

UNIVERSIDAD MAYOR DE SAN ANDRÉS

FACULTAD DE INGENIERÍA

CARRERA DE INGENIERÍA ELECTRÓNICA



PROYECTO DE GRADO

Prototipo de Impresora 3D con el uso de botellas PET como material de impresión y adaptación de desecho electrónico (*e-waste*) para su construcción.

Autor: Paulo Roberto Loma Marconi

Asesor: Javier Sanabria García

DAM: Jorge Antonio Nava Amador

La Paz, Bolivia

2015

Abstract

Basic 3D Printer prototype implementation with FDM technique by the application of plastic PET bottles as printing material replacing the traditional PLA/ABS, e-waste adaptation for the extruder and complete structure construction. Own electronic and Firmware-Software implementation designs, for the PID heating Joule Effect temperature control, Step by Step motors control, general operation and monitoring, Gcode descriptor and 3D positioning algorithm.

Implementación de un prototipo básico de Impresora 3D con la técnica FDM aplicando botellas plásticas PET desechables como material de impresión reemplazando el tradicional PLA/ABS, adaptación de desecho electrónico (*e-waste*) para la construcción del extrusor y la estructura completa. Implementación de diseño propio de electrónica y Firmware-Software, para el control de temperatura PID de calentamiento por Efecto Joule, control de motores paso a paso (mPaP), operación y monitoreo general, algoritmo descriptor de Gcode y posicionamiento 3D.

Agradecimientos

Quisiera agradecer a la Carrera de Ingeniería Electrónica de la Facultad de Ingeniería de la Universidad Mayor de San Andrés, por todo el conocimiento adquirido a través de los docentes durante toda la carrera.

Un especial agradecimiento a mi Tutor; Ing. Javier Sanabria García, por su constante dedicación, consejos y guía durante todo el proceso de realización del presente proyecto.

Finalmente, agradecer los tribunales; Ing. Jorge León Gómez Moreno e Ing. Ruperto Aduviri Rodríguez, por su disponibilidad, atención y observaciones realizadas durante el proceso de calificación.

*Para mi madre, Silvia Marconi
y mi padre, Gerardo Loma.*

*"Era como un niño jugando a la orilla del mar,
recogiendo una piedra mas pulida que la otra,
cuando tenía un océano de conocimiento delante de mi"*

- Issac Newton

Contenido

Abstract	I
Agradecimientos	II
Contenido	VII
Lista de Figuras	IX
Lista de Tablas	X
Lista de Códigos fuente	XI
Lista de Algoritmos	XII
1. Introducción	1
1.1. Antecedentes	1
1.2. Definición del problema	3
1.3. Objetivo	3
1.4. Justificación	3
1.5. Alcances y limitaciones	4
2. Marco Teórico	5
2.1. Modelado por Deposición Fundida (FDM)	5
2.1.1. Alimentación del material	6
2.1.2. Licuefacción	7
2.1.3. Extrusión	7
2.1.4. Robot Cartesiano	8
2.1.5. Posicionamiento 3D	9
2.2. Electrónica	9
2.2.1. Microcontrolador (MCU) - Microchip Familia 18FXX5X	10
2.2.2. Protocolo de comunicación USB	10
2.2.3. Motores Paso a Paso (mPaP)	12
2.2.4. Calentamiento por Efecto Joule	14
2.2.5. Sensor de temperatura	14
2.3. Sistema Operativo en Tiempo Real (RTOS)	16
2.3.1. <i>kernel</i> RTOS	17

2.3.2. Algoritmo RTOS	17
2.4. Herramientas externas necesarias	18
2.4.1. Modelo CAD	18
2.4.2. <i>Slicer</i> → Gcode	19
2.4.3. <i>Viewer</i>	20
3. Desarrollo del Proyecto	21
3.1. Descripción general de funcionamiento	21
3.2. Requerimientos de construcción electromecánica	22
3.3. Construcción mecánica del extrusor y la estructura del prototipo	23
3.3.1. Alimentación del material	25
3.3.2. Calentamiento por Efecto Joule	25
3.3.3. Extrusión	25
3.3.4. Robot cartesiano	26
3.3.5. Posicionamiento 3D	28
3.4. Diseño electrónico	28
3.4.1. MCU PIC18F4550	28
3.4.2. Comunicación USB	30
3.4.3. Control mPaP	30
3.4.4. Calentamiento por Efecto Joule	31
3.4.5. Acondicionamiento de señal de Termocupla Tipo K	33
3.4.6. Modelo CAD y grabado PCB	36
3.5. Análisis matemático para el procesamiento digital de datos	36
3.5.1. Adquisición de datos por puerto ADC PIC18F4550	37
3.5.2. Envío/Recepción de 10 bits a través de 2 Bytes USB PIC18F4550	37
3.5.3. Controlador PID	38
3.5.4. Descriptor Gcode	40
3.6. Desarrollo de <i>Firmware</i>	42
3.6.1. Estructura general	42
3.6.2. Cabecera de configuración MCU PIC18F4550	42
3.6.3. Función principal: main	43
3.6.4. Función genérica: Inicio	44
3.6.5. Función genérica: ADC	45
3.6.6. Tarea RTOS: Comunicación USB/descriptor Bulkmode	45
3.6.7. Tarea RTOS: Control mPaP	47
3.6.8. Tarea RTOS: Control PID discreto de temperatura	47
3.7. Desarrollo de <i>Software</i>	52
3.7.1. Clase: Main GUI	54
3.7.2. Clase Main GUI/Grupo Método: Genérico	55
3.7.3. Clase Main GUI/Grupo Método: Control Manual mPaP XYZEF	55
3.7.4. Clase Main GUI/Grupo Método: Gcode	58
3.7.5. Clase Main GUI/Grupo Método: Monitoreo	58
3.7.6. Clase Main GUI/Método: Enviar Data	59
3.7.7. Clase Main GUI/Método: Referencia r(t)	60
3.7.8. Clase Main GUI/Método: Selección Gla - Glc	61
3.7.9. Clase: Comunicación PICUSBapi	62
3.7.10. Clase: GcodeDocument	62
3.7.11. Clase: Vector5D	63

3.7.12. Clase: Impresion3D	64
3.8. Herramientas externas aplicadas	66
3.8.1. Sketchup CAD →.stl	66
3.8.2. CAM/Slic3r(.stl) →Gcode	66
3.8.3. Viewer Repetier-Host	67
4. Resultados	68
5. Conclusiones y recomendaciones	71
Bibliografía	73
Índice de Palabras	74
Acrónimos	75
Glosario	77
Anexo A: Código fuente de <i>Firmware</i>	78
Anexo B: Código fuente de <i>Software</i>	85
Anexo C	103
.1. Control PID: Discretización general	103
.2. Control PID: Discretización por partes	104

Listado de Figuras

2.1. Técnica FDM	6
2.2. Sistemas de Extrusión Vertical	6
2.3. Sistema de Extrusión Horizontal	7
2.4. Dos tipos de salida en el <i>nozzle</i>	8
2.5. Configuración mecánica cartesiana	8
2.6. Tipos de posicionamiento 3D	9
2.7. Disposición funcional Microcontroladores de la Familia 18FXX5X	10
2.8. Características eléctricas y mecánicas del protocolo USB	11
2.9. Diagrama en bloques de relación lógica para Enumeración/ <i>EndPoint/Pipes</i> USB	12
2.10. Configuración mecánica interna de dos tipos de mPaP	13
2.11. Control básico mPaP Unipolar	13
2.12. Control básico mPaP Bipolar	14
2.13. Tres métodos de movimiento mPaP	15
2.14. Compensación Analógica	16
2.15. Dos tipos de estructuras básicas de aplicación y una estructura básica RTOS	16
2.16. Ejemplo de modelo CAD en Sketchup	19
2.17. Ejemplo de uso del Slicer	19
2.18. Ejemplo de uso del <i>Viewer Repiter-Host</i>	20
3.1. Esquema general de funcionamiento del prototipo	22
3.2. E-waste	23
3.3. Modelo CAD del extrusor	23
3.4. Extrusor ensamblado	24
3.5. Extrusor completado	24
3.6. Alimentación del material plástico PET	25
3.7. Licuefactor eléctrica-caloríficamente aislado	25
3.8. Construcción del pistón	26
3.9. Eje/Motor X	26
3.10. Eje/Motor Y	27
3.11. Eje/Motor Z	27
3.12. Posicionamiento 3D del prototipo completo	28
3.13. Esquema electrónico del MCU PIC18F4550	29
3.14. Recomendación del fabricante	30
3.15. Diseño recomendado	30
3.17. Calentamiento por Efecto Joule	31
3.16. Esquema de control mPaP completo	32

3.18. Acondicionamiento de Termocupla.	33
3.19. Diseño del sensor de Temperatura Termocupla K.	35
3.20. Implementación electrónica completa.	36
3.21. Conversión de escala en referencia $r(t)$, para procesar en el controlador PID.	39
3.22. <i>Toolpath</i> del código 3.1.	41
3.23. Envío/Recepción de paquetes USB.	46
3.24. Respuesta G_p al escalón, muestreado con el GUI.	48
3.25. Identificación del modelo con Ident Matlab.	49
3.26. Controlador $G_c(s)$ con Sisotool.	49
3.27. Simulación Matlab.	51
3.28. Respuesta al escalón del sistema a lazo cerrado G_{lc}	51
3.29. Entorno de trabajo en Visual Studio	53
3.30. DGML general.	54
3.31. DGML Clase principal InterfazMain3DPrinter.	55
3.32. Proceso de Envío de datos mPaP al prototipo.	57
3.33. GUI-DGML Control Manual XYZ.	57
3.34. GUI-DGML Grupo Método: Gcode.	58
3.35. GUI-DGML Grupo Método Monitoreo.	60
3.36. GUI Referencia $r(t)$ PWM.	61
3.37. GUI Selección Gla - Glc,	61
3.38. DGML Clase PICUSBapi.	62
3.39. DGML GcodeDocument.	63
3.40. DGML Vector5D.	64
3.41. DGML Impresion3D.	65
3.42. Interfaz Sketchup y plugin .stl.	66
3.43. Interfaz Slic3r.	67
3.44. Interfaz Repetier-Host.	67
4.1. Modelo CAD de la pieza.	68
4.2. Generación y visualización Gcode.	69
4.3. Operación general a través del GUI.	69
4.4. Pieza final impresa.	70

Lista de Tablas

2.1. Temperaturas de trabajo.	7
2.2. Posicionamiento 3D.	9
2.3. Resumen de características USB.	11
3.1. Condiciones de implementación. Fuente: Elaboración propia.	33
3.2. Condiciones de relación (pulsos vs mm).	41
3.3. Condiciones de relación (mm vs periodo).	42

Listado de Códigos fuente

2.1.	Ejemplo de líneas de comandos Gcode.	19
3.1.	Ejemplo Gcode, rectángulo 2D de 10x5 mm.	40
3.2.	Gp a lazo abierto, Gla.	48
3.3.	Sistema a lazo abierto Gla, y lazo cerrado Glc.	50
1.	Firmware completo.	78
2.	Secciones VID/PID - name de usb_descriptor.h	84
3.	Clase InterfazMain3DPrinter completo.	85
4.	Clase PICUSBapi completo.	94
5.	Clase GcodeDocument completo.	97
6.	Clase Vector5D completo.	97
7.	Clase Impresion3D completo.	98

Lista de Algoritmos

2.1.	Algoritmo ejemplo RTOS.	18
3.1.	Estructura general.	43
3.2.	Cabecera de configuración MCU PIC18F4550.	44
3.3.	Función principal: main	45
3.5.	Función ADC.	45
3.4.	Función genérica: Inicio.	46
3.6.	Tarea RTOS: Comunicación USB.	47
3.7.	Control mPaP.	47
3.8.	Discretización PID por partes.	52
3.10.	Grupo Método: Genérico.	55
3.9.	Clase InterfazMain3DPrinter.	56
3.11.	Grupo Método: Control Manual mPaP XYZ.	58
3.12.	Grupo Método Gcode.	59
3.13.	Grupo Método: Monitoreo.	59
3.14.	Método: Enviar Data.	60
3.15.	Método Referencia $r(t)$ PWM.	61
3.16.	Método: Selección Gla - Glc.	61
3.17.	Clase PICUSBapi.	63
3.18.	Clase: GcodeDocument.	63
3.19.	Clase: Vector5D.	64
3.20.	Clase Impresion3D.	65

1. Introducción

La posibilidad de construir un objeto físico real y tangible a partir de la información de su modelo CAD utilizando solo una máquina que no ocupa más espacio que un microondas comercial, es lo que la tecnología de Impresoras 3D introdujo al imaginario colectivo de las personas.

El concepto una Impresora 3D es similar al de un *Control Numérico Computarizado - Computer Numerical Control* (CNC); que produce piezas por sustracción, el cabezal móvil compuesto por un **taladro** sustrae el material **capa por capa** a través de comandos Gcode generados en un computador. En cambio, una Impresora 3D produce piezas por **adición**, el cabezal móvil compuesto por un **extrusor** funde y deposita el material **capa por capa**, también usa comandos Gcode.

1.1. Antecedentes

Considerada como una de las diez industrias de mayor crecimiento [1, 2], el concepto de una Impresora 3D nació a comienzos de los años 80 debido a que la industria requería producir piezas geométricamente complejas en un solo proceso, bajo el título de la *Manufactura por Adición - Additive Manufacturing* (AM).

Eventualmente en marzo del año 2005 *The RepRap Project* [3] presentó su primer modelo de Impresora 3D autoreponible para consumo doméstico, investigación y desarrollo, con *Licencia Pública General - General Public License* (GPL). En pocos años varios grupos emprendedores siguieron la línea RepRap, creando empresas como *MakerBot* [4], *Printrbot* [3], *Formlabs* [5], el efecto fue que entre el 2010 y 2013 el costo de una Impresora 3D comercial doméstica bajó drásticamente.

Pero, para considerar qué áreas y cuál es la situación actual de la AM, es necesario listar lo expuesto en *Roadmap Wohlers 2009* [6] y en *Additive Manufacturing Technologies* [7]:

- Procesamiento de materiales en general.
- Modelado por Deposición Fundida - FDM.
- Proceso de Estereolitografía - SLA.
- Proceso de Sinterización Selectiva por Láser - SLS.
- Mecanismo de fusión en polvo.

- Proceso de laminación en papel.
- Manufactura por bioadición orgánica de tejidos vivos para el reemplazo de órganos humanos.
- Desarrollo de prótesis humana especializada y dedicada; no invasiva, para el reemplazo de extremidades.
- Industria automotriz y aeroespacial.
- Diseño de métodos y herramientas *Software-Hardware CAD/CAM/CAE*.

Destacando el desarrollo particular en el área FDM, con una perspectiva de uso material de impresión alternativo **Tereftalato de Polietileno - Polyethylene Terephthalate** (PET) para la impresión de las piezas finales; pocos ejemplos pero relevantes, se mencionarán a continuación:

- *Perpetual Plastic Project* [8], proyecto que fomenta el reciclado de vasos plásticos mediante el uso de una microplanta de procesamiento de materiales hasta su reutilización en una Impresora 3D.
- *Developing a plastics recycling add-on for the RepRap 3D printer* [3] y *Granule Extruder* [9], diseños de extrusores para complementar a una Impresora 3D RepRap convencional.
- *Extruder to Recycle Plastic Milk Bottles* [10], *Recyclebot* [11], proyectos de manufactura de plásticos PET enrollados, para su posterior uso en Impresoras 3D tradicionales.

A nivel académico universitario, el "Programa de Investigación, Negocio y Educación" en la Universidad de Tecnología Delft en Holanda [12]; a cargo del Departamento de Diseño de Ingeniería Industrial y el Departamento de Arquitectura, tienen un programa semestral académico formal de "Manufactura por Adición"; con 60 horas teóricas y 194 horas prácticas, dirigido a estudiantes de todas las carreras, que hasta principios de mayo del 2012; después de 3 años, 105 estudiantes concluyeron el programa satisfactoriamente.

En Bolivia, **Sawers3D** [13] es la Impresora 3D comercial de actual relevancia, sin embargo emplean **Poliácido Láctico - Polylactic Acid** (PLA) y/o **Acrilonitrilo Butadieno Estireno - Acrylonitrile Butadiene Styrene** (ABS), y componentes electromecánicos importados. A nivel académico, la Universidad Autónoma Gabriel René Moreno (UAGRM) por medio del grupo autonombado **RepRapBol**, implementaron impresoras 3D tradicionales, y al igual que **Sawers3D** recurrieron a la importación de componentes.

Sawers3D tiene los siguientes componentes importados exclusivamente para impresoras 3D tipo RepRap:

- Motores paso a paso (**mPaP**).
- Controladores electrónicos para **mPaP** (**Sanguinololu**, **Pololu Shield**).
- Microcontrolador **Atmel** embebido en sistemas de desarrollo **Arduino**.
- Extrusor prefabricado y ensamblado.
- Material de impresión enrollado tipo **PLA**.

1.2. Definición del problema

La carrera de Ingeniería Electrónica aún no introdujo el estudio de esta tecnología, que bien podrían servir de motivación para generar documentación académica con métodos y técnicas nuevas y/o mejoradas.

Durante el proceso de desarrollo de una Impresora 3D, existe el problema de la necesidad de importación de componentes específicos electrónicos y electromecánicos, pero esa no es la única dependencia, también hay otra con el material de impresión PLA/ABS que se obtiene únicamente por importación.

Por otro lado, desde el punto de vista medioambiental, existe el problema de generación de desecho electrónico **e-waste** a partir de los componentes de Impresoras 3D dañadas, y está la producción de nuevo plástico PLA/ABS.

1.3. Objetivo

Implementar un prototipo básico de Impresora 3D reutilizando y adaptando desecho electrónico (**e-waste**) en el proceso de construcción electromecánica, además del empleo de botellas PET desechables como insumo de impresión.

Para lograr el objetivo se requiere de:

- Adaptar mecánicamente el sistema de alimentación electromecánica para el material de impresión, sistema de extrusión electromecánico de fundición y sistema de transmisión de energía mecánica en los tres ejes de movimiento.
- Diseñar el sistema de control **Proporcional Integral Derivativo (PID)** de temperatura, control de motores paso a paso, circuito de comunicación USB y circuito de calentamiento por Efecto Joule.
- Desarrollar el *Software* de control, comunicación, impresión y operación.
- Realizar pruebas de impresión 3D y discutir resultados.

1.4. Justificación

Justificación académica

El conocimiento adquirido en Ingeniería Electrónica y el estudio del procedimiento completo de impresión 3D por técnica FDM, sirven de justificación para la realización del presente proyecto. Involucra áreas como; diseño de sistemas de control, diseños de circuitos de control para motores paso a paso, aplicación de microcontroladores de gama media, acondicionamiento de señal para sensores de instrumentación industrial, conversión de señales analógicas en digitales, control por **Modulación por Ancho de Pulso - Pulse Width Modulation (PWM)** para la conversión de energía eléctrica en energía calorífica.

Además del conocimiento en desarrollo de algoritmos y manejo de lenguajes de programación, protocolo de comunicación USB en la interacción *Hardware-Software*, empleo de herramientas de análisis y simulación en *Matlab*, sin olvidar los principios básicos en mecánica para la adaptación de los engranajes.

Justificación medioambiental

La reutilización **e-waste** y uso de botellas plásticas PET desechables califican como propuesta alternativa de bajo costo en el área de reciclaje y tratamiento de desechos.

1.5. Alcances y limitaciones

Los alcances del proyecto son:

- La alimentación del material de impresión será manual.
- El extrusor fundirá y depositará el material capa por capa en forma vertical.
- Sistema de transmisión de energía mecánica será dependiente de los tamaños y tipos de mPaP y engranajes adquiridos por **e-waste**.
- Diseño del sistema electrónico propio estará de acuerdo a las condiciones de funcionamiento del extrusor y el sistema de transmisión de energía mecánica.
- Controlador PID de temperatura será sometido a análisis, diseño, simulación e implementación a través de herramientas *Hardware-Software*.
- El *Firmware-Software* será desarrollado con algoritmos propios y su implementación a través de Programación Imperativa-Estruturada y Programación Orientada a Objetos **POO**.
- Las piezas finales impresas deberán corresponder en dimensiones al modelo **CAD** original.

En cuanto a las limitaciones, varios aspectos se consideran por ser un prototipo:

- Se requiere fragmentar plástico PET de acuerdo a las dimensiones del extrusor porque la alimentación del material es manual.
- La replicación del extrusor es intrínsecamente dependiente de los componentes **e-waste** y materiales locales.
- El sistema de transmisión de energía sufre de la misma dependencia del extrusor pero su replicación es accesible.
- El diseño electrónico es dependiente de las condiciones de funcionamiento en la etapa de calentamiento del plástico por Efecto Joule.
- Los algoritmos *Firmware-Software* desarrollados pueden ser usados en una **CNC**, pero la etapa de impresión 3D es dependiente de las condiciones de funcionamiento del extrusor.

2. Marco Teórico

Con el propósito de introducir los conceptos y definiciones para el desarrollo del proyecto, se describen en cuatro secciones las áreas de mayor relevancia.

La sección [2.1](#) describe la técnica **Modelado por Deposición Fundida - *Fused Deposition Modeling*** (FDM), la segunda sección [2.2](#) define el tipo de **Unidad de Micro Controlador - *Micro Controller Unit*** (MCU), protocolo USB, los motores, sensor de temperatura y método de calentamiento del plástico.

Por último, se explica brevemente los conceptos de un **Sistema Operativo de Tiempo Real - *Real Time Operating System*** (RTOS) y se lista las herramientas de *Software* externas necesarias.

2.1. Modelado por Deposición Fundida (FDM)

Una Impresora 3D estándar manufactura piezas por adición, con un cabezal móvil en coordenadas Cartesianas compuesto por un extrusor electromecánico que funde y deposita el material de impresión PLA/ABS capa por capa, en general este proceso es una variante del **Modelado por Deposición Fundida - *Fused Deposition Modeling*** (FDM).

Esta técnica FDM; fig. [2.1](#), respeta una serie de principios básicos para procesar el material hasta formar una pieza 3D.

- Alimentación del material.
- Licuefacción.
- Extrusión.
- Robot Cartesiano.
- Posicionamiento 3D.

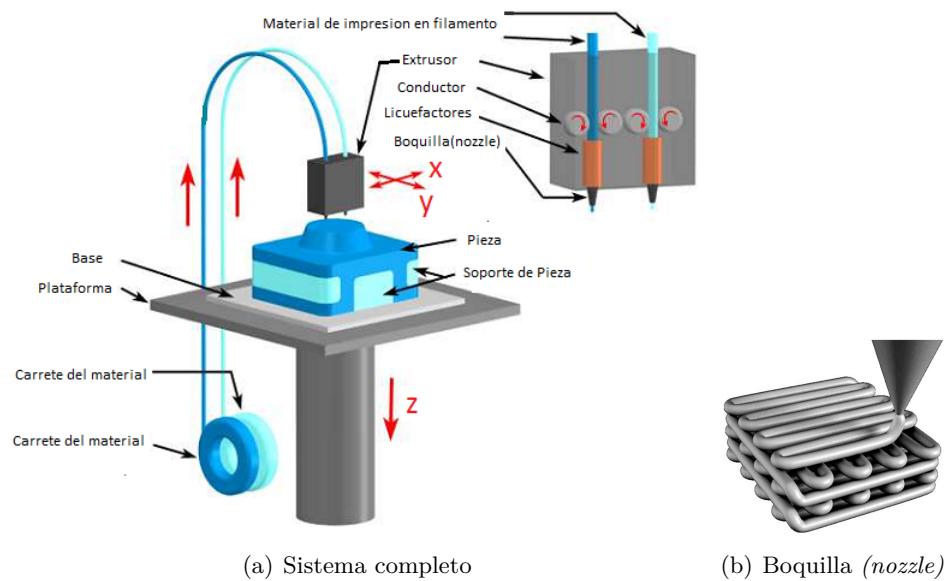


Figura 2.1: Técnica FDM.

Fuente: Tecnología AM [14, 7] Cap. 1, Cap. 10

Algunas de estas condiciones tienen características matemáticas-físicas en mecánica de fluidos Newtonianos; que no se explican en este proyecto, pero para un estudio detallado revisar [15, 16] Cap. 1, Cap. 4.

2.1.1. Alimentación del material

A través de un sistema electromecánico o de forma manual, el material de impresión en estado sólido es transportado hacia la Cámara de Licuefacción, por medio de una tolva o un sistema continuo de rodillos para un filamento de sección uniforme fig. 2.2.

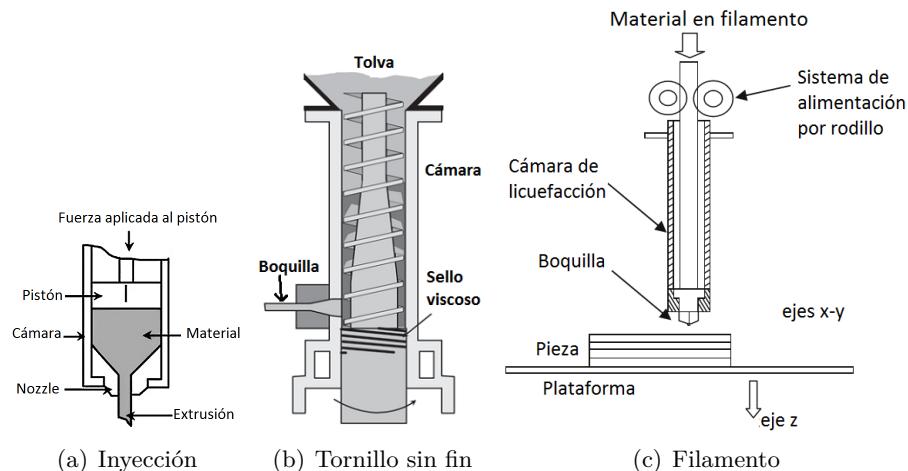


Figura 2.2: Sistemas de Extrusión Vertical.

Fuente: Procesos de extrusión [17, 7]

La ventaja de una tolva, es la introducción del material picado o fragmentado de diferentes dimensiones, facilita la mezcla de colores y plásticos de diferentes tipos y generalmente se usa en extrusores horizontales para obtener plástico enrollado; fig. 2.3, pero puede emplearse en diseños verticales, fig. 2.2(b).

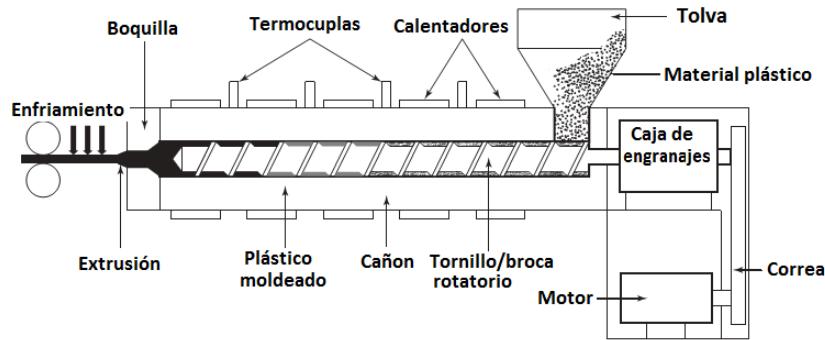


Figura 2.3: Sistema de Extrusión Horizontal.

Fuente: Procesos de extrusión [18] Cap. 1

En los modelos se debe respetar las condiciones de trabajo del extrusor; temperatura de la Cámara de Licuefacción o Licuefactor, velocidad de extrusión, presión interna y viscosidad del material a fin de realizar una extrusión satisfactoria.

2.1.2. Licuefacción

En el Licuefactor; fig. 2.2, el material de impresión sólido es calentado a un estado líquido mediante **Calentamiento por Efecto Joule**, para lograr la temperatura deseada se puede implementar una resistencia eléctrica tipo niquelina (niquel-cobalto) u otro material.

Dependiendo del material y sus características, la cámara debe llegar a un rango de temperatura constante según tabla 2.1.

Para que no sufra combustión; ocasionando atascamiento mecánico y malos olores, generalmente se usa un controlador electrónico PID que garantiza la estabilidad del sistema, mas adelante se cuestionará las ventajas de su implementación.

	Procesamiento [°C]	Licuefactor [°C]	Salida [°C]
PET	120 - 295	n/a	n/a
ABS	180 - 274	190 - 250	210 - 250

Tabla 2.1: Temperaturas de trabajo.

Fuente: Características de polímeros [19]

2.1.3. Extrusión

Cuando se aplica presión suficiente en la cámara, el material líquido es extruido por la boquilla a velocidad controlada si la temperatura de trabajo es la indicada.

La fig. 2.2 muestra tres configuraciones donde la **presión interna** es el resultado de:

- **Caso a:** La fuerza aplicada y área circular aplicado al pistón.
- **Caso b:** El rozamiento, velocidad angular y torque aplicado en la broca, además de la viscosidad del material plástico.
- **Caso c:** La velocidad de entrada y sección transversal del filamento, al mismo tiempo los rodillos pueden otorgar mas velocidad de entrada/retracción al filamento.

El *nozzle* determina la forma y el diámetro del filamento extruido, según:

- **Nozzle corto:** Ideal para impresoras 3D de calidad aceptable, velocidad de extrusión **lenta** y pieza final de mayor **resolución**.
- **Nozzle ancho:** Para extrusores horizontales, ideales para producir rollos de plásticos, velocidad de extrusión **rápida** y pieza final de menor **resolución**.

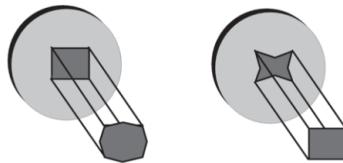


Figura 2.4: Dos tipos de salida en el *nozzle*.

Fuente: Procesos de extrusión [17]

2.1.4. Robot Cartesiano

La técnica FDM no sería posible sin el principio de funcionamiento de un Robot Cartesiano; fig. 2.5, puede moverse linealmente en los 3 ejes XYZ donde cada eje debe formar un ángulo recto respecto del otro. Esta configuración mecánica ofrece ventajas simplificadas en la ecuaciones de movimiento, por eso su uso intensivo a nivel industrial en las maquinas CNC, fresadoras y/o *plotters* de dibujo.

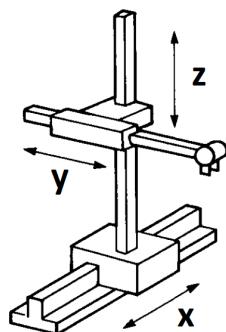


Figura 2.5: Configuración mecánica cartesiana.

Fuente: Introducción a la Robótica [20]

Una Impresora 3D respeta esta configuración, según la resolución de los mPaP e.g. 1.8°, 3.6°, 7.5°; ubicados en los 3 ejes respectivamente, ellos traducen los comandos Gcode en movimiento

lineal para el cabezal móvil (extrusor) y el movimiento de la plataforma para una impresión 3D capa por capa; es decir, un correcto posicionamiento 3D.

2.1.5. Posicionamiento 3D

Existen diferentes tipos de posicionamiento 3D, la fig. 2.6 muestra algunos de ellos.

- **Cube:** Extrusor en movimiento lineal solo en el eje X, la base se mueve en los ejes YZ.
- **PrintrbotLC:** Extrusor en los ejes XZ y la base de impresión en el eje Y.
- **Ultimaker:** Extrusor en los ejes XY y la base de impresión en el eje Z.

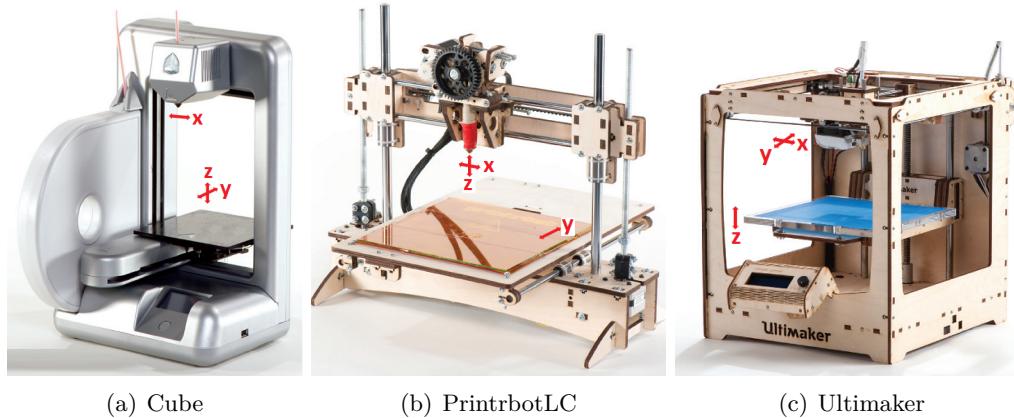


Figura 2.6: Tipos de posicionamiento 3D.

Fuente: Guía de Impresión 3D [21]

Es conveniente tabular estos tipos para identificar el adecuado en la implementación del prototipo, tabla 2.2.

Tipo	Ejes de movimiento	
	Extrusor	Base
A	X	YZ
B	XZ	Y
C	XY	Z

Tabla 2.2: Posicionamiento 3D.

Fuente: Elaboración propia.

2.2. Electrónica

Esta sección describe los componentes principales del MCU, las características del protocolo USB, dos tipos de mPaP, calentamiento por Efecto Joule y el tipo de sensor de temperatura Termocupla K.

2.2.1. Microcontrolador (MCU) - Microchip Familia 18FXX5X

La Familia MCU Microchip 18FXX5X; fig. 2.7, son de los más comerciales y económicos a nivel académico e industrial, su uso extendido desde aplicaciones simples hasta video juegos retro, los han convertido en los favoritos para profesionales:

- Módulo USB integrado tipo HID/MSD/CDC/CC-Bulk.
- Conversor Analógico Digital - *Analogic Digital Converter* (ADC) de 10 bits y 13 canales.
- Oscilador hasta 48MHz.
- Cuatro Timers de 16 bits.
- Dos módulos Capturador/Comparador/PWM - *Capture/Compare/PWM* (CCP) de 16/16/10 bits, respectivamente.
- Encapsulamiento de 28/40/44 pines.

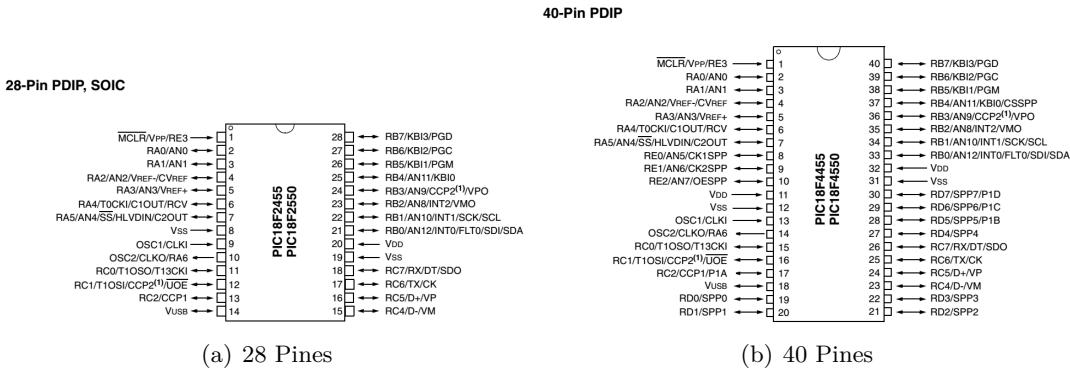


Figura 2.7: Disposición funcional MCU de la Familia 18FXX5X.

Fuente: *Datasheet* del fabricante [22] pág. 2

2.2.2. Protocolo de comunicación USB

Bus Serial Universal - *Universal Bus Serial* (USB) no requiere introducción pero es necesario definir 3 áreas importantes que son consideradas en el proyecto.

Velocidad del Bus

Es la velocidad del par trenzado conectado a los Pines D- / D+; fig. 2.8, estos valores son teóricos y en la práctica bajan bastante porque dependen de las condiciones del dispositivo; para este caso, un MCU.

- **Low Speed:** 1,5 Mbps en USB 1.1/2.0/3.0.
- **Full Speed:** 12 Mbps en USB 1.1/2.0/3.0.

- **High Speed:** 480 Mbps en USB 2.0/3.0.

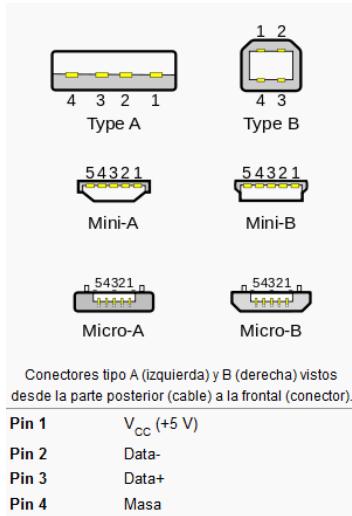


Figura 2.8: Características eléctricas y mecánicas del protocolo USB.

Fuente: Sitio oficial USB [23]

Transferencia

La cantidad de información y el tipo control/flujo de datos que USB puede manejar son:

- **Interrupt:** Con verificación de datos y latencia; *Low Speed*, para teclado, mouse, pantalla táctil.
- **Isochronous:** Sin verificación de datos y latencia; *Full Speed*, para señales de audio y video.
- **Bulk:** Transferencia masiva, con verificación de datos y latencia; *Full/High Speed*, para discos duros, pendrives, escáner e impresoras.

Clase

Un clase USB es parecida a una clase definida en P00, contiene los constructores y métodos para ser utilizados por objetos creados en las aplicaciones *Host*, hay 4 tipos básicos resumidos en la tabla 2.3.

Clase	Transferencia	Velocidad	Dispositivos
HID	Interrupt/Isochronous	Low Speed	mouse, teclado, pantalla táctil
MSD	Interrupt/Isochronous/Bulk	Full/High Speed	disco duro, pendrive
CDC	Interrupt/Isochronous/Bulk	Full/High Speed	escáner, impresora
CC	Interrupt/Isochronous/Bulk	Full/High Speed	personalizado

Tabla 2.3: Resumen de características USB.

Fuente: Elaboración propia.

Enumeración/EndPoints/Pipes

La fig. 2.9 muestra en diagrama de bloques la relación lógica *Host - Slave(MCU)* para la correcta comunicación USB.

- **Enumeración:** Se da cuando el *Host* detecta un dispositivo nuevo en algún puerto I/O y a partir de la información del *Driver Host* y Descriptor *Slave* asigna un PID/VID único para que no tenga conflictos con otros dispositivos que se conecten al mismo puerto.
- **EndPoints:** *Buffers* de memoria; en el *Host* y dispositivo *Slave(MCU)*, necesarios para el envío/recepción de paquetes USB.
- **Pipes:** Conexiones lógicas entre *Endpoints Host - Slave(MCU)* habilitados después de la Enumeración.
- **Descriptor MCU y driver Host:** Ambos deben ser configurados para un número compatible de *Endpoints*; según el *datasheet* [fuente 22], la familia 18FXX5X tiene hasta 32 *Endpoints* unidireccional y 16 bidireccional.

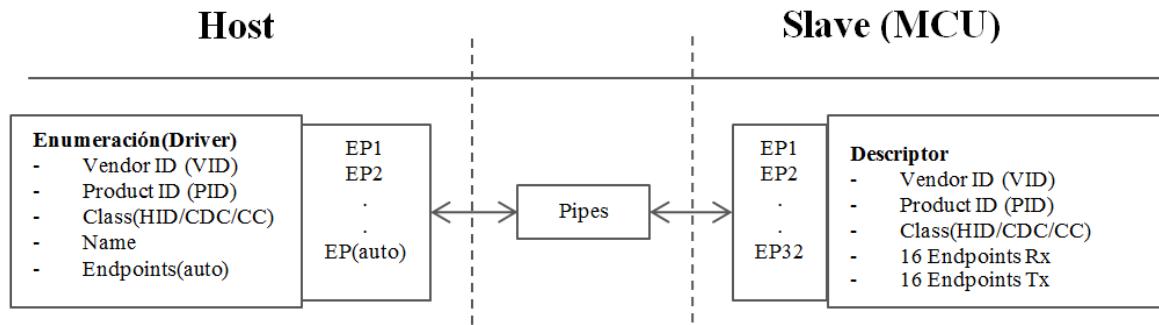


Figura 2.9: Diagrama de bloques de relación lógica para Enumeración/*EndPoints/Pipes* USB.
Fuente: Elaboración propia.

2.2.3. Motores Paso a Paso (mPaP)

Los mPaP son bastante usados por su precisión, varían en número de pasos por ciclo, tipo de embobinado; fig. 2.10, torque y tamaño.

Por tipo de construcción:

- Imán permanente.
- Reluctancia variable.
- Híbrido.

Por tipo de embobinado:

- Unipolar.
- Bipolar.

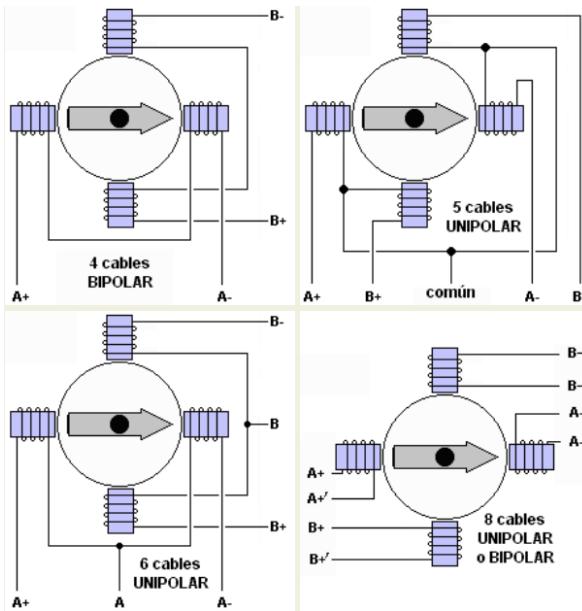


Figura 2.10: Configuración mecánica interna de dos tipos de mPaP.

Fuente: Características básicas mPaP [24]

Unipolar

De 5, 6 u 8 conectores, no requiere inversión de corriente y la fig. 2.11 muestra un circuito de control simple.

En la práctica se aplica un esquema mas completo con control de velocidad y giro, para tal efecto usualmente se recomienda el Circuito Integrado - *Integrated Circuit* (IC) L297 con un array de transistores.

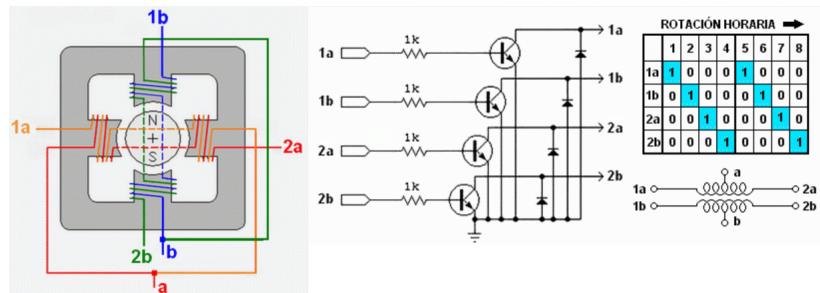


Figura 2.11: Control básico mPaP Unipolar.

Fuente: Características básicas mPaP [24]

Bipolar

Casi siempre viene con 4 conectores y requiere un circuito de control puente H fig. 2.12.

Al igual que el mPaP Unipolar, para el control de giro y velocidad también se recomienda IC L297 y el puente H doble IC L298.

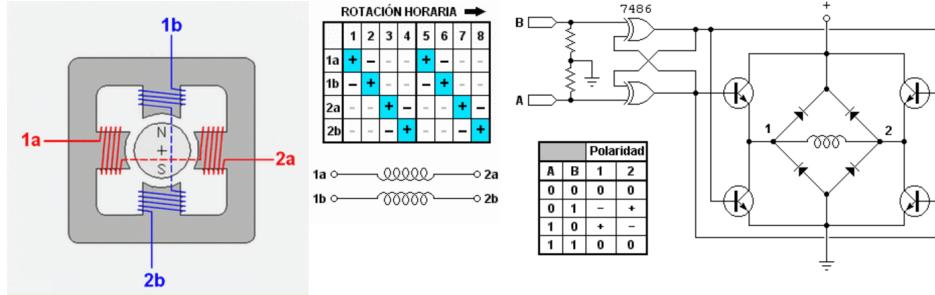


Figura 2.12: Control básico mPaP Bipolar.

Fuente: Características básicas mPaP [24]

En ambos tipos de motores existe 3 métodos de movimiento; paso de una fase, paso de dos fases, medio paso y micropaso, pero el último método no se tratará en el proyecto. La fig 2.13 muestra como el eje magnético se alinea con las bobinas polarizadas para generar torque y movimiento.

- **Paso de una fase:** Solo se energiza una bobina a la vez.
- **Paso de dos fases:** Se energizan dos bobinas al mismo tiempo.
- **Medio paso:** Es una combinación de los dos anteriores, en un paso se energiza una bobina y en el siguiente se energizan dos bobinas, lo que provoca 8 pasos en total.

2.2.4. Calentamiento por Efecto Joule

Es la transformación de energía eléctrica en calorífica, a través del incremento de temperatura en un conductor sometido a una corriente eléctrica. Se emplea en dispositivos de calentamiento como planchas, soldadores, hornillas o duchas eléctricas, etc.

Esta técnica es necesaria en la etapa de calentamiento del material plástico en la Cámara de Licuefacción; subsección 2.1.2, de acuerdo al rango de temperatura deseado según tabla 2.1.

La **Potencia Disipada** en un conductor eléctrico ecuación 2.1 representa la relación matemática con la ley de Ohm para un conductor eléctrico de Resistencia $R[\Omega]$

$$P = V \cdot I = I^2 \cdot R = \frac{V^2}{R} \quad [W] \quad (2.1)$$

Y la **Energía Disipada** en un tiempo $t[s]$, ecuación 2.2.

$$E = P \cdot t = V \cdot I \cdot t = I^2 \cdot R \cdot t = \frac{V^2}{R} \cdot t \quad [J] \quad (2.2)$$

2.2.5. Sensor de temperatura

Es importante el tratamiento de esta señal de realimentación hacia la etapa electrónica porque permite controlar el rango de temperatura deseada en la Cámara de Licuefacción.

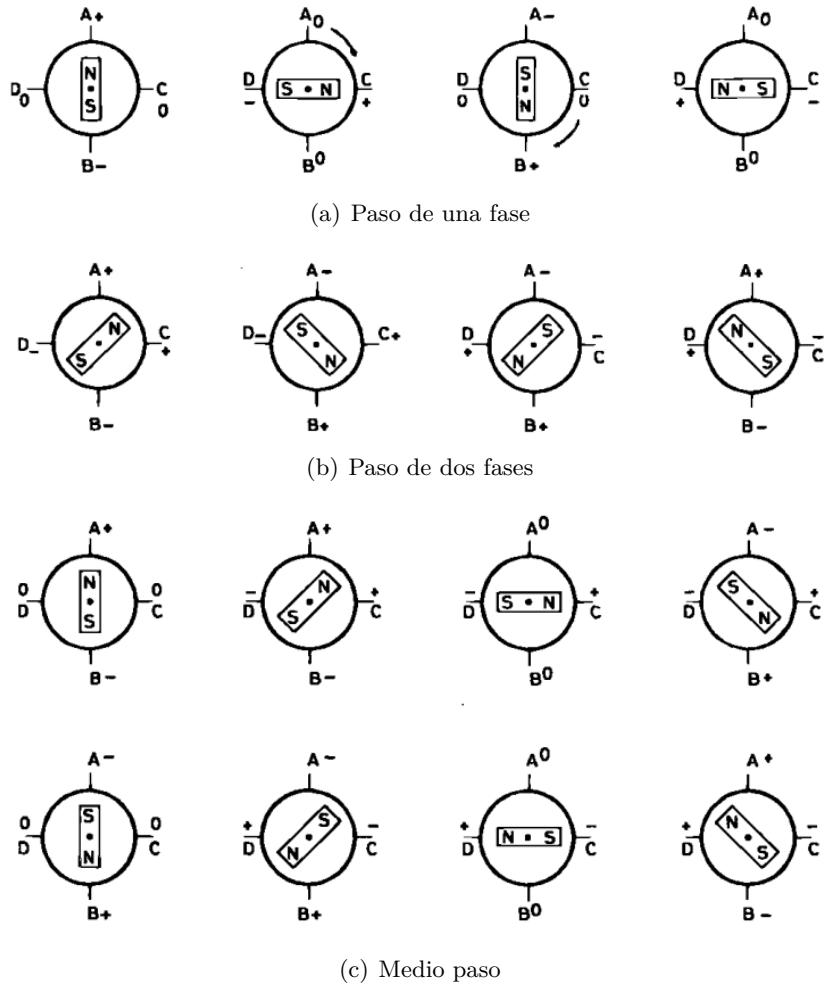


Figura 2.13: Tres métodos de movimiento mPaP.

Fuente: *Datasheet* del fabricante [25]

En la técnica FDM fig. 2.1 y de acuerdo al tipo de material tabla 2.1, se puede emplear 3 tipos de sensores; Detector de Temperatura Resistiva - *Resistance Temperature Detector* (RTD), Termistor Coeficiente de Temperatura Negativo - *Negative Temperature Coefficient* (NTC) y/o Termocupla, pero los dos primeros no serán tratados.

Termocupla

Existen varios tipos; J, K, N, T, R, S y B, pero en todos los casos está compuesto de dos metales diferentes que producen un **Diferencial de Potencial Eléctrico** V_{T_c} proporcional a la **Temperatura Caliente** T_c .

Para cualquier tipo se debe aplicar un diseño electrónico de **Compensación Analógica o Digital**; fig. 2.14, con las siguientes características:

- **Bloque isométrico:** Agrupa la juntura fría y un Sensor de Temperatura Extra para la medición de T_f .

- **Acondicionamiento de señal:** Circuito para el Sensor de Temperatura Extra.
- **Sumador:** Suma los voltajes del Bloque isométrico V_{T_c, T_f} y Acondicionamiento de señal $V_{T_f,0}$ para obtener el voltaje de salida V_{T_c} proporcional a T_c .

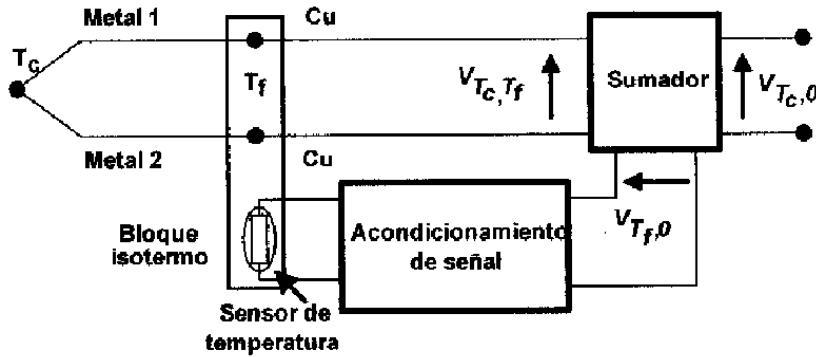
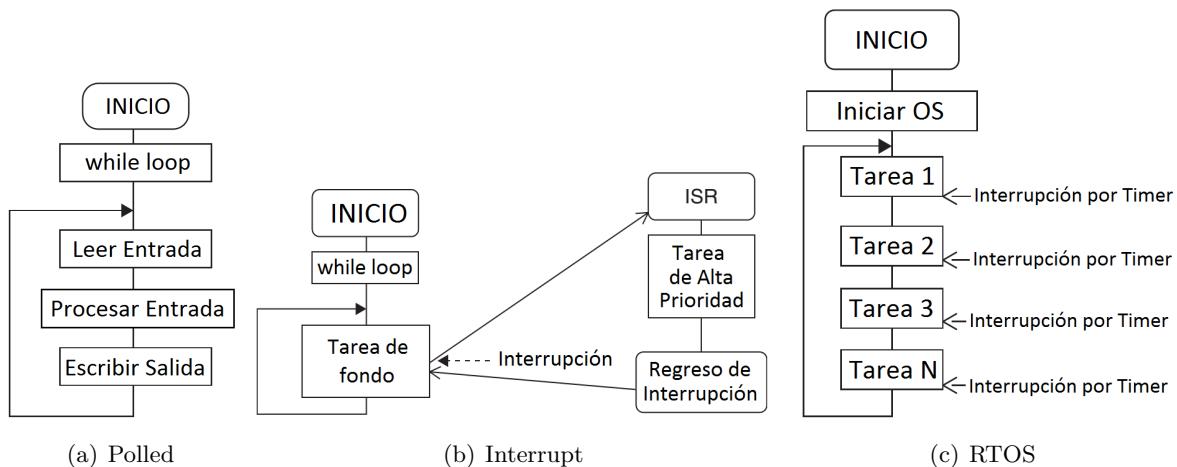


Figura 2.14: Compensación Analógica.

Fuente: Conceptos de Instrumentación [26] Cap. 13

2.3. Sistema Operativo en Tiempo Real (RTOS)

Generalmente, para aplicaciones básicas en **Sistemas Embebidos** es suficiente el empleo de un **Sistema Operativo - Operating System (OS)** bajo Estructura *Polled* fig. 2.15(a); lo que significa ejecutar un tarea por vez en forma descendente dentro del bucle infinito `while loop`, pero si se desea un sistema complejo Multi-tarea de sincronización, lo recomendable es usar un RTOS fig. 2.15(c).

Figura 2.15: Dos tipos de estructuras básicas de aplicación y una estructura básica RTOS.
Fuente: Programación MCU [27] Cap. 5

- **Polled:** Básico y simple, manejado por el tradicional `while loop`, ejecuta las tareas una a una en orden descendente, usado cuando los *delays* no son importantes.

- **Interrupt:** Principalmente empleado para responder a eventos asíncronos de Rutina de Servicio de Interrupción - *Interrupt Service Routine* (ISR), necesario para aplicaciones que requieren constantemente monitorear cambios en algún puerto I/O; e.g. ADC.
- **RTOS:** Complejo y potente, requiere un *kernel* dedicado para la creación, control y distribución de tareas específicas que necesitan; sincronización, respuesta rápida a eventos por Interrupción externa ISR y *delay* mínimo. Al mismo tiempo reemplaza el *while loop* por una llamada de servicio de inicialización.

2.3.1. *kernel* RTOS

Es un grupo de instrucciones y comandos compuesto por:

- **Operación de Tareas:** Crear, eliminar, asignar prioridad y cambiar estado.
- **Scheduler:** Asigna tiempos de ejecución a las tareas por medio de 3 algoritmos:
 - **Cooperativo:** Solo si la tarea fue completada se devuelve el control al *Scheduler*.
 - **Round-robin:** Cada tarea tiene un tiempo de ejecución homogéneo.
 - **Preferencial:** Las tareas se ejecutan en función de su prioridad.
- **Servicios:** Comandos de manejo de Interrupciones, *timers*, *delays*, memoria y puertos I/O.
- **Sincronización y mensajes:** Comandos bandera/semáforo para sincronizar tareas, y envío/recepción de mensajes entre tareas.

En resumen, RTOS es un tipo de OS Multi-tarea compuesto por un conjunto de instrucciones y comandos definidos a través de un *kernel*, que gestiona el tiempo de ejecución, control prioritario, respuesta rápida a eventos por Interrupción externa ISR y sincronización de tareas en una aplicación *Firmware*.

Una **tarea** en RTOS se define con instrucciones generales igual que una función común en programación, lo que hace especial a una tarea es el empleo de servicios RTOS; prioridad y estado.

2.3.2. Algoritmo RTOS

El algoritmo 2.1 define una estructura de aplicación RTOS para un **caso general**, comprende la configuración de librerías, Operación de Tareas, tipo de algoritmo *Scheduler*, uso de servicios y mensajes.

```
config RTOS
    include librerias
        Operacion de Tareas;
        Scheduler(preferencial);
        Servicios(iniciar RTOS, detener RTOS);
        Sincronización y mensajes;
    timerMCU;

def tarea1 RTOS
    prioridad(high);
    estado(enabled);
    delay(ms);
    code...;

def tarea2 RTOS
    prioridad(low);
    estado(waiting);
    mensaje(dato para tareal);
    code..;

main
    call iniciar RTOS;
```

Algoritmo 2.1: Algoritmo ejemplo RTOS. Fuente: Elaboración propia.

2.4. Herramientas externas necesarias

Para que el prototipo imprima una pieza aun requiere de herramientas externas, el modelado CAD por *Software* dedicado y el uso de lenguaje de comandos **Gcode**.

2.4.1. Modelo CAD

Puede ser descargado por Internet o modelado en un *Software CAD* como **Sketchup**, **Autocad** o **SolidWorks**, el modelo no debe sobrepasar los límites en área y altura de la plataforma de impresión fig. 2.1.

El formato de salida deseable es **.stl**, pero también existen otros formatos muy usados como **.3ds** o **.obj**, cualquiera que sea el formato debe ser posible procesar con un **Slicer**.

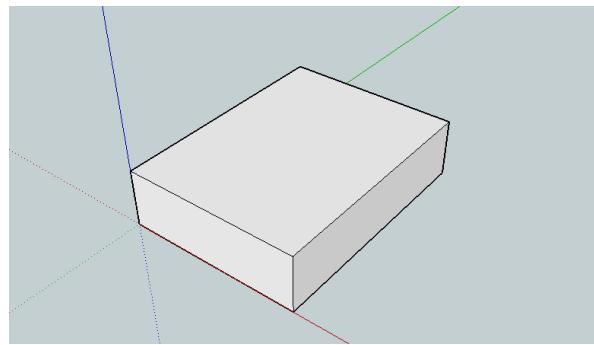


Figura 2.16: Ejemplo de modelo CAD en Sketchup.

Fuente: Elaboración propia.

2.4.2. *Slicer* → Gcode

El objetivo del **Slicer** es la creación del camino de las líneas y capas (*Toolpath*), para que el extrusor deposite el material fundido.

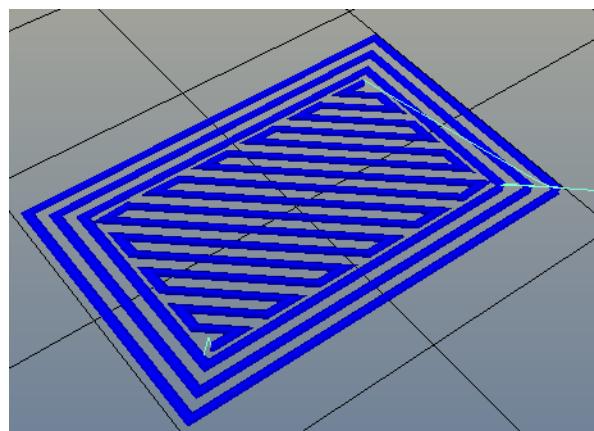


Figura 2.17: Ejemplo de uso del Slicer.

Fuente: Elaboración propia.

Para finalmente generar el Gcode que usará la Impresora 3D, [2.1](#)

```

1 M82 ; use absolute distances for extrusion
2 G1 F1800.000 E-1.00000
3 G92 E0
4 G1 Z0.500 F7800.000
5 G1 X87.630 Y85.130 F7800.000
6 G1 E1.00000 F1800.000
7 G1 X88.670 Y84.270 E1.08522 F540.000
8 G1 X89.860 Y83.630 E1.17054

```

Código 2.1: Ejemplo de líneas de comandos Gcode.

Fuente: Elaboración propia.

2.4.3. Viewer

Viewer o visualizador es un *Software* que permite mostrar en 3D el *toolpath Gcode* que el **Slicer** genera a partir del modelo CAD, fig. 2.18.

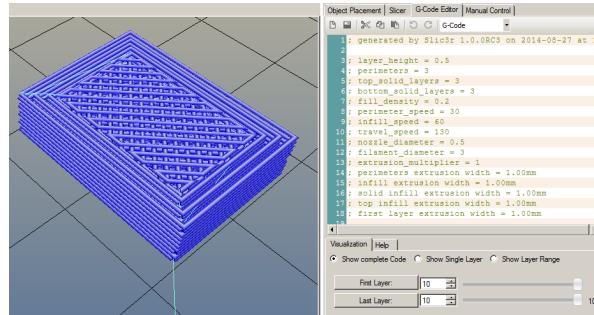


Figura 2.18: Ejemplo de uso del *Viewer* Repiter-Host.

Fuente: Elaboración propia.

3. Desarrollo del Proyecto

Se describe en diagrama de bloques el funcionamiento general del prototipo, luego se establece los requerimientos de construcción y adaptación mecánica, reutilizando e-waste y materiales adquiridos en el mercado local.

El diseño electrónico desarrollado en el *Software Proteus*, contiene las etapas de control, calentamiento por Efecto Joule y acondicionamiento de señal para el sensor de temperatura Termocupla K, y el quemado del PCB.

Las etapas de análisis matemático sirven para el desarrollo de los algoritmos de adquisición de datos por puerto ADC, envío/recepción de 10 bits por 2 Bytes a través del módulo USB PIC18F4550, la conversión de escala para la etapa de control PID discreto de temperatura, además de las ecuaciones de control PID discreto por método general y método por partes.

En la sección de desarrollo de *Firmware* se aplica Programación Imperativa-Estructurada, lenguaje de programación C y OSA-RTOS. El desarrollo del *Software* es a través de Programación Orientada a Objetos en entorno *Visual Studio* con lenguaje de programación C#.

Por último, se lista las tres herramientas externas que se usan para las pruebas de funcionamiento.

3.1. Descripción general de funcionamiento

Básicamente, se requiere una secuencia lógica de etapas que se puntúan a continuación:

- Etapa 1:
 - Modelo CAD 3D de la pieza a imprimir, se usa *Software* dedicado (e.g. Autocad, Sketchup o SolidWorks), que genere un formato de salida compatible con el **Slicer**.
 - El *Software Slicer* genera un modelo **capa por capa** del modelo CAD de la pieza, con formato de salida **Gcode**.
 - El *Software* de comunicación, operación y monitoreo carga el **Gcode** y procesa las instrucciones de impresión capa por capa que mandará al *Firmware* solo datos de distancia, velocidad y sentido de giro de los mPaP a través de protocolo de comunicación **USB**.

- Muestra mediante la **Interfaz Gráfica de Usuario - Graphic User Interface (GUI)**, la temperatura de calentamiento del plástico en el extrusor, control manual de los mPaP y envió de instrucciones automático y manual.
- Etapa 2:
 - El *Firmware* se encarga de recibir y ejecutar las instrucciones de movimiento de cada mPaP, ejecutar el algoritmo de control PID de temperatura y la comunicación USB.
 - El sistema electrónico contiene los circuitos de Envío/Recepción de datos USB a través de puertos de comunicación, circuitos de control para los mPaP y la fuente de alimentación eléctrica general de todo el prototipo.
- Etapa 3:
 - Contiene el sistema de transmisión de energía electromecánica y calorífica del prototipo, donde finalmente imprime la pieza 3D en función a las instrucciones del *Firmware* enviados a través del sistema electrónico.

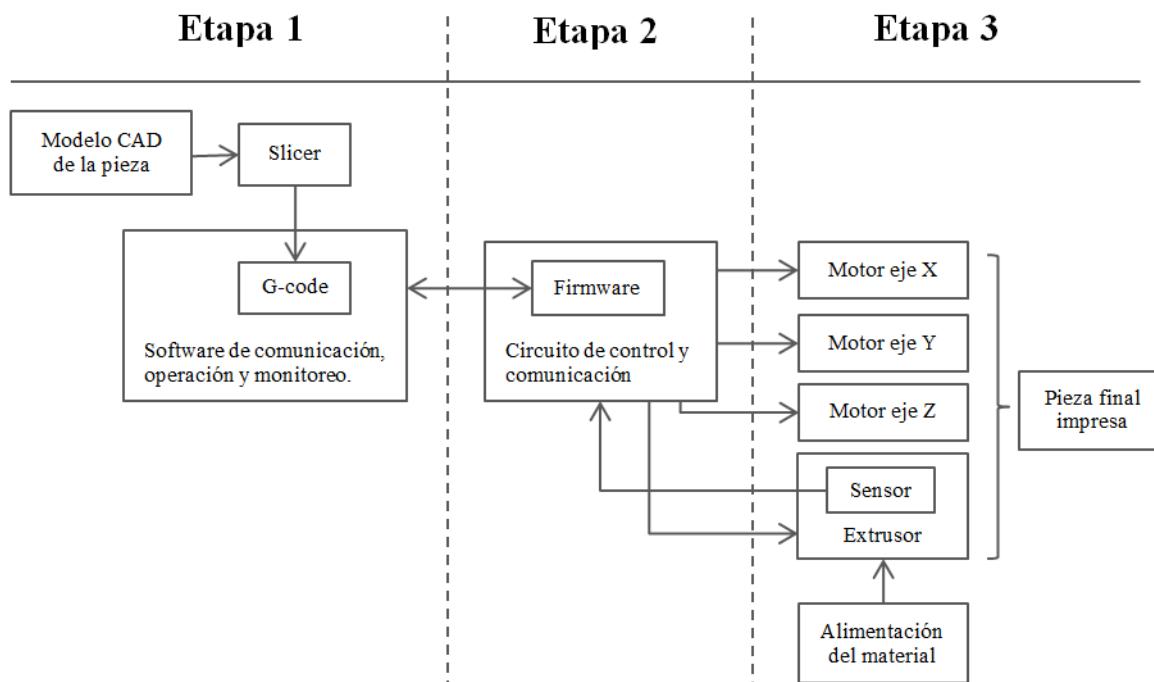


Figura 3.1: Esquema general de funcionamiento del prototipo.

Fuente: Elaboración propia.

3.2. Requerimientos de construcción electromecánica

Se dispuso la adaptación de **e-waste**; fig. 3.2, y componentes de fácil adquisición en tiendas locales, para ninguna etapa se recurrió a la importación de componentes exclusivos para

Impresoras 3D, y así satisfacer parte del objetivo.

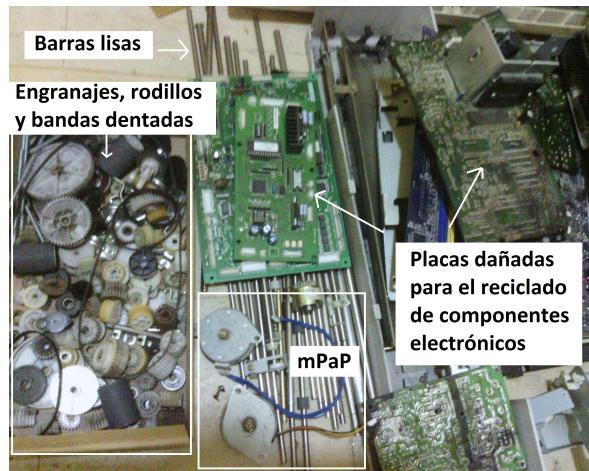


Figura 3.2: E-waste.

Fuente: Elaboración propia.

3.3. Construcción mecánica del extrusor y la estructura del prototipo

La construcción empieza con la elección de los componentes y herramientas electromecánicas, luego se realiza el modelo CAD del extrusor utilizando la herramienta *Sketchup* y finalmente la construcción total.

La fig. 3.3 demuestra la concepción de adaptación para el desarrollo del extrusor definiendo sus características principales.

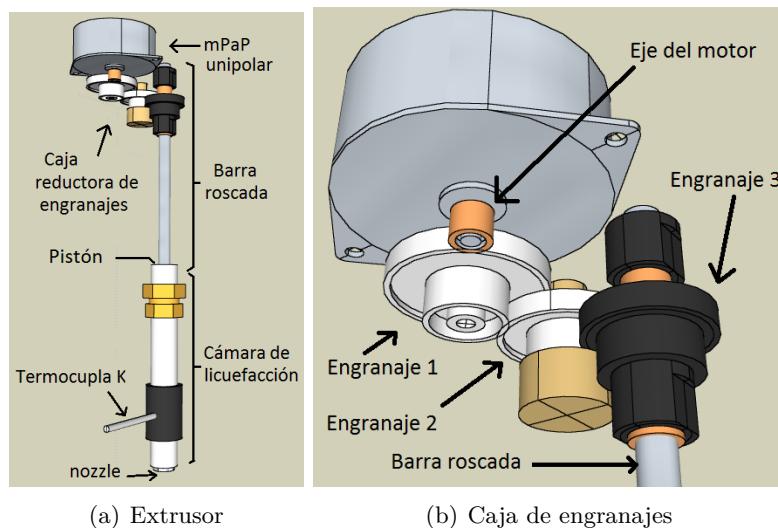
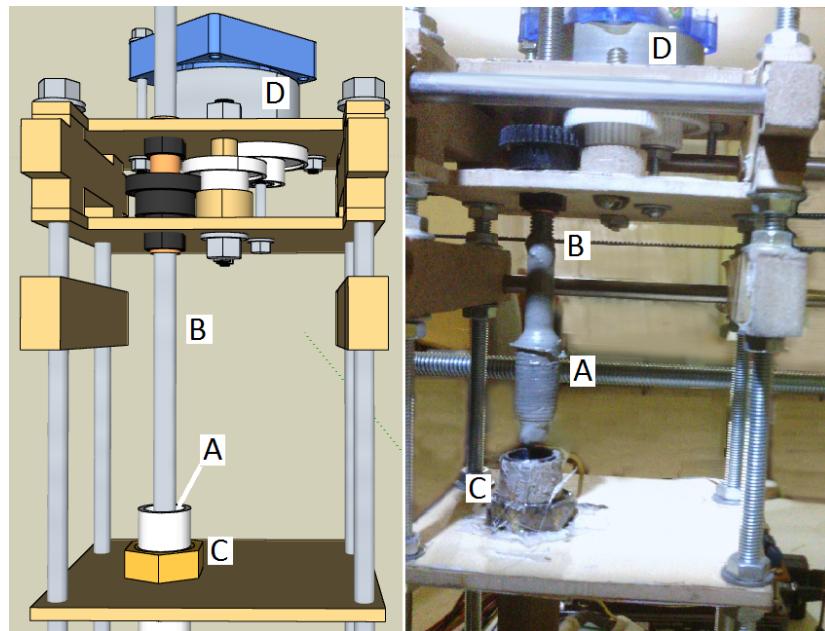


Figura 3.3: Modelo CAD del extrusor.

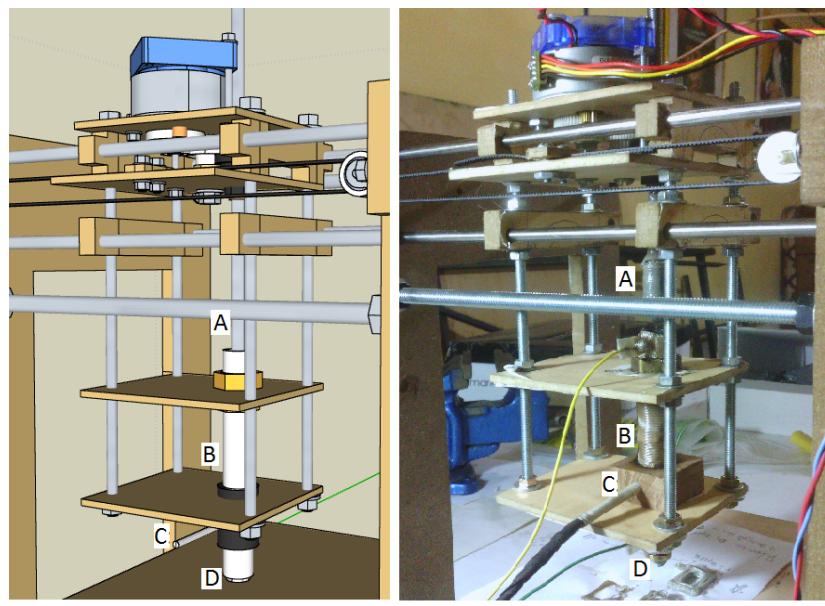
Fuente: Elaboración propia.

Para finalmente construir el extrusor completo fig. 3.4 y posicionarlo en la estructura del prototipo fig. 3.5.



A: Pistón C: Licuefactor
B: Barra rosada D: mPaP unipolar

Figura 3.4: Extrusor ensamblado.
Fuente: Elaboración propia.



A: Pistón C: Termocupla K
B: Licuefactor D: Nozzle

Figura 3.5: Extrusor completado.
Fuente: Elaboración propia.

3.3.1. Alimentación del material

El tipo de alimentación del material plástico PET picado es manual, a través de una pseudo-tolva, luego es transportado con la técnica **Por Inyección** fig. 2.2(a) hacia el Licuefactor.

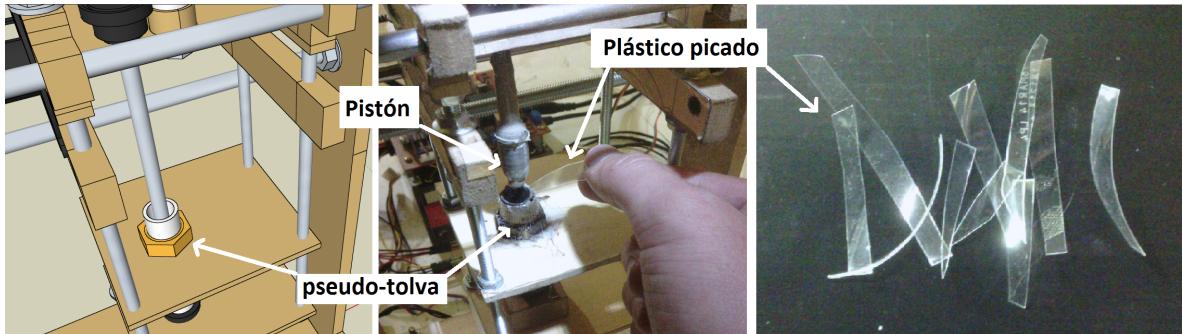


Figura 3.6: Alimentación del material plástico PET.

Fuente: Elaboración propia.

3.3.2. Calentamiento por Efecto Joule

En el proceso de licuefacción se dispone de la construcción del Licuefactor con un *nozzle* de bronce para hornillas a gas y una resistencia eléctrica tipo niquelina, todo debe ser aislado de forma eléctrica y calorífica fig. 3.7.

El calentamiento por Efecto Joule se da cuando se controla la cierta cantidad de corriente eléctrica a través de la niquelina lo que provoca que el Licuefactor aumente su temperatura considerablemente.



Figura 3.7: Licuefactor eléctrica-caloríficamente aislado.

Fuente: Elaboración propia.

3.3.3. Extrusión

La técnica de extrusión que se aplica es **Por Inyección** fig. 2.2(a), la barra rosada esta acoplada al engranaje 3 fig. 3.3(b) y transfiere el movimiento rotacional del motor en movimiento vertical lineal.

Por tanto, la presión interna necesaria para la extrusión del plástico es el resultado del movimiento vertical lineal del pistón y su área transversal de contacto respectivo.

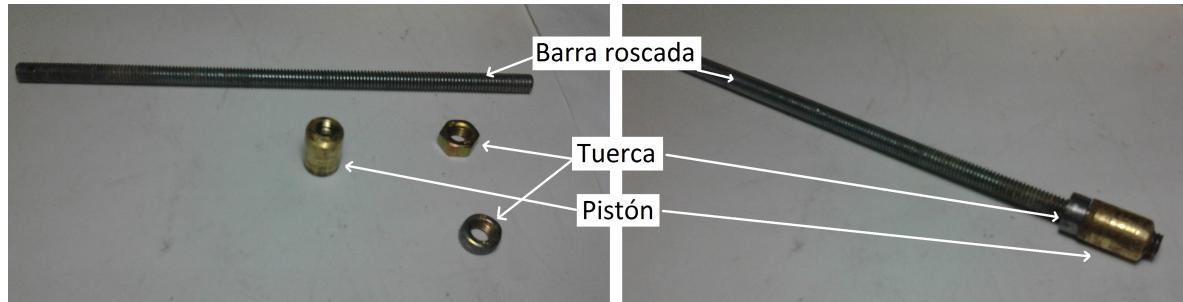


Figura 3.8: Construcción del pistón.

Fuente: Elaboración propia.

3.3.4. Robot cartesiano

La estructura completa del prototipo respeta la configuración del robot cartesiano, para cada eje existe un mPaP unipolar o bipolar.

En los 3 casos, el eje y/o caja de engranajes del motor tiene acoplado una banda dentada que transfiere el movimiento rotacional en lineal, las barras lisas permiten el movimiento lineal uniforme reduciendo la fricción y las abrazaderas conectan eje con eje para completar la configuración de robot cartesiano.

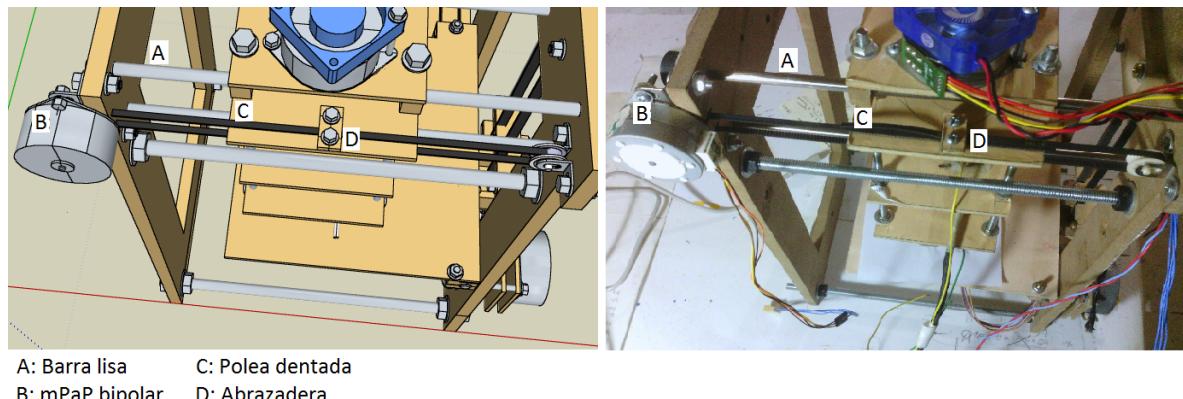
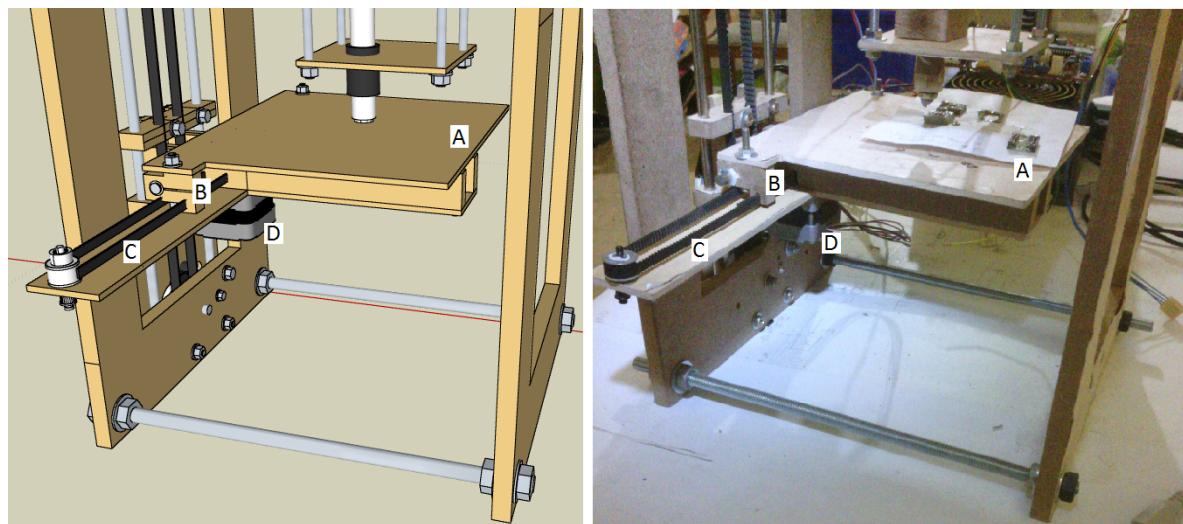


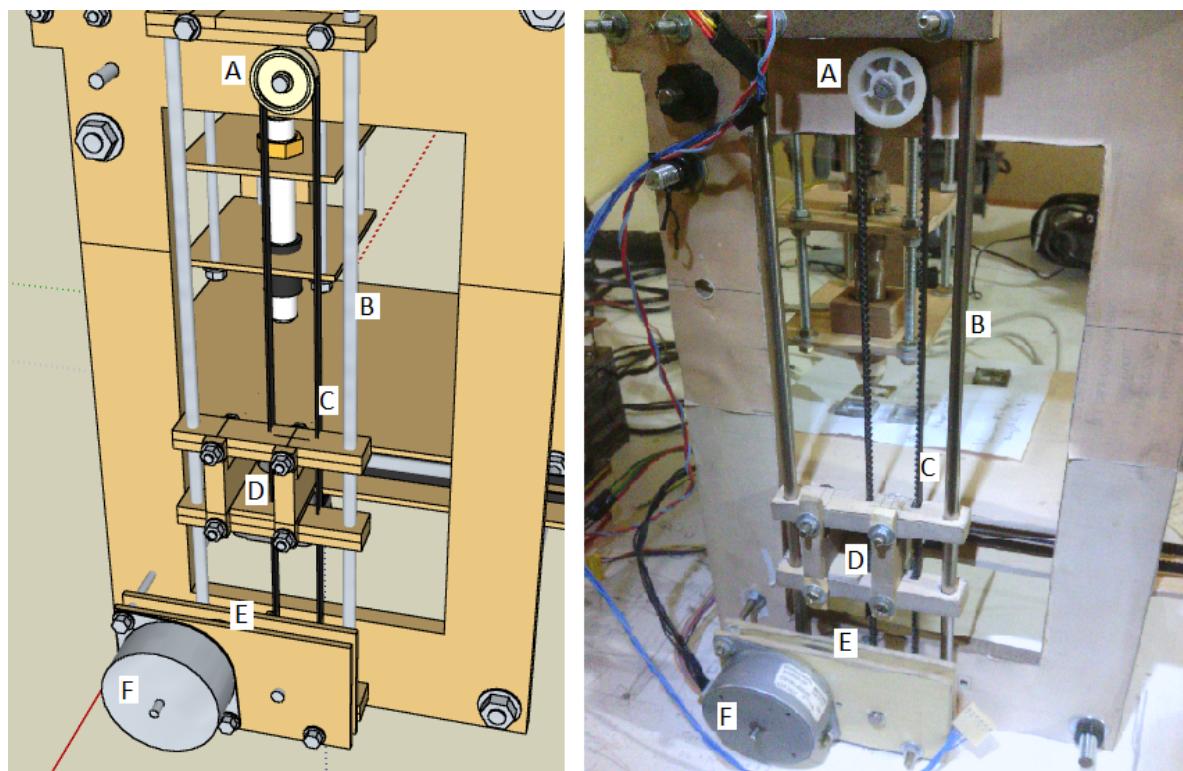
Figura 3.9: Eje/Motor X.

Fuente: Elaboración propia.



A: Plataforma C: Polea dentada
B: Abrazadera D: mPaP unipolar

Figura 3.10: Eje/Motor Y.
Fuente: Elaboración propia.



A: Engranaje C: Polea dentada E: Caja de engranajes
B: Barra lisa D: Abrazadera F: mPaP unipolar

Figura 3.11: Eje/Motor Z.
Fuente: Elaboración propia.

3.3.5. Posicionamiento 3D

Según la tabla 2.2 y la fig. 3.12, el prototipo completo es **Tipo A**, el extrusor se mueve en el eje X y la base en los ejes YZ.

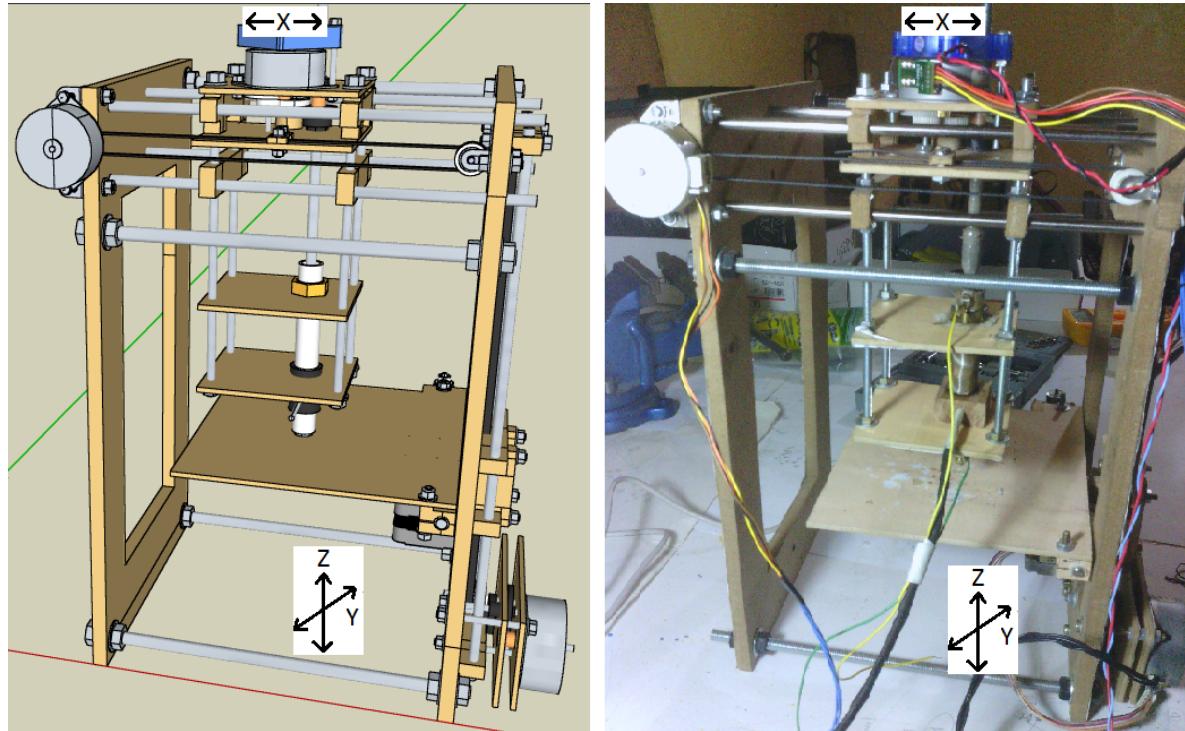


Figura 3.12: Posicionamiento 3D del prototipo completo.
Fuente: Elaboración propia.

3.4. Diseño electrónico

3.4.1. MCU PIC18F4550

La ejecución del *Firmware* está a cargo del MCU PIC18F4550, las ventajas de su empleo ya fueron descritas en la sección 2.2.1.

El diseño electrónico de la fig. 3.13 contiene las siguientes características:

- Conector JPIIC: Alimentación de voltaje +5[V] y GND común.
- Entradas:
 - Pulsador MRCL (*MasterClear*) = PIN 1, reset al PIC18F4550.
 - Pulsador BOOT = PIN 6, inicio/parada del *bootloader*.
 - **Cristal 20MHz** = PIN 13, PIN 14.

- **USB sense** = PIN 35, verificación de conexión/desconexión del cable USB.
 - Datos USB D- = PIN 23.
 - Datos USB D+ = PIN 24.
 - Sensor de temperatura **Termocupla** = PIN 2, puerto ADC 10 bits.
 - Salidas:
 - **Control** de temperatura por PWM = PIN 17, resolución de 10 bits.
 - Led **Pulsos** = PIN 39, verificación de número de pasos mPaP en el proceso de control manual.
 - Led **GlaGlc** = PIN 40, verificación de control de temperatura PID.
 - Dirección mPaP:
 - * **dirX, dirY, dirZ, dirE** = PIN 19, 20, 21, 22
 - Frecuencia (velocidad) mPaP:
 - * **clockX, clockY, clockZ, clockE** = PIN 27, 28, 29, 30

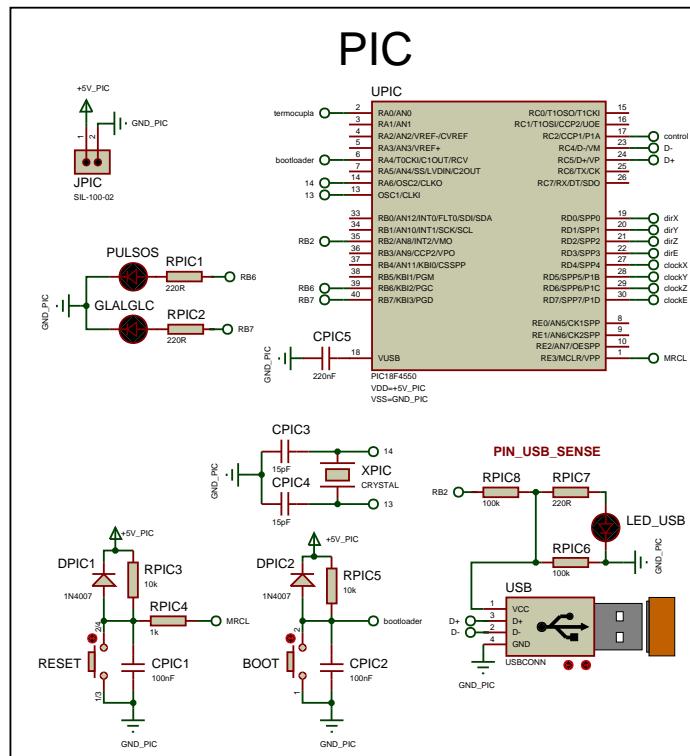


Figura 3.13: Esquema electrónico del MCU PIC18F4550.

Fuente: Elaboración propia.

3.4.2. Comunicación USB

Aunque el diseño fue mostrado en la sección anterior fig. 3.13, se debe notar que se respeta la recomendación del *datasheet* cuando el USB trabaja en modo de auto-alimentación, además de la inclusión del capacitor $CPIC5 = 220nF$ en PIN 18, con todo esto se asegura las corrientes mínimas de trabajo en el módulo USB.

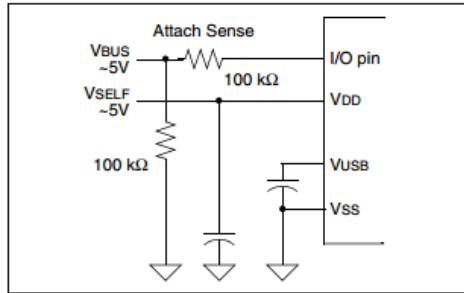


Figura 3.14: Recomendación del fabricante.

Fuente: *Datasheet* del fabricante [22] pág. 183

3.4.3. Control mPaP

Como se usa mPaP e-waste, se considera los circuitos de control para ambos tipos; Bipolar y Unipolar, porque no se tomó en cuenta el tamaño, marca o tipo cuando se seleccionó los mPaP.

El IC L297 es bastante conveniente para controlar ambos tipos y es de fácil adquisición en tiendas locales, la fig. 3.15 refleja el esquema electrónico recomendado por el fabricante para mPaP Unipolares y Bipolares.

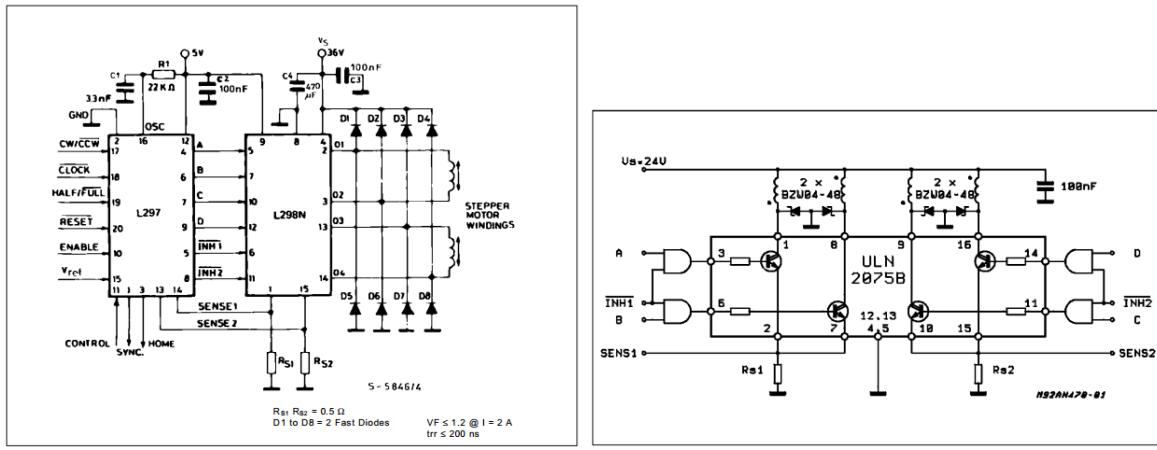


Figura 3.15: Diseño recomendado.

Fuente: *Datasheet* del fabricante [25] págs. 14-15

La fig. 3.16 muestra el diseño completo para los 4 mPaP, con las siguientes características:

- Conectores de alimentación de Voltaje $+5[V]$ y GND común: JX1, JY1, JZ1, JE1.
- Entradas en cada IC L297:
 - `dirX, dirY, dirZ, dirE.`
 - `clockX, clockY, clockZ, clockE.`
- Salidas por cada mPaP:
 - JX2, JY2, JZ2, JE2.
- En el caso Unipolar se cambia el ULN2075B por un array *Darlington* TIP122, respetando la fig. 3.15(b) y con una leve modificación en el mPaP Y.
- Regulador de voltaje VREF: RX5, RY5, RZ5, RE5, controlan la cantidad de corriente que pasa por las bobinas de cada mPaP.

3.4.4. Calentamiento por Efecto Joule

Para aplicar calentamiento por Efecto Joule en el Licuefactor se aplica control por PWM desde el MCU, la fig. 3.17 describe el esquema electrónico simple con las siguientes características:

- Conector JEX1: alimentación de Voltaje $+19,5[V] - 4[A]$ y GND común, necesario para alcanzar la Potencia de disipación en la resistencia (niquelina) $R = 6[\Omega]$ del Licuefactor fig. 3.7.
- Entrada de **control**: regula la corriente suministrada a $R = 6[\Omega]$ en función a la cantidad de voltaje otorgado por PWM del MCU PIC18F4550 a través del Mosfet QEX ILR3803.
- Salida JEX2: conector directo hacia la resistencia(niquelina) $R = 6[\Omega]$.

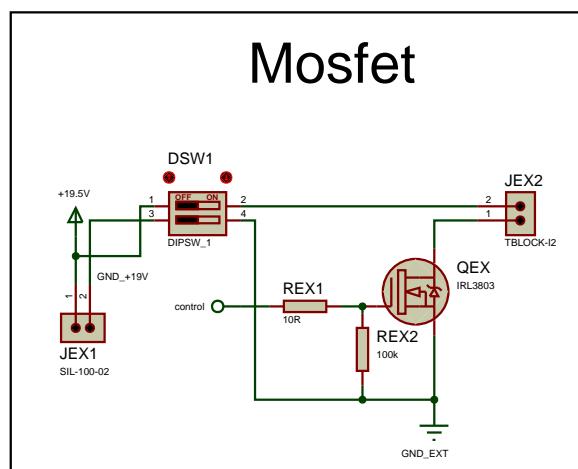


Figura 3.17: Calentamiento por Efecto Joule.
Fuente: Elaboración propia.

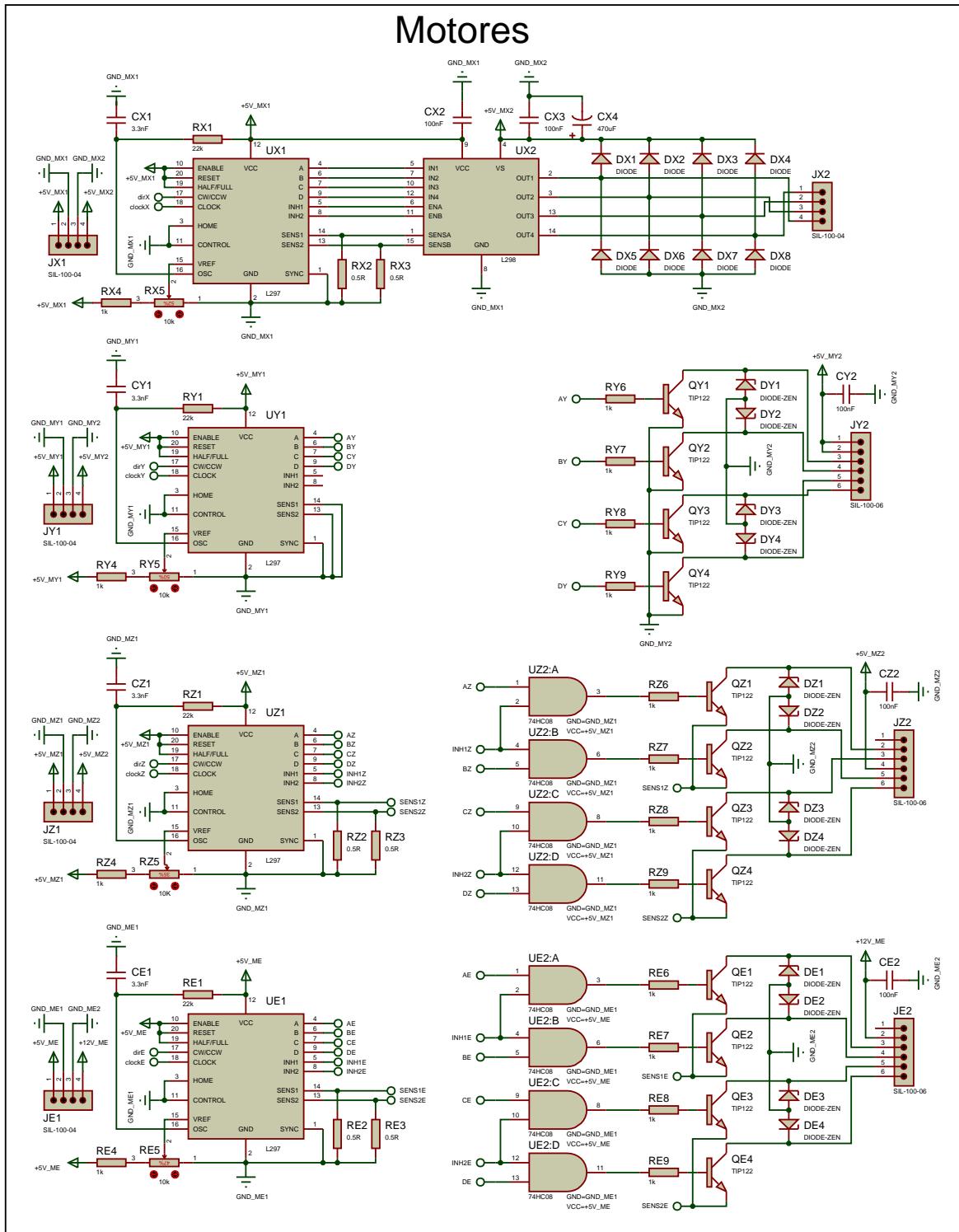


Figura 3.16: Esquema de control mPaP completo.
Fuente: Elaboración propia.

3.4.5. Acondicionamiento de señal de Termocupla Tipo K

Para lograr el acondicionamiento de señal y compensación analógica correctamente, se debe plantear el problema según la tabla 3.1 seguido del análisis y obtención de valores mediante la fig. 3.18 para el desarrollo de las ecuaciones posterior.

Cabe destacar que anteriormente se mostró un esquema de acondicionamiento en marco teórico fig. 2.14, pero hacía falta un bloque de ganancia para la termocupla G_2 , que se refleja en la fig. 3.18.

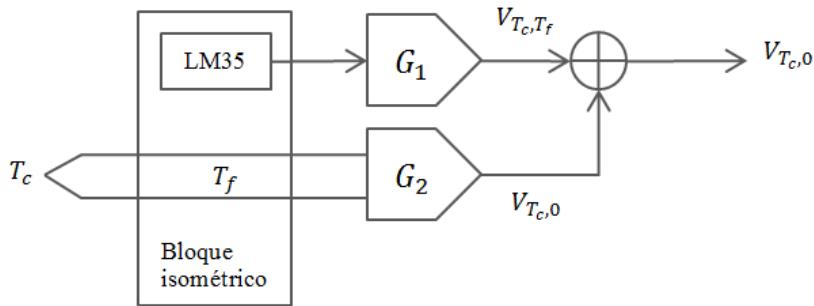


Figura 3.18: Acondicionamiento de Termocupla.

Fuente: Elaboración propia.

	Rango de Temperatura	Rango de Voltaje	Sensibilidad
Termocupla K	0 – 300[°C]	0 – 5[V]	$S_k = 40,5 \left[\frac{uV}{^{\circ}C} \right]$
LM35	0 – 150[°C]	0 – 5[V]	$\alpha = 10,0 \left[\frac{mV}{^{\circ}C} \right]$

Tabla 3.1: Condiciones de implementación.

Fuente: Elaboración propia.

Aplicando Linealización a partir de las condiciones de la tabla 3.1,

$$S = \frac{V_{max} - V_{min}}{T_{max} - T_{min}} \quad (3.1)$$

$$\begin{aligned} S_1 &= \frac{5 - 0}{150 - 0} \left[\frac{V}{^{\circ}C} \right] \\ &= 33,3 \left[\frac{mV}{^{\circ}C} \right] \end{aligned} \quad (3.2)$$

$$\begin{aligned} S_2 &= \frac{5 - 0}{300 - 0} \left[\frac{V}{^{\circ}C} \right] \\ &= 16,7 \left[\frac{mV}{^{\circ}C} \right] \end{aligned} \quad (3.3)$$

bloque de ganancias,

$$\begin{aligned} G_1 &= \frac{S_1}{\alpha} \\ &= \frac{33,3}{10,0} \left[\frac{\text{mV}}{\text{°C}} \right] \\ &= 3,33 \end{aligned} \tag{3.4}$$

$$\begin{aligned} G_2 &= \frac{S_1}{\alpha} \\ &= \frac{16,7}{40,5} \left[\frac{\text{mV}}{\text{°C}} \right] \\ &= 412 \end{aligned} \tag{3.5}$$

obteniendo valores de Resistencia para el **Amplificador Diferencial(OP07) = UT2** de Ganancia **G₁**

$$v_o = \frac{R_2}{R_1} (v_2 - 0) \tag{3.6}$$

$$\begin{aligned} G_1 &= \frac{v_o}{v_2} \\ &= \frac{R_2}{R_1} \\ &= 3,33 \end{aligned}$$

$$R_2 = 330[\Omega] \quad R_1 = 100[\Omega] \tag{3.7}$$

reemplazando en el diseño de la fig. 3.19

$$RT2 = RT4 = 330[\Omega] \quad RT1 = RT3 = 100[\Omega] \tag{3.8}$$

calculando los valores de Resistencia para el **Amplificador Diferencial de Instrumentación(AD620) = UT3** de Ganancia **G₂**, según la ecuación siguiente, obtenido de la fuente: *datasheet [28]*

$$\begin{aligned} R_G &= \frac{49,4[k\Omega]}{G_2 - 1} \\ &= \frac{49,4[k\Omega]}{412 - 1} \\ &= 120[\Omega] \end{aligned} \tag{3.9}$$

reemplazando en el diseño de la fig. 3.19

$$RG = 120[\Omega] \tag{3.10}$$

Para completar el último paso de la fig. 3.19 se debe aplicar la **Ley de Circuitos Homogéneos y Metales Intermedios** según,

$$V_{Tc,0} = V_{Tc,Tf} + V_{Tf,0} \tag{3.11}$$

y calculando los valores de Resistencia para el **sumandor UT4**,

$$v_o = \left(1 + \frac{R_2}{R_1}\right) \frac{\frac{v_1}{R_1} + \frac{v_2}{R_2}}{\frac{1}{R_1} + \frac{1}{R_2}}$$

si, $R_1 = R_2 = 10[k\Omega]$ entonces,

$$\begin{aligned} v_o &= v_1 + v_2 \\ v_{UT4} &= v_{UT2} + v_{UT3} \end{aligned} \quad (3.12)$$

Aplicando los valores obtenidos anteriormente se completa el esquema de la fig. 3.19, con las siguientes características.

- Conector JTCK1: Alimentación de voltaje $-12/+12[V]$ y GND común.
- Conector JTCK2 – JTCK3: Conexión con los bornes de la Termocupla K.
- Esquema OP07 = UT2: Amplificador operacional Diferencial, representa G_1 .
- AD620 = UT3: Amplificador Operacional de Instrumentación con rango de ganancia 1 a 1000, representa G_2 .
- Esquema OP07 = UT4: Sumador de UT2 y UT3, ec. (3.12)
- Esquema TL081 = UT5: Filtro pasa bajo de 1Hz.
- Salida **Termocupla**: Voltaje de salida total para el sensor de temperatura que se conecta directamente al PIC18F4550.

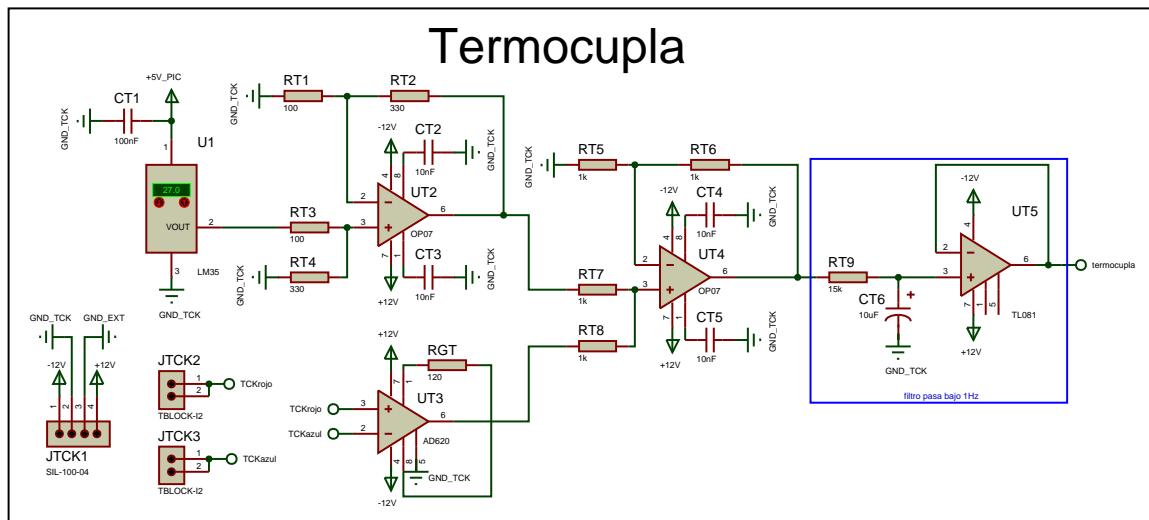


Figura 3.19: Diseño del sensor de Temperatura Termocupla K.

Fuente: Elaboración propia.

3.4.6. Modelo CAD y grabado PCB

Condiciones de diseño Antes de construir la Placa de Circuito Impreso – *Printed Circuit Board* (PCB) el modelo CAD debe respetar lo siguiente:

- El área del PCB no debe exceder demasiado las dimensiones de una fuente de poder ATX-PC.
- Las perforaciones de las tuercas de sujeción deben coincidir con los tornillos de la fuente de poder ATX-PC.
- Debe existir una altura minima de 2[cm] entre la altura máxima de la fuente ATX-PC y la base del PCB.
- De ser posible, se debe grabar sobre una placa de cobre doble cara.
- Usar cuantos componentes electrónicos e-waste se pueda; resistencias, capacitores y conectores.

Finalmente se implementa la electrónica completa y se muestra los resultados en la fig. 3.20.

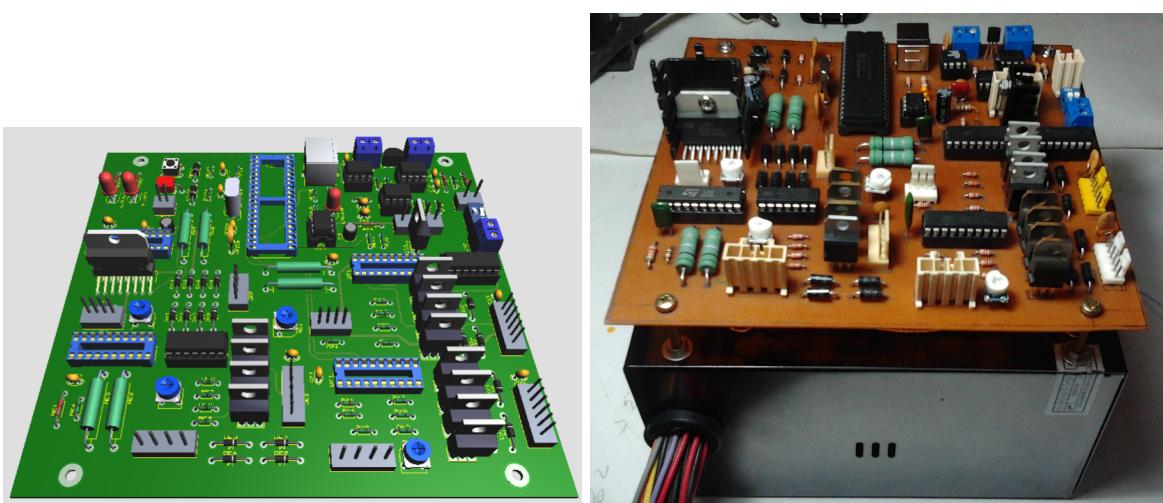


Figura 3.20: Implementación electrónica completa.

Fuente: Elaboración propia.

3.5. Análisis matemático para el procesamiento digital de datos

Antes del desarrollo de *Firmware-Software* es necesario describir las secciones vitales de análisis para implementar sus algoritmos.

3.5.1. Adquisición de datos por puerto ADC PIC18F4550

El primer paso es determinar la máxima resolución del puerto ADC PIC18F4550.

$$Resolución = \frac{V_{max} - V_{min}}{2^N - 1} \quad (3.13)$$

si ADC=10 bits y rango 0 - 5 [V]

$$\begin{aligned} Resolución &= \frac{5 - 0}{2^{10} - 1} \\ &= \frac{5}{1023} \\ &= 0,004887[V] \end{aligned} \quad (3.14)$$

Es decir, por cada dato digital(1 bit en decimal) hay 0,004887[V].

Ejemplo: si por puerto ADC entra 3[V] analógico, entonces su correspondiente voltaje digital es,

$$\begin{aligned} voltaje\ digital\ ADC &= \frac{3}{Resolución} \left[\frac{V}{V} \right] \\ &= \frac{3}{0,004887} \\ &= 613,87 \approx 614 \end{aligned}$$

comprobando,

$$\begin{aligned} voltaje\ analógico &= 614 \cdot Resolución \\ &= 614 \cdot 0,004887[V] \\ &= 3,0006[V] \approx 3[V] \quad \text{lqqd} \end{aligned}$$

3.5.2. Envío/Recepción de 10 bits a través de 2 Bytes USB PIC18F4550

El Envío/Recepción de paquetes USB del PIC18F4550 está limitado a 1 Byte máximo, lo que significa que no se puede enviar datos digitales mayores a 255 por vez.

Este problema es resuelto fácilmente con la partición del dato digital de 10 bits en 2 Bytes.

- Enviar por 2 Bytes desde el MCU PIC18F4550

dato digital ADC_a = dato digital ADC >> 8 ;desplazando 8 bits a la derecha

dato digital ADC_b = dato digital ADC AND 0xFF ;operación AND binario

Enviar Byte[0] = dato digital ADC_a

Enviar Byte[1] = dato digital ADC_b

- Recibir los 2 Bytes en el GUI

$$\text{dato digital ADC} = \text{Recibir Byte}[0] \cdot 256 + \text{Recibir Byte}[1]$$

Ejemplo: continuado el ejemplo anterior, si *voltaje digital ADC* = 614

- Enviar en 2 Bytes desde el MCU PIC18F4550

$$\text{voltaje digital ADC_a} = \text{voltaje digital ADC} >> 8$$

$$\begin{aligned}\text{voltaje digital ADC_a} &= 614 >> 8 \\ &= 10\ 0110\ 0110 >> 8 \\ &= 0000\ 0010 \\ &= 2\end{aligned}$$

$$\text{voltaje digital ADC_b} = \text{voltaje digital ADC AND } 0xFF$$

$$10\ 0110\ 0110$$

AND

$$\begin{aligned}xx\ 1111\ 1111 \\ &= 0110\ 0110 \\ &= 102\end{aligned}$$

$$\text{Enviar Byte}[0] = 2$$

$$\text{Enviar Byte}[1] = 102$$

- Recibir los 2 Bytes en el GUI

$$\text{voltaje digital ADC} = \text{Recibir Byte}[0] \cdot 256 + \text{Recibir Byte}[1]$$

$$= 2 \cdot 256 + 102$$

$$= 614$$

lqqd

3.5.3. Controlador PID

Conversión de escala

Antes de detallar la discretización es necesario una conversión de escala entre las etapas de Adquisición de datos ADC, resolución del PWM y el valor de referencia $r(t)$ enviado por el GUI.

La fig. 3.21 muestra el Envió/Recepción del dato de referencia $r(t)$ a través del módulo USB, donde el *Firmware* lo recibe en un rango [0-255], por tanto es necesario escalar la referencia $r(t)$ al mismo rango que la salida $y(t)$, según ec. (3.15).

$$\begin{aligned}\text{Relación de conversión} &= \frac{y(t)}{r(t)} = \frac{1023}{255} \\ &= 4,012\end{aligned}\tag{3.15}$$

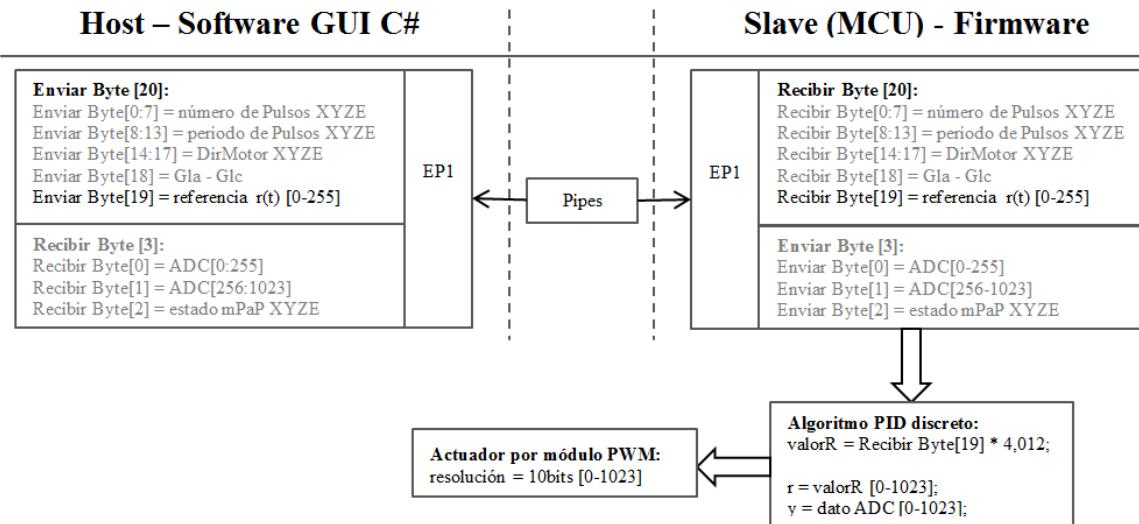


Figura 3.21: Conversión de escala en referencia $r(t)$, para procesar en el controlador PID.
Fuente: Elaboración propia.

Ejemplo: si por GUI deseo enviar 3[V], entonces según ec. (3.14) primero de debe obtener la Resolución de $r(t)$ en el GUI

$$\begin{aligned} \text{Resolución } r(t)_{GUI} &= \frac{5 - 0[V]}{2^8 - 1} \\ &= \frac{5}{255} \\ &= 0,01961[V] \end{aligned}$$

obteniendo su correspondiente voltaje digital,

$$\begin{aligned} \text{voltaje digital } r(t)_{GUI} &= \frac{3}{0,01961} \\ &= 152,9 \approx 153 \end{aligned}$$

$\text{voltaje digital } r(t)_{GUI}$ es enviado por Enviar Byte[19] a través de USB, el Firmware lo recibe por Recibir Byte[19] y lo almacena en valor R, fig. 3.21

$$\text{valorR} = \text{Recibir Byte}[19]$$

aplicando la ec. (3.15),

$$\begin{aligned} \text{valorR digital escalado} &= \text{valorR} \cdot 4,012 \\ &= 153 \cdot 4,012 \\ &= 613,84 \approx 614 \end{aligned}$$

para comprobar, se debe obtener su correspondiente valor de voltaje en escala [0-1023] según la ec. (3.14),

$$\begin{aligned} \text{valorR analógico escalado} &= 614 \cdot 0,004887[V] \\ &= 3,0006[V] \approx 3[V] \end{aligned} \quad \text{lqqd}$$

Discretización general

La aplicación del procedimiento tradicional genera una sobrecarga de procesamiento en el MCU pero es útil en la simulación Matlab; ec. (3.16), y su desarrollo está en Anexo C (5).

$$G_c(z) = \frac{(K_{pz} + K_{iz} + K_{dz}) z^2 + 2(K_{iz} - K_{dz}) z + (K_{pz} + K_{iz} + K_{dz})}{z^2 - 1} \quad (3.16)$$

Discretización por partes

Al aplicar este método, se obtiene 3 ecuaciones independientes que serán ejecutadas una a la vez hasta ser sumados en una ecuación final de control ec. (3.17), pero el aporte más importante es la identificación del **error en estado anterior** $e(kT - T)$ y el **valor integral en estado anterior** $i(kT - T)$. El desarrollo se encuentra en Anexo C (5).

$$u(kT) = p(kT) + i(kT) + d(kT) \quad (3.17)$$

3.5.4. Descriptor Gcode

El Descriptor Gcode tiene la función de obtener los datos de posición XYZ y velocidad F(*Feedrate*) de los mPaP, procesar dichos valores para enviar solamente **número de Pulses XYZ** y **Periodo XYZ** al *Firmware*, que lo ejecutará en los mPaP.

Para empezar, una instrucción simple Gcode es,

G1 X10.000 Y10.000 Z10.000 E10.000 F1200.000

donde G1 indica que la instrucción es de Movimiento Lineal Controlado y los valores XYZ corresponden a 10[mm] cada uno, con velocidad(*Feedrate*) F 1200 [$\frac{\text{mm}}{\text{min}}$].

El código ejemplo 3.1, indica el desarrollo de un rectángulo 2D sin el eje Z, donde el extrusor E deposita el material de acuerdo a las distancias XY, con una velocidad F 1200 [$\frac{\text{mm}}{\text{min}}$].

1	G1 X0.000 Y0.000
2	G1 X10.000 E10.000 F1200.000
3	G1 Y5.000 E5.000 F1200.000
4	G1 X10.000 E10.000 F1200.000
5	G1 Y5.000 E5.000 F1200.000

Código 3.1: Ejemplo Gcode, rectángulo 2D de 10x5 mm.

Fuente: Elaboración propia.

Aunque se necesitan mas instrucciones que dependen del tipo de **Slicer**, lo importante es entender la lógica de instrucciones **Gcode**.

Cada linea de instrucción refiere al *toolpath* necesario para completar el rectángulo 2D, fig. 3.22.

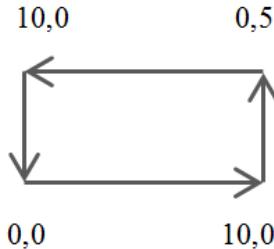


Figura 3.22: *Toolpath* del código 3.1.

Fuente: Elaboración propia.

Por tanto, el procedimiento de desarrollo del algoritmo descriptor consiste en:

- Guardar en *buffer* de memoria el Documento.gcode que contiene todas las instrucciones Gcode.
- Buscar la instrucción de Movimiento Lineal Controlado.
- Procesar el campo numérico para guardar en variables temporales XYZEF.
- Realizar el escalamiento de distancias XYZE, de acuerdo a la siguiente relación matemática.

	1[mm]	$10 \left[\frac{\text{mm}}{\text{s}} \right] \approx 20[\text{ms}]$
X	3,03[pulsos]	
Y	10[pulsos]	
Z	50[pulsos]	
E	70[pulsos]	
F		$\frac{F \cdot 2}{2 \cdot 2 - 60 \cdot 20} \cdot 20[\text{ms}]$

Tabla 3.2: Condiciones de relación (pulsos vs mm).

Fuente: Elaboración propia.

- Realizar el promedio de las distancias y periodos XYE para calcular la ecuación de relación, tomando como referencia al periodo del motor E, tabla 3.3.
- Sin embargo, no es necesario calcular la relación agregando el eje Z, porque su movimiento solo se ejecuta cuando termina una capa.

	X	Y	E
[mm]	$d_{X1} \cdot 3,03 = d_{X2}$	$d_{Y1} \cdot 10 = d_{Y2}$	$d_{E1} \cdot 70 = d_{E2}$
[ms]	$T_{X1} \cdot 3,03 \cdot 7 = T_{X2}$	$T_{Y1} \cdot 7 = T_{Y2}$	$T_E = T_{ref}$

Tabla 3.3: Condiciones de relación (mm vs periodo).

Fuente: Elaboración propia.

- Relación XYE

$$d_{X2} \cdot T_{X2} = d_{Y2} \cdot T_{Y2} = d_{E2} \cdot T_E \quad (3.18)$$

$$T_{Y2} = \frac{d_{E2}}{D_{Y2}} \cdot T_E \quad (3.19)$$

$$T_{X2} = \frac{d_{E2}}{D_{X2}} \cdot T_E \quad (3.20)$$

3.6. Desarrollo de *Firmware*

En el desarrollo del *Firmware* se implementa **Programación Imperativa y Estructurada**, la razón se debe a la aplicación del Lenguaje de Programación C, además de la técnica Multitarea a través de un RTOS *open-source*. El código completo se encuentra en Anexo A (5).

3.6.1. Estructura general

Para empezar, el algoritmo 3.1 describe en pseudo-código la estructura base. En config hay 4 grupos grandes delimitados, MCU y Comunicación USB que están implícitamente relacionados, en cambio RTOS depende del correcto funcionamiento de su propio grupo de instrucciones y la correspondiente compatibilidad con el MCU.

Hay que resaltar la omisión del clásico `while loop` en `main`; se prescinde de él porque un RTOS gestiona mejor las tareas de operación, control y comunicación, la sección 2.3 explica los beneficios de su aplicación.

3.6.2. Cabecera de configuración MCU PIC18F4550

Condiciones de implementación La configuración inicial MCU se desarrolla de acuerdo a:

- Incluir:
 - Librerías PIC18F4550, USB y OSA-RTOS.
 - *bootloader* del Compilador PCWHD.
- Definir:

```

Input: Puertos I/O
Output: Actuadores
config
  ┌── MCU;
  ├── RTOS;
  ├── Comunicación USB;
  └── Control;

def
  ┌── variables;
  ├── funciones;
  └── tareas RTOS;

main
  ┌── call funciones;
  └── call tareas RTOS;

```

Algoritmo 3.1: Estructura general. Fuente: Elaboración propia.

- Cristal de oscilación externo 20 MHz y *clock* interno 48 MHz.
- ADC de 10 bits.
- Tipos de constantes y variables de funciones genéricas.
- TareasOSA-RTOS.
- *Timer* para OSA-RTOS.

3.6.3. Función principal: main

OSA-RTOS [29] es un RTOS *open-source* que sirve bastante bien para el propósito del proyecto:

- Multi-tarea con tipo de *kernel* Cooperativo y Preferencial.
- Ofrece documentación clara e implementación fácil.

Condiciones de implementación El algoritmo 3.3 refleja la función principal `main`, según:

- Inclusión de librería *kernel <osa.h>* : *Scheduler*, Servicios, operación de Tareas, sincronización y mensajes.
- Definición, creación y prioridad de tareas RTOS.
- Iniciar servicios RTOS.
- Definición del *Timer0 = 1 kHz* para RTOS.

```

config PIC
  include <18F4550.h>;
  ADC = 10 bits;
  delay(cristal = 20MHz, clock = 48MHz);
  Directivas I/O (puerto A, B, C, D);
  include <usb_bootloader.h>PCWHD;
  include <osa.h>OsaRTOS;

config USB
  Transferencia Bulk;
  Tamaño bufferEndPoint 1 de recepción = 20[Bytes];
  Tamaño bufferEndPoint 1 de envió = 3[Bytes];
  include usb_descriptor.h
    clase = CC;
    PID = 0x0011;
    VID = 0x04D8;
    name = 3D REPET PL;

def variables tarea USB
  Recibir Byte[20] = [0:19] 20 vectores x 1 Byte (0-255);
  Enviar Byte[3] = [0:2] 3 vectores x 1 Byte (0-255);

def variables tareas mPaP
  número de Pulso(X, Y, Z, E);
  periodo de Pulso(X, Y, Z, E);
  dir mPaP(X, Y, Z, E);

def variables funciones genéricas
  Gla - Glc;
  valor referencia, salida, control;

def tareas OsaRTOS
  tarea USB();
  tarea Gla();
  tarea Glc PID();
  tarea mPaP X();
  tarea mPaP Y();
  tarea mPaP Z();
  tarea mPaP E();

def Timer 0 para OsaRTOS
  OSATimer = Timer0;

```

Algoritmo 3.2: Cabecera de configuración MCU PIC18F4550. Fuente: Elaboración propia.

3.6.4. Función genérica: Inicio

Condiciones de implementación Es la primera función genérica a llamar que inicializará variables y módulos del MCU, algoritmo 3.4.

- Variables mPaP y PID.

```

main
  call Inicio();
  call Iniciar OsaRTOS;
  create tarea USB, def prioridad alta;
  create tarea Gla, def prioridad media;
  create tarea Glc PID, def prioridad media;
  create tarea mPaP X, def prioridad baja;
  create tarea mPaP Y, def prioridad baja;
  create tarea mPaP Z, def prioridad baja;
  create tarea mPaP E, def prioridad baja;

```

Algoritmo 3.3: Función principal: **main**. Fuente: Elaboración propia.

- Módulo CCP para utilizar Timer2 con frecuencia PWM = 3kHz, según:

$$\begin{aligned}
 PWM &= (Pr2 + 1) 4 \cdot Tosc(\text{Prescaler TMR2}) \quad \text{fuente: datasheet [22]} \\
 PWM &= (249 + 1) 4 \cdot \frac{1}{48MHz} \cdot 16 \\
 PWM &= 3kHz
 \end{aligned} \tag{3.21}$$

- ADC con canal AN0.
- Timer0 para OsaRTOS con frecuencia 1kHz.
- Módulo USB.

3.6.5. Función genérica: ADC

Condiciones de implementación El proceso de adquisición de datos comienza con el muestreo del módulo ADC.

- En cabecera de configuración MCU se definió ADC = 10 bits = [0-1023] pero el tamaño máximo por cada vector **Enviar/Recibir Byte**[20] es de 1 Byte = 8 bits = [0-255].
- Por tanto, es necesario partir en 2 Bytes el dato ADC para ser reconstruido en el GUI, según 3.5.2 y finalmente desarrollar el algoritmo 3.5.

Function ADC()

```

  valor digital ADC = Leer ADC();
  Enviar Byte[0] = valor digital ADC >> 8;
  Enviar Byte[1] = valor digital ADC AND 0xFF;

```

Algoritmo 3.5: Función ADC. Fuente: Elaboración propia.

3.6.6. Tarea RTOS: Comunicación USB/descriptor Bulkmode

Condiciones de implementación La fig. 3.23 muestra en diagrama de bloques los Bytes de Envío/Recepción de paquetes USB y se implementa el algoritmo 3.6, según:

Function Inicio()

```

ini valores mPaP
  |  número de Pulsos XYZE = 0;
  |  periodo de Pulsos XYZE = 0;

ini constantes PID
  |  i1 = e1 = d1 = 0;
  |  Kd = 4,372  Kp = 4,387  Ki = 0,01457;
  |  Tiempo de muestreo: T = 10[s];
  |   $K_{pz} = K_p$    $K_{iz} = K_i \frac{T}{2}$    $K_{dz} = \frac{K_d}{T}$  ;

config CCP
  |  Frecuencia_PWM = 3kHz;

config ADC
  |  Canal 0 = AN0;

config Timer0 para OsarRTOS
  |  Frecuencia = 1kHz para clock 48MHz;

config USB
  |  Habilitar módulo USB;
  |  Enumeración USB;

```

Algoritmo 3.4: Función genérica: Inicio. Fuente: Elaboración propia.

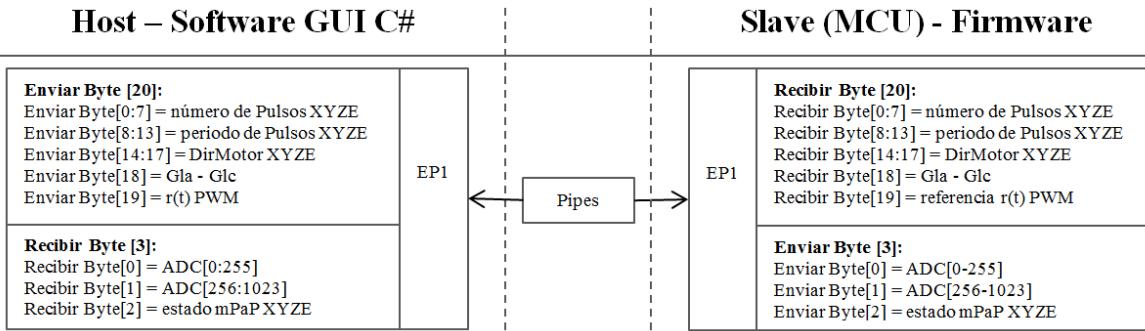


Figura 3.23: Envío/Recepción de paquetes USB.

Fuente: Elaboración propia.

- Configuración `usb_descriptor.h`
- Definición de tamaño *Buffer EndPoint 1* para Recibir/Enviar Datos USB.
- Tarea USB:
 - Recibir datos USB (mPaP, Gla-Glc, referencia $r(kT)$).
 - Enviar datos USB (ADC, cable USB).

```

Function tarea USB()
  if Recibir Byte >0 then
    número de Pulses XYZE = Recibir Byte[0:7];
    periodo de Pulses XYZE = Recibir Byte[8:13];
    DirMotor XYZE = Recibir Byte[14:17];
    dato Gla - Glc = Recibir Byte[18];
    referencia r(kT) PWM = Recibir Byte[19];
  if EnviarByte >0 then
    dato ADC[0-255] = Recibir Byte[0];
    dato ADC[256-1023] = Recibir Byte[1];
    estado mPaP XYZE = Recibir Byte[2];
  delay = 100 [ms];

```

Algoritmo 3.6: Tarea RTOS: Comunicación USB. Fuente: Elaboración propia.

3.6.7. Tarea RTOS: Control mPaP

Condiciones de implementación :

- Definición de variables para los 4 mPaP; número y periodo de Pulses, sentido de giro.
- Definir 4 tareas RTOS, mPaP XYZE.
- *delay* = Periodo de Pulses [ms]; determina la velocidad del mPaP.
- número de Pulses; determina la distancia recorrida por cada mPaP.

```

/* estructura base para las 4 tareas mPaP (X, Y, Z, E) */
Function tareas mPaP
  if número de Pulses >0 then
    if dir mPaP = horario then
      girar horario;
    if dir mPaP = antihorario then
      girar antihorario;
  delay = Periodo de Pulses [ms];

```

Algoritmo 3.7: Control mPaP. Fuente: Elaboración propia.

3.6.8. Tarea RTOS: Control PID discreto de temperatura

Para implementar Control PID se debe obtener el modelo de la Planta(Licuefactor) $G_p(s)$, se analiza los requerimientos de diseño, simulación en Matlab y finalmente según el algoritmo 3.8 se implementa el controlador PID.

Análisis

Aunque se ve posteriormente, el *Software GUI* también cumple la función de obtener la respuesta al escalón del Licuefactor $G_p(s)$ registrando en un **documento.txt** los valores de sensor de Temperatura Termocupla K cada Periodo de Tiempo de muestreo $T_m[ms]$.

Aplicando una señal escalón a G_p a lazo abierto G_{la} se obtiene la fig. 3.24.

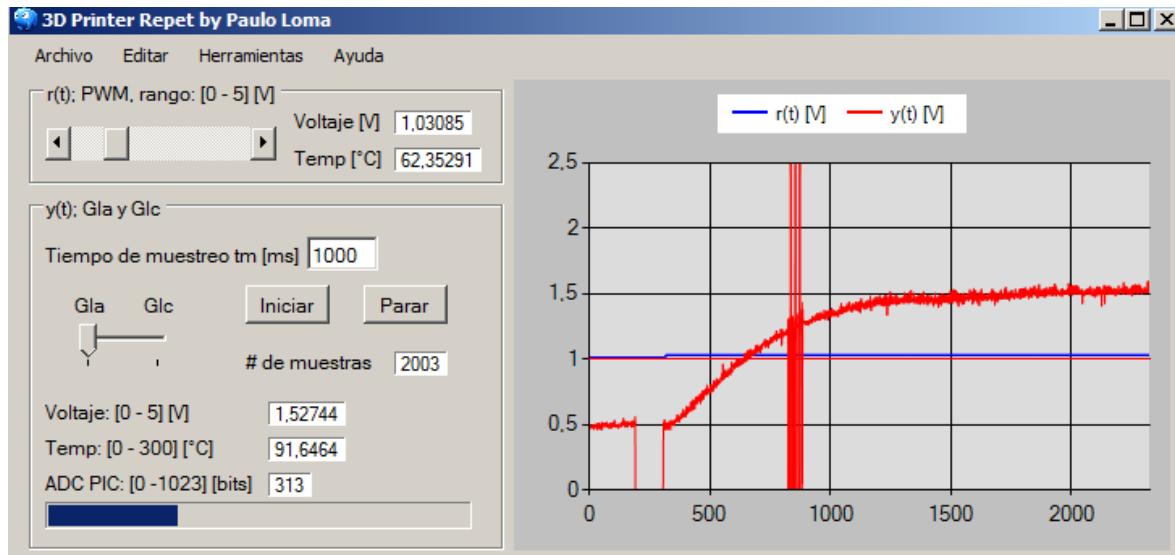


Figura 3.24: Respuesta G_p al escalón, muestreado con el GUI.

Fuente: Elaboración propia.

Que exporta un **documento.txt** con $T_m = 1000[ms]$ para ser analizado en **Matlab**, aunque el código 3.2 no refleja todas las muestras basta con mostrar algunas líneas.

```

1 Datos de la Planta a lazo abierto (Gla) o lazo cerrado (Glc)
2 C:\Users\Paulo\Documents\PAULO\CONTROL\ETN1040\3D printer\3D PRINTER PROJECT\
   analisis\datos recolectados\Gp_tm1000ms_19V_3kHz.txt
3 28/02/2014 01:54:19 p.m.
4
5 r(t): Voltaje [V]    y(t): Voltaje [V]    y(t): Temp [°C]    Tiempo de muestreo: tm [ms]
6 1,03085      0,49288      29,5728      1316,2057
7 1,03085      0,47824      28,6944      2330,2075
8 1,03085      0,4636       27,816       3344,2093
9 1,03085      0,488        29,28        4498,6113
10 1,03085     0,47336      28,4016      5372,2129

```

Código 3.2: Gp a lazo abierto, Gla.

Fuente: Elaboración propia.

Importando con la herramienta **Ident Matlab** para identificación de modelos SISO.

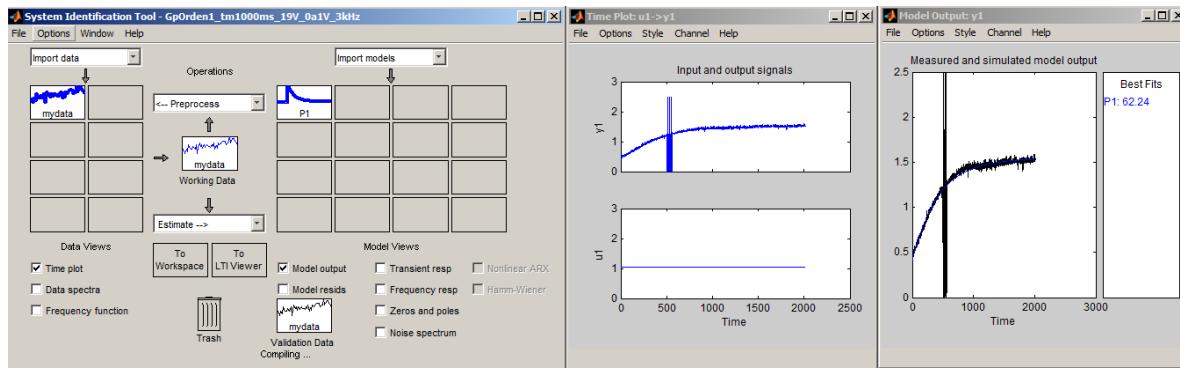


Figura 3.25: Identificación del modelo con Ident Matlab.

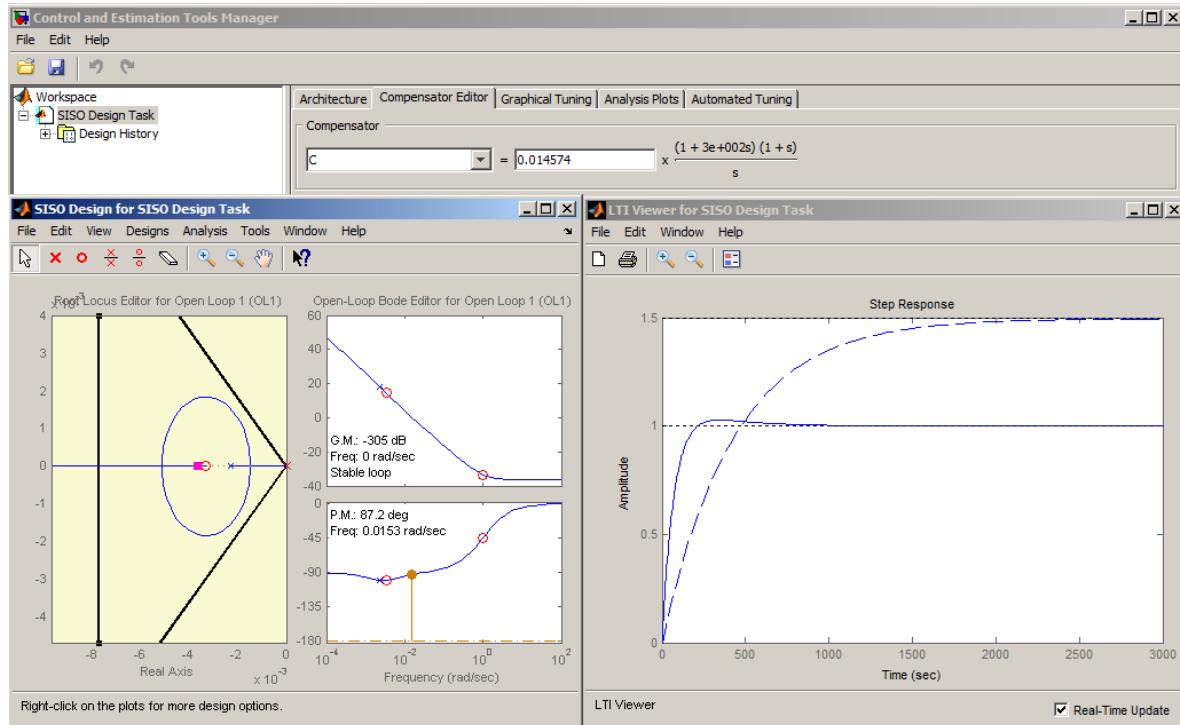
Fuente: Elaboración propia.

Finalmente genera la Función de Transferencia,

$$G_p(s) = \frac{1,4958}{1 + 429,15 s} \quad (3.22)$$

Diseño

Gracias a otra herramienta de diseño de controlador; **Sisotool Matlab**, se genera los parámetros y Función de Transferencia del Controlador $G_c(s)$, fig. 3.26

Figura 3.26: Controlador $G_c(s)$ con Sisotool.

Fuente: Elaboración propia.

Que resulta en:

$$G_c(s) = 0,014574 \frac{(1+s)(1+300s)}{s} \quad (3.23)$$

$$\begin{aligned} K_d &= 4,372 & K_p &= 4,387 & K_i &= 0,01457 \\ M_p &= 2,75\% & t_r &= 127[s] & t_s &= 507 \end{aligned}$$

Simulación

Se discretiza G_p y G_c según el código Matlab 3.3 y se obtiene la fig. 3.27.

```

1 s=tf('s');
2 Gp=1.4958/(1+429.15*s);
3 Kd=4.372; Kp=4.387; Ki=0.01457;
4 Gc=tf([Kd Kp Ki],[1 0]); % Gc=0.014574*(1+s)*(1+300*s)/s
5 T=10; % tr/6 < T < tr/20
6 Kpz=Kp; Kiz=Ki*T/2; Kdz=Kd/T;
7 Gpz=c2d(Gp,T,'zoh');
8 Gcz=tf([(Kpz+Kiz+Kdz) (-Kpz+Kiz-2*Kdz) Kdz],[1 -1 0],T);
9 Glaz=Gpz*Gcz; Gla=Gp*Gc;
10 Glcz=feedback(Glaz,1); Glc=feedback(Gla,1);
11 step(Gpz,Glcz,Glc)
12 pause
13 margin(Glcz)
14 pause
15 bode(Glcz,Glc)

```

Código 3.3: Sistema a lazo abierto Gla, y lazo cerrado Glc.

Fuente: Elaboración propia.

Implementación

Para implementar un algoritmo PID 3.8 en un MCU es necesario discretizar la ecuación general PID, ec. (1), el motivo se debe a que el MCU debe ejecutar el algoritmo en periodos de **Tiempo Discreto** $T[ms]$, el resultado se muestra en la fig. 3.28.

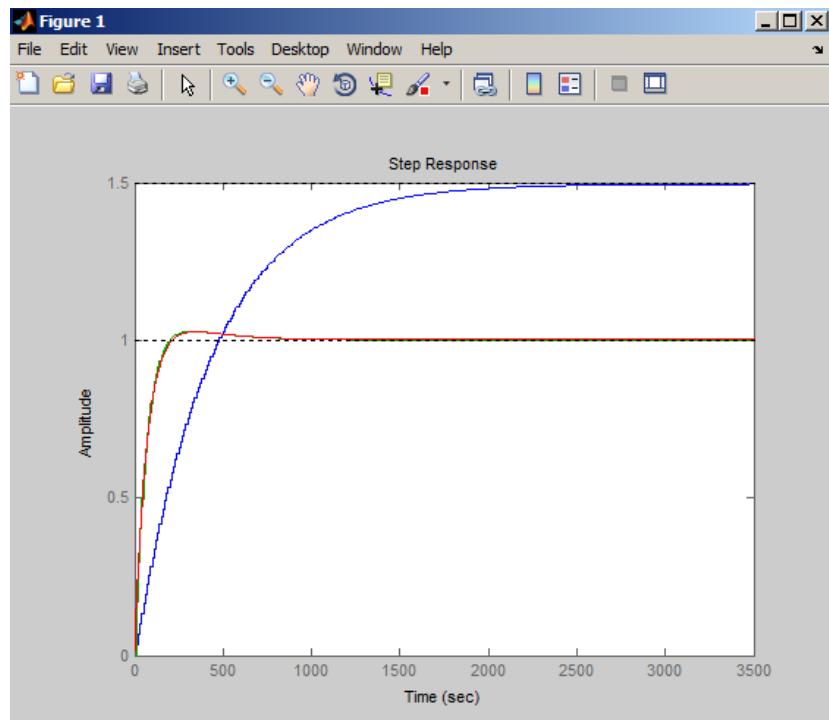


Figura 3.27: Simulación Matlab.
Fuente: Elaboración propia.

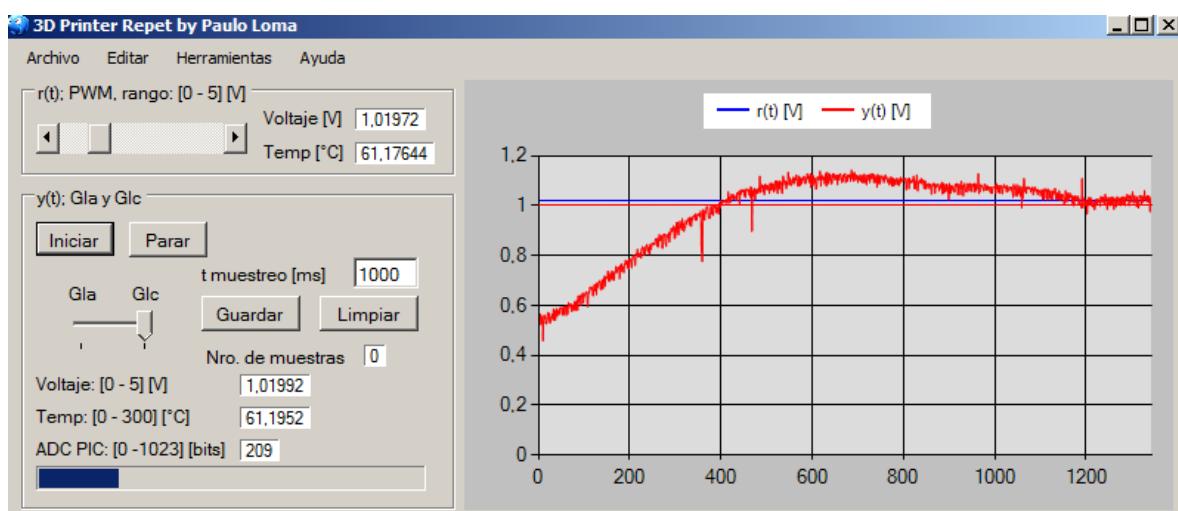


Figura 3.28: Respuesta al escalón del sistema a lazo cerrado G_{lc} .
Fuente: Elaboración propia.

Input: Periodo $T[ms]$, constantes K_{pz}, K_{iz}, K_{dz} , referencia $r(kT)$ y salida $y(kT)$

Output: Controlador $u(kT)$

def

$e(kT) = e$	$e(kT - T) = e1;$
$r(kT) = r$	$y(kT) = y;$
$p(kT) = p;$	
$i(kT) = i$	$i(kT - T) = i1;$
$d(kT) = d;$	
$u(kT) = u;$	

Function control Glc_PID

```

e = r - y ; /* leer error actual */  

p = Kpz · e;  

i = i1 + Kiz · (e + e1);  

d = Kdz · (e - e1);  

/* suma pid */  

u = p + i + d;  

/* enviar dato de control "u" al actuador */  

actuador(u);  

/* error actual se guarda en error anterior y valor integral actual en  

   valor integral anterior */  

e1 = e;  

i1 = i;  

/* Periodo de muestreo */  

delay = T[ms];

```

Algoritmo 3.8: Discretización PID por partes. Fuente: Elaboración propia.

3.7. Desarrollo de *Software*

Condiciones de Implementación El *Software* se realiza aplicando POO a través de Visual Studio C# con diseño de interfaz WFA, la fig. 3.29 muestra el entorno de trabajo con 5 secciones definidas:

- **Referencia r(t)** por PWM: rango de $0 - 5[V] = 0 - 300[^{\circ}C]$.
- **Salida y(t)**: elección *Gla-Glc*, Iniciar/Para Muestreo, tiempo $t_m[ms]$, rango de salida por voltaje $0 - 5[V]$, temperatura $0 - 300[^{\circ}C]$ y $0 - 1023[dec]$.
- **Control Manual mPaP XYZ** con 2 subgrupos: número de Pulso(distancia), periodo de Pulso(velocidad).
- **Gcode**: cargar, instrucción por vez step, reset, timer, enviar una instrucción.
- **Monitoreo**: referencia r(t) vs salida y(t), a través de un *chart* o *plotter*.

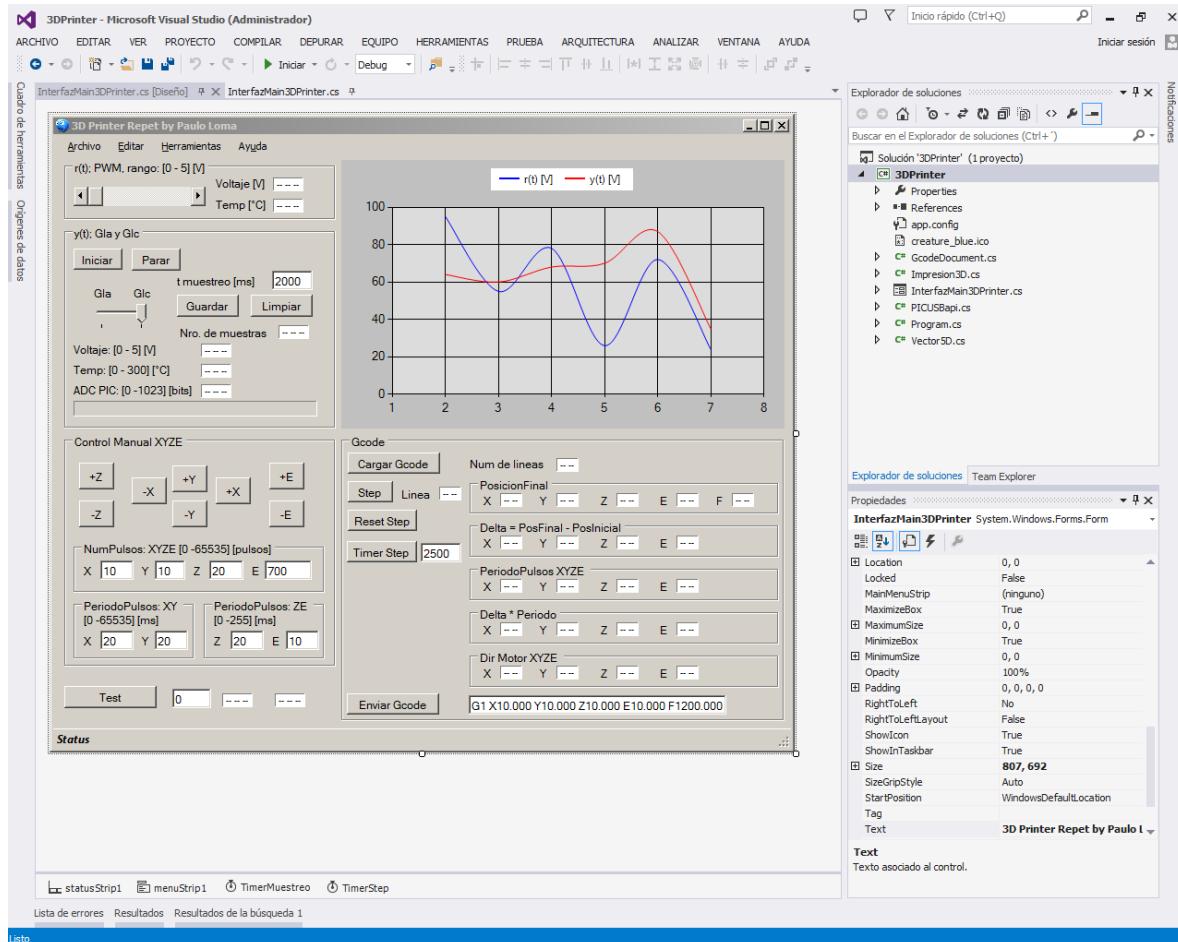


Figura 3.29: Entorno de trabajo en Visual Studio

Fuente: Elaboración propia.

Gracias a la herramienta Mapa de Código DGML de Visual Studio, se presenta en diagrama de bloques relacional las clases implementadas; fig. 3.30, con la siguiente descripción:

- Ejecutable 3D Printer Repet.exe: compilado en .NET4.5 x86.
- ControlUSB: nombre de espacio general.
- Clase InterfazMain3DPrinter: principal enlace al GUI, es preciso aclarar que Visual Studio genera una función main automáticamente para dibujar y enlazar el GUI.
- Clase PICUSBapi: procesamiento Enviar/Recibir paquetes USB, EndPoint1 y VID/PID=04D8/0011.
- Clase GcodeDocument: creación documento Gcode para procesar con Impresion3D.
- Clase Impresion3D: procesamiento Gcode para la impresión de la pieza.
- Estructura Vector5D: creación del vector [X, Y, Z, E, F] para procesar con Impresion3D.

Cada uno de los puntos anteriores será desarrollado y descrito de acuerdo a la sección que corresponda.

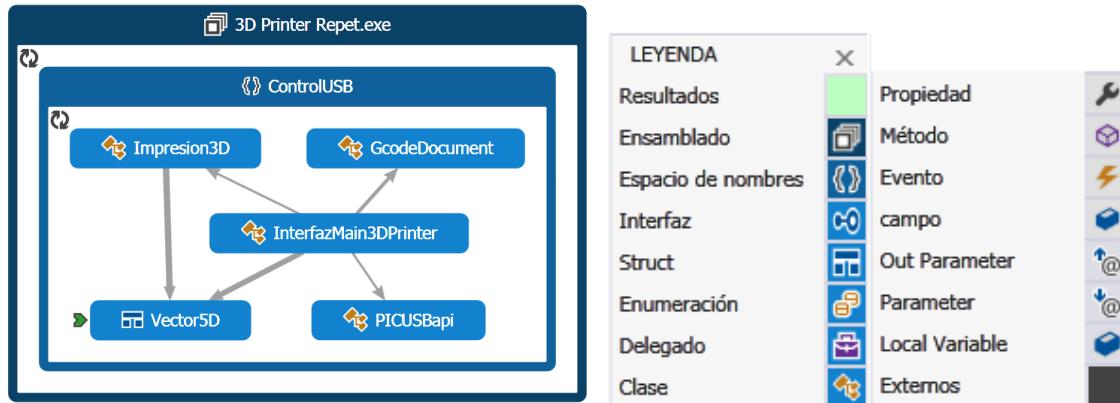


Figura 3.30: DGML general.
Fuente: Elaboración propia.

3.7.1. Clase: Main GUI

Condiciones de implementación El desarrollo de la clase principal contiene los atributos y métodos que relacionan las 5 secciones de la fig. 3.29 y las clases de la fig. 3.30, finalmente se desarrolla el algoritmo 3.9.

- Definir los vectores Envió/Recepción de paquetes USB.
- Agrupar Métodos necesarios que relacionen a las 5 secciones.
 - Grupo Genérico.
 - Grupo Control Manual mPaP XYZEF.
 - Grupo Gcode.
 - Grupo Monitoreo.
- Métodos independientes:
 - Enviar_Data.
 - Referencia r(t) PWM.
 - Selección Gla - Glc.

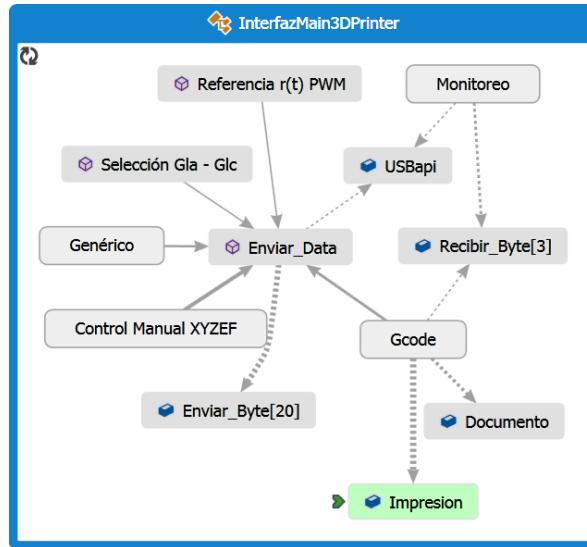


Figura 3.31: DGML Clase principal InterfazMain3DPrinter.

Fuente: Elaboración propia.

3.7.2. Clase Main GUI/Grupo Método: Genérico

Condiciones de implementación

- Verificar estado de Conexión/Desconexión de la Impresora 3D.
- Al cerrar la aplicación se debe limpiar el *buffer* de datos USB.

```

class InterfazMain3DPrinter
    method Estado Dispositivo()
        if conexión cable USB = true then
            mostrar texto = Conectado;
        if desconexión cable USB = true then
            mostrar texto = Desconectado;

    method Cerrar Aplicación()
        if cerrar = true then
            Enviar_Data = (Enviar_Byte[0:20] = 0);
    
```

Algoritmo 3.10: Grupo Método: Genérico. Fuente: Elaboración propia.

3.7.3. Clase Main GUI/Grupo Método: Control Manual mPaP XYZEF

Condiciones de implementación Es necesario definir 2 áreas importantes; distancia y velocidad, fig. 3.32.

- Distancia que recorren los mPaP en base a número de pulsos enviados:

```

class InterfazMain3DPrinter
  attributes USB
    Enviar_Byte[20];
    Recibir_Byte[3];
  attributes Monitoreo
    ADC;
    Dirección documento.txt;
    Contador muestreo;
    tiempo de muestreo inicial;
    tiempo de muestreo final;
  /* creación de objetos, fig. 3.30 */
  attributes objetos
    PICUSBapi USBapi = new PICUSBapi();
    GcodeDocument Documento = new GcodeDocument();
    Impresion3D Impresion = new Impresion3D();
  constructor InterfazMain3DPrinter()
    Iniciar GUI();
  method
    Grupo Genérico: Estado Impresora 3D(), Cerrar Aplicación();
    Enviar_Data();
    Control Manual mPaP XYZEF();
    Selección Gla - Glc();
    Referencia r(t) PWM ();
    Grupo Monitoreo: Iniciar(), Parar(), Limpiar(), Guardar(), timer();
    Grupo Gcode: Cargar(), Enviar();
    Grupo Step: timer(), reset();
  main
    run InterfazMain3DPrinter();

```

Algoritmo 3.9: Clase InterfazMain3DPrinter. Fuente: Elaboración propia.

- Número de Pulsos XYZE: rango 0 – 65535[dec]
- Velocidad mPaP, periodo de cada pulso enviado:
 - Periodo de Pulsos XY: rango 0 – 65535[ms]
 - Periodo de Pulsos ZE: rango 0 – 255[ms]

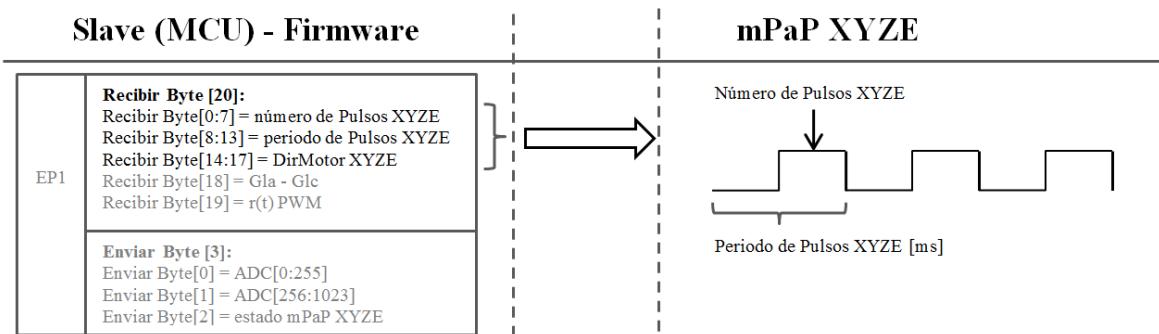


Figura 3.32: Proceso de Envío de datos mPaP al prototipo.

Fuente: Elaboración propia.

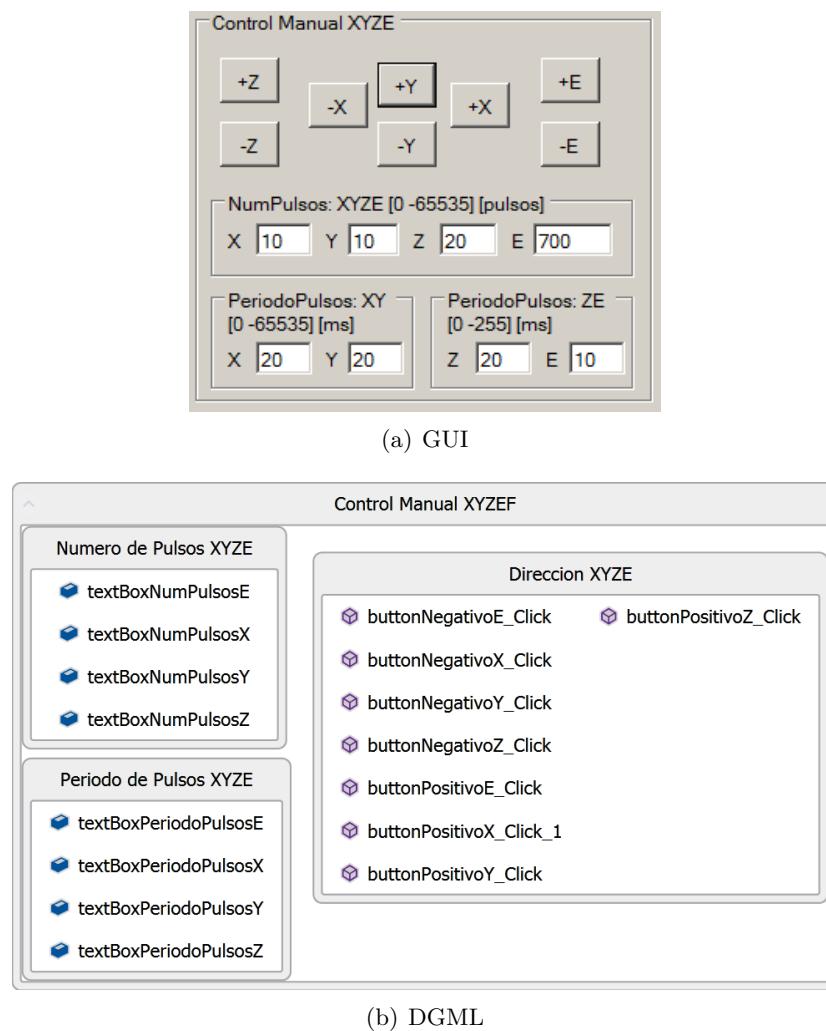


Figura 3.33: GUI-DGML Control Manual XYZ.

Fuente: Elaboración propia.

```

class InterfazMain3DPrinter
    method Control Manual XYZ()
        Enviar_Data(NumPulsos.XYZE) = Enviar_Byte[0:7];
        Enviar_Data(PeriodoPulsos.XYZE) = Enviar_Byte[8:13];
        Enviar_Data(DirPositivo.XYZE) = Enviar_Byte[14:17];
        Enviar_Data(DirNegativo.XYZE) = Enviar_Byte[14:17];
    
```

Algoritmo 3.11: Grupo Método: Control Manual mPaP XYZ. Fuente: Elaboración propia.

3.7.4. Clase Main GUI/Grupo Método: Gcode

Condiciones de implementación Son varios los métodos que se debe definir, además de llamar a las clases PICUSBapi e Impresion3D.

- Método Cargar Documento.gcode
- Método Step: enviar una instrucción del Documento.gcode a la vez.
- Método Timer Step: automatizar el envío de las instrucciones del Documento.gcode.
- Metodo Enviar Gcode: enviar una sola instrucción Gcode.
- Metodo Reset Step: limpiar el *buffer* de instrucciones Gcode.

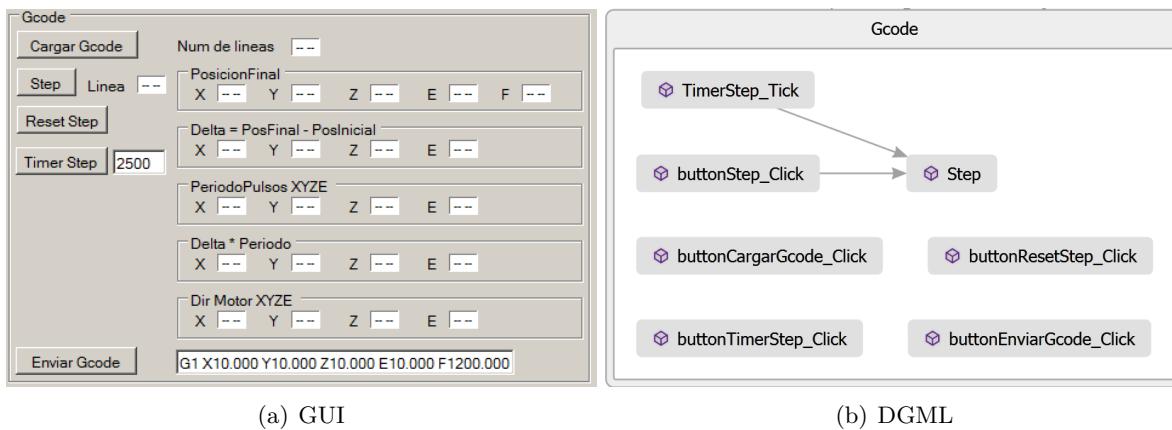


Figura 3.34: GUI-DGML Grupo Método: Gcode.
Fuente: Elaboración propia.

3.7.5. Clase Main GUI/Grupo Método: Monitoreo

Condiciones de implementación Implica el muestreo de datos del sensor del temperatura para ser guardados en un DocumentoMuestreo.txt, que eventualmente sera analizado para diseñar el controlador de temperatura PID correspondiente.

- Métodos Muestreo de datos: Iniciar, Parar y Limpiar datos.

```

class InterfazMain3DPrinter
  method Cargar Gcode()
    if Documento gcode = true then
      guardar instrucciones Gcode = Documento gcode;

  method Step()
    Impresion3D.Iniciar_Impresión(Documento gcode);
    mostrar en GUI el estado de impresion3D;

  method Timer Step()
    iniciar temporizador[ms];

  method Enviar Gcode()
    Enviar_Data(instrucción específica Gcode XYZE);

  method Reset Step()
    Enviar_Data( [0:17] = 0 );
  
```

Algoritmo 3.12: Grupo Método Gcode. Fuente: Elaboración propia.

- Guardar en `DocumentoMuestreo.txt` los datos del sensor: Voltaje, Temperatura y Tiempo.
- Timer Muestreo: Recibir[Byte[3]] del ADC.

```

class InterfazMain3DPrinter
  method Iniciar Muestreo()
    TimerMuestreo.Start(dato Muestreo =  $T_m[ms]$ );

  method Para Muestreo()
    TimerMuestreo.Stop();

  method Limpiar Muestreo()
    chart.clear();

  method Guardar Muestreo()
    Dirección(DocumentoMuestreo);
    Llenar datos de Muestreo en DocumentoMuestreo.txt;
    def
      r(t):Voltaje[V];
      y(t):Voltaje[V];
      y(t):Temp[°C];
      Tiempo de muestreo:tm[ms];

  method Timer Muestreo()
    chart.graph = Recibir[Byte[0:1]];
  
```

Algoritmo 3.13: Grupo Método: Monitoreo. Fuente: Elaboración propia.

3.7.6. Clase Main GUI/Método: Enviar Data

Condiciones de implementación Este método envía los 20 vectores a través de la clase `PICUSBapi`.

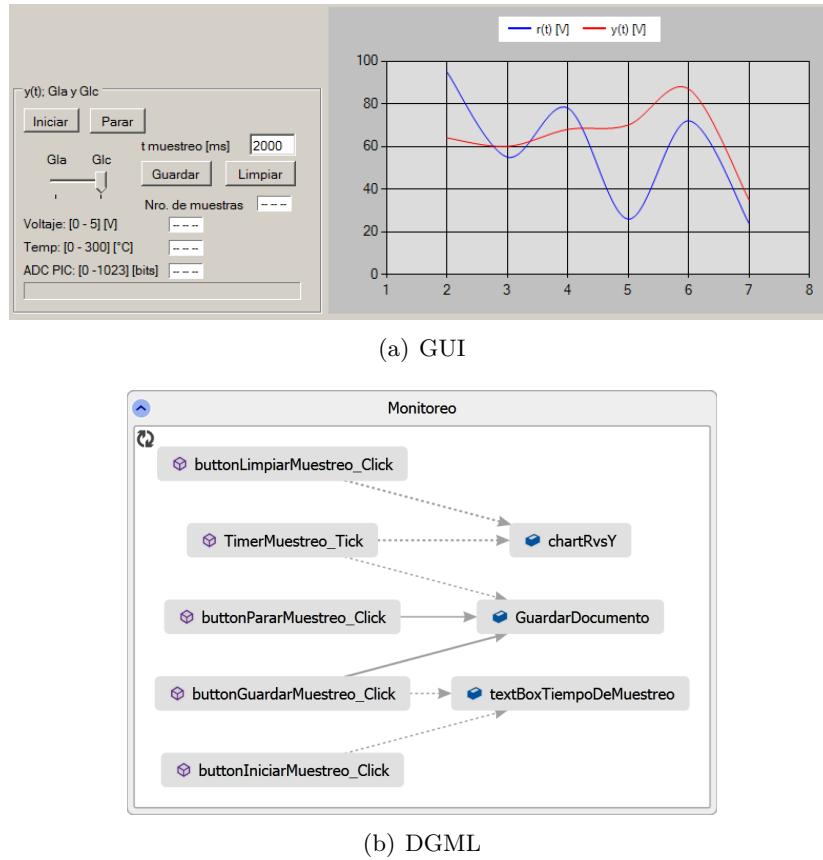


Figura 3.35: GUI-DGML Grupo Método Monitoreo.
Fuente: Elaboración propia.

- Definir los vectores.
- Llamar al método `Enviar_Data` de la clase `PICUSBapi`.

```
class InterfazMain3DPrinter
  method Enviar_Data()
    Envío_Byte[0:20];
    PICUSBapi.EnviarDatos(Envío_Byte[0:19]);
```

Algoritmo 3.14: Método: Enviar Data. Fuente: Elaboración propia.

3.7.7. Clase Main GUI/Método: Referencia r(t)

Condiciones de implementación Para enviar y mostrar los datos de voltaje y temperatura:

- Escalar dato de Barra Horizontal PWM: rango $0 - 255; 2^N - 1; N = 8$

$$\text{Resolución} = \frac{\text{Voltaje}}{2^N - 1} = \frac{5}{255} \left[\frac{V}{\text{dec}} \right] = 0,01961 \left[\frac{V}{\text{dec}} \right]$$

$$- \text{Resolución} = \frac{\text{Temperatura}}{2^N - 1} = \frac{300}{255} \left[\frac{^{\circ}\text{C}}{\text{dec}} \right] = 1,17647 \left[\frac{^{\circ}\text{C}}{\text{dec}} \right]$$

- Mostrar en GUI valor de:
 - Voltaje: rango 0 – 5[V].
 - Temperatura: rango 0 – 300[°C]
- Enviar dato PWM por Byte[19]



Figura 3.36: GUI Referencia r(t) PWM.

Fuente: Elaboración propia.

Según la fig. 3.36 y las condiciones de implementación el algoritmo 3.15 refleja el resultado.

```
class InterfazMain3DPrinter
  method Referencia r(t) PWM()
    Barra_Horizontal_PWM.valor[dec] * 0,01961  $\left[ \frac{V}{\text{dec}} \right]$  = mostrar en GUI Voltaje[V];
    Barra_Horizontal_PWM.valor[dec] * 1,17647  $\left[ \frac{^{\circ}\text{C}}{\text{dec}} \right]$  = mostrar en GUI Temp[°C];
    Barra_Horizontal_PWM.valor[dec] = Enviar_Byte[19] [dec];
```

Algoritmo 3.15: Método Referencia r(t) PWM. Fuente: Elaboración propia.

3.7.8. Clase Main GUI/Método: Selección Gla - Glc

Condiciones de Implementación El método es simple, de acuerdo a la selección Gla-Glc del GUI, enviar dato a través del método Enviar_Data.



Figura 3.37: GUI Selección Gla - Glc.

Fuente: Elaboración propia.

```
class InterfazMain3DPrinter
  method Seleccion Gla - Glc()
    Enviar_Data(Enviar_Byte[18] = Gla - Glc)
```

Algoritmo 3.16: Método: Selección Gla - Glc. Fuente: Elaboración propia.

3.7.9. Clase: Comunicación PICUSBapi

Condiciones de implementación La clase PICUSBapi contiene los métodos y atributos en Envío/Recepción de paquetes USB para Host - Slave MCU, fig. 3.23.

- Atributos:
 - VID/PID = 04D8/0011.
 - EndPoint in
 - EndPoint out
- Métodos:
 - Enviar/Recibir datos.
 - Habilitar/Abrir *pipes* USB.

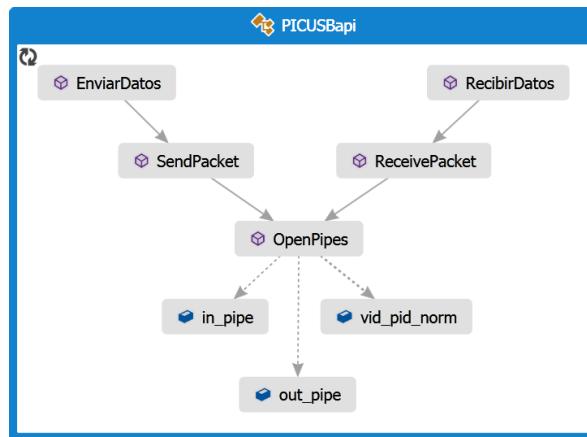


Figura 3.38: DGML Clase PICUSBapi.

Fuente: Elaboración propia.

3.7.10. Clase: GcodeDocument

Condiciones de Implementación Esta clase es necesaria para leer y guardar en un *buffer* todas líneas de instrucción Gcode del Documento.gcode cargado del Grupo Método Gcode.

```

class PICUSBapi
  attributes
    VID/PID = 04D8/0011;
    EndPoint in = 1;
    EndPoint out = 1;
  method Importar abtributos de mpusbapi.dll
    DLLImport(ReceivePacket);
    DLLImport(SendPacket);
  method EnviarDatos()
    SendPacket(Vector[0]) = Byte 0;
    SendPacket(Vector[1]) = Byte 1;
    SendPacket(Vector[..]) = Byte ..;
    SendPacket(Vector[18]) = Byte 18;
    SendPacket(Vector[19]) = Byte 19;
  method RecibirDatos()
    ReceivePacket(Vector[0] = Byte 0);
    ReceivePacket(Vector[1] = Byte 1);
    ReceivePacket(Vector[2] = Byte 2);
  
```

Algoritmo 3.17: Clase PICUSBapi. Fuente: Elaboración propia.

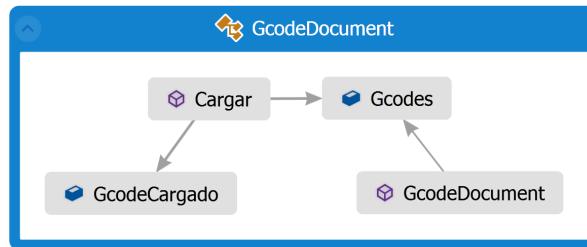


Figura 3.39: DGML GcodeDocument.

Fuente: Elaboración propia.

```

class GcodeDocument
  attributes
    string[ ] Gcodes;
  method Cargar
    if Gcode Cargado = true then
      Gcodes = ReadAllLines(documento)
  
```

Algoritmo 3.18: Clase: GcodeDocument. Fuente: Elaboración propia.

3.7.11. Clase: Vector5D

Condiciones de Implementación Define el manejo de las instrucciones Gcode.

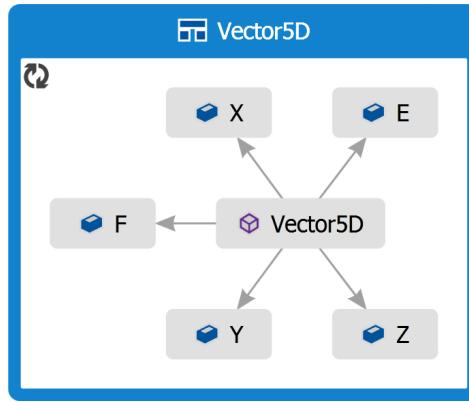


Figura 3.40: DGML Vector5D.

Fuente: Elaboración propia.

```

class Vector5D
    attributes
        decimal X, Y, Z, E, F;
    method operador suma
        Vector5D_1 + Vector5D_2;
    method operador resta
        Vector5D_1 - Vector5D_2;
    method operador absoluto
        abs(Vector5D_1);
        abs(Vector5D_2);
    end
end
  
```

Algoritmo 3.19: Clase: Vector5D. Fuente: Elaboración propia.

3.7.12. Clase: Impresion3D

Condiciones de Implementación Es la clase **descriptor Gcode** que calcula las distancias, velocidades y sentido de giro que deben realizar los mPaP de acuerdo a las instrucciones Gcode previamente guardados y enumerados, para que finalmente imprima una pieza 3D.

- Procesar la instrucción Gcode para obtener los valores numéricos de XYZEF.
- Escalar la relación de movimiento por cada eje según los datos obtenidos empíricamente:
 - Eje X: 1[mm] = 3,03[pulsos]
 - Eje Y: 1[mm] = 10[pulsos]
 - Eje Z: 1[mm] = 50[pulsos]
 - Extrusor E: 1[mm] = 70[pulsos]
 - Velocidad F: $10 \left[\frac{mm}{s} \right] \approx 20[pulsos]$

- Decidir dirección mPaP XYZ.

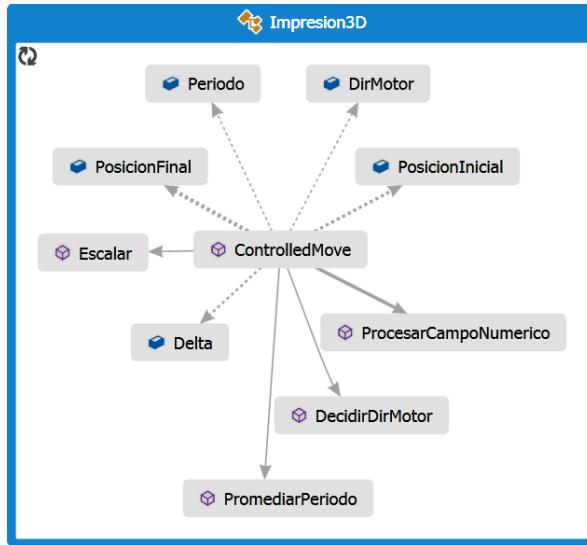


Figura 3.41: DGML Impresion3D.

Fuente: Elaboración propia.

```

class Impresion3D
    attributes
        Vector5D Posición Inicial;
        Vector5D Posición Final;
        Vector5D (Posición Inicial - Posición Final) = Número de Pulso;
        Vector5D Periodo;
        Vector5D DirMotor;

    method Procesar Campo Numerico()
        leer una instrucción a la vez del Documento.gcode;

    method Escalar()
        Posicion X = Número de Pulso X * 3,03;
        Posicion Y = Número de Pulso Y * 10;
        Posicion Z = Número de Pulso Z * 50;
        Posicion E = Número de Pulso E * 70;

    method DirMotor()
        if Posicion Final > Posicion Inicial then
            mover derecha;
        if Posicion Final < Posicion Inicial then
            mover izquierda;

```

Algoritmo 3.20: Clase Impresion3D. Fuente: Elaboración propia.

3.8. Herramientas externas aplicadas

En la etapa final del proyecto converge el *Firmware-Software* y Herramientas externas.

3.8.1. Sketchup CAD →.stl

Aplicando Sketchup y un plugin STL Export & Import.

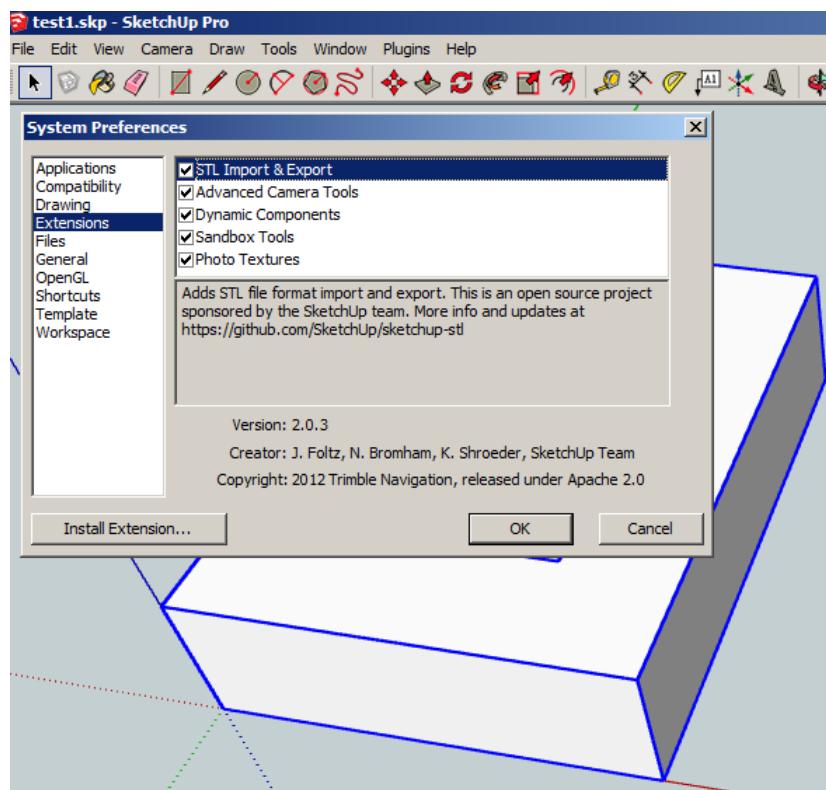


Figura 3.42: Interfaz Sketchup y plugin .stl.
Fuente: Elaboración propia.

3.8.2. CAM/Slic3r(.stl) →Gcode

Slic3r funciona bastante bien en la generación de Gcode

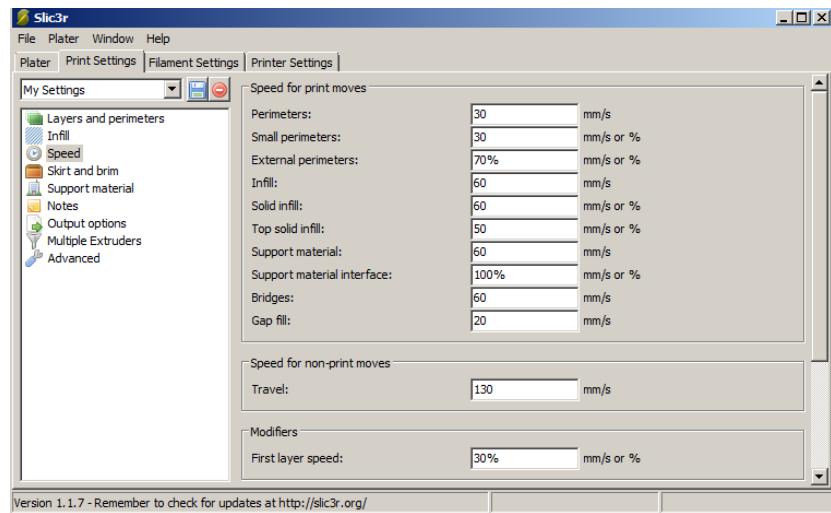


Figura 3.43: Interfaz Slic3r.
Fuente: Elaboración propia.

3.8.3. Viewer Repetier-Host

Por ultimo se aplica Repetier-Host como *viewer* general.

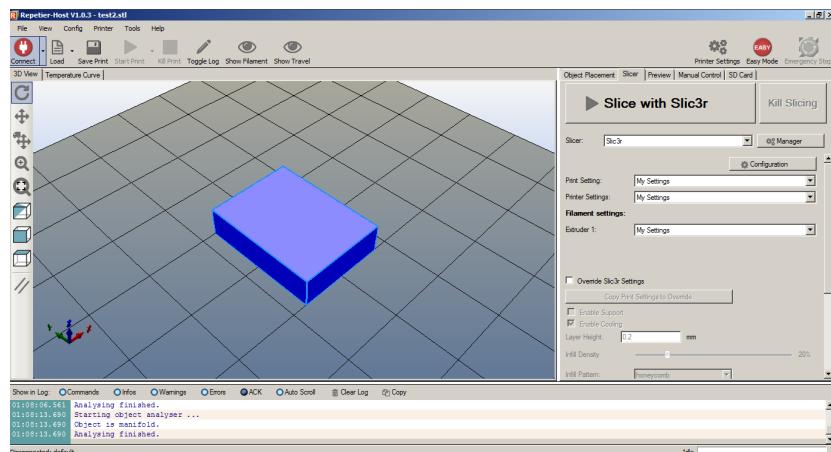


Figura 3.44: Interfaz Repetier-Host.
Fuente: Elaboración propia.

4. Resultados

Implementando el siguiente modelo CAD y exportando el formato .stl

- Interior: 7[mm]x12[mm]
- Exterior: 15[mm]x20[mm]
- Altura: 5[mm]

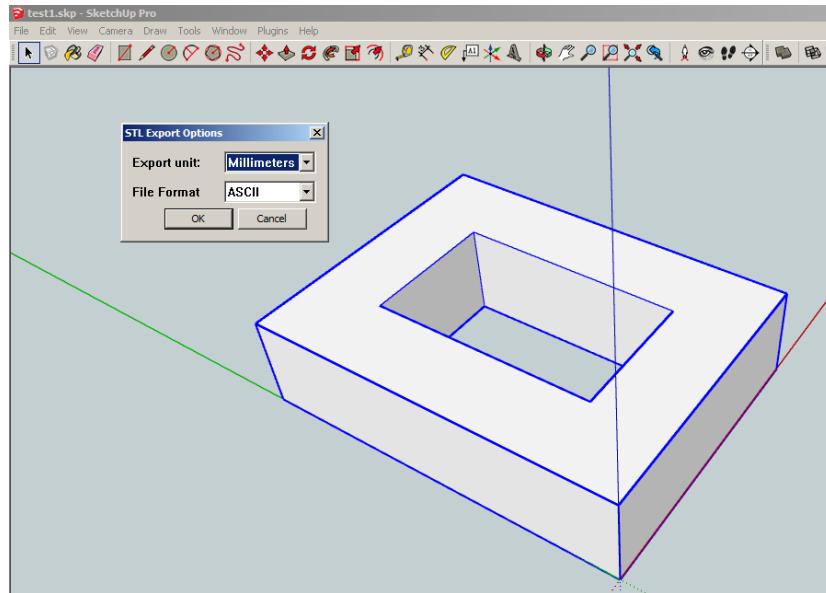


Figura 4.1: Modelo CAD de la pieza.

Fuente: Elaboración propia.

Obteniendo el **Documento.gcode** con **Slic3r** y visualizando con **Repetier-Host**.

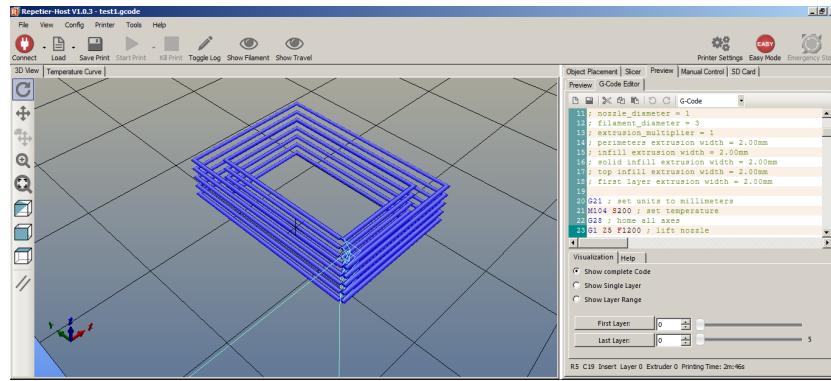


Figura 4.2: Generación y visualización Gcode.

Fuente: Elaboración propia.

Importando el Documento .gcode con el botón Cargar Gcode e imprimiendo la pieza con el botón Step, usando el GUI del Proyecto.

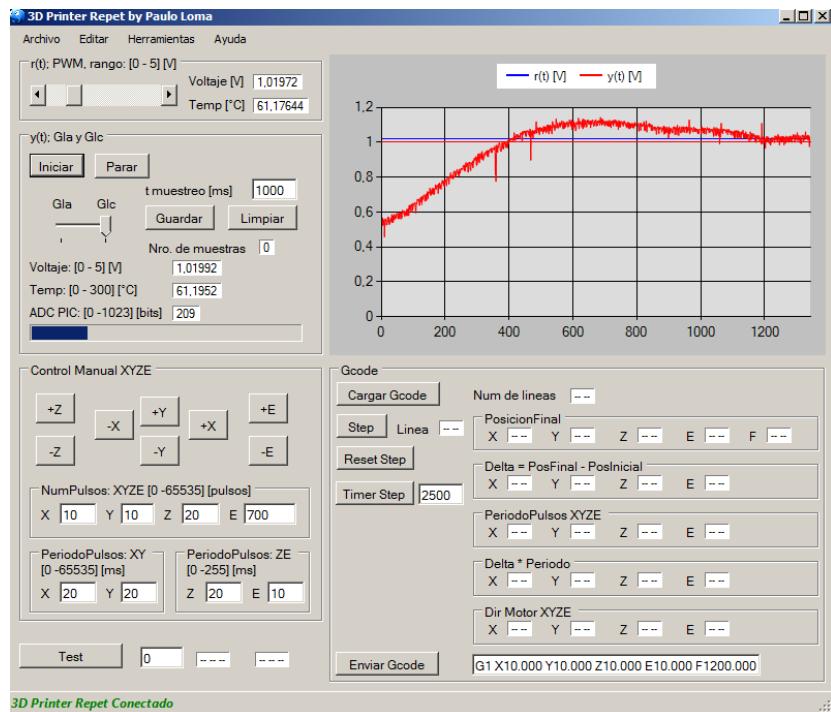


Figura 4.3: Operación general a través del GUI.

Fuente: Elaboración propia.

La pieza final resultante:

- Interior: 7[mm]x13[mm]
- Exterior: 116[mm]x21[mm]
- Altura: 5[mm]

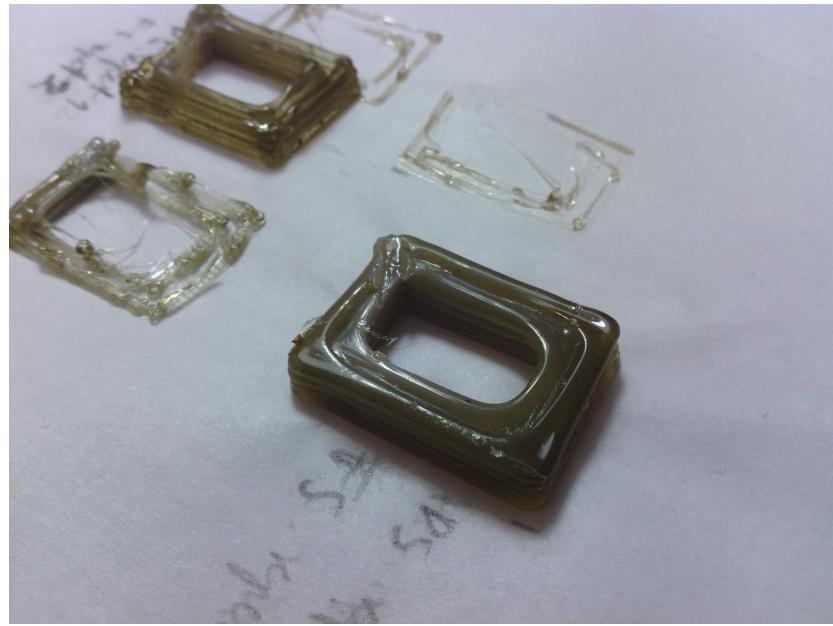


Figura 4.4: Pieza final impresa.

Fuente: Elaboración propia.

5. Conclusiones y recomendaciones

De acuerdo a los objetivos propuestos se concluye su cumplimiento según la siguiente descripción:

- La subsección 3.3.1 detalla la aplicación de material de impresión plástico PET de botellas desechables, alimentado al Licuefactor manualmente.
- El extrusor electromecánico fue construido con e-waste y componentes adquiridos en tiendas locales, sección 3.3.
- El sistema de transmisión de energía mecánica por cada eje de movimiento también esta compuesto por e-waste.
- El procedimiento de control PID se aplicó a través de las herramientas Ident y Sisotool de Matlab, la adquisición de datos de la planta G_p fue por medio del propio GUI del proyecto, y la implementación a través del MCU PIC18F4550, sección 3.6.8.
- El diseño del sistema electrónico y su implementación satisface todos los requerimientos de funcionamiento: control, comunicación y operación del prototipo, fig. 3.20.
- El desarrollo de Firmware sincroniza los mPaP para el correcto posicionamiento 3D, envía/recibe datos del sensor de temperatura por medio del protocolo USB para el correcto control PID, sección 3.6.
- El desarrollo de Software aplicando P00, interactua completamente con el Firmware para la ejecución de las instrucciones Gcode en el proceso de impresión 3D, sección 3.7.

Con respecto a las recomendaciones, se establece lo siguiente:

- La baja resolución de las piezas impresas se debe a la limitación del Medio Paso en los mPaP, fig. 2.13. Por tanto se recomienda el cambio de diseño electrónico que implemente Micro Paso, lo que resultará en una Impresión 3D de calidad media/alta.
- La alimentación del material de impresión es manual y la cantidad de plástico procesado depende del volumen del Licuefactor, se recomienda un diseño nuevo de extrusor que contemple la alimentación continua; fig. 2.2(c), siempre y cuando se logre obtener rollos de plástico de botellas PET.
- La estructura mecánica completa debe ser diseñada con un estilo minimalista para que ocupe la menor cantidad de espacio.

Bibliografía

- [1] Lauren Setar y Matthew MacFarland. *Top 10 Fastest-Growing Industries*. IBIS World, abr. de 2012. URL: <http://www.ibisworld.com/Common/MediaCenter/Fastest%20Growing%20Industries.pdf> (vid. pág. 1).
- [2] James Manyika y col. *Disruptive technologies: Advances that will transform life, business and the global economy*. Inf. téc. McKinsey Global Institute, 2013. URL: http://www.mckinsey.com/~/media/McKinsey/dotcom/Insights%20and%20pubs/MGI/Research/Technology%20and%20Innovation/Disruptive%20technologies/MGI_Disruptive_technologies_Full_report_May2013.ashx (vid. pág. 1).
- [3] Adrian Bowyer. *The RepRap Project*. URL: <http://reprap.org/> (vid. págs. 1, 2).
- [4] Bre Pettis, Adam Mayer y Zach Smith. *MakerBot Industries*. URL: <http://www.makerbot.com/> (vid. pág. 1).
- [5] Formlabs. *Formlabs*. URL: <http://www.formlabs.com> (vid. pág. 1).
- [6] David Bourell, Ming Leu y David Rosen. “Roadmap for Additive Manufacturing Identifying the Future of Freeform Processing”. En: 2009. URL: <http://wohlersassociates.com/roadmap2009.pdf> (vid. pág. 1).
- [7] Ian Gibson, David Rosen y Brent Stucker. *Additive Manufacturing Technologies*. Springer US, 2010. DOI: [10.1007/978-1-4419-1120-9](https://doi.org/10.1007/978-1-4419-1120-9). URL: <http://link.springer.com/book/10.1007/978-1-4419-1120-9> (vid. págs. 1, 6).
- [8] Enrique Canessa, Carlo Fonda y Marco Zennaro. “Perpetual Plastic Project”. En: *Low-cost 3D Printing for Science, Education and Sustainable Development* (mayo de 2013), págs. 191-197. URL: http://sdu.ictp.it/3d/book/Low-cost_3D_printing_screen.pdf (vid. pág. 2).
- [9] Adrian Bowyer. *Granule Extruder*. URL: <http://reprap.org/wiki/GranuleExtruder> (vid. pág. 2).
- [10] Dominik Proschek. “Extruder to Recycle Plastic Milk Bottles”. Tesis de lic. University of ULSTER. URL: http://reprap.org/mediawiki/images/9/90/Master_Thesis_Extruder_Unit.pdf (vid. pág. 2).
- [11] G.B. Braanker y col. *Developing a plastics recycling add-on for the RepRap 3D-printer*. Delft University of Techonology, 2010. 46 págs. URL: <http://reprapdelft.files.wordpress.com/2010/04/reprap-granule-extruder-tudelft1.pdf> (vid. pág. 2).
- [12] Jo Geraedts y col. “Three view on Additive Manufacturing: Business, Research and Education”. En: *TMC2012*. Ed. por I. Horváth y col. Organizing Committee of TMCE 2012. Mayo de 2012. URL: http://www.researchgate.net/publication/235725722_Three_VIEWS_on_ADDITIVE_Manufacturing_Business_Research_and_Education (vid. pág. 2).
- [13] Sawers. *Sawers 3D*. URL: www.sawers.com (vid. pág. 2).

- [14] Jan Eite Bullema. *3d printing to realize innovative electronic products*. Presentation. 2013. URL: http://industrielelektronica.fhi.nl/images/stories/devclub/3d_printing_to_realize_innovative_electronic_products.pdf (vid. pág. 6).
- [15] Pierre Lafleur y Bruno Vergnes. *Polymer Extrusion*. 1th Edition. Wiley-ISTE, 9 de mayo de 2014. URL: <http://www.wiley.com/WileyCDA/WileyTitle/productCd-1848216505.html> (vid. pág. 6).
- [16] Chris Rauwendaal y col. *Polymer Extrusion*. 5th Edition. Carl Hanser Verlag, 2014. URL: <http://www.hanser-elibrary.com/isbn/9781569905166> (vid. pág. 6).
- [17] Harold Giles, John Wagner y Eldridge Mount. *Extrusion: the definitive processing guide and handbook*. English. 2nd. William Andrew, sep. de 2014. 640 págs. ISBN: 9781437734829. URL: http://store.elsevier.com/product.jsp?isbn=9781437734812&_requestid=596110 (vid. págs. 6, 8).
- [18] Keith Luker. "Hot-melt Extrusion: Pharmaceutical Applications". En: ed. por Dennis Douroumis. 1th Edition. A John Wiley, abr. de 2012. Cap. 1. URL: <http://www.wiley.com/WileyCDA/WileyTitle/productCd-1118307879.html> (vid. pág. 7).
- [19] Joshua Pearce. *Waste plastic extruder: literature review*. 2014. URL: http://www.appropedia.org/Waste_Plastic_Extruder:_Literature_Review#Characterization_and_recovery_of_polymer_from_mobile_phone_scrap (vid. pág. 7).
- [20] J. M. Selig. *Introductory Robotics*. Prentice Hall, 1992 (vid. pág. 8).
- [21] Sherry Huss. *Ultimate Guide to 3D Printing*. Inf. téc. 2013 (vid. pág. 9).
- [22] Microchip Technology. *PIC18F2455/2550/4455/4550 Data Sheet : 28/40/44-Pin, High-Performance, Enhanced Flash, USB Microcontrollers with nanoWatt Technology*. English. Ed. por Microchip Technology Inc. Oct. de 2006. 428 págs. URL: <http://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf> (vid. págs. 10, 12, 30, 45).
- [23] USB.org. *USB developers*. URL: <http://www.usb.org> (vid. pág. 11).
- [24] Eduardo Carletti. *Motores paso a paso Características básicas*. 2014. URL: http://robots-argentina.com.ar/MotorPP_basico.htm (vid. págs. 13, 14).
- [25] ST SGS-Thomson Microelectronics. *The L297 Stepper Motor Controller*. 1995. URL: <http://www.alldatasheet.com/datasheet-pdf/pdf/22436/STMICROELECTRONICS/L297.html> (vid. págs. 15, 30).
- [26] Miguel A. Pérez García y col. *Instrumentación Electrónica*. S.A. Ediciones Paraninfo, 2003. 862 págs. ISBN: 9788497321662. URL: <http://m.casadellibro.com/libro-instrumentacion-electronica-incluye-cd/9788497321662/936109> (vid. pág. 16).
- [27] Martin P. Bates. *Programming 8-bit PIC Microcontrollers in C with Interactive Hardware Simulation*. English. 1th. Newnes, 22 de ago. de 2008. 278 págs. ISBN: 9780080560144. URL: http://books.google.com.bo/books?id=lI087dnh07YC&hl=es&source=gbs_navlinks_s (vid. pág. 16).
- [28] Analog Devices. *Low Cost Low Power Instrumentation Amplifier AD620*. 2011 (vid. pág. 34).
- [29] Alex B. *OSA-RTOS*. 2009. URL: <http://www.pic24.ru/doku.php/en/osa/ref/intro> (vid. pág. 43).

Índice de Palabras

- algoritmo, 4, 17, 21, 22, 36, 41–45, 47, 50, 54, 61
Arduino, 2
Atmel, 2
ATX-PC, 36
bipolar, 12, 26, 30
Bulk, 10, 11
clase, 11, 53, 54, 59, 62
control, 3, 11, 13, 17, 21, 22, 29–31, 40, 42, 47, 71
descriptor, 12, 40, 41, 64
Efecto Joule, 3, 4, 7, 9, 21, 25, 31
electrónica, 3, 14, 36
electromecánico, 2, 3, 5, 6, 71
extrusor, 1, 2, 4, 5, 7–9, 19, 22–24, 28, 40, 71
Firmware, 4, 17, 21, 22, 28, 36, 38, 40, 42, 66, 71
Hardware, 2, 4
Ident, 48, 71
impresión, 2–7, 9, 18, 21, 53, 71
Impresora 3D, 1–3, 5, 8, 19, 55, 56
kernel, 17, 43
L297, 13, 30, 31
L298, 13
licuefacción, 5–7, 14, 25
Licuefactor, 7, 25, 31, 47, 48, 71
Microchip, 10
microcontrolador, 2, 3
OSA-RTOS, 21, 42, 43
PIC18F4550, 21, 28, 31, 35, 37, 42, 71
potencia, 14, 31
prototipo, 3, 4, 9, 18, 21, 22, 24, 28, 71
Repetier-Host, 67, 68
RepRap, 1, 2
sensor, 3, 5, 9, 15, 16, 21, 29, 35, 48, 58, 59, 71
Sisotool, 49, 71
Slicer, 18–21, 41
Software, 2–5, 18, 20, 21, 36, 48, 52, 66, 71
Termocupla, 9, 15, 21, 29, 35, 48
toolpath, 41
unipolar, 12, 13, 26, 30, 31

Acrónimos

.stl formato de archivo Estereolitográfico - *STereoLithography file format.* [18](#), [68](#)

ABS Acrilonitrilo Butadieno Estireno - *Acrylonitrile Butadiene Styrene.* [2](#), [3](#), [5](#)

ADC Conversor Analógico Digital - *Analogic Digital Converter.* [10](#), [17](#), [21](#), [29](#), [37](#), [38](#), [43](#), [45](#), [59](#)

AM Manufactura por Adición - *Additive Manufacturing.* [1](#)

CAD Diseño Asistido por Computador - *Computer Aided Design.* [1](#), [2](#), [4](#), [18](#), [20](#), [21](#), [23](#), [36](#), [68](#)

CAE Ingeniería Asistida por Computador - *Computer Aided Engineering.* [2](#)

CAM Manufactura Asistida por Computador - *Computer Aided Manufacturing.* [2](#)

CC Clase Personalizada - *Custom Class.* [10](#), [11](#)

CCP Capturador/Comparador/PWM - *Capture/Compare/PWM.* [10](#), [45](#)

CDC Clase de Dispositivo de Comunicación COMM - *Communication Device Class.* [10](#), [11](#)

CNC Control Numérico Computarizado - *Computer Numerical Control.* [1](#), [4](#), [8](#)

DGML Mapa de Código de Visual Studio. [53](#)

FDM Modelado por Deposición Fundida - *Fused Deposition Modeling.* [1–3](#), [5](#), [8](#), [15](#)

GPL Licencia Pública General - *General Public License.* [1](#)

GUI Interfaz Gráfica de Usuario - *Graphic User Interface.* [22](#), [38](#), [45](#), [48](#), [53](#), [61](#), [69](#), [71](#)

HID Dispositivo de Interfaz Humana - *Human Interface Device.* [10](#), [11](#)

I/O Entrada/Salida - *Input/Output.* [12](#), [17](#)

IC Circuito Integrado - *Integrated Circuit.* [13](#), [30](#), [31](#)

ISR Rutina de Servicio de Interrupción - *Interrupt Service Routine.* [17](#)

MCU Unidad de Micro Controlador - *Micro Controller Unit.* [5](#), [9](#), [10](#), [12](#), [28](#), [31](#), [40](#), [42](#), [45](#), [50](#), [71](#)

mPaP motores paso a paso. [2](#), [4](#), [8](#), [9](#), [12](#), [13](#), [21](#), [22](#), [26](#), [29–31](#), [40](#), [47](#), [52](#), [55](#), [56](#), [64](#), [65](#), [71](#)

MSD Dispositivo de Almacenamiento Masivo - *Mass Storage Device.* [10](#), [11](#)

NTC Coeficiente de Temperatura Negativo - *Negative Temperature Coefficient.* [15](#)

OS Sistema Operativo - *Operating System.* [16](#), [17](#)

PCB Placa de Circuito Impreso - *Printed Circuit Board.* [21](#), [36](#)

PET Tereftalato de Polietileno - *Polyethylene Terephthalate.* [2–4](#), [25](#), [71](#)

PID Proporcional Integral Derivativo. [3](#), [4](#), [7](#), [21](#), [22](#), [29](#), [47](#), [50](#), [58](#), [71](#)

PLA Poliácido Láctico - *Polylactic Acid.* [2](#), [3](#), [5](#)

POO Programación Orientada a Objetos. [4](#), [11](#), [52](#), [71](#)

PWM Modulación por Ancho de Pulso - *Pulse Width Modulation.* [3](#), [29](#), [31](#), [38](#), [45](#), [52](#), [60](#), [61](#)

RTD Detector de Temperatura Resistiva - *Resistance Temperature Detector.* [15](#)

RTOS Sistema Operativo de Tiempo Real - *Real Time Operating System.* [5](#), [16](#), [17](#), [42](#), [43](#), [47](#)

SLA Estereolitográfica - *Stereolithography.* [1](#)

SLS Sinterización Selectiva por Láser - *Selective Laser Sintering.* [1](#)

UAGRM Universidad Autónoma Gabriel René Moreno. [2](#)

USB Bus Serial Universal - *Universal Bus Serial.* [3–5](#), [9–12](#), [21](#), [22](#), [29](#), [30](#), [37–39](#), [42](#), [45](#), [46](#), [53–55](#), [62](#), [71](#)

VID/PID ID Vendedor/ID Producto - *Vendor ID/Product ID.* [53](#), [62](#)

Glosario

e-waste deshecho electrónico - *electronic waste.* [3](#), [4](#), [21](#), [22](#), [30](#), [36](#), [71](#)

Gcode Lenguaje de programación usado en CNC. [1](#), [8](#), [18–22](#), [40](#), [41](#), [53](#), [54](#), [58](#), [62–64](#), [71](#)

Matlab Entorno de desarrollo matemático integrado. [4](#), [40](#), [47–50](#), [71](#)

Proteus Entorno de desarrollo integrado de diseño electrónico. [21](#)

Visual Studio Entorno de desarrollo integrado para aplicaciones de Software. [21](#), [52](#), [53](#)

. Anexo A: Código fuente de *Firmware*

A continuación se muestra el código fuente del documento principal `main.c`

```
1 //--- config: PIC -----
2 #include <18F4550.h>
3 #device adc=10
4 #fuses HSPLL,MCLR,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN
5 #use delay(crystal=20MHz, clock=48MHz)
6
7 //cabecera necesaria para trabajar con el Bootloader "ex_usb_bootloader.c"
8 #include "usb_bootloader.h"
9
10 //--- config: USB bulkmode -----
11 #define USB_HID_DEVICE FALSE
12 #define USB_EP1_TX_ENABLE USB_ENABLE_BULK //turn on EP1(EndPoint1)
13 //for IN bulk/interrupt transfers
14 #define USB_EP1_RX_ENABLE USB_ENABLE_BULK //turn on EP1(EndPoint1)
15 //for OUT bulk/interrupt transfers
16 #define USB_EP1_TX_SIZE 3 //size to allocate for the tx endpoint 1 buffer
17 #define USB_EP1_RX_SIZE 20 //size to allocate for the rx endpoint 1 buffer
18
19 #include <pic18_usb.h> //Microchip PIC18Fxx5x Hardware layer
20 //for CCS's PIC USB driver
21 #include "usb_descriptor.h" //Configuracion del USB y los descriptores
22 //para este dispositivo
23 #include <usb.c> //handles usb setup tokens and get descriptor reports
24
25 #include <osa.h> //OSA RTOS
26
27 //--- config: Directivas I/O -----
28 #use standard_io(A)
29 #use standard_io(B)
30 #use standard_io(C)
31 #use standard_io(D)
32
33 //--- def: variables Tarea USB -----
34 unsigned int8 RecibirByte[20]; //vector [0;19]=20 vectores x 1Byte
35 unsigned int8 EnviarByte[3]; //vector [0;2]=3 vectores x 1Byte
36
37 //--- def: variables Tareas mPaP -----
38 unsigned int16 numPulsosX,numPulsosY,numPulsosZ,numPulsosE;
39 unsigned int16 periodoPulsosX,periodoPulsosY;
40 unsigned int8 periodoPulsosZ,periodoPulsosE;
41 unsigned int8 dirMotorX,dirMotorY,dirMotorZ,dirMotorE;
```

```
42 #define dirX    PIN_D0
43 #define dirY    PIN_D1
44 #define dirZ    PIN_D2
45 #define dirE    PIN_D3
46 #define clockX   PIN_D4
47 #define clockY   PIN_D5
48 #define clockZ   PIN_D6
49 #define clockE   PIN_D7
50
51 //--- def: variables para funciones; ADC PIC, Gla, GlcPID -----
52 unsigned int8 GlaoGlc;
53 unsigned int16 valorR,valorY,valorYusb,control;
54 float Kp,Kd,Ki,Kpz,Kiz,Kdz,T;      //Constantes PID
55 float r,y,u,e,e1,p,i,ii,d,d1;      //Variables PID
56 unsigned int16 max,min;            //Variables anti-windup
57
58 //--- def: Funciones Genericas -----
59 void Inicio (void);
60 void ADC(void);
61
62 //--- def: Tareas OSA -----
63 void USB(void);
64 void Gla(void);
65 void Glc_PIDdiscreto(void);
66 void MotorX(void);
67 void MotorY(void);
68 void MotorZ(void);
69 void MotorE(void);
70
71 //*****
72 //**** main
73 //*****
74 void main(void){
75
76     Inicio();
77
78     OS_Init();           // Init OS
79     OS_Task_Define(USB); // Define tasks.
80     OS_Task_Define(Gla);
81     OS_Task_Define(Glc_PIDdiscreto);
82     OS_Task_Define(MotorX);
83     OS_Task_Define(MotorY);
84     OS_Task_Define(MotorZ);
85     OS_Task_Define(MotorE);
86
87     // Create tasks. // if 0 = no priorities
88     OS_Task_Create(0, USB);
89     OS_Task_Create(0, Gla);
90     OS_Task_Create(0, Glc_PIDdiscreto);
91     OS_Task_Create(0, MotorX);
92     OS_Task_Create(0, MotorY);
93     OS_Task_Create(0, MotorZ);
94     OS_Task_Create(0, MotorE);
95
96     // Create tasks, Task priority. Allowed values from 0(highest) to 7(lowest)
97     /*OS_Task_Create(0, USB);
98     OS_Task_Create(7, Gla);
99     OS_Task_Create(1, Glc_PIDdiscreto);
100    OS_Task_Create(2, MotorX);
```

```

101 OS_Task_Create(2, MotorY);
102 OS_Task_Create(2, MotorZ);
103 OS_Task_Create(2, MotorE);
104
105 OS_Bsem_Set(BS_GLAGLC_FREE);*/
106
107 OS_EI();           // Enable interrupts
108 OS_Run();          // Running scheduler
109 }
110
111 //***** Funciones genericas ****
112 void Inicio(void){
113     //--- ini: mPaP -----
114     numPulsosX=numPulsosY=numPulsosZ=numPulsosE=0;
115     periodoPulsosX=periodoPulsosY=periodoPulsosZ=periodoPulsosE=0;
116     //--- ini: PID -----
117     min=0.0; max=1023 ; //valor Anti-windup
118     i1=0; e1=0; d1=0;
119     Kd=8; Kp=8; Ki=0.02915;
120     T=5;           //Tiempo de muestreo tr/6 < T < tr/20
121     Kpz=Kp; Kiz=Ki*T/2; Kdz=Kd/T;
122
123     //--- ini: CCP -----
124     //--- Pre=16 PR2=249 Pos=1, PWMF=3kHz ,PWMT=300us con Fosc(clock)=48MHz
125     setup_timer_2(T2_DIV_BY_16,249,1);
126     setup_ccp1(ccp_pwm);           //Configurar modulo CCP1 en modo PWM
127     set_pwm1_duty(0);
128
129     //--- ini: ADC -----
130     setup_adc_ports(AN0|VSS_VDD );
131     setup_adc(ADC_CLOCK_INTERNAL);
132     //setup_adc(ADC_CLOCK_DIV_8); //respetar el Tad>1.6us
133     //Tad=8/Fosc=8/20Mhz=400ns
134     set_adc_channel(0);          //Seleccionar Canal(0)=AN0=A0 para ADC
135
136     //--- ini: TIMERO for OS_Timer() -----
137     setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1); //config Timer0 , Pre=1=RTCC_DIV_1
138     //set_timer0(0xF63B);    //carga del Timer0, clock=20MHz, Fout=1kHz=0xF63B
139     set_timer0(0xE88F);        //carga del Timer0, clock=48MHz, Fout=1kHz=0xE88F
140
141     //--- ini: Interrupts -----
142     //enable_interrupts(GLOBAL);
143     enable_interrupts(INT_TIMER0);           //habilita interrupcion Timer0
144     //enable_interrupts(INT_TIMER2);
145
146     //--- ini: USB -----
147     usb_init();                  //inicializamos el USB
148     usb_task();                 //habilita periferico usb e interrupciones
149     usb_wait_for_enumeration(); //esperamos hasta que el PicUSB
150                           //sea configurado por el host
151     delay_ms(50);
152 }
153
154 void ADC(void){
155     valorYusb=read_adc();
156     //delay_us(1); //Taqc minimo de carga
157     //del capacitor(sample&hold)=8/Fosc=8/20MHz=400ns=0.4us
158
159     //-- El ADC es de 10 bits y puedo enviar solo 8, asi que separo

```

```

160 //-- la variable en 2 bytes "EnviarByte[0] y EnviarByte[1]",
161 //-- luego se arma en C# (ver notas)
162 EnviarByte[0]=valorYusb >> 8; //desplazamiento de 8bits a la derecha
163 EnviarByte[1]=valorYusb & 0xFF; //a & b = AND binario
164 }
165
166 //***** Tareas OSA ****
167 #INT_TIMERO
168 void timer0_isr(void){
169     OS_Timer();
170     set_timer0(0xE88F); //se recarga el Timer0
171 }
172
173 void USB(void){
174     for(;;){
175         if(usb_enumerated()){ //True si el USB ha sido enumerado.
176             if(usb_kbhit(1)){ //((endpoint=1 EP1)= TRUE si el EP1 tiene datos
177                 //en su buffer de recepcion.
178
179                 //--- (endpoint,ptr,max)=Reads up to max bytes from
180                 //--- the specified endpoint buffer and saves it to the pointer ptr
181                 //--- Returns the number of bytes saved to ptr
182                 usb_get_packet(1,RecibirByte,20);
183
184                 //--- revisa en orden logico el contenido de
185                 //--- RecibirByte[0],[1],[2],[3],[4]....
186                 //--- *2, OSA usa el doble de pulsos
187                 numPulsosX = (RecibirByte[0]*256+RecibirByte[1])*2;
188                 numPulsosY = (RecibirByte[2]*256+RecibirByte[3])*2;
189                 numPulsosZ = (RecibirByte[4]*256+RecibirByte[5])*2;
190                 numPulsosE = (RecibirByte[6]*256+RecibirByte[7])*2;
191                 periodoPulsosX = (RecibirByte[8]*256+RecibirByte[9]);
192                 periodoPulsosY = (RecibirByte[10]*256+RecibirByte[11]);
193                 periodoPulsosZ = RecibirByte[12];
194                 periodoPulsosE = RecibirByte[13];
195                 dirMotorX = RecibirByte[14];
196                 dirMotorY = RecibirByte[15];
197                 dirMotorZ = RecibirByte[16];
198                 dirMotorE = RecibirByte[17];
199                 GlaoGlc = RecibirByte[18];
200                 valorR = RecibirByte[19];
201             }
202             //--- reviso en orden logico EnviarByte[0],[1] y envio por usb
203             ADC(); //esta funcion contiene los valores de EnviarByte[0;1]
204             if( (numPulsosX || numPulsosY || numPulsosZ || numPulsosE) != 0 ) {
205                 EnviarByte[2]=1; output_high(PIN_B6);}
206                 else {EnviarByte[2]=0; output_low(PIN_B6);}
207
208             //--- (endpoint,data,len,tgl)=Places the packet of data
209             //--- into the specified endpoint buffer.
210             //--- Returns TRUE if success, FALSE if the buffer
211             //--- is still full with the last packet.
212             usb_put_packet(1,EnviarByte,3,USB_DTS_TOGGLE);
213         }
214         OS_Delay(10);
215         OS_Yield();
216     }
217 }
```

```

218 void Gla(void){
219     for(;;){
220         OS_Wait(GlaoGlc==3);
221         //OS_Bsem_Wait(BS_GLAGLC_FREE);
222         output_toggle(PIN_B7);
223
224         /** 1023/255=4.012, necesita adaptarse al mismo rango
225         /** porque el valorR que llega desde el GUI tiene max=255
226         r=(float)(valorR*4.012);
227         control=(unsigned int16)r;
228         set_pwm1_duty(control);
229
230         OS_Delay(1000);
231         //OS_Bsem_Set(BS_GLAGLC_FREE);
232         OS_Yield();
233     }
234 }
235
236 void Glc_PIDdiscreto(void){
237     for(;;){
238         OS_Wait(GlaoGlc==2); //2
239         //OS_Bsem_Wait(BS_GLAGLC_FREE);
240         output_toggle(PIN_B7);
241
242         valorY=read_adc(ADC_READ_ONLY);
243         /** debido al ADC 10bits, la conversion se realiza con 10bits,
244         /** entonces valorY tiene rango de 0 a 1023
245         y=(float)valorY;
246         /** 1023/255=4.012, necesita adaptarse al mismo rango
247         /** porque el valorR que llega desde el GUI tiene max=255
248         r=(float)(valorR*4.012);
249         -----
250         /** Calculo PID por metodo tustin para termino integral
251         /** y metodo de diferencias hacia atras para termino derivativo
252         /** Sea e(kT)=e; e(kT-T)=e1
253         e=r-y;
254         //Sea p(kT)=p; i(kT)=i; i(kT-T)=i1; d(kT)=d
255         p=Kpz*e;
256         //i=i1+Kiz*e;    //diferencia hacia atras
257         i=i1+Kiz*(e+e1); //tustin
258         d=Kdz*(e-e1);   //diferencia hacia atras
259
260         //Sea u(kT)=u
261         u=p+i+d;
262
263         /** Anti-windup solo al termino integral para evitar que se inflie
264         /** y se haga muy grande si la accion de control se satura, por tanto
265         /** es necesario impedir que cambie i
266         //if((u>max) | (u<min)) {i=i-Ki*T*e;}      //diferencia hacia atras
267         //if((u>max) | (u<min)) i=i-Kiz*(e+e1);    //tustin
268         if(u>max)u=max;
269         if(u<min)u=min;
270
271         /** realizar la conversion final
272         control=(unsigned int16)u;
273         set_pwm1_duty(control);
274         e1=e;
275         i1=i;
276

```

```

277     OS_Delay(5000);
278     //OS_Bsem_Set(BS_GLAGLC_FREE);
279     OS_Yield();
280 }
281 }
282
283
284
285 void MotorX(void){
286     for(;;){
287         OS_Wait(numPulsosX>0);
288         if(dirMotorX==64)      output_high(dirX);
289         if(dirMotorX==128)     output_low(dirX);
290         if(--numPulsosX!=0)   output_toggle(clockX);
291         else                  output_low(clockX);
292         OS_Delay(periodoPulsosX);
293         OS_Yield();
294     }
295 }
296
297 void MotorY(void){
298     for(;;){
299         OS_Wait(numPulsosY>0);
300         if(dirMotorY==32)      output_high(dirY);
301         if(dirMotorY==16)      output_low(dirY);
302         if(--numPulsosY!=0)   output_toggle(clockY);
303         else                  output_low(clockY);
304         OS_Delay(periodoPulsosY);
305         OS_Yield();
306     }
307 }
308
309 void MotorZ(void){
310     for(;;){
311         OS_Wait(numPulsosZ>0);
312         if(dirMotorZ==8)      output_high(dirZ);
313         if(dirMotorZ==4)      output_low(dirZ);
314         if(--numPulsosZ!=0)   output_toggle(clockZ);
315         else                  output_low(clockZ);
316         OS_Delay(periodoPulsosZ);
317         OS_Yield();
318     }
319 }
320
321 void MotorE(void){
322     for(;;){
323         OS_Wait(numPulsosE>0);
324         if(dirMotorE==1)      output_high(dirE);
325         if(dirMotorE==2)      output_low(dirE);
326         if(--numPulsosE!=0)   output_toggle(clockE);
327         else                  output_low(clockE);
328         OS_Delay(periodoPulsosE);
329         OS_Yield();
330     }
331 }

```

Código 1: Firmware completo.

Fuente: Elaboración propia.

El siguiente código fuente detalla las configuraciones del descriptor usb para la correcta instalación del driver en el sistema operativo Windows 7 x86.

```

1 ////////////// config options, although it's best to leave alone for this demo
2     //////
3 #define USB_CONFIG_PID          0x0011    //changing this value may make the
4     //driver incompatible
5 #define USB_CONFIG_VID          0x04D8    //changing this value may make the
6     //driver incompatible
7 #define USB_CONFIG_BUS_POWER    100      //100mA (range is 0..500)
8 #define USB_CONFIG_VERSION     0x0100    //01.00 //range is 00.00 to 99.99
9 ////////////// end config
10 //////////////////////////////////////////////////////////////////
11 // Here is where the "CCS" Manufacturer string and "CCS Bulk Demo" are stored.
12 // Strings are saved as unicode.
13 // These strings are mostly only displayed during the add hardware wizard.
14 // Once the operating system drivers have been installed it will usually
15 // display
16 // the name from the drivers .INF.
17 char const USB_STRING_DESC []={
18     //string 0
19         4, //length of string index
20             USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
21             0x09,0x04, //Microsoft Defined for US-English
22     //string 1
23         8, //length of string index
24             USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
25             'C',0,
26             'C',0,
27             'S',0,
28     //string 2
29         24, //length of string index
30             USB_DESC_STRING_TYPE, //descriptor type 0x03 (STRING)
31             '3',0,
32             'D',0,
33             ' ',0,
34             'R',0,
35             'E',0,
36             'P',0,
37             'E',0,
38             'T',0,
39             ' ',0,
40             'P',0,
41             'L',0
42 };

```

Código 2: Secciones VID/PID - name de usb_descriptor.h

Fuente: Elaboración propia.

. Anexo B: Código fuente de *Software*

Los códigos fuente siguientes detallan las **clases** usadas en el desarrollo del **Software** de operación, control, comunicación y monitoreo del prototipo, sobre el entorno de desarrollo **Visual Studio C#**.

```
1 namespace ControlUSB
2 {
3     public partial class InterfazMain3DPrinter : Form
4     {
5         //-- atributos USB
6         private byte[] Enviar_Byte = new byte[20];
7         private byte[] Recibir_Byte = new byte[3];
8         public string Nombre = "3D Printer Repet";
9         public string vid_pid = "vid_04D8&pid_0011";
10
11        //-- atributos Monitoreo
12        public double ADC = 0;
13        public string Direccion;
14        public DateTime tiempoi, tiempof;
15        public Int64 Contador = 0;
16        public byte GuardarDocumento = 0;
17        //-- objetos
18        PICUSBapi USBapi = new PICUSBapi();
19        GcodeDocument Documento = new GcodeDocument();
20        Impresion3D Impresion = new Impresion3D();
21        //-----
22        public InterfazMain3DPrinter()
23        {
24            InitializeComponent();
25        }
26
27        #region Grupo Metodo Generico: Inicio, StatusDispositivo, Cierre_App
28        sirve para limpiar EnviarBytes
29        private void Inicio(object sender, EventArgs e)
30        {
31            Status_Dispositivo();
32        }
33
34        private void Status_Dispositivo()
35        {
36            uint Cuenta = PICUSBapi._MPUSBGetDeviceCount(vid_pid);
37            if (this.WindowState != FormWindowState.Minimized) //Show the
average colors on screen
            {

```

```

38         if (Cuenta == 0)
39         {
40             StatusStrip.Text = string.Format("{0} No Conectado",
41             Nombre);
42             StatusStrip.ForeColor = Color.Red;
43         }
44         else
45         {
46             StatusStrip.Text = string.Format("{0} Conectado", Nombre);
47             StatusStrip.ForeColor = Color.Green;
48         }
49     }
50
51     private void Cierre_App(object sender, FormClosingEventArgs e)
52     {
53         bool close = false;
54         this.TopMost = false;
55
56         if (e.CloseReason != CloseReason.WindowsShutDown)
57         {
58             DialogResult result = MessageBox.Show("Está seguro que desea
59             salir?", string.Format("{0}", Nombre), MessageBoxButtons.YesNo);
60             if (result == DialogResult.Yes)
61             {
62                 close = true;
63                 Enviar_Data(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
64             }
65             else
66             {
67                 close = true;
68             }
69             if (close == true)
70             {
71                 // Al cerrar la aplicación, llevo las salidas a 0.
72                 Enviar_Data(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
73             }
74             else
75             {
76                 e.Cancel = true;
77                 this.TopMost = true;
78             }
79         }
80     #endregion
81
82     #region Metodo Enviar_Data
83     public void Enviar_Data(UInt16 NumPulsosX, UInt16 NumPulsosY, UInt16
84     NumPulsosZ, UInt16 NumPulsosE, UInt16 PeriodoPulsosX, UInt16
85     PeriodoPulsosY, byte PeriodoPulsosZ, byte PeriodoPulsosE, byte DirMotorX,
86     byte DirMotorY, byte DirMotorZ, byte DirMotorE, byte GlaGlc, byte PWM)
87     {
88         Enviar_Byte[0] = (byte)(NumPulsosX >> 8); // Se envia en 2
89         partes para reensamblarse en el PIC
90         Enviar_Byte[1] = (byte)(NumPulsosX & 0xFF);
91         Enviar_Byte[2] = (byte)(NumPulsosY >> 8);
92         Enviar_Byte[3] = (byte)(NumPulsosY & 0xFF);
93         Enviar_Byte[4] = (byte)(NumPulsosZ >> 8);
94         Enviar_Byte[5] = (byte)(NumPulsosZ & 0xFF);

```

```

91     Enviar_Byte[6] = (byte)(NumPulsosE >> 8);
92     Enviar_Byte[7] = (byte)(NumPulsosE & 0xFF);
93     Enviar_Byte[8] = (byte)(PeriodoPulsosX >> 8);
94     Enviar_Byte[9] = (byte)(PeriodoPulsosX & 0xFF);
95     Enviar_Byte[10] = (byte)(PeriodoPulsosY >> 8);
96     Enviar_Byte[11] = (byte)(PeriodoPulsosY & 0xFF);
97     Enviar_Byte[12] = (byte)PeriodoPulsosZ;
98     Enviar_Byte[13] = (byte)PeriodoPulsosE;
99     Enviar_Byte[14] = (byte)DirMotorX;
100    Enviar_Byte[15] = (byte)DirMotorY;
101    Enviar_Byte[16] = (byte)DirMotorZ;
102    Enviar_Byte[17] = (byte)DirMotorE;
103    Enviar_Byte[18] = (byte)GlaGlc;
104    Enviar_Byte[19] = (byte)PWM;
105    this.USBApi.EnviarDatos(Enviar_Byte[0], Enviar_Byte[1],
106    Enviar_Byte[2], Enviar_Byte[3], Enviar_Byte[4], Enviar_Byte[5],
107    Enviar_Byte[6], Enviar_Byte[7], Enviar_Byte[8], Enviar_Byte[9],
108    Enviar_Byte[10], Enviar_Byte[11], Enviar_Byte[12], Enviar_Byte[13],
109    Enviar_Byte[14], Enviar_Byte[15], Enviar_Byte[16], Enviar_Byte[17],
110    Enviar_Byte[18], Enviar_Byte[19]);
111    }
112    #endregion
113
114    #region Grupo Metodo Control Manual XYZ
115    private void buttonPositivoX_Click_1(object sender, EventArgs e)
116    {
117        Enviar_Data(Convert.ToInt16(textBoxNumPulsosX.Text), 0, 0, 0,
118        Convert.ToInt16(decimal.Round(Convert.ToDecimal(textBoxPeriodoPulsosX.
119        Text) * 0.5m)), 0, 0, 0, 128, 0, 0, 0, (byte)trackBarGlaGlc.Value, (byte)
120        hScrollBarPWM.Value);
121    }
122
123    private void buttonNegativoX_Click(object sender, EventArgs e)
124    {
125        Enviar_Data(Convert.ToInt16(textBoxNumPulsosX.Text), 0, 0, 0,
126        Convert.ToInt16(decimal.Round(Convert.ToDecimal(textBoxPeriodoPulsosX.
127        Text) * 0.5m)), 0, 0, 0, 64, 0, 0, 0, (byte)trackBarGlaGlc.Value, (byte)
128        hScrollBarPWM.Value);
129
130    private void buttonPositivoY_Click(object sender, EventArgs e)
131    {
132        Enviar_Data(0, Convert.ToInt16(textBoxNumPulsosY.Text), 0, 0, 0,
133        Convert.ToInt16(decimal.Round(Convert.ToDecimal(textBoxPeriodoPulsosY.
134        Text) * 0.5m)), 0, 0, 0, 32, 0, 0, (byte)trackBarGlaGlc.Value, (byte)
135        hScrollBarPWM.Value);
136
137    private void buttonNegativoY_Click(object sender, EventArgs e)
138    {
139        Enviar_Data(0, Convert.ToInt16(textBoxNumPulsosY.Text), 0, 0, 0,
140        Convert.ToInt16(decimal.Round(Convert.ToDecimal(textBoxPeriodoPulsosY.
141        Text) * 0.5m)), 0, 0, 0, 16, 0, 0, (byte)trackBarGlaGlc.Value, (byte)
142        hScrollBarPWM.Value);
143
144    private void buttonPositivoZ_Click(object sender, EventArgs e)
145    {
146        Enviar_Data(0, 0, Convert.ToInt16(textBoxNumPulsosZ.Text), 0, 0,
147

```

```

0, Convert.ToByte(decimal.Round(Convert.ToDecimal(textBoxPeriodoPulsosZ.
Text) * 0.5m)), 0, 0, 0, 8, 0, (byte)trackBarGlaGlc.Value, (byte)
hScrollBarPWM.Value);
}
}

private void buttonNegativoZ_Click(object sender, EventArgs e)
{
    Enviar_Data(0, 0, Convert.ToInt16(textBoxNumPulsosZ.Text), 0, 0,
0, Convert.ToByte(decimal.Round(Convert.ToDecimal(textBoxPeriodoPulsosZ.
Text) * 0.5m)), 0, 0, 0, 4, 0, (byte)trackBarGlaGlc.Value, (byte)
hScrollBarPWM.Value);
}

private void buttonPositivoE_Click(object sender, EventArgs e)
{
    Enviar_Data(0, 0, 0, Convert.ToInt16(textBoxNumPulsosE.Text), 0,
0, 0, Convert.ToByte(decimal.Round(Convert.ToDecimal(textBoxPeriodoPulsosE
.Text) * 0.5m)), 0, 0, 0, 2, (byte)trackBarGlaGlc.Value, (byte)
hScrollBarPWM.Value);
}

private void buttonNegativoE_Click(object sender, EventArgs e)
{
    Enviar_Data(0, 0, 0, Convert.ToInt16(textBoxNumPulsosE.Text), 0,
0, 0, Convert.ToByte(decimal.Round(Convert.ToDecimal(textBoxPeriodoPulsosE
.Text) * 0.5m)), 0, 0, 0, 1, (byte)trackBarGlaGlc.Value, (byte)
hScrollBarPWM.Value);
}

#endregion

#region Metodo Seleccion Gla - Glc
private void trackBarGlaGlc_Scroll(object sender, EventArgs e)
{
    Enviar_Data(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, (byte)
trackBarGlaGlc.Value, (byte)hScrollBarPWM.Value);
}

#endregion

#region Metodo Referencia r(t) PWM
private void hScrollBarPWM_Scroll(object sender, ScrollEventArgs e)
{
    //int PWM = int.Parse(textBoxDatoPWM.Text);
    labelDatoPWMEniado.Text = Convert.ToString(hScrollBarPWM.Value *
0.01961);      //conversion; 1V=51; 5[V]/255=0.01961(teórico) || 4.96[V
]/255=0.01945(práctico).... (revisar notas)
    labelDatoTempEniado.Text = Convert.ToString(hScrollBarPWM.Value *
1.17647);      //conversion; 1V=51; 300[°C]/255=1.17647

    Enviar_Data(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, (byte)
trackBarGlaGlc.Value, (byte)hScrollBarPWM.Value);
}

#endregion

#region Grupo Metodo Monitoreo: Iniciar + Parar + Limpiar + Guardar +
TimerMuestreo_Tick
private void buttonIniciarMuestreo_Click(object sender, EventArgs e)
{
    TimerMuestreo.Interval = Convert.ToInt16(textBoxTiempoDeMuestreo.
Text);
}

```

```
173         TimerMuestreo.Start();
174     }
175
176     private void buttonPararMuestreo_Click(object sender, EventArgs e)
177     {
178         GuardarDocumento = 0;
179         Contador = 0;
180         labelNumDeMuestras.Text = "0";
181         TimerMuestreo.Stop();
182     }
183
184     private void buttonLimpiarMuestreo_Click(object sender, EventArgs e)
185     {
186         chartRvsY.Series["SeriesY"].Points.Clear();
187         chartRvsY.Series["SeriesR"].Points.Clear();
188     }
189
190     private void buttonGuardarMuestreo_Click(object sender, EventArgs e)
191     {
192         // Configure save file dialog box
193         SaveFileDialog dlg1 = new SaveFileDialog();
194         dlg1.Title = "Guardar Archivo de datos";
195         dlg1.Filter = "Archivo de Texto (.txt) |*.txt";
196         dlg1.DefaultExt = ".txt";
197         dlg1.AddExtension = true;
198         dlg1.RestoreDirectory = true;
199
200         // Show open file dialog box
201         DialogResult result = dlg1.ShowDialog();
202
203         // Process open file dialog box results
204         if (result == DialogResult.OK)
205         {
206             string ruta = dlg1.FileName;
207             Direccion = ruta;
208             TextWriter fichero = new StreamWriter(ruta);
209             fichero.WriteLine("Datos de la Planta a lazo abierto (Gla) o");
210             lazo cerrado (Glc)");
211             fichero.WriteLine(ruta);
212             tiempoi = DateTime.Now;
213             fichero.WriteLine(tiempoi);
214             fichero.WriteLine();
215             fichero.WriteLine("{0} {1} {2} {3}",
216             "r(t):Voltaje[V]", "y(t):Voltaje[V]", "y(t):Temp[°C]", "Tiempo de muestreo");
217             :tm[ms]");
218             fichero.Close();
219             MessageBox.Show("Se ha guardado el archivo: " + dlg1.FileName);
220 ;
221             labelNumDeMuestras.Text = "0";
222
223             int intervalo = int.Parse(textBoxTiempoDeMuestreo.Text);
224             TimerMuestreo.Interval = intervalo;
225
226             GuardarDocumento = 1;
227         }
228         else
229         {
230             GuardarDocumento = 0;
231             MessageBox.Show("Datos no creados.");
232         }
233     }
234 }
```

```

228         }
229         dlg1.Dispose();
230         dlg1 = null;
231     }
232
233     private void TimerMuestreo_Tick(object sender, EventArgs e)
234     {
235         for (int i = 0; i < 3; i++)      //con el incrementador llena los
bytes "Recibir_Byte" con el contenido de "this.USBAPI.RecibirDatos(i)"
236         {
237             Recibir_Byte[i] = this.USBapi.RecibirDatos(i);
238         }
239         ADC = (UInt16)Recibir_Byte[0] * 256 + (UInt16)Recibir_Byte[1];
//rearma el valor de ADC del PIC y pone rango de 0 a 1023
240
241         if (GuardarDocumento == 1)
242         {
243             Contador++;
244             TimeSpan duracion = DateTime.Now - tiempoi;
245             StreamWriter fichero = new StreamWriter(Direccion, true);
246             Thread.CurrentThread.CurrentCulture = CultureInfo.
CreateSpecificCulture("en-US");    //para WriteLine con la notación "en-US"
"23,713.22476" para trabajar directamente en Matlab porque por defecto
fue compilado en "es-BO"
247             fichero.WriteLine("{0} {1}
{2} {3}", Convert.ToString(decimal.Round(Convert.
.ToDecimal(hScrollBarPWM.Value * 0.01961), 3)), Convert.ToString(decimal.
Round(Convert.ToDecimal(ADC * 0.00488), 3)), Convert.ToString(decimal.
Round(Convert.ToDecimal(ADC * 0.00488 * 60), 3)), Convert.ToString(decimal.
Round(Convert.ToDecimal(duracion.TotalMilliseconds), 3)) );
248             fichero.Close();
249             labelNumDeMuestras.Text = Convert.ToString(Contador);
250         }
251
252         progressBarADC.Value = (UInt16)ADC;
253         labelValorADC.Text = Convert.ToString(ADC);
254         labelValorVoltaje.Text = Convert.ToString(ADC * 0.00488);    //5[V
]/1023[bits]=0.00488
255         labelValorTemp.Text = Convert.ToString(ADC * 0.00488 * 60); //300[
°C]/5[V]=60
256
257         chartRvsY.Series["SeriesR"].Points.AddY(hScrollBarPWM.Value *
0.01961);
258         chartRvsY.Series["SeriesY"].Points.AddY(ADC * 0.00488);
259     }
260     #endregion
261
262     #region Grupo Metodo Gcode: CargarGcode + Step + buttonStep +
TimerStep + Enviar Gcode + ResetStep
263     private void buttonCargarGcode_Click(object sender, EventArgs e)
264     {
265         // Configure open file dialog box
266         OpenFileDialog dlg = new OpenFileDialog();
267         dlg.FileName = "Documento"; // Default file name
268         dlg.DefaultExt = ".gcode"; // Default file extension
269         dlg.Filter = "Documento Gcode (.gcode)|*.gcode"; // Filter files
by extension
270
271         // Show open file dialog box

```

```

272     DialogResult result = dlg.ShowDialog();
273
274     // Process open file dialog box results
275     if (result == DialogResult.OK)
276     {
277         // Procesa "dlg.FileName" a traves de la clase "GcodeDocument"
278         try
279         {
280             this.Documento.Cargar(dlg.FileName);
281             if (this.Documento.GcodeCargado == true)
282             {
283                 MessageBox.Show("Gcode cargado correctamente ");
284                 labelSizeLinesGcodeDocument.Text = Convert.ToString(
285                     this.Documento.Gcodes.Length);
286                 return;
287             }
288         }
289         catch (Exception ex)
290         {
291             MessageBox.Show("Error al cargar el Gcode: " + ex.Message)
292 ;
293             return;
294         }
295     }
296
297     private void Step()
298     {
299         if (this.Documento.GcodeCargado == false)
300         {
301             MessageBox.Show("Gcode no cargado ");
302             return;
303         }
304         else
305         {
306             this.Impresion.SetUpGcodeHandlers();
307             this.Impresion.IniciarImpresion(this.Documento.Gcodes);
308             if (this.Impresion.EndOfDocument == false)
309                 Enviar_Data(Convert.ToInt16(this.Impresion.Delta.X),
310                 Convert.ToInt16(this.Impresion.Delta.Y), Convert.ToInt16(this.Impresion.
311                 Delta.Z), Convert.ToInt16(this.Impresion.Delta.E), Convert.ToInt16(this.
312                 Impresion.Periodo.X), Convert.ToInt16(this.Impresion.Periodo.Y), Convert.
313                 ToByte(this.Impresion.Periodo.Z), Convert.ToByte(this.Impresion.Periodo.E)
314                 , Convert.ToByte(this.Impresion.DirMotor.X), Convert.ToByte(this.Impresion.
315                 DirMotor.Y), Convert.ToByte(this.Impresion.DirMotor.Z), Convert.ToByte(
316                 this.Impresion.DirMotor.E), (byte)trackBarGlaGlc.Value, (byte)
hScrollBarPWM.Value);
317             else TimerStep.Stop();
318
319             labelLineaActual.Text = Convert.ToString(this.Impresion.
320             PosicionLinea);
321
322             labelPosFinX.Text = Convert.ToString(this.Impresion.
323             PosicionFinal.X);
324             labelPosFinY.Text = Convert.ToString(this.Impresion.
325             PosicionFinal.Y);
326             labelPosFinZ.Text = Convert.ToString(this.Impresion.
327             PosicionFinal.Z);

```

```

317         labelPosFinE.Text = Convert.ToString(this.Impresion.
318             PosicionFinal.E);
319         labelPosFinF.Text = Convert.ToString(this.Impresion.
320             PosicionFinal.F);
321         labelDeltaX.Text = Convert.ToString(this.Impresion.Delta.X);
322         labelDeltaY.Text = Convert.ToString(this.Impresion.Delta.Y);
323         labelDeltaZ.Text = Convert.ToString(this.Impresion.Delta.Z);
324         labelDeltaE.Text = Convert.ToString(this.Impresion.Delta.E);
325         labelPeriodoX.Text = Convert.ToString(this.Impresion.Periodo.X
326 );
326         labelPeriodoY.Text = Convert.ToString(this.Impresion.Periodo.Y
327 );
327         labelPeriodoZ.Text = Convert.ToString(this.Impresion.Periodo.Z
328 );
328         labelPeriodoE.Text = Convert.ToString(this.Impresion.Periodo.E
329 );
329         labelDeltaPeriodoX.Text = Convert.ToString(this.Impresion.
330             Delta.X * this.Impresion.Periodo.X);
330         labelDeltaPeriodoY.Text = Convert.ToString(this.Impresion.
331             Delta.Y * this.Impresion.Periodo.Y);
331         labelDeltaPeriodoZ.Text = Convert.ToString(this.Impresion.
332             Delta.Z * this.Impresion.Periodo.Z);
332         labelDeltaPeriodoE.Text = Convert.ToString(this.Impresion.
333             Delta.E * this.Impresion.Periodo.E);
333         labelDirX.Text = Convert.ToString(this.Impresion.DirMotor.X);
334         labelDirY.Text = Convert.ToString(this.Impresion.DirMotor.Y);
335         labelDirZ.Text = Convert.ToString(this.Impresion.DirMotor.Z);
336         labelDirE.Text = Convert.ToString(this.Impresion.DirMotor.E);
337     }
338 
339     private void buttonStep_Click(object sender, EventArgs e)
340     {
341         TimerStep.Stop();
342         Step();
343     }
344 
345     private void buttonTimerStep_Click(object sender, EventArgs e)
346     {
347         if (this.Documento.GcodeCargado == false)
348         {
349             MessageBox.Show("Gcode no cargado ");
350             return;
351         }
352         TimerStep.Start();
353     }
354 
355     private void TimerStep_Tick(object sender, EventArgs e)
356     {
357         if (Recibir_Byte[2] == 0)
358         {
359             TimerStep.Interval = Convert.ToInt16(textBoxTimerStep.Text);
360             Step();
361         }
362         else return;
363     }
364 
365     private void buttonEnviarGcode_Click(object sender, EventArgs e)
366     {

```

```

366         this.Impresion.SetUpGcodeHandlers();
367         this.Impresion.Step(textBoxEnviarGcode.Text);
368
369         Enviar_Data(Convert.ToInt16(this.Impresion.Delta.X), Convert.
370 ToUInt16(this.Impresion.Delta.Y), Convert.ToInt16(this.Impresion.Delta.Z)
371 , Convert.ToInt16(this.Impresion.Delta.E), Convert.ToInt16(this.
372 Impresion.Periodo.X), Convert.ToInt16(this.Impresion.Periodo.Y), Convert.
373 ToByte(this.Impresion.Periodo.Z), Convert.ToByte(this.Impresion.Periodo.E)
374 , Convert.ToByte(this.Impresion.DirMotor.X), Convert.ToByte(this.Impresion
375 .DirMotor.Y), Convert.ToByte(this.Impresion.DirMotor.Z), Convert.ToByte(
376 this.Impresion.DirMotor.E), (byte)trackBarGlaGlc.Value, (byte)
377 hScrollBarPWM.Value);
378
379         labelPosFinX.Text = Convert.ToString(this.Impresion.PosicionFinal.
380 X);
381         labelPosFinY.Text = Convert.ToString(this.Impresion.PosicionFinal.
382 Y);
383         labelPosFinZ.Text = Convert.ToString(this.Impresion.PosicionFinal.
384 Z);
385         labelPosFinE.Text = Convert.ToString(this.Impresion.PosicionFinal.
386 E);
387         labelPosFinF.Text = Convert.ToString(this.Impresion.PosicionFinal.
388 F);
389         labelDeltaX.Text = Convert.ToString(this.Impresion.Delta.X);
390         labelDeltaY.Text = Convert.ToString(this.Impresion.Delta.Y);
391         labelDeltaZ.Text = Convert.ToString(this.Impresion.Delta.Z);
392         labelDeltaE.Text = Convert.ToString(this.Impresion.Delta.E);
393         labelPeriodoX.Text = Convert.ToString(this.Impresion.Periodo.X);
394         labelPeriodoY.Text = Convert.ToString(this.Impresion.Periodo.Y);
395         labelPeriodoZ.Text = Convert.ToString(this.Impresion.Periodo.Z);
396         labelPeriodoE.Text = Convert.ToString(this.Impresion.Periodo.E);
397         labelDeltaPeriodoX.Text = Convert.ToString(this.Impresion.Delta.X
398 * this.Impresion.Periodo.X);
399         labelDeltaPeriodoY.Text = Convert.ToString(this.Impresion.Delta.Y
400 * this.Impresion.Periodo.Y);
401         labelDeltaPeriodoZ.Text = Convert.ToString(this.Impresion.Delta.Z
402 * this.Impresion.Periodo.Z);
403         labelDeltaPeriodoE.Text = Convert.ToString(this.Impresion.Delta.E
404 * this.Impresion.Periodo.E);
405         labelDirX.Text = Convert.ToString(this.Impresion.DirMotor.X);
406         labelDirY.Text = Convert.ToString(this.Impresion.DirMotor.Y);
407         labelDirZ.Text = Convert.ToString(this.Impresion.DirMotor.Z);
408         labelDirE.Text = Convert.ToString(this.Impresion.DirMotor.E);
409
410         this.Impresion.Reset();
411     }
412
413     private void buttonResetStep_Click(object sender, EventArgs e)
414     {
415         TimerStep.Stop();
416         this.Impresion.Reset();
417         Enviar_Data(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
418 (byte)trackBarGlaGlc.Value, (byte)hScrollBarPWM.Value);
419
420         labelLineaActual.Text = "0";
421
422         labelPosFinX.Text = "0";
423         labelPosFinY.Text = "0";
424         labelPosFinZ.Text = "0";

```

```

407         labelPosFinE.Text = "0";
408         labelPosFinF.Text = "0";
409         labelDeltaX.Text = "0";
410         labelDeltaY.Text = "0";
411         labelDeltaZ.Text = "0";
412         labelDeltaE.Text = "0";
413         labelPeriodoX.Text = "0";
414         labelPeriodoY.Text = "0";
415         labelPeriodoZ.Text = "0";
416         labelPeriodoE.Text = "0";
417         labelDeltaPeriodoX.Text = "0";
418         labelDeltaPeriodoY.Text = "0";
419         labelDeltaPeriodoZ.Text = "0";
420         labelDeltaPeriodoE.Text = "0";
421         labelDirX.Text = "0";
422         labelDirY.Text = "0";
423         labelDirZ.Text = "0";
424         labelDirE.Text = "0";
425     }
426 #endregion
427
428
429
430     private void buttonTest_Click(object sender, EventArgs e)
431     {
432         //decimal r = decimal.Round(Convert.ToDecimal("13.224",
433         CultureInfo.CreateSpecificCulture("en-US")),1);
434         //labelTest1.Text = Convert.ToString(Math.Round(Convert.ToDouble
435         ("23,713.22476", CultureInfo.CreateSpecificCulture("en-US")), 3));
436         //labelTest2.Text = Convert.ToString(decimal.Round(Convert.
437         ToDecimal("23713,22476", CultureInfo.CreateSpecificCulture("es-BO")), 3));
438         //-- Nota: Double vs Decimal: Decimal tiene mas precision y menos
439         rango, lo que reduce el consumo de ram
440         //--
441         // pero Decimal no tiene conversion
442         // implicita con int, short, float, como lo tiene Double.
443
444         UInt16 Xenviado;    //100[mm]*5.88[pulsos/mm]=588[pulsos]
445         decimal Xprecibido;
446         byte enviaXa,enviaXb;
447
448         Xenviado = Convert.ToUInt16(textBoxTest.Text);
449         enviaXa = (byte)(Xenviado >> 8);
450         enviaXb = (byte)(Xenviado & 0xFF);
451
452         Xprecibido = (byte)enviaXa * 256 + (byte)enviaXb;
453         labelTest2.Text = Convert.ToString(Xprecibido);
454     }
455 }

```

Código 3: Clase InterfazMain3DPrinter completo.

Fuente: Elaboración propia.

```

1 namespace ControlUSB
2 {

```

```

3 unsafe public class PICUSBapi
4 {
5     #region Atributos Strings: EndPoint y VID_PID
6     string vid_pid_norm = "vid_04D8&pid_0011";
7
8     string out_pipe = "\\\\"MCHP_EP1";
9     string in_pipe = "\\\\"MCHP_EP1";
10    #endregion
11
12    #region Metodos de importacion mpusbapi.dll
13    [DllImport("mpusbapi.dll")]
14    private static extern DWORD _MPUSBGetDLLVersion();
15    [DllImport("mpusbapi.dll")]
16    public static extern DWORD _MPUSBGetDeviceCount(string pVID_PID);
17    [DllImport("mpusbapi.dll")]
18    private static extern void* _MPUSBOpen(DWORD instance, string pVID_PID
19     , string pEP, DWORD dwDir, DWORD dwReserved);
20    [DllImport("mpusbapi.dll")]
21    private static extern DWORD _MPUSBRead(void* handle, void* pData,
22     DWORD dwLen, DWORD* pLength, DWORD dwMilliseconds);
23    [DllImport("mpusbapi.dll")]
24    private static extern DWORD _MPUSBWrite(void* handle, void* pData,
25     DWORD dwLen, DWORD* pLength, DWORD dwMilliseconds);
26    [DllImport("mpusbapi.dll")]
27    private static extern DWORD _MPUSBReadInt(void* handle, DWORD* pData,
28     DWORD dwLen, DWORD* pLength, DWORD dwMilliseconds);
29    [DllImport("mpusbapi.dll")]
30    private static extern bool _MPUSBClose(void* handle);
31    #endregion
32
33    #region Metodos para el manejo del mpusbapi.dll
34    void* myOutPipe;
35    void* myInPipe;
36
37    public void OpenPipes()
38    {
39        DWORD selection = 0;
40
41        myOutPipe = _MPUSBOpen(selection, vid_pid_norm, out_pipe, 0, 0);
42        myInPipe = _MPUSBOpen(selection, vid_pid_norm, in_pipe, 1, 0);
43    }
44
45    public void ClosePipes()
46    {
47        _MPUSBClose(myOutPipe);
48        _MPUSBClose(myInPipe);
49    }
50
51    private void SendPacket(byte* SendData, DWORD SendLength)
52    {
53        uint SendDelay = 1000;
54
55        DWORD SentDataLength;
56
57        OpenPipes();
58        _MPUSBWrite(myOutPipe, (void*)SendData, SendLength, &
59        SentDataLength, SendDelay);
60        ClosePipes();
61    }

```

```

57     private void ReceivePacket(byte* ReceiveData, DWORD* ReceiveLength)
58     {
59         uint ReceiveDelay = 1000;
60
61         DWORD ExpectedReceiveLength = *ReceiveLength;
62
63         OpenPipes();
64         _MPUSBRead(myInPipe, (void*)ReceiveData, ExpectedReceiveLength,
65         ReceiveLength, ReceiveDelay);
66         ClosePipes();
67     }
68 #endregion
69
70     #region Metodos de EnviarDatos y RecibirDatos que llaman a SendPacket
71     y ReceivePacket
72     public void EnviarDatos(byte Byte0, byte Byte1, byte Byte2, byte Byte3
73     , byte Byte4, byte Byte5, byte Byte6, byte Byte7, byte Byte8, byte Byte9,
74     byte Byte10, byte Byte11, byte Byte12, byte Byte13, byte Byte14, byte
75     Byte15, byte Byte16, byte Byte17, byte Byte18, byte Byte19)
76     {
77         byte* Enviar_Buffer = stackalloc byte[20]; //define buffer de 20
78         bytes para enviar
79
80         Enviar_Buffer[0] = (byte)Byte0;
81         Enviar_Buffer[1] = (byte)Byte1;
82         Enviar_Buffer[2] = (byte)Byte2;
83         Enviar_Buffer[3] = (byte)Byte3;
84         Enviar_Buffer[4] = (byte)Byte4;
85         Enviar_Buffer[5] = (byte)Byte5;
86         Enviar_Buffer[6] = (byte)Byte6;
87         Enviar_Buffer[7] = (byte)Byte7;
88         Enviar_Buffer[8] = (byte)Byte8;
89         Enviar_Buffer[9] = (byte)Byte9;
90         Enviar_Buffer[10] = (byte)Byte10;
91         Enviar_Buffer[11] = (byte)Byte11;
92         Enviar_Buffer[12] = (byte)Byte12;
93         Enviar_Buffer[13] = (byte)Byte13;
94         Enviar_Buffer[14] = (byte)Byte14;
95         Enviar_Buffer[15] = (byte)Byte15;
96         Enviar_Buffer[16] = (byte)Byte16;
97         Enviar_Buffer[17] = (byte)Byte17;
98         Enviar_Buffer[18] = (byte)Byte18;
99         Enviar_Buffer[19] = (byte)Byte19;
100
101         SendPacket(Enviar_Buffer, 20); //envia el buffer con SendPacket
102     }
103
104     public byte RecibirDatos(int i)
105     {
106         byte* Recibir_Buffer = stackalloc byte[3]; //define buffer de 3
107         bytes para recibir
108         DWORD Longitud = 3;
109
110         ReceivePacket(Recibir_Buffer, &Longitud);
111
112         return Recibir_Buffer[i];
113     }
114 #endregion

```

```

109 }
110 }
```

Código 4: Clase PICUSBapi completo.
Fuente: Elaboración propia.

```

1   public void Cargar(string documento)
2   {
3       try
4       {
5           Gcodes = File.ReadAllLines(documento);
6           GcodeCargado = true;
7       }
8       catch (Exception)
9       {
10           Gcodes = new string[] {};
11           GcodeCargado = false;
12       }
13   }
14 }
15 }
16 }
```

Código 5: Clase GcodeDocument completo.
Fuente: Elaboración propia.

```

1 namespace ControlUSB
2 {
3     [Serializable()]
4     public struct Vector5D
5     {
6         public decimal X, Y, Z, E, F;
7
8         public Vector5D(decimal _X, decimal _Y, decimal _Z, decimal _E,
9 decimal _F)
10        {
11            this.X = _X;
12            this.Y = _Y;
13            this.Z = _Z;
14            this.E = _E;
15            this.F = _F;
16        }
17
18
19        public static Vector5D operator +(Vector5D v1, Vector5D v2)
20        {
21            return new Vector5D(v1.X + v2.X, v1.Y + v2.Y, v1.Z + v2.Z, v1.E +
22 v2.E, v1.F + v2.F);
23        }
24
25        public static Vector5D operator -(Vector5D v1, Vector5D v2)
26        {
27            return new Vector5D(v1.X - v2.X, v1.Y - v2.Y, v1.Z - v2.Z, v1.E -
28 v2.E, v1.F - v2.F);
29        }
30
31        public static Vector5D operator *(Vector5D v1, Vector5D v2)
```

```

30         {
31             return new Vector5D(v1.X * v2.X, v1.Y * v2.Y, v1.Z * v2.Z, v1.E *
32             v2.E, v1.F * v2.F);
33         }
34
35         /*public static Vector5D Abs(Vector5D v)
36         {
37             v.X = Math.Abs(v.X);
38             v.Y = Math.Abs(v.Y);
39             v.Z = Math.Abs(v.Z);
40             v.E = Math.Abs(v.E);
41             v.F = Math.Abs(v.F);
42             return v;
43         }*/
44
45         public static Vector5D Abs(Vector5D v)
46         {
47             return new Vector5D(Math.Abs(v.X), Math.Abs(v.Y), Math.Abs(v.Z),
48             Math.Abs(v.E), Math.Abs(v.F));
49         }
50
51         public override string ToString()
52         {
53             return String.Format("<{0}, {1}, {2}, {3}, {4}>", this.X, this.Y,
54             this.Z, this.E, this.F);
55         }
56     }
57 }
```

Código 6: Clase Vector5D completo.

Fuente: Elaboración propia.

```

1     private string [] Gcodes;
2     public int PosicionLinea = 0;
3
4     public Vector5D PosicionInicial = new Vector5D(0.0m, 0.0m, 0.0m, 0.0m,
4         0.0m);
5     public Vector5D PosicionFinal = new Vector5D(0.0m , 0.0m, 0.0m, 0.0m,
5         0.0m);
6     public Vector5D Delta = new Vector5D(0.0m, 0.0m, 0.0m, 0.0m, 0.0m);
7     //delta = numPulos
8     public Vector5D Periodo = new Vector5D(0.0m, 0.0m, 0.0m, 0.0m, 0.0m);
9     public Vector5D DirMotor = new Vector5D(0.0m, 0.0m, 0.0m, 0.0m, 0.0m);
10
11    private delegate void PrintCommandHandler(string args);
12    private Dictionary<int, PrintCommandHandler> GCodeHandlers = new
13        Dictionary<int, PrintCommandHandler>();
14
15    #region SetUpGcodeHandlers + IniciarImpresion + Reset + EndOfDocument
16    + Step
17    public void SetUpGcodeHandlers()
18    {
19        GCodeHandlers[1] = ControlledMove;
20        GCodeHandlers[92] = SetPosition;
21        GCodeHandlers[28] = GoHome;
22        // gcodes
23        /*
24        GCodeHandlers[0] = RapidMove;
25        GCodeHandlers[1] = ControlledMove;
```

```
23     GCodeHandlers[28] = GoHome;
24     GCodeHandlers[4] = Dwell;
25     GCodeHandlers[20] = SetImperial;
26     GCodeHandlers[21] = SetMetric;
27     GCodeHandlers[90] = AbsolutePositioning;
28     GCodeHandlers[91] = IncrementalPositioning;
29     GCodeHandlers[92] = SetPosition;
30
31     // mcodes
32     MCodeHandlers[0] = ShutDown;
33     MCodeHandlers[140] = SetBedTemp;
34     MCodeHandlers[113] = SetExtruderPot;
35     MCodeHandlers[109] = SetExtruderTempWait;
36     MCodeHandlers[107] = CoolerOff;
37     MCodeHandlers[106] = CoolerOn;
38     MCodeHandlers[104] = SetExtruderTemp;
39     MCodeHandlers[105] = GetExtruderTemp;
40     MCodeHandlers[110] = NewPrint;
41     MCodeHandlers[108] = ExtruderSpeed;
42     MCodeHandlers[101] = ExtruderOn;
43     MCodeHandlers[103] = ExtruderOff;
44     MCodeHandlers[141] = SetChamberTemp;
45     MCodeHandlers[142] = SetHoldingPressure;
46     */
47 }
48
49 public void IniciarImpresion(string[] gcodes)
50 {
51     Gcodes = gcodes;
52
53     if (EndOfDocument == true)
54     {
55         MessageBox.Show("Fin del documento Gcode ");
56         return;
57     }
58     else
59     {
60         Step("");
61     }
62 }
63
64 public bool EndOfDocument
65 {
66     get { return PosicionLinea >= Gcodes.Length; }
67 }
68
69 public void Reset()
70 {
71     PosicionLinea = 0;
72     this.PosicionInicial = new Vector5D(0.0m, 0.0m, 0.0m, 0.0m, 0.0m);
73     this.PosicionFinal = new Vector5D(0.0m, 0.0m, 0.0m, 0.0m, 0.0m);
74     this.Delta = new Vector5D(0.0m, 0.0m, 0.0m, 0.0m, 0.0m);
75 }
76
77 public void Step(string Gcode)
78 {
79     //-- verifica el doc Gcode cargado o la instruccion Gcode enviada
80     manualmente
81     string instr;
```

```

81         if (Gcode != "")
82         {
83             instr = Gcode;
84         }
85         else
86         {
87             //-- obtener siguiente linea de instruccion
88             instr = Gcodes[PosicionLinea++];
89         }
90         //-- buscar Gcode
91         Match match = Regex.Match(instr, @"^G(?:<code>\d+)\b(?:<args>.*)");
92         if (match.Groups["code"].ToString() != "")
93         {
94             int code = Convert.ToInt32(match.Groups["code"].ToString());
95             Debug.Assert(GCodeHandlers.ContainsKey(code), "Unknown G-Code:");
96             " + code.ToString());
97             if (GCodeHandlers.ContainsKey(code))
98             {
99                 string args = match.Groups["args"].ToString();
100                GCodeHandlers[code](args); //llama a la funcion
101                GCodeHandlers[1],[2], etc
102            }
103        }
104    #endregion
105
106    #region ProcesarCampoNumerico + Escalar + PromediarPeriodo +
107    DecidirDirMotor
108    private void ProcesarCampoNumerico(string args, string field, ref
109    decimal val)
110    {
111        Match match = Regex.Match(args, field + @"(?<val>[0-9+-\.]+\b)");
112        if (match.Groups["val"].ToString() != "")
113            val = decimal.Round(Convert.ToDecimal(match.Groups["val"].
114            ToString(), CultureInfo.CreateSpecificCulture("en-US")),0);
115
116        private void Escalar(ref Vector5D Fin, Vector5D Ini)
117        {
118            if (Fin.X != Ini.X) Fin.X = decimal.Round(Fin.X * 3.03m);      //
119            1[mm] = 3.03[pulos]
120            if (Fin.Y != Ini.Y) Fin.Y = decimal.Round(Fin.Y * 10m);      //
121            1[mm] = 10[pulos]
122            if (Fin.Z != Ini.Z) Fin.Z = decimal.Round(Fin.Z * 50m);      //
123            1[mm] = 50[pulos]
124            if (Fin.E != Ini.E) Fin.E = decimal.Round(Fin.E * 70m);      //
125            1[mm] = 70[pulos]
126            if (Fin.F != Ini.F) Fin.F = decimal.Round(2m * Convert.ToDecimal(
127            Math.Pow(2, Convert.ToDouble(-Fin.F * 2m / (60m * 20m)))) * 20m); //10[
128            mm/s] = 20[ms]
129        }
130
131        private void PromediarPeriodo(ref Vector5D T, Vector5D d, decimal f)
132        // 0.5, porque OSA toma PeriodoPulsos como semiPeriodoPulsos
133        {
134            if ((d.E == 0) && (d.Y == 0) && (d.X == 0))
135            {
136                T.E = T.Y = T.X = 0;

```

```

128     }
129     if ((d.E == 0) && (d.Y == 0) && (d.X != 0))
130     {
131         T.X = decimal.Round(3.3m * 7m * f * 0.5m);
132     }
133     if ((d.E == 0) && (d.Y != 0) && (d.X == 0))
134     {
135         T.Y = decimal.Round(7m * f * 0.5m);
136     }
137     if ((d.E == 0) && (d.Y != 0) && (d.X != 0))
138     {
139         T.Y = decimal.Round(7m * f * 0.5m);
140         T.X = decimal.Round((d.Y / d.X) * T.Y);
141     }
142     if ((d.E != 0) && (d.Y == 0) && (d.X == 0))
143     {
144         T.E = decimal.Round(f * 0.5m);
145     }
146     if ((d.E != 0) && (d.Y == 0) && (d.X != 0))
147     {
148         T.E = decimal.Round(f * 0.5m);
149         T.X = decimal.Round((d.E / d.X) * f * 0.5m);
150     }
151     if ((d.E != 0) && (d.Y != 0) && (d.X == 0))
152     {
153         T.E = decimal.Round(f * 0.5m);
154         T.Y = decimal.Round((d.E / d.Y) * f * 0.5m);
155     }
156     if ((d.E != 0) && (d.Y != 0) && (d.X != 0))
157     {
158         T.E = decimal.Round(f * 0.5m);
159         T.Y = decimal.Round((d.E / d.Y) * f * 0.5m);
160         T.X = decimal.Round((d.E / d.X) * f * 0.5m);
161     }
162
163     if (d.Z != 0) T.Z = decimal.Round(7m * f * 0.5m);
164     if (d.Z == 0) T.Z = 0;
165 }
166
167     private void DecidirDirMotor(ref Vector5D dir, Vector5D Fin, Vector5D
168 Ini)
169     {
170         if (Fin.X > Ini.X) dir.X = 128;
171         if (Fin.X < Ini.X) dir.X = 64;
172         if (Fin.Y > Ini.Y) dir.Y = 32;
173         if (Fin.Y < Ini.Y) dir.Y = 16;
174         if (Fin.Z > Ini.Z) dir.Z = 8;
175         if (Fin.Z < Ini.Z) dir.Z = 4;
176         if (Fin.E > Ini.E) dir.E = 2;
177         if (Fin.E < Ini.E) dir.E = 1;
178     }
179 #endregion
180
181 #region Funciones para los comandos Gcode
182     private void ControlledMove(string args)
183     {
184         ProcesarCampoNumerico(args, "X", ref this.PosicionFinal.X);
185         ProcesarCampoNumerico(args, "Y", ref this.PosicionFinal.Y);
186         ProcesarCampoNumerico(args, "Z", ref this.PosicionFinal.Z);
187     }
188 }
```

```

186     ProcesarCampoNumerico(args, "E", ref this.PosicionFinal.E);
187     ProcesarCampoNumerico(args, "F", ref this.PosicionFinal.F);
188
189     Escalar(ref this.PosicionFinal, this.PosicionInicial);
190
191     this.Delta = Vector5D.Abs(this.PosicionFinal - this.
192     PosicionInicial);
193
194     PromediarPeriodo(ref this.Periodo, this.Delta, this.PosicionFinal.
195     F);
196
197     DecidirDirMotor(ref this.DirMotor, this.PosicionFinal, this.
198     PosicionInicial);
199
200     this.PosicionInicial = this.PosicionFinal;
201 }
202
203     private void SetPosition(string args)
204 {
205     ProcesarCampoNumerico(args, "X", ref this.PosicionFinal.X);
206     ProcesarCampoNumerico(args, "Y", ref this.PosicionFinal.Y);
207     ProcesarCampoNumerico(args, "Z", ref this.PosicionFinal.Z);
208     ProcesarCampoNumerico(args, "E", ref this.PosicionFinal.E);
209     ProcesarCampoNumerico(args, "F", ref this.PosicionFinal.F);
210
211     this.PosicionInicial = this.PosicionFinal;
212 }
213
214     private void GoHome(string args)
215 {
216     this.PosicionFinal.X = 0;
217     this.PosicionFinal.Y = 0;
218     this.PosicionFinal.Z = 0;
219     this.PosicionFinal.E = 0;
220     this.PosicionFinal.F = 0;
221 }
222 #endregion
223 }
```

Código 7: Clase Impresion3D completo.

Fuente: Elaboración propia.

. Anexo C

.1. Control PID: Discretización general

Sea la ecuación general de control PID en tiempo continuo,

$$u(t) = K_p e(t) + \frac{K_p}{T_i} \int_0^t e(\tau) d\tau + K_p T_d \frac{de(t)}{dt} \quad (1)$$

aplicando transformada de Laplace,

$$\begin{aligned} U(s) &= K_p E(s) + \frac{K_p}{T_i} \frac{1}{s} E(s) + K_p T_d s E(s) \\ \frac{U(s)}{E(s)} &= K_p + \frac{K_p}{T_i} \frac{1}{s} + K_p T_d s \end{aligned}$$

reemplazando; $\frac{U(s)}{E(s)} = G_c(s)$ $\frac{K_p}{T_i} = K_i$ $K_p T_d = K_d$

$$G_c(s) = K_p + K_i \frac{1}{s} + K_d s \quad (2)$$

$$G_c(s) = \frac{K_d s^2 + K_p s + K_i}{s} \quad (3)$$

transformada Z por método Tustin(bilineal), si $s = \frac{2}{T} \frac{z-1}{z+1}$ en ec. 2

$$G_c(z) = K_p + K_i \frac{T}{2} \frac{z+1}{z-1} + K_d \frac{2}{T} \frac{z-1}{z+1}$$

reemplazando $K_{pz} = K_p$ $K_{iz} = K_i \frac{T}{2}$ $K_{dz} = K_d \frac{2}{T}$ y despejando,

$$G_c(z) = \frac{(K_{pz} + K_{iz} + K_{dz}) z^2 + 2(K_{iz} - K_{dz}) z + (K_{pz} + K_{iz} + K_{dz})}{z^2 - 1} \quad (4)$$

2. Control PID: Discretización por partes

- Proporcional,

$$p(t) = K_p e(t)$$

transformada de Laplace y transformada Z y antitransformada en **Tiempo Discreto**,

$$\begin{aligned} P(s) &= K_p E(s) \\ P(z) &= K_p E(z) \\ &= K_{pz} E(z) \\ p(kT) &= K_{pz} e(kT) \end{aligned} \quad (5)$$

- Integral,

$$i(t) = \frac{K_p}{T_i} \int_0^t e(\tau) d\tau$$

transformada de Laplace y $K_i = \frac{K_p}{T_i}$

$$I(s) = K_i \frac{1}{s} E(s)$$

transformada Z por método **Tustin(Bilineal)**, si $s = \frac{2}{T} \frac{1-z^{-1}}{1+z^{-1}}$

$$I(z) = K_i \frac{T}{2} \frac{1+z^{-1}}{1-z^{-1}} E(z)$$

desarrollando,

$$\begin{aligned} I(z) (1-z^{-1}) &= K_i \frac{T}{2} (1+z^{-1}) E(z) \\ I(z) - I(z) z^{-1} &= K_i \frac{T}{2} [E(z) + E(z) z^{-1}] \end{aligned}$$

antitransformada en **Tiempo Discreto** y $K_{iz} = K_i \frac{T}{2}$

$$i(kT) = i(kT - T) + K_{iz} [e(kT) + e(kT - T)] \quad (6)$$

- Derivativo; aplicando el procedimiento anterior,

$$d(t) = K_p T_d \frac{de(t)}{dt}$$

transformada de Laplace y $K_d = K_p T_d$

$$D(s) = K_d E(s) s$$

transformada Z por método **Diferencia hacia atrás**, si $s = \frac{1 - z^{-1}}{T}$

$$D(z) = K_d \frac{1 - z^{-1}}{T} E(z)$$

despejando,

$$D(z) = \frac{K_d}{T} [E(z) - E(z) z^{-1}]$$

antitransformada en **Tiempo Discreto** y $K_{dz} = \frac{K_d}{T}$

$$d(kT) = K_{dz} [e(kT) - e(kT - T)] \quad (7)$$

sumando **Proporcional 5, Integral 6 y Derivativo 7**

$$u(kT) = p(kT) + i(kT) + d(kT) \quad (8)$$