UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

PUESTA EN MARCHA DE SISTEMA DE DISEÑO DE PIEZAS COMPLEJAS PARA IMPRESORA 3D DE CONCRETO

POR

Matías Nicolás López Barriga

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil Electrónico

Profesor Guía
Juan Pablo Segovia Vera

Junio 2023
Concepción (Chile)

A mis padres y hermano, permanentes referentes de vida

Agradecimientos

Llegar a este punto no ha sido fácil y lo he logrado gracias a la ayuda y comprensión de un grupo de personas. Agradezco a mi madre Miriam por su abrigo y cariño incondicional que me ha entregado, y a mi padre Juan que siempre me apoyo hasta el día que dejó este mundo en 2019.

También a mi hermano Juan, mi mayor y más estricto consejero de vida que me ha ayudado a evitar decisiones de las que hoy estaría arrepentido.

Agradezco también a mis amigos de infancia, quienes hasta el día de hoy no me han abandonado a pesar de desaparecerme por largos periodos de tiempo debido a los estudios.

Por último, pero no menos importante, agradezco al profesor Juan Pablo Segovia por guiarme a lo largo de este proyecto y confiar mi capacidad de llevar adelante las propuestas que generé a mitad de camino en esta memoria de título.

Sumario

Este documento presenta la solución desarrollada en respuesta al problema de optimización del procedimiento utilizado para la generación y envió de instrucciones de movimiento presente en el proceso de impresión diseñado por exalumnos memoristas de la universidad de Concepción. Ellos diseñaron una impresora 3D de concreto tipo delta y un pequeño software que se encarga de generar instrucciones de movimiento. El primer problema presente en el anterior diseño radicó en que el software fue creado con MATLAB 2013b, lo que lo hace complejo de utilizar hoy en día por cuestión de compatibilidad. El segundo problema se halló en que para realizar el proceso completo se necesitaba usar 4 programas distintos, lo que aumenta la probabilidad de errores, los recursos necesarios y el tiempo requerido para imprimir piezas.

Para solucionar estos problemas se desarrolló un software que genere y envíe las instrucciones de movimiento a la impresora, reduciendo el número de programas necesarios a solo 2. Para ello se investigó las herramientas disponibles, escogiendo a los lenguajes de programación Python y QML, junto con el programa rebanador Ultimaker Cura. Luego se listaron los requisitos a cumplir por el software, se diseñaron las pantallas de la nueva interfaz gráfica y se procedió a programarlas. Finalmente se realizó un Factory Acceptance Test (FAT) con los equipos disponibles en el laboratorio de control de procesos.

El software respondió satisfactoriamente a las pruebas, manteniendo abierta la conexión entre el computador y la impresora durante todo el proceso. Fue capaz de enviar ordenes de movimiento a la impresora, monitorear impresiones y cambiar valores de parámetros internos de ella cuando se deseaba. Este programa se testeó correctamente en Windows 10, MacOS Big Sur y Ubuntu 22.04.

Summary

This Document exhibits the solution developed in response to the optimization problem present in the process of creation and delivery of movement instructions designed by ex-students of the university of Concepcion. They designed a delta concrete 3D printer and a small software that generates the needed printing instructions. The first problem in their design is that this software was made on MATLAB 2013b, which makes it difficult to use today for compatibility reasons. The second problem is found when looking at the process flow, which shows the need for 4 computer programs to print something. This increases the chances of making mistakes, the resources needed, and the time consumed when printing pieces.

To solve these issues, a new software capable of producing and sending printing instructions was developed, reducing the required computer programs from four to only two. This started with an investigation of the tools available for this task, settling on Python and QML as the programing languages to use and Ultimaker Cura as the slicer software. Following this, the tasks to be accomplished by the software were listed and its different tabs were designed and later built in code. Lastly, a Factory Acceptance Test was performed operating the equipment available in the process control laboratory.

The software behaved appropriately under testing, keeping the communication open between the PC and the printer for the whole process. It was able to send printing instructions, monitor the printing procedure and change the internal parameters of the machine when ordered. This PC program was tested satisfactorily on Windows 10, MacOS Big Sur and Ubuntu 22.04.

Tabla de contenidos

Lista de Tablas

Lista de Figuras

Abreviaciones

Mayúsculas

U.C. : Ultimaker Cura
C.C. : Contour Crafting
C.P. : Concrete Printing
GUI : Graphical User Interface
PLC : Programable Logic Controller
RDL : Robot Delta Lineal
CNC : Computer Numerical Control
IDE : Integrated Development Environment
CIP : Common Industrial Protocol
FAT : Factory Acceptance Test

1. Introducción

1.1. Introducción General

Las tecnologías de impresión 3D se han insertado rápidamente en la industria manufacturera de piezas complejas y, últimamente, la industria de construcción de grandes estructuras y vehículos espaciales. Para cada uno de estos casos se necesita una impresora enfocada en esa tarea, pues se necesitan crear elementos de distintos tamaños y materiales, para lo cual existen diversas tecnologías de impresión. En particular, hoy en día se construyen piezas en variados plásticos, metales, material cerámico y hormigón con tecnología de manufactura aditiva (A.M.), pero a pesar de los distintos materiales utilizados, todas las piezas a imprimir con A.M. deben pasar por el mismo proceso para generar las instrucciones de impresión.

El listado de ordenes que necesita la impresora para construir la pieza, se obtiene del "laminado" de las figuras con ayuda de softwares orientados al rebanado de modelos. Este proceso divide la figura en capas horizontales, obteniendo las coordenadas por las que debe pasar el extrusor, lo que luego se traduce en las posiciones que deben tomar los motores para que el actuador del robot llegue a dichas coordenadas. La traducción se consigue haciendo uso de las ecuaciones solución del problema inverso de posición del robot impresora, las cuales varían dependiendo de la categoría a la que pertenezca el robot. Algunos de los softwares más sofisticados no solo realizan el rebanado de las piezas, sino también incluyen herramientas para comunicar las instrucciones directamente a sus impresoras al momento de calcularlas, al mismo tiempo que permiten monitorear el proceso.

Es en este ámbito en el que se desarrolló este trabajo. Esta memoria de título se basa en mejorar y optimizar los trabajos previos desarrollados en memorias anteriores [1][2]. En ellos se diseñó un robot delta lineal (R.D.L.) que sirviera como una impresora 3D industrial, además de diseñar las primitivas de control del robot. Junto con esto, se diseñó en MATLAB 2013b un software simple que se encargara de traducir las instrucciones de movimiento a un lenguaje compatible con el PLC que controla los motores.

El presente documento se centra tanto en la generación y traducción de instrucciones de impresión 3D, como además, en lograr una comunicación entre el programa rebanador y la impresora 3D controlada por PLC para entregarle las instrucciones y luego poder monitorear y controlar el proceso de impresión.

1.2. Trabajos Previos

1.2.1 *Sobre el presente y futuro de la impresión 3D con concreto*

La industria de la construcción ha existido por mucho tiempo, sumando nuevas herramientas y tecnologías a sus procesos, pero la impresión 3D sin duda, es una de las más innovadoras. El interés por la impresión 3D enfocada a la construcción ha aumentado exponencialmente los últimos años [9], lo que se ve reflejado en el número de estudios y publicaciones recientes en comparación a 10 años atrás (fig. 1.1).

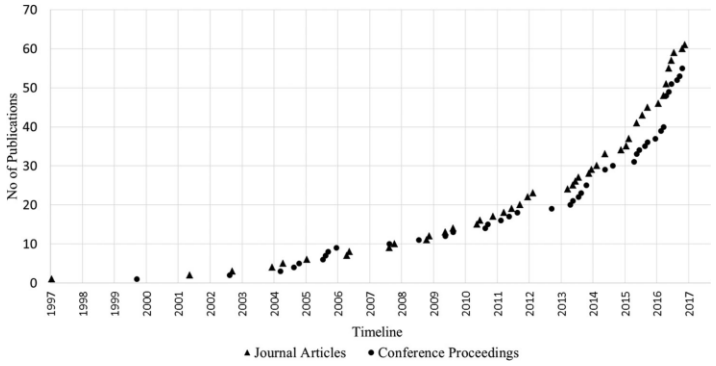


Fig. 1.1 N° de publicaciones centradas en la impresión 3D orientada a la construcción a lo largo de los años. [9]

Dentro de los estudios que han surgido de este interés, se encuentran distintas respuestas al problema de como imprimir piezas con concreto. Las soluciones con mejores resultados corresponden a las tecnologías de impresión por *contour crafting (CC)*, *concrete printing (CP)* y *D-shape printing* [10]. Los métodos CC y CP se basan ambos en el principio de A.M., pero mientras que en CC se utiliza una espátula concretera para lograr una impresión más suave, en CP se omite esta espátula logrando una estructura con vacíos entre sus capas, lo que trae consigo el beneficio de una estructura flexible, aunque también frágil a los golpes. Por último, la técnica *D-shape* construye piezas utilizando capas de material granulado y adhesivo, con lo que se pueden crear estructuras muy detalladas, pero a la vez muy frágiles en comparación al concreto, ya que no pueden resistir los climas húmedos.

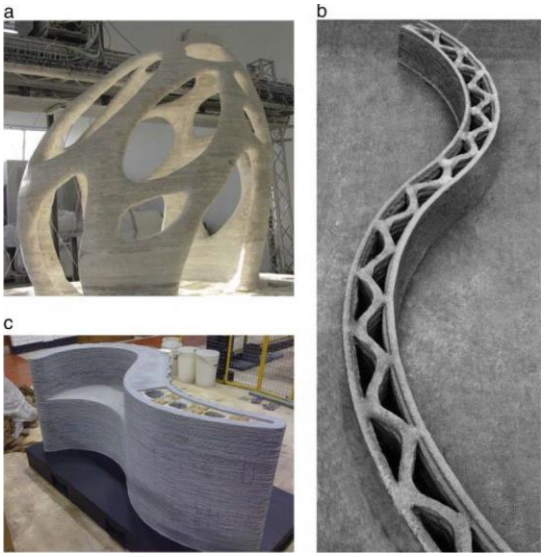


Fig. 1.2 Estructuras impresas con impresoras 3D de concreto usando distintas técnicas [10]
(a) D-shape, **(b)** Contour crafting, **(c)** Concrete printing

Otro problema aún presente en la impresión 3D con concreto, es el reforzamiento de las figuras. Investigadores del centro de impresión 3D de Singapur (SC3DP) ya han probado 2 prototipos de solución [15]. El primero (fig. 1.3a) propone depositar concreto con un cable de acero dentro de él, para lo que se necesita un sistema especial de extrusión y un cable flexible que pueda realizar las curvas presentes en las figuras. El segundo método (fig. 1.3b) plantea un sistema que implante pequeñas fibras de acero dentro de cada capa impresa. Por ahora ambas propuestas necesitan mayor investigación para ser aplicadas de forma práctica en la realidad.

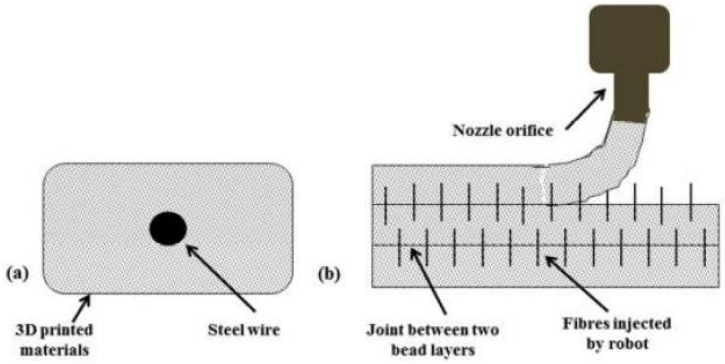


Fig. 1.3 Propuestas para reforzar estructuras impresas. [15]
(a) Cable flexible de acero, **(b)** Fibras de acero

1.2.2 *Sobre la impresora diseñada en memorias anteriores*

Exalumnos memoristas de la Universidad de Concepción, diseñaron un robot delta lineal para la impresión de piezas de concreto utilizando la tecnología de manufactura aditiva [1][2]. En el desarrollo que realizaron se diseñó la estructura de la impresora, las primitivas de control de posición y movimiento y la etapa de traducción de las instrucciones de impresión. Este diseño tomó en cuenta el uso de equipos Allen-Bradley presentes en el laboratorio de control, por lo que la comunicación con el PLC se hizo con el programa RSLogix 5000.

El proceso completo de impresión requiere de 4 etapas: modelación de la figura, rebanado, traducción de instrucciones y envío/monitoreo de ellas. El modelado se realiza en el programa SolidWorks 2016, mientras que el rebanado en el software Slic3r. Para entregarle las instrucciones al PLC se desarrolló una interfaz gráfica en MATLAB 2013 (fig. 1.4) que se encarga de recibir un archivo conteniendo el G-code de la figura y, utilizando las medidas de la impresora y las ecuaciones solución del robot delta lineal [2][28], traducirlo a las posiciones que deben tomar los motores. Estas instrucciones traducidas se insertan dentro de un archivo ".L5K" compatible con RSLogix 5000, el cual es luego importado en dicho programa y cargado en el PLC de la impresora.

El esquema descrito divide las tareas individuales del proceso completo a lo largo de 4 programas: SolidWorks, Slic3r, GUI MATLAB de traducción y RSLogix 5000.

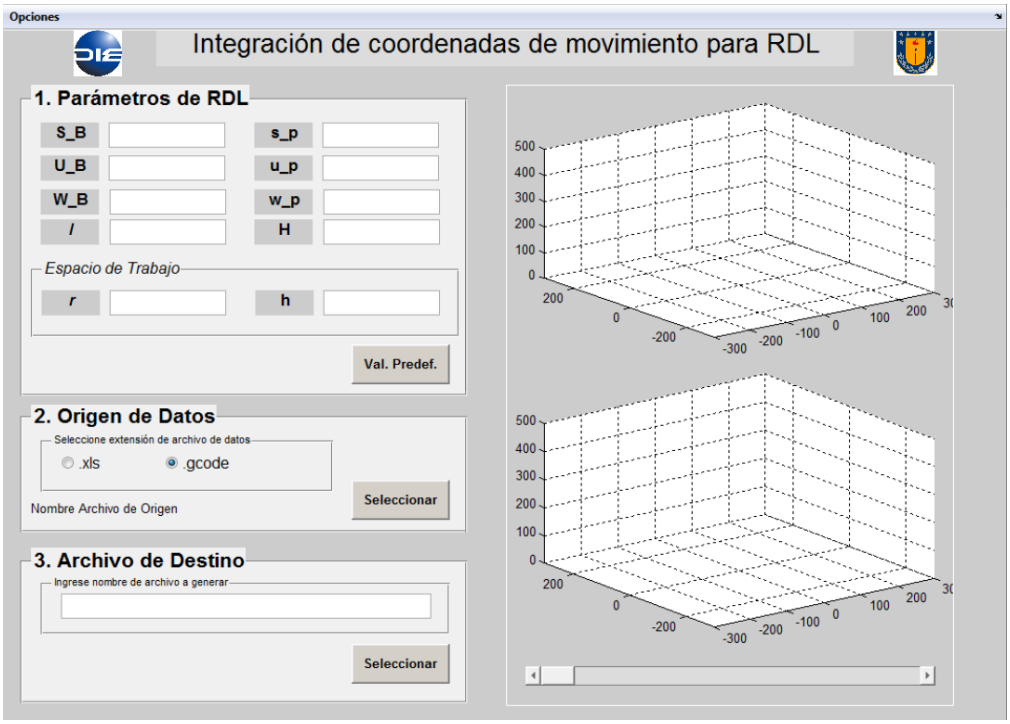


Fig. 1.4 Interfaz de traducción desarrollada en memorias pasadas. [2]

1.2.3 *Sobre robótica y robots orientados a la impresión 3D*

La automatización industrial ha dado origen a distintos robots que se encargan de ensamblar piezas, clasificar objetos, moldear piezas, etc. Dentro de estos diseños tenemos a los robots diseñados para la impresión 3D de piezas de pequeño y gran tamaño [3][11][19]. Estos toman distintas formas, como por ejemplo [30] muestra un diseño de un brazo robótico para imprimir cohetes espaciales en metal, mientras que en [3] habla del diseño de un robot tipo arco para imprimir muros de concreto (fig. 1.5).

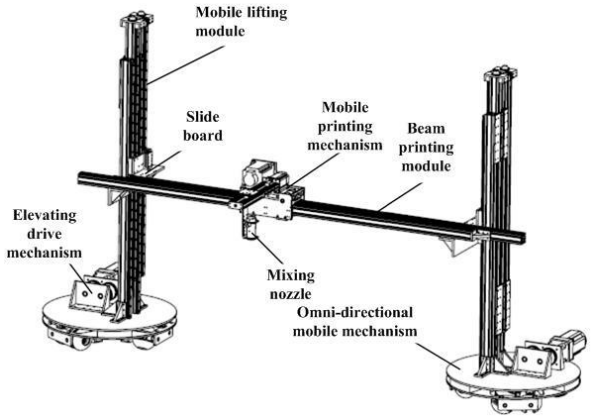
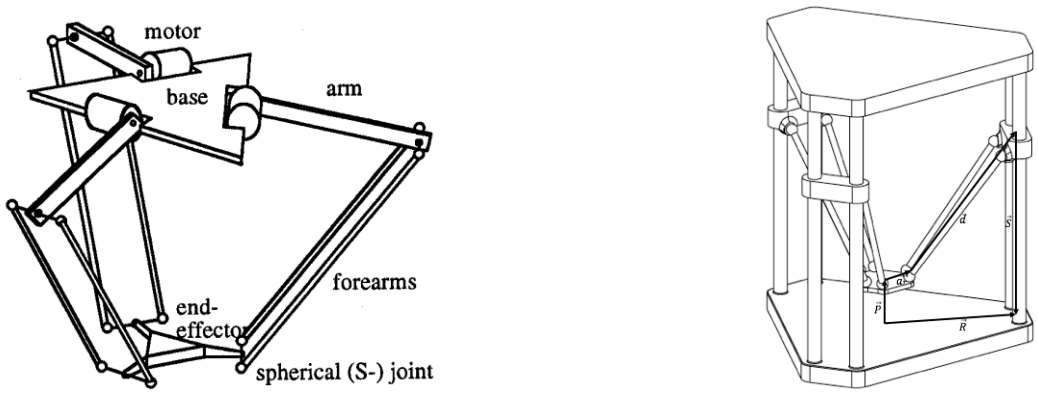


Fig. 1.5 Impresora 3D tipo pórtico controlada por PLC. [3]

Una de las categorías más utilizadas para impresión 3D son las llamadas impresoras 3D tipo delta (fig. 1.6). Estas consisten en una base, 3 pilares y 3 brazos sujetando el efector de la impresora en medio [18]. La posición del efector cambia dependiendo de las posiciones de los motores lineales que sujetan los brazos de la impresora. Tres brazos entregan una mayor velocidad y precisión en los movimientos realizados.

Fig. 1.6 Impresora 3D tipo delta [18]

Estas impresoras pertenecen a la categoría de los robots delta [14], un tipo de robot nacido en la industria chocolatera como una solución alternativa al proceso repetitivo de fabricación del producto. Como la mayoría de los brazos robóticos, se usan en situaciones que requieran mover objetos de un lugar a otro con mucha rapidez y precisión. Esta clase de robots se identifica por poseer una placa base fija que soporta el peso de la estructura, otra placa móvil donde se encuentra el efector y al menos 2 enlaces que las conecten. Estos enlaces están compuestos por un brazo y antebrazo, siendo este último conformado por dos barras paralelas. Esta configuración permite que ambas placas se mantengan paralelas al moverse. Existen 2 principales variantes de este robot: el robot delta angular y el robot delta lineal (fig. 1.7), siendo este último la especie a la que pertenecen las impresoras 3D tipo delta.



(a) **(b)**

Fig. 1.7 Robots delta
(a) Robot delta angular[12], **(b)** Robot delta lineal[29]

Para controlar estos robots se necesita resolver el problema inverso de posición. Este proceso matemático está documentado y ha sido resuelto previamente [28][29]. Las ecuaciones solución para el robot delta lineal son las presentes:

$$L_i^2 + 2zL_i + C_i = 0 \tag{1.1}$$

$$\therefore\ L_i = -z \pm \sqrt{z^2 - C_i} \tag{1.2}$$

Donde,

$$C_1 = x^2 + y^2 + z^2 + a^2 + b^2 + 2ax + 2by - l^2 \tag{1.3}$$

$$C_2 = x^2 + y^2 + z^2 + a^2 + b^2 - 2ax + 2by - l^2 \tag{1.4}$$

$$C_3 = x^2 + y^2 + z^2 + c^2 + 2cy - l^2 \tag{1.5}$$

$$a = \frac{S_B}{2} - \frac{s_p}{2} \tag{1.6}$$

$$\text{b} = W_B - w_p \tag{1.7}$$

$$\text{c} = u_p - U_B \tag{1.8}$$

1.2.4 *Sobre la generación y envío de instrucciones de impresión 3D*

Cuando se quiere imprimir una pieza tridimensional se necesita generar instrucciones de movimiento para la impresora. Esto se logra con los softwares rebanadores, los cuales dividen la figura en capas horizontales y formulan la ruta a seguir por el efector de la impresora. El lenguaje en el que estos programas generan esta ruta es conocido como G-code, el lenguaje estandarizado para la comunicación con dispositivos CNC [31].

Este lenguaje se basa en diferentes comandos y coordenadas XYZ asociadas a esos comandos. Los comandos los utiliza para indicar movimientos del extrusor, velocidad de extrusión del material, velocidad de movimiento del extrusor, temperatura de extrusión, entre otros parámetros de impresión (ver tabla 1.1).

TABLA 1.1: Comandos utilizados en lenguaje G-code.

Variable / Comando	Interpretación
G0, G1	Movimientos lineales
F	Cambio de velocidad de movimiento
E	Cambio de velocidad de extrusión
M	Comandos misceláneos

```
G0 F12000 X94.868 Y52.918 Z0.24
;TYPE:SUPPORT-INTERFACE
G1 F2700 E0
G1 F1800 X94.935 Y52.801 E0.01076
G1 X95.409 Y52.022 E0.08355
G1 X95.509 Y51.871 E0.09801
G1 X96.02 Y51.157 E0.1681
G1 X96.133 Y51.009 E0.18296
G1 X96.709 Y50.318 E0.25477
G1 X96.81 Y50.205 E0.26687
G1 X97.435 Y49.556 E0.33879
G1 X97.56 Y49.436 E0.35262
G1 X98.231 Y48.833 E0.42463
```

Fig. 1.8 Extracto de un archivo escrito en lenguaje G-code

Utilizando este lenguaje se puede crear un listado de órdenes a seguir por las impresoras 3D. En él, las coordenadas se encuentran divididas por cada eje y vienen escritas en milímetros. Las impresoras deben interpretar esas coordenadas y traducirlas a las posiciones que debe tomar cada motor. Esto lo hacen resolviendo el problema inverso de posición correspondiente a su situación particular [2][3][11][19][28][29]. Este proceso se puede realizar antes de enviar las instrucciones a la impresora o puede realizarlo la impresora misma.

El envío de instrucciones se puede hacer con una memoria flash [18][24][25][26][27] o realizando una conexión entre el computador y el controlador de la impresora [17][21][22][32]. Ambos métodos son posibles a través de los softwares rebanadores, pero realizar una comunicación con la impresora tiene la ventaja de poder monitorear y hasta controlar el proceso de impresión.

En el caso de impresoras controladas por PLCs existen diversas bibliotecas de programación que permiten la comunicación con computadores personales [21]. Al seleccionar cual biblioteca utilizar se debe analizar el lenguaje de programación que se desea utilizar y el dispositivo con el que se realizará la comunicación. Por ejemplo, si se requiere una comunicación con dispositivos Allen-Bradley se puede utilizar el lenguaje C para aprovechar la biblioteca "libplctag" [22] o bien, si se prefiere Python se dispone de la biblioteca "Pycomm" o "Pycomm3" [17], dependiendo de la versión de Python preferida.

1.2.5 *Discusión*

Como se ha aclarado previamente, el escenario actual desarrollado en memorias anteriores [1][2] completa el proceso de impresión utilizando 4 softwares independientes, en donde cada uno cumple un solo propósito (fig. 1.9). Si bien esto funciona, existe una ineficiencia en el número de programas y etapas que se necesita, lo que entorpece el procedimiento y aumenta el riesgo de errores humanos en el proceso.

Como ha quedado demostrado en [11][17][21][22][23], es posible programar un software capaz de comunicarse con un PLC a través de lenguajes convencionales y, gracias a las herramientas de Python, no existe impedimento para unir la etapa de traducción y la de comunicación en un solo software. Sin embargo, si se escoge cuidadosamente el software de rebanado, entonces la etapa de rebanado también se podría integrar al software. Existen varios en el mercado [6], pero se debe escoger un programa de rebanado que sea código abierto, escrito en su mayoría en Python y que admita la integración de plugins para poder incorporar las etapas de traducción y comunicación. Bajo estos requerimientos el programa más apto para esta tarea es Ultimaker Cura [25], software multiplataforma de código abierto y escrito en su mayoría en Python y QML.

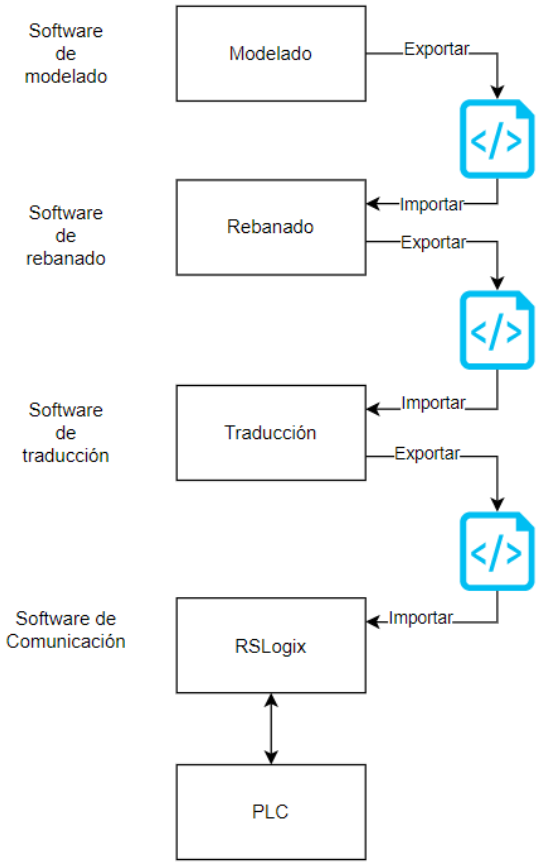


Fig. 1.9 Esquema del proceso de impresión actual desarrollado en memorias anteriores. [1][2]

1.3. Definición del problema

El proceso actual para generar instrucciones y entregárselas a la impresora, ocupa 4 programas de computadora en total para completarse (fig. 1.9). Esto se puede reducir a solo 2 programas con la creación de una interfaz gráfica multiplataforma (Windows, Ubuntu y Mac) que condense las tareas realizadas por los softwares de rebanado, traducción, comunicación y monitoreo (fig. 1.10).

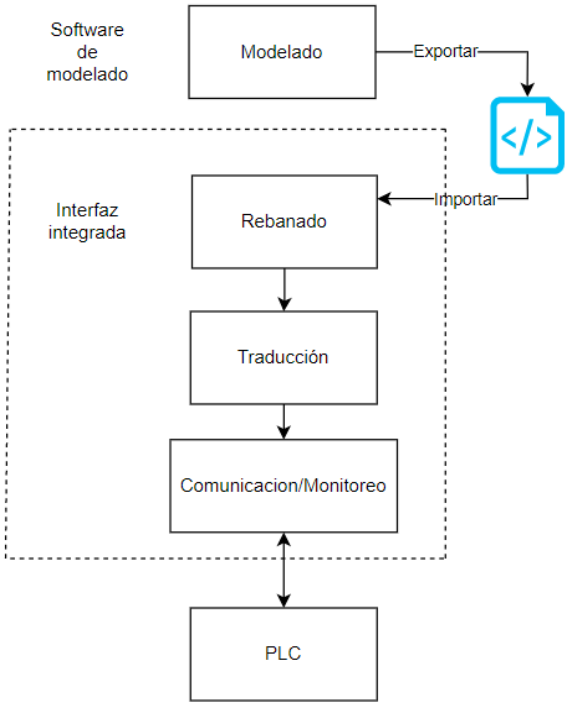


Fig.1.10 Esquema del proceso de impresión propuesto.

1.4. Objetivos

1.4.1 *Objetivo General*

Diseñar y desarrollar una interfaz gráfica multiplataforma que optimice el proceso de generación de instrucciones, comunicación con la impresora y monitoreo del proceso de impresión llevado a cabo por la impresora 3D de concreto industrial desarrollada en memorias anteriores [1][2].

1.4.2 *Objetivos Específicos*

- Integrar la generación de instrucciones de movimiento, comunicación con la impresora y monitoreo del proceso de impresión 3D en un solo programa.
- Diseñar una interfaz gráfica auto explicativa y amistosa al usuario para que cualquier persona pueda usarla sin problemas, independientemente de su nivel de conocimiento del proceso de impresión.
- Comprobar su correcto funcionamiento a través de un Factory Acceptance Test utilizando equipos presentes en el laboratorio de control de procesos de la universidad de Concepción.

1.5. Alcances y Limitaciones

- La parte gráfica de la interfaz será programada mayoritariamente en lenguaje QML.
- La interfaz se desarrollará como un plugin del programa rebanador Ultimaker Cura.
- El código que rige la lógica principal detrás de la interfaz y que será responsable de su funcionamiento será escrito mayoritariamente en Python.
- La comunicación se realizará con las funciones incluidas en la biblioteca Pycomm3 de Python.

1.6. Temario y Metodología

En esta memoria de título se persigue optimizar el proceso descrito en la figura 1.9, siendo el objetivo final una interfaz de usuario integrada que lleve el proceso a como es descrito en la figura 1.10:

En el capítulo 2 (diseño de un bosquejo de la interfaz) se explican las necesidades que se buscan satisfacer y el razonamiento de diseño utilizado en la creación de un bosquejo general de la interfaz para cumplir dichas tareas. El capítulo 3 (desarrollo de la interfaz gráfica) muestra las herramientas utilizadas para la programación de la etapa gráfica de la interfaz, basada en el bosquejo diseñado. En el capítulo 4 (Desarrollo de la generación de instrucciones) se construye la etapa de traducción, correspondiente a la generación de las instrucciones de impresión a partir del G-code creado por Ultimaker Cura. El capítulo 5 (Desarrollo de una comunicación entre el software y PLC) explica cómo se usaron las distintas herramientas de Pycomm3 para llevar a cabo la comunicación entre el software y el PLC de la impresora 3D, todo con el objetivo de enviar las instrucciones al PLC y de monitorear el proceso de impresión. Luego, en el capítulo 6 (implementación y puesta en marcha) se llevará a cabo la implementación de la interfaz en un ambiente virtual y también en uno real, exponiendo el funcionamiento de la interfaz y obteniendo resultados del proceso completo. Por último, en el capítulo 7 se discutirán estos resultados y la importancia de éstos.

2. Diseño de un bosquejo de la interfaz

2.1. Necesidades por satisfacer

La nueva interfaz responde a la oportunidad de optimizar el proceso de impresión con un programa que reemplace las tareas soportadas por los actuales softwares. Debe ser capaz de integrarse con fluidez dentro del programa Ultimaker Cura de forma que el usuario sienta que no utiliza un software externo. Las tareas por cumplir son:

- Generación de instrucciones a partir del G-code creado por U.C.
- Conexión con el controlador de la impresora (PLC).
- Envío de instrucciones a la impresora.
- Dar inicio al proceso de impresión.
- Monitoreo del proceso.
- Control de movimiento de la impresora.

Es con estas misiones en mente que se decidió dividir la interfaz en varias pestañas. Asimismo, cada pestaña puede cumplir a cabalidad la tarea que se le asigne. Es importante hacer notar que los bosquejos son solo una guía básica de lo que se desea lograr, por lo que el diseño evolucionó a lo largo del desarrollo del proyecto.

2.2. Acceso a la interfaz

Ultimaker Cura es un software de código abierto que acepta extensiones desarrolladas por terceros, característica que se aprovechó en este proyecto para integrar la interfaz a este programa rebanador. Al hacer esto el acceso a la interfaz se encontrará en la barra de herramientas presente en la parte superior de U.C., tal como se muestra en la figura 2.1.

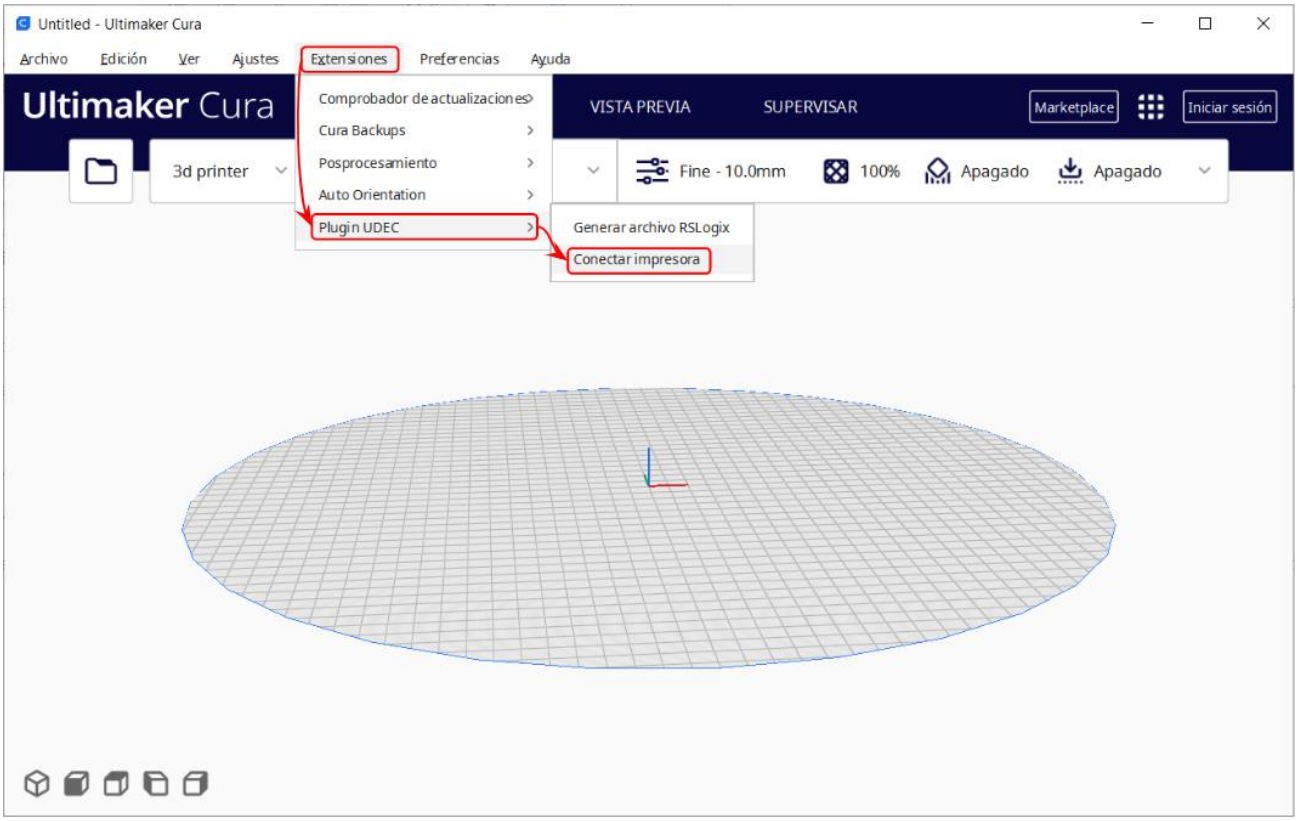


Fig. 2.1 Forma de acceder a la nueva interfaz dentro de Ultimaker Cura

2.3. Pestañas de la interfaz y sus propósitos

2.3.1 *Pestaña de conexión y generación de instrucciones*

Esta pestaña es la primera pantalla que verá el usuario al iniciar la interfaz y se encargará de generar las instrucciones de impresión y de realizar la conexión con la impresora. También entregará al usuario indicaciones sobre como iniciar el proceso de impresión. En ella se encontrará un apartado para ingresar los parámetros de la impresora (tabla 2.1) necesarios para el cálculo de las instrucciones (1), junto a una barra de progreso y el botón para iniciar el cálculo (2). Este debe ser el primer paso del proceso, pues sin instrucciones no se necesita abrir una conexión.

Un segundo apartado se hará cargo de la conexión con la impresora. Para ello se necesita que el usuario ingrese la dirección IP del controlador de la impresora, por lo que se requiere una caja de texto (3). Al realizar la conexión se extraerá la información básica del controlador y se le mostrará al usuario (4) para que compruebe si se conectó al dispositivo correcto. Por último, se agregaron botones globales para enviar las instrucciones a la impresora e iniciar la impresión (5), junto con los botones para acceder a las otras pestañas (6). Con este razonamiento se creó el bosquejo de la figura 2.2.

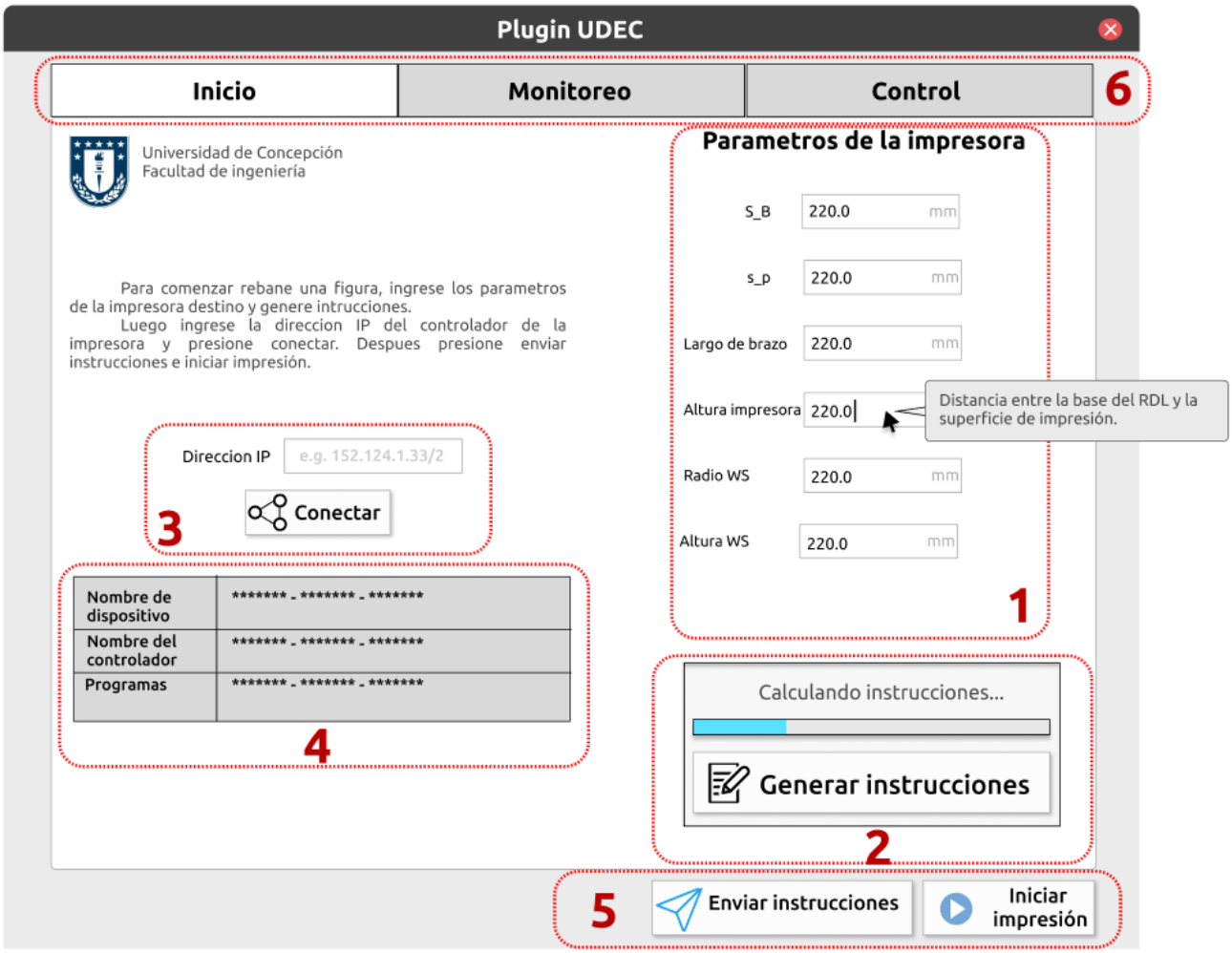


Fig. 2.2 Bosquejo de la pestaña de conexión y generación de instrucciones

TABLA 2.1: Parámetros de la impresora

Parámetro	Descripción
S_B	Distancia entre los actuadores (motores) de la impresora. En la impresora diseñada corresponde a 646 mm.
s_p	Distancia entre los puntos de conexión del efector y los brazos del robot. En la impresora diseñada 108 mm.
Largo del brazo	Es el largo de los brazos de la impresora. En la impresora diseñada corresponde a 983 mm.
Altura de impresora	Es la distancia entre la base del robot y la superficie de impresión. En la impresora diseñada corresponde a 1460 mm.
Radio WS	Es el radio de la base del espacio de trabajo. En la impresora diseñada corresponde a 225 mm.
Altura WS	Es la altura del espacio de trabajo. En la impresora diseñada corresponde a 500 mm.

2.3.2 *Pestaña de monitoreo de variables*

En esta pestaña el usuario debe poder monitorear todos los aspectos relevantes del proceso, como el estado de avance de la impresión. Para esto se tendrá una serie de gráficos que muestran los valores de los tags que el usuario seleccione, y también una barra de carga que muestre el avance del proceso (4). Los gráficos estarán compuestos por 3 secciones: la leyenda (1), el plano que muestra los valores históricos de los tags seleccionados (2) y botones que despliegan el listado de tags disponibles para monitorear, junto con cajas de texto para modificar sus valores (3).

Para seleccionar que es lo que mostrarán los gráficos, se le dará al usuario la opción de seleccionar un máximo de 4 tags a graficar. Los datos mostrados corresponderán a el último minuto pasado. Así, un bosquejo general de esta pestaña se encuentra en la figura 2.3:

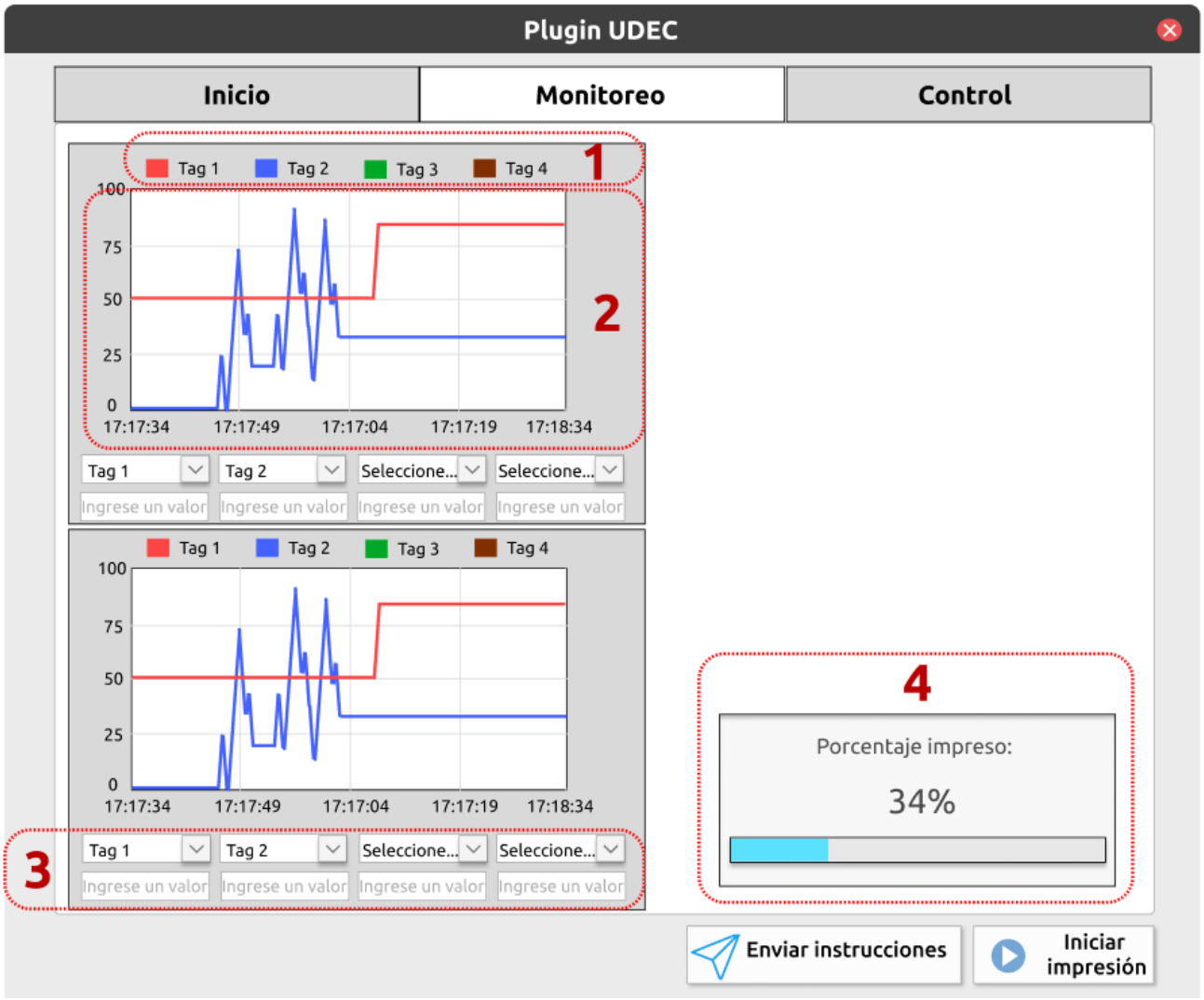


Fig. 2.3 Bosquejo de la pestaña de monitoreo

2.3.3 *Pestaña de control de impresora*

Con esta pestaña se le permitirá al usuario controlar distintos aspectos de la impresora, como la velocidad máxima de movimiento del efector, llevar el efector al origen, mover el efector libremente o pausar la impresión. Para esto debe poseer una sección con botones que aumenten o disminuyan las posiciones de cada eje XYZ. Esto se realizará con botones que aumenten o disminuyan en 10 y 100 unidades las posiciones en cada eje, separados entre plano XY (1) y eje Z (2). Bajo ellos se ubicarán cajas de texto que permitan ingresar una coordenada especifica (3). Cualquier movimiento que se desee realizar se iniciará con el botón “Realizar movimiento”.

Para agregar el resto de las funciones se tendrá un apartado con diferentes botones asignados a cada tarea (4). Aquí se podrá modificar la velocidad de impresión, detenerla completamente e iniciar una rutina para que el actuador vuelva al origen (solo con la impresora detenida). Con estas ideas se llegó al bosquejo de la figura 2.4.

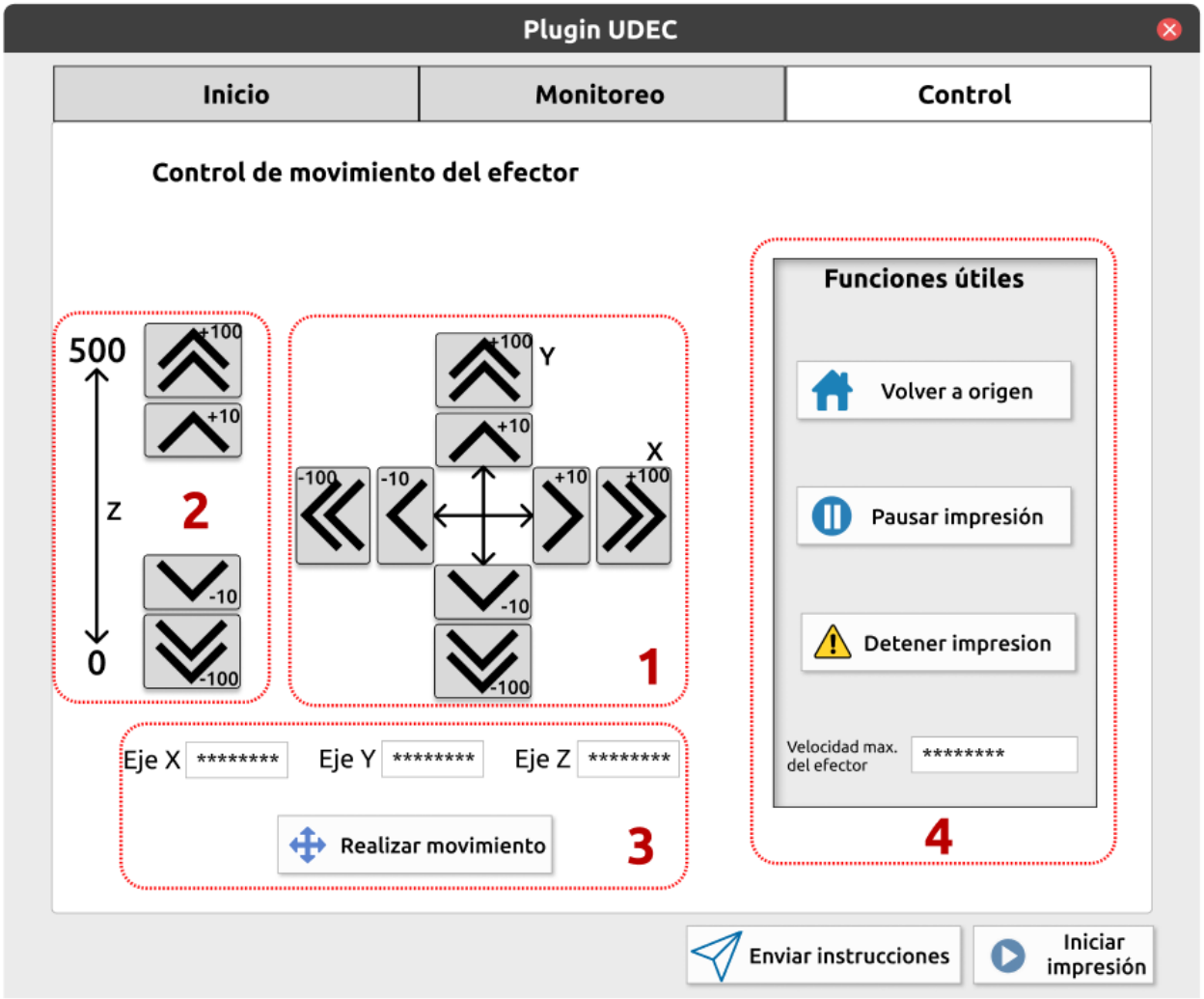


Fig. 2.4 Bosquejo de la pestaña de control

3. Desarrollo de la interfaz gráfica

3.1. Herramientas utilizadas

Para el desarrollo de la interfaz se decidió utilizar el lenguaje de programación QML [33]. Este lenguaje fue creado especialmente para el diseño de interfaces gráficas. Está equipado con bibliotecas que permiten una profunda customización de los componentes de la interfaz, como cajas de texto, botones, imágenes, menú de pestañas, etc. Estos elementos son llamados "componentes QML". Los elementos de la interfaz se declaran como un árbol de elementos y cada uno de ellos posee parámetros propios de cada uno. Ejemplos de parámetros son los colores del elemento, el tamaño, que tipo de texto reciben, cuando hacerse visibles, cuando se les puede hacer clic o que sucede al posar el mouse sobre ellos.

Para la programación de esta etapa (y también las siguientes) se utilizó el IDE QtCreator, el cual viene equipado con las bibliotecas necesarias para este desarrollo. Este IDE ayuda con los lenguajes QML, Python, JavaScript, C++ y C, por lo que es ideal para el actual desarrollo.

3.2. Programación en QML

3.2.1 *Elementos generales de la interfaz*

La interfaz se desarrolló en el archivo QML principal llamado *MainWindow.qml*, pero algunos de los elementos de la interfaz se crearon como componentes independientes en otros archivos. Las bibliotecas utilizadas en *MainWindow.qml* para la creación de la interfaz son las siguientes:

```
import QtQuick 2.15
import QtQml 2.0
import QtQuick.Window 2.15
import QtQuick.Controls
import QtQuick.Layouts
```

Fig. 3.1 Bibliotecas Qt importadas

Para crear las pantallas se necesita dividir la interfaz en pestañas. Esto se logra con el componente QML *TabBar*. Con el podemos agregar pestañas a la interfaz de forma simple y también establecer condiciones de ingreso a ellas. Esto significa que se puede prohibir el acceso a las pestañas de monitoreo y control mientras no se logre una conexión con una impresora.

Declaración de tamaño y ubicación del componente.

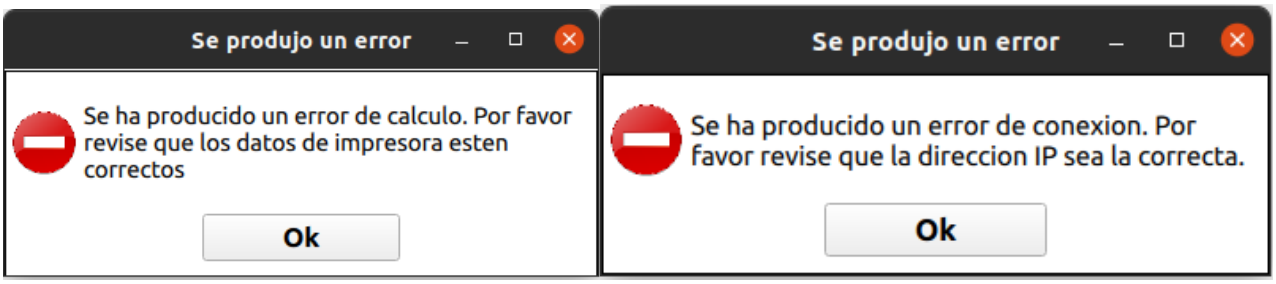
Declaración del botón de la primera pestaña, incluyendo título y posición.

```
header: TabBar{
    id: coreBar
    width: parent.width - 50
    anchors.horizontalCenter: parent.horizontalCenter
    TabButton{
        height: 40
        contentItem: Label {
            text: qsTr("Instrucciones y conexión")
            verticalAlignment: Text.AlignVCenter
            horizontalAlignment: Text.AlignHCenter
        }
    }
```

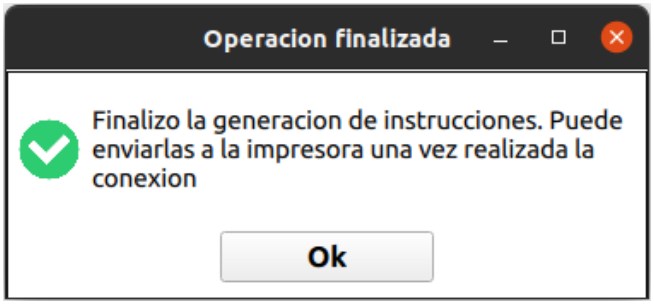
Fig. 3.2 Declaración de pestaña en TabBar

Fuera de las pestañas tenemos los botones globales que se encargan de enviar instrucciones, iniciar el proceso de impresión y detener forzosamente los motores. Estos botones se programan para que no se puedan cliquear si no existe una conexión con la impresora y un listado de instrucciones disponible.

Por último, también se creó una ventana pop-up que se encarga de mantener informado al usuario sobre el proceso. Se utiliza para informar de errores de conexión, errores de cálculo de instrucciones y también del fin del proceso de cálculo o de impresión.



(a) **(b)**



(c)

Fig. 3.3 Diferentes usos de ventana pop-up

(a) Error de cálculo, **(b)** Error de conexión, **(c)** Fin del proceso de calculo

3.2.2 *Pestaña de conexión y generación de instrucciones*

Esta pestaña debe asegurarse de que los datos ingresados en las cajas de texto sean los datos esperados. La dirección IP del PLC debe tener el formato:

$$< IP\ \_\ address >/< PLC\_Slot >$$

Para forzar al usuario a ingresar una dirección IP válida se utiliza el parámetro *RegularExpressionValidator* del componente *TextField*. Con este parámetro podemos indicar el formato exacto que queremos que sea ingresado. Esto se debe hacer con sintaxis JavaScript o el código no lo leerá. Entonces, el parámetro será el siguiente:

Declaración del formato admisible exacto, expresado en lenguaje JavaScript.

```
TextField{
    id: ip_field

    width: 140
    placeholderText: qsTr("e.g. 192.168.0.15/2")
    validator: RegularExpressionValidator{
        regularExpression: /^(([01]?[0-9]?[0-9]|2([0-4][0-9]|5[0-5]))\.){3}
                           //([01]?[0-9]?[0-9]|2([0-4][0-9]|5[0-5]))\
                           //(([0-6]|3([0])|2([0-9]))\.)$/}
}
```

Fig. 3.4 TextField receptor de dirección IP

Lo siguiente son las cajas de texto para ingresar los parámetros de la impresora. En total son 6 parámetros, por lo que para ahorrar espacio en el código se creó un nuevo componente QML y se guardó en el archivo ParamArea.*qml*, el cual se puede utilizar dentro del código principal *MainWindow.qml*. Al interior de este nuevo archivo se definen las características principales del componente. Las cajas de texto que reciben los parámetros deben mostrar un texto explicativo cuando se posa el mouse sobre ellos y deben recibir sólo números. Al utilizar este componente en *MainWindow.qml* se pueden establecer los parámetros internos de él, como el nombre de cada medida de la impresora, el contenido del texto de ayuda y el lado en donde desplegar este texto.

Declaración del nombre del parámetro, texto de ayuda y posiciones.

```
ParamArea{
    id: sb
    paramText: "538"
    name: "S_B"
    help: " Es la distancia entre los actuadores del RDL "
    helpSide: "left"
    Layout.alignment: Qt.AlignHCenter
}
```

Fig. 3.5 ParamArea.qml utilizado dentro de MainWindow.qml

Luego, se colocan el botón para generar instrucciones y la barra de carga asociada. Esta barra corresponde al componente QML *ProgressBar* y muestra el avance del cálculo. Por último, se colocan el botón para realizar conexión y las cajas de texto con información. El botón para generar instrucciones se mantiene inhabilitado hasta que se rebane una figura, mientras que el botón de conexión se activa cuando existen instrucciones en memoria listas para su envío. Con estos pasos, el aspecto final de la pestaña es el de la figura 3.6.

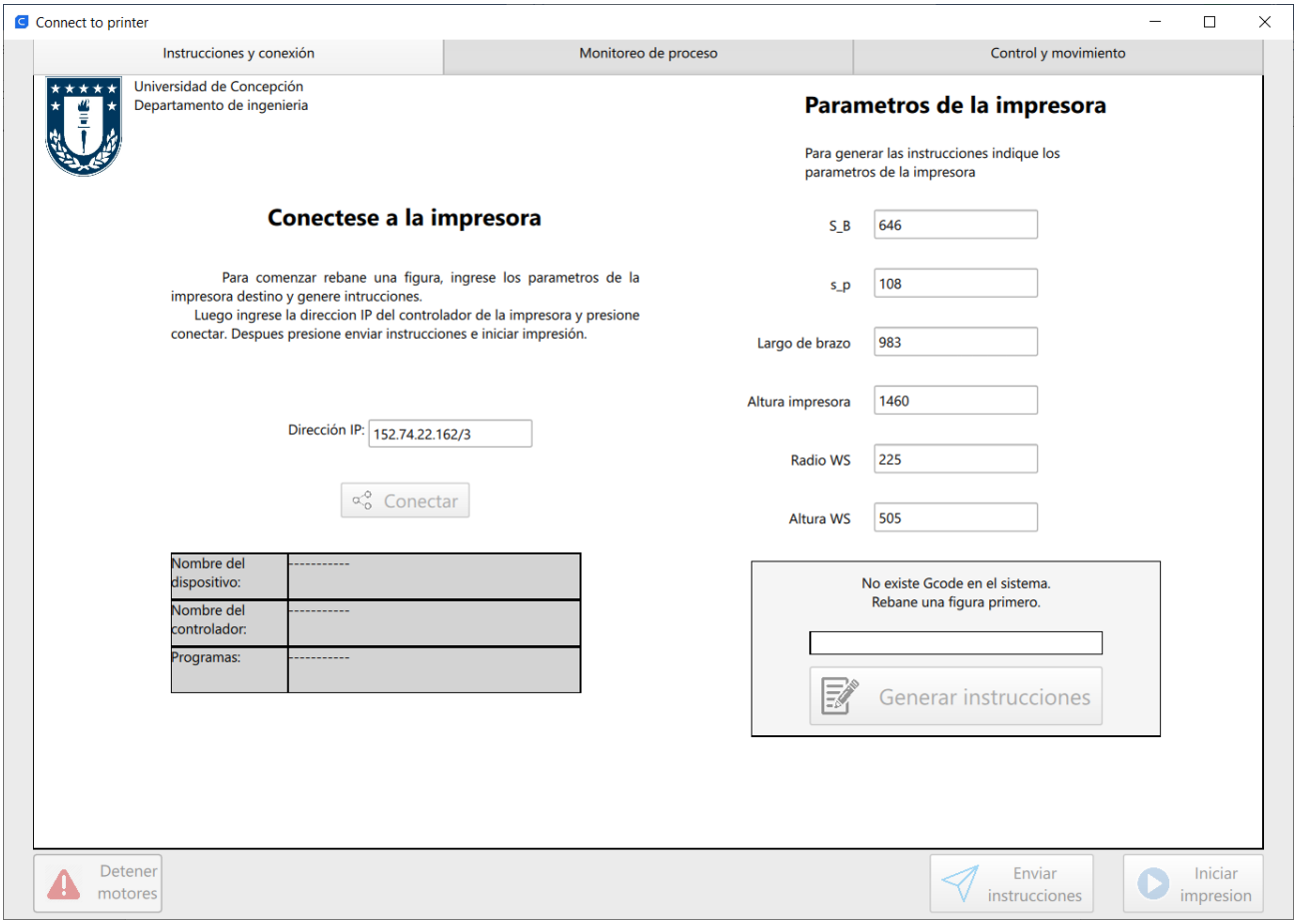


Fig. 3.6 Primera pestaña programada en lenguaje QML

3.2.3 *Pestaña de monitoreo*

Esta pantalla consiste en algunos gráficos temporales con menús desplegables para escoger variables a monitorear. Los gráficos se crean con la biblioteca *matplotlib*, se convierten a imágenes y luego éstas se cargan en la interfaz. Para crear los gráficos se deben generar los arreglos de valores que se mostrarán en él. Cada vez que el usuario selecciona un tag del listado, se realizan las peticiones de lectura y los valores leídos se guardan en arreglos de la biblioteca *NumPy* para mayor rapidez de trabajo. Una vez que se tienen los arreglos de valores se llama a la función *plot()* encargada de generar el grafico y convertirlo a imagen.

Configuración inicial del gráfico: Ejes, título, límites y tamaño.

Se agregan los valores extraídos de la impresora al gráfico configurado anteriormente.

```
def plot(self, name, value_arrays):
    self.imageProvider = matplt.MatplotlibImageProvider()
    gain = 1.4
    figure = self.imageProvider.addFigure(name, figsize=(6.4*gain,4.8*gain))
    ax = figure.add_subplot()
    ax.grid(linewidth=2)
    ax.set(aspect='auto', ylim=[0,100],
           xlim=[self.bias_time(datetime.now(), 0, -1, 0),
           datetime.now()])
    ax.set_xlabel('Hora', fontsize=30)
    ax.set_ylabel('Valor', fontsize=30)
    ax.set_title("Visualización de variables", fontsize=30)
    ax.tick_params(axis='both', which='both', labelsize=15)

    for tag in range(len(value_arrays)):
        n_of_values = len(value_arrays[tag])
        if len(value_arrays[tag]) < 60:
            X = np.array([self.bias_time(datetime.now(), 0, 0, -n_of_values + i)
                for i in range(n_of_values)])
        else:
            X = np.array([self.bias_time(datetime.now(), 0, 0, -60 + i)
                for i in range(60)])
        Y = np.array(value_arrays[tag])
        ax.plot(X,Y, linewidth=3, label='tag'+str(tag))
    ax.legend(fontsize='xx-large', loc='upper left')
```

Fig. 3.7 Función plot() que crea imágenes con gráficos de 2 ejes

Para los ejes Y se utiliza un eje de valores entre 0 y 100, mientras que para el eje X se utiliza un eje temporal que muestra el último minuto. Para que el gráfico se actualice se tiene que usar el componente *Timer*, el cual ejecuta y repite las ordenes definidas 1 vez cada cierto tiempo establecido.

Declaración de intervalo de activación y encendido de función repetidora.

Cada vez que se active el timer se extraen los nombres de los tags seleccionados y se actualizan los gráficos con los valores de ellos.

```
Timer{
    id: date_timer
    interval: 1000
    running: false
    repeat: true
    onTriggered: {
        var upper_tagboxes_info = {
            "tagbox_1" : [tagbox1_active, tagbox_1.combobox.currentText, 0],
            "tagbox_2" : [tagbox2_active, tagbox_2.combobox.currentText, 1],
            "tagbox_3" : [tagbox3_active, tagbox_3.combobox.currentText, 2],
            "tagbox_4" : [tagbox4_active, tagbox_4.combobox.currentText, 3]}
        var lower_tagboxes_info = {
            "tagbox2_1" : [tagbox1_active, tagbox2_1.combobox.currentText, 4],
            "tagbox2_2" : [tagbox2_active, tagbox2_2.combobox.currentText, 5],
            "tagbox2_3" : [tagbox3_active, tagbox2_3.combobox.currentText, 6],
            "tagbox2_4" : [tagbox4_active, tagbox2_4.combobox.currentText, 7]}
        var upper_active_tagboxes = who_active(upper_tagboxes_info)
        var lower_active_tagboxes = who_active(lower_tagboxes_info)

        update_charts(upper_active_tagboxes, lower_active_tagboxes)
        reload_upper_plot()
        reload_lower_plot()
    }
```

Fig. 3.8 Componente QML Timer

Con este componente activándose cada 1 segundo, actualizamos los valores desplegados en el gráfico. Importante es saber, que al iniciar la interfaz este *Timer* se encuentra desactivado y se activa una vez que se selecciona una variable en los menús desplegables.

Los menús desplegables se crean con el componente QML *ComboBox*. Los elementos que se almacenen en el menú se deben agregar a través de su propiedad *model*, el cual recibe un listado (o arreglo) de elementos. Este listado es el grupo de tags del PLC y se agregan a todos los *ComboBox* de la pestaña cuando se logra la conexión con la impresora. Bajo estos componentes se agregaron cajas de texto para visualizar y cambiar los valores de las variables desplegadas. Los valores aceptados son números reales y se envían cuando el usuario presione la tecla “enter”.

Con la ayuda de *matplotlib* se pueden también crear gráficos 3D. Estas herramientas se usaron para generar un gráfico tridimensional que siga en vivo el avance de impresión de la figura, mostrando la ruta por la que ha pasado el efector del RDL. Al igual que ambos gráficos 2D, al iniciar la impresión se inicia un *Timer* que actualizará la ruta recorrida. Finalmente, se agrega una caja que contenga el porcentaje completado de impresión y una barra de carga que represente el avance del proceso de impresión. Así, la pestaña de monitoreo queda como en la figura 3.9.

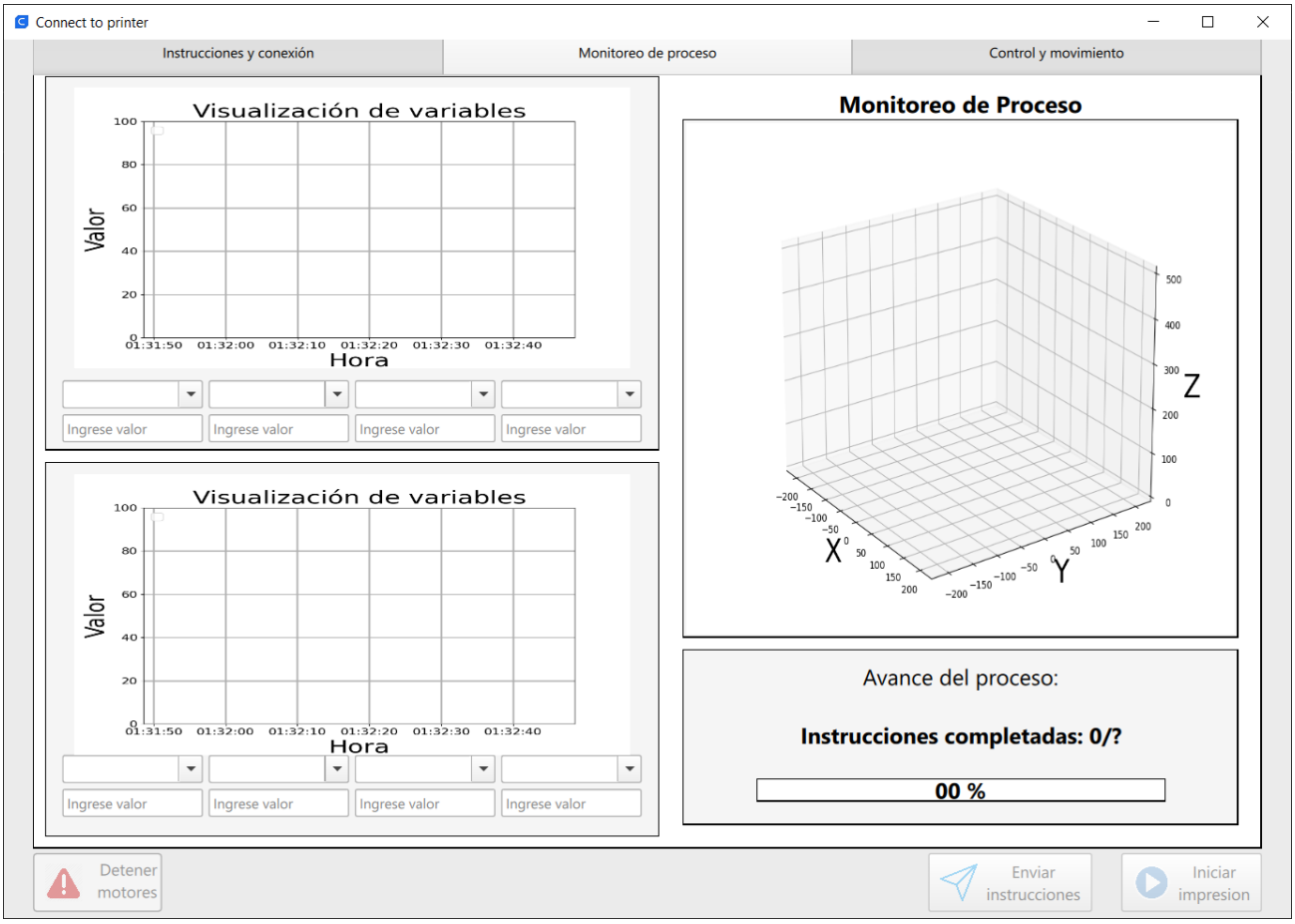


Fig. 3.9 Segunda pestaña programada en lenguaje QML

3.2.4 *Pestaña de control de impresora*

Dentro de esta pestaña se encuentran funciones básicas de control de la impresora. De acuerdo con el bosquejo, se agregaron una serie de botones que permiten mover libremente el efector de la impresora. Estos botones se dividen por ejes, existiendo un botón para movimientos pequeños y otro para movimientos grandes. Dichos botones se repiten a lo largo de la pestaña, por lo que se creó el componente *ArrowButtons.qml*, el que contiene ambos botones.

Declaración de la posición del botón.

Se aplica una rotación al botón correspondiente al eje al que pertenece, en este caso eje Z.

```
ArrowButtons{
    id: zup;
    anchors.left: zreference.left;
    anchors.top: zreference.top ;
    transform: Rotation{
        origin.x: 0;
        origin.y: 0;
        angle: 270
    }
}
```

Fig. 3.10 Componente ArrowButtons dentro de MainWindow.qml

Utilizando este componente, se ubican los botones necesarios para cada eje tal como se propuso en el bosquejo (fig. 2.3). Bajo estos botones se colocan las cajas de texto para ingresar directamente la coordenada de destino, velocidad de desplazamiento y un botón para enviar la orden de movimiento a la impresora. Por último, se crea un apartado en la zona derecha con los botones para volver al origen y para detener la impresión, además de un componente *spinbox* para poder cambiar la velocidad de movimiento del efector al momento de imprimir y cajas de texto para modificar las ganancias de los lazos de control de los motores.

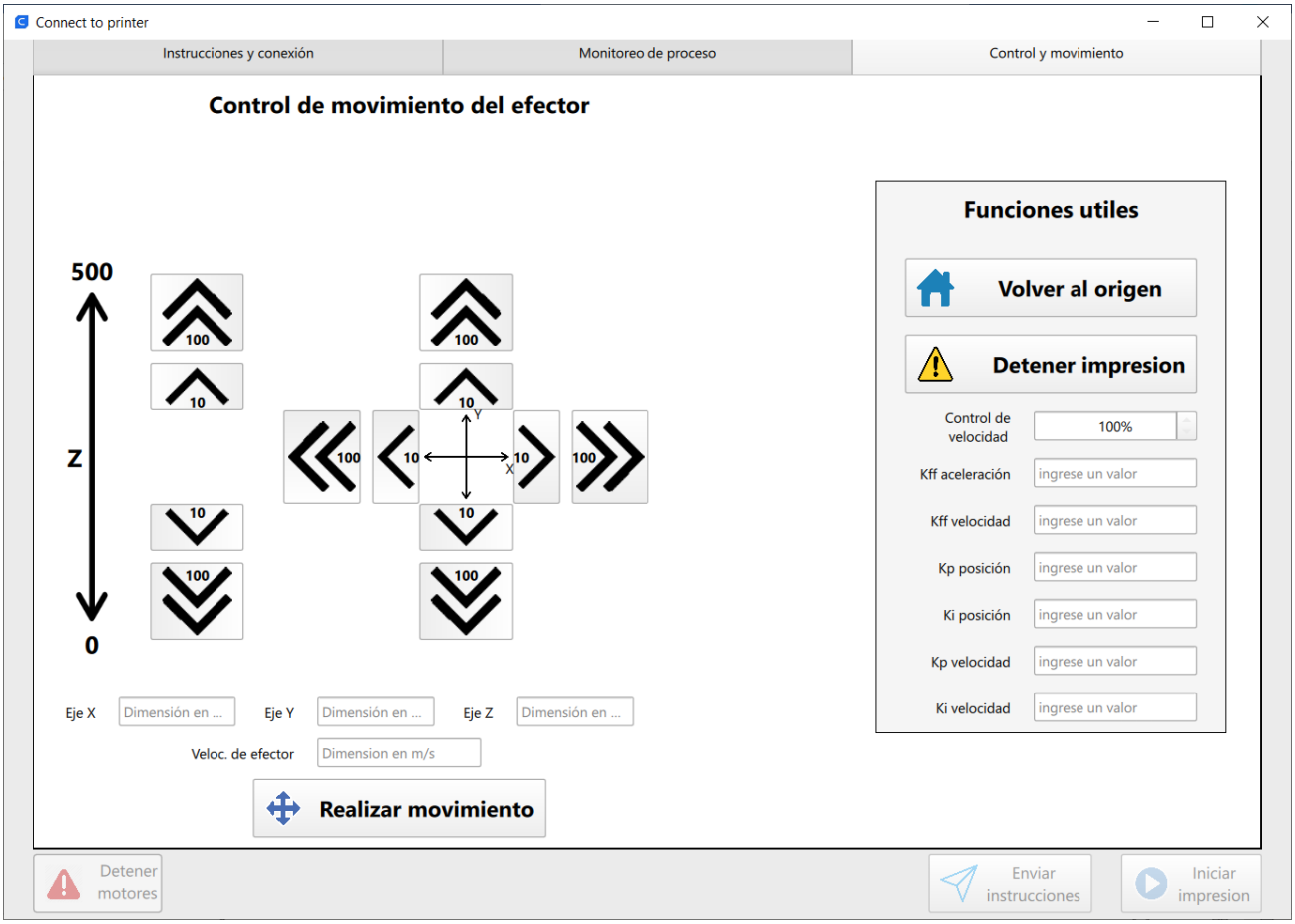


Fig. 3.11 Tercera pestaña programada en lenguaje QML

4. Desarrollo de la generación de instrucciones

4.1. Teoría previa

Esta etapa corresponde a la traducción de las coordenadas de impresión, para obtener las instrucciones de movimiento de los motores. Es por esto, que necesitamos las ecuaciones solución del problema inverso de posición del robot delta lineal. Estas ecuaciones han sido extensamente estudiadas en trabajos previos a este [2][28], llegando a las ecuaciones de la figura 1.8.

Se aprovechará el avance realizado en proyectos anteriores [34], en donde se programó en Python un método para extraer el G-code directamente de la memoria interna de Ultimaker Cura y se utilizaron las ecuaciones solución anterior nombradas para calcular las posiciones de los motores. Para el desarrollo de esta etapa de la presente memoria de título, se hará uso de las mismas funciones ocupadas en [34]. El código completo de cada archivo se puede encontrar en anexos.

Para que la interfaz sea compatible con Ultimaker Cura se necesita integrarla como un plugin. Los plugins tienen una estructura definida en la documentación del programa [25], deben tener por lo menos un archivo *init.py* para iniciar el plugin, un archivo *json* con una descripción de las características del plugin y el archivo Python que se encargue de la lógica principal.

4.2. Generación de instrucciones

El proceso se inicia al presionar el botón “generar instrucciones” en la pestaña de conexión y generación de instrucciones. Con esta acción se llama a la función *generate_instructions_list()*, la cual toma las medidas de la impresora ingresadas por el usuario, para calcular las instrucciones de movimiento y luego guardarlas en un listado. El orden de acciones es el siguiente:

1. Se verifica que los parámetros de impresora ingresados son válidos, es decir que sean distintos de cero.
   1.1. Si se encuentra un elemento invalido se enseña un pop-up que indica el error al usuario.
2. Se extraen las coordenadas y velocidades de movimiento desde el G-code generado por Ultimaker Cura y se almacenan en el listado *coordinates*.
3. Se comprueba que las coordenadas extraídas encajen dentro del espacio de trabajo de la impresora ingresada.
   3.1. Si las coordenadas se extienden fuera del espacio de trabajo se muestra una ventana de error.

4. Se aplica un corrimiento a las coordenadas del eje Z para mover el origen del sistema de coordenadas.
5. Se calculan las instrucciones de movimiento con la solución del problema inverso de posición y se almacenan en el listado *self.positions_list*. Se indica el estado de avance del cálculo en la barra de avance.
   5.1. Si los cálculos producen un error matemático se despliega un mensaje de error. Esto puede pasar si se ingresan medidas físicamente imposibles para la impresora.
6. Se despliega un mensaje indicando que el proceso de creación de instrucciones finalizó.

Se revisa que los parámetros sean válidos. Si lo son, se continúa a extraer las coordenadas del G-code generado previamente.

Se chequea que las coordenadas se encuentren dentro de los límites de la impresora. si lo están, se calculan las instrucciones de movimiento y almacenan en una lista. Se muestra un mensaje al usuario en caso de errores de cálculo.

Finalización del cálculo. Se despliega un aviso al usuario en pantalla.

```
@pyqtSlot(float, float, float, float, float, float)
def generate_instructions_list(self, sb, sp, arm_length, height, ws_radio, ws_height):
    """Responsible for using the given parameters to call the functions which
       calculate the instructions."""

    params = [sb, sp, arm_length, height, ws_radio, ws_height]
    self.params = params
    if not self.are_valid(params):
        Logger.log("e", "Some parameters are invalid.")
        self.progress_end.emit()
        return
    coordinates = self.get_coordinates(self.split_lines(self.get_gcode()))
    self.total_coordinates = len(coordinates)
    if self.fits_in_ws(ws_radio, ws_height, coordinates):
        ws_coordinates = self.z_bias(coordinates, float(height), float(ws_height))
        try:
            self.inv_kin_problem(ws_coordinates, params)
            self.positions_list = self.flatten(self.positions_list)
        except ValueError:
            self.set_message_params('e', 'Se produjo un error',
                                    'Se ha producido un error de calculo. '
                                    '@Por favor revise que los datos de '
                                    'impresora esten correctos.')
            self.progress_end.emit()
            return
        self.set_message_params('i', 'Operacion finalizada',
                                'Finalizo la generacion de instrucciones. '
                                'Puede enviarlas a la impresora una vez '
                                'realizada la conexion.')
        self.progress_end.emit()
```

Fig. 4.1 Función ejecutada al presionar el botón generar instrucciones.

Una vez que las instrucciones son creadas y almacenadas en un listado, quedan a espera que se realice una conexión con una impresora. Cuando se logre una conexión, las instrucciones se tienen que enviar como un listado al PLC. Esto se explica en detalle en el siguiente capítulo.

5. Desarrollo de la comunicación entre el software y PLC

5.1. Herramientas utilizadas

Para conseguir una conexión con un PLC se utilizó la biblioteca Pycomm3 de Python. Esta es una biblioteca orientada a la comunicación con PLCs Allen-Bradley y funciona en Python 3.6.1 en adelante. Para cumplir la comunicación se incluyen 3 drivers: CIPDriver, LogixDriver y SLCDriver. El primero incluye servicios CIP estándar, los cuales son la base para el funcionamiento de los otros 2 drivers. Para esta memoria de título se utilizó el driver LogixDriver, el cual contiene servicios de mensajería propios de los controladores ControlLogix, CompactLogix, SoftLogix y Micro800. Las funciones principales para la comunicación son las explicados en la tabla 5.1:

TABLA 5.1: Funciones de LogixDriver a usar en la comunicación con PLC

Función	Descripción
open()	Abre una conexión Ethernet/IP con el dispositivo objetivo y registra una sesión CIP.
close()	Cierra una conexión Ethernet/IP con el dispositivo objetivo.
get_plc_info()	Lee y retorna información básica sobre el controlador.
get_tag_list()	Lee y retorna un listado con la información de los tags de controlador y programas.
read()	Lee y retorna el valor del tag o tags objetivo. Los tags de programa deben especificar el programa al que pertenecen.
write()	Escribe un valor sobre un tag o tags objetivo. Los tags de programa deben especificar el programa al que pertenecen.

Con estas funciones podemos realizar todas las tareas de envío de instrucciones, monitoreo y control del proceso. Cabe hacer notar que no es recomendable abrir y cerrar la conexión muchas veces por segundo, sino más bien hacer todas las peticiones posibles abriendo una sola conexión o la comunicación puede fallar, razón por la cual se abrirá una conexión única al principio del proceso. Además, se debe tener en cuenta que si no se realizan peticiones al PLC por un largo tiempo (aproximadamente 2 minutos) la conexión será destruida por el PLC. Por último, también se debe tener en cuenta que, al escribir valores sobre tags, el límite de valores que se pueden escribir con un solo mensaje es cercano a 65000, por lo que este es el límite de instrucciones por mensaje.

5.2. Construcción de la comunicación con PLCs

El desarrollo de la comunicación se explicará por cada pestaña en la interfaz, mostrando como se logró llevar a cabo el funcionamiento de cada elemento que hace uso de la conexión con la impresora. Por último, se realiza una breve explicación de las rutinas utilizadas en el proceso de impresión.

5.2.1 *Pestaña de conexión y generación de instrucciones*

En esta pestaña es donde se inicia la comunicación con la impresora. Al presionar el botón "conectar" se llama a la función *plc_info()*:

Se abre una conexión a la impresora (si es que es inexistente) y se extraen los datos del PLC.

En caso de no poder realizar la conexión se despliega un error al usuario.

```
@pyqtSlot(result=list)
def plc_info(self) -> List[str]:
    try:
        if not self.plc.connected:
            self.plc.open()
        is_connected = self.plc.connected
        product_name = self.plc.info["product_name"]
        name = self.plc.info["name"]
        programs = ""
        for program in list(self.plc.info["programs"].keys()):
            programs += str(program)
        return [product_name, name, programs, is_connected]
    except:
        self.set_message_params('e', 'Se produjo un error',
                                'Se ha producido un error de conexion. '
                                'Por favor revise que la direccion IP sea
                                'la correcta.')
        self.progress_end.emit()
        return ["----------", "----------", "----------", False]
```

Fig. 5.1 Función plc_info()

Esta función abre la conexión que se utilizará a lo largo de todo el proceso, para luego obteners el nombre del dispositivo, el del controlador y los programas que se encuentran dentro de él. Esta información se muestra en la interfaz para informar al usuario a cuál dispositivo se conectó. Luego se llama a la función *plc_tag_list()* (fig. 5.2).

Se comprueba el estado de la conexión con la impresora y se extrae el listado de tags disponibles en el PLC.

```
@pyqtSlot(str, result=list)
def plc_tag_list(self, ip) -> List[str]:
    """Gets the list of program tags and returns a list containing
       the names of all tags with less than 100 elements"""

    if not self.plc.connected::
        self.plc.open()
    tag_list = plc.get_tag_list('Program:MainProgram').copy()
    tag_name_list = self.get_names(self.get_tags_info(tag_list))
    return tag_name_list
```

Fig. 5.2 Función plc_tag_list()

La función anterior, obtiene el listado de tags del programa "MainProgram" con la función *get_tag_list()* y ordena la información recibida para generar un listado con los nombres de todos los tags que tengan menos de 100 elementos. Este límite es necesario para que la interfaz no se sobrecargue tratando de manejar cientos de elementos. El listado con los nombres es luego retornado a la interfaz y cargado en los elementos *ComboBox* de la pestaña de monitoreo, para que el usuario los pueda seleccionar y monitorear.

Por último, una vez que se haya generado el listado de instrucciones, se activará el botón para realizar el envío de ellas al controlador con el que se realizó la conexión. Al presionar este botón se llamará a la función *send_instructions()*, la cual usa la función *write()* para escribir el listado de instrucciones en el tag encargado de almacenar las instrucciones.

Se comprueba la conexión a la impresora y se revisa si está imprimiendo.

Si no se encuentra imprimiendo una pieza, se envían las instrucciones a la impresora. En caso contrario se despliega un mensaje anunciando la cancelación del envío de instrucciones.

```
@pyqtSlot()
def send_instructions(self):
    n_instructions = len(self.positions_list)
    if not self.plc.connected:
        self.plc.open()
    is_printing = self.plc.read('Program:MainProgram.sw_beginapp').value
    if self.plc.connected and not is_printing:
        self.plc.write('Program:MainProgram.Matriz_L{'+str(n_instructions)+'}',
                       self.positions_list)
        self.set_message_params('i', 'Operacion finalizada',
                                'Las instrucciones fueron enviadas a la '
                                'impresora. Puede monitorear el proceso en '
                                'las pantallas adyacentes.')
    else:
        self.set_message_params('e', 'Operacion cancelada',
                                'La impresora se encuentra trabajando. '
                                'Detenga la impresion en la pestaña "Control" '
                                'o espere a que finalice.')
    self.progress_end.emit()
```

Fig. 5.3 Función send_instructions()

5.2.2 *Pestaña de monitoreo*

En esta pestaña existen 4 elementos que necesitan comunicarse con el controlador: los componentes *ComboBox*, las cajas de texto para ver y sobrescribir los valores de los tags monitoreados, los gráficos presentes y la barra de progreso. En la subsección anterior se explicó el funcionamiento de los componentes *ComboBox* al iniciar la conexión a la impresora, por lo que no se hablará de ellos aquí.

Para graficar los valores de los tags se utiliza el componente *Timer* introducido en el tercer capítulo (fig. 3.8). Cada 1 segundo se revisa cuáles de los elementos *ComboBox* se encuentran activos y luego actualizan los gráficos con los tags seleccionados llamando a la función *update_series()*.

Se comprueba el estado de conexión con la impresora y se extraen los valores de los tags seleccionados. Esto se ejecuta continuamente para actualizar los gráficos.

En caso de perderse la conexión se despliega un mensaje indicando el intento de reconexión y se utilizan los últimos valores obtenidos para actualizar los gráficos.

```
@pyqtSlot(list, list,  int, int, result=list)
def update_series(self, tag_list, tag_spot, upper_len, lower_len) -> List[float]:
    """Updates both 2D plots with the values of the given tag list. Returns the
    read values to be displayed to the user"""

    try:
        tag_names = self.extract_names(tag_list)
        n_tags = len(tag_names)
        if not self.plc.connected:
            self.plc.open()
        tag_read = self.plc.read(*tag_names)
        values = self.extract_values(tag_read, n_tags)
        self.saved_values = values
        if self.loading_is_open:
            self.connection_achieved.emit()
            self.loading_is_open = False
    except:
        if not self.loading_is_open:
            self.set_message_params('r', 'Se produjo un error',
                                    'Se ha perdido la conexion con la impresora.
                                    '\nReconectando...')
            self.progress_end.emit()
        self.loading_is_open = True
        values = self.saved_values
    self.update_plots(values, tag_spot, upper_len, lower_len)
    return values
```

Fig. 5.4 Función update_series()

Esta función se encarga de leer los valores de los tags seleccionados en la interfaz, desde el controlador asociado a la dirección IP previamente ingresada. Para leer los valores de los tags de programa se debe utilizar el nombre del tag y el nombre del programa al que pertenece. En el caso de valores de los elementos internos de un arreglo o matriz, también se debe especificar la posición del valor a leer. Es por esta razón, que al listado de tags se le debe extraer primero el nombre exacto del elemento a leer con la función *extract_names()*. Una vez que se tienen los nombres de cada tag en el formato correcto, se realiza una petición de lectura para leer todos los valores simultáneamente. Luego se llama a la función *update_plots()* para actualizar los gráficos con los valores leídos. Finalmente, se retorna un listado con los valores de los tags solicitados, para que se desplieguen en la interfaz y los pueda leer el usuario.

En el caso de las cajas de texto, para cambiar los valores de las variables se realiza un mensaje al PLC cada vez que el usuario presione “enter” luego de ingresar un valor. Esta acción gatilla un llamado a la función *write_value()*, la cual toma el tag que se encuentra seleccionado en ese momento por el componente *ComboBox* correspondiente y el valor ingresado para su escritura.

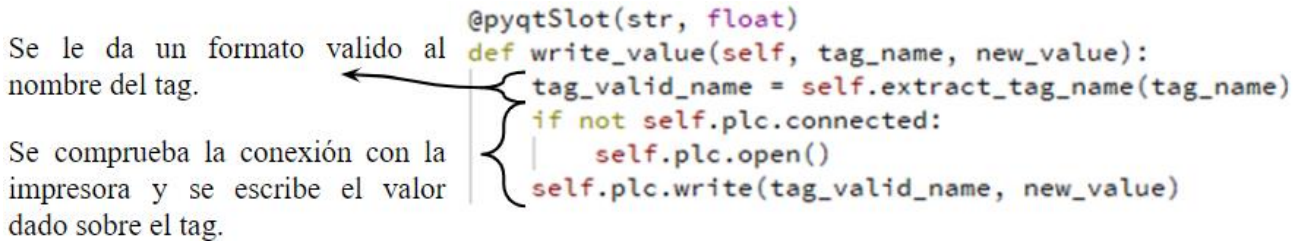


Fig. 5.5 Función write_value()

La barra que muestra el progreso de impresión funciona bajo el mismo principio de lectura. Cada unos pocos segundos se lee en qué paso se encuentra el proceso con respecto al número total de instrucciones. De acuerdo con esto se va modificando el tamaño de la barra.

5.2.3 *Pestaña de control de impresora*

Esta pestaña contiene cajas de texto para modificar las ganancias del lazo de control, además de botones que permiten mover el efector de la impresora a una coordenada especifica. Las flechas modifican la coordenada mostrada en las cajas de texto, pero la petición de movimiento no se realiza hasta que se presiona el botón “Realizar movimiento”. Este botón toma la coordenada y velocidad de efector indicada en pantalla y llama a la función *move_to()*, la que se encarga de traducir dicha coordenada a las posiciones que deben tomar los motores para luego enviarlas al PLC e iniciar el desplazamiento, siempre que no exista un movimiento en proceso.

Se calculan las posiciones de los motores con las coordenadas dadas y se comprueba si la impresora se está moviendo.

Si la impresora se encuentra en reposo el movimiento se lleva a cabo, de lo contrario se despliega un mensaje indicando que la impresora se encuentra ocupada.

```
@pyqtSlot(float, float, float)
def move_to(self, x_pos, y_pos, z_pos):
    x = x_pos
    y = y_pos
    z = z_pos - self.params[3] + self.params[5]
    servo_pos = self.inv_kin_problem([[x, y, z, 6000]], self.params)
    is_printing = self.plc.read('Program:MainProgram.sw_beginapp').value
    is_homing = self.plc.read('sw_startposition').value
    if self.plc.connected and not is_printing and not is_homing:
        self.plc.write('Program:MainProgram.coor_move_array{3}', self.flatten(servo_pos))
        self.plc.write('Program:MainProgram.sw_coor_move', 1)
    else:
        self.set_message_params('e', 'Operacion cancelada',
                                'La impresora se encuentra trabajando. '
                                'Detenga la impresion en la pestaña "Control" '
                                'o espere a que finalice.')
        self.progress_end.emit()
```

Fig. 5.6 Función move_to()

En la mitad derecha de la pestaña están los botones encargados del homing de la impresora y de detener la impresión en proceso. Ambos botones utilizan la función *write()* para activar los interruptores e iniciar rutinas que llevan a cabo las tareas correspondientes. En el caso de desear detener el proceso de impresión, se necesitará una doble confirmación y se corroborará que exista un proceso que detener antes de enviar la orden.

Bajo ellos, se encuentra un elemento *spinbox* para modificar la velocidad de impresión y a las ganancias de lazo de los controladores de cada motor. Para modificar la velocidad, se debe seleccionar el *spinbox* y utilizar las flechas del teclado para cambiar el porcentaje aplicado. Cada vez que el valor cambia se envía una señal al PLC que modifica la velocidad. Por otro lado, para sintonizar los lazos se debe ingresar un valor para la ganancia que se quiere modificar y presionar “enter”. Hacer esto, envía el mensaje al PLC que sobrescribe el parámetro correspondiente.

5.2.4 *Rutinas utilizadas para la comunicación interfaz – PLC*

Las tareas y órdenes que realiza la interfaz a los motores están directamente enlazadas a las rutinas de control. Estas rutinas fueron primeramente desarrolladas por ex memoristas en [2] y luego, se trabajó sobre ellas en el desarrollo de esta memoria de título. En la tabla 5.2 se realiza una breve explicación de cada rutina, aclarando las modificaciones realizadas en caso de existir (marcadas con asterisco).

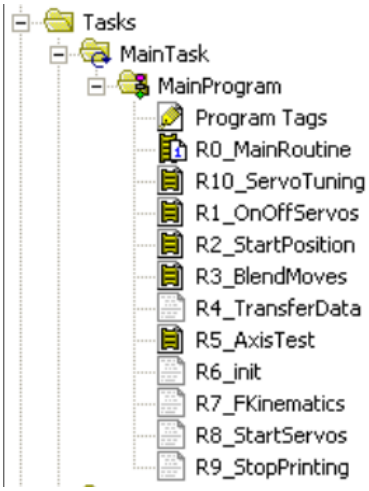


Fig. 5.7 Rutinas que componen las primitivas de control en RSLogix 5000

TABLA 5.2: Rutinas que componen las primitivas de control de la impresora

Función	Descripción
R0_MainRoutine	Ladder que contiene el resto de las rutinas para su libre ejecución.
R1_OnOffServos*	Se encarga de activar o desactivar los servomotores. Se agregó la capacidad de resetear los motores en caso de falla.
R2_StartPosition*	Maneja la lógica utilizada para realizar el homing de la impresora. Se cambió el método utilizado por uno que ocupa instrucciones *Motion Axis Home* (MAH).
R3_BlendMoves*	Ladder responsable de iniciar el proceso de impresión y ejecutar en orden las instrucciones de movimiento. Se agregó instrucción que permite realizar movimientos independientes.
R4_TransferData*	Código estructurado encargado de poner en cola las instrucciones de impresión. Se agregó lógica para seguimiento de progreso y pausa/detención de proceso.
R5_AxisTest	Permite realizar movimientos de prueba en cada eje de forma independiente con el objetivo de depurar del programa. No es utilizado en este proyecto.
R6_Init*	Inicializa todas las variables de programa. Se agregaron algunas variables más para el seguimiento del progreso de impresión.
R7_FKinematics	Calcula la coordenada en que se encuentra el efector a partir de las posiciones reales de los motores utilizando la solución del problema directo de posición.
R8_StartServos	Rutina nueva. Activa todos los motores al ser ejecutada.
R9_StopPrinting	Rutina nueva. Detiene la impresión en proceso y reinicia el contador de instrucciones. Se pierde todo el avance impreso.
R10_ServoTuning	Rutina nueva. Ladder que contiene la lógica responsable de modificar las ganancias de los lazos de control.

Una de las rutinas más importantes dentro del proceso es la rutina 10. Ella permite la sintonización de los lazos de control de los motores. Para realizar esta tarea, se utilizan los bloques Set System Value (SSV) y Get System Value (GSV), los que son capaces de sobrescribir y leer valores de sistema. Normalmente, al crear un eje de movimiento en RSLogix 5000, los valores ingresados como ganancias de lazo son modificables solo offline, lo que convierte el proceso de sintonización en algo tedioso.

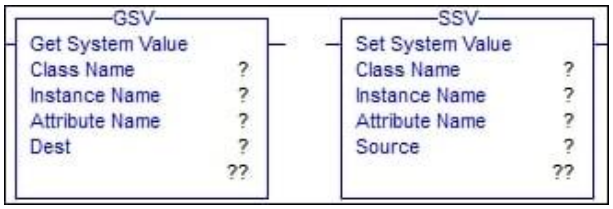


Fig. 5.8 Bloques SSV y GSV

Afortunadamente los bloques SSV y GSV permiten leer y alterar estos valores de sistema en estado online. Estos bloques reciben el nombre de la clase, instancia, atributo y tag de origen/destino. Para modificar las ganancias de lazo, la clase es *Axis*, la instancia es el eje de movimiento a manipular, el atributo es el nombre del parámetro (ganancia) a modificar y los tag de origen/destino son tags de tipo reales. Los bloques GSV se utilizan para poder extraer los valores de las ganancias y poder mostrárselos al usuario, mientras que los SSV se utilizan para sobrescribir las ganancias de los tres motores simultáneamente. La figura 5.10 muestra un extracto de la rutina 10, donde se puede observar los dos tipos de "escalones" que componen la lógica principal:

Escalón 1: Cuando el usuario active el switch *sw_posItgGain,* el valor del tag *SV_positggain* sobreescribirá la ganancia integrativa de posicion de los tres motores.

Escalón 2: Los valores de las 6 ganancias de lazo se leen y almacenan en sus 6 tags correspondientes, los que luego el software diseñado utiliza para mostrarselos al usuario.

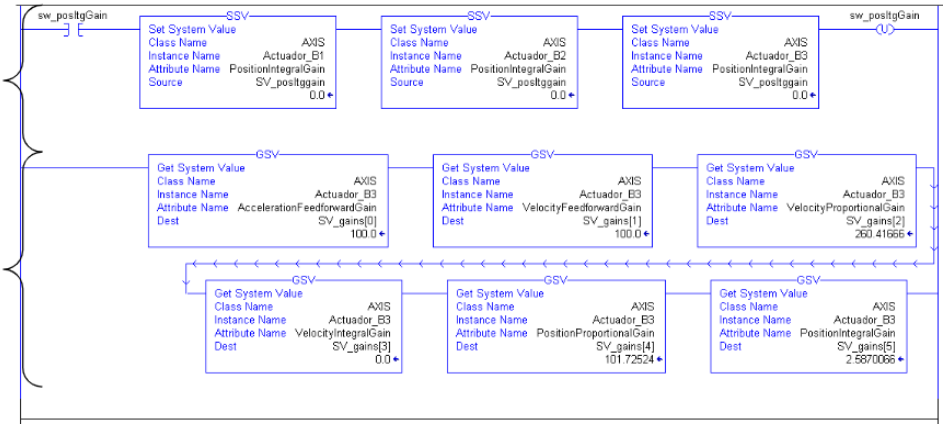


Fig. 5.9 Extracto de rutina R10_ServoTuning

5.2.5 *Diagrama de flujo final de la interfaz*

El uso de la interfaz se puede resumir con el siguiente diagrama de flujo:

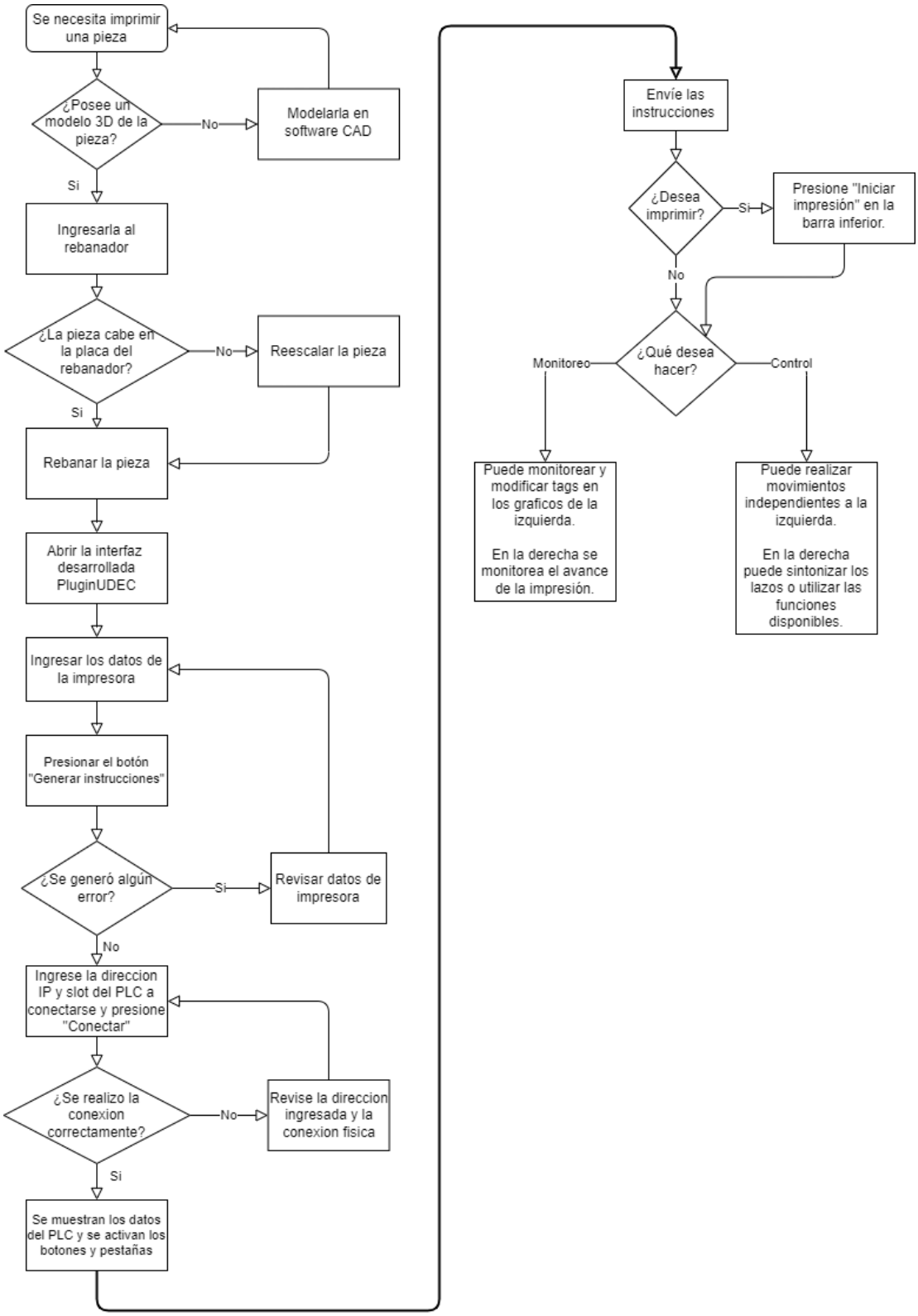


Fig. 5.10 Diagrama de flujo de la interfaz desarrollada

6. Implementación y puesta en marcha

La implementación y puesta en marcha de esta interfaz se realizó utilizando Ultimaker Cura versión 5.2.1 (versión productiva) en Windows 10 y macOS Big Sur, mientras que en Ubuntu 22.04 se usó la versión 5.3.0-alpha (versión de desarrollo). Los sistemas operativos se diferencian por las velocidades de respuesta y rendimiento general de la aplicación.

Dentro de este capítulo se detallarán los pasos necesarios para la implementación de la nueva interfaz en cualquier computador, para luego pasar a las experiencias y puesta en marcha llevada a cabo en el laboratorio de control de procesos de la facultad de ingeniería.

6.1. Implementación del plugin en Ultimaker Cura

Ultimaker Cura acepta nuevos plugins con relativa facilidad, solo basta con agregar una carpeta que contenga el nuevo plugin dentro del directorio adecuado. El proceso de creación e instalación de plugins se encuentra documentado en la página del programa [25]. Mientras el plugin desarrollado ocupe las herramientas y bibliotecas incluidas en la instalación de Ultimaker Cura, no existirán problemas de compatibilidad.

En el caso de la interfaz desarrollada en esta memoria de título se utilizaron bibliotecas externas a U.C. necesarias para el cumplimiento de los objetivos. Matplotlib se utilizó para crear los gráficos que ayudan a monitorear el proceso y Pycomm3 es vital para la comunicación entre la interfaz y la impresora. Ambas bibliotecas incidieron en la instalación del plugin, la que varía entre cada sistema operativo.

6.1.1 *Instalación en Windows 10*

Para instalar la interfaz se deben copiar los contenidos del plugin [35] en el directorio de "plugins" de U.C. Este directorio puede ser encontrado bajo el menú de ayuda del programa seleccionando la opción "Mostrar carpeta de configuración".

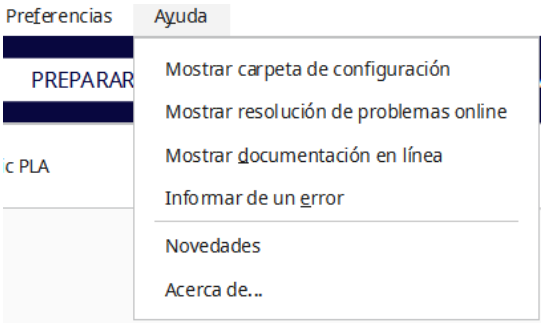


Fig. 6.1 Opción para encontrar el directorio de configuración

Dentro de los contenidos del plugin se encuentran los archivos de las bibliotecas Pycomm3 y Matplotlib. Ambas son externas a U.C., pero Matplotlib requiere ser copiada dentro del directorio de instalación del programa rebanador para ser detectada. Este directorio suele encontrarse en <C:\archivos de programa\Ultimaker Cura *versión*> y contiene las bibliotecas instaladas con U.C., como PyQt6 y Scipy.

Los archivos de Matplotlib incluidos con el plugin se encuentran separados por sistema operativo, ya que cada uno ocupa una configuración diferente. En el caso de Windows 10 se deben copiar los archivos de la carpeta correspondiente y pegarlos en el directorio de instalación de U.C. Estos pasos están descritos en el archivo "Readme.txt" incluido con el plugin. Siguiendo las instrucciones anteriores, la interfaz queda correctamente instalada en Windows 10.

6.1.2 *Instalación en macOS*

El proceso en macOS es similar a Windows 10. Dentro de la carpeta del plugin, se encuentran los archivos de Matplotlib correspondientes a macOS y se deben copiar al directorio de instalación de U.C. Para llegar a él se debe ir a la ventana de aplicaciones, hacer clic izquierdo sobre Ultimaker Cura y seleccionar "mostrar contenido del paquete". Luego entrar al directorio <./Contents/MacOS> y una vez dentro se depositan los archivos de la biblioteca.

Por último, al igual que en Windows, se deposita la carpeta con el plugin dentro de la carpeta "plugins" de la carpeta de configuración de U.C. Se puede acceder a ella con el mismo método utilizado en Windows.

6.1.3 *Instalación en Ubuntu*

Hacer funcionar la interfaz en Ubuntu requiere muchos más pasos que en el resto de los sistemas operativos. En los casos anteriores, para que Ultimaker Cura detecte a Matplotlib, se copian los archivos correspondientes en la carpeta de instalación, pero en Ubuntu no existe tal directorio. En Ubuntu el programa rebanador tiene la particularidad de estar configurado como una imagen de disco, la cual al ejecutarse genera un ambiente virtual temporal que contiene todos los archivos y bibliotecas del programa. Al ser temporal, este ambiente se crea al momento de iniciar U.C. y se elimina el cerrarlo, por lo que el método utilizado en otros sistemas operativos no funciona.

El problema descrito se presenta en la versión productiva de Ultimaker Cura, pero no en la versión para desarrollo. Esta otra versión está hecha para desarrolladores, por lo que permite tener acceso a todos los archivos de instalación, pero se tiene que construir desde el código fuente del programa [25]. El proceso de construcción está documentado y se describe paso a paso lo que se debe hacer en la página de Ultimaker Cura [36].

Una vez que se completa la construcción de U.C. desde su código fuente, se deben seguir los mismos pasos realizados en Windows y macOS. Matplotlib se debe depositar en el directorio </Cura/venv/lib/python3.10/site-packages>, mientras que el plugin se copia en la carpeta “plugin” dentro de la carpeta de configuración. De esta forma la interfaz es compatible con Ubuntu.

6.2. Equipos de laboratorio utilizados

6.2.1 *Controlador Logix 5564*

Controlador lógico programable Allen-Bradley. Pertenece a la línea de controladores ControlLogix estándar reconocidos por los números de catálogo 1756-L6X y 1756-L7X. Capaz de comunicarse con dispositivos por protocolos Ethernet/IP, ControlNet y DeviceNet, entre otros.

Fig. 6.2 Módulos instalados en chasis en el laboratorio de control de procesos

6.2.2 *Modulo interfaz ControlLogix SERCOS 1756-M08SE*

Interfaz física encargada de enlazar el controlador con los equipos servovariadores Kinetix 6000. Permite al PLC comunicarse con hasta 8 servovariadores, enviando y recibiendo información de movimiento de los ejes. Esto incluye las configuraciones de ejes, posiciones, velocidades, torques, alarmas, entre otros. Este módulo se observa en el chasis mostrado en la figura 6.2.

6.2.3 *Servovariadores multiejes Kinetix 6000*

Fig. 6.3 Módulos Kinetix 6000 en laboratorio de control de procesos

Módulos diseñados para manejar los servomotores. Compuesto por un módulo de eje integrado 2094-BC01-MP5-S y 5 módulos de eje 2094-BMP5-S conectados en serie. Aceptan retroalimentación de encoder absoluto o incremental de múltiples vueltas o una sola vuelta. Poseen tecnología safe torque-off y visualizadores de 7 segmentos para indicar estados utilizando códigos numéricos.

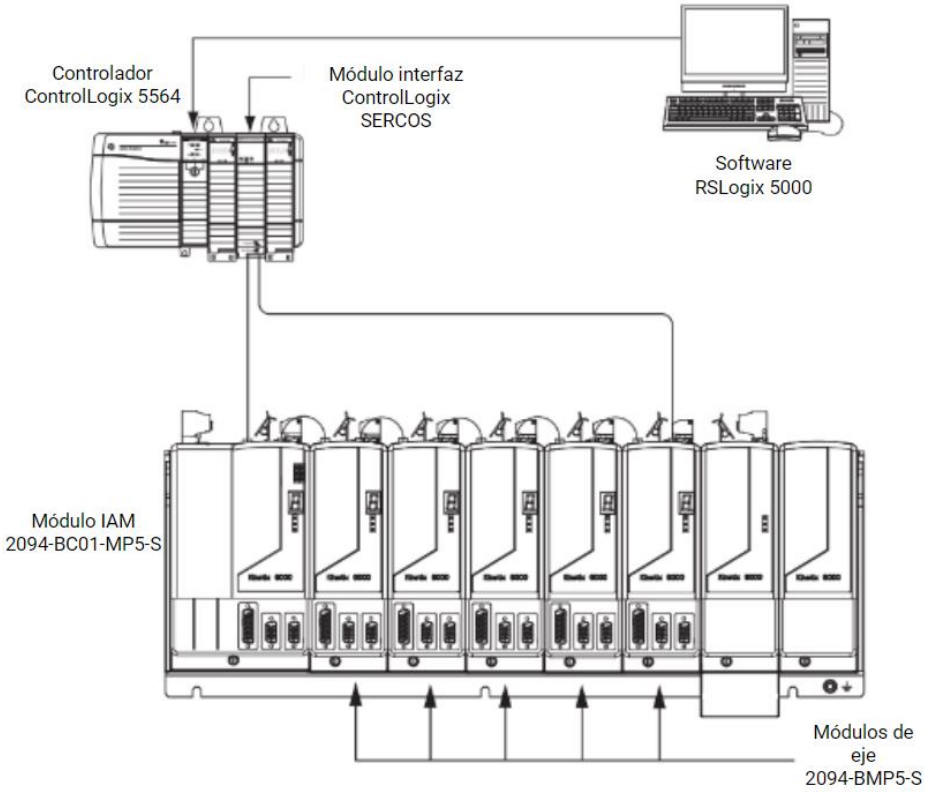


Fig. 6.4 Configuración de conexión de módulos Kinetix 6000

6.2.4 *Accionadores lineales integrados serie-MP*

Los accionadores lineales integrados son un sistema de movimiento lineal serie MPAS-B60662-V20S4A de Allen Bradley, compuesto por tres pares de actuador lineal y servomotor AC. Los actuadores tienen un carril de 660 mm de largo por el que se mueve el actuador, avanzando 20 mm por revolución del motor y con una velocidad máxima de 1124 mm/s (sin carga). El conjunto actuador-servo fue diseñado para aplicaciones de automatización industrial, siendo muy resistente y entregando alta precisión. Los servomotores incluyen sistema de freno y encoder absoluto, para múltiples vueltas y de alta resolución (128 ciclos/rev) [2].

Fig. 6.5 Accionadores lineales integrados serie-MPAS

6.3. Configuraciones de impresión en Ultimaker Cura

Las pruebas que se describirán en este capítulo se realizaron en Ubuntu 22.04, macOS big sur y Windows 10. Se propuso poner a prueba todas las funcionalidades de la interfaz al realizar un ensayo de impresión de un cubo simple. Pero antes de realizar las pruebas, se necesita crear una nueva impresora en U.C. que refleje el R.D.L., junto con una configuración de rebanado adecuada.

6.3.1 *Crear una nueva impresora en Ultimaker Cura*

Para que el proceso de rebanado entregue instrucciones precisas, se necesita indicarle al programa las especificaciones de la impresora que construirá la figura. Esto se hace en el apartado de impresoras, seleccionando las opciones “agregar impresora → Agregar una impresora fuera de red → Custom → Custom FFF printer”. Esto abrirá la ventana “Ajustes de la máquina”, en donde se insertaron las medidas del robot delta lineal diseñado en memorias anteriores [2] (figura 6.7-a).

Junto con las medidas de la máquina, también se necesita configurar los ajustes de impresión, en los que se especifican algunos detalles del método de impresión, como el grueso de los muros, la altura de las capas, densidad de relleno, velocidad de impresión y generación de soportes. Para las pruebas se configuró un grosor de 1cm para los muros y capas, con un relleno total de la figura y una velocidad de impresión de 60 mm/s.

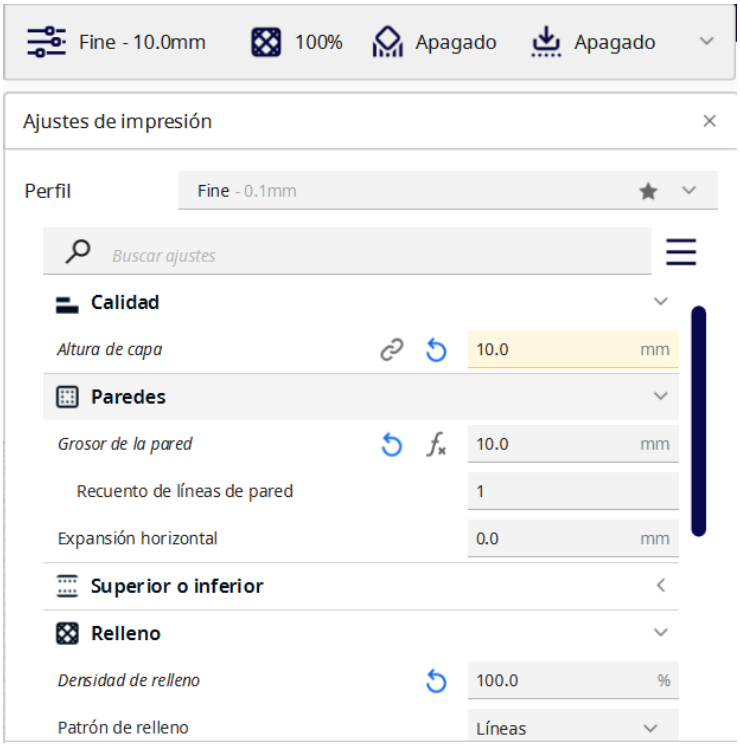
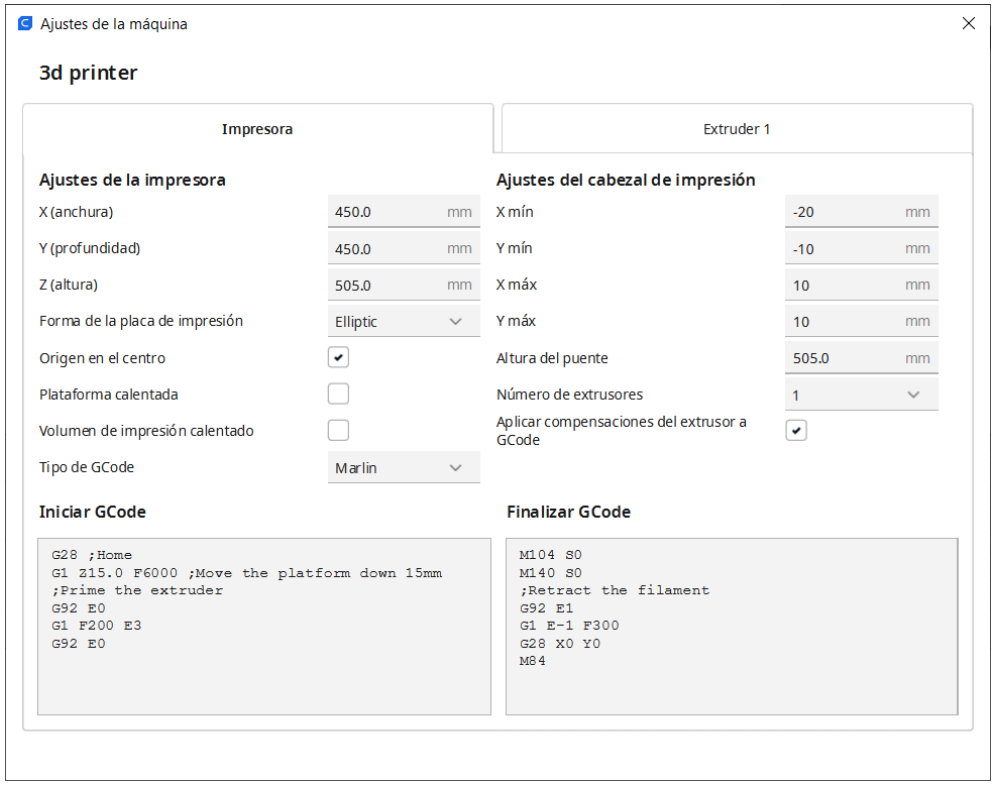
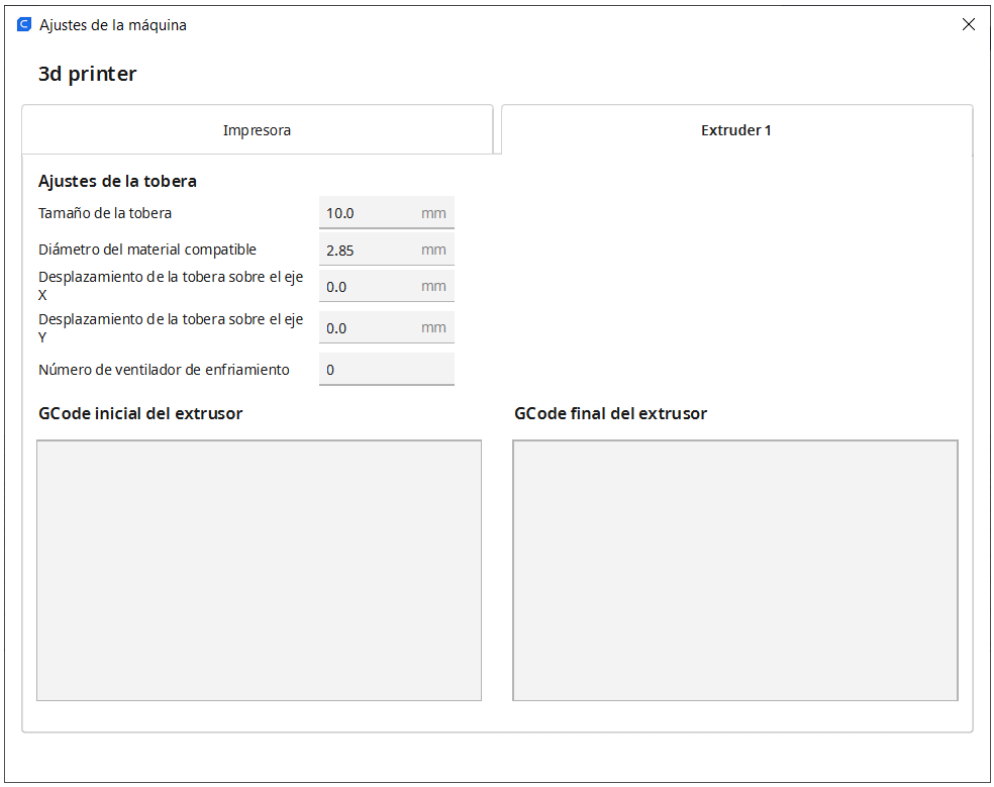


Fig. 6.6 Ajustes de impresión utilizados en pruebas



(a)



(b)

Fig. 6.7 Especificaciones de maquina utilizadas en pruebas
(a) Especificaciones de impresora**, (b)** Especificaciones de extrusor

6.4. Pruebas realizadas y resultados

El plugin se probó realizando un ensayo de impresión de un cubo de 20cm de lado. Con las configuraciones descritas anteriormente se llegó al rebanado observado en la figura 6.8. Con la figura ya rebanada se abrió la interfaz programada.

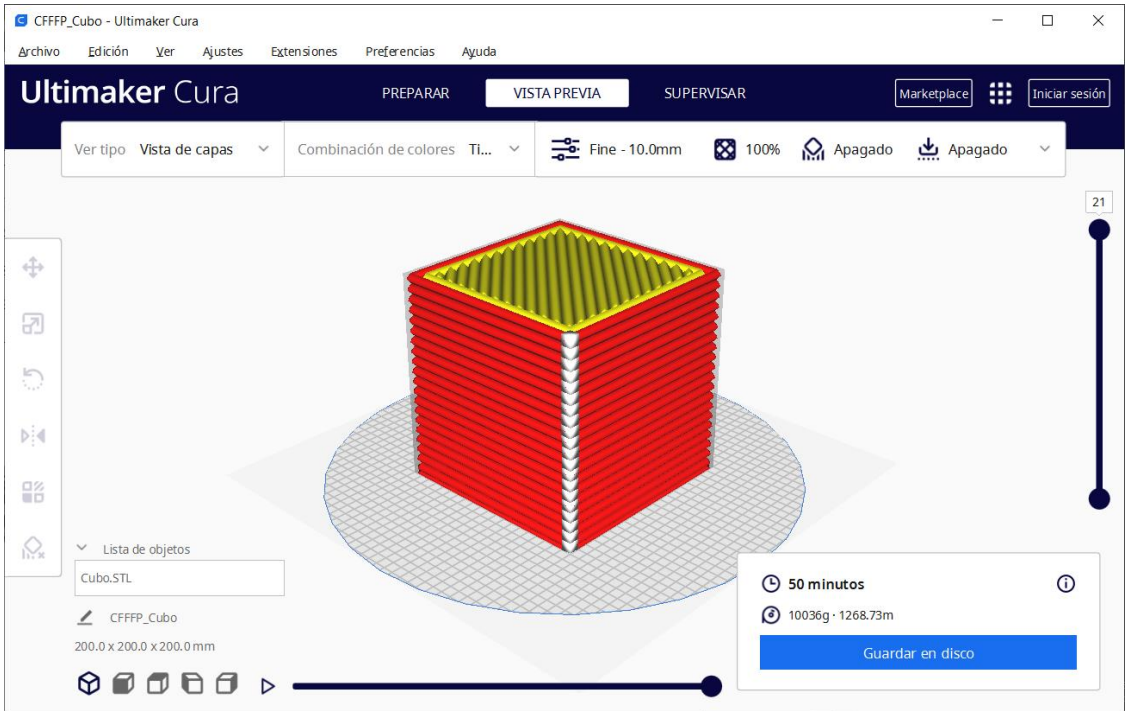


Fig. 6.8 Vista previa de las capas del cubo a imprimir

6.4.1 *Pestaña de conexión y generación de instrucciones*

Al iniciar el plugin se entra a esta pestaña. En ella el plugin informó correctamente que existe un G-code en memoria para traducir. En este estado no se puede entrar a las otras pestañas y solo el botón para generar instrucciones se encuentra disponible. El proceso de generación de instrucciones tomó cerca de 0.5 segundos en todos los sistemas operativos, siendo macOS el más rápido y Windows el más lento. Este tiempo varía dependiendo del tamaño de la figura, siendo instantáneo en figuras pequeñas y tomando poco más de 1 segundo en figuras grandes. Al finalizar el proceso se mostró un mensaje notificando el número de instrucciones generadas (2286 en este caso) y se activó el botón de conexión a la impresora.

Para conectarse al PLC se indicó su dirección IP (indicando slot) y se presionó conectar. La conexión se demoró entre 1 y 2 segundos en concretarse; los datos del PLC se mostraron en el recuadro inferior. En ese instante se habilitan las pestañas de monitoreo y control, además de los botones para

enviar las instrucciones a la impresora, comenzar la impresión y para detener los motores. Finalmente, se enviaron las instrucciones de movimiento al PLC.

6.4.2 *Pestaña de control*

Dentro de esta pestaña se inició la rutina de homing para calibrar la posición de los motores. El movimiento inicia de inmediato y, mientras está en proceso, cualquier otra orden de movimiento despliega un mensaje de error, cancelando la orden. Los desplazamiento realizados con las flechas se produjeron instantáneamente y sin problemas, siguiendo la velocidad que se indicó. La interfaz no permitió coordenadas fuera del espacio de trabajo, siempre acotando las coordenadas a los límites impuestos por la máquina.

La prueba de sintonización de los lazos se realizó cambiando las ganancias y realizando un pequeño movimiento con las flechas de la izquierda. Se comprobó con ayuda de RSLogix 5000 que las ganancias fueron modificadas correctamente, además de corroborar visualmente con las respuestas de los motores.

6.4.3 *Pestaña de monitoreo*

Por último, se inició el proceso de impresión del cubo. Inmediatamente se actualizó la barra de progreso para reflejar el número total de instrucciones por recorrer, al mismo tiempo que el grafico tridimensional comenzó a mostrar la ruta por donde se movía el efector. Por otro lado, los gráficos de dos dimensiones se utilizaron para mostrar las velocidades de los movimientos y las coordenadas XYZ por donde se desplazaba el actuador del robot (figuras 6.9 – 6.17). Esta configuración se mantuvo hasta que finalizó la impresión, momento en que se desplegó el mensaje notificando el fin del proceso (figura 6.17).

Mientras se realizaba la impresión se probó la modificación de la velocidad de movimiento del efector en la pantalla de control. Al llevar la velocidad bajo un 20%, los movimientos comenzaban a ser muy lentos, demorándose algunos hasta 30 segundos al bajar a menos del 10%. Esto generó demoras, pues la velocidad se actualiza entre movimientos y no al momento de dar la orden de cambio.

Finalmente, se presionó el botón de parada de emergencia de los motores. Al presionarlo, los motores se desactivan totalmente, deteniendo cualquier movimiento en el instante. Luego de 2 segundos los motores se vuelven a activar automáticamente y quedan en espera de órdenes de movimiento.

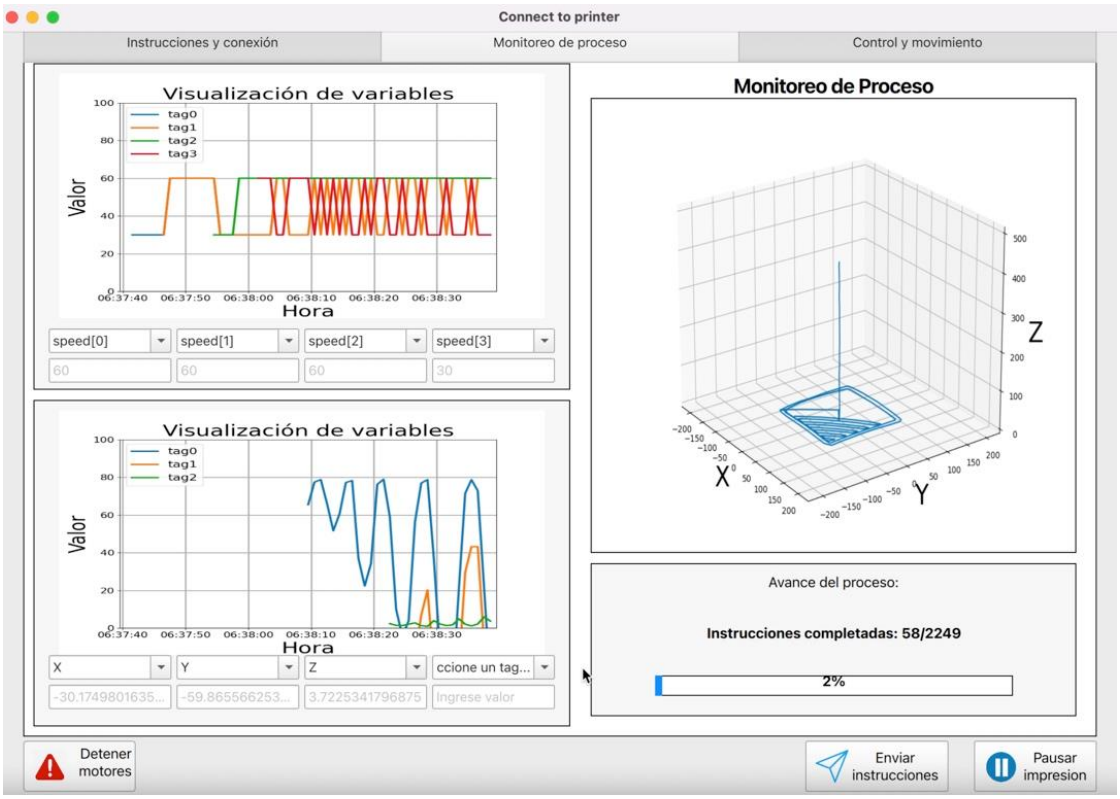


Fig. 6.9 Pestaña de monitoreo durante prueba (2% avance)

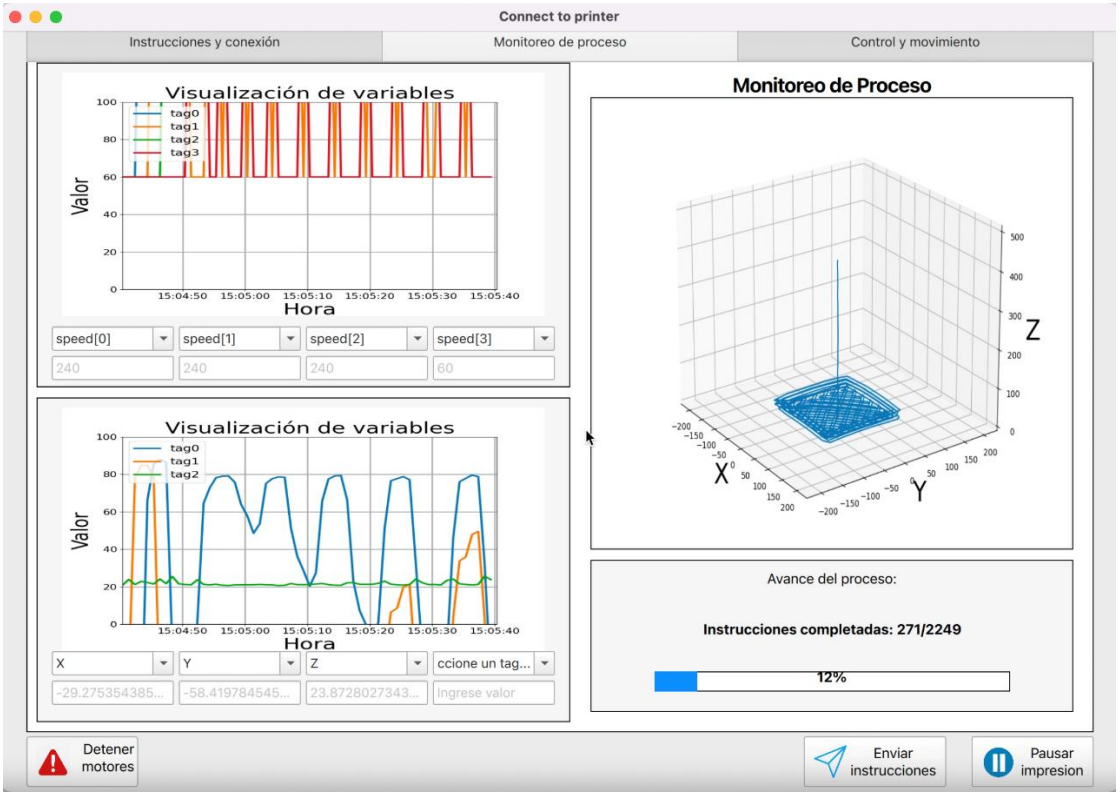


Fig. 6.10 Pestaña de monitoreo durante prueba (12% avance)

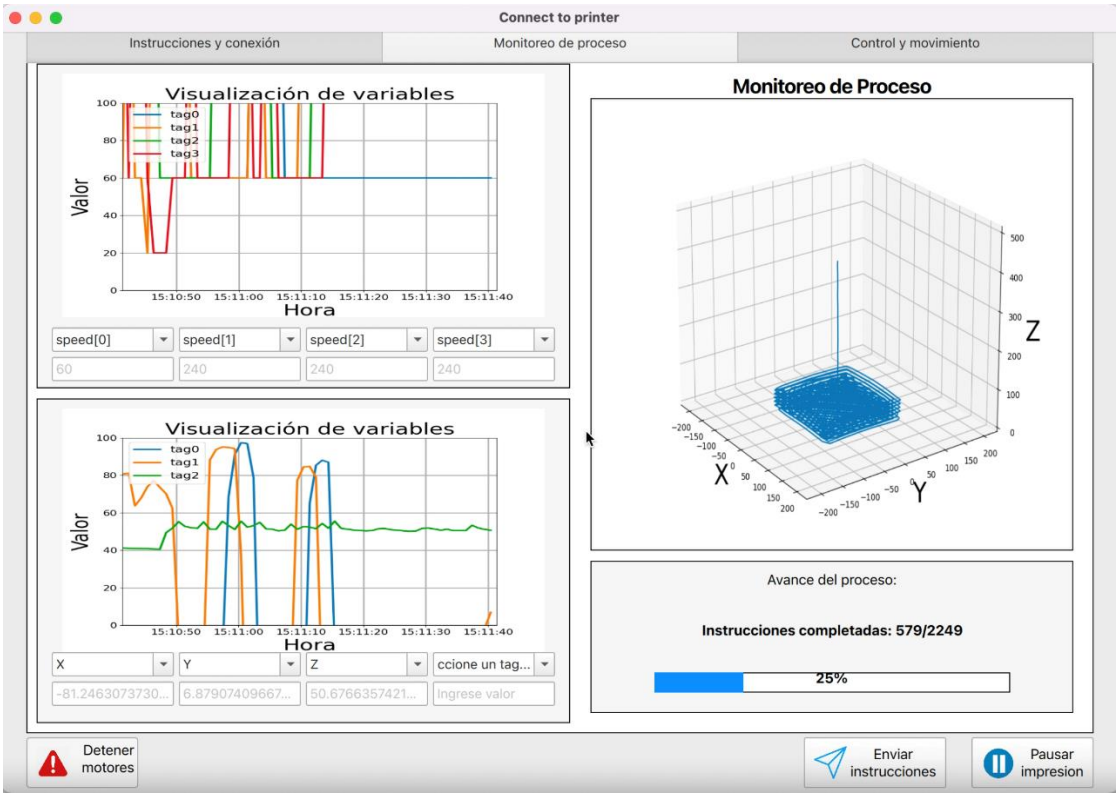


Fig. 6.11 Pestaña de monitoreo durante prueba (25% avance)

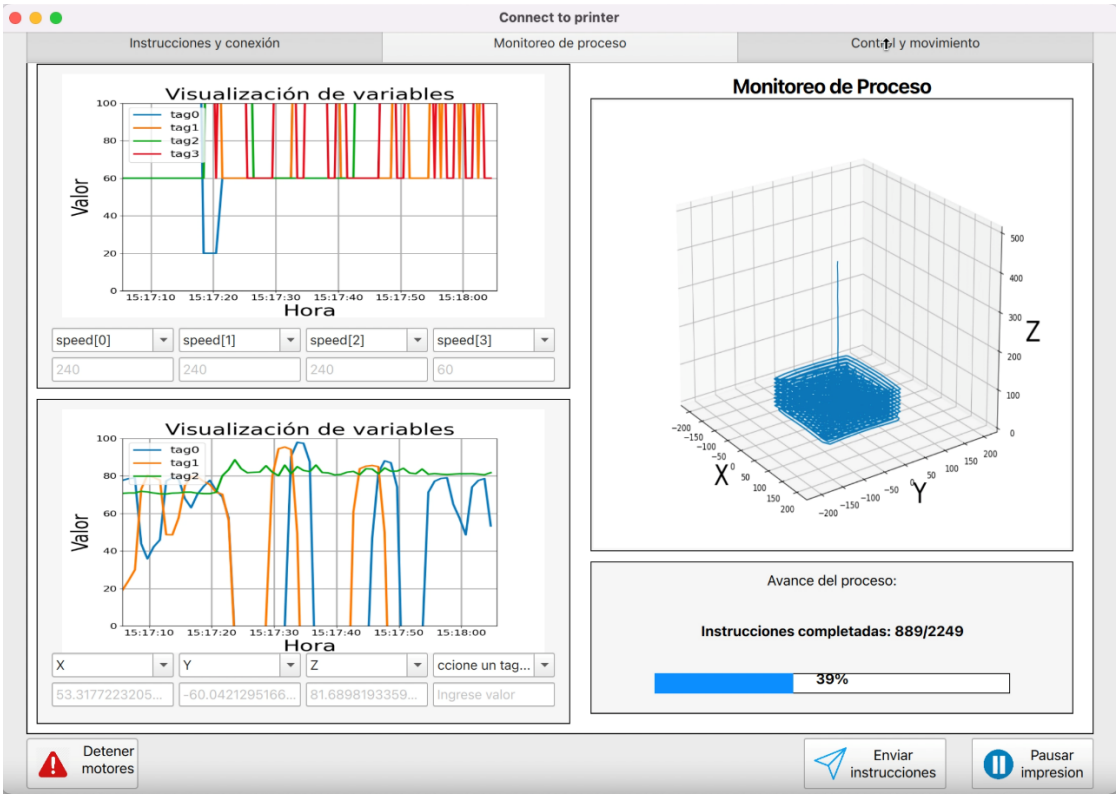


Fig. 6.12 Pestaña de monitoreo durante prueba (39% avance)

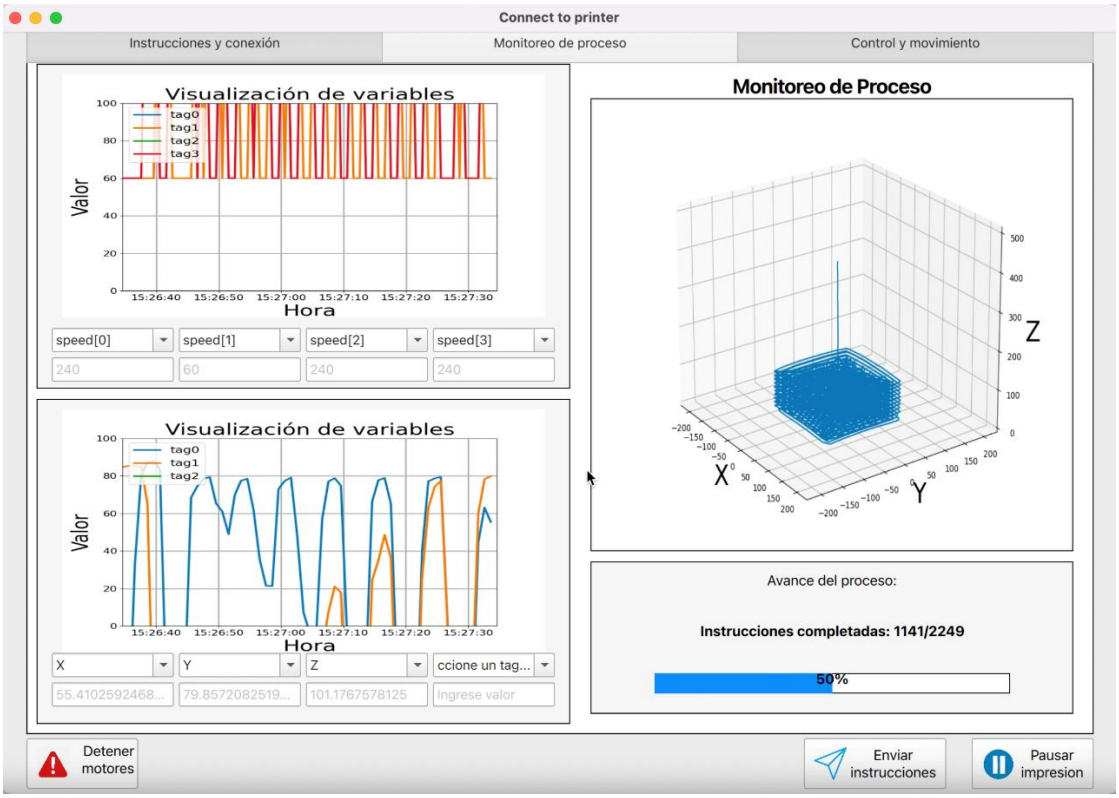


Fig. 6.13 Pestaña de monitoreo durante prueba (50% avance)

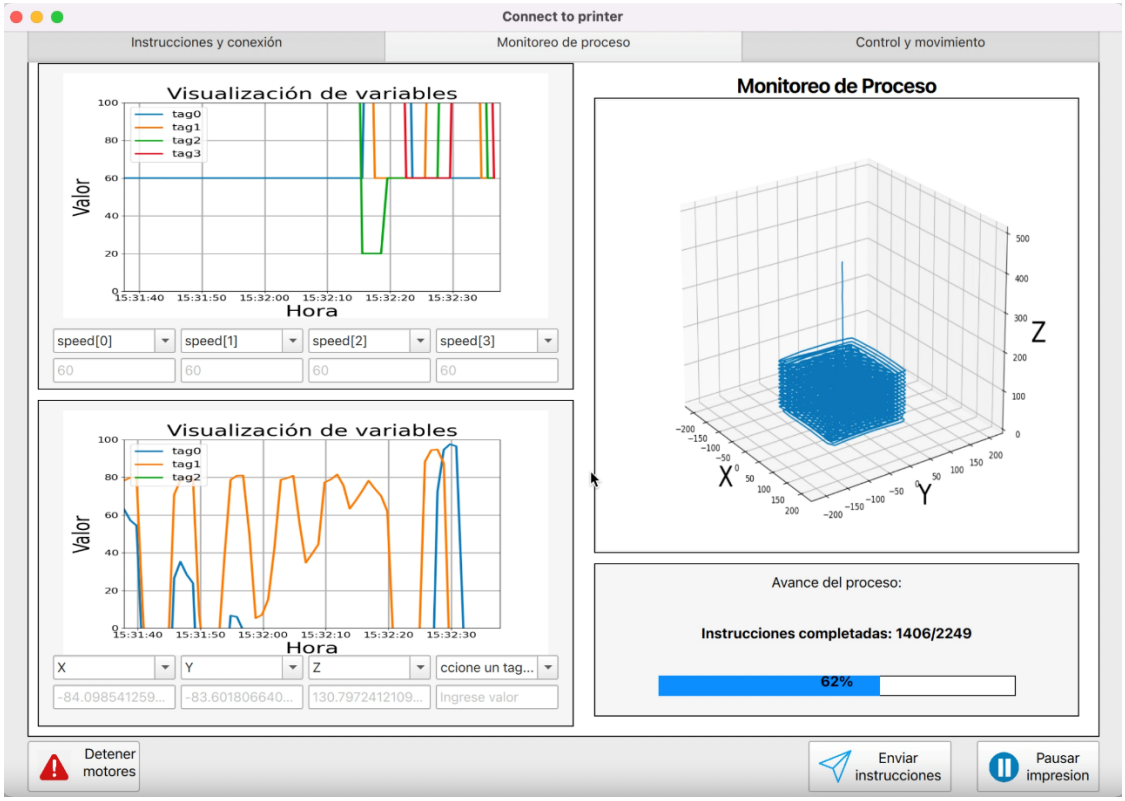


Fig. 6.14 Pestaña de monitoreo durante prueba (62% avance)

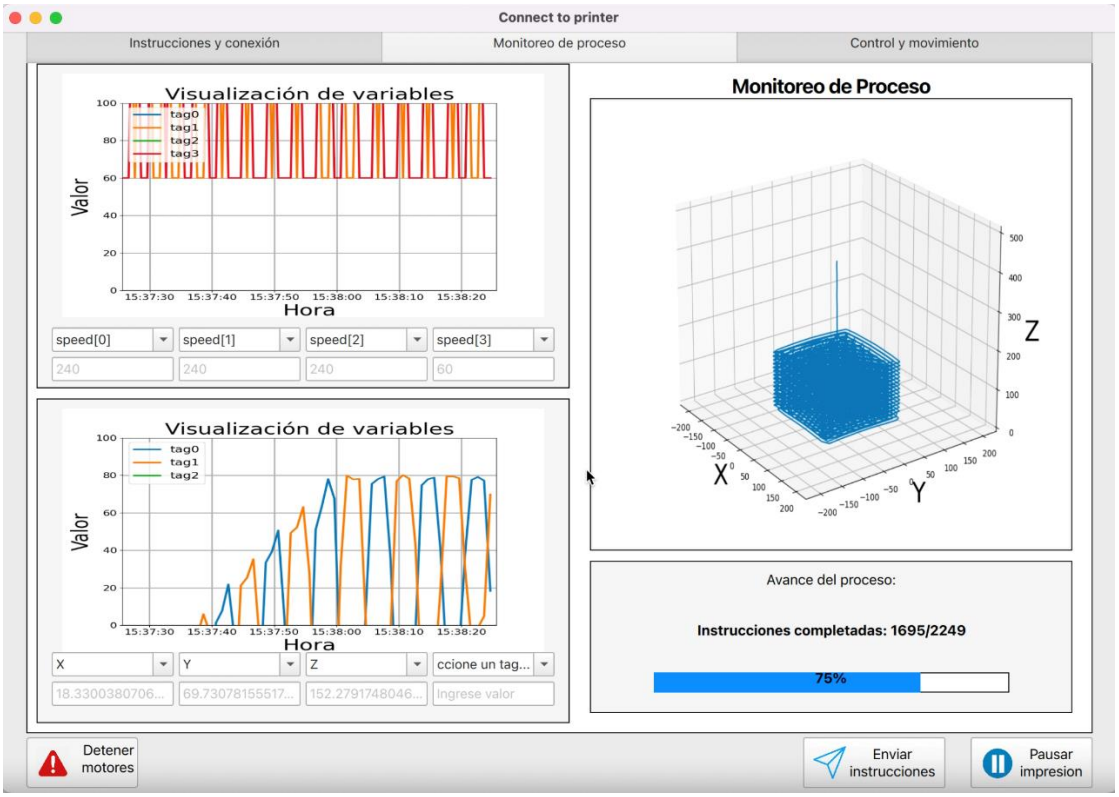


Fig. 6.15 Pestaña de monitoreo durante prueba (75% avance)

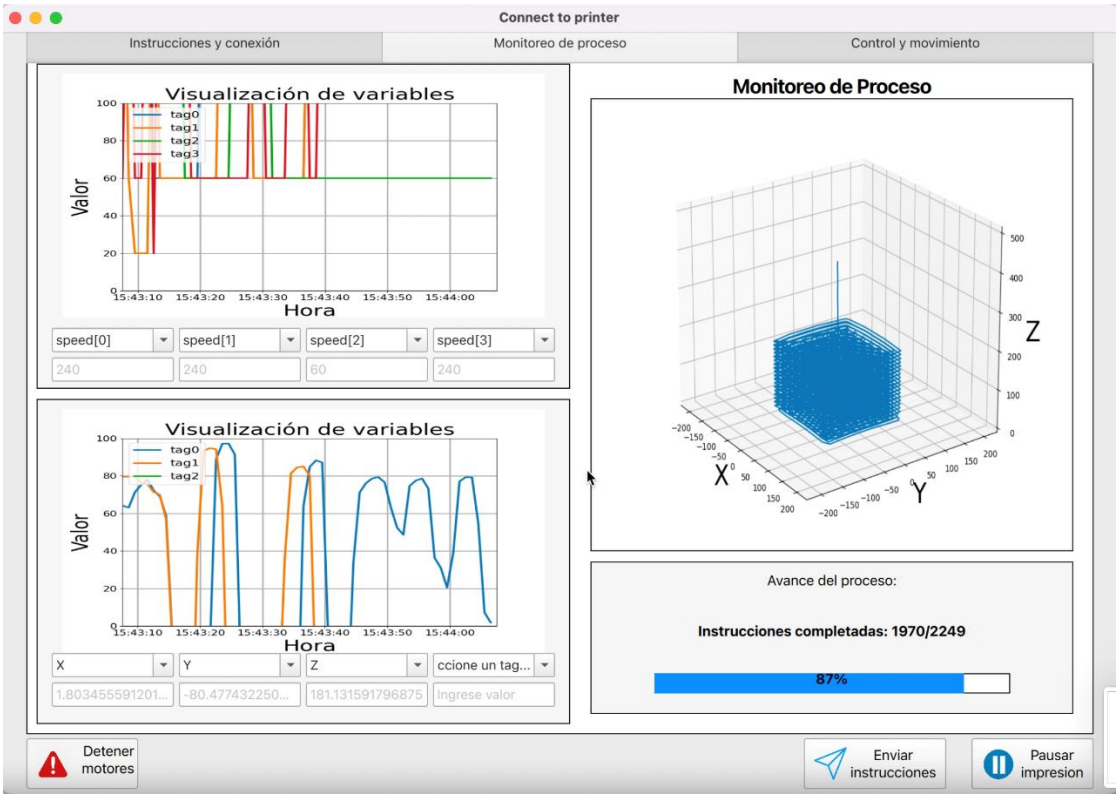


Fig. 6.16 Pestaña de monitoreo durante prueba (87% avance)

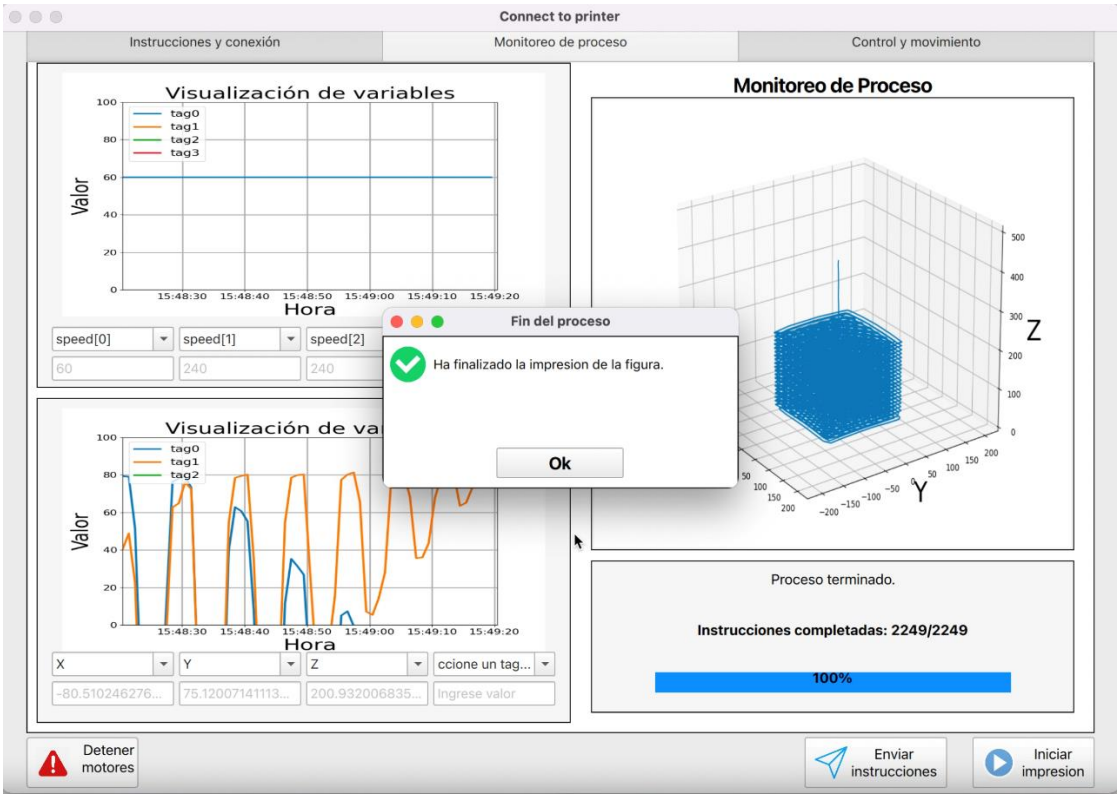


Fig. 6.17 Pestaña de monitoreo durante prueba (100% avance)

7. Discusión de resultados, conclusiones y trabajo futuro

7.1. Discusión y conclusiones

El tiempo que toma desempeñar las tareas varía entre cada sistema operativo, mostrándose siempre una tendencia a ser más lento en Windows. Es visible que, en el caso de la generación de instrucciones, estos tiempos dependen de la figura que se esté trabajando (tamaño, forma, complejidad), pero las diferencias en las pruebas demostraron ser muy pequeñas entre los sistemas operativos como para escoger uno sobre otro, nunca superando 0.5 segundos de diferencia. Evidentemente, dichas diferencias se pueden acentuar al utilizar computadores de menor capacidad a los utilizados en las pruebas.

Con respecto a los resultados obtenidos en el proceso de impresión, es pertinente hablar de la situación que acontece al llevar la velocidad a valores muy bajos. Como los movimientos de los motores son efectivamente órdenes de desplazamiento de un punto “A” a un punto “B” con una velocidad determinada, si se cambia la velocidad en medio del movimiento no se ve respuesta hasta que se inicie una nueva orden. Este comportamiento se vuelve un problema al llegar a velocidades muy bajas (menor a 20%) y es inherente del programa cargado al PLC y de los bloques disponibles para movimientos de motores. Buscar una solución a este comportamiento de los motores es una oportunidad de perfeccionar el funcionamiento del robot y puede ser parte de investigaciones futuras.

A su vez, el FAT realizado también evidenció el cumplimiento de cada objetivo impuesto al comienzo del proyecto. Se lograron integrar los procesos de generación de instrucciones de movimiento, comunicación con la impresora y su monitoreo en el programa Ultimaker Cura a través de una interfaz agregada como extensión de este. De misma forma, esta interfaz contiene guías visuales y textuales que la convierten en un software auto explicativo y amistoso a nuevos usuarios. No menos importante es la capacidad de funcionar en Windows 10, MacOS y Ubuntu 22.04, pues ser multiplataforma facilita el acceso a su uso.

Este proyecto fue evolucionando a lo largo de su desarrollo, obteniendo un producto final que superó lo diseñado en los bosquejos previos. Como se utilizaron las ecuaciones solución de un RDL, con este plugin se pueden generar instrucciones para impresoras delta de cualquier tamaño y luego monitorear y controlar el proceso de impresión, siempre y cuando se utilice un PLC Allen-Bradley que contenga el programa utilizado en esta memoria de título. Esto demuestra que, si bien el software creado cumple plenamente los objetivos propuestos a un comienzo, este proyecto es optimizable y escalable.

7.2. Trabajo Futuro

El trabajo desarrollado demostró la escalabilidad de este proyecto, abriendo al menos las siguientes líneas de investigación y desarrollo:

- Este software está diseñado solo para PLCs Allen-Bradley. Si se investigan las herramientas presentes en Pycomm3 y otras bibliotecas disponibles en internet [21] se puede crear un software universal para el control de impresoras delta industriales que sea capaz de comunicarse con cualquier controlador del mercado.
- Siguiendo el razonamiento anterior, este plugin funciona solo en impresoras tipo delta. Realizando una investigación que cubra otros tipos de robots (brazos mecánicos, por ejemplo) se puede expandir el concepto desarrollado en esta memoria de título a otros tipos de máquinas CNC.
- Las bibliotecas que permiten la comunicación con PLCs merecen ser estudiadas para conocer los límites de ellas. El software desarrollado en esta memoria representa un ejemplo de lo que se puede lograr con el correcto uso de una de estas bibliotecas y perfectamente se pueden utilizar con otros objetivos. Por ejemplo, se pueden crear pequeños scripts que se comuniquen con los PLC a cargo de un proceso para leer datos y guardarlos en hojas históricas, las cuales al mismo tiempo pueden ser respaldadas en nubes.

Referencias

[1] Matias L. Álvarez, "*Diseño Interfaz Gráfica Para Control de Impresora 3D industrial*", Memoria de Título, Ingeniero Civil Electrónico, enero 2017, Departamento de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de Concepción.

[2] Jorge A. Hernández, "*Diseño e implementación de primitivas de control para un Robot Delta Lineal*", Memoria de Título, Ingeniero Civil Electrónico, agosto 2017, Departamento de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de Concepción.

[3] C. Ye, N. Chen, L. Chen and C. Jiang, "A Variable-Scale Modular 3D Printing Robot of Building Interior Wall," 2018 IEEE International Conference on Mechatronics and Automation (ICMA), pags. 1818-1822, 2018, doi: 10.1109/ICMA.2018.8484433.

[4] *K. Mehta, R. Joshi, H. M. Jadav, S. V. Kulkarni, B. H. Soni and A. Mali, "Notice of Removal: Integration of MODBUS/TCP master monitoring and control system using python for high power RF system," 2015 International Conference on Electrical, Electronics, Signals, Communication and Optimization (EESCO), pags. 1-6, 2015, doi: 10.1109/EESCO.2015.7253952.*

[5] R. Celi, A. Sempértegui, D. Morocho, D. Loza, D. Alulema and M. Proaño, "Study, design and construction of a 3D printer implemented through a delta robot," 2015 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON), pags. 717-722, 2015, doi: 10.1109/Chilecon.2015.7404650.

[6] 3Dnatives, (1 de mayo de 2022), TOP 10 con los mejores softwares CAD para todos los niveles - 3Dnatives, dirección: https://www.3dnatives.com/es/mejores-softwares-cad-programa-180320192/#!.

[7] Repetier, (4 de abril de 2022), Repositorio de repetier firmware, GitHub - repetier/repetier-firmware: Firmware for arduino based reprap 3D printer, dirección: https://github.com/repetier/Repetier-Firmware.

[8] Marlin, (4 de abril de 2022), Repositorio de Marlin Firmware, GitHub - marlinfirmware/marlin, dirección: https://github.com/MarlinFirmware/Marlin.

[9] Yi Wei Daniel Tay, Biranchi Panda, Suvash Chandra Paul, Nisar Ahamed Noor Mohamed, Ming Jen Tan & Kah Fai Leong, “3D printing trends in building and construction industry: a review”, 2017 Virtual and Physical Prototyping.

[10] Isaac Perkins & Martin Skitmore, “Three-dimensional printing in the construction industry: A review”, 2015 International Journal of Construction Management,C.

[11] Pattavanitch, J., S. Panyaru. "Development of a new linear delta robot for the fused deposition modelling process.", 2017 Journal of Research and Applications in Mechanical Engineering 5.1, pags. 42-54, 2017.

[12] Vischer Peter, Reymond Clavel. "Kinematic calibration of the parallel Delta robot.", 1998 Robotica 16.2, pags. 207-218, 1998.

[13] Edwards, Mary. "Robots in industry: An overview.", 1984 Applied ergonomics 15.1, pags. 45-53, 1984

[14] Rey, L., R. Clavel. "The delta parallel robot.", 1999 Parallel Kinematic Machines. Springer, London, pags. 401-417, 1999

[15] Paul, Suvash Chandra, et al. "A review of 3D concrete printing systems and materials properties: Current status and future research prospects.", 2018 Rapid Prototyping Journal, 2018.

[16] Wong, Kaufui V., Aldo Hernandez. "A review of additive manufacturing.", 2012 International scholarly research notices 2012, 2012.

[17] I. Ottoway, (4 de marzo de 2022), Pycomm3 Documentation, Release 1.2.6, dirección: docs-pycomm3-dev-en-latest.

[18] Bell, Charles. “3D printing with delta printers.” Apress, 2015.

[19] Putnam, Craig, and A. G. R. Bodine. "Industrial 3D Printer."

[20] Alberto Portero, Jesús Salvador Lozano Rogado, Santiago Salamanca Miño. “Control de robot cartesiano mediante PLC S7-1200 de Siemens y Arduino para impresión 3D.", 5-7 de septiembre de 2018, Actas de las XXXIX Jornadas de Automática, Badajoz, 2018.

[21] PyCon 2019, 6 de mayo de 2019, Jonas Neubert - What is a PLC and how do I talk Python to

it? - PyCon 2019, direccion: https://www.youtube.com/watch?v=a0l29lgDf6k

[22] (19 de junio de 2022), GitHub. GitHub - libplctag: This C library provides a portable and simple API for accessing Allen-Bradley and Modbus PLC data over Ethernet, dirección: https://github.com/libplctag/libplctag (accedido el 19 de junio de 2022).

[23] José Guadalupe Zavala Villalpando, Ricardo Domínguez Guevara, Jesús Iván Orizaba Aguilar. "Implementación de HMI para un plc micrologix 1100 con raspberry pi 2 modelo b.", 2016 Pistas Educativas, 2016, vol. 38, no 121.

[24] (17 de mayo de 2022), "Software Features". Professional 3D Printing Software | Simplify3D. dirección: https://www.simplify3d.com/software/features/.

[25] (30 de marzo de 2022), GitHub - Ultimaker/Cura: 3D printer / slicing GUI built on top of the Uranium framework. GitHub, dirección: https://github.com/Ultimaker/Cura.

[26] (16 de mayo de 2022), GitHub - MatterHackers/MatterControl: 3D printing software for Windows, Mac and Linux. GitHub, dirección: https://github.com/MatterHackers/MatterControl.

[27] (16 de mayo de 2022), GitHub - slic3r/Slic3r: Open Source toolpath generator for 3D printers, dirección: https://github.com/slic3r/Slic3r.

[28] R.L. Williams II, (January 2016). The Delta Parallel Robot: Kinematics Solutions, Internet Publication, dirección: https://www.ohio.edu/mechanical-faculty/williams/html/PDF/DeltaKin.pdf

[29] Can, Fatih Cemal, Murat Hepeyiler, and Özgün Başer. "A novel inverse kinematic approach for delta parallel robot." *International Journal of Materials, Mechanics and Manufacturing* 6.5, pags. 321-326, 2018

[30] (2022), Relativity, dirección: https://www.relativityspace.com/rockets.

[31] Sobieski, Wojciech, and Wojsciech Kiński. "Geometry extraction from GCODE files destined for 3D printers." *Technical Sciences* 23.2, pags. 115-130, 2020.

[32] (2022), HAGE3D, dirección: https://www.hage3d.com/en/industrial-3d-printer/slicing-software-control.

[33] The Qt Company, (2022), Documentación de QML, dirección: https://doc.qt.io/qt-

5/qmltypes.html.

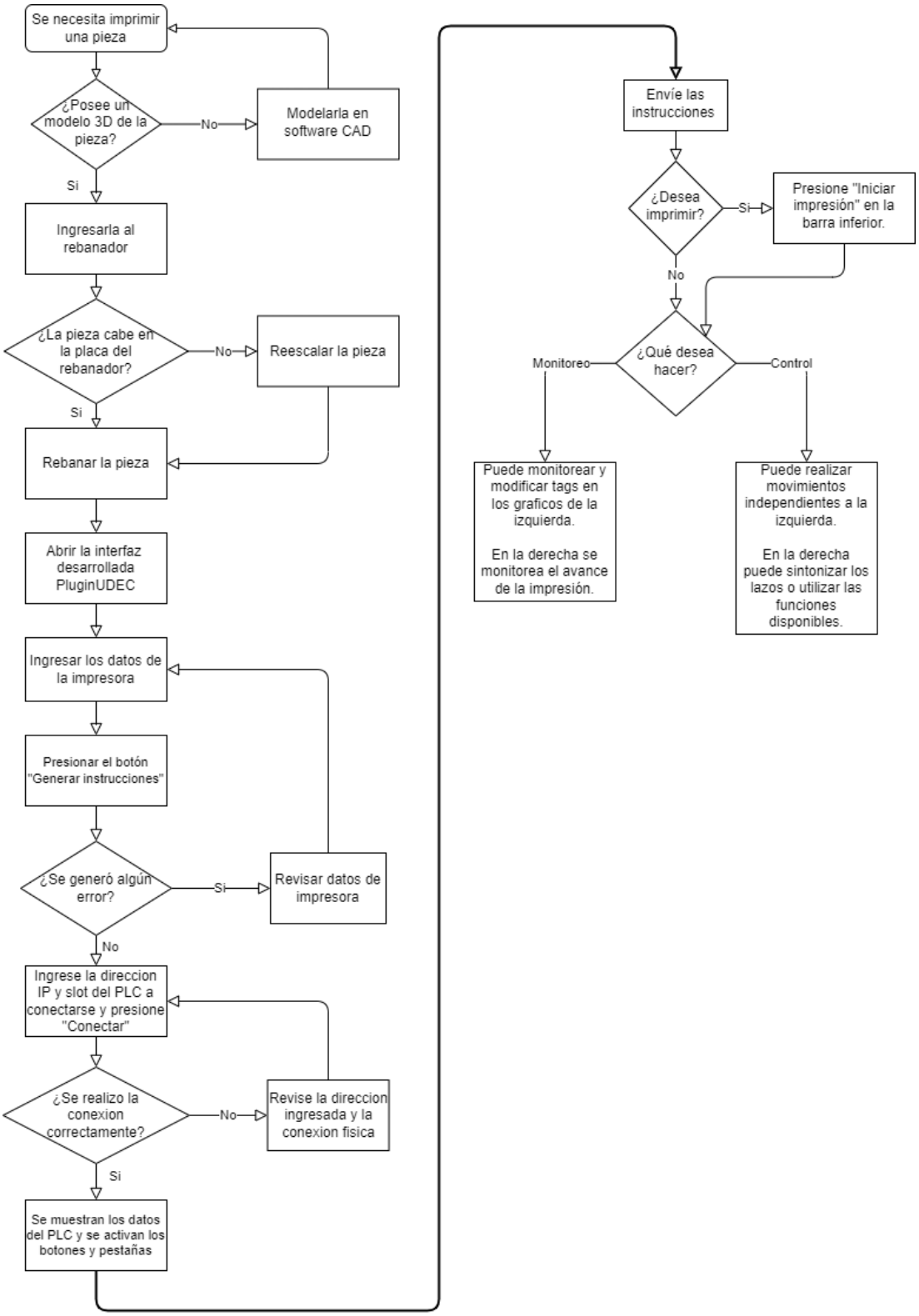
[34] Matias N. López, "*Desarrollo de plugin Cura para la construcción y generación de instrucciones de impresión 3D*", Departamento de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de Concepción, Informe de proyecto, jul. de 2022.

[35] Matias N. López, (2022), Repositorio de proyecto memoria de título, dirección: https://github.com/matlopb/3DPrinterCommUDEC/tree/feature-qt6

[36] (30 de marzo de 2022), GitHub - Ultimaker/Cura: Running Cura from Source. GitHub, dirección: https://github.com/Ultimaker/Cura/wiki/Running-Cura-from-Source.

Anexo A. Diagramas de flujo

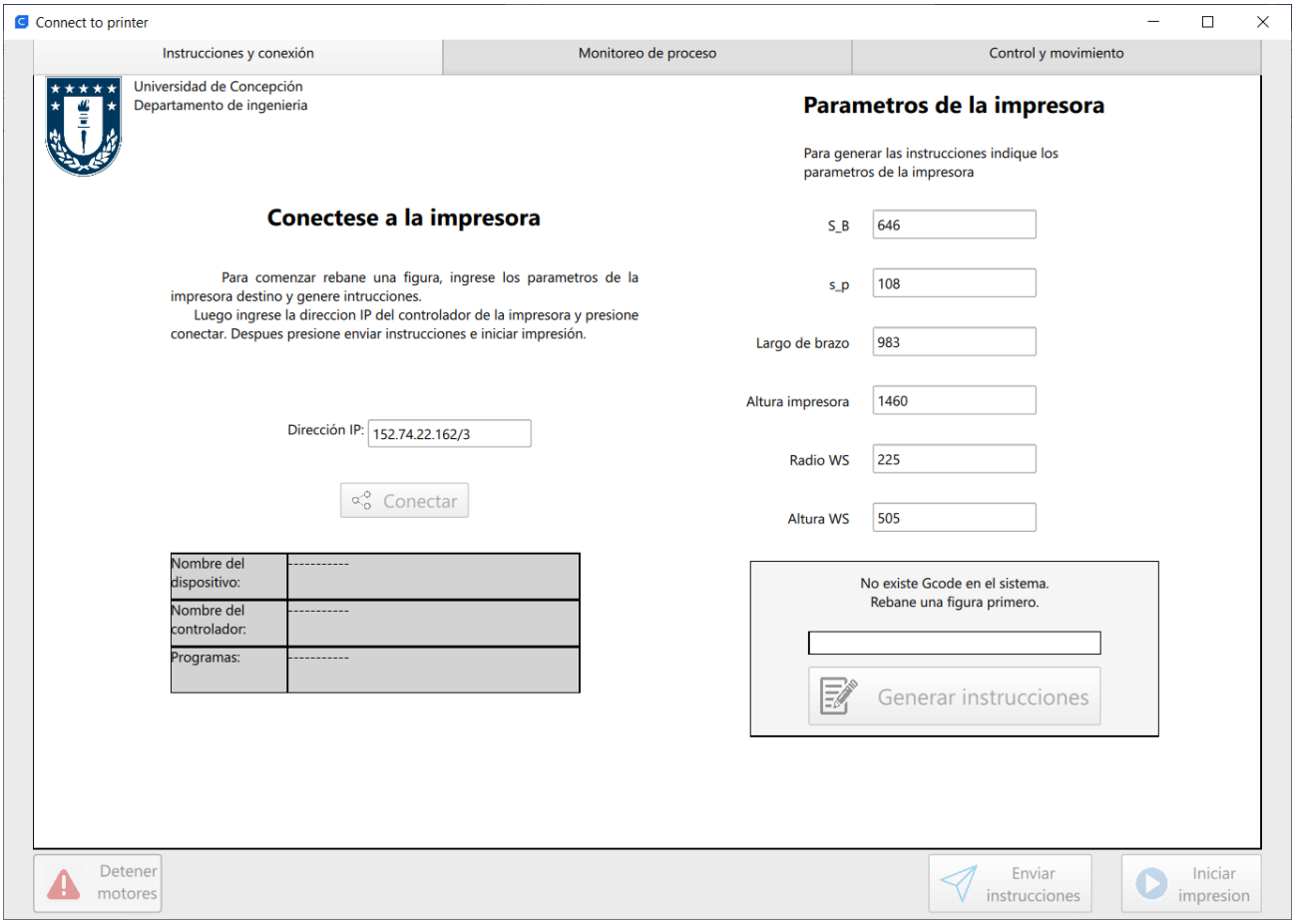
7.3. A.1. Diagrama de flujo de la interfaz



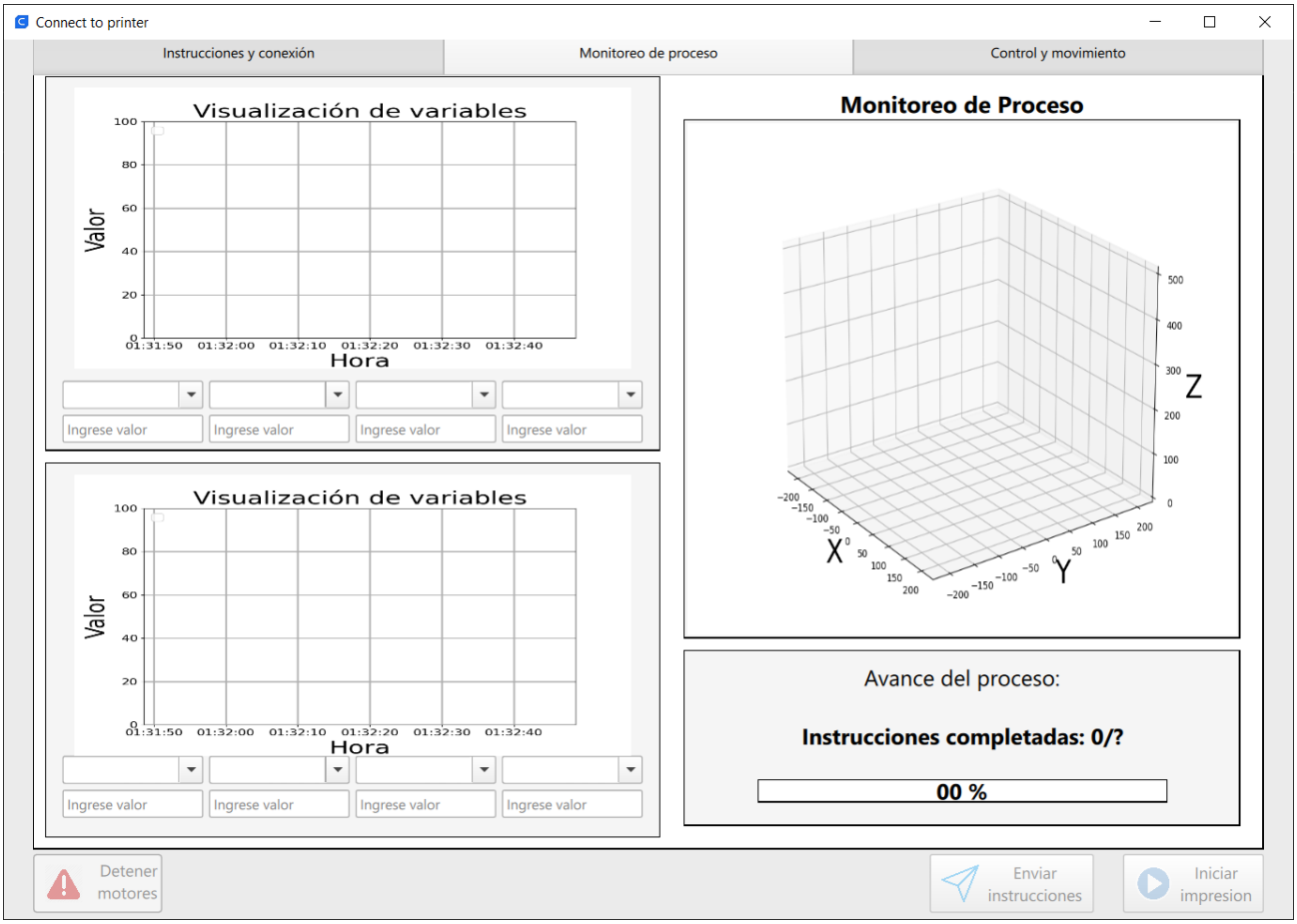
A.1

Anexo B. Pantallas de operación

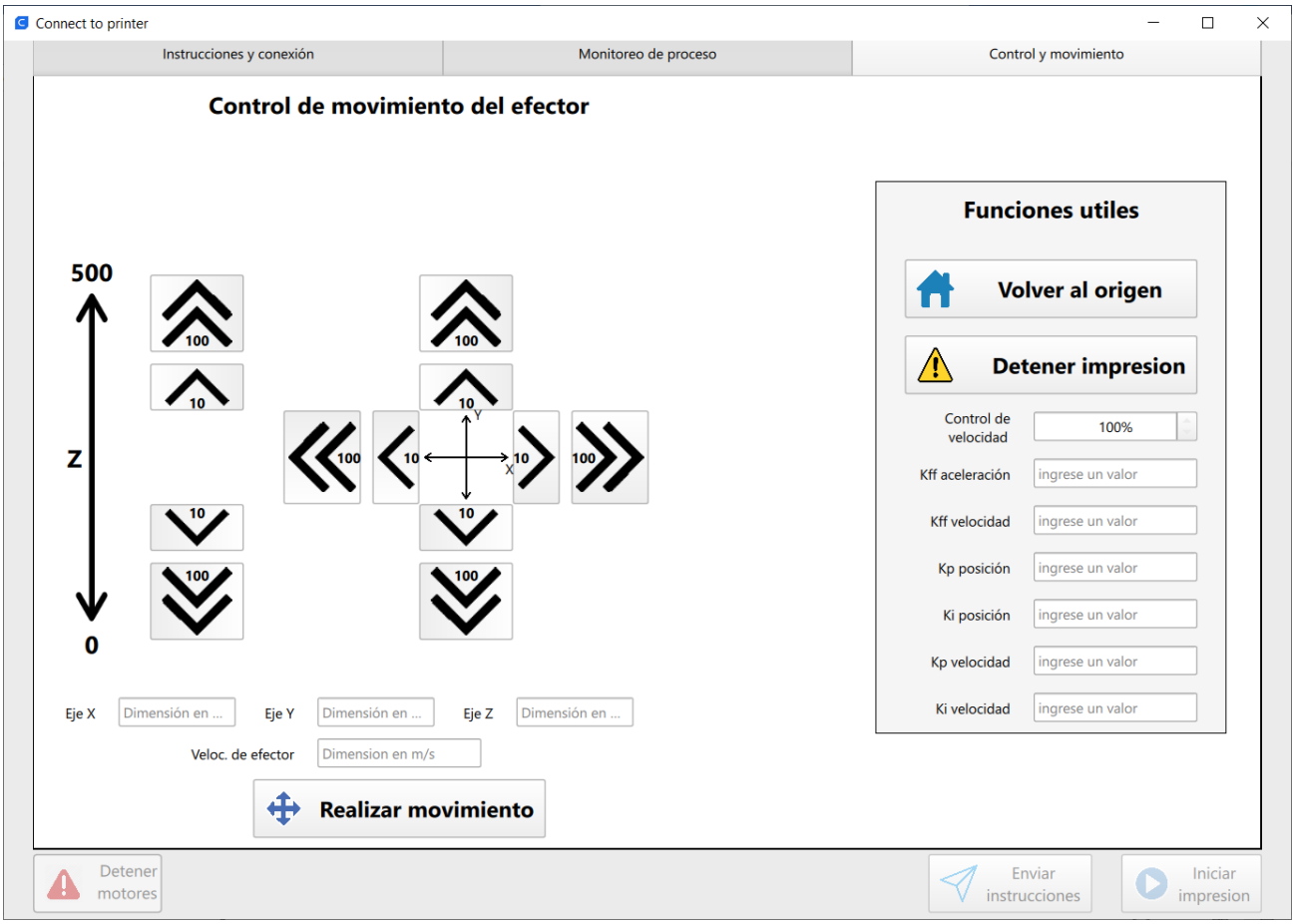
B.1. Pantalla de generación de instrucciones y conexión con impresora



B.2. Pantalla de monitoreo de proceso



B.3. Pantalla de control de impresora



Anexo C. Códigos

C.1. __init__.py

```
from . import PluginUDEC

from UM.i18n import i18nCatalog
i18n_catalog = i18nCatalog("PluginUDEC")

def getMetaData():
   return{}

def register(app):
    return {"extension":PluginUDEC.PluginUDEC()}
```

C.2. plugin.json

```
{
   "name": "PluginUDEC",
   "author": "Matías Nicolás López Barriga",
   "version": "1.0.1",
   "api": 8,
   "description": "Herramienta para transformar g-code a archivos de extension
.l5k",
   "supported_sdk_versions": ["5.0.0", "6.0.0", "7.0.0", "8.0.0"]
}
```

C.3. ArrowButtons.qml

```
import QtQuick.Layouts 1.3
import QtQuick 2.15
import QtQuick.Controls

Item {
    property alias lowButton: xplus10
    property alias highButton: xplus100
    property int rotation: 0
    Button{
        id: xplus10

        width: 40
        height: 80
        icon.source: "./images/arrow.png"
        icon.width: 32
        icon.height: 80

        Text {
            id: amount1
            text: qsTr("10")
            anchors.verticalCenter: parent.verticalCenter
            anchors.left: parent.left
```

```
            font.bold: true
            transform: Rotation{origin.x: amount1.width/2; origin.y:
amount1.height/2; angle: -rotation}
        }
    }
    Button{
        id: xplus100

        width: 66
        height: 80
        anchors.left: xplus10.right
        anchors.leftMargin: 10
        icon.source: "./images/doublearrow.png"
        icon.width: 58
        icon.height: 80

        Text {
            id: amount10
            text: qsTr("100")
            anchors.verticalCenter: parent.verticalCenter
            anchors.left: parent.left
            font.bold: true
            transform: Rotation{origin.x: amount10.width/2; origin.y:
amount10.height/2; angle: -rotation}
        }
    }
}
```

C.4. MessageDialog.qml

```
import QtQuick 2.15
import QtQuick.Controls
import QtQml 2.0
import QtQuick.Window 2.15
import UM 1.2 as UM
import Cura 1.0 as Cura

Window {
    id: messagedialog
    title: manager.get_message_title()
    width: {
        if (Qt.platform.os == "linux"){
            300 * screenScaleFactor
        }
        else if (Qt.platform.os == "windows"){
            350 * screenScaleFactor
        }
        else if (Qt.platform.os == "osx"){
            350 * screenScaleFactor
        }
    }
    height: {
        if (Qt.platform.os == "linux"){
            150 * screenScaleFactor
        }
```

```
    else if (Qt.platform.os == "windows"){
        150 * screenScaleFactor
    }
    else if (Qt.platform.os == "osx"){
        150 * screenScaleFactor
    }
}
minimumWidth: {
    if (Qt.platform.os == "linux"){
        300 * screenScaleFactor
    }
    else if (Qt.platform.os == "windows"){
        350 * screenScaleFactor
    }
    else if (Qt.platform.os == "osx"){
        350 * screenScaleFactor
    }
}
minimumHeight: {
    if (Qt.platform.os == "linux"){
        50 * screenScaleFactor
    }
    else if (Qt.platform.os == "windows"){
        50 * screenScaleFactor
    }
    else if (Qt.platform.os == "osx"){
        50 * screenScaleFactor
    }
}

Component.onCompleted: {
        x = Screen.width / 2 - width / 2
        y = Screen.height / 2 - height / 2
    }

Rectangle {
    id: rectangle
    anchors.centerIn: parent
    width: parent.width
    height: parent.height
    color: "#ffffff"
    border.width: 1
    layer.enabled: false

    Image {
        id: message_image
        source: {
            if (manager.get_message_style() == "e"){
                "./images/error.png"
            }
            else if (manager.get_message_style() == "i"){
                "./images/complete.png"
            }
            else if (manager.get_message_style() == "r"){
                "./images/warning.png"
            }
        }
        fillMode: Image.PreserveAspectFit;
```

```
            anchors.left: parent.left
            anchors.leftMargin: 5
            anchors.right: message.left
            anchors.rightMargin: 5
            anchors.verticalCenter: message.verticalCenter
        }

        Text
        {
            id: message;
            anchors.right: parent.right
            anchors.rightMargin: 10
            anchors.left: parent.left
            anchors.leftMargin: 50
            anchors.top: parent.top
            anchors.topMargin: 20
            width: parent.width - 40
            wrapMode: Text.Wrap
            text: manager.get_message_content()
        }

        AnimatedImage{
            id: loading
            source: "images/loading.gif"
            width: 40
            anchors.top: message.bottom
            anchors.topMargin: 20
            anchors.horizontalCenter: okButton.horizontalCenter
            fillMode: AnimatedImage.PreserveAspectFit
            visible: (manager.get_message_style() == "r") ? true : false
        }

        Button
            {
                id: okButton;
                anchors.horizontalCenter: parent.horizontalCenter
                anchors.bottom: parent.bottom
                anchors.bottomMargin: 10
                width: 125
                height: 30
                visible: (manager.get_message_style() == "r") ? false : true

                Text
                {
                    anchors.centerIn: parent
                    width: contentWidth
                    height: contentHeight
                    font.bold: true;
                    font.pointSize: 10;
                    font.pixelSize: 17;
                    text: qsTr("Ok")
                }
                onClicked: messagedialog.close()
            }
    }

}
```

C.5. ParamArea.qml

```
import QtQuick 2.0
import QtQuick.Controls

Item
{
    id: container

    property alias name: title.text
    property alias paramText: paramfield.text
    property alias help: helpText.text
    property string helpSide: ""
    property alias input: paramfield
    property alias placeholder: paramfield.placeholderText
    width: 140;
    height: 25;

    TextField
    {
        id: paramfield

        width: parent.width;
        height: parent.height;
        placeholderText: qsTr("Dimensión en mm");
        //validator: RegExpValidator{ regExp: /\d{1,7}([.]\d{1,3})$|\d{1,7}/ }
    }

    Text
    {
        id: title;

        text: qsTr(name);
        anchors.right: paramfield.left;
        anchors.verticalCenter: paramfield.verticalCenter;
        anchors.rightMargin: 20;
    }

    Rectangle
    {
        id: helpZone;

        width: helpText.contentWidth+10;
        height: helpText.contentHeight+5;
        border.width: 1;
        color: "#fffaf0";
        anchors.margins: 20
        anchors.verticalCenter: (helpSide == "left" || helpSide == "right") ?
paramfield.verticalCenter : undefined
        anchors.horizontalCenter: (helpSide == "below" || helpSide == "above") ?
paramfield.horizontalCenter : undefined
        anchors.right: (helpSide == "left") ? paramfield.left : undefined
        anchors.left: (helpSide == "right") ? paramfield.right : undefined
        anchors.top: (helpSide == "below") ? paramfield.bottom : undefined
        anchors.bottom: (helpSide == "above") ? paramfield.top : undefined
        visible: (help == "") ? false : paramfield.hovered;
```

```
        z: 100;

        Text
        {
            id: helpText;
            anchors.centerIn: parent
        }
    }
}
```

C.6. ProgressBar.qml

```
import QtQuick 2.15
import QtQuick.Controls 2.15
import QtQuick.Window 2.15

Window {
    id: mainWindow

    width: 300
    height: 150
    visible: true
    color: "#ffffff"
    title: qsTr("Proceso en curso")

    Component.onCompleted: {
            x = Screen.width / 2 - width / 2
            y = Screen.height / 2 - height / 2
        }

    Column {
        spacing: 10
        anchors.centerIn: parent

        TextArea {
            id: loading
            text: qsTr("Espere mientras se completa el proceso")
        }

        ProgressBar {
            id: progressBar
            from: 0
            anchors.horizontalCenter: parent.horizontalCenter
        }

    }

    Connections {
        target: manager

        function onProgressChanged(progress) {
            progressBar.value = progress;
        }
        function onProgressTotalChanged(total) {
            progressBar.to = total;
```

```
        }

    }
}
```

C.7. Tagbox.qml

```
import QtQuick 2.15
import QtQuick.Controls

Item {
    property alias combobox: tagbox
    property alias combobox_input: tagbox_input
    width: childrenRect.width
    height: childrenRect.height

    ComboBox{
        id: tagbox

        editable: true
        TextField
        {
            id: tagbox_input

            width: parent.width;
            height: parent.height;
            placeholderText: qsTr("Ingrese valor");
            validator: RegularExpressionValidator{ regularExpression:
/\d{1,7}([.]\d{1,3})+$|\d{1,7}/ }
            anchors.top: parent.bottom
            anchors.topMargin: 5
            anchors.horizontalCenter: parent.horizontalCenter
        }
    }

}
```

C.8. MainWindow.qml

```
import QtQuick 2.15
import QtQml 2.0
import QtQuick.Window 2.15
import QtQuick.Controls
import QtQuick.Layouts


ApplicationWindow {
    id: login_dialog

    visible: true
    color: "#EBEBEB"
    title: qsTr("Connect to printer")
    property variant win
    property string plc_path: ""
```

```
property var plc_info:["-----", "-----", "-----"]
property bool tagbox1_active: false
property bool tagbox2_active: false
property bool tagbox3_active: false
property bool tagbox4_active: false
property bool connected: false
property bool is_emergency: false
property string total_instructions: ""
property bool is_printing: false
property string ws_radio: ""
property string ws_altura: ""
property var tb_counter: [0,0,0,0,0,0,0,0]

width: {
    if (Qt.platform.os == "linux"){
        900 * screenScaleFactor
    }
    else if (Qt.platform.os == "windows"){
        1100 * screenScaleFactor
    }
    else if (Qt.platform.os == "osx"){
        1100 * screenScaleFactor
    }
}
height: {
    if (Qt.platform.os == "linux"){
        590 * screenScaleFactor
    }
    else if (Qt.platform.os == "windows"){
        750 * screenScaleFactor
    }
    else if (Qt.platform.os == "osx"){
        750 * screenScaleFactor
    }
}
minimumHeight: {
    if (Qt.platform.os == "linux"){
        590 * screenScaleFactor
    }
    else if (Qt.platform.os == "windows"){
        750 * screenScaleFactor
    }
    else if (Qt.platform.os == "osx"){
        750 * screenScaleFactor
    }
}
minimumWidth: {
    if (Qt.platform.os == "linux"){
        900 * screenScaleFactor
    }
    else if (Qt.platform.os == "windows"){
        1100 * screenScaleFactor
    }
    else if (Qt.platform.os == "osx"){
        1100 * screenScaleFactor
    }
}
```

```
Component.onCompleted: {
        x = Screen.width / 2 - width / 2
        y = Screen.height / 2 - height / 2
        frame.height = height - 60
        frame.width = width - 50

        if (manager.look_for_gcode() == false){
            login_zone.notify_gcode_status()
        }
    }
onWidthChanged: frame.width = width - 50
onHeightChanged: frame.height = height - 80

Connections {
    target: manager

    function onProgressEnd() {
        var mDialog = Qt.createComponent("MessageDialog.qml");
        win = mDialog.createObject(login_dialog)
        win.show()
    }

    function onProgressChanged(progress) {
        login_zone.change_progress(progress)
        login_zone.set_instructions_status("Cálculo en proceso...")
    }

    function onProgressTotalChanged(total) {
        console.log(total)
        login_zone.change_bar_total(total)
    }

    function onConnectionAchieved(){
        win.close()
    }
}

onClosing:{
    close.accepted = false
    monitoring_tab.item.stop_timer()
    close.accepted = true
}

Button{
    id: send_instructions

    width: 140;
    height: 50;
    enabled: connected
    anchors.right: start_printing.left
    anchors.rightMargin: 25;
    anchors.bottom: parent.bottom
    anchors.bottomMargin: 5
    Text {
        anchors.right: parent.right
        anchors.rightMargin: 10
        anchors.verticalCenter: parent.verticalCenter
        text: qsTr("Enviar\ninstrucciones")
```

```
            font.weight: Font.Medium
            font.pixelSize: 14
            opacity: enabled ? 1.0 : 0.4
            horizontalAlignment: Text.AlignHCenter
        }
        Image {
            source: "./images/send.png"
            opacity: enabled ? 1 : 0.4
            anchors.left: parent.left
            anchors.leftMargin: 10
            anchors.verticalCenter: parent.verticalCenter
            height: 32
            width: 32
        }
        onClicked:{
            manager.send_instructions()
            manager.check_servos()
        }
    }

    Button{
        id: start_printing

        width: 120;
        height: 50;
        enabled: connected
        anchors.right: parent.right
        anchors.rightMargin: 25;
        anchors.bottom: parent.bottom
        anchors.bottomMargin: 5
        Text {
            anchors.right: parent.right
            anchors.rightMargin: 10
            anchors.verticalCenter: parent.verticalCenter
            text: (is_printing) ? qsTr("Pausar\nimpresion") :
qsTr("Iniciar\nimpresion")
            font.weight: Font.Medium
            font.pixelSize: 14
            opacity: enabled ? 1.0 : 0.4
            horizontalAlignment: Text.AlignHCenter
        }
        Image {
            source: (is_printing) ? "./images/pause.png" : "./images/play.png"
            opacity: enabled ? 1 : 0.4
            anchors.left: parent.left
            anchors.leftMargin: 10
            anchors.verticalCenter: parent.verticalCenter
            height: 32
            width: 32
        }
        onClicked: {
            is_printing = manager.switch_printing()
            monitoring_zone.set_progress_timer(is_printing)
        }
    }

    Button{
        id: emergency_stop
```

```
        width: 110
        height: 50
        enabled: connected
        anchors.left: parent.left
        anchors.leftMargin: 25
        anchors.verticalCenter: start_printing.verticalCenter
        contentItem: Text {
                text: qsTr("Detener \nmotores")
                font.weight: Font.Medium
                font.pixelSize: 14
                font.bold: false
                opacity: enabled ? 1.0 : 0.4
                horizontalAlignment: Text.AlignRight
                color: (emergency_stop.hovered && enabled) ? "white" : "black"
        }
        background: Rectangle {
                border.color: "darkgrey"
                color: (emergency_stop.hovered && enabled) ? Qt.lighter("red",
1.2) : 'whitesmoke'
                border.width: 1
                radius: 3
                Image {
                    source: "./images/emergency.png"
                    opacity: enabled ? 1 : 0.4
                    anchors.left: parent.left
                    anchors.leftMargin: 10
                    anchors.verticalCenter: parent.verticalCenter
                    height: 32
                    width: 32
                }
        }
        onClicked: {
            message.text = qsTr("Está a punto de detener forzosamente los
motores. \n ¿Está seguro que desea continuar?")
            is_emergency = true
            double_confirmation_window.visible = true
        }
    }

    Timer{
        id: emergency_on

        interval: 2000
        repeat: false
        running: false
        onTriggered: {manager.check_servos()}
    }

    Window{
        id: double_confirmation_window

        x: Screen.width / 2 - width / 2
        y: Screen.height / 2 - height / 2
        width: 350
        height: 150
        title: 'Detener impresión'
        visible: false
```

```
        Rectangle{
            anchors.fill: parent
            Image {
                id: message_image
                source: "./images/warning.png"
                fillMode: Image.PreserveAspectFit;
                width: 50
                height: 50
                anchors.horizontalCenter: message.horizontalCenter
                anchors.bottom: message.top
                anchors.topMargin: 20
            }

            Text {
                id: message
                anchors.bottom: buttons_row.top
                anchors.bottomMargin: 20
                anchors.horizontalCenter: buttons_row.horizontalCenter
                color: 'black'
            }

            RowLayout{
                id: buttons_row
                anchors.horizontalCenter: parent.horizontalCenter
                anchors.bottom: parent.bottom
                anchors.bottomMargin: 20
                Button{
                    id: accept

                    Layout.preferredWidth: 100
                    Layout.preferredHeight: 30
                    contentItem: Label {
                        text: qsTr("Detener")
                        verticalAlignment: Text.AlignVCenter
                        horizontalAlignment: Text.AlignHCenter
                    }
                    onClicked: {
                        if (is_emergency){manager.force_stop(); is_emergency =
false; emergency_on.running = true}
                        else{manager.stop_printing()}
                        double_confirmation_window.visible = false
                        is_printing = false}
                }
                Button{
                    id: decline

                    Layout.preferredWidth: 100
                    Layout.preferredHeight: 30
                    contentItem: Label {
                        text: qsTr("Cancelar")
                        verticalAlignment: Text.AlignVCenter
                        horizontalAlignment: Text.AlignHCenter
                    }
                    onClicked: {double_confirmation_window.visible = false}
                }
            }
        }
```

```
    }

    header: TabBar{
        id: coreBar
        width: parent.width - 50
        anchors.horizontalCenter: parent.horizontalCenter
        TabButton{
            height: 40
            contentItem: Label {
                text: qsTr("Instrucciones y conexión")
                verticalAlignment: Text.AlignVCenter
                horizontalAlignment: Text.AlignHCenter
            }
        }
        TabButton{
            height: 40
            contentItem: Label {
                text: qsTr("Monitoreo de proceso")
                verticalAlignment: Text.AlignVCenter
                horizontalAlignment: Text.AlignHCenter
            }
        }
        TabButton{
            height: 40
            contentItem: Label {
                text: qsTr("Control y movimiento")
                verticalAlignment: Text.AlignVCenter
                horizontalAlignment: Text.AlignHCenter
            }
        }
    }

    StackLayout {
        id: frame
        anchors.top: coreBar.bottom
        anchors.horizontalCenter: parent.horizontalCenter
        currentIndex: coreBar.currentIndex

//################################################## Pestaña de conexión
####################################################################

        Item {
            id: login_tab

            Rectangle {
                id: login_zone
                width: login_dialog.width - 50
                height: login_dialog.height - 80
                border.width: 1

                function notify_gcode_status(){
                    set_instructions_status("No existe Gcode en el
sistema.\nRebane una figura primero.")
                    generate_instructions.enabled = false
                }
                function set_instructions_status(status){
                    instructions_status_text.text = qsTr(status)
                    if (progressBar.value == progressBar.to){
```

```
                        total_instructions = manager.get_n_coor()
                        instructions_status_text.text = qsTr("Se generaron "+
total_instructions +" posiciones. Las instrucciones estan listas para su
envío.")
                    }
                }
                function change_progress(progress) {
                    progressBar.value = progress
                    console.log(progress)
                }

                function change_bar_total(total) {
                    progressBar.to = total
                }

                Image {
                    id: udecLogo
                    source: "./images/udeclogo.jpg"
                    anchors.left: parent.left
                    anchors.leftMargin: 10
                    anchors.top: parent.top
                    anchors.topMargin: 10
                    width: 66
                    height: 86

                }

                Text {
                    id: udecText
                    text: qsTr("Universidad de Concepción \nDepartamento de
ingenieria")
                    anchors.left: udecLogo.right
                    anchors.leftMargin: 10
                    anchors.top:  udecLogo.top
                }

                ListModel{
                    id: plc_info_model

                    ListElement{
                        name: "Nombre del \ndispositivo"
                        value: "-----------"
                    }
                    ListElement{
                        name: "Nombre del \ncontrolador"
                        value: "-----------"
                    }
                    ListElement{
                        name: "Programas"
                        value: "-----------"
                    }
                }

                Component{
                    id: my_delegate

                    Rectangle{
                        border.width: 1
```

```
                width: 100
                height: 40
                color: "lightgrey"
                Text {
                    id: model_name
                    text: name + ": "
                }
                Rectangle{
                    border.width: 1
                    width: 250
                    height: 40
                    color: "lightgrey"
                    anchors.left: parent.right
                    Text {
                        id: model_value
                        text: value
                    }
                }
            }
        }

        Text{
            id: login_title

            text: "Conectese a la impresora";
            font.bold: true;
            font.pointSize: 16;
            font.pixelSize: 20;
            anchors.top: udecLogo.bottom;
            anchors.topMargin: 20;
            anchors.left: parent.left;
            anchors.leftMargin: 200;
        }

        ColumnLayout{
            id: main_column

            spacing: 30
            anchors.horizontalCenter: login_title.horizontalCenter
            anchors.top: login_title.bottom
            anchors.topMargin: 30

            Rectangle{
                width: 400
                height: 100

                Text{
                    id: main_text

                    anchors.fill: parent
                    text: qsTr("       Para comenzar rebane una figura,
ingrese los parametros de la impresora destino y genere intrucciones.\n
Luego ingrese la direccion IP del controlador de la impresora y presione
conectar. Despues presione enviar instrucciones e iniciar impresión.")
                    wrapMode: Text.WordWrap
                    horizontalAlignment: Text.AlignJustify
                }
            }
```

```
                    Row{
                        id: content_Item

                        height: 30
                        Layout.preferredWidth: 200
                        Layout.alignment: Qt.AlignHCenter

                        Text{
                            id: field_Indicator

                            text: qsTr("Dirección IP: ")
                        }

                        TextField{
                            id: ip_field

                            width: 140
                            placeholderText: qsTr("e.g. 192.168.0.15/2");
                            text: qsTr("152.74.22.162/3");
                            validator: RegularExpressionValidator{
regularExpression: /^(([01]?[0-9]?[0-9]|2([0-4][0-9]|5[0-5]))\.){3}([01]?[0-
9]?[0-9]|2([0-4][0-9]|5[0-5]))\/(([0-6]|3([0])|2([0-9]))\.)$/}
                        }
                    }
                    Button{
                        id: login_button

                        Layout.preferredWidth: 110
                        Layout.preferredHeight: 30
                        enabled: false
                        Layout.alignment: Qt.AlignHCenter
                        Text {
                            text: qsTr('Conectar')
                            color: generate_instructions.enabled ?
"black":"darkgrey"
                            anchors.right: parent.right
                            anchors.rightMargin: 10
                            anchors.verticalCenter: parent.verticalCenter
                            font.pointSize: 12
                        }
                        Image {
                            source: "./images/connect.png"
                            opacity: enabled ? 1 : 0.4
                            anchors.left: parent.left
                            anchors.leftMargin: 10
                            anchors.verticalCenter: parent.verticalCenter
                            height: 16
                            width: 16
                        }
                        onClicked: {
                            plc_path = ip_field.text
                            plc_info = manager.plc_info(plc_path)
                            connected = plc_info[3]
                            plc_info_model.setProperty(0, "value", plc_info[0])
                            plc_info_model.setProperty(1, "value", plc_info[1])
                            plc_info_model.setProperty(2, "value", plc_info[2])
                            if (connected){
```

```
                            monitoring_tab.enabled = true
                            control_tab.enabled = true
                            monitoring_zone.load_tags()
                            control_zone.get_actual_position()
                            control_zone.get_gains()
                        }
                    }
                }

                ListView{
                    model: plc_info_model
                    delegate: my_delegate
                    height: 200
                }
            }
            Text{
                id: params_title

                text: "Parametros de la impresora";
                font.bold: true;
                font.pointSize: 16;
                font.pixelSize: 20;
                anchors.top: parent.top;
                anchors.topMargin: 20;
                anchors.horizontalCenter: parent.horizontalCenter
                anchors.horizontalCenterOffset: parent.width/4
            }
            ColumnLayout
            {
                id: paramcolumn
                anchors.horizontalCenter: params_title.horizontalCenter
                anchors.top: params_title.bottom;
                anchors.topMargin: 20;
                Layout.preferredWidth: frame.width/2
                Layout.preferredHeight: frame.height
                spacing: 25;
                z:100

                Text{
                    id: params_subtitle

                    Layout.preferredWidth: params_title.width
                    Layout.alignment: Qt.AlignHCenter
                    text: qsTr("Para generar las instrucciones indique los
\nparametros de la impresora")
                }

                ParamArea{id: sb; paramText: "646"; name: "S_B"; help: " Es
la distancia entre los actuadores del RDL "; helpSide: "left"; Layout.alignment:
Qt.AlignHCenter}
                ParamArea{id: sp; paramText: "108"; name: "s_p"; help: " Es
la distancia entre los puntos de conexion \n del efector y los brazos del robot
"; helpSide: "left"; Layout.alignment: Qt.AlignHCenter}
                ParamArea{id: armLen; paramText: "983"; name: "Largo de
brazo"; help: " Es el largo de los brazos de la impresora "; helpSide: "left";
Layout.alignment: Qt.AlignHCenter}
                ParamArea{id: printerH; paramText: "1460"; name: "Altura
impresora"; help: " Es la distancia entre la base del RDL y la \n superficie de
```

```
impresion "; helpSide: "left"; Layout.alignment: Qt.AlignHCenter}
                    ParamArea{id: radio; paramText: "225"; name: "Radio WS";
help: " Es el radio de la base del espacio de trabajo "; helpSide: "left";
Layout.alignment: Qt.AlignHCenter; input.onTextChanged: ws_radio = paramText}
                    ParamArea{id: altura; paramText: "505"; name: "Altura WS";
help: " Es la altura del espacio de trabajo "; helpSide: "left";
Layout.alignment: Qt.AlignHCenter; input.onTextChanged: ws_altura = paramText}

                    Rectangle{
                        id: instructions_status

                        Layout.preferredWidth: login_zone.width/3
                        Layout.preferredHeight: 150
                        color: "whitesmoke"
                        Layout.alignment: Qt.AlignHCenter
                        border.width: 1

                        Text{
                            id: instructions_status_text

                            height: 30
                            width: parent.width - 50
                            anchors.horizontalCenter: parent.horizontalCenter
                            anchors.top: parent.top
                            anchors.topMargin: 10
                            text: qsTr("No existen instrucciones en memoria.")
                            horizontalAlignment: Text.AlignHCenter
                            wrapMode: Text.WordWrap
                        }
                        ProgressBar {
                            id: progressBar
                            width: parent.width - 100
                            height: 20
                            anchors.horizontalCenter: parent.horizontalCenter
                            anchors.top: instructions_status_text.bottom
                            anchors.topMargin: 20
                            from: 0
                            background: Rectangle {
                                            anchors.fill: parent
                                            color: "white"
                                            border.width: 1
                                            border.color: 'black'
                            }
                            contentItem: Rectangle {
                                            anchors.left: parent.left
                                            anchors.verticalCenter:
parent.verticalCenter
                                            height: parent.height - 4
                                            width: parent.width *
(parent.value/parent.to)
                                            color: 'dodgerblue'
                            }
                        }
                        Button{
                            id: generate_instructions

                            width: 250
                            height: 50
```

```
                            anchors.horizontalCenter: parent.horizontalCenter
                            anchors.bottom: parent.bottom
                            anchors.bottomMargin: 10
                            enabled: true
                            Text {
                                text: qsTr("Generar instrucciones")
                                color: generate_instructions.enabled ?
"black":"darkgrey"
                                anchors.right: parent.right
                                anchors.rightMargin: 10
                                anchors.verticalCenter: parent.verticalCenter
                                font.pointSize: 14
                            }
                            Image {
                                source: "./images/write.png"
                                opacity: enabled ? 1 : 0.4
                                anchors.left: parent.left
                                anchors.leftMargin: 10
                                anchors.verticalCenter: parent.verticalCenter
                                height: 32
                                width: 32
                            }
                            onClicked: {
                                manager.generate_instructions_list(sb.paramText,
sp.paramText, armLen.paramText,

printerH.paramText, radio.paramText, altura.paramText)
                                login_button.enabled = true
                            }
                        }
                    }
                }
            }
        }
//################################################## Pestaña de monitoreo
#####################################################################
        Item {
            id: monitoring_tab

            Rectangle {
                id: monitoring_zone

                width: login_dialog.width - 50
                height: login_dialog.height - 80
                border.width: 1

                function load_tags(){
                    var tag_list = ["Seleccione un tag..."]
                    var get_list = manager.plc_tag_list(plc_path)
                    for(let element in get_list){
                        tag_list.push(get_list[element])
                    }
                    tagbox_1.combobox.model = tag_list
                    tagbox_2.combobox.model = tag_list
                    tagbox_3.combobox.model = tag_list
                    tagbox_4.combobox.model = tag_list
                    tagbox2_1.combobox.model = tag_list
                    tagbox2_2.combobox.model = tag_list
```

```
                    tagbox2_3.combobox.model = tag_list
                    tagbox2_4.combobox.model = tag_list
                }
                function set_progress_timer(state) {progress_timer.running =
state}
                function stop_timer() {date_timer.running = false}

                Text{
                    id: tab_title

                    text: "Monitoreo de Proceso"
                    z: 100
                    font.bold: true;
                    font.pointSize: 16;
                    font.pixelSize: 20;
                    anchors.top: parent.top;
                    anchors.topMargin: 20;
                    anchors.horizontalCenter: progress_status.horizontalCenter

                }

                Rectangle{
                    id: chart_rect1

                    border.width: 1
                    width: parent.width/2
                    height: parent.height/2 - 15
                    anchors.left: parent.left
                    anchors.top: parent.top
                    anchors.margins: 10
                    color: "whitesmoke"

                    RowLayout{
                        id: combobox_row

                        spacing: 5
                        anchors.horizontalCenter: chart.horizontalCenter
                        anchors.top: chart.bottom
                        anchors.topMargin: 10

                        Tagbox{
                            id: tagbox_1

                            combobox.onActivated: {
                                manager.clear_series(0)
                                tb_counter[0] = 0
                                date_timer.running = true
                                tagbox1_active = true
                            }
                            combobox_input.onAccepted: {
                                manager.write_value(combobox.currentText,
combobox_input.text)
                            }
                        }
                        Tagbox{
                            id: tagbox_2

                            combobox.onActivated: {
```

```
                                tb_counter[1] = 0
                                manager.clear_series(1)
                                date_timer.running = true
                                tagbox2_active = true
                            }
                            combobox_input.onAccepted: {
                                manager.write_value(combobox.currentText,
combobox_input.text)
                            }
                        }
                        Tagbox{
                            id: tagbox_3

                            combobox.onActivated: {
                                tb_counter[2] = 0
                                manager.clear_series(2)
                                date_timer.running = true
                                tagbox3_active = true
                            }
                            combobox_input.onAccepted: {
                                manager.write_value(combobox.currentText,
combobox_input.text)
                            }
                        }
                        Tagbox{
                            id: tagbox_4

                            combobox.onActivated: {
                                tb_counter[3] = 0
                                manager.clear_series(3)
                                date_timer.running = true
                                tagbox4_active = true
                            }
                            combobox_input.onAccepted: {
                                manager.write_value(combobox.currentText,
combobox_input.text)
                            }
                        }
                    }

                    Image {
                        id: chart
                        width: parent.width -50
                        height: parent.height - 80
                        anchors.horizontalCenter: parent.horizontalCenter
                        anchors.top: parent.top
                        anchors.topMargin: 10
                        source: "image://perflog/plot_one"
                    }
                }

                Rectangle{
                    id: chart_rect2

                    border.width: 1
                    width: parent.width/2
                    height: parent.height/2 - 15
                    anchors.left: parent.left
```

```
                    anchors.bottom: parent.bottom
                    anchors.margins: 10
                    color: "whitesmoke"

                    RowLayout{
                        id: combobox_row2

                        spacing: 5
                        anchors.horizontalCenter: chart2.horizontalCenter
                        anchors.top: chart2.bottom

                        Tagbox{
                            id: tagbox2_1

                            combobox.onActivated: {
                                tb_counter[4] = 0
                                manager.clear_series(4)
                                date_timer.running = true
                                tagbox1_active = true
                            }
                            combobox_input.onAccepted: {
                                manager.write_value(combobox.currentText,
combobox_input.text)
                            }
                        }
                        Tagbox{
                            id: tagbox2_2

                            combobox.onActivated: {
                                tb_counter[5] = 0
                                manager.clear_series(5)
                                date_timer.running = true
                                tagbox2_active = true
                            }
                            combobox_input.onAccepted: {
                                manager.write_value(combobox.currentText,
combobox_input.text)
                            }
                        }
                        Tagbox{
                            id: tagbox2_3

                            combobox.onActivated: {
                                tb_counter[6] = 0
                                manager.clear_series(6)
                                date_timer.running = true
                                tagbox3_active = true
                            }
                            combobox_input.onAccepted: {
                                manager.write_value(combobox.currentText,
combobox_input.text)
                            }
                        }
                        Tagbox{
                            id: tagbox2_4

                            combobox.onActivated: {
                                tb_counter[7] = 0
```

```
                                manager.clear_series(7)
                                date_timer.running = true
                                tagbox4_active = true
                            }
                            combobox_input.onAccepted: {
                                manager.write_value(combobox.currentText,
combobox_input.text)
                            }
                        }
                    }

                    Image {
                        id: chart2
                        width: parent.width -50
                        height: parent.height - 80
                        anchors.horizontalCenter: parent.horizontalCenter
                        anchors.top: parent.top
                        anchors.topMargin: 10
                        source: "image://perflog/plot_two"
                    }
                }

                Rectangle{
                    id: figure_plot_rect

                    anchors.top: tab_title.bottom
                    anchors.bottom: progress_status.top
                    anchors.bottomMargin: 10
                    anchors.right: progress_status.right
                    anchors.left: progress_status.left
                    color:"whitesmoke"
                    border.width: 1

                    Image {
                        id: figure_plot
                        anchors.top: parent.top
                        anchors.bottom: parent.bottom
                        anchors.right: parent.right
                        anchors.left: parent.left
                        anchors.margins: 3
                        source: "image://perflog/figure"
                    }
                    Timer{
                        id: figure_timer
                        interval: 500
                        running: is_printing
                        repeat: true
                        onTriggered: {
                            manager.get_figure_progress()
                            reload3()
                        }
                        function reload3() { var t = figure_plot.source;
figure_plot.source = ""; figure_plot.source = t; }
                    }
                }

                Rectangle{
                    id: progress_status
```

```
                    width: frame.width/2 - 50
                    height: 150
                    color: "whitesmoke"
                    border.width: 1
                    anchors.bottom: parent.bottom
                    anchors.right: parent.right
                    anchors.margins: 20

                    Text{
                        id: progress_text

                        height: 30
                        width: contentWidth
                        anchors.horizontalCenter: parent.horizontalCenter
                        anchors.top: parent.top
                        anchors.topMargin: 10
                        font.pointSize: 14
                        text: qsTr("Avance del proceso:")
                        horizontalAlignment: Text.AlignHCenter
                    }
                    Text{
                        id: progress_step

                        height: 30
                        width: contentWidth
                        anchors.horizontalCenter: parent.horizontalCenter
                        anchors.top: progress_text.bottom
                        anchors.topMargin: 20
                        text: qsTr("Instrucciones completadas: 0/?")
                        font.bold: true
                        font.pointSize: 14
                        horizontalAlignment: Text.AlignHCenter
                    }
                    Text{
                        id: percentage_text
                        z:100

                        height: 30
                        width: contentWidth
                        anchors.horizontalCenter:
process_progressBar.horizontalCenter
                        anchors.verticalCenter:
process_progressBar.verticalCenter
                        anchors.verticalCenterOffset: 2
                        text: qsTr("00 %")
                        font.bold: true
                        font.pointSize: 14
                        horizontalAlignment: Text.AlignHCenter
                    }
                    ProgressBar {
                        id: process_progressBar
                        height: 20
                        width: 350
                        anchors.horizontalCenter: parent.horizontalCenter
                        anchors.bottom: parent.bottom
                        anchors.bottomMargin: 20
                        from: 0
```

```
                        to: 100
                        background: Rectangle {
                                        anchors.fill: parent
                                        color: "white"
                                        border.width: 1
                                        border.color: 'black'
                        }
                        contentItem: Item {
                            implicitWidth: 200
                            implicitHeight: 4
                            Rectangle {
                                        id: content
                                        height: parent.height
                                        width: process_progressBar.width *
(process_progressBar.value/process_progressBar.to)
                                        color: 'dodgerblue'
                            }
                        }
                    }
                    Timer{
                        id: progress_timer
                        interval: 2000
                        repeat: true
                        running: false
                        onTriggered: {
                            var progress = manager.get_progress_percentage()
                            percentage_text.text = progress[0].toString() + "%"
                            process_progressBar.value = progress[0]
                            progress_step.text = "Instrucciones completadas: " +
progress[1]+"/"+total_instructions
                            progress_text.text = "Avance del proceso:"
                            if (progress[0] >= 100){
                                progress_text.text = "Proceso terminado."
                                progress_timer.running = false
                                is_printing = false
                            }
                        }
                    }
                }

                Timer{
                    id: date_timer
                    interval: 1000
                    running: false
                    repeat: true
                    onTriggered: {
                        var upper_tagboxes_info = {"tagbox_1" : [tagbox1_active,
tagbox_1.combobox.currentText, 0],
                            "tagbox_2" : [tagbox2_active,
tagbox_2.combobox.currentText, 1],
                            "tagbox_3" : [tagbox3_active,
tagbox_3.combobox.currentText, 2],
                            "tagbox_4" : [tagbox4_active,
tagbox_4.combobox.currentText, 3]}
                        var lower_tagboxes_info = {"tagbox2_1" :
[tagbox1_active, tagbox2_1.combobox.currentText, 4],
                            "tagbox2_2" : [tagbox2_active,
tagbox2_2.combobox.currentText, 5],
```

```
                            "tagbox2_3" : [tagbox3_active,
tagbox2_3.combobox.currentText, 6],
                            "tagbox2_4" : [tagbox4_active,
tagbox2_4.combobox.currentText, 7]}
                        var upper_active_tagboxes =
who_active(upper_tagboxes_info)
                        var lower_active_tagboxes =
who_active(lower_tagboxes_info)

                        update_charts(upper_active_tagboxes,
lower_active_tagboxes)
                        reload_upper_plot()
                        reload_lower_plot()
                    }

                    function reload_upper_plot() { var t = chart.source;
chart.source = ""; chart.source = t; }
                    function reload_lower_plot() { var t = chart2.source;
chart2.source = ""; chart2.source = t; }

                    function who_active(tagbox_dict){
                        var active_elements = {}
                        for (let tag in tagbox_dict){
                            if (tagbox_dict[tag][0] && tagbox_dict[tag][1] !=
"Seleccione un tag..."){
                                active_elements[tag] = [tagbox_dict[tag][1],
tagbox_dict[tag][2], tagbox_dict[tag][3]]
                            }
                        }
                        return active_elements
                    }

                    function update_charts(upper_tag_dict, lower_tag_dict){
                        var tag_names = []
                        var tag_spot = []
                        var upper_len = 0
                        var lower_len = 0
                        for (let upper_tag in upper_tag_dict){
                            tag_names.push(upper_tag_dict[upper_tag][0])
                            tag_spot.push(upper_tag_dict[upper_tag][1])
                            upper_len++
                        }
                        for (let lower_tag in lower_tag_dict){
                            tag_names.push(lower_tag_dict[lower_tag][0])
                            tag_spot.push(lower_tag_dict[lower_tag][1])
                            lower_len++
                        }

                        if (tag_names.length != 0){
                            var tagbox_values = manager.update_series(tag_names,
tag_spot, upper_len, lower_len)
                            var counter = 0
                            for (let u_element in upper_tag_dict){
                                fill_text(tagbox_values[counter],
upper_tag_dict[u_element][1], "up")
                                counter++
                            }
                            for (let l_element in lower_tag_dict){
```

```
                                fill_text(tagbox_values[counter],
lower_tag_dict[l_element][1], "down")
                                counter++
                            }
                        }
                        else{
                            manager.clear_all_arrays()
                        }
                    }

                    function fill_text(value, tagbox_n, side){
                        var upper_input_boxes = {"0":tagbox_1.combobox_input,
"1":tagbox_2.combobox_input, "2":tagbox_3.combobox_input,
"3":tagbox_4.combobox_input}
                        var lower_input_boxes = {"4":tagbox2_1.combobox_input,
"5":tagbox2_2.combobox_input, "6":tagbox2_3.combobox_input,
"7":tagbox2_4.combobox_input}
                        if (side=="up"){

upper_input_boxes[tagbox_n.toString()].placeholderText = value
                        }
                        else if(side=="down"){

lower_input_boxes[tagbox_n.toString()].placeholderText = value
                        }
                    }

                }
            }
        }
//################################################### Pestaña de control
#####################################################################

        Item {
            id: control_tab

            Rectangle {
                id: control_zone

                width: login_dialog.width - 50
                height: login_dialog.height - 80
                border.width: 1

                function get_actual_position(){
                    eje_x.paramText = 0
                    eje_y.paramText = 0
                    eje_z.paramText = 0
                }

                function get_gains(){
                    var gains = manager.get_gains()
                    kff_a.placeholder = gains[0].toString()
                    kff_v.placeholder = gains[1].toString()
                    kp_v.placeholder = gains[2].toString()
                    ki_v.placeholder = gains[3].toString()
                    kp_p.placeholder = gains[4].toString()
                    ki_p.placeholder = gains[5].toString()
                }
```

```
                Text{
                    id: control_title

                    text: "Control de movimiento del efector";
                    font.bold: true;
                    font.pointSize: 16;
                    font.pixelSize: 20;
                    anchors.top: parent.top;
                    anchors.topMargin: 20;
                    anchors.left: parent.left;
                    anchors.leftMargin: 150;
                }

                Rectangle{
                    id: crossreference
                    width: 80
                    height: 80
                    anchors.verticalCenter: zreference.verticalCenter
                    anchors.left: zreference.right
                    anchors.leftMargin: 150
                }

                ArrowButtons{id: right; anchors.left: crossreference.right;
anchors.top: crossreference.top; rotation: 0; transform: Rotation{origin.x: 0;
origin.y: 0; angle: 0}
                            highButton.onClicked: {eje_x.paramText =
parseInt(eje_x.paramText) + 100}
                            lowButton.onClicked: {eje_x.paramText =
parseInt(eje_x.paramText) + 10}}
                ArrowButtons{id: upup; anchors.left: crossreference.left;
anchors.top: crossreference.top; rotation: 270; transform: Rotation{origin.x: 0;
origin.y: 0; angle: 270}
                            highButton.onClicked: {eje_y.paramText =
parseInt(eje_y.paramText) + 100}
                            lowButton.onClicked: {eje_y.paramText =
parseInt(eje_y.paramText) + 10}}
                ArrowButtons{id: down; anchors.top: crossreference.bottom;
anchors.left: crossreference.right; rotation: 90; transform: Rotation{origin.x:
0; origin.y: 0; angle: 90}
                            highButton.onClicked: {eje_y.paramText =
parseInt(eje_y.paramText) - 100}
                            lowButton.onClicked: {eje_y.paramText =
parseInt(eje_y.paramText) - 10}}
                ArrowButtons{id: left;  anchors.top: crossreference.bottom ;
anchors.left: crossreference.left; rotation: 180; transform: Rotation{origin.x:
0; origin.y: 0; angle: 180}
                            highButton.onClicked: {eje_x.paramText =
parseInt(eje_x.paramText) - 100}
                            lowButton.onClicked: {eje_x.paramText =
parseInt(eje_x.paramText) - 10}}

                Rectangle{
                    id: zreference
                    width: 80
                    height: 80
                    anchors.left: parent.left
                    anchors.leftMargin: 100
```

```
                    anchors.verticalCenter: parent.verticalCenter
                }

                ArrowButtons{id: zup; anchors.left: zreference.left;
anchors.top: zreference.top; rotation: 270; transform: Rotation{origin.x: 0;
origin.y: 0; angle: 270}
                             highButton.onClicked: {eje_z.paramText =
parseInt(eje_z.paramText) + 100}
                             lowButton.onClicked: {eje_z.paramText =
parseInt(eje_z.paramText) + 10}}
                ArrowButtons{id: zdown; anchors.top: zreference.bottom;
anchors.left: zreference.right; rotation: 90; transform: Rotation{origin.x: 0;
origin.y: 0; angle: 90}
                             highButton.onClicked: {eje_z.paramText =
parseInt(eje_z.paramText) - 100}
                             lowButton.onClicked: {eje_z.paramText =
parseInt(eje_z.paramText) - 10}}
                Image {
                    id: doublearrow
                    width: 300
                    height: 350
                    anchors.right: zreference.left
                    anchors.rightMargin: -100
                    anchors.verticalCenter: zreference.verticalCenter
                    source: "./images/2wayarrow.png"
                    Text {
                        id: toptext
                        text: qsTr("500")
                        font.bold: true
                        font.pointSize: 16
                        anchors.horizontalCenter: parent.horizontalCenter
                        anchors.bottom: parent.top
                        anchors.bottomMargin: -30
                    }
                    Text {
                        id: bottomtext
                        text: qsTr("0")
                        font.bold: true
                        font.pointSize: 16
                        anchors.horizontalCenter: parent.horizontalCenter
                        anchors.top: parent.bottom
                        anchors.topMargin: -30
                    }
                    Text {
                        text: qsTr("Z")
                        font.bold: true
                        font.pointSize: 16
                        anchors.verticalCenter: parent.verticalCenter
                        anchors.left: bottomtext.left
                        anchors.leftMargin: -15
                    }
                }
                Image {
                    id: y_arrow
                    source: "./images/2wayarrow.png"
                    anchors.centerIn: crossreference
                    width: 90
                    height: 90
```

```
                    Text {
                        id: y_label
                        text: qsTr("Y")
                        anchors.horizontalCenter: parent.horizontalCenter
                        anchors.horizontalCenterOffset: 10
                        anchors.top: parent.top
                    }
                }
                Image {
                    id: x_arrow
                    source: "./images/2wayarrow.png"
                    anchors.centerIn: crossreference
                    transform: Rotation{origin.x: x_arrow.width/2; origin.y:
x_arrow.height/2; angle: 90}
                    width:90
                    height: 90
                    Text {
                        id: x_label
                        text: qsTr("X")
                        anchors.horizontalCenter: parent.horizontalCenter
                        anchors.horizontalCenterOffset: 10
                        anchors.top: parent.top
                        transform: Rotation{origin.x: x_label.width/2; origin.y:
x_label.height/2; angle: -90}
                    }
                }

                RowLayout{
                    spacing: 30
                    anchors.horizontalCenter: max_speed.horizontalCenter
                    anchors.bottom: max_speed.top
                    anchors.bottomMargin: 10
                    ParamArea{id: eje_x; name: "Eje X"; paramText: "0"; help: "
Indique un valor entre -" + ws_radio + " y " + ws_radio; helpSide: "above";
input.width: 100; input.onTextChanged:  {if (parseInt(eje_x.paramText) >
parseInt(ws_radio)){eje_x.paramText = parseInt(ws_radio)} if
(parseInt(eje_x.paramText) < -parseInt(ws_radio)){eje_x.paramText = -
parseInt(ws_radio)}}}
                    ParamArea{id: eje_y; name: "Eje Y"; paramText: "0"; help: "
Indique un valor entre -" + ws_radio + " y " + ws_radio; helpSide: "above";
input.width: 100; input.onTextChanged:  {if (parseInt(eje_y.paramText) >
parseInt(ws_radio)){eje_y.paramText = parseInt(ws_radio)} if
(parseInt(eje_y.paramText) < -parseInt(ws_radio)){eje_y.paramText = -
parseInt(ws_radio)}}}
                    ParamArea{id: eje_z; name: "Eje Z"; paramText: "0"; help: "
Indique un valor entre 0 y " + ws_altura; helpSide: "above"; input.width: 100;
input.onTextChanged:  {if (parseInt(eje_z.paramText) >
parseInt(ws_altura)){eje_z.paramText = parseInt(ws_altura)} if
(parseInt(eje_z.paramText) < 0){eje_z.paramText = 0}}}
                }

                ParamArea{id: max_speed; name: "Veloc. de efector"; paramText:
"100"; placeholder: qsTr("Dimension en m/s"); anchors.horizontalCenter:
make_move.horizontalCenter; anchors.bottom: make_move.top; anchors.bottomMargin:
10}

                Button{
                    id: make_move
```

```
                    width: 250
                    height: 50
                    anchors.horizontalCenter: control_title.horizontalCenter
                    anchors.bottom: parent.bottom
                    anchors.bottomMargin: 10
                    Text {
                        text: qsTr("Realizar movimiento")
                        color: make_move.enabled ? "black":"darkgrey"
                        anchors.right: parent.right
                        anchors.rightMargin: 10
                        anchors.verticalCenter: parent.verticalCenter
                        font.pointSize: 14
                        font.bold: true
                    }
                    Image {
                        source: "./images/move.png"
                        opacity: enabled ? 1 : 0.4
                        anchors.left: parent.left
                        anchors.leftMargin: 10
                        anchors.verticalCenter: parent.verticalCenter
                        height: 32
                        width: 32
                    }
                    onClicked: {
                        manager.write_value("jerk_speed", max_speed.paramText)
                        manager.move_to(eje_x.paramText, eje_y.paramText, -
eje_z.paramText)
                    }
                }

                Rectangle{
                    id: utilities
                    width: childrenRect.width + 50
                    height: childrenRect.height + 20
                    anchors.right: parent.right
                    anchors.rightMargin: 30
                    anchors.verticalCenter: parent.verticalCenter
                    border.width: 1
                    color: "whitesmoke"

                    Text {
                        id: utilities_title
                        text: "Funciones utiles";
                        font.bold: true;
                        font.pointSize: 16;
                        font.pixelSize: 20;
                        anchors.top: parent.top;
                        anchors.topMargin: 10
                        anchors.horizontalCenter: parent.horizontalCenter
                    }

                    ColumnLayout{
                        spacing: 15
                        anchors.top: utilities_title.bottom
                        anchors.topMargin: 30
                        anchors.horizontalCenter: parent.horizontalCenter
```

```
                        Button{
                            id: move_home

                            Layout.preferredWidth: 250
                            Layout.preferredHeight: 50
                            Layout.alignment: Qt.AlignHCenter
                            Text {
                                text: qsTr("Volver al origen")
                                color: move_home.enabled ? "black":"darkgrey"
                                anchors.right: parent.right
                                anchors.rightMargin: 30
                                anchors.verticalCenter: parent.verticalCenter
                                font.pointSize: 14
                                font.bold: true
                            }
                            Image {
                                source: "./images/home.png"
                                opacity: enabled ? 1 : 0.4
                                anchors.left: parent.left
                                anchors.leftMargin: 10
                                anchors.verticalCenter: parent.verticalCenter
                                height: 32
                                width: 32
                            }
                            onClicked: {
                                manager.run_home()
                            }
                        }
                        Button{
                            id: stop_printing

                            Layout.preferredWidth: 250
                            Layout.preferredHeight: 50
                            Layout.alignment: Qt.AlignHCenter
                            Text {
                                text: qsTr("Detener impresion")
                                color: stop_printing.enabled ?
"black":"darkgrey"
                                anchors.right: parent.right
                                anchors.rightMargin: 10
                                anchors.verticalCenter: parent.verticalCenter
                                font.pointSize: 14
                                font.bold: true
                            }
                            Image {
                                source: "./images/warning.png"
                                opacity: enabled ? 1 : 0.4
                                anchors.left: parent.left
                                anchors.leftMargin: 10
                                anchors.verticalCenter: parent.verticalCenter
                                height: 32
                                width: 32
                            }
                            onClicked: {
                                message.text = qsTr("¿Está seguro que desea
detener la impresión? \n Se perderá todo el progreso hasta ahora")
                                double_confirmation_window.visible = true
                            }
```

```
                        }

                        SpinBox{
                            id: speed_tune_button

                            Layout.preferredWidth: 140
                            Layout.preferredHeight: 25
                            Layout.alignment: Qt.AlignRight
                            to: 200
                            from: 1
                            value: 100
                            textFromValue: function(value){return
value.toString()+'%'}
                            onValueModified: {console.log("valor cambiado a " +
value); manager.tune_speed(value)}

                            Text{
                                id: spinbox_title

                                text: qsTr('Control de\n velocidad')
                                anchors.verticalCenter:
speed_tune_button.verticalCenter
                                anchors.right: speed_tune_button.left
                                anchors.rightMargin: 20
                            }
                        }

                        ParamArea{id: kff_a; name: "Kff aceleración";
placeholder: qsTr("ingrese un valor"); help: "Ganancia de pre-alimentación de
acceleración"; helpSide: "left";
                                  Layout.alignment: Qt.AlignRight;
input.onAccepted: {manager.tune_gain("SV_accelFFgain", "sw_accelFFGain",
kff_a.paramText)}}
                        ParamArea{id: kff_v; name: "Kff velocidad"; placeholder:
qsTr("ingrese un valor"); help: "Ganancia de pre-alimentación de velocidad";
helpSide: "left";
                                  Layout.alignment: Qt.AlignRight;
input.onAccepted: {manager.tune_gain("SV_velocFFgain", "sw_velocFFGain",
kff_v.paramText)}}
                        ParamArea{id: kp_p; name: "Kp posición"; placeholder:
qsTr("ingrese un valor"); help: "Ganancia proporcional de posición"; helpSide:
"left"; Layout.alignment: Qt.AlignRight;
                                  input.onAccepted:
{manager.tune_gain("SV_posPropgain", "sw_posPropGain", kp_p.paramText)}}
                        ParamArea{id: ki_p; name: "Ki posición"; placeholder:
qsTr("ingrese un valor"); help: "Ganancia integral de posición"; helpSide:
"left"; Layout.alignment: Qt.AlignRight;
                                  input.onAccepted:
{manager.tune_gain("SV_posItggain", "sw_posItgGain", ki_p.paramText)}}
                        ParamArea{id: kp_v; name: "Kp velocidad"; placeholder:
qsTr("ingrese un valor"); help: "Ganancia proporcional de velocidad"; helpSide:
"left"; Layout.alignment: Qt.AlignRight;
                                  input.onAccepted:
{manager.tune_gain("SV_velocPropgain", "sw_velocPropGain", kp_v.paramText)}}
                        ParamArea{id: ki_v; name: "Ki velocidad"; placeholder:
qsTr("ingrese un valor"); help: "Ganancia integral de velocidad"; helpSide:
"left"; Layout.alignment: Qt.AlignRight;
                                  input.onAccepted:
```

```
{manager.tune_gain("SV_velocItggain", "sw_velocItgGain", ki_v.paramText)}}
                    }
                }
            }
        }
    }
}
```

C.9. PluginUDEC.py

```
import math
import multiprocessing
import os
import os.path
import platform
from datetime import datetime, timedelta
from functools import reduce
from multiprocessing import Lock
from threading import Thread
from typing import List, cast

import numpy as np
from PyQt6.QtCore import pyqtSlot, QObject, pyqtSignal, QTimer
from UM.Application import Application
from UM.Extension import Extension
from UM.Logger import Logger
from UM.PluginRegistry import PluginRegistry
from UM.i18n import i18nCatalog
from cura.CuraApplication import CuraApplication

from .pycomm3.pycomm3 import LogixDriver
from .resources import MatplotlibImageProvider as matplt

i18n_catalog = i18nCatalog("PluginUDEC")  # Translates to a language Cura can
read

def threaded(fn):
    def wrapper(*args, **kwargs):
        Thread(target=fn, args=args, kwargs=kwargs).start()
    return wrapper

class PluginUDEC(QObject, Extension):
    progress_changed = pyqtSignal(float, name="progressChanged")
    file_changed = pyqtSignal(str, name="fileChanged")
    progress_total_changed = pyqtSignal(float, name="progressTotalChanged")
    progress_end = pyqtSignal(name="progressEnd")
    connection_achieved = pyqtSignal(name="connectionAchieved")
    lock = Lock()
    end_flag = multiprocessing.Value('b', True)
    message_flag = multiprocessing.Value('b', False)

    def __init__(self) -> None:
        super().__init__()
        QObject.__init__(self)
        Extension.__init__(self)
```

```
        self.setMenuName(i18n_catalog.i18nc("@item:inmenu", "Plugin UDEC"))
        self.addMenuItem(i18n_catalog.i18nc("@item:inmenu", "Generar archivo
RSLogix"), self.show_popup)
        self.addMenuItem(i18n_catalog.i18nc("@item:inmenu", "Conectar
impresora"), self.show_connect)
        self._view = None
        self.connect_view = None
        self.message_view = None
        self.positions_list = []
        self.saved_values = []
        self.tag_dict = {}
        self.ip = ""
        self.loading_is_open = False
        self.plc = LogixDriver('152.74.22.162/3', init_program_tags=False)
        self.plot_value_arrays = [[],[],[],[],[],[],[],[]]
        self.figure_array = [[],[],[]]

    def figure_plot(self, name, values):
        """Recieves a list of 3 arrays (x,y,z) to plot on 3 dimensions. Stores
the image with the given name."""

        gain = 1.4
        figure = self.imageProvider.addFigure(name, figsize=(6.4*gain,4.8*gain))
        ax = figure.add_subplot(projection='3d')
        ax.grid(linewidth=2)
        ax.set(aspect='auto')
        ax.set_xlabel('X', fontsize=30)
        ax.set_ylabel('Y', fontsize=30)
        ax.set_zlabel('Z', fontsize=30)
        ax.set_xlim(-225,225)
        ax.set_ylim(-225,225)
        ax.set_zlim(0,505)
        ax.set_title("Figura en proceso", fontsize=30)
        ax.view_init(elev=20., azim=-35, roll=0)
        figure.subplots_adjust(left=0, bottom=0, right=1, top=1, wspace=0,
hspace=0)

        ax.plot(np.array(values[0]),np.array(values[1]),np.array(values[2]))

    @pyqtSlot()
    def get_figure_progress(self):

        if not self.plc.connected:
            self.plc.open()
        x_value = self.plc.read('Program:MainProgram.X').value
        y_value = self.plc.read('Program:MainProgram.Y').value
        z_value = self.plc.read('Program:MainProgram.Z').value
        self.figure_array[0].append(x_value)
        self.figure_array[1].append(y_value)
        self.figure_array[2].append(z_value)
        self.figure_plot('figure',self.figure_array)

    def plot(self, name, value_arrays):
        self.imageProvider = matplt.MatplotlibImageProvider()
        gain = 1.4
        figure = self.imageProvider.addFigure(name, figsize=(6.4*gain,4.8*gain))
        ax = figure.add_subplot()
```

```
        ax.grid(linewidth=2)
        ax.set(aspect='auto', ylim=[0,100], xlim=[self.bias_time(datetime.now(),
0, -1, 0), datetime.now()])
        ax.set_xlabel('Hora', fontsize=30)
        ax.set_ylabel('Valor', fontsize=30)
        ax.set_title("Visualización de variables", fontsize=30)
        ax.tick_params(axis='both', which='both', labelsize=15)

        for tag in range(len(value_arrays)):
            n_of_values = len(value_arrays[tag])
            if len(value_arrays[tag]) < 60:
                X = np.array([self.bias_time(datetime.now(), 0, 0, -n_of_values
+ i) for i in range(n_of_values)])
            else:
                X = np.array([self.bias_time(datetime.now(), 0, 0, -60 + i) for
i in range(60)])
            Y = np.array(value_arrays[tag])
            ax.plot(X,Y, linewidth=3, label='tag'+str(tag))
        ax.legend(fontsize='xx-large', loc='upper left')

    def update_plots(self, values_list, tag_spot, upper_len, lower_len):
        """Grab values and put them in independant arrays. Separate arrays
between upper and lower plot. Call self.plot to create the figure assossiated
with the values."""

        values = self.get_value_arrays(values_list, tag_spot)
        upper_value_arrays = []
        lower_value_arrays = []
        for i in range(upper_len):
            upper_value_arrays.append(values[i])
        for i in range(lower_len):
            lower_value_arrays.append(values[i + upper_len])
        self.plot('plot_one', upper_value_arrays)
        self.plot('plot_two', lower_value_arrays)
        return

    def get_value_arrays(self, values, tag_spot):

        _values = []
        for i in range(len(values)):
            self.plot_value_arrays[tag_spot[i]].append(values[i])
            if len(self.plot_value_arrays[tag_spot[i]]) > 60:
                self.plot_value_arrays[tag_spot[i]].pop(0)
            _values.append(self.plot_value_arrays[tag_spot[i]])
        for element in self.plot_value_arrays:
            if self.plot_value_arrays.index(element) not in tag_spot:
                element.clear()
        return _values

    @pyqtSlot(list, list,  int, int, result=list)
    def update_series(self, tag_list, tag_spot, upper_len, lower_len) ->
List[float]:
        """Updates both 2D plots with the values of the given tag list. Returns
the
        read values to be displayed to the user"""

        try:
            tag_names = self.extract_names(tag_list)
```

```
            n_tags = len(tag_names)
            if not self.plc.connected:
                self.plc.open()
            tag_read = self.plc.read(*tag_names)
            values = self.extract_values(tag_read, n_tags)
            self.saved_values = values
            if self.loading_is_open:
                self.connection_achieved.emit()
                self.loading_is_open = False
        except:
            if not self.loading_is_open:
                self.set_message_params('r', 'Se produjo un error',
                                        'Se ha perdido la conexion con la
impresora. '
                                        '\nReconectando...')
                self.progress_end.emit()
            self.loading_is_open = True
            values = self.saved_values
        self.update_plots(values, tag_spot, upper_len, lower_len)
        return values

    def bias_time(self, original_time, hr_bias, min_bias, sec_bias):
        """Recieves a datetime-time object and modifies it with the given bias
for hr, min and sec."""

        new_time = original_time + timedelta (hours=hr_bias) + timedelta
(minutes=min_bias) + timedelta (seconds=sec_bias)
        return new_time

    def shift_elements(self, arr, num, fill_value):
        result = np.empty_like(arr)
        if num > 0:
            result[:num] = fill_value
            result[num:] = arr[:-num]
        elif num < 0:
            result[num:] = fill_value
            result[:num] = arr[-num:]
        else:
            result[:] = arr
        return result

    @pyqtSlot()
    def clear_all_arrays(self):

        for element in self.plot_value_arrays:
            element.clear()

    @pyqtSlot(int)
    def clear_series(self, index):

        self.plot_value_arrays[index].clear()

    @pyqtSlot(str, result=list)
    def plc_tag_list(self, ip) -> List[str]:
        """Gets the list of program tagsand returns a list containing the names
of all tags
        with less than 100 elements"""
```

```
        self.ip = ip
        plc = LogixDriver(ip, init_program_tags=True)
        plc.open()
        tag_list = plc.get_tag_list('Program:MainProgram').copy()
        tag_name_list = self.get_names(self.get_tags_info(tag_list))
        plc.close()
        return tag_name_list

    def get_names(self, tag_list) -> List[str]:
        tag_name_list = []
        element_number = 0
        tags_info_dict = tag_list

        for name in tags_info_dict:
            tag_elements_name =
self.get_tag_elements(name.replace("Program:MainProgram.",''),
tags_info_dict[name][1])
            for element in tag_elements_name:
                tag_name_list.append(element)
                self.tag_dict[element] = [element_number,
tags_info_dict[name][0], tags_info_dict[name][2]]
                element_number += 1
            element_number = 0
        return tag_name_list

    def get_tags_info(self, tag_list) -> dict:
        tags_info_list = reduce(self.fetch_info, tag_list, dict())
        return tags_info_list

    def fetch_info(self, acc_dict, cur_dict) -> dict:
        tag_name =
cur_dict["tag_name"].replace("Program:Program:MainProgram.","Program:MainProgram
.")
        tag_dim = cur_dict["dimensions"]
        tag_n_dim = cur_dict["dim"]
        tag_total_elements = self.count_elements(tag_dim)
        if tag_total_elements < 100:
            acc_dict[tag_name] = [tag_total_elements, tag_dim, tag_n_dim]
        return acc_dict

    def count_elements(self, dimensions) -> float:
        total_elements = 1
        dim_copy = dimensions.copy()
        for axis_value in dim_copy:
            if axis_value == 0:
                dimensions.remove(0)
            else:
                total_elements *= axis_value
        return total_elements

    def get_tag_elements(self, tag_name, tag_dim) -> List[str]:
        length = len(tag_dim)
        tag_list = []
        if length == 0:
            tag_list.append(tag_name)
        elif length == 1:
            for i in range(tag_dim[0]):
                tag_list.append(tag_name+"["+str(i)+"]")
```

```
        elif length == 2:
            for i in range(tag_dim[0]):
                for j in range(tag_dim[1]):
                    tag_list.append(tag_name+"["+str(i)+","+str(j)+"]")
        elif length == 3:
            for i in range(tag_dim[0]):
                for j in range(tag_dim[1]):
                    for k in range(tag_dim[2]):

tag_list.append(tag_name+"["+str(i)+"]"+"["+str(j)+"]"+"["+str(k)+"]")
        return tag_list

    def extract_names(self, tag_list) -> List:
        tag_names = []
        for tag in tag_list:
            tag_name = tag.split('[')[0]
            tag_dim = self.tag_dict[tag][1]
            tag_n_dim = self.tag_dict[tag][2]
            if tag_dim == 1:
                tag_names.append('Program:MainProgram.' + tag_name)
            elif tag_n_dim == 2:
                tag_names.append('Program:MainProgram.' + tag)
            else:
                tag_element = self.tag_dict[tag][0]
                tag_names.append('Program:MainProgram.' + tag_name +
"["+str(tag_element)+"]")
        return tag_names

    def extract_values(self, tag_list, number_of_elements) -> List:
        values = []
        if number_of_elements > 1:
            for element in tag_list:
                tag_value = element.value
                if tag_value is True:
                    tag_value = 1
                elif tag_value is False:
                    tag_value = 0
                values.append(tag_value)
        else:
            tag_value = tag_list.value
            if tag_value is True:
                tag_value = 1
            elif tag_value is False:
                tag_value = 0
            values.append(tag_value)
        return values

    @pyqtSlot(str)
    def save_value(self, ip):
        with LogixDriver(ip) as plc:
            self.new_value =
plc.read('Program:MainProgram.array_tag{3}').value[0]

    def show_connect(self):
        """Displays an error message with the given title and message"""
        self.plot('plot_one',[])
        self.plot('plot_two',[])
        self.figure_plot('figure', [[],[],[]])
```

```
        self.create_view("MainWindow.qml")
        if self.connect_view is None:
            Logger.log("e", "Not creating MainWindow window since the QML
component failed to be created.")
            return
        self.connect_view.show()

    @pyqtSlot(result=list)
    def plc_info(self) -> List[str]:
        try:
            if not self.plc.connected:
                self.plc.open()
            is_connected = self.plc.connected
            product_name = self.plc.info["product_name"]
            name = self.plc.info["name"]
            programs = ""
            for program in list(self.plc.info["programs"].keys()):
                programs += str(program)
            return [product_name, name, programs, is_connected]
        except:
            self.set_message_params('e', 'Se produjo un error',
                                    'Se ha producido un error de conexion. '
                                    'Por favor revise que la direccion IP sea '
                                    'la correcta.')
            self.progress_end.emit()
            return ["----------", "----------", "----------", False]

    @pyqtSlot()
    def send_instructions(self):
        n_instructions = len(self.positions_list)
        if not self.plc.connected:
            self.plc.open()
        is_printing = self.plc.read('Program:MainProgram.sw_beginapp').value
        if self.plc.connected and not is_printing:

self.plc.write('Program:MainProgram.Matriz_L{'+str(n_instructions)+'}',
                       self.positions_list)
            self.plc.write('Program:MainProgram.total_coordinates',
self.total_coordinates)
            self.set_message_params('i', 'Operacion finalizada',
                                    'Las instrucciones fueron enviadas a la '
                                    'impresora. Puede monitorear el proceso en '
                                    'las pantallas adyacentes.')
        else:
            self.set_message_params('e', 'Operacion cancelada',
                                    'La impresora se encuentra trabajando. '
                                    'Detenga la impresion en la pestaña
"Control" '
                                    'o espere a que finalice.')
        self.progress_end.emit()

    @pyqtSlot(float, float, float, float, float, float)
    def generate_instructions_list(self, sb, sp, arm_length, height, ws_radio,
ws_height):
        """Responsible for using the given parameters to call the functions
which calculate the instructions."""

        params = [sb, sp, arm_length, height, ws_radio, ws_height]
```

```
        self.params = params
        if not self.are_valid(params):
            Logger.log("e", "Some parameters ")
            self.progress_end.emit()
            return
        coordinates = self.get_coordinates(self.split_lines(self.get_gcode()))
        self.total_coordinates = len(coordinates)
        if self.fits_in_ws(ws_radio, ws_height, coordinates):
            ws_coordinates = self.z_bias(coordinates, float(height),
float(ws_height))
            try:
                self.inv_kin_problem(ws_coordinates, params)
                self.positions_list = self.flatten(self.positions_list)
            except ValueError:
                self.set_message_params('e', 'Se produjo un error',
                                        'Se ha producido un error de calculo. '
                                        '0Por favor revise que los datos de '
                                        'impresora esten correctos.')
                self.progress_end.emit()
                return
            self.set_message_params('i', 'Operacion finalizada',
                                    'Finalizo la generacion de instrucciones. '
                                    'Puede enviarlas a la impresora una vez '
                                    'realizada la conexion.')
            self.progress_end.emit()


    def flatten(self, list) -> List:
        flattened_list = [item for sublist in list for item in sublist]
        return flattened_list

    @pyqtSlot(result=bool)
    def look_for_gcode(self):
        scene = Application.getInstance().getController().getScene()
        if not hasattr(scene, "gcode_dict"):
            Logger.warning("no gcode in system")
            return False
        return True

    def get_list_names(self, tag_list) -> List[str]:
        tag_names = []
        for tag in tag_list:
            tag_name = tag.split('[')[0]
            tag_dim = self.tag_dict[tag][1]
            tag_n_dim = self.tag_dict[tag][2]
            if tag_dim == 1:
                tag_names.append('Program:MainProgram.' + tag_name)
            elif tag_n_dim == 2:
                tag_names.append('Program:MainProgram.' + tag)
            else:
                tag_element = self.tag_dict[tag][0]
                tag_names.append('Program:MainProgram.' + tag_name +
"["+str(tag_element)+"]")

    def extract_tag_name(self, tag) -> str:
        tag_name = tag.split('[')[0]
        tag_dim = self.tag_dict[tag][1]
        tag_n_dim = self.tag_dict[tag][2]
```

```
        if tag_dim == 1:
            extracted_name = 'Program:MainProgram.' + tag_name
        elif tag_n_dim == 2:
            extracted_name = 'Program:MainProgram.' + tag
        else:
            tag_element = self.tag_dict[tag][0]
            extracted_name = 'Program:MainProgram.' + tag_name +
"["+str(tag_element)+"]"
        return extracted_name

    @pyqtSlot(str, float)
    def write_value(self, tag_name, new_value):
        tag_valid_name = self.extract_tag_name(tag_name)
        if not self.plc.connected:
            self.plc.open()
        self.plc.write(tag_valid_name, new_value)

    @pyqtSlot(str, str, float)
    def tune_gain(self, tag, switch, gain_value):
        self.write_value(tag, gain_value)
        self.write_value(switch, 1)

    @pyqtSlot(result=list)
    def get_progress_percentage(self):
        try:
            step = self.plc.read('i').value
            self.prev_step = step
            if self.loading_is_open:
                self.connection_achieved.emit()
                self.loading_is_open = False
        except:
            if not self.loading_is_open:
                self.set_message_params('r', 'Se produjo un error',
                                        'Se ha perdido la conexion con la
impresora. '
                                        '\nReconectando...')
                self.progress_end.emit()
            self.loading_is_open = True
            step = self.prev_step
        percentage = 100*step/self.total_coordinates
        if percentage >= 100:
            self.set_message_params('i', 'Fin del proceso',
                                    'Ha finalizado la impresion de la figura. ')
            self.progress_end.emit()
        progress = [int(percentage), step]
        return progress

    @pyqtSlot(result=bool)
    def switch_printing(self):
        self.check_servos()
        if not self.plc.connected:
            self.plc.open()
        read_state = self.plc.read('Program:MainProgram.sw_beginapp').value
        if read_state:
            self.plc.write('Program:MainProgram.sw_beginapp', 0)
            return False
        else:
            self.plc.write('Program:MainProgram.sw_beginapp', 1)
```

```
            return True

    @pyqtSlot()
    def check_servos(self):
        if not self.plc.connected:
            self.plc.open()
        servos_are_active = self.plc.read('Actuador_B1.ServoActionStatus').value
        printing_done =
self.plc.read('Program:MainProgram.sw_is_printing_done').value
        if not servos_are_active:
            self.plc.write('Program:MainProgram.sw_start_servos', 1)
            self.plc.write('Program:MainProgram.sw_init_var', 1)
        if printing_done and servos_are_active:
            self.plc.write('Program:MainProgram.sw_init_var', 1)

    @pyqtSlot(result=str)
    def get_n_coor(self):
        return str(self.total_coordinates)

    @pyqtSlot(float, float, float)
    def move_to(self, x_pos, y_pos, z_pos):
        x = x_pos
        y = y_pos
        z = z_pos - self.params[3] + self.params[5]
        servo_pos = self.inv_kin_problem([[x, y, z, 6000]], self.params)
        is_printing = self.plc.read('Program:MainProgram.sw_beginapp').value
        is_homing = self.plc.read('sw_startposition').value
        if self.plc.connected and not is_printing and not is_homing:
            self.plc.write('Program:MainProgram.coor_move_array{3}',
self.flatten(servo_pos))
            self.plc.write('Program:MainProgram.sw_coor_move', 1)
        else:
            self.set_message_params('e', 'Operacion cancelada',
                                    'La impresora se encuentra trabajando. '
                                    'Detenga la impresion en la pestaña
"Control" '
                                    'o espere a que finalice.')
            self.progress_end.emit()

    @pyqtSlot()
    def run_home(self):
        with LogixDriver(self.ip, init_program_tags=False) as plc:
            is_printing = self.plc.read('Program:MainProgram.is_printing').value
            is_moving = plc.read('Program:MainProgram.sw_coor_move').value
            is_printer_busy = is_moving or is_printing
            if not is_printer_busy:
                plc.write('sw_startposition', 1)
            else:
                self.set_message_params('e', 'Operacion cancelada',
                                        'La impresora se encuentra trabajando. '
                                        'Detenga la impresion en la pestaña
"Control" '
                                        'o espere a que finalice el
movimiento.')
                self.progress_end.emit()

    @pyqtSlot()
    def stop_printing(self):
```

```
        if not self.plc.connected:
            self.plc.open()
        is_done_printing =
self.plc.read('Program:MainProgram.sw_is_printing_done').value
        is_homing = self.plc.read('sw_startposition').value
        step = self.plc.read('i').value
        if step != 0 and not is_done_printing and not is_homing:
            self.plc.write('Program:MainProgram.sw_stop_printing', 1)
        elif is_homing:
            self.set_message_params('e', 'Operacion cancelada',
                                    'La impresora se encuentra en proceso de
homing.')
            self.progress_end.emit()
        else:
            self.set_message_params('e', 'Operacion cancelada',
                                    'La impresora no se encuentra trabajando.')
            self.progress_end.emit()

    @pyqtSlot()
    def force_stop(self):
        if not self.plc.connected:
            self.plc.open()
        self.plc.write('Program:MainProgram.sw_off_servos', 1)

    @pyqtSlot(result=list)
    def get_gains(self):
        if not self.plc.connected:
            self.plc.open()
        gains_list = self.plc.read('Program:MainProgram.SV_gains{6}').value
        return gains_list

    @pyqtSlot(float)
    def tune_speed(self, quotient):
        self.write_value('move_speed_gain', quotient/100)


# ------------------------------------------------------------------------------
----------------------
# ------------------------------------------------------------------------------
----------------------

    @pyqtSlot(str)
    def start_worker(self, path):
        QTimer.singleShot(0, lambda: self.worker.run(path))

    @pyqtSlot(result=str)
    def showGcode(self) -> str:
        """Shows the available gcode (if any) on the corresponding zone."""

        gcode = self.get_gcode_bit()
        if len(gcode) < 20:
            return ''.join(gcode)
        gcode = self.split_lines(gcode)
        gcode_string = ' '.join(gcode)
        return gcode_string

    @threaded
    def generatePositionsAsync(self, sb, sp, arm_length, height, ws_radio,
```

```
ws_height, file_path,
                                overwrite, destination_path=None, dir_path=None)
-> None:
        """Responsible for using the given parameters to call the functions
which calculate the instructions and
        then write them in a L5K file."""

        params = [sb, sp, arm_length, height, ws_radio, ws_height]
        print(str(params))
        file_params = [file_path, overwrite, destination_path, dir_path]
        if self.are_valid(params, file_params) is False:
            self.progress_end.emit()
            self.end_flag.value = False
            return
        coordinates = self.get_coordinates(self.split_lines(self.get_gcode()))
        if self.fits_in_ws(ws_radio, ws_height, coordinates):
            ws_coordinates = self.z_bias(coordinates, float(height),
float(ws_height))
            try:
                self.inv_kin_problem(ws_coordinates, params)
            except ValueError:
                self.set_message_params('e', 'Se produjo un error', 'Se ha
producido un error de calculo. Por favor '
                                                                    'revise que
los datos de impresora esten '

'correctos.')
                self.progress_end.emit()
                self.end_flag.value = False
                return
            self.generateInstructions(file_path, overwrite, self.positions_list,
destination_path)
            self.end_flag.value = False
            self.set_message_params('i', 'Operacion finalizada', 'Se termino la
generacion de instrucciones. Las '
                                                                'instrucciones
fueron guardadas en el archivo '
                                                                'indicado.')
            self.progress_end.emit()

    @pyqtSlot(float, float, float, float, float, float, str, bool, str, str)
    def generatePositions(self, sb, sp, arm_length, height, ws_radio, ws_height,
file_path, overwrite,
                          destination_path=None, dir_path=None) -> None:
        """Called when the user presses the 'generate instructions' button.
Responsible for using the given parameters
         to call the functions which calculate the instructions and then write
them in a L5K file."""

        self.generatePositionsAsync(sb, sp, arm_length, height, ws_radio,
ws_height, file_path,
                                    overwrite, destination_path, dir_path)

    def are_valid(self, params, file_params=None) -> bool:
        print("EN PROCESO DE VALIDACION")
        if 0 in params:
            self.set_message_params('e', 'Faltan datos', 'Ingrese todos los
datos solicitados e intente otra vez.')
```

```
                return False
            if file_params is not None:
                if file_params[1]:
                    if not os.path.isfile(self.get_url(file_params[0])):
                        print("no existe archivo")
                        self.set_message_params('e', 'Archivo inexistente',
                                                'El archivo indicado no es
compatible o no existe.\nPor favor seleccione un '
                                                'archivo valido.')
                        return False
                if not file_params[1]:
                    if not os.path.isfile(self.get_url(file_params[0])):
                        self.set_message_params('e', 'Archivo inexistente',
                                                'El archivo indicado no es
compatible o no existe.\nPor favor seleccione un '
                                                'archivo valido.')
                        return False
                    elif os.path.isfile(self.get_url(file_params[2])):
                        self.set_message_params('e', 'Archivo ya existe',
                                                'El archivo ya existe.\nPor favor
escoja otro nombre.')
                        return False
                    elif not os.path.exists(self.get_url(file_params[3])):
                        self.set_message_params('e', 'Carpeta inexistente',
                                                'La carpeta indicada no existe.\nPor
favor seleccione un directorio valido.')
                        return False
            return True

        def fits_in_ws(self, radio, height, coordinates) -> bool:
            """Checks if the given coordinates fit in the working space"""

            for element in coordinates:
                if abs(element[0]) > radio or abs(element[1]) > radio or
abs(element[2]) > height:
                    print(element[0], radio, abs(element[1]), radio,
abs(element[2]), height)
                    self.set_message_params('e', 'Coordenadas incompatibles',
                                            'La figura revanada es mas grande que '
                                            'el espacio de trabajo disponible.')
                    self.progress_end.emit()
                    return False
            return True

        def z_bias(self, coordinates, height, ws_height) -> List[List[float]]:
            """Returns the given coordinates with an added bias on the Z axis. This
bias depends on the specified RDL
            height """

            for i in range(len(coordinates)):
                coordinates[i][2] *= -1
                coordinates[i][2] -= height - ws_height
            return coordinates

        def inv_kin_problem(self, coordinates, parameters):
            """Returns the positions of the motors for each given coordinate"""

            self.positions_list.clear()
```

```
        sb = parameters[0]
        sp = parameters[1]
        wb = (math.sqrt(3)/6)*sb
        wp = (math.sqrt(3)/6)*sp
        ub = (math.sqrt(3)/3)*sb
        up = (math.sqrt(3)/3)*sp
        l = parameters[2]

        a = (sb - sp) / 2
        b = wb - wp
        c = up - ub
        x = self.get_column(coordinates, 0)
        y = self.get_column(coordinates, 1)
        z = self.get_column(coordinates, 2)
        feed_rate = self.get_column(coordinates, 3)
        print(x, y, z)
        self.progress_total_changed.emit(len(coordinates))
        for i in range(len(coordinates)):
            C1 = x[i] ** 2 + y[i] ** 2 + z[i] ** 2 + a ** 2 + b ** 2 + 2 * a *
x[i] + 2 * b * y[i] - l ** 2
            C2 = x[i] ** 2 + y[i] ** 2 + z[i] ** 2 + a ** 2 + b ** 2 - 2 * a *
x[i] + 2 * b * y[i] - l ** 2
            C3 = x[i] ** 2 + y[i] ** 2 + z[i] ** 2 + c ** 2 + 2 * c * y[i] - l
** 2
            L1 = -z[i] - math.sqrt(z[i] ** 2 - C1)
            L2 = -z[i] - math.sqrt(z[i] ** 2 - C2)
            L3 = -z[i] - math.sqrt(z[i] ** 2 - C3)
            self.positions_list.append([round(L1, 3), round(L2, 3), round(L3,
3), feed_rate[i]/60])
            self.progress_changed.emit(i+1)
        new_list = self.positions_list
        return new_list

    def get_column(self, coordinates, column) -> List[float]:
        """Returns the specified column from the given List"""

        axis = []
        for i in range(len(coordinates)):
            axis.append(coordinates[i][column])
        return axis

    def split_lines(self, base_text) -> List[str]:
        """Splits the given text into lines and saves them onto a list. Returns
the list."""

        lines = [""]
        line_list = [""]
        for line in base_text:
            lines += line
            if self.match_inline(['\n'])(line):
                line_list.append(''.join(lines))
                lines = [""]
        return line_list

    def match_text(self, base_text, find) -> List[str]:
        """Reads a given text and stores its content up until a match is found.
Returns stored content."""
```

```
        content = [""]
        for line in base_text:
            content[0] += line
            if self.match_inline([find])(line):
                break
        return content

    def get_gcode_bit(self) -> List[str]:
        """Gets the available gcode (if any) from the Cura build plate."""

        scene = Application.getInstance().getController().getScene()
        if not hasattr(scene, "gcode_dict"):
            Logger.warning("no gcode in system")
            return ["No gcode in system"]
        gcode_dict = getattr(scene, "gcode_dict")
        if not gcode_dict:
            Logger.log("e", "no gcode in system")
            return ["No gcode in system"]
        active_build_plate_id =
CuraApplication.getInstance().getMultiBuildPlateModel().activeBuildPlate
        gcode_list = gcode_dict[active_build_plate_id]
        return str(gcode_list[1])

    def get_gcode(self) -> List[str]:
        """Gets the available gcode (if any) from the Cura build plate."""

        scene = Application.getInstance().getController().getScene()
        if not hasattr(scene, "gcode_dict"):
            Logger.log("e", "no gcode in system")
            return ["No gcode in system"]
        gcode_dict = getattr(scene, "gcode_dict")
        if not gcode_dict:
            Logger.log("e", "no gcode in system")
            return ["No gcode in system"]
        active_build_plate_id =
CuraApplication.getInstance().getMultiBuildPlateModel().activeBuildPlate
        gcode_list = gcode_dict[active_build_plate_id]
        gcodeList = [""]
        for i in range(len(gcode_list)):
            gcodeList += str(gcode_list[i])
        return gcodeList

    def get_coordinates(self, gcode) -> List[List[float]]:
        """Gets the XYZ coordinates and feed rate speed from the G code and
        stores them in a list of lists."""

        filtered_lines = list(filter(self.match_inline([' X', ' Y', ' Z']),
gcode))
        coordinates = []
        x_axis = 1
        y_axis = 1
        z_axis = 1
        feed_rate = 60
        for line in filtered_lines:
            words = line.split()
            if self.match_inline(['X'])(line):
                x_axis = self.get_values('X', words)
```

```
            if self.match_inline(['Y'])(line):
                y_axis = self.get_values('Y', words)
            if self.match_inline(['Z'])(line):
                z_axis = self.get_values('Z', words)
            if self.match_inline(['F'])(line):
                feed_rate = self.get_values('F', words)
            coordinates.append([x_axis, y_axis, z_axis, feed_rate])
        Logger.log('i', "The number of coordinates is " + str(len(coordinates)))
        return coordinates

    def get_values(self, axis, line) -> float:
        """Returns the values of the coordinates from the G code"""

        axis_values = list(filter(self.match_inline([axis]), line))
        axis_values = self.to_int(axis_values)
        return axis_values

    def to_int(self, word) -> float:
        """Returns the value in the given word as a float"""

        value = list(filter(self.is_int, list(word[0])))
        value = ''.join(value)
        if value is '':
            value = 100
        value = float(value)
        return value

    def is_int(self, element) -> bool:
        """Returns true if the element holds a number representing a component
of a coordinate"""

        if 'X' in element or 'Y' in element or 'Z' in element or 'F' in element:
            return False
        else:
            return True

    def get_url(self, url) -> str:
        """get the real URL for the selected file"""

        if platform.system() == 'Windows':
            fileUrl = self.get_os_path(url, '')
            return fileUrl
        fileUrl = self.get_os_path(url, '/')
        return fileUrl

    def get_os_path(self, url, separator) -> str:
        tmp = [separator]
        for i in range(len(url)):
            if i > 7:
                tmp[0] += str(url[i])
        file_path = ''.join(tmp)
        return file_path

    @pyqtSlot(str, result=str)
    def showFileContent(self, url) -> str:
        """Show the content of the selected file in the bottom right text
box."""
```

```
        file = open(self.get_url(url), mode='r', encoding='utf-8')
        content = file.read()
        file.close()
        return content

    def write_file(self, file_path, destination_path, positions, action) ->
None:
        """Copies the content of file_path in destination_path while adding
extra content"""

        file = open(self.get_url(file_path), mode='r', encoding='utf-8')
        file_read = file.readlines()
        text = self.get_matrix(positions)
        with open(self.get_url(destination_path), action, encoding='utf-8') as
file_write:
            for line in file_read:
                file_write.write(line)
                if self.match_inline(['\t\t\tTAG'])(line):
                    file_write.write(str(text) + '\n')
        file.close()

    def get_matrix(self, positions) -> str:
        """Returns the given positions formatted as a RSLogix 5000 matrix TAG"""

        L: str = '\t\t\t\tMatriz_L : REAL[' + str(
            len(positions)) + ',4](Description := "matriz de posiciones",' +
'\n'
        L += '\t\t\t\t            RADIX := Float) := ['
        for i in range(0, len(positions), 2):
            try:
                if i is not 0:
                    L += '                    ,'
                L += ''.join(str(positions[i][0])) + ',' +
''.join(str(positions[i][1])) + ',' + ''.join(
                    str(positions[i][2])) + ',' + ''.join(str(positions[i][3]))
+ ',' + ''.join(str(positions[i + 1][0])) + ',' + ''.join(
                    str(positions[i + 1][1])) + ',' + ''.join(str(positions[i +
1][2])) + ',' + ''.join(str(positions[i + 1][3])) + '\n'
            except IndexError:
                L += ''.join(str(positions[i][0])) + ',' +
''.join(str(positions[i][1])) + ',' + ''.join(
                    str(positions[i][2])) + '\n'
        L += '                ]' + ';'
        return L

    def generateInstructions(self, file_path, overwrite, positions,
destination_path=None) -> None:
        """Generates the instructions by either overwriting the base file or
creating a new one."""

        if overwrite:
            self.write_file(file_path, file_path, positions, 'w')
        if not overwrite:
            self.write_file(file_path, destination_path, positions, 'x')

    def match_inline(self, find):
        """Looks for a match between 2 strings"""
```

```
        def find_word(line) -> bool:

            for i in range(len(find)):
                if find[i] in line:
                    return True
            return False

        return find_word

    def create_view(self, view) -> None:
        """Creates the view to be used."""

        Logger.log("d", "Creating requested view.")
        path = os.path.join(cast(str,
PluginRegistry.getInstance().getPluginPath("UDECPlugin")), view)
        if view == "main.qml":
            self._view = CuraApplication.getInstance().createQmlComponent(path,
{"manager": self})
            print("-----------------------", self._view.contentItem)
            if self._view is None:
                Logger.log("e",
                           "Not creating Plugin UDEC window since the view
variable is empty.")
                return
            Logger.log("d", "Plugin UDEC view created.")

        if view == "MessageDialog.qml":
            print("SE ESTA CREANDO LA VENTANA DE MENSAJES")
            self.message_view =
CuraApplication.getInstance().createQmlComponent(path, {"manager": self})
            print("SE ASIGNO LA VENTANA A LA VARIABLE MESSAGE VIEW")
            if self.message_view is None:
                Logger.log("e",
                           "Not creating message window since the message view
variable is empty.")
                return
            Logger.log("d", "Message view created.")

        if view == "MainWindow.qml":

CuraApplication.getInstance()._qml_engine.addImageProvider('perflog',
self.imageProvider)
            self.connect_view =
CuraApplication.getInstance().createQmlComponent(path, {"manager": self})
            if self.connect_view is None:
                Logger.log("e",
                           "Not creating MainWindow window since the message
view variable is empty.")
                return
            Logger.log("d", "MainWindow view created.")

    def show_popup(self) -> None:
        """Show the GUI of the UDEC Plugin."""

        self.create_view("main.qml")
        if self._view is None:
            Logger.log("e", "Not creating Plugin UDEC window since the QML
component failed to be created.")
```

```
            return
        self._view.show()

    @pyqtSlot(result=str)
    def get_message_title(self) -> str:
        title = str(self.message_title)
        return title

    @pyqtSlot(result=str)
    def get_message_content(self) -> str:
        content = str(self.message_content)
        return content

    @pyqtSlot(result=str)
    def get_message_style(self) -> str:
        style = str(self.message_style)
        return style

    @pyqtSlot(str, str, str)
    def set_message_params(self, style, title, content) -> None:
        self.message_style = style
        self.message_title = title
        self.message_content = content

    @pyqtSlot()
    def show_message(self) -> None:
        """Displays an error message with the given title and message"""

        self.create_view("MessageDialog.qml")
        if self.message_view is None:
            Logger.log("e", "Not creating message window since the QML component
failed to be created.")
            return
        self.message_view.show()
```

UNIVERSIDAD DE CONCEPCION – FACULTAD DE INGENIERIA
RESUMEN DE MEMORIA DE TITULO

Departamento	**:** Departamento de Ingeniería Eléctrica
Carrera	: Ingeniería civil electrónica
Nombre del memorista	: Matías Nicolás López Barriga
Título de la memoria	**:** Puesta en marcha de sistema de diseño de piezas complejas para impresora 3D de concreto
Fecha de la presentación oral	: 16/06/2023
Profesor(es) guía	: Juan Pablo Segovia Vera
Profesor(es) revisor(es)	: Lautaro Salazar Silva -- Manuel Valenzuela Latorre
Concepto	**:**
Calificación	**:**

Resumen (máximo 200 palabras)

Este documento presenta el desarrollo de un software que optimiza el proceso de generación y entrega de instrucciones de movimiento a una impresora 3D de concreto tipo delta diseñada por exalumnos memoristas de la universidad de Concepción. Ellos establecieron que el proceso ocupara 4 softwares para modelar la figura a imprimir, generar instrucciones de impresión, traducirlas a un lenguaje entendido por el PLC y enviarlas a la memoria interna de éste. El nuevo software unifica las últimas tres etapas un sólo programa, agregando también una etapa de monitoreo del proceso y otra de control de movimientos y sintonización. De esta forma, el proceso completo se puede completar con sólo 2 programas.

Para la programación de la nueva interfaz gráfica, se utilizaron principalmente los lenguajes Python y QML con el fin de separar la lógica principal de traducción y envío de instrucciones (Python) del código correspondiente al sector gráfico (QML). Además, este software se desarrolló como plugin del programa rebanador Ultimaker Cura con el fin de que el proceso se pueda realizar en una sola plataforma.

Por último, se llevó a cabo un FAT del software en un ambiente real, con los equipos presentes en el laboratorio de control de procesos.