UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO DE UNA APLICACIÓN PARA LA IDENTIFICACIÓN Y CONTROL DE UN MOTOR DE CORRIENTE CONTINUA MEDIANTE LABVIEW

AUTOR: Dotor Alonso, Luis

TUTOR: Herrero Durá, Juan Manuel

Curso Académico: 2013-14

RESUMEN

En el proyecto que se desarrolla a continuación se pretende controlar y automatizar un robot para operaciones de taladrado mediante una aplicación en LabVIEW. Este robot incluye tres motores para controlar las 3 coordenadas del espacio (X, Y, Z), de tal manera que pueda controlar la posición de un cabezal (broca). Este cabezal es movido por un cuarto motor que hace girar una broca proporcionándole la velocidad suficiente para poder realizar la operación de taladrado.

Para empezar, se ha identificado un modelo de posición y velocidad de los motores disponibles en el laboratorio (motor de la casa Feedback). Para ello se ha tenido que realizar previamente un programa inicial en LabVIEW capaz de realizar ensayos y almacenar datos. En este programa se ha realizado un algoritmo previo para transformar la información devuelta por el sensor de posición del motor en una información válida para la identificación.

Una vez se han realizados los preparativos oportunos, se ha procedido a realizar los ensayos tanto de velocidad como de posición y, posteriormente, se ha obtenido un modelo matemático mediante la utilización de la herramienta System Identification Tool. Más tarde, utilizando la herramienta rltool de Matlab, se han obtenido los reguladores para los dos modelos, acto seguido se ha procedido a la validación de los mismos en un diagrama de bloques de Simulink.

Obtenidos los reguladores, se ha procedido a construir el bucle de control de los cuatro motores en LabVIEW, incluyendo la creación de un PID programado en C para realizar la función del regulador.

Una vez se han controlado los 3 motores de posición y el motor de velocidad del robot, se ha procedido a hacer un automatismo en la aplicación, capaz de controlar secuencialmente el proceso de taladrado. Se pretende que el automatismo sea capaz de reconocer un fichero de texto creado por el usuario, dicho fichero ha de contener las coordenadas X e Y de los puntos del plano que el robot va a taladrar y un valor entero que represente el tipo de punto a taladrar.

El automatismo se ha realizado para que sea capaz de reconocer secuencialmente cada uno de los puntos del fichero. De esta forma, cada vez que se lea un punto:

1. Los motores X e Y se trasladarán a la posición correcta (coordenadas X e Y del fichero) para situar el cabezal en el plano.
2. Seguidamente, girará el cabezal del robot hasta alcanzar una velocidad que dependerá del tipo de agujero a realizar (valor entero del fichero que define el tipo de punto).
3. Posteriormente, se hará bajar el motor Z para realizar el taladrado y luego se le ordenará subir de nuevo. La distancia recorrida por el motor Z también dependerá del tipo de punto, ya que podría interesar hacer agujeros pasantes o no.
4. A continuación, parará el motor del cabezal.
5. Finalmente, el programa tendrá que ser capaz de distinguir entre dos puntos consecutivos, de tal manera que si el siguiente punto leído en el fichero es distinto, tendrá que volver a una posición de referencia para prepararse antes de realizar el siguiente agujero.

Por último, se ha realizado una interfaz gráfica para el programa, la cual se ha dividido en tres partes:

- Un modelo 3D del robot modelado en LabVIEW, capaz de reproducir cada uno de los movimientos de los cuatro motores.
- Una gráfica cartesiana XY que representa en tiempo real: las coordenadas de los puntos ejecutados, el punto en ejecución y el itinerario de puntos a realizar.
- Unos leds que parpadean en tiempo real en función del tipo de agujero (tipo de punto) que se está realizando.

En el desarrollo de la aplicación se ha considerado como especificaciones del cliente el uso de la maqueta del robot (los cuatro motores) y la tarjeta de adquisición de datos *AD-Link PCI 9112*. Como esta tarjeta dispone únicamente de dos salidas analógicas y se necesitan cuatro (una por cada motor), se ha hecho uso de una segunda tarjeta (idéntica) instalada en otro ordenador, de tal manera que el programa ha sido dividido en dos:

- Una primera parte donde se controla el motor Y, el motor X y el automatismo.

- Una segunda parte donde se controla el motor Z y el motor del cabezal.

Estos dos programas han sido comunicados a través de Ethernet, utilizando el protocolo de comunicación TCP.

ÍNDICE DE DOCUMENTOS

1.- MEMORIA

2.- AÑEXO DE DISEÑO

3.- ANEÑO DE PROGRAMACIÓN

4.- MANUAL DE USUARIO

5.- PRESUPUESTO

6.- PLIEGO DE CONDICIONES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO DE UNA APLICACIÓN PARA LA IDENTIFICACIÓN Y CONTROL DE UN MOTOR DE CORRIENTE CONTINUA MEDIANTE LABVIEW

Documento Nº1: Memoria

AUTOR: Dotor Alonso, Luis

TUTOR: Herrero Durá, Juan Manuel

Curso Académico: 2013-14

CONTENIDO

1.- OBJETO DEL PROYECTO 2
2.- OBJETIVOS DEL PROYECTO 3
3.- MOTIVACIÓN 3
4.- ANTECEDENTES DEL PROYECTO 4
4.1.-Introducción y descripción del equipo 4
4.1.1.- Descripción del fabricante Feedback 4
4.1.2.- Tarjeta de adquisición de datos 7
4.1.3. - Fuente de alimentación 7
4.1.4.- Montaje 8
4.2.- Control por Computador 10
4.2.1.- Introducción 10
4.2.2.- Aplicaciones 10
4.2.3.- Descripción del software utilizado 11
4.2.3.1.- LabVIEW: Introducción y justificación 11
4.2.3.2.- Matlab: Introducción 13
4.2.3.2.1.- System Identification ToolBox 14
4.2.3.2.2.- Rltool 14
4.2.3.2.3.- Simulink 15
5.- DESCRIPCIÓN DE LA SOLUCIÓN OBTENIDA 16
5.1.- Identificación de los modelos matemáticos 16
5.1.1.- Preparación para la identificación 16
5.1.1.1.- Diseño del sensor virtual 16
5.1.1.2.- Diseño del filtro 18
5.1.1.3.- Diseño del PID 19
5.1.2.- Realización de los ensayos 20
5.1.3.- Obtención del modelo y su validación 20

5.2.- Diseño de los reguladores 21
5.2.1.- Obtención de los reguladores 21
5.2.2.- Validación de los reguladores 23
5.3.- Diseño del automatismo del proceso 24
5.3.1.- Descripción del automatismo 25
5.4.- Diseño gráfico 26
5.4.1.- Modelado 3D del robot 26
5.4.2.- Diseño de la gráfica XY 28
5.4.3.- Diseño de Leds 30
6.- CONCLUSIONES 31
7.- FUTURAS MEJORAS 32
8.- BIBLIOGRAFÍA 33

1.- OBJETO DEL PROYECTO

El objeto que se está proyectando es un software para automatizar y controlar un robot diseñado para operaciones de taladrado de precisión en objetos como tablas de madera o placas de circuitos impresos. En este proyecto no existe la maqueta del robot como tal, no obstante, se ha simulado su comportamiento con cuatro motores de corriente continua de la casa Feedback. Estos cuatro motores representan el robot.

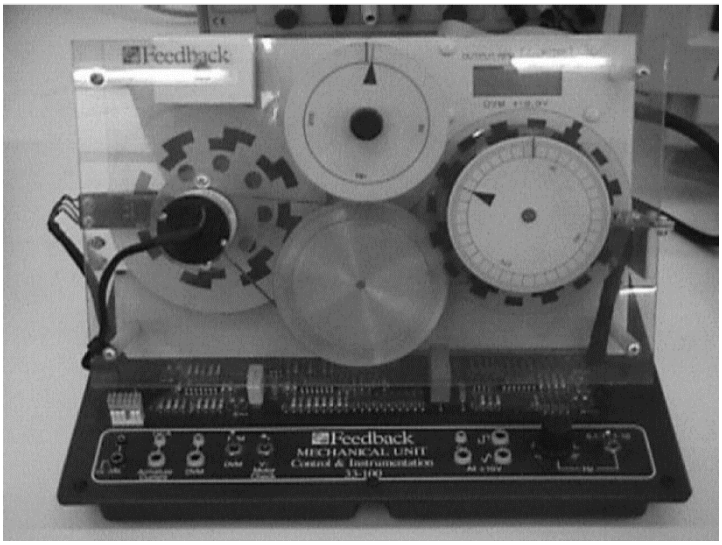


Figura 1: Maqueta del motor.

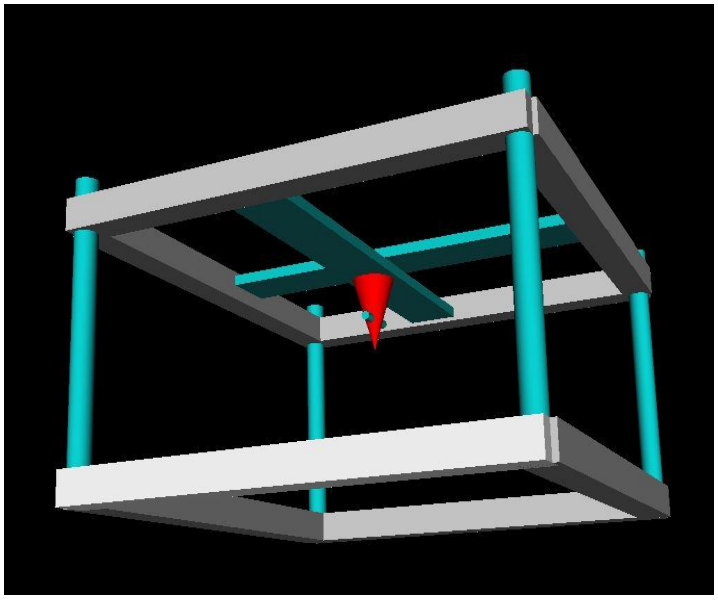


Figura 2: Modelo 3D del robot.

2.- OBJETIVOS DEL PROYECTO

El objetivo principal de este proyecto es el desarrollo de una aplicación en LabVIEW capaz de controlar los cuatro motores del robot y automatizar todo el proceso de taladrado, no obstante, existen una serie de objetivos que se han planteado a largo de todo el proceso de desarrollo:

- Programación del algoritmo de un PID mediante las ecuaciones necesarias, que sirva de regulador en los cuatro controles del proyecto.
- Diseñar un control de posición sin sobreoscilación ni error de posición, para posicionar tres de los cuatro motores en las coordenadas correspondientes del espacio (X, Y, Z). El diseño ha de incluir la obtención del modelo matemático del proceso y el cálculo del regulador.
- Diseñar un control de velocidad sin error de posición para poder alcanzar la velocidad adecuada en el supuesto proceso de taladrado. El diseño ha de incluir la obtención del modelo matemático del proceso y el cálculo del regulador.
- Diseño de un modelo 3D dinámico, capaz de reproducir los movimientos de los motores y representar el robot.
- Representación del itinerario de puntos de trabajo en un gráfico cartesiano, para monitorizar el proceso de automatización.
- Asentar todos los conocimientos adquiridos en las asignaturas cursadas en el Grado en Tecnologías Industriales (Sistemas automáticos, Tecnología automática, Laboratorio de automatización y control).

3.- MOTIVACIÓN

La gran motivación que reside en este proyecto, es fruto del conocimiento adquirido en LabVIEW. Esta plataforma permite desarrollar programas y realizar auténticas filigranas mediante un lenguaje de programación gráfico muy potente. Además, existe motivación de cara a la aplicación práctica de conocimientos adquiridos en otras asignaturas.

4.- ANTECEDENTES DEL PROYECTO

4.1.-Introducción y descripción del equipo

En el desarrollo del presente proyecto se ha hecho uso del siguiente material:

- Cuatro motores de la casa Feedback (servosistema digital de corriente continua 33-002).
- Dos tarjetas de adquisición de datos con sus respectivas borneras.
- Dos ordenadores para el control de los cuatro motores.
- Cuatro fuentes de alimentación.

Ha sido necesaria la utilización de dos tarjetas de adquisición de datos, debido a que cada una de ellas únicamente posee dos salidas analógicas y, por consecuencia, dos ordenadores: uno por tarjeta.

4.1.1.- Descripción del fabricante Feedback

El servosistema digital de corriente continua 33-002 es un motor con el que se ha establecido la comunicación a través de un cable de 34 pines. Es un prototipo que no requiere demasiada instrumentación, únicamente una fuente de alimentación y una regleta que permita el acceso a la información proporcionada por el cable anterior.

La unidad mecánica de este motor contiene un amplificador de potencia que tiene como principal misión excitar al motor a partir de una entrada analógica. El prototipo posee una serie de elementos mecánicos y electrónicos, los cuales se listan a continuación:

- Eje del motor: lleva un disco de freno, junto con una pista de velocidad de 2 fases y el tacogenerador.
- Eje de salida: éste incorpora una pista de medida angular digital y un potenciómetro, el cual proporciona una tensión de ± 10 voltios. Además, este eje está comunicado con el eje del motor a través de una correa de reducción de 32:1.
- Eje de entrada: lleva acoplado un potenciómetro que proporciona señales en el margen de ± 10 voltios y la escala correspondiente.

- Un disco de freno y un freno magnético: lo podemos accionar manualmente mediante una palanca, con el fin de simular una perturbación.
- Un generador de señales sencillo: permite generar hasta 3 tipos de ondas de baja frecuencia: senoidal, cuadrada y triangular. No obstante, para ello necesita una fuente de alimentación externa que proporcione tensiones de 15, 0,-15 voltios a 1.5 amperios y 5, 0 voltios a 0.5 amperios.
- Un display: el cual proporciona la lectura directa de la velocidad del eje de salida en r.p.m. en un rango de 0 a 99 r.p.m. Como la relación de trasmisión es de 32:1, la velocidad del motor en el eje de entrada, es de hasta 3200 r.p.m.
- Pistas y lectores de velocidad, las cuales proporcionan señales de onda cuadrada de 0.5 voltios (8 ciclos por vuelta).
- Un conmutador: para comprobar el arranque inicial del motor.
- Señal de corriente de excitación: se trata de una onda de tensión que comunica la corriente de excitación mediante una escala de 1V/A.
- Conmutador de rango y frecuencia de señal de prueba.
- Lectores y medidas digitales: son pistas que comunican información en código Gray de 16 bits y son leídas mediante lectores de infrarrojos.
- Pulso índice: se trata de un pulso que proporciona información sobre la referencia del eje de salida.

A continuación, en la figura 3 se pueden observar algunos de los elementos anteriormente descritos, además, en la figura 4 se contemplan los elementos de manipulación de prototipo:

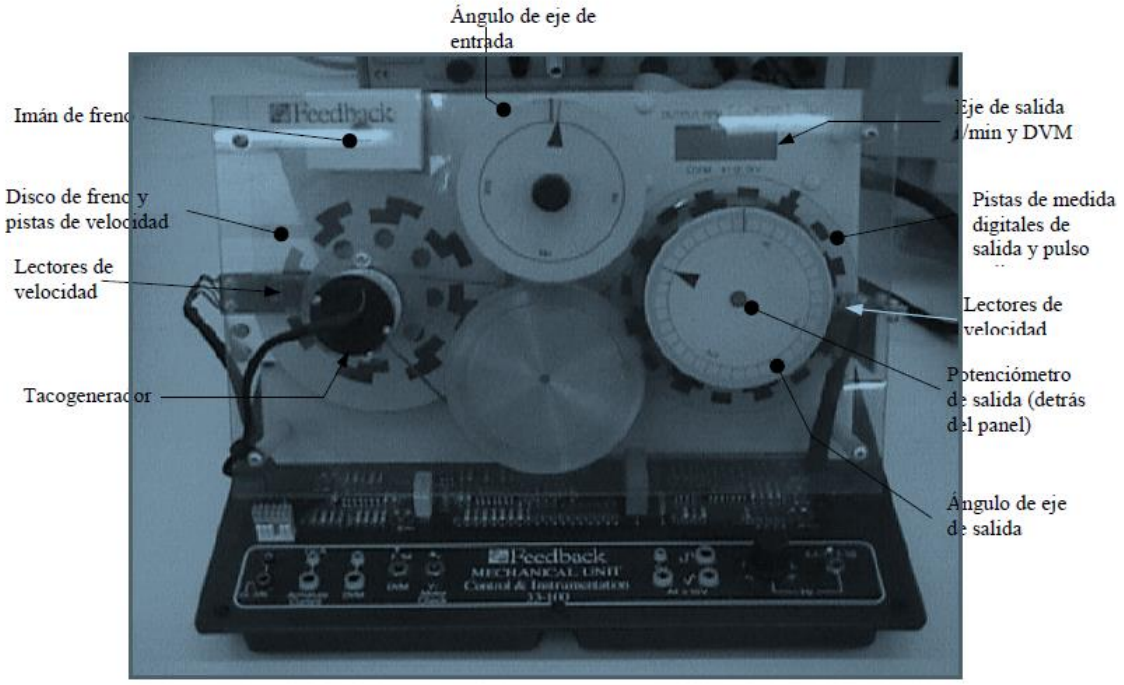


Figura 3: Elementos de la maqueta.

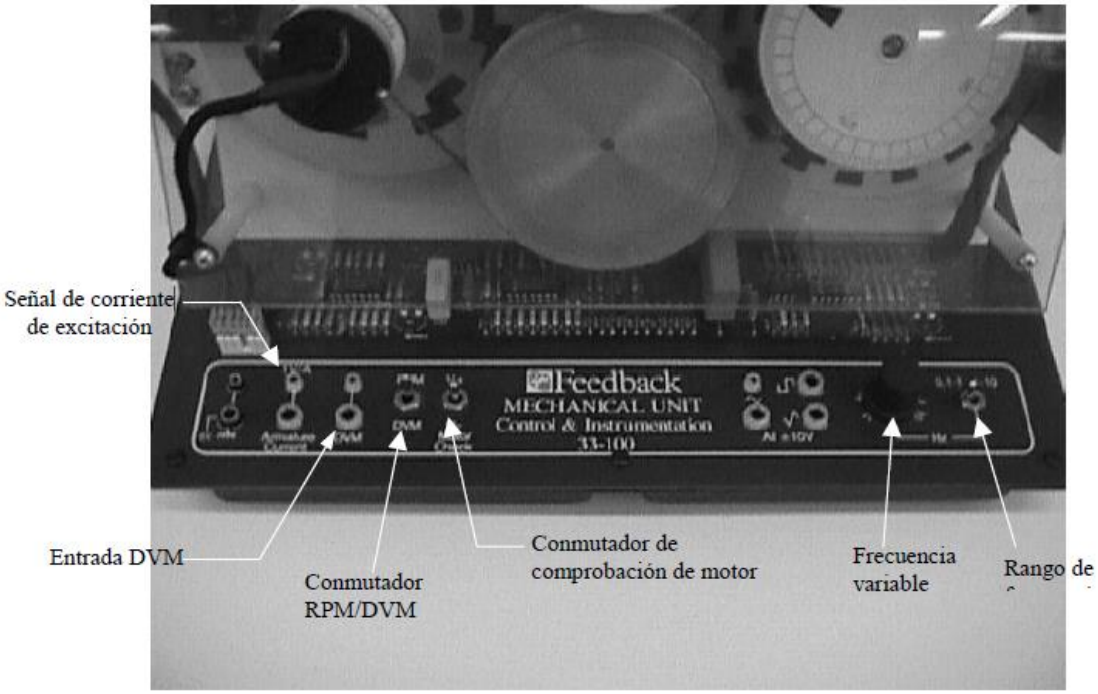


Figura 4: Elementos de manipulación de prototipo.

4.1.2.- Tarjeta de adquisición de datos

La tarjeta de adquisición de datos es un elemento muy importante, ya que nos permite comunicar el motor (proceso) con el ordenador, donde está la base del cómputo. Esta tarjeta de adquisición está instala (pinchada) en la placa base del ordenador y se conecta con el motor a través de la bornera de conexiones.

La tarjeta de adquisición de datos utilizada (AD-Link PCI 9112) posee:

- Seis entradas analógicas.
- Dos salidas analógicas.
- Cinco salidas digitales.

A pesar de que la tarjeta de adquisición de datos posee únicamente 2 salidas, la bornera de conexiones dispone de 4 conexiones de salida. Dos de éstas salidas están en el rango de ±10V y las otras simplemente en el intervalo de 0-10V. En el caso de los motores, se han utilizado las salidas de ±10V.

4.1.3. - Fuente de alimentación

Para poder suministrar energía a cada uno de los motores del robot, se han tenido que utilizar cuatro fuentes de alimentación. Estas fuentes han de proporcionar tensiones de 15, 0,-15 voltios a 1.5 amperios y 5, 0 voltios a 0.5 amperios. Cada uno de los motores incorpora una serie de cables para realizar estas conexiones. En la Figura 5 se puede observar el tipo de fuente de alimentación utilizada.

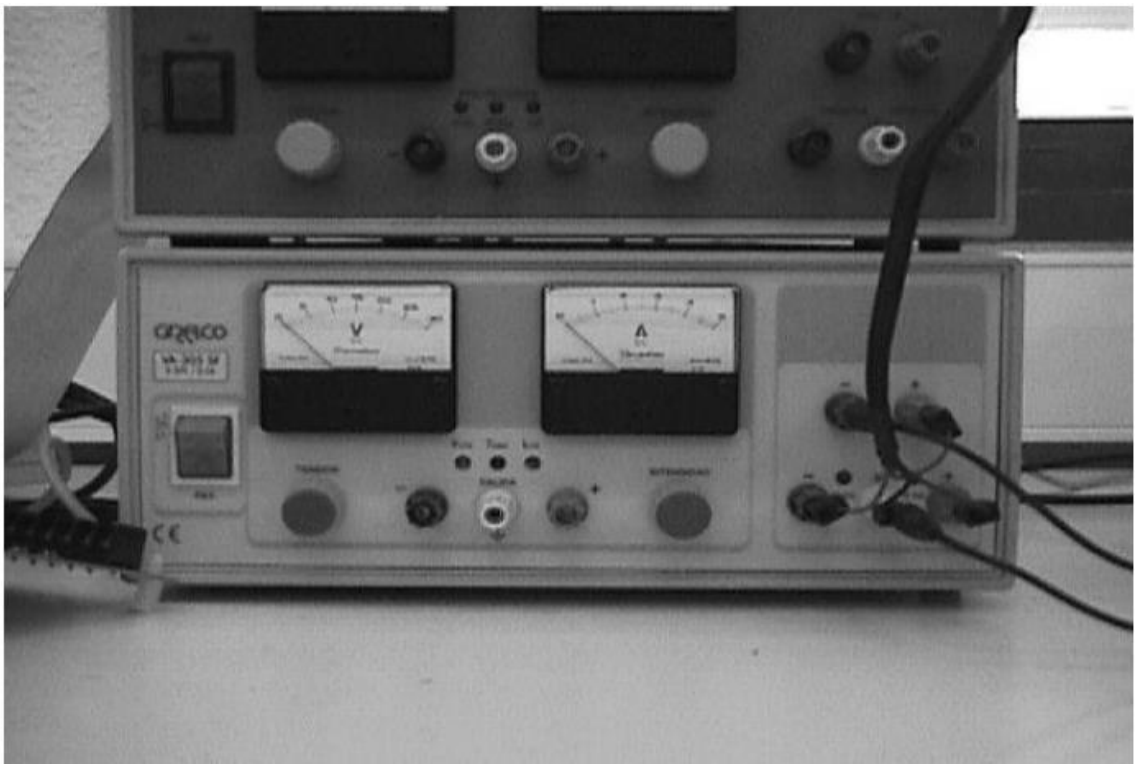


Figura 5: Fuente de alimentación.

4.1.4.- Montaje

El motor está comunicado mediante el cable de 34 pines a una pequeña placa con cada una de las conexiones necesarias para interactuar con las entradas y salidas del motor. A continuación se nombran cada una de las conexiones que ofrece:

- Dispone de una entrada para escribir tensión en el motor. Esta tensión la proporciona la fuente de alimentación.
- Una conexión a masa para establecer un mismo origen de potenciales en todos los circuitos implicados.
- Una salida para leer tensión devuelta por el tacogenerador del motor, que permite medir la velocidad.
- Una salida para leer la tensión devuelta por el potenciómetro, para el caso en que se quiera fijar la referencia del motor manualmente.
- Una salida para leer la tensión devuelta por el sensor del eje del motor.

A continuación, se muestra una imagen de la placa con sus conexiones.

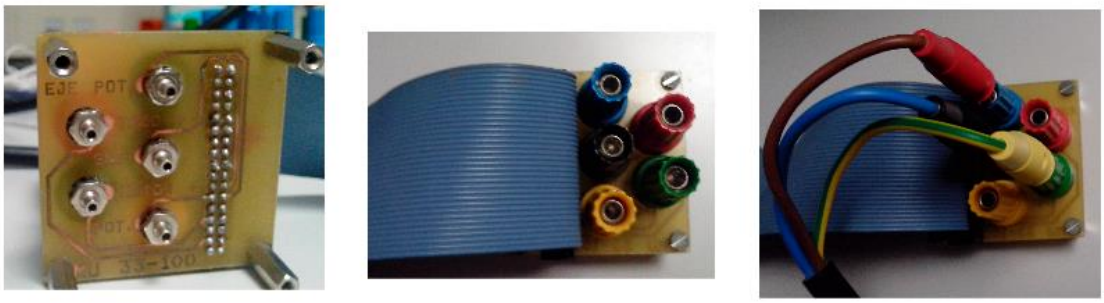


Figura 6: Placa de conexiones.

Para el cableado total de un motor se han utilizado tres cables gracias a los cuales se comunica la bornera con la placa de conexiones, además del cable de 34 pines que comunica la bornera con el ordenador y otro cable de 34 pines que comunica la placa con el motor. A continuación se muestra un esquema con el cableado total de un motor:

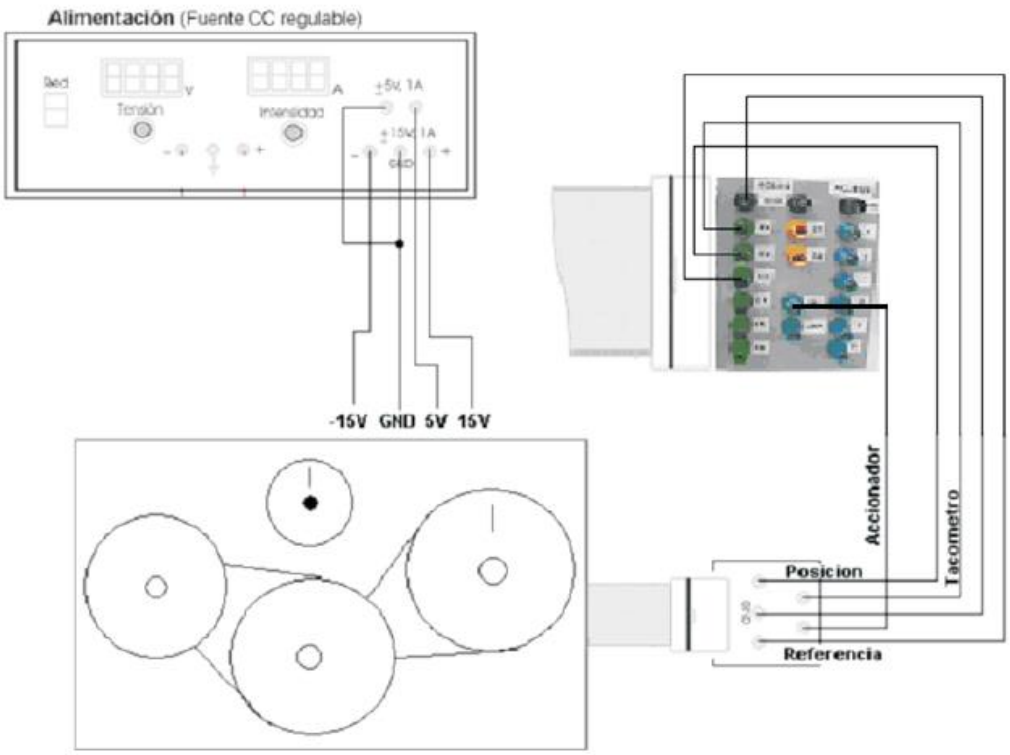


Figura 7: esquema de conexión.

4.2.- Control por Computador

En el presente apartado se va realizar una breve introducción al control por computador y a sus objetivos, además se describirán cada uno de los softwares utilizados en el proyecto.

4.2.1.- Introducción

La aplicación de ordenadores, o computadores, con alta capacidad de cómputo en los procesos industriales con el fin de controlar procesos, ha supuesto un gran salto tecnológico, ya que se ha traducido en la implementación de nuevos sistemas de control en el entorno industrial. Desde la perspectiva de las distintas teorías de control automático, el computador no se limita solo a realizar el cálculo de los reguladores analógicos, sino que también permite la implantación de algoritmos de control más avanzados y complejos. En un principio, el objetivo del control por computador era sustituir y mejorar los reguladores analógicos, no obstante, esta ambición se fue ampliando debido a las crecientes capacidades de los computadores. En la actualidad, se realiza un control integrado en las plantas de fabricación englobando también la gestión de la producción.

4.2.2.- Aplicaciones

Hoy en día los objetivos y las aplicaciones industriales de un computador en el control son muchas, a continuación se describen las más importantes:

- Adquisición de datos: esta función consiste en la recogida, tratamiento y almacenamiento de datos.
- Supervisión: Cuando un computador realiza este tipo de función, se conecta a sistemas que controlan el proceso (tales como autómatas PID, reguladores…), generalmente por medio de una red de comunicación. El objetivo principal de esta función es facilitar o ayudar al operador del proceso o planta. El computador suministra la información necesaria a través de indicadores gráficos, como por ejemplo las alarmas.
- Control secuencial: en este tipo de control el ordenador toma el papel de un autómata programable, secuencia cada una de las tareas que tiene que realizar el proceso.

- Control analógico digital: es una forma antigua de control en la que los computadores se encargaban de establecer la consigna de los bucles analógicos.
- Control digital directo: en este caso el computador ejecuta el control de un proceso continuo, interpretando el papel de un regulador industrial.
- Análisis de datos: esta es una función típica en los ordenadores de gestión, en la que los datos son analizados, generalmente, mediante un software de ofimática.

La mayoría de estas funciones o aplicaciones del control por computador aquí presentadas son llevadas a cabo por el software desarrollado en el proyecto.

4.2.3.- Descripción del software utilizado

A continuación, se van a presentar de manera breve los programas que han sido utilizados en el desarrollo del proyecto.

4.2.3.1.- LabVIEW: Introducción y justificación

LabVIEW (acrónimo de Laboratory Virtual Instrumentation Engineering Workbench) es una plataforma y entorno de desarrollo para diseñar sistemas con un lenguaje de programación visual gráfico. Este software fue diseñado por National Instruments en 1976 para el desarrollo de osciloscopios virtuales, no obstante, su gran potencial de programación le ha conducido a ser una plataforma recomendada para el desarrollo de sistemas (hardware y software) de control, diseño simulado o real. El lenguaje que utiliza este programa se denomina lenguaje G (Gráfico).

Las aplicaciones desarrolladas en LabVIEW reciben el nombre de instrumentos virtuales (VIs), ya que en un origen este programa fue diseñado para el control de instrumentos. No obstante, su aplicación se ha expandido no solo hacia la electrónica, sino también a la programación embebida.

En el actual proyecto se ha elegido desarrollar en la plataforma de LabVIEW, debido a que presenta numerosas ventajas con respecto a otros softwares y leguajes de programación. Algunas de estas ventajas se nombran a continuación:

- Resulta un lenguaje de programación sencillo pero a la vez potente, que resulta válido tanto para programadores con un alto nivel de conocimientos como para usuarios iniciados.

- Le permite realizar al programador inexperto aplicaciones complejas que le resultaría imposible desarrollar con otros lenguajes de programación tradicionales.
- Permite efectuar programas complejos de miles de Vis con decenas de miles de entradas y salidas, proyectos para combinar nuevos Vis, etc.
- Presenta facilidades para el manejo de Interfaces de comunicaciones: Puerto serie, puerto paralelo, GPIB, PXI, VXI, TCP/IP, Irda, Bluetooht, USB, OPC.
- Tiene la capacidad de interactuar con otros lenguajes de programación y aplicación: DLL (librerías de funciones), .NET, ActiveX, multisim, Matlab/Simulink, AutoCAD, SolidWorks, etc.
- Contempla un gran número de facilidades de cara a la edición, como herramientas dinámicas (gráficas), adquisición y tratamiento de imágenes, control de movimientos, modelado de objetos en 3D y control de los mismos en tiempo real.

Por último, cabe destacar que el programa presenta, a grandes rasgos, dos ventanas de edición: una primera llamada “Block Diagram” (diagrama de bloques), donde el usuario puede editar el código mediante un sinfín de posibilidades de programación y una segunda pantalla, llamada “Front Panel” (panel frontal), donde se edita toda la interfaz gráfica consecuente del código implementado. A continuación, se muestra una imagen con estas dos ventanas representado un ejemplo de la codificación de un interruptor con un led:

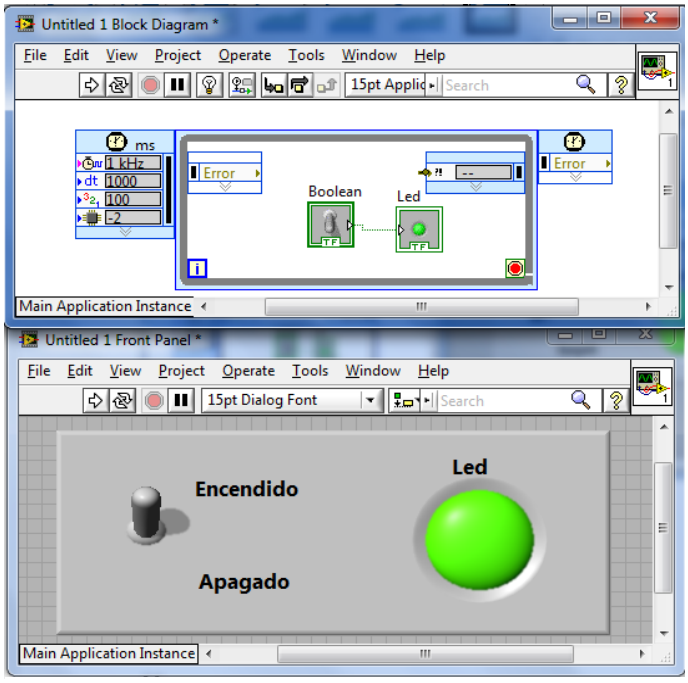


Figura 8: Interfaz de LabVIEW.

4.2.3.2.- Matlab: Introducción

Matlab es un software matemático utilizado internacionalmente en universidades y centros de investigación. Ofrece un entorno de desarrollo integrado y un lenguaje de programación propio basado en C. Este programa alberga una gran cantidad de funciones, opciones de manipulación de datos, posibilidades de comunicación con otros programas con lenguajes de programación distintos, etc. No obstante, lo que lo hace realmente interesante es su creciente expansión en el desarrollo de prestaciones para el programa: a saber Simulink, rltool y System Identification tool, etc.

En el actual proyecto se ha hecho uso de estas prestaciones para la obtención de modelos y para la validación de los mismos. A continuación, se ha realizado una breve descripción de cada una de las herramientas de la cuales se ha hecho uso.

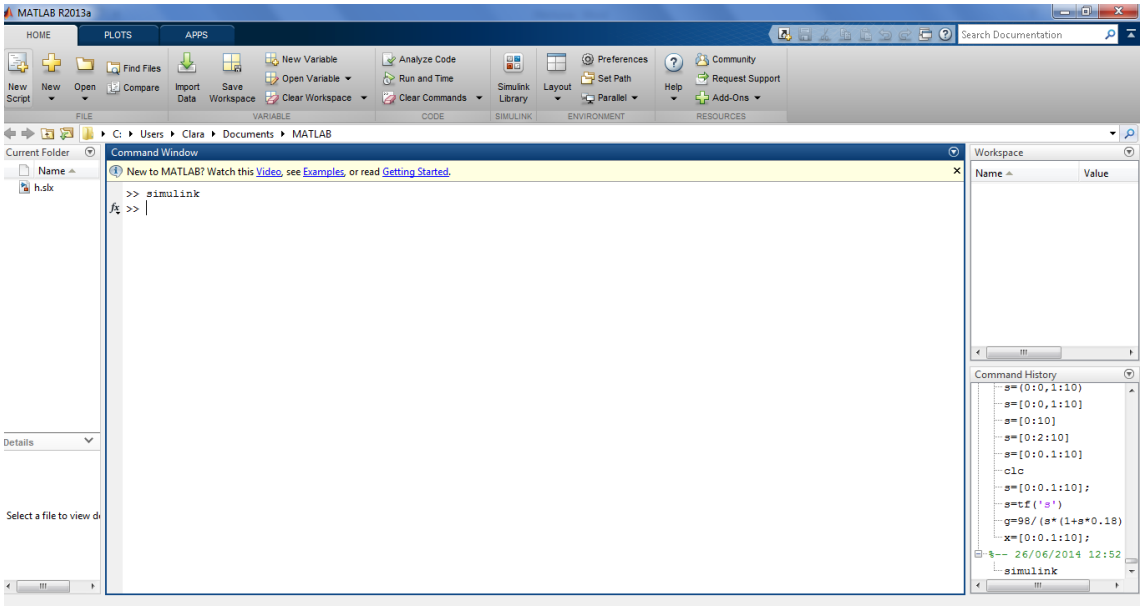


Figura 9: Interfaz de Matlab.

4.2.3.2.1.- System Identification ToolBox

Esta herramienta de Matlab permite obtener un modelo matemático de un sistema dinámico a partir de la información de sus entradas y salidas. El programa utiliza algoritmos de estimación a partir de una aproximación del modelo configurado por el usuario. Este programa ha sido utilizado en el proyecto para estimar un modelo matemático de velocidad y de posición del motor.

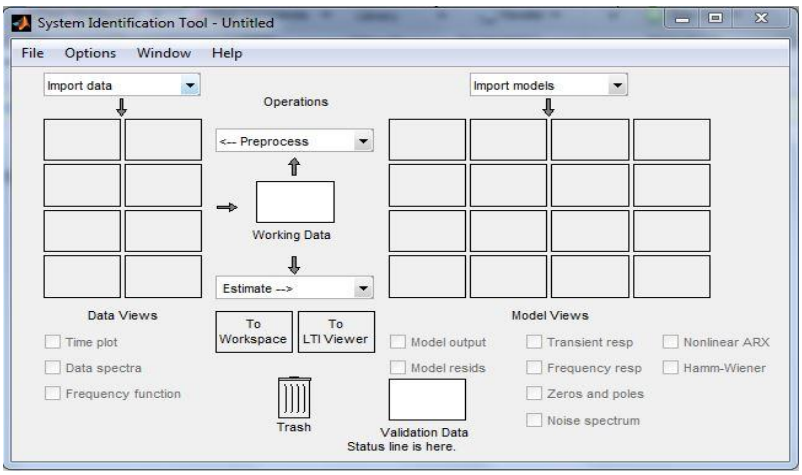


Figura 10: Panel Frontal del System Identification Tool.

4.2.3.2.2.- Rltool

Rltool es una herramienta de MATLAB que proporciona una interfaz gráfica de usuario para el análisis del lugar de las raíces de sistemas SISO (single input, single output), es decir, sistemas con un sola entrada y una sola salida. Esta herramienta permite, de una forma sencilla, obtener el diseño de reguladores y ver su influencia en el lugar de las raíces (lugar geométrico de los polos y ceros de una función de transferencia a medida que varía la ganancia) dibujado sobre el plano complejo. Gracias a esta herramienta se puede manipular la posición de cada uno de los polos y los ceros del regulador, así como su ganancia, de tal manera que se puede observar el conjunto de soluciones posibles para colocar los polos en bucle cerrado. A continuación, se muestra una imagen de la interfaz del programa.

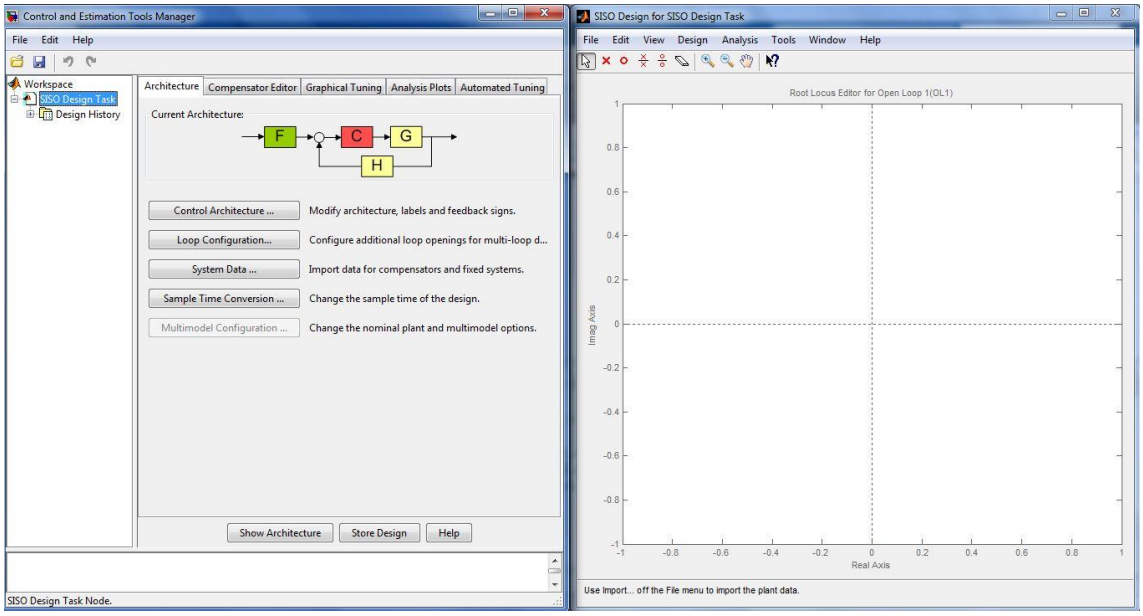


Figura 11: interfaz del rltool.

4.2.3.2.3.- Simulink

Esta herramienta se basa en un entorno de bloques para la simulación multidominio y el diseño basado en modelos. Posee cierto grado de abstracción de los fenómenos físicos involucrados en los sistemas. Se ha utilizado para la validación de los reguladores, previamente obtenidos con el rltool. En la figura 12 se puede observar su interfaz gráfica.

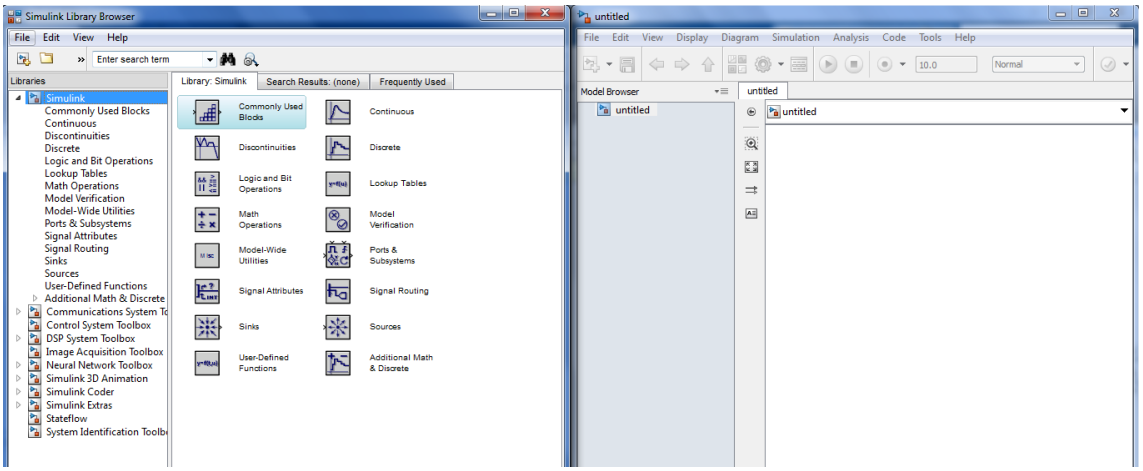


Figura 12: Interfaz gráfica de Simulink.

5.- DESCRIPCIÓN DE LA SOLUCIÓN OBTENIDA

5.1.- Identificación de los modelos matemáticos

Como la mayoría de los proyectos de control, se ha empezado por la identificación de los modelos del proceso. En este caso se han identificado dos modelos:

- Un modelo de posición del eje de salida del motor para controlar la posición de las tres coordenadas del espacio.
- Un modelo de velocidad para controlar la velocidad del motor del cabezal.

5.1.1.- Preparación para la identificación

Antes de empezar con la identificación, ha sido necesario obtener los ensayos pertinentes para poder estimar los modelos matemáticos, no obstante, antes de eso ha sido necesario:

- Realizar la parte del programa encargada de la obtención de datos (necesario para los dos modelos).
- Adaptar la salida del sensor de eje del motor previamente al ensayo (necesario para el modelo de posición).

5.1.1.1.- Diseño del sensor virtual

El sensor del eje de salida del motor devuelve una función que expresa su posición a lo largo de una vuelta. Esta función con forma triangular (Figura 13) no interesa nada para la identificación del modelo de posición, puesto que no expresa la posición del brazo robot, si no la posición del eje de un motor.

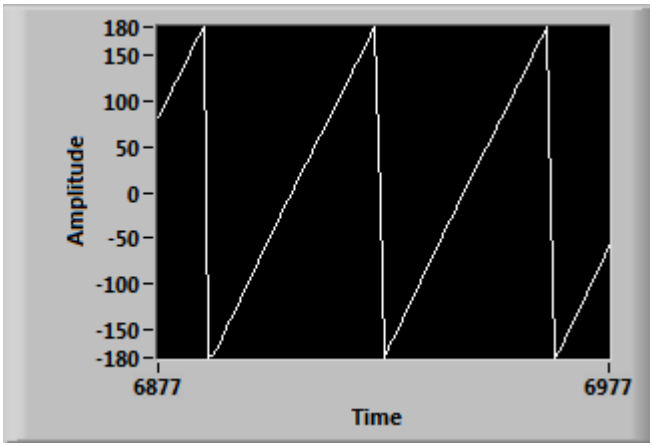


Figura 13: Posición del eje del motor.

Para poder representar la posición de cada eje del robot conforme los motores vayan dando vueltas, ha sido necesario implementar un algoritmo para simular un sensor virtual que contase en todo momento las vueltas dadas por el motor y que, de esta forma, modificase la salida del sensor real en función del número de vueltas, la posición leída del sensor real y la posición del sensor virtual guardada del instante anterior. Este algoritmo ha sido implementado en LabVIEW como un instrumento virtual con 3 entradas y 2 salidas, tal y como se muestra en la figura 14. Su codificación se muestra en el anexo de programación.

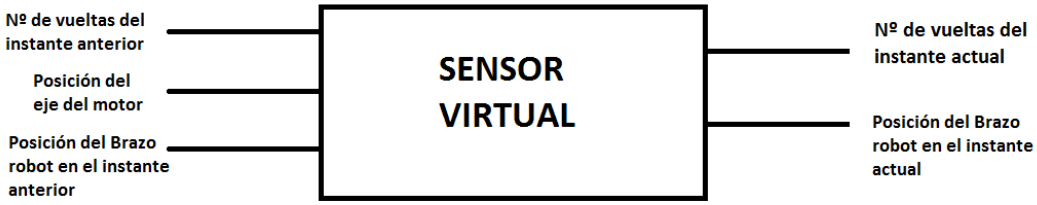


Figura 14: Salidas y entradas del sensor virtual.

Una vez se ha hecho pasar la onda triangular por este sensor virtual, se queda un resultado como el de la Figura 15:

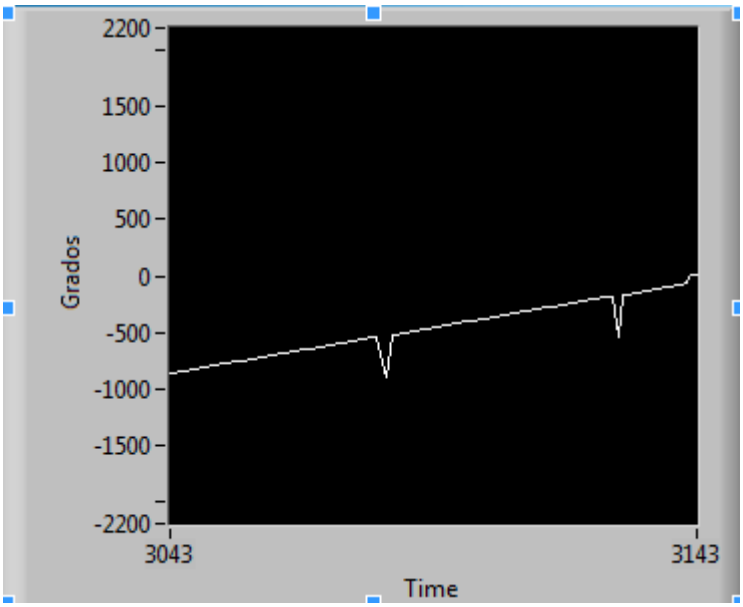


Figura 15: salida del motor después de pasar por el sensor virtual.

5.1.1.2.- Diseño del filtro

Como se puede observar en la Figura 15, la salida del sensor virtual presenta una discontinuidad, esto es debido a que el sensor pasa por un punto donde la tensión que devuelve pasa muy bruscamente de -10 voltios a 10 voltios y viceversa. Debido a que esta transición no es exacta, la salida presenta este pico.

Para eliminar esta discontinuidad, se ha hecho pasar la posición del robot por un filtro, el cual compara en cada instante de tiempo la posición medida con la posición del instante anterior, de tal manera que si la diferencia de ambas varía más de una cierta cantidad, la salida se acota a un cierto valor. En la Figura 16 se puede observar la posición después de haber pasado por el filtro (línea roja).

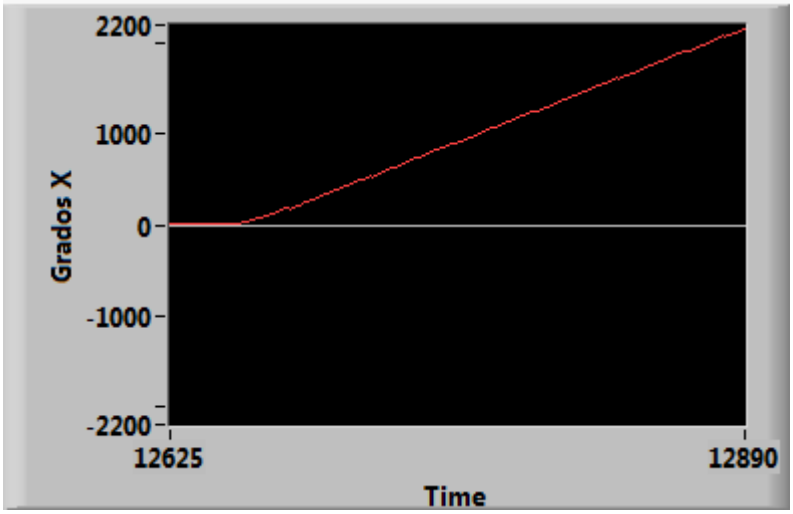


Figura 16: Posición del Brazo robot filtrada.

5.1.1.3.- Diseño del PID

A continuación, se ha realizado un instrumento virtual para implementar en LabVIEW un PID. Este paso no es estrictamente necesario de cara a la obtención de los ensayos, no obstante, se ha realizado a estas alturas del desarrollo con el fin de dejar terminada la estructura de programación del bucle de control. Este PID programado consta de dos modos de funcionamiento:

- El modo manual, donde el usuario controla la variable manipulada del proceso desde un control en LabVIEW. Este modo es el utilizado para obtener los ensayos.
- El modo automático, donde se ha programado la ecuación en diferencias de un PID.

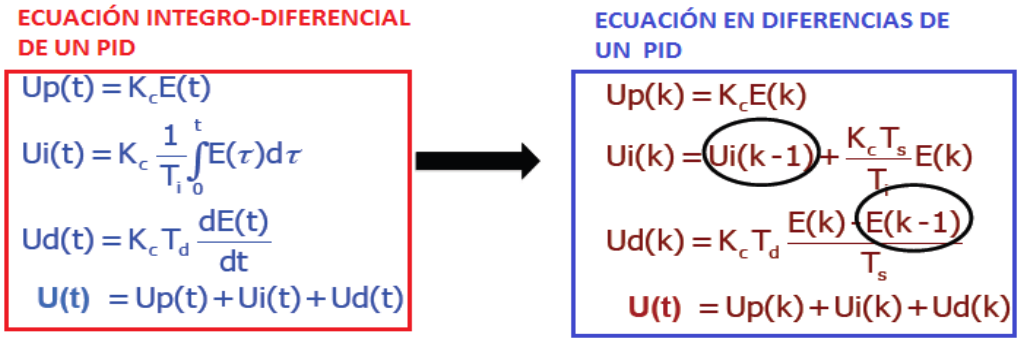


Figura 17: Ecuación integro-diferencial y en diferencias de un PID.

5.1.2.- Realización de los ensayos

Una vez se ha adaptado el programa, se ha procedido a la realización de los ensayos. Se han realizado dos ensayos distintos aplicando manualmente escalones sobre la variable manipulada y guardando los datos pertinentes en cada caso:

- Ensayo de posición: Se han guardado datos de tiempo, posición y tensión aplicada al motor.
- Ensayo de velocidad: se han guardado datos de tiempo, velocidad y tensión aplicada al motor.

5.1.3.- Obtención del modelo y su validación

Una vez obtenidos los ensayos, se han importado a la herramienta System Identification Tool, donde se han obtenido y validado varios modelos, tanto en el caso de velocidad como en el caso de posición. Posteriormente, se ha realizado una comparativa de los distintos modelos, eligiendo finalmente los que presentaban mayor porcentaje de aproximación en la validación:

- Modelo de posición, el cual ha sido aproximado por un primer orden con integrador:

$$G_p(s) = \frac{98.821}{(1 + 0.228 * s) * s} \frac{Grados}{V}$$

- Modelo de velocidad, el cual ha sido aproximado por un primer orden:

$$G_p(s) = \frac{29.472}{(1 + 0.176 * s)} \frac{rpm}{V}$$

Cabe destacar que este motor tiene un comportamiento bastante lineal (no al 100%) y, por tanto, los modelos obtenidos tienen una fiabilidad considerable.

5.2.- Diseño de los reguladores

5.2.1.- Obtención de los reguladores

Para la obtención de los dos reguladores se ha hecho uso de la herramienta rltool de Matlab, mediante la cual se ha obtenido el lugar de las raíces de ambos procesos. Previamente a este diseño, se han fijado una serie de especificaciones estáticas y dinámicas que han de ser cumplidas por los controles:

- El tiempo de establecimiento de los dos sistemas (posición y velocidad) ha de ser menor o igual a dos segundos ($T_{es} \leq 2seg$), siempre que no sature la acción de control.
- En cuanto a la sobreoscilación:
    - Control de posición: ha de ser nula para así evitar futuros problemas mecánicos o de otros tipos $\delta = 0\%$.
    - Control de velocidad: en este caso no es tan importante si la velocidad del motor excede durante un pequeño periodo de tiempo la velocidad deseada, no obstante se ha fijado la sobreoscilación $\delta \leq 4.3\%$.
- En cuanto al error de posición, se ha establecido que en ambos caso debe ser cero. Por una parte, en el control de posición podría dar problemas, ya que no se taladraría en el punto exacto del plano (algo indeseable). Por otro lado, un control de velocidad impreciso podría provocar malos resultados en la ejecución de los agujeros, debido al mal mecanizado.

Teniendo en cuenta las dos primeras especificaciones descritas anteriormente, aparece una región (zona de especificaciones dinámicas) en el plano complejo donde podemos situar los polos en bucle cerrado.

Teniendo en cuenta las especificaciones descritas anteriormente para cada uno de los controles y teniendo el modelo aproximado para cada uno de los procesos, se ha optado por la utilización de los siguientes reguladores:

- Control de Posición: se ha utilizado un regulador PD, con el fin de evitar la sobreoscilación. Al estar controlándose un sistema de primer orden con integrador, no ha de preocupar el error de posición, ya que, gracias a este integrador, el error bajo cambios en el punto de consigna será cero.
- Control de velocidad: se ha optado por un regulador PI con el fin de eliminar el error de posición. En este caso se está controlando un sistema de primer orden, por tanto, hace falta incluir un integrador para garantizar error de posición cero. No obstante, tampoco aparecerá oscilación en el control, puesto que al tratarse de un modelo sencillo se ha conseguido situar los polos en bucle cerrado dentro del eje real.

El diseño de ambos reguladores se ha realizado por cancelación, es decir, cancelando un polo del proceso con el cero del regulador. Una vez se ha obtenido el conjunto de soluciones posibles, se ha calculado una ganancia para colocar los polos en el lugar deseado. Finalmente, se han obtenido los siguientes reguladores:

- Regulador para el control de posición:

$$G_r(s) = 0.5(s + 0.23)$$

- Regulador para el control de velocidad:

$$G_{r(s)} = \frac{2.5(s + \frac{1}{0.18})}{s}$$

5.2.2.- Validación de los reguladores

Una vez obtenidos cada uno de los reguladores en el Rltool se ha procedido a su validación en Simulink, junto con la validación experimental en LabVIEW. Para ello se ha construido un bucle de control en Simulink mediante bloques y se ha simulado el control bajo un cambio de tipo escalón en la referencia. Analizando los resultados, se han obtenido las siguientes conclusiones:

- En el control de velocidad: se ha simulado el control en Simulink y se ha contrastado con el control experimental, llegando a la conclusión de que el control simulado no refleja al 100% el control real. Esto se debe a que el modelo obtenido es una aproximación y, por tanto, el diagrama de Simulink no tiene en cuenta la zona muerta del motor, en la cual se han obtenido los ensayos. Además, tampoco tiene en cuenta el control anti-Windup (implementado en el control real). Finalmente se ha llegado a la conclusión, mediante validación experimental, de que el control es adecuado, ya que cumple con las especificaciones del control.
- En el control de posición: se ha simulado el control de la misma forma que en el caso anterior. En cuanto al error de posición y a la sobreoscilación, se ha observado que desaparecen en la simulación, no obstante, el tiempo de establecimiento del sistema no concuerda con el teórico (marcado por la parte real del polo), esto es debido a que el control de posición que se ha implementado es agresivo y, por tanto, hace saturar a la acción de control. A pesar de lo dicho anteriormente, este comportamiento sigue siendo deseado ya que interesa que los motores X e Y se muevan a sus respectivas posiciones a la máxima velocidad posible, cosa que no se podría conseguir sin la máxima acción de control.

Esta validación se ha justificado con detalle en el anexo de diseño mediante las gráficas correspondientes y los diagramas de bloques utilizados.

5.3.- Diseño del automatismo del proceso

En el siguiente apartado se va a explicar de manera sencilla el automatismo implementado en el software, no obstante, su codificación se desarrolla detalladamente en el anexo de programación. El objetivo de esta automatización es actuar a modo de PLC para controlar secuencialmente el robot. Este automatismo ha sido implementado en Labview, no obstante, se ha realizado un diagrama para entender mejor su funcionamiento:

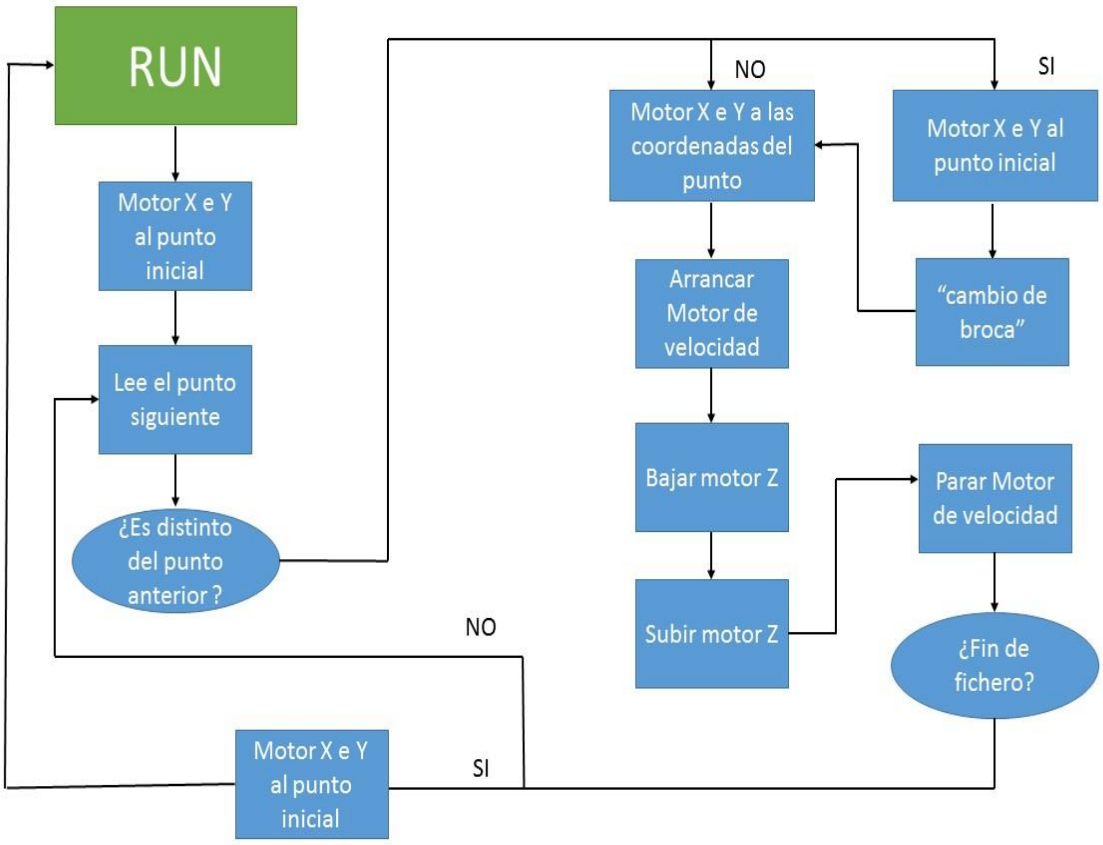


Figura 18: Diagrama de funcionamiento del automatismo.

5.3.1.- Descripción del automatismo

Se ha diseñado un automatismo que controla un robot para el taladrado de precisión de placas. El automatismo debe, a partir del fichero de datos proporcionado por el usuario, realizar los puntos especificados en el mismo.

A continuación, se describen las especificaciones del automatismo:

- El fichero debe poseer un estructura de matriz con tres columnas, donde la primera y la segunda columna sean la coordenadas X e Y, respectivamente, que han de seguir los motores. La tercera y última columna contendrá un número entero entre el 1 y el 5, el cual especificará el tipo de agujero que se quiere realizar. Para garantizar un correcto funcionamiento del automatismo, la primera fila del fichero será la 0, 0, 0. En la Figura 19 se muestra un ejemplo de la estructura del fichero:

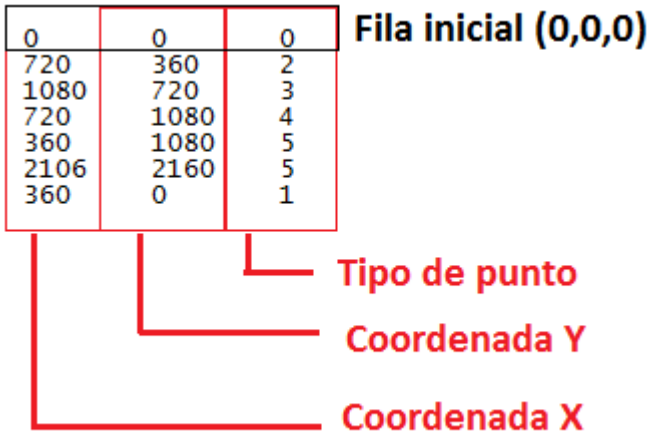


Figura 19: Estructura del fichero de datos.

- El automatismo se iniciará cuando se apriete el botón RUN. Acto seguido, los motores X e Y se moverán a las posiciones marcadas por las sucesivas filas del fichero, que en el caso del inicio será siempre la posición (X=0, Y=0).
- Cada vez que el automatismo lea un punto nuevo, deberá comparar el tipo de punto leído con el tipo de punto anterior, de manera que si son distintos tendrá que volver a la posición X=0 e Y=0 para “cambiar el tipo de broca”. En caso de que el tipo de punto sea el mismo que el anterior, los motores deberán situarse en las coordenadas del punto sin pasar por cero.

- Una vez que los motores X e Y están en las coordenadas correspondientes, se arrancará el motor del cabezal, el cual se pondrá a velocidades diferentes en función del tipo de punto. Si el punto es de tipo 1 o 5 girará a 120 rpm, en cambio, si el tipo de punto es 2, 3 o 4 girará a 80 rpm. En ambos casos se deberá esperar 6 segundos para que el motor alcance la velocidad adecuada.
- Acto seguido, el motor Z deberá bajar para realizar el agujero, no obstante, la distancia que este motor debe bajar también depende del tipo de punto. Si el punto es de tipo 1, 2, 3 o 4 el motor Z girará 720 grados, en cambio, si el punto es de tipo 5 el motor girará 360. En ambos caso se esperará 4 segundos desde el momento en que se le da la orden al robot, ya que se supone que es tiempo suficiente para que alcance esa posición y taladre. Tras estos 4 segundos el motor del cabezal deberá volver a la posición Z=0.
- Una vez el automatismo haya terminado de leer todos y cada uno de los puntos, deberá volver a la posición de cero y esperar a que se le vuelva a dar a RUN para que vuelva a arrancar el automatismo.
- En caso de que se apriete el botón de stop o se apriete de nuevo el botón RUN, el automatismo deberá pararse inmediatamente. Se supone que el operario encargado reiniciará el programa para que, cuando se vuelva a arrancar, los valores se restauren por defecto.

Se ha conseguido que el automatismo cumpla con todas las especificaciones expuestas anteriormente. Como bien se ha dicho al inicio de este apartado, en el Anexo de programación se analiza este diseño más profundamente a nivel de código en LabVIEW.

5.4.- Diseño gráfico

5.4.1.- Modelado 3D del robot

Para finalizar el documento, se ha hecho una breve descripción del objeto 3D modelado en LabVIEW. Como ya se ha dicho en la introducción del anexo actual, este apartado consta de una gran parte de programación que se explicará detalladamente en el anexo de programación.

El robot se ha modelado a partir de figuras con geometrías simples, desplazando unas piezas respecto de otras y cambiando cuidadosamente sus ejes de rotación, es decir, no se han utilizado operaciones de extrusión, ni chaflanes, etc. Hay que tener en cuenta que este modelo es puramente un boceto de lo que podría ser una representación real del robot, sin embargo, no se ha realizado ningún tipo de análisis cinemático ni dinámico.

El robot consta de una serie de elementos cuyas dimensiones han sido expresadas en u.l. (unidades de longitud) ya que no se trata de un prototipo real, sino de una mera representación.

De manera básica, se ha estructurado el robot en los siguientes elementos:

- Una base que soporta la estructura del robot formada por cuatro prismas rectangulares de dimensiones X=2u.l, Y=0.15u.l y Z=0.15u.l.
- Una segunda base que soporta los brazos X e Y, además del cabezal, formada por cuatro prismas rectangulares de dimensiones X=2u.l, Y=0.14u.l y Z=0.14u.l.
- Un brazo movido por el motor X, encargado de situar el cabezal en la coordenada X correspondiente. Este brazo tiene unas dimensiones X=0.25u.l, Y=2u.l y Z=0.05u.l.
- Un brazo movido por el motor Y, encargado de situar el cabezal en la coordenada Y correspondiente. Este brazo tiene unas dimensiones X=1.8u.l, Y=0.25u.l y Z=0.05u.l.
- Un cabezal cónico que representa el elemento taladrador, el cual está atravesado por un pequeño cilindro. Este cilindro tiene como función que se pueda apreciar el giro del cabezal en la simulación. El cabezal tiene una altura de 0.35u.l y un radio de 0.1u.l.
- Por último, cuatro cilindros empotrados a la base del soporte, que representan las guías a través de las cuales se desplaza la segunda base y con ella los dos brazos y el cabezal. Estos cilindros tienen una altura de 1.25u.l y dos bases de radio 0.05u.l.

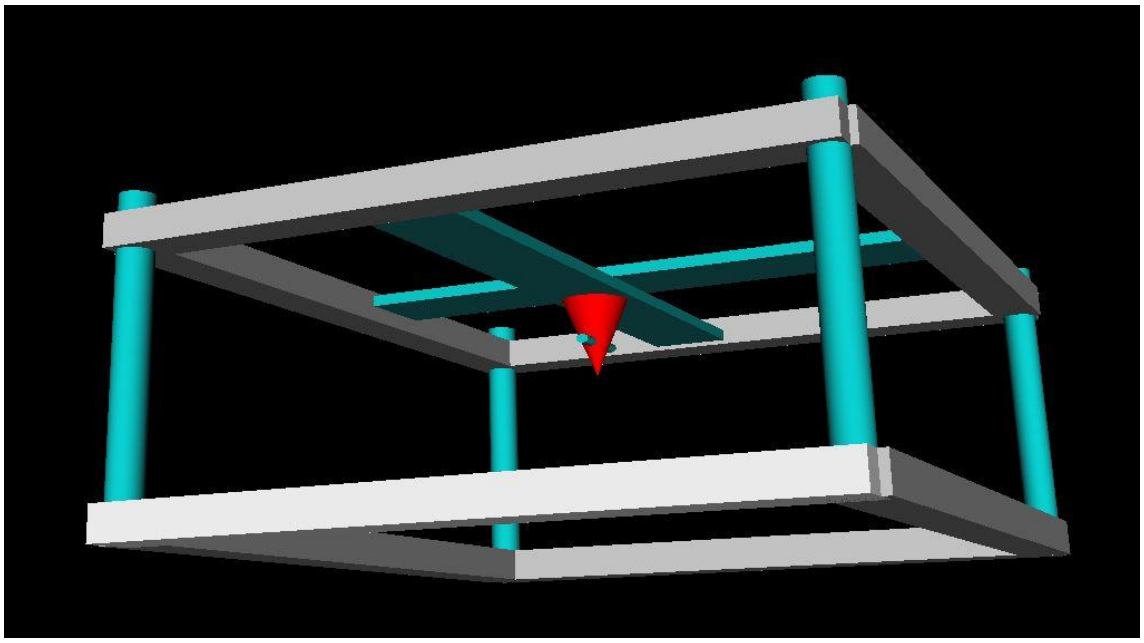


Figura 20: Modelo 3D del Robot.

5.4.2.- Diseño de la gráfica XY

Con el fin de una monitorización y control del recorrido del robot, se ha implementado una gráfica cartesiana en el programa capaz de representar información procedente de tres fuentes distintas. Esta gráfica permite que el usuario visualice en un plano las coordenadas X e Y de los distintos puntos de trabajo. En el transcurso del automatismo, la información mostrada por la gráfica aparece de la siguiente manera:

1. Al arrancar el fichero, se representan instantáneamente cada uno de los puntos de trabajo que el robot ha de ejecutar. Estos puntos se han representado mediante una cruz de color blanco.
2. Acto seguido, el automatismo empieza a leer el fichero, dirigiéndose de manera programada a los sucesivos puntos de trabajo, de tal forma que el punto que se está ejecutando aparece coloreado por un cuadrado azul, proporcionando así una idea de la posición del robot.
3. Por último, cada vez que un punto se ha ejecutado y el programa lo detecta, hace que la gráfica XY lo coloree mediante un cuadrado verde, indicando así el número total de puntos ejecutados. Con esta información el usuario puede saber a qué altura está el proceso, estimando así el tiempo restante para que finalice el automatismo.

A continuación, se puede observar una imagen de la gráfica con la representación de un ensayo a medio ejecutar:

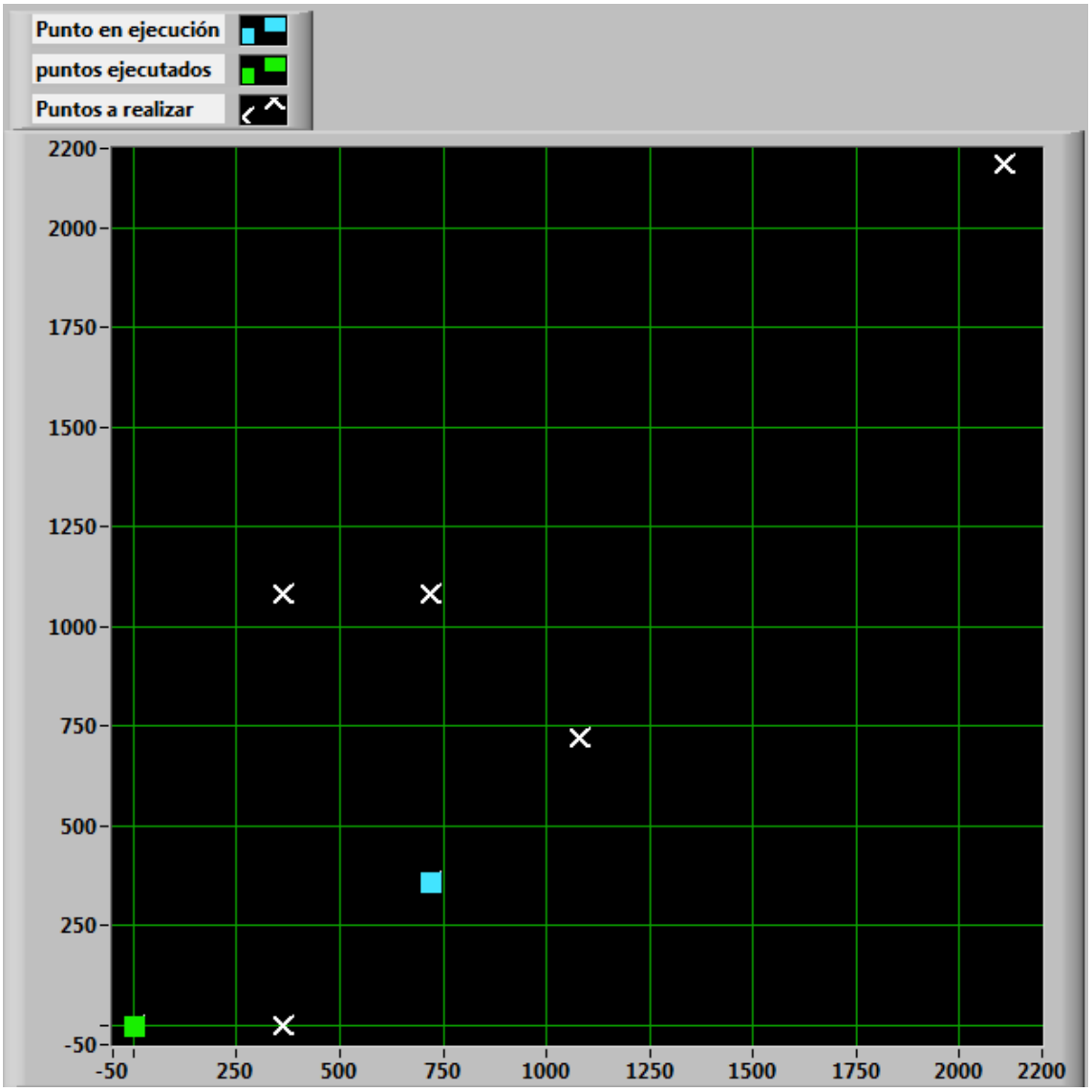


Figura 21: Gráfica XY de la aplicación de control.

5.4.3.- Diseño de Leds

Por último y para terminar con la parte gráfica, se han implementado un conjunto de leds que parpadean en tiempo real en función del tipo de agujero que se esté realizando. Tanto esta codificación como la de la gráfica XY están explicadas detalladamente a nivel de código en el anexo de programación.

Figura 22: Leds de la interfaz gráfica.

Como se puede observar en la Figura 22, los leds presentan unos títulos que especifican el tipo de agujero (Agujero Pasante, Rosca M12, Rosca M20, Rosca M50, Avellanado), no obstante, estos nombres son puramente un ejemplo y no implican absolutamente nada en el programa, ya que si el robot se destinase a operaciones de taladrado de placas para circuitos impresos, estos títulos no tendrían sentido. Por tanto, estos nombres dependen del tipo de aplicación final que se le dé al robot.

6.- CONCLUSIONES

Finalmente, después de haber realizado todo el desarrollo de la solución del proyecto, podemos concluir una serie de aspectos importantes:

- Se ha conseguido un control de posición preciso, el cual no presenta sobreoscilación ni error de posición, además de un tiempo de establecimiento adecuado.
- Se ha conseguido un control de velocidad sin error de posición y sin sobreoscilación, a pesar de no estar fijado en las especificaciones, y también con un tiempo de establecimiento adecuado.
- Se ha conseguido crear un instrumento virtual de un PID programado en el lenguaje en C. Este VIs sirve para cualquier tipo de regulador tipo PID que se quiera implementar en un futuro.
- Se ha conseguido crear el automatismo deseado en LabVIEW sincronizando cada uno de los cuatro motores del robot sin la necesidad de recurrir a un PLC externo. Se ha conseguido que el automatismo tenga una función de auto reseteo cada vez que termine de leer un fichero, de tal manera que se puede cargar un segundo fichero sin necesidad de reiniciar el programa.
- Se ha implementado el robot en 3D a partir de figuras con geometrías simples, consiguiendo así una representación mucho más directa y clara del objetivo de la aplicación.
- Se ha programado la gráfica XY mostrando en ella el transcurso y la ejecución en tiempo real de cada uno de los puntos de trabajo, de esta manera el usuario tendrá una idea más clara de lo que está ocurriendo en el proceso.
- Se ha conseguido implementar un conjunto de leds parpadeantes que aportan información del tipo de agujero realizado en tiempo real.
- En general, se ha conseguido crear una interfaz gráfica bastante completa y amena para el usuario, separada en distas partes (panel principal y configuración).
- Se ha conseguido profundizar, afirmar y poner en práctica gran parte de los conocimientos obtenidos en asignaturas anteriores.
- Por último, se ha logrado realizar el objetivo principal de este proyecto, que era el desarrollo de una aplicación para controlar cada uno de los motores de corriente continua, además de automatizar un proceso secuencial.

7.- FUTURAS MEJORAS

A pesar de ser una aplicación con bastante cierre, siempre hay cosas que se pueden mejorar con vistas al futuro, bien sea de cara a hacer un programa más completo, hacer más fácil la interactuación del programa con el usuario, mejorar y depurar el código, etc. En el presente apartado se presentan una serie de aspectos que resultaría interesante mejorar de cara a la continuación del proyecto, o a su mejora:

- Se podría implementar un quinto motor para la selección del tipo de "broca". Si el robot tuviera en su cabezal un cilindro que incorporase los distintos tipos de brocas y que éste permitiera seleccionar la broca correspondiente en función de un ángulo girado, se conseguiría que el robot no tuviera que pasar por el punto inicial para la preparación del próximo punto, ganando así tiempo de producción. La adición de un quinto motor hubiera implicado una tercera tarjeta de adquisición de datos, además de un tercer ordenador y una quinta fuente de alimentación.
- Otro aspecto importante que se podría mejorar es la depuración del código, de manera que el programa pudiese hacer lo mismo en menos operaciones. No obstante, se han utilizado un número considerable de horas en este trabajo.
- Al tratarse de un software desarrollado en un número limitado de horas y que ha sido utilizado por programadores que conocen su código, no ha sido necesario hacerlo robusto para determinadas puntualidades. De cara a la venta del software a un usuario inexperto en la programación, se podría realizar una mejora en su estabilidad para evitar posibles errores y colapsos del programa.

Todas estas ampliaciones y mejoras del programa no se han realizado por cuestiones de tiempo, ya que se trata de un proyecto con un número de horas limitadas.

8.- BIBLIOGRAFÍA

ISA. (2012-2013). *Identidicación de motor de corriente continua.* Valencia: UPV.

ISA. (2012-2013). *Unidad didáctica 5 de Sistemas Automáticos.* valencia: UPV.

ISA. (2013-2014). *Unidad didactica 2 de Tecnologia Automática.* Valencia: UPV.

ISA. (2013-2014). *Unidad didáctica 4 de Tecnologia Automatica.* Valencia: UPV.

Miquel, M. V. (2012-2013). *Servosistema Digital de Corriente Continua 33-002(SFT254).* Valencia.

Wikipedia. (s.f.). Obtenido de http://es.wikipedia.org/wiki/LabVIEW

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO DE UNA APLICACIÓN PARA LA IDENTIFICACIÓN Y CONTROL DE UN MOTOR DE CORRIENTE CONTINUA MEDIANTE LABVIEW

Documento Nº2: Anexo de Diseño

AUTOR: Dotor Alonso, Luis

TUTOR: Herrero Durá, Juan Manuel

Curso Académico: 2013-14

CONTENIDO

1.- INTRODUCCIÓN Y OBJETIVOS 2

2.- DESCRIPCIÓN DE LA SOLUCIÓN ELEGIDA 3

2.1.- Control de posición 3

2.1.1.- Identificación del modelo del proceso 5

2.1.1.1- Ensayo y obtención de los datos 6

2.1.1.2- Cálculo del modelo y validación mediante la herramienta ident tool de Matlab 7

2.1.1.3.- Importación de los datos 7

2.1.1.4.- Aproximación del modelo y cálculo 9

2.1.1.5.- Comparación de los resultados y elección del modelo 11

2.1.2.- Diseño del regulador 13

2.1.2.1.- Justificación de la elección de un regulador tipo PD 13

2.1.2.2.- Diseño del regulador mediante la herramienta rltool de Matlab 16

2.1.2.3.- Validación del regulador 20

2.2.- Control de velocidad 22

2.2.1.- Identificación del modelo del proceso 22

2.2.1.1.- Ensayo y obtención de los datos 23

2.2.1.2.- Cálculo del modelo y validación mediante la herramienta System Identification Tool de Matlab 24

2.2.1.3.- Importación de los datos 24

2.2.1.4.- Aproximación del modelo y cálculo 26

2.2.1.5.- Comparación de los resultados y elección del modelo 27

2.2.2.- Diseño del regulador 29

2.2.2.1.- Justificación de la elección de un regulador tipo PI 30

2.2.2.2.- Diseño del regulador mediante la herramienta rltool de Matlab 33

2.2.2.3.- Validación del regulador 36

3.- BIBLIOGRAFÍA 39

1.- INTRODUCCIÓN Y OBJETIVOS

En este documento se pretende mostrar la línea de diseño que se ha seguido de cara al desarrollo del software del Robot. Este incluye la fase de identificación del modelo de posición (motores X Y Z) y modelo de velocidad (motor del cabezal), pasando por el diseño de los reguladores y su validación.

Los objetivos principales de este documento son los siguientes:

- Obtención y validación de un modelo matemático para el control de posición de los motores X Y Z mediante la identificación con System Identification Tool.
- Obtención y validación de un modelo matemático para el control de velocidad del motor del cabezal mediante la identificación con System Identification Tool.
- Obtención de los reguladores necesarios para los modelos descritos anteriormente mediante la herramienta rltool de Matlab
- Validación teórica de los reguladores en Simulink junto con la validación experimental en LabVIEW.

2.- DESCRIPCIÓN DE LA SOLUCIÓN ELEGIDA

En este apartado se recoge la solución adoptada en cada uno de los objetivos descritos anteriormente.

2.1.- Control de posición

Se ha realizado el control de posición de 3 de los 4 motores que tiene el robot. Previamente a la identificación del modelo de posición, se ha tenido que realizar un sensor virtual que contase en todo momento las vueltas que daba el motor, de manera que se pudiese saber la posición de cada uno de los brazos del robot en cada instante de tiempo. Este sensor virtual se ha diseñado a partir de un sensor real que proporcionaba información sobre la posición del eje del motor. En el anexo de programación se hablará de este sensor virtual con más detalle.

Hay que tener en cuenta que sin este sensor virtual no se podría haber hecho la identificación, ya que los datos obtenidos en los ensayos se hubieran visto falseados al no representar fielmente la posición del brazo robot, si no la orientación del eje del motor.

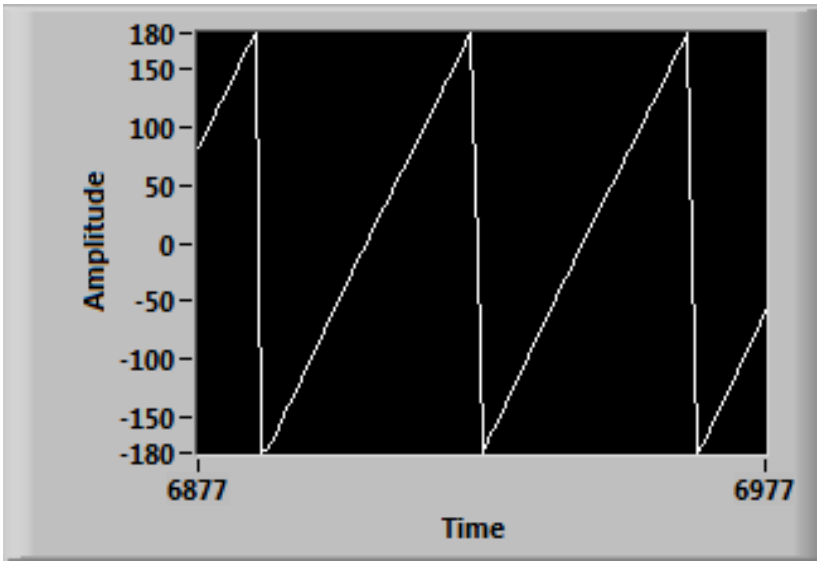


Figura 1: posición del motor antes del sensor virtual.

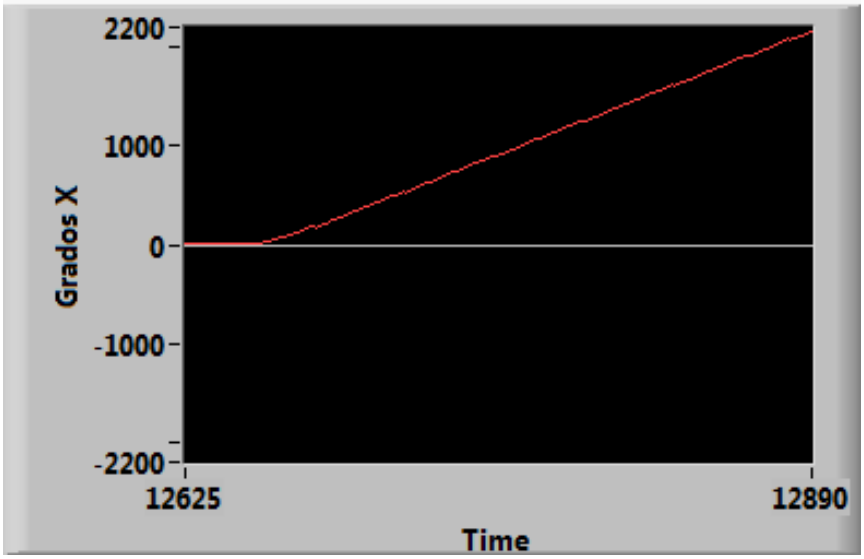


Figura 2: posición de robot después del sensor virtual.

Como se puede observar en la Figura 1, la posición no aumenta conforme se aplica un escalón positivo, si no que describe una especie de onda triangular. No obstante, en la Figura 2 se observa cómo el problema ha quedado resuelto al hacer pasar esta salida por un algoritmo que la rectifica, además de filtrarla. Como ya se ha dicho anteriormente, este algoritmo se explicará más detalladamente en el anexo de programación.

Una vez corregido el efecto de la tensión sobre la posición del brazo robot, podemos pasar a realizar la identificación.

2.1.1.- Identificación del modelo del proceso

La identificación experimental de la Función de Transferencia permite, a partir de una respuesta experimental de un sistema respecto de un punto de equilibrio debida a un cambio en la entrada (rampa, escalón etc), obtener una expresión (Función de transferencia) que aproxima la dinámica dominante del proceso en dicho punto.

La identificación del modelo de posición se ha realizado mediante la herramienta System Identification Tool de Matlab. Esta herramienta permite obtener el modelo de un proceso de manera aproximada a partir de datos obtenidos experimentalmente.

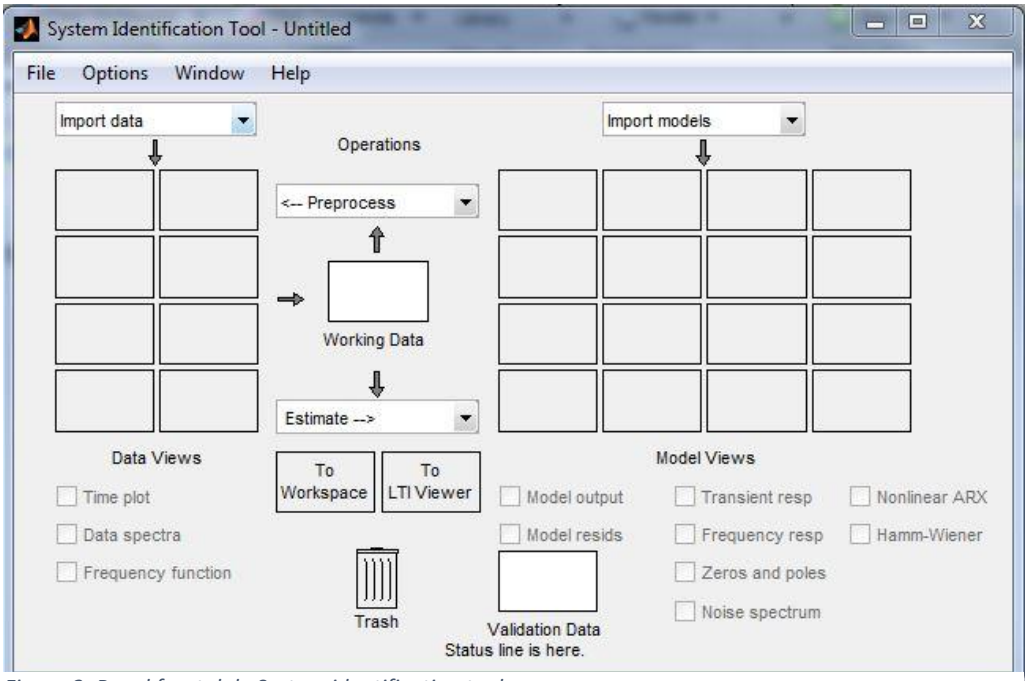


Figura 3: Panel frontal de System identification tool.

2.1.1.1- Ensayo y obtención de los datos

Para realizar los ensayos se ha utilizado LabVIEW. Aplicando unos escalones positivos en la tensión de entrada y estando el motor en el punto de equilibrio, se hace variar la posición del motor en forma de rampa. Esto quiere decir que el modelo de posición del motor es un primer orden con integrador.

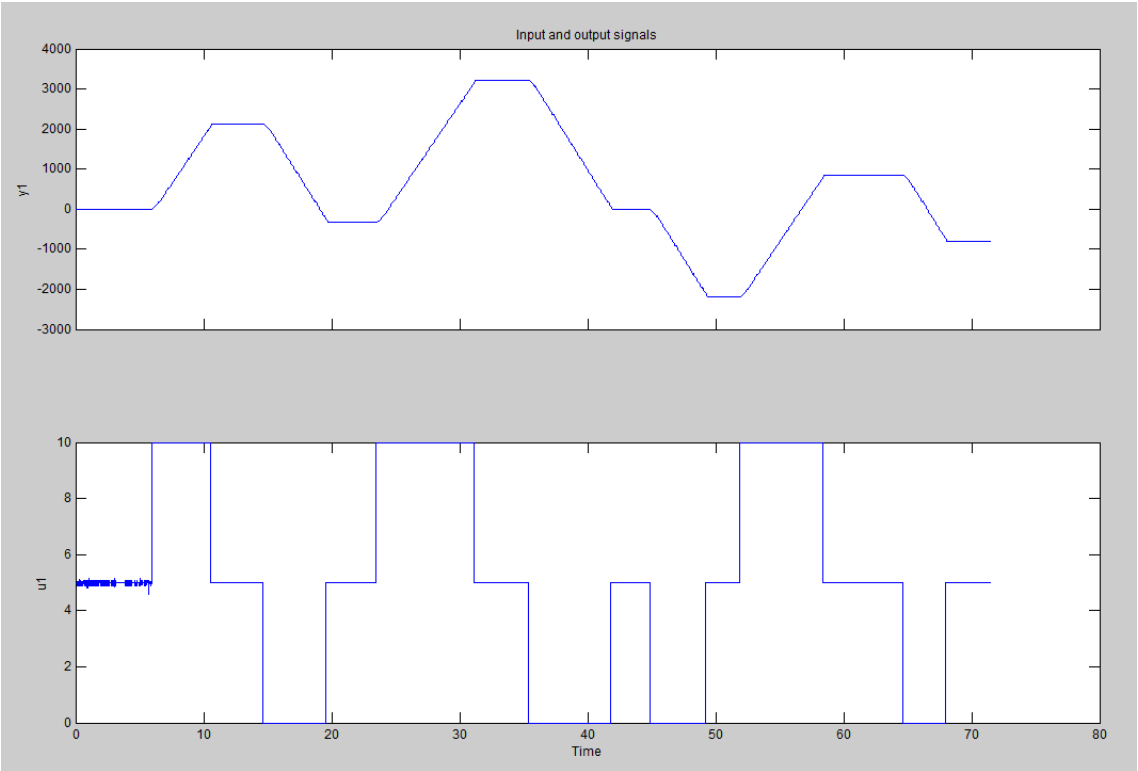


Figura 4: Ensayo importado a System Identification Tool.

LabVIEW guardará en cada periodo de ejecución del bucle de control (periodo de muestreo) un valor de tiempo, tensión aplicada y de grados girados por el motor, generando así una matriz constituida por tres columnas de datos. Una vez terminado el ensayo, se genera un fichero con todos estos valores, el cual nos servirá para realizar la identificación mediante la herramienta System Identification Tool.

Tanto en el control de posición como en el control de velocidad se ha elegido un periodo de muestro (periodo de ejecución del bucle) de 20 milisegundos, el cual es suficiente para mantener una lectura de los sensores actualizada y efectuar un control adecuado.

2.1.1.2- Cálculo del modelo y validación mediante la herramienta ident tool de Matlab

En este apartado se obtendrá finalmente un modelo de posición del motor Feedback, el cual nos servirá para aproximar la dinámica de los tres motores que controlan las coordenadas X, Y, Z.

2.1.1.3.- Importación de los datos

Una vez obtenido los datos en LabVIEW, se procede a su importación en Matlab. Se debe abrir el programa para poder generar dos vectores en el Workspace que representen dos de las tres variables (tensión y posición) del fichero creado en LabVIEW, las cuales servirán para la identificación. A partir de este punto se puede abrir la herramienta System Identification Tool mediante el comando ident en Matlab.

Figura 5: Panel de Command Window de Matlab.

Abierta la herramienta de identificación se debe, a partir del ensayo realizado en LabVIEW, aislar cada uno de los escalones positivos y negativos de éste. De esta manera se obtienen una serie de ensayos (escalones) a los cuales se les debe restar el punto de equilibrio, el cual es el dato inicial de cada uno de los vectores de datos. Restando este punto se obtienen escalones y respuestas que empiezan en cero, es decir, se han transformado los ensayos individuales en ensayos incrementales. Una vez se obtengan estos ensayos incrementales se podrá proceder a estimar un modelo.

Para aislar cada uno de los escalones se ha tenido que importar el ensayo a la herramienta System Ident Tool mediante la opción “Time domain data”.

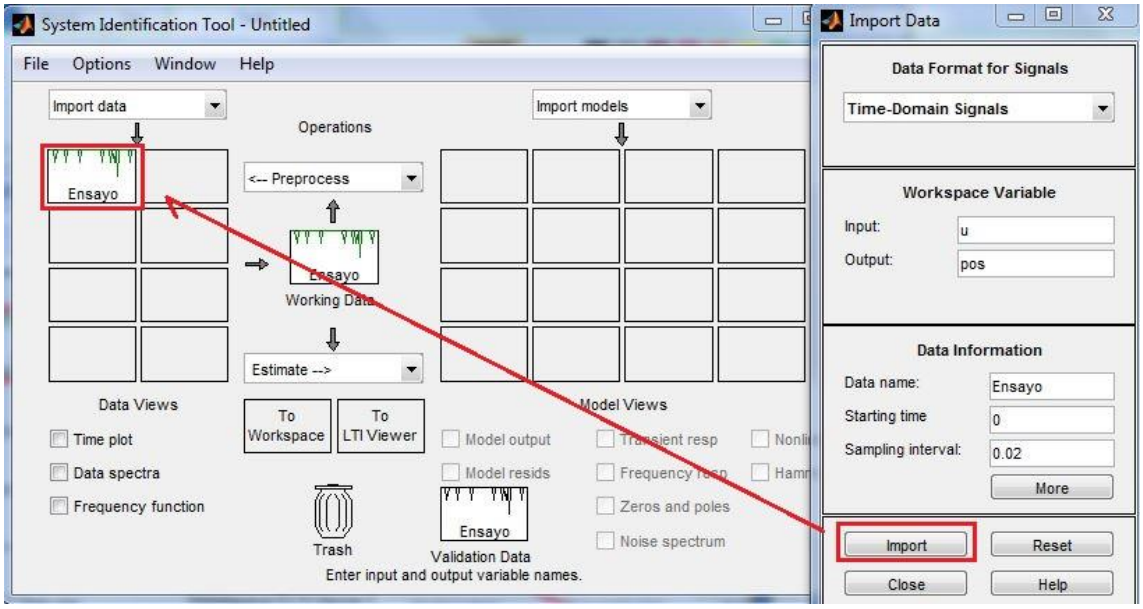


Figura 6: Importación de datos en System Identification Tool.

Cuando se tienen los datos del ensayo, se hace uso de la función “Select range”, mediante la cual aislamos cada uno de los escalones realizados anteriormente. Una vez se obtienen estos ensayos individuales, se trasladan al Workspace.

Como ya se ha comentado anteriormente, hay que restar el punto de equilibro del sistema, lo cual se puede realizar de manera sencilla en Matlab restando el punto inicial de cada vector.

```
>> escalon.u=escalon.u-escalon(1).u

>> escalon.y=escalon.y-escalon(1).y
```

Una vez se han conseguido transformar los ensayos en incrementales, se han de reimportar a la herramienta System Identification Tool mediante la función “Data object”. Posteriormente se procederá a estimar un modelo.

2.1.1.4.- Aproximación del modelo y cálculo

Una vez que a todos los escalones, positivos y negativos, se les ha restado el punto de equilibrio y, por tanto, comienzan en cero, se ha procedido a estimar un modelo de cada uno de ellos.

Se ha utilizado la función "Process Models" de System Identification Tool para estimar el modelo del proceso, la cual permite configurar a través de una ventana el tipo de modelo que se va a aproximar. Entre las opciones posibles permite seleccionar: el orden del modelo (número de polos), ceros, delay, modelo con integrador, etc.

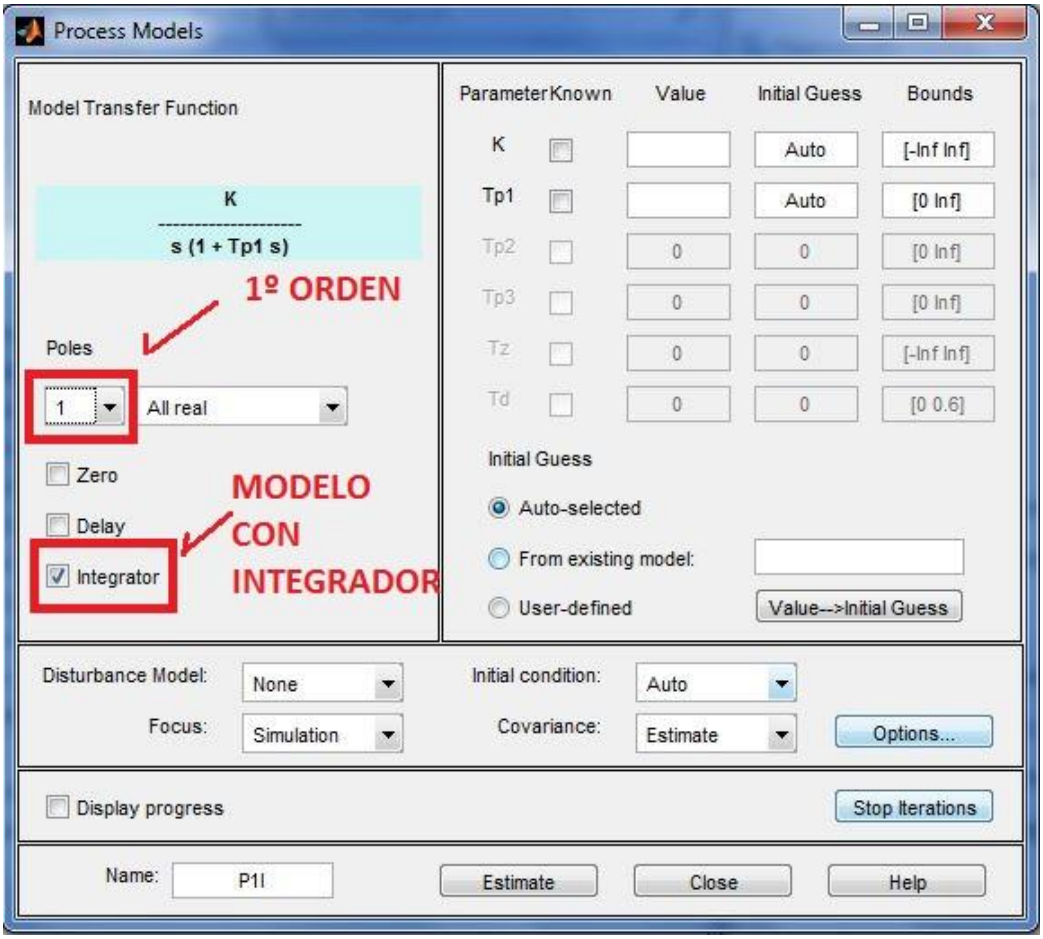


Figura 7: Panel Process Models de System Identification Tool.

Se ha aproximado el modelo por un primer orden con integrador, puesto que al aplicar un escalón en la entrada (tensión), se ha obtenido una respuesta en forma de rampa (Figura 4).

Una vez obtenidos todos los modelos de cada uno de los ensayos, se procede a su validación mediante otros escalones. A partir de un modelo extraído con un escalón dado, se va simulando y analizando como el modelo en cuestión se va aproximando al resto de escalones del ensayo global. En la Figura 8 se muestra un ejemplo de validación. Se puede apreciar como los modelos pos1, pos2, neg1 y neg2, obtenidos mediante sus respectivos escalones, se aproximan al ensayo neg3. El porcentaje de aproximación: 98.49%, 98.2%, 94.28% y 93.75% indica que todos estos modelos reflejan con bastante validez el proceso. A esto cabe añadir que no se pueden utilizar los mismos datos de identificación para validar, puesto que el resultado no sería fiable.

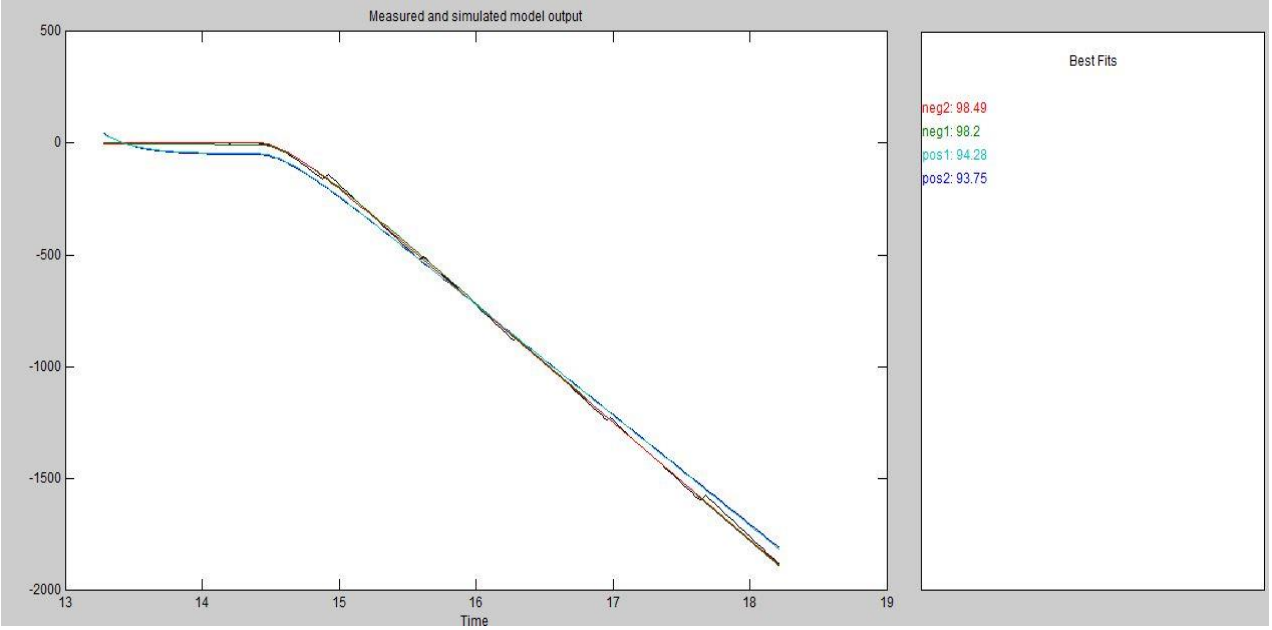


Figura 8: Validación de los modelos de posición.

2.1.1.5.- Comparación de los resultados y elección del modelo

En este apartado se va a comparar todos los modelos obtenidos para controlar la posición del motor y se va a elegir uno de ellos en función de los resultados obtenidos en la validación.

Se han obtenido un total de 5 modelos distintos, a partir de 2 escalones positivos y 3 negativos y se han validado simultáneamente entre sí. En la tabla 1 se recoge el resultado de la identificación de los distintos modelos. Se observa como en la mayoría de la filas, los valores de aproximación más altos son los de la diagonal roja. Esto es debido a que son valores que se han obtenido utilizando los mismos datos en la identificación y en la validación, por tanto, no son datos válidos a tener en cuenta de cara a la elección de la solución.

	Validación				
Modelos	pos1	pos2	neg1	neg2	neg3
pos1	**99,02**	99,02	93,9	94,25	94,28
pos2	98,87	**99,18**	93,43	93,75	93,75
neg1	93,47	92,7	**98,23**	98,43	98,2
neg2	93,82	93,49	98,05	**98,78**	98,49
neg3	94,15	93,83	97,96	98,74	**98,54**

Tabla 1: Porcentaje de aproximación de los distintos modelos de posición.

A pesar de eliminar los resultados remarcados en rojo, se puede concluir que, en general, todos los modelos responden con bastante validez a cualquiera de los ensayos. Es cierto que se observa que los modelos pos1, pos2 obtienen un mayor porcentaje de aproximación en la validación con los escalones positivos, de la misma manera que los ensayos neg1 y neg2 los obtienen con los negativos.

Como el robot va a estar sometido tanto a escalones negativos como positivos en cualquiera de sus tres motores (motor X, motor Y, motor Z) y, además, los valores de ganancia y constante de tiempo (tp1) son muy semejantes en todos los modelos, se ha decidido elegir el modelo que mayor porcentaje medio haya obtenido en la validación. Debido a que hay más ensayos negativos que positivos, para que los ensayos positivos no estén en desventaja se ha optado por descartar el ensayo negativo con menos porcentaje medio de aproximación. De esta manera, descartamos el modelo neg1.

	kp	tp1
pos1	98,821	0,22893
pos2	98,257	0,2311
neg1	106,58	0,2678
neg2	105,2	0,22152
neg3	104,83	0,2213

Tabla 2: Valores de los distintos modelos.

Llegados a este punto, obtenemos la media ponderada de los porcentajes de validación de cada uno de los modelos, llegando así a la tabla 3. De esta manera, se concluye que el modelo más adecuado para ajustar la dinámica del robot es el modelo pos1 con una ganancia de 98.821 y una constante de tiempo de 0.22893.

	MEDIA
pos1	96,58401
pos2	96,315214
neg2	96,080259
neg3	96,258684
MAX	96,58401

Tabla3: Medias de los porcentajes de aproximación.

Por tanto, el modelo elegido para representar la dinámica del proceso ha sido:

$$G_p(s) = \frac{98.821}{s(0.228s + 1)} \left(\frac{grados}{V}\right)$$

2.1.2.- Diseño del regulador

En este apartado se desarrollará el diseño que se ha llevado a cabo para obtener el regulador necesario para nuestro proceso, de manera que se pueda cerrar el lazo de control de posición.

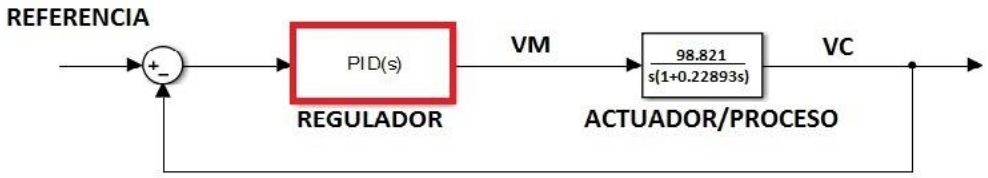


Figura 9: bucle de control de Posición.

2.1.2.1.- Justificación de la elección de un regulador tipo PD

El sistema que se está tratando de controlar es un motor. Se sabe que si un sistema, al cual se le aplica una entrada de tipo escalón, responde a la salida con una rampa, se puede aproximar por un sistema de primer orden con integrador. Por tanto, sabemos que nuestro proceso se trata de uno de este tipo.

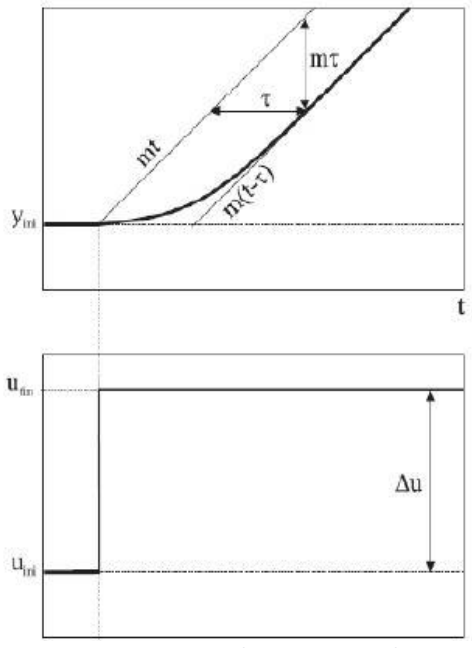


Figura 10: Primer orden con integrador.

$$G(s) = \frac{K}{s(\tau s+1)}$$

$$K = \frac{m}{\Delta u}$$

Se está controlando la posición de un brazo robot, por tanto existen una serie de especificaciones dinámicas y estáticas que se deben cumplir al añadir el regulador:

- El tiempo de establecimiento del robot debe ser menor a 2 segundos ($T_{es} \leq 2\text{seg}$), siempre que no sature la acción de control.
- Se trata del control de posición de un robot, por tanto, no interesa que tenga sobreoscilación, ya que podría provocar problemas en el resto de elementos mecánicos. Por tanto δ=0.
- Por último, cabe señalar que el robot se diseña para operaciones de taladrado, por tanto sería inconcebible que el motor tuviera error en la posición del eje del taladro. Por consiguiente Ep=0.

Una vez se conocen las especificaciones dinámicas, se puede representar en el plano complejo el conjunto de soluciones válidas, para situar nuestros polos en bucle cerrado.

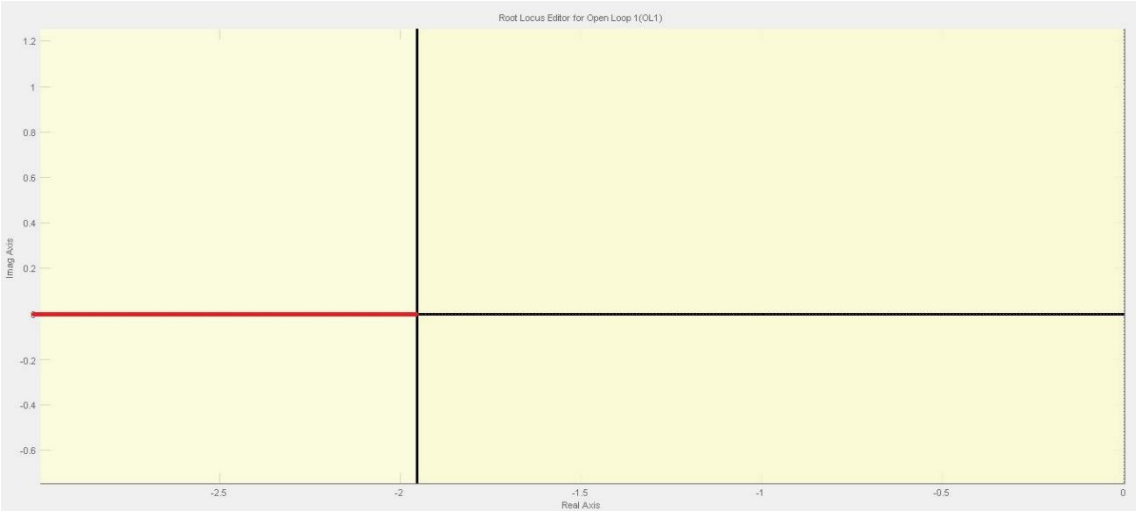


Figura 11: Zona de especificaciones dinámica.

En la figura 11 se puede apreciar la zona de especificaciones dinámicas (línea roja), definida por el tiempo de establecimiento y porcentaje de sobreoscilación. Se puede observar como el conjunto de soluciones del plano complejo queda reducido al eje real.

Para que el regulador que se está diseñando pueda cumplir con todas estas especificaciones, se ha de decidir cuidadosamente qué regulador de tipo PID se necesita (P, PI, PD, PID). Para tomar esta decisión se ha recurrido a la ayuda del siguiente árbol de decisión:

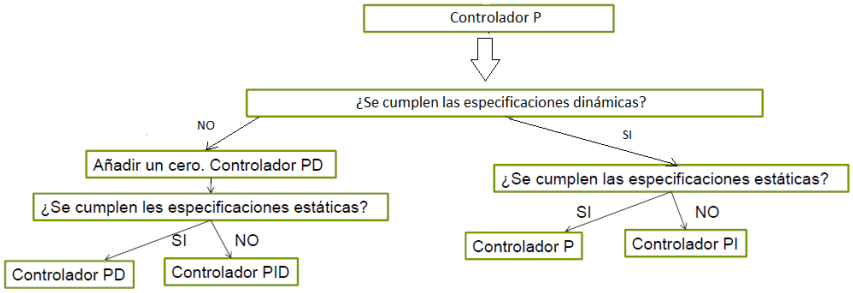


Figura 12: Árbol de decisión.

Se ha analizado cada especificación por separado y se han obtenido las siguientes conclusiones:

- El tiempo de establecimiento de un sistema es aproximadamente el tiempo de establecimiento de su polo dominante (polo más lento). Este tiempo es $\frac{4}{\sigma}$, siendo σ la parte real del polo en valor absoluto. En principio esto no debe preocupar demasiado, debido a que se podrá conseguir modificando la ganancia de nuestro regulador, por tanto hasta este punto solo es necesario un regulador de tipo P.

$$P: G_{R\_P}: (s) = K_p$$

- Para que el sistema no tenga sobreoscilación, se ha de incluir un parte derivativa, dado que con un regulador de tipo proporcional no se podría conseguir el cumplimiento de ambas especificaciones dinámicas (tiempo de establecimiento y sobreoscilación). Por tanto, se necesita un regulador de tipo PD.

$$PD: G_{R_{PD}}(s) = K_P(1 + T_d s) = \frac{K(s+b)}{s}$$

- Por último, no va a ser necesario incluir la parte integral en el regulador, por tanto no va ser necesario utilizar un PID. Esto es debido a que, para garantizar que nuestro sistema no tenga error de posición en bucle cerrado bajo un cambio en el punto de consigna, nuestro regulador o nuestro proceso ha de poseer un integrador. Como ya se ha explicado anteriormente, el modelo de nuestro proceso posee un integrador, es decir, un polo en cero.

Por tanto, para conseguir que se cumplan las especificaciones establecidas, es necesario que el regulador contemple parte proporcional y parte derivativa, es decir, un regulador de tipo PD.

2.1.2.2.- Diseño del regulador mediante la herramienta rltool de Matlab

Una vez se tiene claro qué tipo de regulador se quiere usar para controlar nuestro proceso, procedemos a su diseño. Existen distintos métodos para diseñar un regulador, a continuación se nombran distintas metodologías para su diseño:

- Método Algebraico: es una manera sencilla y mecánica de obtener un regulador, pero tiene problemas con modelos que presentan un gran número de polos, 3 o más. Además, no permite ver de manera gráfica qué especificaciones va a ser posible conseguir. Por tanto, no se ha optado por este método.
- Métodos empíricos: Ziegler-Nichols, CHR o Cohen Coon son métodos experimentales que permiten escoger los parámetros del PID a partir de las propiedades dinámicas más importantes del proceso a controlar. Este tipo de métodos han sido desarrollados para personal con conocimientos insuficientes para realizar un diseño analítico, por tanto no permite hacer un diseño ajustado. En muchas ocasiones es necesario realizar una segunda etapa de sintonía manual para mejorar los resultados.
- Por último, El lugar de las raíces, el cual se ha decidido usar. Es un método que presenta múltiples ventajas:
    1. Ver de manera gráfica qué especificaciones va a ser posible conseguir.
    2. Permite resolver cualquier sistema independientemente del orden que éste presente.
    3. Dispone de un procedimiento sistemático de diseño.
    4. Permite analizar el BC (bucle cerrado) resultante con un PID concreto.

Para el cálculo de El lugar de las raíces se ha utilizado la herramienta rltool de Matlab, la cual permite obtener de una manera sencilla el modelo del regulador y el conjunto de soluciones posibles.

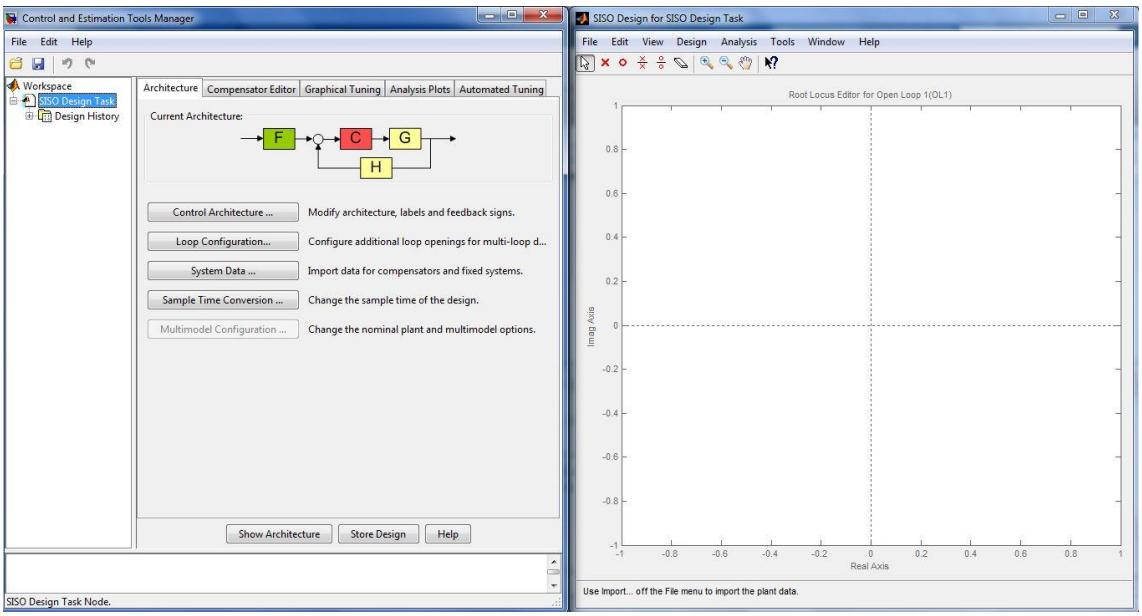


Figura 13: Interfaz del Rltool.

Lo primero de todo es introducir la Función de transferencia (FDT) del modelo, obtenido mediante Ident, en Matlab. Una vez hecho, se procede a abrir la herramienta rltool tecleando su nombre en la venta del Command Window en Matlab.

```
>> s=tf('s');
>> g=98.821/(s*(s*0.22893+1))

g =

       98.82
  ---------------
  0.2289 s^2 + s

Continuous-time transfer function.

>> rltool(g)
```

FDT DEL MOTOR

comando para abrir el motor

Figura 14: Código en el Command Window de Matlab.

Una vez se ha abierto el rltool, el programa muestra el Lugar de las raíces del sistema en bucle cerrado con un regulador proporcional (Figura 15).

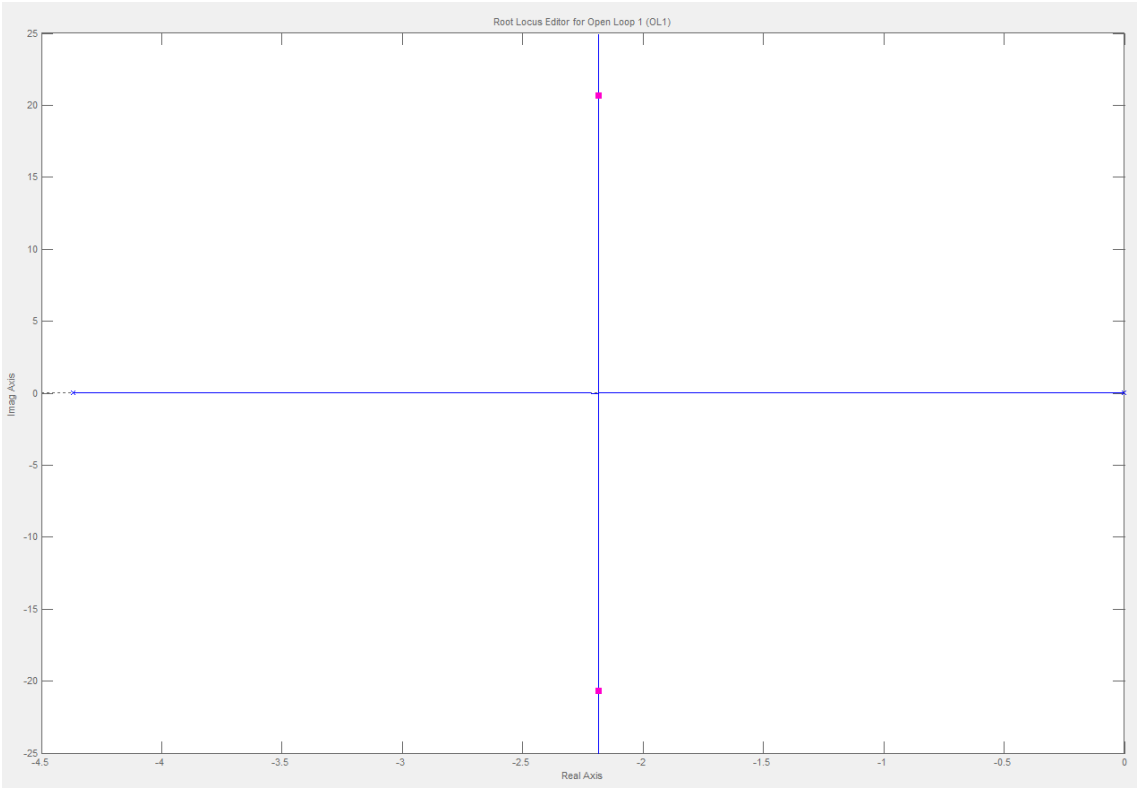


Figura 15: LR del modelo posición del motor.

Para poder realizar el cálculo del regulador es necesario añadir el elemento que aporta el PD, es decir, se ha añadido un cero real.

El regulador se ha diseñado por el método de la cancelación. Se ha cancelado con el cero del regulador uno de los polos del proceso. El proceso posee dos polos:

- Un polo en cero (s=0), un integrador.
- Un polo en -4.368 $\left(s = -\frac{1}{\tau} = \frac{1}{0.2289}\right)$.

Un integrador se considera un polo inestable, ya que se encuentra en el límite de la inestabilidad, por tanto, no se ha cancelado este polo. Si se hubiera cancelado este polo, el sistema hubiese continuado siendo inestable. Por tanto, se ha procedido a cancelar el polo en -4.368.

A continuación, se muestra el Lugar de la raíces de la FDT del sistema en bucle cerrado con un regulador PD:

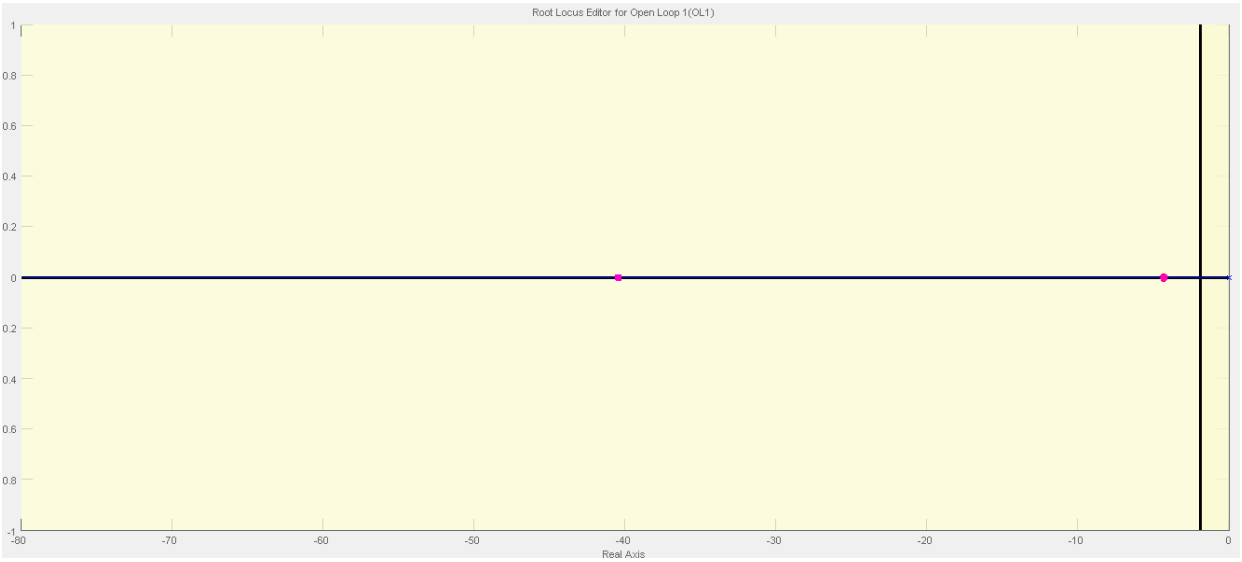


Figura 16: LR del sistema en bucle cerrado con un regulador PD.

En la Figura 16 se puede observar como el Lugar de las raíces queda reducido al eje real, coincidiendo así con la zona de especificaciones dinámicas.

Finalmente, se ha decidido colocar el polo en -49.5, dando lugar a una ganancia de 0.5, quedando el regulador de la siguiente manera:

$$G_r(s) = 0.5(s + 0.23)$$

Calculado el regulador, se procede a la validación del mismo mediante Simulink.

2.1.2.3.- Validación del regulador

Una vez se ha obtenido el regulador, se ha procedido a su validación en Simulink. Se ha construido un diagrama de bloques para simular el control de posición, donde se ha introducido la Función de transferencia del proceso y del regulador. A continuación se muestra una imagen con este diagrama:

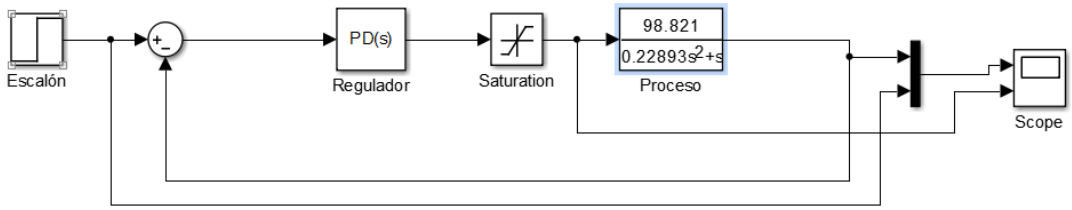


Figura 17: Diagrama de bloques en Simulink.

Posteriormente, se ha aplicado un escalón de 720 grados en la referencia y se ha realizado la simulación, representando en el *Scope* la acción de control, la posición y la referencia:

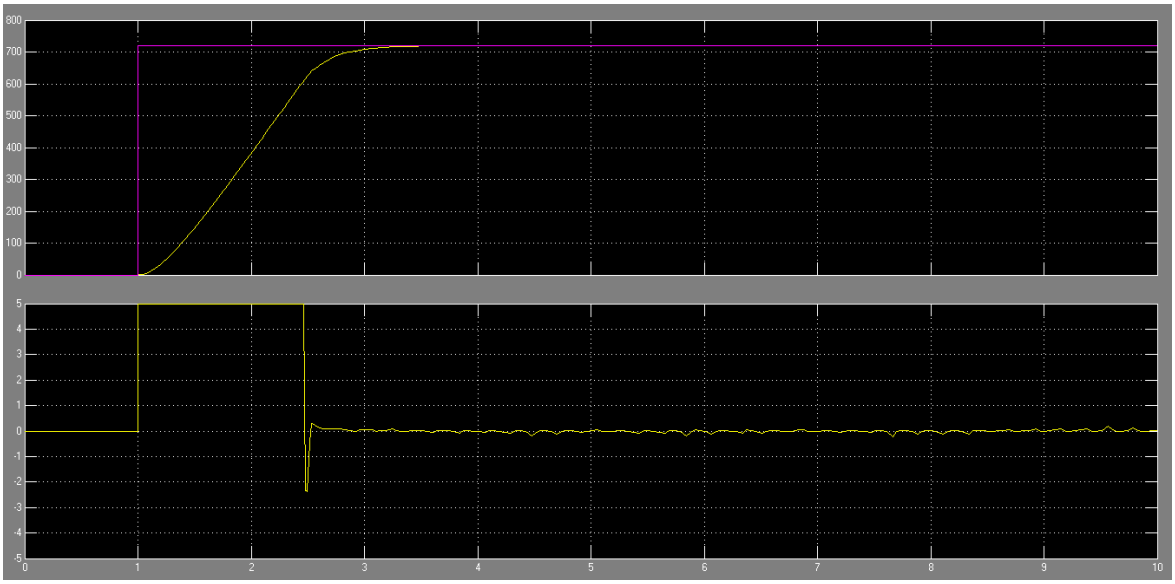


Figura 18: Simulación del control de posición.

A partir de estos resultados se han obtenido las siguientes conclusiones:

- Se puede observar en la Figura 18 que se produce la saturación de la acción de control, esto va a provocar que el tiempo de establecimiento del sistema no sea el teórico estimado en el rltool. Esto se produce debido a que, al saturar la acción de control, el actuador no puede darle al sistema la tensión necesaria para alcanzar el régimen permanente en el tiempo deseado. No obstante, esto significa que el sistema es todo lo rápido que puede, lo cual es deseado, ya que se pretende que los motores alcancen su posición en el menor tiempo posible.
- Se observa que la salida no presenta sobreoscilación, ya que el control derivativo funciona adecuadamente.
- La salida no presenta error de posición gracias al integrador del proceso.

A pesar de que sature la acción de control, el regulador es adecuado, ya que interesa tener un regulador agresivo para alcanzar la posición lo antes posible. A continuación, se muestra una imagen en LabVIEW donde se realiza la validación experimental del regulador, comprobándose que los resultados simulados coinciden con los reales. Finalmente se da el regulador por bueno.

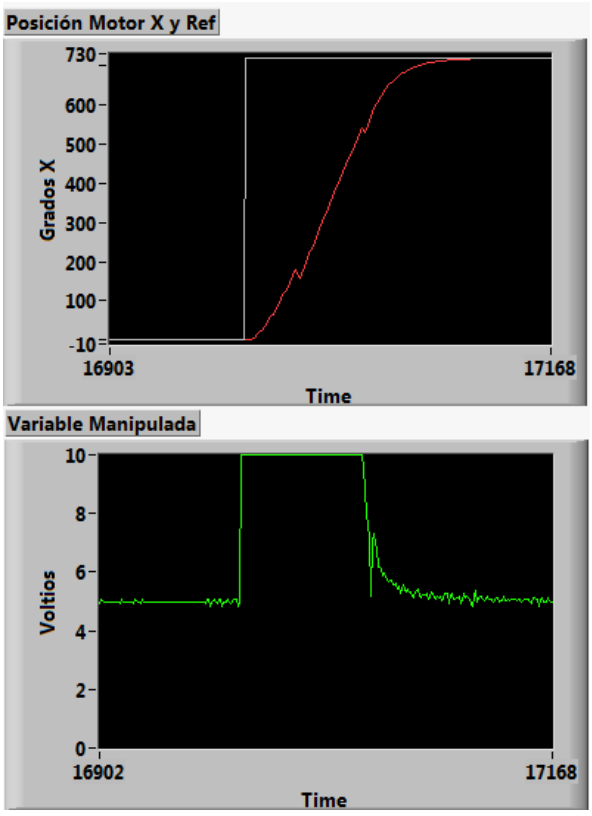


Figura 19: Comprobación del regulador mediante un escalón en LabVIEW.

2.2.- Control de velocidad

En este apartado se va a proceder a explicar el control de velocidad que se ha diseñado para el motor del cabezal del robot. En este caso, únicamente se ha aplicado este control a uno de los cuatro motores.

Uno de los problemas previos que ha surgido durante el desarrollo de este control ha sido ocasionado por el hecho de que la tarjeta de adquisición de datos disponía únicamente de dos salidas. Como el proyecto trata un robot de cuatro motores, se necesitan al menos cuatro salidas en la tarjeta para controlar simultáneamente las tres posiciones de los motores (X Y Z) y la velocidad del motor del cabezal. El problema se ha resuelto utilizando dos tarjetas de adquisición de datos y realizando una comunicación por protocolo TCP a través de ethernet mediante un programa realizado en LabVIEW, del cual se hablará en el anexo de programación.

2.2.1.- Identificación del modelo del proceso

En este apartado se va a proceder de una manera más rápida debido a que gran parte de las cosas ya se han explicado con anterioridad. En este caso no ha sido necesario realizar ningún tipo de adaptación previa al sensor, ya que estamos utilizando directamente el tacómetro para medir la velocidad. No obstante, se ha tenido que construir previamente un nuevo bucle de control para realizar los ensayos y la obtención de los datos.

2.2.1.1.- Ensayo y obtención de los datos

De forma similar a como se ha explicado en el apartado de control de posición, para realizar la obtención de los datos se han aplicado unos escalones positivos y negativos en la entrada del proceso. En este caso la respuesta del sistema no ha sido en forma de rampa, si no sobreamortiguada. Esto significa que se va a aproximar el modelo de velocidad por un primer orden.

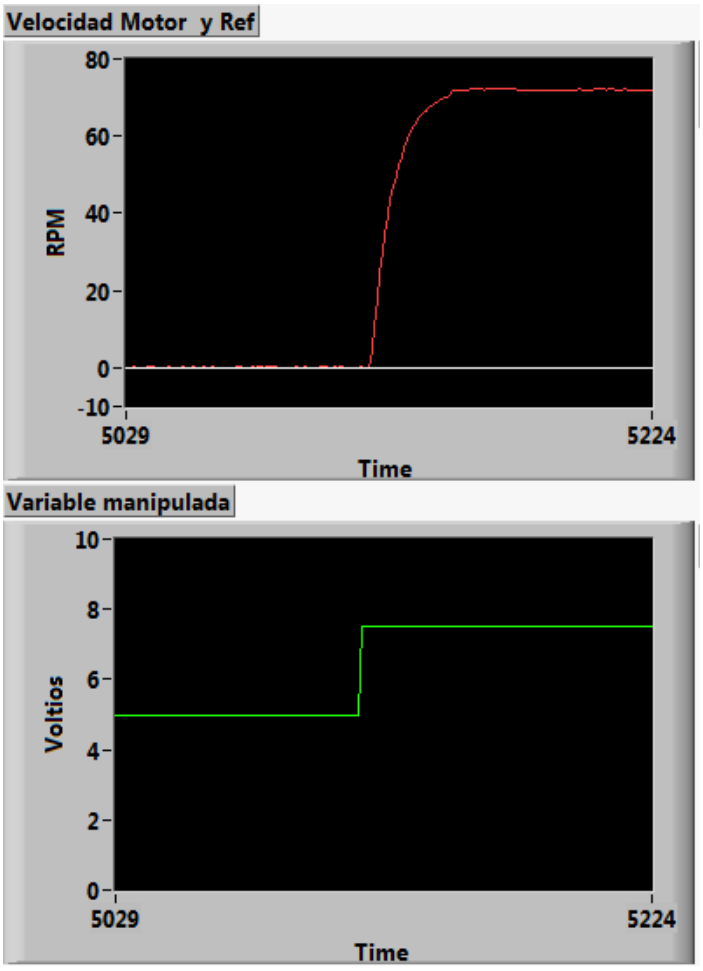


Figura 20: Respuesta sobreamortiguada.

Se ha tenido en cuenta que los escalones aplicados en los ensayos no deben ser demasiado grandes, ya que se podría entrar en zona de saturación. Esto puede generar problemas debido a que en la fase de identificación se obtienen modelos con menos ganancia, lo que provoca que más tarde el regulador sea mucho más agresivo.

En este caso, el bucle de control de LabVIEW ha tomado datos de tiempo, velocidad y tensión aplicada al motor cada tiempo de ejecución del bucle (tiempo de muestreo), generando así el fichero con tres columnas de datos.

2.2.1.2.- Cálculo del modelo y validación mediante la herramienta System Identification Tool de Matlab

De nuevo se ha utilizado la herramienta System Identification tool para el cálculo del modelo. Esta vez se ha utilizado para obtener un modelo de velocidad.

2.2.1.3.- Importación de los datos

Una vez generado el fichero de datos del ensayo, se han vuelto a extraer de la matriz dos vectores, uno de tensiones aplicadas y otro de velocidades (en Matlab). Posteriormente, se abre la herramienta ident para proceder a la identificación del modelo.

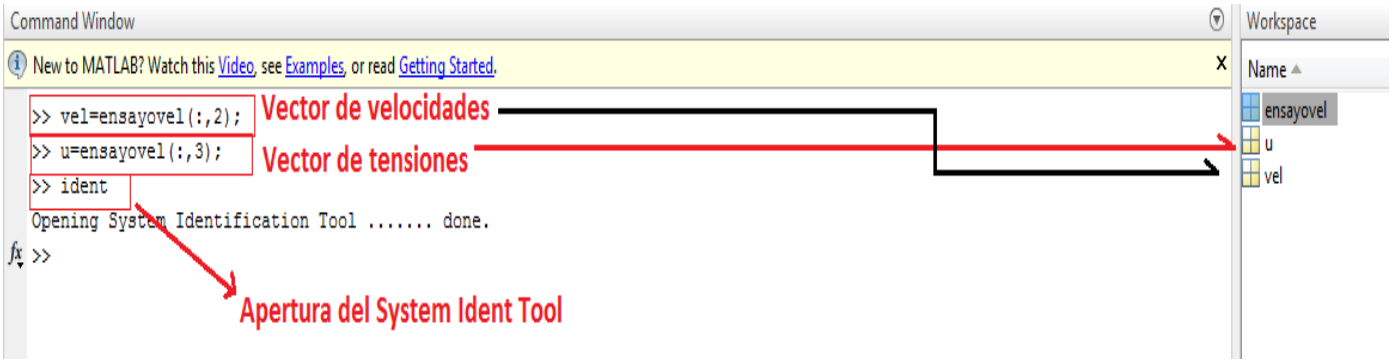


Figura 21: Panel de Command window de Matlab.

Una vez se ha abierto el System Identification Tool, se ha procedido a importar el ensayo, de tal manera que la entrada (input) ha sido la tensión y la salida (output) ha sido la velocidad. En la Figura 22, se muestra como queda el ensayo total realizado en LabVIEW una vez importado a System Identification Tool.

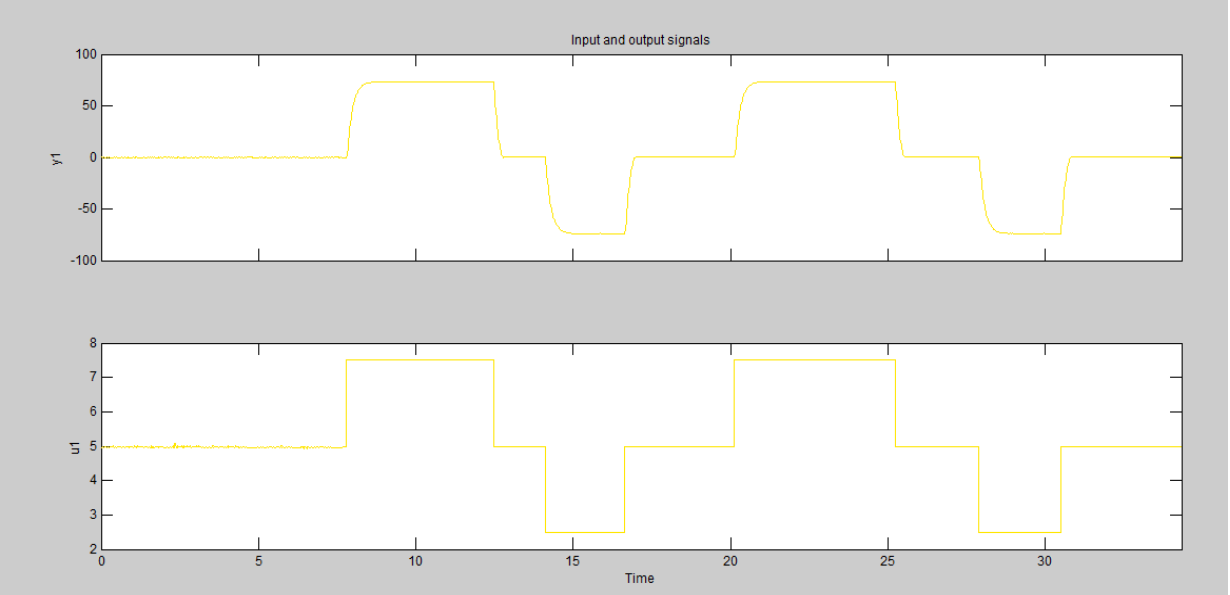


Figura 22: Ensayo importado a System Identification Tool.

Una vez cargado el ensayo en ident, se ha vuelto a discretizar el ensayo global (seleccionando los distintos escalones) en pequeños subensayos formados por escalones individuales positivos y negativos. Posteriormente, se ha procedido a restar el punto de equilibro a estos ensayos individuales. Como bien se ha comentado anteriormente, restando este punto se obtienen escalones y respuestas que empiezan en cero, es decir, se han transformado los ensayos individuales en ensayos incrementales.

Para restar el punto de equilibrio a estos ensayos individuales y transformarlos en ensayos incrementales se ha importado al Workspace de Matlab cada uno de estos ensayos (en forma de vectores) y, posteriormente, se les ha restador el punto inicial. Este punto inicial es un punto donde todas las variables se encuentran en régimen permanente, es decir, todo se mantiene constante. Dicho de otra manera, es el punto de equilibrio.

```
>> escalon.u=escalon.u-escalon(1).u

>> escalon.y=escalon.y-escalon(1).y
```

Una vez restado el punto inicial, se han reimportado los datos a la herramienta System Identification Tool, tal y como se ha hecho en el control de posición. De esta manera, ya tenemos todos los ensayos con variables incrementales.

En la Figura 23 se muestra un ejemplo de un ensayo individualizado.

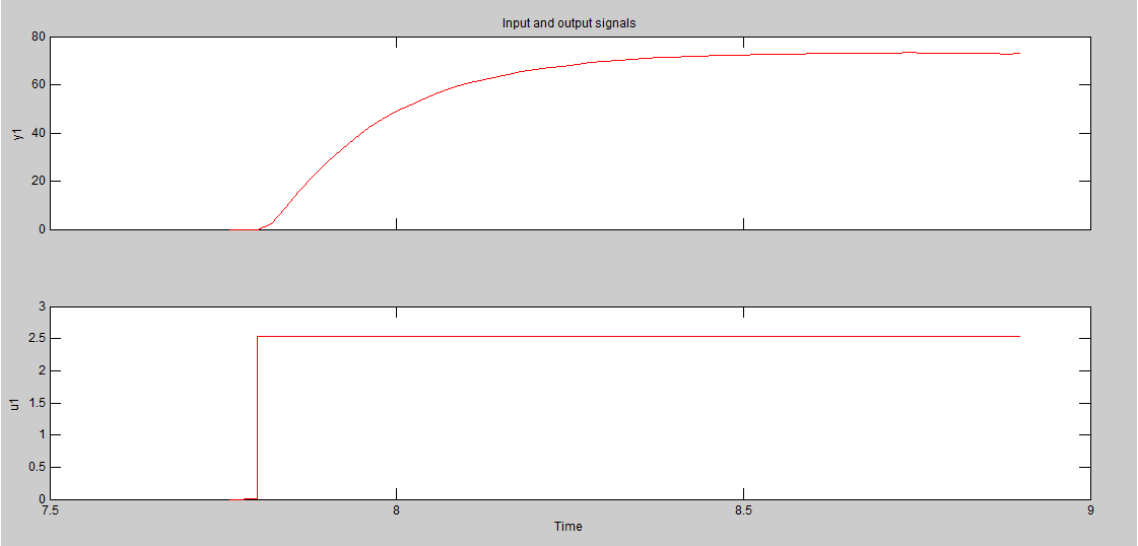


Figura 23: Ensayo individualizado a un escalón.

2.2.1.4.- Aproximación del modelo y cálculo

De nuevo se han calculado los distintos modelos mediante la función Process Models. En esta ocasión, no se ha aproximado por un primer orden con integrador, si no por un primer orden ya que, como ya se ha comentado anteriormente, la respuesta del sistema ante una entrada de tipo escalón ha sido sobreamortiguada. Se han realizado 8 escalones distintos, lo que ha dado lugar a 8 sub-ensayos, de manera que a cada modelo se le ha dado el nombre del ensayo mediante el cual ha sido extraído.

Una vez obtenidos todos los modelos de cada uno de los ensayos, se procede a su validación mediante otros escalones. A partir de un modelo extraído con un escalón dado, se va simulando y analizando cómo el modelo en cuestión se va aproximando al resto de escalones del ensayo global.

En la Figura 24 se muestra un ejemplo de validación donde se ha simulado cada uno de los 8 modelos calculados con el ensayo pos1.

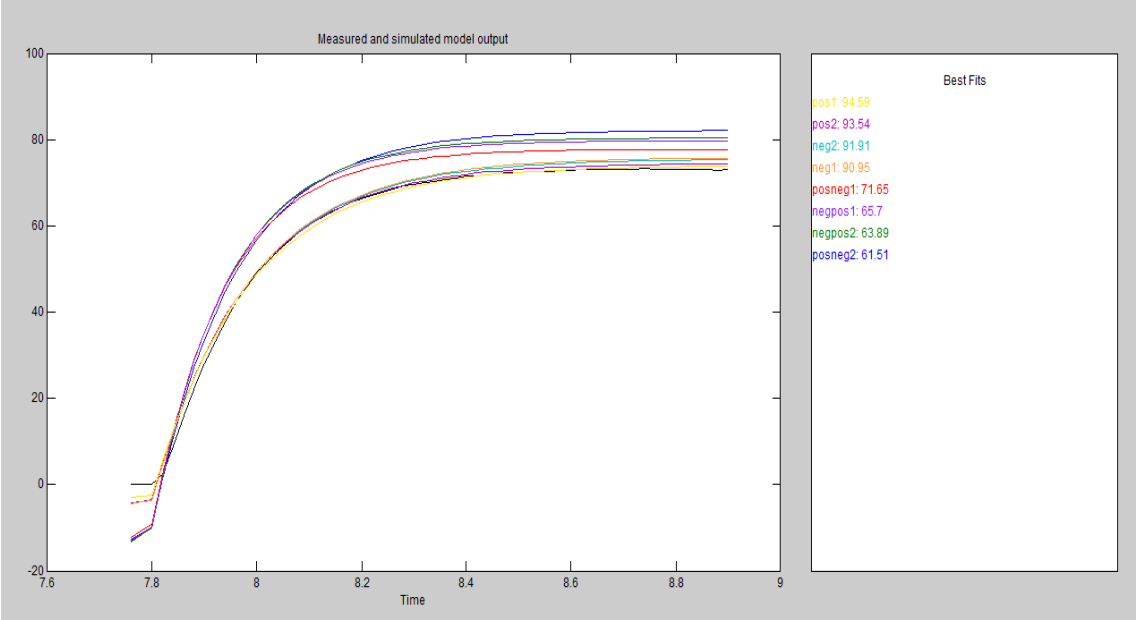


Figura 24: Validación de los modelos de velocidad.

Se puede observar que cada una de las gráficas representadas en la Figura 24 presenta un inicio distinto, esto es debido a que las condiciones iniciales en System Identification Tool no son nulas, sino que son estimadas por la herramienta para cada una.

2.2.1.5.- Comparación de los resultados y elección del modelo

Una vez se ha simulado cada modelo con cada uno de los diferentes ensayos, se ha realizado una tabla comparativa recopilando toda la información:

	validacion							
Modelos	**pos1**	**pos2**	**posneg1**	**posneg2**	**neg1**	**neg2**	**negpos1**	**negpos2**
pos1	**94,59**	93,92	71,77	69,46	91,66	92,59	68,86	68,46
pos2	93,54	**95,03**	74,5	72,18	93,86	94,58	71,62	71,25
posneg1	71,65	75,37	**88,8**	88,38	78,16	78,51	87,9	88,43
posneg2	61,51	65,1	85,36	**89,82**	69,82	69,03	88,11	89,16
neg1	90,95	93,48	76	73,43	**95,21**	95,21	72,87	72,49
neg2	91,91	94,25	75,55	73,04	95,06	**95,39**	72,48	72,1
negpos1	65,7	69,43	87,87	89,66	73,05	73,09	**88,6**	89,51
negpos2	63,89	67,61	87,2	89,81	71,46	71,39	88,54	**89,56**

Tabla 4: Porcentaje de aproximación de los distintos modelos de velocidad.

Se puede apreciar como los modelos posneg1, posneg2, posneg1 y posneg2, obtenidos mediante sus respectivos escalones, no se aproximan con demasiada validez a los ensayos pos1, pos 2, neg1, neg2. Esto se debe a que estos escalones han sido realizados en puntos de equilibrio distintos al punto donde se ha realizado los escalones pos1, pos2, neg1 y neg2. Al no haber hecho estos escalones en el mismo punto de equilibrio, se produce una variación en la dinámica del sistema, que puede verse reflejada, por ejemplo, en la dificultad del sistema para incrementar la velocidad una vez ha alcanzado ciertas velocidades.

Por tanto, se ha decidido en el proceso de selección del modelo descartar los modelos posneg1, posneg2, negpos1 y negpos2, debido a que el motor de velocidad del robot va a partir siempre desde una velocidad de reposo igual a cero, que es el punto de equilibrio donde se han realizado los ensayos pos1, pos2, neg1 ,neg2.

Cabe destacar también que la diagonal roja de la tabla 4 son resultados obtenidos, fruto de utilizar el mismo ensayo en el cálculo del modelo y en la validación, por eso son datos no demasiado fiables para contrastar la validación.

Otra cuestión que se ha tenido en cuenta a la hora de la selección de un modelo, ha sido la valoración de los parámetros estimados por la herramienta System Identification Tool. No obstante, en la tabla 5 se observa que los parámetros que se han obtenido de los diferentes modelos candidatos son prácticamente iguales.

Modelos	Kp	Tp1
pos1	**29.256**	**0.18238**
pos2	**29.472**	**0.17689**
neg1	**30.01**	**0.18178**
neg2	**29.472**	**0.17689**

Tabla 5: parámetros de los modelos de velocidad.

Se ha realizado la media aritmética de porcentajes de aproximación de los modelos pos1, pos2, neg1 y neg2 resultando la siguiente tabla:

Modelos	Media
pos1	**93,17605**
pos2	**94,24886**
neg1	**93,6796134**
neg2	**94,1326209**

Tabla 6: Media de aproximación de los modelos candidatos.

En este caso, el modelo de velocidad con más porcentaje de aproximación es el pos2, con una ganancia de 29.472 y una contante de tiempo (Tp1) τ=0.17689.

$$G_p(s) = \frac{29.472}{0.176s + 1}\left(\frac{rpm}{V}\right)$$

2.2.2.- Diseño del regulador

En el siguiente apartado se ha desarrollado el diseño que se ha llevado a cabo para obtener, en este caso, el regulador necesario para controlar el motor del taladro, cerrando así el bucle de control de velocidad.

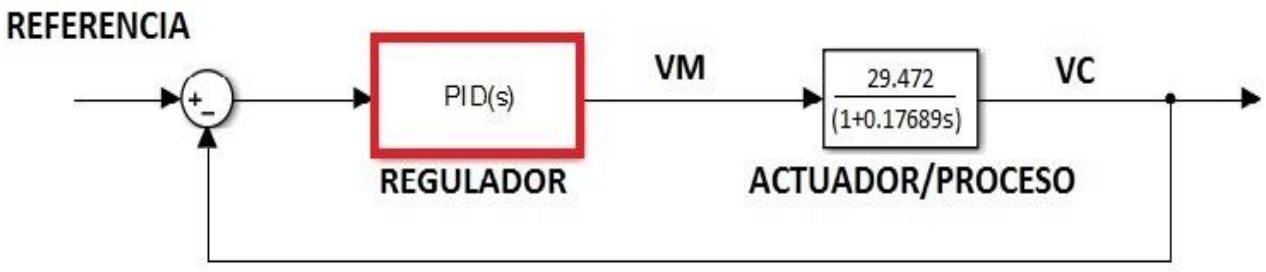
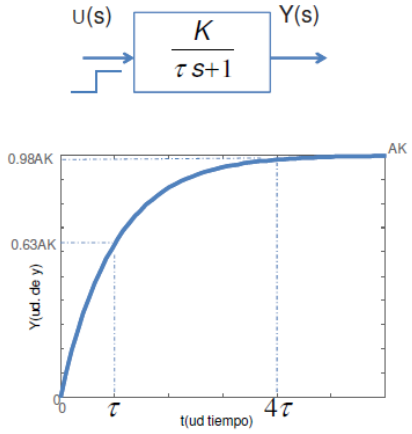


Figura 25: bucle de control de velocidad.

2.2.2.1.- Justificación de la elección de un regulador tipo PI

Como bien se ha comentado con anterioridad, al aplicar escalones positivos de tensión en el motor del cabezal se han obtenido respuestas sobreamortiguadas, por tanto, en este caso se ha aproximado el sistema por un primer orden. Esta información se ha utilizado debido a que la característica del sistema que se quiere controlar influye directamente en la elección del tipo de regulador.



$$Y(s) = G(s)U(s) = \frac{K}{(\tau s+1)} \cdot \frac{A}{s}$$

$$y(t) = A \cdot K(1 - \underbrace{e^{-\frac{t}{\tau}}}_{transitorio})$$

Figura 26: Sistema de 1ª Orden.

En este caso se está controlando la velocidad de un motor, por tanto, existe una serie de especificaciones estáticas y dinámicas distintas a las existentes en el control de posición que debe cumplir el regulador a diseñar:

- El tiempo de establecimiento del motor del cabezal debe ser menor a 2 segundos ($T_{es} \leq 2\text{seg}$) siempre que no se produzca saturación.
- Al tratarse de un control de velocidad no importa demasiado que tenga sobreoscilación, por tanto se ha establecido un límite razonable para sobreoscilación δ≤4.3%.
- Por último, cabe señalar que el robot se diseña para operaciones de taladrado de precisión, por tanto la velocidad ha de tomar un valor fiable para poder conseguir una ejecución práctica exitosa. Por consiguiente, se ha establecido que el error de posición debe ser 0 (Ep=0).

Una vez se conocen las especificaciones dinámicas, se puede representar en el plano complejo el conjunto de soluciones válidas para situar nuestros polos en bucle cerrado.

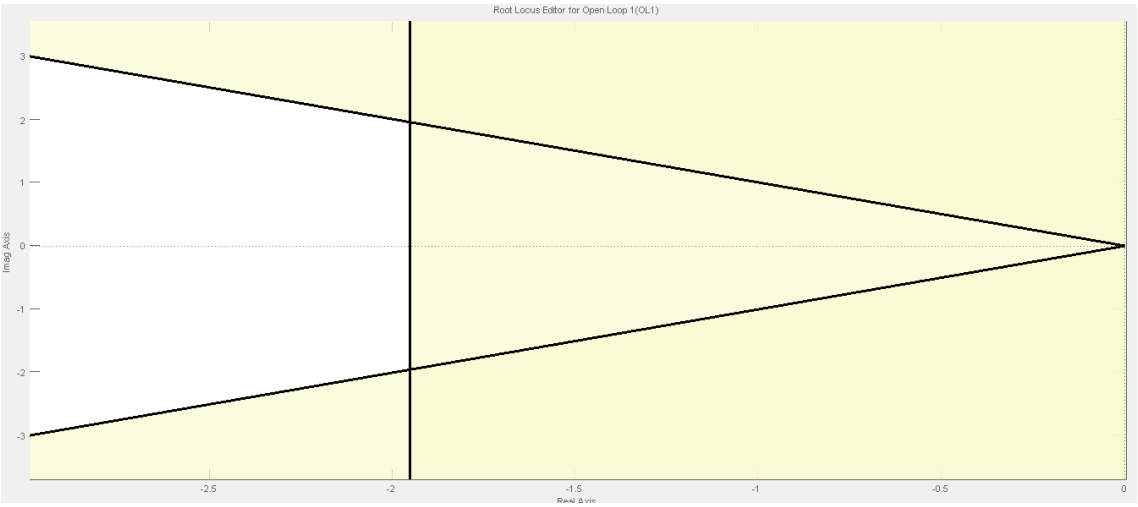


Figura 27: Zona de especificaciones Dinámica para el control de velocidad.

En la figura 27 se puede apreciar la zona de especificaciones dinámicas definida por el tiempo de establecimiento y el porcentaje de sobreoscilación. Se observa cómo el conjunto de soluciones del plano complejo queda reducido a parte del semiplano complejo de la derecha.

Para que el regulador que se está diseñando pueda cumplir con todas estas especificaciones, se ha de decidir cuidadosamente qué regulador de tipo PID necesitamos (P, PI, PD, PID). Para tomar esta decisión se ha utilizado la ayuda del siguiente árbol de decisión, al igual que en control de posición:

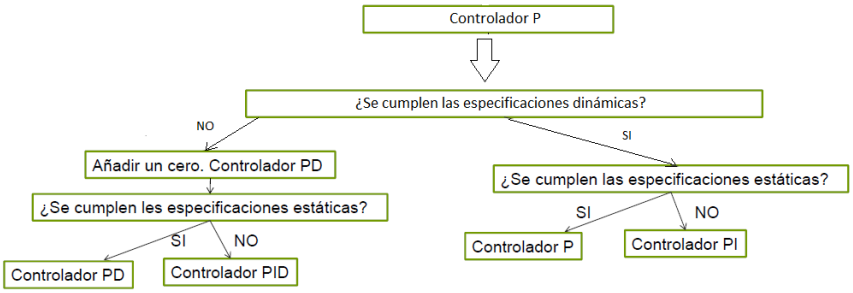


Figura 28: Árbol de decisión.

No obstante, se ha analizado cada especificación por separado y se han obtenido conclusiones:

- El tiempo de establecimiento de un sistema es aproximadamente el tiempo de establecimiento de su polo dominante (polo más lento). Este tiempo es $\frac{4}{\sigma}$, siendo σ la parte real del polo en valor absoluto. En principio esto no debe preocupar demasiado, debido a que la posición de este polo en el plano complejo podrá alterarse modificando la ganancia del regulador. Por tanto, hasta este punto solo es necesario un regulador de tipo P.

$$P: G_{R\_P}: (s) = K_p$$

- Para que el sistema no tenga error de posición, ha sido necesario incluir la parte integral, ya que con un control proporcional es imposible conseguir error de posición nulo en sistemas de 1º orden. A continuación, se muestra la fórmula para el cálculo del error (de posición) de seguimiento de señales:

$$Ep = \frac{1}{1 + \lim_{n \to 0}\left(G_r(s)G_p(s)\right)}$$

Para conseguir que el error de posición sea cero ha sido necesario conseguir que el denominador de la ecuación se anulase, cosa que se puede conseguir añadiendo un integrador. Por esta razón ha sido necesario añadir control integral (añade un integrador).

- Por último, no va a ser necesario incluir la parte derivativa, ya que con un PI se cumplen las especificaciones dinámicas. Por tanto, en este caso tampoco ha sido necesario utilizar un PID en el control de velocidad. Esto es debido a que en este caso el Lugar de las raíces en bucle cerrado (con un PI) ocupa únicamente parte del eje real negativo.

Por tanto, para conseguir que se cumplan las especificaciones establecidas, es necesario que el regulador contemple parte proporcional y parte integral, es decir, un regulador de tipo PI.

$$PI: G_{R\_PI}: (s) = \frac{K_p(s + {}^{1}/_{Ti})}{s} = \frac{K(s + a)}{s}$$

2.2.2.2.- Diseño del regulador mediante la herramienta rltool de Matlab

Como ya se ha comentado en el apartado de control de posición, para el cálculo de El lugar de las raíces se ha utilizado la herramienta rltool de Matlab.

Se ha introducido la Función de transferencia (FDT) del modelo obtenido mediante Ident en Matlab. Una vez hecho, se ha procedido a abrir la herramienta rltool tecleando su nombre en la venta del Command Window del programa.

```
>> s=tf('s');
>> g=29.472/(1+0.17689*s)

g =

      29.47
  ------------
  0.1769 s + 1

Continuous-time transfer function.

>> rltool(g)
```

FDT DEL MODELO DEL MOTOR DE VELOCIDAD

comando para abrir el rltool

Figura 29: Código en el Command Window de Matlab.

Una vez se ha abierto el rltool, el programa muestra el Lugar de las raíces del modelo con un regulador proporcional (Figura 30):

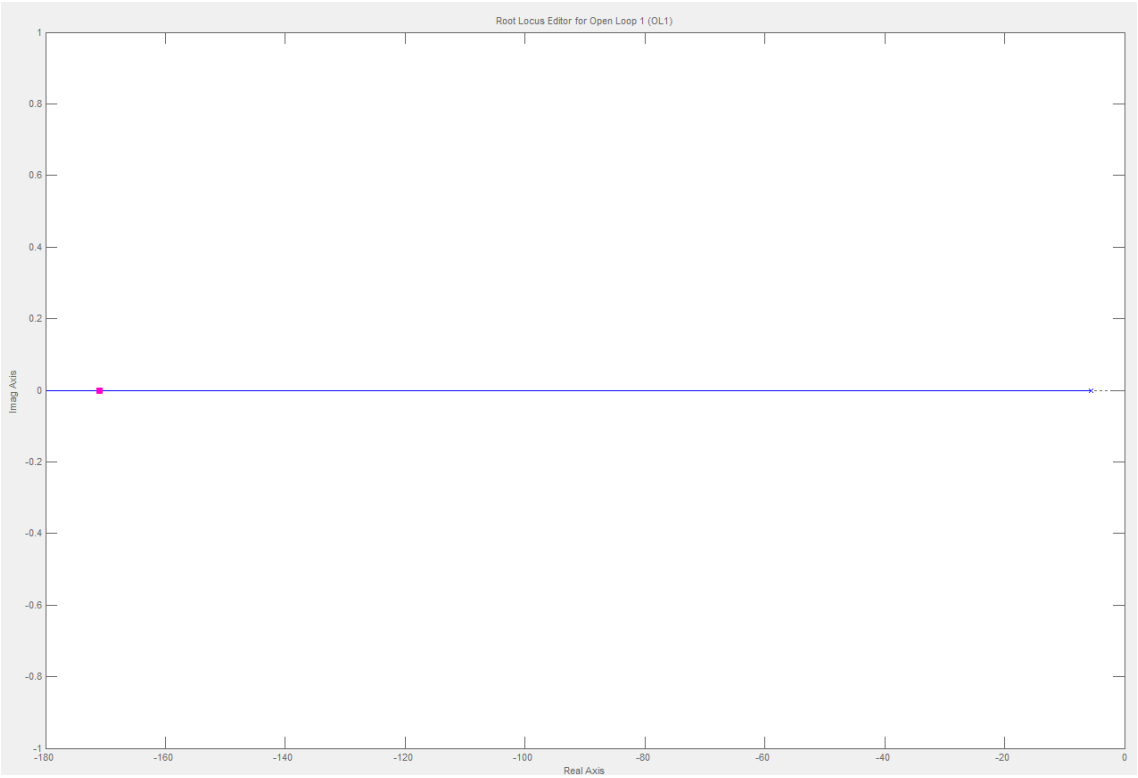


Figura 30: Lugar de las raíces del modelo de velocidad del motor.

Para poder realizar el cálculo del regulador, es necesario añadir los elementos que aportan el PI, es decir, se ha añadido un cero real y un integrador.

El regulador se ha diseñado por el método de la cancelación: se ha cancelado con el cero del regulador el único polo del proceso, el cual está situado en -5.6529 ($s=-1/\tau=1/0.1769$).

A continuación, se muestra el Lugar de la raíces de la FDT del sistema en bucle cerrado con un regulador PI:

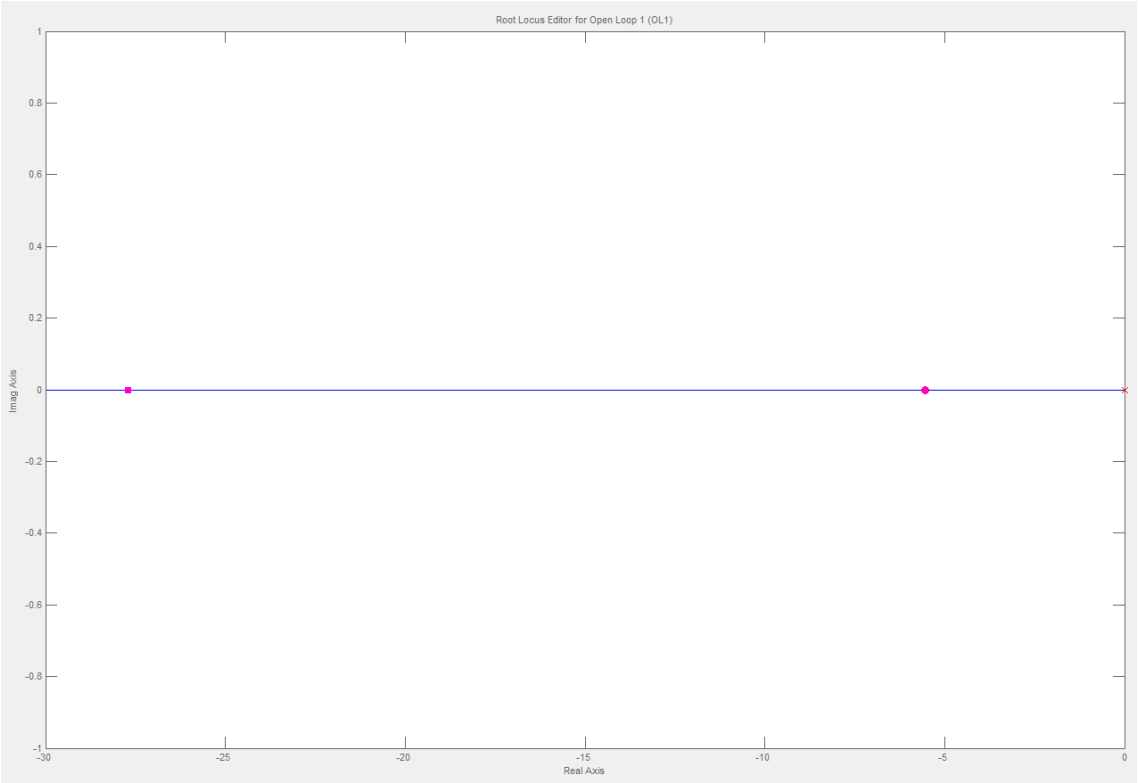


Figura 31: Lugar de la raíces en bucle cerrado con un regulador PI.

En la Figura 31 se puede observar como el Lugar de las raíces con un PI queda prácticamente igual que el Lugar de las raíces con un regulador proporcional (Figura 30). También se puede afirmar que no va a ser necesario añadir un regulador derivativo, ya que todo el Lugar de las raíces está sobre el eje real.

Finalmente, se ha decidido colocar el polo en -73.7, dando lugar a una ganancia de 2.5, quedando el regulador de la siguiente manera:

$$G_r(s) = \frac{2.5(s + {}^{1}/_{0.18})}{s}$$

Calculado el regulador, se procede a la validación del mismo mediante Simulink.

2.2.2.3.- Validación del regulador

Del mismo modo que en el control de posición, se ha realizado la simulación en un diagrama de bloques en Simulink.

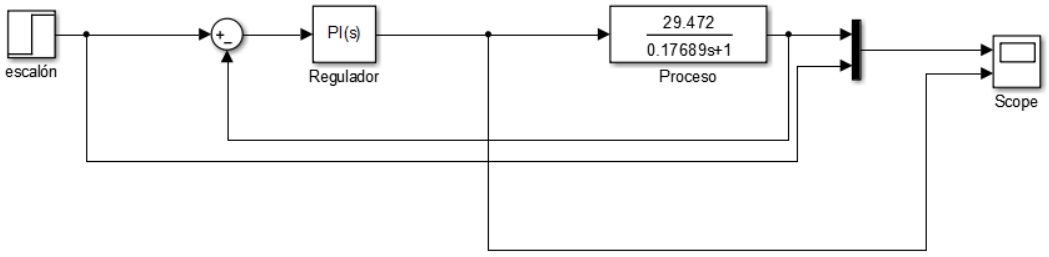


Figura 32: Diagrama de bloques en Simulink..

Se ha realizado un escalón de 50 rpm en la referencia y se ha simulado el control, representado en el *Scope,* la velocidad, la tensión y la referencia.

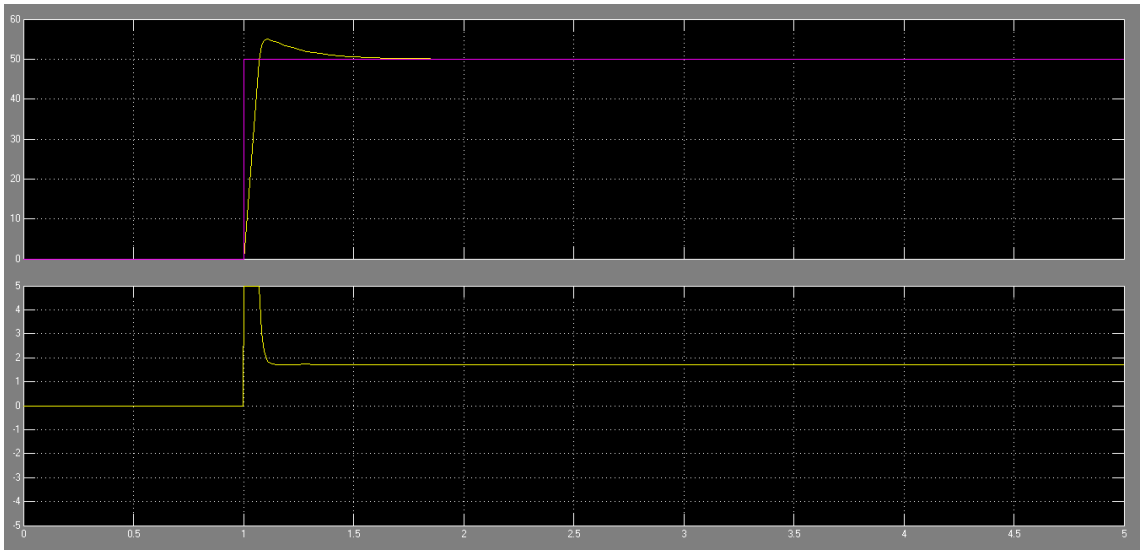


Figura 33: Simulación en Simulink del control de velocidad.

Observando los resultados, se han obtenido las siguientes conclusiones:

- Debido a la saturación de la acción de control no se ha podido alcanzar el tiempo de establecimiento teórico impuesto por la parte real del polo del sistema, no obstante, al tratarse de un regulador agresivo se consigue que el tiempo de establecimiento sea relativamente pequeño.
- Se observa que no existe error de posición, lo que significa que el control integral funciona adecuadamente.
- Existe un cierto nivel de sobreoscilación en la simulación, no obstante, ésta no concuerda con la validación experimental en LabVIEW (Figura 33). Esto se debe a que el modelo obtenido es aproximado y no tiene en cuenta las fricciones iniciales del ensayo que presenta el motor al arrancar, ni la zona muerta. Además, el PID implementado en LabVIEW presenta la función de Anti-Windup, la cual congela la acción integral cuando satura la acción de control, cosa que tampoco tiene en cuenta el diagrama de bloques de Simulink. A pesar de que la validación simulada presenta disconformidades, la validación experimental prevalece frente a esta, por tanto se da el regulador por bueno.

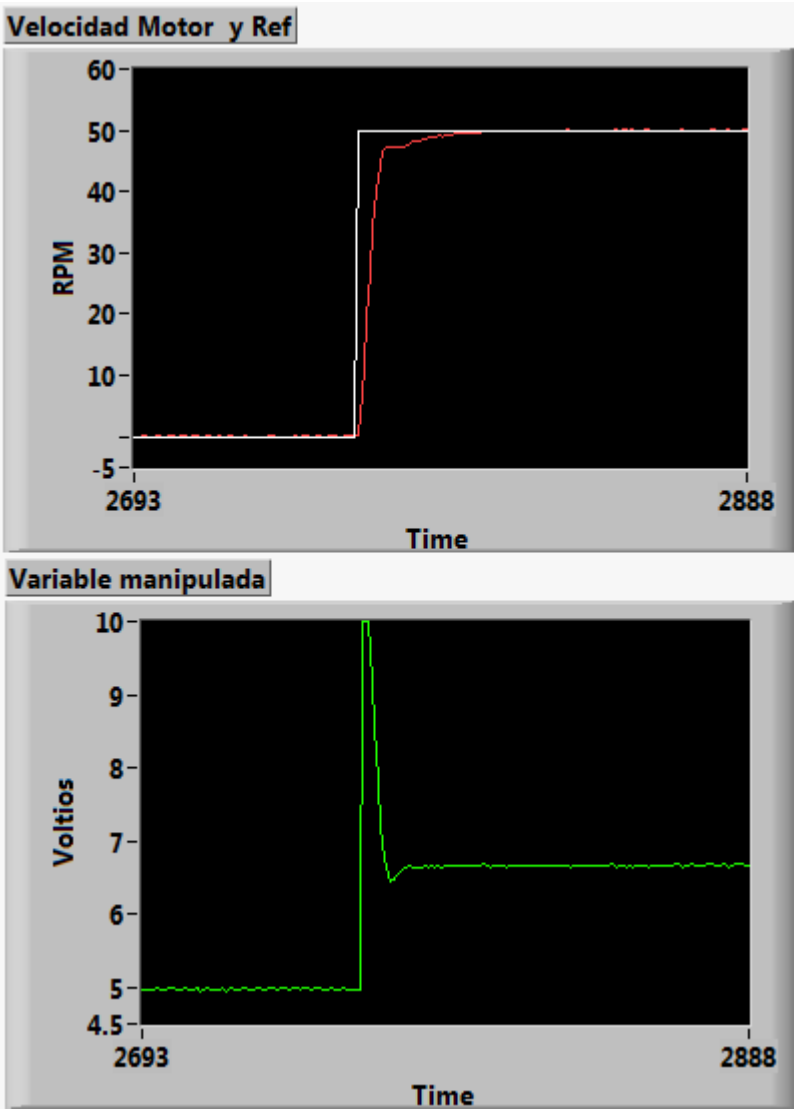


Figura 34: Validación experimental del control de velocidad.

3.- BIBLIOGRAFÍA

B.C.Kuo. (1996). *Sistemas de Control Automático. Séptima Edición.* Pretice Hall.

ISA. (2012-2013). *Control de Posición y velocidad de un motor de corriente continua.* Valencia: UPV.

ISA. (2012-2013). *Identidicación de motor de corriente continua.* Valencia: UPV.

ISA. (2012-2013). *Unidad didactica 5, Sistemas automaticos.* valencia: UPV.

ISA. (2013-2014). *Programación mediante LabVIEW para aplicacion de identidicación y control.* Valencia: UPV.

ISA. (2013-2014). *Unidad didáctica 2, Tecnología Automática.* Valencia: UPV.

ISA. (2013-2014). *Unidad didáctica 4, Tecnología Automática.* Valencia: UPV.

K.Ogata. (2003). *Ingeniería de Control Moderna. 4ª edición.* Pearson Prentice Hall.

Miquel, M. V. (2012-2013). *Servosistema Digital de Corriente Continua 33-002(SFT254).* Valencia.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO DE UNA APLICACIÓN PARA LA IDENTIFICACIÓN Y CONTROL DE UN MOTOR DE CORRIENTE CONTINUA MEDIANTE LABVIEW

Documento Nº3: Anexo de Programación

AUTOR: Dotor Alonso, Luis

TUTOR: Herrero Durá, Juan Manuel

Curso Académico: 2013-14

CONTENIDO

1.- INTRODUCCIÓN 2

2.- MOTIVACIÓN Y OBJETIVOS 2

3.- DESCRIPCIÓN DE LA SOLUCIÓN 3

3.1.- Descripción del programa 3

3.2.- Módulo de control 4

3.2.1.- Introducción al bucle de control Principal 4

3.2.2.- Desarrollo de un PID 4

3.2.3.- Lectura y escritura de datos 9

3.2.4.- Almacenamiento de datos 10

3.2.5.- Desarrollo de un sensor virtual 11

3.2.6.- Desarrollo de un filtro para el sensor de posición 13

3.2.7.- Bucle de control secundario 15

3.3.- Módulo de automatización 16

3.3.1.- Descripción del programa del automatismo 16

3.3.2.- Lectura del Fichero de puntos 17

3.3.3.- Análisis de etapas y transiciones 19

3.4.- Interfaz gráfica 25

3.4.1.- Modelo 3D del robot 25

3.4.1.1.- Modelado de las piezas 25

3.4.1.2- Manipulación de la posición 27

3.4.2.- Representación XY 30

3.4.3.- Leds 31

3.5.- Protocolo de comunicación TCP 32

4.- BIBLIOGRAFÍA 34

1.- INTRODUCCIÓN

En el presente documento se pretende describir y explicar todo el código de programación que conforma el programa. Este anexo está pensado como un manual explicativo para el caso en el que un programador que adquiera el programa pueda interpretarlo, e incluso modificarlo, con el fin de mejorarlo. No obstante, hay que tener en cuenta que el documento puede resultar un tanto complicado para un usuario inexperto del programa, sobre todo si este no tiene conocimientos sobre el lenguaje de programación gráfico de LabVIEW.

En el manual de usuario se explicará el funcionamiento del programa a nivel de interfaz gráfica, de tal manera que un usuario puede aprender a utilizarlo sin necesidad de tener conocimientos de programación.

2.- MOTIVACIÓN Y OBJETIVOS

Este documento pretende cumplir una serie de objetivos, los cuales se listan a continuación:

- Realizar una descripción detallada de la solución adoptada, pero enfocada a lo que ha sido la programación en LabVIEW.
- Mostrar las diferentes partes funcionales de las que está compuesta la aplicación desarrollada (módulo de automatización, módulo de control y diseño gráfico).
- Ser guía y manual de futuros programadores dispuestos a entender el programa.
- Asentar los conocimientos de programación en LabVIEW adquiridos durante el desarrollo del proyecto.

El siguiente documento muestra gran parte de los conocimientos adquiridos en la programación gráfica de LabVIEW, lo cual refleja una fuerte motivación, dado que es un programa creado por National Instruments para el desarrollo de aplicaciones de propósito general. La experiencia adquirida en el código de LabVIEW motiva a seguir realizando ampliaciones futuras para el programa que se está describiendo, o incluso a desarrollar nuevas aplicaciones basadas en este lenguaje de programación gráfico.

3.- DESCRIPCIÓN DE LA SOLUCIÓN

El siguiente apartado cohesiona cada una de la partes en las que está dividido el programa a nivel de programación.

3.1.- Descripción del programa

La aplicación está dividida, fundamentalmente, en cuatro partes:

1. Módulo de control: el control de los cuatro motores del robot está separado en dos bucles: un primer bucle situado en el programa principal, el cual controla la posición de los motores X e Y, y un segundo bucle, situado en el programa secundario, donde se controla la posición del motor X y la velocidad del motor del cabezal. Estos dos bucles poseen elementos comunes muy parecidos, de hecho, en caso de haber podido utilizar una tarjeta de adquisición con cuatro salidas, se hubiera construido todo el control en un mismo bucle.
2. Módulo de automatización: esta parte del programa se encarga de gestionar el automatismo, el cual está compuesto de un bucle que controla todas y cada una de las etapas que ha de cumplir y dos más para gestionar la lectura de datos.
3. Módulo gráfico: esta parte incluye el código de la representación XY, el parpadeo de leds y el modelo 3D del robot. Este último se trata de un bucle de control encargado de actualizar los movimientos del modelo 3D que lo representa. Está formado por geometrías simples desplazadas relativamente unas respecto de otras.
4. Módulo de comunicación: por último, existe una cuarta parte del programa que se encarga de realizar la comunicación a través de Ethernet, la cual está dividida en un programa cliente y otro servidor. Esta comunicación ha sido necesario hacerla por cuestiones de disponibilidad de Hardware, ya que solo se disponía de tarjetas de adquisición con dos salidas, por tanto, se han usado dos tarjetas y se han realizado dos programas distintos, pero comunicados.

3.2.- Módulo de control

Existen cuatro controles distintos en este proceso (control X, Y, Z y control de la velocidad) los cuales han sido divididos en dos bucles de control. Ambos bucles son muy semejantes, por tanto se procederá a explicar el bucle de control principal, encargado de controlar la posición X e Y. Posteriormente, se hará alguna aclaración sobre el bucle de control de velocidad y posición Z.

3.2.1.- Introducción al bucle de control Principal

Este primer bucle consta de una serie de elementos, los cuales se han estudiado de manera independiente, ya que realizar el estudio del bucle a nivel global resultaría un tanto caótico. Este bucle consta de:

- Un algoritmo de control tipo PID que actúa como regulador del proceso.
- Una parte encargada de la lectura y escritura de datos mediante drivers de la tarjeta de adquisición.
- Programación para el almacenamiento de datos para la identificación.
- Un sensor virtual que actúa como un contador de vueltas, elaborado a partir del sensor del eje del potenciómetro.
- Un filtro para la salida del sensor de posición.
- Elementos generales que se comunican con el módulo de automatización.

3.2.2.- Desarrollo de un PID

Se ha realizado un algoritmo para que actúe a modo de un regulador de tipo PID. Esta es un parte importante en la programación de control, dado que se han utilizado tres controladores de tipo PD y uno de tipo PI.

El objetivo es conseguir mediante un Formula Node un código escrito en C que sirva para el cálculo de la ecuación de un PID.

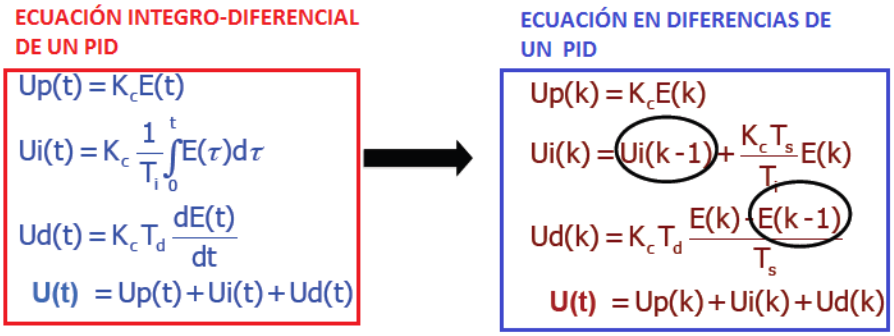


Figura 1: Ecuación integro-diferencial y en diferencias de un PID.

El código implementado en LabVIEW se puede visualizar en la Figura 2:

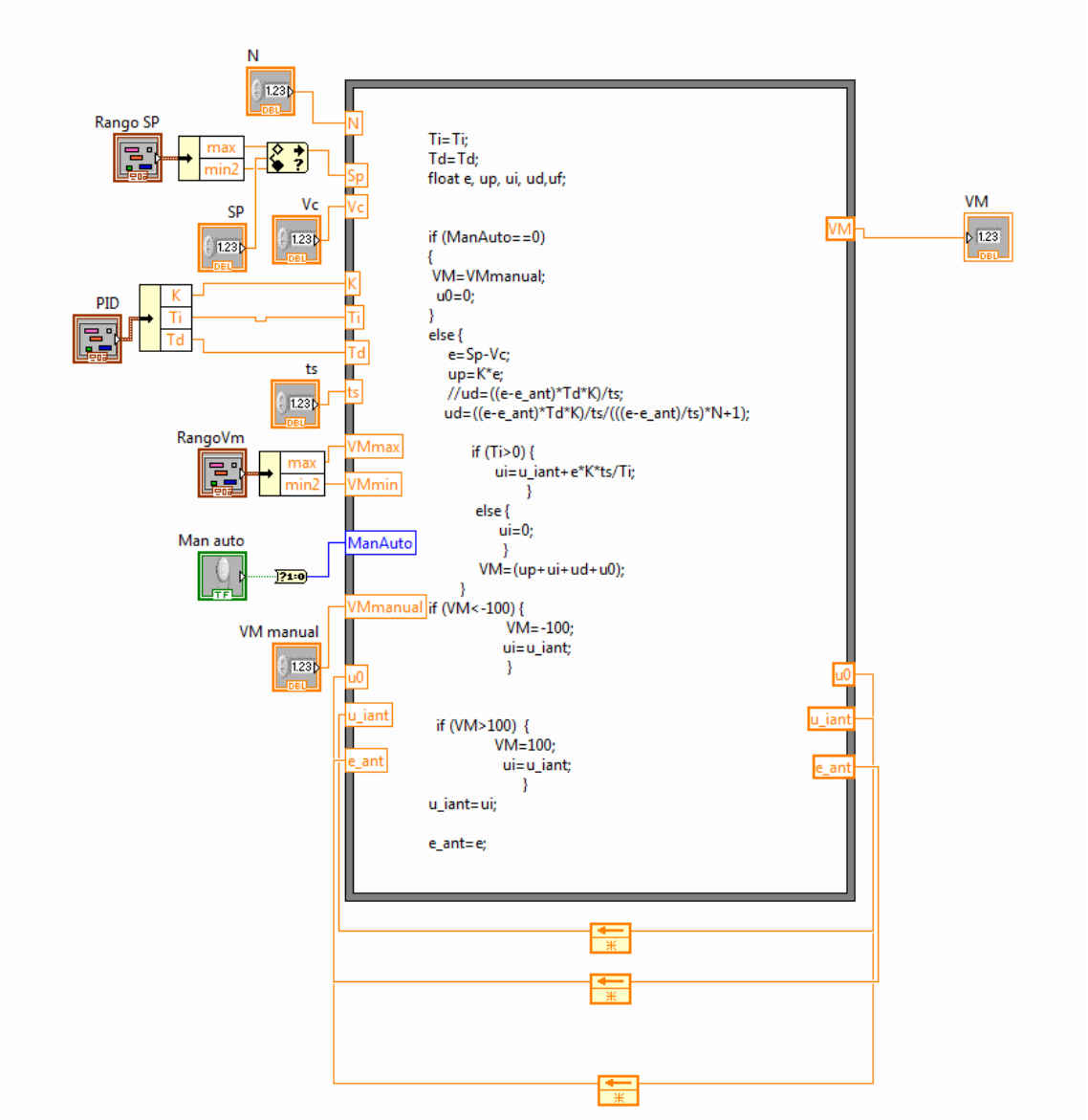


Figura 2: Codificación del PID.

El código consta de nueve entradas para editar la configuración disponible y de una salida de donde se obtiene la variable manipulada, es decir, la acción de control. En la Figura 3 se observan más claramente las entradas y salidas de este subprograma.

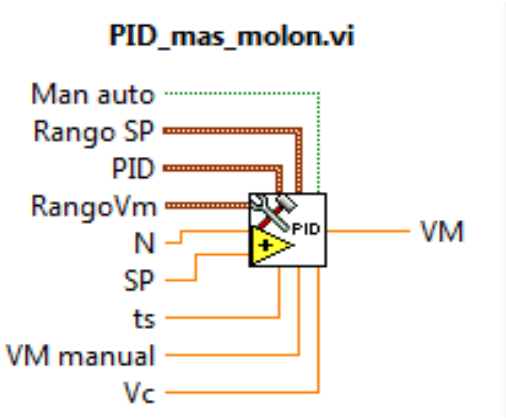


Figura 3: Cableado del PID.

Entre las diferentes entradas podemos encontrar:

- Man auto: Permite conectar un booleano que conmute entre control manual y control automático.
- Rango SP: Establece mediante un cluster los límites superior e inferior de la referencia que va a seguir el proceso.
- PID: Permite introducir los parámetros Kp, Td y Ti necesarios para configurar el regulador y realizar el control automático.
- RangoVm: Establece mediante un cluster el rango en el que puede variar la variable manipulada.
- N: Coeficiente para filtrar la acción derivada.
- SP: Permite conectar un control para establecer la referencia que va a seguir el proceso en cada instante.
- ts: Esta entrada permite configurar el intervalo de tiempo en el que se realiza cada iteración de control. Para que funcione correctamente, se introduce el período de ejecución del bule de control.
- Vm: permite conectar un control para establecer la variable manipulada en caso de estar en control manual.
- Vc: en esta entrada se introduce la variable a controlar, que en este caso será la posición o la velocidad de un motor.

Este algoritmo de control está basado en la ecuación en diferencias de un PID. En el caso concreto en el que se haya activado el control automático, el algoritmo de control del PID se podrá en marcha realizando una serie de operaciones que se resumen a continuación:

1. Se calcula el error a partir de la variable controlada y la referencia.

```
e=Sp-Vc;
```

2. Se calcula la acción proporcional a partir del error.

```
up=K*e;
```

3. En caso de haber introducido control derivativo, se calcula la acción correspondiente, teniendo en cuenta el error del instante actual, el error del instante anterior, y el coeficiente N.

```
ud=((e-e_ant)*Td*K)/ts/(((e-e_ant)/ts)*N+1);
```

4. En caso de haber seleccionado control integral, se calcula dicha acción a partir del error del instante actual y de la acción integral del instante anterior. Hay que tener en cuenta que, en caso de no querer acción integral, seleccionar 0 en el apartado Ti de la entrada PID podría dar lugar a una acción integral infinita, ya que en la ecuación, el coeficiente Ti va dividiendo. Por esta razón se ha implementado un if else que diferencia entre los casos en los que se haya seleccionado acción integral nula o no.

```
if (Ti>0) {

ui=u_iant+e*K*ts/Ti;}

else {

ui=0;}
```

5. Se suman cada una de las acciones calculadas anteriormente en una única acción total:

```
VM=(up+ui+ud+u0);
```

6. Se limita la acción de control en caso de que haya sobrepasado los límites del actuador (fuente de alimentación) y se congela la acción integral del instante actual, para evitar así el fenómeno de Windup (efectuando así un control anti Windup).

```
if (VM<-100) {
VM=-100;
ui=u_iant;}
if (VM>100)  {
VM=100;
ui=u_iant;}
```

7. Por último, se actualizan las variables necesarias para realizar la siguiente iteración de control:

```
u_iant=ui;
e_ant=e;
```

3.2.3.- Lectura y escritura de datos

Para poder realizar una correcta comunicación con el proceso, es necesario implementar en LabVIEW los drivers correspondientes. Se han utilizado dos Drivers: uno para la escritura y otro para la lectura de datos:

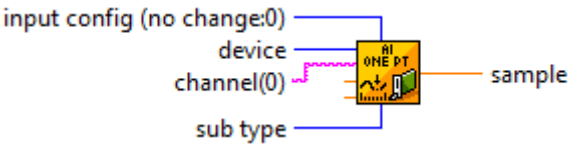


Figura 5: Driver de lectura.

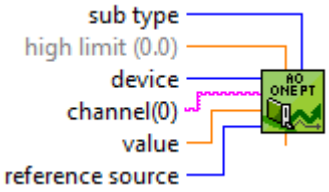


Figura 4: Driver de escritura.

En ambos drivers se ha hecho uso de la entrada “device” y “chanel(0)”, que sirven para especificar el dispositivo y el canal de escritura (o lectura) respectivamente. Además, en el driver de escritura se ha utilizado la entrada “high limit” para asegurar que no se escribe una tensión más grande que la máxima y en el driver de lectura se ha utilizado la salida “simple” para obtener información de los sensores.

En el caso de la lectura en el bucle principal (donde se controlan los motores X e Y), la salida “simple” proporciona información del sensor del eje de motor, el cual devuelve un valor entre [-10,10] (mv), por tanto, se ha multiplicado la salida por 18 para realizar la conversión a grados.

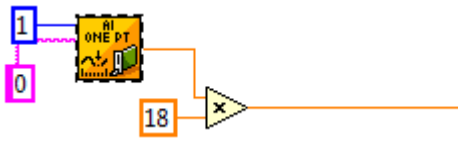


Figura 6: driver de lectura implementado en el código.

En el caso del driver de escritura, se ha realizado una conversión de la variable manipulada devuelta por el PID en términos porcentuales (menos cien y cien) para adaptarlo a la tarjeta de adquisición.

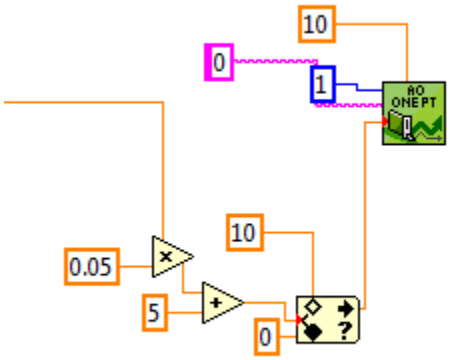


Figura 7: Driver de escritura implementado en el código.

3.2.4.- Almacenamiento de datos

Para poder realizar la identificación es necesario disponer de un fichero con los datos de un ensayo. Por tanto, el bucle de control presenta una parte del código que se ha dedicado al almacenamiento de los datos.

En la Figura 8 se observa cada uno de los cables que transportan consigo un dato de tiempo, tensión y posición del motor en cada periodo de ejecución del bucle. Estos datos salen del bucle y se almacenan en tres vectores mediante la función “build array”. Finalmente, cuando el bucle acaba, el programa guarda un fichero gracias a la función “write to spreadsheet file”.

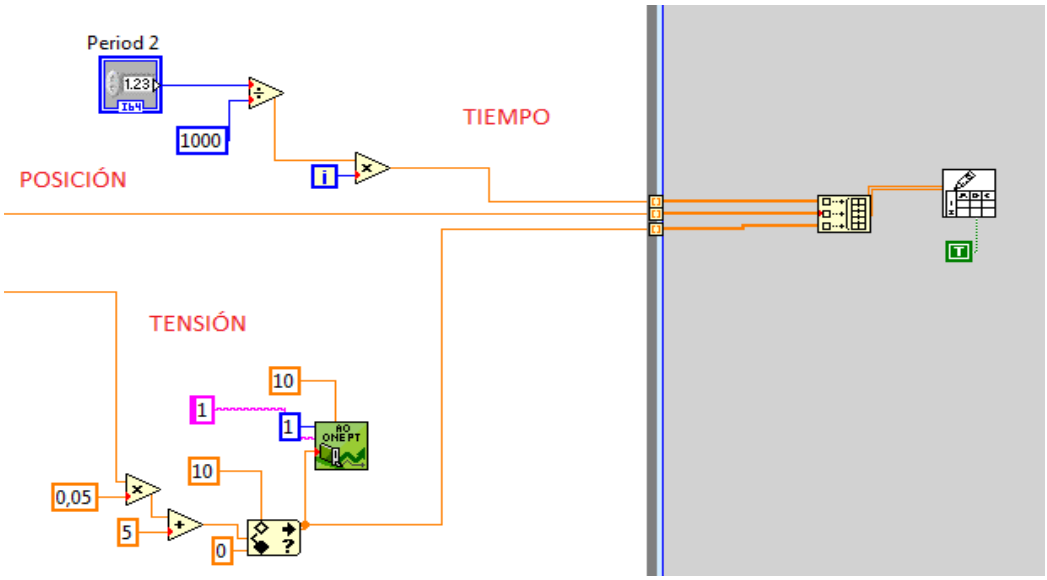


Figura 8: Código de almacenamiento de datos.

3.2.5.- Desarrollo de un sensor virtual

Debido a que la salida del sensor del eje del motor devuelve una función triangular, se ha tenido que implementar en el código un sensor virtual que contase las vueltas y que transformase la salida a partir de este sensor real.

Este sensor devuelve una tensión de [0-10] voltios cuando la posición del eje va de [0-180] grados. De la misma manera, devuelve de [0-10] voltios cuando la posición va de [0-180].

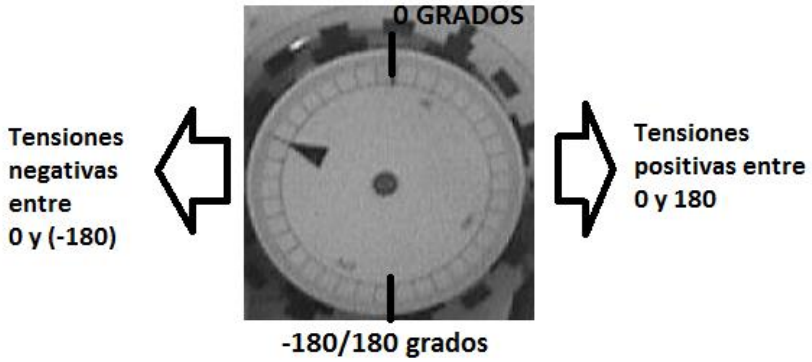


Figura 9: sensor del eje del motor.

El algoritmo se ha diseñado de manera que reconozca cada paso que hace por 180 (o -180), memorizando así el número de vueltas que realiza. Para implementar esta codificación, el programa tiene que comparar los grados devueltos por el sensor real en el instante actual y en el instante anterior. El código realiza las siguientes comprobaciones:

- Si la posición anterior es positiva y la posición actual es negativa, significa que el motor ha pasado de grados positivos a negativos. Esto se reduce a dos posibilidades: una cuando el motor pasa por 0 y otra cuando pasa por 180. Si además la posición anterior está entre [180,160] y la posición actual está entre [-160,-180], entonces el paso del motor se ha realizado por 180. En este caso el contador de vueltas suma una vuelta.
- De la misma manera, si la posición anterior es negativa y la posición actual es positiva, significa que el motor ha pasado de grados negativos a positivos (también dos posibilidades). Si además la posición anterior está entre [-180,-160] y la posición actual está entre [160,181], entonces el paso del motor se ha realizado por 180. En este caso el contador de vueltas resta una vuelta.

Una vez resuelto el asunto de cambio de vueltas, el algoritmo debe modificar la entrada para transformar la onda triangular en una recta. Para ello, el código debe coger la salida del sensor real y sumar 360 grados en función del número de vueltas. De esta manera, cuando el programa reciba los primeros grados entre [0 y 180] (contador=0), sumará 0*360 devolviendo íntegramente los grados que son, pero cuando se produzca el paso por 180, contará un vuelta y le sumará 360 al rango de [-180,0], generando un nuevo rango de [180-360]. De esta manera, conforme se vayan contando las vueltas, el programa irá representado una recta a base de juntar (enlazar) rangos de valores. En la Figura 10 se puede observar toda la codificación resultante de implementar lo anteriormente descrito.

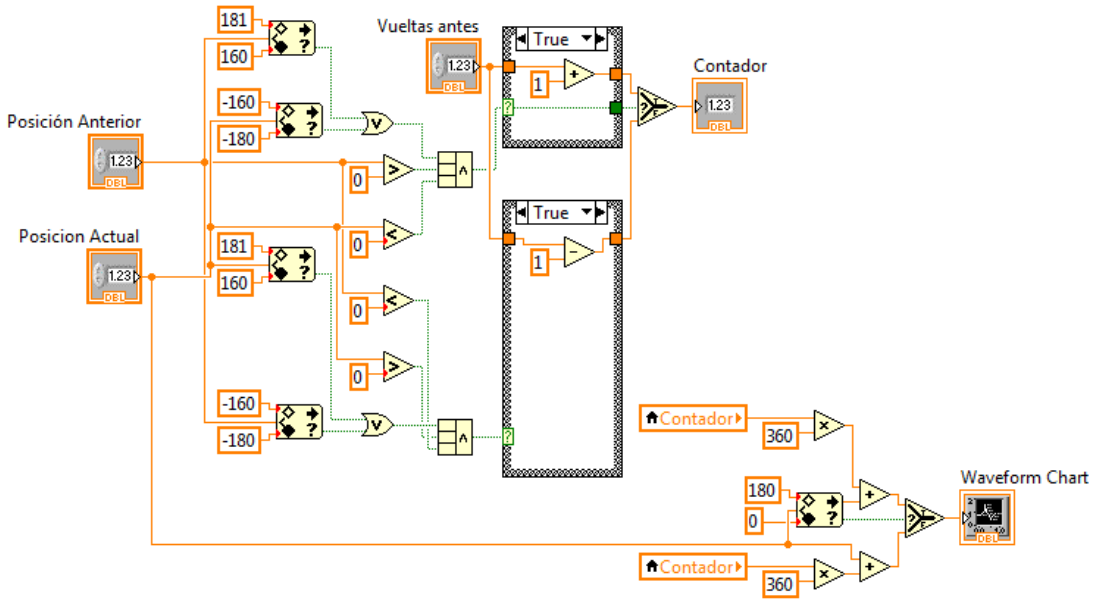


Figura 10: Algoritmo del sensor virtual.

3.2.6.- Desarrollo de un filtro para el sensor de posición

Debido a la mala precisión de los sensores reales y su gran variabilidad en la medida, el sensor virtual presenta un ruido en la salida que impide realizar buenos ensayos para la identificación, ya que falsea los datos. En la Figura 11 se puede presenciar esta discontinuidad:

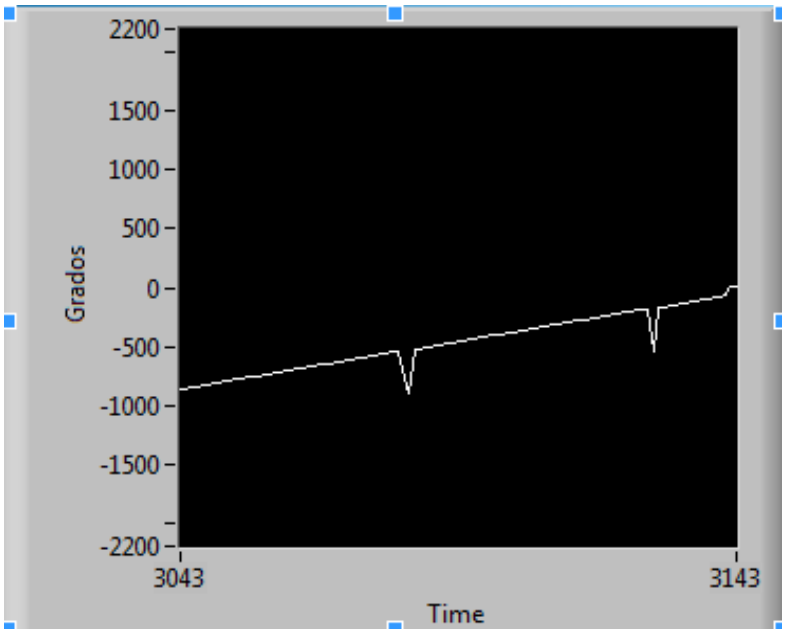


Figura 11: Posición del Robot antes de filtrarla.

Para solucionar este problema se ha implementado un filtro derivativo. Este filtro comprueba la posición del instante anterior y la compara con la posición del instante actual, de manera que si la variación de la posición supera un cierto umbral, la posición de ese instante toma el valor anterior, más un incremento obtenido a partir de una media.

Para implementar esta función, el código realiza una serie de pasos:

- Comprueba si la posición del instante actual es mayor o menor que la del instante anterior, ya que el motor puede estar aumentando o disminuyendo la posición.
- Verifica si la diferencia de valores está en el rango de [-13,13], lo cual implica que la variación es correcta, en ese caso la salida pasa a ser íntegramente del valor actual.

- En caso de que la posición actual sea mayor que la anterior y que la diferencia esté fuera de rango, el valor actual pasa a ser el anterior más 10,5.
- En caso de que la posición actual sea menor que la anterior y que la diferencia esté fuera de rango, el valor actual pasa a ser el anterior menos 11,5.

Cabe destacar que los valores de 10,5 y -11,5 se han obtenido a partir de una media aritmética de variación de puntos. En la Figura 12 se muestra la codificación resultante:

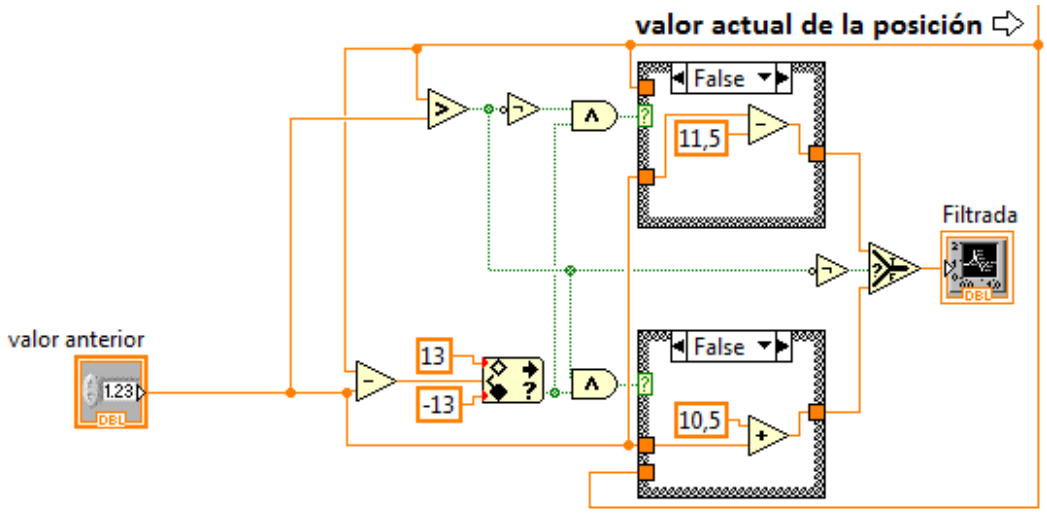


Figura 12: Filtro derivativo.

3.2.7.- Bucle de control secundario

El bucle de control secundario se encarga de controlar la posición del motor Z y la velocidad del motor del cabezal. Los elementos que forman este bucle son prácticamente iguales a los del bucle principal, especialmente en el control del motor Z. No obstante, en lo que respecta al control de velocidad, existen ciertos cambios o diferencias, las cuales se describen a continuación:

- El sensor que se va a utilizar en esta ocasión es un tacómetro para medir la velocidad. La recta de conversión del sensor del eje del motor era simplemente "P$osición = 18 * Voltaje$", ahora la salida "simple" del driver de lectura se hace pasar por la siguiente función:

$$Vel = 15.4569 * Voltaje$$

- En esta ocasión no ha hecho falta utilizar un sensor virtual ni el filtro derivativo (en el control de la posición Z sí), ya que el tacómetro, al hacerlo pasar por la recta de conversión, devuelve íntegramente el valor de la velocidad.
- El almacenamiento de datos es prácticamente igual, lo único que se modifica es que en uno de los vectores ya no se guardan posiciones, si no velocidades.
- Por último, cabe destacar que en la configuración del PID se ha utilizado un regulador de tipo PI, no obstante, esto no afecta al código a nivel de programación. Esta elección se justifica en el anexo de diseño.

Por todo lo demás, el control de velocidad es exactamente igual que el resto de controles de posición, consta de un programa PID de drivers de escritura, de lectura, etc.

3.3.- Módulo de automatización

Una vez se controlan todos y cada uno de los motores, hay que realizar la codificación pertinente para cumplir con las especificaciones del automatismo.

3.3.1.- Descripción del programa del automatismo

El programa se ha diseñado para que sea capaz de reconocer documentos .txt. El usuario podrá, bien con una hoja de cálculo, o bien mediante otros programas, diseñar un fichero estructurado en puntos como el que se describe en el anexo de diseño. El código descompone cada fila de este fichero (fichero de tres columnas y múltiples filas) en tres variables, dos de posición (X e Y) y una de tipo de punto. De esta manera, el automatismo puede proporcionar de manera programada y secuencial una referencia a cada uno de los motores, bien mediante las variables de posición a los motores X e Y, o bien mediante el tipo de broca (tipo de punto), ya que como bien se describe en el anexo de diseño, la referencia del motor Z y del motor de velocidad dependen del tipo de punto. Cada vez que el programa lee un punto nuevo, realiza una secuencia de pasos:

- Ordena a los motores que se muevan hasta el punto inicial (pos x=0, pos y=0). Este punto debe estar incluido en el fichero, ya que es una especificación del automatismo (este paso solo lo realiza la primera vez que lee el fichero).
- Acto seguido, manda a los motores X e Y al siguiente punto de trabajo, hasta que detecta que han llegado a las coordenadas correspondientes.
- A continuación, le manda una referencia de velocidad al motor del cabezal en función del tipo de punto.
- Por último, el programa le ordena bajar y subir al motor Z, también en función del tipo de punto.

A lo largo de todo el proceso el programa contabiliza cada uno de los puntos que va ejecutando, de tal manera que termina cuando esta cuenta alcance el número total de puntos (número de filas).

3.3.2.- Lectura del Fichero de puntos

Para realizar la lectura de los datos, se han utilizado dos bucles, uno para atender la señal de arranque del automatismo y leer el fichero (Botón Run), y otro para descomponer cada una de las filas en las variables necesarias.

El bucle encargado de leer fichero únicamente está pendiente de la pulsación del botón Run (variable OK), para que cuando se pulse, pueda volcar todos estos datos en una matriz (tres vectores).

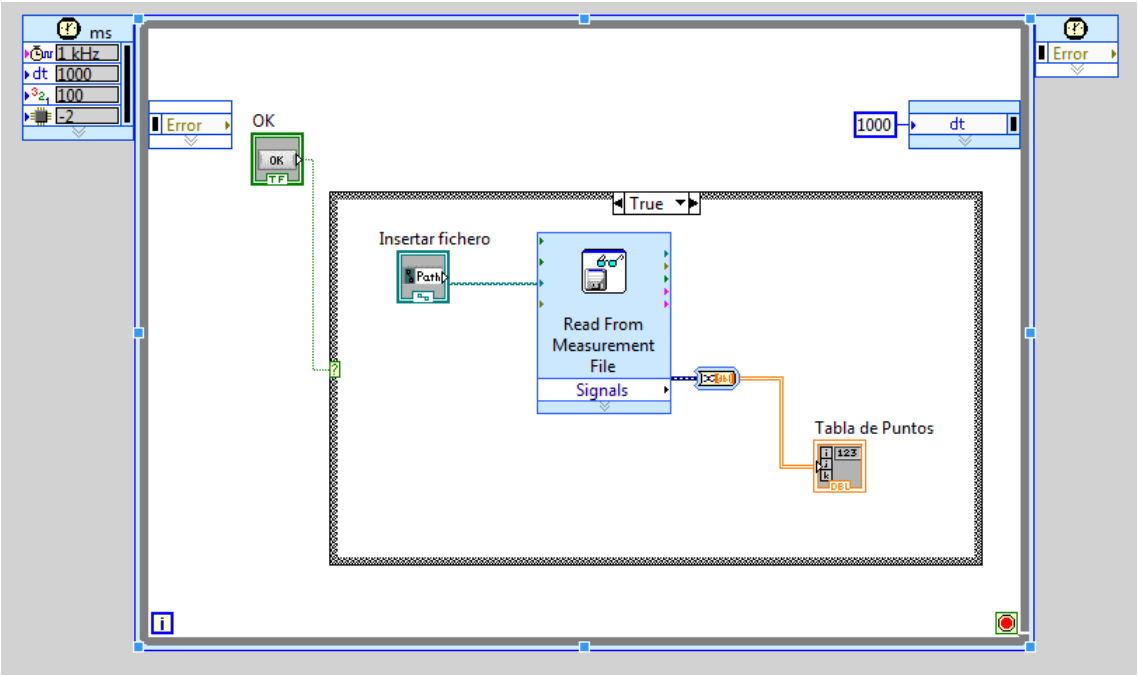


Figura 13: Bucle de consulta de fichero.

El segundo bucle extrae de la matriz anterior cinco variables, mediante las funciones de los array (index array, array size):

- MAX: número de puntos (filas) que posee el fichero.
- POS 1: coordenada x del punto “actual”.
- POS 2: coordenada y del punto “actual”.
- Tipo de broca: tipo de punto “actual”.
- Element: tipo de punto del instante siguiente.

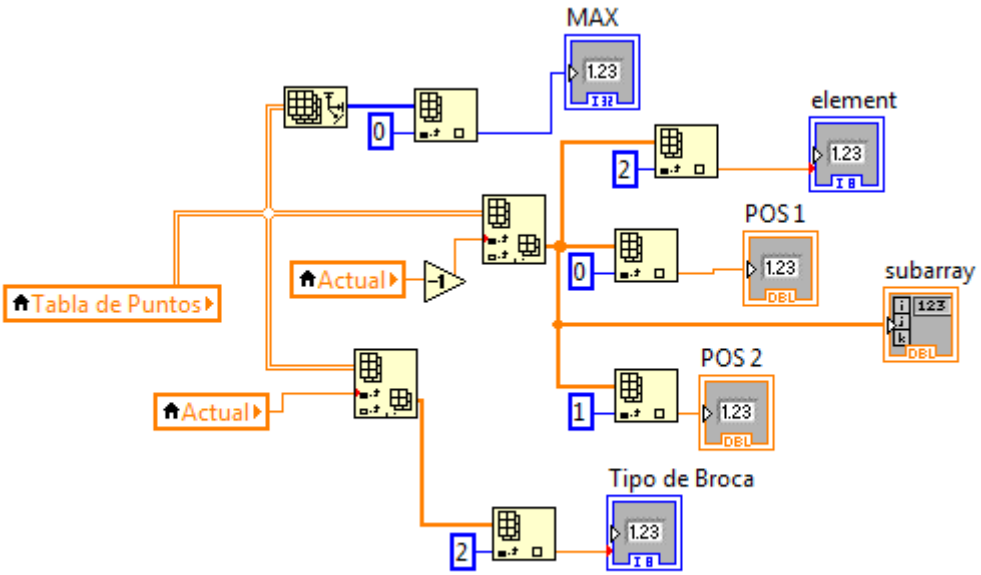


Figura 14: Código de descomposición en variables.

La variable “actual” de este bucle es un contador. Esta actúa a modo de puntero en los diferentes vectores de datos, recorriéndolos de principio a fin. Controlando la variación de esta variable, es decir, incrementándola en el momento justo, se modifica secuencialmente las referencias de los distintos motores, de tal manera que cuando cambia, se modifican todas las variables que dependen de ella (pos1, pos2, tipo de broca y element). Cuando se iguala a la variable MAX, finaliza el automatismo.

Para coordinar todas y cada una de las referencias del programa, se ha creado la variable “indicador de referencia”, de tal manera que en función del valor entero de esta variable, cada uno los PID harán caso a una referencia o a otra. Esto permite que cuando estamos ejecutando el programa (pero no el automatismo) se pueda ordenar a los motores que vayan a la posición deseada desde un control que maneja el usuario. Esto se aplica para cada uno de los motores de manera automática, implementándolo en el código mediante un case Structure de enteros. En la Figura 15 se puede observar que siempre que el automatismo está en marcha (Run= true), la referencia tomará el valor de cero o pos 2 (en este caso) en función del indicador de referencia, sin embargo, cuando el automatismo se apague, la referencia se atenderá a través de un control.

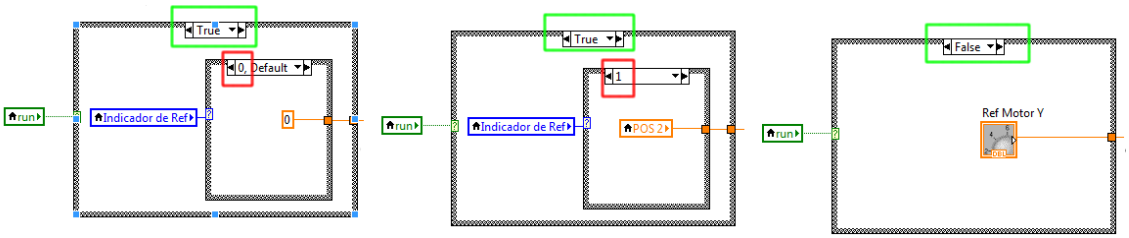


Figura 15: Indicador de referencia.

3.3.3.- Análisis de etapas y transiciones

Una vez se tienen las variables necesarias para la ejecución del automatismo, se necesita una codificación para coordinarlas.

Tal y como se especifica en el anexo de diseño, el automatismo arranca con la pulsación del botón Run, esto activa un case Structure donde se encuentran cada una de las acciones del robot y arranca el bucle de lectura del fichero.

El bucle donde están contenidas las acciones del robot está jerarquizado de la siguiente manera:

1. Un bucle que comprueba la pulsación del botón Run.
2. Un case Structure activado por dicho botón.
3. Un Staked Sequence Structure dentro de otro bucle para secuenciar cada una de las etapas del automatismo.

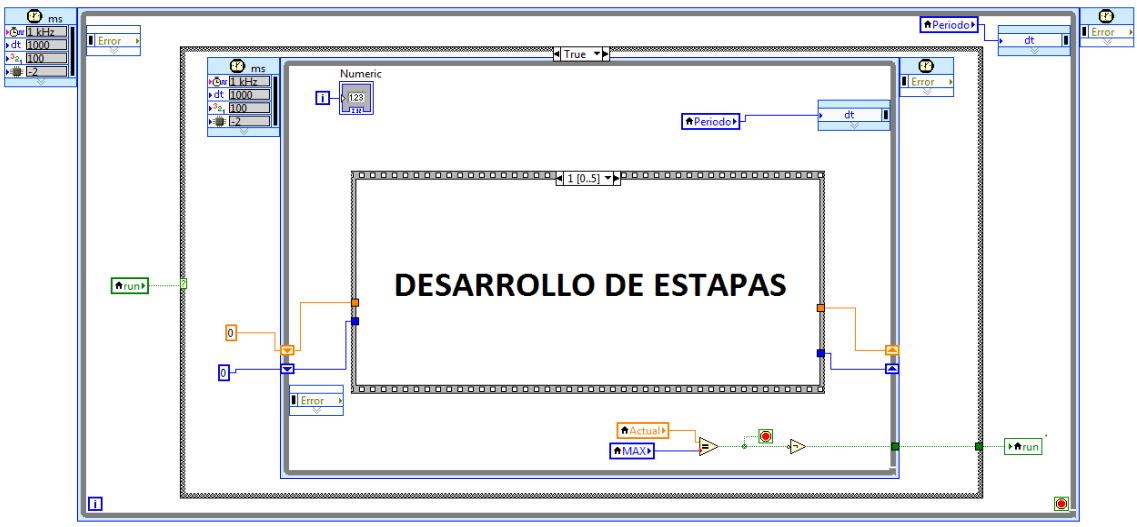


Figura 16: Estructura del automatismo.

A continuación, se va a explicar el código utilizado para ejecutar cada uno de los pasos que realiza el automatismo:

- La primera etapa (etapa 0) es la encargada de hacer volver al motor X e Y al punto inicial. Esto se realiza comparando inicialmente la variable tipo de broca del punto actual con la del punto anterior, de tal manera que si son distintas se activa un condicional (case Structure) que pone el tipo de referencia a cero, es decir, manda al motor X e Y al punto inicial hasta que el propio código detecta que ya han llegado. La manera de detectar si el motor está en el punto correcto, es comprobar en cada instante de tiempo si la diferencia entre la posición del motor y la coordenada cero está dentro de un margen de error razonable. Cuando el programa detecta que ambos motores se encuentran dentro de este margen, activa un condicional que temporiza siete segundos antes de iniciar la siguiente etapa. Cabe destacar que, si se cumplen las especificaciones del automatismo y la primera fila del fichero de datos es la "0 0 0", entonces, cuando se arranque el automatismo siempre pasará por el punto inicial (X=0, Y=0), dado que la primera comparación de tipo de broca será siempre cierta gracias a un shift register que inicializa el tipo de broca del punto anterior a 0, pasando así a la etapa 1, donde los motores X e Y se mueven a sus respectivas coordenadas, que en el caso del primer punto serán las del punto inicial.

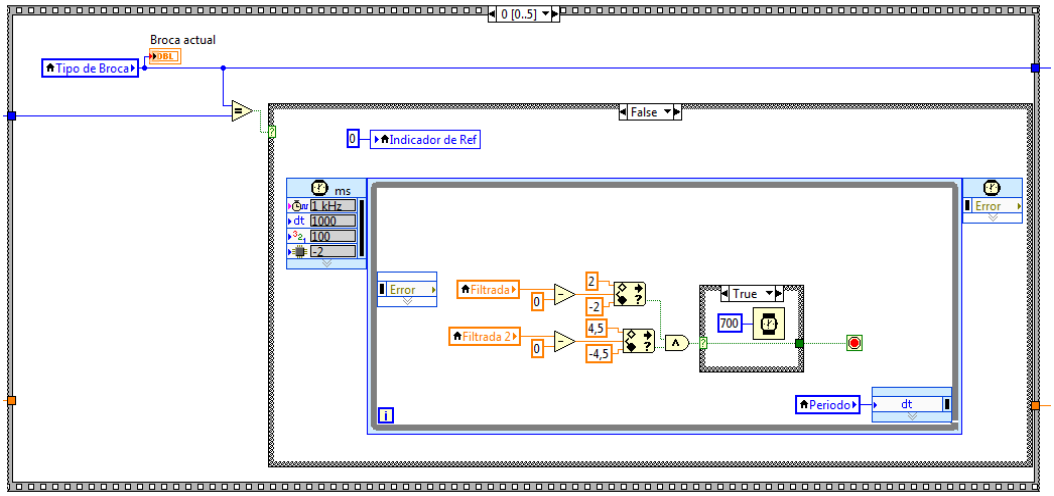


Figura 17: Etapa 0.

- Una vez en la etapa 1, se incrementa en una unidad la variable “actual” y se le ordena a los motores X e Y que vayan a sus respectivas coordenadas, cambiando el valor de “indicador de referencia” a 1. De la misma manera que en la etapa 0, un bucle comprueba en cada instante de tiempo si la diferencia entre la posición de motor y la coordenada correspondiente está dentro de un margen de error razonable (este margen de error depende de la medida de los sensores). Cuando el programa detecta que ambos motores se encuentran dentro de este margen, activa un condicional que temporiza siete segundos antes de iniciar la siguiente etapa.

Incrementar la variable actual antes de que los motores se muevan puede hacer pensar que los motores van a posicionarse en el siguiente punto, creando una descoordinación. No obstante, esto no sucede debido a que las variables “pos 1” y “pos2” obedecen a un puntero que está retrasado una unidad con respecto al puntero de la variable “tipo de broca”.

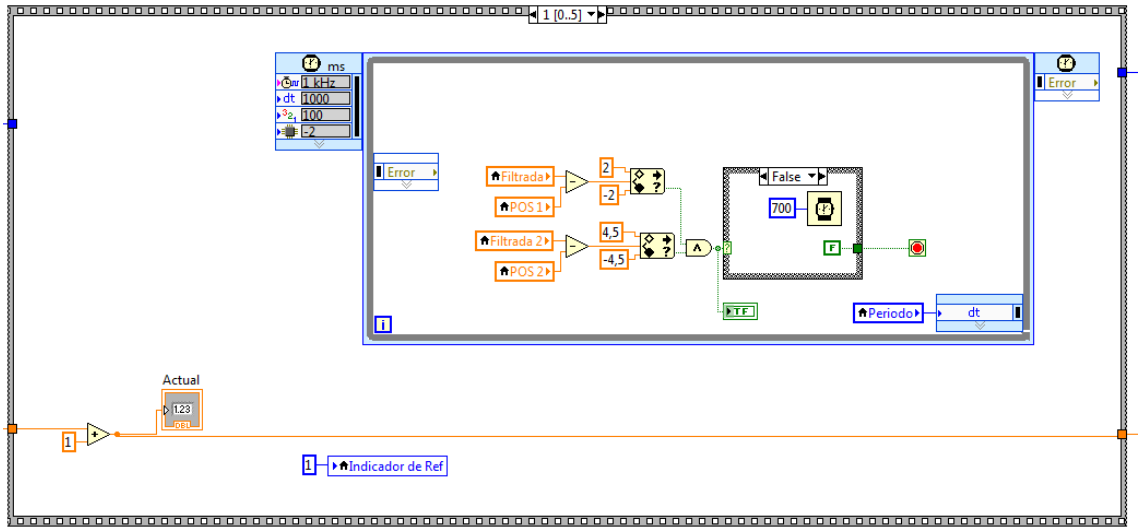


Figura 18: Etapa 1.

- Llegados a la etapa 2, se le ordena al motor del cabezal girar en función del tipo de punto. Si el punto es de tipo 1 o 5 girará a 120 rpm, en cambio, si el tipo de punto es 2, 3 o 4 girará a 80 rpm. Para diferenciar entre puntos se ha utilizado un case Structure accionado por la variable "tipo de broca", donde en cada una de las posibilidades se le ha dado a la variable velocidad el valor correspondiente. Al mismo tiempo, se esperan 6 segundos antes de continuar a la siguiente etapa. Esta variable "velocidad" se manda a través de ethernet para que se escriba en el motor del cabezal.

Para conseguir que el motor del cabezal no girase en el primer y último punto del fichero, se ha excluido mediante un case Structure booleano. Este último punto es un punto no escrito por el usuario en el fichero real, no obstante, es un punto que detecta el programa con coordenadas X=0 e Y=0, que sirve para hacer volver a los motores al punto inicial, sin necesidad de que gire el cabezal y taladre.

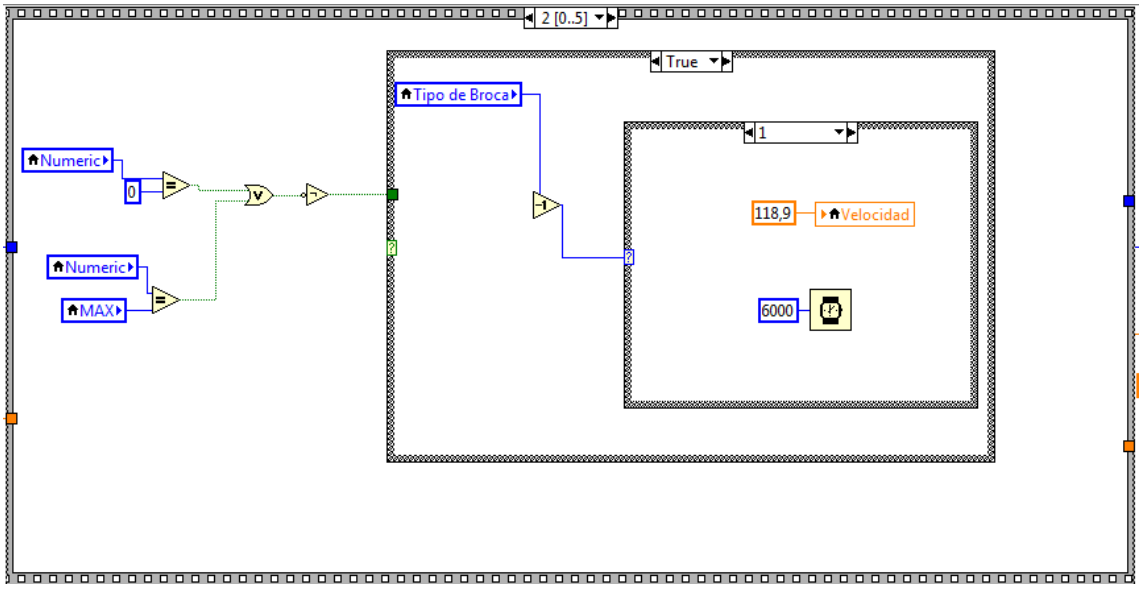


Figura 19: Etapa 2.

- En la etapa 3, se le ordena al motor Z a bajar en función del tipo de punto. Si el punto es de tipo 1, 2, 3 o 4 el motor Z girará 720 grados, en cambio, si el punto es de tipo 5 el motor girará 360. De la misma manera que en la etapa 2, se ha diferenciado entre puntos mediante un case Structure consultando la variable “tipo de Broca”. Al mismo tiempo, se esperan 4 segundos antes de continuar a la siguiente etapa. En este caso la variable que se manda a través de ethernet es “Posición”.

En el caso de esta etapa y de la siguiente (etapa 4), también se ha excluido el punto inicial y el punto final, al igual que en la etapa anterior.

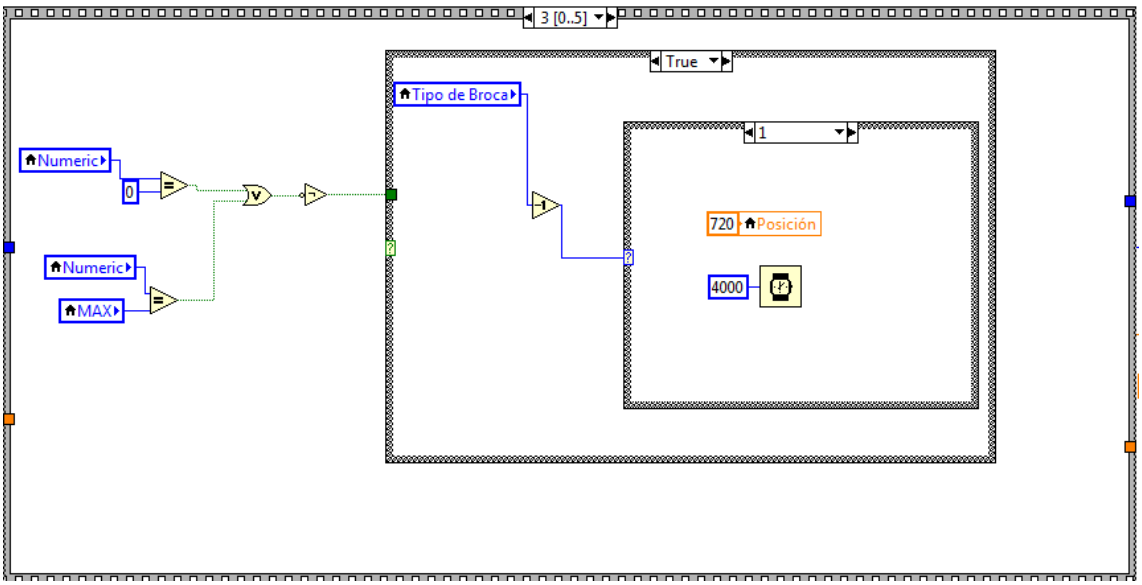


Figura 20: Etapa 3.

- La etapa 4 es exactamente igual que la etapa 3, solo que se hace regresar al motor Z a la posición en la que estaba, es decir, subir otra vez a la coordenada Z=0.

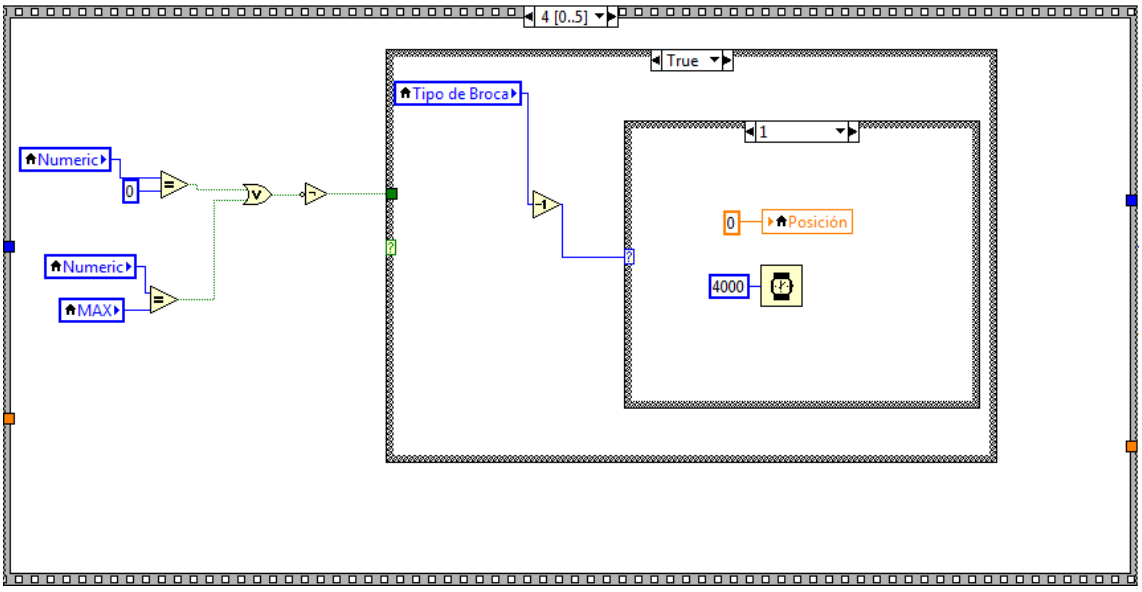


Figura 21: Etapa 4.

- Por último, hay una quinta etapa que sirve para parar el motor del cabezal, es decir, escribir 0 en la variable “velocidad”.

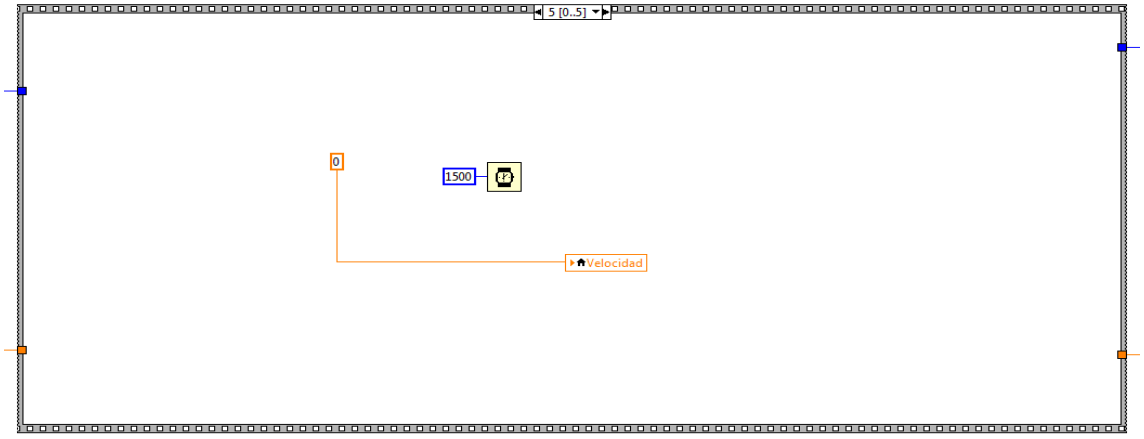


Figura 22: Etapa 5.

Para finalizar, el automatismo comprueba, después de ejecutar cada punto, si la variable “actual” ha alcanzado el valor de la variable “MAX”, lo cual se dará cuando el programa haya ejecutado todos los puntos y haya regresado a cero. Además, cuando estas dos variables se igualan, se produce el reseteo de la variable “Run”. En la Figura 16 se puede observar esta codificación.

3.4.- Interfaz gráfica

Una parte significativa del programa está dedicada a la representación gráfica del robot, la monitorización del automatismo mediante la gráfica XY y los leds parpadeantes. Una buena interfaz gráfica contribuye a hacer que el usuario tenga una comunicación más fácil con el robot.

3.4.1.- Modelo 3D del robot

Como bien se ha comentado en el apartado de diseño, el robot ha sido modelado a partir de piezas con geometrías simples. La finalidad de este objeto virtual es representar lo que podría ser un prototipo del robot.

3.4.1.1.- Modelado de las piezas

Este apartado consta de un código bastante amplio pero con pequeños conceptos, por ello, se ha decidido explicar de manera breve las funciones que han sido necesarias para su elaboración y algún que otro ejemplo, ya que no tendría sentido repetir lo mismo varias veces:

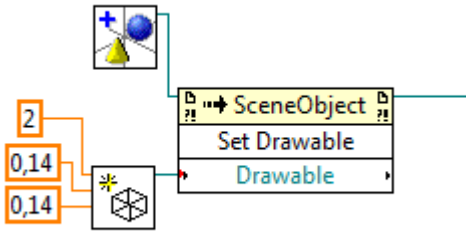


Figura 23: definición de un Prima en 3D.

Para definir cada una de las piezas que conforman el robot, ha sido necesario hacer uso de tres bloques:

- Create object, el cual nos permite crear un nuevo objeto 3D.
- Create box, Create cylinder o Create cone, para definir la geometría de un prisma, un cilindro o un cono de un objeto 3D.
- Invoke node, la cual permite realizar una acción sobre una referencia. En este caso se han utilizado dos acciones:
    1. Add object, para añadir un objeto a una referencia.
    2. Set Drawable, para dibujar un objeto sobre una referencia.

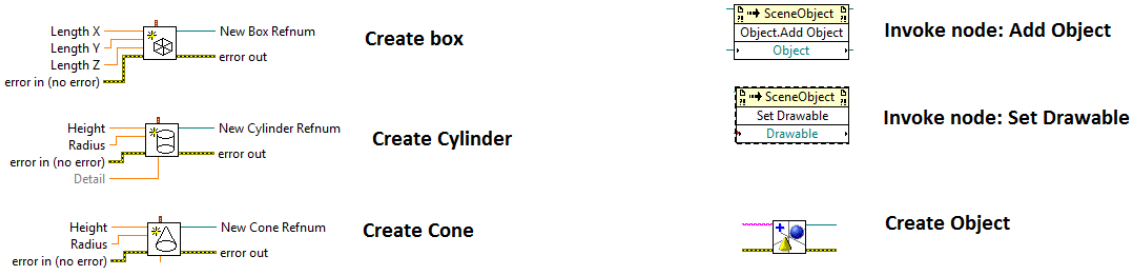


Figura 24: Funciones utilizadas en el modelado de las piezas.

En la Figura 23 se puede observar como se ha creado un prisma y se ha dibujado sobre una referencia (la cual no tenía definido ningún objeto) mediante la acción Set Drawable. De esta manera, la salida del invoke node podrá ser utilizada como referencia de otros nodos distintos. La Figura 24 muestra como se ha creado una de las base del robot. A partir de un objeto creado en un nodo, se le han ido añadiendo más objetos mediante la función Add object. De esta manera, la referencia del último nodo afecta a los cuatro objetos creados.

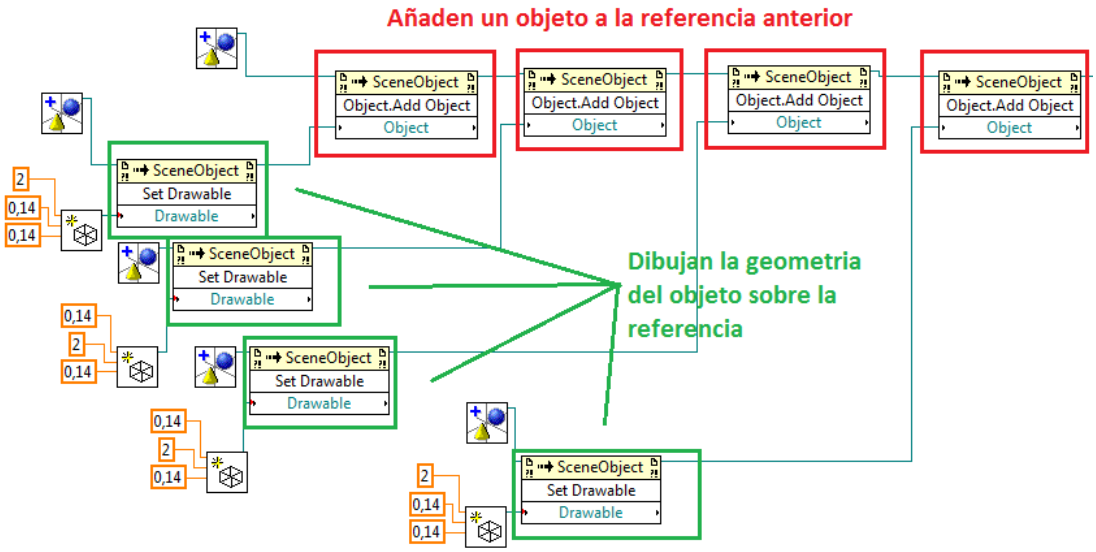


Figura 25: definición de la base del robot.

Siguiendo la metodología explicada anteriormente, se han creado el resto de piezas que conforman el robot. De esta manera se ha ido creando una referencia cada vez más grande, que es la que finalmente se ha representado en un indicador 3D llamado "3D picture". Cabe destacar que la salida "object" de un invoke node cualquiera (Add Object) se puede utilizar para modificar el objeto (uno o varios) que añade, aplicando transformaciones como la translación o el giro respecto a uno de sus ejes. De esta manera podemos posicionar las piezas o incluso simular el movimiento del robot (con controles).

3.4.1.2- Manipulación de la posición

En este apartado se va a explicar cómo se ha conformado el robot a partir de sus piezas. Para poder posicionar cada una de las formas geométricas, de manera que acaben dando cohesión al robot, se ha tenido que modificar la posición de una con respecto de otras. Para simplificar el desarrollo, se va a explicar cómo modificar la posición, giro, escala y color de un objeto de manera que se pueda generalizar a todo el robot.

Una vez generada la pieza, es necesario realizar varias operaciones secuenciales. A continuación, en la Figura 26, se muestran las diferentes operaciones que se han utilizado sobre un objeto:

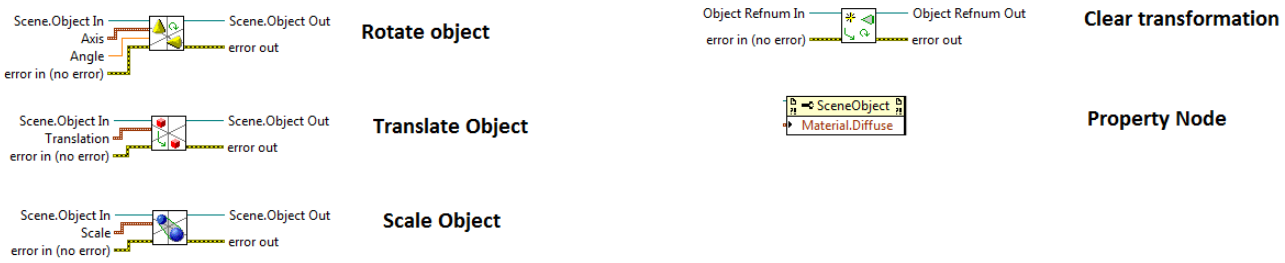


Figura 26: Funciones aplicadas a los objetos 3D.

- Rotate Object: Permite rotar un determinado ángulo el objeto 3D de una escena alrededor de uno, o varios ejes.
- Translate Object: Permite trasladar una determinada cantidad el objeto 3D de una escena en la dirección de uno, o varios de sus ejes.
- Scale Object: Permite modificar la escala de un objeto, en cualquiera de sus tres dimensiones.
- Clear transformation: Elimina cualquier transformación previa realizada sobre un objeto.

- Property Node: Permite modificar alguna propiedad del objeto, en este caso, el color.

Una vez se define una pieza, el origen de cada uno de sus ejes está situado en el centro geométrico. Cuando se hace pasar un objeto u objetos a través de la función Translate Object, se desplaza la pieza pero no sus ejes. Por esta razón, si se quiere hacer girar una pieza respecto a sus ejes de simetría en una posición distinta de donde nacen su ejes, es necesario primero hacer un giro y luego una traslación. Esto es importante de cara a entender el orden de aplicación de las distintas operaciones. Cada vez que se aplique una operación sobre un objeto, es necesario hacer uso de la función Clear transformation, ya que en caso de no utilizarla, no se limpiaría el registro de transformaciones hechas sobre el objeto previamente y, por tanto, las transformaciones serían acumulativas.

Por último, en caso de querer modificar la escala del objeto, se puede hacer al final de las transformaciones, junto con el nodo de propiedad para modificar el color. En conclusión, para realizar las transformaciones pertinentes sobre los objetos del robot se ha utilizado la siguiente secuencia:

1. Se han limpiado las transformaciones anteriores con la función Clear Object.
2. Se ha rotado el objeto con la operación Rotate Object.
3. Se ha trasladado el objeto con la operación Translate Object.
4. Por último, se ha modificado el color.

En la Figura 27 se muestra un ejemplo de cómo se han aplicado estas funciones sobre una escena 3D. En esta ocasión se ha modificado la posición y el giro del cono del robot. En una de las transformaciones de rotación se ha colocado un control para poder controlar posteriormente el giro desde los datos de posición proporcionados por el motor del cabezal. Para realizar las transformaciones del resto de piezas que conforman este robot, se ha seguido la mecánica anteriormente descrita.

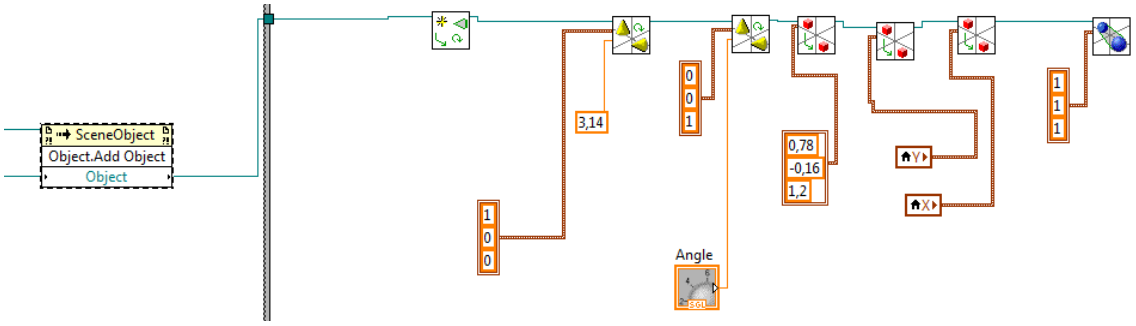


Figura 27: Transformaciones aplicadas sobre el cono (taladro).

Finalmente, para poder comunicar cada uno de los motores con sus elementos del modelo 3D, se ha realizado en el programa principal un bucle que comunica los cuatro controles del robot (posición del brazo X, Y, Z y posición del cabezal) con sus correspondientes motores, a través de rectas de conversión. De esta manera, cada periodo de ejecución del bucle, el modelo actualiza sus transformaciones sobre los objetos 3D.

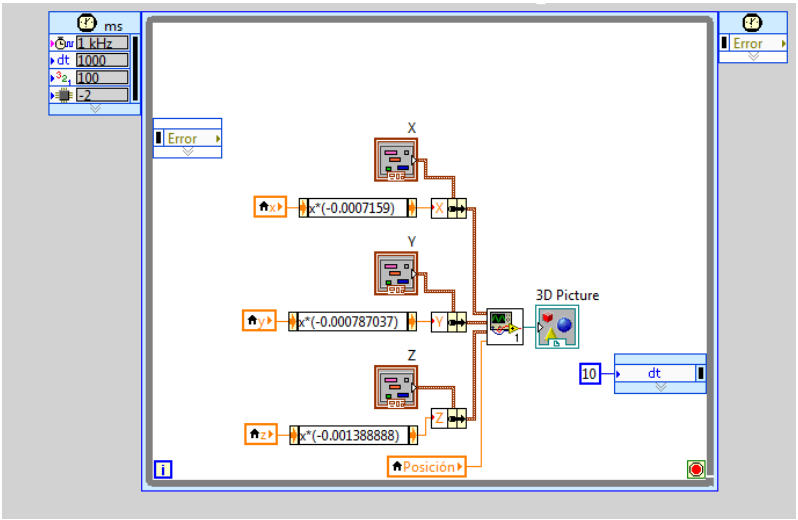


Figura 28: Comunicación del modelo 3D con los motores

3.4.2.- Representación XY

Para poder visualizar adecuadamente el transcurso de la ejecución de los puntos, se ha implantado una gráfica cartesiana que representa las coordenadas X e Y. La gráfica funciona de la siguiente manera:

- Cuando arranca el automatismo, se representan en blanco cada uno de los puntos que se va a realizar, es decir, el itinerario de puntos.
- Cada punto que se esté ejecutando aparece en tiempo real en la gráfica coloreado en azul.
- Los puntos se van coloreando en verde conforme se hayan realizado.

Este funcionamiento se ha implementado utilizado el bucle en el que se descompone el fichero del automatismo en las variables Pos1, Pos2 etc (Figura 29). Este código genera 4 vectores de datos, los cuales van registrando en tiempo real las coordenadas X e Y del fichero de datos conforme se activa la variable "x.and.y?". Esta variable se activa conforme los motores X e Y alcanzan una coordenada de un punto de trabajo, por tanto, los vectores solo se actualizan conforme los motores hayan alcanzado la posición de los puntos. La diferencia entre estos vectores es que dos de ellos, además de registrar las coordenadas de los puntos que se están realizando en tiempo real, también almacenan las coordenadas de los vectores hechos en el pasado, gracias a un Shift Register. De esta manera, representando ambos pares de vectores en la gráfica XY se consigue el objetivo deseado.

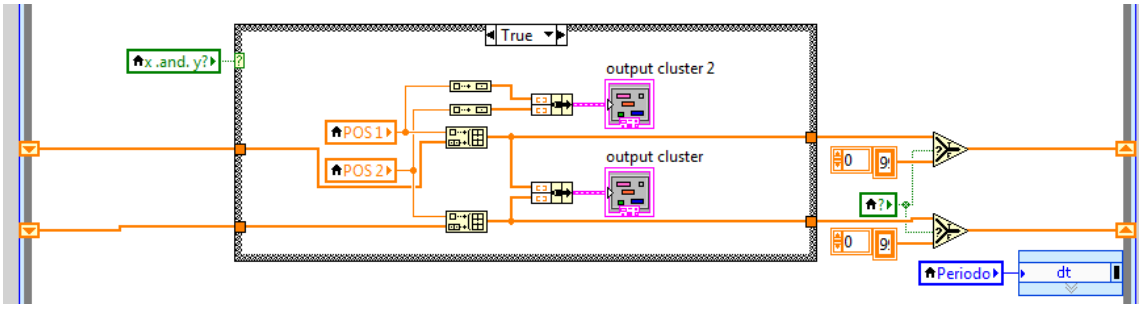


Figura 29: Código para la generación de vectores para la gráfica XY.

Para conseguir representar el itinerario de puntos falta representar un par de vectores más, cuya codificación se ha implementado en el bucle de control. Como se puede observar en la Figura 30, cuando se activa el automatismo (Run=true), se extrae de la matriz del fichero el total de coordenadas X e Y en un par de vectores y acto seguido se representan junto a los pares de vectores anteriormente descritos.

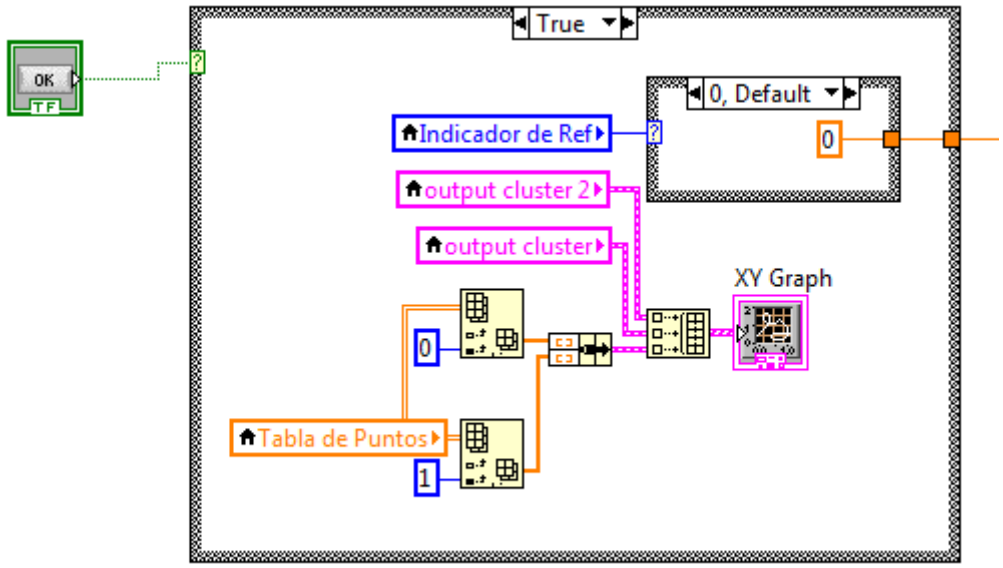


Figura 30: Representación de los vectores en la gráfica X, Y.

3.4.3.- Leds

Por último y para terminar con la parte gráfica del programa, se ha implementado un algoritmo para producir el parpadeo de leds en función del tipo de punto. Cada led representa un tipo de punto distinto, de tal manera que cuando la variable "element" toma un valor entero procedente del fichero de datos, hace parpadear (utilizando nodos de propiedad) al led correspondiente a ese tipo de punto y apaga el resto de leds. En la Figura 31 se observa su codificación en el programa.

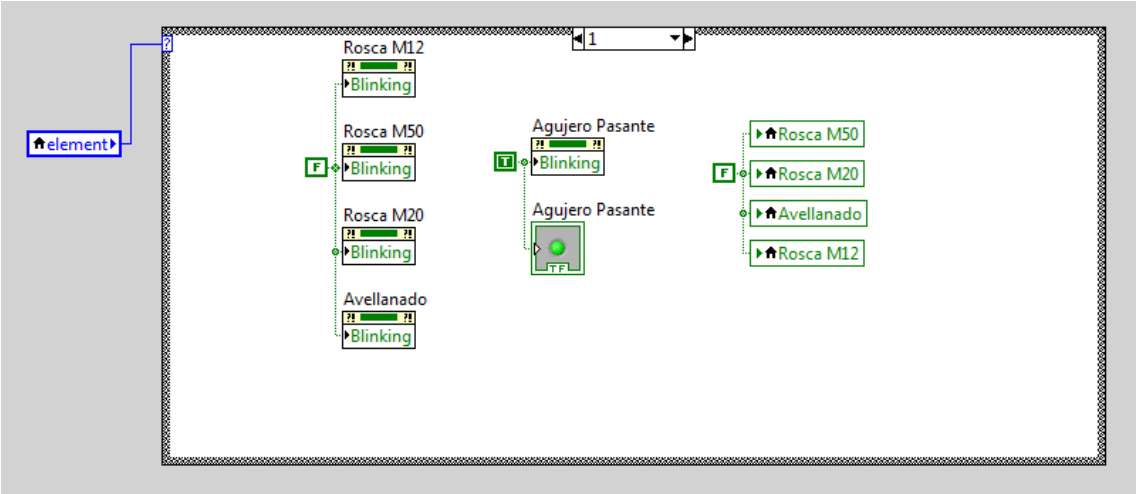


Figura 31: Codificación de los leds.

3.5.- Protocolo de comunicación TCP

Por último y para finalizar con este anexo, se ha utilizado un programa cliente y un programa servidor para establecer la comunicación TCP a través de Ethernet.

Es importante comentar que estos dos programas no son objeto del proyecto y han sido utilizados por el hecho de no disponer de una tarjeta de adquisición de datos con cuatro salidas. En principio el programa está diseñado para ser un único bloque ejecutado en un mismo ordenador, ya que no tiene demasiando sentido controlar un robot desde dos computadores distintos.

El programa servidor está implementado en el programa principal. Este manda en cada instante de tiempo una referencia de velocidad y de posición para el motor del cabezal y el motor Z, respectivamente, de tal manera que, cuando el bucle del automatismo actualice esta referencia, se vea reflejado en el servidor. Toda esta información se envía a través de un puerto que se especifica a través de un control.

El programa cliente, implementado en el programa secundario, recibe esta información siempre que se configure la conexión, es decir, el programa ha de conectarse a la dirección IP del programa servidor y especificar el puerto (control) en el cual el programa servidor está volcando la información.

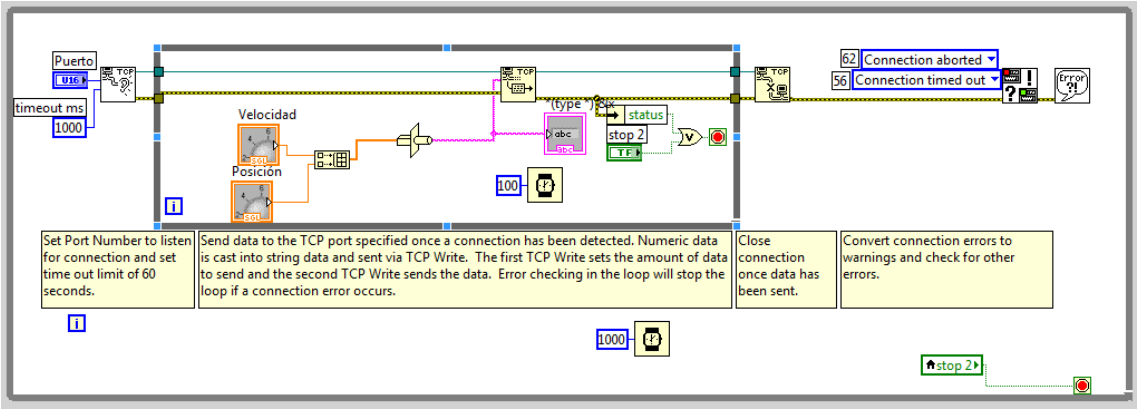


Figura 32: programa servidor.

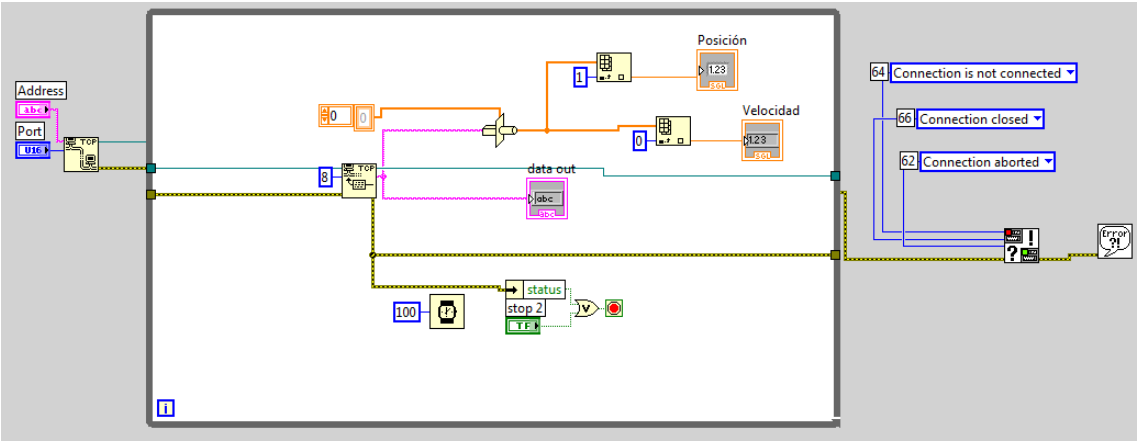


Figura 33: programa cliente.

4.- BIBLIOGRAFÍA

ISA. (2013-2014). *Programación mediante LabVIEW para aplicacion de identidicación y control.* Valencia: UPV.

LAazaro, A. M., & Fernandez, J. R. (2005). *PROGRAMACION GRÁFICA PARA EL CONTROL DE INSTRUMENTACIÓN.* PARANINFO, S.A.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO DE UNA APLICACIÓN PARA LA IDENTIFICACIÓN Y CONTROL DE UN MOTOR DE CORRIENTE CONTINUA MEDIANTE LABVIEW

Documento Nº4: Manual de Usuario

AUTOR: Dotor Alonso, Luis

TUTOR: Herrero Durá, Juan Manuel

Curso Académico: 2013-14

CONTENIDO

1.- INTRODUCCIÓN Y OBJETIVOS 2
2.- REQUERIMIENTOS 2
2.1.- Requerimientos de Hardware 2
2.1.1.- PC 2
2.1.2.- Tarjeta de adquisición de datos 3
2.2.- Requerimientos de Software 3
3.- INTRODUCCIÓN A LA APLICACIÓN 4
3.1.- Arranque del programa 4
3.2.- Página principal 5
3.3.- Configuración y control 9
3.3.1.- Control y monitorización 10
3.3.2.- Configuración 11
3.3.2.1.- Configuración PID 12
3.3.2.2.- Configuración del bucle de control 13
3.3.2.3.- Configuración del protocolo TCP 14

1.- INTRODUCCIÓN Y OBJETIVOS

El presente documento pretende ser una guía para los usuarios iniciados en la aplicación, de manera que puedan conocer el programa a nivel de configuración, controles, indicadores e interfaz, sin la necesidad de tener conocimiento en la programación de LabVIEW.

Los objetivos de este documento son:

- Ser manual de apoyo para el usuario del programa.
- Mostrar el significado de cada uno de los controles e indicadores del programa.
- Especificar la manera en la que se debe realizar la comunicación de Ethernet.
- Mostrar los requisitos mínimos en cuanto al software y al hardware.

2.- REQUERIMIENTOS

A continuación, se van a especificar el conjunto de necesidades de hardware y software para conseguir un correcto funcionamiento de la aplicación.

2.1.- Requerimientos de Hardware

Se ha de disponer tanto de un ordenador como de una tarjeta de adquisición de datos para poder ejecutar la aplicación.

2.1.1.- PC

Las especificaciones mínimas del PC para ejecutar la LabVIEW dependerán del sistema operativo utilizado, no obstante, a continuación se muestran los requisitos mínimos en Windows, ya que ha sido el sistema operativo que se ha empleado.

En caso de un usuario que solamente pretenda ejecutar la aplicación:

- Procesador Pentium III/Celeron 866 MHz o equivalente.
- RAM 256 MB.
- Resolución de pantalla 1024 x 768 píxeles.
- Espacio libre en disco: 407 MB.

Si este usuario pretende programar o modificar el programa:

- Procesador Pentium 4 o equivalente.
- RAM 1GB.
- Resolución de Pantalla 1024 x 768 píxeles.
- Espacio libre en disco: 3.5GB.

2.1.2.- Tarjeta de adquisición de datos

Se ha disponer de una tarjeta de adquisición con cuatro salidas y cuatro entradas analógicas de $\pm$ 10 voltios. En caso de no disponer de una tarjeta con dichas conexiones, se pueden utilizar dos tarjetas distintas, instaladas en un ordenador o en dos diferentes.

En el manual actual se explica el programa utilizando dos tarjetas de adquisición de datos AD-Link PCI-9112, puesto que cada una de ellas dispone únicamente de dos salidas. Se puede utilizar cualquier tarjeta distinta con las especificaciones anteriores, no obstante, se deberán hacer los cambios oportunos en el programa e instalar los drivers de la nueva tarjeta en el ordenador que ejecuta la aplicación.

2.2.- Requerimientos de Software

En cuanto al software, se ha de disponer de un sistema operativo y de la plataforma de trabajo:

- La plataforma de trabajo utilizada ha sido LabVIEW 2010, no obstante, el programa es completamente compatible con las versiones sucesoras.
- A continuación se muestran los sistemas operativos compatibles con este programa:
    - Windows 8.1/8/7/Vista (32 bits y 64 bits)
    - Windows XP SP3 (32 bits)
    - Windows Server 2003 R2 (32 bits)
    - Windows Server 2008 R2 (64 bits)
    - Mac OS X 10.5, 10.6, 10.7, 10.8 o 10.9
    - Linux kernel 2.2x, 2.4x, 2.6x o 3.x y GNU C Library (glibc) Versión 2.4.4 o posterior para la arquitectura Intel x86.

Además de todo este software, el controlador gráfico del ordenador que se esté usando ha de estar actualizado, ya que puede generar problemas con la simulación gráfica 3D.

En el desarrollo del proyecto se ha utilizado también el programa Matlab y las herramientas que contiene, como System Identification Tool, Simulink y Rltool, no obstante, este programa no es necesario tenerlo instalado, puesto que solo se ha utilizado para la obtención de los modelos matemáticos del motor y los reguladores pertinentes.

3.- INTRODUCCIÓN A LA APLICACIÓN

A continuación se va a explicar en detalle el funcionamiento de la aplicación, la cual ha sido estructurada en dos partes diferentes:

- La ventana de Página principal
- La ventana de Configuración y control.

Figura 1: Ventanas de la aplicación.

3.1.- Arranque del programa

Una vez abierta la aplicación, se pulsa el botón  situado arriba a la izquierda para iniciar el programa. Cabe destacar que, previamente al arranque del programa, las conexiones pertinentes han de estar realizadas adecuadamente, ya que en caso contrario el funcionamiento no sería el correcto.

3.2.- Página principal

Esta ventana ha sido desarrollada con un nivel de configuración mínimo, está pensada para que el usuario pueda utilizar el robot sin tener que configurar nada. A continuación, se muestra una imagen con la Pantalla principal:

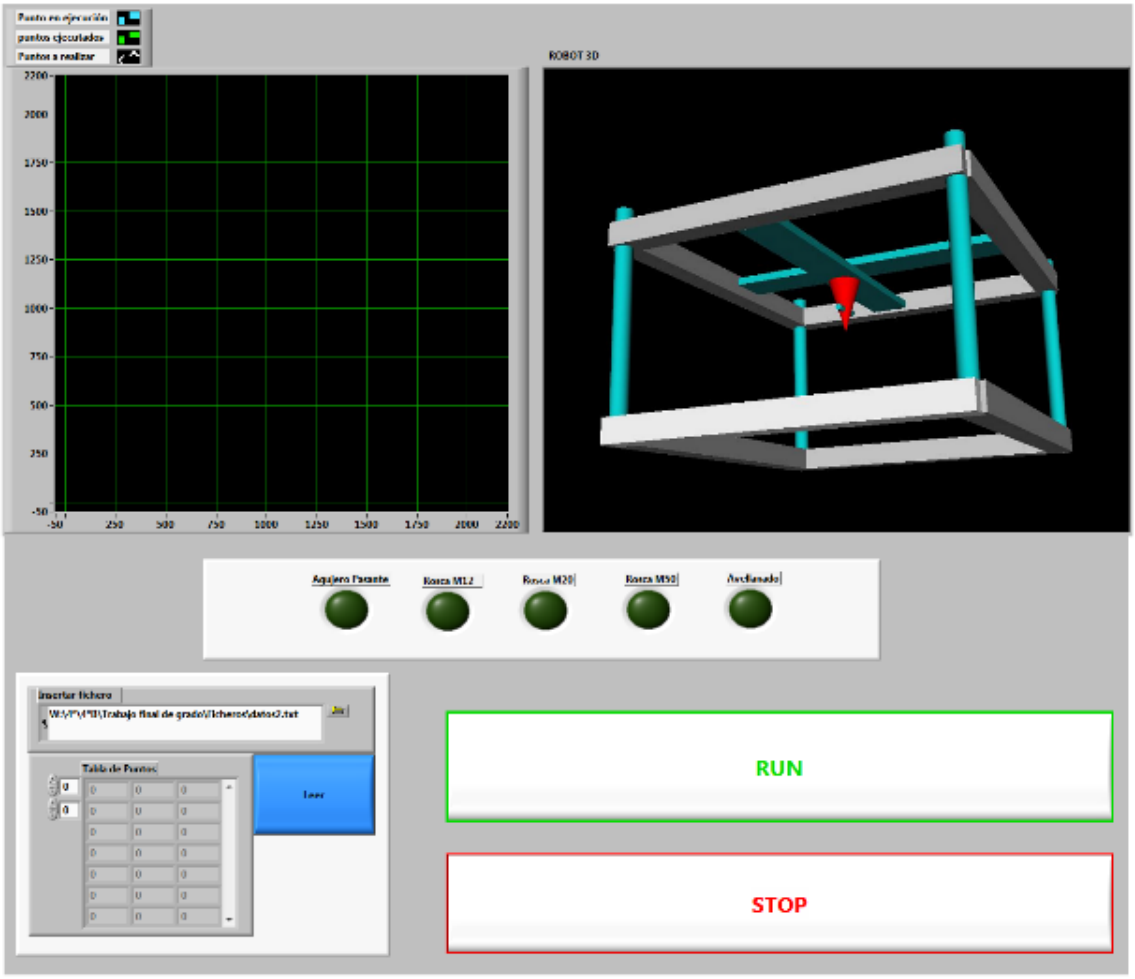


Figura 2: Página principal de la aplicación.

En esta Pantalla lo único que el usuario debe saber es:

- Cargar un fichero de datos y leerlo en el programa.
- Iniciar el automatismo.
- Interpretar la gráfica y el modelo 3D.
- Parar el programa.

Para cargar un fichero de datos se ha de pinchar en el botón  que hay a la derecha del cuadro *Insertar fichero*. De esta manera, se abrirá un buscador que permitirá insertar el directorio del fichero donde se encuentran los datos.

Una vez se ha insertado el fichero, se ha de pinchar en el botón *Leer* para que el programa pueda reconocer los distintos puntos.

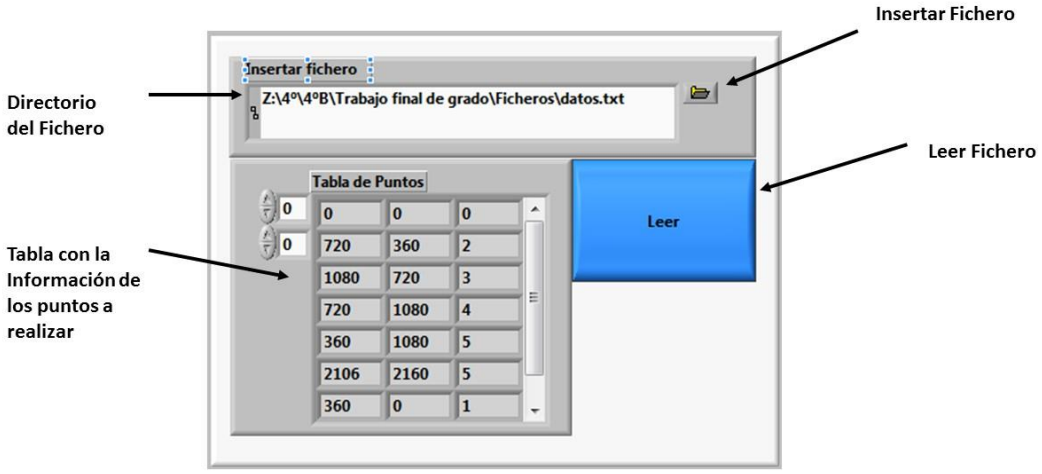


Figura 3: Carga del fichero.

Cabe mencionar que, para crear un fichero de datos, se ha de hacer uso de una hoja de cálculo (por ejemplo Microsoft Excel). Se han de insertar los datos de los puntos por filas, generando 3 columnas en el siguiente orden: Coordenada X, Coordenada Y, Tipo de punto (valor entero del 1 al 5). Además, la primera fila del fichero ha de ser la (0, 0, 0). A continuación, se puede ver una imagen con la confección de un fichero en Excel:

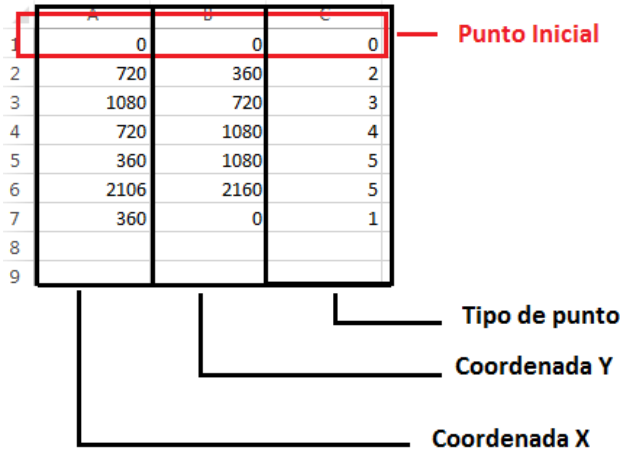


Figura 4: Fichero en Excel.

Una vez el programa reconoce cada uno de los puntos, el usuario ha de pinchar en el botón *RUN* para iniciar el automatismo, de manera que cuando este termine, se podrá volver a pinchar en el botón *RUN* (para iniciarlo de nuevo), o se podrá pinchar en el botón *STOP* para finalizar con el programa (Figura 4). Cada vez que finaliza el automatismo, se podrá cambiar el fichero de datos reiterando el paso de carga del fichero.

Figura 5: Inicio del automatismo y parada de la Aplicación.

Cada vez que se inicie el automatismo, la gráfica XY empezará a actualizarse mostrando los siguientes puntos:

- Punto en ejecución (cuadrado azul)
- Puntos ejecutados (cuadrados verdes)
- Puntos a ejecutar (cruces blancas)

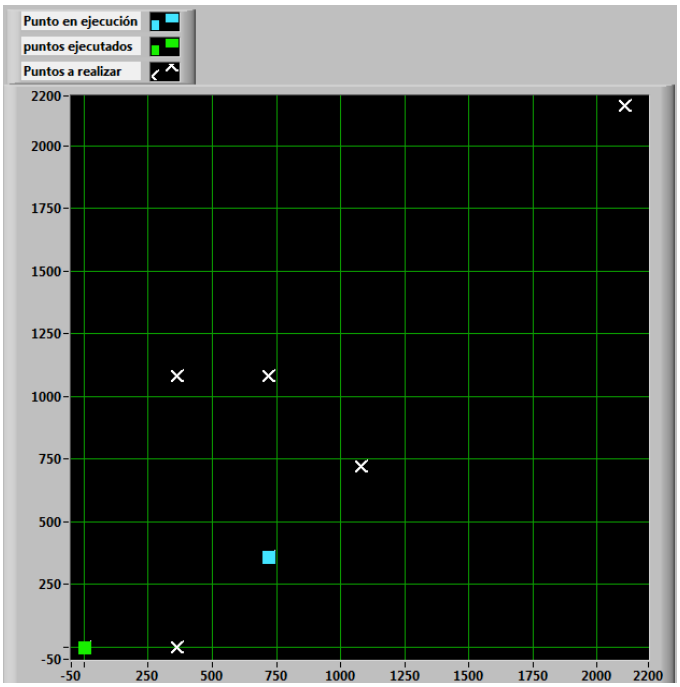


Figura 6: Gráfica XY.

Por último y de la misma forma, al ser pinchado el botón *RUN*, los Leds comenzarán a parpadear en función del tipo de punto del Fichero (1-5) y el modelo 3D se moverá representando el movimiento de los motores en tiempo real.

Figura 7: Leds.

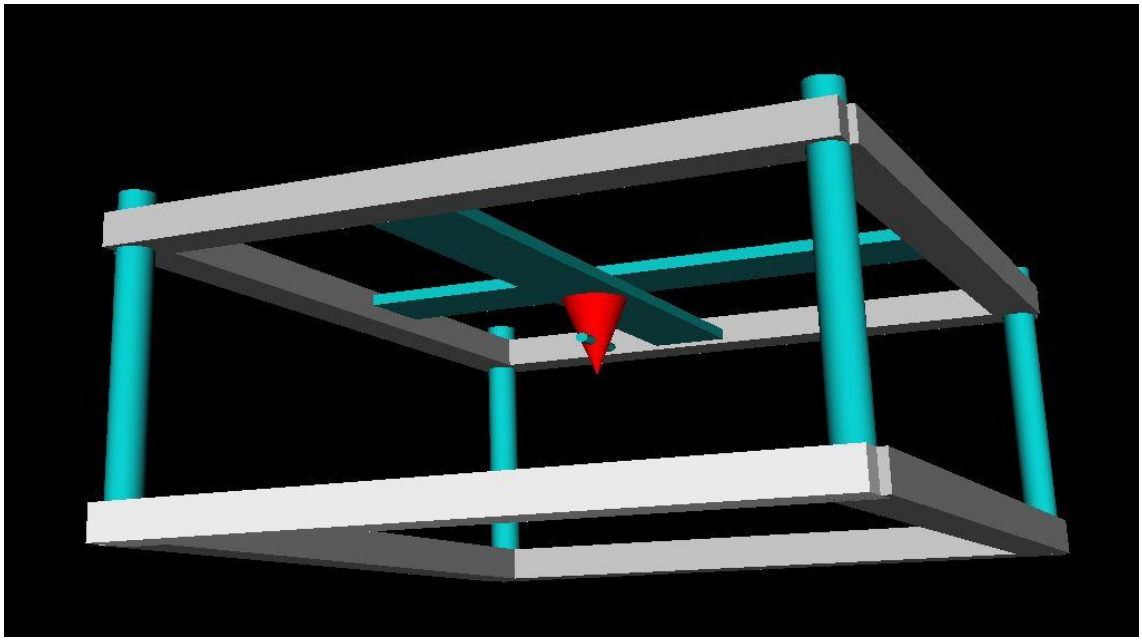


Figura 8: Modelo en 3D.

3.3.- Configuración y control

Esta es la segunda ventana de la aplicación, la cual se ha pensado para un usuario técnico. Se ha desarrollado de tal forma que se pueda monitorizar el proceso, hacer comprobaciones con los motores, configurar controladores, etc.

Cabe destacar que la aplicación que aquí se presenta está dividida en dos programas que son (a nivel gráfico) copia una de otra, por tanto, se va a explicar el programa principal (Motor X e Y), ya que el programa secundario (Motor Z y Motor del cabezal) consta prácticamente de las mismas funciones. La única diferencia es la parte de *Configuración TCP*, ya que un programa es servidor (Programa principal) y otro es cliente, no obstante esta parte se explica al final del documento.

Figura 9: Ventana de configuración y control.

3.3.1.- Control y monitorización

Esta parte del programa se ha realizado para que el técnico del robot pueda monitorizar cada uno de los motores mediante gráficas, las cuales muestran información sobre la posición, velocidad o voltaje aplicado. La interfaz incluye, para cada uno de los motores:

- Un botón para cambiar entre control manual y control automático, de tal manera que se hace encender un led indicando el tipo de control seleccionado.
- Un control (potenciómetro) para manipular manualmente el valor de la referencia de cada uno de los motores, siempre y cuando esté seleccionada la opción de control automático.
- Un control (potenciómetro) para aplicar manualmente tensión a los distintos motores, el cual solo se atenderá cuando esté activa la opción de control manual (este control está expresado en %).

A continuación se muestra una imagen del programa principal con cada uno de los elementos mencionados anteriormente:

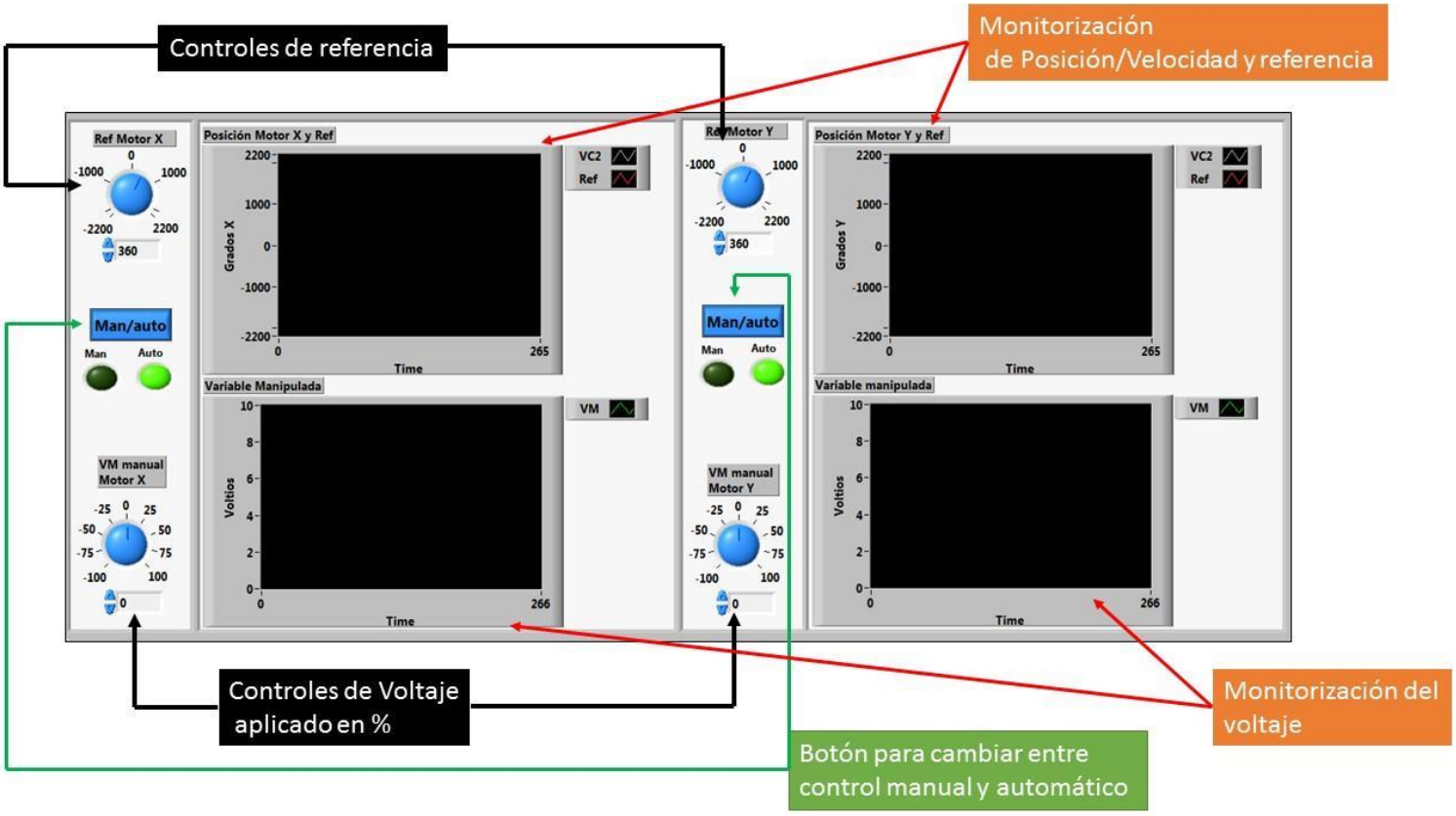


Figura 10: Interfaz de control y monitorización.

3.3.2.- Configuración

La otra parte de esta ventana está dedicada a la configuración, mediante la cual el técnico podrá configurar los parámetros oportunos. En ella podemos encontrar una serie de ventanas que se corresponden con:

- Configuración de reguladores PID.
- Configuración del bucle de control.
- Configuración del protocolo TCP.

En caso de pinchar en cada una de las ventanas, el programa mostrará los parámetros configurables correspondientes.

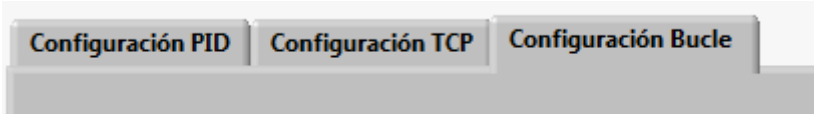


Figura 11: Ventanas de las distintas configuraciones.

3.3.2.1.- Configuración PID

El programa muestra dos recuadros correspondientes a dos motores, dependiendo de si se trata del programa principal o del programa secundario. Dentro de cada uno de estos recuadros hay una serie de controles para modificar diferentes parámetros:

- Dentro del recuadro *PID,* la aplicación permite modificar mediante tres controles los parámetros (K, Ti, Td) de un regulador en concreto.
- Dentro del recuadro *Rango SP* se puede modificar el límite superior e inferior que puede alcanzar la referencia de cada motor.

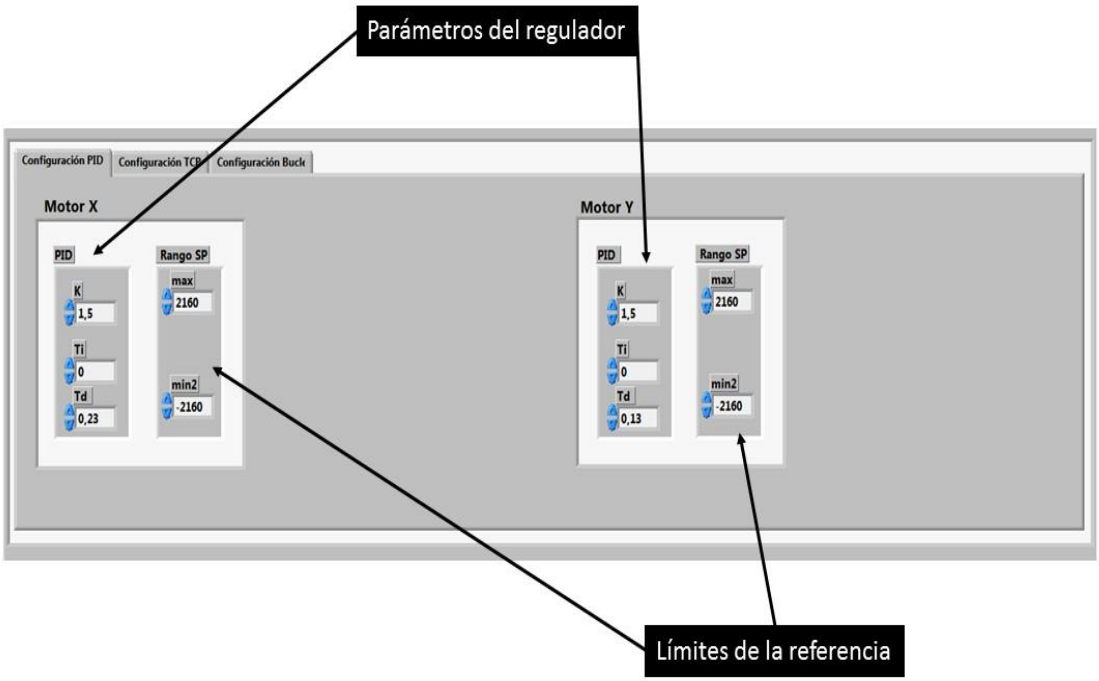


Figura 12: Configuración del PID.

3.3.2.2.- Configuración del bucle de control

En este apartado de la configuración el programa incluye:

- Un control para modificar el periodo de ejecución del bucle de control (y el resto de bucles del programa). Esto puede resultar útil en caso de querer disminuir la carga de cómputo en el ordenador.
- Dos controles para modificar, en cada uno de los motores, el coeficiente *N,* el cual filtra la acción derivada. En el caso del programa secundario solo se muestra un control, puesto que el motor del cabezal no presenta acción de control derivada.
- Indicadores que muestran las vueltas dadas por cada uno de los motores de posición (Motor X, Y, Z). Este indicador sirve para detectar errores de vueltas y así reajustar los rangos del sensor virtual, que transforma la salida del sensor de posición real del motor, por tanto, tiene un uso muy reducido.

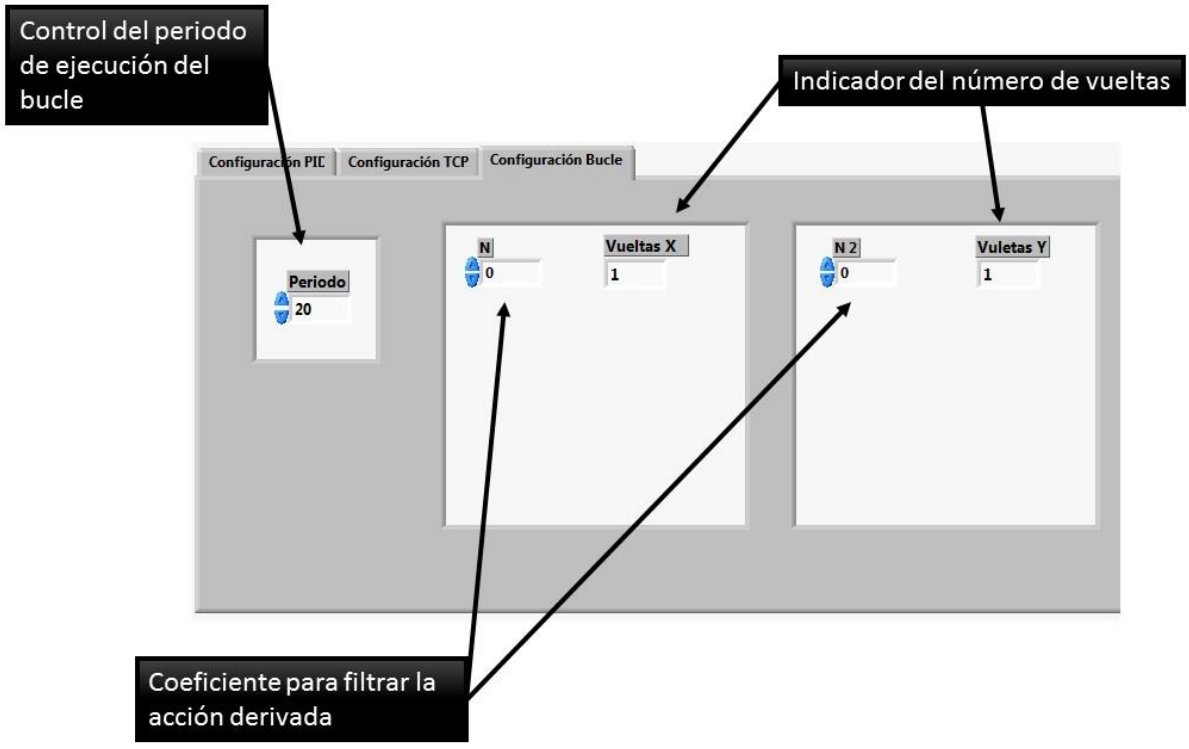


Figura 13: Configuración del bucle de control.

3.3.2.3.- *Configuración del protocolo TCP*

Para poder establecer una buena conexión entre los dos programas (principal y secundario), el programa cliente (secundario) ha de conectarse a la dirección IP del programa servidor (principal) y recibir la información a través del puerto correspondiente.

El programa cliente consta de:

- Un control para seleccionar la dirección IP del programa servidor.
- Un control para seleccionar el puerto donde el programa servidor está volcando la información.
- Un par de indicadores para mostrar los datos de posición y velocidad enviados desde el programa principal.
- Un control booleano para activar o desactivar la comunicación TCP, de tal manera que, si está activo, la referencia del motor Z y del motor del cabezal es enviada a través de Ethernet desde el programa principal. En caso contrario, la referencia de estos motores se indica a través de los controles explicados en el apartado de control y monitorización.
- Un Led que indica si la comunicación TCP está activada o desactivada.
- Un botón de *STOP* para apagar el bucle que controla la comunicación.

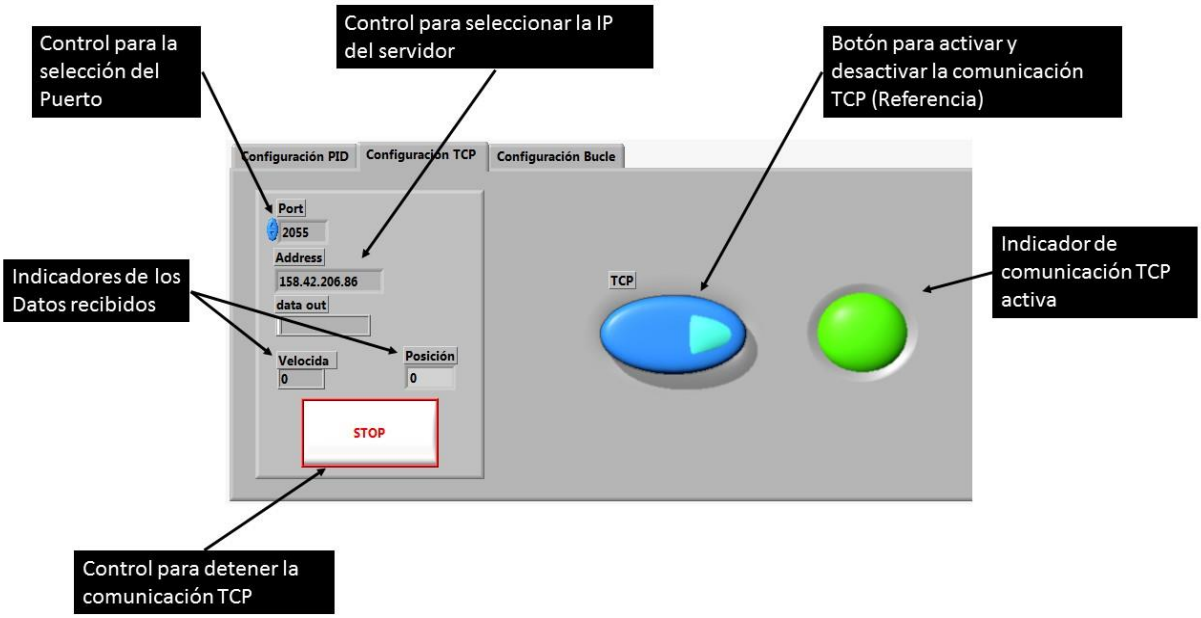


Figura 14: Configuración de la interfaz cliente (Programa secundario).

El programa servidor consta de:

- Un control para la selección del puerto, en el cual se volcará la información.
- Dos controles (potenciómetros) para modificar la referencia de velocidad y la posición enviada al programa cliente.
- Un botón de STOP para apagar la conexión desde el servidor.

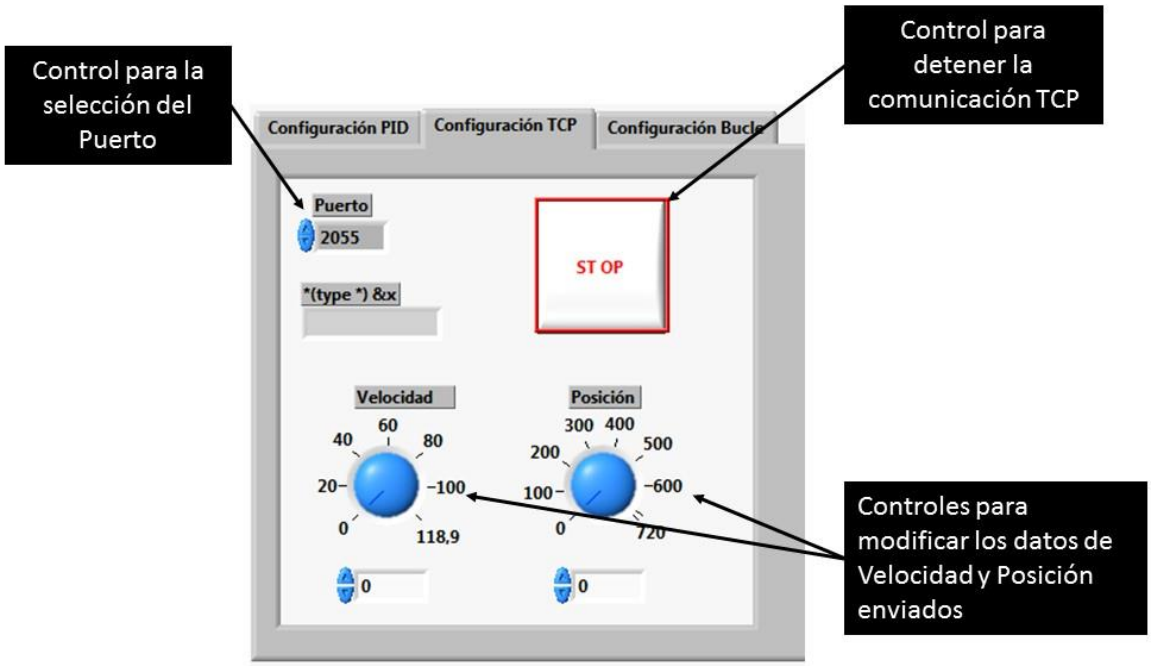


Figura 15: Configuración de la interfaz servidor (Programa principal).

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO DE UNA APLICACIÓN PARA LA IDENTIFICACIÓN Y CONTROL DE UN MOTOR DE CORRIENTE CONTINUA MEDIANTE LABVIEW

DocumentoNº5: Presupuesto

AUTOR: Dotor Alonso, Luis

TUTOR: Herrero Durá, Juan Manuel

Curso Académico: 2013-14

CONTENIDO

1.- INTRODUCCIÓN 2

2.- PRESUPUESTO 3

2.1- Definición de las Unidades de obra 3

2.2.- Cuadro de precios 3

2.2.1- Cuadro de precios Nº1: Mano de obra 4

2.2.1- Cuadro de precios Nº2: Materiales 6

2.2.3.- Cuadro de precios Nº3: precios unitarios 8

2.2.4.- Cuadro de precios Nº4: precios descompuestos 9

2.3.- Presupuesto de ejecución material 9

2.4.- Presupuesto de ejecución por contrata 10

1.- INTRODUCCIÓN

En el siguiente documento se va a realizar un estudio económico para valorar el trabajo final de grado. Hay que tener en cuenta que se trata de un software informático y que, por tanto, se planteará de forma diferente a un proyecto de construcción o una instalación eléctrica.

Para estimar dicho presupuesto se va a considerar que la aplicación informática ha sido desarrollada por un Graduado en Tecnologías Industriales en una empresa dedicada al trabajo en cuestión. Se considerará que el cliente ha solicitado el desarrollo de la aplicación para controlar un robot con una tarjeta de adquisición de datos (AD-Link PCI-9112) específica, suministrados por su compañía. Es decir, no se ha considerado en el presupuesto la maqueta (motores), ni la tarjeta de adquisición de datos. A partir de esta premisa se evaluará el presupuesto total del proyecto.

Por tratarse de Trabajo Final de Grado no se incluirá el IVA en los precios estimados, no obstante, se añadirá el beneficio industrial y los gastos generales al precio final del proyecto.

2.- PRESUPUESTO

2.1- Definición de las Unidades de obra

A continuación, se van a describir cada una de las unidades de obra en las que se ha dividido el presupuesto:

- **UO1-Montaje y mantenimiento de los equipos**: incluye el montaje, almacenamiento, traslado, revisión y ajuste de cada uno de los prototipos, así como la puesta a punto de las unidades de hardware y software necesarios para su funcionamiento (Tarjeta de adquisición de datos, ordenador, fuente de alimentación, Drivers de la tarjeta, LabVIEW, Matlab).
- **UO2-Desarrollo de la aplicación**: incluye las horas de programación, de identificación y cálculo, así como la búsqueda de información de fuentes, la utilización del hardware y software.

2.2.- Cuadro de precios

En este apartado se han recogido los cuadros de precios de:

- **Mano de obra**: en el cual se incluye la determinación del coste del jornal, tanto de la mano de obra directamente implicada como indirectamente implicada.
- **Materiales**: incluye la amortización del hardware y el software utilizados.

2.2.1- Cuadro de precios Nº1: Mano de obra

En el presente proyecto ha participado un Graduado en tecnologías industriales como mano de obra directamente implicada y dos técnicos de laboratorio para el mantenimiento de los equipos como mano de obra indirectamente implicada.

Los dos técnicos nombrados anteriormente realizan el mismo trabajo, la diferencia fundamental es que uno trabaja en horario de mañanas y el otro en horario de tardes, por tanto se valorarán por igual como técnico de mantenimiento.

COSTES	TOTAL (€)
Salario Base (220 días al año)	14000
Pluses (transporte, herramientas etc)	1500
Seguridad Social (Base de cotización, accidentes, desempleo etc)	4650
Extras (Vacaciones)	4000
Otros (días y horas extra etc)	1000
Total a facturar por año	25150
Total a facturar por jornada (8 horas)	114,3
Total a facturar por hora	14,28

Tabla 1: Coste de la mano de obra indirectamente implicada.

Cada técnico de mantenimiento ha invertido una media de dos horas de su jornada semanal en el mantenimiento de los equipos entre montaje, revisión, reparación y almacenamiento de los mismos, así como en la actualización del software. Al tratarse de un proyecto de un mes y medio de duración (12 semanas), el total de horas invertidas por los dos técnicos son:

$$horas\ invertidas = 2 \times 12 = 24\ horas/técnico$$

El precio del jornal de un Graduado en Tecnologías Industriales se puede observar en la siguiente tabla:

COSTES	TOTAL (€)
Salario Base (220 días al año)	20000
Pluses (transporte, herramientas etc)	1500
Seguridad Social (Base de cotización, accidentes, desempleo etc)	6660
Extras (Vacaciones)	5050
Otros (días y horas extra etc)	1250
Total a facturar por año	34460
Total a facturar por jornada (8 horas)	156,63
Total a facturar por hora	19,6

Tabla 2: Costes de la mano de obra directamente implicada.

El tiempo empleado por el Graduado en Tecnologías Industriales en el desarrollo de la aplicación es de un mes y medio, teniendo en cuenta que trabaja 8 horas al día y que cada mes tiene 22 días laborables, el total de horas trabajadas es:

$$horas\ invertidas = 1{,}5 \times 8 \times 22 = 246{,}4\ horas$$

2.2.1- Cuadro de precios Nº2: Materiales

En cuanto a los materiales utilizados en el proyecto, se ha realizado un estudio de amortización, ya que para el desarrollo de la aplicación han hecho falta una serie de programas (software) y de equipos (Hardware), sin los cuales no hubiera sido posible llevarlo a cabo.

En la siguiente tabla se muestran los precios de cada uno de softwares utilizados en el proyecto, junto con sus respectivas licencias. Cabe destacar que la licencia de Matlab es de 2000 € cada dos años y que la licencia de Windows 7 es de 300€ y se renueva cada 3 años.

Software	Precio licencia (€/año)	Precio a amortizar(€/hora)
LabVIEW	1080	0,6
Matlab	1000	0,55
Microsoft Office	70	0,03888
Windows 7	100	0,0555

Tabla 3: Cuadro de precios para la amortización del software

Se ha estimado que cada uno de los softwares utilizados se amortizan cada año en aproximadamente 1800 horas. Teniendo en cuenta las licencias y el tiempo de amortización, el precio (€/hora) para amortizar cada programa se ha calculado como:

$$Precio\ a\ amortizar = \frac{Precio\ de\ licencia(€)}{1800}\ (\frac{€}{hora})$$

En el caso del Hardware, el material amortizado ha consistido únicamente en un ordenador, ya que se ha considerado que el cliente ya es propietario de la maqueta que se va a controlar, de la tarjeta de adquisición y del ordenador que va a ejecutar la aplicación una vez desarrollada. Por tanto, solo se presupuestará la amortización del ordenador del desarrollador.

Hardware	Precio Total (€)	Precio a amortizar (€/hora)
Ordenador (incluyendo periféricos)	800	0,91

Tabla 4: cuadro de precios para la amortización del equipo.

Se considera que la empresa que desarrolla la aplicación amortiza los equipos en un tiempo de 4 años, ya que ésta se dedica al desarrollo de aplicaciones de control y que, por tanto, sus equipos tienen que estar actualizados. Para calcular este precio (€/hora) de amortización se ha tenido en cuenta el precio total de cada equipo y el número de horas laborables en 4 años (880):

$$Precio\ a\ amortizar = \frac{Precio\ del\ equipo(€)}{880}\ (\frac{€}{hora})$$

2.2.3.- Cuadro de precios Nº3: precios unitarios

En la tabla número 5, se muestra el precio unitario de cada unidad de obra, su medición y su importe final en el proyecto.

Descripción	Precios (€/horas)	Medición (horas)	Importe final (€)
UO1-Montaje y mantenimiento de los equipos	28	24	672
UO2-Desarrollo de la aplicación	21,75	246,4	5359,2

Tabla 5: Cuadro de precios unitario

Este cuadro se ha obtenido sumando los recursos específicos (€ por unidad de horas) de cada unidad de obra, y se ha multiplicado por las mediciones de las horas trabajadas, obteniendo así el importe final de cada unidad de obra.

2.2.4.- Cuadro de precios Nº4: precios descompuestos

A continuación, se muestra el cuadro de precios descompuestos del proyecto, en el que se muestran los recursos incluidos en cada unidad de obra.

Descripción	Importe (€/hora)
UO1-Montaje y mantenimiento de los equipos	
1.1 Técnico de mantenimiento de mañanas	14,28
1.2 Técnico de mantenimiento de tardes	14,28
UO2-Desarrollo de la aplicación	
2.1 Amortización del Software LabVIEW	0,6
2.2 Amortización del Software Matlab	0,55
2.3 Amortización del Software Microsoft Office	0,038
2.4 Amortización del Sistema operativo Windows 7	0,055
2.5 Amortización Equipo con periféricos	0,91
2.6 Graduado en Tecnologías industriales	19,6

Tabla 6: Cuadro de precios descompuesto

2.3.- Presupuesto de ejecución material

A continuación, se ha obtenido el presupuesto de ejecución material, que no es más que la suma de los precios de cada una de las unidades de obra por sus respectivas mediciones:

$$P.de\ ejecución\ material = P_{UO1} + P_{UO2} = 672 + 5659{,}2 = \mathbf{6031{,}2}\ €$$

2.4.- Presupuesto de ejecución por contrata

Una vez calculado el presupuesto total del proyecto, se le añade el beneficio industrial y los gastos generales, que son respectivamente el 6% y el 23% del presupuesto de ejecución material.

$$P.\,de\ ejecución\ por\ contrata = (1 + 0{,}23 + 0{,}06) \times 6031{,}2 = \mathbf{7780{,}28}\ €$$

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO DE UNA APLICACIÓN PARA LA IDENTIFICACIÓN Y CONTROL DE UN MOTOR DE CORRIENTE CONTINUA MEDIANTE LABVIEW

Documento Nº6: Pliego de condiciones

AUTOR: Dotor Alonso, Luis

TUTOR: Herrero Durá, Juan Manuel

Curso Académico: 2013-14

CONTENIDO

1.- PLIEGO DE CONDICIONES GENERALES 2
1.1.- Pliego de condiciones facultativas 2
1.1.1.- Programación de trabajos. 2
1.1.2.- Ejecución defectuosa y modificaciones 2
1.1.3.- Recepción definitiva 3
1.2.- Pliego de condiciones económicas 3
1.2.1.- Fianzas 3
1.2.2.- Composición de precios 4
1.2.3.- Mejoras 4
1.2.4.- Revisión de precios 4
2.- PLIEGO DE CONDICIONES PARTICULARES 5
2.1.- Condiciones de funcionamiento del sistema 5
2.1.1.- Limitación por hardware 5
2.1.2.- Limitaciones por sistemas operativos 6
2.1.3.- Control del sistema 6
2.2.- Condiciones de instalación 6
2.3.- Condiciones de cara al programa 7
2.4.- Condiciones de mantenimiento 7
2.4.1.- Condiciones de mantenimiento del programa informático 8
2.4.2.- Condiciones de mantenimiento del material suministrado por el cliente 8
2.5.- Condiciones de garantía 9

1.- PLIEGO DE CONDICIONES GENERALES

En el siguiente documento se describen una serie de normas de carácter general y obligado cumplimiento para este tipo de proyectos.

Este documento se divide en:

- Pliego de condiciones facultativas.
- Pliego de condiciones económicas.

1.1.- Pliego de condiciones facultativas

En este proyecto se exige que el cliente conozca las leyes y el proyecto en su totalidad.

1.1.1.- Programación de trabajos.

La empresa desarrolladora del proyecto dispondrá de dos meses para la realización del mismo.

1.1.2.- Ejecución defectuosa y modificaciones

El cliente tiene la responsabilidad de revisar el funcionamiento del programa suministrado y comunicar los aspectos que, a juicio de él, no considere correctos, extendiéndose dicha responsabilidad a aquellos defectos no detectados pero existentes.

Las modificaciones que el cliente crea oportuno realizar se verán, en cualquier caso, gravadas por su correspondiente incremento de precio, ya que se considerarán como cambios no imprescindibles para el normal funcionamiento del programa y realizados por sus gustos particulares. Atendiendo a esta última razón, el proyectista se compromete a alterar el contenido del programa siempre que no perjudique su ejecución.

En el improbable caso de que las anomalías detectadas se deban a una mala o defectuosa programación, el proyectista se compromete a realizar las modificaciones que estime convenientes para subsanarlas, sin que esto origine costo económico alguno para el comprador.

1.1.3.- Recepción definitiva

Una vez se desarrolle el proyecto, en el plazo de dos meses, se procederá a la recepción definitiva del mismo.

1.2.- Pliego de condiciones económicas

Seguidamente, se describen las relaciones económicas que deben regir entre el cliente y el proyectista, para cada uno de los siguientes apartados:

- Fianzas.
- Composición de precios.
- Mejoras.
- Revisión de precios.

1.2.1.- Fianzas

Las condiciones de pago del proyecto realizado son las siguientes:

- El comprador depositará, en el momento de adjudicación del proyecto y antes del inicio de los trabajos de programación, una fianza como garantía por una cantidad equivalente al 15% del valor total del presupuesto del proyecto.
- En caso de demora en el abono de la fianza, está se incrementará en un 4% semanal.
- El resto del valor total del proyecto será abonado una vez realizada la instalación y comprobación del correcto funcionamiento del programa por el usuario.

- Tras la concertación del período máximo de realización del proyecto entre el proyectista y el cliente, se ha acordado que el proyectista abone una cantidad del 1% del valor total del proyecto por cada día de demora sobre la fecha acordada con el cliente.

Se ha estimado convenientemente realizar un mantenimiento conjunto del programa informático, durante un período de dos años después de la entrega del mismo. Durante este período la garantía del producto será total.

1.2.2.- Composición de precios

Los precios aplicados al proyecto están en consonancia con los dispositivos generales acordados en los convenios en los que tiene ámbito. Así, las tarifas empleadas por la mano de obra corresponden a las vigentes para un Graduado en Tecnologías Industriales realizando tareas de programación.

1.2.3.- Mejoras

Las ampliaciones y mejoras que el cliente estime oportuno realizar en el programa, supondrán un aumento en el importe total del proyecto, de acuerdo con el total que supusiese realizarlas y las tarifas ya estipuladas.

1.2.4.- Revisión de precios

Como el tiempo que puede transcurrir entre la redacción de este proyecto y su entrega y aceptación por parte del cliente puede ser largo, si transcurre el período de tiempo tabulado por el Código Oficial de Peritos e Ingenieros Técnicos Industriales de Valencia, se procederá a una revisión de precios en el acto de entrega del mismo.

2.- PLIEGO DE CONDICIONES PARTICULARES

El presente proyecto trata sobre una aplicación informática desarrollada en LabVIEW, por tanto, existen una serie de aspectos a tratar que lo diferencian de los proyectos típicos de instalaciones industriales.

2.1.- Condiciones de funcionamiento del sistema

Este apartado incluye los mínimos exigidos para el correcto funcionamiento del programa en cuanto a hardware y software, así como la configuración básica del equipo.

2.1.1.- Limitación por hardware

La aplicación software desarrollada sólo puede funcionar en ordenadores que reúnan las condiciones de hardware que se indican a continuación:

- Microprocesador Pentium 4/M o equivalente.
- Memoria RAM mínima de 1 GB.
- Monitor y tarjeta de vídeo, será VGA o superior con al menos 4Mb de RAM de vídeo.
- Espacio libre en el Disco de 200 MB.
- Tarjeta de Adquisición de datos AD-Link PCI 9112.
- Periféricos, Teclado y Ratón.

Estos requisitos son los mínimos requeridos para el correcto funcionamiento del programa, incluyendo las operaciones de simulación que realiza. Son, por tanto, los que se deben incluir en el Pliego de Condiciones como información al Cliente.

2.1.2.- Limitaciones por sistemas operativos

Los requisitos de software se reducen a la posesión de un sistema operativo:

- Windows 7, Windows 8 o Windows 8.1
- Mac OS X 10.5, 10.6, 10.7, 10.8 o 10.9.

2.1.3.- Control del sistema

Se realizará de la forma más sencilla posible. Para ello se han tomado todas las medidas necesarias para que el cliente se pueda desenvolver por el programa de forma cómoda, simplemente debe saber desenvolverse bajo el entorno gráfico Windows, ya que el programa ha incorporado todos los controles de los que dispone un sistema visual como es Windows.

- Botones de control de Funciones sencillas.
- Espacios diferenciados y ordenados.
- Ayuda disponible sobre fondo de pantalla.
- Y por supuesto el uso de Ratón.

2.2.- Condiciones de instalación

El programa puede ser instalado fácilmente por el cliente, sin apenas tener unos vagos conocimientos del entorno gráfico Windows, ya que en el manual de usuario será explicado detalladamente el modo de instalación del programa.

2.3.- Condiciones de cara al programa

A continuación se describen las condiciones de uso de la aplicación para garantizar su integridad y el buen uso de la misma:

- En caso de adjudicación del proyecto, solo se dispondrá de una licencia.
- Si el usuario precisa de otra copia del programa a parte de la suministrada por razones ajenas al funcionamiento de la misma, está terminantemente prohibida la transferencia o copia de un ordenador a otro. En este caso, se deberá proceder a la compra de otra licencia de venta por parte del Graduado en Tecnologías industriales realizador del proyecto.
- En el precio de las nuevas copias se valorará el coste directo de producción del software, sin tener en cuenta desplazamientos y horas de puesta en marcha, siempre y cuando la instalación se realice por parte del cliente. En dicho caso, cualquier anomalía en la instalación que produzca un funcionamiento incorrecto del programa, anula el derecho de garantía posventa de la copia.

2.4.- Condiciones de mantenimiento

Este apartado incluye el mantenimiento del programa informático y el material suministrado por el cliente:

- Dos Tarjetas de adquisición de datos AD-Link PCI 9112.
- Prototipo del Robot.

2.4.1.- Condiciones de mantenimiento del programa informático

El programa informático no necesita revisión por parte de ningún técnico, salvo en el caso de modificaciones en la configuración del hardware o del software. Así mismo, si se actualiza o modifica la versión del equipo y se procediera a modificar alguna característica o ubicación del programa, dicha revisión deberá efectuarse por parte de un técnico designado por el realizador del proyecto.

Queda terminantemente prohibida la alteración del programa y, si se comprobara que éste ha sido modificado, ello acarrearía la pérdida total de la garantía del producto, así como cualquier derecho sobre actualizaciones, mejoras y/o revisiones.

2.4.2.- Condiciones de mantenimiento del material suministrado por el cliente

El Graduado en Tecnologías industriales se compromete a mantener en perfectas condiciones el estado del robot y de las dos tarjetas de adquisición de datos. En caso de cualquier desperfecto, la empresa se hará responsable y reembolsará la cantidad necesaria para su compensación.

2.5.- Condiciones de garantía

Si por cualquier razón se produjera una pérdida accidental de algún fichero, se podrá, previa autorización del proyectista, proporcionar al cliente una clave y contraseña para acceder al espacio web de la empresa y, así, poder descargar los archivos perdidos.

De todo ello se desprende, por tanto, que el período de vigencia de la garantía es de dos años. Más allá, el proyectista no se hará cargo de los gastos ocasionados por posibles anomalías, por otra parte lógicas por el paso del tiempo. Si transcurrido este plazo se requiriera otra copia del programa, se deberá volver a negociar el acuerdo, aunque por condiciones del cliente se otorgarán importantes beneficios en forma de descuentos y rapidez de entrega. Por último, señalar además que la garantía se establece automáticamente en el momento de la firma del contrato de compraventa, sin ningún otro tipo de trámite por parte del comprador del producto de software objeto del proyecto.

Conviene recordar que cualquier cambio o variación en el programa por parte del cliente, así como su copia, préstamo o instalación no autorizada, anula completamente cualquier condición de garantía expuesta anteriormente. Esto incluye también la utilización indebida del disco y/o sus ficheros, bien sea por trabajar bajo condiciones, entornos o configuraciones no recomendadas o por negligencia en su utilización.

El programador no se hace responsable, en ningún caso, de los perjuicios que se pueden desprender de la utilización indebida del programa informático, tanto materiales, como daños a terceras personas.