E.T.S. INGENIERÍA INFORMÁTICA

UNIVERSIDAD DE MÁLAGA

Escuela Técnica Superior de Ingeniería Informática

Grado en Ingeniería de la Salud.

Diseño e implementación de hardware y software para el apoyo a la investigación en laboratorios

Tools suit design and implementation to manage and support research activities on laboratories

Realizado por:

Samuel Delgado Fernández

Tutorizado por:

José Manuel Jerez Aragonés

Co-tutorizado por:

Fernando Moreno Jabato

Departamento:

Lenguajes y Ciencias de la Computación.

UNIVERSIDAD DE MÁLAGA

MÁLAGA, septiembre de 2019.

Fecha defensa: septiembre de 2019.

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA DE LA SALUD MENCIÓN EN INGENIERÍA BIOMÉDICA, MENCIÓN EN BIOINFORMÁTICA.

DISEÑO E IMPLEMENTACIÓN DE HARDWARE Y SOFTWARE PARA EL APOYO A LA INVESTIGACIÓN EN LABORATORIOS

TOOLS SUIT DESIGN AND IMPLEMENTATION TO MANAGE AND SUPPORT RESEARCH ACTIVITIES ON LABORATORIES

Realizado por
Samuel Delgado Fernández
Tutorizado por
José Manuel Jerez Aragonés
Co-tutorizado por
Fernando Moreno Jabato
Departamento
Lenguajes y Ciencias de la Computación.

UNIVERSIDAD DE MÁLAGA
MÁLAGA, septiembre 2019.

Fecha defensa:

Agradecimientos

A mis padres por su amor incondicional y
ser los hombros sobre los que apoyarme
y a mi tío Diego, por ser el referente
y la fuente de motivación que me inspiró a elegir la Ingeniería.

If I have seen further, It is by standing upong the shoulders of giants
Isaac Newton (1676).

Resumen

En este documento se recoge el diseño, desarrollo e implementación de una modificación para equipos de laboratorio utilizados para la toma de muestras, convirtiéndolo en un dispositivo automatizado basado en Arduino. Mediante el control de motores de paso a través de la placa Arduino y una interfaz gráfica sencilla e intuitiva se ha creado un conjunto de herramientas para el soporte a la investigación en los laboratorios.

Este diseño está pensado para poder ser implementada en los equipos de microscopía más comunes en los laboratorios mediante pequeños ajustes del diseño base, configurado para un modelo genérico de microscopio.

El diseño electro-mecánico del dispositivo está realizado sobre Arduino, implementando módulos de control para motores paso a paso, utilizados comúnmente para maquinaria CNC e impresoras 3D. Además, las imágenes se recogen mediante una cámara USB incorporada en el visor del microscopio y conectada a un computador.

En cuanto a la herramienta software está desarrollada en lenguaje JAVA, con un diseño sencillo de interfaz gráfica para facilitar el uso de cara al usuario final y que gestiona toda la información relevante y la muestra de forma organizada mediante distintos paneles.

Palabras clave: Arduino, Herramienta Laboratorio, Toma de muestras, JAVA, Software, Procesamiento de imagen, Automatización, Microscopio.

Abstract

This document includes the design, development and implementation of a modification for laboratory equipment used for sampling, making them an automatic device based on Arduino. Through the control and monitoring of step motors with Arduino and simple and intuitive graphic interface a research support set of tools has been created.

This modification of the equipment is thought to be able to be implemented in almost all microscopes by adding small modifications of the basic design, setted for a generic microscope model.

The electro-mechanical design of the device is based on Arduino and control modules for stepper motors, more widely used for CNC control and 3D printers. In addition, the images are taken by a USB camera incorporated in the microscope viewer and connected to the computer.

On the other hand, the software tool has been developed in JAVA, with a simple graphic interface design to make easier for the final user and which manage all the important information and show it in a organized way through different panels.

Keywords: Arduino, Laboratory equipment, Sampling, JAVA, Software, Image processing, Automatization, Microspopy.

Índice general

Capítulo 1

Introducción

En el campo de la investigación, análisis de muestras y, en general, en las distintas actividades realizadas en los laboratorios científicos actualmente, es habitual encontrar equipos costosos, tanto a la hora de la adquisición como en el mantenimiento de dichos equipos. Un ejemplo claro en el ámbito de la investigación se podría encontrar en las investigaciones fisiológicas y neurofísicas, que requieren de un control extremadamente preciso de múltiples entradas y salidas, recogidas por un hardware dedicado, de elevado coste, y que frecuentemente requiere de un soporte software de licencia privada para el estudio y la gestión de la información recogida[4]. Esto principalmente se debe a que son equipos de precisión, especialmente diseñados para tareas concretas en entornos específicos. Por otro lado, aunque existe una gran diversidad de herramientas para fines de investigación, pocas de estas herramientas son actualizables mediante modificaciones a las nuevas necesidades que requieren los proyectos. Este hecho hace que poseer un equipo actualizado y de calidad sea cada vez más difícil debido al rápido avance de la tecnología que deja obsoletos equipos con pocos años. Una de las muchas ventajas que ofrece el avance de tecnología es, precisamente, la creación de nuevos dispositivos que, bien diseñados, pueden plantear soluciones de bajo coste y accesibles a la mayoría de usuarios.

Uno de los avances que más a revolucionado el campo de la tecnología y el desarrollo de nuevos dispositivos es la aparición de microcontroladores y soluciones integradas en pla-

cas programables como Arduino o Raspberry Pi, que ofrecen la posibilidad de desarrollar nuevos equipos, modificaciones y extensiones de dispositivos con nuevas funcionalidades, con un nivel de requisitos asequibles para la mayoría de usuarios [9]. Esta revolución se ve incentivada por una corriente de desarrollo denominada como DIY por sus siglas en inglés (Do It Yourself), traducido al castellano como "Hazlo tú mismo". Esta corriente ha llevado a la gran comunidad de creadores a desarrollar un conjunto de herramientas y plataformas que facilitan la incorporación de nuevas líneas de desarrollo.

Partiendo de la situación planteada, este documento recoge el proceso de diseño y la realización del prototipo de un conjunto de herramientas que conformen la base para la creación de equipo competente como solución a la inexistencia de equipos de laboratorios precisos, asequibles y ampliables con mejoras externas diseñadas por la comunidad así como un soporte software que permita la gestión y el seguimiento de muestras y el análisis final de los resultados obtenidos.

Este conjunto de herramientas se compondrá de una plataforma diseñada y modelada en 3D, que será posible imprimir mediante una impresora 3D común, sin altos requisitos técnicos y que alojará el mecanismo necesario para sujetar dos motores paso a paso que se encargaran de mover los tornillos de movimiento de la pletina de un microscopio de laboratorio universitario. Estos motores estarán controlados por un microcontrolador conectado a un computador que, mediante un software de interfaz gráfica permitirá al usuario controlar dicho movimiento de forma sencilla e intuitiva. Por último, sendos componentes del proyecto deberán admitir futuras líneas de trabajo que añadan de forma coherente y sencilla nuevas funcionalidades que incrementen la utilidad y la usabilidad de cualquiera de las partes descritas anteriormente.

Capítulo 2

Objetivos

El objetivo principal de este Trabajo Fin de Grado es aplicar los conocimientos adquiridos durante el Grado en Ingeniería de la Salud en sus menciones de Ingeniería Biomédica y Bioinformática, para el diseño y el prototipado de un conjunto de herramientas que consisten en un dispositivo hardware y una herramienta software para el control, la automatización y la captura de imágenes de muestras situadas en placas Petri a través del ocular de un microscopio.

Dicho dispositivo hardware se basa en una plataforma diseñada en 3D y modelada con SolidWork, donde se han tenido en cuenta las especificaciones y requerimientos físicos que plantea el proyecto. La plataforma debe soportar un conjunto de motores paso a paso y permitir las diferentes configuraciones espaciales que se puedan prever para los equipos de microscopía más comunes en los laboratorios universitarios actuales, así como la correcta sujeción del equipo con la plataforma y la rotación precisa y controlada de los tornillos de movimiento de la pletina. El diseño electro-mecánico debe poder llevarse a cabo de forma genérica lo que plantea el uso de materiales y componentes genéricos fácilmente adquiribles, tanto a nivel de coste como de disponibilidad en los comercios.

Por otro lado, el dispositivo utilizado para la captura de imágenes debe permitir obtener imágenes de suficiente calidad y con una tasa de refresco adecuada para la visualización en tiempo real de la muestra a través del ocular del microscopio.

La herramienta software debe poder gestionar y almacenar de forma ordenada la información de la cámara conectada así como la información relevante referente a la posición de los motores paso a paso, computada a partir de una posición base. Además debe permitir de forma sencilla la comunicación entre la computadora y los dispositivos de movimiento y toma de imágenes así como la visualización en tiempo real. Dicha herramienta debe proporcionar una interfaz gráfica de usuario sencilla e intuitiva que facilite el uso del dispositivo al usuario final.

Por último se espera que el dispositivo físico y la herramienta software permitan la incorporación de nuevas funcionalidades de forma que el presente proyecto se establezca como punto de partida para la creación de nuevos equipos más eficientes y con funcionalidades más avanzadas.

Capítulo 3

Estado del arte

En la última década, el rápido avance de la tecnología ha llevado a las ciencias más clásicas a la renovación continua de los instrumentos de medida y análisis. Siendo el caso de los equipos de microscopía convencionales unos equipos desfasados y que ofrecen pocas funcionalidades en comparación con las nuevas tecnologías, basadas en visualizaciones computerizadas, análisis de imágenes a través de equipos sofisticados, mediciones precisas y algoritmos de inteligencia artificial para la detección de estructuras de interés científico [7].

Desde la implantación en el mercado de los autómatas programables como Arduino, que contienen en su arquitectura un procesador capaz de realizar cálculos y algoritmos complejos [2], se han utilizado para numerosas aplicaciones tanto en robótica básica en sus inicios [9], formación específica en campos innovadores como la e-Health [8] o en aplicaciones prácticas enfocadas a la automatización de procesos en laboratorios [6]. Esta versatilidad permite diseñar herramientas precisas y eficaces para el control de actuadores y el muestreo de señales a través de las entradas y salidas que ofrece la placa.

Además, el bajo coste y la curva de aprendizaje de estos autómatas programables hacen posible que herramientas diseñadas por técnicos sean reproducibles por usuarios nóveles en el campo, creando así la oportunidad de que técnicos de laboratorio sin conocimiento en lenguajes de programación y diseño de hardware puedan implementar de forma ase-

quible nuevos equipos [5].

Por otro lado, la integración de lenguajes de programación como C++, JAVA o Python en estos autómatas, permiten añadir a las funcionalidades propias del lenguaje de Arduino, basado en C, utilidades de éstos lenguajes de programación como pueden ser la creación de interfaces gráficas de usuario o la introducción de nuevas tecnologías como el Internet de las Cosas [3].

Dada la curva de aprendizaje de los actuales lenguajes de programación y las funcionalidades que ofrecen junto con Arduino, Java y Python permiten implementar algoritmos de inteligencia artificial a partir de sus paquetes y librerías. Entre otros, Java es un lenguaje de programación perfecto para la creación de una interfaz gráfica que permita añadir funcionalidades al controlador de Arduino [1].

Además, existen diferentes firmwares ya desarrollados para el control de componentes de movimiento mediante las placas de microcontroladores. Uno de estos firmwares es el conocido como GRBL, desarrollado de forma open-source y que cuenta con el *feedback* de una comunidad activa. Este firmware utiliza las comunicaciones del puerto serie entre el computador y la placa Arduino para el control de los motores, lo que hace necesaria el uso del paquete de *JSSC* (JAVA Simple Serial Connector) que establece las interfaces, clases, excepciones y parámetros necesarios para la correcta comunicación por el puerto serie.

Por otro lado, para la comunicación con el dispositivo óptico que se plantea para el proyecto, es necesario el uso del paquete de *webcam-capture* desarrollado por *sarxos* y que, al igual que ocurre con el paquete mencionado anteriormente, establece las bases para una correcta captura y visualización de las imágenes recibidas por cualquier cámara conectada al ordenador, además de permitir la gestión de los dispositivos ópticos disponibles en el computador.

Capítulo 4

Conceptos básicos

Algunos de los conceptos básicos para la comprensión y desarrollo del proyecto se citarán a continuación de modo que este capítulo sirva de introducción a algunos de los conceptos avanzados que se expondrán durante el resto de la memoria.

4.1. Arduino

Arduino es una familia de placas programables basadas en microcontroladores que mediante pines o puertos de conexión de entrada/salida permiten la interacción entre sensores y efectores y la propia placa. Son placas de uso genérico cuya programación está basada en C++ y que disponen de multitud de sensores, efectores y módulos de ampliación que permiten hacer casi cualquier proyecto.
Tanto la plataforma de desarrollo de Arduino como las propias placas son open-source, se pueden encontrar bajo distintas casas comerciales pero a un coste muy reducido.
Dentro de la familia de placas Arduino se encuentran las placas insignia Arduino UNO (Fig.4.1) basada en un microcontrolador ATmega328p que cuenta con 15 pines digitales, 7 pines analógicos, tanto de entrada como de salida, y pines serie de salida que permiten la comunicación con otros dispositivos. Por otro lado, se encuentra la versión de Arduino Mega, basada en ATmega2560, que cuenta con hasta 60 pines digitales y 20 analógicos, de forma que se amplia la conectividad de la placa arduino además de la velocidad y

capacidad de computación que ofrece la placa base.

Además, y como sugerencia de desarrollo para futuros proyectos es posible establecer conexiones inalámbricas con módulos para las placas Arduino o en su defecto, utilizando las placas donde ya se encuentran estos módulos integrados como el NodeMCU basado en el ESP8266 de Espressif.

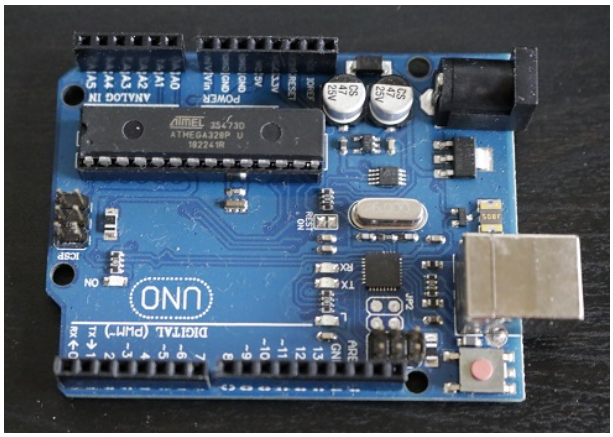

Figura 4.1: Arduino UNO Rev 3.

4.2. Motores NEMA17

Los motores NEMA17 (Fig. 4.2) son un tipo de motores paso a paso, controlados por pines de entrada digitales que controlan la cantidad de pasos que ejecutan en una orden. Los motores NEMA17 son motores validados por una entidad privada que certifica un estándar en diseño. Este tipo de motores se caracterizan por el rango de movimiento que pueden ofrecer, siendo perfectos para un control fino de la rotación. En el caso de los motores elegidos, la sensibilidad de movimiento es de una amplitud de 1.8º.

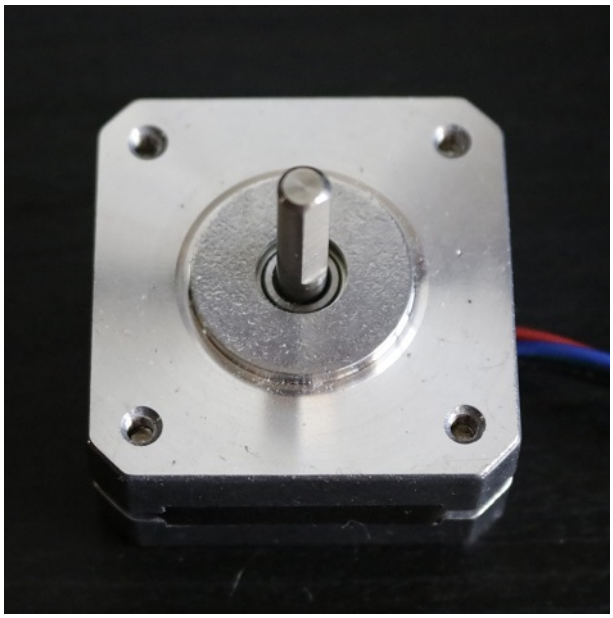

Figura 4.2: Motores NEMA.

4.3. Modulo CNC

El módulo Control Numérico Computerizado (Fig. 4.3) es un módulo de Arduino que facilita mediante un acople entre el módulo y la placa Arduino, la comunicación entre dicha placa y un conjunto de motores paso a paso. Este módulo está diseñado para la creación de máquinas CNC, impresoras 3D o cualquier otro dispositivo que requiera del movimiento tridimensional. Los módulos que se pueden encontrar comercialmente controlan hasta 4 motores paso a paso, mientras que es posible acoplarlo a una placa Arduino, ocupando la totalidad de la placa base de Arduino UNO.

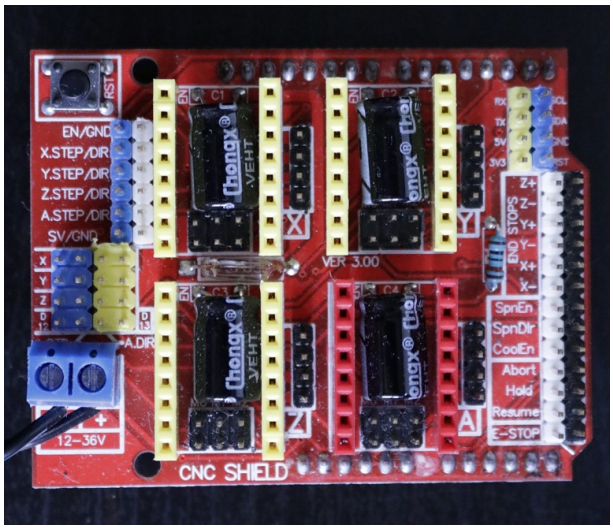

Figura 4.3: Módulo CNC para Arduino.

4.4. Drivers A4988

Los drivers A4988 (Fig. 4.4) o también llamados controladores, son dispositivos que junto con los módulos CNC simplifican el uso y control de los motores paso a paso. Entre sus utilidades encontramos la de regular tanto el voltaje como la intensidad de corriente que necesitan los motores para su funcionamiento. Disponen de protecciones contra la sobreintensidad, cortocircuito y sobretensión además, de contar con disipadores para evitar altas temperaturas, por lo que son dispositivos bien protegidos y que nos aseguran una fiabilidad elevada.

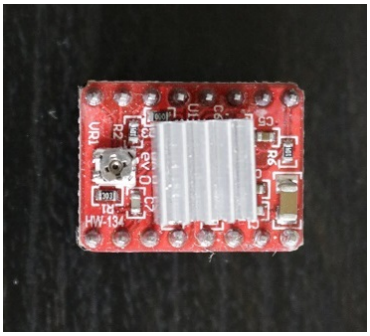

Figura 4.4: Controlador A4988.

4.5. Microscopio

Para una correcta comprensión del documento a nivel técnico, se debe conocer que un microscopio es un instrumento óptico que contiene varias lentes que permiten obtener una imagen ampliada de un objeto microscópico. Para poder desarrollar el proyecto correctamente se expone a continuación una explicación sencilla de las partes ópticas y mecánicas de un microscopio:

- Ocular: Lente situada cerca del ojo del observador. Tiene como función ampliar la imagen del objetivo.
- Objetivo: Lente situada cerca de la muestra. Amplía la imagen de la muestra.
- Soporte: Mantiene la parte óptica. Tiene dos partes: base y brazo.
- Platina o pletina: Lugar donde se deposita la muestra.
- Cabezal: Contiene las lentes oculares. Puede ser monocular o binocular.
- Revólver: Contiene los sistemas de lentes objetivos. Permite mediante giros, cambiar los objetivos.
- Tornillos de enfoque: Macrométrico, que aproxima el enfoque y micrométrico que afina el enfoque con mayor precisión.

Capítulo 5

Requisitos del proyecto

En el siguiente apartado se definen los requisitos funcionales para el conjunto software y hardware del proyecto.

5.1. Requisitos funcionales del hardware

El hardware debe:

- Permitir el movimiento de la pletina en el plano horizontal por medio de motores.
- Ser realizable mediante impresión 3D.
- Ser una estructura modular.
- Ser ajustable a la mayoría microscopios.
- Permitir distintas configuraciones espaciales.
- Permitir la captura imagen a través del ocular.
- Permitir la captura vídeo a través del ocular.
- Mostrar la imagen en tiempo real.

5.2. Requisitos funcionales del software

El software debe:

- Establecer la posición por defecto.
- Calcular los parámetros de calibración.
- Establecer una configuración de los motores.
- Mover la pletina a una posición por defecto.
- Mover la pletina en el plano horizontal.
- Mostrar por pantalla la imagen de la cámara en tiempo real.
- Guardar la imagen de la cámara en un formato estándar.

5.3. Requisitos no funcionales del dispositivo

Los requisitos no funcionales que debe cumplir el dispositivo son:

- El usuario debe poder calibrar el dispositivo.
- El usuario debe poder modificar fácilmente el dispositivo para adaptarlo a cualquier microscopio.

Capítulo 6

Presupuesto del proyecto

El proyecto que se ha desarrollado ha sido especialmente diseñado para favorecer su implementación, es por ello que los materiales utilizados son accesibles a través de comercios locales o tiendas online. Dentro de la lista de materiales se contempla la tornillería y estructura básica como fuselaje mientras que, como otros, se considera el microscopio y fungibles varios como cables o herramientas de trabajo.

En el presupuesto del proyecto no se contemplan ni el firmware utilizado para la Arduino ni el código desarrollado para la interfaz gráfica ya que son de código abierto y están disponibles anexados a este documento y en sus respectivos repositorios web.

A continuación se muestra una lista (Cuadro 6.1) con los materiales utilizados y su precio actual en el mercado. Estos materiales son modificables y su precio variará dependiendo del proveedor y del momento de la compra, por lo que se deben tomar como precios orientativos a fecha de publicación de este documento.

Descripción	Cantidad	Precio	Total
Placa Arduino UNO R3	1	9.99€	9.99€
Controlador de motores paso a paso	1	12.99€	12.99€
Motores NEMA17	2	12.5€	25€
Alimentador 12V 2A o mayor.	1	8.99€	8.99€
Fuselaje	1	30€	30€
Total			95.60€

Cuadro 6.1: Lista de materiales y presupuestos.

Capítulo 7

Desarrollo y diseño de la instrumentación (hardware)

A partir del estudio de los requisitos funcionales del hardware se ha diseñado una estructura base (Fig. 7.1) para impresión 3D con materiales plásticos biodegradables resistentes al calor como el PLA. El diseño cuenta con un estudio de mercado para conocer qué materiales son accesibles fácilmente en comercios locales y vía Internet. La estructura base se ha diseñado mediante un software de diseño 3D y atendiendo a las necesidades del estudio se ha comprobado la resistencia del diseño y de los materiales elegidos de forma que permitan realizar las tareas para las que está destinado el dispositivo. Todas las tuercas y tornillos se han elegido de forma que se trate de una métrica universal, siendo posible una configuración distinta siempre que se modifique también el modelo 3D en base a los requisitos estructurales de cada dispositivo de microscopía.

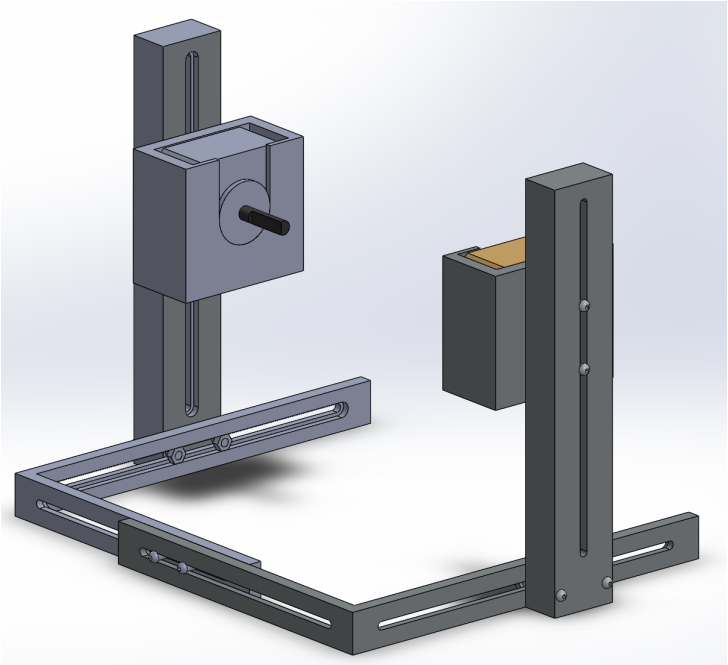

Figura 7.1: Vista general del ensamblaje.

En primer lugar, el diseño de la base es una estructura en escuadra (Fig. 7.2) de un tamaño óptimo para la impresión 3D y que permite múltiples conformaciones, de forma que el diseño en escuadra sea adaptable a distintos tamaños de microscopio. Para el ajuste de precisión se ha troquelado la hendidura de la tornillería con una métrica de 3mm mientras que, en la cara interna de la escuadra se ha conformado una hendidura de 5.5mm capaz de alojar en todo su recorrido tornillos de métrica M3. Este detalle (Fig. 7.3) hace que tanto el tornillo como la tuerca queden ocultos en el interior de la escuadra, mientras que la hendidura de 5.5mm sirve de tope para la tuerca, evitando que rote y facilitando la instalación y el ajuste de la pieza. Además, esta pieza está diseñada con un espesor de 5mm de forma que, junto con la estructura del brazo vertical dejen ocultos los tornillos, evitando dañar el equipo y siendo más estético.

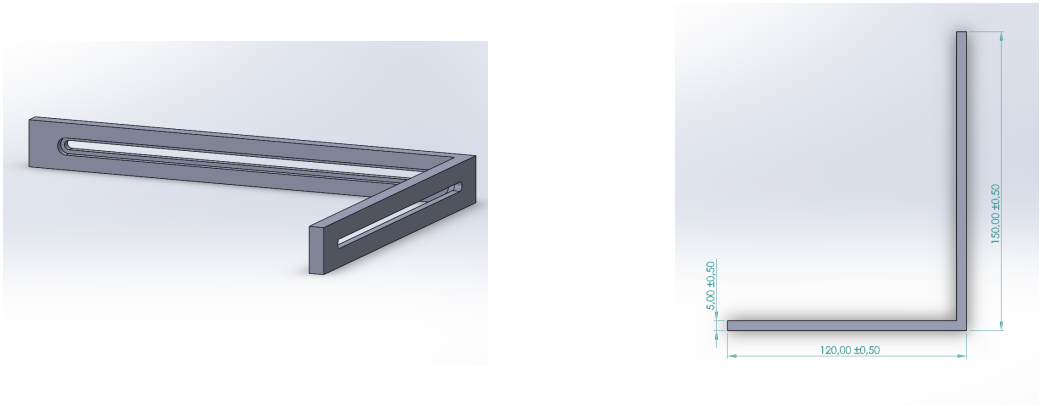

Figura 7.2: Vista de la base de la plataforma.

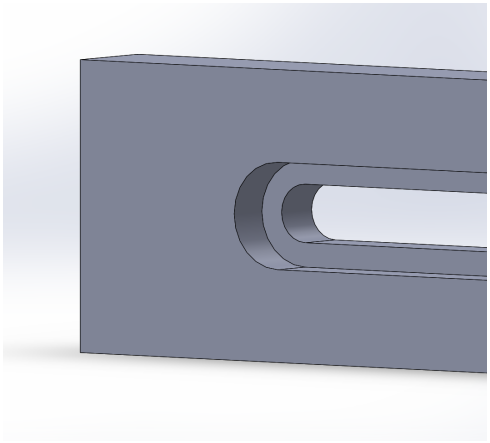

Figura 7.3: Detalle de la base.

La estructura de los brazos verticales (Fig. 7.4) tienen una altura aproximada de 15 cm, con un ancho de 4 cm y un fondo de 1cm. En su parte central se encuentra una hendidura de 3mm de ancho y otra hendidura más superficial de 5.5mm con una profundidad de 2.4mm. En la parte inferior de esta estructura vertical se encuentran dos orificios de 3mm de ancho que alojarían tornillos M3, estos orificios se encuentran a la altura de la hendidura central de la estructura en escuadra, de forma que sirve como anclaje a dicha estructura.

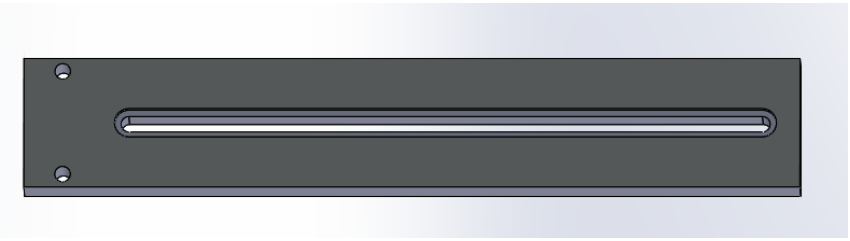

Figura 7.4: Vista del brazo, en horizontal sobre uno de sus lados.

Las cajas de los motores (Fig. 7.5) están diseñadas para alojar motores NEMA17. Cuentan con 4.2cm de lado en su cara cuadrada y 2.1cm de fondo. En la zona central de la caja, y en su pared posterior se encuentra una hendidura para los tornillos M3 y la hendidura propia para las tuercas M3, mientras que en la cara anterior de la caja, se encuentra una hendidura capaz de alojar la protuberancia del motor y el eje de dicho motor. Además, en la parte inferior de la caja, se encuentra un orificio de forma rectangular cuya función es pasar los cables de los motores hasta la caja de electrónica.

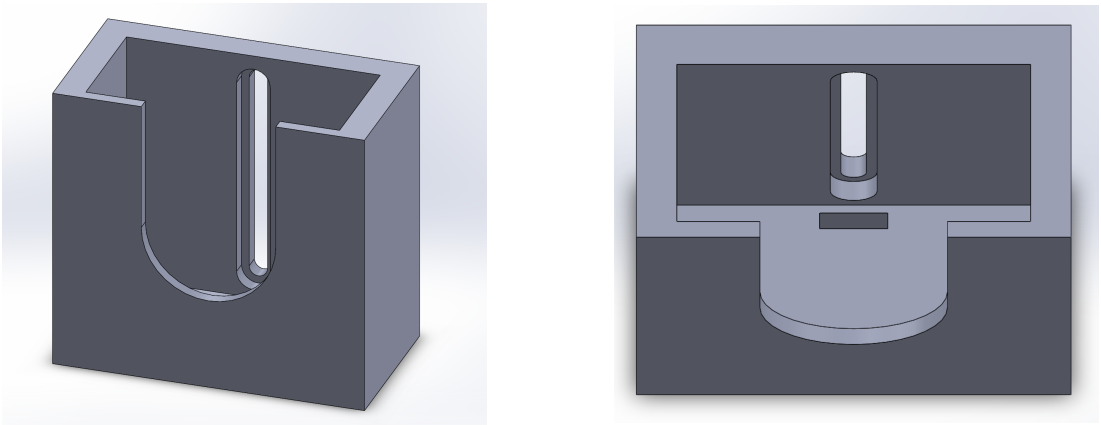

Figura 7.5: Vista de la caja de los motores.

En cuanto a los materiales propuestos para la plataforma, el plástico de impresión PLA es resistente al calor, proporciona una precisión de impresión mayor y es biodegradable. Además es uno de los plásticos de impresión 3D más comunes y asequibles, de forma que son accesibles en comercios locales e internet.

El método de impresión 3D que se propone es mediante un relleno de estructura del

75-80 % para garantizar la estabilidad y robustez de la estructura. Por otro lado, se propone la distribución (Fig. 7.6) de las piezas para una cama caliente de 22x22cm, tamaño estándar de las camas calientes de las actuales impresoras 3D.

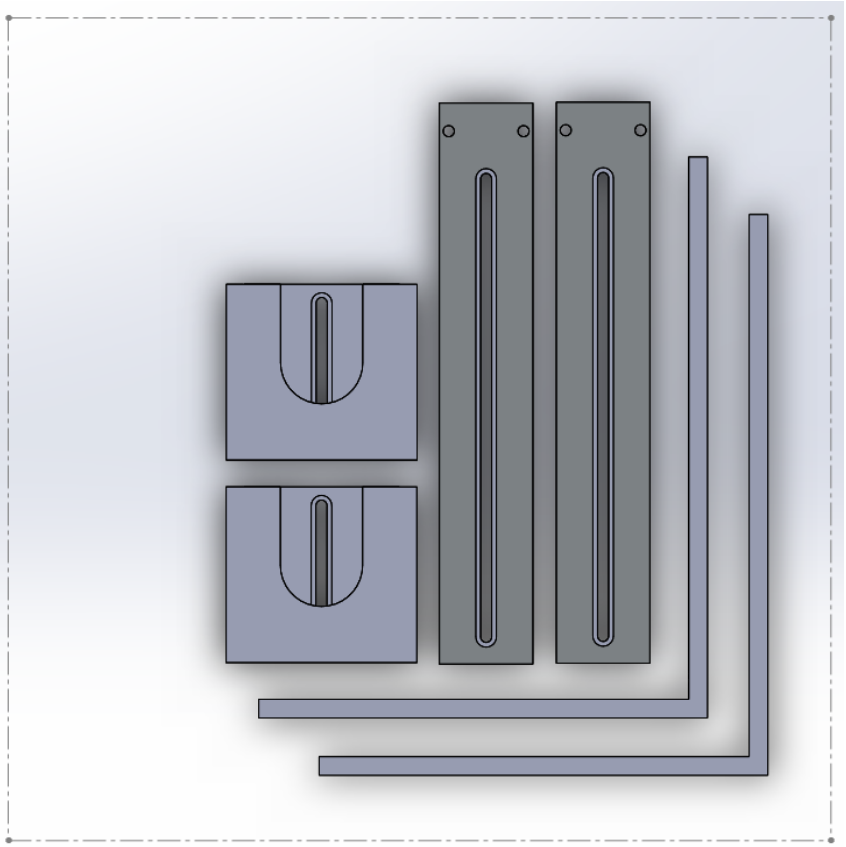

Figura 7.6: Distribución de las piezas para la impresión.

En cuanto a los componentes electro-mecánicos se ha diseñado a partir de una placa Arduino UNO que se comunicará con dos motores paso a paso tipo NEMA17, los cuales tienen un rango de movimiento de aproximadamente 1.8º de giro por cada paso completo. La comunicación entre la placa Arduino y los motores NEMA17 se lleva a cabo a través de un módulo de control numérico computerizado (Módulo CNC), especialmente diseñado para Arduino y que puede encontrarse fácilmente. Este tipo de módulo necesita de controladores independientes para cada motor que se desee utilizar, en el caso del proyecto se han utilizado dos controladores A4988 que disponen de una resistencia variable para

la regulación del voltaje suministrado a los motores. Al tratarse de un movimiento en el plano horizontal y teniendo en cuenta las características de dicho movimiento en los microscopios comunes, se han utilizado dos motores NEMA17 junto con dos controladores A4988. Cada controlador, puede ser configurado para recorrer distintos tamaños de paso desde una decimosexta parte de paso hasta un paso completo, para el proyecto que se ha llevado a cabo se ha utilizado la configuración de paso completo. Por otro lado es necesaria la ventilación de los controladores A4988 mediante pequeñas placas metálicas de disipación, cables dupont con terminación hembra o en su defecto, cables de extensión para los cables de los motores.

Por último, como anexo a esta memoria se incluye un manual detallado de montaje y configuración para el usuario final que provee de la información necesaria para poner en funcionamiento el conjunto de herramientas diseñado durante este proyecto, además de los planos 3D para su modificación, en el caso de que fuese necesaria y para su impresión.

Capítulo 8

Diseño y desarrollo de la herramienta software

El diseño de la herramienta software se basará en la reutilización de código ya existente de otras plataformas desarrolladas para lenguaje JAVA y que entre otras funcionalidades, permiten el control de los motores paso a paso de forma sencilla, así como la posibilidad de automatizar movimientos predefinidos por el usuario y el control de la posición de los motores. Al tratarse de una herramienta software multiplataforma se detallarán por separado el código en C utilizado para la controladora de Arduino y el código desarrollado para la interfaz gráfica de usuario.

8.1. Arduino

Para el control de los motores NEMA a través de la placa de Arduino se ha utilizado el paquete de software GRBL (https://github.com/gnea/grbl), un software desarrollado en C, de código abierto y utilizado para el control de maquinaria de precisión como pueden ser las cortadoras láser, troqueladoras, fresadoras o impresoras 3D.
Este software proporciona un código base que permite controlar mediante comandos sencillos, enviados a través del puerto serie de Arduino, máquinas con movimiento en los

tres ejes sin rotación. Dicho movimiento es precisamente el movimiento necesario para el control de un dispositivo de microscopía convencional.
Además de permitir el movimiento en los tres ejes, permite establecer una posición de referencia a partir de un sistema de coordenadas y realizar movimientos específicos almacenados en archivos de formato abierto.
Mediante la interfaz gráfica desarrollada se controlan los siguientes comandos:

- G0, G1: Movimiento lineal.
- G10 L20: Establecer coordenadas de trabajo.
- G90, G91: Modulación de la distancia.

Estos comandos están descritos en la documentación disponible en el repositorio oficial del proyecto `https://github.com/gnea/grbl` de forma que es accesible a cualquier usuario.

8.2. JAVA

El código desarrollado mediante JAVA se basa en el esquema Modelo-Vista-Controlador y hace uso de diferentes librerías importadas mediante Maven al entorno de trabajo sobre el que se ha trabajado. Aunque generalmente el desarrollo de la herramienta JAVA se ha asemeja al esquema descrito anteriormente hay que que tener en cuenta que tanto el controlador del dispositivo de Arduino como el controlador de la cámara USB están desarrollados a partir de paquetes externos y por tanto se explicarán durante este apartado conforme a las necesidades del proyecto.
Para controlar la comunicación mediante el puerto serie se ha utilizado el paquete JSSC disponible para JAVA 8. Este factor hace que sea necesario el JDK en la versión 1.8.0.221, debido a que, de no ser así, no se asegura la correcta comunicación entre dispositivos. La gestión de dicha comunicación bilateral se lleva a cabo a través de una clase *Arduino*.java desarrollada con los métodos mínimos requeridos para el correcto funcionamiento. Dicha

clase hace uso de la librería JSSC para la comunicación por el puerto serie y de una clase adicional *Position.java* que almacena y gestiona la información referente a la posición actual en la que se encuentran los motores. La comunicación por el puerto serie se lleva a cabo mediante un método denominado *sendCommand()*, el cual espera recibir como argumento del método la cadena de caracteres equivalente al comando que se desea realizar. Dichos comandos están establecidos en la interfaz gráfica como botones. De forma adicional, y con el conocimiento de los comandos se pueden realizar comandos a través de la consola de la propia interfaz gráfica.
Por otro lado, en los mensajes de entrada se ha creado una clase privada *PortReader()* que implementa la interfaz de *SerialPortEventListener* y que permite recibir de forma continua los mensajes emitidos por la placa Arduino. Esta clase permite controlar el estado del dispositivo además permitir comprobar el estado de la conexión y del funcionamiento general del hardware mediante comandos de consola.

8.2.1. Interfaz gráfica de Usuario

La interfaz gráfica (Fig. 8.1) representa el núcleo central de la herramienta software. La información está organizada mediante distintos paneles bien diferenciados y estructurados en base a las funciones básicas que llevan a cabo.

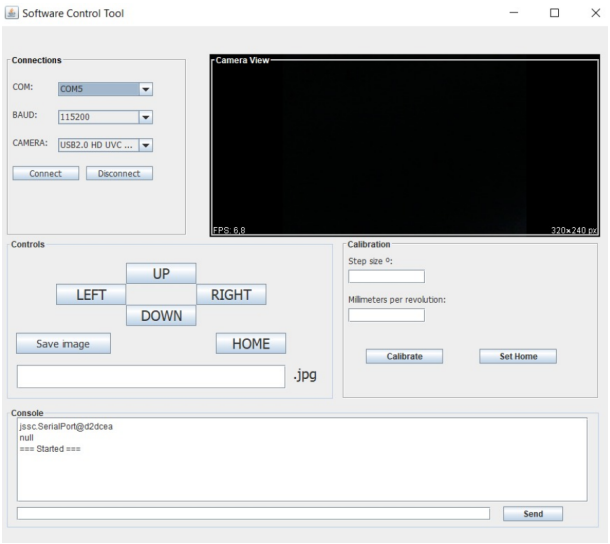

Figura 8.1: Vista general de la interfaz gráfica de usuario.

8.2.1.1. Panel de conexión

En el panel de conexiones (Fig. 8.2) se pueden encontrar tres listas desplegables que gestionan los distintos dispositivos que pueden estar conectados y que son necesarios para el correcto funcionamiento de la herramienta. Este panel utiliza el paquete de *sarxos* para la detección de la cámara web y el paquete de *JSSC* para la detección de los puertos COM utilizados durante la ejecución. Ambos paquetes son utilizados recurrentemente durante todo el código JAVA.

- COM: Lista los dispositivos conectados al puerto serie de la computadora. Por defecto se seleccionará el puerto COM en uso, si existe algún dispositivo Arduino adicional conectado al ordenador aparecerá como elemento de la lista desplegable.

- BAUD: Lista los baudios disponibles para la configuración del puerto serie conectado. Por defecto tiene un valor fijo de 115200 baudios. En caso de ser necesaria la modificación de este elemento, se deberá proporcionar como una modificación del código.

- CAMERA: Lista todos los dispositivos de visualización de imagen disponible en la computadora. Por defecto tiene el valor de la web-cam preestablecida en la computadora. En caso de una cámara distinta, deberá aparecer como elemento seleccionable en la lista desplegable.

Además se encuentran botones funcionales para conexión y desconexión manual del dispositivo hardware. Estos botones pueden servir como método de reseteo manual.

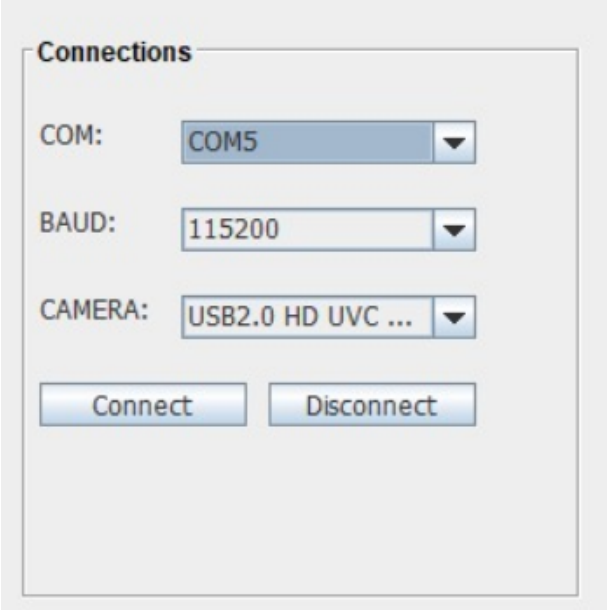

Figura 8.2: Panel de conexiones de la interfaz gráfica de usuario.

8.2.1.2. Panel de cámara

En el panel de cámara o de visualización (Fig.8.3), se encuentra la visualización de la cámara con información básica referente a la imagen recibida en tiempo real. La resolución establecida por defecto es de 320x240 píxeles con una tasa de refresco variable en función de las condiciones lumínicas y de procesamiento tanto de la cámara como de la computadora. En el caso de que se requiera una resolución distinta, se debe modificar el código fuente. En ningún caso es posible establecer una resolución de imagen mayor que la resolución nativa de la cámara, es decir, no se permite el sobre-escalado de la imagen.

Figura 8.3: Panel de visualización de la interfaz gráfica de usuario.

8.2.1.3. Panel de control

El panel de control (Fig. 8.4) está previsto de una botonera sencilla para el control de los motores del dispositivo, así como un botón para el regreso a la posición preestablecida como *HOME* y un campo de texto que funciona junto con el botón *SAVE IMAGE* para guardar una captura de la imagen mostrada por la cámara con un formato que permite el almacenamiento de forma estructurada. Los archivos guardados se almacenan con el nombre de archivo YYYY-MM-DD_NombreDeArchivo_PosX-PosY.jpg de forma que en el nombre del archivo aparece la fecha y las coordenadas en las que se tomó la fotografía.

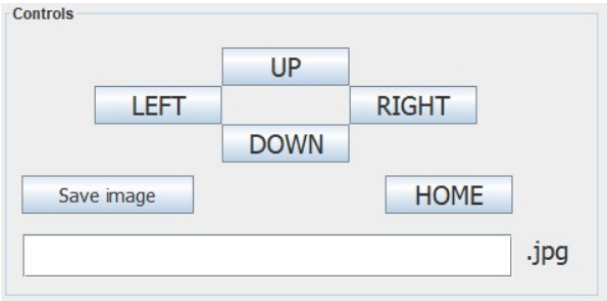

Figura 8.4: Panel de control de la interfaz gráfica de usuario.

8.2.1.4. Panel de calibrado

El panel de calibrado (Fig. 8.5) permite al usuario calibrar el paso de los motores de forma virtual mediante un cálculo computado por el propio programa a partir de la sensibilidad de paso de los motores utilizados, en este caso 1.8º y la distancia recorrida en una vuelta. Este último parámetro se debe comprobar para cada microscopio ya que

depende directamente del nivel de paso del tornillo que posea cada dispositivo. Además, se facilita un botón para la configuración del punto de referencia.

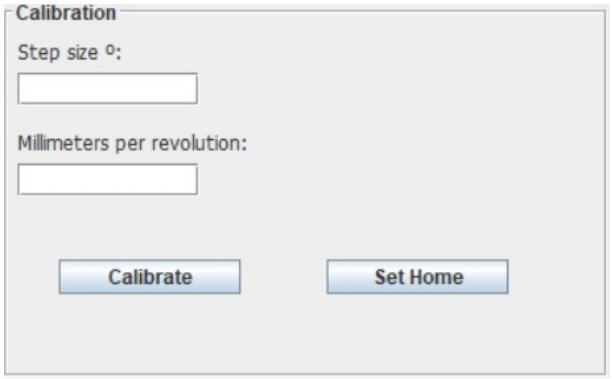

Figura 8.5: Panel de calibrado de la interfaz gráfica de usuario.

8.2.1.5. Panel de consola

En el panel de la consola (Fig. 8.6) se encuentra una consola de comandos que muestra los comandos realizados así como las respuestas que pueda proporcionar Arduino en el transcurso de la conexión. Además se ha configurado un campo de texto editable para la introducción de comandos más complejos que se envían al dispositivo hardware mediante el boton *Send*. Algunas de las funcionalidades que se describen más detalladamente en el manual de usuario son *clear* o *goto*.

Figura 8.6: Panel de la consola de la interfaz gráfica de usuario.

8.2.2. Controlador JAVA de Arduino

En cuanto al controlador del hardware, se ha desarrollado una clase específica *Arduino.java* que contiene las instancias necesarias para controlar el hardware, así como métodos de inicio, configuración y recuperación de los parámetros.

Se han establecido como privadas los parámetros que definen a la clase *Arduino.java* para evitar posibles fallos o una mala configuración por la modificación indebida de dichos parámetros. Además, dentro de dicha clase, se ha creado una clase privada *PortReader* que implementa la interfaz *SerialPortEventListener*. Esta clase permite la comunicación bilateral entre Arduino y el software desarrollado.

La comunicación desde la computadora hacia Arduino se produce a través del método *sendCommand(String cmd)*, el cual espera un comando específico del módulo de Arduino implementado. Si se recibe un comando no reconocido, saltará un error en la consola de comandos del software, evitando el colapso del programa.

Además, la clase *Arduino.java* proporciona un método *moveTo(String cmd)*, el cual espera recibir un *String* similar a *goto(XX, YY)*. Este *String* contiene la instrucción y las coordenadas necesarias para mover los motores a una posición específica.

Capítulo 9

Futuros proyectos

Una de las mayores ventajas del proyecto realizado es la posibilidad de añadir funcionalidades en futuros proyectos. De entre las posibilidades existentes se propone:

- Mejora de la óptica por una cámara con reconocimiento y seguimiento de objetos para el estudio del movimiento.
- Desarrollo de algoritmos basados en técnicas topográficas para el estudio del relieve.
- Análisis de imágenes mediante algoritmos de inteligencia artificial.

Apéndice A

Manual de montaje

Durante el presente capítulo se va a desarrollar una guía de instalación, ensamblado y configuración del conjunto de herramientas. Este capítulo se encuentra además como documento adicional para facilitar su distribución y uso, además de encontrarse disponible en el repositorio del proyecto.

A.1. Ensamblado de la herramienta hardware

En primer lugar, instalaremos, ensamblaremos y configuraremos el hardware de Arduino. Para esta tarea necesitaremos disponer de los siguientes materiales:

- Multímetro.
- Destornillador de precisión de estrella.
- Destornillador de precisión plano.
- Cable USB tipo A-B.
- Dos motores NEMA17.
- Placa Arduino UNO Rev. 3 o MEGA.

- Alimentador 12V a 2A mínimo.
- Dos controladores A4988.
- Dos disipadores para los A4988.
- Cables dupont hembra-macho.
- Alicates de corte (Opcional).
- Soldador (Opcional).
- Estaño (Opcional).

Además de los componentes y herramientas mencionados en la lista anterior, necesitaremos descargar del repositorio el Firmware de GRBL que se puede encontrar en el siguiente enlace `https://github.com/gnea/grbl` y el entorno de desarrollo de Arduino que podemos encontrar en su pagina oficial `https://www.arduino.cc/en/main/software`.
Una vez descargado e instalado el IDE de Arduino, ubicaremos la carpeta donde se almacenan las librerías de Arduino (*Por defecto se suele encontrar en la carpeta* ***Documentos***) y descomprimiremos el Firmware que está en formato *.zip*. Para que el IDE de Arduino detecte el Firmware, debemos colocar la carpeta *grbl* que encontraremos dentro del archivo comprimido dentro de la carpeta *libraries*.
Cuando el IDE y la libreria del Firmware estén correctamente instalados, conectaremos la placa Arduino mediante el cable USB al computador. Para certificar que el computador reconoce correctamente la placa Arduino debemos acceder al IDE y utilizar la opción *Herramientas¿Puerto* (Fig. A.1) y comprobar que se encuentra seleccionado el puerto serie en el que tenemos conectado la placa y el modelo de la placa, que debe coincidir con el modelo que usemos.

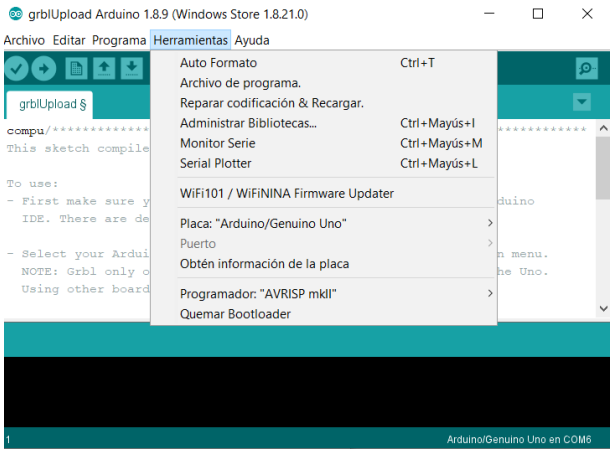

Figura A.1: Entorno de desarrollo de Arduino.

Si no se conoce el puerto al que está conectado la placa Arduino, podemos comprobarlo mediante el menú de administración de dispositivos (*Windows*).

Para las placas de Arduino MEGA debemos comprobar que el chip del procesador esté seleccionado correctamente. Existen dos microchips y puede variar en función del distribuidor de la placa y del modelo.

Seguidamente, con las conexiones comprobadas y el IDE configurado correctamente, utilizamos la opción de *Archivo¿Ejemplos¿grbl¿grblUpload* para cargar el sketch del Firmware que utilizaremos.

No es necesaria la modificación del archivo *grblUpload*

Una vez abierto *grblUpload* utilizamos el botón de subida para cargar el Firmware en nuestra placa Arduino y nos deberá aparecer un mensaje informándonos de que la tarea se ha completado con éxito (Fig. A.2)

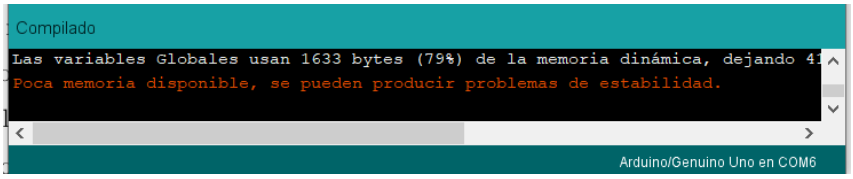

Figura A.2: Mensaje de comprobación del IDE de Arduino.

A continuación debemos conectar el módulo CNC a la placa de Arduino (Fig. A.3)

haciendo coincidir los pines inferiores del módulo con los pines de entrada de Arduino. Debemos tener especial cuidado en que los pines de Arduino coincidan exactamente con los pines inferiores.

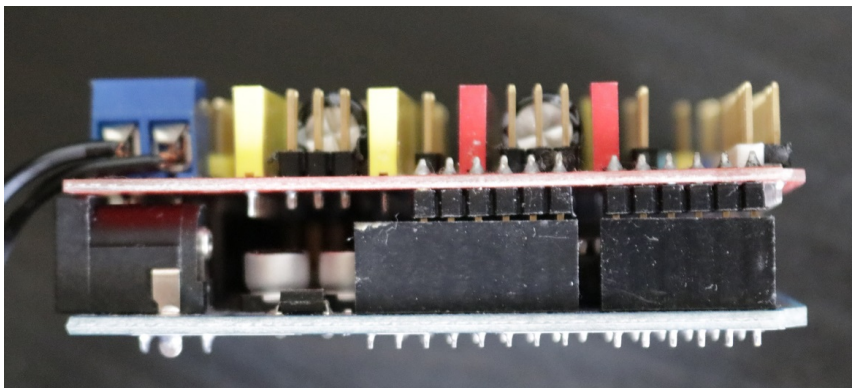

Figura A.3: Conexión entre la placa Arduino (abajo) y el módulo CNC (arriba)

Seguidamente, conectaremos los controladores A4988 al módulo CNC procurando que la resistencia variable situada en los controladores queden correctamente posicionadas orientadas hacia el lado más largo del módulo CNC, de forma que queden los pines de *enable* en la esquina superior iquierda (Fig. A.4).

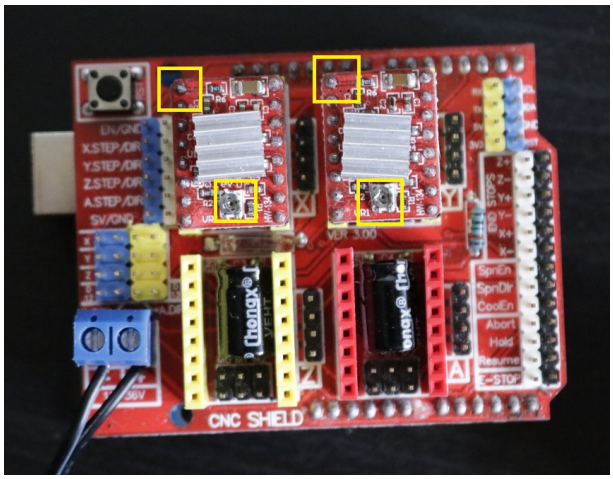

Figura A.4: Conexión de los controladores al módulo CNC. Las resistencias variables y los pines señalados en amarillo.

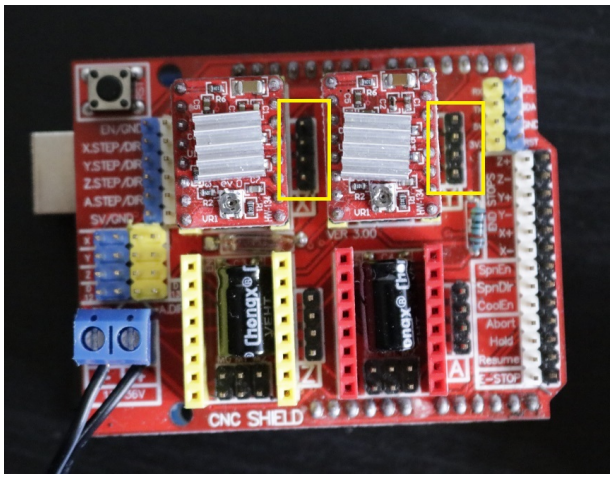

Figura A.5: Pines de conexión de los A4988 en el módulo CNC.

A continuación debemos conectar los cables del alimentador de 12V a la entrada habilitada para ello con la ayuda del destornillador plano de precisión. Una vez conectado a los terminales del módulo CNC, conectamos el alimentador a la red eléctrica.

Para regular la potencia que se les suministra a los motores NEMA17 debemos utilizar el multímetro en una posición adecuada para medir hasta 2V y utilizaremos el la propia resistencia como polo positivo y el conector negativo de 12V como tierra. Debemos variar la resistencia del controlador hasta 956mV aproximadamente.

Para conectar los motores al módulo CNC primero debemos identificar los cables de salida que corresponden a cada una de las bobinas. Para ello utilizaremos el multímetro en comprobación de continuidad y vamos comprobando los cables por parejas, si dan continuidad significa que corresponden a la misma bobina, es decir, corresponden al polo positivo y negativo de la bobina. Una vez identificado los cables, los conectaremos de forma ordenada en los pines correspondientes (Fig. A.5. Las bobinas deben conectarse en secuencia, es decir, polo positivo-negativo de la bobina uno, polo positivo-negativo de la bobina dos.

Con estos pasos, ya hemos terminado la configuración de la parte mecánica. Para la comprobación del funcionamiento de los motores, se recomienda testar los antes de ensamblar la estructura de la base.

Para el ensamblado de la estructura base, se ofrece la siguiente imagen (Fig. A.6) como referencia debido a que puede variar para cada dispositivo de microscopía.

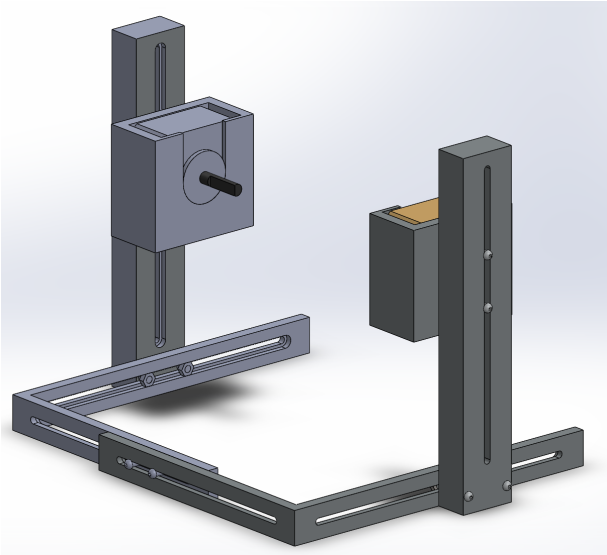

Figura A.6: Ejemplo de ensamblado.

Para facilitar la incorporación de la cámara web al microscopio, se recomienda diseñar un soporte específico para la correcta sujeción de la cámara.

A.2. Instalación de la herramienta software

Para la instalación de la herramienta software debemos descargarla desde el repositorio online `www.github.com/sdelgadofdez/gis-tfg` y ejecutar el archivo *softwareControlTool.jar*.
Para pruebas de desarrollo y nuevas lineas de trabajo se ofrece el directorio completo del proyecto.

A.3. Calibración de la herramienta

Para que la herramienta funcione correctamente, es necesaria la previa calibración de los motores. Para ello, en la herramienta software desarrollada podemos encontrar el panel *Calibration* que nos ofrece la posibilidad de establecer dos parámtros:

- Step size: que se refiere al tamaño en grados del paso del motor. Este parámetro es específico de cada motor y debemos consultarlo en la ficha técnica del fabricante. En el caso del motor utilizado en el proyecto es de 1.8º.

- Distance: que se refiere a la distancia en milímetros que avanza la pletina en cualquiera de las direcciones mediante una vuelta del tornillo. Para calcular dicha distancia, situamos la pletina en un punto de referencia, damos una vuelta completa de forma manual y calculamos la distancia desplazada.

Una vez introducido los parámetros necesarios pulsamos en el botón *Calibrate* y nos aparecerá un mensaje en la consola de comandos referente a los cálculos realizados por el software.

Para establecer un punto de vuelta a casa, por favor, lleve mediante los botones de dirección la pletina al punto deseado y pulse Set Home en el panel de calibración.

Nota: *Si se desplaza la pletina de forma manual, la configuración de la vuelta a casa no será válida, ya que las coordenadas son calculadas a partir de la interacción interfaz-hardware.*

Existen dos comandos disponibles configurados para la consola de comandos:

- clear: limpia el texto existente en la consola de comandos.

- goto(X,Y): mueve los motores hasta la posición X,Y introducida a partir de la coordenada de referencia 0,0 o HOME.

Apéndice B

Códigos realizados

Los códigos que han sido realizados durante el desarrollo de este trabajo se muestran en este anexo.

B.1. Programa principal

```
package test;

import java.awt.EventQueue;
import view.MainView;

public class Main {

	/**
	 * Launch the application.
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
```

```
                                try {
                                        MainView window = new MainView();
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                        }
                });
        }

}
```

B.2. Vista principal de la interfaz

```
package view;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import java.awt.Color;
import java.awt.Dimension;

import javax.swing.JLabel;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

import javax.swing.border.TitledBorder;

import com.github.sarxos.webcam.Webcam;
import com.github.sarxos.webcam.WebcamPanel;
import com.github.sarxos.webcam.WebcamUtils;
import com.github.sarxos.webcam.util.ImageUtils;

import jssc.SerialPortList;
import model.Arduino;

import javax.swing.JComboBox;
```

```
import javax.swing.JButton;
import javax.swing.border.EtchedBorder;
import javax.swing.JTextField;

public class MainView extends JFrame implements ActionListener{

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	private JFrame mainFr;

	private JTextField imgTxtField;

	private JPanel connectionPnl;
	private JPanel viewPnl;
	private JPanel controlPnl;
	private JPanel consolePnl;

	private JComboBox<String> comCB;
	private JComboBox<?> baudCB;
	private JComboBox<?> cameraCB;

	private JLabel lblCom;
	private JLabel lblBaud;
	private JLabel lblCamera;
	private JLabel jpgLbl;
```

```
private JButton upBtn;
private JButton downBtn;
private JButton leftBtn;
private JButton rightBtn;
private JButton saveImgBtn;
private JButton homeBtn;
private JButton connectBtn;
private JButton disconnectBtn;
private JButton sendBtn;

private AppendingTextPane consoleTxt;
private JScrollPane consoleScroll;
private JTextField commandTxt;

//Components to work with.
private Webcam webcam;
private Arduino arduino;

private JPanel calibrationPnl;
private JTextField stepSizeTxt;
private JTextField mmPerStepTxt;

private JLabel lblStepSize;

private JLabel lblMillimetersPerStep;

private JButton btnCalibrate;
```

```
private int imgCount = 0;

private JButton setHomeBtn;




/**
 * Create the application.
 */
public MainView() {
	initialize();
	mainFr.setVisible(true);
}

/**
 * Initialize the contents of the frame.
 */
private void initialize() {
		mainFr = new JFrame();
		mainFr.setBounds(300, 10, 900, 800);
		mainFr.setMinimumSize(new Dimension(900, 800));
		mainFr.setTitle("Software␣Control␣Tool");
		mainFr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE
		    );
```

```
/*
 * Following code makes the connection settings.
 * Those settings are showed in the connection
    panel in the GUI.
 */
connectionPnl = new JPanel();
connectionPnl.setBounds(10, 40, 260, 261);
connectionPnl.setBorder(new TitledBorder(new
   EtchedBorder(EtchedBorder.LOWERED, new Color
   (255, 255, 255), new Color(160, 160, 160)), "
   Connections", TitledBorder.LEADING, TitledBorder
   .TOP, null, new Color(0, 0, 0)));

connectionPnl.setLayout(null);

/*
 * This show and allow to select the serial port
    connection.
 */
String[] portNames = SerialPortList.getPortNames();
String portName = null;
if(portNames.length == 0) {
       comCB = new JComboBox<String>();
}else {
       comCB = new JComboBox<String>(portNames);
       portName = comCB.getSelectedItem().toString
          ();
}
```

```
/*
 * This show in the GUI the most common BAUD rates.
 */
List<String> baudList = new ArrayList<String>();
baudList.add("115200");

baudCB = new JComboBox(baudList.toArray());
baudCB.setSelectedIndex(0);
String baud = baudCB.getSelectedItem().toString();


lblCom = new JLabel("COM:");
lblCom.setFont(new Font("Tahoma", Font.PLAIN, 12));
lblCom.setBounds(10, 40, 55, 13);

comCB.setFont(new Font("Tahoma", Font.PLAIN, 12));
comCB.setBounds(75, 40, 135, 20);

lblBaud = new JLabel("BAUD:");
lblBaud.setBounds(10, 80, 55, 13);
lblBaud.setFont(new Font("Tahoma", Font.PLAIN, 12));


baudCB.setFont(new Font("Tahoma", Font.PLAIN, 12));
baudCB.setBounds(75, 80, 135, 20);
```

```
lblCamera = new JLabel("CAMERA:");
lblCamera.setFont(new Font("Tahoma", Font.PLAIN, 12)
   );
lblCamera.setBounds(10, 120, 55, 13);

List<Webcam> webcams = Webcam.getWebcams();
if (webcams.size() == 0) {
              webcam = null;
              cameraCB = new JComboBox();
              viewPnl = new JPanel();
}else {
              List<String> cameras = new ArrayList<
                 String>();
              for(Webcam wc:webcams) {
                     cameras.add(wc.getName());
              }
              cameraCB = new JComboBox(cameras.
                 toArray());
}

cameraCB.setFont(new Font("Tahoma", Font.PLAIN, 12))
   ;
cameraCB.setBounds(75, 120, 135, 20);

connectBtn = new JButton("Connect");
connectBtn.setFont(new Font("Tahoma", Font.PLAIN,
   12));
connectBtn.setBounds(10, 160, 95, 20);
```

```
disconnectBtn = new JButton("Disconnect");
disconnectBtn.setFont(new Font("Tahoma", Font.PLAIN,
    12));
disconnectBtn.setBounds(115, 160, 95, 20);

connectionPnl.add(lblCom);
connectionPnl.add(comCB);
connectionPnl.add(lblBaud);
connectionPnl.add(baudCB);
connectionPnl.add(lblCamera);
connectionPnl.add(cameraCB);
connectionPnl.add(connectBtn);
connectionPnl.add(disconnectBtn);

if(portName != null) {
       arduino = new Arduino(portName, baud);
}else {
       arduino = null;
}
/*
 * Webcam Panel config
 */

if (webcams.size() == 0) {
              viewPnl = new JPanel();
}else {
```

```
                webcam = Webcam.getWebcamByName((
                    String)cameraCB.getSelectedItem())
                    ;
                webcam.setViewSize(new Dimension
                    (320,240));
                viewPnl = new WebcamPanel(webcam);
                ((WebcamPanel) viewPnl).
                    setFPSDisplayed(true);
                ((WebcamPanel) viewPnl).
                    setDisplayDebugInfo(false);
                ((WebcamPanel) viewPnl).
                    setImageSizeDisplayed(true);
                ((WebcamPanel) viewPnl).setMirrored(
                    true);
        }
        viewPnl.setBounds(301, 40, 557, 261);
        viewPnl.setBorder(new TitledBorder(new EtchedBorder(
            EtchedBorder.LOWERED, new Color(255, 255, 255),
            new Color(160, 160, 160)), "Camera View",
            TitledBorder.LEADING, TitledBorder.TOP, null,
            new Color(255,255,255)));


        viewPnl.setLayout(null);



        /*
```

```
 * Following code makes the control functions.
 * Those settings are showed in the control panel
    in the GUI.
 */
controlPnl = new JPanel();
controlPnl.setBounds(10, 303, 469, 231);
controlPnl.setBorder(new TitledBorder(null, "
   Controls", TitledBorder.LEADING, TitledBorder.
   TOP, null, null));

upBtn = new JButton("UP");
upBtn.setBounds(172, 35, 100, 30);
upBtn.setFont(new Font("Tahoma", Font.PLAIN, 20));

downBtn = new JButton("DOWN");
downBtn.setBounds(172, 95, 100, 30);
downBtn.setFont(new Font("Tahoma", Font.PLAIN, 20));

leftBtn = new JButton("LEFT");
leftBtn.setBounds(72, 65, 100, 30);
leftBtn.setFont(new Font("Tahoma", Font.PLAIN, 20));

rightBtn = new JButton("RIGHT");
rightBtn.setBounds(272, 65, 100, 30);
rightBtn.setFont(new Font("Tahoma", Font.PLAIN, 20))
   ;

saveImgBtn = new JButton("Save␣image");
```

```
saveImgBtn.setBounds(15, 135, 135, 30);
saveImgBtn.setFont(new Font("Tahoma", Font.PLAIN,
   15));

homeBtn = new JButton("HOME");
homeBtn.setBounds(300, 135, 100, 30);
homeBtn.setFont(new Font("Tahoma", Font.PLAIN, 20));


imgTxtField = new JTextField();
imgTxtField.setBounds(16, 181, 384, 34);
imgTxtField.setFont(new Font("Tahoma", Font.PLAIN,
   20));
imgTxtField.setColumns(10);

jpgLbl = new JLabel(".jpg");
jpgLbl.setBounds(410, 181, 110, 25);
jpgLbl.setFont(new Font("Tahoma", Font.PLAIN, 20));

controlPnl.setLayout(null);
controlPnl.add(saveImgBtn);
controlPnl.add(homeBtn);
controlPnl.add(leftBtn);
controlPnl.add(downBtn);
controlPnl.add(upBtn);
controlPnl.add(rightBtn);
controlPnl.add(imgTxtField);
controlPnl.add(jpgLbl);
```

```
/*
 * Following code show the console.
 * Those settings are showed in the console panel
    in the GUI.
 */
consolePnl = new JPanel();
consolePnl.setBounds(10, 544, 848, 174);
consolePnl.setBorder(new TitledBorder(null, "
   Console", TitledBorder.LEADING, TitledBorder.TOP
   , null, null));

mainFr.getContentPane().setLayout(null);

consoleTxt = new AppendingTextPane();
consoleScroll = new JScrollPane(consoleTxt);
consoleScroll.setBounds(16, 15, 815, 121);

commandTxt = new JTextField();
commandTxt.setBounds(16, 143, 676, 19);
commandTxt.setColumns(10);

sendBtn = new JButton("Send");
sendBtn.setBounds(710, 142, 85, 21);
consolePnl.setLayout(null);
consolePnl.add(commandTxt);
```

```
consolePnl.add(sendBtn);
consolePnl.add(consoleScroll);

/*
 * Following code show the calibration panel.
 * Those settings are showed in the calibration
    panel in the GUI.
 */

calibrationPnl = new JPanel();
calibrationPnl.setBorder(new TitledBorder(new
   EtchedBorder(EtchedBorder.LOWERED, null, null),
   "Calibration", TitledBorder.LEADING,
   TitledBorder.TOP, null, null));
calibrationPnl.setBounds(489, 303, 369, 231);
calibrationPnl.setLayout(null);

lblStepSize = new JLabel("Step␣size␣\u00BA:");
lblStepSize.setFont(new Font("Tahoma", Font.PLAIN,
   12));
lblStepSize.setBounds(10, 25, 65, 15);


lblMillimetersPerStep = new JLabel("Millimeters␣per␣
   revolution:");
lblMillimetersPerStep.setFont(new Font("Tahoma",
   Font.PLAIN, 12));
lblMillimetersPerStep.setBounds(10, 80, 157, 15);
```

```
btnCalibrate = new JButton("Calibrate");
btnCalibrate.setBounds(35, 160, 100, 20);


stepSizeTxt = new JTextField();
stepSizeTxt.setBounds(10, 45, 110, 20);

stepSizeTxt.setColumns(10);

mmPerStepTxt = new JTextField();
mmPerStepTxt.setBounds(10, 100, 110, 20);

mmPerStepTxt.setColumns(10);

setHomeBtn = new JButton("Set␣Home");
setHomeBtn.setBounds(195, 160, 100, 20);

calibrationPnl.add(setHomeBtn);
calibrationPnl.add(lblMillimetersPerStep);
calibrationPnl.add(btnCalibrate);
calibrationPnl.add(stepSizeTxt);
calibrationPnl.add(mmPerStepTxt);
calibrationPnl.add(lblStepSize);


mainFr.getContentPane().add(viewPnl);
```

```
                mainFr.getContentPane().add(connectionPnl);


                mainFr.getContentPane().add(controlPnl);
                mainFr.getContentPane().add(calibrationPnl);


                mainFr.getContentPane().add(consolePnl);

                // Add ActionListener for each button.
                connectBtn.addActionListener(this);
                disconnectBtn.addActionListener(this);

                upBtn.addActionListener(this);
                downBtn.addActionListener(this);
                leftBtn.addActionListener(this);
                rightBtn.addActionListener(this);
                saveImgBtn.addActionListener(this);
                homeBtn.addActionListener(this);

                sendBtn.addActionListener(this);

                btnCalibrate.addActionListener(this);

                consoleTxt.appendText("=== Started ===");

    }
    public void actionPerformed(ActionEvent e) {
```

```
try {
        if(e.getSource() == upBtn) {
                arduino.sendCommand("G91␣G0␣Y1");
                arduino.getPos().moveY(1);
                consoleTxt.appendText(arduino.getPos().
                   toString());
                consoleTxt.appendText(arduino.getResponse())
                   ;
        }else if(e.getSource() == downBtn) {
                arduino.sendCommand("G91␣G0␣Y-1");
                arduino.getPos().moveY(-1);
                consoleTxt.appendText(arduino.getPos().
                   toString());
                consoleTxt.appendText(arduino.getResponse())
                   ;
        }else if(e.getSource() == leftBtn) {
                arduino.sendCommand("G91␣G0␣X-1");
                arduino.getPos().moveX(-1);
                consoleTxt.appendText(arduino.getPos().
                   toString());
                consoleTxt.appendText(arduino.getResponse())
                   ;
        }else if(e.getSource() == rightBtn) {
                arduino.sendCommand("G91␣G0␣X1");
                arduino.getPos().moveX(1);
                consoleTxt.appendText(arduino.getPos().
                   toString());
                consoleTxt.appendText(arduino.getResponse())
```

```
                ;
        }else if(e.getSource() == saveImgBtn) {
                if(webcam != null) {

                        String imgName = imgTxtField.getText
                            ();
                        LocalDate date = LocalDate.now();
                        if(arduino != null) {
                                imgName = date + "_" + imgName
                                    + "_" + imgCount + "_" +
                                   arduino.getPos().getX() + "
                                   -" + arduino.getPos().getY
                                   ();
                        }else {
                                imgName = date + "_" + imgName
                                    + "_" + imgCount;
                        }
                        WebcamUtils.capture(webcam, imgName,
                            ImageUtils.FORMAT_JPG);
                        imgCount++;
                }else {
                        consoleTxt.appendText("Impossible␣to␣
                            connect␣with␣the␣camera.␣Please␣
                            check␣the␣connection.");
                }
        }else if(e.getSource() == homeBtn) {
                arduino.sendCommand("G90␣G0␣X0␣Y0");
                arduino.getPos().setPosition(0, 0);
```

```
            consoleTxt.appendText(arduino.getPos().
               toString());
            consoleTxt.appendText(arduino.getResponse())
               ;
        }else if(e.getSource() == connectBtn) {
            arduino.connect();
            consoleTxt.appendText(arduino.getResponse())
               ;

        }else if(e.getSource() == disconnectBtn) {
            arduino.disconnect();
            consoleTxt.appendText("DISCONNECTED");
            consoleTxt.appendText(arduino.getResponse())
               ;
        }else if(e.getSource() == sendBtn) {
            String command = commandTxt.getText();
            if(command.equalsIgnoreCase("clear")) {
                    consoleTxt.setText("");
                    commandTxt.setText("");
            }
            else if(command.toUpperCase().contains("goto
               ")){
                    arduino.moveTo(command.toUpperCase())
                       ;
            }else {
                    arduino.sendCommand(command);
                    consoleTxt.appendText(arduino.
                       getResponse());
```

```
                                commandTxt.setText("");
                        }
                    }else if(e.getSource() == btnCalibrate) {
                        consoleTxt.appendText(arduino.calibrate(
                            stepSizeTxt.getText(), mmPerStepTxt.
                            getText()));
                        stepSizeTxt.setText("");
                        mmPerStepTxt.setText("");
                        consoleTxt.appendText(arduino.getResponse())
                            ;
                    }else if(e.getSource() == setHomeBtn) {
                        arduino.sendCommand("G10␣P0␣L20␣X0␣Y0");
                        consoleTxt.appendText("HOME␣set");
                        consoleTxt.appendText(arduino.getResponse())
                            ;
                    }
                } catch (NullPointerException e1) {
                    // TODO Auto-generated catch block
                    consoleTxt.setText(e1.getMessage());
                }
            }
}
```

B.3. Consola con *scroll* vertical

```
package view;

import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.SimpleDateFormat;
import java.util.Date;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextPane;
import javax.swing.text.BadLocationException;
import javax.swing.text.Document;
import javax.swing.text.Position;
import javax.swing.text.StyledDocument;

public class AppendingTextPane extends JTextPane {
  public AppendingTextPane() {
    super();
  }

  public AppendingTextPane(StyledDocument doc) {
    super(doc);
  }

  // Appends text to the document and ensure that it is visible
  public void appendText(String text) {
```

```
    try {
      Document doc = getDocument();

      // Move the insertion point to the end
      setCaretPosition(doc.getLength());

      // Insert the text
      replaceSelection(text+"\n");

      // Convert the new end location
      // to view co-ordinates
      Rectangle r = modelToView(doc.getLength());

      // Finally, scroll so that the new text is visible
      if (r != null) {
        scrollRectToVisible(r);
      }
    } catch (BadLocationException e) {
      System.out.println("Failed to append text: " + e);
    }
  }
}
```

B.4. Clase para Arduino

```
package model;

import jssc.SerialPort;
import jssc.SerialPortEvent;
import jssc.SerialPortEventListener;
import jssc.SerialPortException;

public class Arduino {
	private SerialPort serialPort;
	private String baud;
	private PortReader portReader;

	private Position position;

	public Arduino(String portName, String baud) {
		try {
			this.serialPort = new SerialPort(portName);
			serialPort.openPort();
			position = new Position(0, 0);

			this.portReader = new PortReader();

			serialPort.setParams(
						SerialPort.BAUDRATE_115200,
						SerialPort.DATABITS_8,
						SerialPort.STOPBITS_1,
						SerialPort.PARITY_NONE);
```

```
                serialPort.setFlowControlMode(SerialPort.
                    FLOWCONTROL_RTSCTS_IN | SerialPort.
                    FLOWCONTROL_RTSCTS_OUT);
                serialPort.addEventListener(portReader, SerialPort.
                    MASK_RXCHAR);

                if(this.getResponse() != null) {

                }
        } catch (SerialPortException e) {
                e.printStackTrace();
        }
}
public void connect() {
        try {
                this.serialPort.openPort();
        } catch (SerialPortException e) {
                e.printStackTrace();
        }
}
public void disconnect() {
        try {
                this.serialPort.closePort();
        }catch(SerialPortException e) {
                e.printStackTrace();
        }
```

```
}
public Position getPos() {
	return position;
}
public String getPortName() {
	return serialPort.toString();
}
public String getBaud() {
	return baud;
}
public void sendCommand(String command) {
	/*
	 * this code should received data to communicate with
	 * arduino through Serial ports.
	 */
	try {
		char ESC = (char) 27; //ESC
		char LN = (char) 10; //Break Line.
		command = ESC + command + LN;

		serialPort.writeString(command);
	} catch (SerialPortException e) {

		e.printStackTrace();
	}
}
public String getResponse() {
	return portReader.getResponse();
```

```
}
public String calibrate(String stepSize, String mmStep) {
        double stepsPerMm = 0;
        try {
                double size = Double.parseDouble(stepSize);
                double distance = Double.parseDouble(mmStep);

                double stepsPerRev = 360 / size;

                stepsPerMm = stepsPerRev/distance;

                this.sendCommand("$100="+stepsPerMm);
                this.sendCommand("$101="+stepsPerMm);

        } catch (NumberFormatException e) {
                e.printStackTrace();
        }
        return "Settings␣calibrate␣for␣step␣size␣=␣" + stepSize + "
           ž;␣distance␣=␣" + mmStep + "mm.␣Factor␣movement␣=␣" +
           stepsPerMm;
}

public void moveTo(String cmd) {
        int currentX = position.getX();
        int currentY = position.getY();

        cmd = cmd.replace("GOTO(", "").replace(")","");
        String[] coordenates = cmd.split(",");
```

```
        int x = Integer.parseInt(coordenates[0]);
        int y = Integer.parseInt(coordenates[1]);

        while(currentX != x && currentY != y) {
                if(x<currentX) {
                        this.sendCommand("G91␣G0␣X1");
                        position.moveX(1);
                }else if(x>currentX) {
                        this.sendCommand("G91␣G0␣X-1");
                        position.moveX(-1);
                }
                if(y<currentY) {
                        this.sendCommand("G91␣G0␣Y1");
                        position.moveY(1);
                }else if(y>currentY) {
                        this.sendCommand("G91␣G0␣Y-1");
                        position.moveY(-1);
                }
        }
}

private class PortReader implements SerialPortEventListener {

        private String receivedResponse;

        public void serialEvent(SerialPortEvent event) {
                if(event.isRXCHAR() && event.getEventValue()>0) {
```

```
                        try {
                                receivedResponse = serialPort.
                                    readString(event.getEventValue());
                                System.out.println(receivedResponse);
                        }catch(SerialPortException e) {
                                System.out.println(e.getMessage());
                        }catch(RuntimeException e) {
                                System.out.println(e.getMessage());
                        }
                }
        }
        public String getResponse() {
                return receivedResponse;
        }
    }
}
```

B.5. Clase para Position

```
package model;

public class Position {
	private int x;
	private int y;

	public Position(int x, int y) {
		this.x = x;
		this.y = y;
	}

	public int getX() {
		return x;
	}
	public void setX(int x) {
		this.x = x;
	}
	public int getY() {
		return y;
	}
	public void setY(int y) {
		this.y = y;
	}

	public void setPosition(int x, int y) {
		this.x = x;
		this.y = y;
```

```
        }
        public void moveX(int step) {
                this.setX(this.getX()+step);
        }
        public void moveY(int step) {
                this.setY(this.getY()+step);
        }

        public String toString() {
                return "PosX: " + this.getX() + "; " + "PosY: " + this.getY
                    ();
        }
}
```

B.6. Excepción de cámara

```
package exceptions;


public class CameraException extends Exception{
	public CameraException() {
		super();
	}
	public CameraException(String mssg) {
		super(mssg);
	}
}
```

Bibliografía

[1] Thomas Abeel, Y Van De Peer, and Y Saeys. Java-ML : A Machine Learning Library. *Journal of Machine Learning Research*, 2009.

[2] Arduino. ARDUINO UNO REV3, 2018.

[3] Gianluca Barbon, Michael Margolis, Filippo Palumbo, Franco Raimondi, and Nick Weldin. Taking Arduino to the Internet of Things: The ASIP programming model. *Computer Communications*, 2016.

[4] Alessandro D'Ausilio. Arduino: A low-cost multipurpose lab equipment. *Behavior Research Methods*, 2012.

[5] Štěpánka Kubínová and Jan Šlégr. ChemDuino: Adapting Arduino for Low-Cost Chemical Measurements in Lecture and Laboratory. *Journal of Chemical Education*, 2015.

[6] Jovana Z. Milanovic, Predrag Milanovic, Rastislav Kragic, and Mirjana Kostic. "Do-It-Yourselfreliable pH-stat device by using open-source software, inexpensive hardware and available laboratory equipment. *PLoS ONE*, 2018.

[7] Firas Mualla, Marc Aubreville, and Andreas Maier. Microscopy. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2018.

[8] S. T. Puente, A. Úbeda, and F. Torres. e-Health: Biomedical instrumentation with Arduino. *IFAC-PapersOnLine*, 2017.

[9] John Sarik and Ioannis Kymissis. Lab kits using the arduino prototyping platform. In *Proceedings - Frontiers in Education Conference, FIE*, 2010.