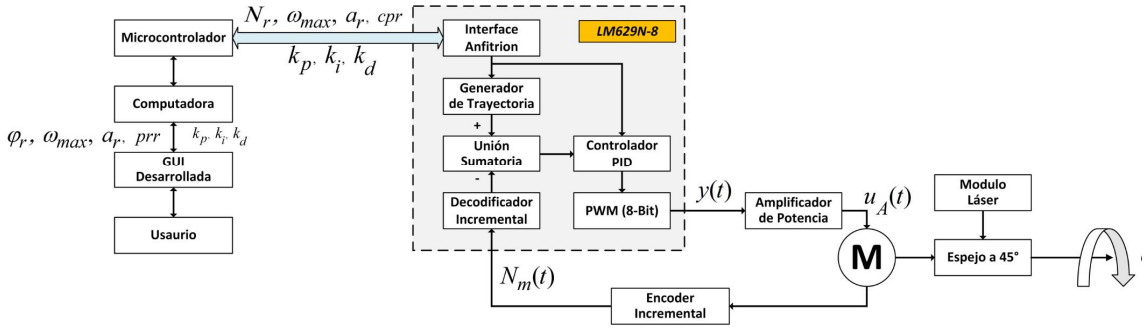

Figura 3.8. Sistema de control de movimiento propuesto para el PL

3.3. Modelo Teórico del Posicionador Láser

EL modelo teórico fue hecho usando un esquema comúnmente conocido que representa la parte eléctrica y parte mecánica de un motor de Corriente Continua CC (M) idealizando su comportamiento [28], este esquema tiene agregado el par motor. Siendo: Voltaje de alimentación para la armadura del motor $u_A(t)$ para la energización del motor CC; hay una Resistencia eléctrica R al paso de la Corriente eléctrica $I(t)$ y una Inductancia eléctrica L generada al cambio de corriente por los devanados del motor. En consecuencia de lo anterior, se presenta una Fuerza Electromotriz (FEM) ε que es el voltaje inducido generado al giro del eje del motor cuando los colectores del rotor están sin contacto con las escobillas activas de corriente eléctrica, al estar girando el eje se producen los siguientes conceptos: J momento de inercia del motor, la cual es la inercia rotacional indicando una resistencia al giro del eje al aplicar una aceleración angular; β como el coeficiente de rozamiento viscoso, $M(t)$ representa el par motor producido por el movimiento angular del eje del motor y este es el momento de fuerza que ejerce el motor sobre el eje; $\omega_o(t)$ es la velocidad angular al producirse un movimiento lineal sobre el eje del motor y finalmente $\varphi_o(t)$ representa la posición angular actual del motor CC. A continuación se presenta un modelo matemático del motor que interactua la parte eléctrica con la parte mecánica.

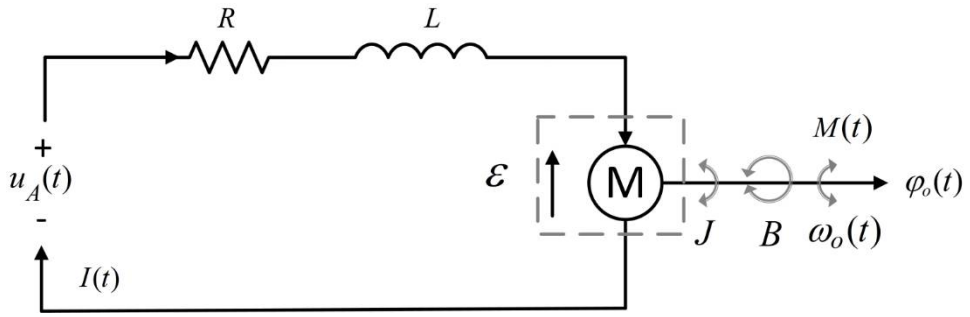

Figura 3.9. Modelado del Motor de CC

La fuerza electromotriz es proporcional a la velocidad angular

$$\varepsilon(t) = k_u \cdot \omega_o(t) \tag{3.3.1}$$

La velocidad es la derivada de la posición angular

$$\omega_o(t) = \frac{d\varphi_o(t)}{dt} \tag{3.3.2}$$

Por la primera ley de Gustav Kirchhoff acerca de la conservación de la energía

$$u_A(t) - \varepsilon(t) = R \cdot I(t) + L\frac{dI(t)}{dt} \tag{3.3.3}$$

La ecuación 3.3.1 se sustituye en la ecuación 3.3.3 para obtener la ecuación 3.3.4

$$u_A(t) - k_u \cdot \frac{d\varphi_o(t)}{dt} = R \cdot I(t) + L\frac{dI(t)}{dt} \tag{3.3.4}$$

Es generado un momento electromagnético al girar el eje del motor por el campo magnético

$$M(t) = k_M \cdot I(t) \tag{3.3.5}$$

Existe un momento de inercia al presentarse al momento electromagnético reducido por el rozamiento dinámico

$$M(t) - \beta\frac{d\varphi_o(t)}{dt} = J\frac{d^2\varphi_o(t)}{dt^2} \tag{3.3.6}$$

Así que el momento electromagnético es la suma de los momentos de inercia y el coeficiente de viscosidad

$$M(t) = J\frac{d^2\varphi_o(t)}{dt^2} + \beta\frac{d\varphi_o(t)}{dt} \tag{3.3.7}$$

De la ecuación 3.3.5 se sustituye la ecuación para obtener la ecuación 3.3.8

$$k_M \cdot I(t) = J\frac{d^2\varphi_o(t)}{dt^2} + \beta\frac{d\varphi_o(t)}{dt} \quad \textbf{(3.3.8)}$$

Se despeja la corriente y se deriva para obtener la ecuación 3.3.9 y 3.3.10 respectivamente

$$I(t) = \frac{J\frac{d^2\varphi_o(t)}{dt^2} + \beta\frac{d\varphi_o(t)}{dt}}{k_M} \quad \textbf{(3.3.9)}$$

$$\frac{dI(t)}{dt} = \frac{J\frac{d^3\varphi_o(t)}{dt^3} + \beta\frac{d^2\varphi_o(t)}{dt^2}}{k_M} \quad \textbf{(3.3.10)}$$

De las ecuaciones 3.3.9 y 3.3.10 se obtiene la ecuación 3.3.11 para definir el modelo de un motor de CC.

$$u_A(t) - k_u\frac{d\varphi_o(t)}{dt} = R \cdot \frac{J\frac{d^2\varphi_o(t)}{dt^2} + \beta\frac{d\varphi_o(t)}{dt}}{k_M} + L \cdot \frac{J\frac{d^3\varphi_o(t)}{dt^3} + \beta\frac{d^2\varphi_o(t)}{dt^2}}{k_M} \quad \textbf{(3.3.11)}$$

3.4. Simulación del Sistema de Control de Movimiento

Para la simulación del modelo de un Sistema de Control de Movimiento (SCM) propuesto para este trabajo de tesis, se desarrollaron diferentes diagramas de bloques usando la paquetería Simulink en la plataforma Matlab, posteriormente se unificaron los diagramas en un modelo completo para representar al SCM. Iniciando con el diagrama de bloques de la Figura 3.10 para representar al motor Maxon DCX22S conforme a las características de la Tabla 3.2. Donde el voltaje de la armadura del motor es $u_A(t)$ que pasa por una unión sumatoria produciendo una señal de error, al sumar $u_A(t)$ y el voltaje inducido con signo negativo para representar a ε (Fuerza electromotriz) multiplicado por una constante *Kd*. La señal que pasa a alimentar al bloque de la parte eléctrica del motor como un sistema lineal; donde *L* es la magnitud de la inductancia que es la oposición a un cambio de corriente de los devanados del rotor que almacenan energía en presencia de un campo magnético y *R* Resistencia eléctrica oponiéndose al flujo de corriente eléctrica *I(t)*, produciendo un momento de fuerza *M(t)* sobre el eje de transmisión de potencia multiplicado por *KM* en seguida este momento de fuerza es afectado por un bloque que representa la parte mecánica del motor presentándose un momento de inercia oponiéndose al giro del eje del motor, la señal que sale del bloque de la parte mecánica del motor es el desplazamiento angular del eje del motor $\varphi_o(t)$ donde *K* es la conversión de radianes a grados, derivando a $\varphi_o(t)$ se obtiene la velocidad angular como variable de control $\omega_o(t)$. La fuerza electromotriz creada por el giro del rotor es retroalimentada con un signo negativo a la unión sumatoria creando una configuración en lazo cerrado [29], [30], [31], [32].

Tabla 3.2. Datos Principales de motor Maxon DCX22S propuestos

Descripción	Parámetro	Unidad	Magnitud
Voltaje de armadura	u_A	*volts*	12
Velocidad nominal	ω_n	*rpm*	6,710
Resistencia eléctrica	R	Ω	2.12
Inductancia eléctrica	L	mH	0.13
Inercia rotacional	J	gcm^2	5.12
Par motor nominal	M_n	mNm	14
Constante de Par motor	k_M	$\frac{mNm}{A}$	13.8
Corriente eléctrica	I	A	1.05

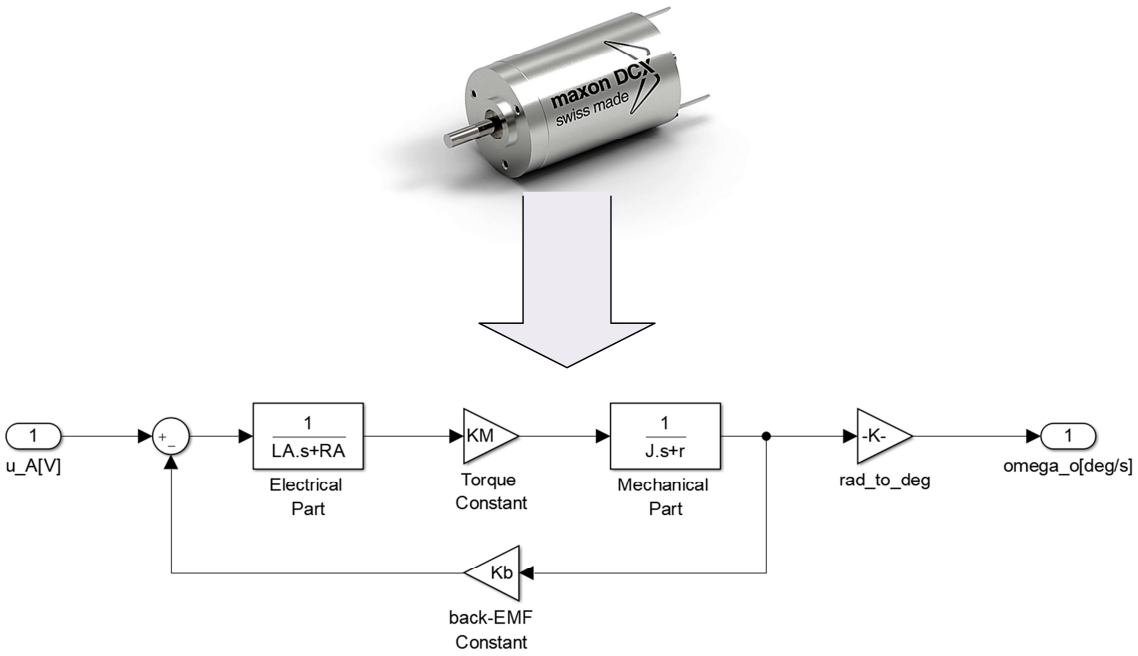

Figura 3.10. Diagrama de bloques para representar al motor Maxon DCX22S

El controlador LM629N-8 tiene entre sus módulos internos un «Generador de trayectoria» este es representado en la Figura 3.11. Para obtener la velocidad trapezoidal, dos bloques de tren de pulsos «Pulso_1» y «Pulse_2» son usados para producir dos señales con la misma amplitud, una es contraria a la otra y el tren de pulsos «Pulse_2» tiene un retardo en el tiempo necesario para obtener la desaceleración. Ambas señales pasan en sincronía por un bloque sumador para formar una señal continua de aceleración y desaceleración en secuencia, observada en la Figura 3.11 en la primer gráfica de color rojo, luego esta señal continua pasa por un bloque «Integrador_1» integrando la aceleración y desaceleración para obtener el

perfil trapezoidal de la velocidad en la segunda grafica de color amarillo, donde es notable pendientes iguales de la aceleración $a(t)$ y desaceleración, la razón de cambio es la misma para ambas, entre estas rampas se muestra una señal con amplitud constante representando la velocidad constante máxima $\omega_{max}(t)$. Esta velocidad pasa por otro bloque «Integrador_2» integrando el área debajo del perfil trapezoidal de la velocidad para obtener la trayectoria de la posición de referencia del eje del actuador aplicado $\varphi_r(t)$ vista en la tercer grafica con la señal de color azul. Confirmando la operación del generador de perfil de velocidad trapezoidal.

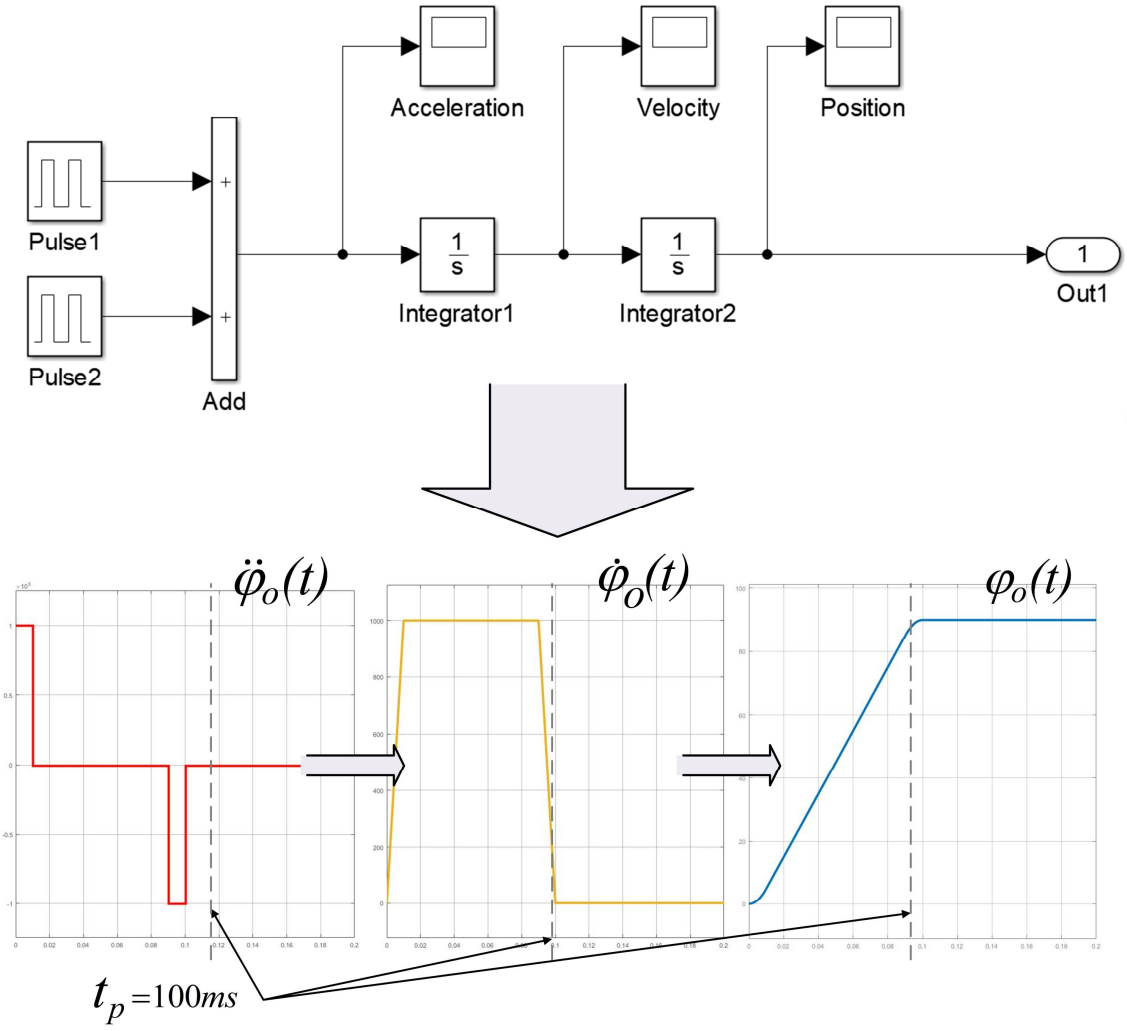

Figura 3.11. Generador del perfil de trayectoria trapezoidal y sus gráficas

El Sistema de Control de Movimiento (SCM) está representado en la Figura 3.12. El bloque inicial es el generador de perfil de la señal de la velocidad trapezoidal $\omega_o(t)$ obteniendo la posición de referencia $\varphi_r(t)$. Observando, la Figura 3.12 la señal $\varphi_r(t)$ proveniente del

generador de trayectoria pasa por un sumador para producir la señal de error de posicionamiento $\varphi_e(t)$ posteriormente este señal pasa por un controlador PID configurable, este controlador produce la señal de control $y(t)$ para accionar al bloque DCX22S-1 representado el sistema controlado, este es un motor Maxon DCX22S analizado en la Figura 3.10. Este motor ejerce una rotación en su eje generando una velocidad angular $\omega_o(t)$ que sale del bloque DCX22S-1 y pasa por un bloque «Encoder1» su función de este es convertir la señal $\omega_o(t)$ en la posición deseada $\varphi_o(t)$ para ser retroalimentada hacia la unión sumatoria, fijando una configuración en lazo cerrado del SCM. La Gráfica de la Figura 3.12 es notable un retraso de tiempo mínimo para la obtención de $\varphi_o(t)$ debido a la inercia rotacional y los efectos no lineales como la fricción que sufre el motor Maxon DCX22S.

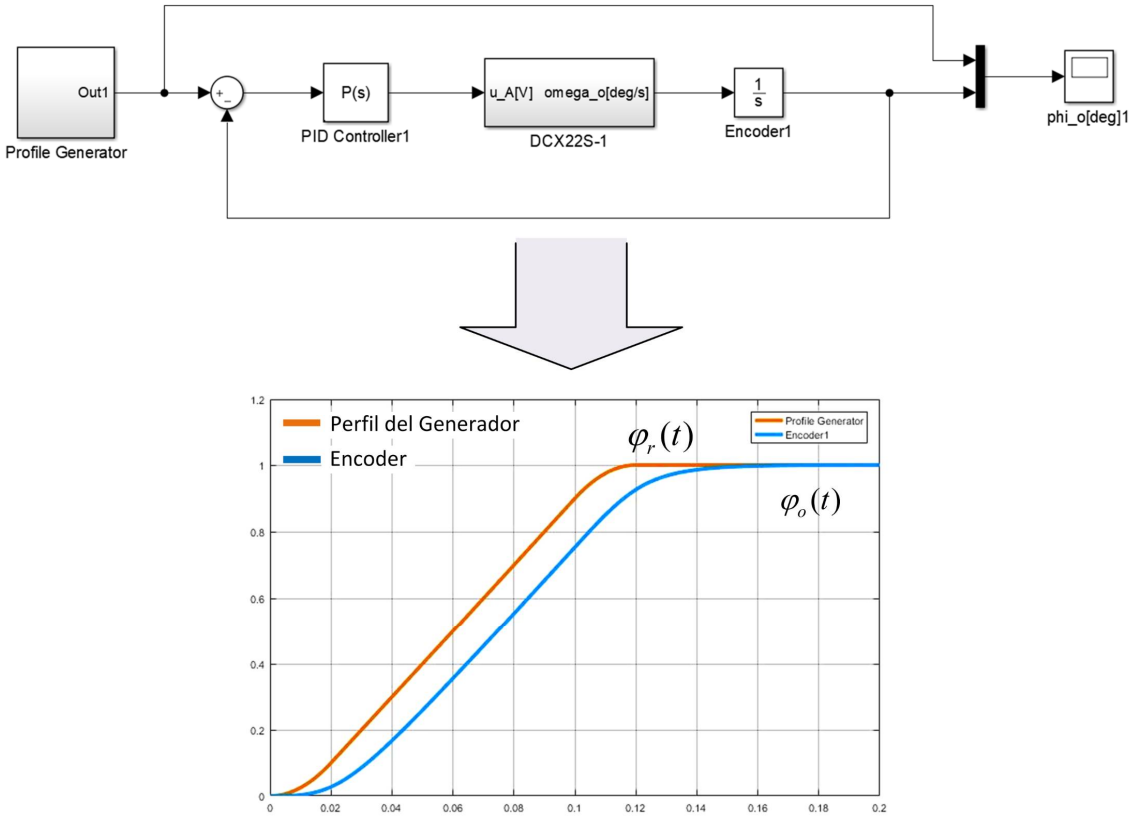

Figura 3.12. Señales de respuesta de entrada y salida del sistema propuesto

Comparación de la señal de salida $\varphi_o(t)$ para diferentes modelos

La Figura 3.13 muestra dos modelos con métodos diferentes para un Sistema de Control de Movimiento (SCM), ambas señales de salida de cada uno de estos modelos pasan por un bloque «Mux» encargado de combinar las dos salidas de posicionamiento de cada modelo para poder visualizar la respuesta de la Figura 3.14, la cual es una comparación entre ambas señales obtenidas por el bloque «phi_0_compare». En el sistema 1, se usa una señal tipo escalón para representar la señal de posición de referencia $\varphi_r(t)$ pasa por el bloque sumador para obtener la señal de error de posicionamiento por el hecho de presentar una configuración en lazo cerrado del SCM al retroalimentar la señal de la posición de salida $\varphi_o(t)$ por el bloque «Encoder-1» este representa el sensor de posición. La señal $\varphi_e(t)$ pasa por el bloque «PID-1» encargado de compensar la señal de error obteniendo la variable de control *y(t)* que pasa a alimentar al bloque DCX22S-S que representa al motor Maxon DCX22S este produce un momento de fuerza sobre el eje de transmisión de potencia para provocar desplazamiento angular y obteniendo la señal de la posición de salida $\varphi_o(t)$. El flujo de señales que pasan por el segundo sistema es implementado usando la Figura 3.12.

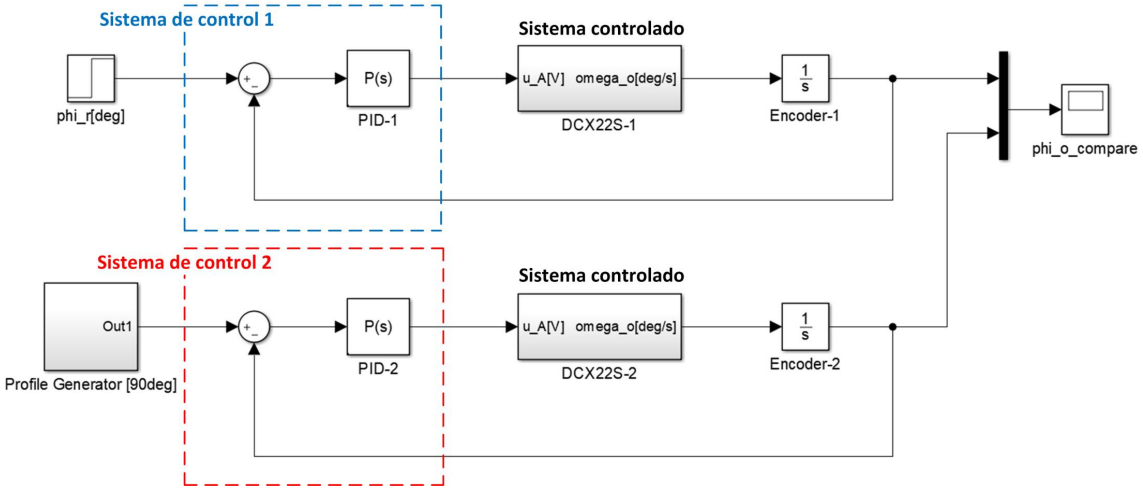

Figura 3.13. Comparación de sistemas: previo y propuesto

La Figura 3.14, muestra el resultado del bloque «phi_o_compare» mostrando dos señales de la posición angular final $\varphi_o(t)$ con amplitud en grados. La señal de color azul corresponde al sistema de control 1 el cual usa como entrada un escalón unitario como la señal de posición angular de referencia $\varphi_r(t) = 90$ grados. Sin embargo, es notorio que $\varphi_o(t)$ presentó una sobreoscilación en el estado transitorio, y rápidamente compenso el error para posicionarse en 93.13 grados. La señal de color naranja representa la señal de salida $\varphi_o(t)$ del segundo sistema de control el cual utiliza un perfil trapezoidal para la velocidad aplicada al motor CC, esta $\varphi_o(t)$ reduce las sobreoscilaciones en estado transitorio y se mantiene estable en la zona

transitoria y estacionaria, permitiendo un movimiento suave sin carga considerable en el eje del motor aplicado obteniendo una respuesta precisa a 89.99 grados. Por lo anterior, hay una evidencia de la ventaja de usar una trayectoria trapezoidal para la de velocidad del motor CC como una variable de control.

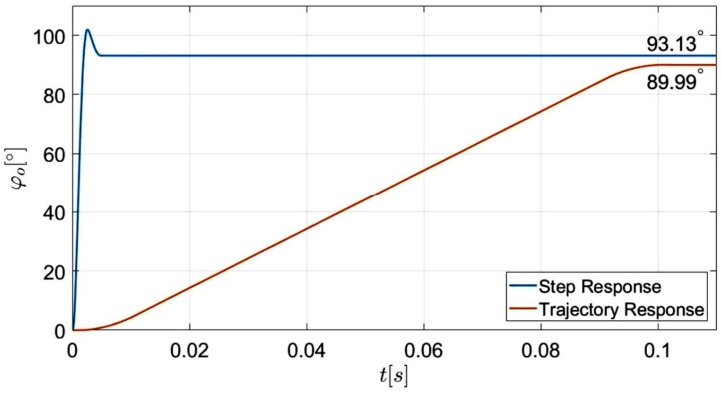

Figura 3.14. Gráfica comparativa entre las posiciones angulares de salida $\boldsymbol{\varphi_o(t)}$

3.5. Metodología de Taguchi en el Diseño de Experimentación

La metodología Taguchi [33] se enfoca en reducir la variación de un proceso usando el concepto «Robustez» en un diseño de experimentación, reduciendo el costo de la producción de productos a fines mostrando una alta calidad en los mismos. Para ello el Ing. Genichi Taguchi [34] propuso los arreglos ortogonales o también llamados «Arreglos Taguchi» involucrando todas las variables que pudieran afectar a un proceso sin probar todas las combinaciones posibles de las variables, en diferentes niveles de prueba.

La metodología Taguchi presenta tres etapas:

- Diseño del sistema
- Diseño de parámetros
- Identificar qué factores
- Diseño de tolerancias

La hipótesis establecida por Taguchi se conforma de la siguiente manera:

1. La calidad de un producto debe ser diseñada y no inspeccionada
2. Obtener el diseño de los parámetros que afectan el proceso
3. Seleccionar el arreglo ortogonal más apropiado a partir del diseño de parámetros donde se indiquen el número y condiciones de cada experimento
4. Realizar las pruebas indicadas por el arreglo
5. Realizar el análisis de datos

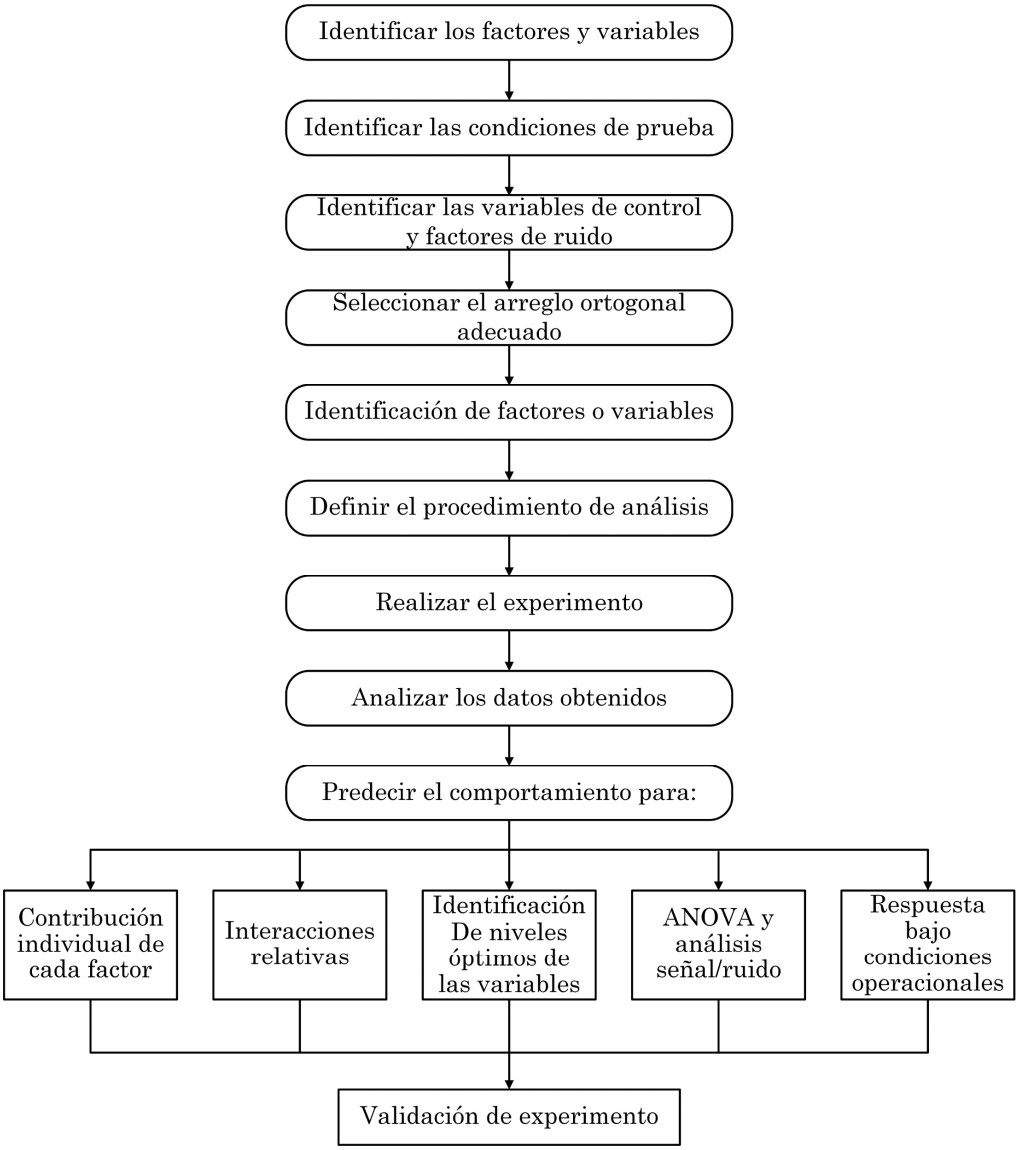

Figura 3.15. Metodología de Taguchi de Diseño de experimentos [33]

4. Metodología de Funcionamiento del Sistema de Control para el PL

Este capítulo, cinco secciones se presentan para definir la metodología de funcionamiento del sistema de control en configuración de lazo cerrado para el Posicionador Láser (PL) del TVS No.3. La sección 4.1 contiene la implementación del sistema de control propuesto dentro del trabajo de esta tesis. La sección 4.2 presenta el programa informático desarrollado para la tarjeta Arduino Mega 2560 encargada de hacer una intercomunicación entre la GUI y el controlador LM629N-8. El desarrollo de la Interfaz Gráfica de Usuario (GUI) para el PL se presenta en la sección 4.3 finalmente, la sección 4.4 contiene el diagrama de flujo para posicionar un rayo láser hacia un entorno circundante limitado del TVS No.3.

4.1. Implementación de Sistema de Control de PL

El sistema de control para el Posicionador Láser (PL) es presentado en el diagrama de bloques de la Figura 4.1. El sistema de control está integrado por los siguientes componentes: Usuario para monitorear el estado del sistema y descargar una configuración de datos para dar de alta una posición angular de referencia (arbitraria) del rayo láser que será proyectado hacia un entorno circundante; el usuario interactúa con una GUI (Graphic User Interface) esta envía la configuración de datos a una tarjeta Arduino Mega 2560, la cual usa un microcontrolador Atmega2560 (CMOS 8-bit) basado en la arquitectura (AVR enhanced RISC- Reduced Instruction Set Computer), esta tarjeta ejecuta un programa máquina (Código Máquina-CM) con las instrucciones asignadas para procesar la configuración de datos y de señales físicas que interactúan con el sistema de control para el PL. El CM es creado al compilar un programa informático (Código Fuente-CF) desarrollado y es descrito a detalle en la sección 4.2, el CF es desarrollado en las plataformas Arduino IDE y Eclipse IDE. El controlador LM629N-8 de Texas Instruments (TI) diseñado para usarse en sistemas de control de movimiento preciso, este procesara datos y comandos salientes de la tarjeta Arduino Mega para obtener una señal PWM de control de salida ; un controlador de potencia modelo XY-160D, se encarga de amplificar la potencia eléctrica (voltaje y corriente) a la señal PWM. Por lo tanto, la señal de salida del XY-160D provee la corriente y voltaje nominales a un motor Maxon DCX22S con un Incremental Encoder Maxon ENX 16 para proporcionar información de la posición angular actual de un rayo láser color verde proyectado al entorno circúndate con la ayuda de un espejo con corte diagonal a 45 grados acoplado al eje del motor Maxon

DCX22S. El rayo láser es proveniente de un Módulo láser: Coherent StingRay-514 de 10mW con una longitud de onda de 518 nm

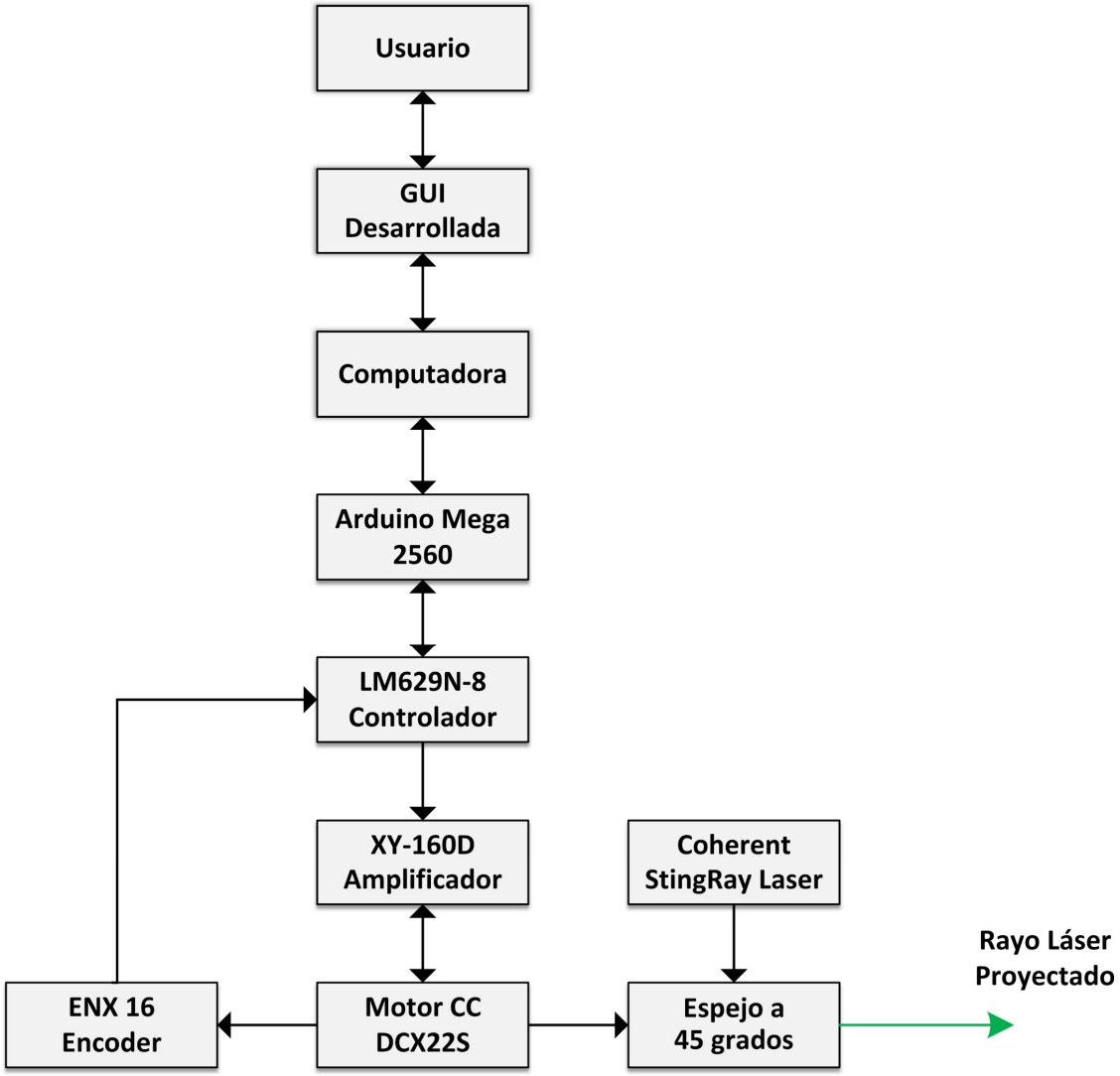

Figura 4.1. Diagrama de bloques para el Posicionador Láser

A continuación se explicara con carácter técnico el flujo de datos del diagrama de bloques presentado en la Figura 4.1.

Usuario y GUI desarrollada

La Figura 4.2 se presentan los bloques de inicio para el flujo de datos del sistema de control para el Posicionador Láser. El usuario es el operador de la GUI ejecutable y desarrollada usando instrumentos virtuales en la plataforma LabVIEW 2015, la GUI es compatible con el sistema operativo Windows. El usuario es retroalimentado del estado del sistema visualmente

por la GUI usando un monitor de datos y comandos. Al ingresar datos a la GUI, se crean paquetes de datos y comandos de tipo cadena, estos llevan la configuración requerida para la tarjeta Arduino Mega. La configuración de datos contiene comandos de funciones específicas del controlador LM629N-8 estas funciones están explicadas a detalle en la sección 4.2, la configuración de datos contiene números hexadecimales para definir las ganancias del controlador PID e información de una perfil de velocidad trapezoidal; tales como la aceleración de referencia, velocidad máxima, posición angular de referencia, pulsos por revolución del Enconder Incremental, si se desea obtener variables relativas o absolutas.

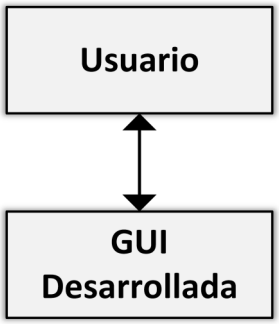

Figura 4.2. Usuario y GUI desarrollada

Computadora y Microcontrolador

La Figura 4.3, presenta el bloque de la computadora debido a que la GUI es ejecutada. La tarjeta Arduino Mega tiene un conector USB (Universal Serial Bus) 2.0 tipo «B hembra». Por lo tanto, la comunicación serial es usada entre la computadora y la tarjeta Arduino Mega. Al tener un cable USB de tipo «A macho» a tipo «B macho», conectado entre los puertos USB entre la Tarjeta Arduino Mega y la Computadora para escribir o leer datos entre ambos componentes.

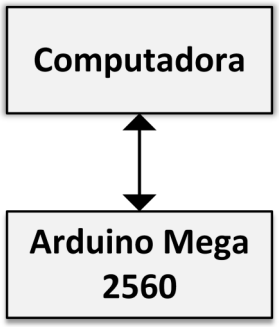

Figura 4.3. Computadora y microcontrolador

LM629N-8, Amplificador XY-160D, ENX 16 Encoder y motor Maxon DCX22S

La Figura 4.4 presenta el bloque del controlador LM629N-8, el cual usa una comunicación paralela para la escritura y lectura de datos y comandos hacia o desde la tarjeta Arduino Mega. El LM629N-8 usa una señal de control PWM (Pulse Width Modulation) de salida, a dos pines en configuración; el primer pin es la magnitud del PWM (Ciclo de trabajo) 0 a 5 v_{cc} usando lógica transistor a transistor-TTL, el segundo pin es la dirección del PWM. La señal del PWM es amplificada en potencia por el controlador XY-160D, obteniendo un voltaje y una corriente nominales para el motor Maxon DCX22S con un sensor de posición acoplado al eje del motor, el sensor de posición es un Encoder Incremental (EI) para retroalimentar al LM629N-8 la información de la posición actual del eje del motor. El eje del motor tiene acoplado un espejo a 45 grados para proyectar un rayo láser proveniente del módulo «Coherent StingRay Laser» hacia el campo de visión del TVS No.3.

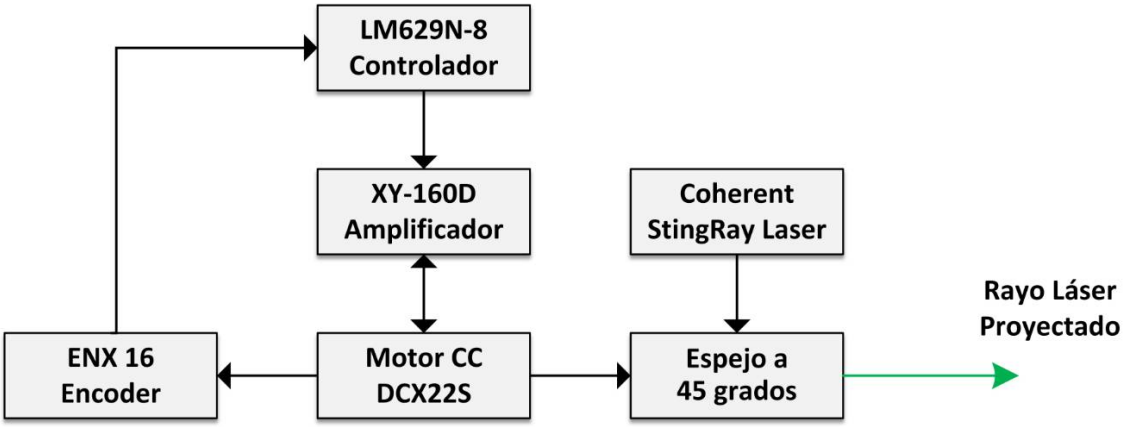

Figura 4.4. Controlador LM629N-8, Amplificador XY-160D, Sensor de Posición y el motor Maxon DCX22S

4.2. Implementación de Software para el Hardware

El Código Fuente (CF) es desarrollado en las plataformas Arduino (Integrated Development Environment-IDE) y Eclipse IDE C++, el CF contiene las instrucciones para procesar los datos, comandos y señales físicas que interactúan con el microcontrolador Atmega2560 montado en la Tarjeta Arduino Mega. La Figura 4.5 muestra la herencia de clases implementadas, habilitando todas las funciones que ofrece el controlador de movimiento, cada una de las clases tiene sus atributos y operaciones específicas para procesar un conjunto de variables, constantes y funciones para interactuar con los datos y comandos descritos en cada una de ellas.

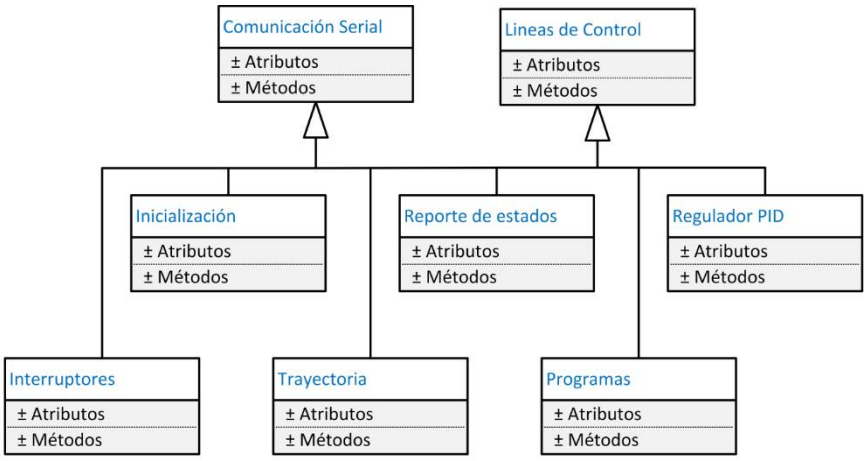

Figura 4.5. Esquema de herencia de las clases desarrolladas

Como un ejemplo de lo anterior se define la clase «Interrupt»

La Figura 4.6, muestra la estructura de la clase «Interrupt» que fue desarrollada y es parte de la implementación del software para la tarjeta Arduino Mega, para habilitar las interrupciones del controlador LM629N-8, los atributos con la bandera – corresponden a las constantes y variables privadas, los métodos con el signo + indican que son públicos para ser llamadas entre otras clases propuestas en la Figura 4.5. Las 8 clases desarrolladas para la implementación dan difusión a un Programa Orientado a Objetos (POO) claro, ordenado y flexible con la posibilidad de realizar modificaciones de manera simple y rápida. Las clases en grupo forman una biblioteca para ser llamada en el algoritmo principal.

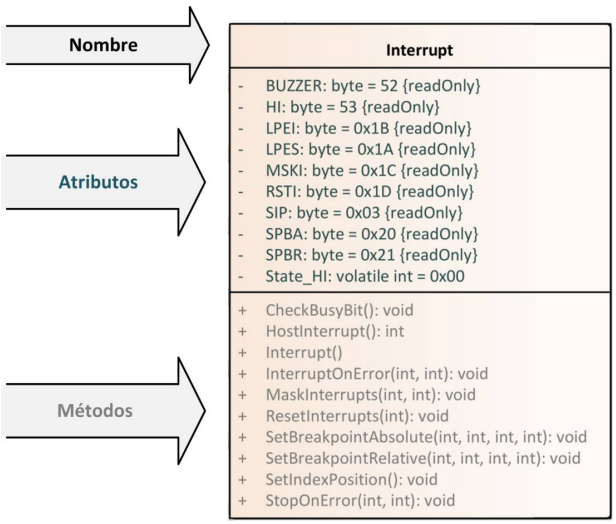

Figura 4.6. Bloque que representa la Clase «Interrupt»

4.3. Implementación de Software para Interfaz Gráfica de Usuario

La planificación del software para la interface gráfica de usuario fue hecha usando un esquema específico para el lenguaje visual de la plataforma LabVIEW, este esquema es notable en la Figura 4.7; tres estructuras cíclicas (Ciclos While) corren en paralelo, donde cada uno de ellos tiene sus procesos de datos específicos ejecutando ciertas acciones tales como: reporte de datos, cálculos de aceleración, velocidad y posición. El flujo de datos será de izquierda a derecha, comenzando con el bloque «Puerto Serial COM #» este es el encargado de abrir sesión con el puerto serial a la comunicación serial entre la computadora y la tarjeta Arduino Mega 2560, este tiene dos caminos de comunicación uno es para la escritura de datos o comandos (TX) de la GUI hacia la tarjeta Arduino Mega 2560 y el segundo camino es para la lectura de datos y comandos (RX) desde la tarjeta Arduno Mega 2560.

De arriba hacia abajo, el primer ciclo tiene una estructura de eventos por interrupción esperando a que una interrupción se active para procesar las diferentes acciones programadas. Las interrupciones se activan al presionar algún botón del panel de control de la GUI. Algunas acciones internas en la estructura de eventos contienen el bloque de escritura de datos y comandos para que sean transferidos al puerto serial del microcontrolador Atmega2560 y posteriormente al controlador LM629N-8, el segundo ciclo monitorea si hay caracteres en el búfer de lectura correspondiente el puerto serial de la computada cada 500 milisegundos (Timeout) de esta forma existe una retroalimentación de la correcta funcionalidad del controlador LM629N-8, algunos de los caracteres serán presentados en la GUI y otros activan interrupciones por datos o comandos, de esta manera el último ciclo procesa los eventos por interrupción por datos y presenta las gráficas de la trayectoria de la posición y velocidad vistos en el capítulo 5.

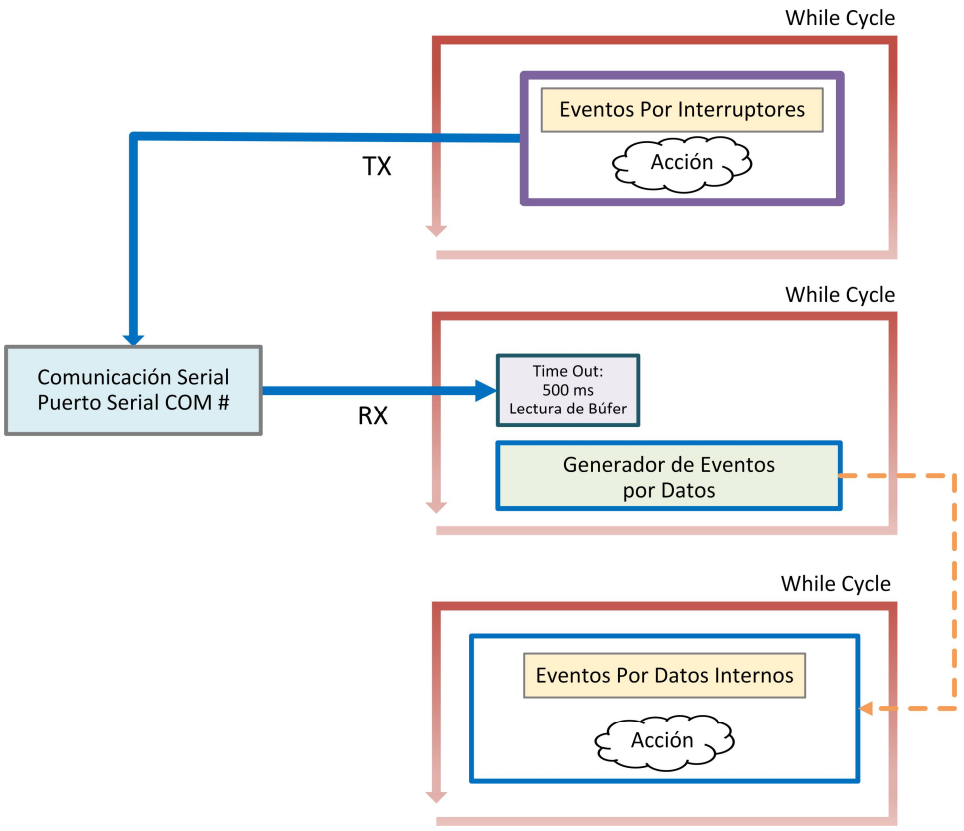

Figura 4.7. Esquema de la lógica del software para la GUI mediante Virtual Instruments usando LabVIEW

4.4. Diagrama de Flujo para el Proceso de Posicionar el Rayo Láser

El proceso de posicionar el rayo láser saliente del módulo «Coherent StingRay Laser» es hecho por el usuario a través de una seria de pasos, logrando posicionar al eje del motor Maxon DCX22S. La Figura 4.8, muestra el flujo acciones a ejecutar para hacer funcional el sistema de control de movimiento del PL usando el controlador LM629N-8 para el control de un simple movimiento.

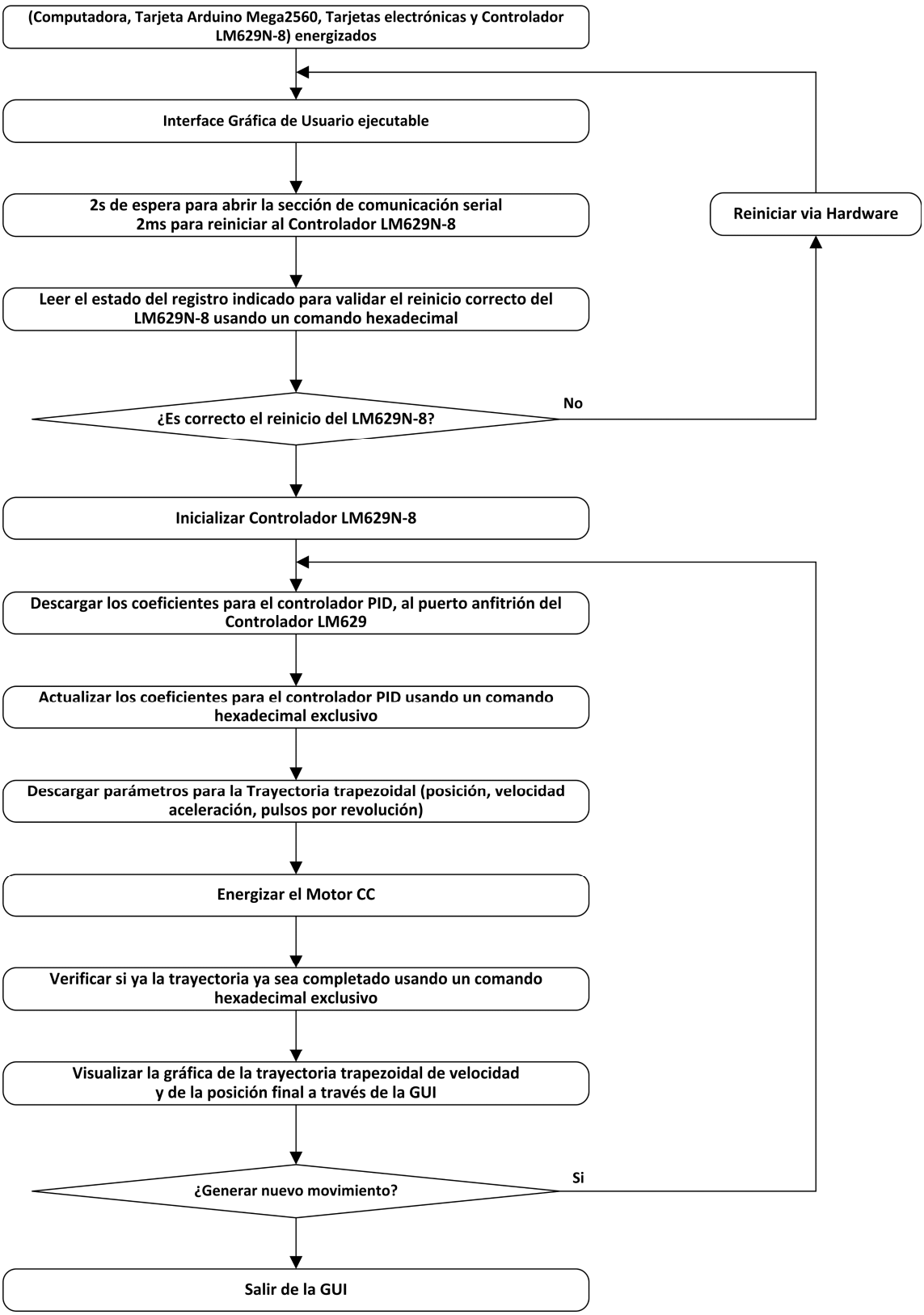

Figura 4.8. Diagrama de flujo de proceso del PL en el TVS No. 3

5. Experimentación de la Reducción del Error de la Posición Angular

Este capítulo presenta los trabajos hechos mediante la metodología del capítulo 4. Las evidencias de los trabajos están distribuidas en seis secciones; sección 1: Integración de tarjetas electrónicas este presenta los circuitos eléctricos involucrados para el control eléctrico requerido en la metodología; la sección 2 contribuye a la integración de software, presenta un ejemplo de cómo fue hecha la programación orientada a objetos (POO), usando las estructura de clases y el desarrollo de una librería. Después, la sección 3: Ejecución del Posicionador Láser (PL) desde una GUI; demuestra cómo se llevó a cabo la experimentación para mover el rayo láser del sistema TVS prototipo 3 desde una Interface de Usuario Grafica desarrollada, posteriormente se presentan las condiciones experimentales, buscando alta precisión y tiempos de posicionamientos cortos. Finalmente, se demuestra cómo se llevó a cabo el diseño de experimentación determinado por las características que lo definen.

5.1. Integración de Tarjetas Electrónicas

La integración de las tarjetas electrónicas fue planificada usando la metodología presentada en esta tesis. Los primeros trabajos de la integración se desarrollaron usando una tarjeta de pruebas electrónicas (protoboard). Con el objetivo de hacer modificaciones de manera sencilla y reduciendo el presupuesto económico de materiales. Las modificaciones fueron hechas durante la experimentación, detectado errores de programación para minimizarlos.

La integración fue hecha al comprender cuales eran las conexiones eléctricas básicas para poner en operación el controlador LM629N-8, la Figura 5.1 muestra la imagen A, mostrando las primeras conexiones de prueba para el controlador LM629N-8, los cables azules habilitan la comunicación en paralelo del módulo « Interface Host » a 8 bits. La Imagen B, muestra la primer integración de la experimentación mediante la metodología del capítulo 4, en ella se presenta la tarjeta Arduino Mega 2560, el motor aplicado y el modulo que funciona como controlador de potencia L298.

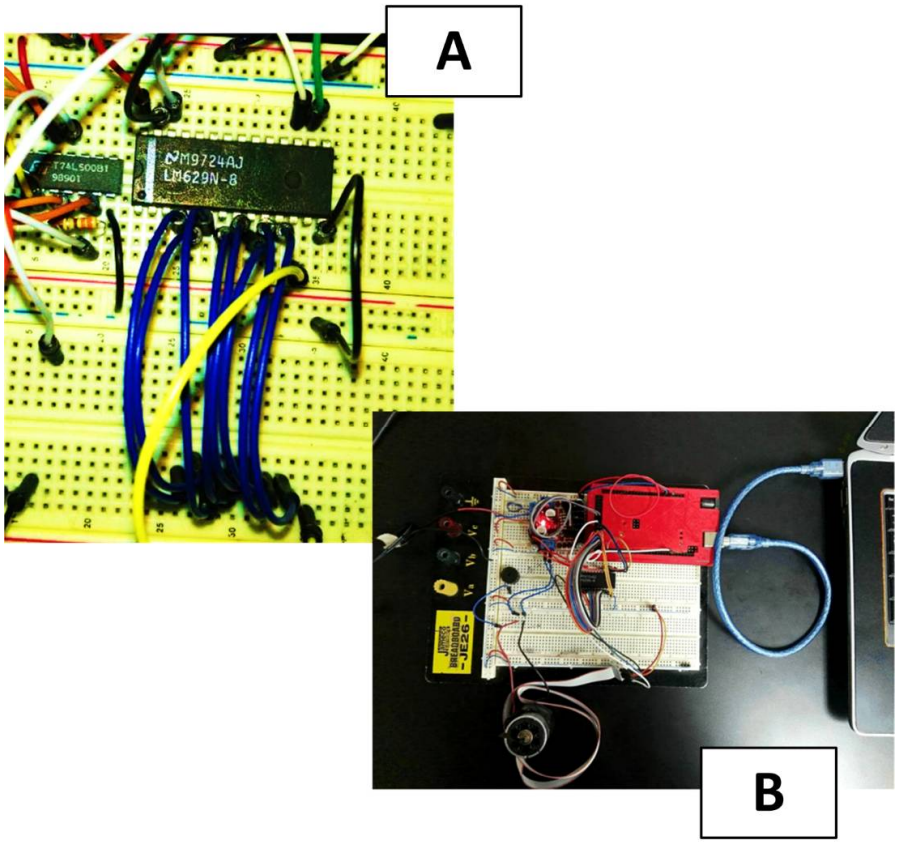

Figura 5.1. A) LM629N-8 habilitando su puerto E/S; B) Primera experimentación funcional

La primera experimentación de la integración del sistema de control mostrada en la imagen B de la Figura 5.1, contribuyo a realizar los primeros movimientos del eje del motor Maxon DCX22S. Los resultados son mostrados en el capítulo 6. Sin embargo, eventualmente se presentaban falsos contactos de los cables utilizados en la placa de prueba debido a lo anterior, surge el diseño de una tarjeta electrónica para fijarla sobre la tarjeta Mega 2560, sin perder la accesibilidad de las entradas y salidas del microcontrolador LM629N-8 para futuros trabajos. El diseño de primera tarjeta electrónica es mostrado en la Figura 5.2, en ella la imagen de arriba representa el esquema electrónico requerido para la tarjeta y abajo se muestra las locaciones de los componentes a insertar, el diseño fue desarrollado en la plataforma «Proteus Design Suite» encargado de crear diseños de tarjetas electrónicas simples hasta complejas y simularlos inclusive.

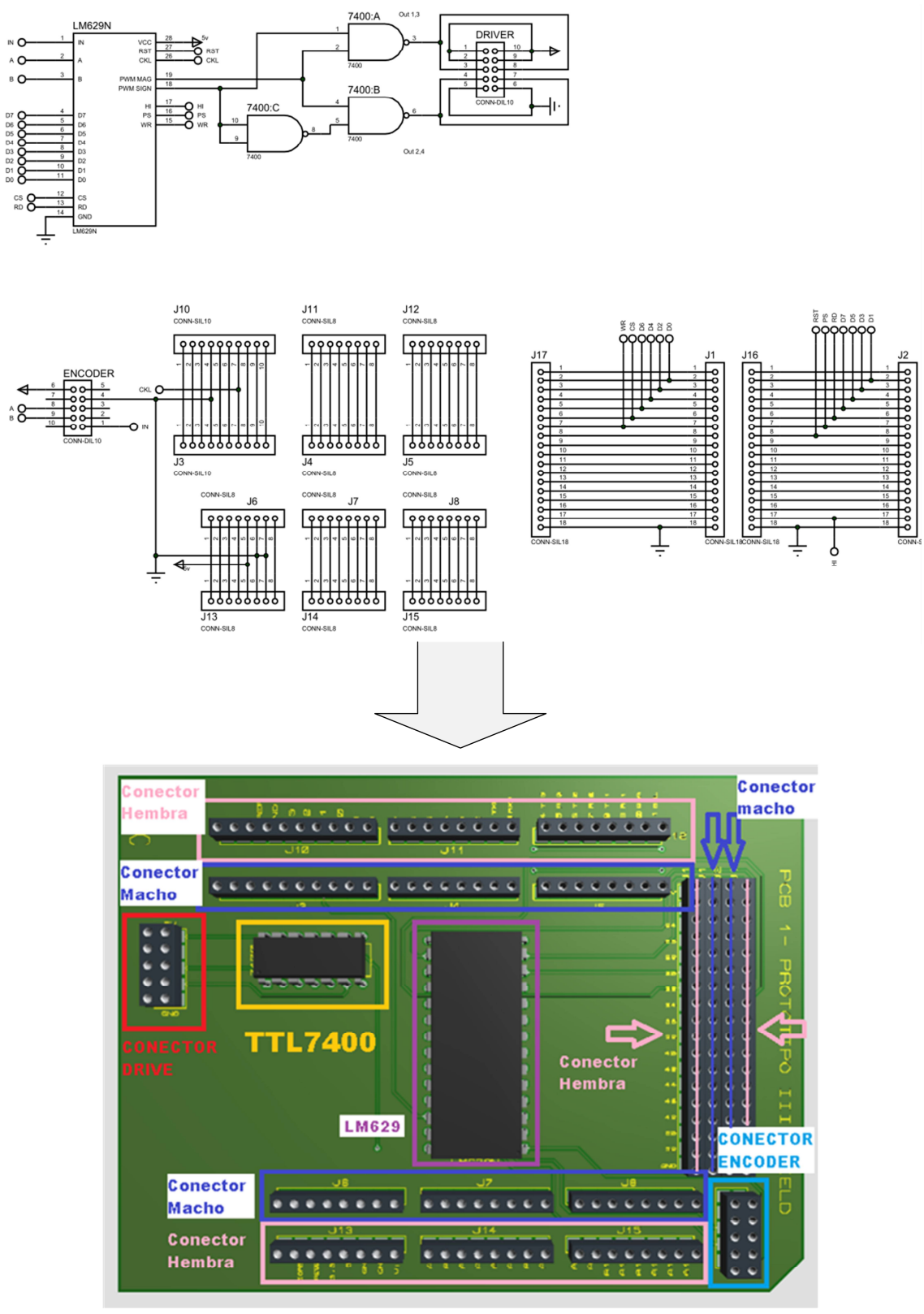

Figura 5.2. Planos eléctricos de la primera tarjeta electrónica bajo diseño

En la primera tarjeta electrónica se montan los componentes electrónicos de baja potencia y responsables del control lógico, la primer tarjeta contiene los siguientes componentes: un controlador LM629N-8, un circuito integrado 7408 contiene las compuertas lógicas tipo AND para el proceso de la señal de control PWM saliente del controlador, un zumbador para validar auditivamente una interrupción por parte del controlador, un conector de 10 pines para la conexión del Encoder Incremental con el modulo interno «Incremental Decoder » del controlador LM629N-8. La segunda tarjeta electrónica contiene los componentes que forman parte del controlador de potencia XY-160D para operar motores de corriente continua de hasta 7A por lo anterior fue útil usar un relevador para manipular la corriente demandante, la tarjeta fue diseñada bajo un plano electrónico y ambos son mostrados en la Figura 5.3 y 5.4.

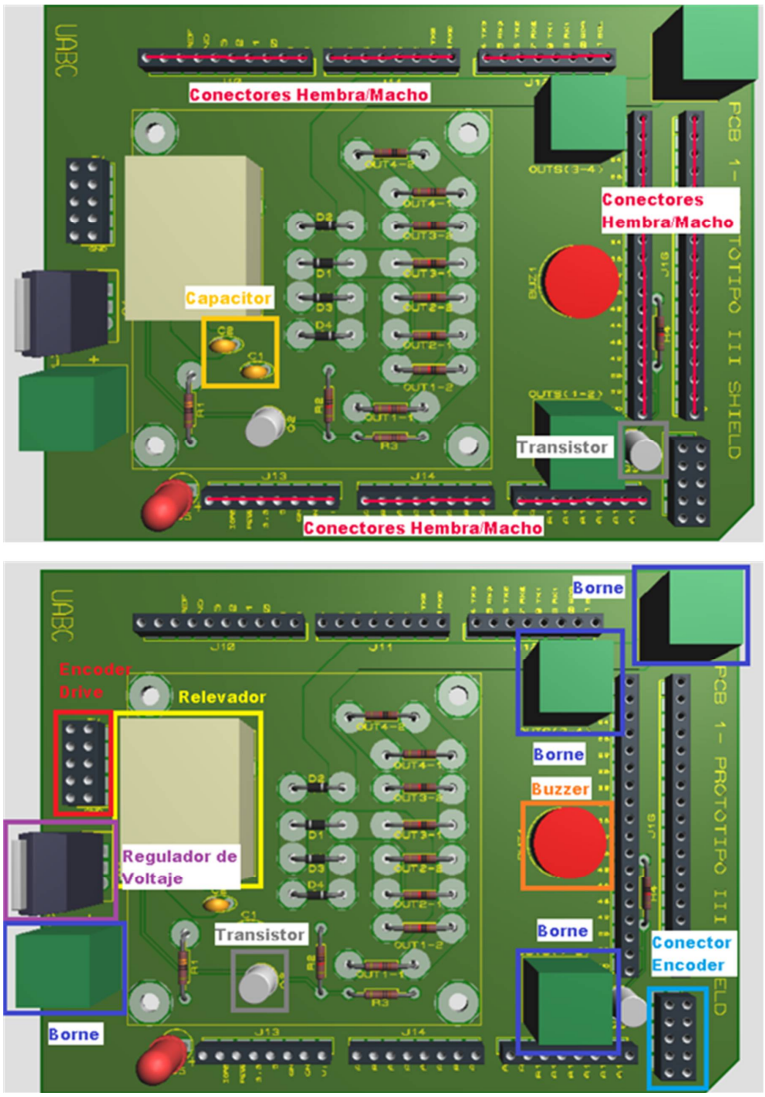

Figura 5.3. Segunda tarjeta electrónica bajo diseño

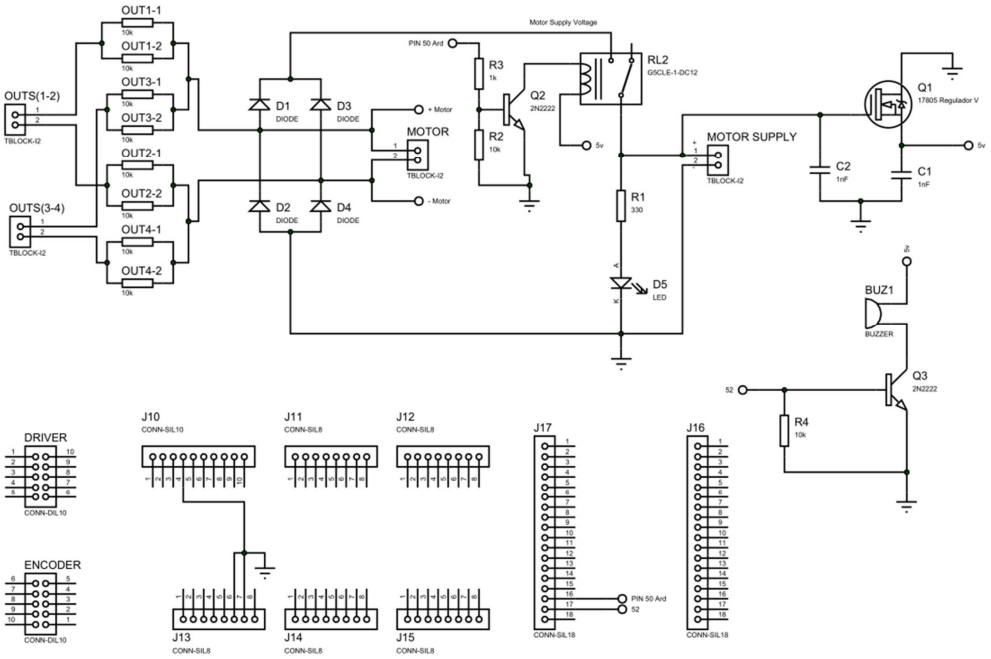

Figura 5.4. Planos eléctricos de la segunda tarjeta electrónica bajo diseño

La Figura 5.5, muestra el alcance del sistema de control propuesto en el capítulo 4 para el TVS No. 3, debido a que el sistema de control cuenta con dos modos de operación: modo posición y modo velocidad, el sistema de control propuesto en esta tesis es funcional para girar el sistema TVS No. 3 usando su brazo principal en modo posición y en modo velocidad al motor de corriente continua de la apertura de escaneo.

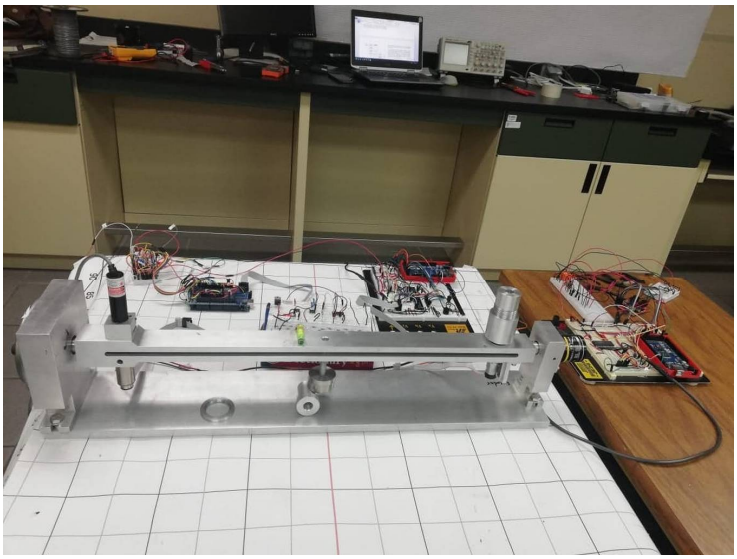

Figura 5.5. Sistemas de control de movimiento funcionales para el TVS No. 3

La Figura 5.6, presenta a vista general las dos tarjetas electrónicas, que fueron desarrolladas, al par del trabajo de esta tesis, estas tarjetas permiten evitar los falsos contactos eventuales durante el desarrollo de la experimentación. Estas tarjetas son montables sobre la tarjeta Arduino Mega 2560.

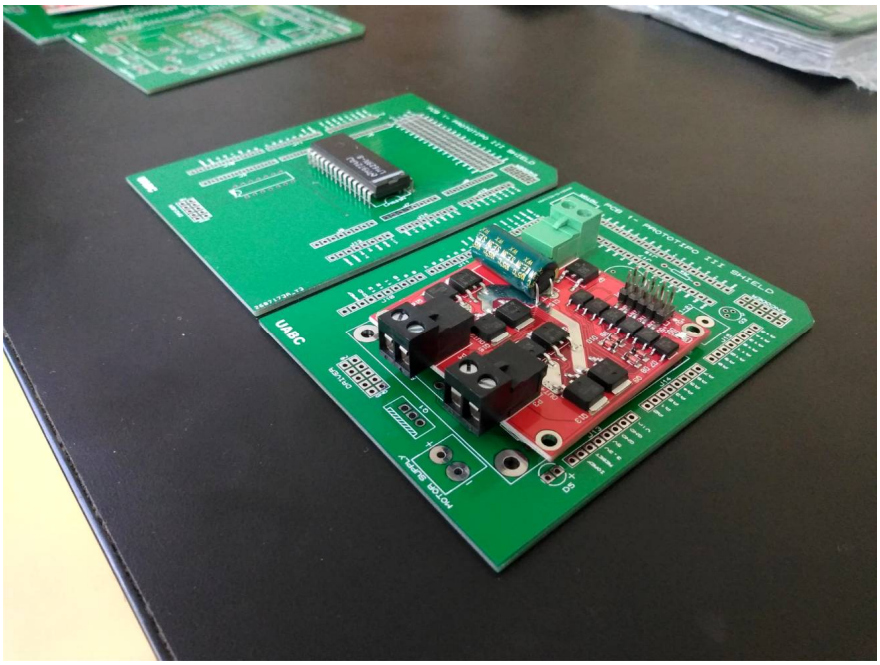

Figura 5.6. Tarjetas electrónicas impresas

5.2. Integración de Software

La metodología de la Programación Orientada a Objetos (POO) fue usada dentro de la experimentación y para la integración de software usando el esquema de la herencia de las clases a desarrollar en la Figura 4.5 en la sección 4.2 del capítulo Metodología. Esta sección presenta como se llevó a cabo la integración del software involucrado al Sistema de Control de Movimiento (SCM) como ejemplos de esta implementación fueron tomados dos archivos hechos; el archivo «.h» y «.ccp» demostrando la estructura en cada una de ellas. El archivo .h en Fig. 5.7 le corresponde a «ControlLines» colocando los atributos (Variables y constantes iniciales públicos) y las operaciones (funciones correspondientes a las líneas de control). Las líneas de control son los pines físicos que deberán estar en diferentes estados lógicos para configurar al puerto paralelo de comunicación como puerto de escritura o lectura.

```
/*
   ControlLines.h - library for Motion Control code using LM629 controller.
   Created by Miguel Reyes, June 30, 2018.
   Released into the public domain.
*/

#ifndef ControlLines_h
#define ControlLines_h
#include "Arduino.h"

class ControlLines
{
 public:

  const byte cs = 30; //ok
  const byte rd = 31; //ok
  const byte wr = 32; //ok
  const byte ps = 33; //ok

   ControlLines( ); // Constructor

    void WritingCommands(void);
    void ReadingStatus(void);
    void ReadingData(void);
    void WritingData(void);
    void HostInput(void);
    void HostOutput(void);
    void StandbyPortC(void);

  private:

  };

#endif
```

Figura 5.7. Ejemplo de sintaxis para crear la clase «ControlLines.h»

La Figura 5.8 muestra la estructura de la clase «*COMSerial.h*» para interpretar el tipo de dato del primer comando que recibe la tarjeta Arduino Mega por parte de la GUI estableciendo comandos hexadecimales. La función desarrollada para la clase COMSerial.h es requerida cada vez que es llamada y convierte comandos de tipo cadena a tipo hexadecimal, el primer dato tipo cadena proveniente de la GUI está definido a dos caracteres los cuales representan un numero hexadecimal para interpretarlo como un comando.

```
/*
   COMserial.cpp - library for Motion Control code using LM629 controller.
   Created by Miguel Reyes, July 15, 2018.
   Released into the public domain.
*/

#include "Arduino.h"
#include "COMserial.h"


COMserial::COMserial() // Constructor
{

}
int  COMserial::StringToHex(String hexNum)
{
      int value = 0;
      int valueByte = 0;
      char arrayDigit[3] = {'\0'};
      int digit[3] = {'\0'};
      boolean flag_00 = false;
      hexNum.toCharArray(arrayDigit, 3);

      for (int i = 0; i <= 1; i++)
      {
            if (arrayDigit[i] >= '0' && arrayDigit[i] <= '9')
            digit[i] = String(arrayDigit[i]).toInt();
            if (arrayDigit[i] == 'A')
            digit[i] = 10;
            if (arrayDigit[i] == 'B')
            digit[i] = 11;
            if (arrayDigit[i] == 'C')
            digit[i] = 12;
            if (arrayDigit[i] == 'D')
            digit[i] = 13;
            if (arrayDigit[i] == 'E')
            digit[i] = 14;
            if (arrayDigit[i] == 'F')
            {
                  digit[i] = 15;
            }
            flag_00 = true;
      }
      while (flag_00)
      {
      value = (digit[0] * 16 + digit[1] * 1);
            flag_00 = false;
            return value;
      }
}
```

Figura 5.8. Ejemplo de sintaxis para crear la función de la clase «COMSerial.h»

Panel frontal de la Interface Gráfica de Usuario GUI

La Interface Gráfica de Usuario es desarrollada en la plataforma LabVIEW usando la metodología de la sección 4.3. La Figura 5.9, presenta las 9 secciones (The tools palette) para habilitar todas las funciones del controlador LM629N-8. La descripción de Izquierda a derecha a continuación: la sección «Interrupt» contiene 7 botones sostenidos, los cuales al ser liberados después de ser presionados (Latch when released) cambian su estado «OFF-Apagado» a «ON-Encendido» activando un evento configurado para accionar una interacción al proceso indicado para cada uno de los botones, al procesar los datos y comandos que forman parte de la función que es determinada por cada botón. La sección «Serial Monitor» contiene un monitor para desplegar la información almacenada en el búfer de lectura del puerto serial. El despliegue de la información dentro del monitor es esencial para conocer el estado del controlador LM629N-8. La sección «Control Panel» presenta un cuadro de información de los parámetros a tomar en cuenta a la hora de descargar la configuración de la trayectoria trapezoidal, al final se presenta el botón para cerrar sesión de la GUI. Posteriormente en la parte derecha abajo del botón «Close session» se encuentra la sección «Filter» en ella contienen dos botones para enviar la configuración de las ganancias y actualizar el registro del controlador PID interno del controlador LM629N-8, debajo de la sección «Filter» se encuentra la sección «Trajectory» en esta se presentan dos botones, el primer botón sirve para ingresar los datos que configuran el perfil de la trayectoria trapezoidal de la varible de control, y el segundo botón sirve para energizar el motor dando un desplazamiento angular controlado.

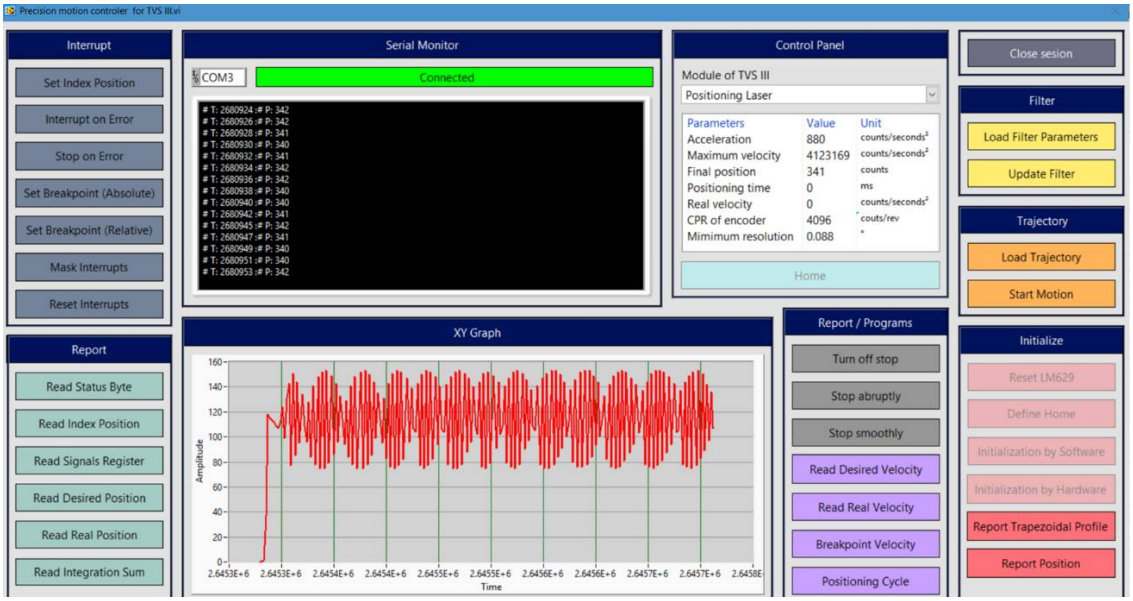

Figura 5.9. Panel frontal de GUI desarrollada en LabVIEW

Debajo de la sección de «Trajectory» se encuentra la sección «Initialize» en esta se presentan 6 botones dedicados para ejecutar rutinas de instrucciones dedicadas a la inicialización del controlador LM629N-8, apoyándose de la sección «Serial Monitor» obteniendo la respuesta de salida del controlador para determinar la correcta o incorrecta inicialización. A un lado de la sección de «Initialize» se presenta la sección «Report-Programs» en este se concentra 8 botones, los primeros 3 se disponen a parar el motor Maxon DCX22S, los botones de color morado son dedicados para reportar los datos indicados en la etiqueta de cada botón y enfocados al estado de los parámetros de la velocidad cargada al controlador, con excepción del ultimo botón, debido a que este botón crea una rutina de desplazamientos angulares configurables. A lado de la sección «Report-Programs» se encuentra un monitor para desplegar las trayectorias posición o velocidad trapezoidal obtenidas previamente solicitadas al dar clic al botón «Report Position» dentro de la sección «Initialize», finalmente existe la sección «Report» en esta se concentra los siete botones para reportar datos descargados en los registros relacionados con la etiqueta de los botones. Las respuestas del controlador son mostradas de inmediato en la pantalla de la sección «Serial monitor».

La Figura 5.10 presenta una parte del diagrama de bloques (programación visual) en un instrumento virtual en la plataforma LabVIEW, en la Figura tenemos el primer ciclo para eventos por interrupción conforme al esquema de la Figura 4.7 de la sección 4. El ciclo while mostrado en la Figura 5.10 contiene una estructura evento, el cual contiene 32 eventos posibles en cada uno de ellos contiene procesos diferentes y referentes para cada botón activado del panel de control de la Figura 5.9.

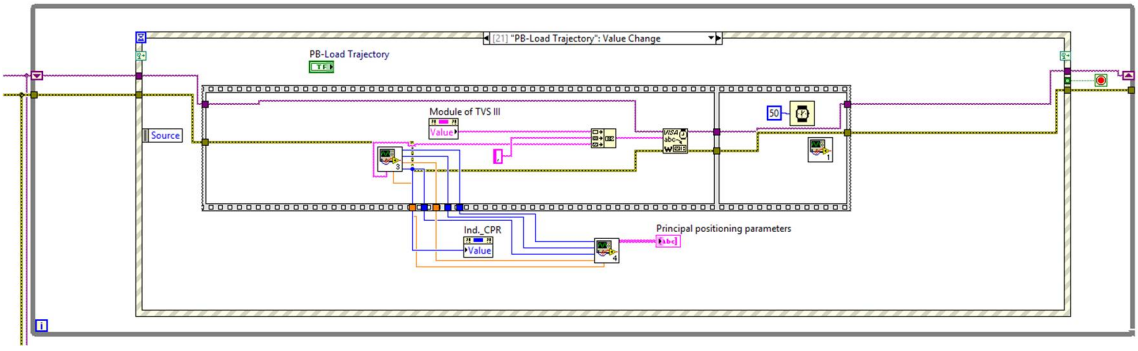

Figura 5.10. Ciclo de eventos activados por los botones del Panel Frontal de la GUI

La Figura 5.11, presenta el ciclo encargado de ejecutar tres posibles generadores de eventos por datos o comandos los cuales son obtenidos por el bloque de lectura, cada 500 microsegundos está leyendo el búfer de lectura del puerto serial, el bloque de lectura es configurable con 300 bytes por paquete en cada lectura de forma arbitraria. Existen tres generadores de eventos por datos, el primero crea una interrupción para reportar la posición

actual del eje del motor usando la información que nos proporciona el controlador LM629N-8. El segundo generador habilita un paro de motor siempre y cuando el bloque de lectura procesa alguna de estas posibles frases «Turn off», «Stop Abruptly» o «Stop Smoothty»; finalmente el tercer generador activa el evento para almacenar los datos de la trayectoria a graficar.

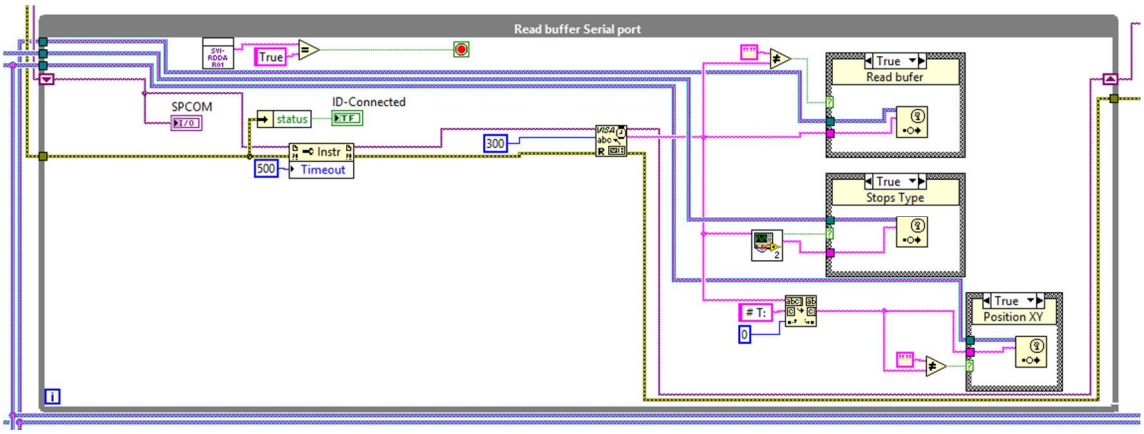

Figura 5.11. Ciclo para habilitar la comunicación serial

Ciclo de eventos por interrupción de datos visto en la Figura 5.12, hay cuatro diferentes eventos en cada uno contiene las acciones que son ejecutadas cuando algún generador de interrupciones dentro el ciclo de lectura serial, detecta una frase configurada para la interrupción por eventos. Los eventos son los siguientes: « PB-Close session » al activar el botón pulsador (close session) desde el panel frontal, «Read buffer» es activado al detectar las cuentas del módulo «summing union» interno del controlador LM629N-8 para brindar la información del Encoder Incremental, «Plot XY» es activado por la detección de los datos posteriores de la frase: «# T:» y «Turn off motor» es activado por estas posibles frases «Turn off», «Stop Abruptly» o «Stop Smoothty».

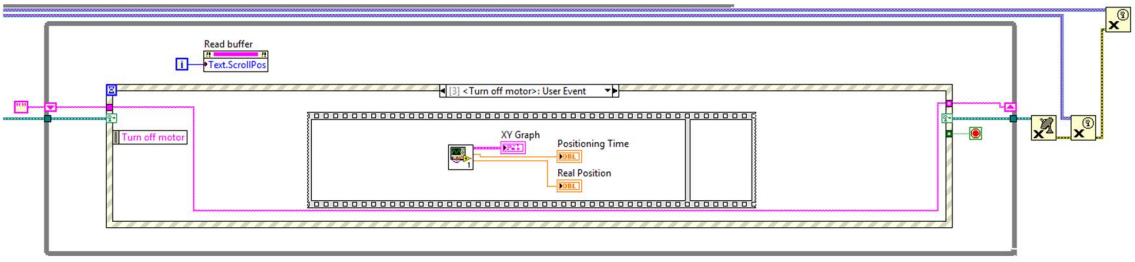

Figura 5.12. Ciclo de eventos por interrupción de datos

La Figura 5.13, presenta el inicio del bus de eventos de usuario y datos, al presentarse el bloque de crear los registros por eventos. De igual forma se presenta el bloque de la apertura

de comunicación serial, creando el « Handshaking » entre la tarjeta Arduino 2560 y el puerto de la comunicación serial.

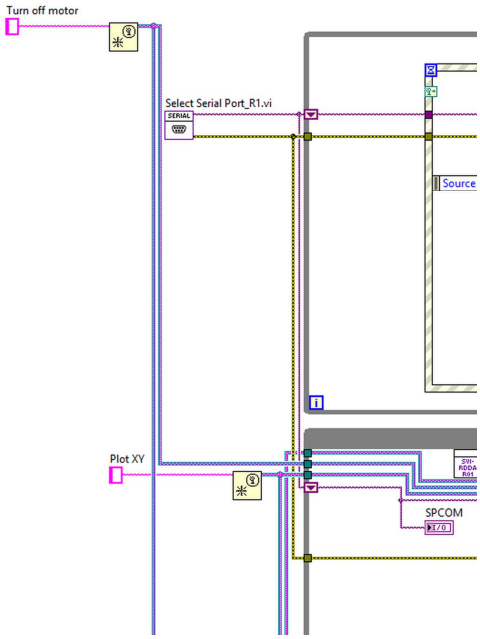

Figura 5.13. Inicialización de generador de eventos y abrir sesión serial

La Figura 5.14, presenta el bloque «Reg Events» el cual representa el nodo del bus de la interfaz de los eventos por usuario.

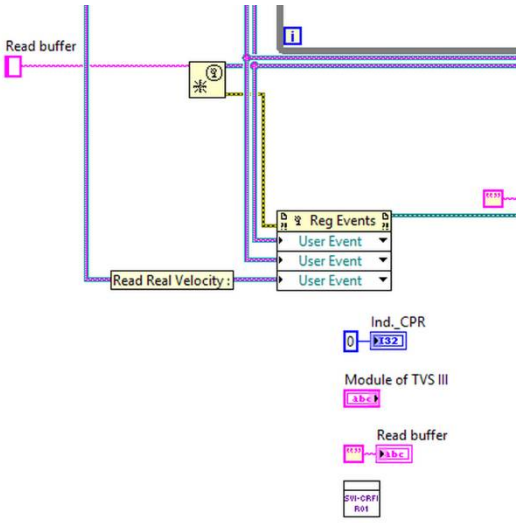

Figura 5.14. Bus de cola para el registro de eventos

La Figura 5.15, presenta los bloques que son ejecutados cada vez que el usuario desea salir de la sesión de la interface GUI. Hay una estructura secuencial dentro de ella, contiene un subVI (Sub Instrumento Virtual) esté limpia los registros usados durante la sesión abierta. Posteriormente, al ser ejecutado la estructura secuencia cierra la sesión de la comunicación serial.

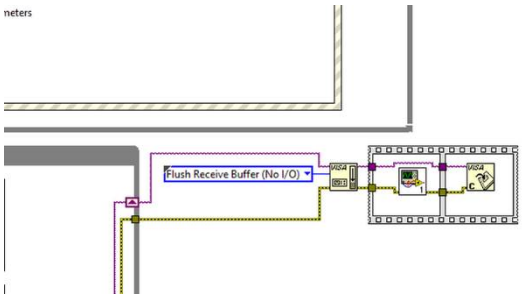

Figura 5.15. Bloques responsables para cerrar la sesión del GUI

5.3. Ejecución de Posicionador Láser desde la GUI

Al ejecutar la GUI desarrollada, se presenta la ventana emergente en la Figura 5.16 para abrir la sesión de la comunicación serial entre la tarjeta Arduino 2560 y la computadora, solo es necesario seleccionar el puerto que está ligado a la caja de control para ingresar el número de puerto (COM #).

Figura 5.16. Ventana emergente para la apertura de la comunicación serial

Una vez que se habilita la comunicación serial al seleccionar el puerto serial (COM #) se tiene que inicializar el controlador LM629N-8, y después de validar la inicialización del controlador al esperar la respuesta del controlador mediante el serial monitor, para la sintonización del controlador PID es necesario usar las ganancias de los coeficientes del controlador (k_p, k_i, k_d). La Figura 5.17, muestra la ventana emergente para dar de alta los valores de los coeficientes del controlador PID. Al cargar los valores es presionado el botón «Load» para transferir esos datos al controlador LM629N-8.

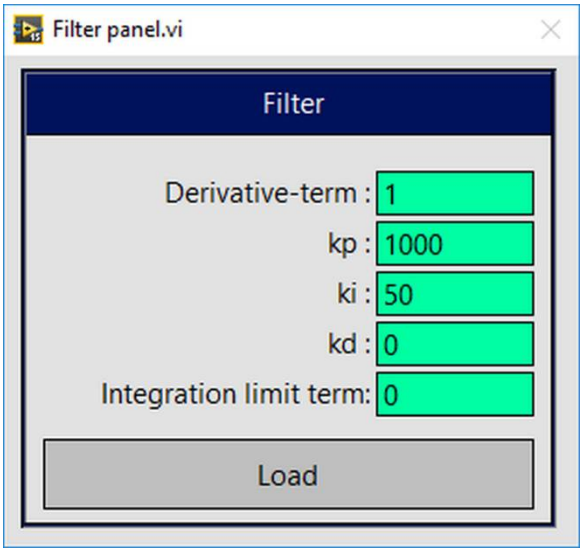

Figura 5.17. Ventana emergente para ajustar el controlador PID

Después de descargar los coeficientes del controlador PID, se actualiza el registro del ajuste del controlador PID, al dar clic al botón pulsador «Update Filter» deberá ser necesario la actualización de datos para validar la correcta manera de ajustar el controlador PID. Posteriormente, es necesario dar de alta una configuración de trayectoria para descárgala al módulo «Generador de trayectoria» interno del controlador LM629N-8. La Figura 5.18, presenta todas las funcionalidades para configurar las trayectorias, desde habilitar las magnitudes relativas o absolutas de aceleración, velocidad y posición, el modo de la trayectoria; velocidad o posición, el tipo de parada que se desea habilitar, posteriormente 4 campos para ser llenados los cuales cargan los valores de la aceleración en $\frac{revoluciones}{s^2}$, máxima velocidad $\frac{revoluciones}{s}$, Posición final en grados (*deg*) y los pulsos por revolución *ppr* del Encoder Incremental aplicado. Finalmente, el botón «Load T. Parameters» al ser presionado se descargara la configuración de esa ventana hacia la tarjeta Arduino Mega 2560 y después al controlador LM629N-8.

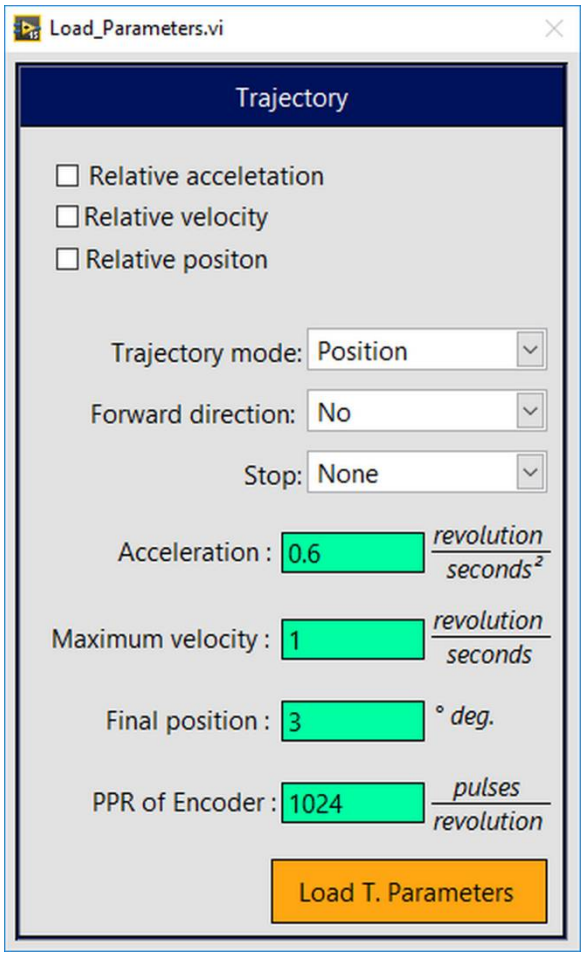

Figura 5.18. Ventana emergente para configurar la trayectoria trapezoidal

5.4. Diseño de Experimentación

La experimentación se llevó a cabo al tener funcional el Sistema de Control de Movimiento usando la metodología del capítulo 4, durante el seguimiento de la experimentación se hicieron modificaciones favorables para el mismo sistema, con el objetivo central de obtener una reducción del error de posicionamiento y operar el SCM con el mínimo tiempo de posicionamiento para PL del sistema TVS No. 3 usando el controlador LM629N-8 y el motor Maxon DCX22S. Los factores principales para la experimentación están en la Tabla 5.1. La sintonización del controlador PID fue empírica conforme a prueba y error, observando el comportamiento del motor. Al conseguir los coeficientes favorables del controlador PID se realizaron las primeras pruebas experimentales. De hecho, más de 200 pruebas fueron hechas

en diferentes condiciones. Sin embargo, para esta tesis, solo se darán a conocer la experimentación que tiene trascendencia para los objetivos de la propuesta de esta tesis.

Los principales factores para la experimentación son presentados en la Tabla 5.1. Donde, la aceleración de referencia a_r es requerida e ingresada en la GUI, sus unidades de medida son $\frac{revolucion}{s^2}$, la velocidad máxima de referencia ω_{max} usa las unidades $\frac{revolucion}{s}$ y es requerida de igual forma que la aceleración. La posición angular de referencia φ_r es arbitraria y está dentro de cualquier posición angular > ρ_{min}, la posición angular final φ_o es obtenida al calcular la función trigonométrica $arctan\left(\frac{a}{b}\right)$ en las mediciones del plano XY limitado al campo de visión del TVS No.3. El tiempo de posicionamiento es obtenido al usar un osciloscopio Tektronix TBS 2000 Series mediante los canales A y B del Encoder Incremental Maxon ENX 16, la posición discreta angular de referencia N_m es obtenido al solicitar información por el modulo interno «Summing Union» usando un comando desde la GUI para reportar la cantidad del conteo de flancos de subida y bajada, los demás factores son definidos con las siguientes ecuaciones mostradas (5.4.1 a 5.4.8) usando la hoja de datos del Controlador LM629N-8 [27].

Tabla 5.1. Principales Factores de la experimentación

No.	Principales Factores		
	Parámetros	**Símbolo**	**Unidad**
1.	Aceleración de referencia	a_r	$\frac{revolution}{s^2}$
2.	Velocidad máxima de referencia	ω_{max}	$\frac{revolution}{s}$
3.	Posición angular de referencia	φ_r	deg
4.	Posición angular final	φ_o	deg
5.	Tiempo de posicionamiento	t_φ	ms
6.	Cuentas por posición angular de referencia	N_r	$\frac{cuentas}{revolution}$
7.	Cuentas por posición angular final	N_m	$\frac{cuentas}{revolution}$
8.	Mínima resolución	ρ_{min}	deg
9.	Resolución de Encoder Incremental	ppr	$\frac{cuentas}{revolution}$
10.	Error de posición angular relativo	φ_e	deg
11.	Promedio de error de posición angular relativo	$\bar{\varphi}'_e$	$\%$
12.	Promedio de tiempo de posicionamiento	$\bar{t}'_\varphi$	ms
13.	Promedio de posición angular final	$\bar{\varphi}'_o$	deg

La aceleración de referencia está definida por la ecuación 5.4.1, a_r es aproximado por la equivalencia al multiplicar los siguientes factores: Pulsos Por Revolución *cpr* por 4 debido a que el controlador LM629N-8 cuadruplica la resolución del Encoder Incremental, tiempo de muestreo al cuadrado $t_s^2 = 256\,\mu s$ al usar 8 MHz del reloj del sistema, aceleración de referencia ingresada en la sección «Trajectory» en la GUI y la constate 2^{16} para convertir a la a_r en el formato del registro de 2 bytes de la aceleración en memoria del controlador.

$$a_r \approx ppr \cdot 4 \cdot t_s^2 \cdot a \cdot 2^{16} \tag{5.4.1}$$

La velocidad de referencia está definida por la ecuación 5.4.2, a_r es aproximado por la equivalencia al multiplicar los siguientes factores: Pulsos Por Revolución *ppr* por 4 debido a que el controlador LM629N-8 cuadruplica la resolución del Encoder Incremental, tiempo de muestreo al cuadrado $t_s^2 = 256\,\mu s$ al usar 8 MHz del reloj del sistema, velocidad máxima ingresada en la sección «Trajectory» en la GUI y la constate 2^{16} para convertir a la ω_r en el formato del registro de la velocidad 2 bytes en memoria del controlador.

$$\omega_r \approx ppr \cdot 4 \cdot\ t_s^2 \cdot \omega_{max} \cdot 2^{16} \tag{5.4.2}$$

El desplazamiento mínimo para ser identificado por el Enconder Incremental es definido por la ecuación 5.4.3. Al dividir 360 grados sobre la máxima resolución de la información del Encoder Incremental.

$$\varphi_{min} = \frac{360}{ppr \cdot 4} \tag{5.4.3}$$

Posición discreta angular de referencia es discreta por ende la aproximación a la equivalencia de la ecuación 5.4.4.

$$N_r \approx \frac{\varphi_r}{\varphi_{min}} \tag{5.4.4}$$

EL error relativo de posición es definido por la ecuación 5.4.5.

$$\varphi_e = \frac{|\varphi_r - \varphi_o|}{\varphi_r} \cdot 100 \tag{5.4.5}$$

Las ecuaciones 5.4.6 a 5.4.8 definen los promedios error relativo de posición, el tiempo de posicionamiento, y la posición angular de salida, respectivamente, definiendo n como el número de pruebas experimentales.

$$\bar{\varphi}_e' = \frac{1}{n}\sum_{i=1}^{n} \varphi_{ei} \tag{5.4.6}$$

$$\bar{t}'_{\varphi} = \frac{1}{n}\sum_{i=1}^{n} t_{\varphi i} \tag{5.4.7}$$

$$\bar{\varphi}'_{o} = \frac{1}{n}\sum_{i=1}^{n} \varphi_{oi} \tag{5.4.8}$$

Un plano cuadriculado fue considerado para obtener dimensiones del campo de visión experimental limitado en el área de trabajo del TVS No.3. Las dimensiones son relacionadas al plano *XY* en la Figura 5.19, se define la función trigonométrica para la obtención de la posición angular de salida φ_o fijando una distancia entre la apertura del rayo láser proveniente del Posicionador Láser a la pared del cuadriculada donde es reflejado el rayo láser. Para cuantificar el tiempo de posicionamiento de cada prueba dentro de la experimentación se usó un osciloscopio TEKTRONIX TBS 2000 SERIES semejante al de la Figura 5.20, usando los canales A y B para un monitorio de las señales incrementales de ambos canales.

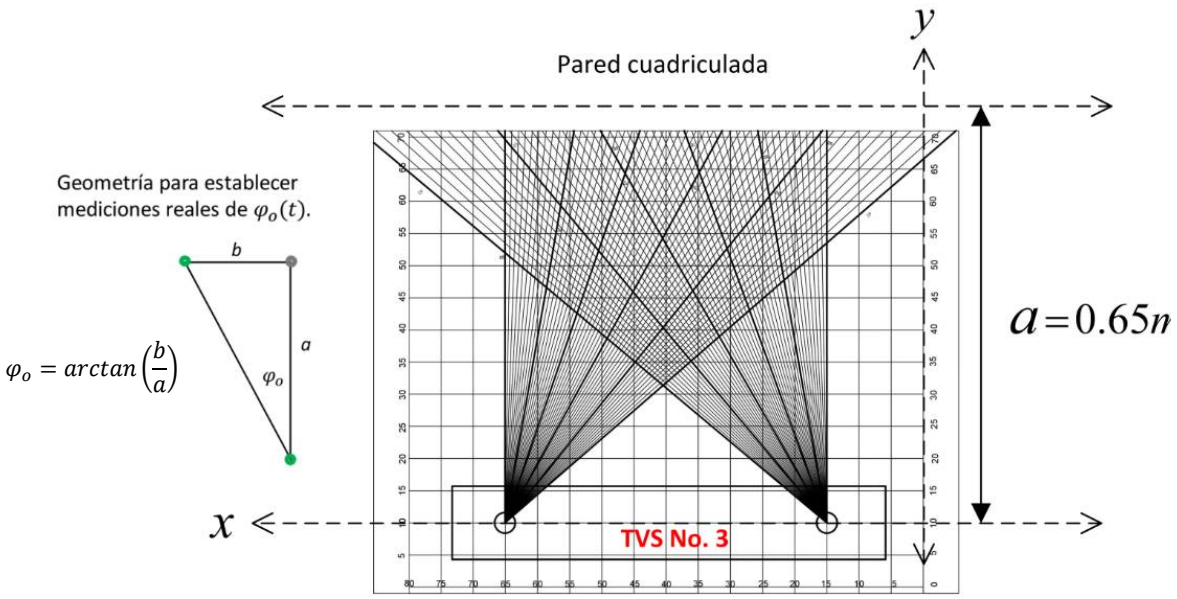

Figura 5.19. Plano cuadriculado para obtener resultados de $\varphi_o(t)$

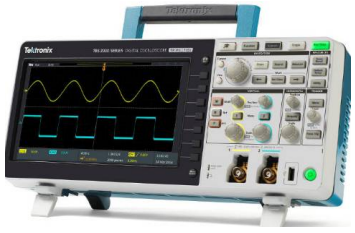

Figura 5.20. Osciloscopio Tektronix TBS 2000 SERIES

Un diseño de experimentación fue desarrollado conforme a la metodología Taguchi vista en el capítulo 3 con la finalidad de conocer los efectos al probar 4 factores sospechosos usando 2 niveles. Los factores son: (A) Aceleración de referencia a_r, (B) Velocidad máxima de referencia ω_{max}, (C) Ganancia proporcional k_p, (D) Ganancia integrativa k_i. Estos son mostrados en la Tabla 5.2. Para evaluar los factores significativos que influyen al obtener el mínimo tiempo de posicionamiento t_φ manteniendo un desempeño optimo del SCM para el Posicionado Láser del TVS No. 3.

Tabla 5.2. Factores experimentales a dos niveles

Factor	Descripción	Nivel 1	Nivel 2	Unidad
A	Aceleración de referencia	$0.1 \leq A_1 \leq 100$	$0.1 \leq A_2 \leq 100$	$\frac{revolution}{s^2}$
B	Velocidad máxima	$0.1 \leq B_1 \leq 100$	$0.1 \leq B_2 \leq 100$	$\frac{revolution}{s}$
C	Ganancia proporcional	$1 \leq C_1 \leq 3000$	$1 \leq C_2 \leq 3000$	-
D	Ganancia integrativa	$1 \leq D_1 \leq 3000$	$1 \leq D_2 \leq 3000$	-

En la Tabla 5.3, presenta el arreglo ortogonal con las 16 pruebas diferentes posibles, usando dos niveles: 1 y 2.

Tabla 5.3. Arreglo Ortogonal L_{16} usado en la experimentación

No. de Prueba	A	B	C	D
1.	A_1	B_1	C_1	D_1
2.	A_1	B_1	C_1	D_2
3.	A_1	B_1	C_2	D_1
4.	A_1	B_1	C_2	D_2
5.	A_1	B_2	C_1	D_1
6.	A_1	B_2	C_1	D_2
7.	A_1	B_2	C_2	D_1
8.	A_1	B_2	C_2	D_2
9.	A_2	B_1	C_1	D_1
10.	A_2	B_1	C_1	D_2
11.	A_2	B_1	C_2	D_1
12.	A_2	B_1	C_2	D_2
13.	A_2	B_2	C_1	D_1
14.	A_2	B_2	C_1	D_2
15.	A_2	B_2	C_2	D_1
16.	A_2	B_2	C_2	D_2

6. Resultados de la Experimentación

Este capítulo, divulga los resultados de la experimentación vista en el capítulo anterior obteniendo los alcances y limitaciones del sistema de control para el Posicionador Láser (PL) del TVS No. 3 propuesto en la metodología del capítulo 4. Cuantificando el promedio de error de posición angular relativo $\bar{\varphi}'_e$, el valor promedio del tiempo de posicionamiento $\bar{t}'_\varphi$ como el tiempo mínimo en *ms* para desplazar el eje del motor Maxon DCX22S, conforme a diferentes posiciones angulares $\varphi_r(t)$ = (1°, 2°, 3°, 5°, 15°, 30°, 45°, 90°, 180° y 360°). En consecuencia, trabajando con un sistema de control estable y con un desempeño óptimo para el PL del TVS No. 3 usando el controlador LM629N-8.

6.1. Usando el Controlador Embebido LM629N-8

Al tener el sistema de control operacional para el Posicionador Láser (PL) visto en la Figura 5.5 del capítulo 5, fue posible conocer sus alcances y límites bajo la experimentación realizada. La interface gráfica de usuario pasó por varias modificaciones para mejorar el desempeño de la misma con más incidencias en la comunicación serial al tener retardos de la transferencia bidireccional de datos entre la computadora y la tarjeta Arduino 2560. La sintonización del controlador PID del controlador LM629N-8 se llevó a cabo empíricamente, con la finalidad de evitar sobreoscilaciones en la zona transitoria y estacionaria de la posición angular $\varphi_o(t \to \infty)$ del eje del motor Maxon DCX22S. En la Figura 6.1 se presentan seis imágenes diferentes de A) a E) son algunas de las respuestas de la posición angular de salida $\varphi_o(t \to \infty)$ durante la sintonización. En la imagen A), usando φ_r = 2° es notable un comportamiento senoidal obteniendo 1.93°$\leq \varphi_o(t) \leq$ 2.2°. Así mismo para las siguientes imágenes: B) a F), la sintonización fue comenzando con un coeficiente proporcional $\leq$ *100* unidas, observando el comportamiento del motor y la posición angular de salida $\varphi_o(t \to \infty)$) aproximándose a la posición angular de referencia φ_r. Posteriormente se uso el coeficiente proporcional hasta 1,500 unidades como límite. Por lo tanto, al usar solo la parte proporcional del controlador, no fue posible aproximarse al valor de referencia de φ_r bajo tiempos de posicionamientos cortos $t_\varphi \leq 500$ *ms*, el comportamiento oscilatorio de las imágenes A-C se eliminaron al incluir la parte integrativa del controlador PID, los efectos del controlador cambiaron registrando la imagen D) hasta F), donde la parte estacionaria de la señal $\varphi_o(t \to \infty)$ es estable. Sin embargo los comportamientos del motor Maxon DCX22S fueron semejantes, más nunca iguales bajo las mismas condiciones de la experimentación debido a presentar efectos de fricción y características físicas del motor DCX22S que fueron omitidos por la complejidad de análisis dentro del trabajo propuesto de esta tesis.

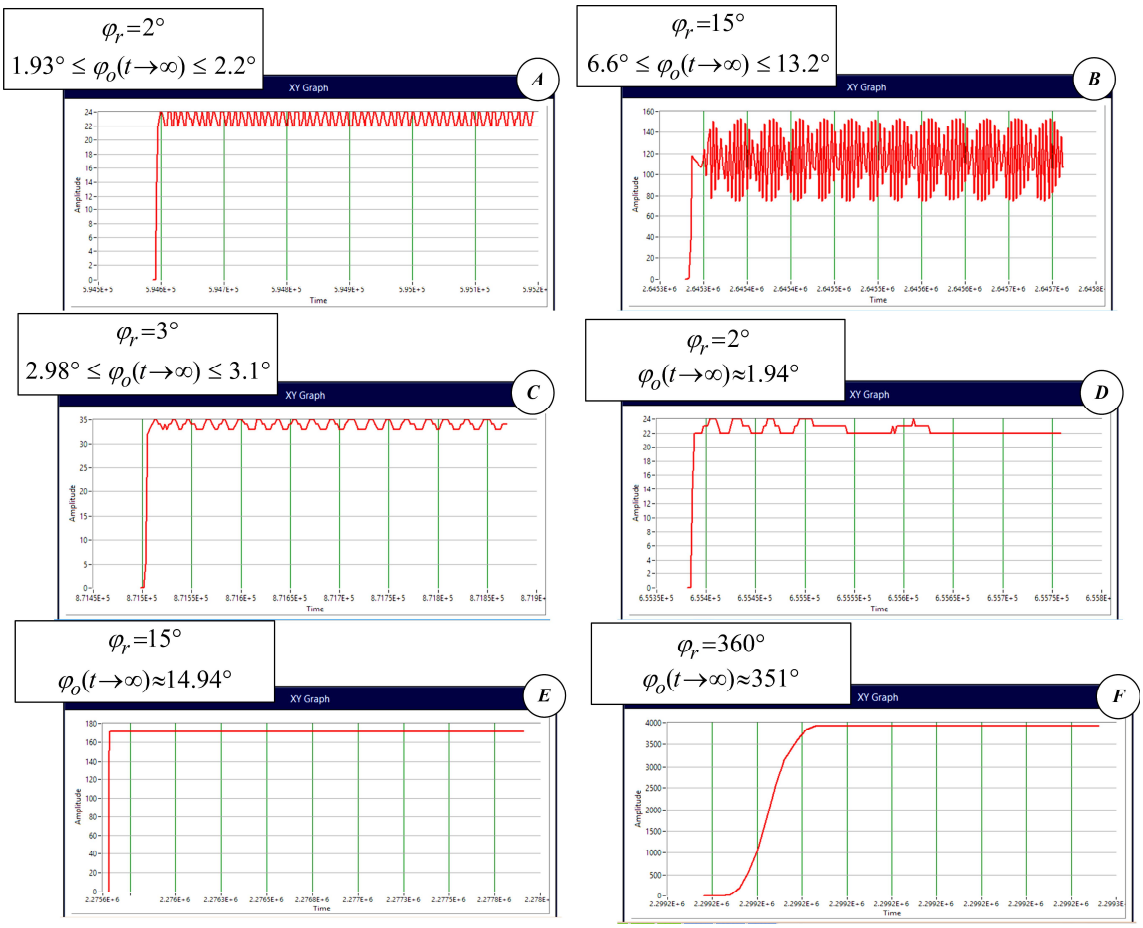

Figura 6.1. Diferentes comportamientos del eje del motor Maxon DCX22S bajo la lectura de $N_m(t)$

Después de que la sintonización fue lograda, al ajustar el controlador PID, con los valores propuestos a dos niveles en la Tabla 6.1, mostrando resultados de $\varphi_o(t)$ con un alto grado de precisión, durante las pruebas empíricas.

Tabla 6.1. Factores involucrados para la sintonización

Factores de Controlador PID	Nivel 2	Nivel 1
Ganancia proporcional	1,000, 1,050,1,100, 1,200,1,300, 1,500	1,000, 1200
Ganancia integrativa	10, 15, 100, 200, 250	5, 10, 50, 100

Para validar la respuesta del tiempo de posicionamiento t_φ para cada una de las pruebas realizadas, fueron utilizadas las señales incrementales de los canales A y B del sensor de posición Maxon ENX 16 empotrado con el motor Maxon DCX22S. Las señales incrementales fueron leídas por el osciloscopio Tektronix TBS Series. Para validar los

valores de cada posición angular de salida $\varphi_o(t)$ se usó la función trigonométrica $\varphi_o = arctag\left(\frac{a}{b}\right)$ siendo, a como el cateto opuesto y b como el cateto adyacente, trabajando conforme a la experimentación vista en la Figura 5.5, la Figura 6.2 es una evidencia de la instalación de la experimentación, donde se muestran líneas imaginarias para obtener φ_o donde, $a = 2\ m$ y b el valor en metros desplazados desde un punto de luz reflectada como la posición cero $\varphi_o = 0$ en la superficie de la pared debido al rayo láser emitido por el PL, hasta que el movimiento es parado al quedar estático el punto de luz reflectada llegando a la posición angular de salida $\varphi_o(t)$, también se utilizaron los parámetros de a y b propuestos en la Figura 5.19 del capítulo 5.

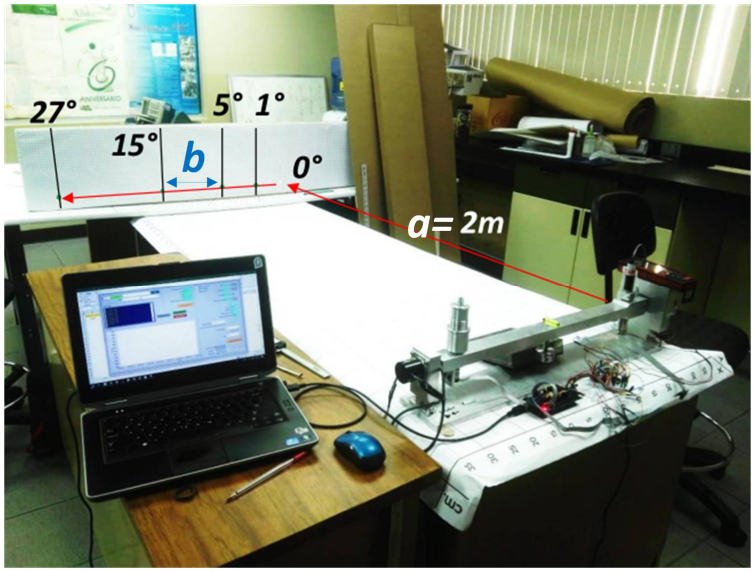

Figura 6.2. Primera instalación de la experimentación

6.2. Comparativa de los Factores Significantes

Los factores $k_p, k_i, a_r, \omega_{max}, N_m, \varphi_o, \varphi_e, t_\varphi$ presentados en las Tablas 6.2 a la 6.11, dan a conocer el comportamiento del eje del motor Maxon DCX22S bajo las condiciones experimentales vistas en el capítulo 5. Los valores en color verde son los considerados como los óptimos no repetitivos, sin embargo en términos cuantitativos es posible llegar a esos valores si se mejora la sintonización del controlador PID para velocidades y aceleraciones altas sin perder estabilidad para trabajos futuros. La precisión es notoria debido a que hay oscilaciones de $\mp$ 1 cuenta y la palabra «Osc» es referente a que el desplazamiento no fue estable, presentando oscilaciones durante la prueba en la zona transitoria y estacionaria

representada por $\varphi_o(t \to \infty)$ los valores de los factores fueron determinados conforme a un alto grado de precisión.

Tabla 6.2. Resultados de la experimentación usando $\varphi_r = 1°$ y $N_r = 11$

No. Prueba	k_p	k_i	a_r	ω_{max}	N_m	$\varphi_o \approx$	φ_e	t_φ
1.	1000	100	15	30	10	0.88°	12%	15 ms
2.	1000	100	15	40	10	0.88°	12%	12 ms
3.	1000	100	10	30	10	0.88°	12%	22.4 ms
4.	1000	100	10	40	10	0.88°	12%	22 ms
5.	1000	200	15	30	Osc	-	-	-
6.	1000	200	15	40	10	0.88°	12%	11.2 ms
7.	1000	200	10	30	11	0.96°	4%	827 ms
8.	1200	200	10	40	Osc	-	-	-
9.	1200	100	15	30	10	-	-	15.2 ms
10.	1200	100	15	40	Osc	-	-	-
11.	1200	100	10	30	10	0.88°	12%	22 ms
12.	1200	100	10	40	Osc	-	-	-
13.	1200	200	15	30	10	0.88°	12%	11.9 ms
14.	1200	200	15	40	10	0.88°	12%	12.3 ms
15.	1200	200	10	30	Osc	-	-	-
16.	1200	200	10	40	11	0.96°	4%	178 ms

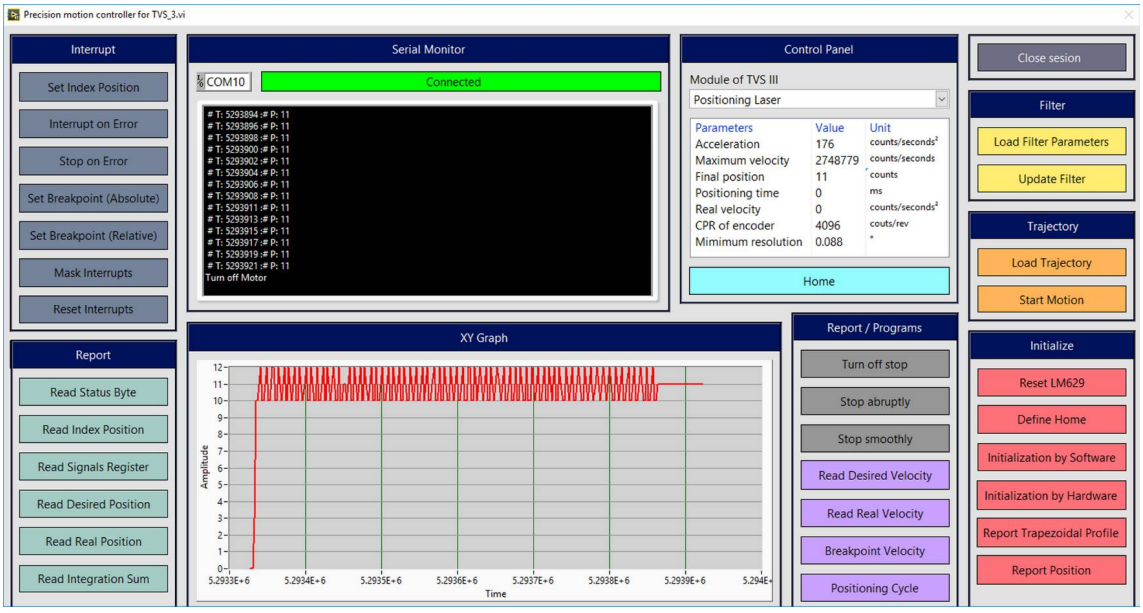

Figura 6.3. Prueba No. 7 para $\varphi_r = 1°$ con oscilaciones de ± 1 cuenta

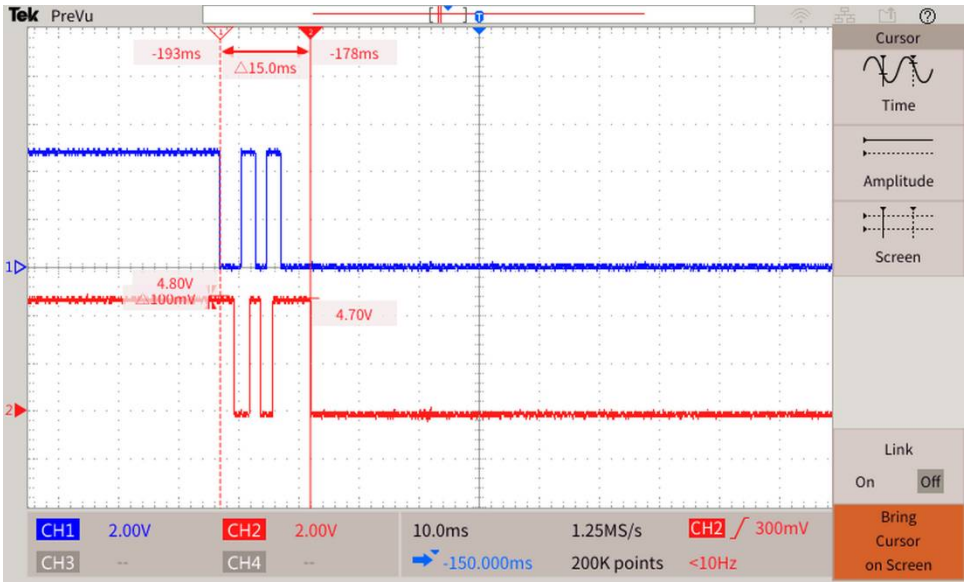

Figura 6.4. Prueba No. 1 para $\varphi_o \approx 1°$ obteniendo $N_m = 10$, $t_\varphi = 15\ ms$

Tabla 6.3. Resultados de la experimentación usando $\varphi_r = 2°$ y $N_r = 23$

No. Prueba	k_p	k_i	a_r	ω_{max}	N_m	$\varphi_o \approx$	φ_e	t_φ
1.	1000	100	15	30	22	1.93°	3.5%	118 ms
2.	1000	100	15	40	Osc	-	-	-
3.	1000	100	10	30	22	1.93°	3.5%	36 ms
4.	1000	100	10	40	22	1.93°	3.5%	38.4 ms
5.	1000	200	15	30	22	1.93°	3.5%	28 ms
6.	1000	200	15	40	22	1.93°	3.5%	29.6 ms
7.	1000	200	10	30	22	1.93°	3.5%	34 ms
8.	1200	200	10	40	22	1.93	3.5%	34 ms
9.	1200	100	15	30	22	1.93°	3.5%	38 ms
10.	1200	100	15	40	22	1.93°	3.5%	28.8 ms
11.	1200	100	10	30	22	1.93°	3.5%	63.2 ms
12.	1200	100	10	40	Osc	-	-	-
13.	1200	200	15	30	Osc	-	-	-
14.	1200	200	15	40	22	1.93°	3.5%	28 ms
15.	1200	200	10	30	23	2.02	1%	1 s
16.	1200	200	10	40	22	1.93°	3.5%	32.4 ms

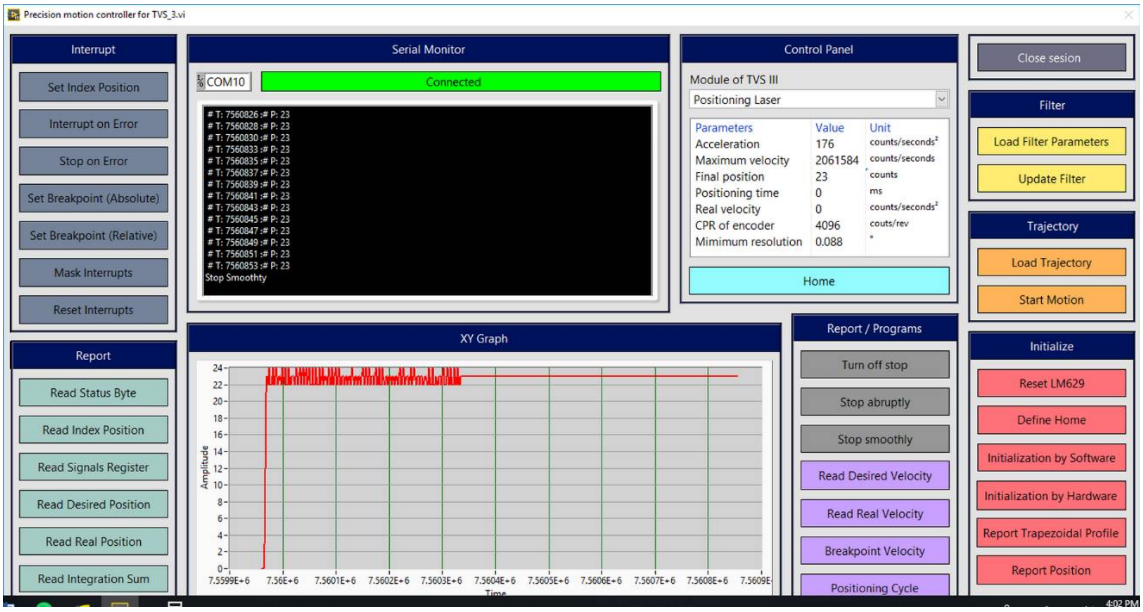

Figura 6.5. Prueba No. 15 para $\varphi_r = 2°$ con oscilaciones de ± 1 cuenta

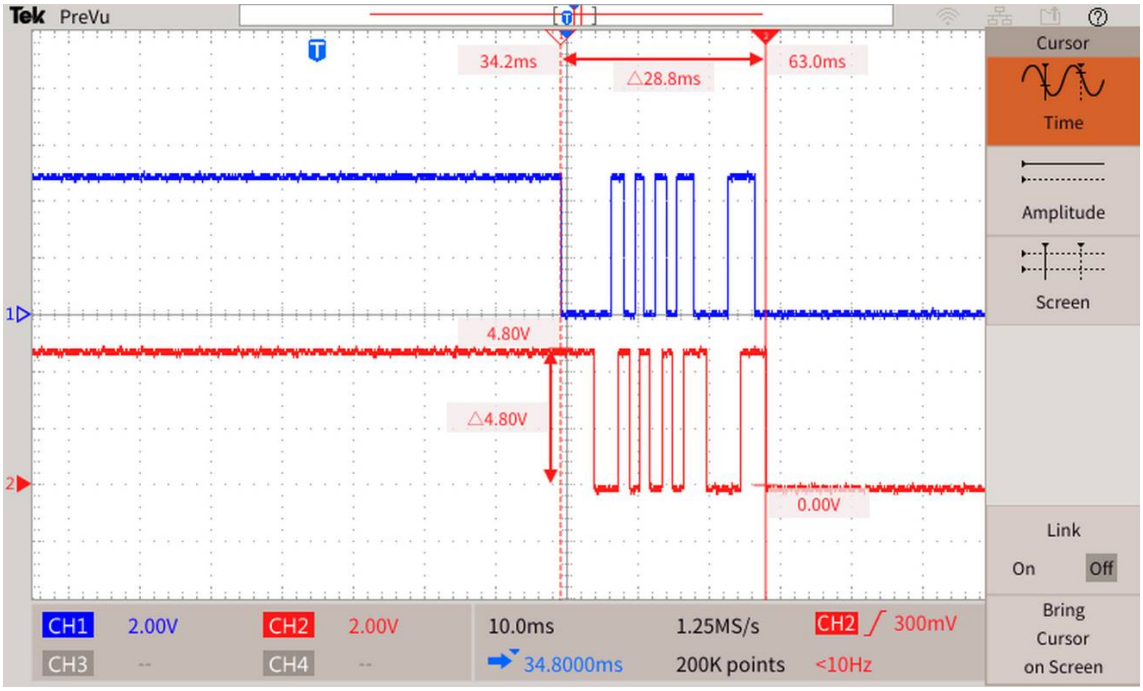

Figura 6.6. Prueba No. 10 para $\varphi_o \approx 2°$ obteniendo $N_m = 22, t_\varphi = 28.8\ ms$

Tabla 6.4. Resultados de la experimentación usando $\varphi_r = 3°$ y $N_r = 34$

No. Prueba	k_p	k_i	a_r	ω_{max}	N_m	$\varphi_o \approx$	φ_e	t_φ
1.	1200	10	15	30	34	2.98°	0.67%	34 ms
2.	1200	10	15	40	34	2.98°	0.67%	38 ms
3.	1200	10	10	30	34	2.98°	0.67%	97.2 ms
4.	1200	10	10	40	Osc	-	-	-
5.	1200	5	15	30	Osc	-	-	-
6.	1200	5	15	40	Osc	-	-	-
7.	1200	5	10	30	33	2.9°	3.34%	44.4 ms
8.	1200	5	10	40	Osc	-	-	-
9.	1300	10	15	30	Osc	-	-	-
10.	1300	10	15	40	34	2.98°	0.67%	58.2 ms
11.	1300	10	10	30	34	2.98	0.67%	654 ms
12.	1300	10	10	40	Osc	-	-	-
13.	1300	5	15	30	Osc	-	-	-
14.	1300	5	15	40	Osc	-	-	-
15.	1300	5	10	30	Osc	-	-	-
16.	1300	5	10	40	Osc	-	-	-

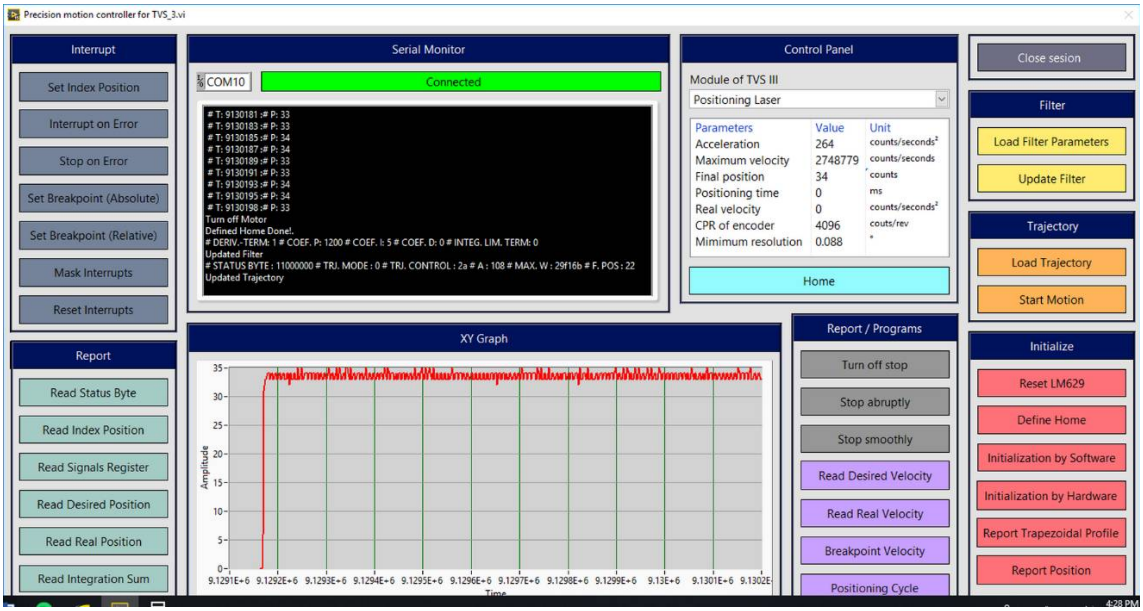

Figura 6.7. Prueba No. 4 para $\varphi_r = 3°$ con oscilaciones de ± 1 cuenta

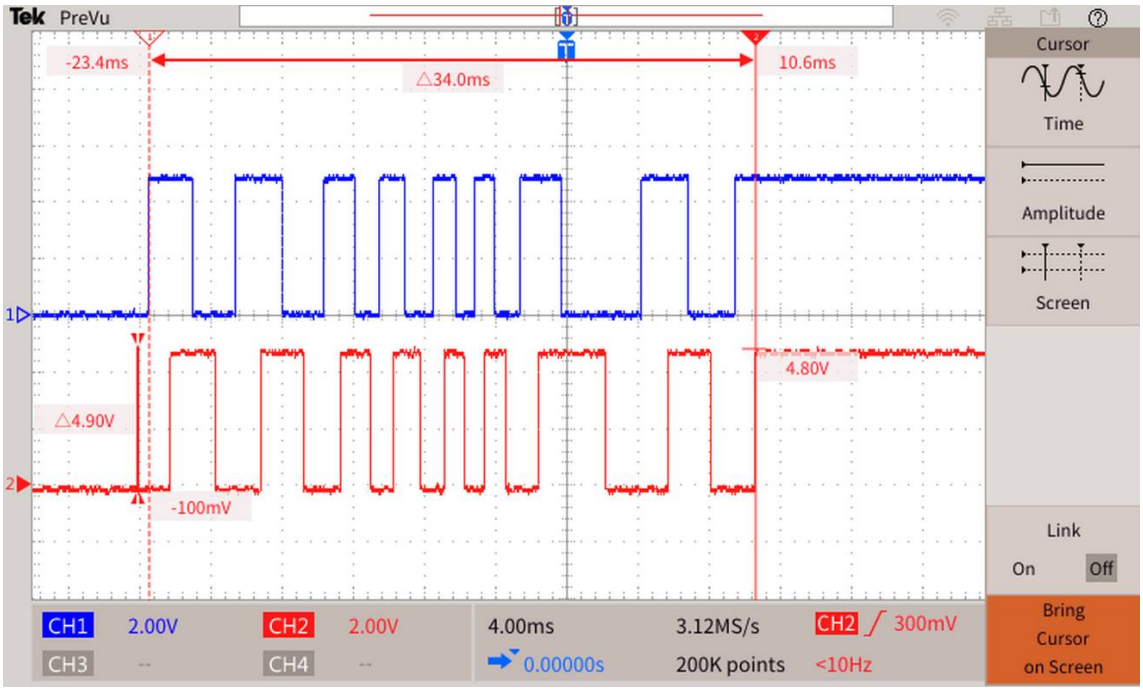

Figura 6.8. Prueba No. 1 para $\varphi_o \approx 3°$ obteniendo $N_m = 34$, $t_\varphi = 34\ ms$

Tabla 6.5. Resultados de la experimentación usando $\varphi_r = 5°$ y $N_r = 57$

No. Prueba	k_p	k_i	a_r	ω_{max}	N_m	$\varphi_o \approx$	φ_e	t_φ
1.	1200	10	15	30	57	5°	0%	438 ms
2.	1200	10	15	40	56	4.92°	1.6%	49.6 ms
3.	1200	10	10	30	57	5°	0%	95.2 ms
4.	1200	10	10	40	Osc	-	-	-
5.	1200	5	15	30	57	5°	0%	62.4 ms
6.	1200	5	15	40	56	4.92°	1.6%	45.6 ms
7.	1200	5	10	30	57	5°	0%	70 ms
8.	1200	5	10	40	57	5°	0%	62.1 ms
9.	1300	10	15	30	Osc	-	-	-
10.	1300	10	15	40	57	5°	0%	60 ms
11.	1300	10	10	30	Osc	-	-	-
12.	1300	10	10	40	Osc	-	-	-
13.	1300	5	15	30	Osc	-	-	-
14.	1300	5	15	40	Osc	-	-	-
15.	1300	5	10	30	Osc	-	-	-
16.	1300	5	10	40	Osc	-	-	-

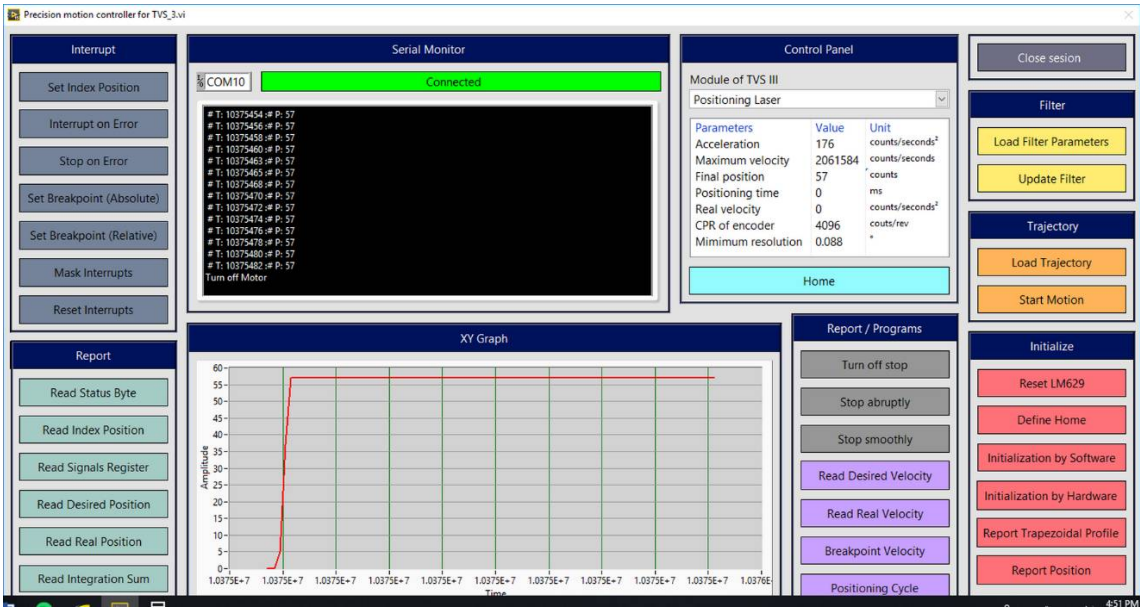

Figura 6.9. Prueba No. 1 para $\varphi_r = 5°$ sin oscilaciones

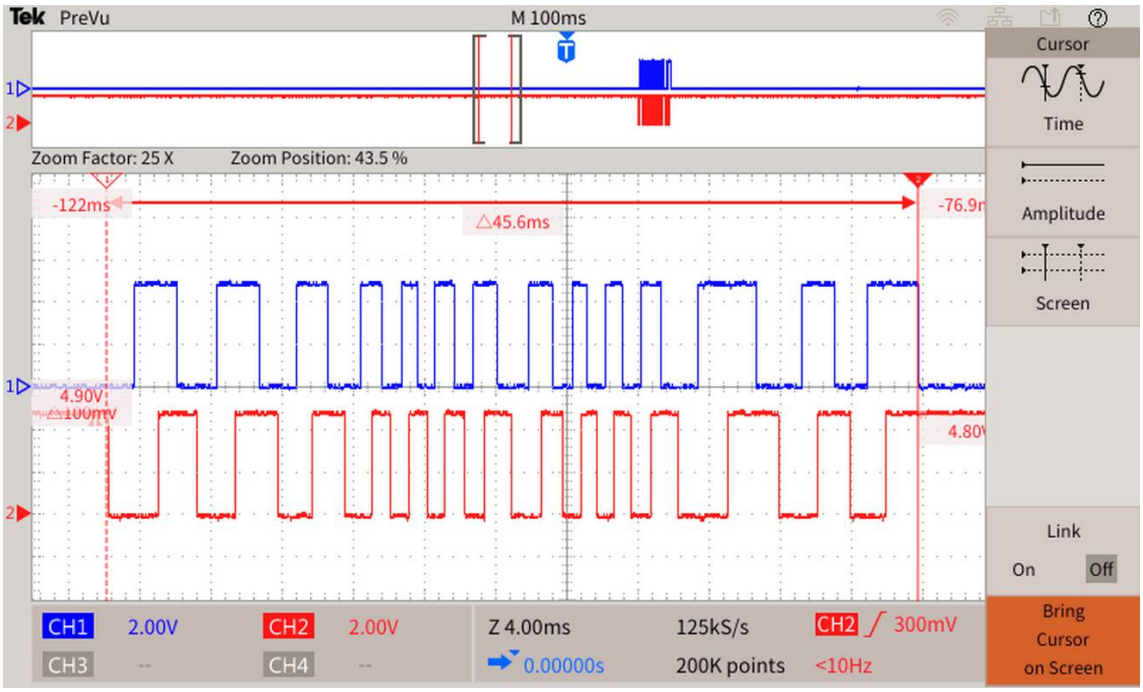

Figura 6.10. Prueba No. 6 para $\varphi_o \approx 5°$ obteniendo $N_m = 56, t_\varphi = 45.6\ ms$

Tabla 6.6. Resultados de la experimentación usando $\varphi_r = 15°$ y $N_r = 171$

No. Prueba	k_p	k_i	a_r	ω_{max}	N_m	$\varphi_o \approx$	φ_e	t_φ
1.	1050	10	15	30	171	15°	0%	1 s
2.	1050	10	15	40	171	15°	0%	191 ms
3.	1050	10	10	30	171	15°	0%	616 ms
4.	1050	10	10	40	171	15°	0%	385 ms
5.	1050	5	15	30	170	14.94°	0.4%	300 ms
6.	1050	5	15	40	171	15°	0%	118 ms
7.	1050	5	10	30	Osc	-	-	-
8.	1050	5	10	40	171	15°	0%	174 ms
9.	1000	10	15	30	170	14.94°	0.4%	92.6 ms
10.	1000	10	15	40	Osc	-	-	-
11.	1000	10	10	30	170	14.94°	0.4%	114 ms
12.	1000	10	10	40	Osc	-	-	-
13.	1000	5	15	30	Osc	-	-	-
14.	1000	5	15	40	171	15°	0%	169 ms
15.	1000	5	10	30	Osc	-	-	-
16.	1000	5	10	40	170	14.94°	0.4%	112 ms

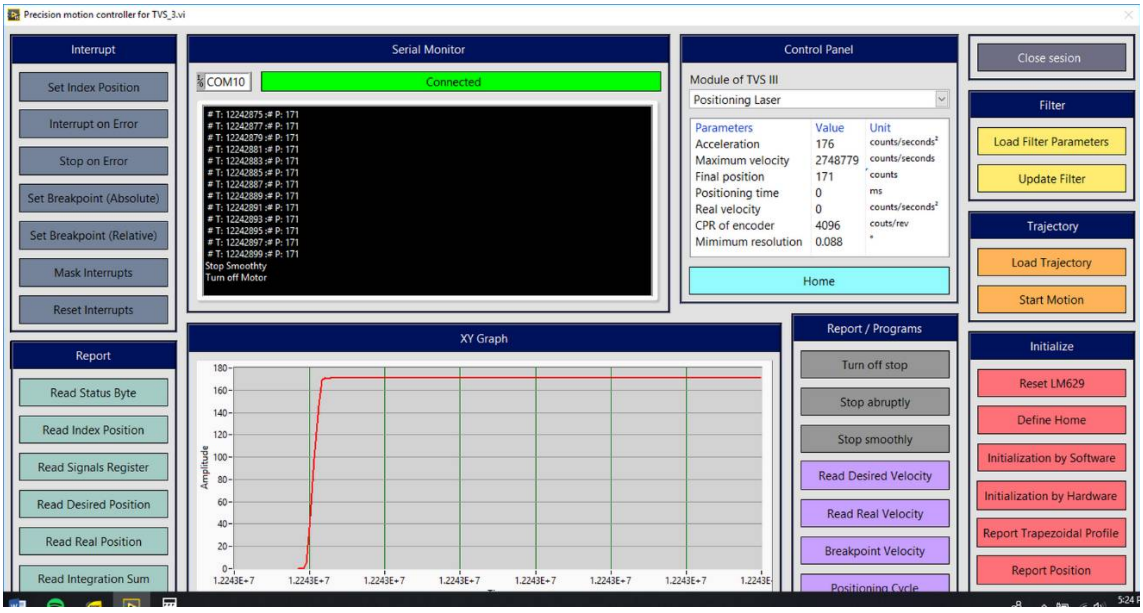

Figura 6.11. Prueba No. 2 para $\varphi_r = 15°$ sin oscilaciones

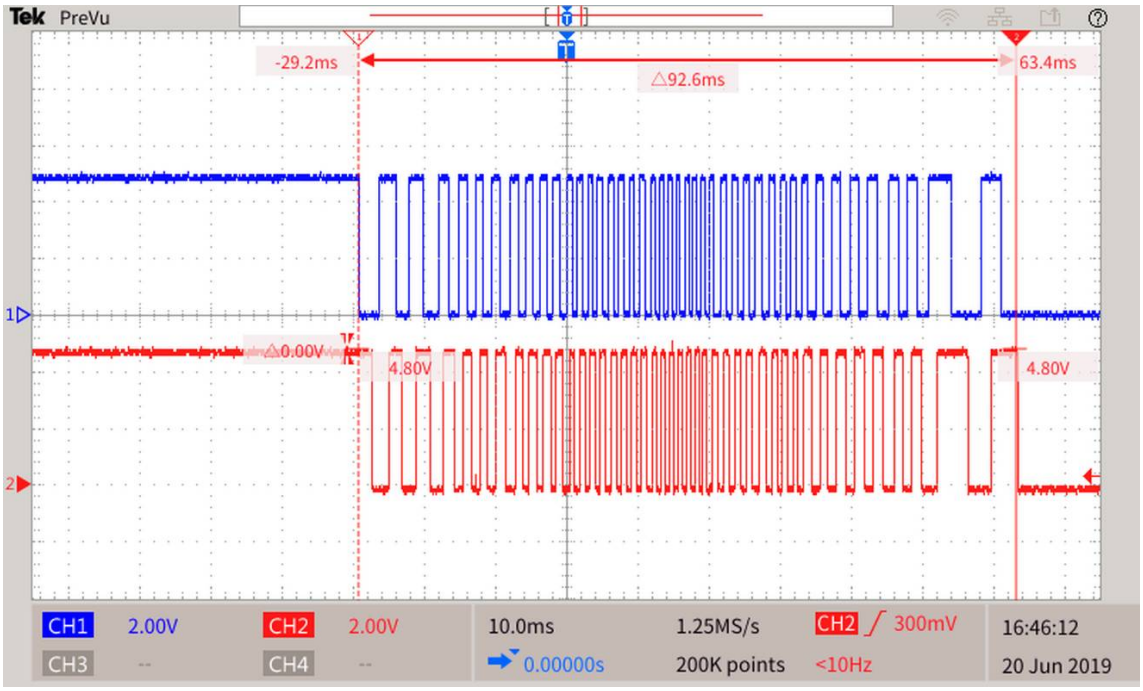

Figura 6.12. Prueba No. 9 para $\varphi_o \approx 15°$ obteniendo $N_m = 170$, $t_\varphi = 92.6\ ms$

Tabla 6.7. Resultados de la experimentación usando $\varphi_r = 30°$ y $N_r = 341$

No. Prueba	k_p	k_i	a_r	ω_{max}	N_m	$\varphi_o \approx$	φ_e	t_φ
1.	1500	250	30	60	Osc	-	-	-
2.	1500	250	30	40	341	29.97°	0.1%	111 ms
3.	1500	250	15	60	Osc	-	-	-
4.	1500	250	15	40	Osc	-	-	-
5.	1500	50	30	60	341	29.97°	0.1%	115 ms
6.	1500	50	30	40	341	29.97°	0.1%	125 ms
7.	1500	50	15	60	Osc	-	-	-
8.	1500	50	15	40	Osc	-	-	-
9.	1000	250	30	60	341	29.97°	0.1%	132 ms
10.	1000	250	30	40	341	29.97	0.1%	173 ms
11.	1000	250	15	60	340	29.88°	0.4%	134 ms
12.	1000	250	15	40	340	29.88°	0.4%	135 ms
13.	1000	50	30	60	Osc	-	-	-
14.	1000	50	30	40	Osc	-	-	-
15.	1000	50	15	60	341	29.97°	0.1%	570 ms
16.	1000	50	15	40	Osc	-	-	-

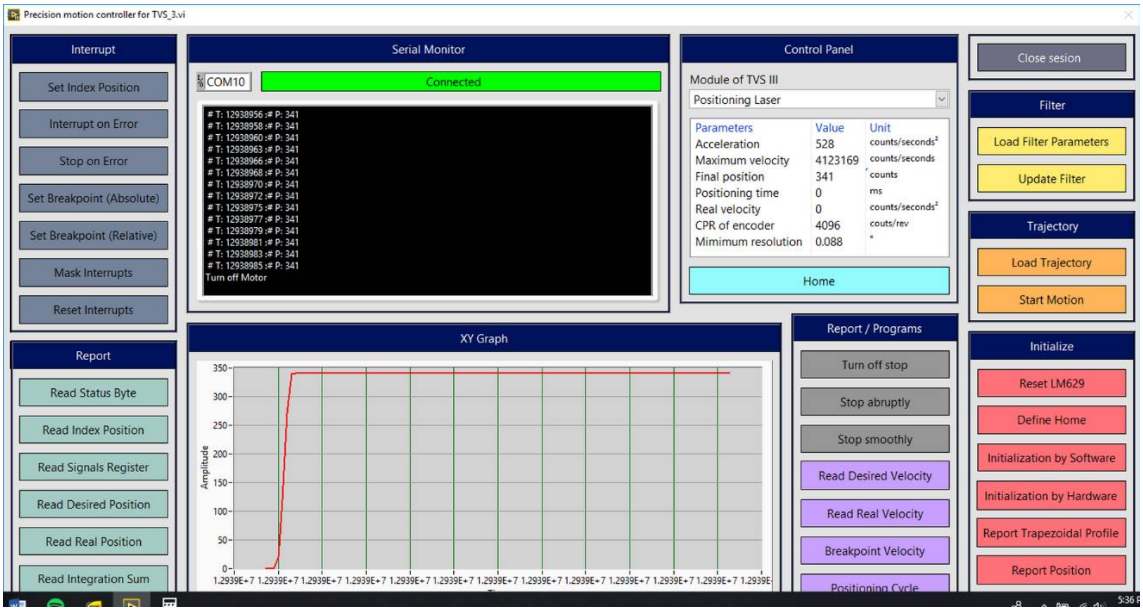

Figura 6.13. Prueba No. 2 para $\varphi_r = 30°$ sin osiclaciones

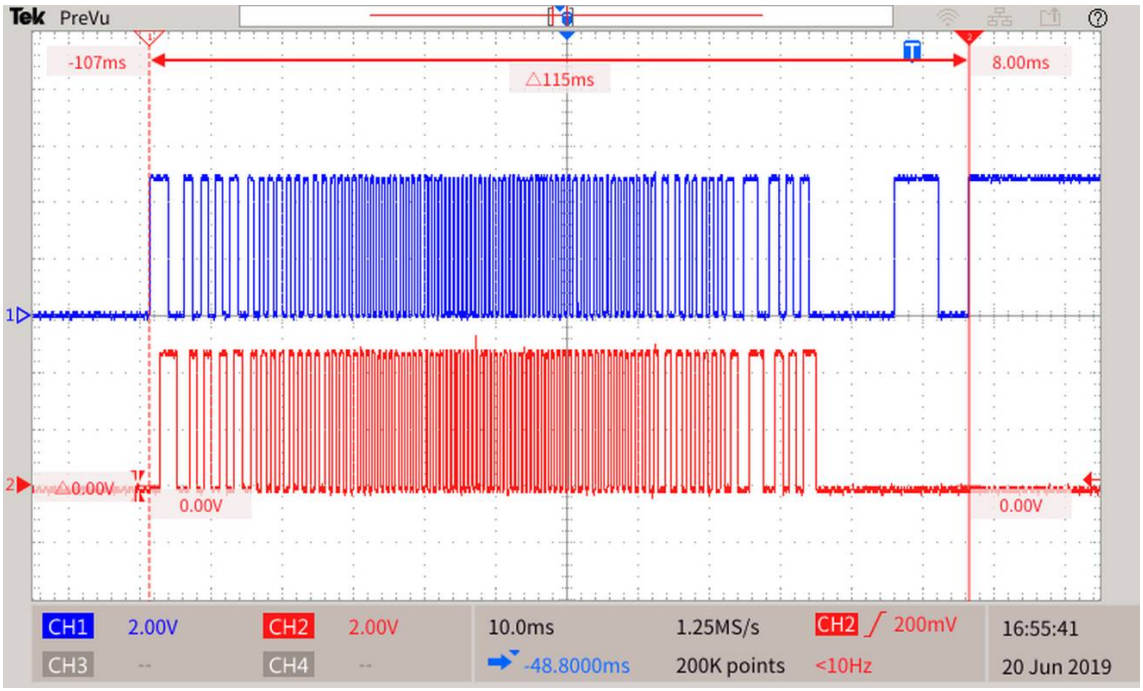

Figura 6.14. Prueba No. 5 para $\varphi_o \approx 30°$ obteniendo $N_m = 341$, $t_\varphi = 115\ ms$

Tabla 6.8. Resultados de la experimentación usando $\varphi_r = 45°$ y $N_r = 512$

No. Prueba	k_p	k_i	a_r	ω_{max}	N_m	$\varphi_o \approx$	φ_e	t_φ
1.	1200	100	30	60	512	45°	0%	161 ms
2.	1200	100	30	40	512	45°	0%	139 ms
3.	1200	100	15	60	512	45°	0%	214 ms
4.	1200	100	15	40	512	45°	0%	135 ms
5.	1200	50	30	60	Osc	-	-	-
6.	1200	50	30	40	512	45°	0%	138 ms
7.	1200	50	15	60	Osc	-	-	-
8.	1200	50	15	40	512	45°	0%	247 ms
9.	1000	100	30	60	512	45°	0%	159 ms
10.	1000	100	30	40	512	45°	0%	136 ms
11.	1000	100	15	60	512	45°	0%	209 ms
12.	1000	100	15	40	511	44.91°	0.2%	166ms
13.	1000	50	30	60	Osc	-	-	-
14.	1000	50	30	40	Osc	-	-	-
15.	1000	50	15	60	Osc	-	-	-
16.	1000	50	15	40	512	45°	0%	310 ms

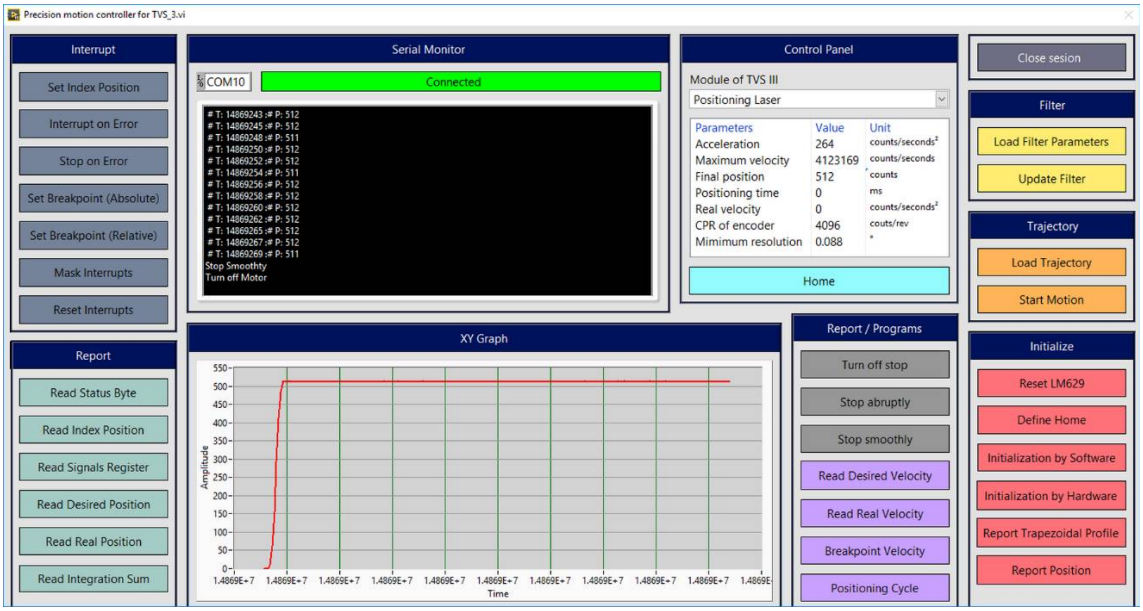

Figura 6.15. Prueba No. 14 para $\varphi_r = 45°$ con oscilaciones en zona estacionaria

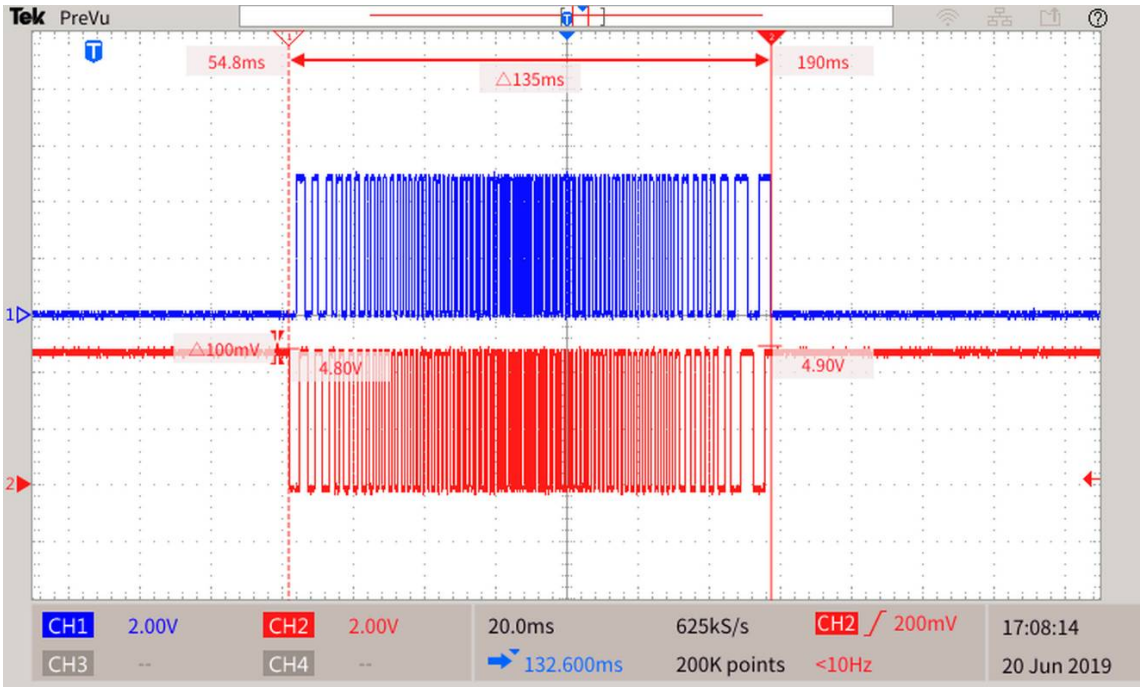

Figura 6.16. Prueba No. 4 para $\varphi_o \approx 45°$ obteniendo $N_m = 512$, $t_\varphi = 135\ ms$

Tabla 6.9. Resultados de la experimentación usando $\varphi_r = 90°$ y $N_r = 1024$

No. Prueba	k_p	k_i	a_r	ω_{max}	N_m	$\varphi_o \approx$	φ_e	t_φ
1.	1100	250	30	60	Osc	-	-	-
2.	1100	250	30	40	Osc	-	-	-
3.	1100	250	15	60	Osc	-	-	-
4.	1100	250	15	40	1024	90°	0%	464 ms
5.	1100	100	30	60	1024	90°	0%	241 ms
6.	1100	100	30	40	1024	90°	0%	1s
7.	1100	100	15	60	1023	89.91°	0.1%	243 ms
8.	1100	100	15	40	1024	90°	0%	2.17 s
9.	1000	250	30	60	1024	90°	0%	547 ms
10.	1000	250	30	40	1023	90°	0%	170 ms
11.	1000	250	15	60	Osc	-	-	-
12.	1000	250	15	40	1023	89.91°	0.1%	244 ms
13.	1000	100	30	60	1024	90°	0%	360 ms
14.	1000	100	30	40	Osc	-		-
15.	1000	100	15	60	1023	89.91°	0.1%	248 ms
16.	1000	100	15	40	Osc	-	-	-

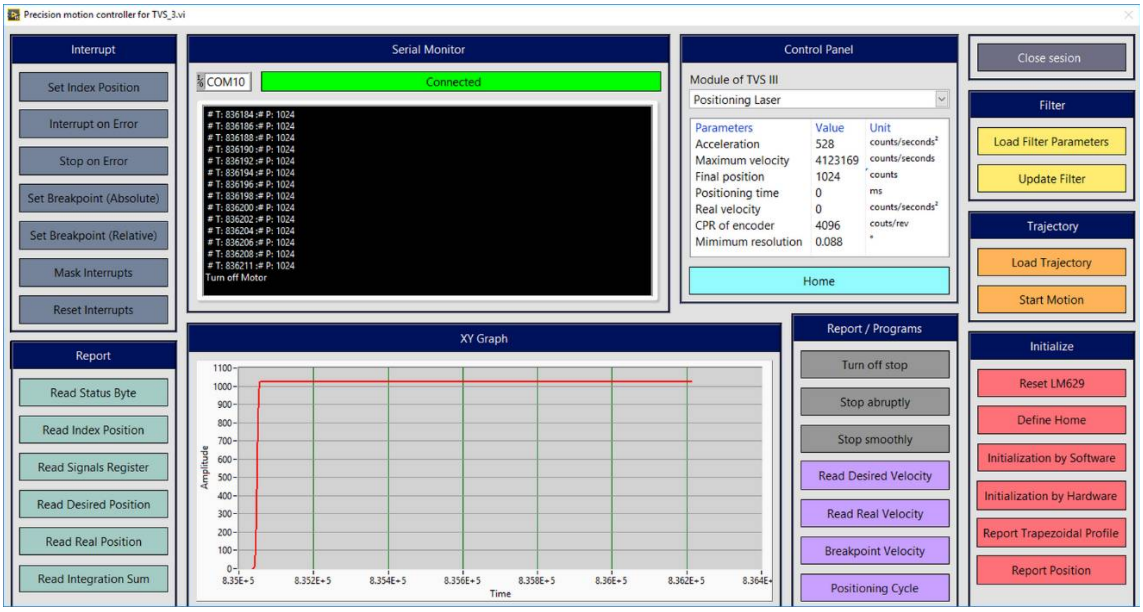

Figura 6.17. Prueba No. 4 para $\varphi_r = 90°$ sin oscilaciones en zona estacionaria

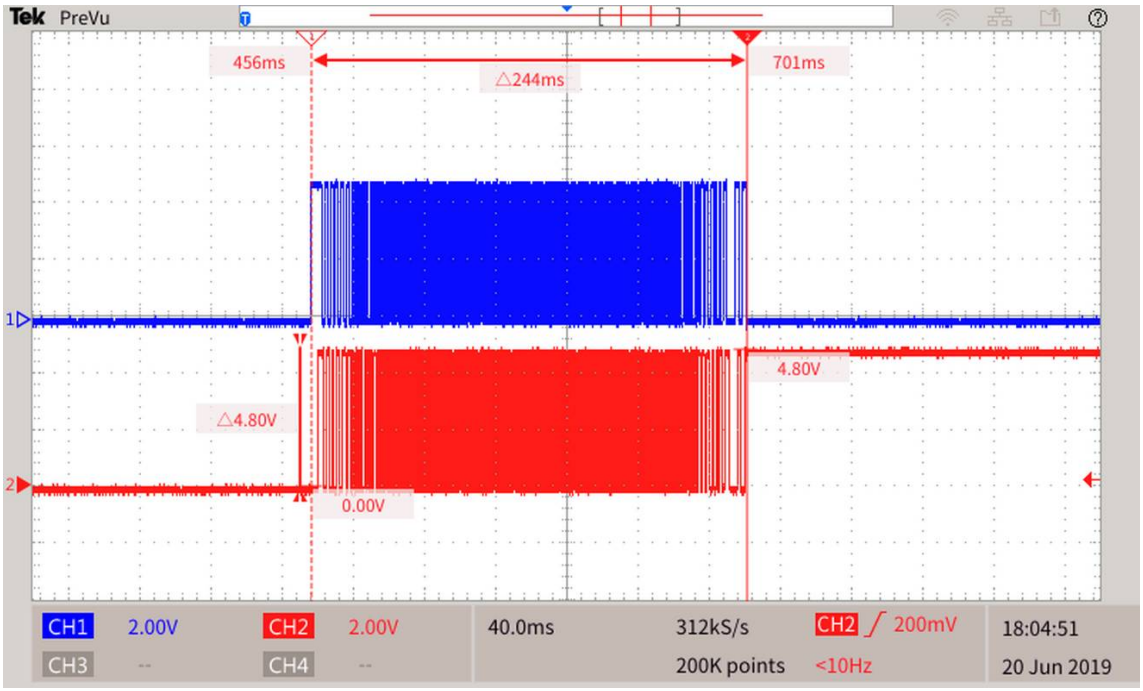

Figura 6.18. Prueba No. 12 para $\varphi_o \approx 90°$ obteniendo $N_m = 1023$, $t_\varphi = 244\ ms$

Tabla 6.10. Resultados de la experimentación usando $\varphi_r = 180°$ y $N_r = 2048$

No. Prueba	k_p	k_i	a_r	ω_{max}	N_m	$\varphi_o \approx$	φ_e	t_φ
1.	1200	250	30	60	2048	180°	0%	283 ms
2.	1200	250	30	40	Osc	-	-	-
3.	1200	250	15	60	Osc	-	-	-
4.	1200	250	15	40	2048	180°	0%	391ms
5.	1200	100	30	60	2048	180°	0%	1.12s
6.	1200	100	30	40	2048	180°	0%	391 ms
7.	1200	100	15	60	Osc	-	-	-
8.	1200	100	15	40	2048	180°	0%	402 ms
9.	1000	250	30	60	Osc	-	-	-
10.	1000	250	30	40	Osc	-	-	-
11.	1000	250	15	60	2048	180°	0%	1 s
12.	1000	250	15	40	Osc	-	-	-
13.	1000	100	30	60	Osc	-	-	-
14.	1000	100	30	40	Osc	-	-	-
15.	1000	100	15	60	Osc	-	-	-
16.	1000	100	15	40	2047	179.91°	0.05%	360 ms

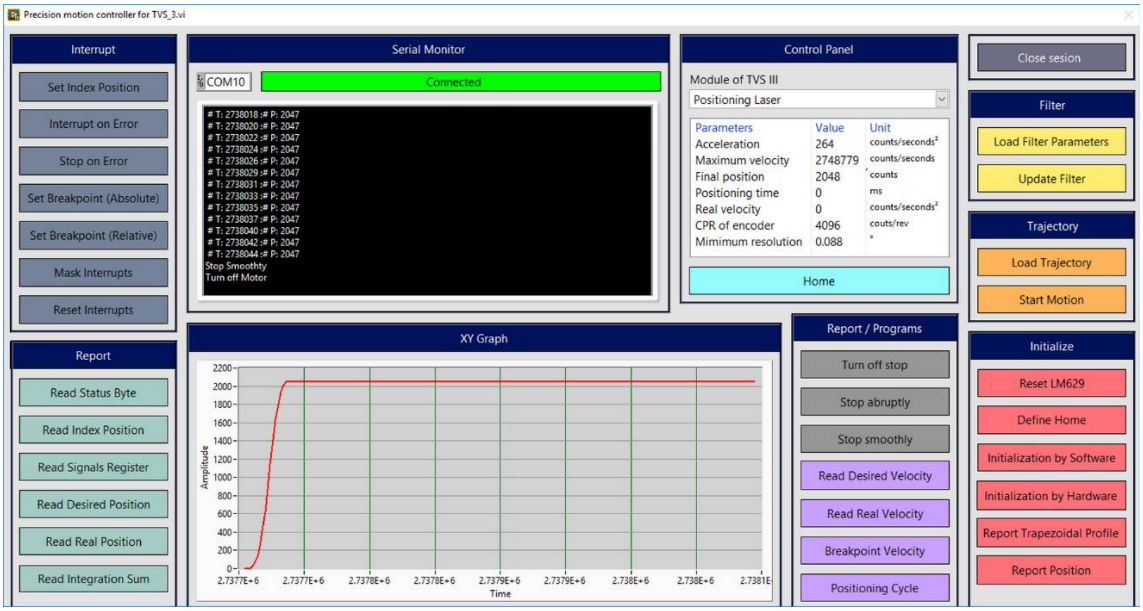

Figura 6.19. Prueba No. 16 para $\varphi_r = 180°$ sin oscilaciones en zona estacionaria

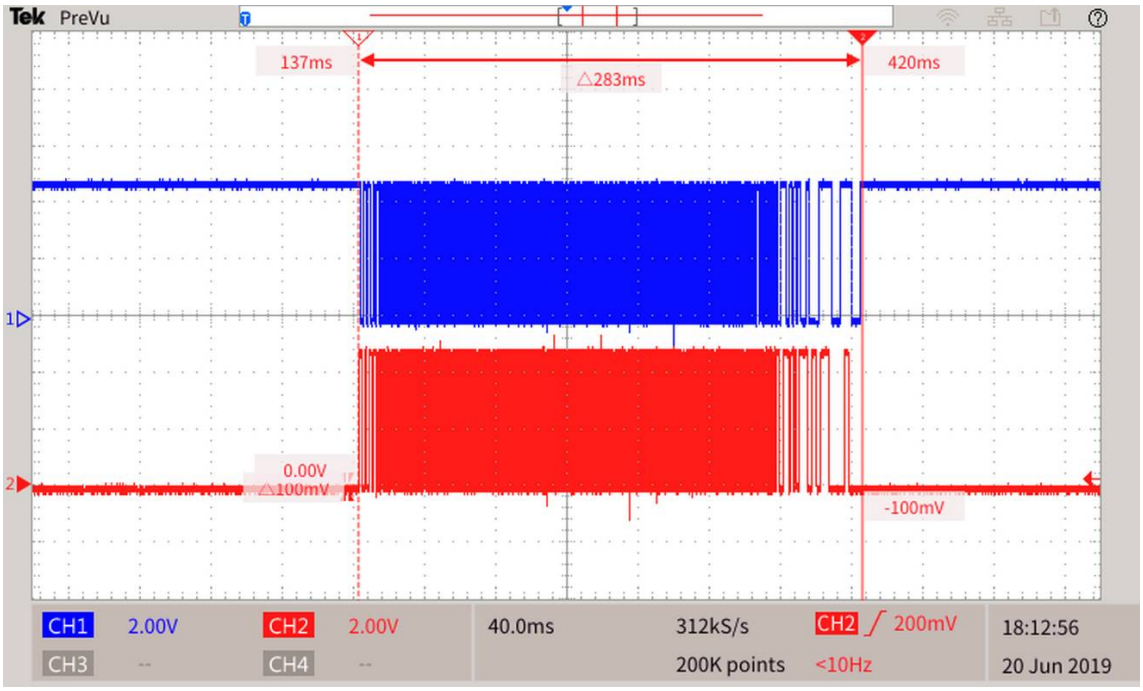

Figura 6.20. Prueba No. 1 para $\varphi_o \approx 180°$ obteniendo $N_m = 2048$, $t_\varphi = 283\ ms$

Tabla 6.11. Resultados de la experimentación usando $\varphi_r = 360°$ y $N_r = 4096$

No. Prueba	k_p	k_i	a_r	ω_{max}	N_m	$\varphi_o \approx$	φ_e	t_φ
1.	1200	15	30	60	4096	360°	0%	371 ms
2.	1200	15	30	40	4096	360°	0%	387 ms
3.	1200	15	15	60	4096	360°	0%	547 ms
4.	1200	15	15	40	Osc	-	-	-
5.	1200	10	30	60	Osc	-	-	-
6.	1200	10	30	40	4096	360°	0%	2s
7.	1200	10	15	60	Osc	-	-	-
8.	1200	10	15	40	4096	360°	0%	575 ms
9.	1000	15	30	60	Osc	-	-	-
10.	1000	15	30	40	4096	360°	0%	367 ms
11.	1000	15	15	60	4096	360°	0%	703 ms
12.	1000	15	15	40	Osc	-	-	-
13.	1000	10	30	60	4096	360°	0%	360 ms
14.	1000	10	30	40	4095	359.91°	0.02%	355 ms
15.	1000	10	15	60	Osc	-	-	-
16.	1000	10	15	40	4096	360°	0%	508 ms

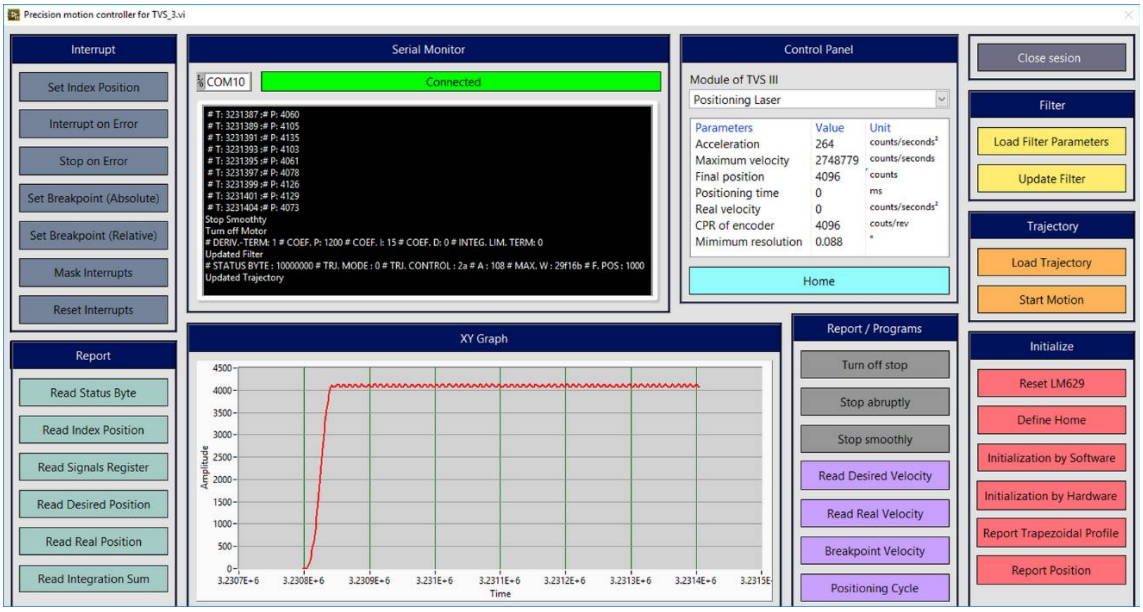

Figura 6.21. Prueba No. 16 para $\varphi_r = 360°$ con oscilaciones en zona estacionaria

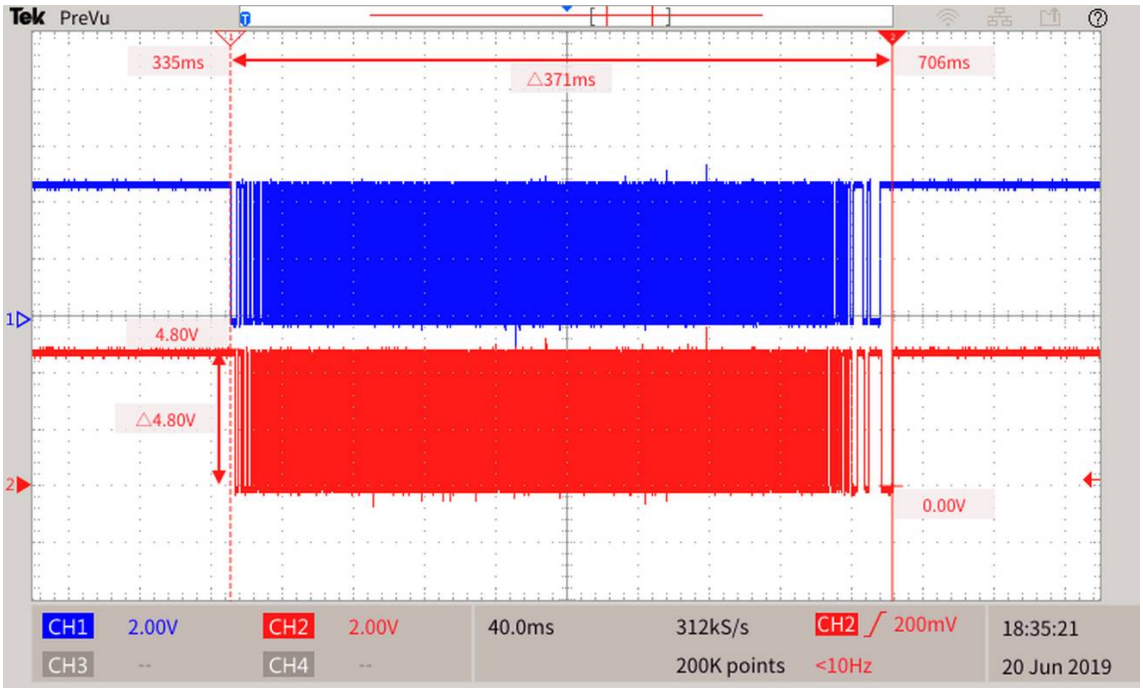

Figura 6.22. Prueba No. 1 para $\varphi_o \approx 360°$ obteniendo $N_m = 4096$, $t_\varphi = 371\ ms$

Conforme a las Tablas de arriba usando 16 pruebas para cada posición angular descrita fue posible obtener los valores para $\bar{\varphi}'_e, \bar{t}'_\varphi$ y $\bar{\varphi}'_o$ definidos en el capítulo 5 usando una resolución mínima como $\rho_{min} = 0.088°$. A continuación se presenta la Tabla 6.12 para describir los

valores obtenidos para la propuesta de esta tesis. Tomando como una reducción de error de posicionamiento promediado menor a 12 por ciento y un tiempo de posicionamiento promediado menor a 500 ms.

Tabla 6.12. Respuestas experimentales de los factores $\bar{\varphi}'_e, \bar{t}'_\varphi$ y $\bar{\varphi}'_o$

Motor Maxon DCX22S	**1°**	**2°**	**3°**	**5°**	**15°**	**30°**	**45°**	**90°**	**180°**	**360°**	**Unidad**
$\bar{\varphi}'_e$	10.4	3.30	1.11	0.4	0.14	0.17	0.01	0.03	0	0.02	*%*
$\bar{t}'_\varphi$	104	42	154	110	227	186	183	314	365	463	*ms*
$\bar{\varphi}'_o$	0.89	1.93	2.96	4.98	14.97	29.94	44.99	89.97	179.98	359.99	*deg*

7. Comparación entre el Controlador LM629N-8 y el Controlador Maxon EPOS 24/1

Este capítulo, presenta los trabajos hechos de una comparación realizada entre los controladores de movimiento siguientes: el controlador de posicionamiento EPOS 24/1 y el controlador LM629N-8. Tomando en consideración, el controlador EPOS 24/1 es industrial y cuenta con una interfaz propia con herramientas efectivas para maximizar los recursos del mismo controlador. Por otra parte, el controlador LM629N-8 es de bajo costo y con una capacidad eficiente para controlar servomotores que requieren movimientos precisos. En consecuencia, la sección de Respuestas de los Controlador Involucrados contiene los resultados de una serie de pruebas obteniendo datos del tiempo de posicionamiento promedio $\bar{t}'_{\varphi}$ para desplazamientos angulares de referencia en $\varphi_r(t) =$ (1°, 5°, 15°, 27°,360°) y comparándolos con los datos promedio del tiempo de posicionamiento conforme a los mismos ángulos de referencia usando el controlador LM629N-8.

7.1. Respuestas de los Controladores Involucrados

Usando la interfaz EPOS Studio diseñada por Maxon Motors [16] para operar el controlador de posicionamiento EPOS 24/1 creando una configuración que involucra los siguientes datos: modo de comunicación serial entre la computadora y el controlador EPOS 24/1, parámetros del motor Maxon RE 25, y fueron: la velocidad nominal, voltaje nominal, resolución en pulsos por revolución del Enconder Incremental HEDL-5540-C02 y el tipo del motor EC, de igual forma presenta una herramienta para hacer una auto sintonización del controlador PID interno del controlador. Un recurso a destacar es el auto sintonización realizada por EPOS Studio, debido a contar con un algoritmo adaptativo para el tipo de motor aplicado y obtener un alto grado de desempeño. Una vez, creado la configuración pertinente para el motor Maxon RE 25 con los siguientes datos solicitados por la interfaz: tipo motor voltaje de armadura nominal u_A=12 v_{cc}, pulsos por revolución *100 ppr*, constante de tiempo térmico de bobinado *12.5 s*, máxima velocidad permisible en *8,330 rpm*, corriente eléctrica nominal *1.16A* el tipo correcto de sensor de posición en relación al número de canales, usando los canales A, B e Índice. Figura 7.1, muestra los tipos de controladores disponibles con diferentes configuraciones físicas.

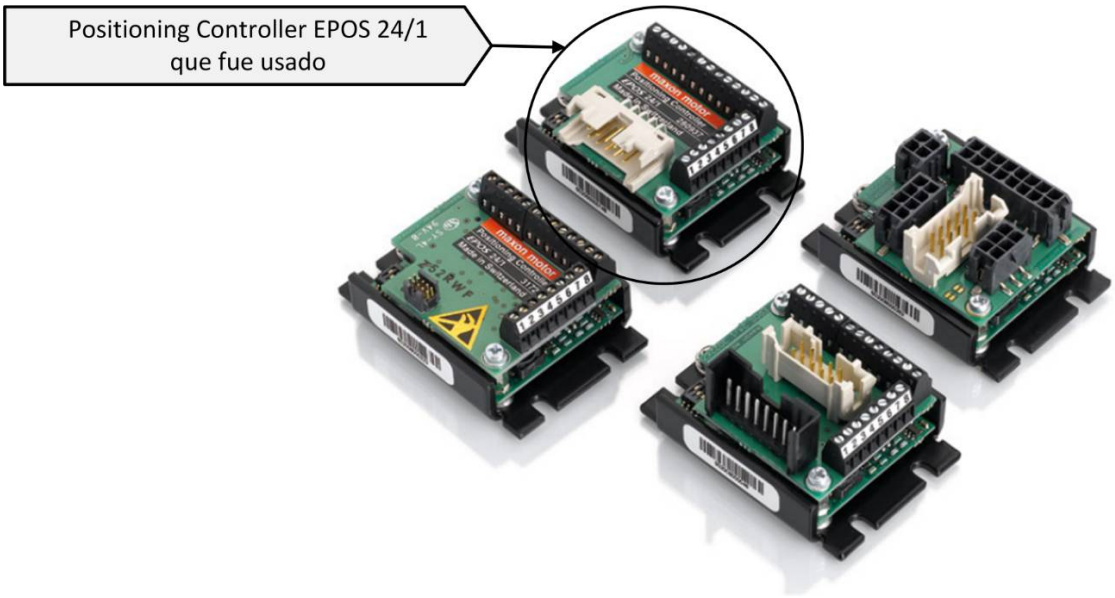

Figura 7.1. Controladores de posicionamiento Maxon EPOS 24/1

Figura 7.2 muestra la experimentación realizada para obtener el auto sintonización para el controlador PID del EPOS 24/1, a través de monitorear la respuesta de corriente y voltaje suministrado al motor Maxon RE 25, una vez logrado la auto sintonización, fueron realizadas las pruebas con la posición angular de referencia $\varphi_r(t) = (1°, 5°, 15°, 27°, 360°)$.

Figura 7.2. Controlador de posición EPOS 24/1 y ventana de auto sintonización

Figura 7.3 también muestra la experimentación realizada con el controlador propuesto en esta tesis, sin embargo para obtener la sintonización para el controlador PID interno del LM629N-8, fue conforme a un método empírico iniciando el con la ganancia proporcional. Posteriormente, con la ganancia integrativa dejando $k_p = 1{,}000$ unidades y k_i=100 unidades observando estabilidad para el SCM usando el motor Maxon RE 25, una vez logrado la

sintonización, fueron realizadas las pruebas para las posiciones angulares $\varphi_r(t)$= (1°, 5°, 15°, 27°,360°).

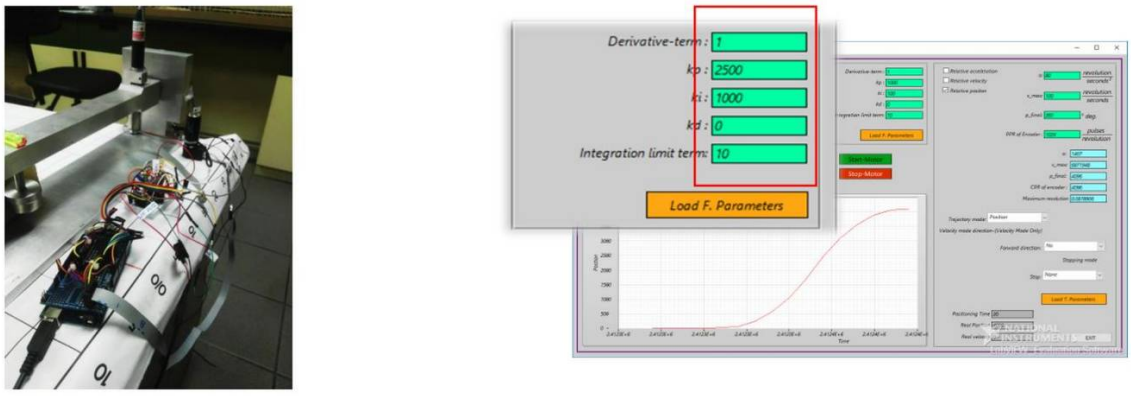

Figura 7.3. Controlador LM629N-8 y ventana del controlador PID de la GUI

Las respuestas de los controladores conforme al tiempo de posicionamiento t_φ estan en la Figura 7.4. Mostrando, dos gráficas donde el eje vertical es la posición angular final discreta N_m la cual representa el número de conteos de flancos altos y bajos leídos por el módulo interno «Decodificador de encoder incremental» el cual se alimenta de los canales A y B que suministran señales cuadráticas de salida a 100 *ppr* por el Encoder Incremental (EI) HEDL-5540-C02 acoplado al motor Maxon RE 25. El eje horizontal de las gráficas es el tiempo de posicionamiento para llegar a la posición angular final en N_m. Usando las posiciones angulares de referencia $\varphi_r(t)$ =(1°, 5°, 15°, 27°,360 °) es notable que para alcanzar la posición angular de referencia $\varphi_r(t)$ =1° el controlador EPOS 24/1 se lleva $t_\varphi = 14\ ms$, mientras que para el controlador LM629N-8, no fue definido el tiempo que toma para aproximarse a 1° solo fue posible obtener el tiempo del flanco bajo t_F = 624.7 μs, debido a no tener una correcta medición para validar t_φ desde un tiempo cero hasta presentarse un solo flanco el cual representa 0.9° por causa de $\rho_{min} = 0.9°$ conforme a la resolución del EI-HEDL-5540-C02. En consecuencia, para $\varphi_r(t) = 5°$ y $\varphi_r(t) = 15°$ el controlador LM629N-8 fue rápido para posicionar el eje del motor comparándolo con el EPOS 24/1 usando la variable tiempo de posicionamiento t_φ. Sin embargo, para $\varphi_r(t) = 27°$ y $\varphi_r(t) = 360°$ el EPOS 24/1 fue rápido en obtener el posicionamiento conforme a t_φ. De hecho, se puede notar que el controlador LM629N-8 presenta estabilidad en la zona estacionaria y en la zona transitoria reduciendo o evitando sobreoscilaciones. Conforme a lo anterior, el controlador LM629N-8 es capaz de manejar tiempos parecidos de posicionamiento como lo hacen los controladores de posición industriales, presentado estabilidad inclusive en su desempeño.

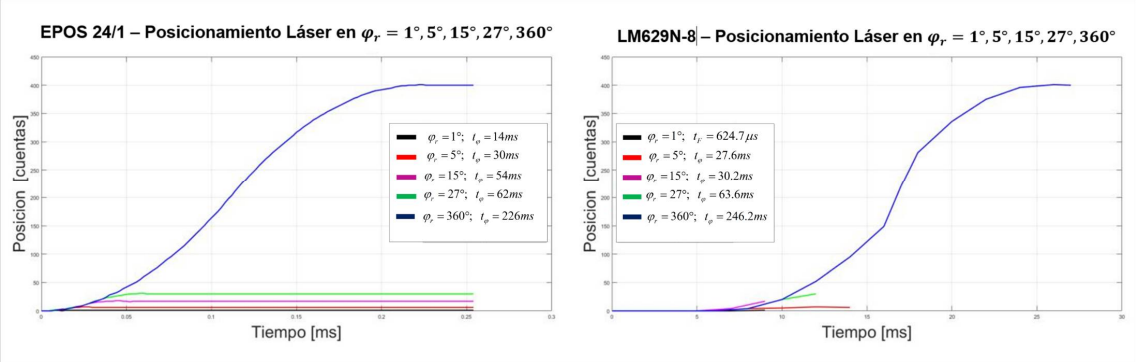

Figura 7.4. Comparación de los tiempos de posicionamiento obtenidos

Figura 7.5, muestra el tiempo de posicionamiento $t_\varphi = 66.3\,ms$ para un desplazamiento angular de 27° representado por $N_m = 24$ cuentas. Usando el motor Maxon RE 25 y el controlador LM629N-8.

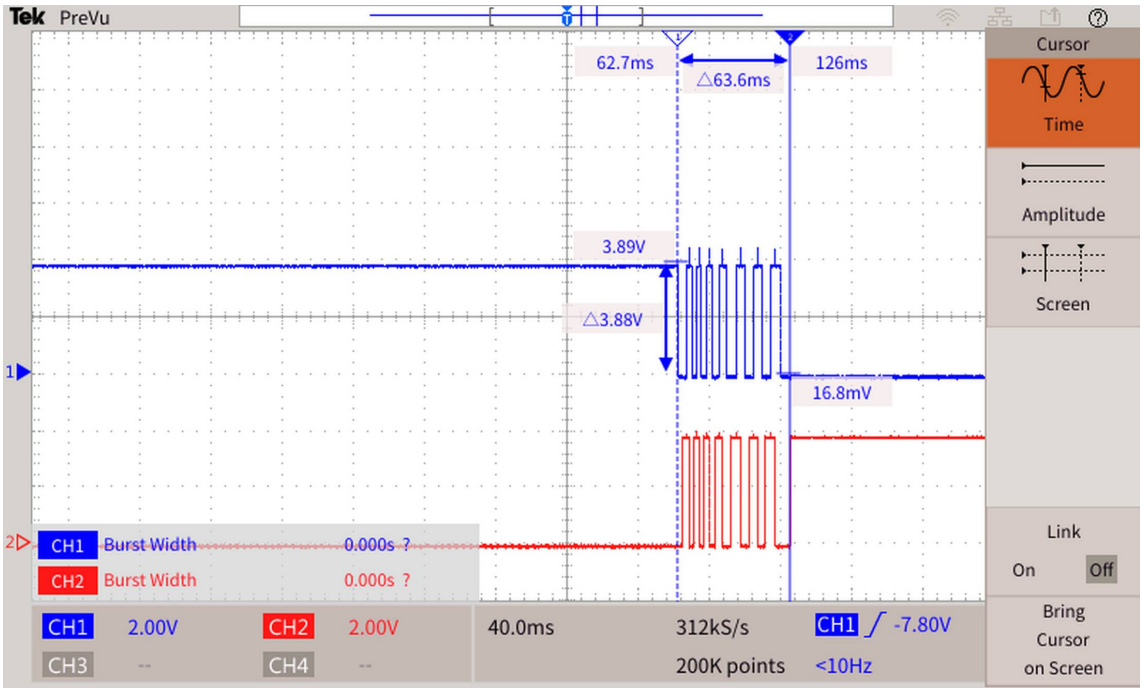

Figura 7.5. Tiempo de posicionamiento t_φ para $\varphi_r = 27°$ usando el LM629N-8

Conclusión

Los logros obtenidos son los siguientes: fue posible medir y obtener una reducción al error de posicionamiento del Posicionador Láser (PL) del Technical Vision System (TVS) prototipo No.3, usando un perfil de velocidad trapezoidal arbitrario y generado por el controlador digital LM629N-8. El actuador del PL fue el motor Maxon DCX22S durante todas las pruebas experimentales. Los resultados de la disminución del error, fueron obtenidos por diseño de experimentación usando metodología de Taguchi. Para lograr este objetivo general, a continuación se presentan los resultados específicos obtenidos:

1. El método teórico propuesto define la trayectoria de posicionamiento angular con una precisión de ± 1 cuenta como la minima resolución del encoder usado, sin presentar inestabilidad en comparación a la respuesta obtenida por el algoritmo proporcional ejecutado en Arduino Uno de investigaciones previas. Este método teórico se implementó, usando el controlador digital LM629N-8 en lazo cerrado, ajustando parámetros que constituyen el perfil de la trayectoria de velocidad trapezoidal de referencia.

2. Modelado y simulado del funcionamiento ideal para el controlador digital LM629N-8 usando Matlab/Simulink conforme al método teórico. Permitió definir parámetros del método teórico obtenido, a la respuesta satisfactoria de la trayectoria de velocidad trapezoidal para el motor Maxon DCX22S. Obteniendo la variable de la posición final del eje del motor con error menor a 1.0%.

3. Código fuente documentado del programa informático para la Interfaz Gráfica de Usuario (GUI), el cual forma parte de la implementación del sistema de control propuesto para aplicar e indicar posiciones deseadas al eje del motor Maxon DCX22S, con el uso del controlador embebido LM629N-8. Con la posibilidad de modificarlo para futuros trabajos a diversas aplicaciones donde intervengan sistemas de control de movimiento.

4. Código fuente documentado del programa informático para la Interfaz Hardware (Gateway), actuando como intercomunicador entre la GUI y el controlador LM629N-8. Tomando las configuraciones codificadas de los perfiles de velocidad, parámetros del control PI entre otros, al control de movimiento por parte de la GUI. Este código al igual que el anterior permite ser modificado para trabajos futuros.

5. Diseño e implementación de una tarjeta electrónica de control en lazo cerrado de la integración de hardware definido al sistema de control propuesto a esta tesis, usando el controlador LM629N-8 para motores CC. Evitando falsos contactos frente al uso de placas de prueba (protoboards).

6. Diseño de experimentación usando la metodología de Taguchi, obteniendo factores significativos del desempeño de la implementación del sistema de control para el Posicionador Láser del TVS No.3, presentado los resultados favorables por el desarrollo de este trabajo de tesis.

Trabajos Futuros

Los trabajos hechos de este proyecto de posgrado, permiten seguir desarrollando recursos necesarios para la funcionalidad correcta del TVS prototipo No.3. La implementación hecha para el control de movimiento con resultados satisfactorios conforme a la estabilidad, exactitud, alta precisión, con tiempos de posicionamiento menores a 500 ms y la reducción del error relativo promediado de posicionamiento del rayo láser menor o igual al 12 por ciento ($\bar{\varphi}'_e \leq 12\ \%$) para el PL del TVS No.3, son ventajas para los trabajos futuros relacionados con el TVS prototipo No.3. Como ejemplo, la apertura de escaneo tiene áreas de oportunidad por mejorar para la obtención de la información del sensor detector de rayo láser reflejado a la superficie del objetivo en estudio dentro del campo de visión, obteniendo el ángulo de la apertura de escaneo como una variable. Además, la geometría de la triangulación dinámica para proveer coordenadas espaciales de cualquier objeto a estudio.

Bibliografía

[1] L. C. Basaca-Preciado, J. C. Rodriguez-Quinonez, O. Sergiyenko, V. V. Tyrsa, W. Hernandez, J. I. Nieto Hipolito y O. Starostenko, «3D Laser Scanning Vision System for Autonomous Robot Navigation,» de *Industrial Electronics (ISIE) International Symposium Bari*, 2010.

[2] W. Flores-Fuentes, M. Rivas-Lopez, D. Hernandez-Balbuena, O. Sergiyenko, J. C. Rodriguez-Quinonez, J. Rivera-Castillo, L. Lindner and L. C. Basaca-Preciado, "Applying Optoelectronic Devices Fusion in Machine Vision: Spatial Coordinate Measurement," in *Developing and Applying Optoelectronics in Machine Vision*, O. Sergiyenko and J. C. Rodriguez-Quinonez, Eds., Hershey, Pensilvania: IGI Global, 2016, p. 37.

[3] G. Fu, A. Menciassi y P. Dario, «Development of a low-cost active 3D triangulation laser scanner for indoor navigation of miniature mobile robots,» *Robotics and Autonomous Systems,* vol. 60, nº 10, pp. 1317-1326, October 2012.

[4] L. Lindner, O. Sergiyenko, M. Rivas-Lopez, B. Valdez-Salas, J. C. Rodriguez-Quinonez, D. Hernandez-Balbuena, W. Flores-Fuentes, V. Tyrsa, M. Medina, F. Murietta-Rico, P. Mercorelli, A. Gurko and V. Kartashov, "Machine vision system for UAV navigation," in *Electrical Systems for Aircraft, Railway, Ship Propulsion and Road Vehicles & International Transportation Electrification Conference (ESARS-ITEC), International Conference on*, Toulouse, 2016.

[5] M. Rivas-Lopez, O. Sergiyenko and V. Tyrsa, Machine Vision: Approaches and Limitations, InTech, 2008.

[6] L. Lindner, O. Sergiyenko, J. C. Rodriguez-Quinonez, M. Rivas-Lopez, D. Hernandez-Balbuena, W. Flores-Fuentes, F. N. Murrieta-Rico and V. Tyrsa, "Mobile robot vision system using continuous laser scanning for industrial application," *Industrial Robot,* vol. 43, no. 4, pp. 360-369, 2016.

[7] X. Li, H. Zhao, Y. Liu, H. Jiang y Y. Bian, «Laser scanning based three dimensional measurement of vegetation canopy structure,» *Optics and Lasers in Engineering,* vol. 54, pp. 152-158, 2014.

[8] W. Flores-Fuentes, M. Rivas-Lopez and O. Sergiyenko, "Energy Center Detection in Light Scanning Sensors for Structural Health Monitoring Accuracy Enhancement," *IEEE Sensors Journal,* vol. 17, no. 7, pp. 2355-2361, 06 March 2014.

[9] L. Lindner, O. Sergiyenko, V. Tyrsa and P. Mercorelli, "An approach for dynamic triangulation using servomotors," in *Industrial Electronics (ISIE), 2014 IEEE 23rd International Symposium on*, Istanbul, 2014.

[10] L. C. Basaca-Preciado, O. Sergiyenko, J. C. Rodriguez-Quinonez y M. Rivas-Lopez, «Optoelectronic 3D Laser Scanning Technical Vision System based on Dynamic Triangulation,» de *Photonics Conference (IPC) in Burlingame California*, 2012.

[11] L. Lindner, O. Sergiyenko, M. Rivas-Lopez, J. C. Rodriguez-Quinonez, D. Hernandez-Balbuena, W. Flores-Fuentes, V. Tyrsa, J. I. Nieto Hipolito, F. N. Muerrieta Rico and V. M. Kartashov, "Issues of exact laser ray positioning using DC motors for vision-based target detection," in *2016 IEEE 25th International Symposium on Industrial Electronics (ISIE)*, Santa Clara, 2016.

[12] O. Y. Sergiyenko, "Optoelectronic System for Mobile Robot Navigation," *Optoelectronics, Instrumentation and Data Processing,* vol. 46, no. 5, pp. 414-428, 2010.

[13] L. Lindner, O. Sergiyenko, M. Rivas-Lopez, M. Ivanov, J. Rodriguez-Quinonez, D. Hernandez-Balbuena, W. Flores-Fuentes, V. Tyrsa, F. N. Muerrieta-Rico and P. Mercorelli, "Machine vision system errors for unmanned aerial vehicle navigation," in *Industrial Electronics (ISIE), 2017 IEEE 26th International Symposium on*, Edinburgh, 2017.

[14] L. Lindner, O. Sergiyenko, J. Rodriguez-Quinonez, V. Tyrsa, P. Mercorelli, W. Fuentes-Flores, F. Murrieta-Rico and J. I. Nieto-Hipolito, "Continuous 3D scanning mode using servomotors instead of stepping motors in dynamic laser triangulation," in *Industrial Electronics (ISIE), 2015 IEEE 24th International Symposium on*, Buzios, 2015.

[15] L. Lindner, Tesis: Theoretical method to increase the speed of continuous mapping in a Three-Dimensional Laser Scanning System using servomotors control, Mexicali, 2017.

[16] "Maxon Academy," Maxon Motor, [Online]. Available: www.maxonmotor.com. [Accessed 09 November 2016].

[17] O. Sergiyenko, L. Burtseva, M. Bravo, I. G. Rendon y V. Tyrsa, «Scanning Vision System For Mobile Vehicle Navigation,» de *Electronics and Photonics Multiconferences in Guanajuato*, 2006.

[18] L. Lindner, O. Sergiyenko, J. Rodriguez-Quinonez y W. Flores-Fuentes, «Sustitución de motores de pasos con servomotores en triangulación dinámica de láser para escaneo continúo de coordenadas espaciales,» de *ARGOS 2014*, Mexicali, 2014.

[19] L. Lindner, "Laser Scanners," in *Developing and Applying Optoelectronics in Machine Vision*, O. Sergiyenko and J. C. Rodriguez-Quinonez, Eds., Hershey, Pensilvania: IGI Global, 2016, p. 38.

[20] L. Lindner, O. Sergiyenko, M. Rivas-Lopez, D. Hernandez-Balbuena, W. Flores-Fuentes, J. C. Rodriguez-Quinonez, F. N. Murrieta-Rico, M. Ivanov, V. Tyrsa and L. C. Basaca, "Exact laser beam positioning for measurement of vegetation vitality," *Industrial Robot: An International Journal,* vol. 44, no. 4, pp. 532-541, 2017.

[21] S. K. Singh, Kinematics Fundamentals, Houston, Texas: Connexions, 2008.

[22] F. Golnaraghi and B. C. Kuo, Automatic Control Systems, McGraw-Hill Education, 2017, p. 864.

[23] P. Hong-Seok y S. Chintal, «Development of High Speed and High Accuracy 3D Dental Intra Oral Scanner,» de *25th DAAAM International Symposium on Intelligent Manufacturing and Automation*, 2015.

[24] F. A. Ferreira, J. d. Vicente y Oliva and A. M. Sanches Perez, "Evaluation of the Performance of Coordinate Measuring Machines in the Industry, Using Calibrated Artefacts," in *The Manufacturing Engineering Society International Conference, MESIC 2013*, 2013.

[25] S. Gustavo, Q. M. Olga, M. Vicente y d. S. Fernando, «Seguimiento de trayectoria de robots moviles usando metodo de integracion trapezoidal,» de *XX Congreso Argentino de Control Automatico, AADECA 2006*, Buenos Aires, Argentina, 2006.

[26] T. Wescott, Applied Control Theory for Embedded Systems, Newnes, 2006, p. 320.

[27] «Presicion Motion Controller LM629,» Texas Instruments, 06 May 2013. [En línea]. Available: http://www.ti.com/product/LM629?jktype=recommendedresults. [Último acceso: 27 February 2019].

[28] S. J. Chapman, Electric Machinery Fundamentals, 4. Edition ed., M. H. Education, Ed., 2005.

[29] A. Abramovici and J. Chapsky, Feedback Control Systems, 1 ed., Springer US, 2000, p. 181.

[30] W. Bolton, Ingeniería de control, 1 ed., MARCOMBO, 2002.

[31] K. Ogata, Ingenieria de Control Moderna., 5 ed., Pearson, 2010, p. 904.

[32] E. Hendricks, O. Jannerup and P. H. Sørensen, Linear Systems Control, 1 ed., Springer-Verlag Berlin Heidelberg, 2008, p. 555.

[33] C. P. Tello, «Metodología Taguchi».

[34] R. K. Roy, Design of Experiments using the Taguchi Approach: 16 Steps to Product and Process Improvement, 2001.

[35] L. C. Basaca-Preciado, J. C. Rodriguez-Quinonez, O. Sergiyenko, V. V. Tyrsa, W. Hernandez, J. I. Nieto Hipolito y O. Starostenko, «Resolution improvement of Dynamic Triangulation method for 3D Vision System in Robot Navigation Task,» de *IECON 2010 – 36th Annual Conference on IEEE in Glendale Arizona*, 2010.

Publicaciones

Miguel Reyes-García; Lars Lindner; Moisés Rivas-López; Mykhailo Ivanov; Julio C. Rodriguez-Quiñonez; Fabian N. Murrieta-Rico; Alexander Gurko; Voktor Melnyk. "***Reduction of Angular Position Error of a Machine Vision System using the Digital Controller LM629***" Published in: IECON 2018- 44th Annual Conference of the IEEE Industrial Electronics Society, Washington, DC, USA. 21-23 Oct. 2018, Date Added to IEEE Xplore: 31 December 2018, 3200-3205.

Miguel Reyes-García; Oleg Sergiyenko; Mykhailo Ivanov; Lars Lindner; Julio C. Rodriguez-Quiñonez; Daniel Hernandez-Balbuena; Wendy Flores-Fuentes; Vera Tyrsa; Luis O. Moreno-Ahedo; Fabian N. Murrieta-Rico. "***Defining the Final Angular Position of DC Motor shaft using a Trapezoidal Trajectory Profile***" Published in: ISIE 2019 the 28th International Symposium on Industrial Electronics, Vancouver, Canada. 12-14 Jun. 2019.

Anexos

A continuación, se presenta la subrutina principal y una librería usando 8 clases para obtener el Código Fuente (CF) de la propuesta de esta tesis. Logrando operar el controlador LM629N-8 usando la tarjeta Arduino Mega 2560 a través de la GUI desarrollada en la plataforma de LabVIEW. Este CF escrito en este apartado fue desarrollado usando la plataforma Eclipse C/C++ versión 2018-12 (4.10.0) para hacer funcional el TVS No. 3 usando sus tres actuadores.

```
#include <avr/interrupt.h>
#include <avr/io.h>
#include <COMserial.h>
#include <ControlLines.h>
#include <Filter.h>
#include <Initialize.h>
#include <Interrupt.h>
#include <Report.h>
#include <Trajectory.h>
#include <Programs.h>
#include <avr/wdt.h>

/*(CLK-8MHz for LM629)----------------------------------*/
#ifdef __AVR_ATmega2560__
const byte CLOCKOUT = 11;  // Mega 2560
#endif
/*(Estancias)-------------------------------------------*/
Initialize _Initialize;
Interrupt _Interrupt;
Filter _Filter;
Trajectory _Trajectory;
COMserial _COM;
Report _Report;
Programs _Programs;
/*-----------------------------------------------------*/
int data[20] = {'\0'};
int inData = 0x00;
volatile unsigned int count = 0;
//used to keep count of how many interrupts were fired
volatile unsigned int StateHI = 0x00;
String inputString = "";
int graphType = 0x00;
boolean stringComplete = false;

/*-----------------------------------------------------*/

/*-----------------------------------------------------*/
void setup()
{
  digitalWrite(_Initialize.tps, LOW);
```

```
  // enable power supply of PWM H driver XY-160D.
  /*(CLK-8MHz for LM629)--------------------------------*/
  pinMode (CLOCKOUT, OUTPUT);
  // set up Timer 1
  TCCR1A = bit (COM1A0);  // toggle OC1A on Compare Match
  TCCR1B = bit (WGM12) | bit (CS10);   // CTC, no prescaling
  OCR1A =  0; // output every cycle
  /*-------------------------------------------------------*/
  Serial.begin(9600);
  // reserve 200 bytes for the inputString:
  //inputString.reserve(300);
  Serial.println("Arduino was Reseted.");
  // Arduino Reset via Watchdog Reset - The Recommended Method
  Serial.println("Graphic of Position laser by default");
  /*-------------------------------------------------------*/
  //Setup Timer1/2/3/4/5/6 to fire every Interval time Timer_0
  //-Timer_1 -Timer_2 -Timer_3 -Timer_4 -Timer_5.
  /*(Timer_5 with Overflow Interrupt to 1 ms)--------------*/
  TCCR5A = 0x00; //Timer_5 Control Reg A: Wave Gen Mode normal
  TCCR5B = 0x00; //Disbale Timer_5 while we set it up.
  TCNT5  = 65519;//Reset Timer_5 Count to (number of tick) out of 65535.
  //Each 1.088ms real 1.143ms to 1.247ms
  TIFR5  = 0x00; //Timer_5 INT Flag Reg: Clear Timer Overflow Flag
  TIMSK5 = 0x01; //Timer_5 INT Reg: Timer2 Overflow Interrupt Enable
  TCCR5B = 0x05; //Timer_5 Control Reg B: Timer Prescaler set to 1024
  /*-------------------------------------------------------*/
  /*(Timer_4 with Overflow Interrupt to 10ms)--------------*/
  TCCR4A = 0x00; //Timer_4 Control Reg A: Wave Gen Mode normal
  TCCR4B = 0x00; //Disbale Timer_4 while we set it up.
  TCNT4  = 65519;//Reset Timer_4 Count to (number of tick) out of 65535.
  //Each 1.088ms real 1.143ms to 1.247ms
  TIFR4  = 0x00; //Timer_4 INT Flag Reg: Clear Timer Overflow Flag
  TIMSK4 = 0x01; //Timer_4 INT Reg: Timer2 Overflow Interrupt Enable
  TCCR4B = 0x05; //Timer_4 Control Reg B: Timer Prescaler set to 1024
  /*-------------------------------------------------------*/

}

void softwareResetArduino( uint8_t prescaller)
{
  // start watchdog with the provided prescaller
  wdt_enable( prescaller);
  // wait for the prescaller time to expire
  // without sending the reset signal by using
 // the wdt_reset() method
  while(1) {}
}

void loop()
{

  if (stringComplete)
  {
    int index = 0;
    char dataArray[50] = {'\0'};
```

```
    char *token = NULL;
    int stringLength = inputString.length();
    inputString.toCharArray(dataArray, 50);
    token = strtok(dataArray, ",");
    String dataBuffer[sizeof(dataArray)];
    while (token != NULL)
    {
      dataBuffer[index] = (token);
      token = strtok(NULL, ",");
      index++;
    }
    for (int i = 0 ; i < index; i++)
    {
      data[i] = _COM.StringToHex(dataBuffer[i]);
    }
//-------------------------------------------------------------
    switch (data[0])
    {
      /*Initialize-------------------*/
      case  0xFF: //Software Reset Arduino Mega 2560
      softwareResetArduino( WDTO_4S);
        break;
      case  0xFE: // Hardware Reset LM629
      _Initialize.HardwareReset(data[0]);
        break;
      case  0xFD: // Software Reset LM629
      _Initialize.SoftwareReset(data[0]);
        break;
      case  0xFC: // Define Home
      _Initialize.DefineHome();
        break;
      /*Interrupt-------------------*/
      case  0xFB: // Set Index Position
      _Interrupt.SetIndexPosition();
        break;
      case  0xFA: // Interrupt On Error
        _Interrupt.InterruptOnError(data[1], data[2]);
        break;
      case  0xF9: // Stop On Error
        _Interrupt.StopOnError(data[1], data[2]);

        break;
      case 0xF8: // Set Breakpoint Absolute
        //_Interrupt.SetBreakpointAbsolute();
        Serial.println("set Breakpoint Absolute Done!");
        break;
      case 0xF7: // Set Breakpoint Relative
        _Interrupt.SetBreakpointRelative(data[1],data[2],data[3],data[4]);
        break;
      case 0xF6: // Mask Interrupts
        _Interrupt.MaskInterrupts(data[0], data[1]);
        break;
      case 0xF5: // Reset Interrupts
        _Interrupt.ResetInterrupts(data[0]);
        break;
```

```
/*Report-------------------*/
 case 0xF4: // Read Status Byte
  inData = _Report.ReadStatusByte();
  Serial.println("Status Byte : ");
  Serial.println(inData,BIN);
  break;
case 0xF3: // Read Signals Register
  _Report.ReadSignalsRegister();
  break;
case 0xF2: // Read Index Position
  _Report.ReadIndexPosition();
  break;
case 0xF1: // Read Desired Position
  _Report.ReadDesiredPosition();
  break;
case 0xF0: // Read Real Position
  _Report.ReadRealPosition(data[0]);
  break;
case 0xEF: // Read Desired Velocity
  _Report.ReadDesiredVelocity();
  break;
case 0xEE: // Read Real Velocity
  _Report.ReadRealVelocity(data[0]);
  break;
case 0xED: // Read Integration Sum
  _Report.ReadIntegrationSum();
  break;
/*Filter------------------*/
case 0xEC: // Load Filter Parameters
  if (stringLength == 33)//Nodo 35 //Stand alone 33
  {
    _Filter.LoadFilterParameters(data[1], data[2], data[3],
  data[4], data[5], data[6], data[7],
  data[8], data[9], data[10]);
  }
  else
  {
    Serial.println("Incomplete commands or error by syntax");
    Serial.println(":: incorrect Length of the parameters is :");
    Serial.println(stringLength);
    Serial.println(" . The correct Length is 33.");
  }
  break;
case 0xEB: // Update Filter
  _Filter.UpdateFilter();
  break;
/*Trajectory--------------------------*/
case 0xEA: // Load Trajectory
  if (stringLength == 45)//Nodo 46 //Stand alone 45
  {
    _Trajectory.LoadTrajectory(data[1], data[2], data[3], data[4],
 data[5], data[6], data[7], data[8], data[9], data[10], data[11],
   data[12],data[13], data[14]);

  }
```

```
      else
      {
        Serial.println("Incomplete commands or error by syntax");
        Serial.println(":: incorrect Length of the parameters is :");
        Serial.println(stringLength);
        Serial.println(" . The correct Length is 45. ");
      }
      break;
    case 0xE9: // Start Motion
      _Trajectory.StartMotion(data[0]);
      break;
    /*Control Panel---------------*/
    case 0xE8: // Stop Motion
      _Trajectory.turnOffstop(data[0]);
      break;
    case 0xE7: // Initializacition
      _Initialize.HardwareInitialization();
      break;
    case 0xE6: // Initializacition
      _Initialize.SoftwareInitialization();
      break;
    case 0xE5: //Basic Velocity Mode Move with Breakpoints
     //- VelBreakP.
      _Programs.VelMovBreakp();
      break;
    case 0xE3: // Positioning Breakpoints
       _Programs.PositioningCycles(data[0], data[1],
       data[2], data[3], data[4], data[5], data[6],
            data[7], data[8], data[9], data[10], data[11],
            data[12], data[13], data[14], data[15], data[16]);
      break;
    case 0xE2: //Go Home position.
      _Programs.goHome(data[1], data[2]);
      break;
    case 0xE1: // Stop Smoothly
      _Trajectory.stopSmoothly(data[0]);
      break;
    case 0xE0: // Stop Abruptly
      _Trajectory.stopAbruptly(data[0]);
      break;
    case 0xDF: // Graph Positioning laser
      graphType = 0x00;
      Serial.println("Enable Graph of positioning laser");
      break;
    case 0xDE: // Graph Trapezoidal velocity
      graphType = 0x01;
      Serial.println("Enable Graph of trapezoidal velocity profile");
      break;
      default: // Default
      Serial.println("Incorrect Command: ");
      Serial.println(data[0]);
      Serial.println("Try again");
    }

inputString = "";
```

```
  stringComplete = false;
  }
}

void serialEvent() {
  while (Serial.available()) {
    char inChar = (char)Serial.read();
    inputString += inChar;
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}

ISR(TIMER5_OVF_vect)
{
  if(_Trajectory.movementState && graphType == 0x00)
  // start motion and graph position.
  {
  _Report.ReadRealPosition(0x00);
  }
  if(_Trajectory.movementState && graphType == 0x01)
  // start motion and graph trapezoidal velocity profile
  {
  _Report.ReadDesiredVelocity();
  }

  TCNT5 = 65519;//Reset Timer-for 1ms-->
  TIFR5 = 0x00;//Timer5 INT Flag Reg: Clear Timer Overflow Flag
};
//digitalWrite(buzzer, !digitalRead(buzzer));
  ISR(TIMER4_OVF_vect)
  {
  StateHI = _Interrupt.HostInterrupt();
  TCNT4 = 65519;//Reset Timer-for 1ms -->
  TIFR4 = 0x00;//Timer5 INT Flag Reg: Clear Timer Overflow Flag
  };
```

```
/*
Programs.h - library for Motion Control code using LM629 controller.
Created by Miguel Reyes, January 09, 2019.
Released into the public domain.
*/

#ifndef Programs_h
#define Programs_h
#include "Arduino.h"


class Programs
{
public:
    volatile boolean movementState = false;

       Programs(); // Constructor

    void LoopPhasingProgram(void);
       void VelMovBreakp(); //Basic Velocity Mode Move with Breakpoints -
VelBreaks.
       void goHome(int encoderHb_1, int encoderLb_1);
       void PositioningCycles(int command, int trjmode, int trjcontrol, int
afwHB_1,int afwLB_1, int aswHB_2, int aswLB_2, int vfwHB_3, int vfwLB_3, int
vswHB_4, int vswLB_4,int pfwHB_5,int pfwLB_5, int pswHB_6, int pswLB_6, int
HB_encoder, int LB_encoder);



private:


};
#endif
```

```
/*
Programs.cpp - library for Motion Control code using LM629 controller.
Created by Miguel Reyes, January 9, 2019.
Released into the public domain.
*/

#include "Arduino.h"
#include "Trajectory.h"
#include <ControlLines.h>
#include <Interrupt.h>
#include <Initialize.h>
#include <Filter.h>
#include <Report.h>
#include "Programs.h"


Programs::Programs( ) // Constructor
{

}
void Programs::LoopPhasingProgram(void)
{
}
void Programs::VelMovBreakp(void) //E5
{
       ControlLines C;
       Interrupt Inte;
       Initialize In;
       Filter F;
       Trajectory T;
       Report R;

       int indata = 0x00;
       int StateHI = 0x00;

       F.LoadFilterParameters(0x01, 0x0D, 0x09, 0xC4, 0x00, 0x32, 0x00, 0x00,
0x00, 0x64); //EC,01,0D,09,C4,00,32,00,00,00,64
    F.UpdateFilter();
       Inte. MaskInterrupts(0xE5, 0x40);
       Inte.SetBreakpointRelative(0x00, 0x00, 0x9C, 0x40); //40 000
       T.LoadTrajectory(0x18, 0x28, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x34,
0x6E, 0x00, 0x00, 0x00, 0x00);
       T.StartMotion(0xE5);
       T.LoadTrajectory(0x18, 0x0C, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0A, 0x3D,
0x71, 0x00, 0x00, 0x00, 0x00);

       while(StateHI == 0x00) // true
              {
              R.ReadRealPosition(0x00);
           StateHI = Inte.HostInterrupt();
              }
       StateHI = 0x00;
       indata = R.ReadStatusByte();
       Serial.println("Status Byte : " + String(indata,BIN));
```

```
      T.StartMotion(0xE5);
    Inte.ResetInterrupts(0xE5); // Reset interrupts
      Inte.SetBreakpointAbsolute(0x00, 0x01, 0x38, 0x80); //80 000

    while(StateHI == 0x00) // true
          {
             R.ReadRealPosition(0x00);
          StateHI = Inte.HostInterrupt();
          }
    StateHI = 0x00;
    indata = R.ReadStatusByte();
      Serial.println("Status Byte : " + String(indata,BIN));

      T.stopAbruptly(0xE5);
      Inte.ResetInterrupts(0xE5);

    indata = R.ReadStatusByte();
      Serial.println("Status Byte : " + String(indata,BIN));
}

//-------------------------------------
void Programs::goHome(int encoderHb_1, int encoderLb_1)
{
      Trajectory _T;
      Initialize _I;
      Interrupt _In;
      Report _R;
      long CPR = 0x00;
      int StateHI = 0x00;

      _In.ResetInterrupts(0xE2);
      //CPR = (((encoderHb_1 & 0xFFFF) << 8) + (encoderLb_1 & 0xFF));
   //T.LoadTrajectory(0x18, 0x28, 0x00, 0x00, 0x00, 0x56, 0x00, 0x08, 0x31,
0x27, 0x00, 0x00, 0x00, 0x00);//EA,00,28, 00,00,00,56,   00,08,31,27,
00,00,00,00
      _T.LoadTrajectory( 0x00, 0x2A, 0x00, 0x00, 0x00, 0x11, 0x00, 0x03, 0x12,
0x6F, 0x00, 0x00, encoderHb_1, encoderLb_1);
      _In.MaskInterrupts(0xE2, 0x04); // Trajectory-Complete Interrupt
      _T.StartMotion(0xE2);

    while(StateHI == 0x00) // true
    {
      StateHI = _In.HostInterrupt();
    }

    _T.stopAbruptly(0xE2);
    StateHI = 0x00;
    Serial.println("Home position Done!");
    _R.ReadRealPosition(0xE2);
    _In.ResetInterrupts(0xE2);
      _I.DefineHome();

}
//-------------------------------------
```

```
void Programs::PositioningCycles(int command, int trjmode, int trjcontrol, int
afwHB_1,int afwLB_1, int aswHB_2, int aswLB_2, int vfwHB_3, int vfwLB_3, int
vswHB_4, int vswLB_4,int pfwHB_5,int pfwLB_5, int pswHB_6,int pswLB_6, int
HB_encoder, int LB_encoder)
{
	Trajectory _T;
	Interrupt _I;
	Report _R;

	int StateHI = 0x00;
	long Countpos = 0x00;
	volatile long O_Countpos = 0x00;

	O_Countpos = (((pfwHB_5 &0xFFFFFFFF) << 24)+ ((pfwLB_5 & 0xFFFFFF)<<
16)+((pswHB_6 & 0xFFFF)<< 8)+((pswLB_6 & 0xFF)));


	_T.LoadTrajectory(trjmode, trjcontrol, afwHB_1, afwLB_1, aswHB_2,
aswLB_2, vfwHB_3, vfwLB_3, vswHB_4, vswLB_4, pfwHB_5, pfwLB_5, pswHB_6,
pswLB_6);
	_I.MaskInterrupts(command, 0x04); // Trajectory-Complete Interrupt
	_T.StartMotion(command);
	Countpos = O_Countpos;


    while(StateHI == 0x00) // true
    {
      StateHI = _I.HostInterrupt();
    }

    _T.stopAbruptly(0xE3);
    StateHI = 0x00;
    _R.ReadRealPosition(command);
    _I.ResetInterrupts(0xE4);

    delay(1500);
    //-------------------------------------------------------[ 1 ]-----------------
-----------------------
    Countpos = Countpos + O_Countpos;

    pfwHB_5 = (0xFFFFFFFF & Countpos) >> 24;
    pfwLB_5 = (0xFFFFFF & Countpos) >> 16;
    pswHB_6 = (0xFFFF & Countpos) >> 8;
    pswLB_6 = 0xFF & Countpos;

    _T.LoadTrajectory(trjmode, trjcontrol, afwHB_1, afwLB_1, aswHB_2, aswLB_2,
vfwHB_3, vfwLB_3, vswHB_4, vswLB_4, pfwHB_5, pfwLB_5, pswHB_6, pswLB_6);
	_I.MaskInterrupts(command, 0x04); // Trajectory-Complete Interrupt
	_T.StartMotion(command);

	while(StateHI == 0x00) // true
    {
      StateHI = _I.HostInterrupt();
    }
```

```
    _T.stopAbruptly(0xE3);
    StateHI = 0x00;
    _R.ReadRealPosition(command);
    _I.ResetInterrupts(0xE4);
    delay(1500);
      //-------------------------------------------------------[ 2 ]--------------
-------------------------
    Countpos = Countpos + O_Countpos;

    pfwHB_5 = (0xFFFFFFFF & Countpos) >> 24;
    pfwLB_5 = (0xFFFFFF & Countpos) >> 16;
    pswHB_6 = (0xFFFF & Countpos) >> 8;
    pswLB_6 = 0xFF & Countpos;

    _T.LoadTrajectory(trjmode, trjcontrol, afwHB_1, afwLB_1, aswHB_2, aswLB_2,
vfwHB_3, vfwLB_3, vswHB_4, vswLB_4, pfwHB_5, pfwLB_5, pswHB_6, pswLB_6);
    _I.MaskInterrupts(command, 0x04); // Trajectory-Complete Interrupt
    _T.StartMotion(command);

    while(StateHI == 0x00) // true
    {
      StateHI = _I.HostInterrupt();
    }

    _T.stopAbruptly(0xE3);
    StateHI = 0x00;
    _R.ReadRealPosition(command);
    _I.ResetInterrupts(0xE4);
    delay(1500);
      //-------------------------------------------------------[ 3 ]--------------
-------------------------
    Countpos = Countpos + O_Countpos;

    pfwHB_5 = (0xFFFFFFFF & Countpos) >> 24;
    pfwLB_5 = (0xFFFFFF & Countpos) >> 16;
    pswHB_6 = (0xFFFF & Countpos) >> 8;
    pswLB_6 = 0xFF & Countpos;

    _T.LoadTrajectory(trjmode, trjcontrol, afwHB_1, afwLB_1, aswHB_2, aswLB_2,
vfwHB_3, vfwLB_3, vswHB_4, vswLB_4, pfwHB_5, pfwLB_5, pswHB_6, pswLB_6);
    _I.MaskInterrupts(command, 0x04); // Trajectory-Complete Interrupt
    _T.StartMotion(command);

    while(StateHI == 0x00) // true
    {
      StateHI = _I.HostInterrupt();
    }

    _T.stopAbruptly(0xE3);
    StateHI = 0x00;
    _R.ReadRealPosition(command);
    _I.ResetInterrupts(0xE4);
    delay(1500);
      //-------------------------------------------------------[ 4 ]--------------
-------------------------
```

```
    Countpos = Countpos + O_Countpos;

    pfwHB_5 = (0xFFFFFFFF & Countpos) >> 24;
    pfwLB_5 = (0xFFFFFF & Countpos) >> 16;
    pswHB_6 = (0xFFFF & Countpos) >> 8;
    pswLB_6 = 0xFF & Countpos;

    _T.LoadTrajectory(trjmode, trjcontrol, afwHB_1, afwLB_1, aswHB_2, aswLB_2,
vfwHB_3, vfwLB_3, vswHB_4, vswLB_4, pfwHB_5, pfwLB_5, pswHB_6, pswLB_6);
    _I.MaskInterrupts(command, 0x04); // Trajectory-Complete Interrupt
    _T.StartMotion(command);

    while(StateHI == 0x00) // true
    {
       StateHI = _I.HostInterrupt();
    }

    _T.stopAbruptly(0xE3);
    StateHI = 0x00;
    _R.ReadRealPosition(command);
    _I.ResetInterrupts(0xE4);
    delay(1500);
      //-------------------------------------------------------[ 5 ]--------------
--------------------------
    Countpos = Countpos + O_Countpos;

    pfwHB_5 = (0xFFFFFFFF & Countpos) >> 24;
    pfwLB_5 = (0xFFFFFF & Countpos) >> 16;
    pswHB_6 = (0xFFFF & Countpos) >> 8;
    pswLB_6 = 0xFF & Countpos;

    _T.LoadTrajectory(trjmode, trjcontrol, afwHB_1, afwLB_1, aswHB_2, aswLB_2,
vfwHB_3, vfwLB_3, vswHB_4, vswLB_4, pfwHB_5, pfwLB_5, pswHB_6, pswLB_6);
    _I.MaskInterrupts(command, 0x04); // Trajectory-Complete Interrupt
    _T.StartMotion(command);

    while(StateHI == 0x00) // true
    {
    StateHI = _I.HostInterrupt();
    }

    _T.stopAbruptly(0xE3);
    StateHI = 0x00;
    _R.ReadRealPosition(command);
    _I.ResetInterrupts(0xE4);
    //Reverse
    delay(1500);
    //-------------------------------------------------------[ 6 ]----------------
----------------------
    Countpos = Countpos - O_Countpos;

    pfwHB_5 = (0xFFFFFFFF & Countpos) >> 24;
    pfwLB_5 = (0xFFFFFF & Countpos) >> 16;
    pswHB_6 = (0xFFFF & Countpos) >> 8;
    pswLB_6 = 0xFF & Countpos;
```

```
    _T.LoadTrajectory(trjmode, trjcontrol, afwHB_1, afwLB_1, aswHB_2, aswLB_2,
vfwHB_3, vfwLB_3, vswHB_4, vswLB_4, pfwHB_5, pfwLB_5, pswHB_6, pswLB_6);
    _I.MaskInterrupts(command, 0x04); // Trajectory-Complete Interrupt
    _T.StartMotion(command);

    while(StateHI == 0x00) // true
    {
    StateHI = _I.HostInterrupt();
    }

    _T.stopAbruptly(0xE3);
    StateHI = 0x00;
    _R.ReadRealPosition(command);
    _I.ResetInterrupts(0xE4);

    delay(1500);
    //-------------------------------------------------[ 7 ]-----------------
----------------------
    Countpos = Countpos - O_Countpos;

    pfwHB_5 = (0xFFFFFFFF & Countpos) >> 24;
    pfwLB_5 = (0xFFFFFF & Countpos) >> 16;
    pswHB_6 = (0xFFFF & Countpos) >> 8;
    pswLB_6 = 0xFF & Countpos;

    _T.LoadTrajectory(trjmode, trjcontrol, afwHB_1, afwLB_1, aswHB_2, aswLB_2,
vfwHB_3, vfwLB_3, vswHB_4, vswLB_4, pfwHB_5, pfwLB_5, pswHB_6, pswLB_6);
    _I.MaskInterrupts(command, 0x04); // Trajectory-Complete Interrupt
    _T.StartMotion(command);

    while(StateHI == 0x00) // true
    {
    StateHI = _I.HostInterrupt();
    }

    _T.stopAbruptly(0xE3);
    StateHI = 0x00;
    _R.ReadRealPosition(command);
    _I.ResetInterrupts(0xE4);

    delay(1500);
    //-------------------------------------------------[ 8 ]-----------------
----------------------
    Countpos = Countpos - O_Countpos;

    pfwHB_5 = (0xFFFFFFFF & Countpos) >> 24;
    pfwLB_5 = (0xFFFFFF & Countpos) >> 16;
    pswHB_6 = (0xFFFF & Countpos) >> 8;
    pswLB_6 = 0xFF & Countpos;

    _T.LoadTrajectory(trjmode, trjcontrol, afwHB_1, afwLB_1, aswHB_2, aswLB_2,
vfwHB_3, vfwLB_3, vswHB_4, vswLB_4, pfwHB_5, pfwLB_5, pswHB_6, pswLB_6);
    _I.MaskInterrupts(command, 0x04); // Trajectory-Complete Interrupt
    _T.StartMotion(command);
```

```
    while(StateHI == 0x00) // true
    {
    StateHI = _I.HostInterrupt();
    }

    _T.stopAbruptly(0xE3);
    StateHI = 0x00;
    _R.ReadRealPosition(command);
    _I.ResetInterrupts(0xE4);

   delay(1500);
   //-----------------------------------------------------[ 9 ]-----------------
----------------------
   Countpos = Countpos - O_Countpos;

   pfwHB_5 = (0xFFFFFFFF & Countpos) >> 24;
   pfwLB_5 = (0xFFFFFF & Countpos) >> 16;
   pswHB_6 = (0xFFFF & Countpos) >> 8;
   pswLB_6 = 0xFF & Countpos;

   _T.LoadTrajectory(trjmode, trjcontrol, afwHB_1, afwLB_1, aswHB_2, aswLB_2,
vfwHB_3, vfwLB_3, vswHB_4, vswLB_4, pfwHB_5, pfwLB_5, pswHB_6, pswLB_6);
   _I.MaskInterrupts(command, 0x04); // Trajectory-Complete Interrupt
   _T.StartMotion(command);

   while(StateHI == 0x00) // true
   {
   StateHI = _I.HostInterrupt();
   }

   _T.stopAbruptly(0xE3);
   StateHI = 0x00;
   _R.ReadRealPosition(command);
   _I.ResetInterrupts(0xE4);

   delay(1500);
   //-----------------------------------------------------[ 10 ]----------------
----------------------
   Countpos = Countpos - O_Countpos;

   pfwHB_5 = (0xFFFFFFFF & Countpos) >> 24;
   pfwLB_5 = (0xFFFFFF & Countpos) >> 16;
   pswHB_6 = (0xFFFF & Countpos) >> 8;
   pswLB_6 = 0xFF & Countpos;

   _T.LoadTrajectory(trjmode, trjcontrol, afwHB_1, afwLB_1, aswHB_2, aswLB_2,
vfwHB_3, vfwLB_3, vswHB_4, vswLB_4, pfwHB_5, pfwLB_5, pswHB_6, pswLB_6);
   _I.MaskInterrupts(command, 0x04); // Trajectory-Complete Interrupt
   _T.StartMotion(command);

   while(StateHI == 0x00) // true
   {
   StateHI = _I.HostInterrupt();
   }
```

```
   _T.stopAbruptly(0xE3);
   StateHI = 0x00;
   _R.ReadRealPosition(command);
   _I.ResetInterrupts(0xE4);

   delay(1500);
   //-----------------------------------------------------[ 11 ]-----------------
-----------------------
   //((pswHB_6 & 0xFFFF)<< 8)+((pswLB_6 & 0xFF));

   Countpos = (((HB_encoder & 0xFFFF) << 8) + (LB_encoder & 0xFF));
   Serial.println("Position calculada : "+String(Countpos,HEX));

   pfwHB_5 = (0xFFFFFFFF & Countpos) >> 24;
   pfwLB_5 = (0xFFFFFF & Countpos) >> 16;
   pswHB_6 = (0xFFFF & Countpos) >> 8;
   pswLB_6 = 0xFF & Countpos;

   Serial.println("pfwHB_5 : "+String(pfwHB_5,HEX));
   Serial.println("pfwLB_5 : "+String(pfwLB_5,HEX));
   Serial.println("pfwHB_6 : "+String(pswHB_6,HEX));
   Serial.println("pfwLB_6 : "+String(pswLB_6,HEX));

   _T.LoadTrajectory(trjmode, trjcontrol, afwHB_1, afwLB_1, aswHB_2, aswLB_2,
vfwHB_3, vfwLB_3, vswHB_4, vswLB_4, pfwHB_5, pfwLB_5, pswHB_6, pswLB_6);
   _I.MaskInterrupts(command, 0x04); // Trajectory-Complete Interrupt
   _T.StartMotion(command);

   while(StateHI == 0x00) // true
   {
   StateHI = _I.HostInterrupt();
   }

   _T.stopAbruptly(0xE3);
   StateHI = 0x00;
   _R.ReadRealPosition(command);
   _I.ResetInterrupts(0xE4);
    Serial.println("------------Finished Cycle----------------------");
}
```

```
/*
   COMserial.h - library for Motion Control code using LM629 controller.
   Created by Miguel Reyes, July 15, 2018.
   Released into the public domain.
*/

#ifndef COMserial_h
#define COMserial_h
#include "Arduino.h"

class COMserial
{
 public:


   COMserial( ); // Constructor


       int StringToHex(String hexNum);

   private:

  };

#endif
```

```
/*
   COMserial.cpp - library for Motion Control code using LM629 controller.
   Created by Miguel Reyes, July 15, 2018.
   Released into the public domain.
*/

#include "Arduino.h"
#include "COMserial.h"


COMserial::COMserial() // Constructor
{

}
int  COMserial::StringToHex(String hexNum)
{
       int value = 0;
       int valueByte = 0;
       char arrayDigit[3] = {'\0'};
       int digit[3] = {'\0'};
       boolean flag_00 = false;
       hexNum.toCharArray(arrayDigit, 3);

       for (int i = 0; i <= 1; i++)
       {
              if (arrayDigit[i] >= '0' && arrayDigit[i] <= '9')
              digit[i] = String(arrayDigit[i]).toInt();
              if (arrayDigit[i] == 'A')
              digit[i] = 10;
              if (arrayDigit[i] == 'B')
              digit[i] = 11;
              if (arrayDigit[i] == 'C')
              digit[i] = 12;
              if (arrayDigit[i] == 'D')
              digit[i] = 13;
              if (arrayDigit[i] == 'E')
              digit[i] = 14;
              if (arrayDigit[i] == 'F')
              {
                     digit[i] = 15;
              }
              flag_00 = true;
       }
       while (flag_00)
       {
       value = (digit[0] * 16 + digit[1] * 1);
              flag_00 = false;
              return value;
       }
}
```

```
/*
   ControlLines.h - library for Motion Control code using LM629 controller.
   Created by Miguel Reyes, June 30, 2018.
   Released into the public domain.
*/

#ifndef ControlLines_h
#define ControlLines_h
#include "Arduino.h"

class ControlLines
{
 public:

  const byte cs = 30; //ok
  const byte rd = 31; //ok
  const byte wr = 32; //ok
  const byte ps = 33; //ok

   ControlLines( ); // Constructor

    void WritingCommands(void);
    void ReadingStatus(void);
    void ReadingData(void);
    void WritingData(void);
    void HostInput(void);
    void HostOutput(void);
    void StandbyPortC(void);

  private:

  };

#endif
```

```
*
   ControlLines.cpp - library for Motion Control code using LM629 controller.
   Created by Miguel Reyes, June 30, 2018.
   Released into the public domain.
*/

#include "Arduino.h"
#include "ControlLines.h"

ControlLines::ControlLines() // Constructor
{
  pinMode(cs, OUTPUT);
  pinMode(ps, OUTPUT);
  pinMode(wr, OUTPUT);
  pinMode(rd, OUTPUT);
}
void ControlLines::WritingCommands(void)
{
  digitalWrite(cs, LOW);
  digitalWrite(ps, LOW);
  delayMicroseconds(3);
  digitalWrite(wr, LOW);
  digitalWrite(rd, HIGH);
}
void ControlLines::ReadingStatus(void)
{
  digitalWrite(cs, LOW);
  digitalWrite(ps, LOW);
  delayMicroseconds(3);
  digitalWrite(rd, LOW);
  digitalWrite(wr, HIGH);
}
void ControlLines::ReadingData(void)
{
  digitalWrite(cs, LOW);
  digitalWrite(ps, HIGH);
  delayMicroseconds(3);
  digitalWrite(rd, LOW);
  digitalWrite(wr, HIGH);
}
void ControlLines::WritingData(void)
{
  digitalWrite(cs, LOW);
  digitalWrite(ps, HIGH);
  delayMicroseconds(3);
  digitalWrite(wr, LOW);
  digitalWrite(rd, HIGH);
}
void ControlLines::HostInput(void)
{
  DDRA   = 0x00; // Inputs
  PORTA  = 0x00; // Status
}
void ControlLines::HostOutput(void)
{
```

```
  DDRA  = 0xFF; //Outputs
  PORTA = 0x00; //Status
}
void ControlLines::StandbyPortC(void)
{
  DDRC  = 0xF7;
  PORTC = 0xF7;
  delayMicroseconds(3);
}
```

```
/*
Filter.h - library for Motion Control code using LM629 controller.
Created by Miguel Reyes, July 12, 2018.
Released into the public domain.
*/

#ifndef Filter_h
#define Filter_h
#include "Arduino.h"

class Filter
{
public:
       Filter(); // Constructor

       void LoadFilterParameters(int der_term,int type_fc,int kp_HB,int
kp_LB,int ki_HB,int ki_LB,int kd_HB,int kd_LB,int il_HB,int il_LB);
    void UpdateFilter(void);


private:

 const byte LFIL = 0x1E; /* -Filter- Load Filter Parameters*/
 const byte UDF = 0x04;  /* -Filter- Update Filter*/
};

#endif
```

```
/*
  Filter.cpp - library for Motion Control code using LM629 controller.
  Created by Miguel Reyes, July 12, 2018.
  Released into the public domain.
*/

#include "Arduino.h"
#include "Filter.h"
#include <ControlLines.h>
#include <Interrupt.h>
#include <Report.h>


Filter::Filter( ) // Constructor
{

}
void Filter::LoadFilterParameters(int der_term, int type_fc, int kp_HB, int
kp_LB, int ki_HB, int ki_LB, int kd_HB, int kd_LB, int il_HB, int il_LB)
{
  ControlLines CL1;
  Interrupt I;
  Report rep;

  int indata = 0x00;

  CL1.HostOutput();
  CL1.WritingCommands();
  PORTA = LFIL;
  digitalWrite(CL1.wr, HIGH);
  delayMicroseconds(3);
  I.CheckBusyBit();

  CL1.WritingData();
  PORTA = byte(der_term);
  digitalWrite(CL1.wr, HIGH);
  delayMicroseconds(3);

   CL1.WritingData();
   PORTA = byte(type_fc);
   digitalWrite(CL1.wr, HIGH);
   delayMicroseconds(3);
   I.CheckBusyBit();

  unsigned int Count_kp = (((kp_HB & 0xFFFF)<< 8)+(kp_LB & 0xFF));
  if (Count_kp > 0x00)
  {
    CL1.WritingData();
    PORTA = byte(kp_HB);
    digitalWrite(CL1.wr, HIGH);
    delayMicroseconds(3);

    CL1.WritingData();
    PORTA = byte(kp_LB);
    digitalWrite(CL1.wr, HIGH);
```

```
    delayMicroseconds(3);
    I.CheckBusyBit();
  }
  unsigned int Count_ki = (((ki_HB & 0xFFFF)<< 8)+(ki_LB & 0xFF));
  if (Count_ki > 0x00)
  {
    CL1.WritingData();
    PORTA = byte(ki_HB);
    digitalWrite(CL1.wr, HIGH);
    delayMicroseconds(3);

    CL1.WritingData();
    PORTA = byte(ki_LB);
    digitalWrite(CL1.wr, HIGH);
    delayMicroseconds(3);
    I.CheckBusyBit();
  }
  unsigned int Count_kd = (((kd_HB & 0xFFFF)<< 8)+(kd_LB & 0xFF));
  if (Count_kd > 0x00)
  {
    CL1.WritingData();
    PORTA = byte(kd_HB);
    digitalWrite(CL1.wr, HIGH);
    delayMicroseconds(3);

    CL1.WritingData();
    PORTA = byte(kd_LB);
    digitalWrite(CL1.wr, HIGH);
    delayMicroseconds(3);
    I.CheckBusyBit();
    }
  unsigned int Count_il = (((il_HB & 0xFFFF)<< 8)+(il_LB & 0xFF));
  if (Count_il > 0x00)
   {
    CL1.WritingData();
    PORTA = byte(il_HB);
    digitalWrite(CL1.wr, HIGH);
    delayMicroseconds(3);

    CL1.WritingData();
    PORTA = byte(il_LB);
    digitalWrite(CL1.wr, HIGH);
    delayMicroseconds(3);
    I.CheckBusyBit();
   }
  CL1.StandbyPortC();
indata = rep.ReadStatusByte();
String stringOne = ("# DERIV.-TERM: " + String(der_term,DEC)+ " # COEF. P: " +
String(Count_kp,DEC) + " # COEF. I: " + String(Count_ki,DEC)+ " # COEF. D: " +
String(Count_kd,DEC) +" # INTEG. LIM. TERM: " + String(Count_il,DEC));
Serial.println(stringOne);
}

void Filter::UpdateFilter(void)
{
```

```
  ControlLines CL1;
  Interrupt I;

  CL1.HostOutput();
  CL1.WritingCommands();
  PORTA = UDF;
  digitalWrite(CL1.wr, HIGH);
  delayMicroseconds(3);
  I.CheckBusyBit();
  CL1.StandbyPortC();

  Serial.println("Updated Filter");
}
```

```
/*
   Initialize.h - library for Motion Control code using LM629 controller.
   Created by Miguel Reyes, June 30, 2018.
   Released into the public domain.
*/
//-------------------------------------------------------

#ifndef Initialize_h
#define Initialize_h
#include"Arduino.h"


//-------------------------------------------------------

class Initialize
{
  public:
       const byte tps = 50; // Turn On_Off_PowerSupply of DC Motor.

    Initialize( ); // Constructor Initialize( )

    void HardwareReset(int command);
    void SoftwareReset(int command);
    void DefineHome(void);
    void HardwareInitialization(void);
    void SoftwareInitialization(void);

  private:

   const byte RESET = 0x00; /* -Initialize- Reset LM629/LM628*/
   const byte PORT8 = 0x05; /* -Initialize- Select 8-Bit Output*/
   const byte DFH = 0x02; /* -Initialize- Define Home*/
   const byte rst = 35;   /*Reset*/
   const byte RSTI = 0x1D; /*Interrupt - Reset Interrupts*/


};

#endif
```

```
/*
   Initialize.cpp - library for Motion Control code using LM629 controller.
   Created by Miguel Reyes, June 30, 2018.
   Released into the public domain.
*/
//-------------------------------------------------------------

#include "Arduino.h"
#include "Initialize.h"
#include <ControlLines.h>
#include <Interrupt.h>
#include <Report.h>
#include <Trajectory.h>


//-------------------------------------------------------------


Initialize::Initialize( ) //Constructor
{

  pinMode(rst, OUTPUT);
  pinMode(tps, OUTPUT);


}
//-----------------------------
void Initialize::HardwareInitialization(void)
{
  Interrupt intr;
  ControlLines CL1;
  Report rep;
  //Trajectory _T;

  int readPortA = 0x00;

 // _T.movementState = false;
  delayMicroseconds(2000);
  HardwareReset(0xE7);
  readPortA = rep.ReadStatusByte();
  Serial.println(readPortA,HEX);

  if (readPortA == 0xC4 || readPortA == 0x84)
  {
    intr.MaskInterrupts(0x00,0x00);
    intr.ResetInterrupts(0xE7);
    intr.CheckBusyBit();
    readPortA = rep.ReadStatusByte();
    Serial.println(readPortA,HEX);

    if (readPortA == 0xC0 || readPortA == 0x80)
    {
      intr.CheckBusyBit();
         CL1.StandbyPortC();
         digitalWrite(tps,HIGH);
```

```
        Serial.println("Initialization by Hardware Done!");

       }
    else
      HardwareInitialization();
  }
  else
        HardwareInitialization();
}
//-----------------------------
void Initialize::SoftwareInitialization(void)
{
  Interrupt I;
  ControlLines CL1;
  Report rep;
  //Trajectory _T;

  int readPortA = 0x00;

  //_T.movementState = false;
  delayMicroseconds(2000);
  SoftwareReset(0xE6);
  delayMicroseconds(2000);
  /*Port8 by Defalut*/
  I.CheckBusyBit();

  CL1.HostOutput();
  CL1.WritingCommands();
  PORTA = RSTI;
  digitalWrite(CL1.wr, HIGH);
  delayMicroseconds(3);
  I.CheckBusyBit();

  CL1.WritingData();
  PORTA = 0xFF;
  digitalWrite(CL1.wr, HIGH);
  delayMicroseconds(3);

  CL1.WritingData();
  PORTA = 0x00;
  digitalWrite(CL1.wr, HIGH);
  delayMicroseconds(3);
  I.CheckBusyBit();

  readPortA = rep.ReadStatusByte();
  Serial.println(readPortA,HEX);
  digitalWrite(tps,HIGH);
  Serial.println("Initialization by Software Done!");



}
//-----------------------------
void Initialize::SoftwareReset(int command)
{
```

```
  Interrupt I;
  ControlLines CL1;

  CL1.HostOutput();
  CL1.WritingCommands();
  PORTA = RESET;
  digitalWrite(CL1.wr, HIGH);
  delayMicroseconds(3);
  I.CheckBusyBit();
  /*PORT8 by Default*/
  CL1.StandbyPortC();
  if (command == 0xFD)
  {
  Serial.println("Software Reset LM629 Done!.");
  }
  }
//-----------------------------
void Initialize::HardwareReset(int command)
{

  digitalWrite(rst, LOW);
  delayMicroseconds(4000);
  digitalWrite(rst, HIGH);
  delayMicroseconds(4000);
  if (command == 0xFE)
  {
  Serial.println("Hardware Reset LM629 Done!.");
  }
}
//-----------------------------
void Initialize::DefineHome(void)
{
 ControlLines ctrL;
 Interrupt intr;

ctrL.HostOutput();
ctrL.WritingCommands();
PORTA = DFH;
digitalWrite(ctrL.wr, HIGH);
delayMicroseconds(3);
intr.CheckBusyBit();
ctrL.StandbyPortC();
Serial.println("Defined Home Done!.");
}
//-----------------------------
```

```
/*
   Interrupt.h - library for Motion Control code using LM629 controller.
   Created by Miguel Reyes, June 30, 2018.
   Released into the public domain.
*/

#ifndef Interrupt_h
#define Interrupt_h
#include "Arduino.h"

class Interrupt
{
  public:
    Interrupt(); // Constructor

    void SetIndexPosition(void);
    void InterruptOnError(int Hbyte, int Lbyte); //threshold from 0x00 until
ox7FFF
    void StopOnError(int Hbyte, int Lbyte); //Adds the feature of turning off
the motor upon detecting excessive position error.
    void SetBreakpointAbsolute(int posHB_4,int posLB_3, int posHB_2,int
posLB_1);
    void SetBreakpointRelative(int posHB_4,int posLB_3, int posHB_2,int
posLB_1);
    void MaskInterrupts(int command,int enableInterrupt);
    void ResetInterrupts(int command);
    void CheckBusyBit(void);
    int HostInterrupt(void);

  private:

    const byte SIP = 0x03;  /* -Interrupt- Set Index Position*/
    const byte LPEI = 0x1B; /* -Interrupt- on Error*/
    const byte LPES = 0x1A; /* -Interrupt- stop on Error*/
    const byte SPBA = 0x20; /* -Interrupt- Set Breakpoint, Absolute*/
    const byte SPBR = 0x21; /* -Interrupt- set Breakpoint, Relative*/
    const byte MSKI = 0x1C; /* -Interrupt- Mask Interrupts*/
    const byte RSTI = 0x1D; /* -Interrupt- Reset Interrupts*/
    const byte HI = 53; /* Host Interrupt*/
    const byte BUZZER = 52; /* Buzzer */
    volatile int State_HI = 0x00; /* State Host Interrupt */

};
#endif
```

```
/*
   Interrupt.cpp - library for Motion Control code using LM629 controller.
   Created by Miguel Reyes, June 30, 2018.
   Released into the public domain.
*/


#include "Arduino.h"
#include "Interrupt.h"
#include <Report.h>
#include <ControlLines.h>
#include <Trajectory.h>

Interrupt::Interrupt()
{
        pinMode(HI, INPUT);
        pinMode(BUZZER, OUTPUT);
}

//-------------------------------------
void Interrupt::SetIndexPosition(void)
{
   ControlLines Cntrl;

   Cntrl.HostOutput();
   Cntrl.WritingCommands();
   PORTA = SIP;
   digitalWrite(Cntrl.wr, HIGH);
   delayMicroseconds(3);
   CheckBusyBit();
   Serial.println("Set Index Position Done!.");
}
//-------------------------------------
void Interrupt::InterruptOnError(int Hbyte, int Lbyte)
{
       ControlLines Cntrl;
    int threshold = 0x0000;

       Cntrl.HostOutput();
       Cntrl.WritingCommands();
       PORTA = LPEI;
       digitalWrite(Cntrl.wr, HIGH);
       delayMicroseconds(3);
       CheckBusyBit();

       Cntrl.WritingData();
       PORTA = byte(Hbyte);
       digitalWrite(Cntrl.wr, HIGH);
       delayMicroseconds(3);

       Cntrl.WritingData();
       PORTA = byte(Lbyte);
       digitalWrite(Cntrl.wr, HIGH);
       delayMicroseconds(3);
       CheckBusyBit();
```

```
      Serial.println("Interrupt on Error Done! :: Threshold = ");
      threshold = (((Hbyte & 0xFFFF)<< 8) + (Lbyte & 0xFF));
    Serial.println(String(threshold,DEC));
}
//-------------------------------------
void Interrupt::StopOnError(int Hbyte, int Lbyte)
{
      ControlLines Cntrl;
    int threshold = 0x0000;

      Cntrl.HostOutput();
      Cntrl.WritingCommands();
      PORTA = LPEI;
      digitalWrite(Cntrl.wr, HIGH);
      delayMicroseconds(3);
      CheckBusyBit();

      Cntrl.WritingData();
      PORTA = byte(Hbyte);
      digitalWrite(Cntrl.wr, HIGH);
      delayMicroseconds(3);

      Cntrl.WritingData();
      PORTA = byte(Lbyte);
      digitalWrite(Cntrl.wr, HIGH);
      delayMicroseconds(3);
      CheckBusyBit();

      Serial.println("Stop on Error Done! :: Threshold = ");
      threshold = (((Hbyte & 0xFFFF)<< 8) + (Lbyte & 0xFF));
    Serial.println(String(threshold,DEC));
}
//-------------------------------------
void Interrupt::SetBreakpointAbsolute(int posHB_4,int posLB_3, int posHB_2,int
posLB_1)
{
      ControlLines CL1;
      unsigned long breakpoint = 0x00;

      CL1.HostOutput();
      CL1.WritingCommands();
      PORTA = SPBA; // This command initialtes loading a relative breakpoint
      digitalWrite(CL1.wr, HIGH);
      delayMicroseconds(3);
      CheckBusyBit();

      CL1.WritingData();
      PORTA = byte(posHB_4);
      digitalWrite(CL1.wr, HIGH);
      delayMicroseconds(3);

      CL1.WritingData();
      PORTA = byte(posLB_3);
      digitalWrite(CL1.wr, HIGH);
```

```
        delayMicroseconds(3);
        CheckBusyBit();

        CL1.WritingData();
        PORTA = byte(posHB_2);
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);

        CL1.WritingData();
        PORTA = byte(posLB_1);
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);
        CheckBusyBit();

        breakpoint = (((posHB_4 & 0xFFFFFFFF) << 24)+ ((posLB_3 & 0xFFFFFF)<<
16)+((posHB_2 & 0xFFFF)<< 8)+((posLB_1 & 0xFF)));

        Serial.println("Updated Breakpoint Absolute at = " +
String(breakpoint,DEC) + " Done!");
}
//-------------------------------------
void Interrupt::SetBreakpointRelative(int posHB_4,int posLB_3, int posHB_2,int
posLB_1)
{

ControlLines CL1;
unsigned long breakpoint = 0x00;

CL1.HostOutput();
CL1.WritingCommands();
PORTA = SPBR; // This command initialtes loading a relative breakpoint
digitalWrite(CL1.wr, HIGH);
delayMicroseconds(3);
CheckBusyBit();

CL1.WritingData();
PORTA = byte(posHB_4);
digitalWrite(CL1.wr, HIGH);
delayMicroseconds(3);

CL1.WritingData();
PORTA = byte(posLB_3);
digitalWrite(CL1.wr, HIGH);
delayMicroseconds(3);
CheckBusyBit();

CL1.WritingData();
PORTA = byte(posHB_2);
digitalWrite(CL1.wr, HIGH);
delayMicroseconds(3);

CL1.WritingData();
PORTA = byte(posLB_1);
digitalWrite(CL1.wr, HIGH);
delayMicroseconds(3);
```

```
CheckBusyBit();

breakpoint = (((posHB_4 & 0xFFFFFFFF) << 24)+ ((posLB_3 & 0xFFFFFF)<<
16)+((posHB_2 & 0xFFFF)<< 8)+((posLB_1 & 0xFF)));

Serial.println("Updated Breakpoint Relative at = " + String(breakpoint,DEC) + "
Done!");

}
//-------------------------------------
void Interrupt::MaskInterrupts(int command, int enableInterrupt)
{

ControlLines CL1;

CL1.HostOutput();
CL1.WritingCommands();
PORTA = MSKI;
digitalWrite(CL1.wr, HIGH);
delayMicroseconds(3);
CheckBusyBit();

CL1.WritingData();
PORTA = 0xFF;//Word(2Bytes)-1Byte-HighByte don't care.
digitalWrite(CL1.wr,HIGH);
delayMicroseconds(3);

CL1.WritingData();
PORTA = byte(enableInterrupt); //2Byte-LowByte
digitalWrite(CL1.wr, HIGH);
delayMicroseconds(3);
CheckBusyBit();
CL1.StandbyPortC();

if(command == 0xF6)
{
Serial.println("Enabled Mask Interrupt : ");
Serial.println(enableInterrupt,BIN);
}

}
//-------------------------------------
void Interrupt::ResetInterrupts(int command)
{

ControlLines CL1;
if(command == 0xF5)
{
      MaskInterrupts(0x00,0x3F);
}

CL1.HostOutput();
CL1.WritingCommands();
PORTA = RSTI;
digitalWrite(CL1.wr, HIGH);
```

```
delayMicroseconds(3);
CheckBusyBit();

CL1.WritingData();
PORTA = 0x00;//Word(2Bytes)-1Byte-HighByte don't care.
digitalWrite(CL1.wr,HIGH);
delayMicroseconds(3);

CL1.WritingData();
//((pswHB_6 & 0xFFFF)<< 8)
PORTA = 0x00; //2Byte-LowByte
digitalWrite(CL1.wr, HIGH);
delayMicroseconds(2000);
CheckBusyBit();
CL1.StandbyPortC();

if(command == 0xF5)
{
Serial.println("Reset Interrupts Done!");
}
}
//-------------------------------------
void Interrupt::CheckBusyBit(void)
{
Report rep;
while(rep.ReadStatusByte() & 0x01);
}

//-------------------------------------
int Interrupt::HostInterrupt(void)
{
	Trajectory _T;
	State_HI = digitalRead(HI);
	if(State_HI)
	{
		digitalWrite(BUZZER, HIGH);
		delayMicroseconds(500);
		digitalWrite(BUZZER, LOW);
		digitalWrite(_T.stopwatchEnable,LOW);
	}

	return State_HI;
}
//-------------------------------------
```

```
/*
Report.h - library for Motion Control code using LM629 controller.
Created by Miguel Reyes, July 12, 2018.
Released into the public domain.
*/

#ifndef Report_h
#define Report_h
#include "Arduino.h"

class Report
{
public:

       Report(); // Constructor
        int ReadStatusByte(void);
       void ReadSignalsRegister(void);
       void ReadIndexPosition(void);
       void ReadDesiredPosition(void);
       long ReadRealPosition(int command);
       void ReadDesiredVelocity(void);
       void ReadRealVelocity(int command);
       void ReadIntegrationSum(void);

private:

// const int RDSTAT  = none  -Report- Read Status Byte*/
const byte RDSIGS = 0x0C; /* -Report- Read Signals Register*/
const byte RDIP = 0x09; /* -Report- Read Index Position*/
const byte RDDP = 0x08; /* -Report- Read Desired Position*/
const byte RDRP = 0x0A; /* -Report- Read Real Position*/
const byte RDDV = 0x07; /* -Report- Read Desired Velocity*/
const byte RDRV = 0x0B; /* -Report- Read Real Velocity*/
const byte RDSUM = 0x0D; /* -Report- Read Integration Sum*/

};

#endif
```

```
/*
Report.cpp - library for Motion Control code using LM629 controller.
Created by Miguel Reyes, July 12, 2018.
Released into the public domain.
*/

#include "Arduino.h"
#include "Report.h"
#include <ControlLines.h>
#include <Interrupt.h>

Report::Report() // Constructor

{

}
//-------------------------------------
int Report::ReadStatusByte(void)
{
  ControlLines CL1;

int readStatus = 0;
CL1.StandbyPortC();
CL1.ReadingStatus();
CL1.HostInput();
readStatus = PINA;
digitalWrite(CL1.rd, HIGH);
delayMicroseconds(3);
CL1.HostOutput();
CL1.StandbyPortC();
return readStatus;
}
//-------------------------------------
void Report::ReadSignalsRegister(void)
{
ControlLines C;
Interrupt I;

int HB_1 = 0x00;
int LB_2 = 0x00;

C.HostOutput();
C.WritingCommands();

PORTA = RDSIGS;
digitalWrite(C.wr,HIGH);
I.CheckBusyBit();

C.HostInput();
C.ReadingData();
HB_1 = PINA;
digitalWrite(C.rd, HIGH);
delayMicroseconds(3);

C.ReadingData();
```

```
LB_2 = PINA;
digitalWrite(C.rd, HIGH);
delayMicroseconds(3);
I.CheckBusyBit();

C.StandbyPortC();

Serial.println( " Read Signals Register :: |" + String(HB_1,BIN) + " | " +
String(LB_2,BIN));

}
void Report::ReadIndexPosition(void)
{
       ControlLines Cntrl;
       Interrupt I;

         byte HB_0 = 0x00;
         byte LB_0 = 0x00;
         byte HB_1 = 0x00;
         byte LB_1 = 0x00;
         long int IndxPos_= 0x00;

         Cntrl.HostOutput();
         Cntrl.WritingCommands();
         PORTA = RDIP;
         digitalWrite(Cntrl.wr, HIGH);
         delayMicroseconds(3);
         I.CheckBusyBit();

         Cntrl.HostInput();
         Cntrl.ReadingData();
         HB_0 = PINA;
         digitalWrite(Cntrl.rd, HIGH);
         delayMicroseconds(3);

         Cntrl.ReadingData();
         LB_0 = PINA;
         digitalWrite(Cntrl.rd, HIGH);
         delayMicroseconds(3);
         I.CheckBusyBit();

         Cntrl.HostInput();
         Cntrl.ReadingData();
         HB_1 = PINA;
         digitalWrite(Cntrl.rd, HIGH);
         delayMicroseconds(3);

         Cntrl.ReadingData();
         LB_1 = PINA;
         digitalWrite(Cntrl.rd, HIGH);
         delayMicroseconds(3);
         I.CheckBusyBit();

         Cntrl.StandbyPortC();
```

```
        IndxPos_ = (((HB_0 & 0xFFFFFFFF) << 24)+ ((LB_0 & 0xFFFFFF)<<
16)+((HB_1 & 0xFFFF)<< 8)+((LB_1 & 0xFF)));

        Serial.println("Read Index Position :: ");
        Serial.println(IndxPos_,DEC);
}
void Report::ReadDesiredPosition(void)
{
      ControlLines CL;
      Interrupt I;
long int Countpos_= 0;
int HB_2 = 0x00;
int LB_2 = 0x00;
int HB_1 = 0x00;
int LB_1 = 0x00;

      CL.HostOutput();
      CL.WritingCommands();
      PORTA = RDDP;
      digitalWrite(CL.wr, HIGH);
      delayMicroseconds(3);
      I.CheckBusyBit();

      CL.HostInput();
      CL.ReadingData();
      HB_2 = PINA;
      digitalWrite(CL.rd, HIGH);
      delayMicroseconds(3);

      CL.ReadingData();
      LB_2 = PINA;
      digitalWrite(CL.rd, HIGH);
      delayMicroseconds(3);
      I.CheckBusyBit();

      CL.HostInput();
      CL.ReadingData();
      HB_1 = PINA;
      digitalWrite(CL.rd, HIGH);
      delayMicroseconds(3);

      CL.ReadingData();
      LB_1 = PINA;
      digitalWrite(CL.rd, HIGH);
      delayMicroseconds(3);
      I.CheckBusyBit();

      CL.StandbyPortC();

      Countpos_ = (((HB_2 & 0xFFFFFFFF) << 24)+ ((LB_2 & 0xFFFFFF)<< 16)+((HB_1
& 0xFFFF)<< 8)+((LB_1 & 0xFF)));
      Serial.println("Read Desired Position : " + String(Countpos_,DEC));
}
long Report::ReadRealPosition(int command)
{
```

```
ControlLines CL;
Interrupt I;

  byte HB_0 = 0;
  byte LB_0 = 0;
  byte HB_1 = 0;
  byte LB_1 = 0;
  long Countpos_= 0;
  CL.HostOutput();
  CL.WritingCommands();
  PORTA = RDRP;
  digitalWrite(CL.wr, HIGH);
  delayMicroseconds(3);
  I.CheckBusyBit();

  CL.HostInput();
  CL.ReadingData();
  HB_0 = PINA;
  digitalWrite(CL.rd, HIGH);
  delayMicroseconds(3);

  CL.ReadingData();
  LB_0 = PINA;
  digitalWrite(CL.rd, HIGH);
  delayMicroseconds(3);
  I.CheckBusyBit();

  CL.HostInput();
  CL.ReadingData();
  HB_1 = PINA;
  digitalWrite(CL.rd, HIGH);
  delayMicroseconds(3);

  CL.ReadingData();
  LB_1 = PINA;
  digitalWrite(CL.rd, HIGH);
  delayMicroseconds(3);
  I.CheckBusyBit();

  CL.StandbyPortC();
  Countpos_ = (((HB_0 & 0xFFFFFFFF) << 24)+ ((LB_0 & 0xFFFFFF)<< 16)+((HB_1 &
0xFFFF)<< 8)+((LB_1 & 0xFF)));

  unsigned long currentTime = millis();

        String stringOne = (" # T: "+String(currentTime,DEC)+ " :# P: "
+String(Countpos_,DEC));
        Serial.println(stringOne);
  return Countpos_;

}
void Report::ReadDesiredVelocity(void)
{
      ControlLines CL;
      Interrupt I;
```

```
        long  Countvel_= 0x00;
        int HB_2 = 0x00;
        int LB_2 = 0x00;
        int HB_1 = 0x00;
        int LB_1 = 0x00;

        CL.HostOutput();
        CL.WritingCommands();
        PORTA = RDDV;
        digitalWrite(CL.wr, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();

        CL.HostInput();
        CL.ReadingData();
        HB_2 = PINA;
        digitalWrite(CL.rd, HIGH);
        delayMicroseconds(3);

        CL.ReadingData();
        LB_2 = PINA;
        digitalWrite(CL.rd, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();

        CL.HostInput();
        CL.ReadingData();
        HB_1 = PINA;
        digitalWrite(CL.rd, HIGH);
        delayMicroseconds(3);

        CL.ReadingData();
        LB_1 = PINA;
        digitalWrite(CL.rd, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();

        CL.StandbyPortC();

        unsigned long currentTime = millis();
        Countvel_ = (((HB_2 & 0xFFFFFFFF) << 24)+ ((LB_2 & 0xFFFFFF)<< 16)+((HB_1
& 0xFFFF)<< 8)+((LB_1 & 0xFF)));


        String stringOne = (" # T: "+String(currentTime,DEC)+ " :# V: "
+String(Countvel_,DEC));
        Serial.println(stringOne);
         //Serial.println("Read Desired Velocity : " + String(Countvel_,DEC));
}
void Report::ReadRealVelocity(int command)
{
        ControlLines CL;
        Interrupt I;
        //-----------------------------------------------
```

```
            signed long  Countvel_= 0;
              int HB_2 = 0x00;
              int LB_2 = 0x00;
              int HB_1 = 0x00;
              int LB_1 = 0x00;

              CL.HostOutput();
              CL.WritingCommands();
              PORTA = RDDV;
              digitalWrite(CL.wr, HIGH);
              delayMicroseconds(3);
              I.CheckBusyBit();

              CL.HostInput();
              CL.ReadingData();
              HB_2 = PINA;
              digitalWrite(CL.rd, HIGH);
              delayMicroseconds(3);

              CL.ReadingData();
              LB_2 = PINA;
              digitalWrite(CL.rd, HIGH);
              delayMicroseconds(3);
              I.CheckBusyBit();

              CL.HostInput();
              CL.ReadingData();
              HB_1 = PINA;
              digitalWrite(CL.rd, HIGH);
              delayMicroseconds(3);

              CL.ReadingData();
              LB_1 = PINA;
              digitalWrite(CL.rd, HIGH);
              delayMicroseconds(3);
              I.CheckBusyBit();

              CL.StandbyPortC();

              Countvel_ = (((HB_2 & 0xFFFFFFFF) << 24)+ ((LB_2 & 0xFFFFFF)<<
16)+((HB_1 & 0xFFFF)<< 8)+((LB_1 & 0xFF)));

       //---------------------------------------------
       unsigned long realVel_= 0x00;
       int HB_3 = 0x00;
       int LB_3 = 0x00;

       CL.HostOutput();
       CL.WritingCommands();
       PORTA = RDRV;
       digitalWrite(CL.wr, HIGH);
       delayMicroseconds(3);
       I.CheckBusyBit();

       CL.HostInput();
```

```
        CL.ReadingData();
        HB_3 = PINA;
        digitalWrite(CL.rd, HIGH);
        delayMicroseconds(3);

        CL.ReadingData();
        LB_3 = PINA;
        digitalWrite(CL.rd, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();

        CL.StandbyPortC();

        if(command == 0xEE)
          {
                Serial.println("Read Real Velocity :");
          }

        unsigned long currentTime = millis();

        int a = 2;
        int b = 6;
      int number = (((0xFF & a) << 8) + (0xFF & b));
        realVel_ = ((0xFFFF & number) << 8);


        String stringOne = ( " @ TESTING, " + String(currentTime,DEC) + " , " +
String(realVel_,BIN));
        //+ " , " + String(LB_3,DEC)
        Serial.println(stringOne);
}
void Report::ReadIntegrationSum(void)
{
        ControlLines CL;
        Interrupt I;

        long int integterm_= 0x00;
        int HB_1 = 0x00;
        int LB_1 = 0x00;

        CL.HostOutput();
        CL.WritingCommands();
        PORTA = RDSUM;
        digitalWrite(CL.wr, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();

        CL.HostInput();
        CL.ReadingData();
        HB_1 = PINA;
        digitalWrite(CL.rd, HIGH);
        delayMicroseconds(3);

        CL.ReadingData();
        LB_1 = PINA;
```

```
        digitalWrite(CL.rd, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();

        CL.StandbyPortC();

        integterm_ = (((HB_1 & 0xFFFF)<< 8)+(LB_1 & 0xFF));

        Serial.println("Read Integration-Term summation value : " +
String(integterm_,DEC));

}
```

```
/*
Trajectory.h - library for Motion Control code using LM629 controller.
Created by Miguel Reyes, July 15, 2018.
Released into the public domain.
*/

#ifndef Trajectory_h
#define Trajectory_h
#include "Arduino.h"


class Trajectory
{
public:
    volatile boolean movementState = false;
    volatile int  stopwatchEnable = 51;

       Trajectory(); // Constructor

       void LoadTrajectory(int trjmode, int trjcontrol, int afwHB_1, int
afwLB_1, int aswHB_2, int aswLB_2, int vfwHB_3, int vfwLB_3, int vswHB_4, int
vswLB_4, int pfwHB_5, int pfwLB_5, int pswHB_6, int pswLB_6);
       void StartMotion(int command);
       void turnOffstop(int command);
       void stopAbruptly(int command);
       void stopSmoothly(int command);

private:

 const byte  LTRJ = 0x1F;
 const byte STT = 0x01;


};
#endif
```

```
/*
Trajectory.cpp - library for Motion Control code using LM629 controller.
Created by Miguel Reyes, July 12, 2018.
Released into the public domain.
*/

#include "Arduino.h"
#include "Trajectory.h"
#include <ControlLines.h>
#include <Interrupt.h>
#include <Report.h>
#include <Initialize.h>


Trajectory::Trajectory( ) // Constructor
{
        pinMode(stopwatchEnable, OUTPUT);
}
void Trajectory::LoadTrajectory(int trjmode, int trjcontrol, int afwHB_1,int
afwLB_1, int aswHB_2, int aswLB_2, int vfwHB_3, int vfwLB_3, int vswHB_4, int
vswLB_4,int pfwHB_5,int pfwLB_5, int pswHB_6,int pswLB_6)
{
 ControlLines CL1;
 Interrupt I;
 Report rep;

 int indata = 0x00;
 unsigned long Countpos = 0x00;
 unsigned long Countsvel = 0x00;
 unsigned long Countacel = 0x00;
 long CPR = 0x0000;
 CL1.HostOutput();
 CL1.WritingCommands();
 PORTA = LTRJ;
 digitalWrite(CL1.wr, HIGH);
 delayMicroseconds(3);
 I.CheckBusyBit();

CL1.WritingData();
PORTA = byte(trjmode);
digitalWrite(CL1.wr, HIGH);
delayMicroseconds(3);

CL1.WritingData();
PORTA = byte(trjcontrol);
digitalWrite(CL1.wr, HIGH);
delayMicroseconds(3);
I.CheckBusyBit();

Countacel = (((afwHB_1 &0xFFFFFFFF) << 24)+ ((afwLB_1 & 0xFFFFFF)<<
16)+((aswHB_2 & 0xFFFF)<< 8)+((aswLB_2 & 0xFF)));
if(Countacel > 0x00)
{
       CL1.WritingData();
       PORTA = byte(afwHB_1);
```

```
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);

        CL1.WritingData();
        PORTA = byte(afwLB_1);
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();

        CL1.WritingData();
        PORTA = byte(aswHB_2);
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);

        CL1.WritingData();
        PORTA = byte(aswLB_2);
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();

}
Countsvel = (((vfwHB_3 &0xFFFFFFFF) << 24)+((vfwLB_3 & 0xFFFFFF)<< 16)+((vswHB_4
& 0xFFFF)<< 8)+((vswLB_4 & 0xFF)));
if(Countsvel > 0x00)
{
        CL1.WritingData();
        PORTA = byte(vfwHB_3);
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);

        CL1.WritingData();
        PORTA = byte(vfwLB_3);
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();

        CL1.WritingData();
        PORTA = byte(vswHB_4);
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);

        CL1.WritingData();
        PORTA = byte(vswLB_4);
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();

}
Countpos = (((pfwHB_5 &0xFFFFFFFF) << 24)+ ((pfwLB_5 & 0xFFFFFF)<< 16)+((pswHB_6
& 0xFFFF)<< 8)+((pswLB_6 & 0xFF)));
if(Countpos > 0x00)
{
        CL1.WritingData();
        PORTA = byte(pfwHB_5);
        digitalWrite(CL1.wr, HIGH);
```

```
        delayMicroseconds(3);

        CL1.WritingData();
        PORTA = byte(pfwLB_5);
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();

        CL1.WritingData();
        PORTA = byte(pswHB_6);
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);

        CL1.WritingData();
        PORTA = byte(pswLB_6);
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();
}
indata = rep.ReadStatusByte();
String stringOne = ("# STATUS BYTE : " + String(indata,BIN) + " # TRJ. MODE : "
+ String(trjmode,HEX)+ " # TRJ. CONTROL : " + String(trjcontrol,HEX) + " # A : "
+ String(Countacel,HEX)+ " # MAX. W : " + String(Countsvel,HEX) +" # F. POS : "
+ String(Countpos,HEX));
Serial.println(stringOne);
CL1.StandbyPortC();
Serial.println("Updated Trajectory");



}
void Trajectory::StartMotion(int command)
{
ControlLines CL1;
Interrupt I;


CL1.HostOutput();
CL1.WritingCommands();
PORTA = STT;
digitalWrite(CL1.wr, HIGH);
delayMicroseconds(3);
digitalWrite(stopwatchEnable,HIGH);
I.CheckBusyBit();

Serial.println("Motor on");

        if(command != 0xE5 || command != 0xE3) //VelMovBreakp();
        {
                movementState = true;
        }

        CL1.StandbyPortC();
}
void Trajectory::turnOffstop(int command)
```

```
{

	ControlLines CL1;
	Interrupt I;
	Initialize In;

	CL1.HostOutput();
	CL1.WritingCommands();
	PORTA = LTRJ;
	digitalWrite(CL1.wr, HIGH);
	delayMicroseconds(3);
	I.CheckBusyBit();

	CL1.WritingData();
	PORTA = 0x01;
	digitalWrite(CL1.wr, HIGH);
	delayMicroseconds(3);

	CL1.WritingData();
	PORTA = 0x00;
	digitalWrite(CL1.wr, HIGH);
	delayMicroseconds(3);
	I.CheckBusyBit();

	CL1.WritingCommands();
	PORTA = STT;
	digitalWrite(CL1.wr, HIGH);
	delayMicroseconds(3);
	I.CheckBusyBit();

	if(command != 0xE5 || command != 0xE3)
	{
		movementState = false;
	}

	digitalWrite(In.tps,LOW);
	Serial.println("Turn off Motor");
	CL1.StandbyPortC();
	I.ResetInterrupts(0xE8);

	}
//-------------------------------------
void Trajectory::stopAbruptly(int command)
{

	ControlLines CL1;
	Interrupt I;

	CL1.HostOutput();
	CL1.WritingCommands();
	PORTA = LTRJ;
	digitalWrite(CL1.wr, HIGH);
	delayMicroseconds(3);
	I.CheckBusyBit();
```

```
        CL1.WritingData();
        PORTA = 0x02;
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);

        CL1.WritingData();
        PORTA = 0x00;
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();

        CL1.WritingCommands();
        PORTA = STT;
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();

        if(command != 0xE5 || command != 0xE3)
        {
              movementState = false;
        }

        Serial.println("Stop Abruptly");
        CL1.StandbyPortC();
        }
//--------------------------------------
void Trajectory::stopSmoothly(int command)
{

        ControlLines CL1;
        Interrupt I;

        CL1.HostOutput();
        CL1.WritingCommands();
        PORTA = LTRJ;
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();

        CL1.WritingData();
        PORTA = 0x03;
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);

        CL1.WritingData();
        PORTA = 0x00;
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();

        CL1.WritingCommands();
        PORTA = STT;
        digitalWrite(CL1.wr, HIGH);
        delayMicroseconds(3);
        I.CheckBusyBit();
```

```
if(command != 0xE5 || command != 0xE3)
{
	movementState = false;
}

Serial.println("Stop Smoothty");
CL1.StandbyPortC();
}
```