Desarrollo de sistema de control para alimentación automática de muestras en equipos de análisis de metabolitos

Por
Blanca de la Torre Fuertes

UNIVERSIDAD COMPLUTENSE
MADRID

Grado en Ingeniería Informática
Facultad de Informática

Tutor académico: Juan Francisco Jiménez Castellanos
Tutor profesional: Dr. Guillermo Vidal de Miguel

Madrid, 2018–2019

Sobre TEFLON

Teflon(cc0 1.0(documentación) MIT(código))es una plantilla de LaTeX creada por David Pacios Izquierdo con fecha de Enero de 2018. Con atribuciones de uso CC0.

Esta plantilla fue desarrollada para facilitar la creación de documentación profesional para Trabajos de Fin de Grado o Trabajos de Fin de Máster. La versión usada es la 1.3.

V:1.3 Overleaf V2 with pdfLaTeX, margin 1in, NO-bib

Índice general

Capítulo 1

Resumen

El proyecto desarrollado en este Trabajo de Fin de Grado surge como un complemento a SUPER SESI, la instrumentación de Fossil Ion Tech que permite analizar muestras de respiración de pacientes en tiempo real y de manera on-line. Este sistema ayuda a identificar los biomarcadores en la respiración del paciente, eliminando riesgos de contaminación en el análisis.
Con el sistema de alimentación automática propuesto en este proyecto, se amplía el rango de acceso de pacientes al SUPER SESI, pudiendo ser analizado el aliento de una persona sin que ésta tenga que estar junto a la máquina. Esto es una importante ayuda en el caso de pacientes que, por razones variadas, no puedan desplazarse hasta el sistema de análisis SUPER SESI.
Para recoger correctamente la muestra, el paciente ha de exhalar en una bolsa de materia inerte con baja emisión de gases para evitar su contaminación. Esta bolsa se introduce en la máquina de alimentación automática o Desampler donde la muestra se mantiene a una temperatura constante. Una vez la bolsa ha quedado correctamente insertada en el Desampler, éste se lleva al laboratorio, donde la máquina se conecta al SUPER SESI y, mediante instrucciones recibidas en la interfaz del Desampler, la muestra se expulsa simulando una descarga en tiempo real del paciente.

Para llevar este proyecto a cabo se ha profundizado en el funcionamiento de la programación multithreading en python. Su uso ha permitido la modularización del comportamiento del programa en clases, siendo cada clase responsable del funcionamiento de una parte concreta del sistema: movimiento de placas de presión, cálculos matemáticos de velocidad, apertura y cierre de compuerta de control, interfaz de usuario y control de inicio y final de recorrido de las placas. Para el diseño de la interfaz de usuario se ha usado Tkinter, una biblioteca gráfica para python. Gracias a ello se ha podido conseguir una comunicación continuada entre la interfaz y las diferentes clases del programa, resultando en un programa fluido y la actualización de información en tiempo real.
Asegurando la integridad de las diferentes variables de control del sistema se ha obtenido un código que comparte información de manera fiable y fluida entre sus múltiples módulos; permitiendo una conexión paralela entre motores paso a paso, interfaz, servos de control de seguridad y cálculo de variaciones en la velocidad según la fuerza establecida por el usuario.
Todo ello se ha podido probar exitosamente en un banco de pruebas, simulando el comportamiento de la máquina tanto en un uso cotidiano de la misma como en situaciones límite que podrían desestabilizar dicho sistema.

Capítulo 2

Palabras Clave

2.1. Castellano

Bag Desampler, motores paso a paso, programación multi-hilo, interfaz gráfica, ley de control, control de velocidad, SUPER SESI, raspberry pi, python, arduino, oscilador armónico, señal sinusoidal, placas de presión.

2.2. Inglés

Bag Desampler, stepper motors, multithreading programming, graphic interface or GUI, control law, speed control, SUPER SESI, raspberry pi, python, arduino, harmonic oscillator, sinusoidal signal, pressure plates.

Capítulo 3

Introducción

3.1. Antecedentes y objetivos

3.1.1. Castellano

La empresa Fossil Ion Technology S.L. ha desarrollado una fuente SESI (Secondary Electro-Spray Ionization) llamada SUPER SESI que proporciona análisis en tiempo real de la composición química del aliento humano. Mediante una alta eficiencia en la ionización, permite el análisis optimizado de moléculas de baja volatilidad. Una de las mayores ventajas de este sistema frente al resto es la rapidez de uso: la muestra se introduce directamente en el SUPER SESI, sin necesidad de ser preparada y obteniendo resultados en un tiempo aproximado de 1 segundo.

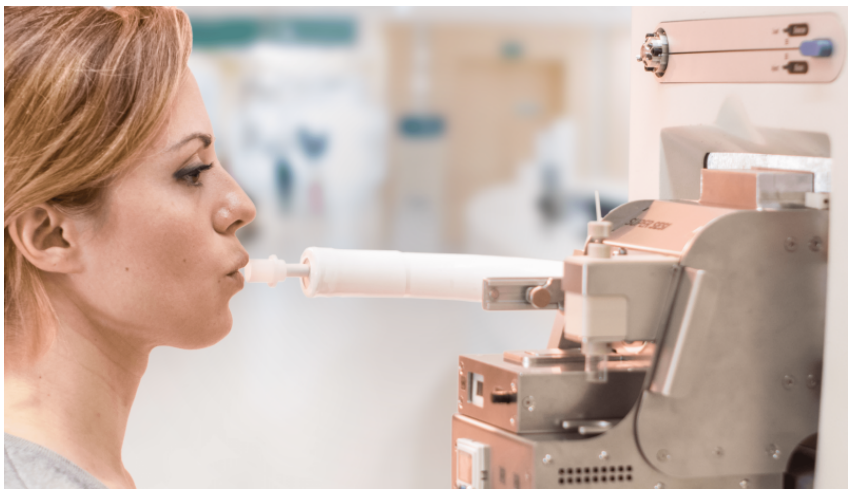

Figura 3.1: Ejemplo de uso de SUPER SESI.

Teniendo el SUPER SESI en un laboratorio al alcance tanto de los técnicos como del paciente, su uso aporta rapidez y fiabilidad a la hora de realizar un análisis inicial, ya que la recolección de datos se realiza de manera directa y on-line. Pero a su vez se genera un problema, puesto que la accesibilidad de ciertos pacientes se puede ver comprometida ya sea por factores externos o de la propia persona. Partiendo de este supuesto, surge la necesidad de un sistema que permita obtener una muestra fiable de un paciente y asegu-

rar su integridad hasta ser analizada en el SUPER SESI.

Tomando esa necesidad como base, se ha diseñado el Bag Desampler: una máquina que permite la alimentación automática del SUPER SESI, asegura el correcto mantenimiento y portabilidad de la muestra y ofrece la posibilidad de una manera segura y off-line de análisis de la exhalación del paciente. Primero se obtiene el aliento del paciente mediante una bolsa diseñada específicamente para ser incorporada al Bag Desampler. Está fabricada con materia inerte que asegura una mínima interacción química con la muestra tomada, y en los extremos tiene dos cierres de seguridad que se acoplan a la máquina, evitando filtraciones de ningún tipo.

Una vez el Bag Desampler ha sido correctamente encendido, la bolsa ha sido acoplada y la compuerta de seguridad cerrada; éste mantiene una temperatura constante que asegura que no haya pérdidas de humedad. La máquina se porta al laboratorio, donde se encaja al SUPER SESI.

Figura 3.2: Imagen del Bag Desampler.

Una vez correctamente colocado, el Bag Desampler permite múltiples acciones al usuario. Se ofrece la posibilidad de definir una fuerza de aplastamiento para la introducción de la muestra en el SUPER SESI y al seleccionar Close, que el aliento pase a su análisis simulando el uso que realizaría un paciente. El usuario puede en cualquier momento pulsar Stop para parar el movimiento de las placas de presión, y pulsar Open para abrir de nuevo las placas y evitar que se siga aplastando la bolsa.

También permite el control de cierre y apertura de las dos agarraderas que sujetan la bolsa por cada extremo. Éstas deberán estar cerradas para proceder a la inserción de la muestra en el SUPER SESI. A su vez, se muestra en tiempo real información de variables de análisis del movimiento de las placas para permitir un mayor conocimiento y control de uso al técnico encargado. Estas variables consisten en la velocidad, fuerza de cierre y posición de las placas; la posición de la compuerta de seguridad, y valores referentes a la temperatura del espacio de almacenamiento de la bolsa.

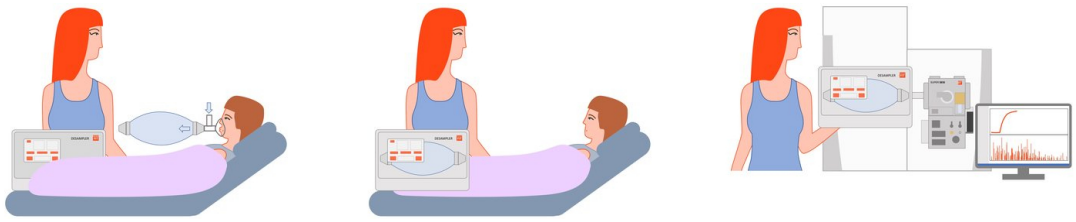

Figura 3.3: Explicación de uso del Bag Desampler.

Con este método se asegura una extracción segura de la muestra, eliminando posibles contaminantes y evitando dependencia de absorción desorción. Las muestras son conservadas y analizadas en las mismas condiciones; minimizando tanto la interacción química entre el gas obtenido y la bolsa como su manejo y tiempo de almacenamiento. Una vez el Bag Desampler se acopla con SUPER SESI, el control de cierre de las placas según la fuerza establecida por el usuario permite que la muestra llegue inalterada a la máquina para su posterior análisis, siendo los datos obtenidos totalmente compatibles con aquellos obtenidos mediante exhalación directa.

El diseño y desarrollo del Bag Desampler ha seguido los siguientes requisitos funcionales:

- Se precisa una máquina capaz de ejercer una fuerza constante mediante dos placas a una bolsa para la extracción de su contenido, específicamente aire. Dichas placas deben moverse mediante un sistema compuesto por dos motores paso a paso conectados a dos tornillos sin fin.
- La máquina debe tener una pantalla interactiva táctil desde donde el usuario controle la máquina y establezca valores que afecten al funcionamiento del sistema.
- El equipo debe tener sistemas de seguridad que aseguren la falta de riesgo para el personal que lo opere. El sistema debe funcionar solo si la tapa de seguridad ha sido correctamente cerrada.

3.1.2. Inglés

The company Fossil Ion Technology S.L. has developed a high sensitivity instrumentation built upon the Secondary Electro-Spray Ionization (SESI) called SUPER SESI. It provides real time analysis of the chemical composition of the human breath. Thanks to its high ionization efficiency it allows an optimized analysis of low volatility molecules in the gas phase.

One of the biggest advantage of this system versus the rest is the speed in its use: the sample is directly introduced in the SUPER SESI, without the need to be prepared and achieving results in an approximate time of 1 second.

Having the SUPER SESI in a laboratory within reach of both technicians and patients, its use brings quickness and reliability when performing an initial analysis, since data recollection is carried out directly and on-line. But at the same time an issue is generated, since the accessibility of certain patients can be compromised either by external factors or by the person himself. Based on this premise the necessity arises for a system that allows to obtain a reliable sample from a patient and ensures its integrity until its analysis in the SUPER SESI.

The Bag Desampler has been designed taking this requirement as a foundation: a machine that allows the automatic supply of the SUPER SESI, securing the correct maintenance and portability of the sample and offering the possibility of a safe and off-line way of the analysis of the patient's breath.
First, the patient's breath is obtained through a bag specifically designed to be incorporated into the Bag Desampler. It is made out of low-out-gassing inert material that assures a minimal chemical interaction with the taken sample, and in each end it has a safety fastener that docks with the machine, avoiding any type of filtration. When the Bag Desampler has been correctly turned on, the bad has been attached and the security cover closed; it maintains a constant temperature that ensured that there are no moisture losses. The Bad Desampler is brought to the laboratory, where it is fitted into the SUPER SESI.

Once correctly placed, the Bag Desampler allows multiple actions to the user. It offers the possibility of defining a squishing force for the introduction of the sample in the SUPER SESI and when selecting Close, the breath passes to its analysis simulating the use that a patient would make. The user can press Stop at any time to stop the movement of the pressure plates, and press Open to open the plates again and prevent the bag from continue being crushed.
It also allows the control of closing and opening of the two clamps that hold the bag at each end. These need to be closed to proceed to the insertion of the sample in the SUPER SESI.
At the same time, information on the variables of analysis of the plates movement is shown in real time to allow greater knowledge and use control to the technician in charge. These variables consist on the speed, closing force and position of the plates; the position of the safety cover, and the reference values for the temperature of the storage space of the bag.
This method ensures a safe extraction of the sample, eliminating possible contaminants and avoiding absorption desorption dependence. The samples are stored and analyzed under the same conditions; minimizing both the chemical interaction between the gas obtained and the bag and its handling and storage time.

Once the Bag Desampler is coupled with SUPER SESI, the squishing control of the plates according to the force established by the user allows the sample to arrive unaltered to the machine for later analysis, being the data obtained fully compatible with those obtained by direct exhalation.

The design and development of the Bag Desampler has pursued the following functional requirements:

- It is required a machine capable of pressing a constant force through two plates to a bag to extract its contents, specifically air. These plates must be moved by a system consisting of two stepper motors connected to two endless screws.
- The machine must have an interactive touch screen from where the user controls the machine and sets values that affect the behaviour of the system.
- The machine must have security systems that ensure the lack of risk for the personnel who operates it. The system should work only if the safety cover has been properly closed.

3.2. Alcance del proyecto

El presente proyecto tiene como objetivo el desarrollo del software asociado a la máquina Bag Desampler. Este software ha de encargarse de diferentes funcionalidades:

- Dada una fuerza como valor de consigna, debe ajustar la velocidad de dos motores paso a paso siguiendo la física establecida por la ley de control diseñada específicamente para este sistema.
- Debe responder adecuadamente a las órdenes dadas por un usuario a través de una interfaz gráfica, coordinando los diferentes módulos de manera fluida y acorde a las necesidades de dicha orden.
- Deber ser capaz de controlar la dirección y velocidad a la que se mueve cada uno de los motores paso a paso, respondiendo oportunamente tanto a los nuevos comandos recibidos como a situaciones límite dentro del funcionamiento interno del sistema.
- Habiendo sido establecidos unas trayectorias para el movimiento de los motores, debe detectar correctamente cuando se ha llegado al fin del recorrido y ser capaz de reaccionar a ello apropiadamente.
- Debe controlar correctamente el envío de las señales de cierre y apertura de cada una de las agarraderas, controladas por pares de servos.

Una vez diseñado y desarrollado el código capaz de llevar a cabo los puntos anteriormente descritos de manera fluida y sincronizada, se requiere la comprobación del correcto funcionamiento en un banco de pruebas capaz de simular fielmente tanto una utilización normal de la máquina como las situaciones extremas derivadas de un uso incorrecto o prolongado, casos altamente probables una vez el Bad Desampler sea distribuido.

3.3. Plan de trabajo

El desarrollo de este proyecto se ha basado en las siguientes fases:

- **Requisitos**: consistente en la definición y análisis de las especificaciones. A su vez, se diseña la arquitectura inicial del sistema y se eligen los componentes.
- **Desarrollo**: al inicio de esta fase se realiza el diseño hardware y software del sistema. Una vez establecidos los diseños, se procede a programar los submódulos del proyecto, realizando pruebas iniciales del funcionamiento individual de cada uno.
- **Integración**: engloba el proceso de unión de los submódulos y su consiguiente sincronización. Se configura y prueba el flujo general del sistema.
- **Puesta a punto**: se procede a las pruebas de verificación del funcionamiento del software en el banco de ensayos; lo que conlleva un proceso metódico de depuración.

Durante las fases de desarrollo e integración, tanto las pruebas como la depuración han seguido un ciclo de vida iterativo, asegurando así la agilidad y flexibilidad del proceso.

En la siguiente página se muestra el Diagrama de Gantt del proyecto, donde cabe destacar el camino crítico (remarcado en rojo).

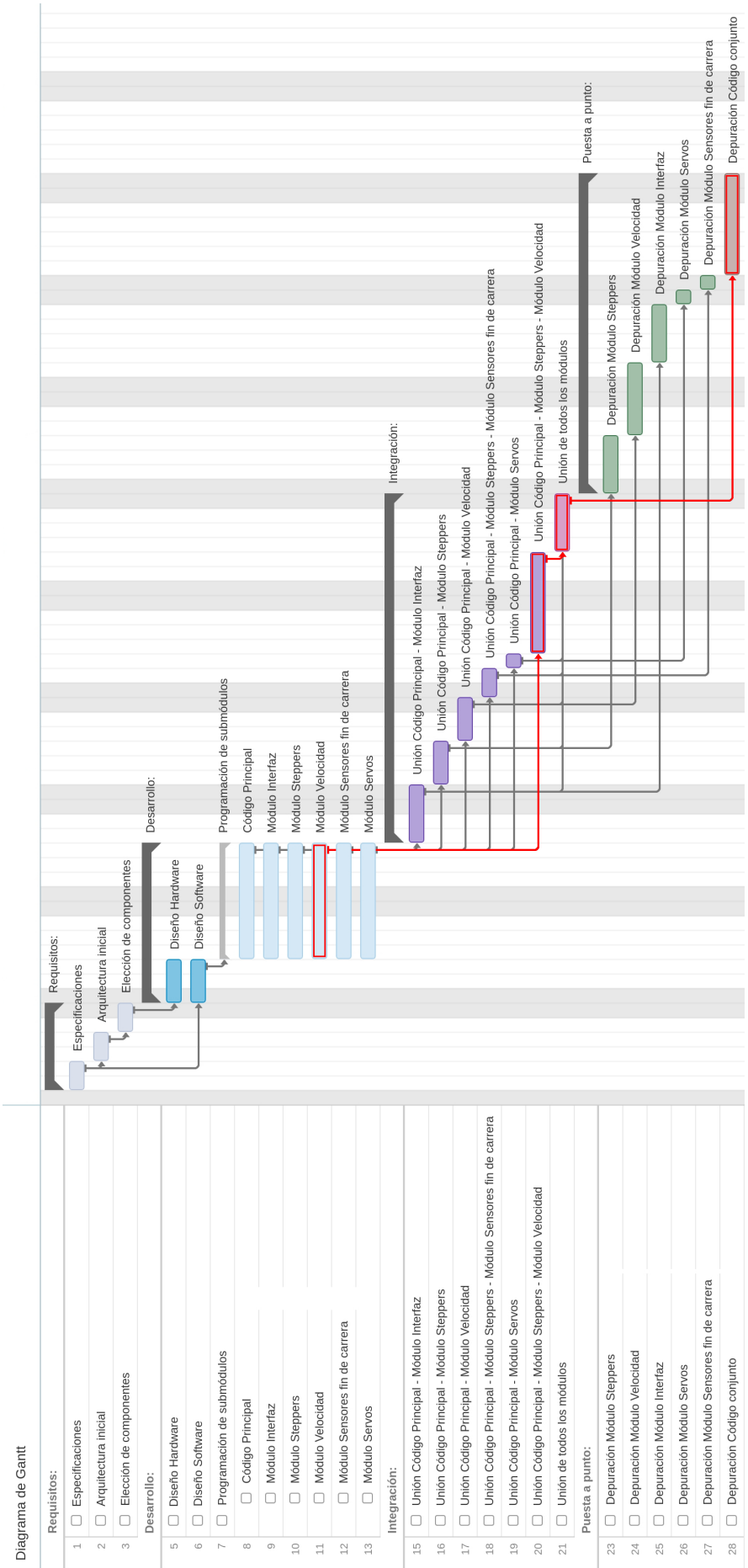
Diagrama de Gantt
Requisitos:
1 Especificaciones
2 Arquitectura inicial
3 Elección de componentes
Desarrollo:
5 Diseño Hardware
6 Diseño Software
7 Programación de submódulos
8 Código Principal
9 Módulo Interfaz
10 Módulo Steppers
11 Módulo Velocidad
12 Módulo Sensores fin de carrera
13 Módulo Servos
Integración:
15 Unión Código Principal - Módulo Interfaz
16 Unión Código Principal - Módulo Steppers
17 Unión Código Principal - Módulo Velocidad
18 Unión Código Principal - Módulo Steppers - Módulo Sensores fin de carrera
19 Unión Código Principal - Módulo Servos
20 Unión Código Principal - Módulo Steppers - Módulo Velocidad
21 Unión de todos los módulos
Puesta a punto:
23 Depuración Módulo Steppers
24 Depuración Módulo Velocidad
25 Depuración Módulo Interfaz
26 Depuración Módulo Servos
27 Depuración Módulo Sensores fin de carrera
28 Depuración Código conjunto
Requisitos:
Especificaciones
Arquitectura inicial
Elección de componentes
Desarrollo:
Diseño Hardware
Diseño Software
Programación de submódulos
Código Principal
Módulo Interfaz
Módulo Steppers
Módulo Velocidad
Módulo Sensores fin de carrera
Módulo Servos
Integración:
Unión Código Principal - Módulo Interfaz
Unión Código Principal - Módulo Steppers
Unión Código Principal - Módulo Velocidad
Unión Código Principal - Módulo Steppers - Módulo Sensores fin de carrera
Unión Código Principal - Módulo Servos
Unión Código Principal - Módulo Steppers - Módulo Velocidad
Unión de todos los módulos
Puesta a punto:
Depuración Módulo Steppers
Depuración Módulo Velocidad
Depuración Módulo Interfaz
Depuración Módulo Servos
Depuración Módulo Sensores fin de carrera
Depuración Código conjunto

Capítulo 4

Resultados

4.1. Modelado de física del sistema y Ley de Control

Con el modelado de la física se busca conocer la función de transferencia, que describiría como el sistema dado responde a los valores de entrada. Estos cálculos han sido proporcionados por la empresa Fossil Ion Tech S.L.

4.1.1. Ecuaciones de la física del sistema a controlar

Se supone un sistema comprensible bajo la acción de dos placas. Ambas placas avanzan con una velocidad v, que se toma como variable de control. Estas placas tienen una superficie B; y el sistema se conecta al reservorio de presión P_0. Suponemos condiciones ideales para el aire contenido en el sistema:

$$PV = mRT \tag{4.1}$$

Derivando dicha fórmula con respecto al tiempo, se obtiene la evolución del sistema en función del tiempo; donde se consideran condiciones isotermas $T = cte$:

$$\frac{dP}{dt}V + P\frac{dV}{dt} = \frac{dm}{dt}RT \tag{4.2}$$

Hacemos la estimación de la pérdida de masa en el sistema empleando la ec. de Bernoulli, siendo A el área de la válvula de salida del aire:

$$P - P_0 = \frac{1}{2}\rho v_f^2 \tag{4.3}$$

$$v_f = -\sqrt{2\frac{P - P_0}{\rho}} \tag{4.4}$$

Para poder eliminar la variable ρ se sustituye:

$$\rho = \frac{P}{RT} \rightarrow \frac{dm}{dt} = -A\rho v_f = -A\sqrt{\frac{2}{RT}}\sqrt{P(P - P_0)} \tag{4.5}$$

Con esto la ecuación para la evolución temporal del sistema 4.2 queda:

$$\frac{dP}{dt}V + P\frac{dV}{dt} = -A\sqrt{2RT}\sqrt{P(P-P_0)} \tag{4.6}$$

Una vez obtenida la ecuación, se procede a establecer una relación entre las variables de estado P, V con las variables de control F, v:

$$P - P_0 = \frac{F}{B} \rightarrow P = P_0 + \frac{F}{B} \tag{4.7}$$

$$BV = -\frac{dV}{dt} \rightarrow V = V_0 - B\int vdt \tag{4.8}$$

Sustituyendo en 4.6:

$$\frac{F}{B}(V_0 - B\int vdt) - BV(P_0 + \frac{F}{B}) = -A\sqrt{2RT}\sqrt{\frac{F}{B}(P_0 + \frac{F}{B})} \tag{4.9}$$

Para simplificar la notación se realizan los siguientes cambios:

$$f = \frac{F}{B} \qquad \omega = BV \qquad k = A\sqrt{2RT} \tag{4.10}$$

Con ello la ecuación 4.9 se puede representar como:

$$f(V_0 - \int \omega dt) - (P_0 + f)\omega = -K\sqrt{f(P_0 + f)} \tag{4.11}$$

4.1.2. Linearización de la ecuación de la dinámica del sistema

Se considera la siguiente aproximación lineal a las funciones f y ω:

$$f = f_0 + \delta f \qquad \omega = \omega_0 + \delta\omega \tag{4.12}$$

Si se linealiza a partir de esos valores la ecuación 4.11, se obtiene:

$$\delta\dot{f}V - (P_0 + f_0 + \delta f)(\omega_0 + \delta\omega) = -K\left[(P_0 + f_0 + \delta f)(f_0 + \delta f)\right]^{\frac{1}{2}} \tag{4.13}$$

$$\delta\dot{f}V - (P_0+f_0)\omega_0 - (P_0+f_0)\delta\omega - \delta f\omega_0 - \delta f\delta\omega = -K\left[(P_0 + f_0)f_0 + \delta f(2f_0 + P_0) + \delta f^2\right]^{\frac{1}{2}} \tag{4.14}$$

Empleando el desarrollo en serie de Taylor de la Raíz y eliminando términos de segundo orden ($\delta\dot{f} \equiv \dot{f}$):

$$\delta\dot{f}V - (P_0+f_0)\omega_0 - (P_0+f_0)\delta\omega - \delta f\omega_0 = -K\left[(P_0 + f_0)f_0\right]^{\frac{1}{2}} - \frac{k}{2}\frac{P_0 + 2f_0}{\left[(P_0 + f_0)f_0\right]^{1/2}}\delta f \tag{4.15}$$

Una vez igualando todos los incrementos de perturbación $\delta(\,\cdot\,) = 0$, obtenemos la solución estacionaria:

$$(P_0 + f_0)\omega_0 = -K\left[(P_0 + f_0)f_0\right]^{\frac{1}{2}} \rightarrow \left[(P_0 + f_0)f_0\right]^{\frac{1}{2}} = K\frac{f_0}{\omega_0} \tag{4.16}$$

Asumiendo que las variaciones $\delta(\cdot)$ de la ecuación 4.15 están relacionadas entre sí, se desarrolla una ecuación dinámica de la perturbación como una variación de 4.16:

$$\delta\dot{f}V - (P_0 + f_0)\delta\omega - \delta f\omega_0 = -\frac{k}{2}\frac{P_0 + 2f_0}{[(P_0 + f_0)f_0]^{1/2}}\delta f \tag{4.17}$$

$$\delta\dot{f}V - (P_0 + f_0)\delta\omega + \delta f(-\omega_0 + \omega_0\frac{P_0 + 2f_0}{2f_0}) = 0 \tag{4.18}$$

$$\delta\dot{f}V - (P_0 + f_0)\delta\omega + \delta f(\omega_0\frac{P_0}{2f_0}) = 0 \tag{4.19}$$

Aplicando la transformada de Laplace:

$$\lambda\delta fV + \delta f\frac{P_0}{2f_0}\omega_0 - (P_0 + f_0)\delta\omega = 0 \tag{4.20}$$

Si la velocidad no cambia, el sistema es estable:

$$\delta\omega = 0 \quad \rightarrow \quad \lambda = \frac{-\omega_0\frac{P_0}{2f_0}}{V} \tag{4.21}$$

4.1.3. Definición de la ley de control

La función de transferencia varía según varía el volumen de la bolsa. La solución de equilibrio cambia conforme se cambia la regulación de restricción de la salida:

$$\omega_0 = K(\frac{f_0}{P_0 + f_0})^{1/2} = P_0\delta\omega \tag{4.22}$$

Se reescribe de nuevo la ecuación del sistema, empleando 4.22 para eliminar ω_0 de la ecuación:

$$\delta\dot{f}V + \delta f\frac{P_0}{2f_0}K(\frac{f_0}{P_0 + f_0})^{1/2} = P_0\delta\omega \tag{4.23}$$

Se realizan varias propuestas de realimentación, buscando un oscilador armónico controlado. Teniendo en cuenta ambos estados, consideramos:

$$\delta\omega = -G_0\delta f - G_1\delta\dot{f} \tag{4.24}$$

En este caso, la ecuación del sistema quedaría:

$$\delta\dot{f}V + \delta f\frac{P_0}{2f_0}K(\frac{f_0}{P_0 + f_0})^{1/2} = P_0(-G_0\delta f - G_1\delta\dot{f}) \tag{4.25}$$

$$\delta\dot{f}(V + P_0G_1) + \delta f[\frac{P_0}{2f_0}K(\frac{f_0}{P_0 + f_0})^{1/2} + P_0G_0] = 0 \tag{4.26}$$

Si calculamos donde va el polo, haciendo transformada de Laplace:

$$\lambda = \frac{-[\frac{P_0}{2f_0}K(\frac{f_0}{P_0 + f_0})^{1/2} + P_0G_0]}{V + P_0G_1} \tag{4.27}$$

Al desconocerse ω_0, se sube un orden las ecuaciones del problema:

$$\delta\ddot{f}V + \frac{P_0}{2f_0}K(\frac{f_0}{P_0 + f_0})^{1/2}\delta\dot{f} = P_0\delta\dot{\omega} \tag{4.28}$$

Se propone una ley de control que supone la realimentación de tres estados, donde se controla la variación de velocidad en función de la fuerza.

$$\delta\dot{\omega} = -H_0\delta f - H_1\delta\dot{f} - H_2\delta\ddot{f} \tag{4.29}$$

4.2. Definición de arquitectura

4.2.1. Hardware

Con respecto a la arquitectura hardware de sensores y actuadores, el diseño ha sido proporcionado por la empresa Fossil Ion Tech S.L. al igual que el modelado de la física. Este diseño combina los módulos desarrollados para este sistema con un módulo de temperatura independiente al proyecto actual. Esta arquitectura sienta las bases de la física del sistema controlado y se compone de los siguientes elementos:

- Raspberry pi 3 Model B+.
- 2 Motor paso a paso, 1.8°, Bobinado Unipolar (M).
- 2 Controller Motores paso a paso A4988.
- 4 Servos.
- Controller Servos (ATMEGA328).
- 4 Sensores de Fuerza Resistivos (Force-Sensing Resistors, FSR).
- 2 Controller Sensores FSR (ATMEGA328).
- 4 Sensores de Fin de Carrera (Limit Switch, LS).
- Controller Temperatura (ATMEGA328).
- 3 Dispositivos de conversión de transmisiones serial a USB (FTDI).
- Clock.
- Interruptor Cover.
- Pantalla táctil.
- Módulo externo de Temperatura.

Estos elementos se conectan e interactúan según se detalla en el siguiente esquema, estableciendo conexiones del tipo:

- I2C (azul).
- Serial (rojo).
- HDMI (amarillo).
- USB (marrón).
- PWM (verde).
- Digital (morado).
- Corriente (gris).

En el esquema mostrado a continuación es muy fácil distinguir los diferentes módulos que componen este sistema. Como epicentro se encuentra la raspberry pi, que ejecuta el programa y maneja el flujo de información. A su derecha se encuentran los motores paso a paso, con sus correspondientes controladores que manejan las órdenes de movimiento recibidas de la raspberry. Unidos a los motores y a la raspberry se encuentra el interruptor de la tapa de seguridad, que actúa para asegurar que los motores se paran inmediatamente

en caso de emergencia. Para el funcionamiento de la interfaz gráfica se cuenta con una pantalla táctil. por su izquierda se encuentran los servos que controlan las agarraderas de la bolsa a la máquina y su respectivo controlador, acompañado de un conversor de señal serial a USB. Además, se cuenta con cuatro sensores de fin de carrera para el control de recorrido de las placas en el movimiento de aplastamiento. Otra parte vital son los cuatro sensores de fuerza, que detectan la fuerza ejercida por las placas a la bolsa en el movimiento de aplastamiento y que pasan su información a la raspberry por medio de controladores y conversores de transmisiones. Por último se encuentra el reloj, clave para las mediciones de tiempo de ejecución.

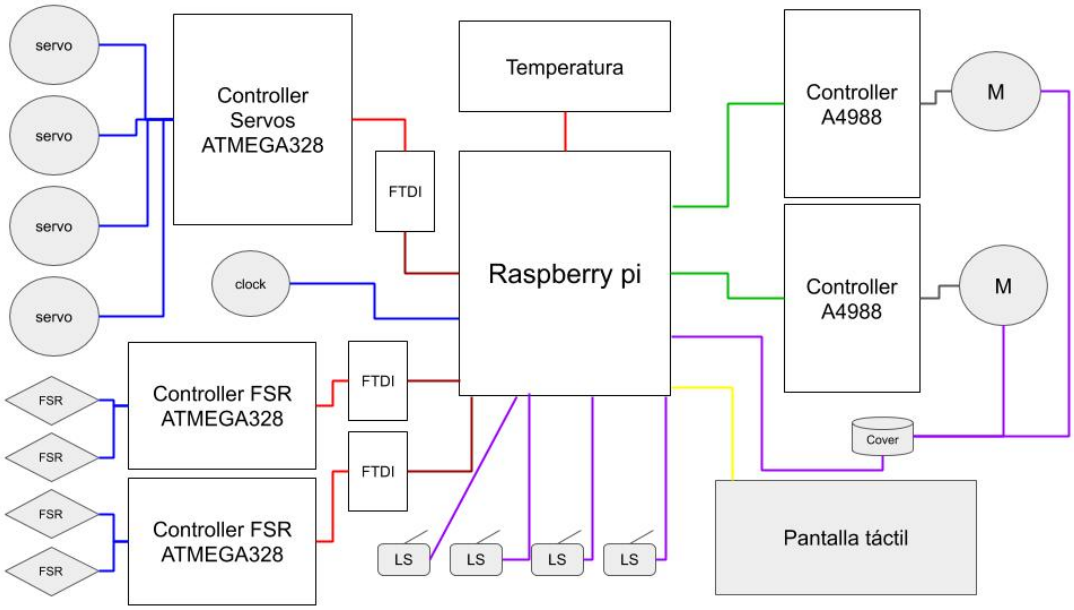

Figura 4.1: Esquema de Hardware

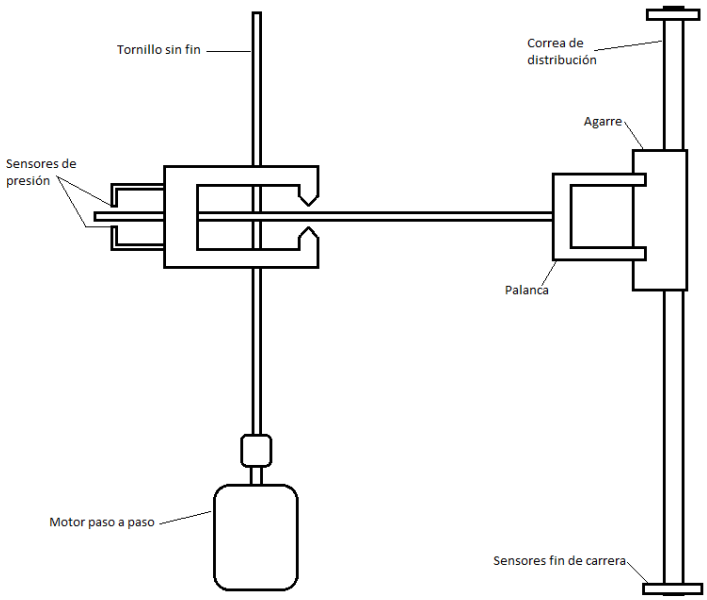

Figura 4.2: Colocación motores, tornillo sin fin y sensores.

Otra arquitectura hardware clave en este proyecto ha sido la del **banco de ensayos**, que ha sido clave tanto en el proceso de depuración del programa como en las pruebas finales de funcionamiento del software. Éste se compone de:

- Una raspberry pi. Tiene conectados tanto una pantalla como teclado y ratón.
- Un arduino.
- Un sensor de fin de carrera.
- Un controlador A4988 con un motor paso a paso conectado.

El programa principal se ejecuta en la raspberry pi. Gracias a la pantalla y el ratón se puede interactuar con la interfaz gráfica, y con el motor paso a paso conectado se puede probar toda la parte de control del movimiento. El arduino se emplea para generar señales externas de fuerza que simulen el comportamiento de las placas, y con el sensor de carrera se simula la llegada al fin de recorrido del motor o una parada brusca por una emergencia.

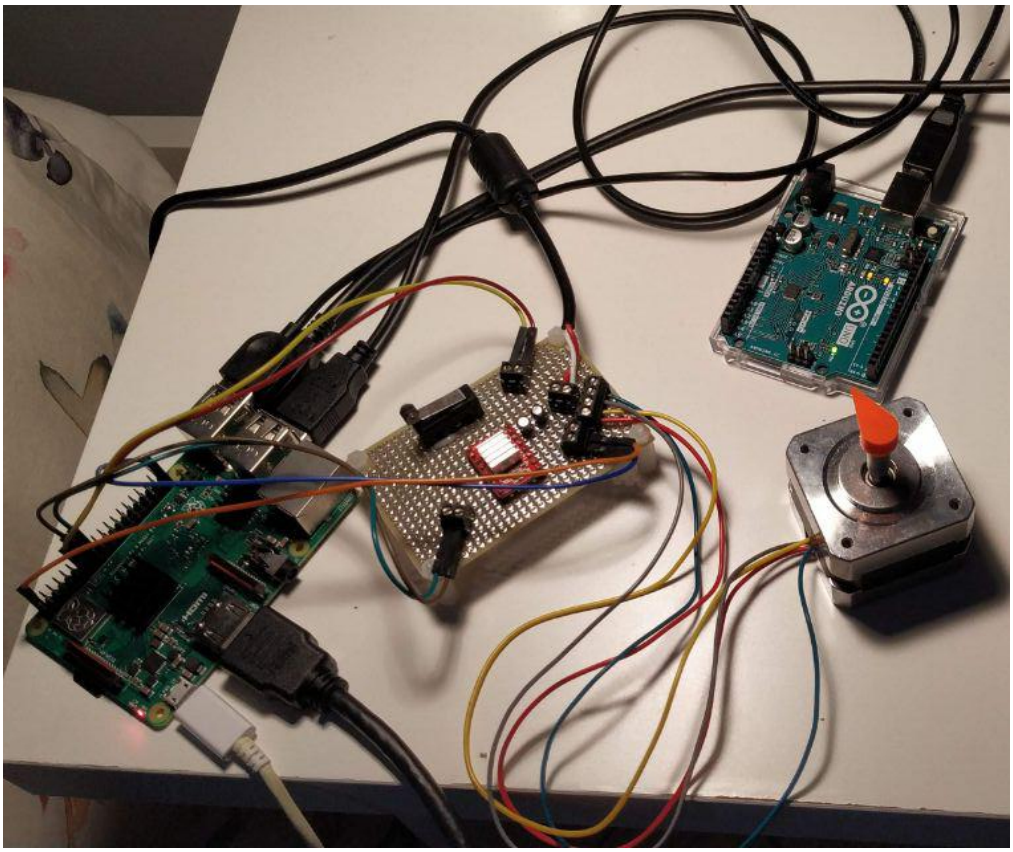

Figura 4.3: Banco de ensayos.

4.2.2. Software

La clave de la fluidez del funcionamiento del software del sistema es la **programación multi-hilo**. Gracias a este tipo de modularización de código, se ha podido aislar cada uno de los controles de los elementos independientes (motores, sensores, interfaz....).
A su vez, ha permitido mantener un flujo general controlado del sistema gracias al uso de variables globales, compartidas por diferentes módulos y con valores actualizados en tiempo real. Esto permite que la información compartida sea accesible a todos los elementos que la requieran sin poner en riesgo la veracidad de dichos valores.
Una vez los módulos han sido asignados a diferentes hilos por medio de la definición de clases, estos se ejecutan simultáneamente, evitando colisiones o ralentizaciones.

A continuación se proporciona un listado de las variables utilizadas en el código principal del programa y los valores que pueden tomar.

- Status: Start, Running, Exit.
- Direction: open, close.
- Stop: True, False.
- Block_Stepper[2]: True, False.
- WaitTime: numérico.
- Status stepper 1: open, close, opening, closing, stopped.
- Status stepper 2: open, close, opening, closing, stopped.
- Force Reading 1: numérico.
- Force Reading 2: numérico.
- Speed Reading 1: numérico.
- Speed Reading 2: numérico.
- Limit Switches Readings: True, False.
- Set Point Force: numérico.
- Cover Status: True, False.
- Open Desampler Flag: True, False.
- Close Desampler Flag: True, False.
- Open_Clamp1: True, False.
- Open_Clamp2: True, False.
- Errors: True, False.
- ShutDown_Signal: True, False.

El código principal del software del sistema se estructura en las siguientes partes o módulos, con sus clases y funciones correspondientes:

Setup

- Definición e inicializado de variables de control.
- Definición y configuración de pines de la Raspberry pi.
- Definición de funciones auxiliares.
- Inicio en cascada de los hilos de control y testeo del correcto funcionamiento y status de los elementos.

Clases

Stepper Control
Control del movimiento de un stepper. Indica al controller una dirección, da un paso y espera un tiempo determinado hasta el siguiente paso.

IN	OUT
Direction	Status stepper 1
Stop	Status stepper 2
WaitTime	Errors

Speed Control
Análisis de lecturas de fuerzas y cálculo del nuevo valor de velocidad correspondiente.

IN	OUT
Force Reading 1	Speed Readings
Force Reading 2	WaitTime
Set Point Force	Force readings
	Errors

Limit Switches Control
Control del cierre/apertura total del Bag Desampler mediante el análisis de lecturas de los sensores de fin de carrera.

IN	OUT
Stop	Speed Readings
Limit Switches Readings	Stop
	Status stepper 1 & 2
	Error

Interface Control
Control de la User Interface: muestra valores de control y permite definir un valor fuerza constante y un comando de movimiento.

IN	OUT
Force Readings	Stop
Speed Readings	Open/Close Desampler Flag
Status stepper 1 & 2	Open_Clamp1
Cover Status	Open_Clamp2
	Shut down
	Set Point Force
	Errors

Servos Control
Control de cierre y apertura de los agarres de la bolsa.

IN	OUT
Open_Clamp1, Open_Clamp2	Errors

Apagado

- Guardado de valores de los status de los diferentes elementos en fichero.
- Unión de hilos y fin de ejecución ordenado de las clases de control.
- Limpieza de configuración de pines de la Raspberry pi.

Una vez descritas las diferentes clases que toman parte del código del programa, se procede a analizar el flujo de información y funcionamiento del Bag Desampler y cómo dichas clases interactúan entre sí.

Se comienza con el encendido de la máquina pulsando el botón de encendido. Esto desencadena el comienzo de ejecución del código, que cambia el status global a START.
Una vez hecho esto, comienza a ejecutarse el Setup; que como he descrito anteriormente realiza el análisis de la correcta inicialización de funciones, variables y clases. Tras su ejecución se produce una comprobación de si todo ha ido correctamente.

- Si algo ha fallado, se muestra el fallo a la vez que se registra en el documento de log del sistema; y el programa se queda a la espera de que el usuario apague la máquina correctamente.
- Si todo ha ido bien, el sistema pasa a cambiar el status global a RUNNING.

El programa pasa a obtener el último estado que se conserva de los diferentes elementos. Esto se hace extrayendo la información guardada en el archivo last_state.txt. Si dentro la información está corrompida o incompleta, el código inicializa las variables de control a su valor por defecto. Con los últimos valores cargados, se procede al inicio de los hilos declarados en las clases correspondientes a los módulos. A su vez, el programa comienza la medición de tiempo de ejecución, clave a la hora de hacer cálculos posteriormente.

Ahora se describen los flujos correspondientes a cada hilo ejecutado:

- **Control de Sensores de fin de carrera** (Limit Switches Control): al comienzo del hilo, se obtiene el valor recibido de cada sensor. Si el valor de cada uno es False, es decir, que nada ha activado ese sensor; se vuelve a obtener el valor. Si se recibe un True de un sensor (ha sido activado), se procede a informar al motor correspondiente de la llegada al final del recorrido cambiando el valor de su variable Block_Stepper a True. Si ambos motores han recibido la información de parada por parte de sus sensores correspondientes, se cambia el valor de la variable de control Stop a True, informando al hilo encargado del movimiento de que debe parar. Si no, se sigue obteniendo el valor de los sensores.
- **Control de Motores paso a paso** (Stepper Control): al comienzo del hilo, comprueba si el valor de la variable Block_Stepper de ese motor tiene el valor True. Si es así, el motor no da un nuevo paso, quedando bloqueado y en espera de recibir un valor diferente en dicha variable. En cambio si su valor es False, el código comprueba el valor de la variable Stop. Si es True, el programa suelta el motor correspondiente. Si es False, se define la dirección del movimiento comprobando el valor de la variable Direction y se ordena al motor que dé un paso en esa dirección. Tras dar el paso, el programa espera un tiempo indicado en la variable WaitTime hasta volver a comprobar si debe bloquear el motor o no.

- **Control de velocidad** (Speed Control): obtiene el valor de las lecturas de los sensores de fuerza dados por su controlador. Con estos valores, se calcula el nuevo valor de la velocidad que deben tener los motores teniendo en cuenta la fuerza requerida por el usuario y accesible en la variable SetPoint Force. Una vez se tiene el nuevo valor de la velocidad, se calcula el tiempo entre pasos requerido para ajustar el movimiento a los nuevos parámetros. Esta información se actualiza a los hilos encargados de los motores gracias a la variable WaitTime.

- **Control de Servos** (Servos Control): se comprueba el valor de Open_ClampX, que informa de si la agarradera correspondiente debe estar abierta o cerrada. Tanto si su valor es True o False, se informa al controlador de servos y se guarda su valor.

- **Control de la Interfaz** (Interface Control): se inicializa el funcionamiento de la interfaz. Ésta tiene asignada a cada componente una función de actualización de valores, lo que permite poder mostrar los valores en tiempo real y mandar las órdenes marcadas por el usuario a los hilos correspondientes. La información que muestra se compone de:

  - Estado de los motores con la variable Steppers Status.
  - Estado de la tapa de seguridad con la variable Cover Status.
  - Valor de la fuerza requerida con SetPoint Force.
  - Valores de las lecturas de los sensores de fuerza.
  - Valor de la velocidad calculada de los motores.

  Los botones que ofrecen funcionalidades al usuario son:

  - Botón de apagado que pone a True la variable ShutDown_Signal.
  - Botones de apertura y cierre de cada una de las agarraderas que cambian el valor de las variables Open_ClampX.
  - Botones de Open, Close y Stop de las placas, que cambian el valor de las variables de control Stop y Direction.
  - Campo del nuevo valor requerido de fuerza, que cambia la variable SetPoint Force.

Recibida la señal de apagado con ShutDown_Signal, se cambia el status global a EXIT. Esto desencadena el guardado del último estado de los elementos en el fichero last_state.txt y la terminación de los hilos correspondientes a cada una de las clases. Una vez hecho esto, se ejecuta el apagado de la máquina. A continuación se muestra el diagrama de software, que representa el flujo anteriormente descrito. Para poder entender el sistema de símbolos y colores, se indica su significado:

- Caja verde: estados
- Caja naranja: acciones
- Caja morada: clases
- Caja verde claro: hardware
- Línea roja: compartir información
- Línea negra: ciclo máquina
- Rombo azul: comparaciones
- Línea verde claro: conexión con hardware

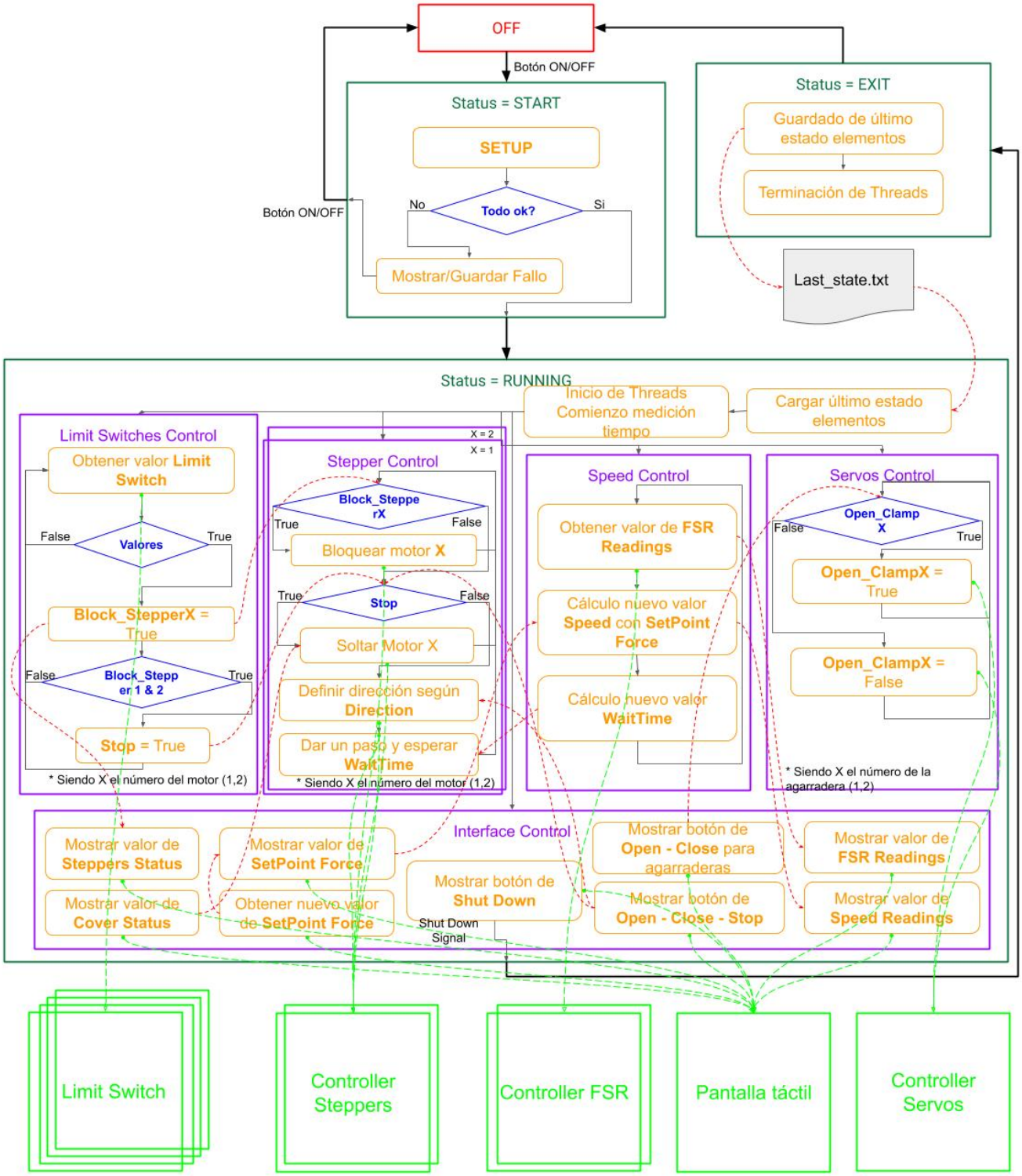

Figura 4.4: Diagrama de Software.

Para poder entender mejor el funcionamiento del movimiento de las placas de la máquina, se proporciona un diagrama de máquina de estados.
Los estados son:

- → **S0**: placas en reposo.
- → **S1**: placas cerrando.
- → **S2**: placas paradas por llegada al final (cerrado).
- → **S3**: placas abriendo.
- → **S4**: placas paradas en el cierre por error encontrado.
- → **S5**: placas paradas en la apertura por error encontrado.
- → **S6**: placas paradas por parada externa (llamada a Stop).

Las transiciones son:

- → **00**: señal de parar (mandada por sensores, ya sea fin de recorrido o atasco).
- → **01**: señal de cerrar.
- → **10**: señal de abrir.
- → **11**: señal de parar (llamada externa).

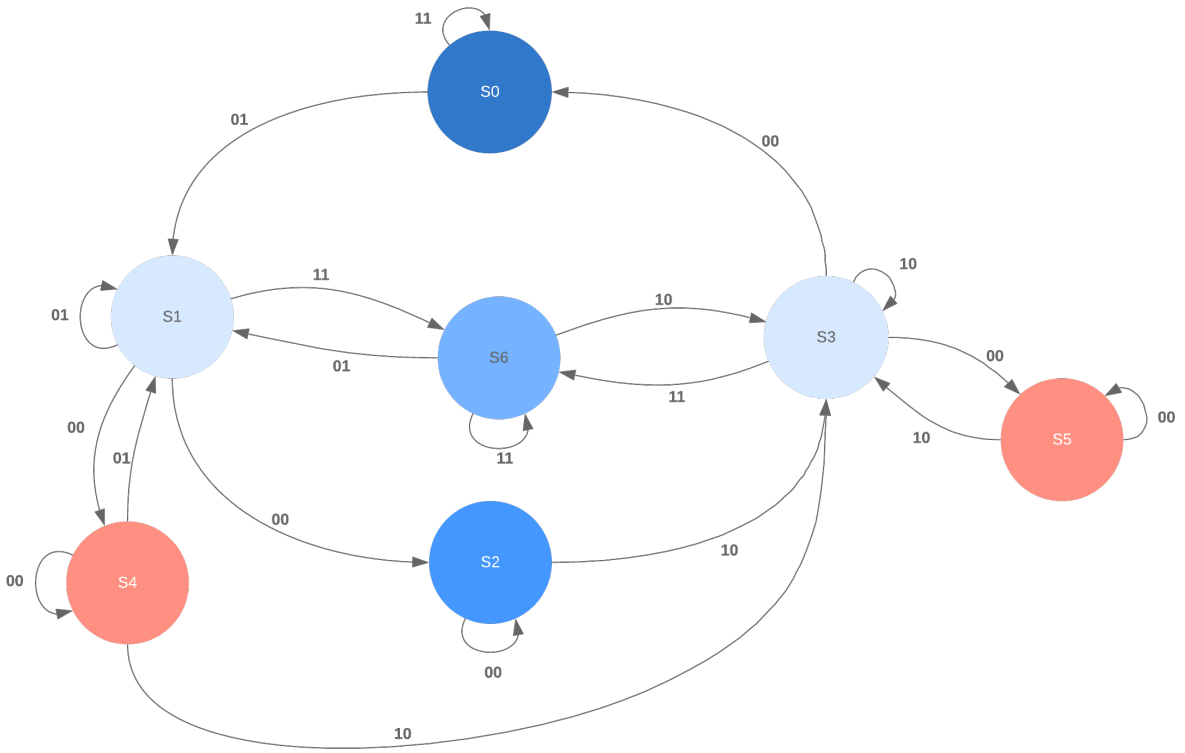

Figura 4.5: Diagrama de máquina de estados

Capítulo 5

Requisitos funcionales

5.1. Interfaz de Usuario

La Interfaz de usuario ha sido diseñada partiendo de un modelo preliminar ofrecido por la empresa. Se ha desarrollado usando el módulo Tk Interface, la interfaz estándar de Python para la interfaz gráfica Tcl/Tk.

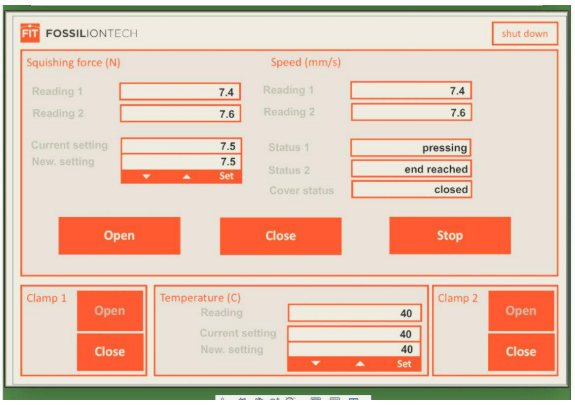

Figura 5.1: Diseño de Interfaz de Usuario aportado por la empresa.

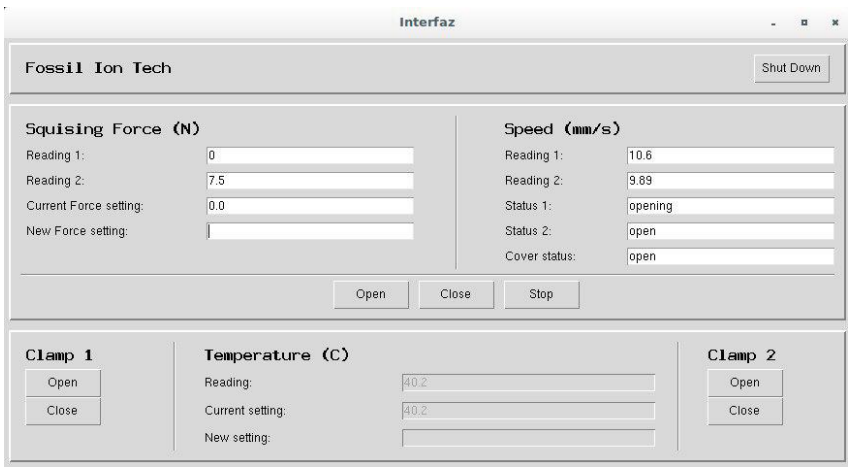

Figura 5.2: Interfaz de Usuario desarrollada en el proyecto.

La Interfaz Usuario de este proyecto es de tipo gráfica, presentando los elementos de control y medida en una pantalla táctil. El tipo de usuarios a los que está destinada es a personal cualificado del ámbito médico y que hayan recibido unas nociones básicas del funcionamiento tanto del SUPER SESI como del Bag Desampler y su correcta utilización. El usuario que utilizará la interfaz debe ser capaz de provocar el apagado y encendido de la máquina, visualizar a tiempo real los valores de control del sistema, modificar las variables de funcionamiento y controlar el movimiento de las placas.

A continuación se detalla el funcionamiento de los elementos principales de la interfaz:

a). El **encabezado de la interfaz** muestra en la parte izquierda el nombre de la empresa. En la parte derecha se encuentra el botón de "Shut Down", que permite apagar el sistema. Una vez se pulsa dicho botón, aparece una ventana emergente informativa donde se requiere que el usuario acceda al apagado de la máquina.

Figura 5.3: Ventana emergente de apagado.

b). El primer área central se divide en dos columnas y una barra inferior. La **primera columna** corresponde a la Fuerza de Aplastamiento de las placas (Squishing Force in Newtons (N)). Ésta muestra:

a) Lectura 1 (Reading 1): valor de la fuerza registrada en una de las placas.

b) Lectura 2 (Reading 2): valor de la fuerza registrada en la otra placa.

c) Fuerza establecida (Current Force setting): valor de la variable de fuerza tomada como referente para el movimiento de las placas.

d) Nueva Fuerza a establecer (New force setting): permite al usuario establecer un nuevo valor de fuerza para el cierre de placas, tanto escribiendo con el teclado como seleccionando con las flechas en el extremo de la caja.

La **segunda columna** corresponde a la Velocidad de las placas (Speed in mm/s). Ésta muestra:

a) Lectura 1 (Reading 1): valor de la velocidad registrada en uno de los motores.

b) Lectura 2 (Reading 2): valor de la velocidad registrada en el otro motor.

c) Estado de la primera placa (Status 1): informa del estado de la primera placa, ya sea parada (stopped), abriendose (opening), cerrándose (closing), abierta (open) o cerrada (close).

d) Estado de la segunda placa (Status 2): informa del estado de la segunda placa, ya sea parada (stopped), abriendose (opening), cerrándose (closing), abierta (open) o cerrada (close).

e) Estado de la tapa de cierre (Cover status): informa del estado de la tapa de seguridad, pudiendo ser abierta (open) o cerrada (close).

La **barra inferior** muestra tres botones en el centro, siendo sus funcionalidades:

a) Open: ejecuta la petición de apertura de placas. Una vez pulsado, muestra una ventana emergente que informa al usuario de la apertura de las placas.

Figura 5.4: Ventana emergente de apertura de placas.

b) Close: ejecuta la petición de cierre de placas. Una vez pulsado, muestra una ventana emergente que informa al usuario del cierre de las placas y de la fuerza de cierre tomada como referencia.

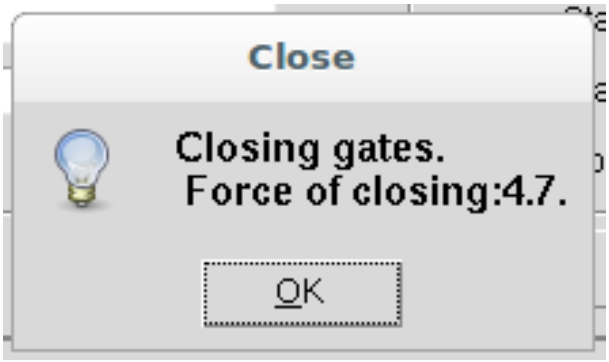

Figura 5.5: Ventana emergente de cierre de placas.

c) Stop: ejecuta la parada inmediata de las placas. Una vez pulsado, muestra una ventana emergente que informa al usuario de la parada de las placas.

c). El segundo área central se divide en tres columnas. La **primera columna** corresponde a los controles de la primera agarradera de la bolsa. Muestra dos botones con las siguientes funcionalidades:

a) Open: ordena la apertura de la agarradera. El servo correspondiente a esa agarradera se mantendrá así hasta que el botón de cierre sea pulsado. Una vez pulsado, se muestra una ventana emergente informando al usuario de la apertura de la agarradera.

Figura 5.6: Ventana emergente de parada de placas.

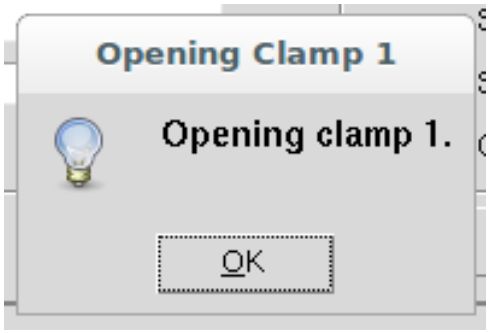

Figura 5.7: Ventana emergente de apertura de la primera agarradera.

b) Close: ordena el cierre de la agarradera. El servo correspondiente a esa agarradera se mantendrá así hasta que el botón de apertura sea pulsado. Una vez pulsado, se muestra una ventana emergente informando al usuario del cierre de la agarradera.

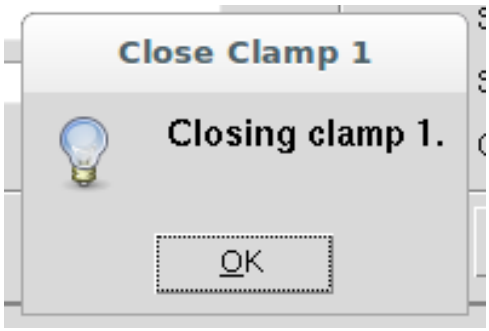

Figura 5.8: Ventana emergente de cierre de la primera agarradera.

La **segunda columna** corresponde al módulo externo de Temperatura (Temperature in C). Muestra tres campos que están opacos al no ser parte de la funcionalidad desarrollada en este proyecto.
La **tercera columna** corresponde a los controles de la segunda agarradera de la bolsa. Muestra dos botones con las mismas funcionalidades que los descritos para la primera columna de este área.

5.2. Respuesta del Hardware

Para cada una de las funcionalidades de la interfaz de usuario detalladas en el apartado anterior, se corresponde una respuesta del hardware correspondiente.
Con esa premisa se repasan ahora las diferentes opciones de funcionamiento del sistema.

a). **Apagado del sistema**: cuando el software recibe la orden de apagado, los motores paso a paso se quedan parados en la última posición registrada, al igual que los servos. Esto asegura el correcto mantenimiento de la sujeción de la bolsa. La pantalla táctil se apaga.

b). **Apertura de las placas**: cuando se selecciona la apertura de las placas, los motores paso a paso se mueven en la dirección indicada hasta que las placas hagan contacto con los sensores de fin de carrera. *

c). **Cierre de las placas**: cuando se selecciona el cierre de las placas, los motores paso a paso se mueven en la dirección indicada hasta que las placas hagan contacto con los sensores de fin de carrera. Mientras las placas están en movimiento, los sensores FSR mandan continuamente los valores registrados.*

d). **Parado de las placas**: cuando se selecciona la parada de las placas, los motores paso a paso se bloquean hasta que reciben orden de moverse de nuevo. Esto hace que las placas no se puedan mover en ninguna dirección. *

e). **Apertura de las agarraderas**: cuando se selecciona la apertura de una agarradera, el servo correspondiente se mueve en esa dirección. No cambiará hasta que la opción contraria sea seleccionada.

f). **Cierre de las agarraderas**: cuando se selecciona el cierre de una agarradera, el servo correspondiente se mueve en esa dirección. No cambiará hasta que la opción contraria sea seleccionada.

* Las placas podrán ejecutar este movimiento siempre y cuando la tapa de cierre de seguridad esté cerrada. En el momento en que el sensor detecte la apertura de dicha tapa, los motores se soltarán, parando el movimiento y pudiendo moverse las placas manualmente.

5.3. Herramientas de desarrollo

El lenguaje seleccionado para el código principal de este proyecto ha sido **Python**. Esto ha sido uno de los requerimientos iniciales establecidos por la empresa, ya que aporta flexibilidad y está en auge.
De entre las librerías más importantes usadas cabe destacar:

- threading: un módulo que construye interfaces multi-hilo. Su uso ha permitido la modularización en clases de los diferentes controles de hardware-software y su ejecución independiente compartiendo variables de control de flujo. Como referencia se ha consultado el libro [1].

- datetime: crucial en el correcto cálculo de la ley de control, pues ha permitido obtener registros de tiempo en milisegundos.

- RPi.GPIO: necesaria para la correcta configuración y comunicación con los pines de la raspberry pi.

- numpy: el paquete fundamental para el cómputo científico en Python. En este proyecto se usa su potente capacidad de trabajar con arrays de N-dimensiones para los cálculos matemáticos más complejos pertenecientes al módulo encargado de la ley de control. Para su uso se ha consultado el libro [2] y el artículo [3].

- logging: módulo necesario en la implementación de un sistema flexible de registro de eventos. Gracias a su amplia funcionalidad, se ha podido diseñar un sistema de registro adaptado a este proyecto, que cataloga los eventos por importancia y es clave en la posible detección de errores y testeo del sistema.

- tkinter: ya mencionado con anterioridad, ha permitido el desarrollo de una interfaz gráfica eficiente, sencilla y funcional.

Otro de los lenguajes utilizado ha sido **Arduino**. Con él se ha desarrollado un código esencial para el banco de pruebas que permite simular los valores de fuerza obtenidos por las placas en movimiento como una señal sinusoidal, y mandarlos correctamente a la raspberry para su posterior procesamiento y uso en el módulo de control de velocidad.
Además, se ha usado para los códigos de control de sensores. El código del controlador de servos permite mandar la orden de cierre o de apertura de las agarraderas, y el de FSR recibir los diferentes valores leídos por los sensores. Para poder manejar este lenguaje con más soltura se ha consultado el libro [4].

Capítulo 6

Control General

El proceso de desarrollo del código principal se ha llevado a cabo construyendo el esquema definido en el diagrama de software. Para ello se ha estructurado en tres partes principales:

6.1. Start

Comienza importando todas las librerías necesarias. Después se pasa a la parte definida como Setup, donde se configuran las conexiones serial a los elementos externos: conexión para el módulo externo de temperatura, conexión para el controlador de los servos y conexión para el controlador de los sensores de fuerza.
A su vez, se configuran los pines de la raspberry para conectarse correctamente a los controladores de los dos steppers, los cuatro sensores de fin de carrera y el interruptor de la tapa de seguridad.

Una vez hecho esto, se pasa a la definición de las variables de control del flujo general del programa (listadas en el apartado anterior de definición de arquitectura software), y a darle sus valores por defecto.

Posteriormente se encuentra la definición de dos funciones usadas en la funcionalidad de las clases: millis, que devuelve el tiempo transcurrido desde el inicio del programa; y str_to_bool, que devuelve un valor booleano para los valores string de True y False.

Luego se configura el sistema de registro de eventos o logging. Se establece que todos los eventos se muestren por pantalla y el formato que tendrán dichos mensajes en terminal. Además, se declara "app.log" como el formato de archivo donde se van a guardar los logs más importantes (nivel de Alerta, Error o Error Crítico), estableciendo un límite máximo de Bytes en 80000. Gracias a esta configuración, los eventos se irán guardando en archivos app.logX cuya X representa la numeración (según anterioridad de los eventos) con un máximo configurado de 50.

Tras esto, se pasa a la declaración de las diferentes clases. Todas ellas serán inicializadas posteriormente como hilos, así que comparten parte de la estructura inicial. Ésta se compone de: función ___init___ donde se añaden las variables globales a las que va a necesitar acceso esa clase y se crea un mensaje de inicio de hilo para ser guardado en el registro de logs; función terminate, que ejecuta la terminación del hilo instanciado; y función run, donde se desarrolla todo el código de la funcionalidad correspondiente a cada clase. Todas

las clases comparten este esquema menos la de la interfaz, que solo tiene init puesto que tanto su terminación como su funcionalidad se define y ejecuta dentro de dicha función.

Teniendo el esquema explicado en el párrafo anterior como base, se pasa a detallar el contenido de cada una de las clases:

- **App()**: corresponde a la interfaz y tiene varias partes. Primero de todo se indican las variables globales a compartir por la clase. Después se van creando los elementos que componen la interfaz que hemos visto en el apartado de requisitos funcionales, comenzando por un elemento raíz que representa a toda la ventana y añadiendo los diferentes marcos, textos y botones. A cada una de estas declaraciones se le detalla su elemento padre, es decir, dentro de qué parte está incluido; en algunos casos un texto asociado, el tamaño de los bordes y márgenes; y, en caso de botones o recuadros de texto, la función o variable que tiene asignada.
  Una vez hecho esto, se detalla la configuración de cada uno de los elementos ya declarados: su posición por columna y fila, su separación por eje x e y, y la referencia que toma al cambiarse el tamaño de pantalla (North, South, East, West). A continuación se realiza la asignación de pesos a los elementos, lo que hace que se coloquen en el orden deseado.
  Por último, se definen las funciones asignadas a los diferentes botones y actualización de los valores mostrados (se refrescan cada segundo): actualización del valor de la fuerza requerida, de las lecturas de fuerza de cada par de sensores, de los valores de velocidad de los motores, las posiciones de las placas, el estado de la tapa de seguridad; apertura, cierre y parada de las placas, apertura y cierre de cada una de las agarraderas y apagado del sistema.

- **SteppersControl()**: controla el funcionamiento de cada uno de los motores paso a paso. Para poder entender mejor el funcionamiento de estos motores se ha analizado el contenido del libro [5]. Al comienzo indica las variables globales a compartir. Después se comprueba si la tapa de seguridad está abierta. Si es así, se genera un mensaje informativo en el registro de eventos y se suelta el motor informando al controlador correspondiente. Si no, se pasa a comprobar si el motor debería estar bloqueado o no. Si debería estarlo, se manda un mensaje informativo y no se hace nada, pasando a una nueva ejecución del hilo.
  Si no debería estar bloqueado, se comprueba si se debería parar el movimiento gracias a la variable global stop. Si la variable indica que debería pararse, se suelta el motor y se comprueba si ha llegado al final del recorrido. Si es así, se cambia la información del estado del motor a cerrado o abierto dependiendo de la dirección que llevara en el movimiento. Si no, se indica su estado como parado.
  Por otro lado si el motor debe moverse, se manda al controlador que dé un paso en la dirección correspondiente y se le indica esperar durante un tiempo marcado por WaitTime y que controla la velocidad a la que se realiza el movimiento del motor.

- **SpeedControl()**: realiza los cálculos matemáticos necesarios para el ajuste de la velocidad de los motores según la fuerza registrada por los sensores. Primero se inicializan todas las variables y arrays usados en el cálculo, y se llama a las variables globales necesarias. La primera comprobación necesaria es si las placas están en movimiento y si ese movimiento es de cierre (aplastamiento de la bolsa). Si no es así, no se necesita realizar ningún cálculo y los valores de lectura se ponen

a 0. En cambio, si están moviéndose, se obtiene el valor medio de los valores de fuerza leídos por los sensores, y con esa información y las lecturas de tiempo del momento de ejecución se calcula la variación de velocidad necesaria para alcanzar la fuerza de presión requerida en las placas.
Estos cálculos se analizarán con mayor profundidad en el siguiente apartado, donde se explica el proceso para llegar a una ley de control robusta y su traducción en código ejecutable (realizado en el hilo que instancia esta clase). Una vez se ha calculado el tiempo de espera entre pasos para los motores paso a paso, se actualiza la variable global WaitTime y se espera un tiempo de 25 milisegundos hasta la nueva ejecución del hilo. Esto permite obtener una nueva medición de los sensores con un valor que refleje correctamente el ajuste proporcionado por dicho cambio en los parámetros calculados.

- **LimitSwitchesControl()**: controla la recepción de las señales detectadas por los sensores de fin de carrera y el análisis de su significado. Tras incluir las variables globales necesarias, lo primero es obtener los valores de los sensores. Después se asegura que la señal recibida es coherente y no ruido. Para ello, una vez recibido un True de un sensor, se establece un contador y si se recibe el mismo valor diez veces, se toma esa señal como válida, mandado un mensaje de bloqueo al motor correspondiente.
  Una vez hecho esto, si se ha recibido la misma señal de los dos sensores asociados a un mismo motor, eso significa que éste ha llegado a su fin de recorrido y las placas han terminado de abrirse o cerrarse; por lo que se actualizan las variables globales de Stop y End para avisar al resto de hilos. En cambio, si se han recibido señales erróneas o contradictorias, se manda una señal de parada a los motores y se crea una excepción en el sistema de registro de eventos.

- **ServosControl()**: controla la apertura y cierre de las agarraderas de la máquina por medio de señales mandadas al controlador de los servos. Añade las variables globales necesarias y comprueba si sus valores son True o False. En cada caso, manda la orden necesaria al controlador. Si lee un True, es que esa agarradera debe abrirse, y cerrarse si recibe un False.

Una vez se han declarado las clases, se comprueba la correcta conexión serial con los componentes externos: módulo de temperatura, controlador de servos y controlador de los sensores de fin de carrera. Si todos devuelven un ok, se continúa con la ejecución del programa. Si no, se activa el flag de error, se guarda un mensaje de log y se salta directamente a la finalización del programa (en la parte de Exit).

6.2. Running

El programa continúa con la obtención de los últimos valores de control guardados en el archivo last_state.txt. Estos valores corresponden al último estado de las agarraderas, la última fuerza requerida y la posición de las placas. Para la obtención de cada uno de los valores se analiza una línea del archivo, comprobando que el valor leído sea correcto; y si no se asigna el valor por defecto. A su vez, si ha habido problemas accediendo al archivo, ya sea por corrupción de los datos o su inexistencia, se asignan los valores por defecto y se guarda un evento en el sistema de logging.
Una vez actualizadas las variables globales de control, se procede a la instanciación de los

diferentes hilos asignados a las clases. Para ello se crean dos hilos de la clase Steppers-Control (uno por cada motor), uno de SpeedControl, uno de LimitSwitchesControl y uno de ServosControl. Después se lanza la ejecución de dichos hilos, seguida del comienzo de medición del tiempo y la creación de la interfaz. Cuando ésta comienza a ejecutarse, solo se pasa al resto del código cuando la interfaz ha sido cerrada (orden de apagado).

6.3. Exit

Cuando el status global cambia a Exit, es decir, se ha dado la orden de apagado del sistema, lo primero es guardar los estados de las variables de control en el fichero last_state.txt. Si se produce algún fallo en dicho proceso, se generará un mensaje de error que será registrado para su posterior análisis.
Tras el cierre del fichero, se procede a la terminación de los hilos. Una vez hecho esto, se cierra el sistema de logging y se limpia la configuración de pines de la raspberry.

6.4. Identificación y resolución de fallos

Para realizar las pruebas primero se ejecutó el código en un pc para mayor rapidez y eficacia en la detección de fallos.

El proceso comenzó por probar los módulos por separado, comprobando el correcto funcionamiento con generación de archivos auxiliares con registros de ejecución y muestra por pantalla del proceso de depuración.

Una vez conseguida esa parte, se pasó a la unión de módulos. Esta fue la parte más complicada y lenta del proceso de testing del control general, ya que se encontraron bastantes dificultades a la hora de conseguir que los valores se actualizaran y compartieran correctamente y en tiempo real. Para ello se fueron juntando módulos dos a dos, siempre siendo uno de ellos la interfaz, capaz de ayudar con la muestra de valores y mandar órdenes de prueba a los hilos en ejecución. Conseguido esto, se fueron añadiendo módulos hasta poder comprobar que entre todos se mantenía el flujo apropiado de información y respuesta a las órdenes que llegaban de terminal.

El siguiente paso fue probar esto en la raspberry. Al usar una raspberry con su propio Sistema Operativo, teclado, pantalla y ratón, el proceso fue bastante rápido, pues solo fue adaptar ciertos términos. Cuando esto se consiguió, se procedió a profundizar en el ajuste de la ley de control, que se analiza a continuación.

Capítulo 7

Ley de control robusta y Control de Motores

Partiendo de la base matemática definida anteriormente para la ley de control, se procede al cálculo de una función continua que aproxime los valores de fuerza obtenidos a unos valores de velocidad finales por el método de mínimo error cuadrático (profundizado gracias al libro [6]). Esta técnica busca minimizar la suma de cuadrados de las diferencias en las ordenadas (llamadas residuos) entre los puntos generados por la función elegida y los correspondientes valores en los datos. Estos datos en nuestro caso son pares de valores representando la fuerza leída por los sensores FSR y el tiempo en milisegundos en el que se tomó la muestra.

El método matemático usado enuncia que dado un conjunto n de pares de valores del tipo:

$$\{(t_k, f_k)\}_{m=1}^{n} \tag{7.1}$$

Siendo t la variable independiente que representa el tiempo y f la variable dependiente representando la fuerza, tenemos un conjunto de m funciones linealmente independientes, que se llamarán funciones base, y son del tipo:

$$\{(f_j(t_k))\}_{j=1}^{m} \tag{7.2}$$

Se desea encontrar una función $\phi(t_k)$ combinación lineal de las funciones base y sea la mejor aproximación a los n pares de valores descritos en 7.1 empleando, como criterio de "mejor", el criterio del mínimo error cuadrático medio de la función $\phi(t_k)$ con respecto a dichos pares. La forma de dicha función será:

$$\phi(t_k) = a_0 + a_1 t_k + a_2 t_k^2 \tag{7.3}$$

Para ello, se define un valor de fuerza que es la media de los dos valores leídos por los sensores FSR. A este valor se le resta el valor requerido de fuerza SetPoint Force, con lo que se obtiene el valor del error de la fuerza.
A su vez, se obtiene el valor del tiempo registrando el tiempo actual con la función millis() y restándole el tiempo que había transcurrido hasta el inicio de la ejecución del hilo. Con esto, tenemos calculado nuestro par de valores necesario para la ecuación. El desarrollo de la función f y sus derivadas son:

$$f = a_0 + a_1 t_k + a_2 t_k^2 \quad \rightarrow \quad f' = a_1 + 2a_2 t_k \quad \rightarrow \quad f' = 2a_2 \tag{7.4}$$

La aproximación por mínimos cuadráticos se basa en minimizar el error individual de cada par sobre el conjunto total cuadrático, cuya función según el número de muestras n es:

$$E_c = \sum_{k=1}^{n} (f_k - \phi(t_k))^2 = \sum_{k=1}^{n} (f_k - a_0 - a_1 t_k - a_2 t_k^2)^2 \tag{7.5}$$

Para poder obtener $a0, a1, a2$ que minimizan a E_c, se deriva e iguala a cero dicha ecuación para cada variable:

$$\frac{dE}{da_0} = \sum 2(f_k - a_0 - a_1 t_k - a_2 t_k^2)(-1) = 0 \tag{7.6}$$

$$\frac{dE}{da_1} = \sum 2(f_k - a_0 - a_1 t_k - a_2 t_k^2)(-t_k) = 0 \tag{7.7}$$

$$\frac{dE}{da_2} = \sum 2(f_k - a_0 - a_1 t_k - a_2 t_k^2)(-t_k^2) = 0 \tag{7.8}$$

Con ello se obtiene un sistema de m ecuaciones con m incógnitas, que recibe el nombre de "Ecuaciones Normales de Gauss".

$$\sum f_k - \sum a_0 - \sum a_1 t_k - \sum a_2 t_k^2 = 0 \tag{7.9}$$

$$\sum f_k t_k - \sum a_0 t_k - \sum a_1 t_k^2 - \sum a_2 t_k^3 = 0 \tag{7.10}$$

$$\sum f_k t_k^2 - \sum a_0 t_k^2 - \sum a_1 t_k^3 - \sum a_2 t_k^4 = 0 \tag{7.11}$$

Operando con ellas y desarrollando la suma, se obtiene la ecuación i-ésima del sistema de m ecuaciones normales que se expresa en forma matricial como:

$$T = \begin{bmatrix} \sum 1 & \sum t_k & \sum t_k^2 \\ \sum t_k & \sum t_k^2 & \sum t_k^3 \\ \sum t_k^2 & \sum t_k^3 & \sum t_k^4 \end{bmatrix} \quad A = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \quad B = \begin{bmatrix} \sum f_k \\ \sum f_k t_k \\ \sum f_k t_k^2 \end{bmatrix} \tag{7.12}$$

$$T \cdot A = B \quad \rightarrow \quad T^t \cdot T \cdot A = T^t \cdot B \quad \rightarrow \quad A = T^t \cdot B \tag{7.13}$$

Una vez llegados a este punto, se requiere la obtención de los sumatorios a sustituir en las matrices 7.12 con los valores de tiempo y fuerza calculados en la parte anterior de código. Para poder tener en cuenta los últimos valores obtenidos, se va guardando en cada ejecución el valor calculado de tiempo y fuerza en dos arrays, uno para cada variable correspondiente. Estos arrays son de tamaño 10, y éste se mantiene en todas las ejecuciones siguiendo un método FIFO, es decir, eliminando el valor más antiguo para añadir el nuevo. Hecho esto se generan los arrays gracias a la librería numpy y se les añaden los valores calculados. Con ello se calcula la ecuación para obtener los valores de $A : a0, a1, a2$.

Una vez obtenidos, se sustituyen en la función f y sus derivadas 7.4. Estos valores se usan en la función de la ley de control especificada en el apartado anterior Modelado de la física del sistema y Ley de control 4.29 y con ello se obtiene un nuevo valor de la aceleración del sistema.

Teniendo el valor de la aceleración, se pasa a la obtención de la nueva velocidad a establecer y con ello el nuevo valor de tiempo que deben esperar los motores entre paso y

paso (WaitTime).

Primero de todo se obtiene el tiempo transcurrido entre este nuevo cálculo y el de la anterior ejecución del hilo (lo que permite saber el tiempo que ha pasado desde que se le dió el anterior valor de velocidad). Si todos los valores de tiempo a tener en cuenta, que están acumulados en su array correspondiente, son válidos, se calcula la nueva velocidad:

$$\text{nueva velocidad} = \text{antigua velocidad} + (\text{aceleración} \cdot \text{tiempo transcurrido}) \tag{7.14}$$

Con el signo del nuevo valor de velocidad se establece la dirección del movimiento de los motores, y si ese valor es válido (entra dentro de los límites establecidos) se muestra por pantalla. Obtenido el nuevo valor de la velocidad, se pasa al cálculo del tiempo entre pasos. Con el nuevo valor de velocidad se obtiene el tiempo ideal de espera que deberían cumplir los motores paso a paso:

$$\text{tiempo ideal de espera} = \frac{1}{|\text{nueva velocidad}| + 10^{-8}} \tag{7.15}$$

Este valor representa que a mayor velocidad, menor es el tiempo de espera (inverso). Una vez obtenido, no se puede asignar sin más, pues se ha de tener también en cuenta el tiempo que puedan llevar esperando los motores. Por eso se definen dos nuevas variables: tiempo real de espera, que será el valor final que se pasará a los motores, y tiempo acumulado de espera, que representa el tiempo que llevan los motores esperando desde que dieron un último paso. También se establece un tiempo máximo de espera para los motores, que evita que se obtengan tiempos elevados en zonas por las que la velocidad pasa a 0 y que pueden hacer que el motor se quede esperando indefinidamente.

Teniendo en cuenta el valor obtenido de tiempo ideal de espera, se calcula el tiempo real de espera hasta un nuevo paso:

$$\text{Tiempo real de espera} = \text{Tiempo ideal de espera} - \text{tiempo acumulado de espera} \tag{7.16}$$

Si el valor de tiempo real de espera es demasiado elevado (mayor que el tiempo máximo de espera), se da el valor del tiempo máximo al tiempo ideal de espera, se añade al tiempo acumulado de espera 25 milisegundos (tiempo de ejecución del hilo medido, es decir, el tiempo que transcurre hasta un nuevo cálculo) y se manda la señal de bloqueo a los motores. Si el valor de tiempo real de espera es válido, se manda la orden de movimiento a los motores poniendo la variable Block de cada uno a False.
Si el tiempo real de espera es menor o igual a 0, se pone el valor de tiempo ideal de espera y de tiempo acumulado a 0; lo que hará que el motor no espere y de un paso más.
Si el tiempo real de espera es mayor que 0, se tiene en cuenta si hay tiempo acumulado de espera, es decir, si el motor ya lleva tiempo esperando para dar un nuevo paso. Si es así, si establece el tiempo de espera al máximo posible en movimiento (menos que el tiempo máximo de espera) y se actualiza el valor del tiempo acumulado. Para ello se realiza la siguiente comparación:

$$\text{tiempo acumulado de espera} < (\text{tiempo real de espera} - \text{tiempo máximo de movimiento}) \tag{7.17}$$

Si el tiempo acumulado cumple dicha comparación 7.17, se iguala su valor a 0. Si no, será:

$$\text{tiempo acumulado de espera} = \text{tiempo real de espera} - \text{tiempo máximo de movimiento} \tag{7.18}$$

Si el tiempo acumulado de espera es menor o igual a 0, es decir, el motor no ha estado esperando para el siguiente paso, el tiempo ideal de espera es igual al tiempo real de espera y el tiempo acumulado se iguala a 0. Con ello se pasa el valor de tiempo ideal de espera a los motores con la variable WaitTime y se guarda el nuevo valor de la velocidad para su uso en la fórmula de la siguiente ejecución del hilo.

7.1. Identificación y resolución de fallos

Dentro de la parte de debugging de la ley de control y funcionamiento de los motores se pueden identificar diferentes fases para alcanzar un correcto funcionamiento del programa.

Fase 1: Dada una fuerza sinusoidal, ¿se obtiene una velocidad sinusoidal?

Primero, se procede al análisis y obtención de los valores de fuerza. Estos valores, de acuerdo con las diferentes fases de prueba, son obtenidos inicialmente con una fórmula matemática calculada directamente por el hilo encargado de los cálculos matemáticos de la velocidad, por lo que el valor de tiempo se obtiene tomando como referencia el inicio del tiempo de ejecución de dicho hilo.
Para poder estudiar mejor el comportamiento del algoritmo a diseñar, se define como input una fuerza de comportamiento sinusoidal del tipo indicado,siendo A la amplitud y F la frecuencia de la onda:

$$f_k = A \cdot \sin(2\pi \cdot F \cdot t_k) \tag{7.19}$$

Una vez diseñada la fórmula, se eligieron los valores de amplitud y frecuencia de manera empírica. Para poder analizar y ver correctamente el comportamiento del algoritmo y visualizar los datos se usó la librería matplotlib.pyplot y se programó un script para la representación gráfica de los datos obtenidos. Gracias a ello podemos observar el proceso de ajuste de la velocidad para responder correctamente a un input de la fuerza sinusoidal. Para ello se fueron modificando valores de amplitud:

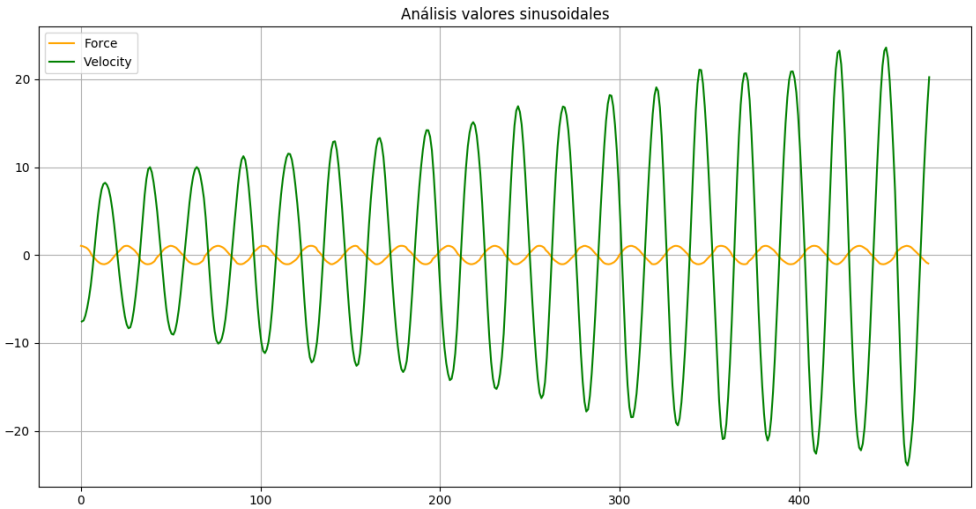

Figura 7.1: Análisis de valores sinusoidales con amplitud 1,05.

Como se puede observar en uno de los múltiples análisis, el mayor problema observado fue el comportamiento exponencial de la velocidad. En un proceso de depuración más profundo, se pasó a analizar por separado cada uno de los elementos de la función f y sus derivadas 7.4:

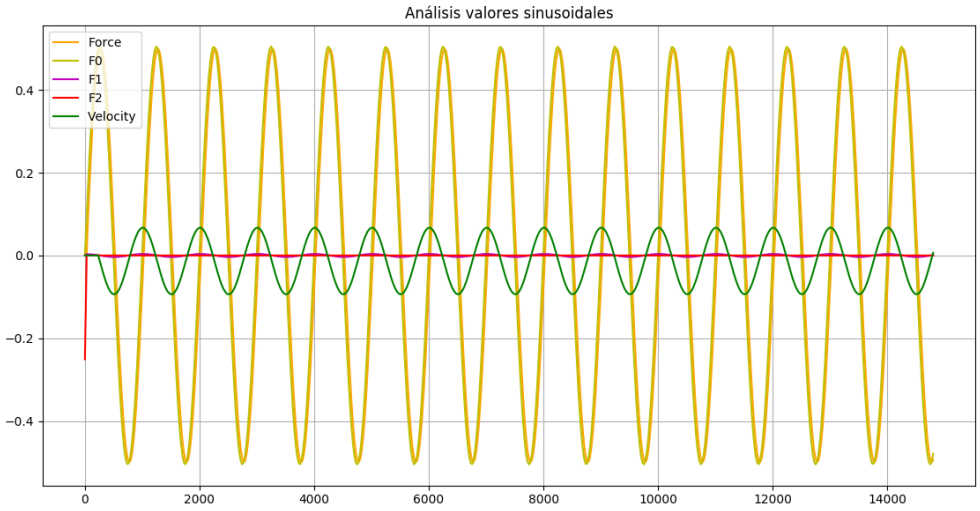

Figura 7.2: Análisis de valores de derivadas de fuerza.

Gracias a esto se pudieron comprobar fallos en el cálculo de las derivadas de f que afectaban al cómputo general de la velocidad. A su vez, se detectaron anomalías en el cálculo con valores iniciales de tiempo, que pudieron ser subsanadas con un filtro para eliminar valores nulos de tiempo en el cálculo.

Fase 2: Dada una velocidad sinusoidal, ¿se obtiene un wait time apropiado?

Una vez conseguida un correcta respuesta de la velocidad a una fuerza sinusoidal, se pasó al análisis del cálculo del tiempo de espera.

Para esta parte fue crucial el paso al testeo en el banco de ensayos. Su uso ayudó en esta fase de las pruebas al hacer mucho más fácil la detección de irregularidades en el movimiento, como trompicones o paradas bruscas.

Una de los primeros fallos detectados fue el no considerar el signo de la velocidad para establecer su dirección, lo que no permitía percibir una velocidad sinusoidal en el motor. A continuación se muestra un gráfico obtenido para el análisis del funcionamiento de esta parte del código donde se ve como la dirección cambia correctamente con el oscilamiento de la velocidad.

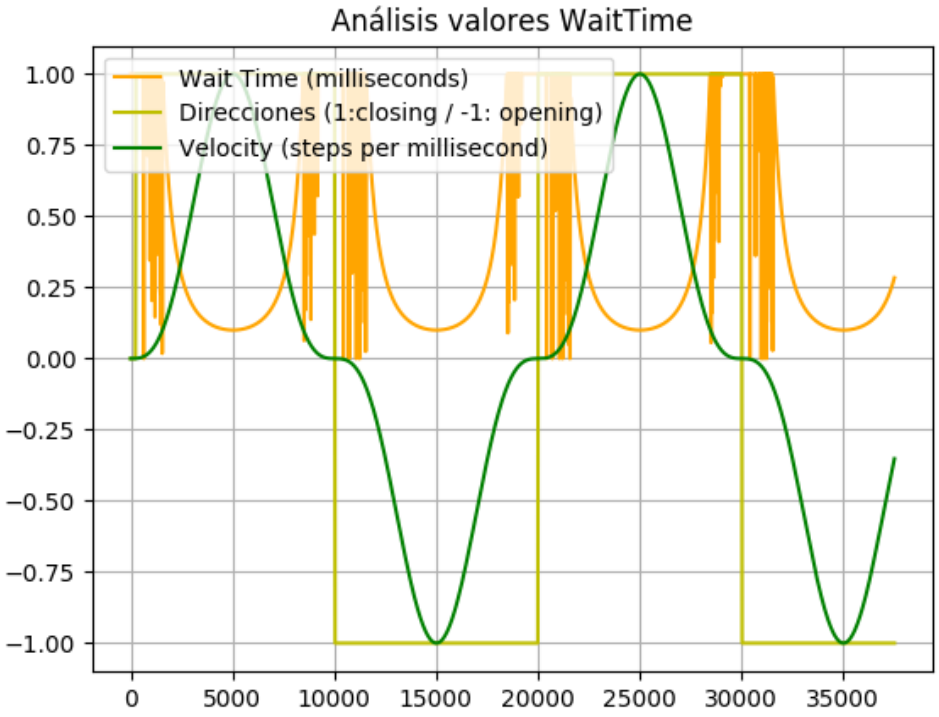

Figura 7.3: Gráfica de análisis de dirección de movimiento.

Como también se puede observar por el gráfico señalado, se obtenía un número sospechosamente grande de valores límite por ambos extremos en el tiempo de espera. Esto hacía que el motor se moviera dando trompicones e incluso se parara y no volviera a arrancar. Para poder analizar estos errores con más detalle se aplicaron cambios a la fórmula de cálculo de la velocidad:

$$\text{velocidad} = 0,1 \cdot (\sin 2\pi \cdot f \cdot \text{tiempo})^3 \tag{7.20}$$

$$\text{velocidad} = 0,1 \cdot (\sin 2\pi \cdot f \cdot \text{tiempo})^5 \tag{7.21}$$

Esto hace que la velocidad dé más valores de arranque cercanos al 0, permitiendo centrarse en la parte problemática de los cálculos.

Con ello se detectaron fallos en la acumulación de tiempo con el tiempo real, y para su mejor análisis se desgranaron las diferentes variables que intervenían en el cálculo erróneo. La siguiente gráfica describe ese proceso, representando un valor de velocidad obtenido usando la ecuación 7.21, y se puede apreciar como la velocidad toma muchos valores cercanos al 0:

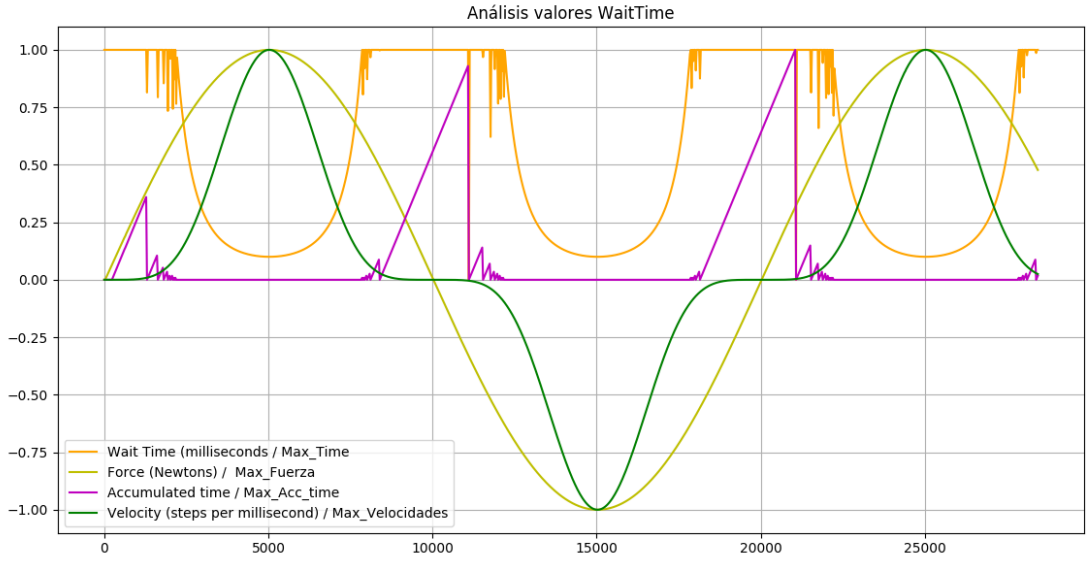

Figura 7.4: Análisis de valores de tiempo de espera.

Como se puede ver en el gráfico anterior, los picos en el valor de tiempo acumulado provocaban una alteración en los valores de WaitTime, que se traducía en un arranque del motor con idas y venidas y falto de fluidez.
Para subsanar este fallo se fueron añadiendo un mayor número de casos en el cálculo del tiempo; hasta que solo se obtuvieron valores límite cuando la velocidad era igual a 0, un comportamiento esperado y correcto. Una vez la velocidad aumentaba, los tiempos de espera se ajustaban a dicha velocidad, haciendo que el motor oscilara correctamente en una dirección y otra.

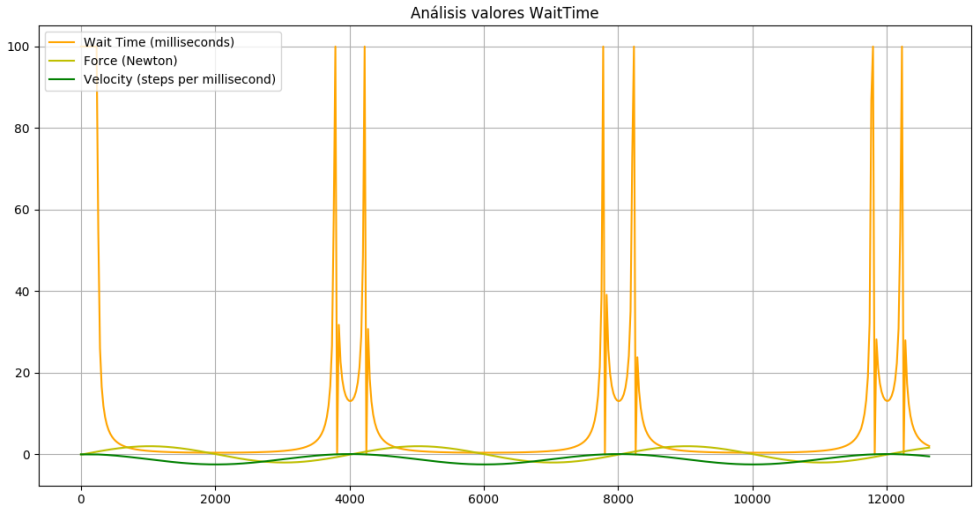

Figura 7.5: Análisis de valores de tiempo de espera finales.

Entre la figura 7.6 y la figura 7.7 se puede observar un cambio entre los valores del tiempo de espera, siendo los del segundo mucho más naturales y alcanzando valores límite (0 y el tiempo máximo de espera) sólo cuando la velocidad es 0.

Para poder asegurar un correcto funcionamiento de los motores, primero se probó dando la velocidad sinusoidal directamente calculada en la raspberry pi. Una vez se comportaron correctamente, se pasó a que el cálculo partiera de una fuerza sinusoidal recibida del arduino. En esta fase se tuvieron problemas con el envío de los valores entre los dos dispositivos, pues los números de fuerza recibidos eran a veces erróneos, lo que paralizaba el cálculo. Para arreglar estos problemas, se diseñó un método de cálculo de valores de fuerza más complejo en el arduino.
Se tomaba el valor del tiempo actual, se calculaba la fuerza con una amplitud a de 0.1 y una frecuencia f de 0.00005:

$$\text{valor de la fuerza} = a \cdot \sin\left(2\pi \cdot f \cdot tiempo\right) \tag{7.22}$$

Esta fuerza se multiplicaba por 10000 para recoger los decimales. Una vez hecho esto, se pasaba a int y dependiendo de si era negativo o positivo el valor, se le restaba o sumaba 10000. Con ello no se perdían los decimales que daban la característica sinusoidal a los valores recibidos, y el cálculo podía hacerse correctamente.
Una vez realizados estos cambios, al ejecutar el programa en la raspberry pi, todo el flujo de información funciona correctamente: recibe una fuerza del arduino, la usa para hacer cálculos y obtiene el valor correspondiente de tiempo de espera entre pasos del motor y con ello el motor se mueve adecuadamente.

Capítulo 8

Conclusiones

8.1. Castellano

Durante el desarrollo de este proyecto, he podido adquirir multitud de experiencias pertenecientes a varios ámbitos del mundo de la informática, tanto de hardware como de software.
Aunque contaba con un amplio conocimiento sobre programación aportado por el estudio del Grado, Python era un lenguaje con el que había trabajado muy poco y el multithreading algo que solo había estudiado desde la perspectiva de la asignatura de Programación de GPUs y Aceleradores. Gracias al desarrollo de este trabajo, he podido familiarizarme con este lenguaje tan dinámico y funcional, profundizando en el uso de la modularización y programación multi-hilo en búsqueda de una mayor eficiencia y fluidez.

A su vez, el trabajar dentro de un proyecto desarrollado para una empresa del nivel de Fossil Ion Tech S.L. me ha permitido aplicar conocimientos tanto de gestión de un proyecto como de documentación detallada y metódica del proceso de trabajo, lo que ha podido facilitar su posterior presentación en esta memoria.

Otro de los aspectos en los que he ganado experiencia ha sido en el apartado hardware de este proyecto. En esta parte he recibido la inmejorable ayuda de un compañero de la empresa, que ha podido guiarme y complementar el conocimiento y aplicación de distintas técnicas para una correcta comprensión y utilización de los componentes usados.

Se puede concluir que se han cumplido los objetivos marcados para este proyecto, pues se han obtenido resultados satisfactorios tanto en la ejecución del código como en los ensayos en el banco de pruebas. En ellos se ha podido probar el funcionamiento del flujo general, la correcta sincronización y respuesta de la interfaz, la eficacia del sistema de registro de eventos y la apropiada ejecución de los hilos. Cumpliendo los requisitos establecidos por la empresa al inicio del proyecto, se prueba que el programa desarrollado es capaz de realizar un control de velocidad de los motores paso a paso certero como respuesta a unos valores de fuerza recibidos desde un arduino que simula una entrada de parámetros por las placas en funcionamiento.

Como futuros objetivos dentro del desarrollo del Bag Desampler queda la finalización del montaje de la máquina y su prueba con el código desarrollado en este proyecto.

8.2. Inglés

During the development of this project, I have been able to acquire a multitude of experiences belonging to various areas of the computer world, both hardware and software related.
Although I had an extensive knowledge about programming provided by the study of the Degree, Python was a language with which I had worked very little and multithreading something I had only studied from the perspective of the subject of Programming of GPUs and Accelerators. Thanks to the development of this project, I have been able to familiarize myself with this dynamic and functional language, deepening in the use of modularization and multi-thread programming in search of greater efficiency and fluidity.

At the same time, working within a project developed for a company of the level of Fossil Ion Tech S.L. has allowed me to apply knowledge of both project management and detailed and methodical documentation of the work process, which has facilitated its subsequent presentation in this report.

Another aspect in which I have gained experience has been in the hardware section. In this part I have received the unbeatable help of a company partner, who has been able to guide me and complement the knowledge and application of different techniques for a correct understanding and use of the components used.

It can be concluded that the objectives set for this project have been met, since satisfactory results have been obtained both in the execution of the code and in the test bench. They have been able to test the operation of the general flow, the correct synchronization and response of the interface, the efficiency of the event registration system and the proper execution of the threads. Fulfilling the requirements established by the company at the beginning of the project, it is proven that the program developed is able to perform a speed control of the stepper motors accurately in response to force values received from an arduino that simulates an input of parameters from the plates in operation.

The future objectives within the development of the Bag Desampler are the completion of the assembly of the machine and its test with the code developed in this project.

Índice de figuras

Bibliografía

[1] Tarek Ziade. *Expert Python Programming.* PACKT, 2008.

[2] Wes McKinney. *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython.* O'REILLY, 2013.

[3] Stéfan van der Walt, Chris S. Colbert, and Gaël Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 22, 2011.

[4] Michael McRoberts. *Beginning Arduino.* APRESS, 2013.

[5] Carl O. Markkanen, Dale B. Blackwell, George A. Knaust, Christian R. Kropac, and John M. Mccall. Stepper motor control system. *Fairchild Camera Instr. Co.*, 1967.

[6] Gabriel Ruiz Garzón. Los orígenes del método de mínimos cuadrados. *Suma 43*, 22:31 – –37, 2003.

PASCAL
ENERO 2018
Ult. actualización 20 de septiembre de 2019
LaTeX lic. LPPL & powered by TEFLON CC-ZERO