GRADO EN INGENIERÍA EN TECNOLOGIAS INDUSTRIALES

TRABAJO FIN DE GRADO

DISEÑO Y CONSTRUCCIÓN DEL SISTEMA DE CONTROL DE UN SISTEMA DE DOS EJES CARTESIANOS CON SIMOTION

Alumno: López Arnaiz, Alfredo

Director: Orive Revillas, Darío

Departamento: Ingeniería de Sistemas y Automática

Curso: 2018-2019

Fecha: 18/09/2018

Agradecimientos:

Quiero expresar mi más profundo agradecimiento

Al Departamento de Ingeniería de Sistemas y Automática y al Prof. Darío Orive Revillas, por su inestimable ayuda y por las facilidades recibidas para la realización de este trabajo.

Al Ing. Ion Olazábal, que ha realizado su TFG en el mismo campo y me ha ayudado en todo lo que le ha sido posible.

Igualmente agradecer al Servicio de Asistencia Técnica de Siemens en la persona del Sr. D. Josué Castilla González su labor de ayuda continuada.

Muchas gracias a todos ellos.

Resumen: El presente trabajo de fin de grado describe el diseño de una mesa con dos ejes para realizar operaciones de corte. Se contemplará en el la programación y puesta en marcha de los sistemas de control y generación de consigna y el montaje de los ejes propiamente dichos. No se considerará la parte correspondiente a la herramienta de corte y, debido a la falta de material, no se han podido montar los motores ni los sensores, por lo que no se han podido realizar las pruebas finales. Sin embargo, se ha completado la programación de la máquina y se han comprobado los programas mediante simulación. La programación de la CPU permite la realización de polígonos, circunferencias, tangencias y curvas alabeadas, abarcando de este modo la gran mayoría de las piezas presentes en la industria.

Laburpena: Gradu-amaierako lan honek ebaketa lanak egiteko bi ardatzeko mahai baten diseinua deskribatzen du. Bertan kontrol sistemen eta aginteen sorreren programazioa eta abiaraztea aztertuko dira, baita ardatzen beraien muntaketa ere. Ez da kontuan hartuko ebaketa-erremintari dagokion zatia eta, material gabeziagatik, ezin izan dira muntatu ez motorrak ezta sentsoreak ere, eta horrexegatik ezin izan dira azken probak egin. Halere, makinaren programazioa burutu da, eta simulazio bidez programak egiaztatu dira. CPUren programazioari esker poligonoak, zirkunferentziak, ukitzaileak eta kurba kopatuak egin daitezke, industrian dauden geometrien gehiengo nagusia barnean hartzen direlarik.

Abstract: The present final degree project describes the design of a table with two axis which makes cutting operations. It will be considered the programation, launch of the control systems, consign generation and the assembly of the already mentioned axis. It will not be considered the cutting tool corresponding part, and due to the lack of material, neither the motors or the sensors could be assembled. Therefore, the final tests could not be performed. Nevertheless, the programation of the machine was completed and the programs were checked by simulation. The programation of the CPU allows the execution of polygons, circumferences, tangentials and skew curves, covering most of the objects present in the industry.

Palabras Clave

Palabras Clave

CPU, Simotion, Eje, encoder, driver, accionamiento, algoritmo, PLC, Simosim, Simatics, Drive-CliQ, Profibus, Telegram.

Hits Gagoak

CPU, Simotion, ardatz, encoder, driver, eragintza, algoritmo, PLC, Simosim, Simatics, Drive-CliQ, Profibus, Telegram.

Key Words

CPU, Simotion, Axis, encoder, driver, drive, algorithm, PLC, Simosim, Simatics, Drive-CliQ, Profibus, Telegram.

Contenido

Lista de Figuras

Lista de Tablas

Memoria del Proyecto

Introducción

Se pretende construir un sistema que guíe de forma automática una antorcha para el corte de chapa. El movimiento se realizará por medio de dos motores que controlarán la posición (X,Y) de la antorcha. Un tercer eje Z se añadirá si así lo precisa la herramienta de corte (figura 1). Este sistema estará preparado para realizar las geometrías más comunes en la industria: polígonos, circunferencias y polinomios de interpolación (para poder realizar curvas si fuese necesario).

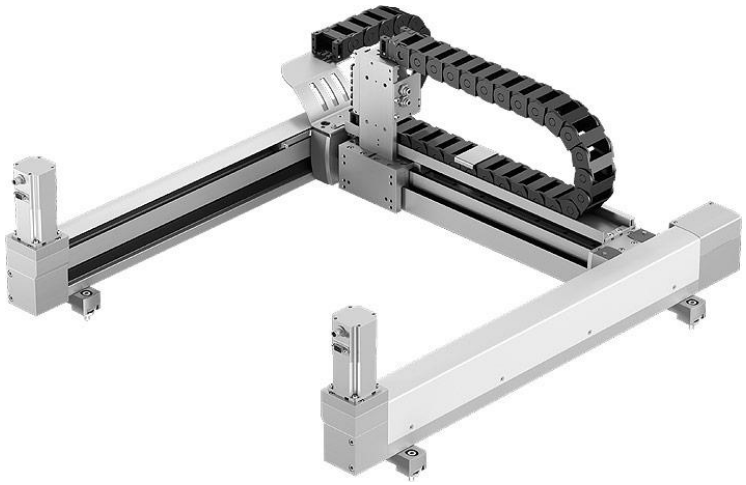


FIGURA 1.EJEMPLO DE MESA DE 2 EJES

Para conseguir esto será necesario escoger y programar la CPU (Unidad central de procesamiento) que va a controlar los motores, y los elementos auxiliares necesarios para que funcionen (drivers, fuentes de alimentación, soportes…), los ejes físicos que van a soportar la antorcha y los sensores que se emplearán para asegurar la precisión. No se realizará la implementación de la herramienta de corte en sí, sin embargo, esta no entraña una gran complejidad técnica y no es necesaria para comprobar el correcto funcionamiento del sistema.

De forma paralela se planteará un programa de ensayos para comprobar la buena marcha del proyecto y evitar que un error en una fase previa pueda ocasionar desperfectos en los equipos.

Contexto

Este proyecto surge como una propuesta de un alumno para realizar un trabajo de fin de grado (TFG) en el departamento de automática de la Escuela de Ingeniería de Bilbao. La idea inicial era construir una mesa de 2 ejes controlada por una CPU, sin necesidad de usar un control CNC (control numérico por computadora) propiamente dicho, para la realización de operaciones sencillas.

Al presentarlo al departamento, surgió la posibilidad de usar el modelo D425 de Siemens. Se trataba de una versión algo antigua que fue adquirida por el departamento y que aún no se había puesto en marcha. Es un tipo de controlador muy interesante para el proyecto ya que combina funciones de los PLCs de Siemens (como los que se emplean para formación en los cursos del departamento) con funciones avanzadas de posicionamiento de ejes.

Después de plantear el Departamento la propuesta del TFG, otro alumno más se interesó por la idea, con lo que el trabajo se dividió en dos TFGs. El primero se centró más en la puesta en marcha de la máquina y el manejo básico de la herramienta de ingeniería, mientras que el presente trabajo se centra en el control de las funcionalidades de la CPU y del resto del equipo para conseguir una aplicación concreta.

Objetivos y alcance del proyecto

En este apartado se exponen los objetivos que se buscan con la realización de este proyecto. El objetivo principal será conseguir que con la CPU de Siemens se consiga poner en marcha una aplicación como es la mesa de corte automática. Para ello se desarrollarán los siguientes pntos:

- Algoritmos de generación de consignas (Polígono, Circunferencia, Polinomio)
- Implementación en CPU
- Implementación en Mesa

Esta aplicación consistirá en el diseño y la implementación de dos ejes capaces de guiar en el plano una antorcha de corte. Ya que el interés principal del proyecto se centra en el área de la automatización, no se considerará la ingeniería relativa a la herramienta de corte.

Además de la aplicación en sí, se habrá conseguido el Know-how necesario para manejar la CPU, así como la puesta en marcha de una mesa con 2 ejes con sus respectivos sensores y accionamientos de control. Esto será muy útil para el Departamento de Automática y Sistemas de la escuela, ya que podrá ser usado por los alumnos de cursos avanzados como parte de su formación práctica en la correspondiente especialidad. De esta forma, el montaje físico servirá como soporte y se aprovechará la programación como ejemplo.

Beneficios que aporta el Trabajo

En primer lugar, éste trabajo permitirá obtener unos algoritmos de generación de consignas de control que después podrán ser utilizadas más adelante en otros proyectos si fuese necesario. Únicamente sería necesario cambiar las funciones de adquisición de datos del encoder y del lazo de control de los motores, los cuales serán propios de cada aplicación. En todo caso la simulación realizada con Matlab de estas consignas seguirá siendo aplicable para las otras aplicaciones de los algoritmos desarrollados.

También se ha conseguido un gran nivel en la programación y el manejo de la CPU, sus entradas y salidas digitales y de los motores y la información de los encoders. Esto le va a resultar muy útil al Departamento ya que es una gama de productos que hasta este momento no se manejaba en la escuela.

Otra aportación del proyecto es el montaje de un sistema de dos ejes cartesianos, con su sistema de acople para motores y sensores de fin de carrera. Este equipo podrá ser usado por el Departamento para realizar pruebas, demostraciones y ejercicios.

Estado del Arte

En este apartado se van a explicar algunos da los conceptos y herramientas que se van a abordar en el proyecto para ayudar al lector no familiarizado con ellos.

- **Cortadoras de 2 Ejes**

  Existe una gran demanda en la industria de estas máquinas de corte que se emplean en gran cantidad de sectores (naval, automóvil, aviación, electrodomésticos…). Es por esto que actualmente existe una gran variedad de diseños para cortadoras. La elección del tipo de diseño depende principalmente del tipo de herramienta de corte que se vaya a emplear. Si se usan herramientas que no entran en contacto con la pieza (oxicorte, laser…) o si lo hacen a través de una herramienta cortante (fresadoras). Esto se debe a que las cargas que deberán soportar la estructura y los elementos móviles serán mucho mayores en el segundo caso.

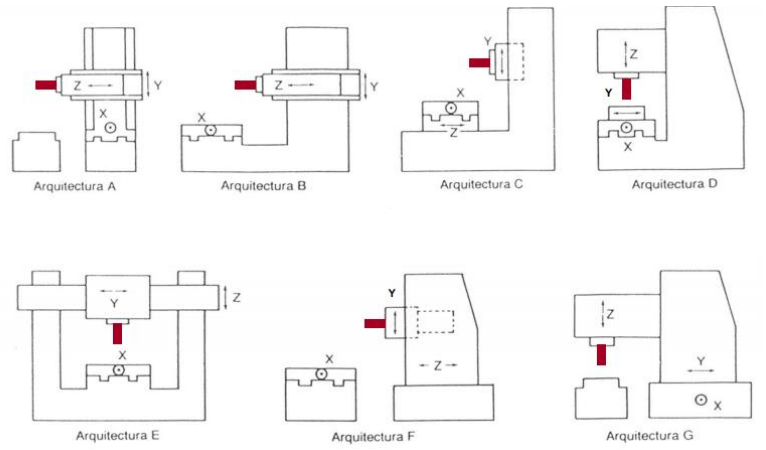


FIGURA 2. ARQUITECTURAS MAQUINA CNC

El otro aspecto técnico a considerar en el diseño será la precisión que se desea obtener. Esto condicionará la elección del sistema de translación del eje (por correa, varilla roscada, husillo, husillo a bolas…). Los sistemas más precisos y con menor rozamiento son también los más caros, por lo tanto, hay que conseguir un equilibrio entre el coste que se está dispuesto a asumir y la precisión que se desea obtener. En la figura 3 se pueden apreciar distintos tipos de ejes, de izquierda a derecha, de bolas, de husillo simple y por correa.

FIGURA 3.TIPOS DE EJES

- **Controladores Industriales**

  Debido al proceso de automatización cada vez mayor que se está dando en la industria existe una gran oferta de CPUs y controladores. Normalmente, cada empresa suministradora ofrece distintas gamas de productos para que se adapten a las necesidades de cada proyecto. Generalmente, el catálogo de productos incluye también los elementos auxiliares (motores, encoders, fuentes de alimentación…) y herramientas de ingeniería (programas para programar los controladores) asociados a cada gama de CPUs. De esta forma, al trabajar con elementos construidos por la misma empresa queda asegurada, en general, la compatibilidad de los componentes. Sin embargo, esto también puede ser una limitación, ya que en muchas ocasiones productos de distintas empresas resultan incompatibles, limitando la libertad del diseño.

FIGURA 4.EJEMPLOS DE CPUS INDUSTRIALES

- **Controladores *Simotion D* de siemens**

  Constituye una gama de PLCs (controladores lógicos programables, por sus siglas en inglés) de Siemens especializada en el control de ejes (posición y velocidad de motores). Se emplea para el control de maquinaria automatizada. Tiene la ventaja de ser un producto modular estando separadas en distintos módulos la CPU, la fuente de alimentación y los drivers de los motores. Esto permite al usuario optimizar el coste del hardware instalado al evitar sobredimensionar alguna parte del equipo. Otro aspecto importante es la

capacidad de interconexión de la CPU, ya que dispone de distintos puertos que soportan varios protocolos de comunicación. En este caso se empleará el sistema *profinet* para crear la red de comunicación entre los distintos elementos del equipo.

FIGURA 5.EJEMPLO DE INSTALACIÓN SIMOTION D

- **Herramienta de Ingeniería *Simotion Scout***

  Es la herramienta de ingeniería de Siemens para la programación y configuración de los equipos de la gama *Simotion* entre otras. Actualmente está siendo sustituido por el programa *TIA Portal* que pretende dar soporte a todos los PLCs de siemens. Sin embargo, es un proceso lento, por lo tanto, aún se emplea SCOUT para las nuevas versiones de la gama Simotion, aunque es posible integrar el programa en el TIA Portal. Se trata de una herramienta muy potente que permite editar, compilar y cargar programas en distintos lenguajes (de bajo nivel, alto nivel y gráficos) además de realizar labores de monitorización y control de los equipos en tiempo real, así como administrar varios dispositivos trabajando en una misma red.

FIGURA 6.CARATULA SIMOTION SCOUT

- **Maquias CNC**

  Son muy utilizadas en la industria. Normalmente se emplea un sistema de control numérico que se encarga únicamente de la interpolación de ejes para la realización de figuras mientras que una CPU tipo PLC se encarga de realizar el resto de funciones auxiliares (cambio de herramienta, refrigeración, sensores…). Sin embargo, en la aplicación que se expone aquí únicamente se usará una CPU para controlar todas las funciones, reduciendo el número de máquinas necesarias. Si que es verdad que se trata de una CPU especializada en movimiento de ejes muy sofisticada. Presenta la ventaja de que, además del control de ejes dispone de entradas y salidas digitales que, en este proyecto permitirán controlar los sensores (una función que en este tipo de máquinas suele estar a cargo del PLC).

  Una desventaja que tendrá respecto de las maquinas CNC al uso es que no dispone de un programa de diseño asistido para el diseño de la estrategia de mecanizado, debiéndose realizar directamente desde la herramienta de ingeniería que programa la CPU.

- **Lenguaje MCC**

  Lenguaje grafico de programación propio de Simotion Scout, "*Motion Control Chart".* La principal ventaja de este lenguaje es que permite controlar muy fácilmente las funciones de sistema. Al ser gráfico, se puede apreciar de forma muy clara como está estructurado el programa y detectar errores de lógica. Esto es una gran ventaja en programas cortos pero que se deben de ejecutar reiteradamente.

- **Lenguaje ST**

  Lenguaje de alto nivel, similar a pascal, que, al igual que el MCC es propio de Simotion Scout, "*Structured Text".* Al ser un lenguaje de alto nivel, acciones tales como los cálculos o el direccionamiento indirecto son fáciles de realizar, además de ser más práctico para programas largos. Sin embargo, es menos intuitivo que el lenguaje gráfico y resulta más fácil la detección de errores de lógica.

Análisis de Alternativas

Tipo de Eje

Como se ha explicado en el apartado correspondiente del estado del arte, existen muchas tipologías de ejes que se pueden usar. En este proyecto se ha primado, sobre todo, el diseño de la parte correspondiente al sistema de control, por lo tanto, el eje en si debe de ser suficientemente preciso para soportar las pequeñas cargas a las que estará sometido durante las demostraciones. Es por esto que en vez de un sistema de husillo de bolas se ha empleado un sistema de correa, más económicos, pero suficientemente rígido para las características del proyecto.

Controlador

Se ha escogido como controlador el D425 de Siemens. Es un controlador especializado en control de ejes que, además, incorpora funcionalidades propias de los PLCs.

Se consideró la posibilidad de utilizar uno de los PLCs empleados habitualmente en el departamento o emplear algún microcontrolador de código abierto comercial. La principal desventaja de ambos es que no integran la funcionalidad de posicionar y controlar ejes del Simotion D425, por lo que de emplear alguna de estas otras opciones el trabajo de programación hubiera sido mucho más laboriosa. Eso sí, no hay que olvidar que esa funcionalidad hay que pagarla: la CPU escogida es más cara que las otras opciones. Si bien es cierto que en el caso de los microcontroladores libres existe mucha información en red debido al auge de herramientas para público no industrial como los drones o las impresoras 3D domésticas.

Otra razón por la que se escogió esta CPU es que este proyecto es una excelente oportunidad para ponerla en marcha y obtener los conocimientos necesarios para manejarla.

Motores

Se han empleado en el proyecto unos servomotores síncronos de Siemens. Tienen la ventaja de ser muy precisos y de disponer de encoder incorporado. Además, al tratarse de un motor de la misma gama de producto del fabricante, lo que facilita su integración con el resto de equipos.

El principal inconveniente es que al ser un motor *"de marca"* es más caro que otras opciones de la competencia. Se pueden encontrar otros motores más accesibles económicamente con una tecnología más simple. Una opción son los motores DC con reductora, mucho más baratos y controlados mediante una salida digital o un servomotor con la electrónica de control apropiada. La principal desventaja de estas opciones es la falta de precisión (sobre todo teniendo en cuenta el gran avance por

vuelta de los ejes escogidos). Además, sería necesario contar con un encoder y realizar la programación necesaria para el lazo de control y la adquisición de información.

Por lo tanto, concluimos que los motores siemens empleados son la mejor opción debido a su precisión, facilidad de manejo e integración con el resto del equipo respecto al resto de opciones, aunque tengan un coste sustancialmente más elevado.

Obtención de consigna

A la hora de obtener la consigna de posición para los motores, existe la posibilidad de usar funciones propias de simotion como son los perfiles de velocidad (*Cam*) o programas para sincronizar ejes (*Paths*).

Los Cams permiten indicar a los motores que posición, velocidad o aceleración deben tener en función del tiempo. El programa genera además una representación de la consigna que se le va a mandar al motor.

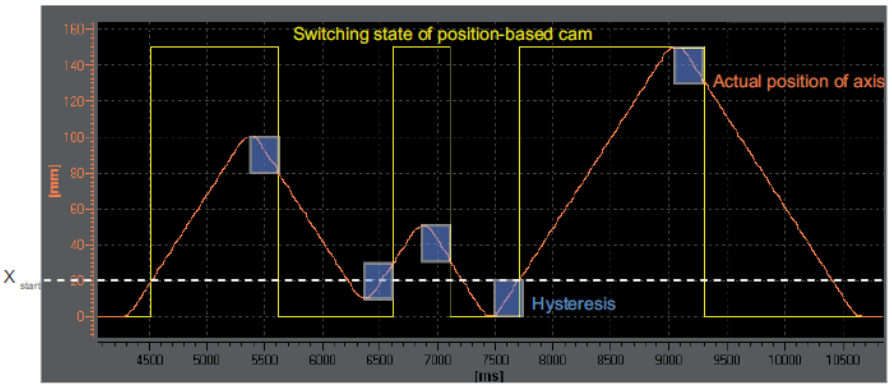


FIGURA 7.DIAGRAMA DE CONSIGNA DE EJE

Aunque para algunas aplicaciones esto puede resultar muy útil, para esta aplicación en concreto resultaría muy engorroso de manejar ya que no se puede "automatizar" mediante código dichos valores de posición, por lo que habría que hacer un cam para cada motor para cada operación. Por ejemplo, para trazar una línea recta con un determinado ángulo habría que hacer dos cams distintos para cada motor calculando previamente la velocidad que deben de tener. Y aun sería más complicado para circunferencias ya que habría que programar una sinusoide en cada cam y sería una sinusoide diferente para cada ángulo.

Otra de las funciones de las que dispone Simotion para sincronizar ejes son los Paths. Se trata de unas funciones de fabrica pensadas para mandar a los motores consignas en función de geometrías por defecto (polígonos, circunferencias…). Esto es, incluyendo funciones de interpolado de ejes propias de una máquina de control numérico:

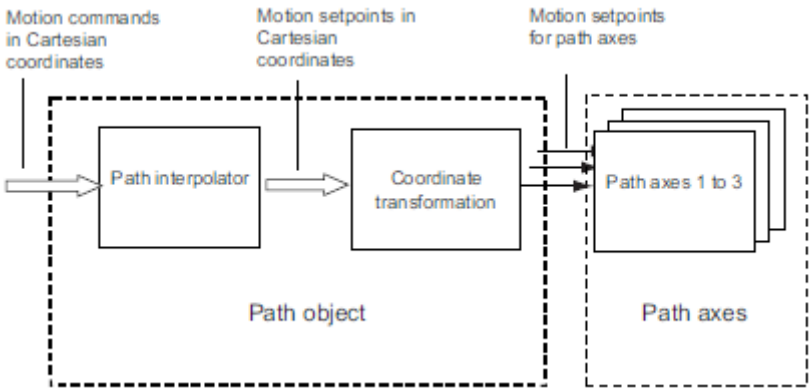


FIGURA 8.DIAGRAMA FUNCIONAMIENTO TO PATH

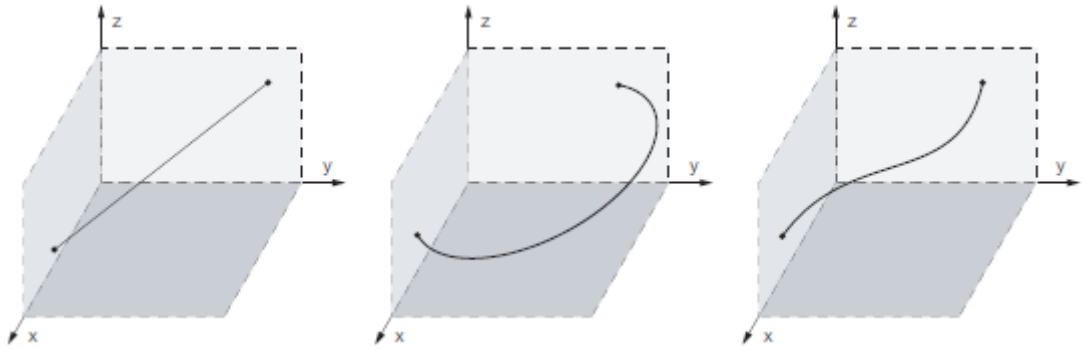


FIGURA 9.GEOMETRÍAS POSIBLES CON HERRAMIENTA PATH

Se trata por lo tanto de una herramienta muy potente que simplificaría mucho el trabajo de programación, ya que evitaría tener que realizar los algoritmos de cálculo de consigna. Sin embargo, este paquete de funciones solo está disponible para versiones de Simotion superiores o iguales a la 4.5, mientras que la CPU de la que se dispone solo soporta Simotion hasta la versión 4.0.

Puede parecer que no se está aprovechando correctamente el potencial de la CPU al no utilizar estas funciones dadas por el fabricante. Sin embargo, no hay que olvidar que la mayor potencialidad de esta CPU es el control tan preciso que tiene de los motores (el lazo cerrado de control). Esto se aprovecha con las funciones _pos() y con los comandos que permiten obtener la posición exacta de cada eje.

Descripción de la solución propuesta. Diseño

En este apartado se va a explicar detalladamente el diseño de la mesa de corte. En primer lugar, se explicarán los algoritmos utilizados para generar la consigna que se ha de enviar a los ejes para generar la figura deseada. A continuación, se explicará cómo implementar dichos algoritmos en el lenguaje de la CPU. Después, los programas de control que llamarán a esos algoritmos y realizarán otras labores auxiliares (como el control de los sensores de final de carrera). Mas adelante, se indicará el protocolo de puesta en marcha de la CPU (configuración de la herramienta de ingeniería para cargar el proyecto) y el cableado de los distintos elementos auxiliares y los motores. Por último, se comentará el montaje de la mesa y el acoplamiento de los motores y los sensores en los ejes físicos.

Algoritmos de Control

La labor de estos algoritmos es generar las consignas de velocidad y posición que se han de enviar a los motores para la realización de las figuras. No se considerará aquí los algoritmos del lazo de control (esto es, los que comparan los datos del encoder con la consigna del motor para asegurar que se sigue la consigna) ya que la CPU dispone de funciones de sistema (ver figura 10) que incluyen este lazo empleando los encoder de los motores.

ST - [D425.Pruebas]

```
21          // bo_EjeY:=FALSE;
22           M_EjeX :=_pos(
23                axis:=EjeX
24               ,direction:=POSITIVE
25               ,positioningMode:=ABSOLUTE
26               ,position:=100
27               ,velocityType:=DIRECT
28               ,velocity:=0.2
29               ,positiveAccelType:=DIRECT
30               ,positiveAccel:=100.0
31           );
```

FIGURA 10.FUNCIÓN PARA POSICIONAR CADA EJE

En la máquina, esto se traduce en el movimiento del punto de la antorcha:

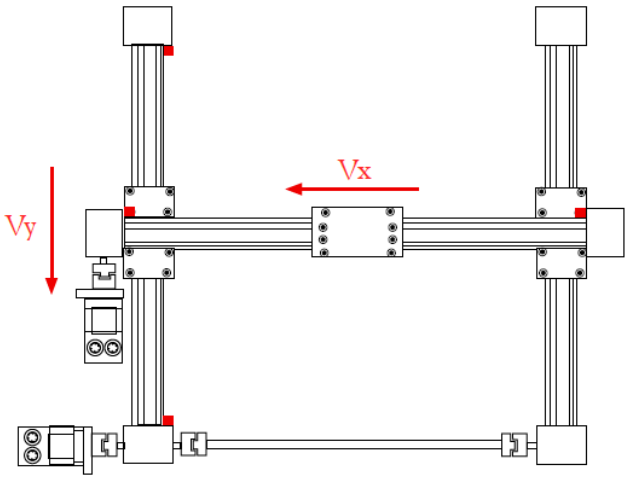


FIGURA 11.MOVIMIENTO REAL DE LA MESA

Simulación:

Para abordar el problema se crearán una serie de funciones lo más sencillas posibles que permitan al mismo tiempo programar las complejas geometrías presentes actualmente en la industria y la estrategia de mecanizado para llevarlas a cabo. Antes de implementarlos en la CPU, **se realizará una simulación empleando el programa Matlab.** De esta forma, se asegurará que los algoritmos funcionan en un entorno *ideal,* libre de los problemas de cualquier máquina real (fallos en los sensores, posibles roturas y sobrecargas, fallos de cableado…). Además, esta simulación será muy útil al programar futuras piezas cuando la maquina esté acabada, ya que permite depurar errores de la estrategia de mecanizado (generalmente fallos de coordenadas o la falta de algún movimiento de translación sin corte).

El primer paso es crear una función de Matlab que simule la función _pos de Simotion, la cual dibujará una línea en un gráfico. También se han añadido los ejes y el punto de la antorcha a la simulación.

```
function TrazarLinea(P1,P2,v,T)
%Funcion que simula el trazado de una linea entre los puntos P1 y
P2, para
%una velocidad de avance de los ejes max de v y un tiempo de ciclo
T

%Se ha puesto en una funcion separada ya que la f. consigna se va a
tener
%que implementar en el plc mientras que Tazar es solo para tener
una
%representacion visual en matlab
V=ConsignaLinea(P1,P2,v,T);

%Punto de inicio
Pant=P1;

%Bucle de Trazado
```

```
for i=1:V(3)
    %Posicion del siguiente punto:
    P(1)=Pant(1)+V(1)*T;
    P(2)=Pant(2)+V(2)*T;
    %Corte realizado en Ti:
    corte(i)=line([Pant(1) P(1)],[Pant(2) P(2)]);
    %SimulacionEjes
    PuntoLaser=viscircles(P,0.1);
    EjeX=line([P(1) P(1)],[0 70]);
    EjeY=line([0 70],[P(2) P(2)]);
    %Tiempo de ciclo
    pause(T);
    %Borra graf. de ejes (el corte se conserva)
    delete(PuntoLaser);
    delete(EjeX);
    delete(EjeY);
    %P pasa a ser Pant para la proxima ejecucion
    Pant(1)=P(1);
    Pant(2)=P(2);
end
%Corte del Ultimo Tramo
P(1)=Pant(1)+V(4)*T;
P(2)=Pant(2)+V(5)*T;
%Corte realizado en Ti:
corte(i+1)=line([Pant(1) P(1)],[Pant(2) P(2)]);
%SimulacionEjes
PuntoLaser=viscircles(P,0.1);
EjeX=line([P(1) P(1)],[0 70]);
EjeY=line([0 70],[P(2) P(2)]);
  %Borra graf. de ejes (el corte se conserva)
    delete(PuntoLaser);
    delete(EjeX);
    delete(EjeY);
end
```

Resaltada en amarillo, está la llamada a la función que genera la consigna de línea, la cuan se explica más adelante y se ha incluido en la función MoverEjes() por comodidad. Los límites de la mesa se dibujan en cada programa de pieza (cada simulación). Las dimensiones que se han dado de forma no se ajustan a las dimensiones de la mensa.

```
%Zona de trabajo
LimSup=line([0 70],[70 70]);
LimInf=line([0 70],[0 0]);
LimIzq=line([0 0],[0 70]);
LimDer=line([70 70],[0 70]);
```

El resultado es muy esquemático, pero perfectamente funcional:

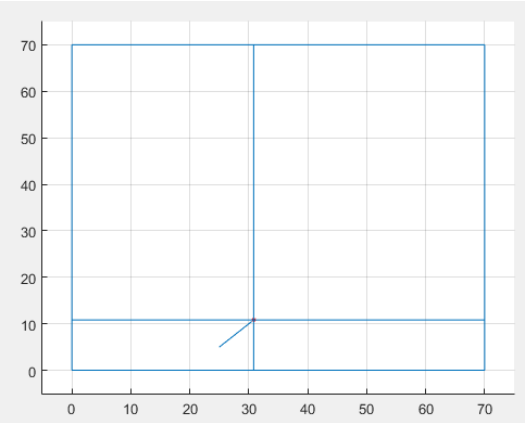


FIGURA 12.SIMULACION REALIZADA CON MATLAB (MESA Y EJES)

Ahora que la parte "grafica" está resuelta, pasamos a los algoritmos en sí. Se realizarán 3 funciones, la primera calculará las consignas para trazar la línea entre dos puntos en coordenadas cartesianas. La siguiente trazará un arco de circunferencia y la tercera un polinomio de interpolación entre una serie de puntos dados. Las funciones de circunferencia y polinomio estarán basadas en la de línea, ya que dichas figuras estarán formadas por una sucesión de llamadas a la función `TrazarLinea()` modificando de forma automática sus parámetros de entrada para crear una aproximación de una circunferencia o un polinomio. La calidad de dicha aproximación dependerá de la precisión de los motores y la rigidez de las guías.

Consigna de línea

Se pretende trazar una línea entre dos puntos A y B conocidos:

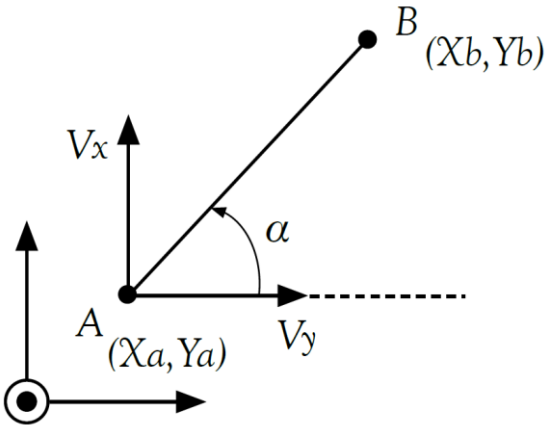


FIGURA 13. CONSIGNA RECTA

Para que el punto de intersección de los dos ejes describa una línea con un ángulo α los dos ejes tendrán que desplazarse a unas velocidades v_x y v_y respectivamente, tal que:

$$v_x = v \cdot cos(\alpha)$$

$$v_y = v \cdot sen(\alpha)$$

Siendo v la velocidad máxima que van a alcanzar los motores durante la operación. El ángulo α se calcula a partir de los puntos inicial y final:

$$\alpha = \text{atan}(\frac{y_B - y_A}{x_B - x_A})$$

Se pasan estos cálculos a Matlab para crear la función `ConsignaLinea()`:

```
function V=ConsignaLinea(P1,P2,v,T)
%Salida: consigna de velocidades para linea
%Entrada: puntos de inicio y fin P1 y P2, velocidad max de los ejes
v
%y periodo de ejecucion T

%Parametros en coordenadas polares
L=sqrt((P2(2)-P1(2))^2+(P2(1)-P1(1))^2);
tita=atan((P2(2)-P1(2))/(P2(1)-P1(1)));
%n°de veces que habra que enviar la consigna
V(3)=round(L/(v*T));
%Velocidaes de Ejex y EjeY rsp.
V(1)=abs(v*cos(tita));
V(2)=abs(v*sin(tita));
%Velocidades para el ultimo tramo
V(4)=((L-V(3)*v*T)/T)*cos(tita);
V(5)=((L-V(3)*v*T)/T)*sin(tita);
%Signos de las velocidades
if P2(1)<P1(1) %Signo del coseno
    V(1)=V(1)*-1;
    V(4)=V(4)*-1;
end
if P2(2)<P1(2) %signo del seno
   V(2)=V(2)*-1;
   V(5)=V(5)*-1;
end
end
```

Un problema que fue descubierto al realizar las funciones fue que, ya que las funciones trigonométricas no son unívocas, había errores de signo cuando el ángulo α queda fuera del 1º cuadrante. Por lo que se tuvo que añadir una sección de código (resaltada en verde) para poner el signo a las componentes de la velocidad. Este problema no aparece al implementar estos algoritmos en la CPU ya que con la función `pos()` se pasa a cada motor tanto la posición como la velocidad deseadas, poniendo la función automáticamente el signo de la velocidad.

Consigna de Circunferencia

Se pretende trazar un arco de circunferencia de centro O, radio R, que comience en el punto P1 y barra un arco de θ grados. Para ello se van calculando una serie de n puntos P_n tal que cada uno de ellos este desplazado un ángulo α (ángulo de

paso) respecto del anterior. De esta forma se obtiene un polígono de múltiples lados que se aproxima a una circunferencia.

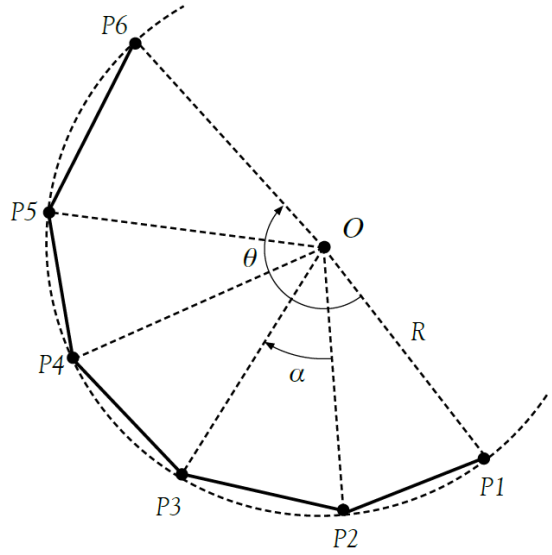


FIGURA 14. APROXIMACIÓN POLIGONAL DE UNA CIRCUNFERENCIA

El algoritmo, por lo tanto, debe de calcular la posición del punto P_i y ordenar a los motores que tracen una recta entre P_i y P_{i-1}.

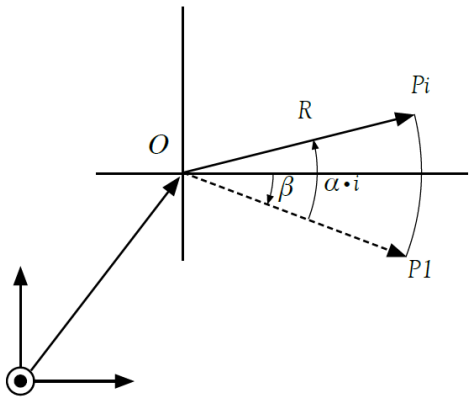


FIGURA 15. POSICIÓN POLAR DE CADA PUNTO DE INTERPOLACIÓN

$$P_{ix} = O_x + R \cdot cos(\beta + \alpha \cdot i)$$

$$P_{iy} = O_y + R \cdot sen(\beta + \alpha \cdot i)$$

Otro problema consiste en calcular el ángulo α óptimo. El límite inferior vendrá impuesto por la rapidez de la CPU mientras que el límite superior vendrá impuesto por la precisión que se desea obtener. En base a esto, siendo T el tiempo de ejecución del programa y sabiendo que, tal como se ha definido la función `TrazarLinea()`, la distancia recorrida en un tiempo T será $v \cdot T$, podemos decir que: el ángulo α con el

que se obtendrá mayor precisión será aquel tal que los puntos P_i y P_{i-1} estén separados una distancia $v \cdot T$.

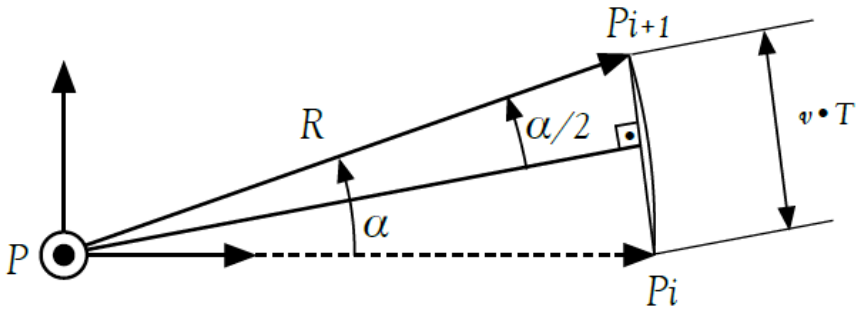


FIGURA 16. SEGMENTO QUE DESCRIBE EL PUNTO EN UN TIEMPO T

Aplicando relaciones trigonométricas, tenemos que: $\alpha = 2 \cdot asen(\frac{3 \cdot v \cdot T}{2 \cdot R})$

Implementamos todos estos cálculos en una nueva función de Matlab:

```
function TrazarArco2(O,P1,tita,v,T)
%Traza arcos con centro en O desde P1 barriendo un angulo tita (en
rad)

%Pasamos P1 a polares (rsp. de O)
vctOP1=P1-O;
beta=atan(vctOP1(2)/vctOP1(1));
%el calculo de beta no es directo pq la f. tan no es univoca
%Primero se calcula el angulo como perteneciente al 1°cuadrante
beta=atan(abs(vctOP1(2)/vctOP1(1)));
%luego se le añade un desfase en funcion del cuadrante en el que
este
if vctOP1(1)<0 && vctOP1(2)>0 %2°Cuadrante
beta=beta+pi/2;
end
if vctOP1(1)<0 && vctOP1(2)<=0 %3°Cuadrante
beta=beta+pi;
end
if vctOP1(1)>=0 && vctOP1(2)<0 %4°Cuadrante
beta=2*pi-beta;
end
%Calculo Radio
R=sqrt(vctOP1(1)^2+vctOP1(2)^2);
%Calculo del angulo de paso:
alfa=2*asin((3*v*T)/(2*R));
%Punto de inicio
Pant=P1;
%Ciclo de generacion de puntos
for i=1:round(tita/alfa)
    P(1)=R*cos(beta+alfa*i)+O(1);
    P(2)=R*sin(beta+alfa*i)+O(2);
    TrazarLinea(Pant,P,v,T);
    Pant=P;
end
```

```
%Calculo del Punto final
    Pf(1)=R*cos(tita+beta)+O(1);
    Pf(2)=R*sin(tita+beta)+O(2);
    TrazarLinea(P,Pf,v,T);

end
```

Consigna Polinomio

Aunque la mayoría de figuras que se realizan en la industria consisten únicamente en polígonos y arcos de circunferencia, cada vez son más comunes las formas alabeadas. Para poder realizar estas geometrías, se parte de una nube de puntos y se emplea un polinomio de interpolación para crear una función que pase por todos ellos. Una vez calculada dicha función, se empleará para ir calculando puntos y uniéndolos por rectas, de la misma forma que se hacía con la circunferencia. En el caso de que se conozca la función que une dichos puntos, se puede modificar el código de tal forma que en vez de calcular los puntos usando el polinomio los calcule directamente de la función dato.

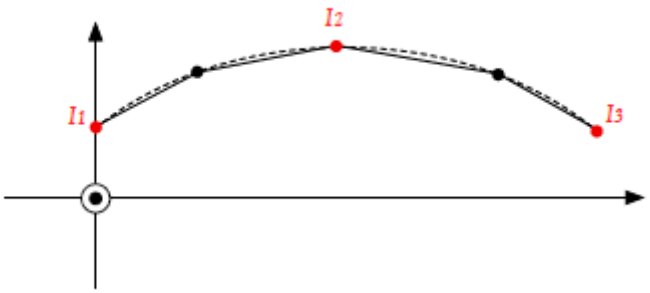


FIGURA 17. APROXIMACIÓN POLIGONAL DE UN POLINOMIO

El método empleado será la representación de Newton del polinomio interpolador. Como la geometría es conocida, se podrán elegir los puntos de interpolación, para simplificar, se usarán 6 puntos unifórmeme espaciados (interpolación en nodos equidistantes).

Calculo teórico del Polinomio:

Partimos de n +1 puntos que se denotarán $x_0, x_1, x_2, \dots, x_n$ y dada una función $f(x)$ se trata de encontrar un polinomio *p(x)* tal que:

$$p(x_i) = f(x_i) \ \ para\ i = 0{,}1, \dots, n$$

El polinomio de interpolación de Newton para esos puntos será:

$$p(x) = f(x_0) + a_1(x - x_0) + a_1(x - x_0)(x - x_1) + \cdots + a_n(x - x_0) \dots (x - x_{n-1})$$

Siendo $a_k = \frac{f(x_k) - q_{k-1}(x_k)}{\prod_{j=0}^{k-1}(x_k - x_j)}$

Para el cálculo de los n coeficientes se emplea la *tabla de diferencias divididas*:

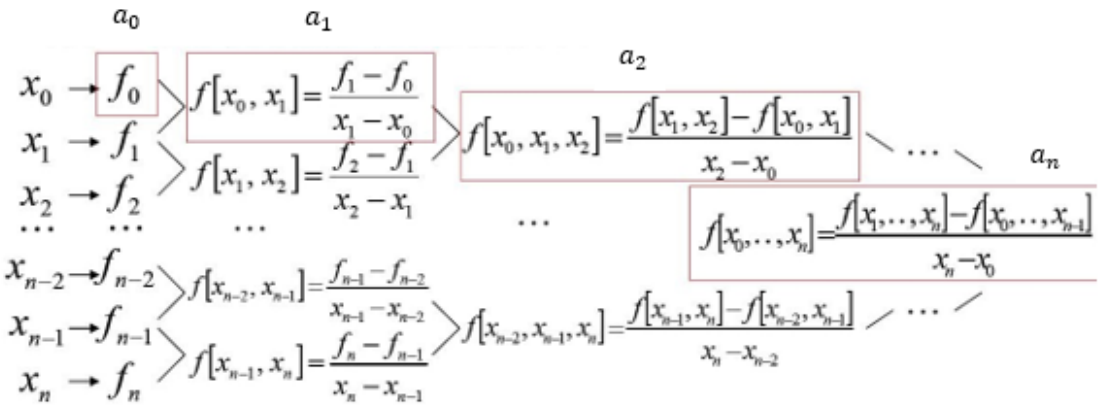


FIGURA 18. TABLA DE DIFERENCIAS DIVIDIDAS

Sin embargo, en el caso particular de que los nodos estén uniformemente espaciados, con una distancia h entre nodos consecutivos. Por lo tanto, cada nodo se puede reescribir de la forma:

$$x_i = x_0 + j \cdot h \; para \; j = 0,1, \dots, n$$

Esto hace que la tabla necesaria para calcular los coeficientes del polinomio se simplifique ya que se en vez de ser *diferencias divididas* tendremos *diferencias finitas progresivas*:

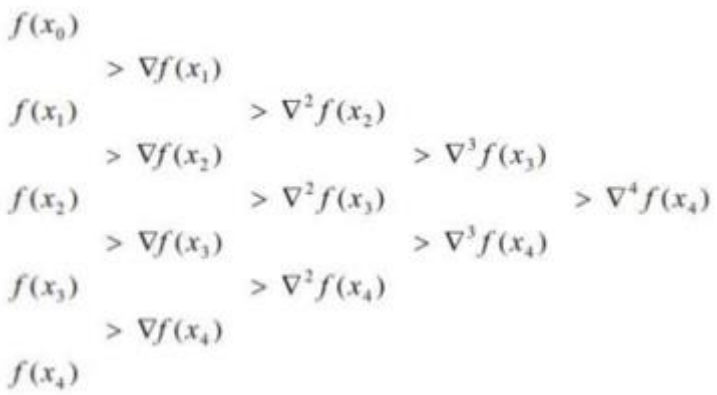


FIGURA 19. TABLA DE DIFERENCIAS FINITAS

Las cuales tienen la ventaja de ser más fáciles de calcular ya que:

$$\Delta^k f(x) = \Delta[\Delta^{k-1} f(x)] = \Delta^{k-1} f(x+h) - \Delta^{k-1} f(x) \quad siendo \;\; \Delta^0 f(x) = f(x)$$

Por lo tanto, los cálculos para obtener los coeficientes del polinomio serán una serie de restas en vez de una serie de divisiones. Esto reduce el tiempo de cálculo de los coeficientes. Sin embargo, a la hora de evaluar los él polinomio en los distintos puntos el cálculo será algo más laborioso. Sin embargo, el error cometido será más pequeño. La evaluación del polinomio se realiza mediante un cambio de variable:

$$p(x) = q(t)\ con\ t = \frac{x - x_0}{h}$$

Siendo, para un polinomio de sexto orden:

$$q(t) = \sum_{i=1}^{6} \Delta^i f(x) \binom{t}{i}$$

Función para el cálculo de los coeficientes del polinomio:

Esta función calcula los n coeficientes del polinomio empleando la tabla de diferencias finitas. Como no se necesitan todos los elementos que forman la tabla, se trabajará con dos arrays, uno en el que se irán guardando las diferencias y otro auxiliar con la columna anterior para los cálculos. De esta forma se optimiza el uso de la memoria.

```
function Dif = ClcDifPolNwt(ArrayY)
%Entrada: Array de 6 elementos que contiene el prod de la
función a
%interpolar
%Salida: Array de 6 elementos que contiene los coeficientes del
polinomio
%interpolador.
%Calculo de coeficientes
    Dif=ArrayY;
    DifAux=Dif;
    L=length(ArrayY);

    for i=1:(L-1)  %Antes 5
        for j=i:(length(ArrayX)-1)
            Dif(j+1)=DifAux(j+1)-DifAux(j);
        end
        DifAux=Dif;
    end
%División por el factorial
    fact=1;
    for i=1:(length(ArrayX)-1) %Antes 6
        fact=fact*i;
        Dif(i+1)=Dif(i+1)/fact;
    end
end
```

El interés de separar el apartado de cálculo de coeficientes de la función `TrazarPolinomio()` es que más adelante, cuando se implemente en la CPU, se llamará una sola vez a la función cálculo de coeficientes y se guardarán estos en memoria para reducir al máximo el número de cálculos necesarios a realizar mientras se evalúan nuevos puntos dentro del algoritmo de trazado de polinomio. Por esta razón, también se ha incluido en la función la división por el factorial de la diferencia dividida correspondiente.

Función Trazar polinomio:

Esta función trazará el polinomio de forma similar a como se hacía para la circunferencia, esto es, se van calculando sucesivos puntos del polinomio y se unen mediante la función `TrazarLinea` para crear un polígono de múltiples lados con la forma aproximada del polinomio.

```
function TrazarPolinomio(ArrayX,ArrayY,v,T)
%Entrada: arrayX y arrayY con los nodos de interpolación. Deben ser 6
y
%estar uniformemente espaciados. Tambien se indica la velocidad v y el
%periodo T.

%Calculo de las diferencias divididas por los factoriales
    coef=ClcDifPolNwt(ArrayY);

%Clc para polinomio
    h=ArrayX(2)-ArrayX(1);
    paso=(ArrayX(length(ArrayX)-1)-ArrayX(1))/60;
%Inicialización bucle
    Pant(1)=ArrayX(1);
    Pant(2)=ArrayY(1);

%Algoritmo de evaluación
    for cont=1:60
        P(1)=Pant(1)+paso;
        P(2)=ArrayY(1);
        %Cambio de variable
        t=(P(1)-ArrayX(1))/h;

        %Calculo de coeficientes en funciónn de t
        coef2=1;
        for k=0:(length(ArrayX)-2) %Antes 4
              coef2=(t-k)*coef2;
              coef3(k+1)=coef2;
        end

        %Calculo del valor del polinomio
        y=coef(1);
        for i=2:(length(ArrayX)-1) %Antes 6
             P(2)=P(2)+coef(i)*coef3(i-1);
        end
        %TrazarLinea
        TrazarLinea(Pant,P,v,T)
        Pant=P;
end
```

Está programada de forma que se pueda introducir un conjunto de puntos tan largo como se quiera, sin embargo, hay que tener en cuenta que el polinomio de interpolación es bastante inestable para conjuntos de más de 4 o 5 puntos, y el error cometido siempre es muy grande en el nodo final. Hubiera sido más preciso utilizar otro método más avanzado, como un spline de 2º o 3º orden. Sin embargo, debido a que la CPU de Siemens no está pensada para realizar cálculos matemáticos resulta muy difícil programar las operaciones matriciales que estos requieren. A modo de ejemplo se muestra la siguiente figura obtenida por interpolación:

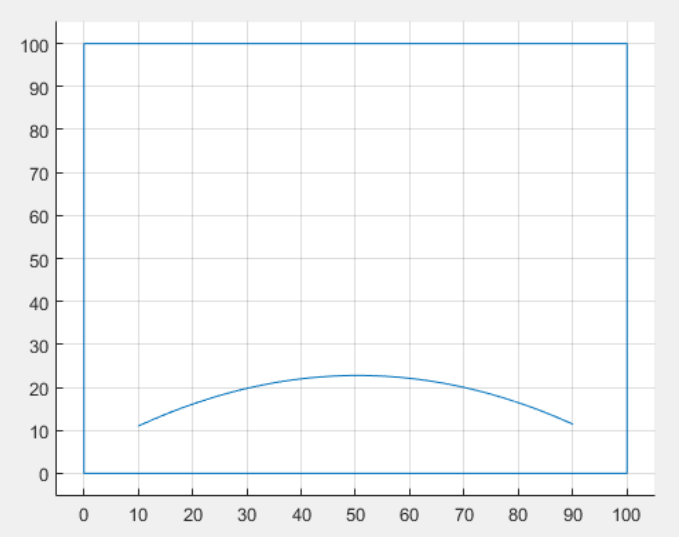


FIGURA 20. EJEMPLO DE POLINOMIO

Empleando los siguientes dados de interpolación:

```
ArrayX=[10,40,60,90];
ArrayY=[11,22,20,5];
```

Obsérvese que para x=90 el valor del nodo de interpolación es de y=5, sin embargo, el polinomio da como resultado un valor de 11 (aprox.). Para solucionarlo, se emplearán más nodos de los que se dibujan, de forma que la interpolación en los nodos centrales sea más exacta y nunca más de 5 nodos en total. Esto ya se explicará con más detalle en el apartado de implementación en la CPU y en las piezas de ejemplo.

En todo caso, aunque este método permita resolver algunas situaciones, en el caso de que se requiera realizar geometrías alabeadas con cierta precisión se deberá recurrir a otro método mas eficiente, como los splines citados anteriormente. Otra opción es contar con la función que define la curva. En dicho caso, el código de la función `TrazarPolinomio()` puede ser fácilmente modificable para evaluar la función en una serie de puntos y obtener su aproximación polinómica. Esto puede ser

muy útil por ejemplo para construir levas o engranajes, ya que la función que define su geometría se obtiene en su diseño.

Función MoverEjes

Aunque la maquina no tenga conectado ningún sistema de corte, si lo tuviese, habría que considerar una función que permita posicionar la antorcha sin cortar para pasar de un contorno a otro (hacer vaciados, agujeros, huecos…). En la simulación esto lo realiza la función `MoverEjes()`, que posiciona los ejes sin trazar ninguna línea. Se usa de la misma manera que la función TrazarLinea.

```
function MoverEjes(P1,P2,v,T)
%Posicionar los ejes sin hacer cortes (para hacer vaciados P.E.

%Se ha puesto en una funcion separada ya que la f. consigna se va a
%tener
%que implementar en el plc mientras que Tazar es solo para tener
una
%representacion visual en matlab
V=ConsignaLinea(P1,P2,v,T);
%Punto de inicio
Pant=P1;

%Bucle de Trazado
for i=1:V(3)
    %Posicion del siguiente punto:
    P(1)=Pant(1)+V(1)*T;
    P(2)=Pant(2)+V(2)*T;

    %SimulacionEjes
    PuntoLaser=viscircles(P,0.1);
    EjeX=line([P(1) P(1)],[0 70]);
    EjeY=line([0 70],[P(2) P(2)]);
    %Tiempo de ciclo
    pause(T);
    %Borra graf. de ejes (el corte se conserva)
    delete(PuntoLaser);
    delete(EjeX);
    delete(EjeY);
    %P pasa a ser Pant para la proxima ejecucion
    Pant(1)=P(1);
    Pant(2)=P(2);
end
%Corte del Ultimo Tramo
P(1)=Pant(1)+V(4)*T;
P(2)=Pant(2)+V(5)*T;

%SimulacionEjes
PuntoLaser=viscircles(P,0.1);
EjeX=line([P(1) P(1)],[0 70]);
EjeY=line([0 70],[P(2) P(2)]);
 %Borra graf. de ejes (el corte se conserva)
    delete(PuntoLaser);
    delete(EjeX);
    delete(EjeY);
end
```

Implementación de los Algoritmos en la CPU

En este apartado se detalla como pasar los algoritmos creados en la programación de la simulación a lenguaje ST de Simotion (el lenguaje estructurado de Siemens, similar a pascal). Se dejará para el siguiente apartado la organización el SO de la máquina y los programas auxiliares. Solo se incluyen aquí las instrucciones que formas la función, en el anexo de programas se puede encontrar el código completo.

FB_TrazarLinea

El principal cambio respecto al algoritmo de Matlab es la supresión del código para dar el sentido de las componentes de la velocidad, ya que (como se ha comentado anteriormente) este lo dará automáticamente la función _pos mediante el parámetro de posición. Otro cambio es el paso de los valores locales a globales con el propósito de poder monitorizarlas desde el panel de control en tiempo real. Por último, en el caso de que el segmento a trazar sea horizontal o vertical, los cálculos darán como resultado un valor de velocidad 0, y esto provoca un error al pasar el parámetro de velocidad a la función _pos. Para solucionarlo, se da un valor arbitrario a estas velocidades en el caso de ser 0 pero, como la posición es la misma que la etapa anterior, el eje no se moverá.

```
//Calculo de la velocidad de cada eje
    IF (PFin[0]-PIn[0])=0 THEN //Da error si den=0 en la TAN
        Vx:=0;
        Vy:=V;
    ELSE
        Alfa:=ATAN((PFin[1]-PIn[1])/(PFin[0]-PIn[0]));
        Vx:=ABS(V*COS(Alfa));
        Vy:=ABS(V*SIN(Alfa));
    END_IF;
    // Solucion error cuando velocidad eje es 0
    IF Vx=0 THEN
        Vx:=V;
    END_IF;
    IF Vy=0 THEN
        Vy:=V;
    END_IF;
    //Se pasan las variables calculadas a las variables globales
        Vx_g:=Vx;
        Vy_g:=Vy;
        Px_g:=PFin[0];
        Py_g:=PFin[1];
     //Envío de ls consignas de movimiento de cada eje:
        M_EjeX :=_pos(
```

```
            axis:=EjeX
            ,direction:=POSITIVE
            ,positioningMode:=ABSOLUTE
            ,position:=Px_g
            ,velocityType:=DIRECT
            ,velocity:=Vx_g
            ,positiveAccelType:=DIRECT
            ,positiveAccel:=100.0
        );
        M_EjeY :=_pos(
            axis:=EjeY
            ,direction:=POSITIVE
            ,positioningMode:=ABSOLUTE
            ,position:=Py_g
            ,velocityType:=DIRECT
            ,velocity:=Vy_g
            ,positiveAccelType:=DIRECT
            ,positiveAccel:=100.0
        );
```

A la hora pedir a la CPU que realice operaciones de forma secuencial, no se pueden realizar llamadas sucesivas como en Matlab (ver simulaciones de ejemplo en el anexo de programación). Esto se debe a que, mientras que en Matlab la operación era instantánea, en la aplicación real es necesario esperar a que la operación haya finalizado para comenzar la siguiente. A efectos de programación, esto implica comparar la posición actual de los ejes con la posición en la que deben de estar al acabar dicha operación. La posición actual vendrá dada por la variable del sistema `<Nombre Eje>.basicmotion.position`. Cada operación vendrá condicionada por un disparador Di (una variable booleana). Cuando el disparador este a TRUE se ejecutará la llamada y después se pondrá a FALSE, después se comprobará la posición de los ejes hasta que lleguen ambos estén en la posición final de la operación. Este proceso se repite con cada operación:

```
    IF Di=TRUE THEN
            F_TrazarLinea( PIn:=Pix,PFin:=Piy);
        END_IF;
        IF Pi+1[0]=ejeX.basicmotion.position AND
P4[1]=ejeY.basicMotion.position THEN
            Di:=FALSE;// desactiva el disparador de la operacion que ha
acabado
            Di+1:=TRUE;//activa el disparador de la proxima operacion
        END_IF;
        IF Di+1=TRUE THEN
            F_TrazarLinea( PIn:=Pi+1,PFin:=Pi+2);
        END_IF;
```

FB_TrazarCirculo

En este caso, es necesario automatizar los disparadores de tal forma que se vayan dibujando en secuencia los múltiples segmentos que componen la circunferencia. A diferencia de Matlab, donde el bucle para realizar las iteraciones está en cada función, en la CPU se ejecutan todas las operaciones a la vez (debido a la necesidad de comprobar continuamente los sensores de carrera). Por lo tanto, el algoritmo de trazado de circunferencia tendrá que adaptarse a esta nueva arquitectura de programa.

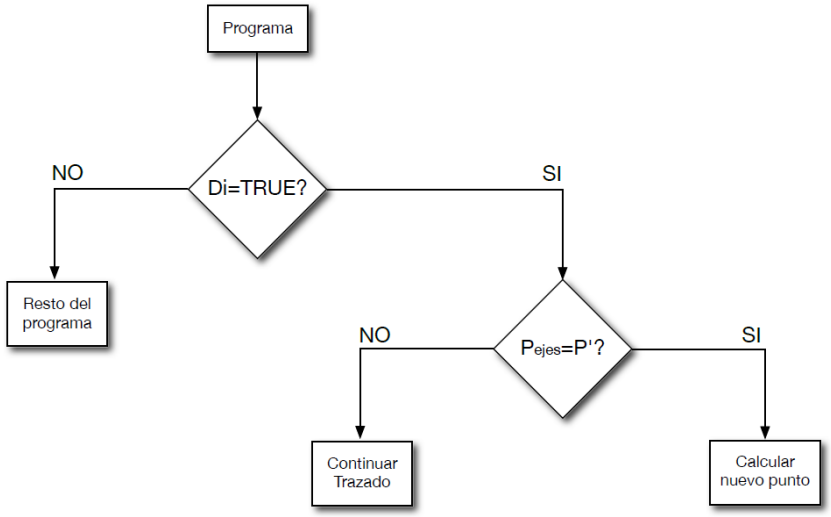


FIGURA 21. DIAGRAMA ALGORITMO CIRCUNFERENCIA

Cuando la operación anterior acaba y el disparador de la operación circunferencia para a TRUE, se inicia el bucle de iteraciones danto valores al punto 0 y al punto 1 del polígono que va a formar la circunferencia. El punto 0 será el punto donde ha acabado la operación anterior y el punto 1 lo dará la función `FB_ClcCirculo` (la cual se explicará mas adelante):

```
IF Pi-1[0]=ejeX.basicmotion.position AND Pi-1[1]=
ejeY.basicMotion.position THEN
            D1:=FALSE;// desactiva el disparador de la operacion que ha
acabado
            Dcirc1:=TRUE;//activa el disparador de la proxima operación
            //Se aprovecha este if para iniciar el ciclo de la
circunferencia
            F_ClcCirculo (O:=P2,P1:=P3,Tita:=Pi/2);
            P[0]:=PCx_g;
            P[1]:=PCy_g;
            PCxant_g:=P3[0];
            PCyant_g:=P3[1];
            Pant[0]:=P3[0];
            Pant[1]:=P3[1];
```

```
            F_TrazarLinea( PIn:=P3,PFin:=P);
        END_IF;
```

Es importante que todas las variables que se emplean en la operación de trazado de circunferencia (El disparador, las coordenadas de los dos puntos de cada segmento y el contador de avance) sean variables globales. Esto es debido a que el programa va a empezar de 0 con cada ciclo de la CPU, reiniciando todas las variables. Esto no era un problema en la realización de polígonos, ya que los disparadores se activaban en una sola vez y eso era suficiente para mandar los motores a donde fuese necesario. Las variables auxiliares Pant y P se emplean para poder pasar los datos a la función `F_TrazarLinea`.

Una vez iniciado el algoritmo está el bloque de código que va a encargarse de seguir iterando. Se trata de una sentencia condicional que comprueba que los ejes han llegado al punto objetivo (P'). Si no lo han hecho, no se realiza ninguna acción. En el caso de que lleguen a ese punto, se aumenta el valor del contador cont y se calcula un nuevo punto P' empleando la función `F_ClcCirculo`, pasando el punto P a tener el valor de P' anterior. Por último, se llama a la función `F_TrazarLinea` para que los motores muevan los ejes a esa nueva posición.

```
        //Circunferencia interior
        IF Dcirc1=TRUE THEN
            IF 0.005>=ABS(PCx_g-ejeX.basicmotion.position) AND
0.005>=ABS(PCy_g-ejeY.basicmotion.position) THEN
                PCxant_g:=PCx_g;
                PCyant_g:=PCy_g;
                F_ClcCirculo(O:=P2,P1:=P3,Tita:=Pi/2);
                Pant[0]:=PCxant_g;
                Pant[1]:=PCyant_g;
                P[0]:=PCx_g;
                P[1]:=PCy_g;
                F_TrazarLinea( PIn:=Pant,PFin:=P);
                PosX_0:=PosX_0+1;
            END_IF;
        END_IF;
```

El último paso consiste en acabar la circunferencia (el criterio de parada del algoritmo). Hay que considerar que el punto en el que queremos que acabe la circunferencia (punto Pfin, en la figura 20.) no tiene por qué coincidir con uno de los puntos por los que pasa el algoritmo.

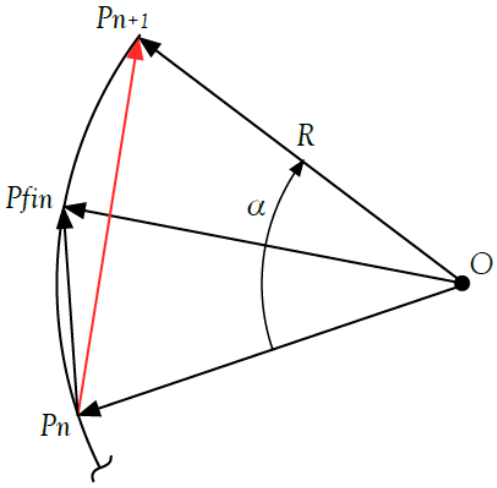


FIGURA 22. CRITERIO DE PARDA ALGORITMO CIRCUNFERENCIA

Por lo tanto, el criterio de parada será que el eje esté a una determinada distancia del punto de fin de circunferencia. La operación Di+1 unirá el último punto del algoritmo con el punto de fin de la circunferencia.

```
            IF 0.1>=ABS(in:= Pfin[0]-ejeX.basicmotion.position) AND
0.1>=ABS(in:=Pfin[1]-ejeY.basicmotion.position) THEN
            Dcirc1:=FALSE;// desactiva el disparador de la operacion
que ha acabado
            cont:=0; //pone el contador de pasos de la circunferencia
a 0
            Di+1:=TRUE;//activa el disparador de la proxima operacion

        END_IF;
```

Por último, está la función `F_ClcCirculo` encargada de ir dando valores a los puntos de la circunferencia. Los cálculos son iguales que para la función de Matlab, sin embargo, en vez de incluir el código necesario para la mover los ejes, únicamente emplea el valor actual del contador para calcular el punto P'. Código:

```
            //Cdv: P1 a polares rsp. de O:
        vctOP1[0]:=P1[0]-O[0];
        vctOP1[1]:=P1[1]-O[1];
            //El clc. del argumento no es directo pq tan no es univoca.
            //Primero se calcula como pertenecente al 1º cuadrante
        beta:=ATAN(in := (ABS(vctOP1[1]/vctOP1[0])));
            //Luego se añade el desfase en función del cuadrante en el
que este
        IF vctOP1[0]<0 AND vctOP1[1]>0 THEN  //2ºcuadrante
            beta:=beta+Pi/2;//(Las F.Trig dan angulos en RADIANES)
        END_IF;
        IF vctOP1[0]<0 AND vctOP1[1]<=0 THEN  //3ºcuadrante
```

```
        beta:=beta+Pi/2;
    END_IF;
    IF vctOP1[0]>=0 AND vctOP1[1]>0 THEN  //4ºcuadrante
        beta:=beta+Pi/2;
    END_IF;

    //Calculo del Radio
    R:=SQRT(in :=(vctOP1[0]*vctOP1[0]+vctOP1[1]*vctOP1[1]) );

    //Angulo de paso
    alfa:=0.03;//----------------------------------->Paso angular
    //Signo de alfa
    IF Tita<0 THEN
        Alfa:=Alfa*-1;
    END_IF;

     //Se calcula el nuevo punto y se guarda en la variable global
     PCx_g:=R*COS(in :=(beta+alfa*cont))+O[0];
     PCy_g:=R*SIN(in :=(beta+alfa*cont))+O[1];

     //Truncatura para igualar precisión encoder
     Paux[0]:=TRUNC(PCx_g*1000);
     Paux[1]:=TRUNC(PCy_g*1000);
     PCx_g:=Paux[0];
     PCy_g:=Paux[1];
     PCx_g:=PCx_g/1000;
     PCy_g:=PCy_g/1000;

     //Avanza un paso el contador
     cont:=cont+1;
```

El valor del paso angular se discutirá mas adelante en el apartado de estudio de la precisión.

FB_TrazarPolinomio

Para trazar el polinomio son necesarias dos funciones auxiliares. La primera se encarga de calcular las diferencias finitas y dividirlas por la factorial correspondiente. Este valor se guarda una variable global, `DifPol_g`, de esta forma solo es necesario calcularlos una vez por polinomio. Por simplicidad, ya que las variables globales no admiten arrays de longitud variable, se emplearán 5 nodos de interpolación. Un número mayor haría inestable al polinomio, como ya se explicó anteriormente.

```
FUNCTION_BLOCK FB_ClcDifPolNwt
        VAR_INPUT
            ArrayY:ARRAY[1..5] OF REAL; //Array con las coordenadas y
de los nodos de interpolación
        END_VAR
```

```
        VAR
            Dif,DifAux:ARRAY[1..5] OF REAL;  //los Arrays se empiezan
en 1 para que sea como matlab
            Fact,i,j:INT;
        END_VAR
        //Asignación de variable de entrada
        Dif:=ArrayY;

        //Calculo de coeficientes (diferencias finitas)
        DifAux:=ArrayY;
        FOR i:=1 TO 5 DO
            FOR j:=i TO 5 DO
                Dif[j+1]:=DifAux[j+1]-DifAux[j];
            END_FOR;
            DifAux:=Dif;
        END_FOR;

        //División por factorial
        fact:=1;
        FOR i:=1 TO 4 DO
            fact:=fact*1;
            Dif[i+1]:=Dif[i+1]/fact;
        END_FOR;

        //Se pasa el valor a variables globales
        FOR i:=1 TO 5 DO
            DifPol_g[i-1]:=Dif[i];
        END_FOR;
```

La siguiente función es la encargada de evaluar el polinomio cuando sea necesario, se trata de una función similar a `F_ClcCirculo`. Cuando se ejecuta se le da un valor de entrada x y evalúa el polinomio en el punto, guardando el valor en la variable global y. De esta forma, el programa accede a ella trazando un nuevo lado del polígono de aproximación. Obsérvese que para calcular el punto necesita de los datos guardados en `DifPol_g`, por lo tanto, es necesario que la función `FB_ClcDifPolNwt` se ejecute antes del ciclo de dibujado.

```
FUNCTION_BLOCK FB_EvaluarPolNwt
        VAR_INPUT
            ArrayX:ARRAY[1..5] OF REAL;
            x:REAL; //punto de evaluación del polinomio
        END_VAR
        VAR
            Coef,coef3:ARRAY[1..5] OF REAL;
            coef2,h,t:REAL;
            i:INT;
        END_VAR

        //OJO: Antes de ejecutarse esta funcion debe de haberse
ejecutado FB_ClcDifPolNwt
```

```
         //Se pasa el valor a variables globales
        FOR i:=1 TO 5 DO
           Coef[i]:=DifPol_g[i-1];
        END_FOR;


        //Calculo de separación entre nodos
        h:=ArrayX[2]-ArrayX[1];

        //Cambio de variable
        t:=(x-ArrayX[1])/h;

        //Calculo de los coeficientes en función de t
        coef2:=1;
        FOR i:=0 TO 3  DO
            coef2:=(t-i)*coef2;
            coef3[i+1]:=coef2;
        END_FOR;

        //Calculo del valor del polinomio
        y:=coef[1];
        FOR i:=2 TO 4 DO
            y:=y+coef[i]*coef3[i-1];
        END_FOR;
    END_FUNCTION_BLOCK
```

Intersecciones

Por último, hay que considerar el caso de que dos operaciones pasen por el mismo punto (fin de un contorno o una operación de posicionamiento que pase por encima de un punto de fin de operación). Cuando se den estos casos, de añade otro if al programa principal para comprobar que operación se encuentra activa. De esta forma solo se iniciará la operación si la precedente esta activa:

```
//Op1---->Posicionar ejes
IF Disp[0]=TRUE THEN
            F_TrazarLinea( PIn:=P1,PFin:=P2);
            IF P2[0]=ejeX.basicmotion.position AND
P2[1]=ejeY.basicMotion.position THEN
                Disp[0]:=FALSE;// desactiva el disparador de la
operacion que ha acabado
                Disp[1]:=TRUE;//activa el disparador de la proxima
operación
            END_IF;
        END_IF;
```

Sistema Operativo de la CPU

En este apartado se comentará como se organizan los programas de la CPU y los programas necesarios para realizar una pieza y otras tareas auxiliares. En primer lugar, se hará una breve explicación sobre cómo se programa en la CPU los programas que deben de ejecutarse en cada caso.

Dentro del árbol de proyecto está el apartado “execution system”:

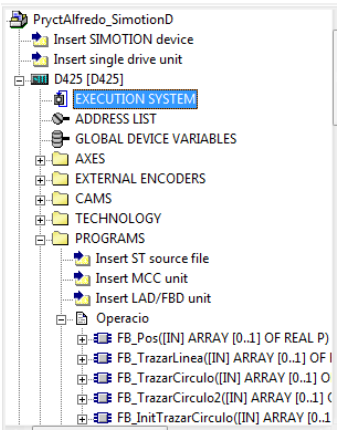


FIGURA 23. VISTA DEL ÁRBOL DE PROYECTO

A su vez, dentro de este apartado hay varias categorías de tareas en las que se pueden incluir los programas del proyecto. Cada categoría varía en función de cuándo o en qué condiciones se van a ejecutar los programas.

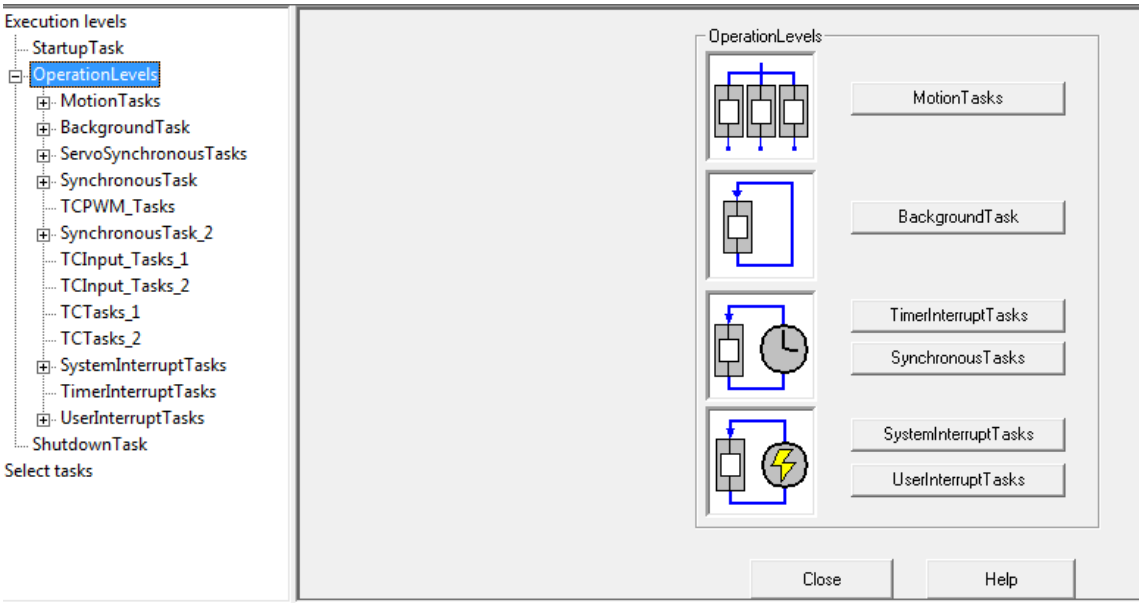


FIGURA 24. TAREAS DEL SISTEMA OPERATIVO DE LA CPU

Las categorías utilizadas en este proyecto son:

- **Startup Task**: los programas incluidos en esta categoría se ejecutarán únicamente al encenderse la CPU.
- **MotionTask**: existen 32 MotionTasks. Se ponen aquí programas que solo deben ejecutarse en algunos casos. A nivel de programación, esto se traduce en que podemos llamar a cualquier MotionTask desde un programa y se ejecutarán los programas que hayamos cargado en ella. Una vez terminados los programas que contiene se vuelve al programa que la ha llamado.
- **BackgroundTask:** los programas contenidos aquí se ejecutan con cada ciclo. Aquí irá por lo tanto el programa principal que decide que MotionTask se ejecuta.
- **SistemInterruptTask:** son programas que se ejecutan en el caso de que se produzca una interrupción. Dentro de esta categoría hay varias subcategorías en función del tipo de interrupción (error en periféricos, en objeto tecnológico…)

Dentro de cada categoría se pueden añadir tantos programas como se quiera y se puede repetir un programa en categorías distintas. Dentro de cada categoría se ejecutarán los programas que contenga de forma secuencial. Como ejemplo, el cuadro de dialogo para añadir programas a una Motion Task:

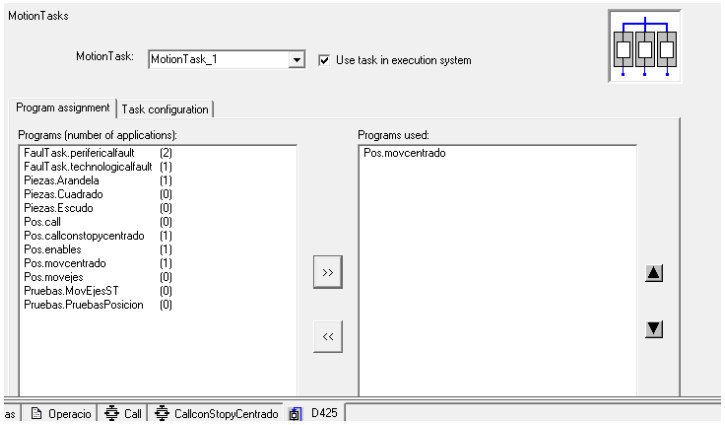


FIGURA 25. ADMINISTRADOR DE PROGRAMAS PARA UNA MOTION TASK

Programa Principal

Sera el programa que se ejecutará en la backgroud task. Su misión es llamar al programa de centrado de ejes cada vez que se inicia la CPU y, una vez centrados iniciar las operaciones de posicionado de ejes (piezas). Además, comprueba periódicamente los sensores de final de carrera para evitar accidentes. Está programado en el lenguaje MCC ya que es el más adecuado para llamar a las Motion Tasks y, en general, para controlar el orden de ejecución de los distintos programas.

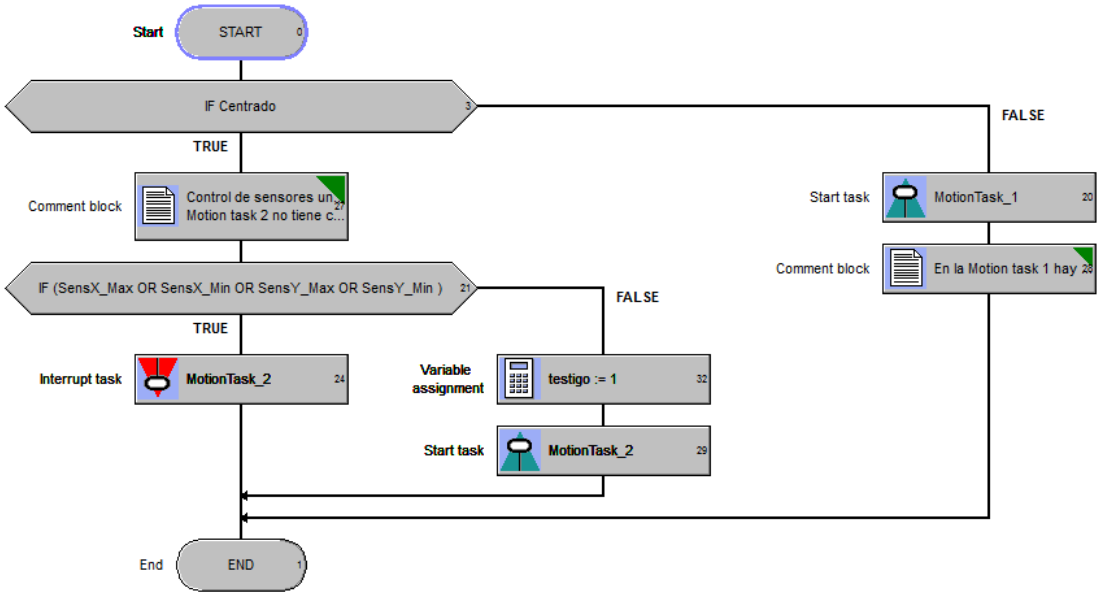


FIGURA 26. PROGRAMA PRINCIPAL

Al iniciarse la CPU, la variable booleana *Centrado* está a FALSE, con lo que se ejecutará la MotionTask_1 en la que está el programa de centrado. Este programa cambiará la variable *Centrado* a TRUE cuando los ejes estén en posición. Entonces se realiza el control de los finales de carrera. Se trata de 4 sensores de contacto conectados a las salidas digitales de la maquina (puerto X137). Cuando el eje alcanza un final de carrera, la entrada digital correspondiente se pone a 1, pasando el valor a la variable booleana correspondiente (ver apartado *Configuración de entradas digitales*). Si alguna de estas variables está a TRUE se ejecuta el comando que interrumpe la MotionTask_2 y, por lo tanto, los ejes se detienen.

En el caso de que no se produzca ningún choque (funcionamiento normal de la maquina) se ejecutará la MotionTask_2, en la cual estarán las operaciones que forman las figuras que la maquina debe de fabricar.

Programa de Centrado

Para centrar la maquina se aprovecharán los sensores de final de carrera, de forma que la posición inicial de la máquina no será el centro de la mesa propiamente dicho, si no una de las esquinas. El programa completo es:

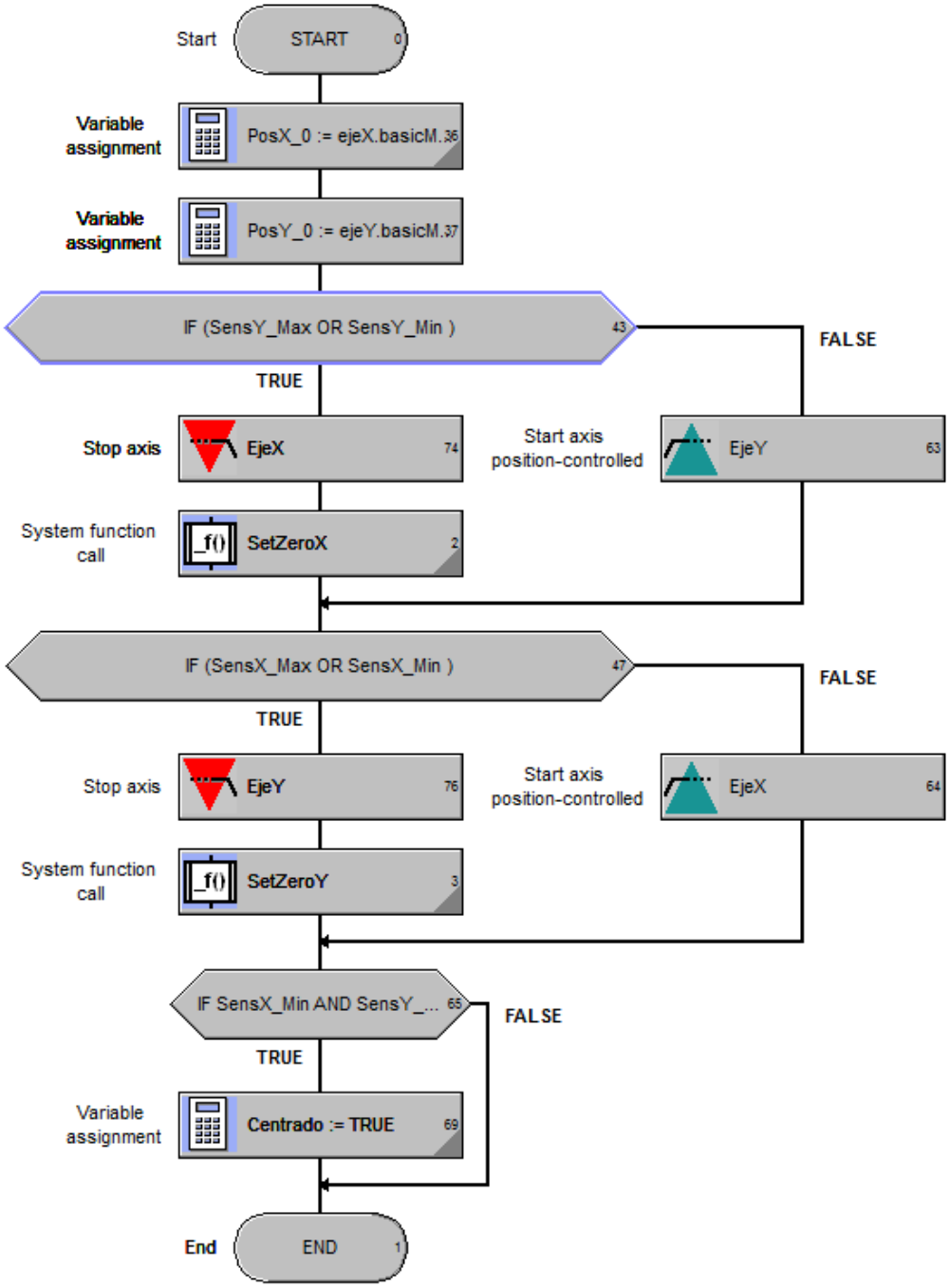


FIGURA 27. PROGRAMA CENTRADO

En primer lugar, se va registrando la posición actual de cada eje en las variables globales PosX_0 y PosY_0, de forma que al centrarse los ejes se pueda ajustar el nuevo cero (funciones `SetZeroY()` y `SetZeroX()`). Después vienen los dos sentencias condicionales de cada motor. Por defecto se le da al motor la orden de avanzar en sentido negativo hasta que toque el sensor de final de carrera correspondiente. Al hacerlo se para el motor, se llama a la función Set correspondiente y por último se le ordena avanzar una distancia de 30 mm en dirección contraria. Esto es importante ya que, tal como se ha planteado el sistema anti-colisión, si se toca algún sistema final de carrera la maquina se para, por lo tanto, el usuario debería tener en cuenta los límites de la máquina y no podría trabajar en los extremos. Por lo tanto, se hace que el área de trabajo sea un poco más pequeña y de esta forma no se dan falsas alarmas.

FautlTask

Para que la CPU funciones es necesario incluir un programa (aunque este vacío) en las categorías de SistemInterruptTask. Se hará por lo tanto un bucle vacío en lenguaje MCC.

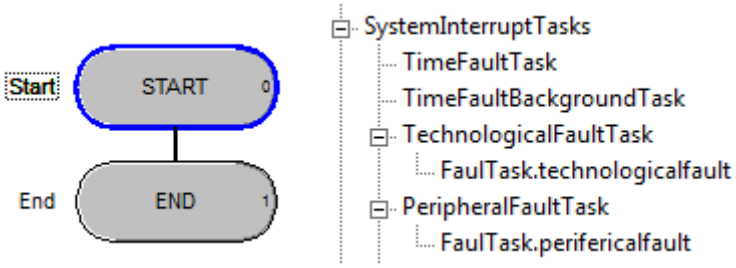


FIGURA 28. PROGRAMA FAULT TASK Y DONDE SE EJECUTA

Sistema de simulación de Simotion (Simosim)

Simosim es un programa de simulación que trabaja con la herramienta de ingeniería que se usa para programar la CPU. Permite comprobar el funcionamiento del software, pero a diferencia de otros programas más modernos y potentes no permite realizar una simulación del hardware de la instalación.

FIGURA 29. INTERFAZ SIMOSIM

Su única interfaz consiste en una ventana donde se muestra una representación del panel de control de la CPU. Desde la CPU, existe un cuadro de dialogo de simulación que activa el programa, esta es la única diferencia respecto a cargar el programa en una maquina física, el resto de la configuración de la conexión se realiza de la misma manera. Para la simulación habrá que configurar el <Acces Point> como <Device> y la interfaz será la creada por simotion “Siemens Simosim Virtual Adapter”.

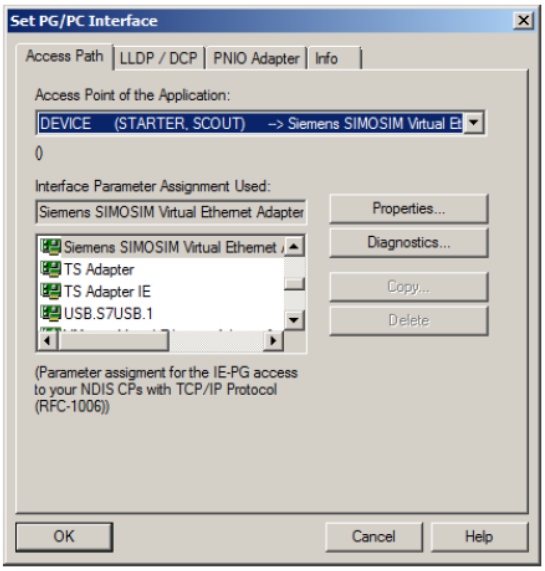


FIGURA 30. INTERFAZ PG/PC PARA SIMULACIÓN

Para evitar un fallo en la conexión entre Scout y el simulador es necesario cambiar el parámetro por defecto de Windows que deshabilita drivers sin firma. Para ello se inicia Windows con F8 y se deselecciona esta opción. Esto es muy importante porque si no la simulación no se iniciará.

Una vez realizada la conexión se podrá probar todo el programa salvo la parte de Drives (ya que la tarjeta integrada SINAMICS no aparece en la simulación. Sin embargo, si que se podrán simular ejes, el panel de control y la tabla de variables globales, por lo que la parte lógica y numérica del proyecto se ha probado en la simulación.

La gran ventaja de Simosim es, primero, la posibilidad de probar programas en cualquier lugar sin tener la instalación, pero también la seguridad, ya que, aunque la CPU es muy segura, los ejes físicos podrían romperse si los motores reciben una consigna de posición que se sale del rango. Ya se ha explicado el sistema de sensores que evita estas situaciones, pero este sistema debe ser probado antes de montar los motores y ponerlos en marcha.

Descripción del equipo y sus distintos módulos (Hardware)

En el siguiente esquema están representados todos los elementos que forman el sistema de control de la mesa:

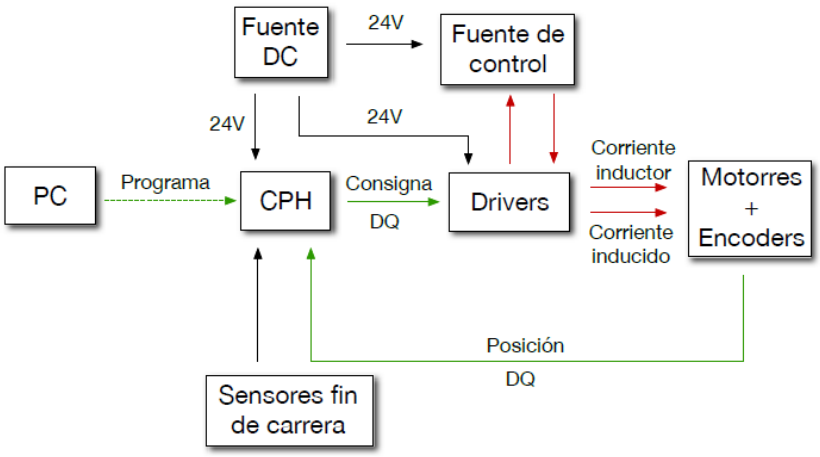


FIGURA 31.ESQUEMA HARDWARE

El elemento central del sistema es la CPU, la cual se controla a través de la herramienta de ingeniería instalada en un PC convencional. La CPU recibe información de los sensores de fin de carrera instalados en los ejes y de los encoders de cada motor. En función de esta información va a mandar una consigna al driver que es el que manda la tensión adecuada a los motores. El driver a su vez necesita de una alimentación de corriente continua de alta potencia suministrada por la fuente de potencia (*Smart line module*). Por último, todos estos equipos necesitan de una fuente de corriente continua para que funcione su electrónica de control, la cual es suministrada por la fuente SITOP.

CPU-Simotion D425

FIGURA 32. MÓDULO DE LA CPU D-425 DE SIEMENS

CPU programable de control. Emplea conexiones ethernet para conectarse tanto con la herramienta de ingeniería (PC) como con los distintos módulos. Es la única parte del equipo con electrónica “inteligente”.

Fuente de Alimentación de electrónica de control (SIPTOP 6EP1336-3BA00)

FIGURA 33. FUENTE DE ALIMENTACIÓN 24V

Se encarga de alimentar la electrónica de control de todos módulos del equipo. Se alimenta de la red y proporciona una tensión constante de 24V de corriente continua.

Fuente de Alimentación Motores (Smart Line Module 6sl3130-6ae15-0ab0)

FIGURA 34. MODULO SMART LINE

Se trata de una fuente de alimentación/retroalimentación (con alimentación por puente de diodos y retroalimentación por fuente de alimentación conmutada usando transistores mosfet IGTB). En modo retroalimentación puede devolver hasta el 100% de s potencia nominal a la red. Esta función puede ser desconectada mediante una entrada digital (para los modelos de 5kw y 10kw) o via Drive-CliQ (para los modelos de 16 kw y 32 kw). También puede trabajar en instalaciones con tierra o con

tierra independiente. Requiere una fuente externa de corriente continua (24V DC) para funcionar.

Alimentación (potencia)	380-480 V AC
Alimentación (electrónica)	24 V DC
Potencia	10 kw

TABLA 1. CARACTERÍSTICAS FUENTE DE ALIMENTACIÓN

Driver Motores (Double motor module 6sl3120-2te19-0aa0)

FIGURA 35. DOUBLE MOTOR MODULE

Los drivers de los motores están controlados directamente por la CPU y se encargan de controlar el motor en función de la información que le llega de la CPU, esto es toma la alimentación continua de la fuente de alimentación de potencia y manda al motor la corriente de inductor y la de inducido en función de la información que le llega de la CPU a través del puerto ethernet con protocolo Drive-CliQ.

Motores (Siemens 1FK70225AK711LG0)

FIGURA 36. MOTORES

Servomotores síncronos con refrigeración por aire y encoder incorporado con una precisión de 512 puntos por revolución. Disponen de conexión Drive-CliQ para control.

Potencia nominal	400 W
Par Máximo	2.65 Nm
Velocidad óptima	6000 rpm
Velocidad Máxima	10000 rpm
Corriente nominal	1.4 A
Eficiencia	86%

TABLA 2. CARACTERÍSTICAS MOTOR

Ejes (Igus Dryline ZLW 1040)

FIGURA 37. EJE ZW1040

Ejes lineales sin lubricación, de gran rigidez y movimiento silencioso (ver tabla). Pertenecen a la gama Dryline, dentro de la cual hay disponibles gran cantidad de accesorios para combinarlos en diferentes geometrías y acoplar distintos tipos de motores, sensores o actuadores. Se emplearán dos ejes (según los ejes cartesianos) para posicionar la antorcha d la herramienta de corte (la cual, como se ha explicado antes no se llegará a montar).

Cableado y conexiones

Se distinguirán dos cableados distintos. Por un lado, esta la red de datos, mediante el sistema Drive-CliQ. Por otro, están las conexiones eléctricas de alimentación de cada módulo y de los motores.

El sistema Drive-CliQ conecta mediante un latiguillo el puerto X100 (parte superior de la CPU) con el puerto X200 de driver de los motores (*Double Motor Module*) este, a su vez, se conecta con los encoder de los motores a través de los puertos X202 y X203 (ver figura 39).

La conexión trifásica del taller va a los magnetotérmicos de entrada. Desde aquí se manda tanto a la fuente de alimentación SITOP (electrónica de control) como a la fuente de alimentación de los motores (electrónica de potencia). La fuente de alimentación manda una tensión de 24V a todos los módulos.

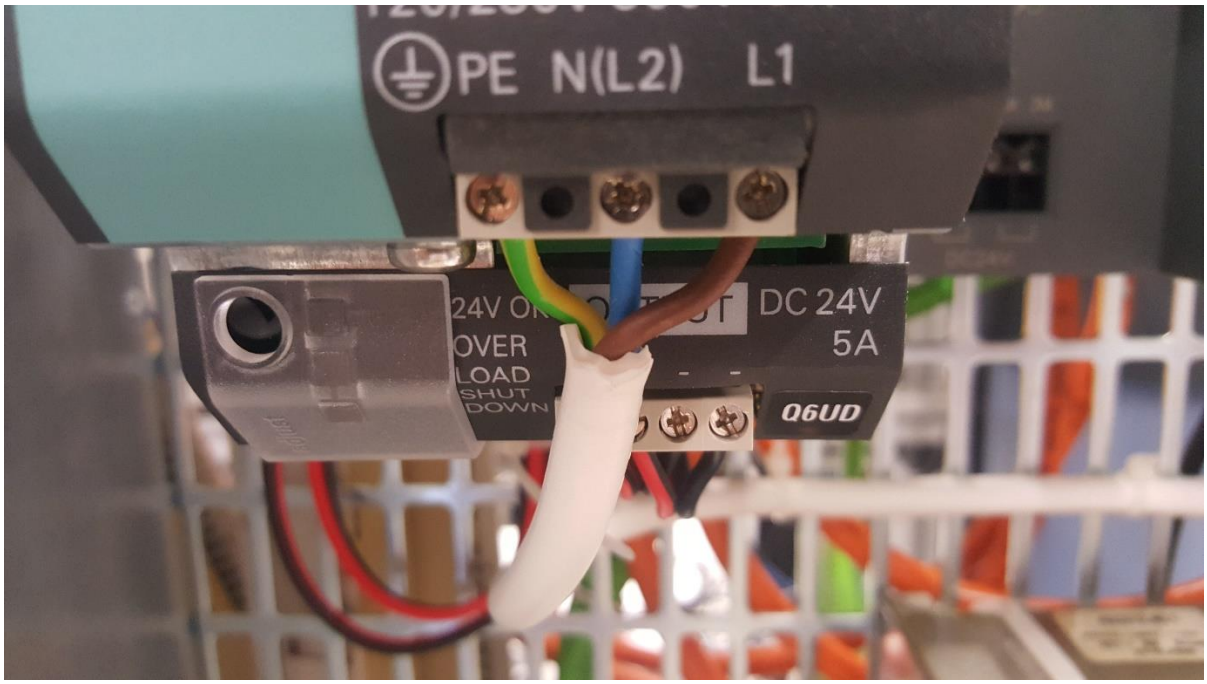


FIGURA 38. CABLEADO SITOP

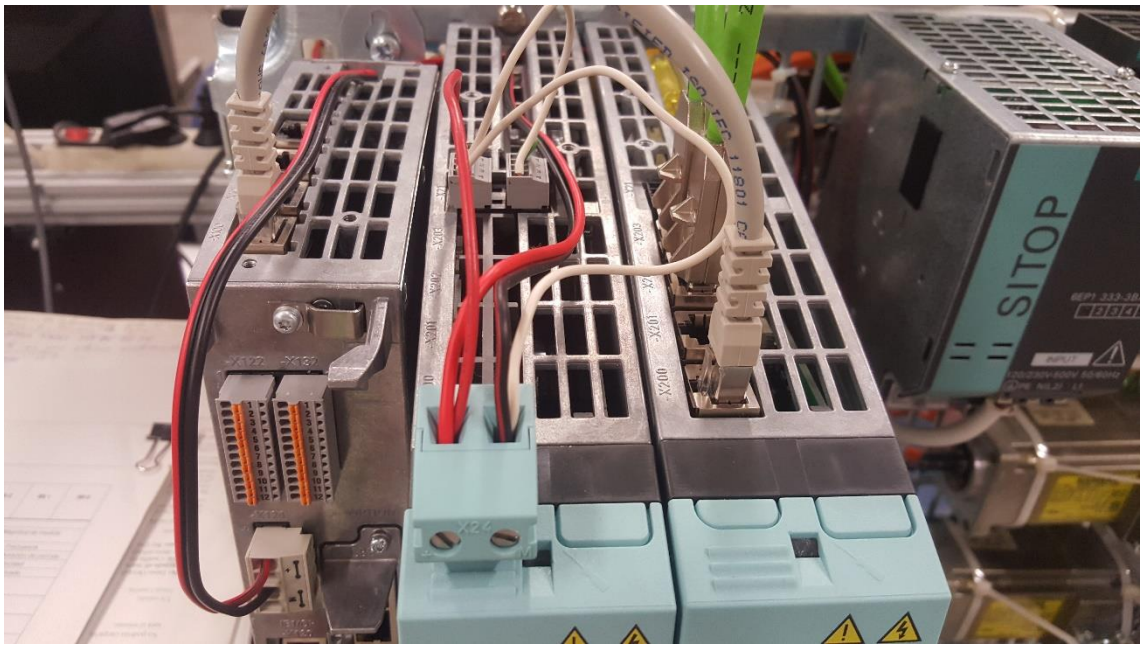


FIGURA 39. CABLEADO DRIVE-CLIQ Y DE ELECTRÓNICA DE CONTROL

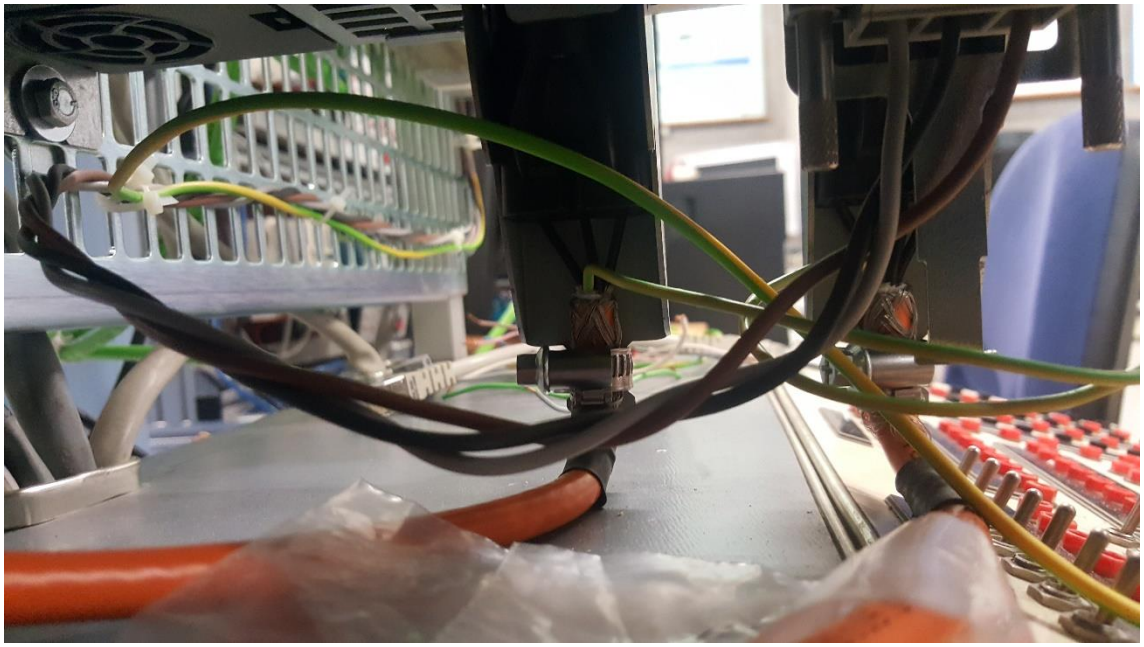


FIGURA 40. CABLEADO DE POTENCIA DE LOS MOTORES

Configuración y procedimiento de carga

Una vez el programa está listo y todos los elementos se han cableado correctamente, se puede iniciar el protocolo de carga.

1. Con la maquina encendida (los dos magnetos a ON) y conectada a la computadora a través del puerto X130. Se espera hasta que en el display de la CPU aparezca un 6 con punto parpadeando (esto indica que la máquina está lista para cargar el programa, ver el manual *D4x5 Comissioning* para mas información)

FIGURA 41. PUERTO X130 Y PANEL DE CONTROL DE LA CPU

2. Se configura el protocolo de comunicación y el punto de acceso desde el cuadro de dialogo <Accesible Nodes> . Desde este cuadro de dialogo se configura <Acces Point> cono <S7 Online> y la interfaz PG/PC al puerto <Conexión de Red Intel(R) PRO/1000MT.TCPIP.1>.

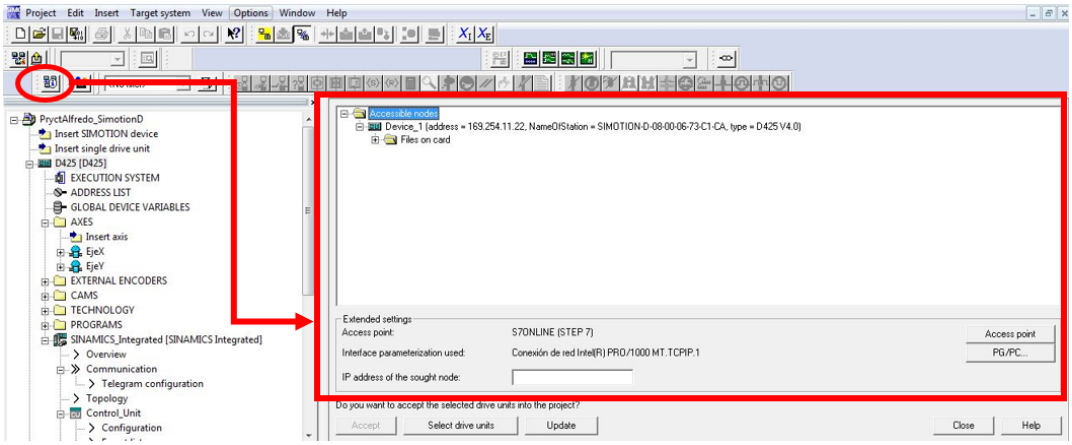


FIGURA 42. PANEL "ACCESBLE NODES"

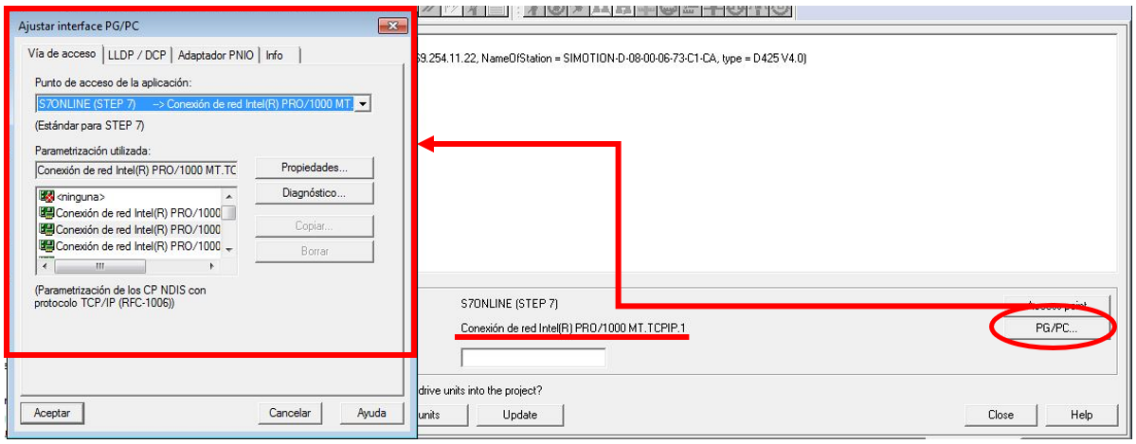


FIGURA 43. PANEL CONFIGURACIÓN PC/PG

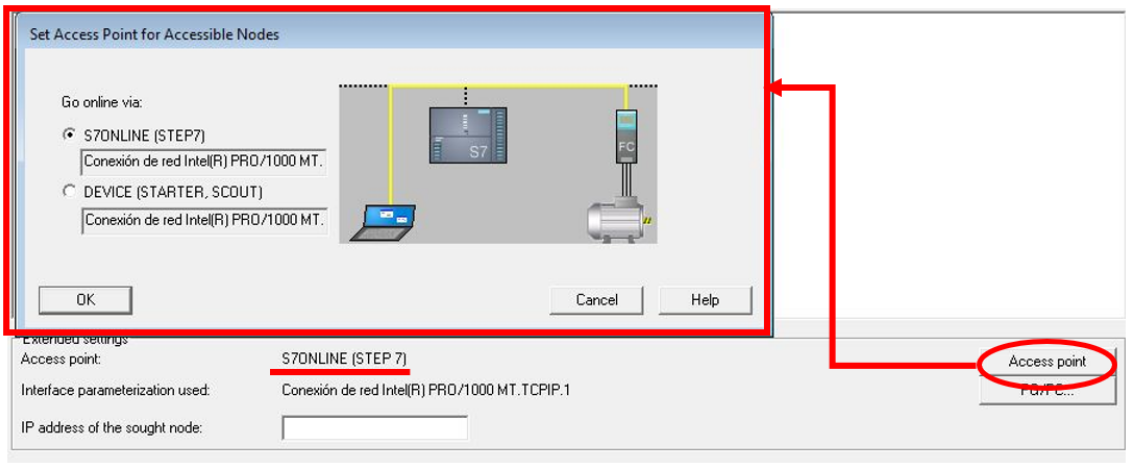


FIGURA 44. PANEL DE CONFIGURACIÓN "ACCESS POINT"

En este punto, la herramienta de ingeniería debería de ser capaz de encontrar a la CPU (D425), condición necesaria para continuar.

3. Se accede al menú <Target> y desde el al submenú <Select Target Devices…>. En el cuadro de diálogo se selecciona la CPU (D425) y el controlador (SIMATICS integrated) así como la opción <S7Online> como punto de acceso.

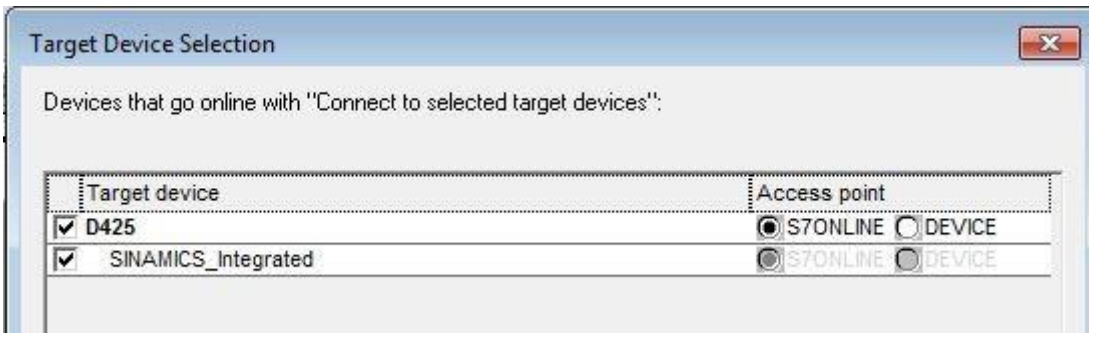


FIGURA 45. CUADRO DE DIALOGO PARA SELECCIÓN DE CPU (DE ENTRE LAS CONECTADAS)

4. Se establece conexión entre la herramienta de ingeniería y la CPU (botón amarillo):

   En la parte inferior de la ventana, un indicador mostrará que la CPU está en línea:

5. Los otros dos botones permiten cargar el programa en la CPU (solo lo permitirá si el compilador no detecta errores) y deshacer la conexión. Es recomendable deshacer la conexión antes de apagar la CPU o el PC.

Cuando la CPU está en línea se puede ver en el árbol de proyecto qué partes del proyecto están sincronizadas (coinciden con) las que están cargadas en ese instante en la CPU.

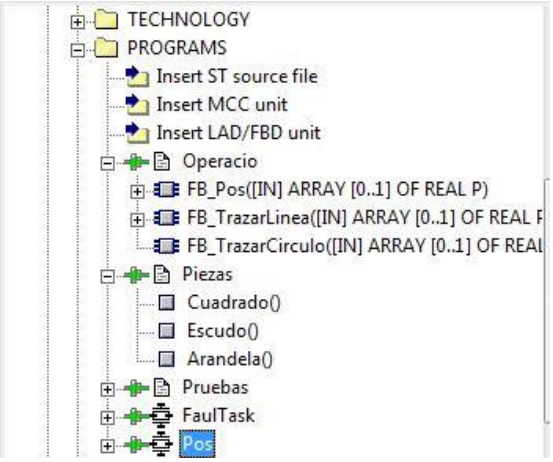


FIGURA 46. SECCIÓN DE PROGRAMAS DEL ÁRBOL DE PROYECTO

Esto se indica mediante un icono que representa las conexiones. Cuando las dos partes están unidas significa que hay conexión. Si el lado derecho está en rojo, indica que el código o la configuración que está cargada en la CPU no se corresponde con la del proyecto. Después de cada carga se actualiza el proyecto, por lo que todas las conexiones deberían estar en verde.

Configuración de Motores (ejes y drivers)

El control de los motores (consigna y lazo de control) se hace mediante dos OT (objetos tecnológicos) dentro del árbol de proyecto. El eje ("*Axis*") se controla desde las instrucciones del programa y el drive es el que ataca al motor.

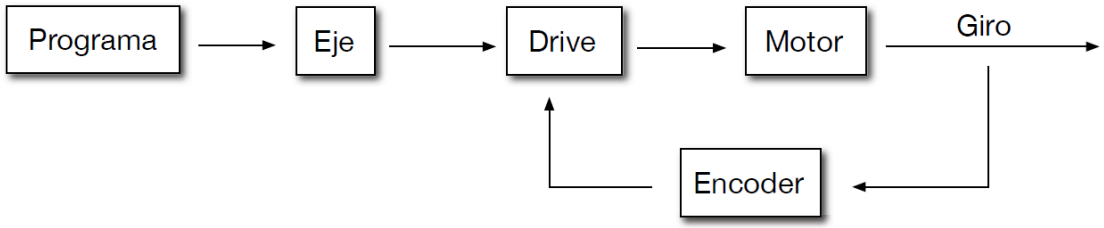


FIGURA 47. LAZO DE CONTROL DE CADA EJE

Primero el drive, que incluye las especificaciones técnicas del motor y es la parte del programa que maneja la consigna que se le manda y recibe la información del encoder integrado en el motor para cerrar el lazo de control. Gracias a la comunicación Drive-CliQ no es necesario configurar los parámetros del motor, ya que la herramienta de ingeniería puede acceder a él mediante la red que se ha creado y saber el modelo y las especificaciones. No hay que confundir el OT (parte del programa) con la tarjeta de electrónica de potencia que manda energía a los motores (no inteligente). Como se emplean dos motores, hay dos drives para controlarlos. Cada drive este asociado a un motor real.

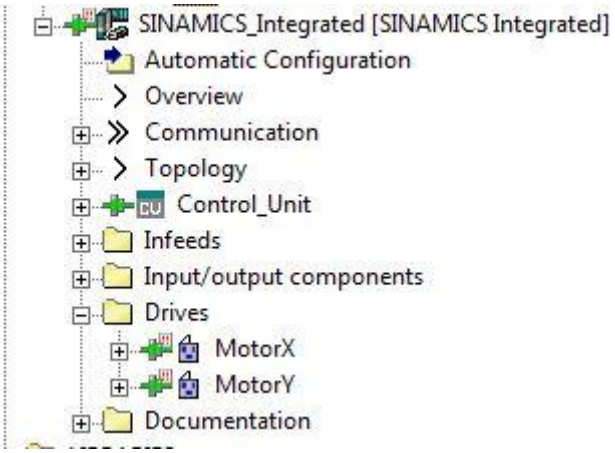


FIGURA 48. SECCIÓN DEL ÁRBOL DE PROYECTO CORRESPONDIENTE A TARJETA SINAMICS

Como se puede apreciar, los drives forman parte de la unidad SINAMICS, que es una unidad de control de ejes (accionamientos en general) alojada en la misma carcasa que la CPU, pero independiente de esta. De hecho, se puede encontrar como producto independiente en el catálogo de Siemens.

Por otro lado, el OT eje es una herramienta que permite mandar una consigna en unidades de ingeniería (una velocidad en m/s, por ejemplo) desde el programa de control y se va a encargar de mandar al drive esa consigna. Un aspecto importante del eje es que no está asociado a un motor real, sino que se le puede asociar a distintos

motores o a ninguno. Esto es muy útil en la etapa de simulación ya que Simosim no puede simular la parte física de la instalación.

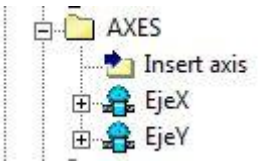


FIGURA 49. SECCIÓN DEL ÁRBOL DE PROYECTOS CORRESPONDIENTE A LOS EJES

A la hora de crear un nuevo eje, hay que configurar que tipo de eje va a ser en función del accionamiento que vaya a controlar (lineal, rotativo, eléctrico, hidráulico…) y dar los valores que relacionan las variables de ingeniería con las variables de control. Por último, se asocia con uno de los drives existentes.

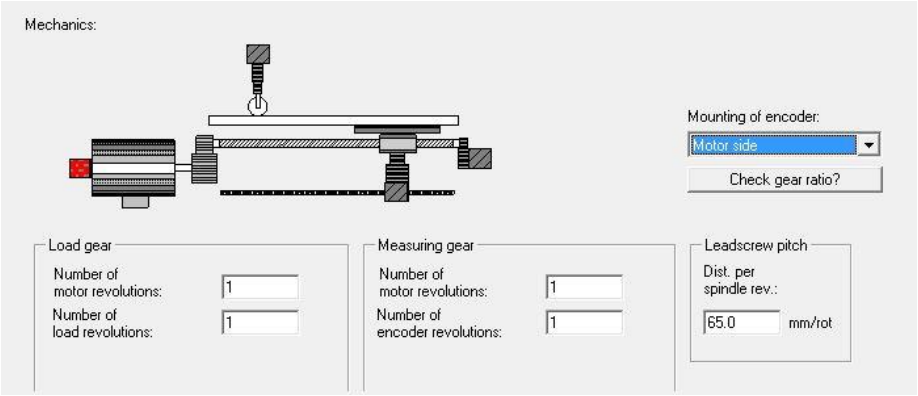


FIGURA 50. CONFIGURACIÓN DE LOS PARÁMETROS FÍSICOS DEL EJE

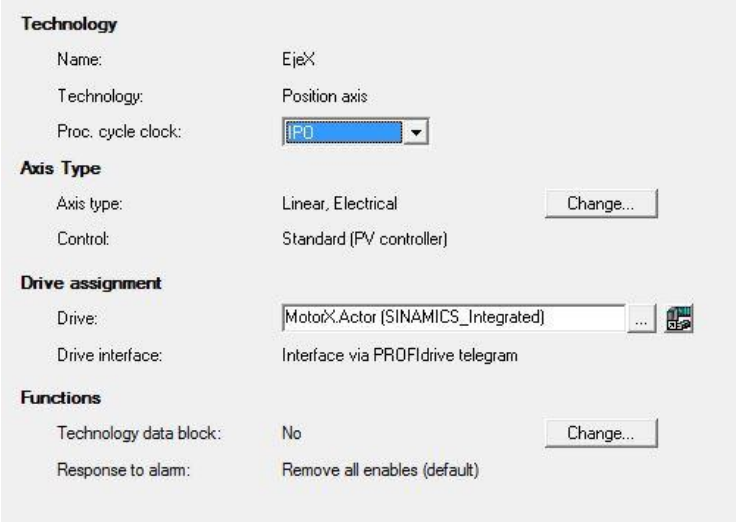


FIGURA 51. CONFIGURACIÓN DEL EJE Y CONEXIÓN CON DRIVE

Nótese que todas las propiedades pueden modificarse en cualquier momento después de crear el eje.

Estudio de la precisión de las geometrías obtenidas

El primer dato con el que se estimará la precisión será el grado de definición que tiene el encoder, esto es 512 posiciones por vuelta. Se considerará que el lazo de control de la CPU es capaz de llevar cada motor exactamente a la posición de la consigna según el encoder. Tampoco se considerarán errores debidos al mal montaje de los ejes o a la falta de rigidez de algún elemente (correas principalmente y, en menor medida, elementos estructurales). El otro dato necesario es el avance por radian de los ejes, que es de $\frac{65}{2\pi}$ mm/rad.

Tenemos la primera relación: $L=\frac{65}{2\pi}\beta$ siendo L la distancia recorrida por el eje y beta el ángulo girado por el motor.

Por otro lado, el encoder tiene 512 divisiones, por lo tanto, cada división cubrirá un ángulo de $\frac{2\pi}{512} = \alpha$ rad/div.

Esto permite obtener el error angular cometido al posicionar el motor:

$$\beta_{real} \in [\beta_{encoder} - \frac{\alpha}{2}, \beta_{encoder} + \frac{\alpha}{2}]$$

Aplicando la primera relación pasamos los valores angulares a valores de longitud mediante la primera relación obtenemos (con L en mm)

$$L_{real} \in [L_{encoder} - \frac{0.127}{2}, L_{encoder} + \frac{0.127}{2}]$$

La posición del punto vendrá determinada por la posición a la que se haya enviado a cada eje, ya que cada uno define una coordenada cartesiana del punto, por lo que tenemos que:

$$x_{real} \in [x_{encoder} - \frac{0.127}{2}, x_{encoder} + \frac{0.127}{2}]$$

$$y_{real} \in [y_{encoder} - \frac{0.127}{2}, y_{encoder} + \frac{0.127}{2}]$$

Concluimos que se podrá asegurar que la posición real del punto estará en el interior de un cuadrado de 0.127 mm de lado siendo la posición marcada por el encoder el centro de dicho cuadrado (ver figura 52).

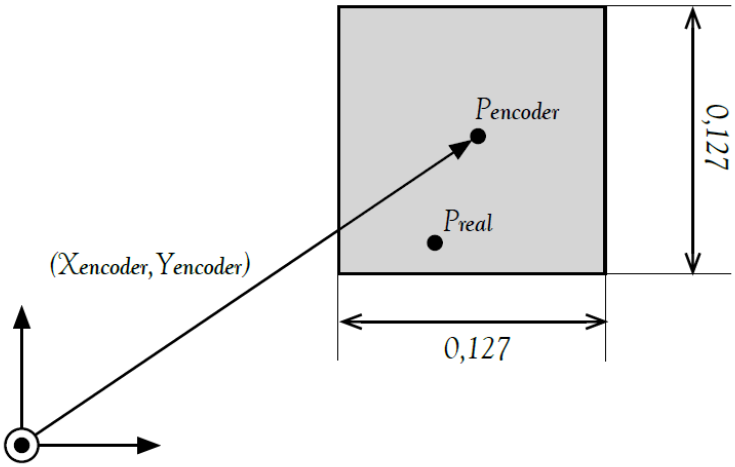


FIGURA 52. ERROR DE POSICIÓN POSIBLE

Entradas digitales y Sensores

El equipo dispone de dos puertos (el X122 y el X132) con su correspondiente soporte físico para emplearlos como entradas o salidas digitales. Estas salidas están eléctricamente aisladas de la tarjeta de control para protegerla de posibles cortocircuitos o sobrecargas. Funcionan a 24V, por lo que se emplea la misma fuente de alimentación que la usada para la CPU. El esquema de conexiones se puede encontrar en el manual *"D4x5-2 operating manual", p58.*

FIGURA 53. PUERTOS I/O X122 Y X132

Este puerto pertenece al controlador Simatic integrado. En versiones modernas del la herramienta de ingeniería se puede emplear el *direccionamiento simbólico* para asignar un entrada o salida de una variable directamente desde la *adres list* de la CPU D425, sin embargo, en esta versión es necesario configurarlas en la sección *Control*

Unit del árbol de proyecto. El primer paso es hacer que la tarjeta S120 envíe y reciba datos de las I/O a la CPU. Para ello, hay que ir al cuadro de diálogo *Telegram Configuration* y añadir el *telegram* 390:

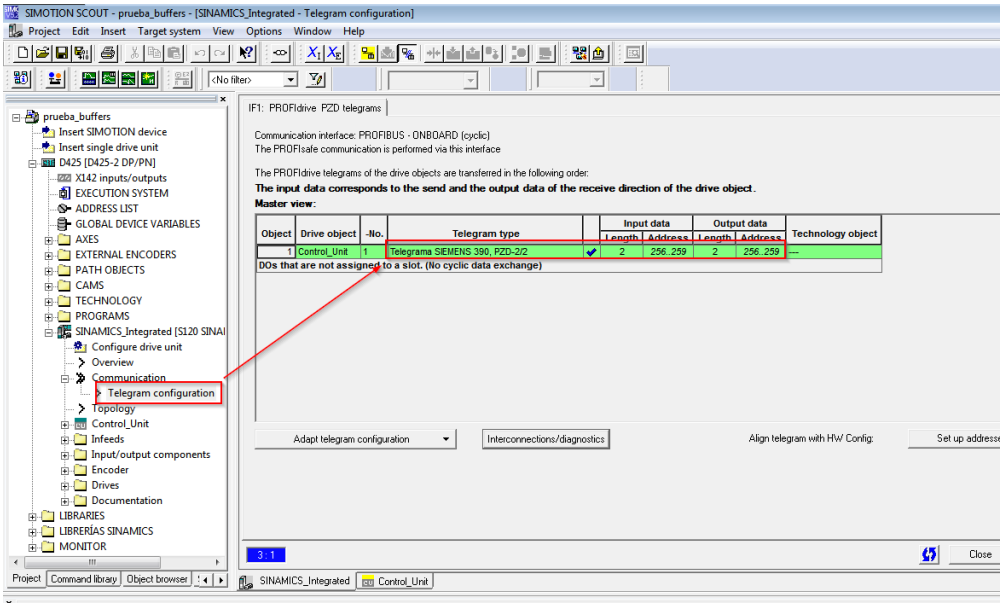


FIGURA 54. CONFIGURACIÓN TELEGRAM 390

Una vez seleccionado el *telegram* hay que conectarlo al puerto de entradas digitales con la opción *Set up Adress.* Una vez configurado, es conveniente comprobar que en efecto se esta enviando información abriendo el cuadro de dialogo *Interconnections/Diagnostics* y una vez dentro accediendo a la pestaña *transmit directions.* Haciendo clic en el botón *E/Digital* aparecerá la ventana indicando las direcciones de las entradas/salidas a las que está conectado. Esta ventana es muy importante, ya que indica en qué posición del *telegram* esta la información de un pin en concreto (números del 0 al 15 a la derecha del recuadro).

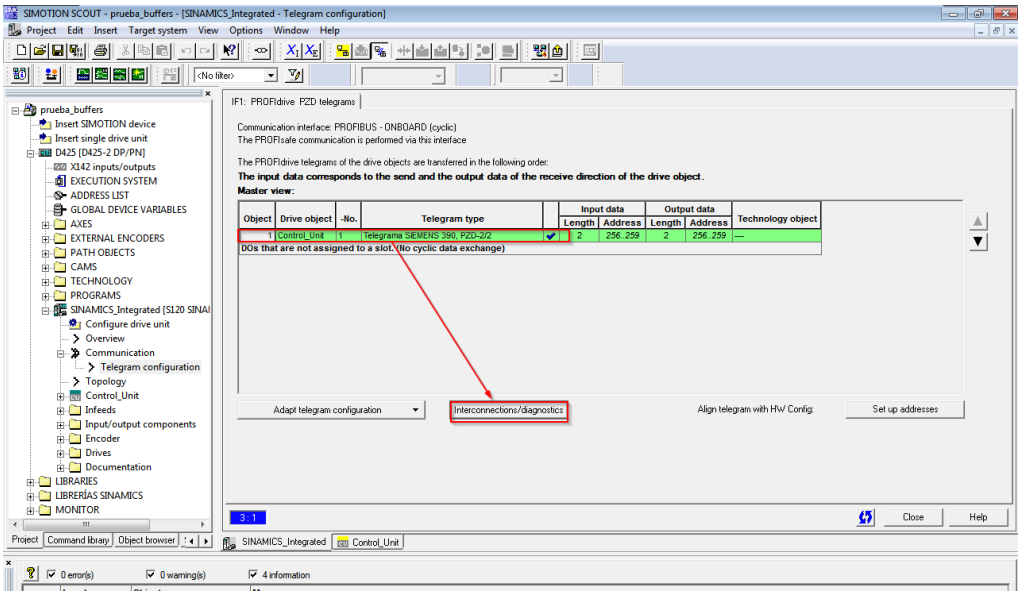


FIGURA 55. CONFIGURACIÓN TELEGRAM

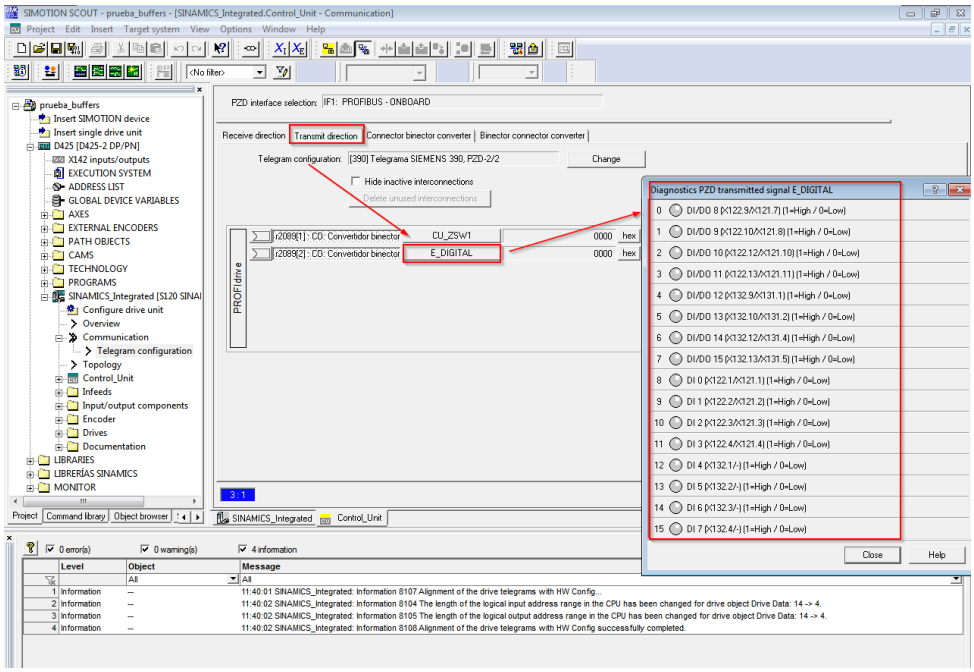


FIGURA 56. COMPROBACIÓN CONEXIONES

El siguiente paso es acceder configurar cada pin de las I/O como entrada o salida. Esto se hace desde el cuadro de dialogo Inputs/Outputs, dentro de la sección *Control Unit* del árbol de proyecto. Aparece un esquema de las salidas en cada puerto y un icono que representa una entrada o salida, haciendo clic en el icono se configura ese pin como una entrada o como una salida.

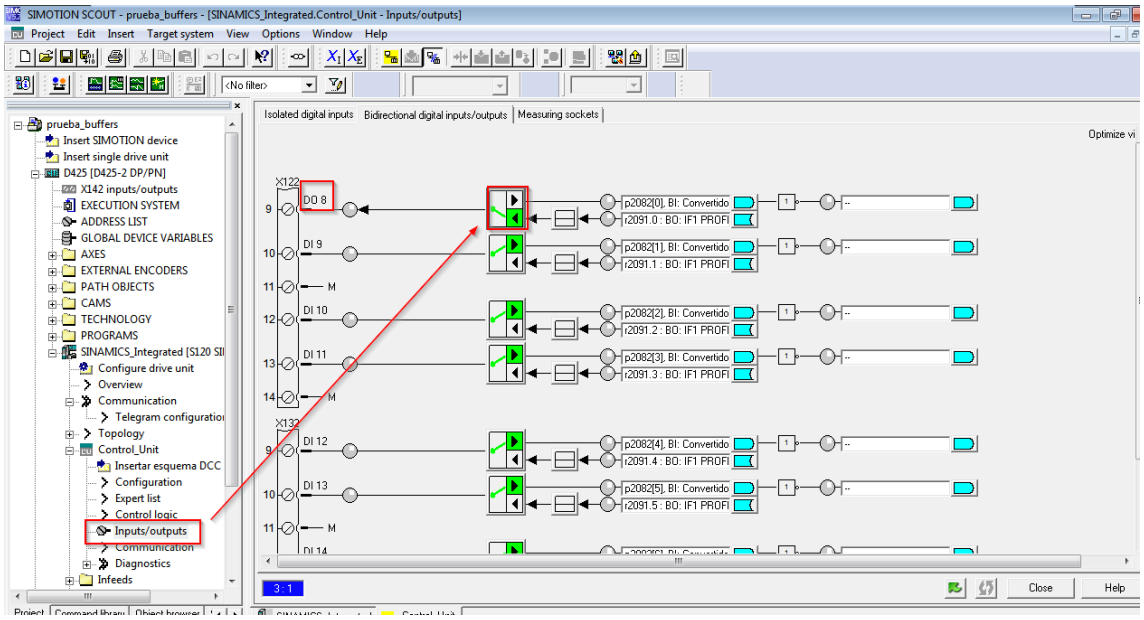


FIGURA 57. CONFIGURACIÓN PINES DE PUERTOS I/O

El siguiente paso es acceder a la información de los puertos. La información estará en la dirección del *telegram,* las capturas de pantalla son de un proyecto de ejemplo. En el proyecto principal la dirección será distinta debido a que además del telegram de I/O están los *telegrams* de control de motores.

Object	Drive object	-No.	Telegram type		Input data		Output data		Technology object
					Length	Address	Length	Address	
1	MotorX	2	SIEMENS telegram 105, PZD-10/10	✔	10	256..275	10	256..275	EjeX
2	MotorY	3	SIEMENS telegram 105, PZD-10/10	✔	10	276..295	10	276..295	EjeY
3	Control_Unit	1	SIEMENS telegram 390, PZD-1/1	✔	2	296..299	2	296..299	---
DOs that are not assigned to a slot. (No cyclic data exchange)									

FIGURA 58. DIRECCIONES TELEGRAMS DE TARJETA S120

La dirección a la que mandará la información será la 298. Por lo tanto, y según viene explicado en la tabla de conexiones, la entrada DI8 estará en la dirección PI 298.0 del *telegram*. PI para entrada y PQ para salida. Las variables se asignan desde la tabla *adress list* de la CPU D425 en el árbol de proyecto.

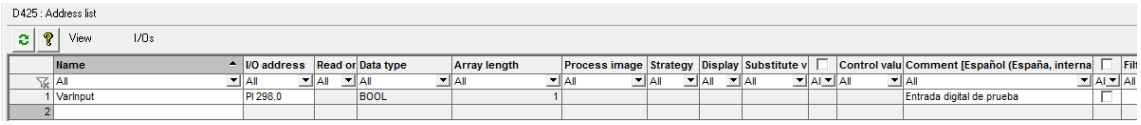


FIGURA 59. VARIABLE ASIGNADA A ENTRADA DIGITAL

Por último, hay que realizar la conexión de las entradas a los sensores. A efectos prácticos los sensores de fin de carrera se comportan como interruptores. Al llegar un eje al fin del recorrido un interruptor pasa a estar encendido, poniendo la

entrada digital a ON y la variable asociada a esta a TRUE. Se usarán los pines 1 a 4 del puerto X122, dejando los puertos configurables para entradas/salidas libres.

El esquema pertenece al manual de hardware de la CPU D425-2 la cual dispone de otro puerto más para I/O (el X142). Sin embargo, la distribución de pines para los puertos X132 y X122 es la misma. Un esquema detallado de la electrónica de entradas y salidas puede encontrarse en el anexo Esquemas.

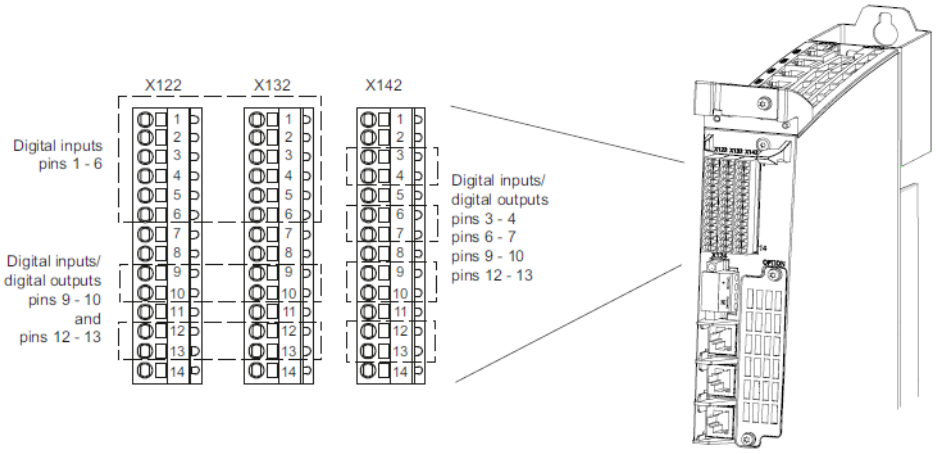


FIGURA 60. DISTRIBUCIÓN DE LOS PINES (*D4X5 OPERATION MANUAL P58)*

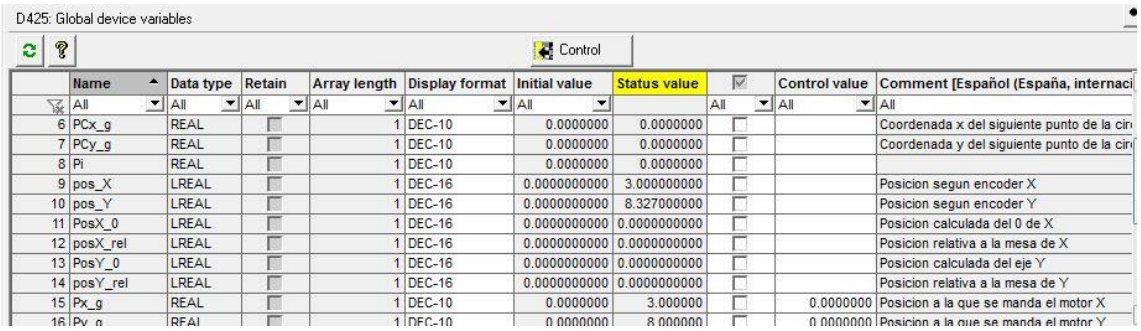
Monitorización y control de la CPU

Cuando la CPU esta en línea se puede controlar en todo momento desde la herramienta de ingeniería permite controlar el encendido y apagado de la maquina desde un panel de control virtual. De esta forma, no es necesario manipular físicamente la máquina para poder cargar proyectos o pararla en caso de emergencia:

FIGURA 61. PANEL VIRTUAL DE CONTROL DE LA CPU

Cuando la CPU está en RUN, se puede usar la lista de variables globales para controlarla desde el PC. Esta lista permite conocer el valor inicial de las variables, su valor actual y forzar un valor. Esto permite usar el PC como interfaz provisional entre el usuario y la CPU durante las fases de desarrollo. Sin embargo, esto no es práctico para la aplicación acabada, ya que obliga a disponer de una licencia (bastante onerosa) de la

herramienta de ingeniería, por lo que se suele diseñar una interfaz propia entre usuario y dispositivo.



	Name	Data type	Retain	Array length	Display format	Initial value	Status value		Control value	Comment [Español (España, internaci
	All	All	All	All	All	All		All	All	All
6	PCx_g	REAL		1	DEC-10	0.0000000	0.0000000			Coordenada x del siguiente punto de la cir
7	PCy_g	REAL		1	DEC-10	0.0000000	0.0000000			Coordenada y del siguiente punto de la cir
8	Pi	REAL		1	DEC-10	0.0000000	0.0000000			
9	pos_X	LREAL		1	DEC-16	0.0000000000	3.000000000			Posicion segun encoder X
10	pos_Y	LREAL		1	DEC-16	0.0000000000	8.327000000			Posicion segun encoder Y
11	PosX_0	LREAL		1	DEC-16	0.0000000000	0.0000000000			Posicion calculada del 0 de X
12	posX_rel	LREAL		1	DEC-16	0.0000000000	0.0000000000			Posicion relativa a la mesa de X
13	PosY_0	LREAL		1	DEC-16	0.0000000000	0.0000000000			Posicion calculada del eje Y
14	posY_rel	LREAL		1	DEC-16	0.0000000000	0.0000000000			Posicion relativa a la mesa de Y
15	Px_g	REAL		1	DEC-10	0.0000000	3.000000		0.0000000	Posicion a la que se manda el motor X
16	Py_g	REAL		1	DEC-10	0.0000000	8.000000		0.0000000	Posicion a la que se manda el motor Y

FIGURA 62. TABLA DE VARIABLES GLOBALES

Otra opción de monitorización interesante que ofrece es la posibilidad de controlar directamente ejes y drives mediante un panel de control virtual. Estos paneles permiten comprobar el funcionamiento de los motores y los ejes directamente sin necesidad de código, con lo que se pueden depurar más fácilmente los errores al ir acotando las fuentes de error. Por ejemplo, al comprobar que el motor gira correctamente al mandarle una orden desde el panel de control del eje se puede asegurar que no hay fallos en el hardware, que el drive está bien configurado y que el eje esta correctamente emparejado con el drive.

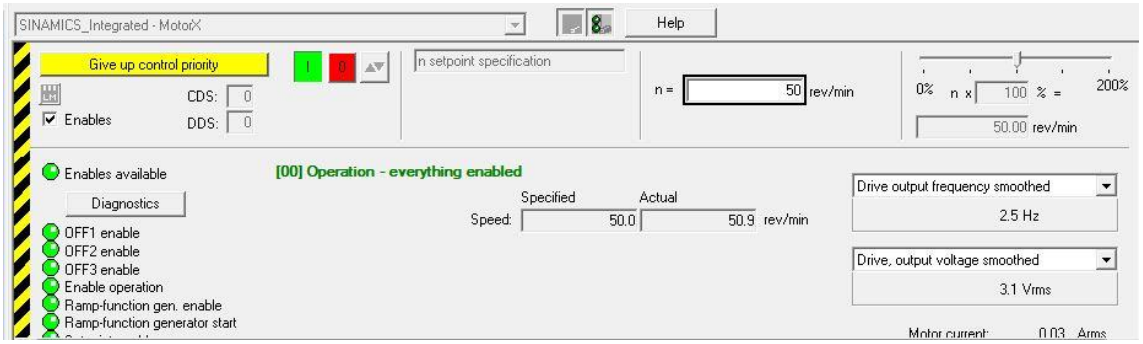


FIGURA 63. PANEL VIRTUAL DE CONTROL DEL MOTOR X

Ejemplos de Piezas

Pieza de Ejemplo 1-Escudo

Descripción Pieza

Es una pieza compuesta por rectas, por lo tanto, se puede realizar con la operación `TrazarLinea()`.También se emplea la función `MoverEjes()` para realizar el posicionamiento antes de empezar a trazar la figura.

Programa Matlab:

```
%Velocidad y periodo
v=2;
T=0.025;
%Puntos
P1=[25 5];
P2=[35 15];
P3=[35 35];
P4=[32 37];
P5=[16 37];
P6=[14 35];
P7=[14 15];
P8=[25 5];

%Simulacion de la mesa
LimSup=line([0 70],[70 70]);
LimInf=line([0 70],[0 0]);
LimIzq=line([0 0],[0 70]);
LimDer=line([70 70],[0 70]);

%Ajustes grafico:
axis([-5 75 -5 75]);
grid on;

TrazarLinea(P1,P2,v,T)
TrazarLinea(P2,P3,v,T)
TrazarLinea(P3,P4,v,T)
TrazarLinea(P4,P5,v,T)
TrazarLinea(P5,P6,v,T)
TrazarLinea(P6,P7,v,T)
TrazarLinea(P7,P8,v,T)
```

Simulación Matlab:

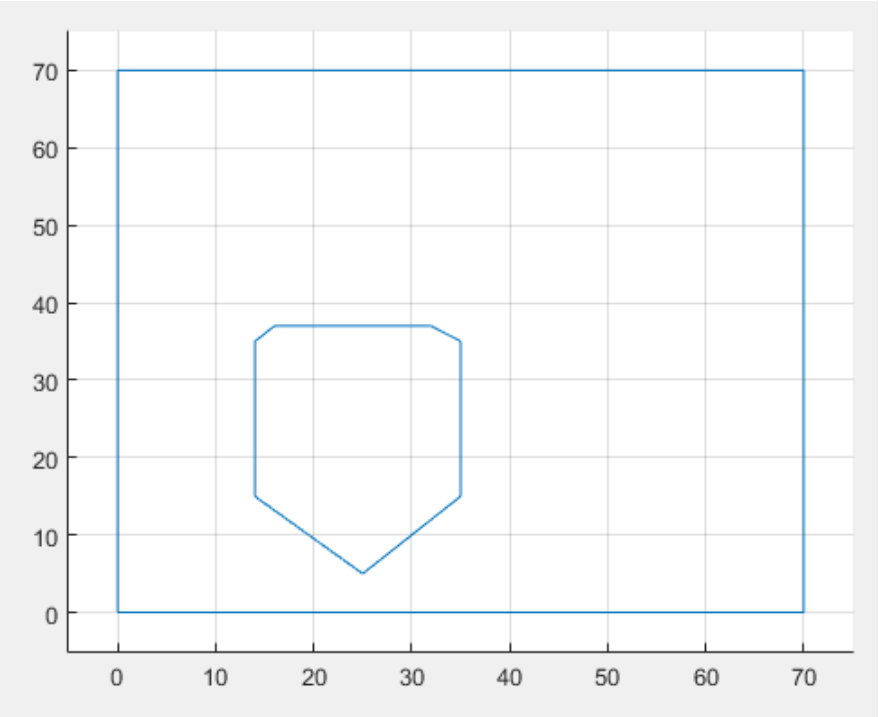


FIGURA 64. SIMULACIÓN PIEZA ESCUDO EN MATLAB

Programa

Para el programa que se cargará en la CPU se han escogido otros puntos para adaptar la figura a las dimensiones de la mesa. Como la figura es poligonal y no hay intersecciones se ha empleado unos disparadores guardados en memoria local. De esta forma se mantiene la estructura que tendrán programas con operaciones más avanzadas.

```
INTERFACE

    PROGRAM Escudo;

END_INTERFACE

IMPLEMENTATION
    USES Operacio;
    PROGRAM Cuadrado
        VAR
            F_TrazarLinea:FB_TrazarLinea;
           // F_Pos:FB_Pos;
            P1,P2,P3,P4:ARRAY[0..1] OF REAL;
            D1:BOOL:=TRUE;
            D2,D3,D4,D5:BOOL:=FALSE;
        END_VAR
        P1[0]:=0;P3[0]:=15.0;
        P1[1]:=0;P3[1]:=15.0;
        P2[0]:=15.0;P4[0]:=0;
        P2[1]:=0;P4[1]:=15.0;
        (*
        IF disp0=TRUE THEN
            D1:=TRUE;
```

```
            disp0:=FALSE;
        END_IF;
        *)
        IF D1=TRUE THEN
            F_TrazarLinea( PIn:=P1,PFin:=P2);
        END_IF;
        IF P2[0]=ejeX.basicmotion.position AND
P2[1]=ejeY.basicMotion.position THEN
            D1:=FALSE;// desactiva el disparador de la operacion que ha
acabado
            D2:=TRUE;//activa el disparador de la proxima operacion
        END_IF;
        IF D2=TRUE THEN
            F_TrazarLinea( PIn:=P2,PFin:=P3);
        END_IF;
        IF P3[0]=ejeX.basicmotion.position AND
P3[1]=ejeY.basicMotion.position THEN
            D2:=FALSE;// desactiva el disparador de la operacion que ha
acabado
            D3:=TRUE;//activa el disparador de la proxima operacion
        END_IF;
        IF D3=TRUE THEN
            F_TrazarLinea( PIn:=P3,PFin:=P4);
        END_IF;
        IF P4[0]=ejeX.basicmotion.position AND
P4[1]=ejeY.basicMotion.position THEN
            D3:=FALSE;// desactiva el disparador de la operacion que ha
acabado
            D4:=TRUE;//activa el disparador de la proxima operacion
        END_IF;
        IF D4=TRUE THEN
            F_TrazarLinea( PIn:=P4,PFin:=P1);
        END_IF;
      (*  IF P1[0]=ejeX.basicmotion.position AND
P1[1]=ejeY.basicMotion.position AND THEN
            D4:=FALSE;// desactiva el disparador de la operacion que ha
acabado
            D1:=TRUE;//activa el disparador de la proxima operacion
        END_IF;*)
        //PosicionReal
        pos_X:=ejeX.basicmotion.position;
        pos_Y:=ejeY.basicMotion.position;
    END_PROGRAM
    PROGRAM Escudo
        VAR
            F_TrazarLinea:FB_TrazarLinea;
           // F_Pos:FB_Pos;
            P1,P2,P3,P4,P5,P6,P7,P8:ARRAY[0..1] OF REAL;
            D1,D2,D3,D4,D5,D6,D7,D8,d9:BOOL:=FALSE;
        END_VAR
        P1[0]:=0;P3[0]:=7;
        P1[1]:=0;P3[1]:=8;
        P2[0]:=5;P4[0]:=7;
        P2[1]:=5;P4[1]:=12;
        P5[0]:=6;P6[0]:=4;
        P5[1]:=13;P6[1]:=13;
        P7[0]:=3;P8[0]:=3;
        P7[1]:=12;P8[1]:=8;
        IF disp0=TRUE THEN
```

```
            D1:=TRUE;
            disp0:=FALSE;
        END_IF;

        //Op1
        IF D1=TRUE THEN
            F_TrazarLinea( PIn:=P1,PFin:=P2);
        END_IF;
        IF P2[0]=ejeX.basicmotion.position AND
P2[1]=ejeY.basicMotion.position THEN
            D1:=FALSE;// desactiva el disparador de la operacion que ha
acabado
            D2:=TRUE;//activa el disparador de la proxima operacion
        END_IF;

        //Op2
        IF D2=TRUE THEN
            F_TrazarLinea( PIn:=P2,PFin:=P3);
        END_IF;
        IF P3[0]=ejeX.basicmotion.position AND
P3[1]=ejeY.basicMotion.position THEN
            D2:=FALSE;// desactiva el disparador de la operacion que ha
acabado
            D3:=TRUE;//activa el disparador de la proxima operacion
        END_IF;

        //Op3
        IF D3=TRUE THEN
            F_TrazarLinea( PIn:=P3,PFin:=P4);
        END_IF;
        IF P4[0]=ejeX.basicmotion.position AND
P4[1]=ejeY.basicMotion.position THEN
            D3:=FALSE;// desactiva el disparador de la operacion que ha
acabado
            D4:=TRUE;//activa el disparador de la proxima operacion
        END_IF;

        //Op4
        IF D4=TRUE THEN
            F_TrazarLinea( PIn:=P4,PFin:=P5);
        END_IF;
        IF P5[0]=ejeX.basicmotion.position AND
P5[1]=ejeY.basicMotion.position  THEN
            D4:=FALSE;// desactiva el disparador de la operacion que ha
acabado
            D5:=TRUE;//activa el disparador de la proxima operacion
        END_IF;

        //Op5
        IF D5=TRUE THEN
            F_TrazarLinea( PIn:=P5,PFin:=P6);
        END_IF;
        IF P6[0]=ejeX.basicmotion.position AND
P6[1]=ejeY.basicMotion.position THEN
            D5:=FALSE;// desactiva el disparador de la operacion que ha
acabado
            D6:=TRUE;//activa el disparador de la proxima operacion
        END_IF;
```

```
        //Op6
        IF D6=TRUE THEN
            F_TrazarLinea( PIn:=P6,PFin:=P7);
        END_IF;
        IF P7[0]=ejeX.basicmotion.position AND
P7[1]=ejeY.basicMotion.position THEN
            D6:=FALSE;// desactiva el disparador de la operacion que ha
acabado
            D7:=TRUE;//activa el disparador de la proxima operacion
        END_IF;

        //Op7
        IF D7=TRUE THEN
            F_TrazarLinea( PIn:=P7,PFin:=P8);
        END_IF;
        IF P8[0]=ejeX.basicmotion.position AND
P8[1]=ejeY.basicMotion.position THEN
            D7:=FALSE;// desactiva el disparador de la operacion que ha
acabado
            D8:=TRUE;//activa el disparador de la proxima operacion
        END_IF;

        //Op8
        IF D8=TRUE THEN
            F_TrazarLinea( PIn:=P8,PFin:=P2);
        END_IF;
        IF P2[0]=ejeX.basicmotion.position AND
P2[1]=ejeY.basicMotion.position THEN
            D8:=FALSE;// desactiva el disparador de la operacion que ha
acabado
            D9:=TRUE;//activa el disparador de la proxima operacion
        END_IF;



        //PosicionReal
        pos_X:=ejeX.basicmotion.position;
        pos_Y:=ejeY.basicMotion.position;
    END_PROGRAM

END_IMPLEMENTATION
```

Pieza de Ejemplo 2- Rectángulo Achaflanado

Descripción Pieza

Esta pieza combina la función `TrazarLinea()` con la función `TrazarCirculo()` para obtener un rectángulo con las esquinas achaflanadas gracias a las tangencias formadas por los arcos de circunferencia.

Programa Matlab:

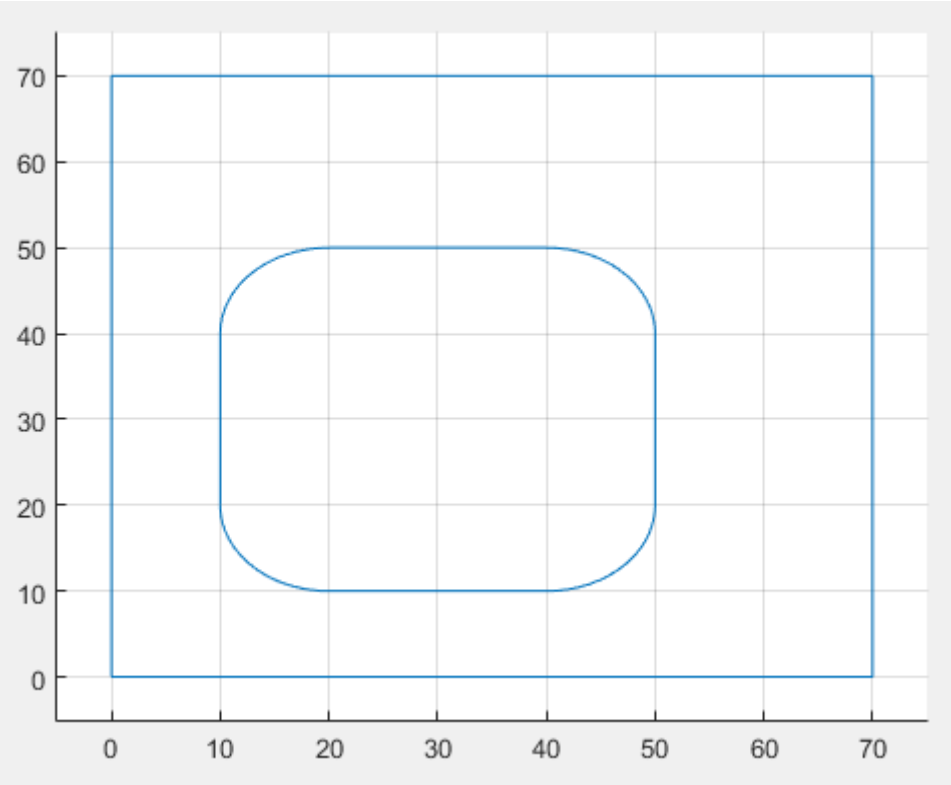
```
%Velocidad y periodo
v=8;
T=0.025;
%Zona de trabajo
LimSup=line([0 70],[70 70]);
```

```
LimInf=line([0 70],[0 0]);
LimIzq=line([0 0],[0 70]);
LimDer=line([70 70],[0 70]);

%Ajustes grafico:
axis([-5 75 -5 75]);
grid on;

%ContornoExterior
TrazarLinea([20 10],[40 10],v,T);
TrazarArco2([40 20],[40 10],pi/2,v,T);
TrazarLinea([50 20],[50 40],v,T);
TrazarArco2([40 40],[50 40],pi/2,v,T);
TrazarLinea([40 50],[20 50],v,T);
TrazarArco2([20 40],[20 50],pi/2,v,T);
TrazarLinea([10 40],[10 20],v,T);
TrazarArco2([20 20],[10 20],pi/2,v,T);
```

Simulación Matlab



Programa

```
PROGRAM RectanguloAchaflanado
        VAR
            F_ClcCirculo:FB_ClcCirculo;
            F_TrazarLinea:FB_TrazarLinea;
            P1,P2,P3,P4:ARRAY[0..1] OF  REAL; //puntos del diseño
            PC.Pin,Pant:ARRAY[0..1] OF  REAL; //puntos auxiliares para
```

```
circunferencias
        END_VAR
        //Inicio de secuencia con variable global
        IF disp0=TRUE THEN
            D1:=TRUE;
            disp0:=FALSE;
        END_IF;

        //Op1---->Posicionar ejes
        IF Disp[0]=TRUE THEN
            P1[0]:=0;
            P1[1]:=0;
            P2[0]:=20;
            P2[1]:=10;
            F_TrazarLinea( PIn:=P1,PFin:=P3);
            IF P2[0]=ejeX.basicmotion.position AND
P2[1]=ejeY.basicMotion.position THEN
                Disp[0]:=FALSE;// desactiva el disparador de la
operacion que ha acabado
                Disp[1]:=TRUE;//activa el disparador de la proxima
operación
            END_IF;
        END_IF;

        //Operacion2----->Lado inferior
        IF Disp[1]=TRUE THEN
            P1[0]:=20;
            P1[1]:=10;
            P2[0]:=40;
            P2[1]:=10;
            F_TrazarLinea( PIn:=P1,PFin:=P3);
            IF P2[0]=ejeX.basicmotion.position AND
P2[1]=ejeY.basicMotion.position THEN
                Disp[1]:=FALSE;// desactiva el disparador de la
operacion que ha acabado
                Disp[2]:=TRUE;//activa el disparador de la proxima
operación
                //Se aprobecha este if para iniciar el ciclo de la
circunferencia
                Pc[0]:=40;
                Pc[1]:=20;
                PIn[0]:=50;
                PIn[1]:=20;
                F_ClcCirculo(O:=PC,P1:=PIn,Tita:=Pi/2);
                P[0]:=PCx_g;
                P[1]:=PCy_g;
                PCxant_g:=P2[0];
                PCyant_g:=P2[1];
                Pant[0]:=P2[0];
                Pant[1]:=P3[1];
                F_TrazarLinea( PIn:=Pant,PFin:=P);
```

```
            END_IF;
        END_IF;


        //Circunferencia inferior derecha
        IF Disp[2]=TRUE THEN
            //Algoritmo de trazado
            IF 0.005>=ABS(PCx_g-ejeX.basicmotion.position) AND
0.005>=ABS(PCy_g-ejeY.basicmotion.position) THEN
                Pc[0]:=40;
                Pc[1]:=20;
                PIn[0]:=50;
                PIn[1]:=20;
                PCxant_g:=PCx_g;
                PCyant_g:=PCy_g;
                F_ClcCirculo(O:=PC,P1:=PIn,Tita:=Pi/2);
                Pant[0]:=PCxant_g;
                Pant[1]:=PCyant_g;
                P[0]:=PCx_g;
                P[1]:=PCy_g;
                F_TrazarLinea( PIn:=Pant,PFin:=P);
                PosX_0:=PosX_0+1;
            END_IF;
            //Punto fin circunferencia
            P[0]:=50;
            P[1]:=20;
           //Criterio de Parada
            IF 0.1>=ABS(in:= P[0]-ejeX.basicmotion.position) AND
0.1>=ABS(in:=P[1]-ejeY.basicmotion.position) THEN
                Disp[2]:=FALSE;// desactiva el disparador de la
operacion que ha acabado
                cont:=0; //pone el contador de pasos de la
circunferencia a 0
                Disp[3]:=TRUE;//activa el disparador de la proxima
operacion
            END_IF;
        END_IF;

        //Operacion2----->Lado derecho
        IF Disp[3]=TRUE THEN
            P1[0]:=50;
            P1[1]:=20;
            P2[0]:=50;
            P2[1]:=40;
            F_TrazarLinea( PIn:=P1,PFin:=P3);
            IF P2[0]=ejeX.basicmotion.position AND
P2[1]=ejeY.basicMotion.position THEN
                Disp[3]:=FALSE;// desactiva el disparador de la
operacion que ha acabado
                Disp[4]:=TRUE;//activa el disparador de la proxima
operación
```

```
                //Se aprobecha este if para iniciar el ciclo de la
circunferencia
                PC[0]:=40;
                PC[1]:=40;
                Pin[0]:=50;
                Pin[1]:=40;
                F_ClcCirculo(O:=PC,P1:=PIn,Tita:=Pi/2);
                P[0]:=PCx_g;
                P[1]:=PCy_g;
                PCxant_g:=PIn[0];
                PCyant_g:=PIn[1];
                F_TrazarLinea( PIn:=PIn,PFin:=P);
            END_IF;
        END_IF;

        //Circunferencia superior derecha
        IF Disp[2]=TRUE THEN
            //Algoritmo de trazado
            IF 0.005>=ABS(PCx_g-ejeX.basicmotion.position) AND
0.005>=ABS(PCy_g-ejeY.basicmotion.position) THEN
                PC[0]:=40;
                PC[1]:=40;
                Pin[0]:=50;
                Pin[1]:=40;
                PCxant_g:=PCx_g;
                PCyant_g:=PCy_g;
                F_ClcCirculo(O:=PC,P1:=PIn,Tita:=Pi/2);
                Pant[0]:=PCxant_g;
                Pant[1]:=PCyant_g;
                P[0]:=PCx_g;
                P[1]:=PCy_g;
                F_TrazarLinea( PIn:=Pant,PFin:=P);
                PosX_0:=PosX_0+1;
            END_IF;
            //Punto fin circunferencia
            P[0]:=40;
            P[1]:=40;
           //Criterio de Parada
            IF 0.1>=ABS(in:= P[0]-ejeX.basicmotion.position) AND
0.1>=ABS(in:=P[1]-ejeY.basicmotion.position) THEN
                Disp[3]:=FALSE;// desactiva el disparador de la
operacion que ha acabado
                cont:=0; //pone el contador de pasos de la
circunferencia a 0
                Disp[4]:=TRUE;//activa el disparador de la proxima
operacion
            END_IF;
        END_IF;

       //*************El resto de operaciones seran iguales a las
precedentes************ //
```

```
    //PosicionReal
        pos_X:=ejeX.basicmotion.position;
        pos_Y:=ejeY.basicMotion.position;
END_PROGRAM
```

Metodología seguida en el desarrollo del Trabajo

En este apartado se va a describir la planificación que se ha realizado del proyecto de TFG. Primero, se describen en el primer apartado las tareas y subtareas en las que se ha dividido el proyecto, así como las razones por las que se ha escogido dicha distribución. A continuación, para representarlo de forma más visual, se ha empleado un diagrama de Gantt. Por último, se hace un resumen de los objetivos a lograr.

Planificación

Se ha hecho la planificación de manera que las tareas se van sucediendo de forma secuencial. Esto es principalmente debido a que estando una sola persona trabajando en el proyecto, es más productivo dedicarse por completo a una sola tarea para no perder tiempo volviendo a repasar conceptos. Además, se trata de un proyecto que por naturaleza esta dado a este tipo de planificación puesto que para hacer una de las tareas deben de estar acabadas todas las anteriores, así, por ejemplo, no se puede empezar a programar la consigna de polinomio sin haber realizado antes la función de calculo de consigna para trazar una línea.

	Nombre de la tarea	Duración	Inicio	Finalizar
1	Planteamiento del Proyecto	1d	04/07/18	04/07/18
2	Diseño de Consignas	8d	05/07/18	13/07/18
3	Consigna Recta	2d	05/07/18	06/07/18
4	Consigna Circunferencia	2d	07/07/18	09/07/18
5	Consigna Polinomio	3d	10/07/18	12/07/18
6	Prueba en Simulación (Matlab)	1d	13/07/18	13/07/18
7	Puesta en marcha de la CPU	9d	14/07/18	24/07/18
8	Configuración de proyecto	3d	14/07/18	17/07/18
9	Gestión de SO	2d	18/07/18	19/07/18
10	Control de Motores	2d	20/07/18	21/07/18
11	Adquisición de Información de Encoders	1d	23/07/18	23/07/18
12	Control de entradas I/O	1d	24/07/18	24/07/18
13	Implementación de operaciones en la CPU	10d	25/07/18	06/09/18
14	Implementación de algoritmo de Recta	2d	25/07/18	26/07/18
15	Implementación de algoritmo de Circunferencia	3d	27/07/18	30/07/18
16	Implementación de algoritmo de Polinomio	3d	31/07/18	04/09/18
17	Diseño de Programas Tipo para piezas	2d	05/09/18	06/09/18
18	Montaje Mesa	1d	07/09/18	07/09/18
19	Sistema de Sensores	6d	08/09/18	14/09/18
20	Programa de Centrado	2d	08/09/18	10/09/18
21	Programa Anti-Impactos	1d	11/09/18	11/09/18
22	Pruebas sensores (simulación)	2d	12/09/18	13/09/18
23	Implementación conjunto en la CPU	1d	14/09/18	14/09/18

TABLA 3. LISTADO DE TAREAS

Planteamiento del Proyecto:

Se plantea el alcance y los objetivos que se van a conseguir y la metodología a emplear para lograrlos. Es una parte importante, ya que se va a decidir el tipo de solución que se va a dar al problema y las distintas tareas que va a ser necesario realizar para conseguirlo.

Diseño de consignas

Es la primera parte del proyecto. Consiste en el diseño y prueba en Matlab de los algoritmos que van a generar las consignas para guiar a los motores. Consiste en una parte teórica y en su aplicación posterior en Matlab con su propio lenguaje de programación. Habrá que desarrollar 3 algoritmos:

- Algoritmo de generación de consigna de recta.
- Algoritmo de generación de consigna de circunferencia
- Algoritmo de generación de consigna de polinomio.
- Simulación en Matlab.

Puesta en marcha de la CPU

En esta etapa, el objetivo es familiarizarse con los rudimentos básicos de la CPU. Se empieza con programas sencillos que permiten utilizar las funciones que más tarde serán empleadas para implementar los algoritmos diseñados en la etapa anterior. A esto hay que añadir el control de los sistemas físicos que no aparecen en la simulación (motores, I/O y encoders). Se pueden diferenciar las siguientes sub-etapas:

- Configuración de proyecto
- Gestión de sistema operativo
- Control de motores
- Adquisición de información de encoders
- Control de entradas I/O

Implementación de operaciones en la CPU

Esta etapa consiste en la “traducción” de los algoritmos de Matlab a lenguaje ST. Después hay que organizar los programas de tal forma que la maquina vaya completando las operaciones de forma sucesiva. Se pueden distinguir las siguientes subetapas:

- Implementación de algoritmo recta
- Implementación de algoritmo circunferencia
- Implementación de algoritmo polinomio
- Diseño de programa tipo para piezas

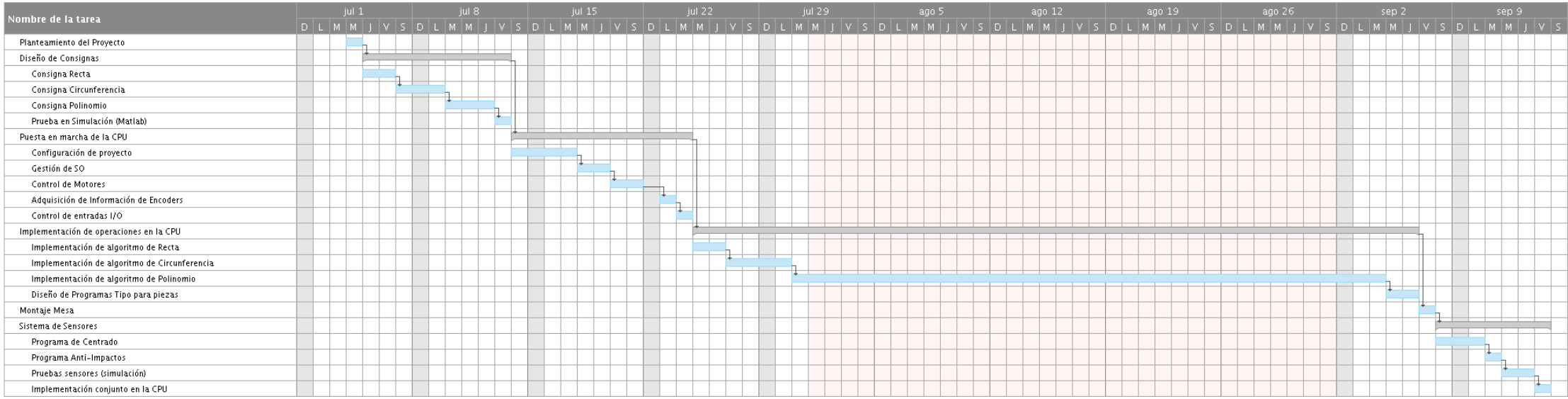
Montaje de la mesa

Sistema de sensores

Es la parte central ya que mientras se ejecuta el programa pieza, existe otro programa que hace las veces de sistema operativo de la CPU. Su misión es centrar la mesa cada vez que empieza a funcionar e iniciar las piezas asegurándose de que no haya choques. Mientras se diseña este programa hay que ir probando los distintos sensores (las variables que guardarán la información de los sensores) con los motores en vacío. Se pueden distinguir las siguientes subetapas:

- Programa de centrado
- Sistema anti-impactos
- Pruebas de sensores (simulación)
- Implementación de lo anterior en la CPU

Diagrama de Gantt



En el diagrama de Gantt se han representado las tareas principales, así como las subtareas en las que están divididas. Como se puede apreciar, únicamente hay un camino, debido a que las tareas se han realizado de forma secuencial. El mes de agosto la escuela se encuentra cerrada, por lo que no se pudieron continuar los progresos durante este mes.

Descripción de los Resultados

Simulación:

Se ha conseguido simular en Matlab las distintas operaciones que pueden realizar con la mesa. Esta simulación es una herramienta muy útil y, gracias a la experiencia y el código obtenido en este proyecto se pueden modificar para poder simular otras operaciones que pudiese realizar la máquina, es por lo tanto una aportación muy útil para futuros desarrollos.

Control de la CPU:

En el desarrollo del proyecto se ha conseguido dominar los aspectos fundamentales de la programación de este tipo de máquinas y las principales funcionalidades de las que disponen. Esto puede ser de gran utilidad en futuros proyectos. Estas funcionalidades son, principalmente, posicionamiento de ejes (la CPU puede controlar hasta 128), cálculos para interpolar ejes y funciones auxiliares con las entradas y salidas digitales (sensores, botoneras y accionamientos sencillos).

Consignas para la generación de geometrías:

El código desarrollado y probado en este proyecto se puede utilizar mas adelante en cualquier otra CPU si fuese necesario. Ya que el código es independiente de los lazos de control se pueden sustituir las funciones de lectura de encoder y control de motores manteniendo exactamente igual el resto de funciones.

Aspectos económicos

En este apartado se detalla el presupuesto del proyecto realizado. Se tendrán en cuenta el precio de todos los elementos que se han utilizado en este proyecto. Todo el material que se ha empleado en este proyecto se va a utilizar en otros ya que no hay ningún consumible ni elemento solamente se puede utilizar una sola vez. Para hacer el cálculo de la vida útil, no se ha considerado el tiempo de desgaste, ya que, en principio la electrónica de control (CPU, driver y fuentes de alimentación) pueden durar casi indefinidamente si están en buenas condiciones de trabajo, sin embargo, ya que son unos equipos que se emplean para propósitos formativos, se calculará su vida útil en función del momento en el que pasan a estar obsoleto. Se va a considerar una vida útil de 20 años según este criterio. En cuanto a los motores, aunque el uso poco intensivo que se les va a dar también les haría durar mucho tiempo, hay que recordar que disponen de una electrónica bastante sofisticada que emplea un protocolo muy concreto para comunicarse con el resto de elementos, por lo tanto, tendrá la misma vida útil limitada por su vida útil. El último elemento a considerar son los ejes de la mesa, al igual que los motores, no van a sufrir un uso que les vaya a producir un gran desgaste. En cuanto a accidentes, la parte más débil (las correas) puede ser fácilmente sustituible y no tiene ningún tipo de electrónica. Por lo tanto, se puede esperar que tengan una vida bastante larga (de unos 30 años) en la escuela. El último elemento a considerar sería el ordenador, con una vida útil de 5 años.

Amortizaciones	Coste unitario (€)	Horas utilizadas	Vida útil	Coste Total (€)
Simotion D425	6000	272	38400	42,5
Smart Line Module	6000	272	38400	42,5
Double Motor Module	12000	272	38400	85
Ordenador	900	272	9600	25,5
SITOP	150	272	34800	1,172413793
Software	3000	272	34800	23,44827586
Mesa de 2 ejes	899	272	57600	4,245277778
			Total	224,3659674

TABLA 4. AMORTIZACIONES

El resultado es que el proyecto total ha tenido un coste de 224.36€. Los cálculos de vida útil se han hecho considerando 240 días hábiles de media por año a 8h por jornada.

Análisis de Riesgos

En este apartado se van a plantear los riesgos a los que se enfrenta este proyecto durante su realización. Después se va a estudiar la importancia de dichos riesgos empleando el método de la matriz de probabilidad-impacto. Esto va a permitir clasificar los distintos riesgos según su importancia para centrar los esfuerzos preventivos en los más graves, asegurando de esta forma el uso óptimo de los recursos de prevención.

El primer gran grupo de riesgos son los relacionados con la rotura de alguna parte del equipo durante las pruebas. Debido a que se trata de material muy especializado tienen un precio muy alto y además son difíciles de encontrar, por lo tanto, si deben de sustituirse, los plazos de entrega son bastante largos. Los principales elementos que pueden romperse son los ejes, la electrónica de potencia que controla los motores, los devanados de los motores y la electrónica de control de los pines I/O de la CPU. El factor de probabilidad de estos riesgos variará en función de lo resistentes que sean (debido tanto a su fortaleza intrínseca como a las medidas de protección que implementen de fábrica. En cuanto al factor impacto, variará en función de lo caro que resulte sustituir o reparar el equipo, así como del tiempo que tarden los repuestos en ser suministrados.

El segundo grupo esta relacionado con la dificultad de los problemas a resolver. Una dificultad muy grande a la hora de programar o configurar la máquina podría suponer un retraso o, en el peor de los casos, la imposibilidad de concluir el proyecto. Dentro de los problemas a resolver estarían los 3 algoritmos de obtención de consigna (tanto en Matlab como en Simotion Scout) así como el hacerlos trabajar de forma sucesiva para obtener figuras. En cuanto a los problemas a la hora de configurar la máquina, podemos considerar la organización de programas (control de sistema operativo), la configuración de entradas digitales y el control de los motores.

Con las principales fuentes de fallo analizada de forma cualitativa, falta por hacer el análisis cuantitativo para determinar la importancia relativa de cada riesgo. Se emplearán 3 tablas para realizar este análisis.

En primer lugar, está la tabla de ponderación de frecuencia, la cual asigna el factor de frecuencia a cada riesgo. Relaciona el análisis cualitativo de frecuencia con el valor cuantitativo del factor de frecuencia.

Ponderación	Probabilidad
1	Improbable
2	Remota
3	Elevada
4	Segura

TABLA 5- TABLA DE PONDERACIÓN-FRECUENCIA

En segundo lugar, está la tabla de ponderación impacto, la cual asigna un factor de impacto a cada riesgo. Relaciona del análisis cualitativo de impacto con el valor cuantitativo del factor de impacto.

Ponderación	Impacto
1	Insignificante
2	Leve
3	Grave
4	Fatal

TABLA 6. TABLA PONDERACIÓN-IMPACTO

Por último, se emplea la matriz de Probabilidad-impacto para calcular la importancia relativa de cada riesgo en función de los valores obtenidos en las dos tablas anteriores. El valor de importancia relativa será igual al producto de ambos factores.

Probabilidad	Impacto			
	Insignificante 1	Leve 2	Grave 3	Fatal 4
Improbable 1	1	2	3	4
Remota 2	2	4	6	8
Elevada 3	3	6	9	12
Segura 4	4	8	12	16

TABLA 7. MATRIZ DE PROBABILIDAD-IMPACTO

En función de la valoración obtenida para cada riesgo, serán clasificados en 3 rangos de importancia:

- **Peligros significativos**: son peligros que tienen tanto una incidencia como una frecuencia muy elevadas (en rojo en la matriz). Por lo tanto, es en estos en los que hay que centrar los esfuerzos preventivos.
- **Peligros no significativos**: son aquellos que o bien tienen un gran impacto, pero son muy poco frecuentes, o son muy frecuentes, pero tienen un impacto muy

reducido (en amarillo en la matriz). Se les dedicarán ciertos esfuerzos preventivos.

- **Peligros despreciables**: son peligros que tienen un impacto muy bajo y además son muy poco frecuentes (en verde en la matriz). Como es muy poco probable que afecten al proyecto, se les destinarán un mínimo de esfuerzos preventivos.

A continuación, se listarán y evaluarán los peligros citados anteriormente empleando el método descrito. También se hará una breve descripción de las medidas preventivas tomadas en cada caso.

- Rotura de ejes: si al motor se le manda una consigna de avance tal que el avance del eje implique que se va a salir de sus guías se rompería la correa que mueve el eje y, además supondría una sobrecarga del motor. La posibilidad de que esto ocurra es bastante alta, debido a que la consigna y los sistemas de control están hechos por el usuario y los motores no tienen una referencia absoluta (solo cuentas vueltas). Por otro lado, para el eje, supondría una rotura de la correa, que puede ser fácilmente sustituida a un precio reducido, por lo que el impacto sería relativamente bajo.

Pond. Frecuencia	elevada	3
Pond. Impacto	leve	2
Valoración	No significativo	6

TABLA 8. VALORACIÓN ROTURAS DE EJE

- Rotura de electrónica de control: se refiere a la rotura por sobrecarga de la CPU, su tarjeta integrada, alguna de las fuentes de alimentación o el driver de los motores. El impacto sería muy alto, ya que son elementos muy caros y difícilmente remplazables. Sin embargo, al tratarse de equipos industriales muy sofisticados, disponen de abundantes sistemas de protección para evitar que una sobrecarga accidental pueda dañarlos. Por lo tanto, con realizar un correcto cableado y emplear las medidas de seguridad habituales a cualquier instalación eléctrica es suficiente como medida de prevención.

Pond. Frecuencia	elevada	1
Pond. Impacto	grave	4
Valoración	Peligro despreciable	4

TABLA 9. VALORACIÓN FALLO EN ELECTRÓNICA

- Rotura puertas I/O: aunque las puertas, al igual que otros sistemas electrónicos cuentas con protección, el sistema eléctrico que las va a activar va a ser diseñado por el usuario. Por lo tanto, será mas posible que se produzca un fallo. Por otro lado, forman un módulo independiente de la CPU y su

electrónica no es tan compleja, por lo que, aunque sería difícil, podrían ser reparadas.

Pond. Frecuencia	elevada	3
Pond. Impacto	leve	3
Valoración	No significativo	9

TABLA 10. VALORACIÓN ROTURA PUERTAS I/O

- Sobrecarga devanados de motores: ante cargas importantes, si a un motor se le pide mas par del que esta preparado para soportar podría sobrecalentarse y fundir los devanados. Desde luego tendría un impacto grave, y mientras traajen en vacío no hay problema. Sin embargo, como se ha explicado en el caso de los ejes, si se fuerza el eje, e vez de romperse la correa podría sobrecargarse el motor y ya se ha comentado que la posibilidad de que esto ocurra es muy alta. Por lo tanto es un peligro que hay que tener muy presente y habrá que probar a conciencia el sistema de seguridad que evite que los ejes se salgan de rango.

Pond. Frecuencia	elevada	3
Pond. Impacto	fatal	4
Valoración	significativo	12

TABLA 11. VALORACIÓN SOBRECARGA MOTORES

- Algoritmos consigna: estos algoritmos se diseñan en Matlab y pueden ser probados en la simulación que realiza este programa. Ya que se trata de una herramienta que se ha usado ampliamente durante el grado y los conocimientos de algebra y métodos numéricos, por lo tanto, es de esperar que todos los problemas que surjan se pueden resolver sin problemas.

Pond. Probabilidad	Improbable	1
Pond. Impacto	Fatal	4
Valoración	despreciable	4

TABLA 12. VALORACIÓN PROBLEMA AL OBTENER ALGORITMOS DE CONSIGNA

- Programación Scout: ser capaz de trasformar las consignas en Matlab en un programa coherente que las realice de forma sucesiva al ser cargado en la cpu es un gran reto, debido principalmente a que el personal del departamento no dispone de experiencia en este modelo. Por lo tanto, si una dificultad o problema de lógica aparece puede retrasar enormemente el proyecto. Para evitarlo, se cuenta con un contacto en el departamento de asistencia técnica de siemens para que solucione alguna situación desesperada.

Pond. Probabilidad	Segura	4
Pond. Impacto	Grave	3
Valoración	significativo	12

TABLA 13. VALORACIÓN DIFICULTAD IRRESOLUBLE EN PROGRAMACIÓN SCOUT

- Control Motores: en principio, al ser el control de motores algo muy habitual en esta clase de máquinas, debería de existir suficiente información para poder realizarlo sin problemas. Aún asi, en caso de que no se pudiera realizar, sería fatal para el proyecto.

Pond. Probabilidad	Improbable	1
Pond. Impacto	Fatal	4
Valoración	despreciable	1

TABLA 14. DIFICULTADES EN EL CONTROL DE MOTORES

- Control puertas I/O: el control de entradas y salidas digitales es de vital importancia para el proyecto, además, tienen la dificultad añadida de que además de la parte de programación hay que realizar el cableado con los sensores, por lo tanto, es una parte del proyecto que habrá que vigilar.

Pond. Probabilidad	elevada	3
Pond. Impacto	Grave	3
Valoración	no significativo	12

TABLA 15. DIFICULTADES CONTROL DE PUERTAS I/O

Conclusiones

Aunque, debido a la falta de material no se ha podido completar las pruebas finales y tener la mesa en funcionamiento si que se han conseguido completar todas las etapas de diseño previo. Por lo tanto, solo faltaría con realizar el montaje para tener una mesa completamente operativa capaz de realizar las distintas geometrías que se han descrito a lo largo del proyecto.

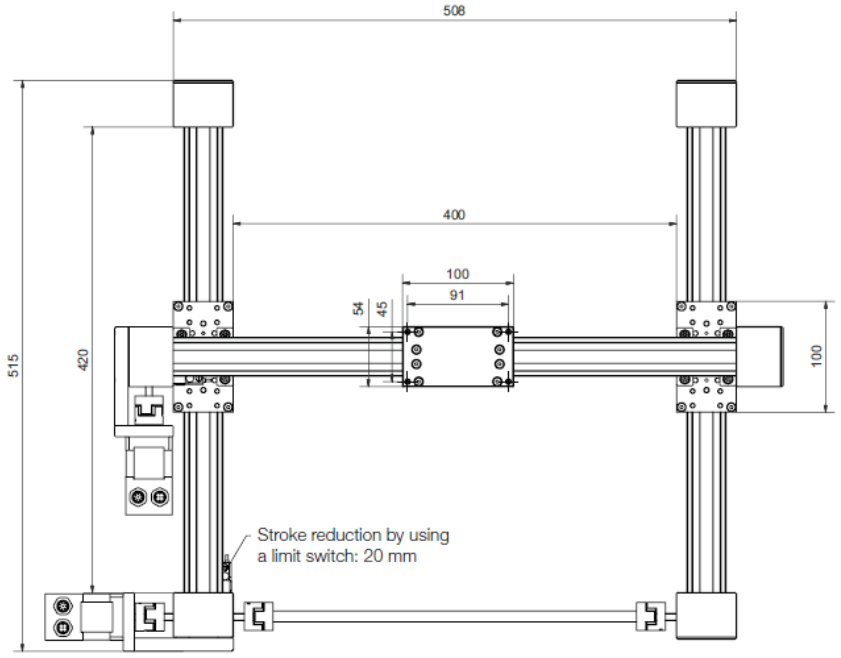
Además, se ha obtenido un conocimiento muy importante en la programación y el funcionamiento de la CPU. Esto le será muy útil al Departamento para poder ofrecer a los estudiantes otro tipo de máquina con la que trabajar. Lo mismo puede decirse de la mesa con 2 ejes que puede ser utilizada con esta CPU o con alguno de los otros PLCs que tiene el Departamento.

Por último, en cuanto a la programación y la estructura general del código de control, se podría haber mejorado. Hay que tener en cuenta que al planificar como se iba a programar la máquina aún no se tenían los conocimientos que se fueron adquiriendo a lo largo del proyecto, por lo tanto, muchas de las soluciones dadas en las primeras etapas, aunque válidas, podrían haber sido mejores de haberse iniciado el proyecto con el bagaje en programación adquirido. Solo hay que ver la diferencia entre los programas en Matlab, mucho más sencillos. En caso de proseguir con el desarrollo del proyecto, se podría rehacer de forma que fuese más sencillo y manejable gracias a los conocimientos adquiridos. Lógicamente la experiencia acumulada en el desarrollo del proyecto ya es un valor en sí mismo y es buen punto de partida para el desarrollo de futuros proyectos.

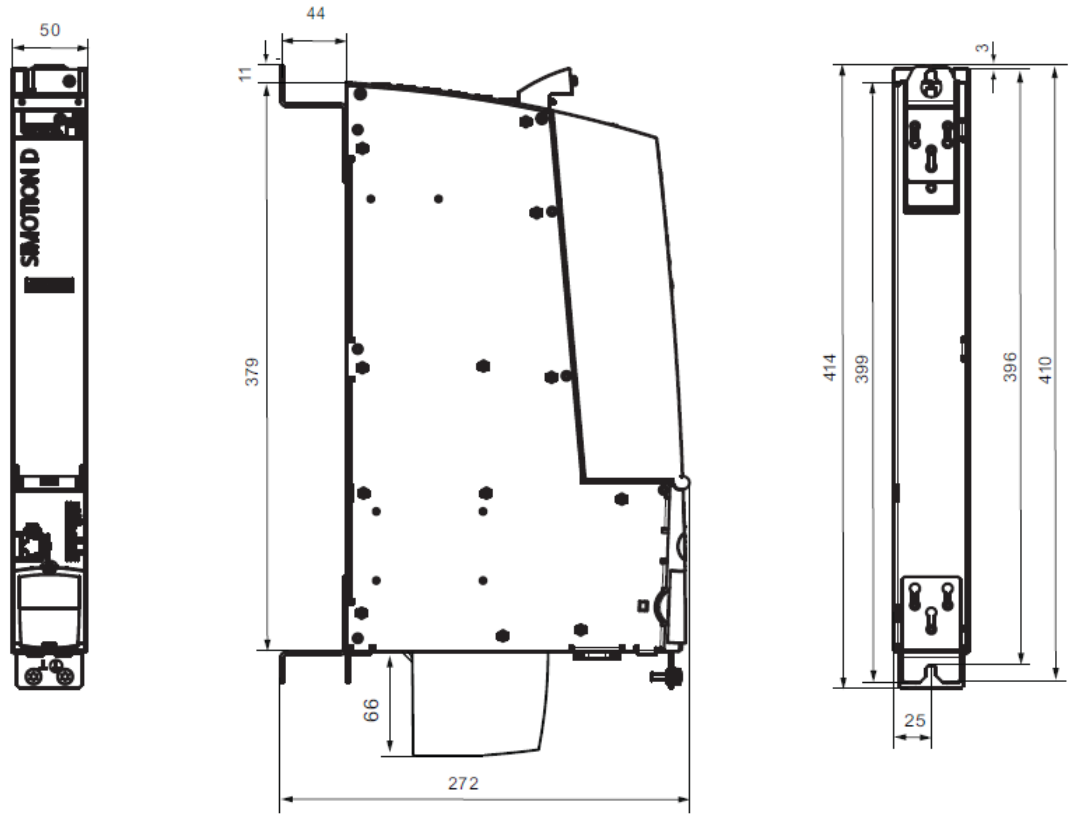
Anexo: Planos

(cotas en mm)

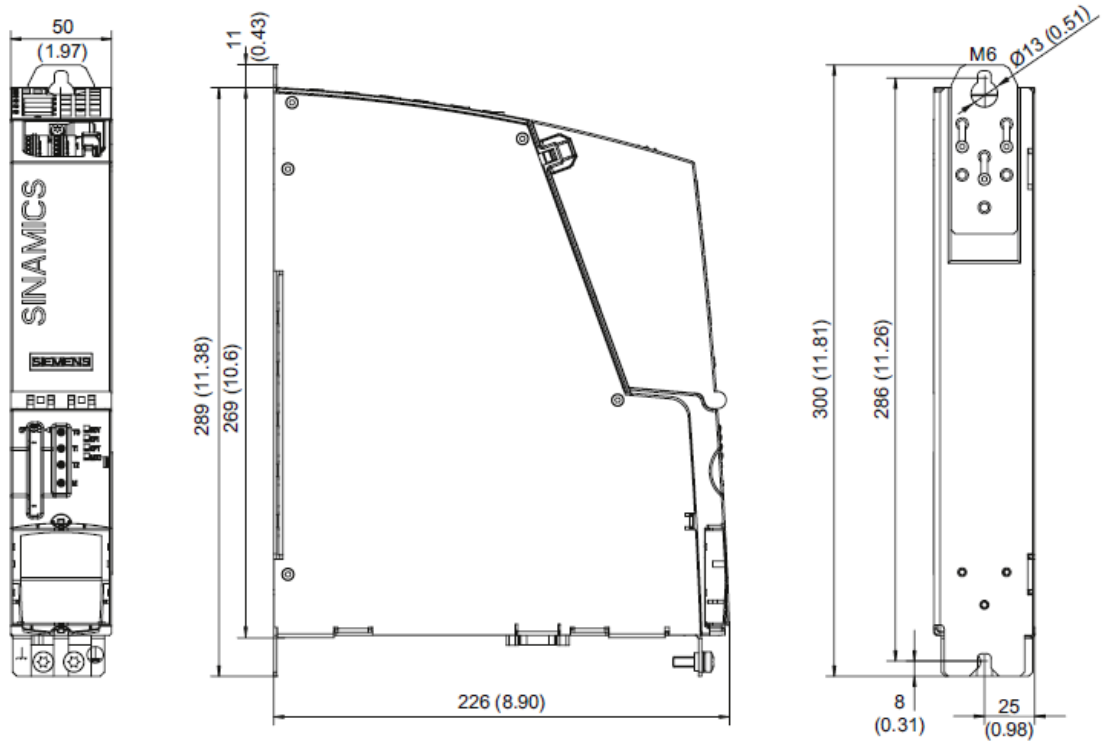
Plano mesa:



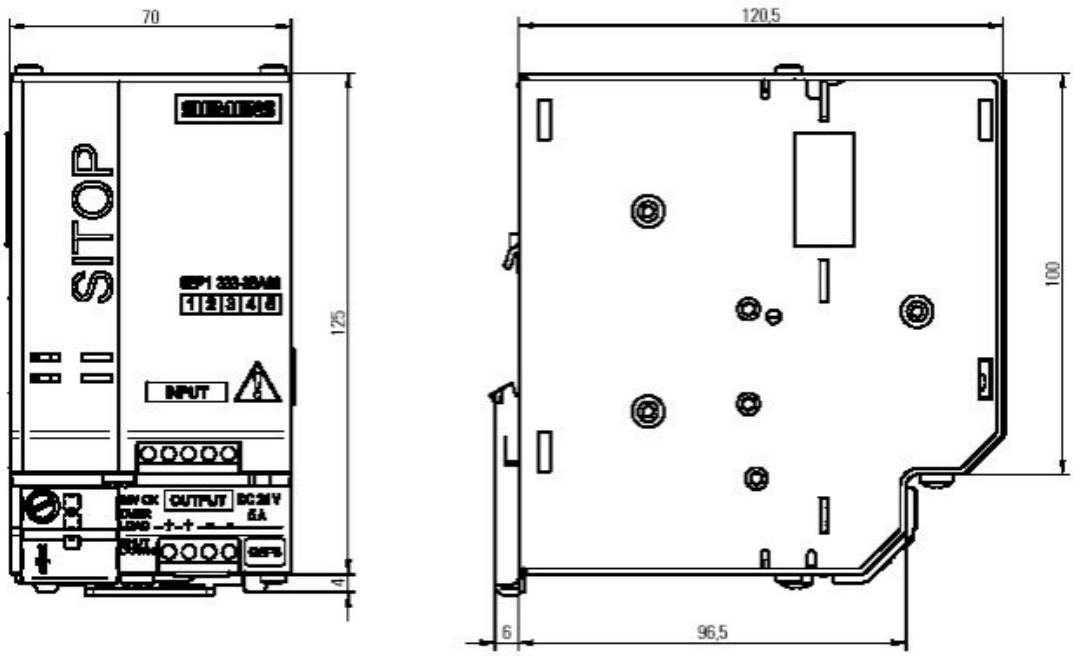
Dimensiones CPU:



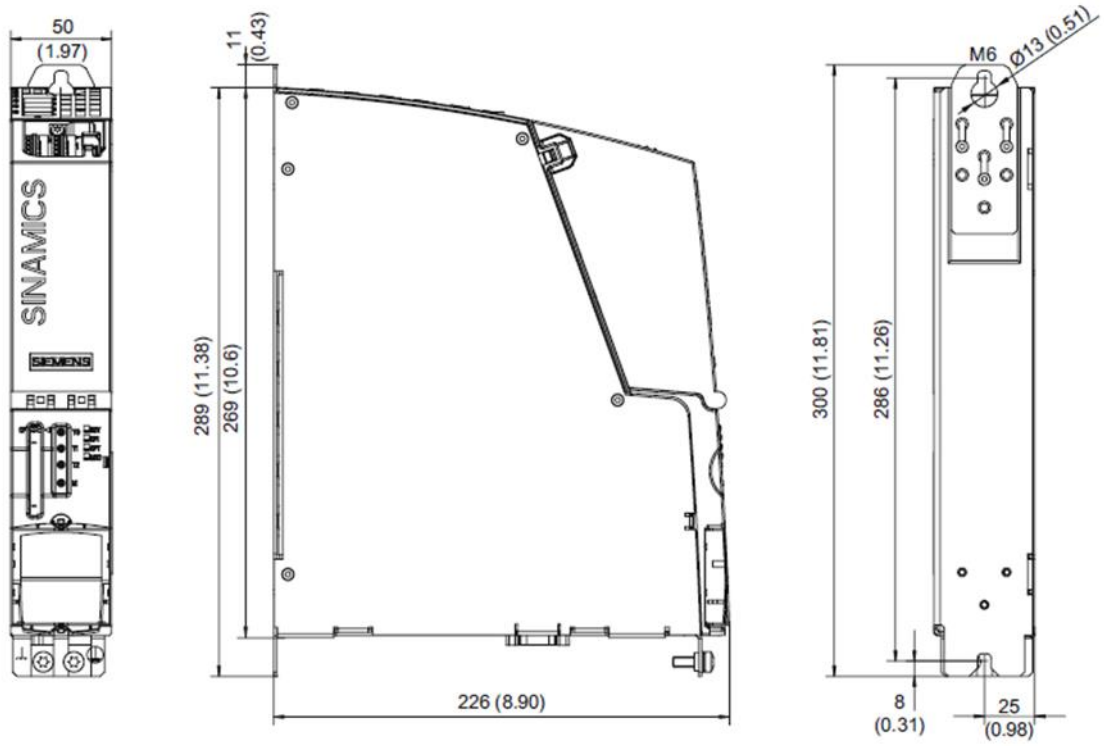
Dimensiones Double Motor Module:



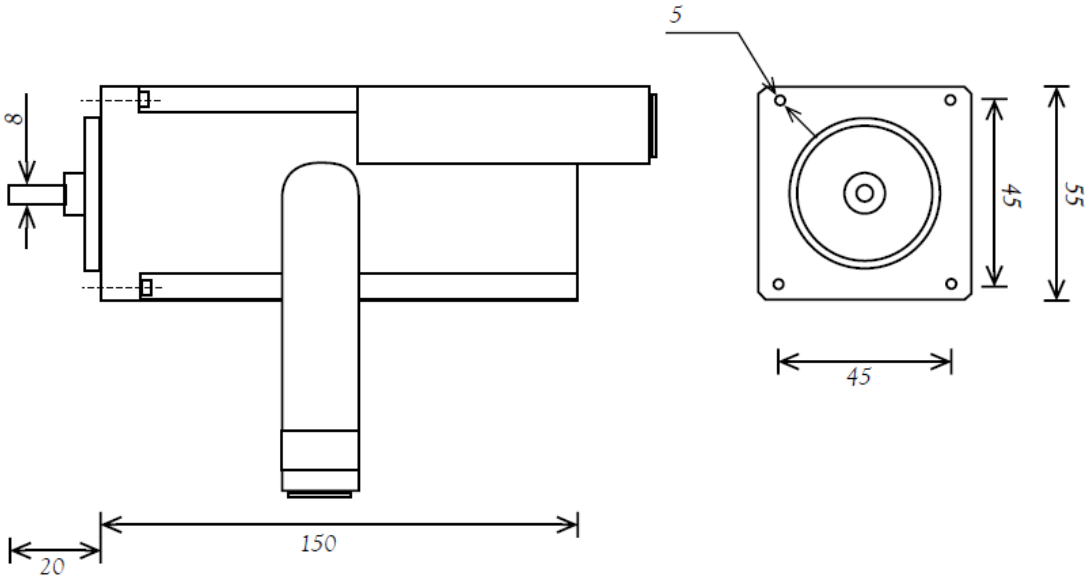
Dimensiones fuente de alimentación SITOP:



Dimensiones Smart Line Module:



Dimensiones Motores:



Bibliografía

Manuales de Siemens:

- **SIMOTION documentation overview**
- **SIMOTION SCOUT Getting Started SIMOTION SCOUT- sample project SIMOTION D435-2.**
- **Motion Control SIMOTION SCOUT, Configuration Manual.**
- **SIMOTION, Motion Control TO Axis Electric / Hydraulic, External Encoder, Function Manual**
- **SIMOTION ST Structured Text, Programming and Operating Manual**
- **SIMOTION MCC Motion Control Chart, Programming and Operating Manual**
- **SIMOTION D4x5-2, Manual**

Datasheets:

- **SITOP modular 5A 1/2phasig 6EP1 333-3BA00, Operating Manual**
- **SIMOTICS S-1FK7, Datasheet**
- **SINAMICS S120 Control Units and Supplementary System Components, Manual 11/2009, RGB ELEKTRONIKA AGACIAK CIACIEK, SPÓŁKA JAWNA**
- **SINAMICS S120 Motor Modules in booksize format C/D type, Datasheet**

Libros:

- **"Ampliación de Métodos Numéricos, Notas de Clase", Departamento de matemática aplicada, ETSI de Bilbao.**