Tecnológico de Costa Rica
Área Académica de Ingeniería Mecatrónica


# Desarrollo de un algoritmo de control de masa para una máquina MCV 1000 Kovosvit MAS

José Andrés Guardia Romero
201224030

11 de Julio del 2017

Instituto Tecnológico de Costa Rica
Área Académica de Ingeniería Mecatrónica
Proyecto de Graduación
Tribunal Evaluador

Proyecto de Graduación defendida ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Mecatrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

MSc. Marta Vílchez Monge
Profesora Lectora

Ing. Ignacio del Valle Granados
Profesor Lector

Dr. Juan Luis Crespo Mariño
Profesor Asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por el Área Académica de Ingeniería Mecatrónica.

Cartago, 31 de agosto de 2017

Declaro que el presente documento de tesis ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos y resultados experimentales propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de tesis realizado y por el contenido del presente documento.

José Andrés Guardia Romero

Cartago, 1 de septiembre de 2017

Céd: 1-1552-0892

## Resumen

Como parte del proyecto TE01020075 "Centro de Competencia - Tecnología de Fabricación" del gobierno de República Checa, el cual busca la investigación, el desarrollo y la innovación de medios técnicos, herramientas, soluciones y tecnologías con el fin de aumentar las principales características funcionales de las máquinas-herramienta y las máquinas de conformación se desarrolla el presente proyecto. Este desarrolla un algoritmo de control con el fin de automatizar el proceso de selección de parámetros acorde a la carga de la máquina para la optimización de sus características dinámicas.

En este trabajo se detalla la creación de un algoritmo en código Python capaz de calcular la carga, previo a un proceso, en la máquina Kovosvit MAS MCV 1000, de manera que a partir del valor obtenido sea posible ajustar los parámetros de los reguladores de la máquina para mejorar las características dinámicas de la misma. También se explica la creación de una rutina PLC que se encarga de realizar las mediciones necesarias para el cálculo de la carga, el llamado del proceso de Python y la modificación de los parámetros de la máquina, todo esto de manera automática.



**Palabra claves:** Características dinámicas, Kovosvit MAS MCV 1000, Python, PLC.

## Summary

As part of the TE01020075 "Competence Center - Manufacturing Technology" project of the government of the Czech Republic, which seeks the research, development and innovation of technical means, tools, solutions and technologies to increase the main Functional characteristics of machine tools and forming machines. The present project develops a control algorithm to automate the process of the selection of parameters according to the load of the machine for the optimization of its dynamic characteristics.

This work details the creation of an algorithm in Python code capable of calculating the load, previous a process in the machine Kovosvit MAS MCV 1000, and from the obtained value adjust the parameters of the machine controls to improve the dynamic characteristics of the machine. It also explains the creation of a PLC routine that is responsible for making the necessary measurements for the calculation of the load, the calling of the Python process and the modification of the parameters of the machine, everything automatically.



**Key words:** Dynamic characteristics, Kovosvit MAS MCV 1000, Python, PLC.

# Dedicatoria

***A DIOS***

En primera instancia agradezco a Dios por guiarme en este camino, por haberme dado los dones necesarios para poder desarrollarme como profesional y por su amor altruista y misericordioso que me ha entregado siempre. Dedico el máximo de mi capacidad y de mis posibilidades en este proyecto para Él.

***A MI FAMILIA***

Por su apoyo incondicional a lo largo de todos mis años de estudio, sobre todo en este último año de finalización de carrera y de vivir en otro país. Por ayudarme a ser la persona que soy ahora y por seguir confiando en mí y apoyándome en mis proyectos y sueños.

# Agradecimiento

Agradezco a Dios por darme la oportunidad de realizar mi proyecto de graduación y los dones necesarios para entregar lo mejor de mí en este trabajo. Por su constante compañía y su amor incondicional.

A mi familia por ayudarme económicamente para realizar mi proyecto de graduación en el extranjero, y por siempre estar ahí junto a mí para cualquier consejo o cualquier ayuda necesaria.

Agradezco al Movimiento de Vida Cristiana, por introducirme a una vida católica y ayudarme a formarme como persona en el ámbito psicológico y espiritual. No sería la persona que soy ahora sin mi formación dentro de la espiritualidad.

Al RCMT de Praga, y a todos sus miembros y mis compañeros de trabajo. Por hacerme sentir en casa, y por la constante ayuda a todas mis preguntas y dudas a lo largo del semestre y del desarrollo del proyecto. Sobre todo, a mi profesor guía, Petr Fotju.

A mis amigos que me acompañaron en la experiencia de vivir un semestre en Europa, por darme unos de los meses más felices de mi vida y experiencias inolvidables que hicieron la experiencia de desarrollar este proyecto en República Checa aún más gratificante.

A mis amigos aquí en Costa Rica, por apoyarme a lo largo de estos meses viviendo lejos, y por las amistades verdaderas que me han dado a lo largo de los años que me han hecho seguir adelante ante cada obstáculo.

Finalmente, a mi tutor Juan Luis Crespo por ayudarme con las dudas y creer en mi proyecto como trabajo digno y completo para el título de ingeniero en el área de ingeniería mecatrónica. Y, sobre todo, por apoyarme en realizar mi proyecto en el RCMT en Praga, República Checa.

# Tabla de Contenido

# Índice de Figuras

# Índice de Tablas

# 1. Introducción

Desde los inicios de la sociedad, los humanos han utilizado herramientas e instrumentos para facilitar sus labores, y para prosperar como civilización. Sin embargo, para usar estas herramientas primero debió existir un proceso de fabricación de la misma. Lógicamente, las primeras herramientas de los humanos no tenían diseños complejos y normalmente solo cumplían una función general y de una manera poco eficiente. Con el paso del tiempo, estas herramientas fueron volviéndose más complejas y más útiles hasta convertirse en herramientas capaces de convertir materia prima en casi cualquier geometría deseada.

Estas herramientas en verdad resultan ser una fusión de una máquina y una herramienta y por eso llevan ese nombre; máquina-herramienta. Entre las más destacadas están el torno, para la creación de perfiles cilíndricos, y las fresas para perfiles rectos. Cada una de estas máquinas busca ser lo más versátil posible y de brindar una ayuda a la industria de producción en masa.

Las máquinas-herramientas solían pertenecer a la ingeniería mecánica, ya que sus creaciones y procesos eran casi todos mecánicos. Pero, con la fusión de la electrónica, la mecánica y la informática, estas máquinas evolucionaron a las máquinas-herramientas CNC. Las CNC (Control numérico por computadora), dieron un paso más en la industria de las máquinas herramientas, ya que al ser programadas se mejoró la exactitud de las piezas, la calidad de las mismas, la velocidad de producción, la eficiencia del costo, la confiabilidad del producto y el rendimiento de producción entre otros. En general, el incremento en la utilización de máquinas herramientas con CN se debe a que un gran número de problemas, que se consideraban bien resueltos por los métodos de trabajo clásicos, que pueden tener una respuesta ventajosa desde el punto de vista técnico mediante la utilización de dichas máquinas.

En los últimos años estas máquinas CNC se siguen mejorando y cada vez son más comunes en la industria tal y como la conocemos. Sin embargo, la mejora siempre es posible, por lo tanto, es el trabajo de investigadores buscar las mejoras en estas máquinas para así cada vez obtener una mejora en la calidad que estas herramientas producen.

La ‘Industria de Maquinaria y Equipo’ resulta de alta relevancia para el desarrollo productivo de una Nación principalmente porque: es fuente de empleos de alta calificación y remuneración, es el sector que provee de maquinarias para la producción de todos los bienes que componen el producto, repercutiendo en su calidad y en su costo, debido a que los esfuerzos hacia el desarrollo suelen generar tensión en las cuentas externas nacionales, es deseable el desarrollo de sectores que contribuyan a permitir incrementos del producto sin que esto genere incrementos considerables de las importaciones.

Por estos motivos, muchos países de primer mundo han tomado las riendas de la industria de máquinas herramientas y se han coronado como líderes en este mercado. Uno de estos países es República Checa. República Checa decidió entrar en el negocio de las máquinas herramientas hace no más de una década, y en este momento se sitúa en el número 14 de mayores productores del mundo y en número 13 como mayor exportador de dichas máquinas. En la siguiente imagen se puede observar el censo tomado en el 2015 respecto a la situación.

**Producers**

| | Country | | $-Millions 2013 (est.) | % Cut | % Form | $-Millions 2012 (rev.) | Change in local currency | Change in U.S. dollars |
|---|---|---|---|---|---|---|---|---|
| 1. | Germany | | 14,687.7 | 72% | 28% | 13,824.9 | 3% | 6% |
| 2. | Japan | | 12,326.4 | 84% | 16% | 18,231.3 | -18% | -32% |
| 3. | China, P. Rep. | | 8,743.0 | 60% | 40% | 9,236.7 | $ | -5% |
| | *as reported* | | *24,980.0* | *60%* | *40%* | *27,990.0* | *$* | *-11%* |
| 4. | Italy | | 5,710.4 | 50% | 50% | 5,606.1 | -1% | 2% |
| 5. | South Korea | | 5,306.0 | 71% | 29% | 5,485.0 | $ | -3% |
| 6. | United States | | 4,956.1 | 75% | 25% | 4,983.2 | $ | -1% |
| 7. | Taiwan | | 4,537.0 | 82% | 18% | 5,414.0 | $ | -16% |
| 8. | Switzerland | | 3,129.1 | 83% | 17% | 3,282.2 | -6% | -5% |
| 9. | Spain | | 1,218.6 | 58% | 42% | 1,095.1 | 8% | 11% |
| 10. | Austria | | 1,094.3 | 54% | 46% | 1,000.1 | 6% | 9% |
| 11. | United Kingdom | | 891.7 | 69% | 31% | 911.7 | -1% | -2% |
| 12. | Canada | c | 803.4 | 61% | 39% | 752.2 | 10% | 7% |
| 13. | Turkey | | 709.2 | 26% | 74% | 644.2 | 7% | 10% |
| 14. | Czech Republic | | 705.6 | 82% | 18% | 720.0 | -2% | -2% |
| 15. | France | | 686.6 | 64% | 36% | 752.2 | -12% | -9% |
| 16. | India | | 658.0 | 85% | 15% | 798.0 | $ | -18% |
| 17. | Brazil | | 420.1 | 81% | 19% | 643.3 | $ | -35% |
| 18. | Netherlands | | 415.7 | 20% | 80% | 402.5 | 0% | 3% |
| 19. | Mexico | c | 374.4 | 62% | 38% | 389.4 | $ | -4% |
| 20. | Belgium | | 324.0 | 20% | 80% | 304.7 | 3% | 6% |
| 21. | Russia | c | 210.9 | 41% | 59% | 263.0 | $ | -20% |
| 22. | Sweden | | 208.5 | 38% | 62% | 201.9 | 0% | 3% |
| 23. | Finland | | 184.6 | 20% | 80% | 187.7 | -5% | -2% |
| 24. | Australia | | 160.0 | 88% | 12% | 148.0 | $ | 8% |
| 25. | Portugal | | 74.4 | 46% | 54% | 70.7 | 2% | 5% |
| 26. | Denmark | | 73.0 | 40% | 60% | 70.7 | 0% | 3% |
| 27. | Argentina | | 43.1 | 53% | 47% | 39.7 | $ | 9% |
| | **Total** | | **68,651.8** | | | **75,458.5** | | **-9%** |

*Figura 1 - 1: Listado de los mayores productores de máquinas herramientas en el mundo. Garder Research (2016). World Machine Tool Survey.*

República Checa posee una larga tradición en la fabricación de máquinas herramientas y conformadoras y tiene gran fama como país exportador de bienes de equipo. También los resultados en el marco del CEFTA (Acuerdo sobre la Zona Centroeuropea de Libre Comercio) son favorables para la República Checa. Por su participación sobre las exportaciones de los países del CEFTA a la UE, esta nación ocupa la primera posición, lo cual es una evidente prueba de que se mantiene la competitividad de los productos checos.

El sector debe ir asegurando su capacidad de competir mediante un incremento continuo del nivel técnico de los productos finales y del nivel tecnológico, procedimientos de prefabricación mejorados, seguimiento sistemático de la calidad, respuesta flexible a las exigencias del cliente, suministros integrales, servicio de posventa accesible y rápido, así como mediante el aumento del prestigio de las empresas.

## 1.1 Planteamiento del problema

En marzo de 2012, la Agencia de Tecnología de la República Checa, que había convocado el primer concurso público de investigación, desarrollo e innovación para el programa de Centros de Competencia, concedió el proyecto TE01020075 "Centro de Competencia - Tecnología de Fabricación" para el periodo 2012-2019.

El proyecto se centra en la investigación, el desarrollo y la innovación de medios técnicos, herramientas, soluciones y tecnologías con el fin de aumentar las principales características funcionales de las máquinas-herramienta y las máquinas de conformación. Con este proyecto, el país busca entrar en los mejores 10 países exportadores de máquinas-herramientas en el mundo para el 2020.

A partir de este gran proyecto surgen muchos proyectos de investigación con el objetivo de mejorar las principales características funcionales de una máquina herramienta. Estas son:

- Exactitud;
- Calidad;
- Rendimiento de la producción;
- Confiabilidad;
- Eficiencia de costo;
- Rendimiento ambiental.

Uno de dichos proyectos es aquel tratado en este documento. La implementación de un algoritmo de control de masa en la serie de máquinas MCV 1000. Estas máquinas provienen del productor Kovosvit, que es uno de los mayores productores de máquinas herramientas en

república checa. El algoritmo de control de masa buscaría mejorar las máquinas herramientas en las características funcionales mencionadas anteriormente, centrándose en la exactitud como meta principal. La máquina Kovosvit MAS MCV 1000 se presenta en la siguiente figura.


*Figura 1 - 2: máquina-herramienta Kovosvit MAS MCV 1000.*

Muchos usuarios de máquinas-herramienta operan las mismas y no prestan atención al ajuste de la unidad de regulación. La configuración de los regulados de cada eje se guarda generalmente como fue fijado por el constructor de la máquina-herramienta y parece, que solamente se realizan cambios cuando ocurre un comportamiento "dramático" de la máquina. Sin embargo, existe la posibilidad de realizar un ajuste óptimo del avance de cada pieza de trabajo y aprovechar al máximo la dinámica y la precisión de la máquina disponible. El ajuste óptimo del eje de avance de alimentación depende directamente de la carga y está limitado por frecuencias naturales. En general, las frecuencias naturales cambian con la carga, aunque este efecto es a veces insignificante.

Para esto, el algoritmo tiene como función automatizar el proceso configurar los parámetros de los reguladores de velocidad de la máquina cada vez que se empiece un proceso en la misma. Esta regulación actualmente se hace de manera manual para cada pieza que se vaya a trabajar y suele dar errores de precisión pequeños en el mecanizado de la pieza. La regulación está directamente relacionada a la masa total del sistema en movimiento.

### 1.1.1 Síntesis del Problema

La falta de configurar los reguladores de velocidad acorde a la masa total del sistema en movimiento, en las máquinas MCV 1000 Kovosvit MAS, son fuente de imprecisiones.

## 1.2 Objetivos del Proyecto

### 1.2.1 Objetivo General

Realizar un algoritmo de control de masa que se pueda implementar en las máquinas MCV 1000 Kovosvit de República Checa para mejorar el comportamiento dinámico de la misma, acorde a la suma de la masa de sus piezas móviles.

### 1.2.2 Objetivo Específicos

#### *1.2.2.1 Calculo de masa*

- Diseñar un proceso para obtener la masa total del sistema de movimiento de la máquina MAS MCV 1000 en un proceso, ya que las propiedades dinámicas de la máquina dependen significativamente de esta variable.
- Implementar el proceso de medición de masa dentro de la estación de programación del sistema de control iTNC 530 con una incertidumbre de no más del 5% de la masa en movimiento.

#### *1.2.2.2 Programación PLC*

- Crear una rutina que permita al PLC de la máquina obtener los datos necesarios del sistema para realizar el proceso de cálculo de carga, que llame al proceso Python encargado de calcular la masa y finalmente varíe los parámetros de la máquina acorde al resultado del proceso Python.

#### *1.2.2.3 Cambio de parámetros*

- Diseñar un algoritmo en lenguaje Python que obtenga los datos necesarios del sistema para realizar el proceso de cálculo de masa, que realice el cálculo de la masa y partir de este valor realice los cambios necesarios en un archivo CN encriptado para el cambio de parámetros.

#### *1.2.2.4 Implementación de proceso en la máquina*

- Verificación del algoritmo en la máquina Kovosvit MAS MVC 1000. Que se realice el cambio de parámetros de manera automática acorde a la carga en el proceso.

# 2. Marco Teórico

En esta sección se describen los conceptos teóricos necesarios para entender el procedimiento del proyecto a realizar. Esta sección se divide en cinco subsecciones, donde cada una entra en detalle de uno de los aspectos del proyecto.

La primera sección describe que es una máquina herramienta y la implementación del control numérico por computadora (CNC) en ellas. La segunda sección trata sobre la estación de programación y de control iTNC 530 dada por la empresa Heidenhain. En la tercera sección se describe superficialmente qué es el lenguaje de programación de alto nivel Python, y como este se ha logrado implementar en los sistemas de Heidenhain. La cuarta sección describe lo que es un controlador lógico programable, ya que la máquina cuenta con uno para sus funciones, y también describe como acceder y manipular este desde la estación de control iTNC 530. Por último, la quinta sección se refiere a la teoría de conceptos matemáticos usados a lo largo del proyecto.

## 2.1 Máquina-herramienta

La máquina herramienta, descrita por Agüero y compañía (2013), es un tipo de máquina que se utiliza para dar forma a piezas sólidas, principalmente productos de la siderurgia y/o sus derivados. Suelen ser máquinas estacionarias. El moldeado de la pieza se realiza por la eliminación de una parte del material. Entre las maquinas-herramientas que existen podemos encontrar las fresadoras, los tornos, las taladradoras entre otros. También se les conoce como centros de mecanizado.

Existen distintas maneras de clasificar los tipos de máquina herramienta, Herriko (s.f) en su trabajo de Máquinas-Herramienta: Funciones, tipos y arquitecturas, utiliza el método de dividirlas en tres grupos según la forma en que manipulan la pieza de trabajo. Estos grupos son:

- De desbaste: aquellas máquinas herramientas que dan forma a la pieza por medio del arranque de viruta, que se define como un fragmento de material residual en forma de lámina.

- De prensado: este grupo de máquinas dan la forma a la pieza por medio del cortado, prensado o estampado.

- Especiales: son las máquinas-herramientas que no entran en ninguno de los dos grupos anteriores ya que usan otro tipo de técnica para dar forma a sus piezas, entre estos tipos de forma podemos encontrar: láser, electroerosión, plasma, entre otros.

A continuación, se describen algunas de las más comunes máquinas-herramientas en la actualidad.

- Fresadora: es una máquina herramienta utilizada para realizar mecanizado por arranque de viruta mediante el movimiento de una herramienta rotativa de varios filos de corte denominada fresa. En las fresadoras tradicionales, la herramienta o fresa gira en una posición fija y el desbastado se realiza acercando la pieza a la herramienta.

  La fresadora se emplea para realizar trabajos en superficies planas o perfiles irregulares, pudiendo también utilizarse para tallar engranajes y roscas, taladrar y mandrilar agujeros, ranuras, chaveteras y graduar con precisión medidas regularmente espaciadas. En la figura 2-1 se muestra una fresadora convencional.

*Figura 2- 1: fresadora convencional.*

- Torno: máquina herramienta que permite mecanizar piezas de forma geométrica de revolución (cilindros, conos, hélices). Estas máquinas-herramienta operan haciendo girar la pieza a mecanizar mientras una o varias herramientas de corte son empujadas en un movimiento de avance contra la superficie de la pieza, cortando las partes sobrantes en forma de viruta, la figura 2-2 muestra una de estas máquinas.

*Figura 2- 2: torno convencional.*

- Taladradora: una taladradora típica, como la mostrada en la figura 2-3, El principio de la operación de esta máquina - herramienta es perforar o hacer un agujero en una pieza de cualquier material. Estas máquinas tienen dos movimientos: el de rotación de la broca que le imprime el motor eléctrico de la máquina a través de una transmisión por poleas y engranajes, y el de avance de penetración de la broca, que puede realizarse de forma manual sensitiva o de forma automática si la máquina lo permite.

*Figura 2- 3: Taladradora.*

### 2.1.1 Control Numérico por Computadora (CNC).

El control numérico computarizado (CNC) lo describe Díaz (agosto del 2008) en su documento de Programación automática de máquinas CNC como un sistema de automatización de máquinas herramienta que son operadas mediante comandos programados en un medio de almacenamiento, en comparación con el mando manual mediante volantes o palancas. Este sistema ha revolucionado la industria debido al abaratamiento de microprocesadores y a la simplificación de la programación de las máquinas de CNC. También se le conoce como CN (control numérico) o por sus siglas en inglés, NC.

Normalmente se siguen dos estándares mundiales, estos son estándares de instrucciones de programación (código) que permiten a la máquina herramienta llevar a cabo ciertas operaciones en particular. A estos estándares se les conoce como:

- ISO 6983 (Organización internacional de estandarización)
- EIA RS274 (Asociación de industrias electrónicas)

Estos estándares definen el sistema de coordenadas de la máquina y los movimientos de la misma. En la figura 2-4 se observa cómo se coloca el sistema de coordenadas en una máquina herramienta, en este caso en una fresadora.

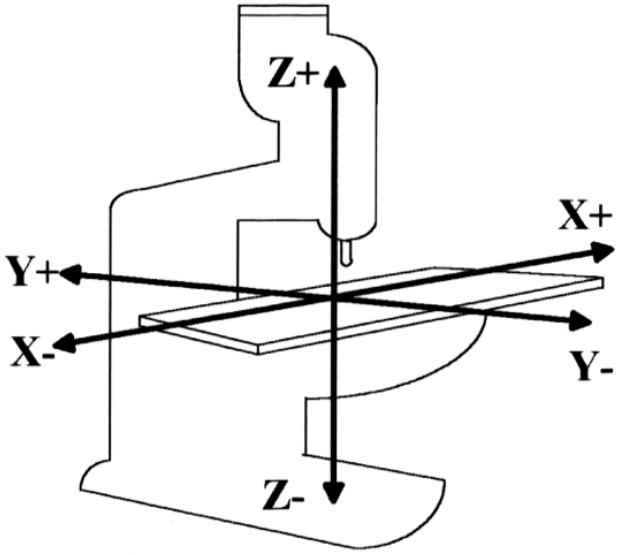



*Figura 2- 4:sistema de coordenadas para una fresadora. Jiménez, R (s.f) Control Numérico por Computadora.*

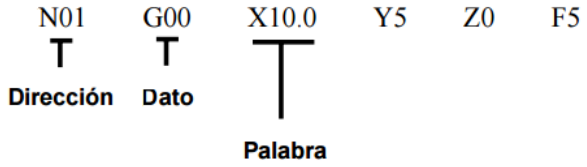
Podemos observar que el eje z es el eje de ataque de la herramienta, el eje x es paralelo a la mesa de la máquina, mientras que el eje y es perpendicular a la mesa.

### 2.1.2 Programación CNC

Jiménez (s.f), en su documento de Control Numérico por Computadora (CNC) describe un programa como una lista secuencial de instrucciones de maquinado que serán ejecutadas por

la máquina de CNC A las instrucciones se les conoce como CODIGO de CNC, las cuales deben contener toda la información requerida para lograr el maquinado de la pieza.

Hay que tener en cuenta que cada movimiento o acción se realiza secuencialmente. A una línea de código se le conoce como bloque y normalmente están constituidos por una dirección (número de bloque), por un dato, una palabra, las coordenadas finales de la acción a realizar y la velocidad de avance.




Estos bloques, o instrucciones se pueden dividir en dos grupos. Los códigos G, y los códigos M. Los primeros se encargan de funciones de movimiento de la máquina (movimientos rápidos, avances, avances radiales, pausas, ciclos), mientras que los segundos se encargan de funciones misceláneas que se requieren para el maquinado de piezas, pero no son de movimiento de la máquina (arranque y paro del husillo, cambio de herramienta, refrigerante, paro de programa).

Los códigos G contienen variables ya definidas por los estándares para cada función específica. En la table 2-1 se definen las variables más importantes. Esta tabla fue obtenida del documento de Jiménez (s.f).

*Tabla 2- 1: Funciones de variables definidas en código G. Extraída de Jiménez, R (s.f)*

| Variable | Función |
|---|---|
| N | Número de bloque |
| G | Función preparatoria |
| X | Coordenada X |
| Y | Coordenada Y |
| Z | Coordenada Z |
| I | Localización en X del centro de un arco |
| J | Localización en Y del centro de un arco |
| K | Localización en Z del centro de un arco |
| S | Velocidad del husillo |
| F | Asigna velocidad de corte |
| M | Función Miscelánea |

A continuación, se presenta a manera de ejemplo un código en CNC y la descripción de cada bloque de código.

| | |
|---|---|
| N5 G90 G20 | Unidades absolutas, programación en pulgadas. |
| N10 T0202 | Paro para cambio de herramienta, Usar #2 |
| N15 M03 S1200 | Prender husillo a 1200 rpm CW |
| N20 G00 X1 Y1 | Movimiento rápido a (X1, Y1) |
| N25 Z0.125 | Movimiento rápido a Z0.125 |
| N30 G01 Z-0.125 F 5 | Avance a Z-0.125 a 5 ppm (pulgada por minuto) |
| N35 G00 Z1 | Movimiento rápido a Z1 |
| N40 X0 Y0 | Movimiento rápido a X0, Y0 |
| N45 M05 | Apagar el husillo |
| N50 M30 | Fin del programa |

### 2.1.3 Ajuste dependiente de la carga de la unidad de alimentación

El sistema de control para la unidad de alimentación de la máquina Kovosvit MCV 1000 MAS se puede representar como un sistema de control en cascada con realimentación. En la siguiente figura se muestra el sistema.

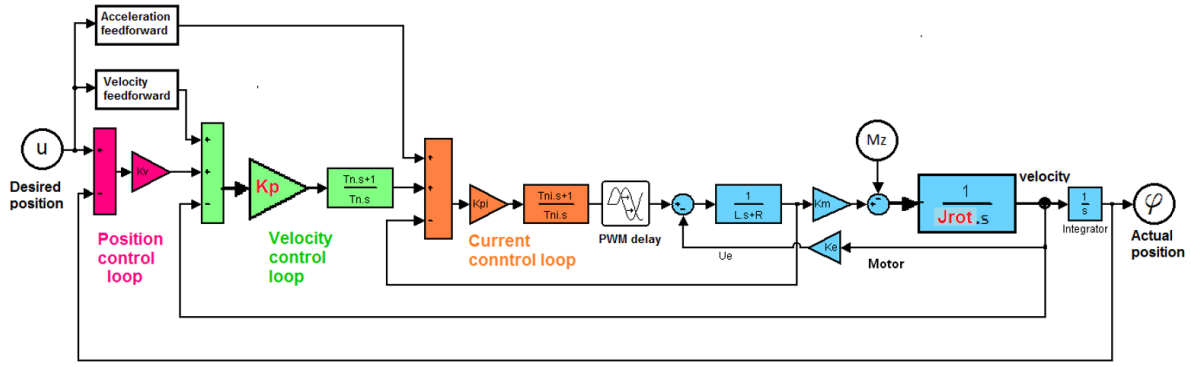



*Figura 2- 5: sistema de control para el sistema de realimentación de la máquina Kovosvit MAS MCV 1000. Imagen otorgada por el RCMT.*

Donde:

- Kv [1 / s] ... factor Kv
- Kp [As / rad] ... ganancia proporcional del regulador de velocidad
- Tn [s] ... constante de tiempo del controlador de velocidad
- Kpi [V / A] ... ganancia proporcional del controlador de corriente
- Tni [s] ... constante de tiempo del controlador de corriente
- L [H] ... inductancia del devanado del motor
- R [Ω] ... resistencia del devanado del motor
- Km [Nm / A] ... constante de par
- Ke [Vs / rad] ... velocidad EMF constante
- Jrot [kg.m2] ... momento de inercia reducido del eje de accionamiento de avance
- Mz [Nm] ... carga externa

Si consideramos la función de transferencia de unidad del control de corriente, podemos derivar la fórmula clara para la ganancia total del control de velocidad KR,

$$K_R = \frac{K_P.K_m}{J_{ROT}.T_N} \qquad (2\text{-}1)$$

Con respecto a la estabilidad podemos ver que la ganancia proporcional del regulador de velocidad Kp (numerador) puede aumentarse proporcionalmente al reducir el momento de inercia Jrot del eje de accionamiento de alimentación (denominador) y la ganancia global del control de velocidad $K_R$ permanece en el mismo valor. La misma idea es válida para el eje lineal, Jrot tiene que ser sustituido por la masa reducida m. Si se cumple esta suposición, la dinámica del control de velocidad permanece en el valor óptimo.

Esta propiedad es importante especialmente para ejes rotatorios, debido a que su característica de masa - momento de inercia reducido - aumenta significativamente a pesar de que la masa de la pieza de trabajo aumenta humildemente. Esto se ejemplifica en la figura 2-6.

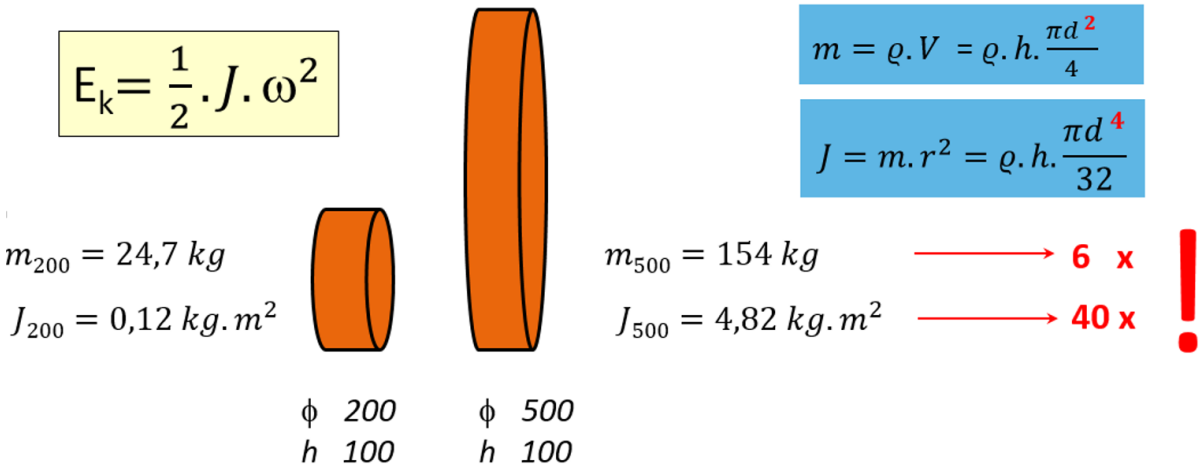



*Figura 2- 6: ejemplo de aumento de masa y de momento de inercia al aumentar el tamaño de un eje rotatorio. Imagen otorgada por el RCMT.*

Con base en a figura 2-6, observamos dos volantes con el mismo grosor h, pero con diámetro diferente ⌀, ambos están hechos del mismo material. De las fórmulas anteriores podemos ver, que la masa cambia con el cuadrado del diámetro mientras que el momento de la inercia con la potencia a la cuatro del diámetro. La masa de ambos volantes difiere aproximadamente 6 veces mientras que el momento de inercia difiere 40 veces. La consecuencia más importante es que el momento de inercia determina la energía cinética del movimiento y, por consiguiente, la dinámica del movimiento y las siguientes exigencias sobre la ganancia del bucle de velocidad.

Esta es la razón por la cual el cambio de la ganancia proporcional Kp del control de velocidad para cada pieza de trabajo es tan importante para la utilización óptima de la capacidad de la máquina herramienta. No sólo la ganancia proporcional del controlador de velocidad tiene que ser cambiado, sino también el factor Kv tienen que ser optimizado. Sin embargo, su ajuste es relativamente fácil en comparación con el ajuste del bucle de control de velocidad.

### 2.1.4 Comportamiento dinámico a diferentes cargas.

La importancia de este proyecto de investigación reside en la premisa que la carga del proceso dentro de una máquina-herramienta puede cambiar el comportamiento dinámico de la misma de manera considerable. El comportamiento dinámico de la máquina puede ser estudiado desde la función de velocidad entregada por el regulador de velocidad respecto al tiempo. Esta función es una función senoidal con un período de tiempo T definido por la frecuencia otorgada por el regulador de velocidad dentro de la máquina.

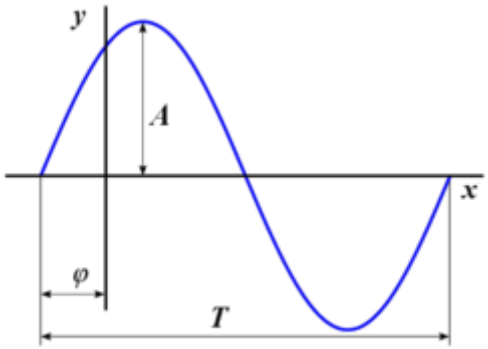



*Figura 2- 7: Función senoidal de velocidad respecto al tiempo. Ciudad universitaria virtual de San Isidoro (s.f)*

La amplitud A y el ángulo de desfase φ se pueden observar mejor en los diagramas de bode para observar el comportamiento dinámico de la máquina. En la figura 2-8 se representan estas dos gráficas.

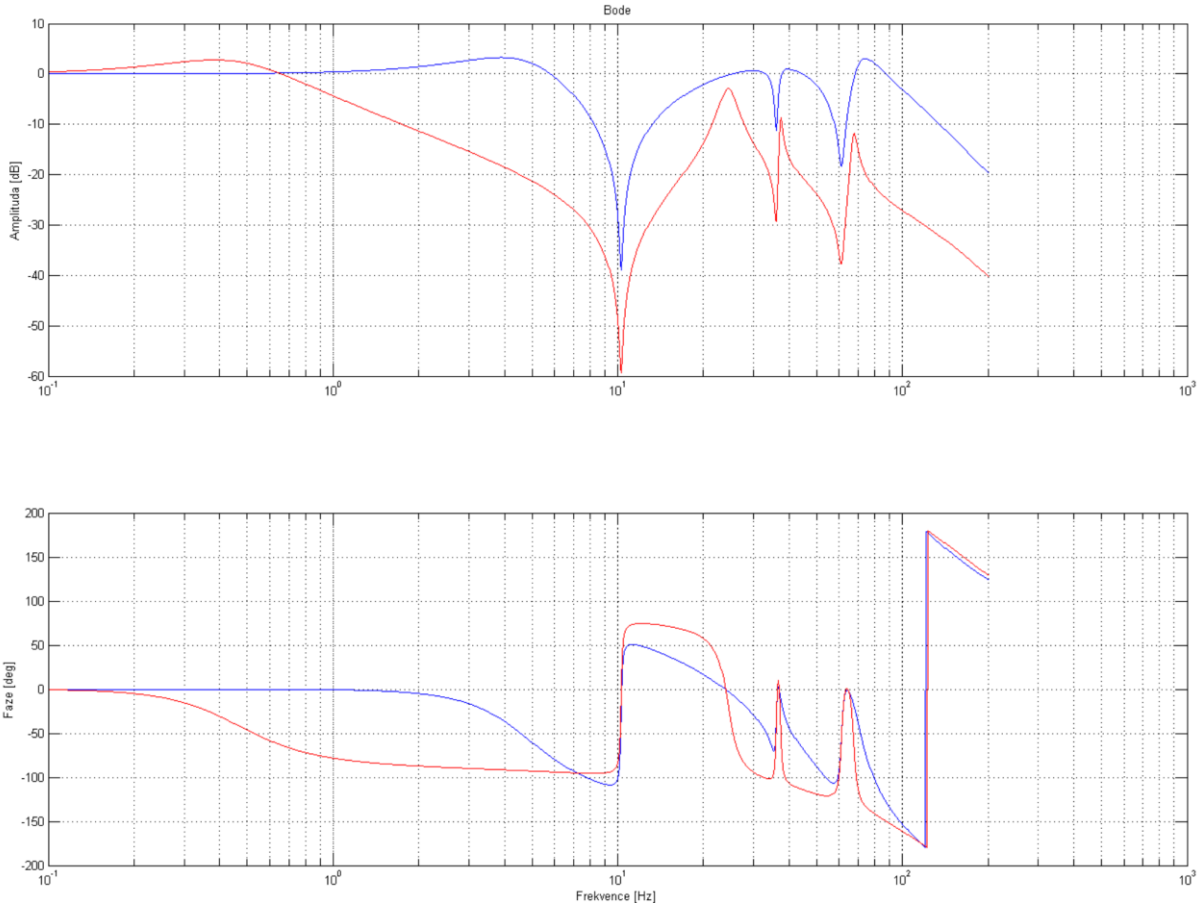



*Figura 2- 8: comportamiento dinámico del regulador de velocidad de la máquina Kovosvit MAS MCV 1000. Gráficas otorgadas por el RCMT.*

En la figura 2-8 se puede observar que la frecuencia tiene un gran peso en el valor la amplitud y en el desfase del ángulo de la función senoidal representada en la figura 2-7. La función azul representa el comportamiento dinámico de la máquina Kovosvit MAS MCV 1000 sin ninguna pieza de trabajo para un proceso. La función roja representa el comportamiento dinámico de la misma máquina al agregar una pieza de trabajo, por ende, aumentar el valor de la carga del proceso.

La frecuencia que se debe elegir en el regulador de velocidad debe ser una donde la amplitud esté entre los -3 db y los 3 db, a esto se le conoce como el margen de seguridad para el comportamiento dinámica de una estación de mecanizado. También, por el tipo de comportamiento dinámico que se obtiene, existe un pico de amplitud negativa que representa una zona de mucho peligro en la máquina. Por esta razón nunca se debe elegir una frecuencia para el regulador que se ubique posterior a este pico, ya que la máquina podría fallar. También se recomienda elegir frecuencia que no se acerquen a esta zona de peligro para evitar posibles fallas en la máquina.

El problema ocurre cuando se elige una frecuencia segura a una carga y al cambiar la carga el comportamiento dinámico varía de tal manera que dicha frecuencia representa un peligro para la máquina. Para solucionar esto se deben usar los filtros dentro del regulador de

velocidad que permiten cambiar la forma de la función del comportamiento dinámico. Cabe destacar que el uso de filtros suele ayudar con la amplitud del comportamiento, pero afecta de manera negativa al ángulo de desfase del comportamiento.

Por último, la frecuencia a elegir también afecta considerablemente el tiempo de sentamiento del valor de velocidad. En la figura 2-9 se puede observar este fenómeno.

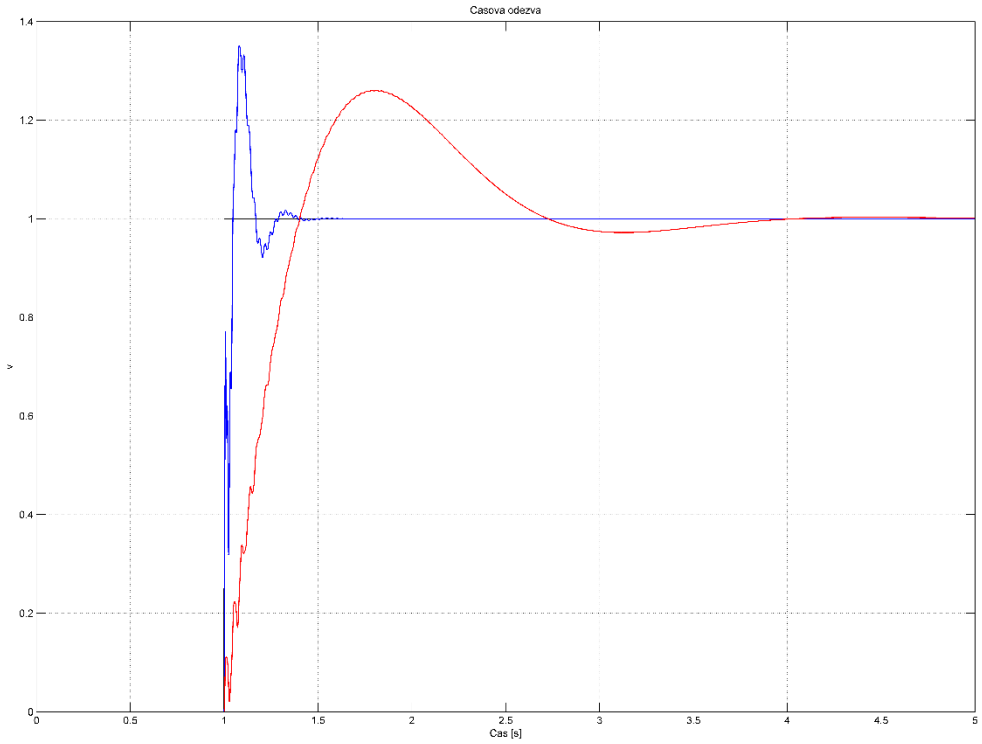



*Figura 2- 9:tiempo de asentamiento del valor de velocidad respecto a la carga del proceso. Gráfica otrogado por el RCMT.*

En esta figura se puede observar el comportamiento sin carga (línea azul) y al agregar una carga (línea roja). El tiempo de asentamiento se incrementa de manera considerable, aunque se mejora el valor del ángulo de desfase en la sobrecarga. Por estas razones de sebe seleccionar valores dentro del regulador que permitan optimizar el tiempo de asentamiento sin tener un mayor efecto negativo en el ángulo de desplazo.

## 2.2 iTNC 530

El iTNC 530 de HEIDENHAIN es un control de contorneado versátil, orientado al taller, para fresadoras, perforadoras y taladradoras, así como centros de mecanizado. El iTNC 530 es universal, y su amplia y compleja gama de aplicaciones lo demuestra.

- Fresadoras universales

- Fresado a alta velocidad
- Mecanizado de cinco ejes con cabeza giratoria y mesa giratoria
- Mecanizado de cinco ejes en máquinas muy grandes
- Taladradoras
- Centros de mecanizado y mecanizado automatizado

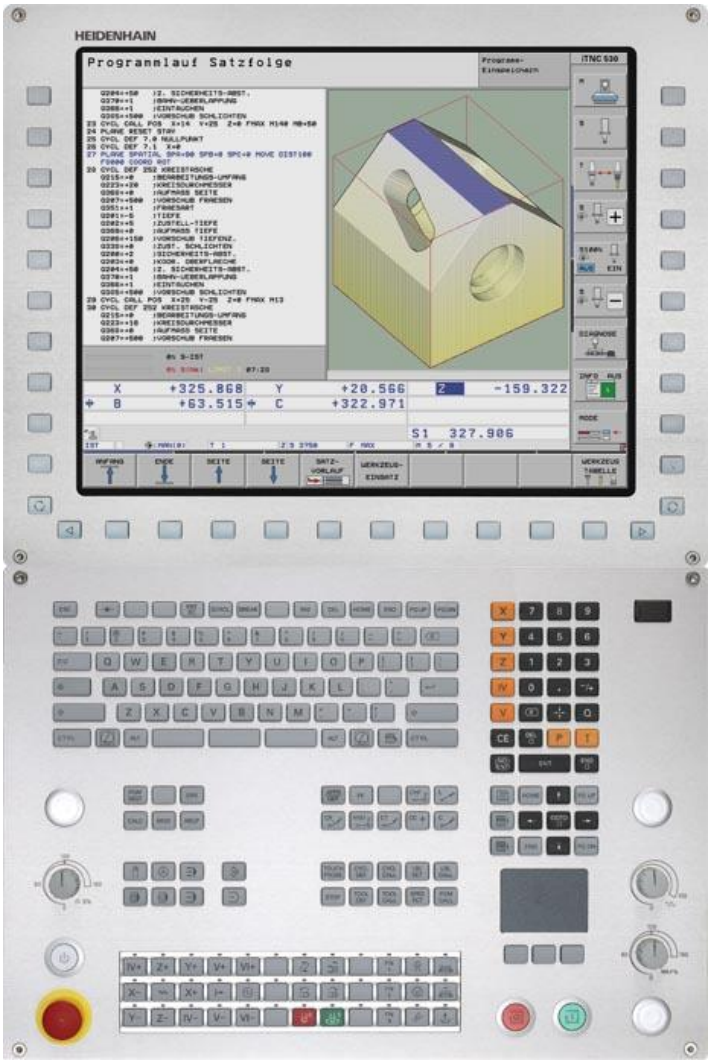



*Figura 2- 10: iTNC 530 de Heidenhain. Heidenhain (2011) iTNC 530 Technical Manual.*

El iTNC 530 cuenta con control de movimiento optimizado, cortos tiempos de procesamiento de bloques y estrategias especiales de control. Junto con su diseño digital uniforme y su control de accionamiento digital integrado que incluye inversores, le permite alcanzar velocidades de mecanizado muy altas y la mejor precisión de contorno posible, especialmente cuando se usan contornos 2-D o formas 3D.

El monitoreo dinámico de colisión DCM (opcional) del iTNC 530 monitorea cíclicamente el espacio de trabajo de la máquina para posibles colisiones con componentes de la máquina y accesorios. Con el control de alimentación adaptable de AFC (opcional), el iTNC regula la velocidad de alimentación automáticamente dependiendo únicamente del consumo de

energía del husillo y de otros datos de proceso respectivos. Esto optimiza el tiempo de mecanizado, soporta la supervisión de la herramienta y reduce el desgaste de la máquina.

El iTNC 530 cuenta con un software separado para el CN y el PLC. El software de CN se identifica con un número de ocho dígitos. El software del PLC se almacena en el disco duro del iTNC.

En el anexo 9-5 se detallan características más específicas del sistema de control iTNC 530. Entre estas se mencionan los controles más importantes, la interfaz del software, los parámetros de la máquina y como modificar los mismos. También se explica el uso del software TNCremo que sirve para conectar un centro de computación con la estación de control del iTNC 530

## 2.3 Python

Python, descrito por Van Rossum (septiembre 2009), es un lenguaje de programación fácil de aprender y potente. Tiene estructuras de datos eficientes de alto nivel y un enfoque simple pero efectivo para la programación orientada a objetos. La elegante sintaxis y la tipificación dinámica de Python, junto con su naturaleza interpretativa, lo convierten en un lenguaje ideal para el desarrollo de scripts y aplicaciones rápidas en muchas áreas en la mayoría de las plataformas.

Python es fácil de usar, además es un lenguaje de programación real, que ofrece mucho más estructura y soporte para programas de gran tamaño que el Shell. Por otro lado, también ofrece mucho más control de errores que C y, siendo un lenguaje de muy alto nivel, tiene tipos de datos de alto nivel incorporados, como array flexibles y diccionarios que le costaría días implementar de manera eficiente en C. Debido a sus tipos de datos más generales, Python es aplicable a un dominio de problema mucho mayor que Awk o incluso Perl, sin embargo, muchas cosas son al menos tan fáciles en Python como en esos idiomas.

Python permite dividir un programa en módulos que pueden ser reutilizados en otros programas de Python. Viene con una gran colección de módulos estándar que puede utilizar como base de sus programas o como ejemplos para comenzar a aprender a programar en Python.

Python es un lenguaje interpretado, que puede ahorrar tiempo considerable durante el desarrollo del programa porque no es necesario recopilar y vincular. El intérprete puede utilizarse de manera interactiva, lo que facilita la experimentación con las características del lenguaje, la escritura de programas de desechado o la comprobación de funciones durante el desarrollo del programa ascendente.

Los programas escritos en Python suelen ser mucho más cortos que los programas C o C ++ equivalentes, por varias razones:

- Los tipos de datos de alto nivel permiten expresar operaciones complejas en una única sentencia;
- El agrupamiento de instrucciones se realiza mediante sangría en lugar de los paréntesis de inicio y fin;
- No son necesarias declaraciones de variables o argumentos.

### 2.3.1 Python en Heidenhain

A partir del software 340 49x-04, la opción "Python OEM Process" (Opción de software # 46) le proporciona al iTNC una herramienta eficaz para usar un lenguaje de programación orientado de alto nivel en el control (PLC).

Los procesos de Python se pueden utilizar para funciones de máquina y cálculos, así como para mostrar información especial. Es especialmente útil para la implementación de soluciones específicas de cada usuario o de la máquina, independientemente de si se deben crear algoritmos o interfaces especiales para funciones especiales.

En la siguiente figura se muestra el diagrama de bloques que muestra las asociaciones básicas cuando se utiliza Python en los controles de Heidenhain.

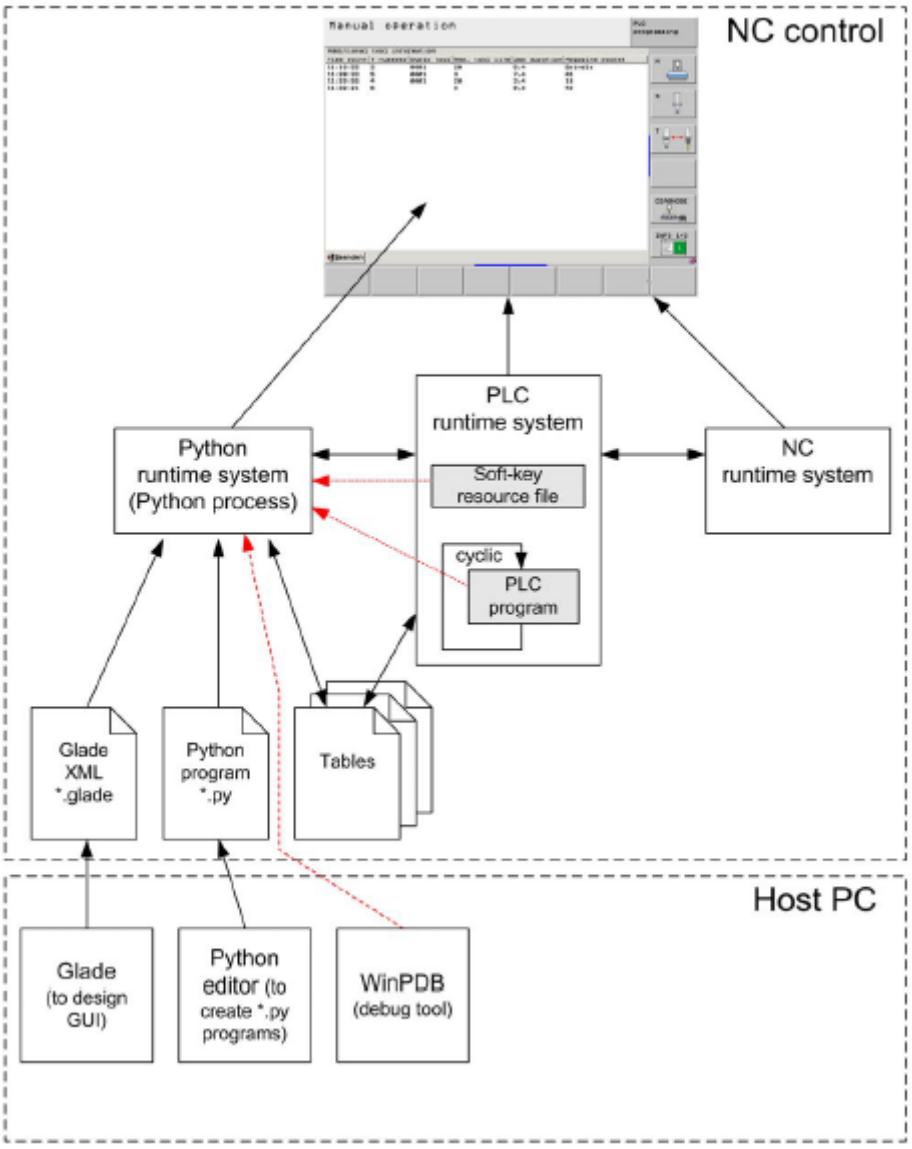



*Figura 2- 11: Diagrama de bloque para controles de Heidenhain. Extraído de (Python in Heidenhain).*

A continuación, se explican los bloques más importantes mostrados en la figura 2-11.

- **Editor de Python**: se debe utilizar un editor de Python (IDLE, PLCdesignNT) para crear los scripts de Python. Los scripts .py generados deben ser transferidos al control.

- **Glade**: con esta herramienta se puede crear la interfaz de usuario deseada arrastrando y colocando elementos en una interfaz gráfica. Glade genera un archivo XML que describe esta interfaz de usuario. Debido a la implementación de pyGTK, Python puede leer y mostrar archivos XML. Los scripts. glade deben ser transferidos al control.

- **Depurador (debug tool):** Heidenhain recomienda utilizar WinPDB como una herramienta de depuración. Puede usarlo para depurar los procesos de Python en el control.

- **Glade XML (*. glade):** Un archivo XML, generado por Glade, que describe la interfaz de usuario deseada y es llamado por un script Python. El archivo *. glade debe ser transferido al control con el script Python.

- **Proceso de Python:** Un proceso de Python se inicia a través del PLC o de una tecla de función (softkey). El nombre del proceso se asigna cuando se inicia este proceso de Python. El nombre del proceso puede diferir del nombre del script Python.

- **Tablas:** Tablas disponibles en el control. Los procesos de Python pueden utilizar las funciones del módulo PyJH para acceder a las tablas.

- **Sistema de tiempo de ejecución del PLC:** El programa PLC se procesa en el ciclo del PLC dentro del sistema de tiempo de ejecución del PLC. Los procesos Python se pueden iniciar, su estado se interroga y finaliza desde el programa PLC (Módulos PLC 9295, 9296, 9297). También se puede iniciar y finalizar un proceso Python mediante teclas programables (entrada correspondiente en el archivo de recursos de la tecla de función).

En el anexo 9-6 y anexo 9-7 se mencionan las librerías estándares de Python en el software del iTNC 530, como también una explicación de las funciones PyJH dadas por Heidenhain para ser usadas en lenguaje Python.

## 2.4 PLC

En esta sección se describe que es un controlador lógico programable (PLC), ya que el control de la máquina Kovosvit MAS 1000 usa uno para controlar todas sus variables y procesos. También se menciona el software de PLC design que se utiliza para la escritura del código de programación PLC. Por último, se mencionan los módulos a usar en este proyecto y la definición de “programas enviados” para utilizar estos módulos.

Los Controladores Lógicos Programables (PLC) son máquinas secuénciales que ejecutan correlativamente las instrucciones indicadas en el programa de usuario almacenado en su memoria, generando unas órdenes o señales de mando a partir de las señales de entrada leídas de la planta o sistema.

Al detectarse cambios en las señales, el autómata reacciona según el programa hasta obtener las órdenes de salida necesarias. Esta secuencia se ejecuta continuamente para conseguir el control actualizado del proceso. La secuencia básica de operación del autómata se puede dividir en tres fases principales:

- Lectura de señales desde la interfaz de entradas.
- Procesado del programa para obtención de las señales de control.

- Escritura de señales en la interfaz de salidas.

A fin de optimizar el tiempo, la lectura y escritura de las señales se realiza a la vez para todas las entradas y salidas. Entonces, las entradas leídas de los módulos de entrada se guardan en una memoria temporal. A esta acude la CPU en la ejecución del programa, y según se va obteniendo las salidas, se guardan en otra memoria temporal. Una vez ejecutado el programa completo, estas salidas se transfieren todas a la vez al módulo de salida.

El funcionamiento del Controlador Lógico Programable es, salvo el proceso inicial que sigue a un Reset, de tipo secuencial y cíclico, es decir, las operaciones tienen lugar una tras otra, y se van repitiendo continuamente mientras el autómata esté bajo tensión.

En el anexo 9-8 se muestra el paso a paso de como acceder al PLC desde la estación de programación o desde la estación de control del iTNC 530.

### 2.4.1 PLC Design Software

El PLC Design es un software de diseño de algoritmos para cualquier PLC con la opción de transferir los archivos creados al PLC que se va a utilizar en una aplicación deseada. Este software será usado en el proyecto para crear la rutina principal en el PLC que se encargará de controlar las variables y los programas a utilizar.

Ya que el PLC es el responsable de controlar todos los aspectos en la máquina herramienta a usar, es importante conocer de cuáles archivos está compuesto este y la función de sus principales archivos. En el software de diseño se pueden modificar y ver el contenido de los archivos más importantes dentro del PLC. En la figura 2-12 se puede observar el árbol de archivos que están contenidos dentro del programa.

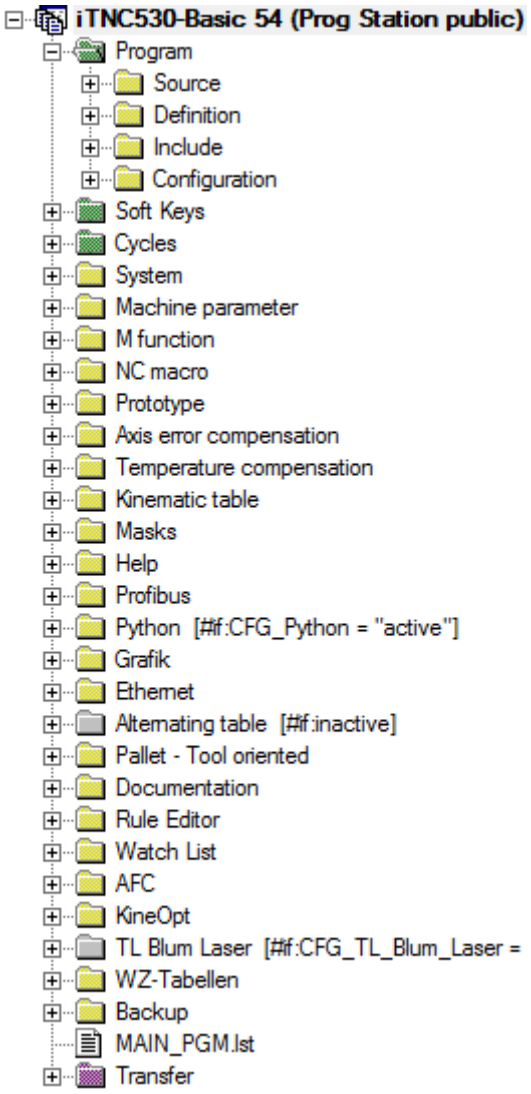



*Figura 2- 12: árbol de archivos en PLC Design Software. Extraída de PLC Design Software.*

Los archivos por destacar de esta imagen son aquellos contenidos en la carpeta de programa. Allí está el archivo de definiciones y de configuración que se encargan de definir las variables globales y activar o desactivar configuraciones del PLC respectivamente. También tenemos un archivo de 'Source' o fuente donde se encuentran todos los módulos creados para el PLC, allí se creará el módulo con la rutina a construir (Ver sección 3.5.2).

Otros archivos por destacar en la imagen 2-12, son NC macro y Transfer. En NC Macro o macros de CN es donde se guardarán todos los macros a utilizar en la máquina y se definirán las variables con las direcciones de cada uno de estos macros. En transfer se ubican todos los archivos que se desean transferir al PLC de la estación de programación. Se puede, si se desea, seleccionar todos los archivos, o en otros casos, solo aquellos modificados o aquellos deseados.

Esta herramienta de diseño también ayuda con la verificación del código a crear. Ya que tiene la opción de depurar el código y señalar donde y cuál es el error encontrado en caso de que exista alguno. Además de la depuración, el software puede conectarse con la estación de programación del iTNC 530 para observar a tiempo real los valores de variables deseadas por

medio de una ‘watchlist’ o lista de observación. En la siguiente figura se muestra la lista de observación.

| Operand | Module | Type | Address | Value |
|---|---|---|---|---|
| NP_M4176_control_in_operation... | <Global> | Marker | 4176 | 0 |
| NP_M4175_control_in_operation... | <Global> | Marker | 4175 | 0 |
| NP_M4153_mode_program_run_sin... | <Global> | Marker | 4153 | 0 |
| NP_M4154_mode_program_run_ful... | <Global> | Marker | 4154 | 0 |
| MG_key_NC_start | <Global> | Marker | 8032 | 0 |
| ML_python_finished | OEM_FUNC.SRC | Marker | 9338 | 0 |
| DL_line_read | OEM_FUNC.SRC | DWord | 8760 | 0 |
| BL_identifier2 | OEM_FUNC.SRC | Byte | 8740 | 0 |
| DL_message_9297 | OEM_FUNC.SRC | DWord | 8744 | 0 |

*Figura 2- 13: Lista de observación. Extraída de PLC Design Software.*

En la figura 2-13 se puede observar que la lista de observación no solo muestra el valor de las variables a visualizar, sino que también provee más información de las mismas. La segunda columna nos dice si la variable es global o local, y si es local nos da el nombre del módulo donde se encuentra definida. La tercera columna da la información del tipo de cada variable, ya sea marca, constante, byte, palabra, o doble palabra. Y la cuarta columna nos da la dirección en la memoria de cada una de estas variables.

Por último, esta herramienta de diseño también cuenta con la opción de visualizar que partes del código están activas por medio de la función de rastrear en la fuente (Trace in Source). Con estas opciones se facilita el funcionamiento de la rutina creada y de los resultados obtenidos por medio de esta. En la siguiente figura se muestra esta opción en funcionamiento.

```
112          1          1          L      MG_SK_program_enable
113          0          0          A      MG_SK_program
114                                ;AN     NP_M4203_error_module_9xxx
115          0          0          AN     M4203
116          0          0          AN     ML_program_start_request
117          0          0          S      ML_SK_program_pressed
118                                ;--------------------------------------
119          0          0          L      ML_SK_program_pressed
120          0          0          S      MG_SK_program
121          0          0          S      ML_program_start_request
122          0          0          R      ML_SK_program_pressed
123
124
125          0          0            L      ML_program_start_request
126                                  IFT
127                                      RPLY   BL_Macro_start
128          - $--------                  <>     K+0
```

*Figura 2- 14: función rastrear en fuente. Extraída de PLC Design Software.*

En la figura observamos una parte del código a la derecha de la pantalla, y a la izquierda podemos ver el valor de cada variable que la función está leyendo. Si la opción está en verde significa que el PLC está leyendo esa variable en ese momento. Por ejemplo, los bloques de comando ‘LOAD (L)’ o cargar siempre están siendo leídos por el PLC si no existe un

condicional que lo evite. Si el rastreo está en amarillo significa que el PLC no está leyendo esa variable ya que no tiene acceso a ella en ese momento. Esta herramienta es de gran uso para visualizar cuando y como el PLC entra a cada parte de su rutina.

El software de diseño del PLC es una herramienta de gran poder y gran capacidad. Muchas de sus funciones no han sido expuestas en esta sección, sin embargo, se han comentado los archivos a usar y las herramientas de depuración necesarias para la verificación del código a crear. En la siguiente sección se hablará de los comandos que se utilizan para programar en código PLC y el significado de cada uno de ellos.

En el anexo 9-8 se presentan los operandos disponibles en este software, como también el conjunto de comandos disponibles y ejemplos de los más importantes entre ellos.

### 2.4.2 Módulos a usar en este proyecto

La lista de todos los módulos disponibles en el PLC se encuentra en el manual técnico del iTNC 530. Los módulos que trabajan con programas en código Python son descritos en el manual de Python in Heidenhain- Controls, en la sección 3. En este documento solo se describirán los módulos necesarios para la rutina de PLC creada durante el desarrollo de la solución (sección 3.5).

A continuación, se mencionan los módulos usados durante este proyecto, y en el anexo 9-9 se describe cada uno de estos módulos.

- Módulo 9291: Iniciar un macro de CN
- Módulo 9297: Consultar el estado de un programa de Python
- Módulo 9296: Mandar una señal a un programa de Python (Terminar o matar)
- Módulo 9295: Iniciar un proceso de Python
- Módulo 9240: Abrir un documento
- Módulo 9243: Leer línea por línea un documento
- Módulo 9241: Cerrar un documento

### 2.4.3 'Programas de envíos'

Los programas de envío son sub-rutinas que el PLC envía al NC para tratamiento. Esto le permite resolver problemas que son muy intensivos al procesador, requieren bucles de programa o deben esperar resultados externos. Es un requisito previo que estos programas no se limiten a un marco de tiempo definido. Dependiendo de la utilización del procesador, el

iTNC proporciona una cierta potencia de cálculo para un programa de envío. Comienza a enviar programas desde el programa PLC.

Estos pueden acceder a todas las áreas de memoria de datos (M / B / W / D) como puede hacer el programa principal. Para evitar problemas, se debe asegurar de que los datos procesados por el programa PLC estén claramente separados de los datos procesados por el programa de envío. Se puede colocar hasta ocho programas de envío en una cola (cola de envío). Cada programa de envío recibe un "identificador" (un número entre 1 y 255, asignado por el NC), que el iTNC entra en el acumulador de palabra. Con este identificador y la función REPLY puede interrogar si el programa está en la cola, está siendo procesado o ya ha sido procesado. El iTNC procesa los programas de envío en la secuencia en la que fueron introducidos en la cola.

#### *2.4.2.1 Llamar a un programa de envío (SUBM)*

Para llamar a un programa de envío se debe asignar un identificador (1 a 255) a una subrutina marcada y colocarlo en la cola. Al mismo tiempo, el iTNC escribe el número asignado en la palabra acumulador. Si ya se han introducido programas en la cola de envío, el iTNC no ejecuta el programa direccionado hasta que se terminen los programas anteriores. Una presentación a la cola sólo puede tener lugar desde un programa de PLC. Un comando SUBM en un programa de envío no es posible.

Si no hay espacio en la cola, o si programa el comando SUBM en un programa de envío (anidamiento), el iTNC asigna el valor "0" a la palabra acumulador.

#### *2.4.2.2 Interrogar el estado de un programa de envío (RPLY)*

Para interrogar el estado del programa de envío se debe usar el identificador especificado. Se debe haber almacenado el identificador en un byte o una palabra al llamar al programa de envío. Con el comando RPLY y la dirección de memoria definida (byte o palabra que contiene el identificador), el iTNC transfiere uno de los siguientes estados de proceso a la palabra acumulador:

- Acumulador de palabra = 0: Programa completo/no en cola
- Acumulador de palabra = 1: El programa está corriendo
- Acumulador de palabra = 2: El programa está en cola

## 2.5 Conceptos matemáticos

En esta sección se explora la teoría detrás de algunos conceptos matemáticos usados en el desarrollo de la solución para este proyecto. Cada uno de estos conceptos es mencionado en

algunas de las subsecciones de la sección 3 de este documento. Aquí podemos encontrar de donde provienen las fórmulas que se han usado para obtener soluciones al problema planteado en la sección 3.

### 2.5.1 Regresión

Solís (2014) describe la regresión como una forma de aproximar y predecir datos de un fenómeno a partir del estudio de datos existentes (históricos). Sin embargo, a diferencia de la interpolación polinomial, las gráficas de las funciones de regresión no contienen, necesariamente, los datos conocidos.

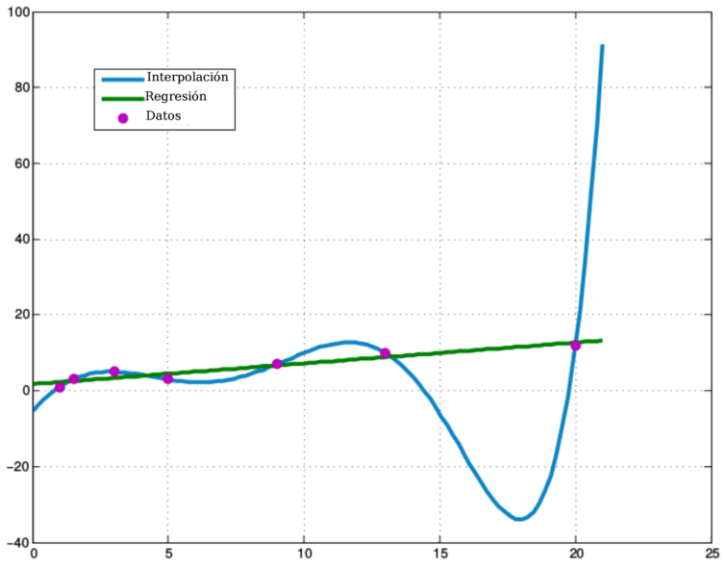



*Figura 2- 15: Gráfica para ejemplificar diferencia entre interpolación y regresión. Solís, R (s.f) Interpolación numérica.*

#### *2.5.1.1 Regresión por el método de mínimo cuadrados*

El método de mínimos cuadrados se utiliza dado un conjunto de datos (xi, yi) para i = 1, 2, ..., n. Se quiere determinar la recta que mejor se ajuste a los datos dados, así la recta tiene la forma:

$$y = \ mx + b \tag{2-3}$$

Como no siempre es posible encontrar una recta que contenga a todos los datos del conjunto, el error absoluto en cada uno de esos puntos es:

$$|mx_i + b - y_i| \tag{2-4}$$

Así el error absoluto total de todos los datos dados es:

$$\sum_{i=1}^{n} |mx_i + b - y_i| \tag{2-5}$$

Así, se podrán buscar los valores de m y b de manera tal que se minimice el error anterior, para evitar el problema de que la función dada no es diferenciable, se usará como referencia la siguiente función:

$$E(m,b) = \sum_{i=1}^{n}(mx_i + b - y_i)^2 \tag{2-6}$$

Derivando parcialmente con respecto a las variables m y b se tiene que:

$$\frac{dE}{dm} = \sum_{i=1}^{n} 2(mx_i + b - y_i)x_i \tag{2-7}$$

$$\frac{dE}{db} = \sum_{i=1}^{n} 2(mx_i + b - y_i) \tag{2-8}$$

Para determinar los puntos críticos de la función E, se debe resolver el sistema de ecuaciones:

$$\frac{dE}{dm} = 0 \tag{2-9}$$

$$\frac{dE}{db} = 0 \tag{2-10}$$

Así podemos expresar el sistema como:

$$m\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i \tag{2-11}$$

$$m\sum_{i=1}^{n} x_i + nb = \sum_{i=1}^{n} y_i \tag{2-12}$$

Resolviendo el sistema anterior obtenemos los valores que estamos buscando:

$$m = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \tag{2-13}$$

$$b = \frac{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \tag{2-14}$$

El sistema adquirido puede ser escrito utilizando matrices:

$$\begin{bmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \end{bmatrix} = \begin{bmatrix} n & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i{}^2 \end{bmatrix} \begin{bmatrix} b \\ m \end{bmatrix} \tag{2-15}$$

Este sistema se puede construir usando estas matrices:

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \tag{2-16}$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{2-17}$$

$$\beta = \begin{bmatrix} b \\ m \end{bmatrix} \tag{2-18}$$

Entonces, el sistema tiene la forma:

$$X'Y = X'X\beta \tag{2-19}$$

Por lo tanto:

$$\beta = (X'X)^{-1}X'Y \tag{2-20}$$

### 2.5.2 Representación de punto flotante con estándar IEEE

Para la representación de los números Racionales existen dos métodos muy conocidos como el del punto fijo, y la representación en punto flotante.

En punto fijo el sistema usa palabras divididas en 3 campos: Signo, parte del número precedente al punto binario, parte posterior al punto binario. En este método existen varias desventajas, como por ejemplo que sólo se puede representar una pequeña cantidad de números con este método.

Con el sistema de Punto Flotante, se resuelve el problema del punto fijo, ya que amplía el rango. Los números que conocemos como escritos en notación científica o notación exponencial se asemejan a la notación de punto flotante.

Los parámetros que caracterizan el sistema de números flotantes de base diez, son:

- La base B
- La mantisa m, que representa a la parte fraccionaria del número
- El exponente E, que varía entre dos cotas: Emin ≤ E ≤ Emax
- La precisión t referida a la cantidad de dígitos d i donde: 0 ≤ di ≤ B-1

A comienzos de 1980, en base al esfuerzo de muchos científicos de la computación, como W. Kahan, un sistema de punto flotante fue desarrollado, el que mereció el seguimiento

de los primeros fabricantes de chips para la construcción de las PC, como Intel y Motorola. Ese sistema se transformó en el sistema de punto flotante IEEE que existe para números en binario y decimal.

El estándar IEEE tiene 3 requerimientos:

1) La representación de los números en punto flotante debe ser consistente en todas las máquinas que lo adopten.
2) La aritmética de redondeo debe ser correcta.
3) El tratamiento de casos excepcionales como la división por cero debe ser consistente.

Existen varios tipos de representaciones de punto flotante según IEEE. Las más comunes son de simple precisión y de doble precisión. En la figura 2-16 se observa un cuadro comparativo entre estas dos.

| Tipo | Tamaño | Signo | Mantisa(Frac.) | Exponente | Rango (≈) |
|---|---|---|---|---|---|
| Simple | 32 bits | 1 bit | 23+1 bits | 8 bits | $1.2*10^{-38}$ a $3.4*10^{+38}$ |
| Doble | 64 bits | 1 bit | 52+1 bits | 11 bits | $2.2*10^{-308}$ a $1.8*10^{+308}$ |

*Figura 2- 16: tabla comparativa entre punto flotante simple y punto flotante de doble precisión. Arnau, V (12 de noviembre del 2010). Aritmética en coma flotante*

# 3. Desarrollo de la Solución

En esta sección se presenta el desarrollo y la explicación de las labores realizadas para cumplir con los objetivos del proyecto. Las secciones se presentan en el orden de que se fueron realizando y cumpliendo, siguiendo el orden de los objetivos específicos planteados en la sección 1.2.2 de este documento.

Esta sección se divide en 7 subsecciones que se realizaron para obtener los resultados a los objetivos planteados. En las primeras dos secciones se comentan los archivos obtenidos de medición y un archivo de ajustes y entradas construido para ser modificado por el usuario dependiendo de las entradas deseadas. En la cuarta sección se desarrolla la solución implementada para calcular la masa total de las piezas móviles en un proceso. La siguiente sección trata sobre la implementación de esta solución en el sistema de control iTNC 530. En la quinta sección se crea la rutina de PLC que controla la activación de los macros a usar y del algoritmo Python. La sexta sección trata sobre la solución dada al cambio de parámetros dentro de la estación de programación. Y, por último, la séptima sección muestra la instalación del algoritmo y los resultados del mismo al ser probado en la máquina Kovosvit Mas MCV 1000.

## 3.1 Archivos de Calibración, Medición, Ajustes y Entradas.

El primer reto planteado para este proyecto es la lectura de la masa de las piezas en movimiento en un proceso dentro del centro de mecanizado. Ya que la máquina-herramienta no puede leer directamente la masa de todas sus piezas, se debió buscar una solución utilizando las variables que la máquina sí puede medir.

La máquina-herramienta cuenta con un proceso de medición que realiza pruebas en la pieza a procesar y genera un archivo de texto donde se puede leer la posición, velocidad, aceleración y torque del motor para un barrido de tiempo programado en la máquina. El reto consiste en realizar un programa que pueda interpretar dichos archivos y usar las variables para obtener la masa total del sistema en movimiento.

Primero se debe hacer una prueba de calibración ya que el torque cuenta con un desfase entre la corriente entregada por el motor y la posición del sistema. Este se debe calibrar antes de crear el archivo de medidas. Este archivo de calibración tiene la misma estructura que el de medición. Cabe resaltar que estos archivos no deben ser manipulados o modificados de ninguna manera por el usuario.

Más adelante en la 3.5.4.3 se explicará cómo se obtienen estos archivos de calibración y de medición, ya que el algoritmo final debe realizar las pruebas para generar dichos archivos. Sin embargo, al empezar este proyecto ya se contaba con estos archivos por lo tanto la primera tarea era realizar un script que pudiera interpretar estos archivos de texto.

Se debe tomar en cuenta que el código una vez implementado en la máquina no se va a poder configurar, o al menos no se desea que se puedan realizar cambios al código. Por lo tanto,

todos los valores de entrada necesarios para el cálculo de la masa deben estar externos al código, para poder realizar cambios en ellos si así se desea.

La solución que se planteó para este reto fue crear un archivo de texto con las entradas necesarias para el cálculo de masa, que fuera fácil de modificar por el usuario de la máquina-herramienta y que fuera fácil de entender. En la siguiente figura se muestra como se ve este archivo de texto finalizado.

```
Settings.

CHANNELID# = 2 : ACTUAL POSITION
CHANNELID# = 6 : ACTUAL VELOCITY
CHANNELID# = 19: ACTUAL ACCELERATION
CHANNELID# = 29: ACTUAL MOMENTUM

Inputs


Pos_conv = 0.001       ([system units of position] to [SI units of position])
Vel_conv = 1.66667e-5  ([system units of velocity] to [SI units of velocity])
Acc_conv = 1           ([system units of acceleration] to [SI units of acceleration])

Start_Cal = -480   ([system units of position] Start position of calibration measurement)
End_Cal = -120     ([system units of position] End position of calibration measurement)

Start_m = -450       ([system units of position] Start position of main measurement of inertia)
End_m = -150       ([system units of position] End position of main measurement of inertia)

A_lim = 0.22        ([system units of acceleration] Limit acceleration set on machine)


MinT = 0.2        (Minimum time interval of constant acceleration)
Margin = 0.2      (Estimation of relative part of acceleration signal where it is not constant yet)

Km_sys = 1.687   (Torque constant in control system)
Correction = 1    (Correction of torque constant)

Acc_uncert = 0.05  (Accepted uncertaity for reduced mass value)
Ratio =  523.5987756 (Ratio of change)
```

*Figura 3- 1: archivo de ajustes y entradas. Extraído de archivo de texto de elaboración propia.*

El archivo se divide en dos partes, una de ajustes y otra de entradas. Los ajustes se refieren a los canales de medición de la máquina, cada uno de estos se refiere a una variable específica a medir, y el usuario solo debe modificar el número del canal para que estos coincidan con los canales mostrados en los archivos de calibración y de medición.

La segunda parte son las entradas necesarias para obtener la masa de las partes en movimiento, estas son modificables por el usuario y se deben modificar solo para tener un valor más preciso de la masa si fuera necesario o si las unidades de los archivos de calibración y de medición son distintas a las unidades SI.

Se recomienda usar los valores mostrados en la figura 3, ya que fueron pensados para la máquina Kovosvit MAS MCV 1000. Sin embargo, si se tratara de una máquina distinta, de un proveedor distinto a Kovosvit, donde las unidades de los archivos de calibración y de medición

no estuvieran en unidades SI, se recomienda solo cambiar los valores de ‘conv’ que se encargan de convertir las unidades de los valores medidos a unidades SI.

La explicación de cada variable viene en inglés para que pueda ser interpretado por usuarios checos y además por un gran porcentaje del mundo industrial. Igualmente, en el anexo 9-10 se explica el significado de cada una de estas variables y los valores posibles para las mismas.

## 3.2 Lectura de archivos y creación de matrices de datos.

En esta sección se desarrolló el primer script en Python que luego sería parte del algoritmo final del proyecto. Este código tiene como función abrir tanto los archivos de calibración y de medición como también el archivo de ajustes y entradas, luego de abrirlos debe empezar la lectura en busca de los datos a guardar o de algún error que exista en alguno de los archivos.

A continuación, se presenta un diagrama de flujo en la figura 3-2, con la lógica que se usó para construir el algoritmo deseado.

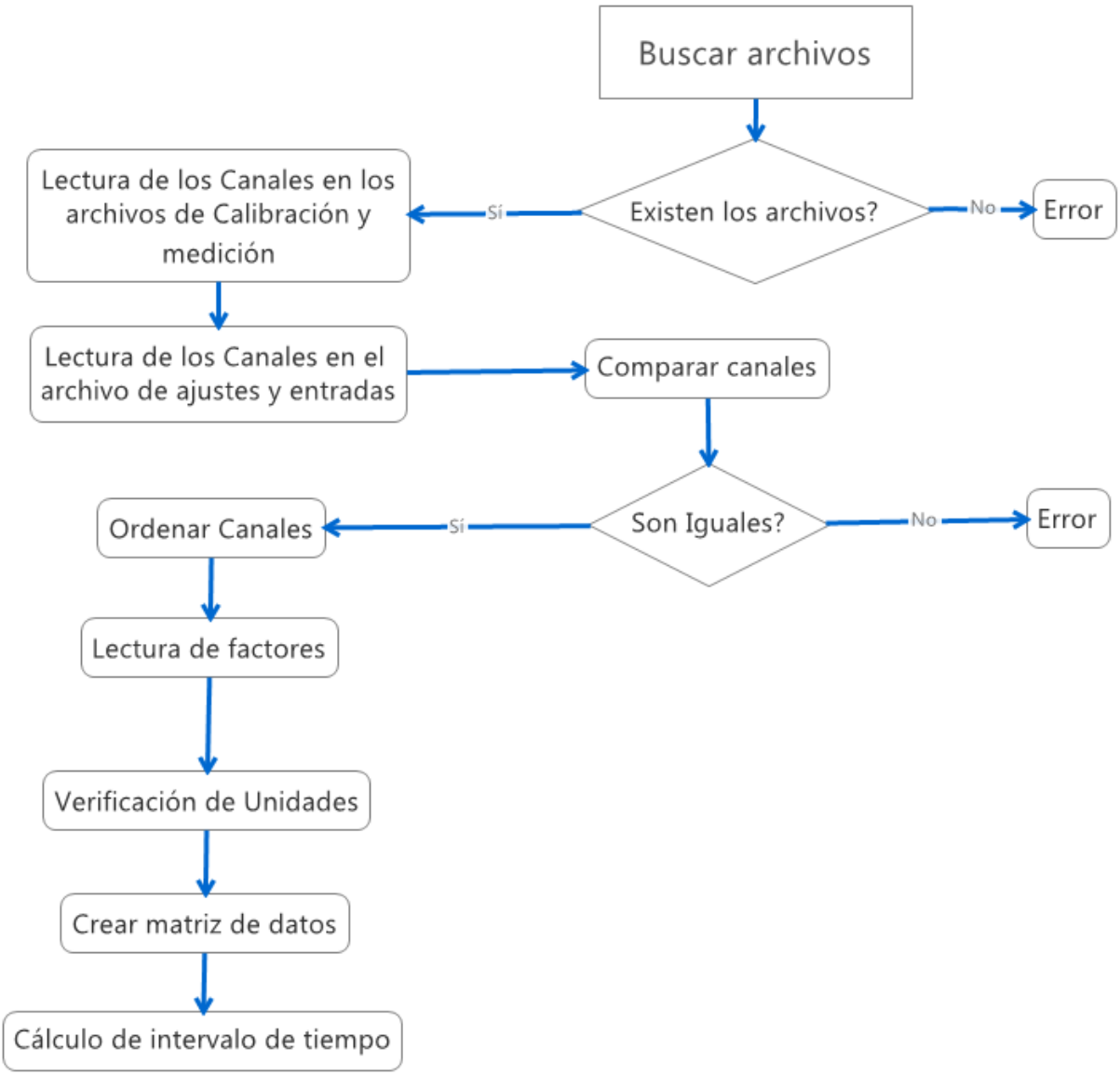



*Figura 3- 2: Diagrama de flujo para función de crear matriz de datos a partir de los archivos de calibración y medición. Desarrollado en Xmind 2012*

Luego de revisar si los archivos existen y pueden ser encontrados se empieza la lectura de canales. Primero se lee el número asociado a cada uno de los 5 canales en el archivo de calibración, estos son los mismos que el de medición ya que son dados por la misma máquina en el mismo eje. Luego se hace la lectura del número de cada canal dado por el usuario en el archivo de ajustes y entradas. Estos 5 canales deben tener el mismo número de ID en ambos archivos, consta destacar que no deben estar en el mismo orden, pero sí deben ser los mismos 5 números, en caso contrario se imprime un mensaje de error en pantalla.

Siguiendo el diagrama de flujo, una vez que se haya verificado que los ID de los canales son iguales se prosigue a ponerlos en el mismo orden. En esta etapa se hace la suposición de que los datos entrados por el usuario están en el orden correcto: posición, velocidad, aceleración y momento, tal y como vienen en el archivo de ajustes y entradas. Con estos IDs se hace una comparación con los IDs de los canales del archivo de calibración y se ordenan estos según lo que pidió el usuario. Ejemplo: si el usuario puso que el canal con ID =2 representa la velocidad

actual, el programa se encarga de buscar el canal con ID =2 en el archivo de calibración, sin importar en qué posición esté, ese canal va a representar los datos de velocidad actual.

Seguidamente cada canal tiene un factor por el cual tienen que ser multiplicados todos los datos tomados. Por lo tanto, se hace una lectura de cada factor y se le asigna al canal donde pertenezca dicho factor. Luego se verifican las unidades del archivo de calibración, y si no están en unidades SI se hace la conversión necesaria. En la figura 3-3 se observa parte de la información desplegada por el archivo de calibración, en ella podemos observar el ID de cada canal, el factor del mismo y las unidades de este.

```
CHANNELID1=2
CHANNEL1="s aktuální"
AXIS1="A"
AMPL1=1.000
DIM1="mm"
FACTOR1=1.0000000E-04

CHANNELID2=6
CHANNEL2="v aktuální"
AXIS2="A"
AMPL2=1.000
DIM2="mm/min"
FACTOR2=1.0728836E-01

CHANNELID3=19
CHANNEL3="a aktuální"
AXIS3="A"
AMPL3=1.000
DIM3="m/(s*s)"
FACTOR3=5.9604645E-04

CHANNELID4=29
CHANNEL4="M aktual"
AXIS4="A"
AMPL4=1.000
DIM4="Nm"
FACTOR4=1.0000000E-03
```

*Figura 3- 3: Información de canales del archivo de calibración. Extraída de archivo de texto de elaboración del RCMT.*

Una vez leída la información de cada canal, se prosigue a guardar los datos en una matriz. Esta matriz cuenta de cinco columnas, una para cada canal y una para el tiempo calculado al final del diagrama de flujo. Esta matriz se construye con todos los datos en el archivo de calibración, para esta máquina son 8192 datos por cada canal. El script se encarga de leer la cantidad de datos y de guardar en cada columna esa misma cantidad. También hay que tener en cuenta que cada dato desplegado viene acompañado con un: ‘0 0’ estos son datos basura desplegado en el archivo de texto y deben ser eliminados al momento de guardar cada dato.

Por último, se realiza el cálculo de los datos de tiempo, que no son desplegados en el archivo de calibración, sino que deben ser calculados. Para esto se debe usar el número de datos por el número de muestras y pasarlo a micro a segundos. El número de datos y el valor de la muestra son leídos en el archivo como se muestra en la figura 3-4.

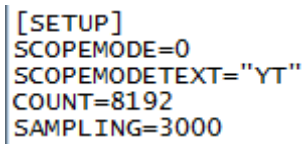



*Figura 3- 4: Numero de datos y valor de la muestra del archivo de calibración. Extraída de archivo de texto de elaboración del RCMT.*

Una vez obtenido este dato tiempo que representa el último valor en microsegundos se calculan todos los demás valores restándole el valor de muestra a cada valor, para que los valores de tiempo estén igualmente espaciados, y al final se tenga la misma cantidad de datos que los demás canales. Estos datos son agregados a una de las columnas de la matriz principal en orden ascendente. Con esto, toda la información necesaria dada por el archivo de calibración ya ha sido guardada y este archivo por el resto del proceso no toma mayor importancia.

## 3.3 Cálculo de masa

En esta sección se desarrollan los pasos necesarios para obtener el valor de la masa total de las piezas en movimiento a partir de los datos obtenidos en la sección 3.2.

Para obtener el valor de la masa, usaremos la segunda ley de Newton.

$$F = ma \tag{3-1}$$

Por lo tanto, debemos buscar el valor de la aceleración y de la fuerza en esta ecuación. En las siguientes secciones desarrollan los pasos para obtener dichos valores.

### 3.3.1 Calibración de corriente

Para calcular masa se necesitan los datos tomados del archivo de medición, sin embargo, antes de tomar esos datos se ha encontrado que en la prueba de medición realizada en la máquina los datos del torque no eran correctos. Ya que por variables físicas de la máquina siempre existe una pequeña dependencia entre la corriente eléctrica (torque del motor) y la posición de las partes móviles. Por lo tanto, se deben usar los datos tomados del archivo de calibración para encontrar esta dependencia y sustraerla de los datos de medición. Es por esta razón que existe una prueba de calibración y su respectivo archivo.

Para obtener los coeficientes de calibración se debe realizar regresión en los datos de corriente y de posición. Para obtener la corriente simplemente se transforma el torque usando la siguiente formula:

$$Corriente = -\frac{torque}{km_{sys}} \tag{3-2}$$

El valor de km_sys es un valor dado por el Sistema de control, y se ingresa en el archivo de ajustes y entradas.

Con los valores de corriente y posición se debe realizar regresión por medio del método de mínimos cuadrados (Ver sección 2.5). En la figura 3-5 se observa la gráfica de corriente vs posición y su debida línea de mejor ajuste.

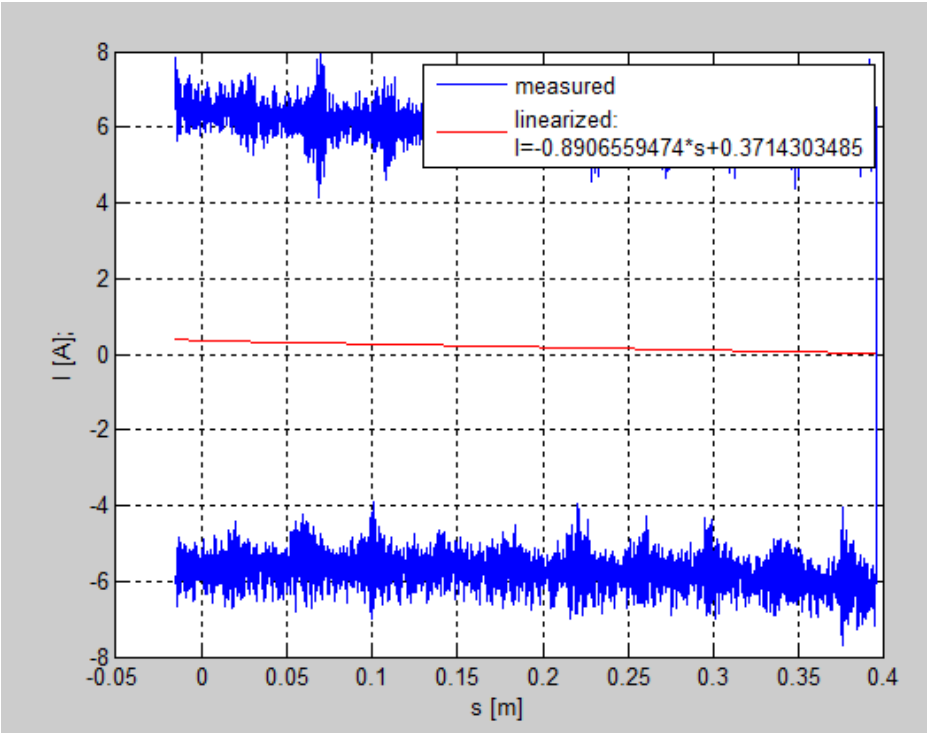



*Figura 3- 5: Gráfica de corriente eléctrica vs posición, y su respectiva regresión por mínimo cuadrados, datos tomados del archivo de calibración. Gráfica construida en Matlab.*

Podemos observar en la gráfica que los datos medidos de la corriente entregan una señal con mucho ruido, y además se repite en mayor medida tanto en el eje positive como en el negativo ya que la prueba mueve la pieza y las partes móviles tanto para una dirección como para la contraria. La fricción que se genera al tiempo que se realiza la prueba ocasiona la desviación, positiva en una dirección y negativa en la contraria, entre los datos medidos y los ideales que deberían estar centrados con valores de corriente aproximados a 0 A.

Con la regresión de los datos medidos en ambas direcciones ignoramos la fricción y obtenemos una recta de mejor ajuste con datos más cercanos al cero. Sin embargo, esta recta al no ser de pendiente cero genera una dependencia pequeña entre la corriente y la posición, por esto es por lo que debemos calibrar la corriente en el archivo de medición.

Los valores obtenidos son:

$$\beta 1 = 0.8906559474$$
$$\beta 2 = 0.3714303485$$

Donde β1 representa la pendiente de la recta, y β2 representa el punto de intersección con el eje Y.

Con estos valores proseguimos a corregir los valores de corriente en el archivo de medición. Para esto primer debemos convertir los valores de torque del archivo de medición a valores de corriente, de la misma manera que se hizo con los datos en el archivo de calibración.

Una vez obtenido estos valores se calibra la corriente substrayéndole la dependencia de corriente a la posición obtenida por medio de regresión en el paso anterior. Esto se realiza por media de la siguiente formula.

$$Current_{cal} = Current - (Position * \beta 1 + \beta 2) \quad (3\text{-}3)$$

Donde Current_cal es cada valor de corriente calibrada, Current es cada valor de corriente del archivo de medición, y Position es cada valor de posición del archivo de medición. Con esto se ha logrado calibrar los datos de corriente.

### 3.3.2 Graficas de velocidad, aceleración y corriente

Una vez obtenido los valores de corriente calibrada se prosigue a trabajar con los datos en el archivo de medición. En este archivo nos interesan los datos de velocidad, aceleración, corriente y tiempo.

En esta sección se muestran tres figuras representando la señal que muestran los datos en el tiempo, esto para dar una idea al lector de lo que se va a realizar en las siguientes secciones con los datos en estas gráficas.

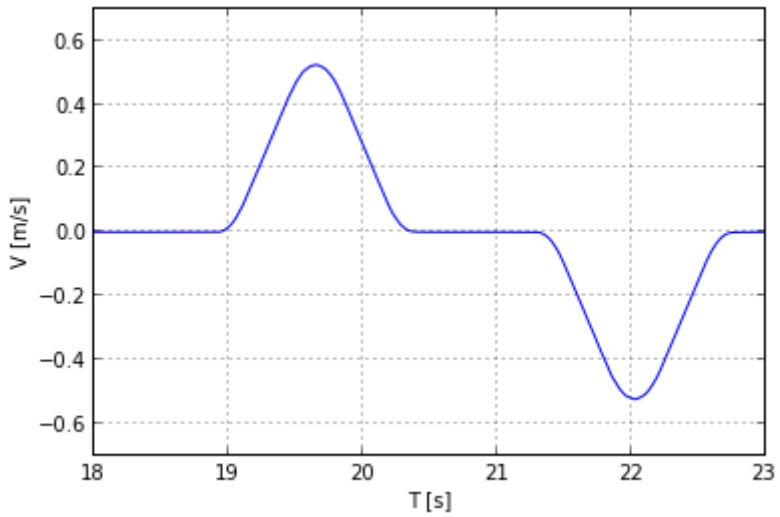



*Figura 3- 6: Gráfica de velocidad vs tiempo, datos tomados del archivo de medición. Gráfica construida en Python.*

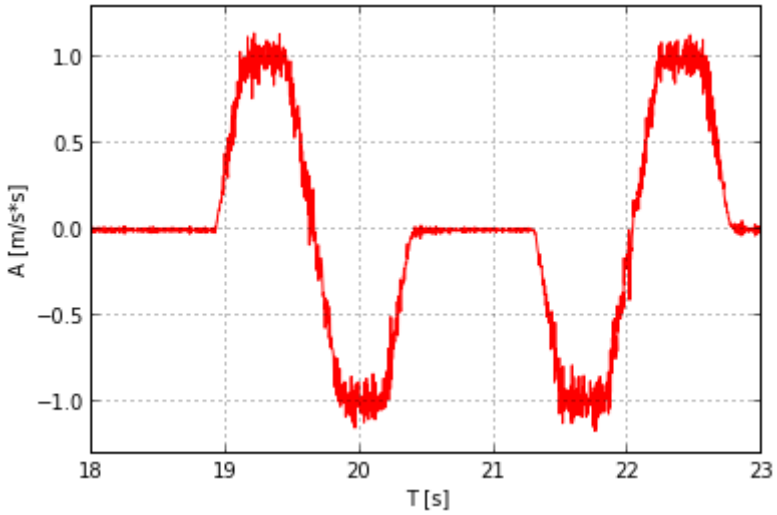



*Figura 3- 7: Gráfica de aceleración vs tiempo, datos tomados del archivo de medición. Gráfica construida en Python.*

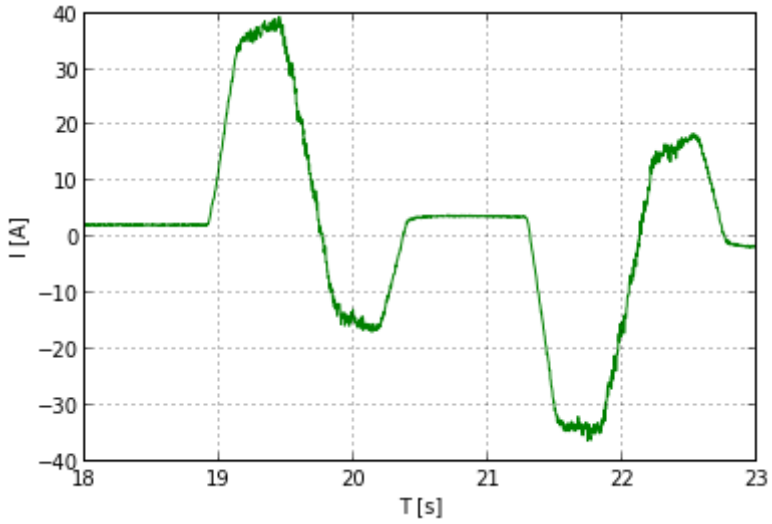



*Figura 3- 8: Gráfica de corriente eléctrica vs tiempo, datos tomados del archivo de medición. Gráfica construida en Python.*

### 3.3.3 Método para obtener el cambio de aceleración en un punto.

Para obtener el valor de la masa debemos encontrar los valores donde la aceleración hace un cambio abrupto. Una señal ideal sería tipo cuadrada, donde el punto donde se encuentre un cambio de aceleración sea fácil de percibir. Sin embargo, este no es el caso para la señal en la figura 3-7. Por lo tanto, debemos usar un método que aproxime un valor para encontrar donde ocurre el cambio de aceleración.

Observando la figura 3-6, la señal de velocidad tiene dos momentos donde los datos pasan de ser una señal creciente a una decreciente o viceversa. En estos 'máximos' hay un cambio fuerte de velocidad, causado por el cambio de aceleración abrupto que estamos buscando. Por lo tanto, si se encuentra un aproximado del punto donde la velocidad cambia su signo de pendiente, podemos encontrar el punto donde la aceleración, idealmente, realiza el cambio que buscamos.

Para realizar esto, se deben seguir 3 pasos, explicados en las siguientes secciones.

### *3.3.3.1 Identificación de intervalos de tiempo*

El primer paso por realizar es encontrar los intervalos donde la aceleración es constante, esto porque en esos intervalos la señal de velocidad crece o decrece de manera constante, y encontrando esos intervalos de velocidad es posible encontrar el punto donde se intersecan y ocurre el cambio de velocidad.

Por lo tanto, antes de encontrar esos intervalos, debemos verificar que la gráfica de aceleración cumpla con 4 momentos o intervalos donde la aceleración sea igual al valor de aceleración máxima dada por el usuario en el archivo de ajustes y entradas. Para esto se plantea un algoritmo en el siguiente diagrama de flujo.

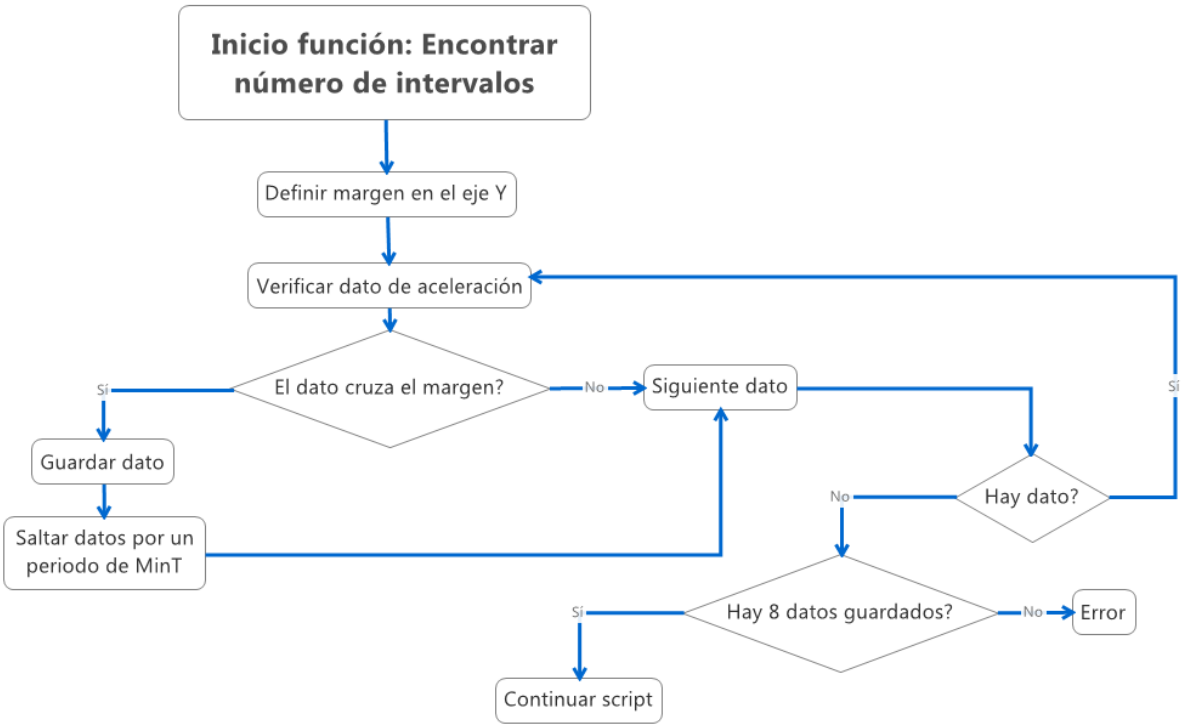



*Figura 3- 9: Diagrama de flujo de función: 'Encontrar número de intervalos.' Diagrama construido en Xmind 2012.*

Lo primero que hace la función es definir un margen donde se vayan a guardar los datos que lo crucen. Este margen se define como la mitad de la aceleración máxima (dato entrado por el usuario), se define tanto en el eje positive como en el eje negativo. En la figura 3-10 se muestran estos márgenes.

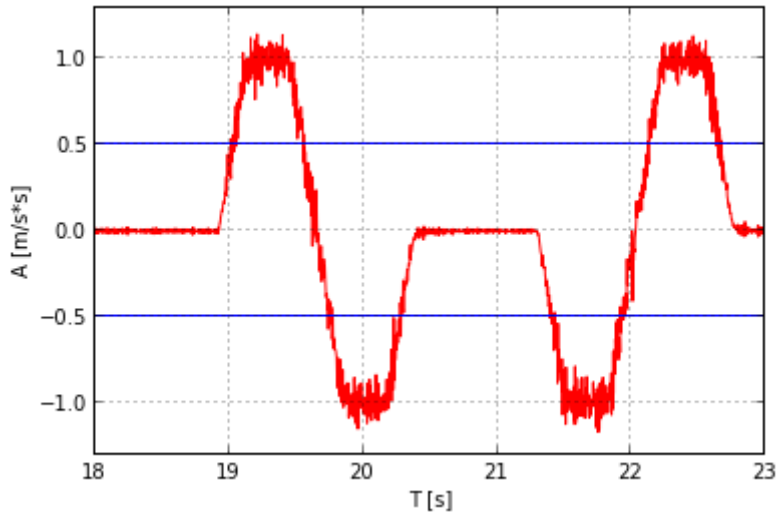



*Figura 3- 10: Gráfica de aceleración vs tiempo con márgenes para encontrar puntos de corte. Gráfica construida en Python.*

Se observa en la figura 3-10 que la señal cruza en 8 partes estos márgenes, pero por ser una señal sumamente ruidosa hay que tener cuidado en las zonas de cruce, ya que por el ruido en la señal habrá muchos cruces en un peque; o período de tiempo.

Luego, se empieza el barrido de los datos de aceleración. Por cada dato, se debe verificar si esta cruza el margen, si no es así se continúa al siguiente dato, pero si el dato cruza el margen se debe guardar ese dato y hacer un salto de un período de tiempo de MinT (dato ingresado por el usuario en el archivo de ajustes y entradas.) Este salto en el tiempo se hace para evitar que el ruido de la señal guarde más de un dato.

Una vez terminado el barrido de datos, se verifican que haya 8 datos guardados. Si no es así el programa finaliza y se imprime un mensaje de error. En la figura 3-11 se observa la gráfica con los 8 cruces por el margen.

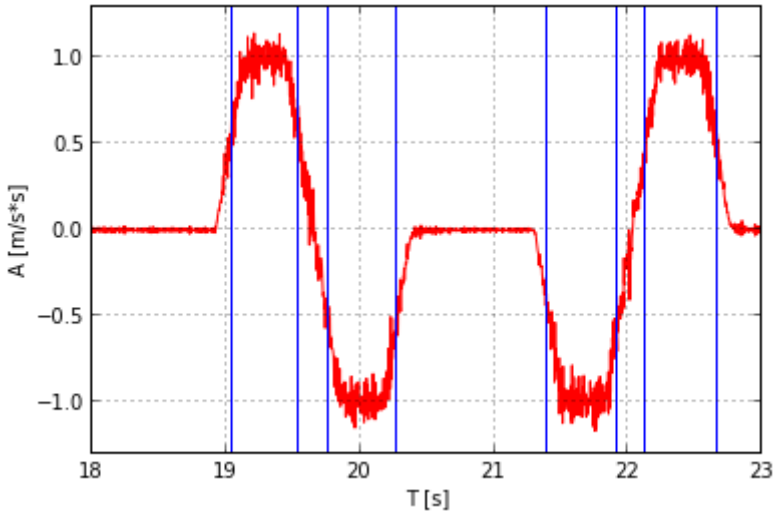



*Figura 3- 11: Gráfica de aceleración vs tiempo con los ocho puntos de corte identificados. Gráfica construida en Python*

En la figura 3-11 se pueden observar los ocho cruces, y que dentro de cada dos de ellos está contenido uno de los cuatro intervalos que estamos buscando. A estos ocho cruces, les llamaremos **puntos de corte** para las siguientes secciones.

### *3.3.3.2 Definición de intervalos*

El siguiente paso, es definir en el tiempo esos intervalos que buscamos. Para esto debemos encontrar los cuatro intervalos donde la aceleración es constante, estos cuatro intervalos están contenidos dentro de los ocho datos previamente hallados. Para esto se plantea el siguiente diagrama de flujo.

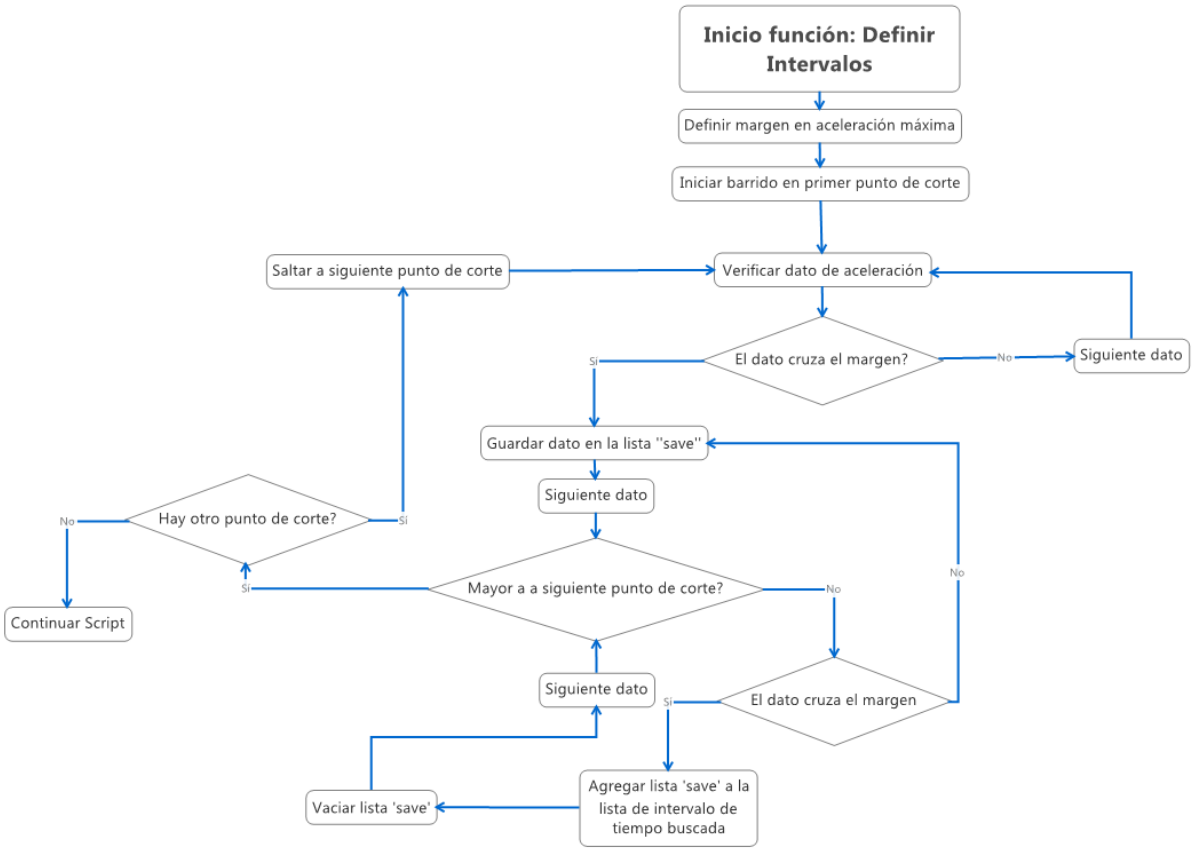



*Figura 3- 12: Diagrama de flujo de función: 'Definir intervalos'. Diagrama construido en Xmind 2012.*

Se inicia la función definiendo el margen de aceleración máxima, este se define con el dato de aceleración máxima ingresado por el usuario en el archivo de ajustes y entradas. Este margen trabaja para datos en valor absoluto, sin importar su signo.

Luego se inicia el barrido empezando en el primer punto de corte, este barrido finaliza una vez que se hayan verificado todos los datos entre el primer y segundo punto de corte, luego salta al tercero y repite el paso, luego al quinto y por último al séptimo, por lo tanto, al final de la función se habrán verificado todos los datos entre los intervalos deseados.

Para una señal ideal, la lógica sería que cuando algún dato cruzara el margen, se guardara en una lista todos los datos hasta que un dato volviera a cruzar el margen. Así, se tendría el intervalo donde la aceleración es constante, sin embargo, como tenemos una señal ruidosa tenemos que usar otra lógica, como la que se presenta en el diagrama de flujo de la figura 3-12.

La lógica se basa en una lista vacía llamada 'save', una vez que el primer dato cruza el margen este se guarda en la lista 'save' y los siguientes datos se guardan en la misma lista hasta que alguno vuelva a cruzar el margen o hasta que se alcance el siguiente punto de cruce, que da por finalizado ese intervalo. Si algún dato cruza el margen, todos los datos de la lista 'save' se agregan a la lista de intervalos de tiempo (los que estamos buscando), se vacía la lista 'save' y se empiezan a guardar nuevos datos en esta lista. Si se llega al siguiente punto de cruce se salta al próximo punto de cruce y se empieza el barrido de nuevo, esto se hace cuatro veces, para así obtener los 4 intervalos de tiempo deseados.

Una vez obtenidos estos 4 intervalos se debe substraer un porcentaje de los datos, ya que se usa el 100% de los datos, al momento de pasar el intervalo a la gráfica de velocidad, puede haber datos que no sean parte de las rectas crecientes o decrecientes, y esto generaría erros en los siguientes pasos. Por lo tanto, se substrae de cada intervalo un porcentaje dado por el usuario como 'Margin' en el archivo de ajustes y entradas. Este porcentaje de substrae dos veces por cada intervalo, substrayendo los datos más pequeños y más grandes en el tiempo.

En la figura 3-13 se muestra cómo se ven estos intervalos una vez que se les haya sustraído el margen.

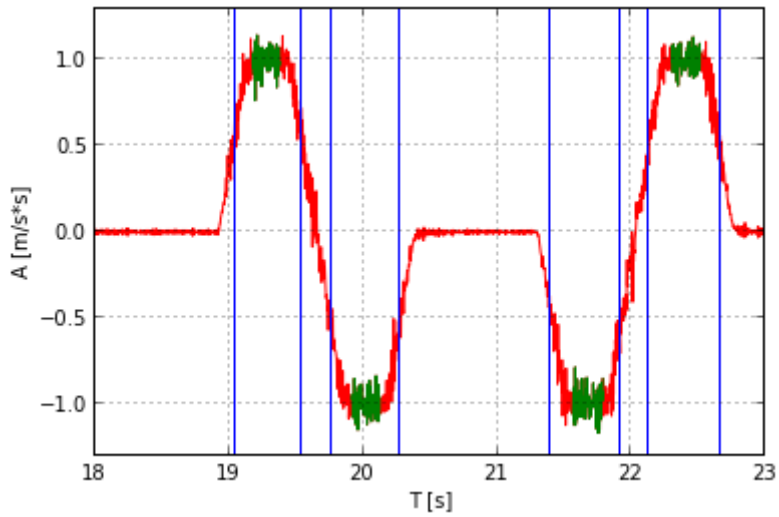



*Figura 3- 13: Gráfica de aceleración vs tiempo con los intervalos de aceleración deseados. Gráfica construida en Python.*

Por siguiente se debe usar esos períodos de tiempo obtenidos para obtener los intervalos de velocidad y de corriente buscados. En las figuras 3-14 y 3-15, se muestran las gráficas junto a sus intervalos definidos.

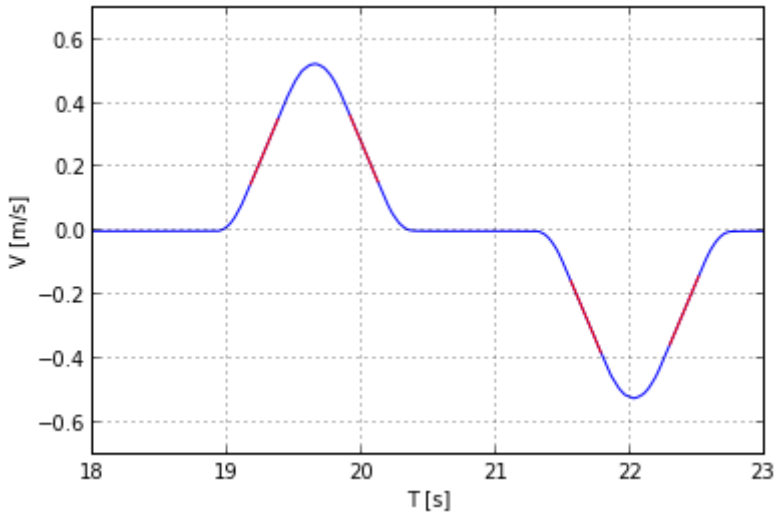



*Figura 3- 14: Gráfica de velocidad vs tiempo con los intervalos de velocidad obtenidos a partir de la gráfica de aceleración. Gráfica construida en Python.*

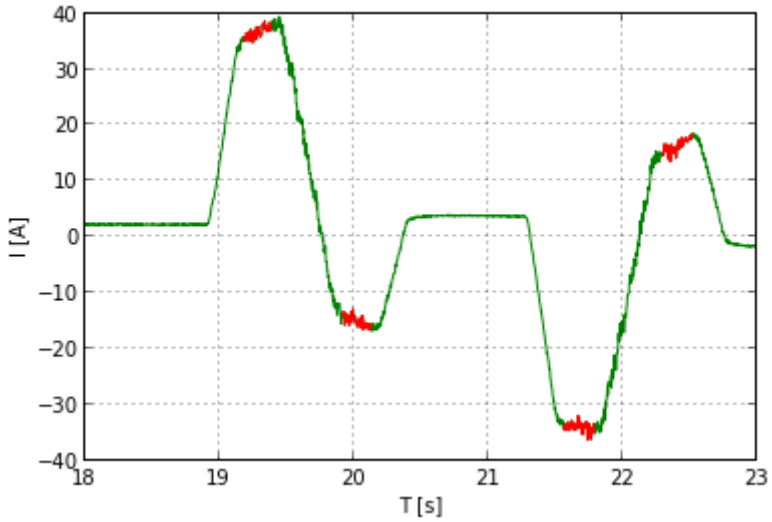



*Figura 3- 15: Gráfica de corriente eléctrica vs tiempo con los intervalos de corriente eléctrica obtenidos a partir de la gráfica de aceleración. Gráfica construida en Python.*

### *3.3.3.3 Regresión para dos intervalos*

El siguiente paso por realizar es hacer regresión por mínimos cuadrados para cada dos intervalos de velocidad y para cada dos intervalos de corriente obtenidos en el punto anterior. Pero la regresión para cada intervalo no se puede hacer individual ya que ocupamos que ambas regresiones compartan la misma pendiente, pero con signo opuesto.

Vamos a suponer las siguientes matrices:

$$X = \begin{bmatrix} X1 & 1 & 0 \\ \dots & \dots & \dots \\ -X2 & 0 & 1 \end{bmatrix},\ Y = \begin{bmatrix} Y1 \\ \dots \\ Y2 \end{bmatrix},\ \beta = \begin{bmatrix} k \\ b1 \\ b2 \end{bmatrix} \qquad \text{(3-4), (3-5), (3-6)}$$

Estas matrices son muy similares aquellas construidas en la sección 2.5. Sin embargo, vemos que β tiene una variable más que representa el corte con el eje y del segundo intervalo de datos. Y y X tienen los datos tanto del primer intervalo como del segundo. En X hay una columna de 1 y otra de 0 para el primer intervalo y para el segundo intervalo estas columnas se alteran a 0 y a 1, ya que b1 solo está presente para el primer intervalo, mientras que para el segundo solo está presente b2. También los datos de X1 y X2 son de signo opuesto ya que el valor de pendiente para cada intervalo es opuesto también.

A partir de estas matrices montamos la ecuación para obtener regresión por medio de mínimos cuadrados. Nos basamos en la ecuación 2-20.

$$X'Y = X'X\beta \tag{3-7}$$

Por lo tanto, para este caso tenemos:

$$[X^T X]\begin{bmatrix} k \\ b1 \\ b2 \end{bmatrix} = [X^T Y] \tag{3-8}$$

Sustituyendo los valores:

$$\begin{bmatrix} \sum_{i=1}^{n1} X1_i^2 + \sum_{i=1}^{n2} X2_i^2 & \sum_{i=1}^{n1} X1_i & -\sum_{i=1}^{n2} X2_i \\ \sum_{i=1}^{n1} X1_i & -n1 & 0 \\ -\sum_{i=1}^{n2} X2_i & 0 & n2 \end{bmatrix}\begin{bmatrix} k \\ b1 \\ b2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n1} X1_i Y1_i - \sum_{i=1}^{n2} X2_i Y2_i \\ \sum_{i=1}^{n1} Y1_i \\ \sum_{i=1}^{n2} Y2_i \end{bmatrix} \tag{3-9}$$

Con esta ecuación de matrices podemos obtener los resultados para los dos pares de intervalos de velocidad y los dos pares de intervalos de corriente.

### 3.3.4 Cálculo de la masa

El primer paso por realizar es calcular el punto de intersección de los intervalos de velocidad, este es el punto donde la aceleración cambia. En este caso los valores de k, b1 y b2 provienen de realizar regresión con el intervalo de velocidad. Para esto usamos la siguiente ecuación:

$$pr = \frac{(b2-b1)}{2k} \tag{3-10}$$

El siguiente paso es realizar regresión para los intervalos de corriente. Con estos valores calculamos el cambio de corriente en el cambio de aceleración. Usamos la siguiente fórmula.

$$dI = (b1 + pr \times k) - (b2 - pr \times k) \tag{3-11}$$

Estos cálculos deben ser realizados para ambos pares de intervalos de corriente. Teniendo ambos valores de cambio de corriente dI, podemos calcular el cambio de corriente promedio en el sistema total.

$$dI = \frac{dI_{primero} + dI_{segundo}}{4} \tag{3-12}$$

Donde dI_primero es el valor de cambio de corriente para el primer par de intervalos de corriente en valor absoluto, y dI_segundo es el valor de cambio de corriente para el segundo par de intervalos de corriente en valor absoluto.

Con este valor de corriente y el valor de la constante de torque en el sistema de control (ver sección 3.2) podemos calcular el cambio de momento en el sistema, con la siguiente ecuación:

$$dM = km_{sys} * dI \tag{3-13}$$

El siguiente paso es calcular el cambio de fuerza en el sistema, para esto ocupamos el valor de proporción entre la velocidad lineal y la velocidad angular. Este valor se puede variar en el archivo de ajustes y entradas, y depende de cada eje y de cada máquina. Para este caso tenemos:

$$p = \frac{2\pi}{0.012} \tag{3-14}$$

Por lo tanto, el cambio de fuerza es:

$$dF = dM \times p \tag{3-15}$$

El último dato que ocupamos es la aceleración interpolada promedio del sistema. Para esto usamos k1 como el valor de la pendiente de la regresión del primer par de intervalos de velocidad, y k2 como el valor de la pendiente de la regresión del segundo par de intervalos de velocidad. La aceleración promedio se calcula con la siguiente ecuación:

$$a = \frac{(k1 + k2)}{2} \tag{3-16}$$

Para obtener la masa, usamos la segunda ley de Newton con los datos obtenidos previamente.

$$m = \frac{dF}{a} \tag{3-17}$$

En la siguiente figura se imprime el dato de masa total del sistema en movimiento obtenido a partir del ejemplo que se ha seguido durante esta sección.

```
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)]
32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
MASA
22786.4534097
```

*Figura 3- 16: Pantallazo de resultado de algoritmo de cálculo de masa. Resultado de algoritmo construido en Python.*

## 3.4 Implementación de cálculo de masa en el Sistema de control Heidenhain iTNC 530

El siguiente reto de este trabajo es mostrar los resultados de la sección 3.3 en el Sistema de control iTNC 530 de Heidenhain. Para esto se usa la estación de programación en el computador que es una simulación del sistema de control en la máquina.

En esta sección se muestra el resultado obtenido de la sección 3.3 de cálculo de masa implementado en la estación de programación del iTNC 530. Para esto se debe crear una "ventana" con el código Python y las funciones PyJH mencionadas en el anexo 9-7.

Cabe destacar que para la implementación de dicho cálculo en la estación de programación se deben llevar a cabo una serie de pasos. Sin embargo, esos pasos son muy metódicos y no tienen gran peso en el desarrollo de la solución. Por lo tanto, esos pasos se explican en el anexo 9-11.

### 3.4.1 Creación de 'ventana' para visualizar resultados

El siguiente paso es imprimir en pantalla el resultado de la masa para verificar que el script corra de manera correcta. Para esto se deben usar las funciones PyJH mencionadas en el anexo 9-7. El código se puede editar desde la estación de programación, pero es de mayor facilidad editar el script en el IDLE de Python, aunque no se pueda probar desde ahí ya que estas funciones son estrictamente para procesos de Python en sistemas de Heidenhain.

Para crear la ventana se debe usar la función jh.gtk. Window. Esta función crea una ventana que se muestra en el contexto gráfico del control de acuerdo con el parámetro de uso. La posición y el tamaño corresponden a la descripción del archivo de disposición. La ventana se vuelve visible cuando se selecciona el diseño correspondiente. El registro de la ventana en el contexto de control se produce automáticamente.

La ventana debe ser destruida antes de terminar el proceso por lo tanto también es necesaria la función jh.gtk.destroy. El método destroy () destruye la instancia de jh.gtk.Window. La ventana se anula automáticamente cuando se destruye.

También hay que tomar en consideración que al usar funciones gtk se debe usar dos funciones extras, una para empezar el programa y otra para terminarlo. Estas son jh.Main y jh. QuitMain. La función jh.Main () constituye el ciclo de eventos del script de Python. Aquí se procesan los eventos GTK + y de control, y se llaman las funciones correspondientes. La función jh.QuitMain () se encarga de terminar este ciclo de eventos y finalizar el programa.

Para cerrar la ventana de manera manual, se debe conectar la ventana a la función destroy. Para esto se usa la función connect. Una vez conectadas solo hace falta convertir la ventana en un botón para que sea destruido al presionarlo. Para esto usamos la función gtk. Button, y este botón también debe conectarse a la ventana de la misma manera que la función destroy. En las siguientes figuras se muestra el código en Python para crear la ventana asignarle un botón y destruir la ventana.

```
##create the window

myWindow = jh.gtk.Window( setTransient = True )
myWindow.set_title('Mass')


## connect the destroy event with a function
myWindow.connect( 'destroy', on_main_window_destroy )


## resize and move the window
myWindow.resize( 300, 200 )
myWindow.move( 350, 350 )
```

*Figura 3- 17: Código en lenguaje Python para crear una ventana en la estación de programación del iTNC 530. Código construido en Python.*

```
button=gtk.Button('VALOR DE MASA')
button.connect( 'clicked', on_main_window_destroy )

## add the button into the window
myWindow.add( button )

## show the window
myWindow.show_all( )
```

*Figura 3- 18: Código en lenguaje Python para crear un botón en la estación de programación del iTNC 530. Código construido en Python*

```
def on_main_window_destroy( widget ):
    """
#This method runs when the user closes the window.
#"""
 stop the main loop
    jh.MainQuit( )
```

*Figura 3- 19: Código en lenguaje Python para destruir una ventana y terminar un proceso de Python en la estación de programación del iTNC 530. Código construido en Python.*

Una vez modificado el código se vuelve a guardar usando el TNCremo y depura para revisar que los cambios estén correctos. Luego se corre usando la softkey WORLD, y en la figura 3-29 se pueden observar los resultados obtenidos.

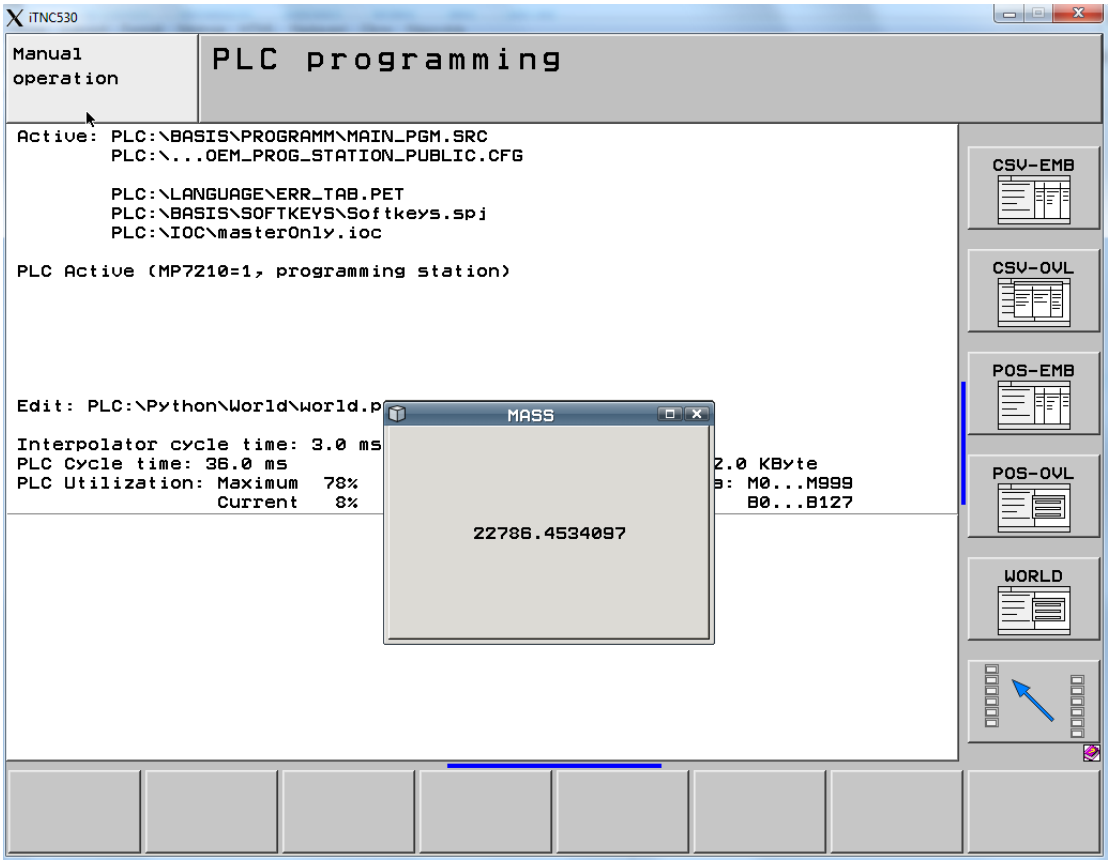



*Figura 3- 20 :Resultado de código Python implementado en la estación de programación iTNC 530 de Heidenhain. Extraída de software Programing Station iTNC 530.*

## 3.5 Creación de Rutina en el PLC

En esta sección se usa el software de PLC Design para crear una rutina en el control que pueda controlar el orden de los macros en código CN y de los scripts de Python, y las condiciones necesarias que se deben cumplir en la máquina para correr estos programas.

Lo que se buscó con la programación del PLC fue crear una rutina que respondiera a un softkey y de inmediato corriera las pruebas de calibración y de medición, calculara la masa a

partir de los datos obtenidos de los archivos creados por las pruebas anteriores, y una vez obtenida la masa corriera el macro que se encarga de sustituir los valores de los MP deseados. Por lo tanto, el usuario solo tendría que presionar la softkey para realizar el cambio de parámetros de la máquina.

### 3.5.1 Creación de variables globales y su conexión a la 'softkey'

La primera tarea por realizar es crear una nueva softkey dentro del plc y asignarle una marca que cambie al ser esta tecla presionada. Esto con el propósito de que el cambio en la marca se encargue de empezar la rutina que se programará luego.

Para la creación de la softkey desde el PLC Design, se debe presionar el archivo de SoftKeys en el árbol de la izquierda del programa. En este archivo se encontrará el software de Softkeys MDF para la creación de las mismas. En la siguiente figura se muestra la pantalla del software.

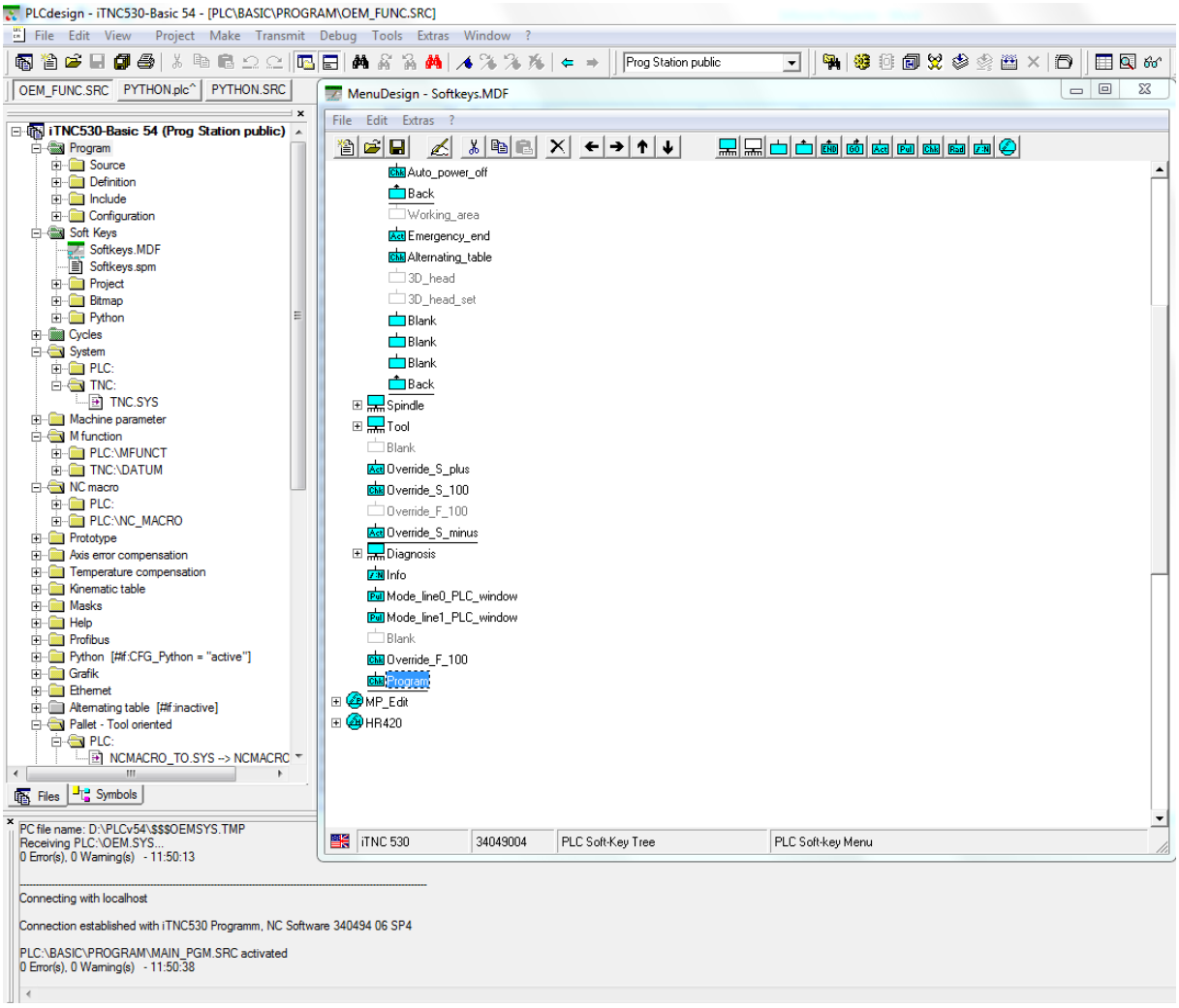



*Figura 3- 21: pantalla del Softkey.MDF para creación de SoftKeys. Extraída del software PLC Design*

Para crear una softkey simplemente se presiona alguna de las teclas superiores derechas del programa. Para crear una softkey que tenga asignada nada más una marca se debe seleccionar ‘nueva softkey check’, y se inserta en donde haya un lugar vacío en el árbol de las SoftKeys, para así no sustituir alguna otra softkey ya existente en la estación de programación.

El siguiente paso, es asignar una variable a esta softkey. Para esto, primero debemos definir la variable en el archivo de definiciones globales. Cabe destacar, que una variable asignada a una softkey solo puede ser global y no local de un módulo. Para definir una variable abrimos el archivo de definiciones globales como se muestra en la figura 3-23.

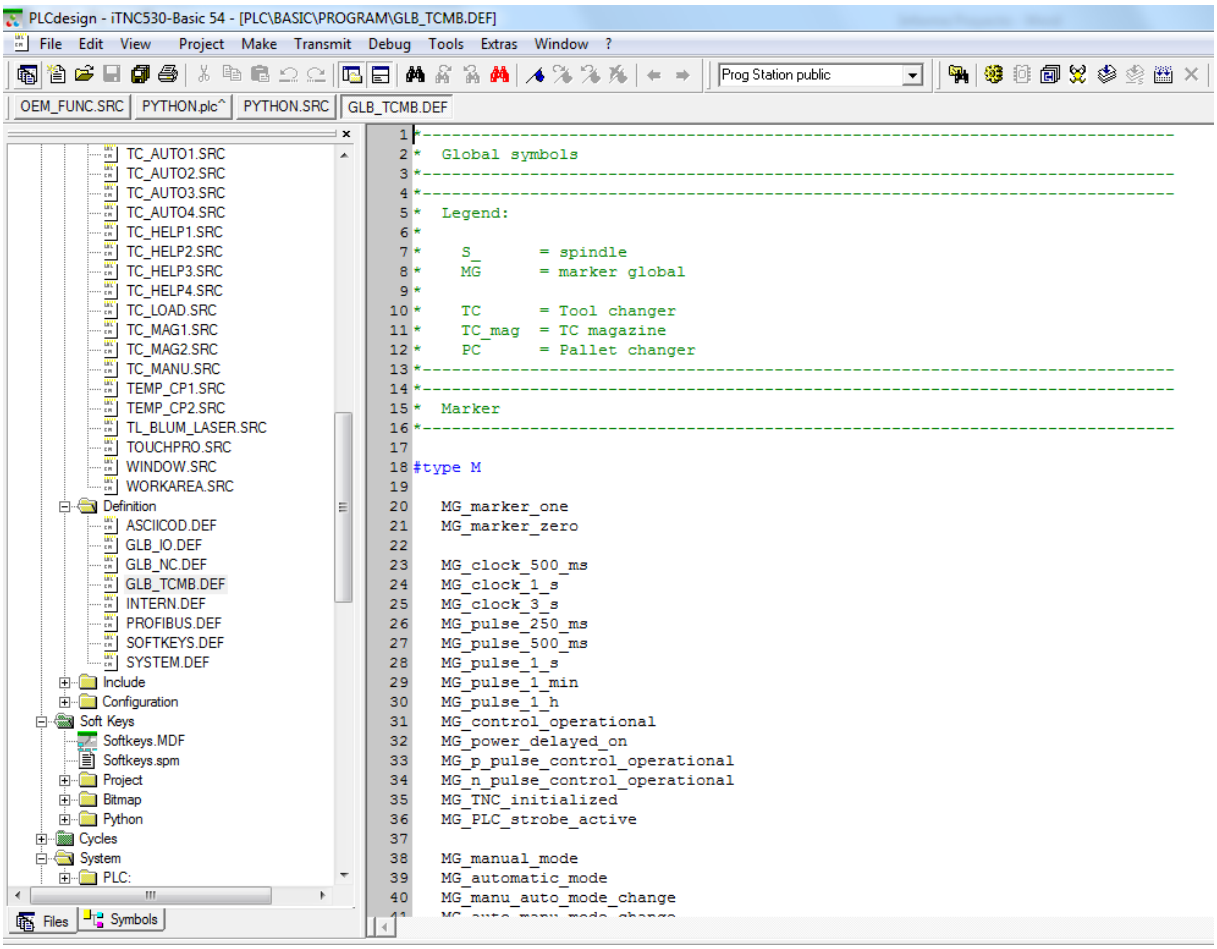



*Figura 3- 22: archivo de variables globales. Extraída del software PLC Design*

En este archivo podemos definir todo tipo de variable global, ya sea una marca, una constante, una palabra o una doble palabra. Para este caso, necesitamos definir dos variables globales tipo marca para asignar a la softkey creada. Una de estas marcas se encarga de cambiar de valor cuando se presiona la tecla, y otra se refiere al ‘enable’ de la tecla creada, por lo tanto, cuando esta variable esté en cero la tecla no estará disponible para utilizarse.

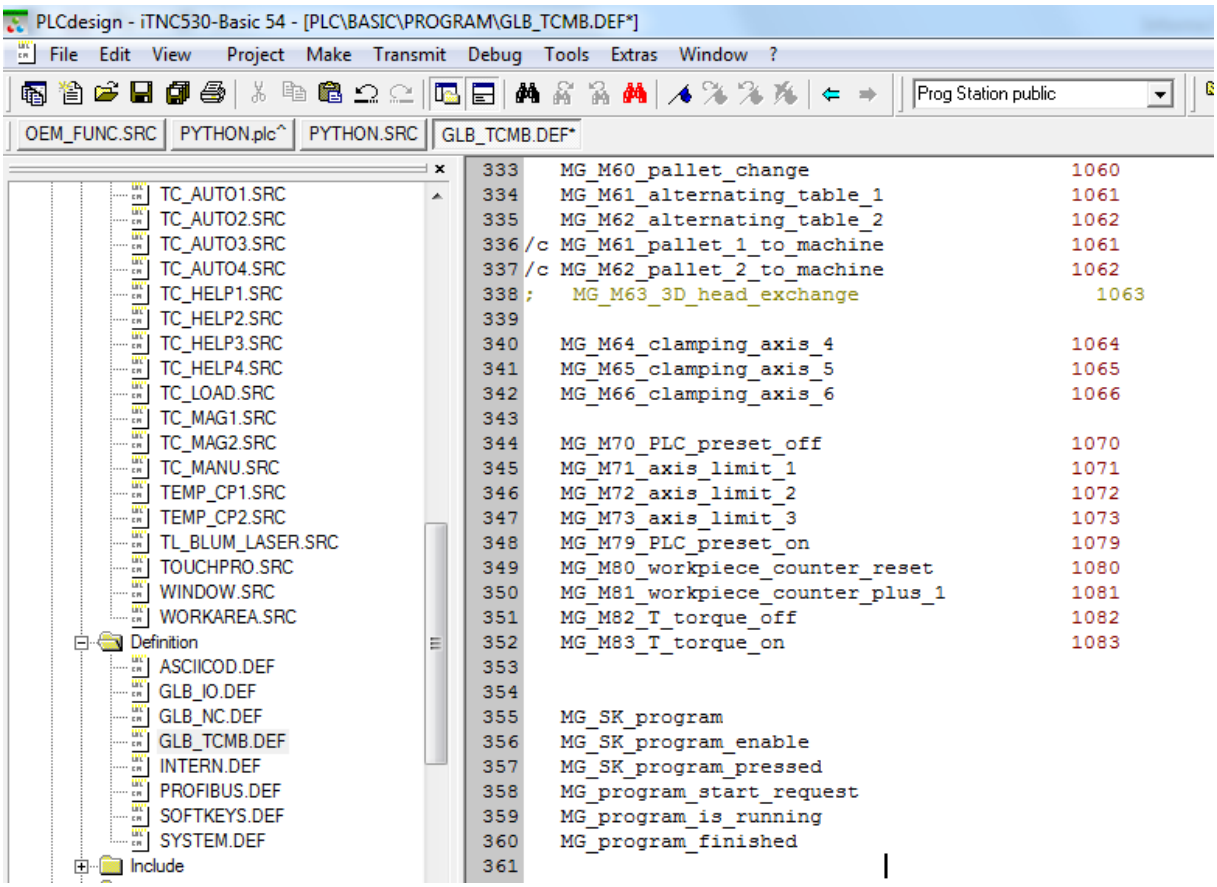



*Figura 3- 23: variables globales creadas para la rutina. Extraída del software PLC Design*

En la figura 3-24. se observan las 6 variables globales que se crearon para el uso de la rutina de PLC deseada. A continuación, se explica el uso de cada una. En la sección 3.5.4 se utiliza cada una de estas variables.

- MG_SK_program: marca global asignada al estatus de la softkey.
- MG_SK_program_enable: marca global asignada a la habilitación de la softkey.
- MG_SK_program_pressed: marca global que solicita la activación del programa una vez que MG_SK_program ha sido presionada y ningún otro proceso esté corriendo dentro de la máquina.
- MG_program_start_request: marca global que se encarga de empezar el proceso y deshabilitar la softkey mientras este esté activo.
- MG_program_is_running: marca global que se mantiene en alto mientras el programa corre.
- MG_program_finished: marca global que se activa una vez que finaliza el primer macro del programa.

Una vez creadas las variables globales necesarias para la rutina a programar, debemos asignar las dos variables (MG_SK_program y MG_SK_program_enable) a la softkey. Para esto nos vamos a editar la softkey en el Menú Design donde la creamos inicialmente y en la casilla de 'assignment' escribimos las dos variables que queremos asignar. En la figura 3-25 se observa esta asignación.

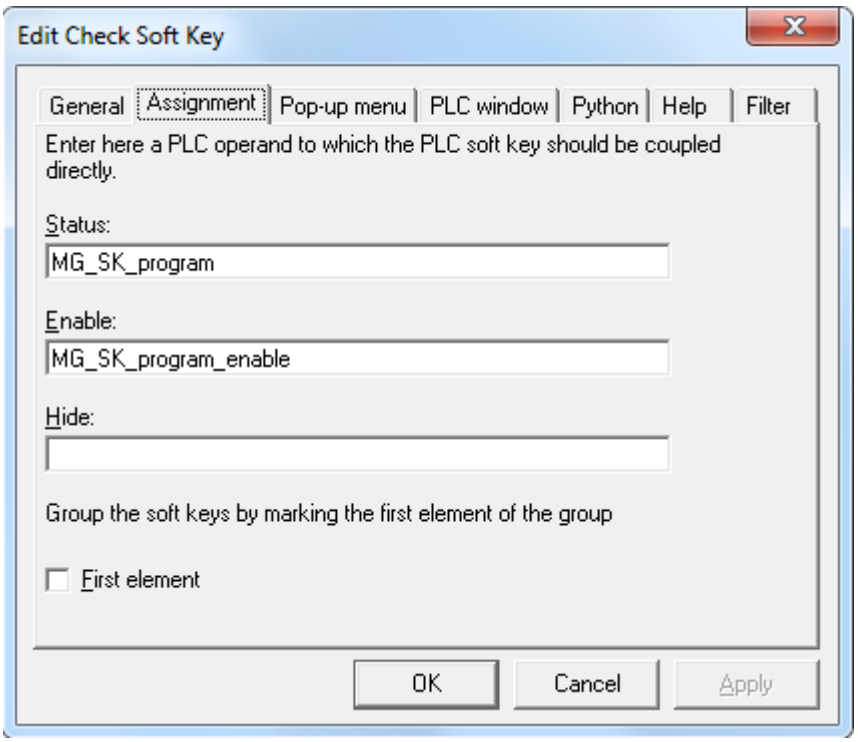



*Figura 3- 24: asignación de variables globales a softkey. Extraída del software PLC Design*

Para nombrar la softkey y asignarle un diseño gráfico usamos la casilla de 'general'. Cabe destacar que el diseño de la softkey fue otorgado por el RCMT en un archivo. bpj y este mismo se le asignó a la softkey. Este diseño consta de un semáforo que cambia su luz de color entre verde y rojo según el valor de la marca asignada al estatus de la softkey.

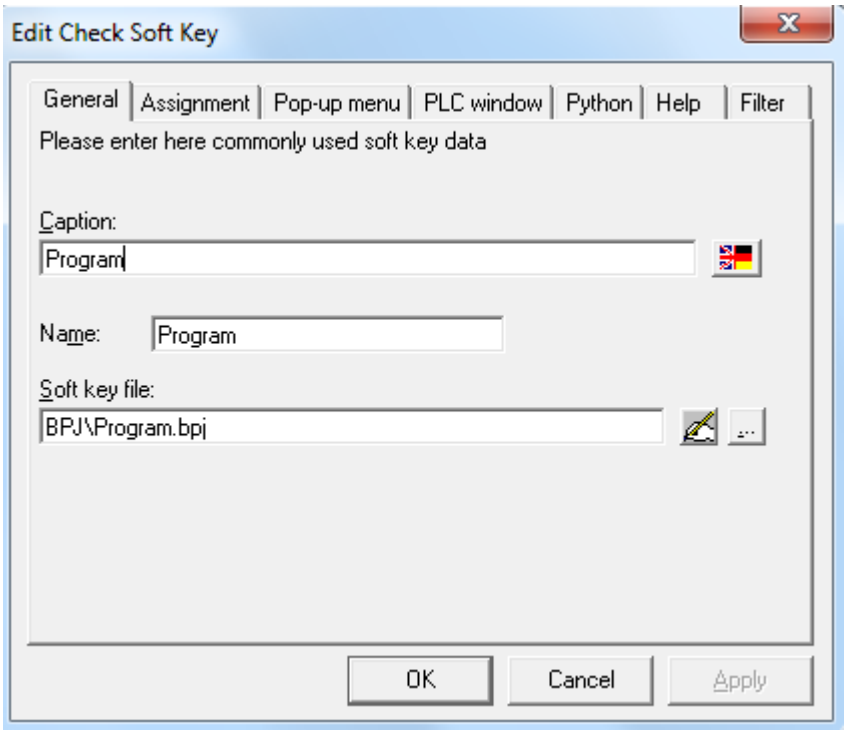



*Figura 3- 25: archivo y nombre de la softkey creada. Extraída del software PLC Design*

En las figuras 3-27 y 3-28, se muestra la estación de programación con la softkey agregada, tanto como en su modo de habilitación y su modo deshabilitado respectivamente.

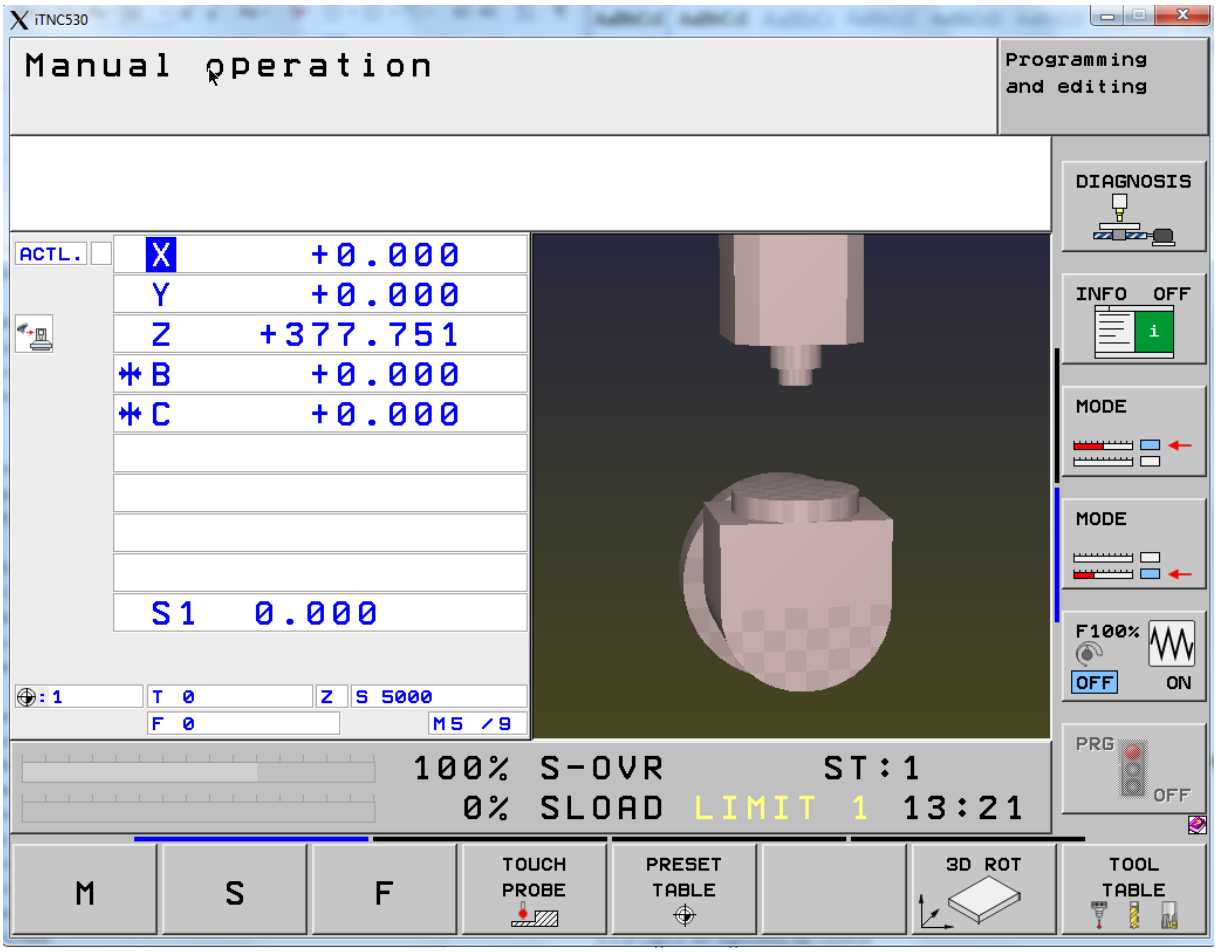



*Figura 3- 26: Estación de programación con la softkey deshabilitada. Extraída de software Programing Station iTNC 530*

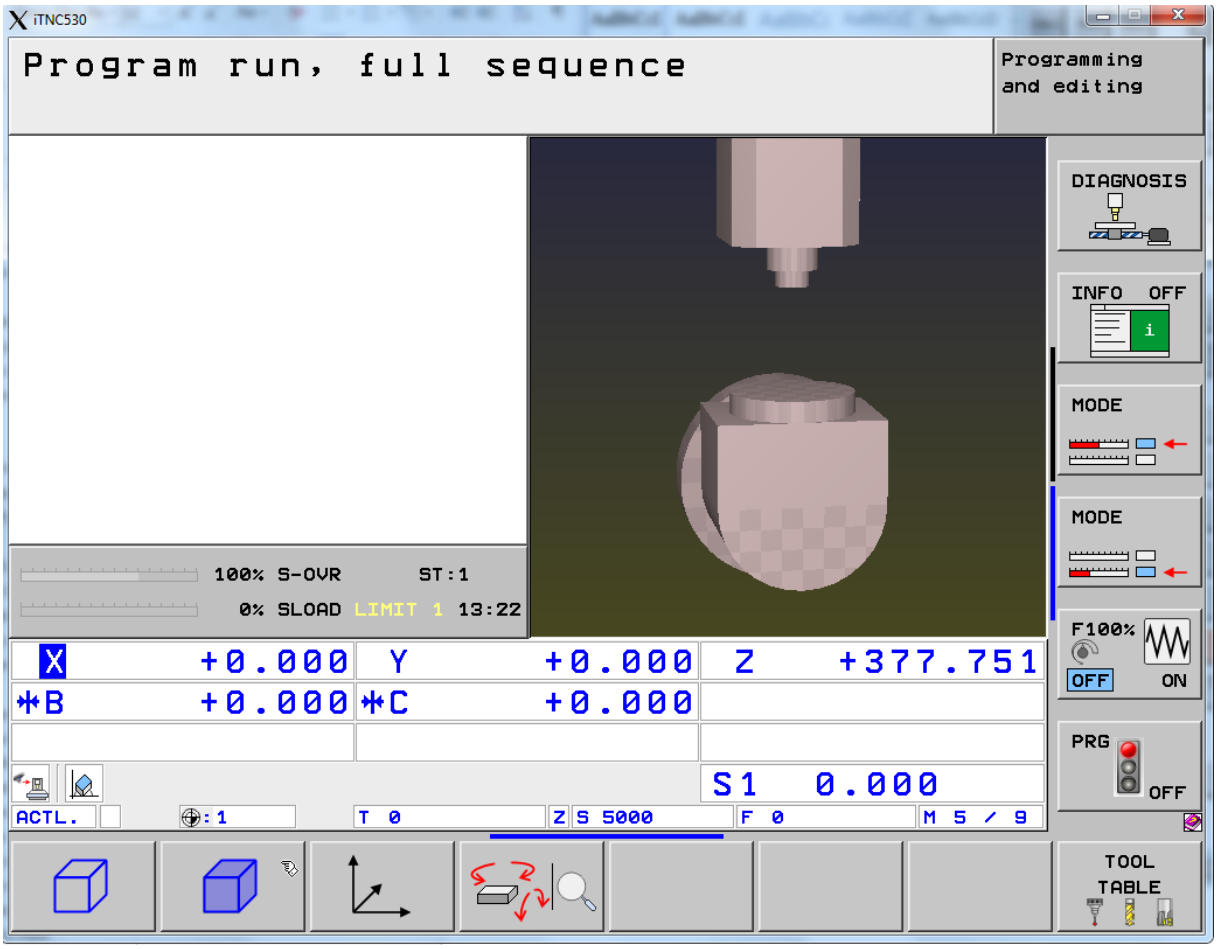



*Figura 3- 27: estación de programación con la softkey habilitada. Extraída de software Programing Station iTNC 530*

### 3.5.2 Creación de modulo

El siguiente paso para crear la rutina, es crear el módulo o función del PLC donde se va a encontrar dicha rutina. Para esto le damos a nuevo archivo y creamos un archivo tipo SRC llamado OEM_FUNC. En este archivo se hallará la rutina deseada. Para habilitar el módulo de OEM_FUNC debemos habilitarlo desde el programa principal, por lo tanto, en MAIN_PGR.SRC escribimos en la definición de módulos el módulo que queremos activar y la función o funciones dentro del mismo. Para este caso solo tendremos una función y a solicitud del RCMT se le llamará OEM_function_jose.

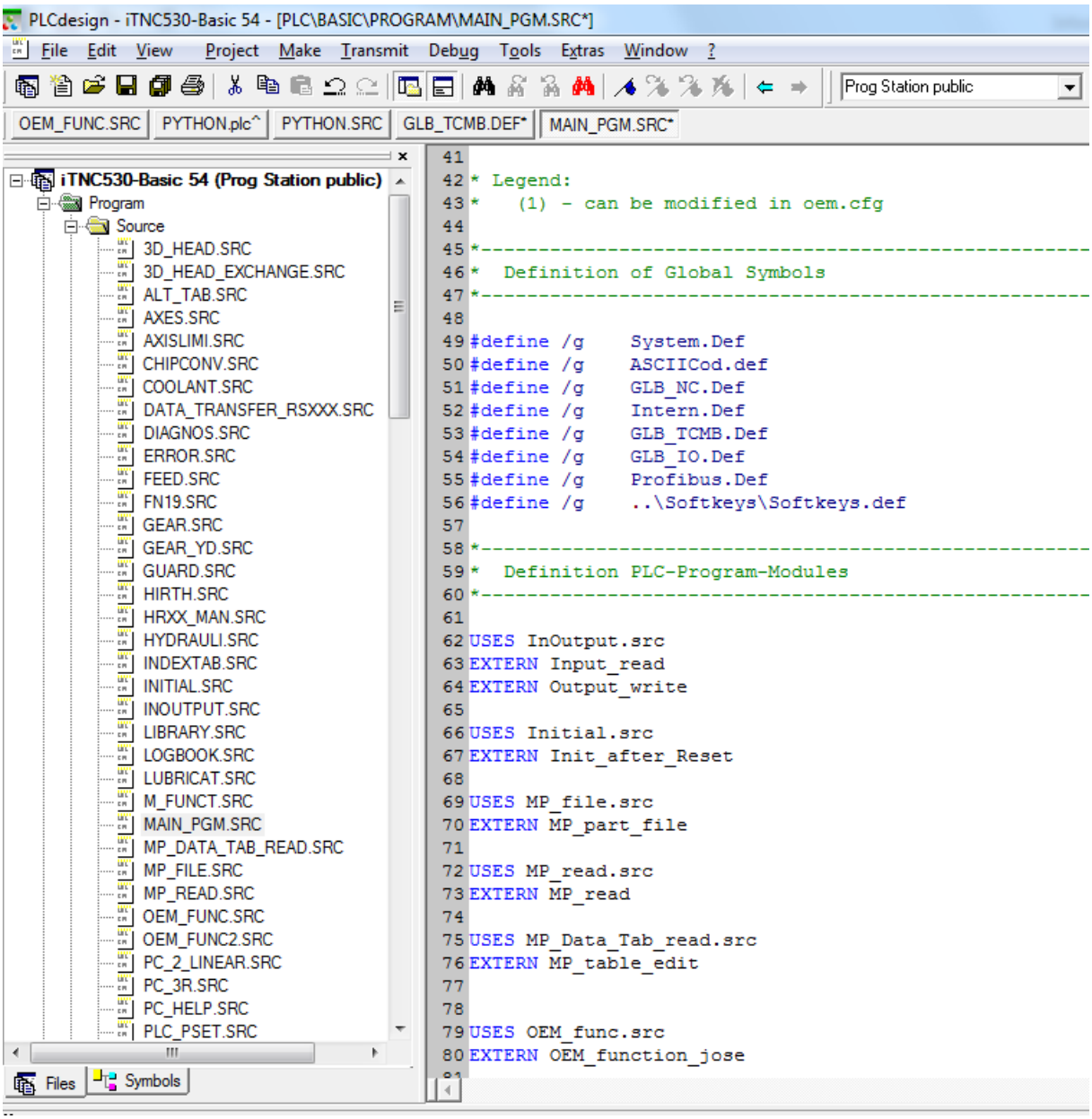



*Figura 3- 28:creación de nuevo módulo. Extraída del software PLC Design*

Una vez definido el módulo, este debe ser llamado desde el programa principal. A este se le puede poner alguna condición de definición, pero en este caso queremos que la máquina siempre tenga acceso a la rutina, por lo tanto, basta con una simple línea de código para llamar al módulo. Para llamar al módulo, se debe llamar a la función dentro del módulo deseada y no al módulo en sí, por lo tanto, la línea 'CM OEM_function_jose'se encarga de acceder a la función desde el programa principal.

Para finalizar la creación del módulo, se debe asignar las funciones dentro del mismo como globales para así poder ser llamadas desde el programa principal. Para esto abrimos archivo del nuevo módulo creado y asignamos la función como global. Luego, escribimos la función dentro del mismo modulo como un salto de etiqueta. También se deben asignar las librerías necesarias en esta sección.

```
46 GLOBAL OEM_function_jose
47
48 #include Mac_Lib.inc
49
50
51 ;----------------------------
52 LBL OEM_function_jose
53 ;----------------------------
```

*Figura 3- 29: definición del módulo como global y de su función como etiqueta. Extraída del software PLC Design*

### 3.5.3 Variables locales

El último paso, antes de empezar la lógica del algoritmo de la rutina del PLC, es definir las variables locales dentro del módulo creado. A diferencia de las variables globales, las variables locales solo pueden ser llamadas desde dicho modulo, y no tienen ninguna relación con otras variables locales de otros módulos. Estas variables pueden ser marcas, constantes, palabras, palabras dobles, o caracteres.

A continuación, se nombra cada una de las variables locales utilizadas en el módulo creado y la función de cada una de ellas. En la sección 3.5.4 se muestra su uso dentro de la rutina del PLC.

- ML_python_finished: marca local que se pone en alto una vez finalizado el script de Python.

- ML_reading_file: marca local que se mantiene en alto mientras la lectura del archivo de verificación no de la señal de que se ha terminado el proceso de Python y se ha calculado la masa correctamente.

- BL_Macro_start: bit local que interroga el estado del ‘programa de envío’ del primer Macro

- BL_Macro_start2: bit local que interroga el estado del ‘programa de envío’ del segundo Macro

- BL_identifier: bit local que interroga el estado del ‘programa de envío’ del script principal de Python

- BL_identifier2: bit local que interroga el estado del ‘programa de envío’ del script secundario de Python

- BL_return_value: bit local de respuesta al módulo 9296 que se encarga de terminar un proceso de Python.

- WL_error: palabra local que devuelve el error de cualquier modulo usado.

- DL_message_9297: palabra doble local que devuelve la respuesta del módulo 9297 al interrogar un proceso de Python.

- DL_error: palabra doble local que retorna el error del módulo 9295 al iniciar un proceso Python.

- DL_exit: palabra doble local que retorna el código de salida del módulo 9295 al iniciar un proceso Python.

- DL_File_handle: palabra doble local con el código de un archivo abierto por el módulo 9240, y se usa para cerrar dicho archivo con el módulo 9241.

- DL_line_read: palabra doble local que retorna el número de bits leídos de un archivo con el módulo 9243.

- KL_mode_asynchronous: constante local con el valor de 1 que se usa para asignar a cada proceso Python el modo de asincrónico.

- KL_python_process_active: constante local con el valor de 0 que se utiliza para verificar que un proceso Python no esté active cuando se le trata de inicializar.

- KL_terminate_python_process: constante local con el valor de 1 que se usa para 'matar' un proceso Python con el módulo 9296

- KL_memory_5_MB: constante local con el valor de 5 que se utiliza para poner un límite de memoria que todos los procesos Python en conjunto pueden utilizar.

- KL_file_open: constante local con el valor de 0 que se utiliza para verificar si se logró abrir un archivo con el módulo 9240

- KL_file_acces: constante local con el valor de cero que se utiliza para definir el tipo de acceso que se desea para el archivo abierto con el módulo 9240, en este caso cero ya que es un tipo de acceso de solo lectura.

### 3.5.4 Lógica del algoritmo de control

En esta sección se detallan los pasos que se utilizaron para crear la rutina de control del PLC dentro del módulo creado en la sección 3.5.2. Cabe destacar que esta rutina fue trabajada

y modificada a lo largo del proyecto. Sin embargo, en esta sección solo se muestra el resultado final del algoritmo y la lógica y pasos necesarios para su correcto funcionamiento.

A continuación, se presenta en la figura 3-31 el diagrama de flujo con la lógica usada en la rutina de control.

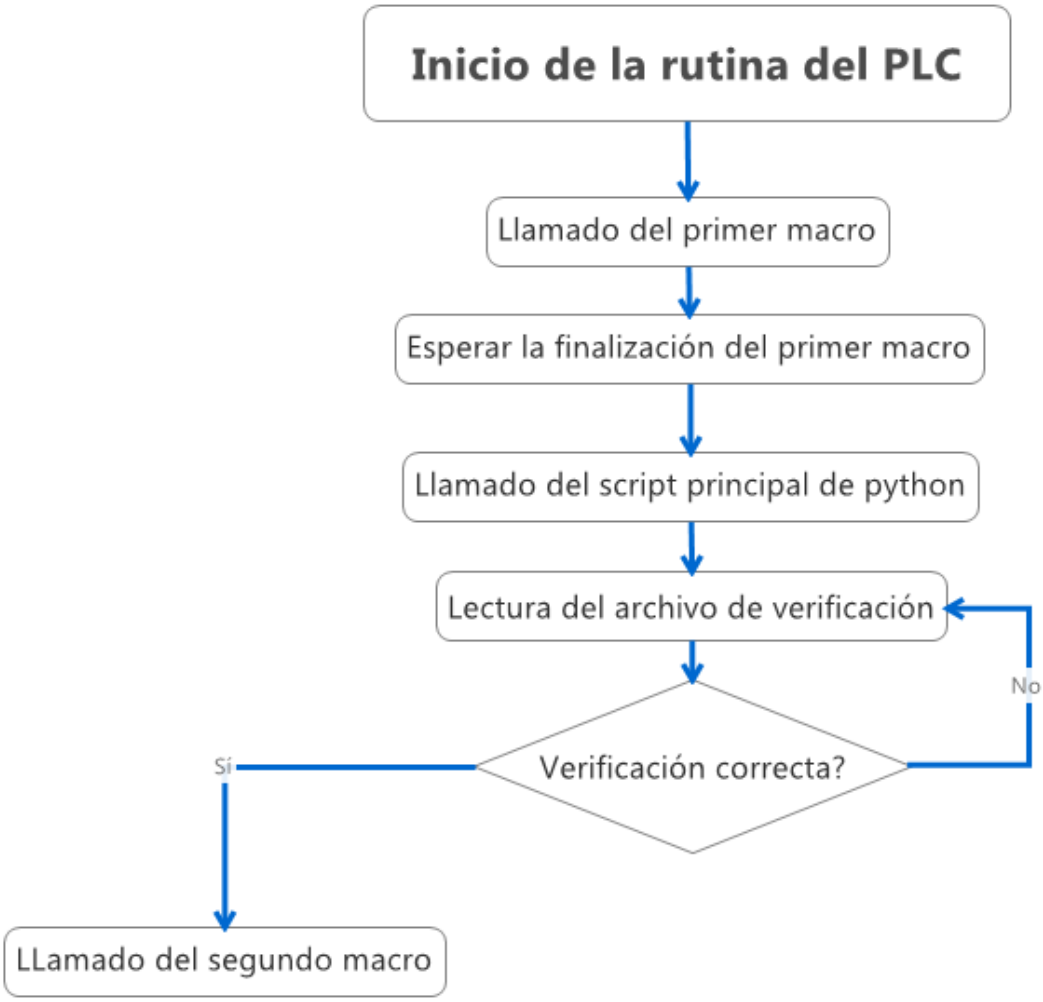



*Figura 3- 30:Diagrama de flujo con lógica de rutina de control del PLC. Diagrama construido en Xmind 2012.*

En las siguientes secciones se explica con detalle cada paso mencionado en el diagrama de flujo de la figura 3-39.

#### *3.5.4.1 Inicio de la rutina del PLC: Condiciones para habilitar la softkey*

Antes de iniciar la rutina, la softkey creada en el anexo 9.11 debe estar habilitada. Para esto, la marca global MG_SK_program_enable debe estar en alto. Sin embargo, no siempre se desea que dicha softkey esté habilitada. Por lo tanto, se ocupan programar algunas condiciones que activen o apaguen la habilitación de dicha softkey. Para esto creamos dos funciones, una que ponga en alto la marca de habilitación y otra que la ponga en bajo. En la siguiente figura se muestra el código usado para este fin.

```
LN      NP_M4153_mode_program_run_single_block
AN      NP_M4154_mode_program_run_full_sequence
O       NP_M4176_control_in_operation_on_or_flashes
O       NP_M4175_control_in_operation_flashes
R       MG_SK_program_enable


;----------------------------------------

L       NP_M4153_mode_program_run_single_block
O       NP_M4154_mode_program_run_full_sequence
AN      NP_M4176_control_in_operation_on_or_flashes
AN      NP_M4175_control_in_operation_flashes
S       MG_SK_program_enable
R       MG_program_is_running
```

*Figura 3- 31: funciones para habilitar y deshabilitar la softkey. Extraída del software PLC Design*

La primera función se encarga de resetear la habilitación, y para esto se deben cumplir cuatro condiciones. La marca 4153 y la marca 4154 se refieren al tipo de modo que la estación de programación está utilizando para correr sus procesos. Estas, específicamente, se refieren a la ejecución de bloque único y a la ejecución de secuencia completa (Ver anexo 9.5 para visualizar los controles del iTNC). La marca 4176 y 4175 se activan cuando la máquina está realizando un proceso. Por lo tanto, ambas deben estar en bajo para que la softkey esté habilitada. La lógica de la primera función se puede leer como: si la ejecución de bloque único o de secuencia complete están en bajo o alguna de las marcas 4176 y 4175 están en alto, la marca, la softkey se deshabilitará.

La segunda función mantiene las mismas condiciones solo que de manera inversa. Se puede leer como: si el programa está en modo de ejecución de bloque único o de secuencia complete, y ambas marcas 4175 y 4176 están en bajo la softkey estará habilitada. Además, esta función también pone en bajo la marca 'program_is_running' ya que, si la softkey está habilitada, el programa no puede estar corriendo.

#### *3.5.4.2 Inicio de la rutina del PLC: Función al presionar la softkey*

Al presionar la softkey la marca de MG_SK_program se pondrá en alto, pero solo por el instante que se presione dicha tecla, una vez que se deje de presionar retornará a su valor de bajo. Por lo tanto, se necesita una marca que se mantenga en alto una vez que la softkey ha sido presionada. Para esto, usamos dos funciones como se muestran en la figura 3-33.

```
L       MG_SK_program_enable
A       MG_SK_program
AN      NP_M4203_error_module_9xxx
AN      MG_program_start_request
S       MG_SK_program_pressed
-------------------------------------
L       MG_SK_program_pressed
S       MG_SK_program
S       MG_program_start_request
R       MG_SK_program_pressed
```

*Figura 3- 32: funciones al presionar softkey. Extraída del software PLC Design*

La primera función pone en alto la marca MG_SK_program_pressed si tanto como la habilitación y la marca de estado de la softkey están en alto, y además la marca 4203, que se refiere a errores en la máquina, está en bajo junto a MG_program_start_request, esta última nos dice que el programa no ha sido ya activado.

La segunda función apaga la marca de program_pressed y pone en alto el estado de la softkey y la marca de solicitud de inicio de programa. Por lo tanto, con estas condiciones en alto, el programa no puede ser llamado nuevamente hasta que no haya finalizado.

#### *3.5.4.3 Llamada del primer macro*

Una vez que la marca de solicitud de inicio en programa está en alto se hace el llamado al programa 'subido' de inicio del primer macro. En la siguiente figura se muestra el código para hacer el llamado.

```
L       MG_program_start_request
IFT
    RPLY    BL_Macro_start
    <>      K+0
    EMT
    SUBM    Running_macro
    =       BL_Macro_start
    ==      K+0
    S       PN_error_submit_spawn_queue_full
ENDI
```

*Figura 3- 33: código para llamar programa de envío del primer macro. Extraída del software PLC Design*

Esta función descrita en la figura 3-34, hace el salto al submódulo 'Running_macro', que utiliza el módulo 9291 para llamar al primer macro. En la siguiente figura se muestra el código para usar el módulo 9291.

```
LBL Running_macro

  PS      S"TNC:\Inert_meas\Private\INERT_MEAS_VS.H"
  CM      9291
  M_display_module_error

  L       ML_program_start_request
  R       ML_program_start_request
  S       ML_program_is_running
  R       ML_SK_program
  R       ML_SK_program_enable
  S       ML_program_finished


  L       NP_W1022_module_error_status
  =       WL_error


EM
```

*Figura 3- 34: función de llamado del primer macro*

Además de usar el módulo 9291, el submódulo también pone en alto y en bajo a varias marcas utilizadas anteriormente. Esta función pone en bajo el estado y la habilitación de la softkey, ya que ambas deben estar apagadas cuando el programa corre, también pone en bajo la solicitud de inicio de programa ya que este ya ha sido inicializado y una vez terminado de correr el programa se pueda utilizar nuevamente. Además, pone en alto la marca de corrido del programa y de programa terminado, esta última se debe a que al estar en alto se cumple una de las dos condiciones para iniciar el script principal de Python. La otra condición es que la marca 4176 que está en alto mientras corre el macro, vuelva a estar en bajo.

El primer macro es llamado INERT_MEAS_VS.H y es el código CN que se encarga de realizar la prueba de calibración y de medición en la máquina y guardar los archivos creados dentro del PLC de la máquina. Estos archivos serán, luego, usados por el script principal de Python. Ver sección 3.1 y 3.2 para saber más sobre estos archivos y como el script de Python los utiliza para el cálculo de masa.

El código CN fue creado por el RCMT en un proyecto anterior, y se muestra en el anexo 9.1. Este código básicamente genera un movimiento horizontal en las piezas móviles dentro de la máquina y por medio de los sensores dentro de esta, mide la posición, la velocidad, la aceleración y el torque entregado por los motores para cada punto en el tiempo definido por un intervalo y un salto entre puntos.

#### *3.5.4.4 Inicio del script principal de Python*

Una vez concluido el proceso del primer macro, la marca 4176 se pone en bajo y al tener la marca de finalización de programa en alto, la rutina de PLC llama al script de Python por medio de otro programa ‘subido’. Este submódulo es llamado ks_py_prueba, y contiene 3

módulos usados para los procesos Python en Heidenhain. En la figura 3-36 se muestra el código de este submódulo.

```
LBL KS_py_prueba

  L       ML_program_finished
  R       ML_program_finished


  PS      S"PYTHON_CODE"                          ;process name
  CM      9297                                    ;Status of a Python instance
  PL      DL_message_9297

  L       DL_message_9297
  ==      KL_python_process_active
  IFT

    PS      KL_terminate_python_process           ;Mode
    PS      S"PYTHON_CODE"                        ;process name
    CM      9296                                  ;Close / interrupt of python instance
    PL      BL_return_value

    LN      ML_program_finished
    S       ML_program_finished

   M_display_module_error

  ELSE

    PS      KL_mode_asynchronous                        ;Mode
    PS      KL_memory_5_MB
    PS      S"PLC:\\Python\\Project\\PYTHON_CODE.py"    ;Script Name Python
    PS      S"PYTHON_CODE"                              ;process name
    PS      S"PYTHON_CODE"                              ;Parameter
    CM      9295                                        ;Starten Python Prozess
    PL      DL_error                                ;Fehlercode
    PL      DL_exit                                 ;Exit-Code Python
    M_display_module_error
```

*Figura 3- 35: Función de llamado de un proceso Python. Extraída del software PLC Design*

Este submódulo verifica el estatus del script de Python usando el módulo 9297 y retorna un valor distinto a cero si este no está activo. Si el script está activo, regresa 0 y el módulo 9296 se encarga de 'matar' el script. En caso contrario, el código llama al módulo 9295 que se encarga de empezar el proceso de Python de manera asincrónica, con capacidad de 5 MB de memoria, y en la dirección deseada dentro del PLC.

El proceso debe ser llamado de manera asincrónica. Ya que, si otro proceso está activo, el proceso de Python no debe tener prioridad sobre este, y no debe llamarse correctamente hasta que el otro proceso finalice. En el parámetro necesario para el módulo se puede escribir cualquier parámetro, ya que el script de Python no tiene ningún parámetro de entrada.

#### *3.5.4.5 Verificación de finalización de script de Python*

Ya que el PLC no cuenta con ninguna marca que notifique si un proceso Python está activo o no, se debe implementar otro método para comprobar que el proceso ha sido terminado.

Para esto se crea un archivo de texto que sirva como señal de que el proceso ha terminado, la masa ha sido calculada, y los parámetros han sido cambiados. Este archivo de texto recibe el nombre de 'Verification.txt' y es guardado dentro del PLC. El archivo tiene escrito 'ON' y nada más. La rutina de Python se encarga de leer dicho archivo y mientras haya menos de 3 caracteres no se invocará el segundo macro. A continuación, se presenta la figura 3-37 con el código de lectura del archivo de verificación.

```
LBL Read_mass

  PS      KL_file_acces                                   ;B/W/D/K    <Type of access>
  PS      S"PLC:\\Python\\Project\\Verification.txt"       ;B/W/D/K/S  <String with file name>
  CM      9240                                            ;Open a file by the PLC
  PL      DL_file_handle                                  ;D          <File handle>
  M_display_module_error

  L       DL_file_handle
  <>      KL_file_open
  IFT
    PS      DL_file_handle              ;D          <File handle>
    PS      K+1                         ;B/W/D/K  <Number of the target string>
    CM      9243                        ;Read from a file line-by-line
    PL      DL_line_read                ;B/W/D    <Number of bytes read (including LF)>


    L       DL_line_read
    >       K+3
    S       ML_python_finished
```

*Figura 3- 36:Código de lectura de archivo de verificación. Extraída del software PLC Design*

Este código es un programa 'subido' que usa el módulo 9240 para abrir el archivo de Verification.txt y usa el módulo 9243 para leer los caracteres dentro de él. La palabra doble DL_line_read retorna la cantidad de caracteres que hay en el archivo, si estos son mayores a tres, significa que el proceso Python hay terminado y la rutina está lista para llamar al segundo macro.

La manera por la cual la cantidad de caracteres en el archivo de verificación varía se debe a que el código Python escribe en este archivo una vez que haya terminado. Luego del cálculo de masa (ver sección 3.3) y el cambio de MP (ver sección 3.6), el código muestra en pantalla el valor de la masa por medio de una ventana (ver sección 3.4.1). Esta ventana muestra la masa y su incertidumbre al usuario. El usuario presiona la ventana para continuar el proceso y el script de Python escribe 'finish' en el archivo de Verification.txt, esto solo para modificar el número de caracteres en dicho archivo y poner en alto el valor de la marca de proceso de Python finalizado.

```
def on_main_window_destroy(widget):
    try:
        path=jh.ResPath('PLC:/Python/Project/Verification.txt')
        files=open(path,'w')
        files.write('finish)
        files.close()

    except error_to_catch:
        pass
    jh.MainQuit()
```

*Figura 3- 37:Función de terminación de código Python*

Con esta función, el código de Python termina y la rutina de Python reconoce su finalización. Sin embargo, sería imposible volver a correr el mismo programa ya que el archivo de verificación fue modificado y posee más de tres caracteres. Por lo tanto, se debe modificar nuevamente el archivo a su estado original. Se vuelve a abrir el archivo y se sustituye 'finish' por 'ON' nuevamente.

Una vez modificado el archivo de verificación el PLC para de leer dicho archivo y usando el módulo 9241 lo cierra. Cabe destacar que el módulo 9241 utiliza la palabra doble de file_handle entregada por el módulo 9240, que se encargaba de abrir el archivo.

#### *3.5.4.6 Llamado del Segundo macro.*

Para finalizar la rutina del PLC se debe llamar al segundo macro llamado MPSet.H. En este se describen los parámetros de la máquina a modificar y los comandos necesarios en CN para modificarlos (Ver anexo 9-2 para ver código CN de este macro).

El llamado del segundo macro es igual que el del primero. Se debe usar un programa 'subido' para poder acceder al macro invocado por el módulo 9291. La única diferencia es que este macro se encarga de poner en bajo la marca de proceso de Python finalizado, para que el proceso se pueda repetir. A continuación, se muestra el código usado para llamar al segundo macro.

```
LBL Running_macro2

  PS      S"P$OEM01"
  CM      9291
  M_display_module_error

  L       ML_python_finished
  R       ML_python_finished

  L       NP_W1022_module_error_status
  =       WL_error

EM
```

*Figura 3- 38: función de llamado de segundo macro. Extraída del software PLC Design*

La palabra W1022 se usa para preguntar que tipo de error devuelve el módulo en caso de que haya un error. Esta variable se usa a lo largo del código en los distintos módulos que se usan.

La dirección del macro dentro del PLC se guarda en la variable P$OEM1, donde esta se encuentra definida en el archivo de NC_MACRO.SYS dentro del PLC con las direcciones de cada macro a usar.

Terminado de correr el segundo macro, el proceso finaliza y las variables terminan en sus condiciones iniciales, listas para repetir dicho proceso. En la sección 3.6 se desarrolla el contenido del macro 2 y el cambio de parámetros a realizar con el código de Python.

El último paso por desarrollar dentro del PLC es la compilación del código y su transferencia a la estación de programación. Los pasos para llevar esto a cabo se describen en el anexo 9-12.

## 3.6 Reemplazo de parámetros

En esta sección se desarrolla la solución utilizada para sustituir los parámetros de la máquina según la masa calculada anteriormente. Para esto se utiliza el código Python y un archivo de código CN que, posteriormente, será usado como el segundo macro en la rutina del PLC. (Ver sección 3.5.4.6).

### 3.6.1 Archivo MPSetTemp.H y MPSet.H

Antes de crear la lógica del algoritmo en lenguaje Python, se debe conocer los archivos que se van a manipular desde el código para lograr el reemplazo de parámetros en la máquina. Como se mencionó en la sección 3.5.4.6 (llamado del segundo macro), el código CN dentro del archivo de MPSetTemp.H se encarga de modificar los parámetros necesarios en la máquina. Por lo tanto, el objetivo del código Python es modificar los valores de estos parámetros reescribiendo el archivo, para que este luego sea utilizado como segundo macro en la rutina del PLC.

El archivo de MPSet.H es una copia exacta de MPSetTemp.H. Esto se debe a que el archivo de MPSetTemp.H funciona como un modelo de solo lectura, y no debe ser modificado bajo ninguna condición, por motivos de seguridad. El archivo MPSet.H será modificado con el cambio de parámetros y será el macro llamado desde la rutina del PLC para hacer los cambios en los MP dentro de la máquina.

Realizar la reescritura de los parámetros en un archivo dentro del PLC desde Python es bastante sencillo, y no sería necesario dedicar toda una sección de este documento a tal objetivo. Sin embargo, estos archivos no son simples archivos de código CN con texto escrito para modificar. Ambos archivos, al estar dentro de los macros del PLC, están encriptados por un

algoritmo desarrollado en Heidenhain y al abrirlos desde fuera de la estación de programación se muestran en este código encriptado. En la siguiente figura se muestra la encriptación de MPSetTemp.H, en el anexo 9.2 se muestra la forma no encriptada de tal archivo.

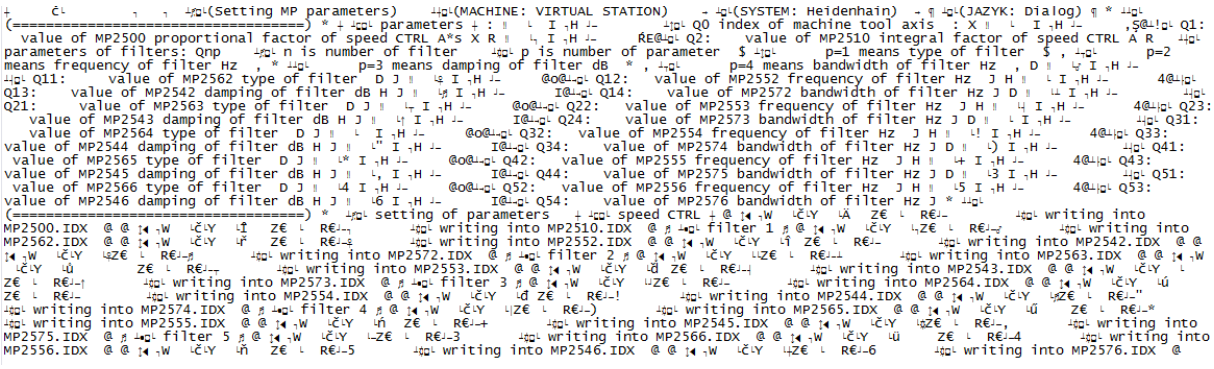


*Figura 3- 39: Archivo encriptado por algoritmo Heidenhain.*

Se debe trabajar con el archivo como se muestra en la figura 3-40. Y de alguna forma lograr desencriptar el algoritmo de Heidenhain para poder hacer cambios funcionales dentro del archivo y que la máquina pueda leerlo nuevamente con los cambios realizados. Para esto se construye un algoritmo en lenguaje Python desarrollado en la sección 3.6.3.

### 3.6.2 Tabla de parámetros a modificar

Antes de empezar la lógica de reemplazo de parámetros, se deben conocer los parámetros que se desean cambiar y la tabla que contiene los valores de estos según la masa calculada de las partes móviles dentro de la máquina. A continuación, se presentan los parámetros a modificar.

- MP2500: factor proporcional de velocidad
- MP2510: factor integral de velocidad.
- MP256n: tipo de filtro
- MP255n: frecuencia de filtro en Hz
- MP254n: ‘damping’ de filtro en db
- MP257n: ancho de banda de filtro en Hz

El valor de n va de 2 a 6, ya que se usan 5 filtros diferentes en los procesos de la máquina Kovosvit MCV MASS 1000, y los cinco filtros dependen de las características dinámicas del proceso, que se puede entender como el valor de la masa de las piezas móviles durante el proceso.

Los valores de estos 22 parámetros (4 por filtro, 5 filtros, y dos MP de velocidad de motor) se encuentran en una tabla agregada al archivo de ajustes y entradas detallado en la sección 3.1 de este documento. En esta tabla tenemos también el número de eje que se está midiendo, ya sea cualquier de los ejes lineales o de los dos rotacionales, y además el valor de la masa máximo para cada valor de los parámetros.

```
idx = 0
#\param inert MP2500 MP2510 MP2562 MP2552 MP2542 MP2572
    1,     0,  3550,  44.5,     0,     0,     0,     0,
    2, 20000,  3650,  45.5,     1,   700,     x,     0,
```

*Figura 3- 40: sección de la tabla de parámetros con sus valores disponibles para cambiar.*

En la figura 3-41, se muestra una sección de la tabla. Bajo la columna de 'inert' está el valor de la masa mínimo para dichos valores de MP. El valor de idx se refiere al número de eje, en este caso al eje 'x', cabe destacar que este valor también debe ser reescrito en el archivo de MPSet.H para que luego sea leído por la máquina.

### 3.6.3 Lógica de algoritmo de remplazo de parámetros

En esta sección se desarrolla la lógica para reemplazar los parámetros en el archivo MPSet.H desde el script de Python. En el siguiente diagrama se presenta el algoritmo que se usó para obtener el resultado deseado.

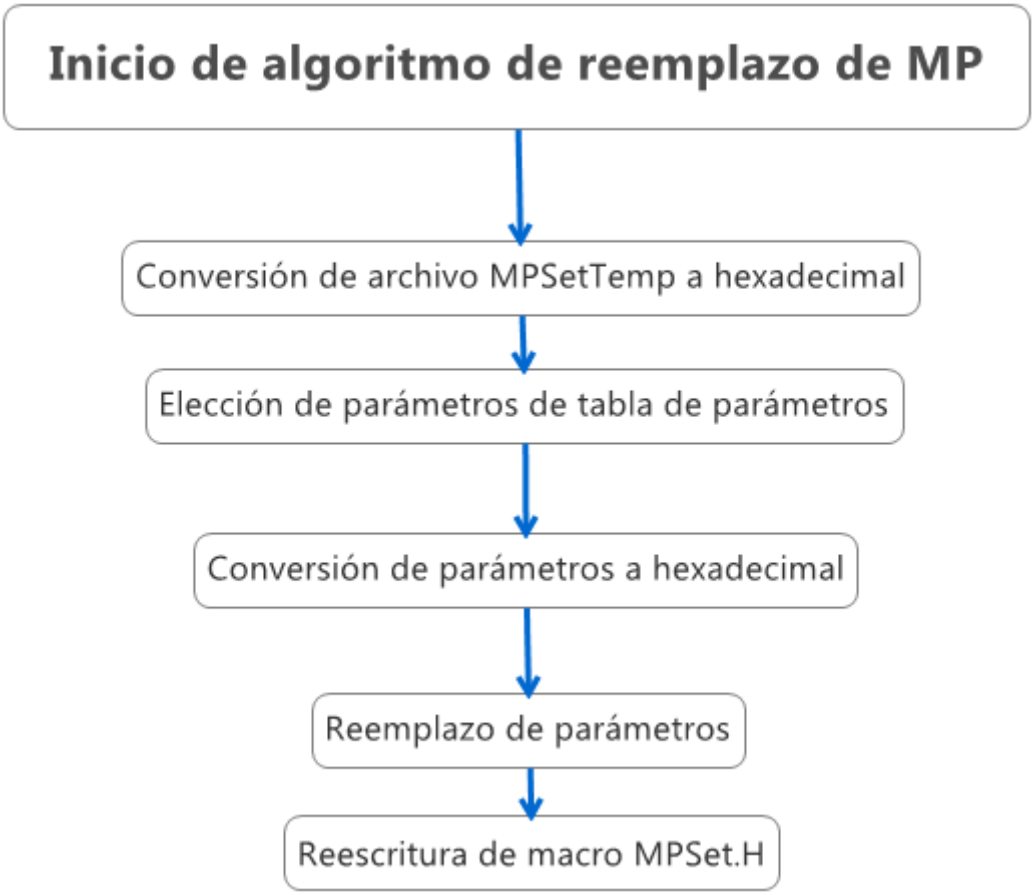



*Figura 3- 41: Diagrama de lógica de algoritmo de reemplazo de parámetros. Diagrama construido en Xmind 2012.*

La lógica principal de este algoritmo es el uso del Sistema hexadecimal. Ya que los archivos encriptados están en código ASCII, sabemos que cada sigo tiene su propio valor hexadecimal. Por lo tanto, es más sencillo manipular el código en hexadecimal que en simbología ASCII. Para esto convertimos el archivo de MPSetTemp a hexadecimal usando la función de Python toHex (). En la siguiente figura se muestra cómo se visualiza una sección del archivo en lenguaje hexadecimal.

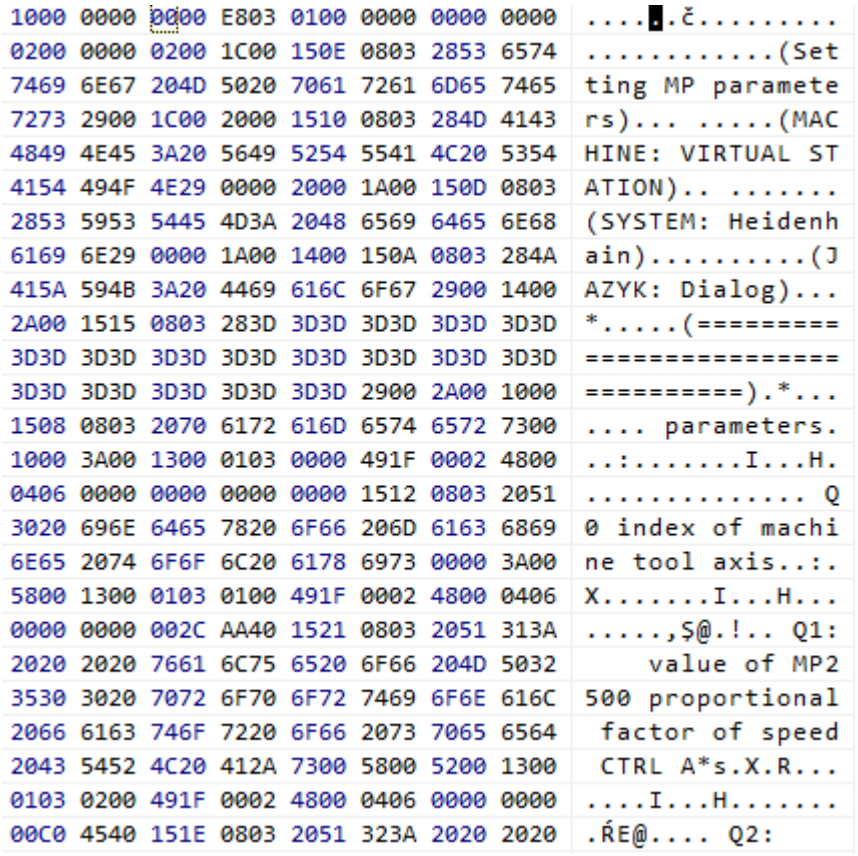



*Figura 3- 42:sección de archivo encriptado en lenguaje hexadecimal.*

Al derecho de esta figura tenemos el archivo en lenguaje ASCII, como se había mostrado en la figura 3-43. Y a la izquierda el respectivo código hexadecimal de cada carácter. Una vez obtenido el código hexadecimal del archivo MPSetTemp, se prosigue con el algoritmo mostrado en la figura 3-42.

### *3.6.3.1 Elección de parámetros de tabla de parámetros y conversión de los mismos a hexadecimal*

Luego de haber convertido el archivo a hexadecimal, se deben leer los parámetros deseados del archivo de ajustes y entradas. Estos se seleccionan según el valor de la masa calculado en la sección 3.3

Los parámetros pueden ser cualquier número real positivo o cero. También estos parámetros pueden tener un valor de ‘x’ que significa que dicho parámetro debe permanecer igual al parámetro en el archivo modelo. Estos valores de ‘x’ han sido una condición solicitada por el RCMT al algoritmo en construcción.

Según el valor de la masa, se selecciona una fila de la tabla. De esa fila cada parámetro es guardado en una lista, y convertido a un valor hexadecimal acorde a la encriptación de Heidenhain.

Para descifrar el algoritmo que usó Heidenhain para encriptar sus archivos, se hicieron pruebas en un macro más simple de la máquina. Este con las mismas condiciones de encriptación. En este macro se sustituyeron números desde la estación de programación y se observe la diferencia en el código hexadecimal al abrir dicho archivo en un lector de texto. Poco a poco se encontró un patrón al variar los números.

Como primera solución, se decidió construir una matriz que transformara todo número positivo con siete dígitos o menos a un valor hexadecimal. Luego, acorde a la posición en esta matriz se buscaba el valor del parámetro en la tabla. El problema, es que dicha matriz era una solución muy ineficiente en código y consumía mucha capacidad de memoria.

Luego, se encontró que dicho código hexadecimal tenía un patrón muy cercano al concepto de punto flotante de doble precisión (Ver sección 2.5.2). La única diferencia es que Heidenhain optaba por invertir cada el orden de cada par de dígitos entre los 16 dígitos del concepto de punto flotante de doble precisión. Por lo tanto, se decidió programar una función que realizara la matemática de transformar cualquier número a punto flotante de doble precisión y finalmente variar el orden de sus dígitos. La función se encuentra en el anexo 9.4.

#### *3.6.3.2 Reemplazo de parámetros*

Una vez obtenidos todos los valores de parámetros de la tabla del archivo de ajustes y entradas, y convertidos estos mismos a hexadecimal, solo se necesitan sustituir en el código hexadecimal obtenido del modelo MPSetTemp.H. Para esto debemos encontrar la posición de cada uno de estos valores en el código hexadecimal.

La solución que se propuso fue buscar cada punto y coma del archivo. Ya que el archivo es inmodificable, la cantidad de puntos y comas y sus respectivas posiciones nunca variará. Cada valor de parámetro a sustituir se encuentra previo a un punto y coma del archivo. Por lo tanto, se creó una función que encontrara las posiciones de todos esos puntos y comas, y en cada uno de esos que siguiera de un parámetro guardar dicha posición en una lista. El punto y coma en el código hexadecimal encriptado de Heidenhain tiene como código 15xx 0803. Donde las ‘x’ pueden ser cualquier valor hexadecimal.

Encontrado las 22 posiciones donde se desea sustituir los parámetros, debemos sustituir cada valor de MP en hexadecimal en los 16 dígitos previos al de estas posiciones. Ya que estos 16 valores representan el número previo al punto y coma. Por lo tanto, el número que deseamos convertir. En la siguiente figura vemos resaltado los 16 dígitos del número (0000 0000 002C A440), seguidos de los 8 del punto y coma (15xx 0803).

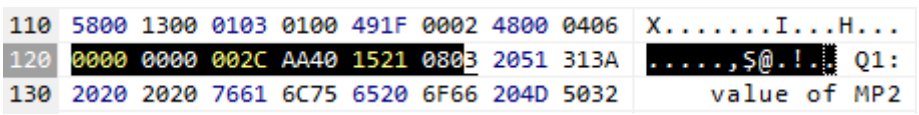



*Figura 3- 43: resaltado de un número y del “punto y coma” en lenguaje hexadecimal de la encriptación Heidenhain. Elaboración propia.*

Estos 16 dígitos representan al número 3350, que es el valor del parámetro MP2500 en el archivo modelo. En la siguiente figura se observa este valor.

```
7  Q0 = 0 ; Q0 index of machine tool axis
8  Q1 = 3350 ; Q1:    value of MP2500 proportional factor of speed CTRL A*s
9  Q2 = 43.5 ; Q2:    value of MP2510 integral factor of speed CTRL A
```

*Figura 3- 44: parámetro 2500 representado en texto no encriptado.*

Sin embargo, la transformación de Python del archivo de código ASCII a hexadecimal, retorna una matriz con valores pares. En vez de un '0,0,1,2' regresa un ''00,12'. Por lo tanto, antes de sustituir los valores hexadecimales de los MP, estos se deben unir en pares. Haciendo así una lista de 8 valores y no una de 16.

Cuando se tiene la lista de 8 valores para cada MP, se prosigue a sustituirlos en las 8 posiciones anteriores a las posiciones de los punto y coma encontradas previamente. Con esto finaliza la sustitución de parámetros y se obtiene el código hexadecimal modificado que sustituirá al código ASCII del macro MPSet.H

#### *3.6.3.2 Reescritura de macro MPSet.H*

Para finalizar el reemplazo de parámetros y el proceso de Python en general, se debe realizar la reescritura del macro MPSet.H que posteriormente será usado por la rutina del PLC para finalizar el proceso del algoritmo de control (Ver sección 3.5.4.6). Este proceso se hace por medio del script de Python. El objetivo es reescribir los parámetros del archivo modelo de MPSetTemp por los obtenidos en la sección 3.6.2 de la tabla de parámetros.

Antes de reescribir el archivo de parámetros debemos transformar el código hexadecimal obtenido de la sección 3.6.3 vuelta a código de simbología ASCII. Esto se debe a que el macro debe estar en simbología ASCII para luego ser desencriptado por la estación de control de la máquina y pueda correrlo como un macro de código CN. Para transformar el código hexadecimal a lenguaje ASCII, se usa la función de Python '. decode('hex')'

Luego, se prosigue a la reescritura del archivo MPSet.H. Ya que se tiene todo el código en ASCII con los parámetros modificados en una lista en Python, se borra todo el archivo de MPSet.H para luego ser sustituido con dicha lista de Python.

```
path=jh.ResPath('TNC:\NC_MACRO\MPsetTemp2.H')
files=open(path,'r+')
files.seek(0)
files.truncate()
files.close()
```

*Figura 3- 45: código Python para reescritura de archivo de reemplazo de parámetros. Elaboración propia.*

La función de seek (0) busca la posición inicial del archivo. La función .truncate() se encarga de borrar todo lo que haya en el archivo desde el punto encontrado con la función .seek. Por lo tanto, con el código mostrado en la figura 3-46, se borra todo el contenido del archivo.

Una vez borrado todo el contenido, el archivo se vuelve a abrir en modo de escritura binaria, ya que así era la forma en que dicho archivo estaba escrito originalmente antes de empezar el proceso de cambios de parámetros. Con el archivo abierto, se escribe la lista en código ASCCI en él y se cierra el archivo. Con esto, se finaliza la sustitución de parámetros en el macro de MPSet.H.

### 3.6.4 Resultados en la estación de programación

Para finalizar la sección 3.6 de reemplazo de parámetros se muestran los resultados en la estación de programación al llevar a cabo la sustitución de los MP. Para mostrar estos resultados se muestra el archivo de MPSet.H antes de realizar el proceso de cambios y después de los cambios.

Recordar que el archivo de MPSet.H es una copia directa del archivo MPSetTemp.H, por lo tanto, antes de realizar el cambio de parámetros, estos dos archivos deben ser exactamente iguales. En la siguiente figura se muestra el archivo MPSet.H dentro de la estación de programación.

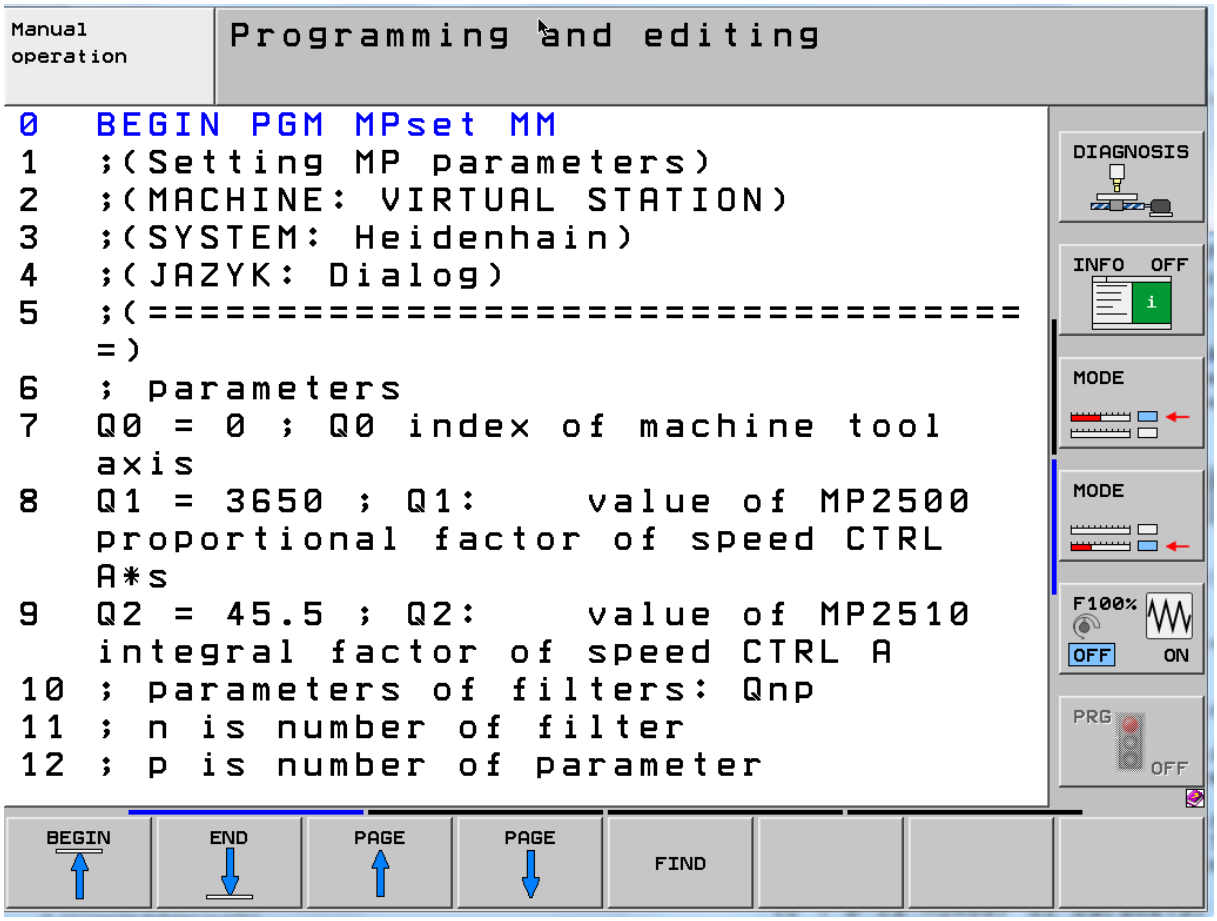



*Figura 3- 46: sección de archivo MPSet.H sin modificar. Extraída de software Programing Station iTNC 530*

En la figura 3-47 se puede observar la primera sección del código CN del archivo MPSet.H en la estación de programación antes de que se realicen los cambios de MP. En esta imagen hay 2 de los 22 parámetros a cambiar, y también el número del eje de la máquina (Q0).

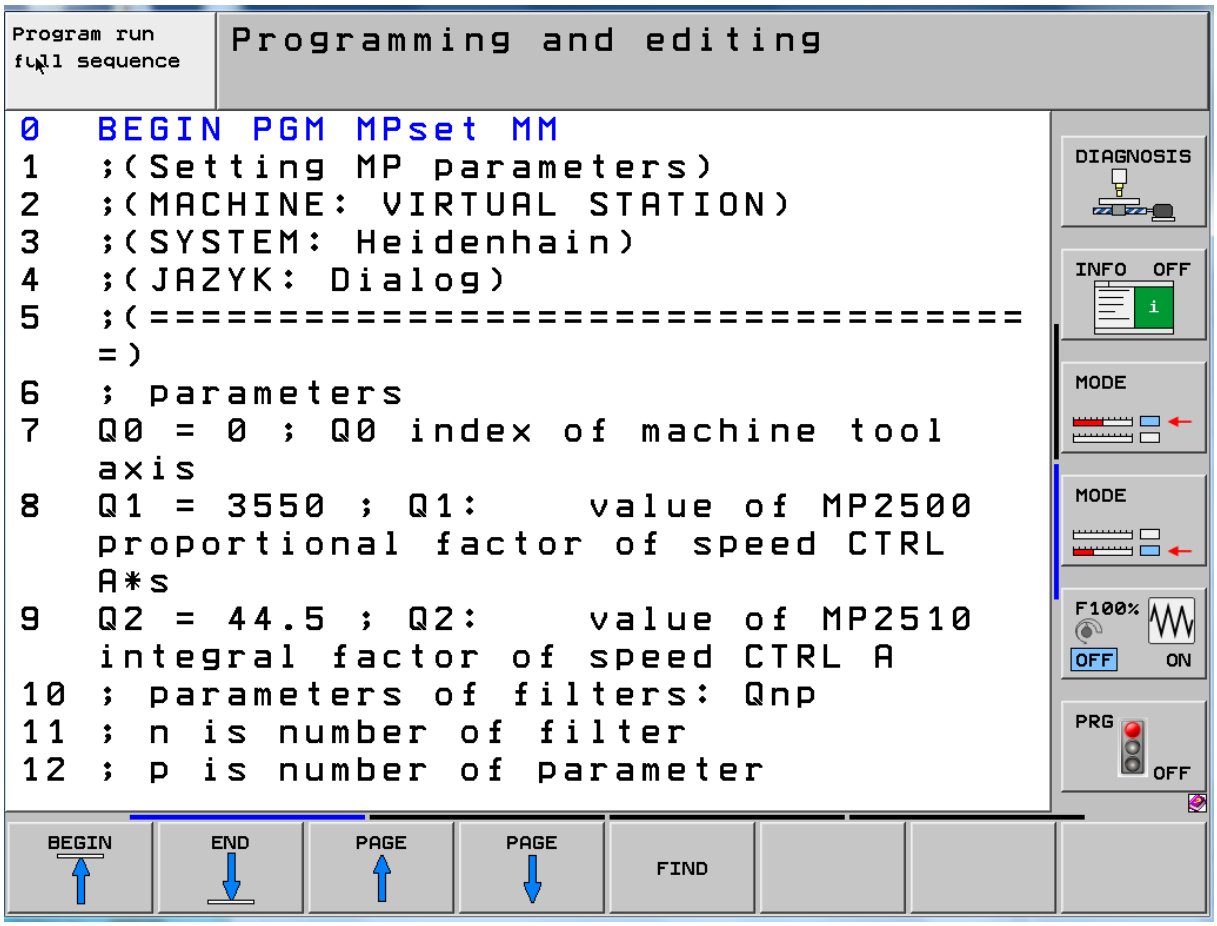



*Figura 3- 47: sección de archivo MPSet.H modificada por el código Python. Extraída de software Programing Station iTNC 530*

En esta segunda figura, el script de Python ya ha finalizado, y el proceso de cambio de parámetros se ha realizado en el archivo de MPSet.H. Se puede observar que los valores de Q1 y Q2, que representan los parámetros MP2500 Y MP2510 respectivamente han sido cambiado con respecto a aquellos en la figura 3-47.

## 3.7 Instalación de algoritmo en la máquina Kovosvit MAS MCV 1000

Para finalizar el desarrollo del proyecto, se realizó la instalación y compilación de la rutina del PLC, de los archivos de Python y de texto en la máquina Kovosvit MCV 1000 MAS, para el correcto funcionamiento del algoritmo de control de masa. Existe una gran diferencia entre la estación de programación y el sistema de control de esta máquina en particular, que se debe tomar en cuenta. El sistema de control de la máquina en el RCMT solo puede compilar archivos de tipo PLC, y no de tipo SCR como los que se crean en el software de diseño de código PLC.

Por lo tanto, como se explica en el anexo de “Compilación de código y transferencia a estación de programación” (Anexo 9-12), se debe compilar el código con la opción de transmitir código PLC y no código fuente.

### 3.7.1 Definición de variables locales en la máquina

Con la correcta compilación del código todos los archivos deberían transmitirse como tipo PLC y la estación de control de la máquina estaría lista para correr la rutina de manera correcta. Sin embargo, ya que el PLC de la máquina fue programado para no ser modificado de ninguna manera, no existe una manera sencilla de definir variables globales ya que no existe un archivo de PLC para definición de variables globales como si lo existe en código fuente.

La solución a este problema es definir todas las variables como variables locales con excepción a las dos variables relacionadas con la softkey creada. Para definir variables locales y transmitirlas a la máquina como código PLC se debe seguir la siguiente sintaxis.

```
#define /s/c    DL_line_read                    D

#define /s/c    KL_mode_asynchronous            K+1
```

*Figura 3- 48: definición de variables locales para transmisión a la máquina. Extraída del software PLC Design*

El / s y / c son para la transmisión a la máquina. La máquina sólo acepta archivos PLC y no archivos SRC, por lo que con el / c en cada variable se transmiten las variables locales y no es necesario agregarlas a ningún archivo de definición. También es necesario definir un rango para estas variables locales, por lo que no toman el espacio de otras variables. Para definir un rango se utiliza el siguiente comando:

```
#define /mn 6000 6499

#define /bn 6000 6499
```

*Figura 3- 49: definición de rangos para variables locales. Extraída del software PLC Design*

El primer comando se usa para definir el rango de marcadores. El segundo para definir el rango de palabras, palabras dobles y bits. Para ver si estos espacios en la memoria están libres, se debe verificar con la tabla en la ventana de edición del PLC en la estación de programación. En estas tablas se puede ver todas las variables de existencia y cuáles están libres.

No existe una manera fácil de definir una variable global en la máquina. Por lo tanto, se recomienda crear sólo variables locales. Pero, en caso de utilizar una softkey, las variables deben ser globales. Así que antes de asignar marcadores globales a la softkey, se debe comprobar si esos marcadores están disponibles y no están asignados a ninguna otra variable. Para comprobar, se utiliza la softkey de tabla en la sección de edición del PLC en la estación de programación.

### 3.7.2 Definición de variables globales en la máquina

Para agregar o modificar una softkey, se debe ir al archivo denominado Softkey_MCV_A.spj. En este archivo, todas las SoftKeys se definen con sus respectivas variables. Las variables se definen en el archivo Sofkey_MCV_A.def.

En este proyecto se modificó la tecla virtual Word de una acción a una softkey de verificación y le asignamos dos marcadores, uno representando el estado y el otro el habilitar. En las siguientes imágenes se muestra el código original a esta softkey y el nuevo código que lo reemplaza.

```
ACTION SK_Python_world PyName:WORLD PyMem:5 PYSCRIPT:PLC:\PYTHON\Test\pos.py
```

*Figura 3- 50: código original de la softkey como softkey de acción.*

```
CHECK SK_Python_world STATUS:M6498 ENABLE:6499
```

*Figura 3- 51: nuevo código para softkey para correr rutina de PLC.*

Los marcadores utilizados son M6498 y M6499, ambos son globales de ahora en adelante, y se asignan a la softkey.

Con las variables locales y globales definidas, y todos los archivos en formato PLC se debe compilar el PLC en la ventana de edición de PLC (sección 2.4.1). Con esto, la rutina de PLC está lista para correr. Por último, se deben agregar los archivos de Python y los archivos de texto a utilizar en alguna carpeta dentro del PLC y verificar que el código Python use las direcciones correctas para acceder a estos archivos.

### 3.7.3 Resultados

En la figura 3-53 podemos ver el resultado de la aproximación, donde se hizo el ajuste individual para cada carga. Se muestra la desviación de posición para el ajuste original con Kv = 4 [m / (min * mm)] y para un ajuste optimizado con Kv = 6,6 [m / (min * mm)]. En nuestro ejemplo el ajuste original fue hecho para la tabla vacía. A continuación, la tabla se cargó por pieza y la desviación de posición se ve en la figura 3-53 (curva verde). Después de que el proceso de la optimización fue hecho la desviación de la posición mejora de 0.075 grados a aproximadamente 0.045 grados (curva roja). Esta mejora representa un 40% de menor desviación de posición con el ajuste hecho. El momento de inercia se incrementó en 1,2.

En la primera vista hay una mejora significativa, mayor ganancia con menor y más suave desviación de posición. Obsérvese que la suposición básica para este resultado fue el ajuste correcto del bucle de velocidad. Después de la optimización de la ganancia del controlador de velocidad y de los filtros de corriente, la mejora respecto al juego original fue: La ganancia proporcional del controlador de velocidad KP se incrementó en 1,9, la ganancia global del lazo de control de velocidad KR fue 2,5 más alta, la ganancia de posición fue 1,65

más alta. Significa mayor rigidez dinámica del eje de avance. Se puede observar que el ajuste original de control de velocidad y posición era robusto para tipos de variedad de piezas de trabajo, pero no era óptimo para cada uno de ellos: la dinámica de la máquina disponible no se aprovechó al máximo. La configuración optimizada permite aprovechar al máximo la dinámica de la máquina disponible.

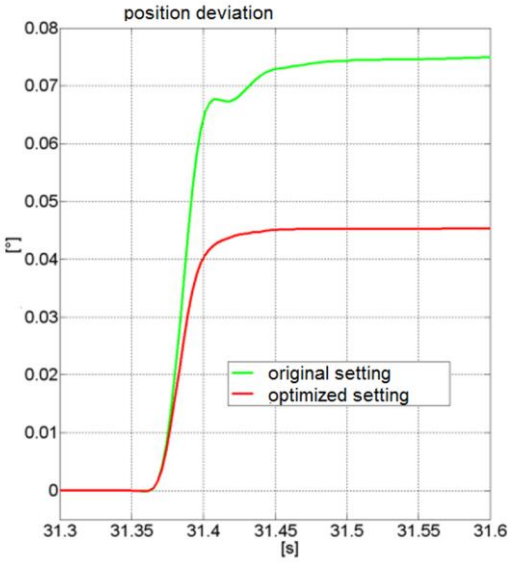



*Figura 3- 52: resultado de la configuración optimizada de las características dinámicas de la máquina. Gráfica elaborada por el RCMT a partir de los resultados presentados en este proyecto.*

# 4 Verificación de Objetivos

En este capítulo del documento se realiza la verificación del cumplimiento de los objetivos planteados en la sección 1.2 (Objetivos del Proyecto). Con esto se busca demostrar, de manera resumida, la labor realizado durante el proyecto y que esta fue de acuerdo con los objetivos propuestos. En el caso de que un objetivo no se cumpliera, se especificaría la razón por el cual no se pudo cumplir con el mismo.

## 4.1 Calculo de masa

En la sección 3.3 del desarrollo de la solución se especificó el proceso por el cual se realizó el cálculo de la carga de un proceso de la máquina. Para esto se usaron el archivo de ajustes y entradas (sección 3.1) y los archivos de calibración y medición (sección 3.2). Se usaron los archivos de calibración y de medición para obtener los datos de posición, de velocidad, de aceleración y de torque de un proceso previamente medido. Con estos datos se creó un proceso Python capaz de leer y de procesar los datos de estos archivos.

El proceso Python utiliza el archivo de calibración para calibrar los valores de corriente obtenidos a partir de los valores de torque, ya que en estos valores existe un desfase relacionado a la posición. Una vez obtenido los datos calibrados, se utiliza el archivo de medición para encontrar la masa del proceso. Esto se realiza utilizando la segunda ley de Newton por media de la obtención del cambio de aceleración y de la fuerza dada por los motores. Con esto se verifica el indicador del primer objetivo específico del proyecto.

El segundo objetivo específico, planteaba realizar pruebas en la estación de programación donde la masa mostrada no tuviera una incertidumbre mayor al 5%. Este objetivo se cumple en la sección 3.4, donde se implementa el cálculo anterior a la estación de programación del sistema de control del iTNC 530.

Para cumplir el objetivo se creó una softkey que corriera el proceso Python y al finalizar dicho proceso se creara una ventana que mostrara al usuario la masa calculada y su incertidumbre asociada. Para el proceso de prueba con los archivos de calibración y de medición dados por el RCMT la masa calculada fue de: 22786.45 kg con una incertidumbre asociada de 398.11 kg. Cabe destacar que estos valores de carga no son los de un proceso en la máquina Kovosvit MAS MCV 1000 sino de otra máquina con una mesa de trabajo mucho más pesada que la de la máquina a probar en este proyecto. Por esta razón el valor de masa es tan grande. Sin embargo, el indicador del objetivo se cumple ya que se calculó correctamente el valor de masa (el valor fue comparado con el valor teórico del RCMT, de 22699 kg) con una incertidumbre que no supera el 5% del proceso.

Cabe destacar de que en caso de que el proceso tenga más de 5% de incertidumbre un error será mostrado en pantalla dando a conocer el error. El valor de 5% puede ser cambiado por el usuario dentro del archivo de ajustes y entradas.

## 4.2 Programación PLC

La segunda sección de los objetivos específicos era la de crear una rutina que se encargara de realizar todo el proceso deseado de manera automática. Esta solución es desarrollada en la sección 3.5, aquí solo se resumirá lo realizado y se verificará si el objetivo propuesto fue cumplido.

Se creó una nueva softkey que se encargara de activar el proceso un proceso CN, seguido por un proceso Python y finalmente activar un segundo proceso CN. Todo esto con solo presionar la softkey creada. La lógica y el funcionamiento del proceso se especifica en la sección 3.5.4 (Lógica del algoritmo de control).

El primer proceso CN llamado por la rutina del PLC es el proceso de calibración y de medición de la máquina. Este proceso genera dos archivos (calibración.dta y medición.dta) que serán usado por el proceso Python llamado al finalizar la creación de los archivos. Luego la rutina llama al proceso Python que calcula la carga del proceso a partir de los archivos generados. Y por último se llama al proceso CN que se encarga de cambiar los parámetros en la maquina (ver sección 3.6.1). Los parámetros son leídos de una tabla en el archivo de ajustes y entradas, y como se especifica en la sección 3.2, esta tabla contiene dos modos de valores de parámetros dados por el RCMT. El proceso Python debe elegir el modo según el valor de la carga encontrada.

La rutina logra cumplir con el objetivo propuesto, sin embargo, ya que para el momento de su creación el código Python aún no podía cambiar los parámetros de la máquina, a primera instancia la rutina no podía variar entre los dos modos posibles. Solo podía variar en un modo. Sin embargo, al cumplir el siguiente objetivo, se verifica que la rutina Python pueda variar entre los dos modos y muestre el cambio de parámetros. Por lo tanto, el objetivo es cumplido. Ver sección 3.6.4 para ver los dos modos en el segundo proceso CN.

## 4.3 Cambio de parámetros

La tercera sección de los objetivos específicos dictaba realizar el proceso Python capaz de realizar el cálculo de masa y el cambio de parámetros por si solo. Ya que para el cumplimiento del primer objetivo ya se había creado un proceso Python que pudiera calcular la carga a partir de los datos de los archivos de calibración y de medición, lo único faltante para cumplir el objetivo era que el código Python lograra realizar el cambio de parámetros.

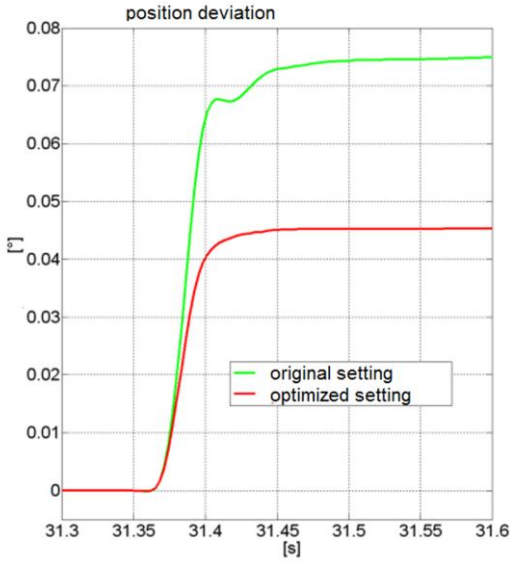
En la sección 3.6 se desarrolla el proceso para realizar el cambio de parámetros por medio del código Python, aquí tan solo se presenta un resumen para verificar el cumplimiento del objetivo planteado. Cabe destacar que el reto en esta sección fue desencriptar el archivo CN que realizaba el cambio de parámetros por media de la rutina del PLC creada anteriormente.

Para desencriptar el archivo se llamó desde Python y se encontró que Heidenhain usaba un código numérico similar al punto flotante de doble precisión según IEEE (ver sección 2.5.2). Con este conocimiento se pudo cambiar los parámetros leídos en la tabla de ajustes y entradas al código de encriptación Heidenhain en lenguaje hexadecimal. Luego se encontró en el código encriptado las posiciones de los cambios de parámetro en el archivo CN y se realizó el cambio en lenguaje hexadecimal. Finalmente se reescribió el archivo encriptado y se abrió desde la estación de programación para ver su correcto funcionamiento. Ver resultados en la sección 3.6.4

## 4.4 Implementación de proceso en la máquina

Por último, era necesario implementar la rutina del PLC creada y el proceso Python a la máquina Kovosvit MAS MCV 1000 y obtener resultados reales en la máquina. Esto se desarrolla en la sección 3.7. En esta sección se especifica la manera de la implementación de la rutina de PLC y los cambios necesarios para que la máquina la pudiera correr.

Finalmente se muestran los resultados para la máquina al realizar los cambios de parámetros con la rutina PLC y los resultados al no realizar ningún cambio para la misma pieza de trabajo. Se observa en los resultados, en la sección 3.7.3 que el comportamiento dinámico mejora notablemente con el cambio implementado.




Con el algoritmo implementado se obtiene la curva roja con los ajustes optimizados, donde se observa un menor tiempo de asentamiento (10 ms) y una desviación de posición mucho menor (0.045 grados). Esto presenta una mejora a los ajustes originales con un tiempo de asentamiento de aproximademte 20 ms y y una desviación de posición de 0.075 grados.

Cabe destacar que más pruebas se realizarán en el RCMT en los siguientes meses, pero en este proyecto solo se pudo realizar la prueba de los resultados mostrados en la sección 3.7.3, ya que el tiempo del internado se acabó y además el contrato de confidencialidad no permitía acceder a más de dos modos distintos de parámetros de máquinas.

# 5 Análisis de presupuesto

En esta sección se lleva a cabo el presupuesto total necesario para realizar el proyecto de investigación planteado en este documento. En la tabla 5-1 se pueden observar estos resultados.

*Tabla 5- 1: presupuesto del proyecto. Elaboración propia*

| Gasto | Precio (dólares) |
|---|---|
| Kovosvit MAS MCV 1000 con Sistema de control Heidenhain iTNC 530 | 40 000 $ |
| Recursos eléctricos del taller de investigación | 100 $ |
| Instrumentos de medición para resultados obtenidos | 200 $ |
| Software Python | Gratuito |
| Horas de trabajo | 5 500$ |
| Otros Gastos | 45 800$ |

Cabe destacar que el taller de investigación ya cuenta con la máquina Kovosvit MAS MCV 1000, y con los softwares instalados en sus computadores. Por lo tanto, el costo del proyecto es muy reducido, se podría decir que lo único que se necesita es el tiempo para realizar el mismo y los recursos que se usaran del laboratorio para llevarlo a cabo.

La ganancia financiera que representa este proyecto no se ha podido medir al finalizar el mismo. Sin embargo, el proyecto añade una mejora en la serie de máquinas Kovosvit MAS haciéndolas más atractivas al mercado internacional y nacional. Al mismo tiempo, elimina los tiempos de paro debido a malos ajustes de parámetros de los reguladores de velocidad en industrias que usan este tipo de máquinas. Estos paros representan una pérdida de recursos que varían para cada laboratorio o industria que utiliza esta serie de máquinas. Con los resultados de este proyecto esos paros pueden ser reducidos y el tiempo de ajustar los reguladores de forma manual se realiza en segundos. Reduciendo los tiempos de ajuste, se puede suponer que existirá una ganancia económica en el proceso.

# 6 Conclusiones

- Se desarrolló un proceso Python capaz de leer y procesar los datos de los archivos de calibración y de medición, y obtener a partir de estos el valor de la carga total del sistema y su incertidumbre asociada.

- Se diseñó una rutina de control en el PLC asociada a una softkey creada, que inicia un proceso de CN, seguido por un proceso Python, y termina con un segundo proceso de CN. El primer proceso CN realiza el proceso de calibración y de medición en la máquina y crea los archivos asociados a estos procesos. El proceso Python calcula la carga del proceso y realiza los cambios en el segundo proceso CN acorde al valor obtenido de masa. El segundo macro CN se encarga de cambiar los parámetros dentro de la máquina.

- Se logró desencriptar los archivos de código de CN para ser usados por el proceso Python para así realizar un cambio en ellos, y que este cambio permitiera cambiar los parámetros de la máquina a los elegidos de una tabla de parámetros óptimos. El cambio se lleva a cabo una vez que se corre el macro de CN.

- Se creó un archivo de texto modificable con las entradas y ajustes para el proceso a realizar. Entre estos parámetros se encuentran los parámetros de cambio de unidades a SI para la posición, velocidad y aceleración, como también el parámetro de cambio de proporción entre velocidad lineal y velocidad angular, con estos parámetros se puede usar el proceso para distintas máquinas inclusive si las unidades de medición de sus sensores son distintas entre las mismas máquinas.

- Se implementó el prototipo del algoritmo en la máquina Kovosvit MAS MCV 1000 del RCMT, y se verificó su funcionamiento para un proceso con una pieza de trabajo. Se realizó la comparación del proceso con y sin el algoritmo y se observó una mejor ganancia de tiempo de asentamiento y una menor desviación de posición en el proceso.

# 7 Recomendaciones y Trabajos Futuros.

- El proceso Python consume casi 40 MB de los 64 MB disponibles para procesos Python en el sistema de control iTNC 530. Por esta razón, se recomienda mejorar los procesos del código y encontrar la manera de que este consuma menos memoria del sistema.

- También hay que tomar en cuenta que el proceso Python tarda aproximadamente 17 segundos en concluir en la máquina Kovosvit MAS MCV 1000, y se debe a la cantidad de procesos que debe realizar y la cantidad de datos que debe procesar. Se recomienda encontrar la forma de optimizar el código para que los proceso sean más eficientes y no tarden tanto tiempo en obtener un resultado.

- Se recomienda implementar el algoritmo en máquinas donde la mesa de trabajo y los motores sean de un peso mucho menos considerables que los de la máquina Kovosvit MAS MCV 1000, ya que así el peso de la pieza de trabajo tiene un mayor efecto en las características dinámicas del sistema y el algoritmo de control tendrá resultados más importantes.

- Para trabajos futuros se debe implementar el algoritmo en máquina donde las unidades de medición sean distintas a las de la máquina Kovosvit MAS MCV 1000, para verificar el correcto funcionamiento del algoritmo a nivel universal en máquinas herramientas con sistemas de control HEIDENHAIN.

# 8 Referencias Bibliográficas

A, Olmos. (2004) Cálculo Numérico. Sistemas numéricos y Errores. Recuperado de: http://mmc2.geofisica.unam.mx/cursos/mcst-2007-II/arch/SisNum.pdf

Agüero, J. Dascal, G. Kolaric, L. Buisan, P. (2013). Máquinas Herramientas. Análisis internacional: ASIA-EEUU-BRASIL. Recuperado de: http://www.uba.ar/archivos_secyt/image/Monograf%C3%ADa%20IMH%2001.pdf

Bolívar, F (2012). Módulo Control numérico computarizado. Universidad Nacional abierta y a distancia CCAV Neiva. Recuperado de: http://datateca.unad.edu.co/contenidos/243008/Modulo_Control_Numerico_Computarizado.pdf

El mundo de la Máquina-herramienta (2007). IMHE: Información De Máquinas-herramienta, Equipos y Accesorios, (339), 81-157.

EMO-HANNOVER (2013): “Feria Máquina-herramienta Emo-Hannover 2013”. www.emo-hannover

Díaz, F (agosto del 2008). Programación automática de máquinas CNC. Facultad de estudios superiores Cuatitlán. Departamento de Ingeniería. Laboratorio de tecnología de los materiales. Recuperado de: http://olimpia.cuautitlan2.unam.mx/pagina_ingenieria/mecanica/mat/mat_mec/m4/master_cam.pdf

Gardner Research (2014). The World Machine-Tool Output and Consumption Survey. Recuperado de: https://www.gardnerweb.com/cdn/cms/2014wmtocs_SURVEY.pdf

Galiana, J., Redondo, I. y Estévez, J., (1991): “Introducción de la Máquina-herramienta de CN en las PME”, Instituto Madrileño de Tecnología, IMPI, Madrid.

González, A. y Plaza, M. B. (1994): “El sector de la Máquinaherramienta: Análisis de las economías de escala y el tamaño empresarial”. Ekonomiaz Revista Vasca De Economía, (30), 234-255. Vitoria-Gasteiz, Retrieved from http://dialnet.unirioja.es/descarga/articulo/274320.pdf

HEIDENHAIN. (Julio 2010). User´s Manual Heidenhain Conversational Programing iTNC 530.

HEIDENHAIN (Febrero 2011). Technical Manual iTNC 530.

HEIDENHAIN (Febrero 2014). Technical Manual. Python in HEIDENHAIN Controls.

HEIDENHAIN (2007). Python in HEIDENHAIN NC. User documentation for the programing language Python in the Heidenhain NC.

HEIDENHAIN (2007). PyJH: Python in the HEIDENHAIN NC.

HEIDENHAIN (29 de octubre del 2008). NC Error Messages.

Herriko, E (s.f). Máquinas-Herramienta: Funciones, tipos y arquitecturas. Tecnologías de fabricación y tecnología de máquinas. DPTO. De ingeniería mecánica. Universidad del país Vasco. Recuperado de: http://www.ehu.eus/manufacturing/docencia/419_ca.pdf

Icex (2013): "Impulso internacional de la Máquina-herramienta". http://www.icex.es/icex/es/navegacion-superior/revista-el-exportador/ observatorio2/el-impulso-internacional-de-la-maquina-herramienta.html.

Jiménez, R (s.f). Control Numerico por Computadora (CNC). Recuperado de: http://materias.fi.uba.ar/7565/U4-control-numerico-por-computadora.pdf

Python (20 de mayo del 2004). Python documentation.

RCMT (2016) RCMT today. Recuperado de: http://www.rcmt.cvut.cz/about-us/en/today

Van Rossum, G (Septiembre 2009). Tutorial de Python. Recuperado de: http://docs.python.org.ar/tutorial/pdfs/TutorialPython2.pdf

# 9 Anexos

## 9.1 Macro para proceso de Calibración y de Medición.

```
0  BEGIN PGM INERT_MEAS_VS MM
1  ;(INERTIA MEASUREMENT)
2  ;(MACHINE: VIRTUAL STATION)
3  ;(SYSTEM: Heidenhain)
4  ;(JAZYK: Dialog)
5  ;(==================================)
6  ;parameters
7  Q0 = 12000 ; FEED RATE measurement mm/min
8  Q20 = 2000 ; FEED RATE calibratiion mm/min
9  Q1 = 100 ; Measurement distance mm
10 Q2 = 30 ; Calibration overlap mm
11 Q21 = Q1 + Q2 + Q2 ; Calibration distance mm
12 Q3 = 0.4 ; Acceleration limit m/s2
13 Q4 = 0 ; Acceleration backup m/s2
14 Q5 = 0.5 ; time delay s
15 Q6 = 0 ; Start position of measurement mm
16 Q26 = Q6 - Q2 ; Start position of calibration mm
17 Q7 = Q6 + Q1 ; End position of measurement mm
18 Q27 = Q26 + Q21 ; End position of calibration mm
19 Q8 = 0 ; Inicialization position mm
20 ;(==================================)
21 L  X+Q8  Y+0  Z+0 F3000 ; Inicialization position
22 ;(Calibration)
23 L  X+Q26  Y+0  Z+0 F3000 ; Starting position
24 CYCL DEF 9.0 DWELL TIME
25 CYCL DEF 9.1 DWELL1.5
26 FUNCTION SCOPE START JOB "plc:\IMTem_.dta" RESULT "plc:\IMCal_.dta"
27 CYCL DEF 9.0 DWELL TIME
28 CYCL DEF 9.1 DWELL1.5
29 L  X+Q27 FQ20 ; calibration movement positive part
30 CYCL DEF 9.0 DWELL TIME
31 CYCL DEF 9.1 DWELLQ5
32 L  X+Q26 FQ20 ; calibration movement negative part
33 CYCL DEF 9.0 DWELL TIME
34 CYCL DEF 9.1 DWELL1.5
35 FUNCTION SCOPE STOP
36 FUNCTION SCOPE STORE
37 CYCL DEF 9.0 DWELL TIME
38 CYCL DEF 9.1 DWELL1.5
39 ;(==================================)
40 ;(Measurement)
41 L  X+Q6  Y+0  Z+0 F3000 ; Starting position
42 FN 18: SYSREAD Q4 = ID1000 NR1060 IDX0 ; Acceleration backup by reading from MP1060.IDX
43 FN 17: SYSWRITE ID 1000 NR1060 IDX0 =+Q3 ; Acceleration limitation by writing into MP1060.IDX
44 CYCL DEF 9.0 DWELL TIME
45 CYCL DEF 9.1 DWELL1.5
46 FUNCTION SCOPE START JOB "plc:\IMTem_.dta" RESULT "plc:\IMMea_.dta"
47 CYCL DEF 9.0 DWELL TIME
48 CYCL DEF 9.1 DWELL1.5
49 L  X+Q7 FQ0 ; measurement movement positive part
50 CYCL DEF 9.0 DWELL TIME
51 CYCL DEF 9.1 DWELLQ5
52 L  X+Q6 FQ0 ; measurement movement negative part
53 CYCL DEF 9.0 DWELL TIME
54 CYCL DEF 9.1 DWELL1.5
55 FUNCTION SCOPE STOP
56 FUNCTION SCOPE STORE
57 CYCL DEF 9.0 DWELL TIME
58 CYCL DEF 9.1 DWELL1.5
59 FN 17: SYSWRITE ID 1000 NR1060 IDX0 =+Q4 ; Acceleration restore by writing into MP1060.IDX
60 L  X+Q8 F3000 ; return to inicialization position
61 END PGM INERT_MEAS_VS MM
```

## 9.2 Macro para cambio de parámetros

```
0  BEGIN PGM MPsetTemp MM
1  ;(Setting MP parameters)
2  ;(MACHINE: VIRTUAL STATION)
3  ;(SYSTEM: Heidenhain)
4  ;(JAZYK: Dialog)
5  ;(==================================)
6  ; parameters
7  Q0 = 0 ; Q0 index of machine tool axis
8  Q1 = 3350 ; Q1:    value of MP2500 proportional factor of speed CTRL A*s
9  Q2 = 43.5 ; Q2:    value of MP2510 integral factor of speed CTRL A
10 ; parameters of filters: Qnp
11 ; n is number of filter
12 ; p is number of parameter
13 ;      p=1 means type of filter
14 ;      p=2 means frequency of filter Hz
15 ;      p=3 means damping of filter dB
16 ;      p=4 means bandwidth of filter Hz
17 Q11 = 0 ; Q11:     value of MP2562 type of filter
18 Q12 = 250 ; Q12:   value of MP2552 frequency of filter Hz
19 Q13 = 20 ; Q13:    value of MP2542 damping of filter dB
20 Q14 = 50 ; Q14:    value of MP2572 bandwidth of filter Hz
21 Q21 = 0 ; Q21:     value of MP2563 type of filter
22 Q22 = 250 ; Q22:   value of MP2553 frequency of filter Hz
23 Q23 = 20 ; Q23:    value of MP2543 damping of filter dB
24 Q24 = 50 ; Q24:    value of MP2573 bandwidth of filter Hz
25 Q31 = 0 ; Q31:     value of MP2564 type of filter
26 Q32 = 250 ; Q32:   value of MP2554 frequency of filter Hz
27 Q33 = 20 ; Q33:    value of MP2544 damping of filter dB
28 Q34 = 50 ; Q34:    value of MP2574 bandwidth of filter Hz
29 Q41 = 0 ; Q41:     value of MP2565 type of filter
30 Q42 = 250 ; Q42:   value of MP2555 frequency of filter Hz
31 Q43 = 20 ; Q43:    value of MP2545 damping of filter dB
32 Q44 = 50 ; Q44:    value of MP2575 bandwidth of filter Hz
33 Q51 = 0 ; Q51:     value of MP2566 type of filter
34 Q52 = 250 ; Q52:   value of MP2556 frequency of filter Hz
35 Q53 = 20 ; Q53:    value of MP2546 damping of filter dB
36 Q54 = 50 ; Q54:    value of MP2576 bandwidth of filter Hz
37 ;(==================================)
```

```
38 ; setting of parameters
39 ; speed CTRL
40 FN 1  7: SYSWRITE ID 1000 NR2500 IDXQ0 =+Q1 ; writing into MP2500.IDX
41 FN 17: SYSWRITE ID 1000 NR2510 IDXQ0 =+Q2 ; writing into MP2510.IDX
42 ; filter 1
43 FN 17: SYSWRITE ID 1000 NR2562 IDXQ0 =+Q11 ; writing into MP2562.IDX
44 FN 17: SYSWRITE ID 1000 NR2552 IDXQ0 =+Q12 ; writing into MP2552.IDX
45 FN 17: SYSWRITE ID 1000 NR2542 IDXQ0 =+Q13 ; writing into MP2542.IDX
46 FN 17: SYSWRITE ID 1000 NR2572 IDXQ0 =+Q14 ; writing into MP2572.IDX
47 ; filter 2
48 FN 17: SYSWRITE ID 1000 NR2563 IDXQ0 =+Q21 ; writing into MP2563.IDX
49 FN 17: SYSWRITE ID 1000 NR2553 IDXQ0 =+Q22 ; writing into MP2553.IDX
50 FN 17: SYSWRITE ID 1000 NR2543 IDXQ0 =+Q23 ; writing into MP2543.IDX
51 FN 17: SYSWRITE ID 1000 NR2573 IDXQ0 =+Q24 ; writing into MP2573.IDX
52 ; filter 3
53 FN 17: SYSWRITE ID 1000 NR2564 IDXQ0 =+Q31 ; writing into MP2564.IDX
54 FN 17: SYSWRITE ID 1000 NR2554 IDXQ0 =+Q32 ; writing into MP2554.IDX
55 FN 17: SYSWRITE ID 1000 NR2544 IDXQ0 =+Q33 ; writing into MP2544.IDX
56 FN 17: SYSWRITE ID 1000 NR2574 IDXQ0 =+Q34 ; writing into MP2574.IDX
57 ; filter 4
58 FN 17: SYSWRITE ID 1000 NR2565 IDXQ0 =+Q41 ; writing into MP2565.IDX
59 FN 17: SYSWRITE ID 1000 NR2555 IDXQ0 =+Q42 ; writing into MP2555.IDX
60 FN 17: SYSWRITE ID 1000 NR2545 IDXQ0 =+Q43 ; writing into MP2545.IDX
61 FN 17: SYSWRITE ID 1000 NR2575 IDXQ0 =+Q44 ; writing into MP2575.IDX
62 ; filter 5
63 FN 17: SYSWRITE ID 1000 NR2566 IDXQ0 =+Q51 ; writing into MP2566.IDX
64 FN 17: SYSWRITE ID 1000 NR2556 IDXQ0 =+Q52 ; writing into MP2556.IDX
65 FN 17: SYSWRITE ID 1000 NR2546 IDXQ0 =+Q53 ; writing into MP2546.IDX
66 FN 17: SYSWRITE ID 1000 NR2576 IDXQ0 =+Q54 ; writing into MP2576.IDX
67 END PGM MPsetTemp MM
```

## 9.3 Lista complete de commandos para PLC

| Group of functions | Syntax | Function |
|---|---|---|
| **Loading and saving commands** | | |
| | L | Load |
| | LN | Load NOT |
| | L– | Load two's complement |
| | LB | Load BYTE |
| | LW | Load WORD |
| | LD | Load DOUBLE WORD |
| | = | Assignment |
| | B= | Assign BYTE |
| | W= | Assign WORD |
| | D= | Assign DOUBLE WORD |
| | =N | Assign NOT |
| | =– | Assign two's complement |
| **Setting commands** | | |
| | S | Set |
| | R | Reset |
| | SN | Set NOT |
| | RN | Reset NOT |
| **Logical operations** | | |
| | A | And |
| | AN | And NOT |
| | O | or |
| | ON | Or NOT |
| | XO | Exclusive OR |
| | XON | Exclusive OR NOT |
| **Arithmetical commands** | | |
| | + | Addition |
| | – | Subtraction |
| | x | Multiplication |
| | / | Division |
| | MOD | Remainder |

| Group of functions | Syntax | Function |
|---|---|---|
| **Increment** | | |
| | INC | Increment operand |
| | INCW | Increment word accumulator |
| | INCX | Increment index register |
| **Decrement** | | |
| | DEC | Decrement operand |
| | DECW | Decrement word accumulator |
| | DECX | Decrement index register |
| **Comparisons** | | |
| | == | Equal to |
| | < | Less than |
| | > | Greater than |
| | <= | Less than or equal to |
| | >= | Greater than or equal to |
| | <> | Not equal to |
| **Parenthetical expression in logical operations** | | |
| | A[ ] | And [ ] |
| | AN[ ] | And NOT [ ] |
| | O[ ] | Or [ ] |
| | ON[ ] | Or NOT [ ] |
| | XO[ ] | Exclusive OR [ ] |
| | XON[ ] | Exclusive OR NOT [ ] |
| **Parenthetical expressions with arithmetical instructions** | | |
| | +[ ] | Addition [ ] |
| | –[ ] | Subtraction [ ] |
| | x[ ] | Multiplication [ ] |
| | /[ ] | Division [ ] |
| | MOD[ ] | Remainder [ ] |
| **Parenthetical expressions in comparisons** | | |
| | ==[ ] | Equal to [ ] |
| | <[ ] | Less than [ ] |
| | >[ ] | Greater than [ ] |
| | <=[ ] | Less than or equal to [ ] |
| | >=[ ] | Greater than or equal to [ ] |
| | <>[ ] | Not equal to [ ] |

| Group of functions | Syntax | Function |
|---|---|---|
| **Shifting commands** | | |
| | << | Shift left |
| | >> | Shift right |
| **Bit commands** | | |
| | BS | Bit set |
| | BC | Bit clear |
| | BT | Bit test |
| **Stack operations** | | |
| | PS | Push data onto the data stack |
| | PL | Pull data from the data stack |
| | PSL | Push logic accumulator onto the data stack |
| | PSW | Push word accumulator onto the data stack |
| | PLL | Pull logic accumulator from the data stack |
| | PLW | Pull word accumulator from the data stack |
| **Jump commands** | | |
| | JP | Unconditional jump |
| | JPT | Jump if logic accumulator = 1 |
| | JPF | Jump if logic accumulator = 0 |
| | CM | Call module |
| | CMT | Call module if logic accumulator = 1 |
| | CMF | Call module if logic accumulator = 0 |
| | EM | End of module, program end |
| | EMT | End of module if logic accumulator = 1 |
| | EMF | End of module if logic accumulator = 0 |
| | LBL | Label |

## 9.4 Función para convertir un numero de decimal a encriptación HEIDENHAIN

```
def double2hex(x):
    d=['0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f']
    real_min=2**(-1022)
    s = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    if x == 0:
        s=['0','0','0','0','0','0','0','0','0','0','0','0','0','0','0','0']
        return s
    if abs(x)<real_min:
        log=[]
        log[0] = 0
        log[1] = 16*abs(x)/real_min
    else:
        log=[]
        log = log_2(abs(x))
        log[0]=log[0]+1022
        log[1]=float(32*(log[1]-0.5))

    if x > 0:
        r=float(log[0]/256)
        F=[]
        F.append(str(r).split('.'))
        s[0]=d[int(F[0][0])]
    else:
        r=float(log[0]/256)
        F=[]
        F.append(str(r).split('.'))
        s[0]=d[int(F[0][0])+8]

    r=float(rem_pos(log[0],256)/16)
    F=[]
    F.append(str(r).split('.'))
    s[1]=d[int(F[0][0])]


    r=float(rem_pos(log[0],16))
    F=[]
    F.append(str(r).split('.'))
    s[2]=d[int(F[0][0])]

    for j in range(3,16):
        s[j]=d[int(math.floor(log[1]))]
        log[1]=16*rem_pos(log[1],1)

    myorder=[14,15,12,13,10,11,8,9,6,7,4,5,2,3,0,1]
    s = [ s[i] for i in myorder]
    return s
```

## 9.5 Especificaciones del iTNC 530.

### 9.5.1 Controles del iTNC 530

En la siguiente tabla se muestran las teclas, más importantes para este proyecto, del iTNC 530 y la función de cada una de estas. Esta tabla fue obtenida del manual de usuario de

Heindenhain del iTNC 530 (HEIDENHAIN, Julio 2010). A lo largo de este documento se hace mención de varias de estas teclas, por lo tanto, es importante contar con esta sección que describe la función de cada una de ellas.

*Tabla 9- 1: función de teclas en el sistema de control iTNC 530. Elaboración propia.*

| Tecla | Función |
|---|---|
|  | Diseño de pantalla dividida |
|  | Alternar la pantalla entre el mecanizado y modos de programación |
|  | Teclas programables para seleccionar funciones en pantalla |
|  | Desplazamientos entre filas de teclas programables |
|  | Operar manualmente |
|  | Manivela electrónica |
|  | Posicionamiento con entrada manual de datos |
|  | Ejecución del programa, bloque único |
|  | Ejecución del programa, secuencia completa |
|  | Programar y editar |
|  | Prueba de funcionamiento |
|  | Seleccione o elimine programas y archivos, |
|  | Seleccionar funciones MOD |
|  | Mostrar texto de ayuda para mensajes de error NC.<br>Llama a la guía del iTNC |

| | |
|---|---|
| ERR | Mostrar todos los mensajes de error actuales |
| ENT | Confirmar diálogo de entrada y reanudación |
| CE | Borrar entrada numérica o error TNC mensaje |
| END | Concluir bloque y salir |
| DEL | Abortar diálogo, borrar sección de programa |

### 9.5.2 Interfaz del programa

El iTNC 530 se puede programar directamente desde el centro de maquinado, sin embargo, también se puede contar con una estación de programación desde cualquier computadora para tener una facilidad para programar y simular los procesos. En la siguiente figura se muestra como es la interfaz de esta estación de programación.

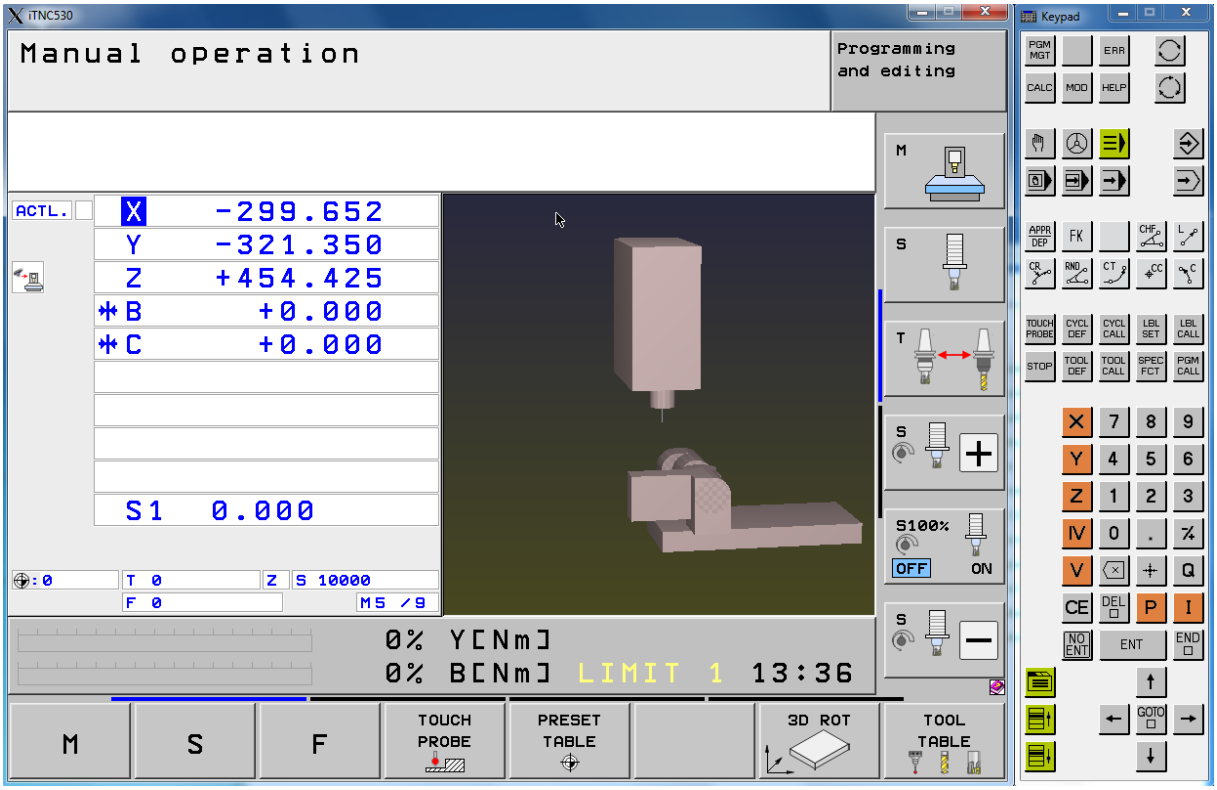



*Figura 9- 1: Interfaz de la estación de programación del iTNC 530, pantalla de operación manual. Extraída de interfaz del software Programming Station iTNC 530.*

La estación de programación se divide en dos partes, la de operación y simulación, y la de programación y edición. En la figura 9-1 observamos la interfaz de operación y simulación.

Podemos observar que la estación cuenta con teclas virtuales también conocidas como ‘softkeys’ tanto abajo y al lado derecho de la pantalla. También, la estación viene acompañada con un teclado virtual a la derecha, con las mismas teclas que se encuentran en la máquina-herramienta.

En la pantalla de operación y simulación, se puede ver el proceso de mecanizado de la pieza simulado, encontrar errores y corregirlos. En esta pantalla corre el código CN y se puede observar la posición de la herramienta por cada bloque y también la herramienta que está en uso.

Cabe destacar que para este trabajo el verdadero peso está en la otra gran sección de la estación de programación, la sección de programación y edición. Esta pantalla se puede observar en la siguiente figura:

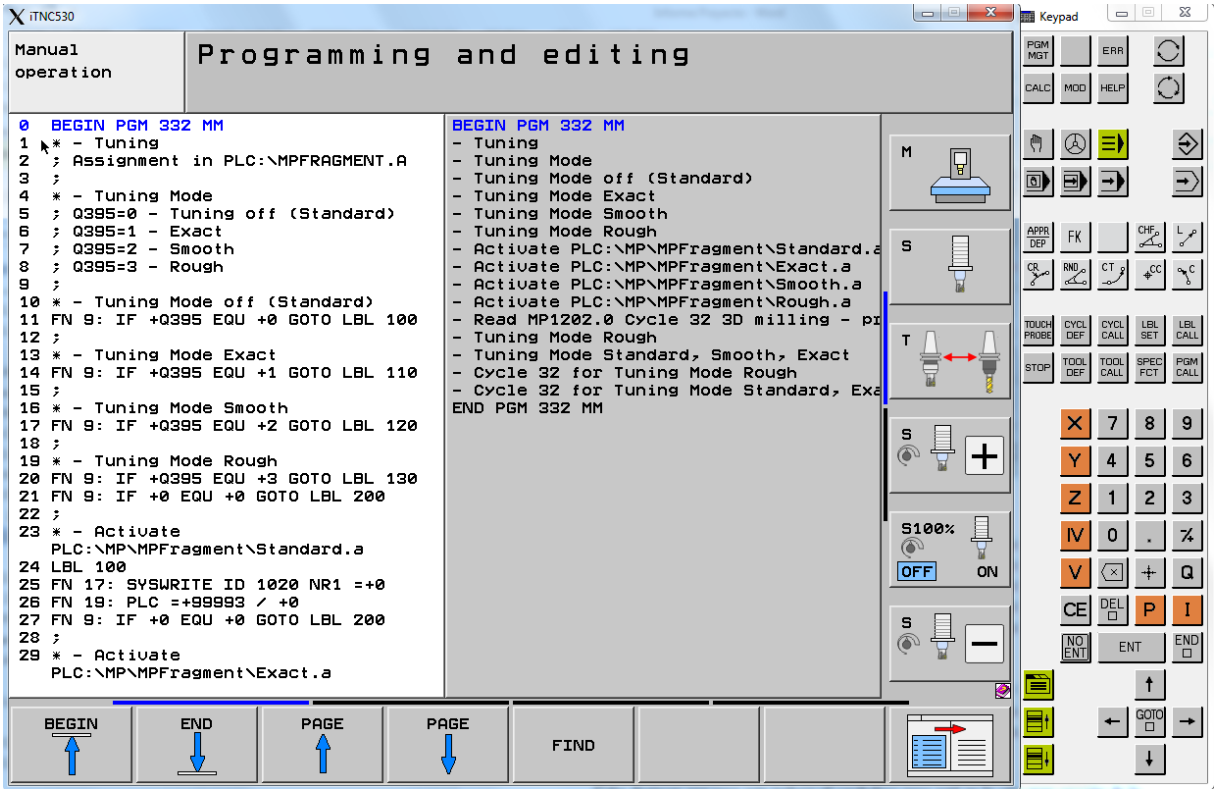



*Figura 9- 2: Interfaz de la estación de programación del iTNC 530, pantalla de programación y edición. Extraída de interfaz del software Programming Station iTNC 530.*

En esta interfaz se puede editar y corregir el código CN, como también entrar a distintas partes del software con la tecla MOD, entre las cuales están el PLC y los parámetros de la máquina.

### 9.5.3 Parámetros de la máquina

Un control de contorno debe tener acceso a datos específicos (por ejemplo, distancias, aceleración) antes de que pueda ejecutar sus instrucciones programadas. Estos datos se definen en parámetros de máquina. Esta lista de parámetros de máquina se divide en grupos de acuerdo con el tema. La tabla 9-1 fue obtenida del manual técnico de Heidenhain del iTNC 530 (HEIDENHAIN, febrero 2011).

*Tabla 9- 1: Parámetros de la máquina. Extraída del manual técnico de Heidenhain del iTNC 530.*

| **Parámetros de la máquina** | **Tema** |
|---|---|
| 10 - 999 | Máquinas y encoders |
| 1000 - 1399 | Posicionamiento |
| 1400 - 1699 | Funcionamiento con control de avance de velocidad. |
| 1700 - 1999 | Funcionamiento con el siguiente error (servo lag) |
| 2000 - 2999 | Control integrado de velocidad y corriente |
| 3000 - 3999 | Huso |
| 4000 - 4999 | PLC |
| 5000 - 5999 | Interfaz de datos |
| 6000 - 6199 | Sonda de contacto 3-D |
| 6500 - 6599 | Medición de herramienta con palpador de palpación |
| 7100 - 7199 | Tapping |
| 7200 - 7349 | Visualización y programación |
| 7350 - 7399 | Colores |
| 7400 - 7599 | Mecanizado y Ejecución del Programa |
| 7600 - 7699 | Hardware |

### 9.5.4 Modificar los parámetros de la máquina

Para modificar los parámetros de la máquina se debe entrar al modo de programación del iTNC y presionar la tecla MOD. Con esta tecla se solicita un código o clave para acceder a distintas áreas del software. Para accede a los parámetros de la máquina se debe ingresar el código: **95148**

Si aún no se han introducido los parámetros de la máquina en el software de HEIDENHAIN (P. Ej., Antes de la puesta en marcha), el iTNC presenta la lista de parámetros

de la máquina después de la prueba de memoria. Se deben introducir los valores de los parámetros de la máquina a mano con el teclado o descargarlos a través de la interfaz de datos.

Se puede introducir los valores de entrada en varios formatos: decimal, binario (%) o hexadecimal ($). Se debe introducir un número para cada parámetro de máquina. El valor representa, por ejemplo, la aceleración en mm / s2 o tensión eléctrica en V. También es posible añadir un comentario precediéndolo con un Punto y coma (;). La entrada binaria (%) es el mejor formato para parámetros de máquina que activan funciones individuales codificadas por bits

Después de introducir todos los valores de los parámetros de la máquina se debe salir de la lista de parámetros de la máquina pulsando la tecla END. Las entradas perdidas o incorrectas generan mensajes de error del control que pedirán que corrija la entrada. Se muestran los siguientes errores:

*Tabla 9- 2: Número y definición de posible error en los parámetros de la máquina. Extraída del manual técnico de Heidenhain del iTNC 530.*

| Numero de error | Definición |
|---|---|
| 0 | No se encontró número de MP |
| 1 | Invalido número de MP |
| 2 | No se encontró separador (;) |
| 3 | Valor de entrada incorrecto |
| 4 | MP doblemente definido |
| 5 | --- |
| 6 | MP no puede ser guardado |

Si el control no reconoce ningún error, automáticamente sale del editor de parámetros de la máquina y está listo para funcionar. Si no se realiza ninguna entrada en la lista de parámetros de la máquina durante la puesta en marcha y al salir del editor con la tecla END, el iTNC genera la lista de parámetros de máquina estándar. En esta lista, el iTNC se define como una estación de programación con los colores estándar HEIDENHAIN. En todos los demás parámetros de la máquina se introduce un valor por defecto.

Puede introducir más de una lista de parámetros de máquina en el iTNC. Para esto, se selecciona las listas con la tecla PGM MGT y la tecla SELECT. La última lista de parámetros de máquina seleccionada se activa cuando sale del editor de parámetros de la máquina

Una lista de parámetros de máquina se puede cambiar con el editor de parámetros de máquina o directamente a través del PLC. La "Lista de parámetros de máquina" incluye los siguientes símbolos:

*Tabla 9- 3: Indicadores para cambiar los parámetros de la máquina. Extraída del manual técnico de Heidenhain del iTNC 530.*

| Indicador | Se puede cambiar por medio de |
|---|---|

| | |
|---|---|
| CN123 | El MP también es accesible a través del número de código 123 |
| PLC | El MP se puede cambiar a través del PLC; También puede ser cambiado en un programa NC en ejecución durante una salida estroboscópica. |
| RUN | El MP también se puede cambiar mientras un programa NC es corriendo |
| RESET | Se pueden cambiar los resultados de un MP en un reset. |
| REF | Para cambiar el MP el eje debe moverse de nuevo sobre la marca de referencia. |

Es importarte destacar que los Parámetros de la máquina solo se pueden sobrescribir cuando los drivers están estacionarios, y no en movimiento.

Existen varias maneras de modificar estos parámetros, pero en cuestión de este proyecto lo importante es saber modificar dichos parámetros desde el PLC. Para esto hay que tomar en cuenta la siguiente tabla de módulos de PLC. Estos también han sido obtenidos del manual técnico de Heidenhain sobre el iTNC 530 (HEIDENHAIN, febrero 2011)

*Tabla 9- 4: Módulos del PLC con relación a los MP. Extraída del manual técnico de Heidenhain del iTNC 530.*

| **Modulo** | **Función** |
|---|---|
| 9031 | Sobrescribir MP |
| 9032 | Leer MP |
| 9310 | Leer MP desde la memoria de tiempo de ejecución |
| 9033 | Seleccionar archivo de MP |
| 9034 | Cargar archivo de MP |
| 9312 | Sobrescribir MP en el archivo activo o en la memoria de tiempo de ejecución |
| 9313 | Leer MP en el archivo activo o en la memoria de tiempo de ejecución |

### 9.5.5 TNCremo

TNCremo es un software proveído por Heidenhain que garantiza una transferencia de datos sin fallos entre el PC y el control cuando se desea utilizar una estación de programación de PC o configurar el acceso remoto al control.

Con TNCremo y Ethernet u otra interfaz de datos, puede transferir bidireccionalmente programas de piezas guardados externamente, tablas de herramientas y tablas de paletas, iniciar

la máquina, crear copias de seguridad del disco duro y probar las condiciones de funcionamiento de la máquina.

## 9.6 Librerías Estándares en el iTNC

La instalación de Python en el iTNC contiene las siguientes librerías estándares de Python. Es importante tomar estas en cuenta a la hora de construir un script. La tabla 9-5 fue obtenida del manual técnico de Python en los controles de Heidenhain (Technical Manual. Python in HEIDENHAIN Controls, febrero 2014).

*Tabla 9- 5: Librearías disponibles en la estación de programación del iTNC. Extraída del manual técnico de Python en los controles de Heidenhain.*

| | | | | |
|---|---|---|---|---|
| Binop | Codeop | Doctest2 | Fpformat | Htmlparser |
| Bisect | Coerción | Dumbdbm | Frozen | Httplib |
| Bool | Compare | Dummy_thread | Funcattrs | Imaplib |
| Bufio | Compile | Dummy_threading | Future | Imp |
| Calendar | Complex | Email | Gc | Import |
| Call | Contains | Enumarte | Generators | Importhooks |
| Capi | Cookie | Eof | Getargs | Inspect |
| Cfgparser | Copy | Erno | Getargs2 | Isintance |
| cgi | Copy_reg | Extcall | Gzip | Iter |
| Charmapcodec | Cpickle | File | Hash | itertools |
| Class | Csv | Filecmp | Heapq | Logging |
| Cmath | Dircache | Fileinput | Hexoct | Long |
| Codecallbacks | dis | Fnmatch | hmac | Long_future |
| Codecs | doctest | format | htmlib | longexp |
| macpath | mailbox | marshal | math | Md5 |
| datetime | descr | descrtut | difflib | Multifile |
| mutants | netrc | new | ntpath | Operator |
| Optparse | os | parser | Pep247 | Pep263 |
| pickle | pickletools | pkg | pkgimport | popen |
| Popen2 | posixpath | pow | pprint | Profile |
| profilehooks | pyclbr | pyexpat | getopt | Gettext |
| glop | global | repr | Rfc822 | Richcmp |
| robotparser | rotor | sax | scope | Select |
| sets | Sgmllib | sha | Shelve | shlex |
| shutil | slice | TcpComm | Softspace | Unicodedata |
| posixpath | sort | univnewlines | pow | Str |
| unpack | pprint | strftime | urllib | Profile |
| string | Urllib2 | profilehooks | stringprep | urlparse |
| pyclbr | strop | userdict | pyexpat | Strptime |
| userlist | queque | struct | userstirng | Quopri |
| structseq | uu | random | sundry | warnings |
| re | symtable | wave | regex | syntax |
| weakref | whichdb | xpickle | zipimport | xmllib |

## 9.7 Funciones PyJH

El módulo PyJH se creó específicamente para usar funciones que comuniquen los scripts de Python con los sistemas de Heidenhain, entre estos sistemas se encuentra el iTNC 530. Este módulo se divide en tres clases o submódulos que se mencionan a continuación con sus respectivas funciones.

**El módulo jh**

- Funciones para solicitar y modificar datos desde el control
- Funciones para controlar el bucle de eventos
- Funciones para interrogar códigos de error
- Función para iniciar la ayuda en línea

**El módulo jh.gtk**

- Clase jh.gtk.Window como expansión de la clase gtk.Window para su inclusión en el diseño de control
- Funciones para registrar y anular el registro de gtk.Window
- Funciones para el control del enfoque y el registro del teclado
- Funciones para seleccionar la pantalla lógica
- Declaración de constantes para tipos de enfoque (jh.focus)
- Declaración de constantes para grupos clave (jh.keys)

**El módulo jh.gtk.glade**

- Clase de jh.gtk.glade.XML como expansión de la clase gtk.glade.XML para su inclusión en el diseño de control
- Función para registrar la ventana gtk.Window del formulario Glade
- Función para crear un jh.gtk.glade.XML desde un búfer en lugar de un archivo

## 9.8 Especificaciones del PLC

### 9.8.1 Acceso al PLC desde el iTNC

Para entrar al PLC y poder configurar sus archivos y rutinas se debe presionar la Tecla MOD en la sección de programación y edición. La siguiente pantalla será revelada.

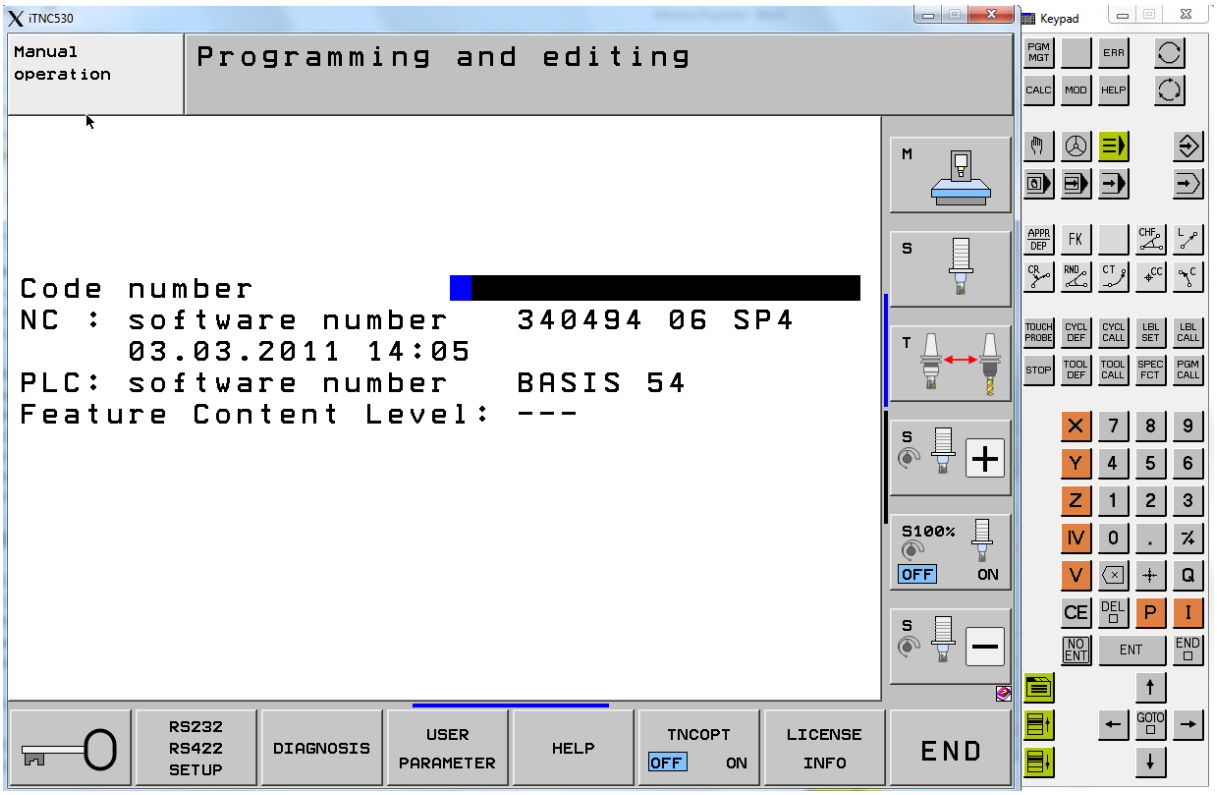



*Figura 9- 3: Pantalla para acceso al PLC. Extraída de interfaz del software Programming Station iTNC 530.*

Luego solo se debe entrar el código, que para entrar al PLC es **807667**. Ya con esto se puede programar el PLC de la máquina y editar las rutinas previas. En la siguiente imagen se observa la pantalla principal de programación en PLC

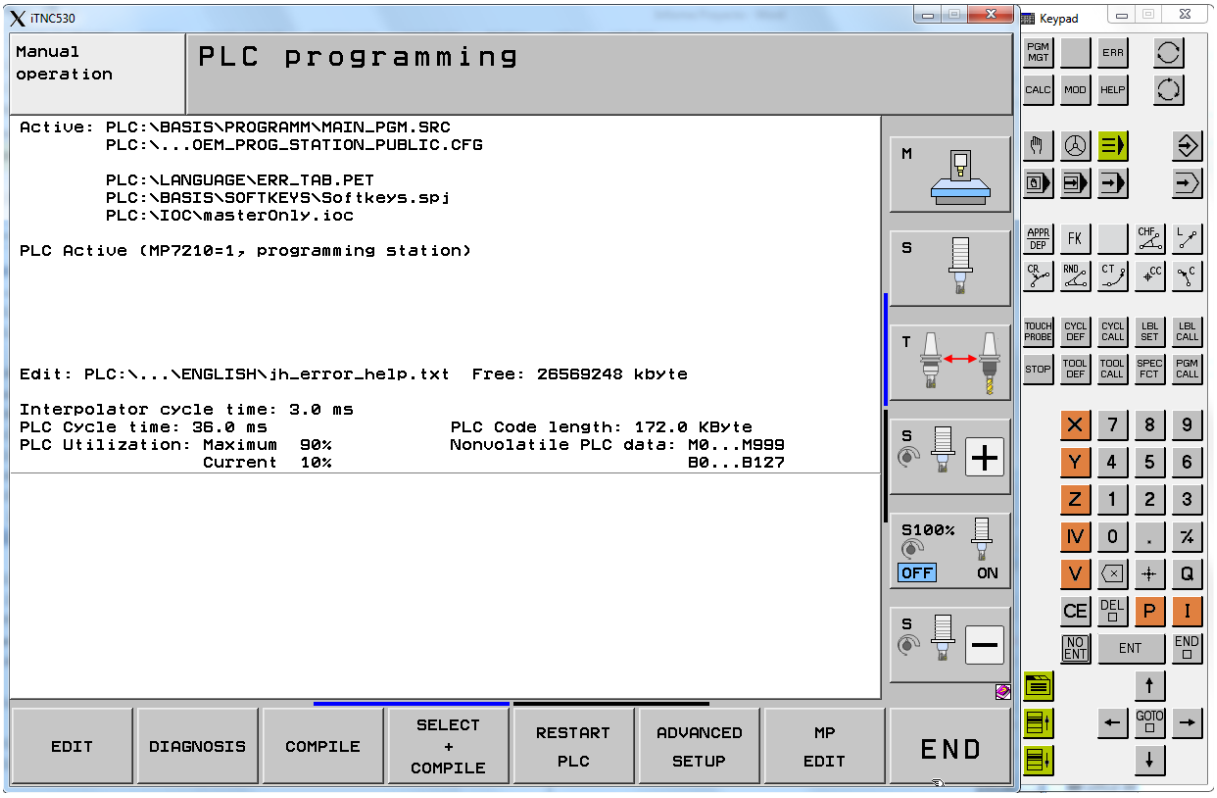



*Figura 9- 4: Pantalla principal de programación del PLC. Extraída de interfaz del software Programming Station iTNC 530*

A continuación, se definen las partes principales de este menú:

- **Activo:** Archivos del PLC integrado en la memoria del proceso. Éstos incluyen el programa del PLC, la tabla del error del PLC, el archivo de proyecto de la llave programable y posiblemente el archivo de configuración de Profibus o el archivo de configuración para la programación del código de fuente. En la puesta en marcha, el iTNC compila automáticamente los archivos definidos en OEM.SYS. Los archivos sólo se activan después de haber sido compilados.

- **Libre (Free):** La memoria disponible en la partición del PLC (PLC: \) se muestra en kilobytes. Además, se comprueba la partición del PLC para ver si hay al menos 10 MB de memoria disponible. Si la memoria disponible es inferior a 10 MB, el siguiente mensaje de error se muestra: '**PLC partition: Not enough memory**'

- **Tiempo de ciclo del PLC:** Tiempo de ciclo del PLC actual, ajustado a través de los parámetros de máquina MP7600.0 y MP7600.1

- **Utilización de PLC Máximo:** Tiempo máximo de ejecución del programa PLC. El tiempo de procesamiento del PLC (tiempo para un ciclo PLC) se da como un porcentaje del tiempo máximo: 100% es el equivalente de un tiempo de ejecución de 1 ms a un tiempo de ciclo de 21 ms. Se utiliza la siguiente fórmula para calcular el tiempo de

ejecución trun [ms] en función del tiempo de ciclo del autómata tPLC [ms] y del tiempo de procesamiento tcalc. [%]:

$$t_{run} = \frac{t_{PLC} * t_{calc}}{21} \quad (2\text{-}2)$$

Si se excede el tiempo máximo de ejecución del programa secuencial, el iTNC visualiza el mensaje de error PLC: time out.

- **Utilización de PLC Actualmente:** El tiempo necesario para la última exploración del PLC en%.

- **Longitud del código del PLC:** Longitud del programa secuencial compilado en KB. Valor máximo: 512 KB

### 9.8.2 Operandos

En la programación de un PLC es necesario conocer el tipo de las variables que se desean usar. Existen varios tipos llamados operandos que pueden ser asignados a las variables creadas ya sea locales o globales. Cada uno de estos operandos tiene un lugar en la memoria dentro del PLC. Al abarcar un espacio en la memoria se debe considerar el rango de memoria que se desea usar al definir variables, y cuidar que estas mismas no substituyan los espacios de memoria ocupadas por otras variables previamente definidas.

A continuación, se presenta una tabla con la lista de operandos existentes y los rangos de memoria disponibles para cada uno de estos operandos.

*Tabla 9- 6: rango de direcciones para operandos existentes. Extraída del manual técnico de Heidenhain del iTNC 530.*

| Operando | Abreviación | Rango de direcciones |
|---|---|---|
| Marcas | M | M0 - M9999<br><br>M0 – M999 están libres; son borrados solo al entrar el código 531210, no durante un reseteado.<br><br>M1000-M3999 están libres y son borrados durante el reseteado.<br><br>M4000 – M5999 están reservados para la transferencia de datos entre el PLC y los códigos CN. |

| | | |
|---|---|---|
| | | M6000- M9999 están libres y son borrados durante el reseteado. |
| Entradas | I | I0-I31 (MC 42x(B,C))<br>I128 – I152 (Panel de operación de la máquina)<br>I64-I127 (primer PLC)<br>I192-I255(Segundo PLC)<br>I256-I319 (Tercer PLC)<br>I320-I383(Cuarto PLC) |
| Salidas | O | O0 – O30 (MC 42x(B,C))<br>O0 – O7 (Panel de operación de la máquina)<br>O32 - O62 (Primer PLC)<br>O64 – O94 (Segundo PLC)<br>O128 – O158 (Tercer PLC)<br>O160 – O190 (Cuarto PLC) |
| Contadores | C | Contador de activación: C0-C47<br>Contenido de contadores: C48-C95<br>Contadores de pulso: C96 – C143 |
| Temporizador | T | Iniciar temporizador: T0-T47<br>Temporizador activo: T48-T95 y T96-T999 |
| Byte<br><br>Palabra<br><br>Doble Palabra | B<br><br>W<br><br>D | B0-B9999 (8 bits)<br><br>B256 – B2047 están reservados para la transferencia de datos entre el PLC y el código CN.<br><br>B2048 – B9999 están libres y solo son borrados durante el reseteado. |
| Constantes | K | -2 147 483 647 - +2 147 483 647 |
| Caracteres | S | S0 – S99 |

### 9.8.3 Conjunto de Comandos

Para la programación del PLC se deben usar los comandos existentes para realizar la función deseada. La programación es distinta a la de un código de alto nivel ya que todas las líneas del código deben tener un comando definido. El código no sigue un orden de línea a línea como un lenguaje de alto nivel, sino que lee todos los comandos donde haya acceso al mismo tiempo. Por esta razón se debe tener mucho cuidado a la hora de programar una nueva función en una rutina del PLC.

La lectura del código funciona por medio de un acumulador lógico que representa el valor que es modificado por el comando actual y comparado con el comando posterior. Cada comando afecta al acumulador lógico de manera distinta, los más importantes de estos comandos para este proyecto son explicados a continuación. Para ver la tabla de todos los comandos disponibles en la programación del PLC ver el anexo 9.3

#### *9.8.3.1 Load (Cargar) y Load Not (Cargar No)*

El comando de cargar y de cargar no, son los más importantes en el código PLC, ya que toda función debe empezar con uno de estos comandos para darle un primer valor al acumulador lógico. Este comando sirve para cualquier tipo de operadores. El comando cargar, carga el valor de la variable dirigida al acumulador lógico. El comando cargar no, carga el valor opuesto de la variable dirigida al acumulador lógico.

En la tabla 9-7 se presenta un ejemplo usando el comando cargar.

*Tabla 9- 7: Ejemplo de función cargar (Load). Elaboración propia.*

| Función | Código | Acumulador Lógico | Contenido del Operando |
|---|---|---|---|
| Cargar el valor de la entrada 4 al acumulador lógico | L I4 | =1 | 1 |
| Asignar el valor del acumulador lógico a la salida 2 | = O2 | | 1 |

#### *9.8.3.2 Set (activar) y Reset (Resetear)*

Los comandos de activar y resetear se usan para asignar un valor a una variable después de una operación lógica. Siempre se debe iniciar el proceso con un comando de cargar. Varios comandos de activar y resetear pueden ser usados seguidamente con el mismo acumulador lógico. Cuando el acumulador lógico tiene el valor de 1, la función de activar pondrá en alto la

marca o el bit de la variable a la cual la orden se refiere. La función resetear funciona de la misma manera, solo que pone en bajo el valor de la marca o del bit.

En la tabla 9-8 se presenta un ejemplo usando los comandos de activar y resetear.

*Tabla 9- 8: Ejemplo de función activar y resetear (Set y Reset). Elaboración propia.*

| Función | Código | Acumulador Lógico | Contenido del Operando |
|---|---|---|---|
| Cargar el valor de la entrada 4 al acumulador lógico | L I4 | =1 | 1 |
| Usar la operación lógica OR entre I4 e I5. (El valor de I5 es 0) | O I5 | 1 | 0 |
| Activar el valor de la salida 2 ya que el acumulador lógico es 1 | S O2 | 1 | 1 |
| Resetear el valor de la marca 500 ya que el acumulador lógico es 1 | R M500 | 1 | 0 |

### *9.8.3.3 Operaciones lógicas*

La programación PLC usa operaciones lógicas para elegir el valor de una variable después de una o de varias de estas operaciones. Estas operaciones pueden ser AND, AND NOT, OR, OR NOT, XOR, XOR NOT. Las operaciones lógicas se usan para comparar el valor del acumulador lógico con la variable en el comando de la operación lógica, y el resultado es guardado en el acumulador lógico. Un ejemplo de cómo usar la operación OR se describe en la tabla N.

A continuación, se describe el funcionamiento de cada una de las operaciones lógicas mencionadas.

- **AND y AND NOT:** para el caso de la operación AND, si ambas entradas a comparar tienen el valor de 1, el acumulador lógico recibe el valor de 1. Usando AND NOT el valor del acumulador lógico debe ser 1 y el valor de la variable a comparar debe ser cero para así guardar el valor de 1 en el acumulador lógico.

- **OR y OR NOT:** para el caso de la operación OR, se necesita que cualquier valor de entrada sea de 1 para guardar en el acumulador lógico el valor de 1. Para OR NOT, ya sea el valor del acumulador lógico o el opuesto de la variable a comparar sea 1, se guardará el valor de 1 en el acumulador lógico.

- **XOR y XOR NOT:** estas operaciones lógicas funcionan igual que las operaciones de OR y OR NOT, tan solo con la diferencia de que si ambos valores de entrada son 1 el acumulador lógico guarda un 0 en vez de un 1.

#### *9.8.3.4 Comparadores*

Los comparadores se usan para comparar el valor de dos variables y guardar en el acumulador lógico el valor de 1 si dicha comparación se cumple, o el valor de cero si la comparación no se cumple. A continuación, se presenta la tabla 9-9 con los existentes comparadores en la programación PLC.

*Tabla 9- 9: Comparadores disponibles en programación PLC. Extraída del manual técnico de Heidenhain del iTNC 530.*

| Sintaxis | Función |
|---|---|
| == | Igual a… |
| > | Mayor a… |
| < | Menor a… |
| >= | Mayor o igual a… |
| <= | Menor o igual a… |
| <> | Diferente a… |

#### *9.8.3.5 Comandos de salto*

Por último, existen varios comandos de salto que llevan al código a distintas partes dentro el mismo módulo, o a un módulo completamente distinto. Estos saltos pueden ir acompañados con un comparador y solo en caso de que se cumpla una condición (falso o verdadero) se realiza el salto, A continuación, se presenta la tabla 9-10 con los existentes comparadores en la programación PLC.

*Tabla 9- 10: Lista de comandos de salto disponibles en programación PLC. Extraída del manual técnico de Heidenhain del iTNC 530.*

| Sintaxis | Función |
|---|---|
| JP | Salto incondicional |
| JPF | Salto si el acumulador lógico es 0 |
| JPT | Salto si el acumulador lógico es 1 |
| CM | Llamar módulo |
| CMT | Llamar módulo si el acumulador lógico es 1 |
| CMF | Llamar módulo si el acumulador lógico es 0 |

| | |
|---|---|
| EM | Terminar módulo |
| EMT | Terminar módulo si el acumulador lógico es 1 |
| EMF | Terminar módulo si el acumulador lógico es 0 |
| LBL | Etiqueta de salto |

## 9.9. Módulos

### 9.9.1 Módulo 9291

Con el módulo 9291 se puede llamar a un macro CN en cualquier modo de funcionamiento. Estos son ejecutados como ciclos, sin visualización de bloque. El símbolo de control en funcionamiento aparece en pantalla mientras se ejecuta la macro. No se pueden activar macros si existe actualmente un mensaje de error de paro de emergencia externo.

El módulo 9291 inicia una macro NC con los mismos mecanismos que seleccionando un programa manualmente y arrancándolo en la interfaz de usuario, es decir, con la selección de la NC Macro se restablecen los mismos ajustes modales que con PGM MGT y Enter. Esto significa que la continuación de un programa NC ya iniciado debe ser prevenido después de que se haya llamado al Módulo 9291.

### 9.9.2 Módulo 9297

El módulo 9297 se utiliza para consultar el estado de una instancia de Python que se inició con el módulo 9295. Este módulo solo puede ser llamado desde un programa de envío (sección 2.4.3). El llamado del mismo es descrito a continuación

*Tabla 9- 11: Uso de módulo 9297. Extraída del manual técnico de Heidenhain del iTNC 530.*

| Comando | Variable | Descripción |
|---|---|---|
| PS | SXX | Nombre del proceso |
| CM | 9297 | |
| PL | B/W/D | Valor de retorno:<br>0: proceso active<br>1: proceso no llamado desde un programa de envío<br>2: proceso de Python no activo<br>3: valor de transferencia incorrecto |

### 8.9.3 Módulo 9296

El módulo 9296 se utiliza para enviar una señal de "cancelación" a una instancia de Python que se inició con el módulo 9295. El final real del proceso no se espera.

Dependiendo del script Python, la cancelación del proceso puede conducir a un sistema inconsistente. Por ejemplo: que las tablas sólo se cambian parcialmente. Esto se aplica en particular si se utiliza el modo 1 para cancelar. Este modo sólo debe utilizarse si una cancelación precedente con Modo 0 no ha podido finalizar el proceso (interrogar el estado con el Módulo de PLC 9297).

A continuación, se presenta el llamado del módulo 9296 en la tabla 9-12.

*Tabla 9- 12: Uso de módulo 9296. Extraída del manual técnico de Heidenhain del iTNC 530.*

| Comando | Variable | Descripción |
|---|---|---|
| PS | B/W/D/K | Modo de llamado:<br>0: Terminar<br>1: Matar |
| PS | SXX | Nombre del proceso |
| CM | 9296 | |
| PL | B/W/D | Valor de retorno:<br>0: señal enviada<br>1: proceso no llamado desde un programa de envío<br>2: valor de transferencia incorrecto (modo de llamado<br>3: valor de transferencia incorrecto (nombre del proceso)<br>4: proceso Python no activo |

### 9.9.4 Módulo 9295

El módulo 9295 se utiliza para iniciar una instancia o proceso de Python. El nombre del proceso Python debe especificarse cuando se llama. Debe ser elegido independientemente del script Python asociado. Esto hace posible iniciar varios procesos de Python basados en el mismo script de Python.

El modo de llamado se utiliza para seleccionar entre llamado asincrónico o llamado sincrónico. Estos son descritos a continuación.

- Inicio sincrónico: El programa de envío espera en el módulo 9295 hasta que el proceso de Python que se ha iniciado termina por sí mismo. El inicio síncrono de los procesos de Python puede utilizarse si el proceso Python devuelve un resultado que se necesita posteriormente en el trabajo de presentación o generación (como realizar cálculos matemáticos).

- Inicio asincrónico: El programa de envío continúa ejecutándose inmediatamente, no se espera en el módulo 9295. El inicio asíncrono se puede utilizar para iniciar procesos de Python que no devuelven un resultado (como la visualización de ventanas).

A continuación, se presenta el llamado del módulo 9295 en la tabla 9-13.

*Tabla 9- 13: Uso de módulo 9295. Extraída del manual técnico de Heidenhain del iTNC 530.*

| Comando | Variable | Descripción |
|---|---|---|
| PS | B/W/D/K | Modo de llamado:<br>0: sincrónico<br>1: asincrónico |
| PS | B/W/D/K | Memoria por utilizar (64 MB disponibles) |
| PS | | Nombre del código (.py) |
| PS | | Nombre del proceso |
| PS | | Parámetro de entrada |
| CM | 9296 | |
| PL | B/W/D | Valor de retorno:<br>0: proceso iniciado<br>1: proceso no llamado desde un programa de envío<br>2: no hay licencia para utilizar Python<br>3: memoria excedida<br>4: valor de transferencia incorrecto (memoria a utilizar)<br>5: valor de transferencia incorrecto (nombre del código)<br>6: valor de transferencia incorrecto (nombre del proceso)<br>7: el proceso no se pudo iniciar<br>8: un proceso con ese mismo nombre ya está corriendo<br>9: parámetro incorrecto para el proceso Python<br>10: la memoria de trabajo total excede los 512 MB |
| PL | | Código de salida del proceso de Python |

### 9.9.5 Módulo 9240

Se pueden abrir hasta ocho archivos simultáneamente. Se accede desde el proceso en el que se abrieron (programa de envío). Después de abrir el archivo, el módulo 9240 siempre transfiere un "identificador de archivo". El identificador de archivo es un número de serie que se puede utilizar para seleccionar este archivo de nuevo en otros módulos. Para añadir datos a un archivo existente, se debe establecer el bit 0 = 1 (lectura y escritura) y el bit 2 = 0 (orientado al registro).

A continuación, se presenta el llamado del módulo 9240 en la tabla 9-14.

*Tabla 9- 14: Uso del módulo 9240. Extraída del manual técnico de Heidenhain del iTNC 530.*

| Comando | Variable | Descripción |
|---|---|---|
| PS | B/W/D/K | Modo de llamado:<br>Bit 0 = 0: solo lectura<br>Bit 0 = 1: lectura y escritura<br>Bit 1 = 0: Archivo no bloqueado<br>Bit 1 = 1: Archivo bloqueado<br>Bit 2 = 0: Tablas<br>Bit 2 = 1: archivos ASCII<br>El Bit 3 solo se evalúa si Bit 2 = 0<br>Bit 3 = 0: no crear archivo<br>Bit 3 = 1: crear archivo si este no existe |
| PS | SXX | Dirección de archivo y nombre de archivo |
| CM | 9296 | |
| PL | D | Identificador de archivo |

### 9.9.6 Módulo 9243

Para leer de una tabla, se debe utilizar el módulo 9245. Primero, se debe abrir el archivo con el módulo 9240. Con el módulo 9243 se lee línea por línea desde un archivo ASCII.

A continuación, se presenta el llamado del módulo 9243 en la tabla 9-15.

*Tabla 9- 15: Uso del módulo 9243. Extraída del manual técnico de Heidenhain del iTNC 530.*

| Comando | Variable | Descripción |
|---|---|---|
| PS | D | Identificador de archivo (obtenido del módulo 9240) |
| PS | B/W/D/K | Número del carácter para el resultado |
| CM | 9243 | |
| PL | B/W/D | Número de bits leídos:<br>> 0: La línea ha sido leída<br>= 0: Final de archivo<br>-1: Error |

### 9.9.7 Módulo 9241

Con este módulo se cierra un archivo que se ha abierto con el módulo 9240. Se debe cerrar el archivo en el proceso (programa de envío) en el que lo abrió. El llamado se describe a continuación.

*Tabla 9- 16: Uso del módulo 9241. Extraída del manual técnico de Heidenhain del iTNC 530.*

| Comando | Variable | Descripción |
|---|---|---|
| PS | D | Identificador de archivo (obtenido del módulo 9240) |
| CM | 9241 | |

## 9.10 Descripción de variables en archivo de ajustes y entradas.

- **Pos_conv:** convierte las unidades de las medidas de posición a unidades SI ([m]), normalmente las unidades vienen en [mm], por lo tanto, el valor ajustado en el archivo es de 0.001.

- **Acc_conv:** convierte las unidades de las medidas de aceleración a unidades SI ([$m/s^2$]), normalmente las unidades vienen en unidades SI, por lo tanto, el valor ajustado en el archivo es de 1.

- **Vel_conv:** convierte las unidades de las medidas de velocidad a unidades SI ([m/s]), normalmente las unidades vienen en [mm/min], por lo tanto, el valor ajustado en el archivo es de 1.66667e-5.

- **Start_cal y End_cal:** se refieren a los valores de posición de inicio y de final de calibración, estos dados en las unidades del Sistema, normalmente milímetros. Estos valores serán usados en la parte de la calibración de la corriente del motor, sección 3.3.1.

  Las únicas restricciones para estos valores, es que no pueden ser iguales y que el intervalo entre ellos contenga al intervalo de inicio y de final de medición.

- **Start_m y End_m:** se refieren a los valores de posición de inicio y de final de medición, estos dados en las unidades del Sistema, normalmente milímetros. Estos valores serán usados en la parte de la calibración de la corriente del motor, sección 3.3.1.

  Las únicas restricciones para estos valores, es que no pueden ser iguales y que el intervalo entre ellos esté contenido dentro del intervalo de inicio y de final de calibración.

- **A_lim:** este valor se refiere al máximo de la aceleración, el valor donde la aceleración se vuelve constante por un período de tiempo. Esto es un valor ideal, ya que en la señal de calculada hay mucho ruido como para encontrar un valor exacto estable.

  La única restricción es que este valor no puede ser cero.

- **MinT:** es el mínimo intervalo de tiempo que debe pasar para marcar uno de los 8 intervalos donde la aceleración corta con el margen de aceleración (la mitad del límite de aceleración. Este valor se usa para evitar intervalos que se darían por el ruido de la señal. Se usa en la sección 3.3.3.

  Este valor tiene que ser positivo y mayor a cero, también se recomienda un periodo de tiempo menor a 1 y mayor a 0.1.

- **Margin:** es el porcentaje que se substraerá del intervalo de aceleración tomado en la sección 3.3.3. Este valor puede ser 0, pero se recomienda siempre remover un porcentaje del intervalo ya que si no puede haber errores por el ruido de la señal y la masa medida sería incorrecta.

  Este valor debe estar entre 0 y 1, ya que es un porcentaje ese valor luego será multiplicado por cien.

- **Km_sys:** es la constante de torque dada por el Sistema de control. En este caso el valor es de 1.7036

- **Acc_uncert:** es el valor en porcentaje de la incertidumbre aceptada para el valor de la carga. Se recomienda que no sea mayor a 5% para que no haya problemas al cambiar parámetros.

- **Ratio:** es la proporción de cambio entre la velocidad lineal y la velocidad angular entra las piezas móviles. Para cada eje y para cada máquina es distinta, entonces se debe variar si se cambia de eje o de máquina.

## 9.11 Pasos necesarios para obtener resultados de calculo de masa en la estación de programación del iTNC 530.

### 9.11.1 Guardar script en los archivos de Python dentro del PLC

Antes de iniciar las pruebas del script de Python dentro de la estación de programación, debemos guardar dicho script dentro de los archivos del PLC. Ya que como se explicó en la sección 2.3 del marco teórico, los scripts de Python solo pueden ser puestos en marcha dentro del PLC de la estación de control.

Para lograr esto debemos usar el software TNCremo. Este nos permite comunicar el disco duro de nuestra computadora con la estación de programación, inclusive con la máquina directamente si fuera necesario realizar las pruebas en la máquina.

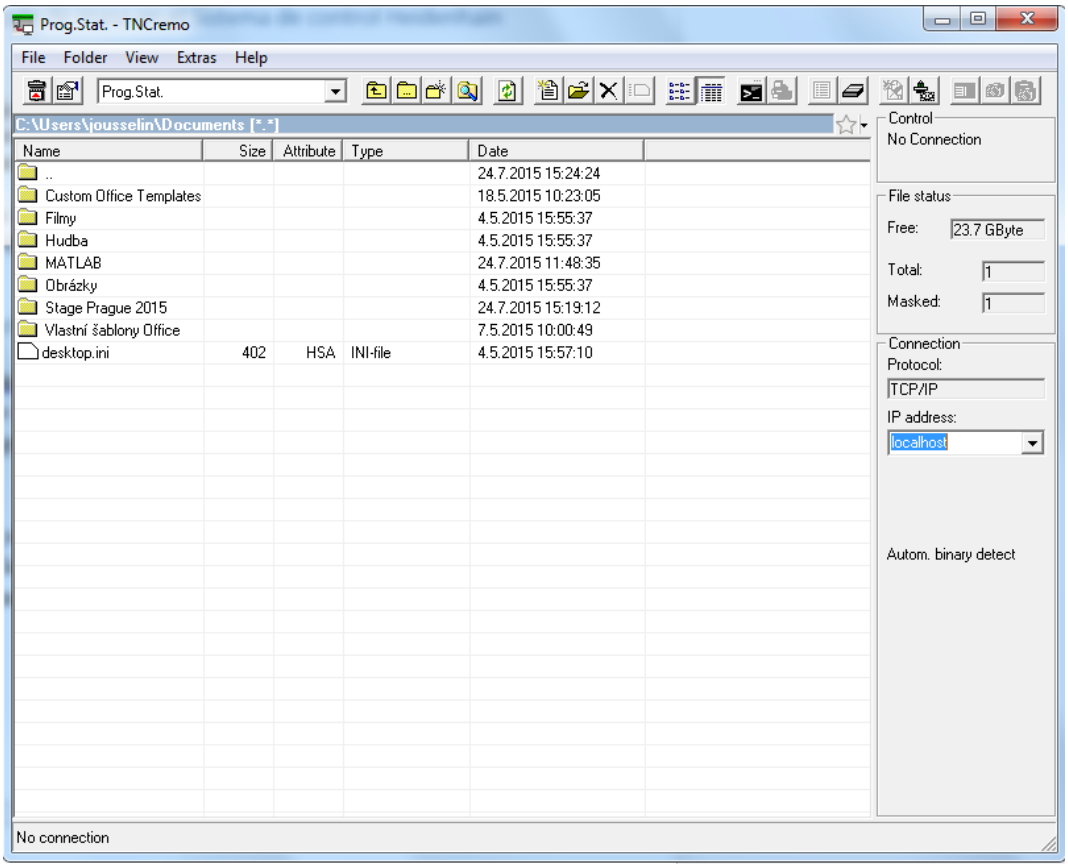



*Figura 9- 5: Interfaz de software TNCremo. Extraída de software TNCremo.*

En la figura 9-5 se muestra la pantalla principal del software TNCremo. En esta se pueden ver las carpetas del computador. Para conectar la estación de programación debemos revisar la dirección IP, esta debe estar en 'local host', y luego usamos el botón de conectar que se encuentra en la esquina superior izquierda de la pantalla, con una flecha roja.

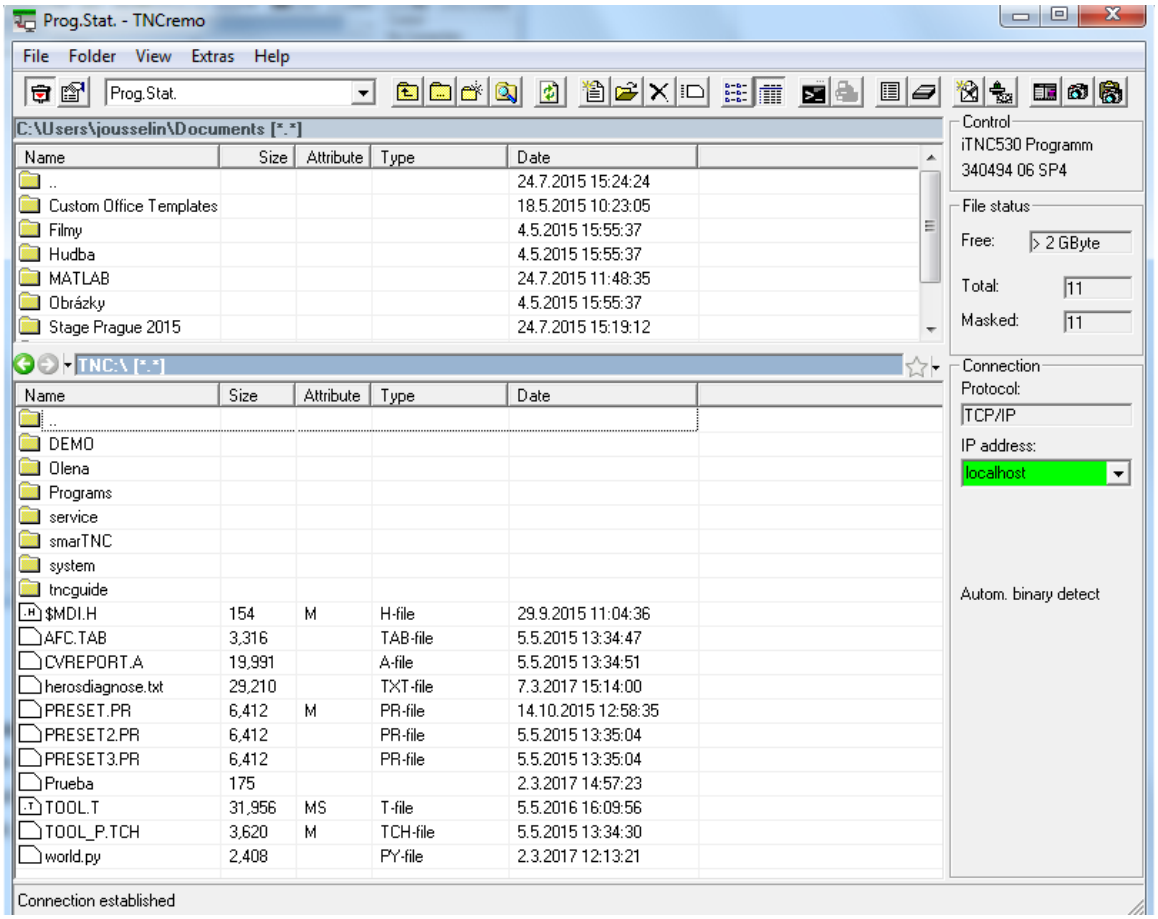



*Figura 9- 6: Conexión de computador con Programing Station iTNC530. Extraído de software TNCremo.*

Con esto ya se ha conectado la estación de programación y se pueden guardar archivos del computador al Sistema de control y viceversa. Sin embargo, el PLC no es accesible aún, ya que al igual que en la programación ocupa una clave para ingresar a él y modificar sus archivos.

Para acceder al PLC se debe ir a carpetas dentro del TNCremo, darle a cambiar el archivo y elegir PLC. El software solicita una clave numérica para ingresar al PLC, y esta es **807667,** al igual que en la sección de edición y programación de la estación de programación.

En los archivos del PLC, se encuentra una carpeta llamada Python reservada para los scripts de este lenguaje. Por lo tanto, guardamos el script dentro de esta carpeta y con eso el archivo ya se puede editar dentro de la estación de programación. Para verificar que el script se haya guardado se debe presionar la tecla ‘PGM MGT’ dentro del PLC en la estación de programación y buscar en el directorio el script. Con la ‘softkey’ editar podemos ver el código y editarlo si fuera necesario. En la figura 9-7, se muestra un ejemplo de un script simple guardado en la máquina.

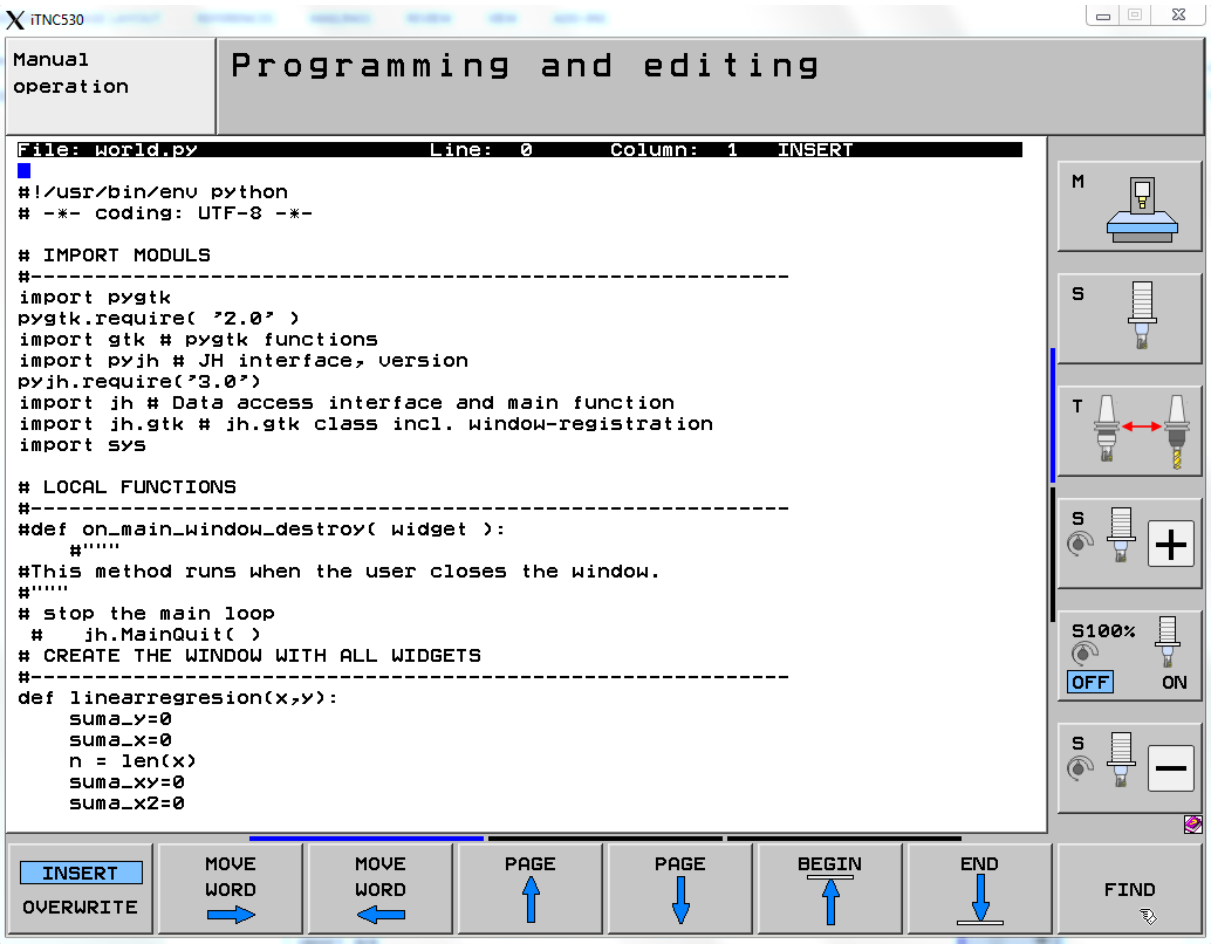



*Figura 9- 7: pantalla de edición de la estación de programación del iTNC 530. Extraída de software Programing Station iTNC 530*

En esta pantalla se puede editar el script, pero no se recomienda editarlo desde la estación de programación ya que no es tan fácil como editarlo en Idle u otro programa para scripts de Python. En la parte de arriba de la pantalla se ve el nombre del archivo y el número de línea y columna donde se encuentra el cursor.

### 9.11.2 Creación de 'SoftKey' para correr script

Para correr un script de Python en la estación de programación existen dos maneras. La primera es usar uno de los módulos del PLC que están hechos específicamente para Python dentro de; Sistema de control, o la segunda opción es crear una Tecla ''softkey' que corra el script al ser pulsada. Para usar los módulos de PLC debemos programar una rutina dentro del PLC y esto lleva su tiempo, por lo tanto, para este caso específico (probar el script de Python que muestre la masa total de las piezas en movimiento) se decidió crear una 'softkey'.

Primero se debe activar la softkey de Python antes de poder crear una sofkey para el proceso o script que se desea correr. Por lo tanto, se debe cambiar en el archivo de configuración del PLC el parámetro de Python de 'inactive' a 'active'. Con esto la softkey se mostrará de Python se mostrará al lado derecho de la estación de programación.


*Figura 9- 8: Softkey de Python activada. Extraída de software Programing Station iTNC 530.*

En la figura 9-8 se muestra la softkey de Python esta, a la vez, contiene dentro más softkeys y espacios vacíos para nuevas.

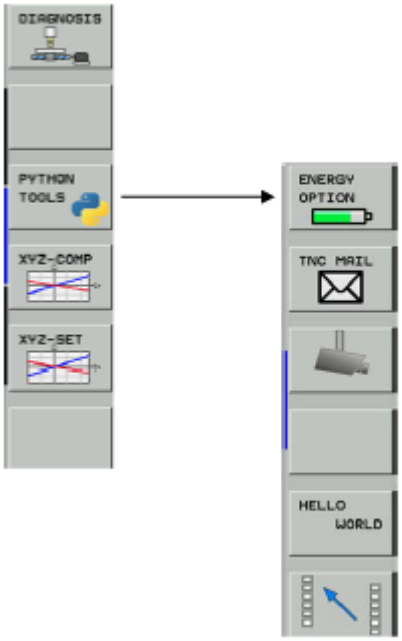



*Figura 9- 9: Softkeys dentro de Python tools. Extraída de software Programing Station iTNC 530.*

En la figura 9-9 podemos ver una softkey para un script de Python llamado Hello World, este es un script sencillo que imprime en pantalla una ventana con las palabras Hola Mundo. Ahora como ya está activado la softkey de Python tools, podemos crear una nueva softkey que corra un proceso de Python deseado.

Para crear una softkey tenemos que entrar nuevamente a los directorios del PLC y buscar el archivo BASIS donde está el archivo SOFTKEYS, dentro de este hay un documento llamado softkeys.spj es ahí donde crearemos la nueva softkey. Debemos buscar la sección que contiene el condicional que verifica si la configuración de Python está activa. En la figura 9-10 se muestra esta sección.

```
#if CFG_Python = "aktiv"

SKMENU Python_demo
ACTION SK_Python_csv_emb                PyName:CSV_EMB PyMem:10 PYSCRIPT:PLC:\Py»
ACTION SK_Python_csv_ovl                PyName:CSV_OVL PyMem:10 PYSCRIPT:PLC:\Py»
PULSE SK_Python_pos_emb                 STATUS:MG_Softkey_Python_Pos_Emb        »
PULSE SK_Python_pos_ovl                 STATUS:MG_Softkey_Python_Pos_Ovl        »
ACTION SK_Python_world                  PyName:WORLD PyMem:10 PYSCRIPT:PLC:\Pyth»
```

*Figura 9- 10: Código para creación de Softkey dentro de archivos del PLC. Extraída de software Programing Station iTNC 530.*

En la figura 9-10 se observa cómo se activan nuevas softkeys, para crear la softkey se debe poner la función ACTION, y luego escribir el nombre del proceso (no tiene que ser el mismo que el del script), la memoria a utilizar en Megabytes, y la dirección exacta del script en el directorio del PLC.

Para este caso, usaremos una softkey llamada WORLD para realizar las pruebas necesarias en la estación de programación. Una vez que el código esté terminado se deberá crear otra softkey para el usuario de la máquina.


*Figura 9- 11: Softkey creada para correr script de Python. Extraída de software Programing Station iTNC 530.*

.

### 9.11.3 Proceso para depurar un código

Antes de probar el código se debe realizar el proceso de 'debugg' en busca de errores, ya que, aunque el código corra en el IDLE de Python puede ser que no lo haga en la estación de programación por la razón que no se cuentan con todas las librerías de Python. Las librerías disponibles en Python de Heidenhain están en la tabla 9-5, en el anexo 9-6.

Se deben seguir los siguientes pasos para depurar un proceso de Python:

- Cambiar al modo de funcionamiento de programación y edición, pulsar la tecla MOD e introducir el número de código apropiado para cambiar al modo de funcionamiento de programación del PLC.

- Presionar la tecla programable PROCESAR MONITOR en la segunda fila de softkeys.

- Pulsar la tecla de función PYTHON DEBUG. El proceso Python actualmente seleccionado para la depuración se mostrará automáticamente.

- Introducir el nombre del proceso Python que se va a depurar en la línea de diálogo y presionar la tecla SET. Si se presiona la tecla de función SET sin introducir un nombre, el último proceso de Python seleccionado para la depuración será sugerido automáticamente. Se puede borrar la ventana informativa del proceso de Python para la depuración con la tecla CE.

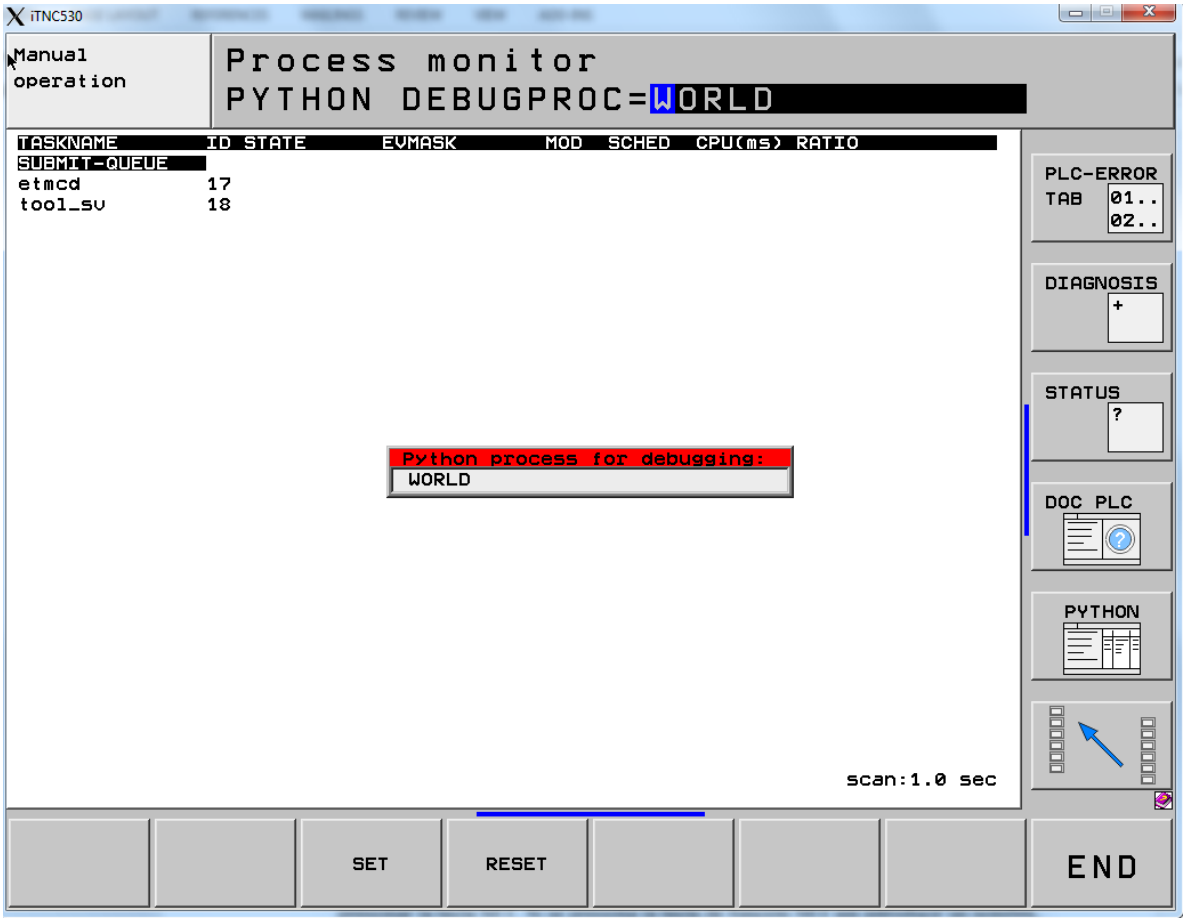



*Figura 9- 12: Pantalla para depurar scripts de Python. Extraída de software Programing Station iTNC 530*

En la figura 9-12 se observa la pantalla para la depuración de un proceso, y en esta se ha seleccionado el proceso WORLD previamente creado por medio de la softkey en la sección anterior. Finalmente, al momento de tartar de correr el código presionando la softkey, el código se depurará antes de correr.

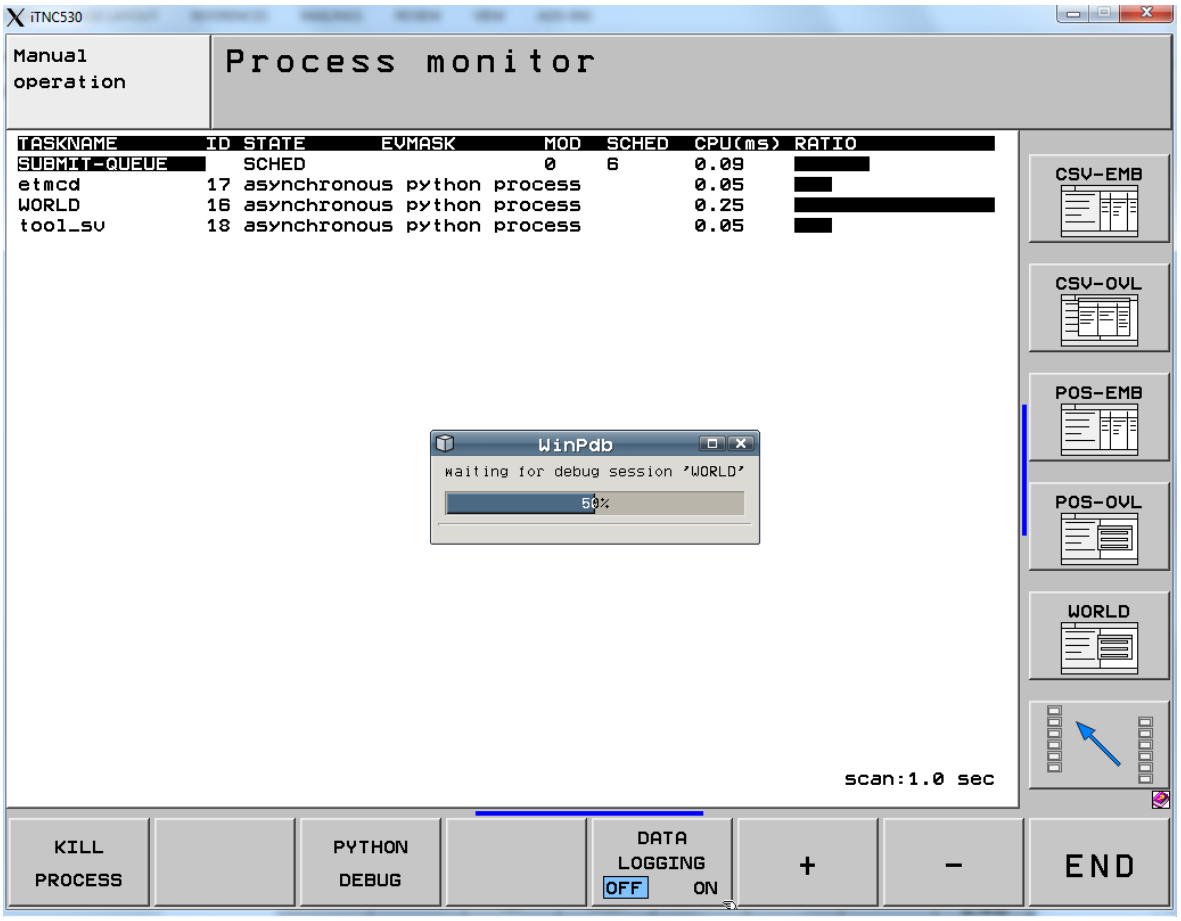



*Figura 9- 13: Código de Python siendo depurado. Extraída de software Programing Station iTNC 530.*

## 9.12 Compilación de código PLC y transferencia a estación de programación

Para finalizar la programación de la rutina del PLC, esta debe ser compilada y transferida a la estación de programación para verificar su correcto funcionamiento. El proceso consta de tres pasos.

- Compilación: proceso por el cual se verifica si hay errores en el código del PLC
- Construcción: proceso por el cual se construye el archivo de transmisión a partir de todos los archivos del PLC
- Trasmisión: proceso en el cual se transmite el archivo de transmisión a la estación de programación.

Antes de la compilación del código se deben elegir los parámetros de transmisión deseados en ajustes del programa. En la figura 9-14 se muestra la ventanilla de ajustes.

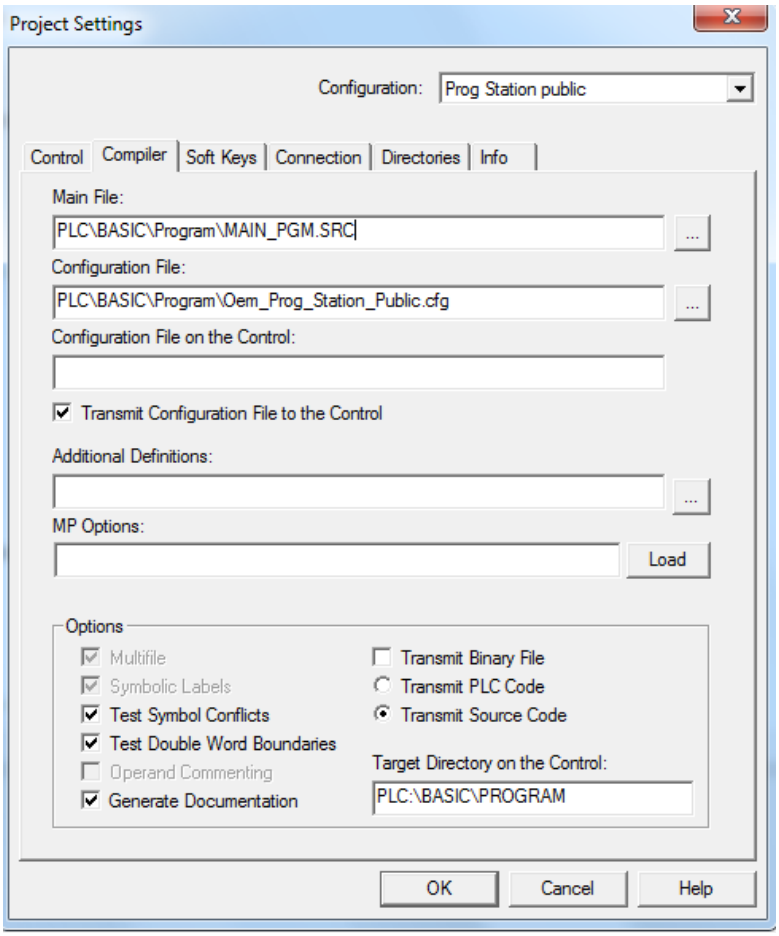



*Figura 9- 14: ventanilla de ajustes de compilación. Extraída del software PLC Design*

Lo que se debe tomar en cuenta en esta ventanilla, es el tipo de transmisión que se desea realizar. Tenemos dos opciones. Transmitir el código en PLC o el código en SRC (código fuente). La diferencia radica en cómo se lee el archivo transmitido. En código Fuente se puede ver igual que aquel programado en el PLC design. Sin embargo, en código PLC es distinto y es más difícil de manipular manualmente.

La mejor opción es usar código Fuente, para que este sea fácilmente manipulable desde la máquina en un futuro. Sin embargo, dependiendo del Sistema de control de la máquina, puede ser que la máquina no pueda leer archivos SRC, y solo pueda leer archivos PLC. Este es el caso de la Kovosvit MAS MCV 1000 disponible en el RCMT. Por lo tanto, al transferir a la máquina, el código debe ser tipo PLC. Para definir las variables locales y variables globales existe una diferencia en este tipo de código, estas diferencias son comentadas en la sección 3.7

Una vez elegidos los ajustes deseados, se debe hacer la compilación del código. Si este compila sin ningún error, se prosigue a construir el mismo, y por último a transferir ya sea todos los archivos o tan solo los archivos modificados. Una vez transferidos a la estación de programación se debe reiniciar la misma o entrar a la sección de editar del PLC (ver sección 2.4.1) y compilar el PLC desde ahí.