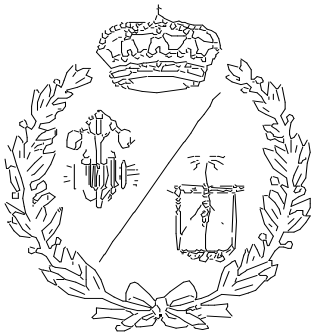
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN
UNIVERSIDAD DE CANTABRIA



Proyecto Fin de Grado

Construcción de un robot móvil para impresión de documentos transferidos mediante comunicación inalámbrica.

(Construction of a mobile robot for printing documents transferred by Wireless communication.)

Para acceder al Título de

GRADUADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: David Rodríguez Fernández

Julio - 2024

RESUMEN

Este trabajo trata de partir de un diseño anterior del TFG realizado por Javier López Martínez “Implementación de un robot móvil para impresión de documentos”, construir un prototipo funcional, en el que se busca actualizar mejorar e implementar el diseño, así como establecer una nueva forma de comunicación, que se lleve a cabo mediante una aplicación instalada en un dispositivo móvil, desarrollada en el software de diseño Android Studio. La comunicación se realiza vía bluetooth, mediante la cual el módulo bluetooth HC-05 instalado en el Arduino recibirá los planos que debe imprimir, el Arduino interpretará el código (G-code) enviado traduciendo este al movimiento sincronizado de cuatro motores paso a paso, desplazándose por el plano XY mientras traza una trayectoria que define la imagen deseada.

Abstract

This poject, based on a previous design of the TFG by Javier López Martínez, seeks to build, improve and implement the design, and establish a new form of communication, which is carried out by an application installed on a mobile device, developed in the Android Studio design software. The communication is done via bluetooth, through which the HC-05 bluetooth module installed on the Arduino will receive the plans to be printed, the Arduino will interpret the code (G-code) sent translating it to the synchronized movement of four stepper motors, moving along the XY plane while tracing a trajectory that defines the desired image.

ÍNDICE GENERAL

Documento 1: Memoria

ÍNDICE

1. Introducción

1.1 Motivación

La motivación detrás del desarrollo de este proyecto radica en la aplicación práctica de los conocimientos adquiridos a lo largo del grado de electrónica industrial y automática de la universidad de Cantabria. La implementación de estos conocimientos para desarrollar un prototipo físico no solo representa un reto, sino también una oportunidad para aportar mi visión y nuevas ideas a un diseño previo. Además, la realización de este proyecto permitirá un enriquecimiento significativo de mis habilidades en electrónica y en el uso de programas complementarios. Este enfoque integral de juntar lo aprendido en un solo proyecto, no solo fortalecerá mi comprensión teórica, sino que también proporcionará una experiencia práctica valiosa.

1.2 Objetivo

El objetivo es realizar un prototipo físico funcional y portátil de bajo coste, que permita la impresión de imágenes sencillas en superficies planas de gran tamaño. De esta forma vamos a poder imprimir imágenes (planos, señales…) sin la necesidad de depender de una impresora convencional, pues vamos a eliminar la restricción del tamaño de papel, habitual en las impresoras comunes. Se pretende que colocando el robot sobre una superficie plana este obtenga unas instrucciones vía bluetooth, desde un smartphone y plasme sobre dicha superficie el diseño deseado. Se ha mejorado el sistema de comunicación con la placa Arduino respecto al anterior diseño, pues antes era necesario tener el prototipo conectado a un ordenador, lo que limitaba su ventaja de portabilidad.

Plasmar la imagen deseada será posible gracias al movimiento coordinado de cuatro motores paso a paso instalados en cada una de las ruedas omnidireccionales y a un servomotor que acciona el útil de escritura, todo esto alimentado por una batería recargable, con tres salidas, que suministra de 5 V, a los cuatro motores, y al Arduino. Este último a su vez alimenta, el módulo bluetooth y el servomotor, que requieren menos voltaje. Se ha realizado el cuerpo del robot mediante impresión 3D, dando pie a diseñar el robot de forma que utilice menos material que el anterior diseño.

2. Contexto

En este apartado vamos a introducir los métodos de impresión actuales y parte de su historia, luego vamos a tratar la historia de la robótica y una forma en la que se estas dos tecnologías se han complementado.

2.1 Impresión

Conocemos la impresión como la acción y efecto de imprimir, este proceso ya sea en papel u otra materia, consiste en la producción de textos, imágenes y formas físicas, se realiza mediante una máquina, que variará en forma y método de impresión en función de lo que se quiera imprimir. Se puede realizar de forma doméstica, artesanal, comercial o industrial a gran escala, y es una parte esencial de la edición de libros y toda clase de publicaciones impresas. Las nuevas técnicas de impresión en 3D ofrecen nuevas aplicaciones en diversos campos. A continuación, veremos los métodos de impresión.

2.1.1 Métodos de impresión 2D

Impresión Inkjet

La impresión inkjet es una tecnología de impresión, de textos e imágenes, sin contacto que consiste en una proyección controlada de minúsculas gotas de tinta sobre el sustrato a imprimir, que va formando la imagen o texto deseado, enviadas desde el soporte informático al dispositivo de impresión.

Impresión Laser

Este método está basado en la tecnología láser y se utiliza para la obtención de texto e imágenes. En este se utiliza un haz de luz, que se proyecta hasta un tambor fotoconductor que ioniza en determinadas zonas, atrayendo el polvo de “tinta seca” o “toner”. Al entrar en contacto con el papel, solo quedan impregnadas las zonas ionizadas, dando así lugar al documento que se haya enviado el usuario a la impresora. Para imprimir imágenes a color será necesario instalar cuatro “toners”, uno para cada color base (CMYK).

Impresión flexográfica

El sistema de impresión flexográfica es un sistema directo y principalmente rotativo con rodillos y material en bobinas de distintos tamaños en el cual se emplean planchas flexibles con altos relieves y una estructura principal formada por rodillos. La principal ventaja de la impresión flexográfica es la capacidad de imprimir en distintos sustratos.

Impresión por Sublimación

La impresión por sublimación es el proceso de transferencia de tintes de sublimación a un papel de transferencia que luego se sublima en otro material por transferencia de calor. La impresión por sublimación es un proceso químico que funciona a través de una impresora de sublimación que toma el diseño y lo imprime usando tinta de sublimación transferida a papel de sublimación. El artículo impreso se coloca en la máquina, donde el diseño se imprimirá en el objeto a través de una prensa de calor.

2.1.2 Métodos de impresión 3D

Impresión por deposición fundida (Método selecionado para el prototipo)

En este método (FDM) se deposita progresivamente un material fundido capa a capa, este material depositado es calentado a lo largo de una trayectoria de extrusión, de esta forma se irá formando las capas de las piezas. En este trabajo se ha elegido este como el método mediante el cual se va a formar el cuerpo del robot. Esta elección se ha tomado por las ventajas que representa, para un trabajo como este, respecto a los demás métodos que veremos a continuación. Destaca del resto por la gran selección de materiales para imprimir, su precisión, la posibilidad de imprimir piezas grandes y su coste competitivo. Las piezas de FDM son excelentes para el uso final, la producción de bajo volumen y la creación rápida de prototipos.

Impresión por sintrerizado por láser

El sinterizado selectivo por láser (SLS) es una de las tecnologías de impresión 3D más populares que produce piezas muy precisas y duraderas que son perfectas para el uso final, la producción de bajo a medio volumen o para la creación rápida de prototipos. En este método un láser de alta potencia dibuja cada capa en un lecho de polvo, normalmente de nailon. El láser sintetiza las partículas formando así estructuras sólidas. Cuando se termina una capa, la placa de construcción baja y un “recubridor” de polvo distribuye más polvo sobre la capa anterior, así hasta terminar las capas de la pieza y retirar el polvo sobrante, que será reutilizado.

Estereolitografía

La estereolitografía produce piezas extremadamente precisas y de alta resolución, capaces de ser utilizadas directamente en el uso final, en la producción de bajo volumen o en la creación rápida de prototipos. Ofrece una mayor resolución de impresión que muchas otras tecnologías de impresión 3D, lo que permite imprimir piezas con detalles finos y acabados superficiales. Este método usa una laser ultravioleta y una cuba de resina para construir las piezas. El haz de luz traza los patrones, por capas, en la superficie de la resina líquida. Esta exposición de la resina al laser cura la resina, la solidifica y la une a la capa inferior. Tras la impresión las piezas se elevan de la cuba y se acaban de curar en un horno de luz ultravioleta.

Multi jet fusion

La Multi Jet Fusion (MJF) es conocida por su rapidez y su excelente acabado superficial, según un informa técnico de HP se determinó que estas máquinas son hasta 10 veces más rápidas que el resto, aunque esto no ha sido verificado por entidades independientes. También es capaz de imprimir en múltiples materiales a la vez. En este caso, se utiliza una fuente de energía de infrarrojos, combinada con un agente de fusión para producir cada una de las capas de la pieza. Primero, un “recubridor” de polvo esparce una fina capa de polvo en la placa de construcción y los cabezales de tinta de la impresora pueden empezar a imprimir la primera capa. Esto se hace inyectando de forma selectiva un agente de fusión (dicho de otra forma, una especie de cola) sobre el polvo, y con cada pasada se dibujan las capas de las piezas, hasta que todas las capas estén completas.

2.2 Robótica

La robótica es, citando la definición de la Administración Nacional de Aeronáutica y el Espacio (NASA), es el estudio de los robots, entendiendo a estos como máquinas que pueden emplearse para realizar trabajos humanos, habiendo algunos que pueden hacerlo por su propia cuenta y otros que requieren de una persona que les indique previamente lo que hacer.

La robótica surge a a partir del trabajo multidisciplinar de la ciencia, la ingeniería y la tecnología, donde se trabaja con diferentes ramas de cada disciplina pues dentro del diseño de un robot hay un diseño mecánico, desarrollo, programación, producción y aplicación.

2.2.1 Grados de libertad

Una de las características más generales de un robot y que más se utiliza en la robótica, son los grados de libertad. El número de grados de libertad define la cantidad de partes independientes de un robot o máquina. Entre los grados de libertad distinguimos dos posibles movimientos: de traslación y rotación. Los movimientos de traslación se refieren a la capacidad de una figura de desplazarse de forma lineal, mientras que el movimiento rotativo hace referencia a un movimiento giratorio de un objeto en torno a su propio eje.

En robótica, el número de grados de libertad sirve para designar las habilidades motrices de los robots y los androides y está ligado a la cantidad de articulaciones y ejes de movilidad de un robot. En la ilustración 1, se muestran los diferentes tipos de conexiones y los grados de libertad que proporciona cada una.

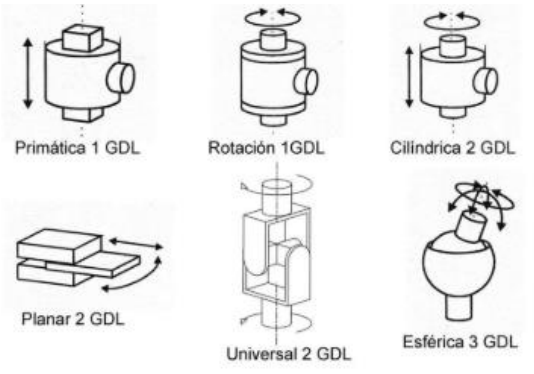


Ilustración 1 Tipos de articulaciones y sus grados de libertad

2.2.2 Clasificación de robots según su estructura

Una alternativa para clasificar y distinguir entre los diferentes tipos de robots es basarse en sus características de diseño, la estructura que presentan y la forma en la que operan. Esta clasificación permite identificar con mayor precisión las capacidades y aplicaciones específicas de cada tipo de robot, facilitando así su estudio y utilización en distintos ámbitos tecnológicos e industriales.

Robots móviles

Se trata esencialmente de máquinas automáticas capaces de desplazarse ampliamente, pudiendo realizar las funciones para las que están programados con autonomía en diversos entornos, hacen uso de sistemas de telemando, información del entorno, o movimientos previamente establecidos por el operador del robot. Están propulsados gracias a un sistema de motores y ruedas, las cuales dependiendo de la tarea para la que esté diseñado el robot, pueden varias de tradicionales a omnidireccionales que permiten un movimiento en cualquier dirección, o ruedas “mecanum” que proporcionan una maniobrabilidad excepcional. En el sector industrial, estos robots aseguran el transporte de piezas u objetos desde un punto a otro en una cadena de fabricación, automatizando procesos con enorme eficiencia.

El robot de este trabajo podría ser un ejemplo. Se desplaza gracias a unos motores con unas ruedas omnidireccionales que siguen una trayectoria con la finalidad de plasmar una figura sobre una superficie plana.

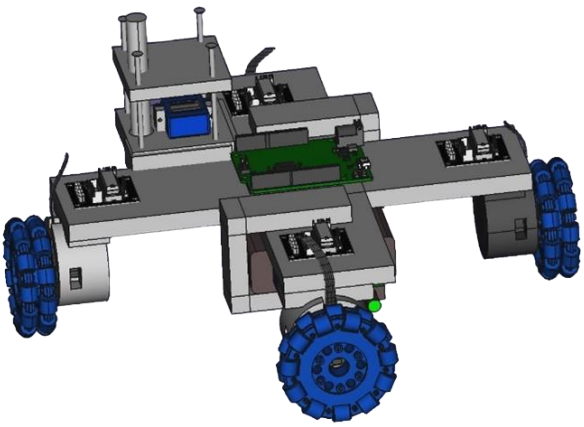


Ilustración 2 Robot móvil para impresión de documentos

Robots androides y ginoides

Los Robots antropomorfos, diseñados para tratar imitar la forma y cinemática del cuepo humano, se construyen para facilitar la interacción con personas y adaptarse a entornos humanos sin necesidad de modificaciones significativas. Son utilizados en diversas áreas como asistencia en el hogar, atención médica, educación, entretenimiento, y exploración. En la industria, trabajan junto a humanos en fábricas, realizan tareas repetitivas o peligrosas, y colaboran en la logística de almacenes, manejando herramientas y equipos diseñados para humanos, lo que mejora la eficiencia y seguridad en el entorno industrial. Un ejemplo de un robot antropomorfo utilizado en la industria es el RoboThespian, un robot humanoide interactivo que se utiliza en entornos industriales y comerciales para demostraciones, educación y entretenimiento, diseñado por Engineered Arts.

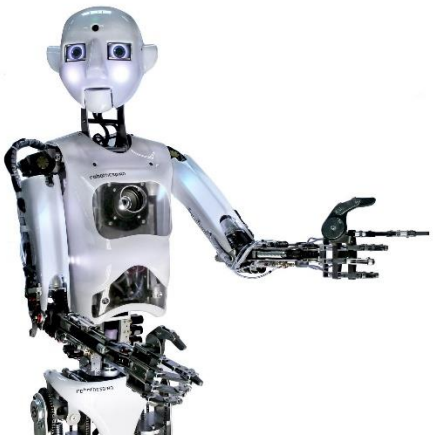


Ilustración 3 RoboThespian

Robots zoomórficos

Son robots que han sido físicamente diseñados con una estructura que imita a las de animales. Estos se caracterizan especialmente por buscar replicar la cinemática de determinadas especies y sus habilidades motora incluyendo aspectos como la flexibilidad, la eficiencia energética y la adaptabilidad a diferentes terrenos.

Dentro de esta categoría también suelen distinguirse otros dos tipos: los robots zoomórficos que caminan y los que no. Los del primer tipo presentan un desarrollo sensiblemente mayor y tienen, además, una proyección muy amplia, con un alto

potencial para su aplicación en la exploración de lugares como el espacio exterior, las profundidades marinas o los volcanes.

El robot ANYmal de ANYbotics es un excelente ejemplo de cómo la biocinética animal se aplica en la industria ya que tiene la capacidad de navegar a través de entornos difíciles como plantas de energía nuclear o centrales eléctricas, donde se requiere una inspección detallada pero puede ser peligroso o difícil para los humanos acceder.

Ilustración 4 Robot ANYmal

Robots poli articulados

A este tipo se le asocia una amplia gama de robots diseñados para realizar tareas repetitivas eficientemente. Estos comparten ciertas características como una posición estática y una estructura ideada para mover únicamente sus elementos terminales, en un espacio de trabajo limitado y según uno o más sistemas de coordenadas, así como un reducido nivel de libertad y autonomía.

En este grupo se encuentran, por ejemplo, los robots manipuladores, los robots industriales y los robots cartesianos, los cuales suelen aplicarse sobre todo cuando es preciso abarcar una zona de trabajo muy amplia o alargada, actuar sobre objetos con un plano simétrico vertical o bien reducir el espacio que se utiliza en el suelo, así como en la automatización del trabajo, ya que aumentan notablemente la productividad.

Una buena representación de este grupo es el UR10e de Universal Robots. Un brazo robótico flexible y colaborativo que cuenta con seis grados de libertad, lo que le permite realizar una amplia gama de tareas en entornos industriales.

Ilustración 5 Robot UR10e

Robots híbridos

Finalmente, existen muchos tipos de robots que comparten características de cada una de estas clasificaciones, por lo que se los suele considerar como robots híbridos. Un claro ejemplo podría ser el de los rover desarrollados por el Jet Propulsion Laboratory de la NASA para la exploración espacial.

Ilustración 6 Robot rover de la nasa

2.3 Comunicación inalámbrica de corto alcance

La comunicación inalámbrica de corto alcance se utiliza para la trasmisión de datos sin cables dentro de un rango limitado. Este tipo de comunicación es ampliamente utilizada

aportando gran valor en la conectividad y en el Internet de las Cosas (IOT), que es un sistema de dispositivos interconectados que pueden comunicarse y compartir datos entre sí de forma inalámbrica, permitiendo la automatización y el control remoto de diversas funciones.

Esta tecnología, la comunicación inalámbrica de corto alcance, es ampliamente utilizada en sistemas que necesitan intercambiar información. Estos métodos buscan la eficiencia y optimización de los recursos que emplean, ofreciendo una alta velocidad de transferencia y generalmente abarcando un radio de 100 metros.

Esta tecnología ha sido utilizada en este trabajo por las ventajas que presenta de portabilidad y versatilidad en el movimiento, pues así el prototipo es capaz de moverse, mientras recibe información, sin las restricciones físicas de un cable.

Bluetooth

La tecnología Bluetooth nació de la necesidad de las empresas de telecomunicaciones de encontrar una a interfaz abierta y de bajo coste para facilitar la comunicación entre dispositivos sin utilizar cables. Bluetooth es una tecnología inalámbrica de muy corto alcance diseñada para permitir la comunicación entre dispositivos como computadoras, sistemas de entretenimiento y otros dispositivos electrónicos sin el uso de cables ni conectores.

Bluetooth es una tecnología inalámbrica que opera en la banda de frecuencia de uso general sin licencia (ISM) de 2.4 GHz y cumple con normas internacionales de hasta 20 dBm. Se conecta a dispositivos mediante puertos en placas del sistema, USB o tarjetas de PC. En una piconet, una red temporal en la que los dispositivos los dispositivos se comunican directamente entre sí sin necesidad de un punto de acceso central (Ad Hoc), un dispositivo actúa como maestro y los demás como esclavos, sincronizándose con el reloj maestro y la secuencia de salto de frecuencias. Las scatternets combinan múltiples piconets, permitiendo la interconexión de redes.

Los dispositivos reciben una dirección de miembro activo (AMA) de 3 bits al unirse a una piconet y, si hay ocho dispositivos activos, uno se estaciona y obtiene una dirección de miembro pasivo (PMA) de 8 bits, permitiendo que más dispositivos se conecten. Bluetooth puede interconectarse con redes cableadas a través de dispositivos de acceso tipo puente.

El salto de frecuencia distribuye la señal en 79 canales de 1 MHz para minimizar interferencias. La comunicación entre dispositivos es automática cuando están dentro

del alcance. Las tramas multiranura permiten mayores velocidades de datos, con paquetes de una ranura alcanzando hasta 172 kbps y tramas de cinco ranuras hasta 721 kbps.

En resumen esta tecnlología permite que los dispositivos se comuniquen entre sí, sincronicen datos entre sí y se conecten sin necesidad de usar cables. Para agregar la funcionalidad Bluetooth a una computadora un dispositivo host, se puede instalar una radio Bluetooth y un controlador de banda base en un dispositivo que se vincula a un puerto integrado en una placa del sistema, un puerto de bus serie universal (USB) u otro puerto compatible. Estos componentes se muestran en la ilustración 7.

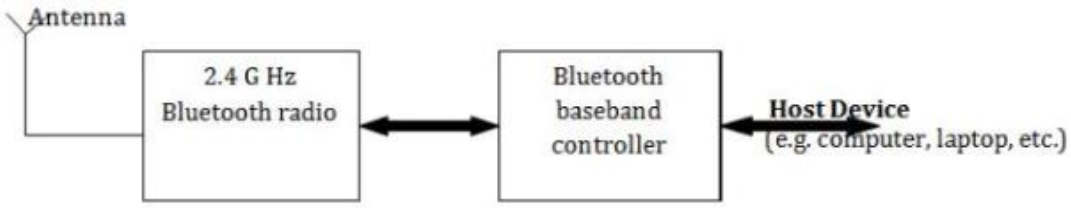


Ilustración 7 Componentes de Bluetooth

Wi-fi Direct

La tecnología Wi-Fi Direct permite que dispositivos Wi-Fi se conecten de manera directa, permitiendo realizar acciones como por ejemplo sincronizar, imprimir, compartir archivos o internet. Con el término de manera directa o Peer to Peer (P2P) se quiere decir que, a diferencia de una conexión Wi-Fi normal, Wi-Fi Direct no necesita un punto de acceso intermedio para la gestión de la comunicación entre los dispositivos, sino que uno de estos dispositivos funcionará como punto de acceso.

Wi-Fi Direct es una tecnología que permite conexiones inalámbricas directas entre dispositivos sin necesidad de un punto de acceso a Internet. Admite velocidades de hasta 250 Mbps, ofreciendo un gran rendimiento para la transferencia de contenido multimedia. Su capacidad de cobertura alcanza hasta 200 metros, permitiendo conexiones tanto a corta como a larga distancia. Aunque una red Wi-Fi Direct puede formarse entre dos o más dispositivos, el número de dispositivos soportados en un grupo es menor que en un punto de acceso tradicional, y no todos los dispositivos certificados pueden conectarse a múltiples dispositivos. Integrada en dispositivos Android 4.0 desde 2011, esta tecnología es compatible con versiones posteriores gracias a la API de Wi-Fi Peer to Peer. Además, soporta el descubrimiento de servicios en la capa de enlace, permitiendo a los dispositivos intercambiar solicitudes para conocer los servicios

disponibles antes de formar un Grupo P2P. Wi-Fi Direct amplía el uso de la tecnología Wi-Fi más allá de la conexión a Internet, permitiendo aplicaciones como compartir contenido, sincronizar datos, jugar y reproducir audio y vídeo directamente entre dispositivos sin necesidad de estar conectados a la red.

NFC (Near Field Communication)

La tecnología NFC, es una tecnología de comunicación inalámbrica de corto alcance que permite el intercambio de datos entre dispositivos. Al estar basada en la tecnología RFID (Identificación por Radiofrecuencia), el NFC es capaz de establecer una comunicación entre dos dispositivos simplemente acercándolos a una distancia de 4 cm o menos. Esta tecnología ayuda a convertir en una realidad el fenómeno conocido como Internet de las cosas (IoT), al permitir una conexión fácil y segura entre objetos.

El NFC funciona gracias a un principio físico conocido como inducción electromagnética. Cuando dos dispositivos equipados con NFC se encuentran a una distancia de aproximadamente yubos 4cm como mínimo, uno de ellos, conocido como el iniciador genera un campo electromagnético de baja frecuencia. El otro dispositivo conocido como el objetivo y es el que recibe este campo y puede extraer la información codificada que se encuentra en él. El intercambio de información de esta tecnología se puede dar de dos formas, Activo o Pasivo. En el modo activo los dos dispositivos se comunican produciendo una señal, el proceso trata en que uno de los dos apague si campo mientras está recibiendo, en este modo se permite una comunicación bidireccional. En el modo pasivo el emisor genera un campo electromagnético y el receptor lo modula. En otras palabras, el dispositivo receptor no necesita su propia fuente de energía, ya que se alimenta de la intensidad del campo electromagnético generado por el emisor. Esto permite que el receptor funcione con energía mínima, obtenida directamente del campo del emisor. Este modo solo permite una comunicación omnidireccional.

Infra red (IR)

La luz comunicación por infrarrojos para transmisión de datos es una tecnología versátil que ha encontrado numerosas aplicaciones en diversas industrias. Desde la transferencia de datos entre dispositivos móviles hasta la comunicación inalámbrica en

controles remotos, la luz infrarroja ha simplificado y mejorado la forma de comunicación y transmisión de información. Una de las dificultades de este método es, que por la falta de verificación en sus protocolos de trasmisión, provocando interferencias en las comunicaciones o comunicaciones no deseadas.

Para transmitir datos binarios con un LED IR, se enciende para enviar un '1' lógico y se apaga para un '0' lógico, pero para evitar interferencias, se usa una frecuencia portadora, comúnmente 38 kHz.

El protocolo NEC, utilizando 38 kHz, codifica un '1' lógico con una ráfaga de 562.2 µs seguida de un período bajo de 1.687 ms, y un '0' lógico con una ráfaga de 562.2 µs seguida de un período bajo de 562.2 µs. Un mensaje incluye un pulso de inicio de 9 ms, un período bajo de 4.5 ms, 8 bits de dirección y comando, sus inversos lógicos, y una ráfaga final de 562.5 µs.

El protocolo RC5 de Philips, con 36 kHz y codificación Manchester, define un '1' lógico con un período bajo de 889 µs seguido de una ráfaga de 889 µs, y un '0' lógico con una ráfaga de 889 µs seguida de un período bajo de 889 µs. Un mensaje incluye dos bits iniciales altos, un bit de conmutación, cinco bits de dirección y seis bits de comando, y tarda 24.892 ms en transmitirse

3. Construcción del prototipo

En este apartado se tratarán los componentes que forman parte del robot, su funcionamiento y por qué han sido seleccionados frente a otras opciones. También se compararán la versión actual de ciertos componentes con la del anterior diseño,

explicando el motivo de su cambio. Como se puede ver en la ilustración cada apartado está asociado a cada uno de los componentes. 3.1 Módulo de Comunicación, 3.2 Batería, 3.3 Microprocesador, 3.4 Ruedas del Robot, 3.5 Servomotor, 3.6 Módulo de control de motores, 3.7 Chasis del robot y 3.8 Motores.

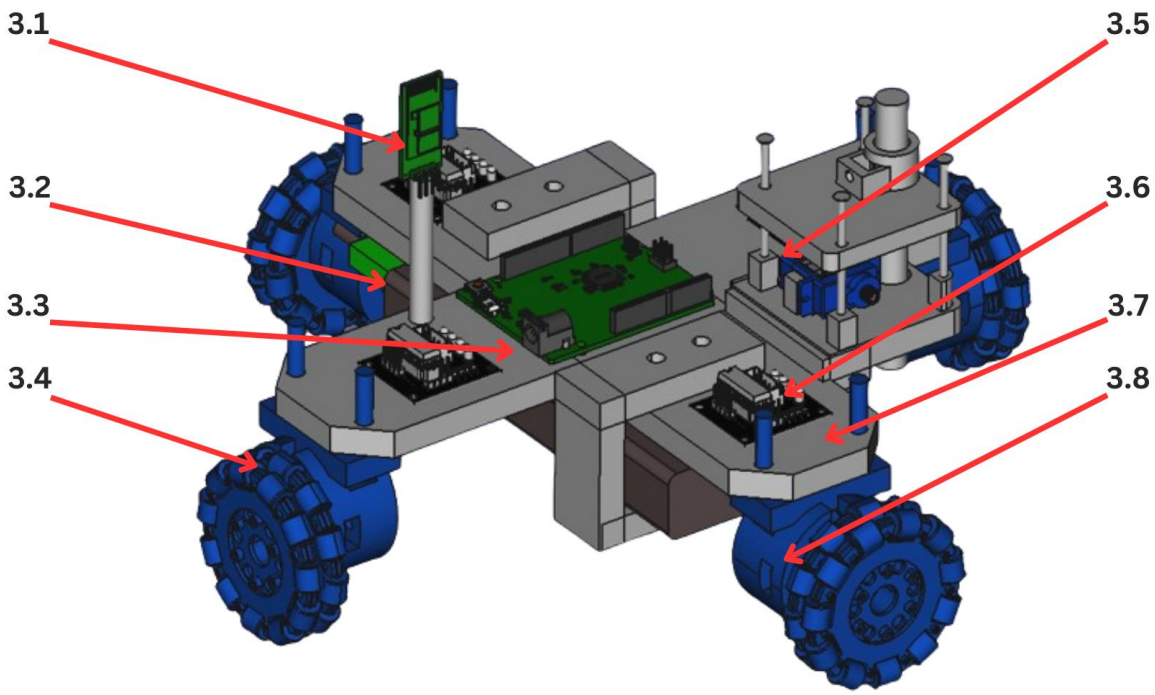


Ilustración 8 Componentes del Robot

3.1 Módulo de comunicación

El dispositivo seleccionado, HC-05 es un módulo de protocolo de puerto serie Bluetooth, utilizado para establecer una comunicación inalámbrica entres dispositivos. Es ampliamente utilizado en el ámbito de la electrónica y la robótica. Ha sido diseñado para ser utilizado con microcontroladores como Arduino y Raspberry Pi, permitiendo la transferencia inalámbrica de datos.

Una de las características principales de este dispositivo y que le distinguen de otros de su clase es que se puede utilizar como maestro y esclavo, esta dualidad le hace un componente muy versátil. En modo maestro, puede iniciar una comunicación con otros dispositivos. En este proyecto este módulo está siendo usado en modo esclavo, espera

a que otro dispositivo se conecte a é. En el caso de este trabajo ese dispositivo será el smartphone que tenga instalada la aplicación diseñada para este trabajo.

El dispositivo opera en la banda (ISM) de 2.4GHz y su alcance efectivo puede llegar hasta los 10 metros, la velocidad de trasmisión es configurable y va de los 1200 Baudios a los 1382400 Baudios.

Es un módulo configurable mediante Attention comands (AT), con estos puedes cambiar los parámetros del dispositivo. Esta configuración se hace a través de la configuración serie y utilizando un software terminal para enviar comando AT.

En la parte del hardware el módulo HC-05 cuenta con varios pines destinados a diferentes funciones, tanto para la entrada y salida de datos como para el control y la configuración del dispositivo. Los pines más importantes y comúnmente utilizados incluyen el pin VCC, que se conecta a la fuente de alimentación del módulo y opera típicamente entre 3.3V y 5V, y el pin GND, que se conecta a tierra. Para la comunicación serial, el pin TXD envía datos desde el módulo al dispositivo conectado, mientras que el pin RXD recibe datos del dispositivo hacia el módulo HC-05. Otro pin relevante es el pin STATE, que indica el estado de conexión del módulo Bluetooth, y el pin EN (o KEY), que se utiliza para poner el módulo en modo de configuración de comandos AT cuando está en estado alto. El módulo incluye un LED interno que parpadea cuando no está conectado y permanece encendido cuando la conexión está establecida. Además de

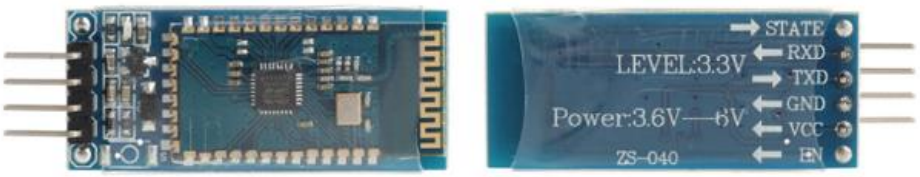


Ilustración 9 HC-05

estos pines esenciales, el HC-05 tiene otros pines internos y menos comunes, como los pines PIO (Programmable Input/Output), que son programables y pueden ser utilizados para diferentes propósitos según la configuración del módulo, y los pines AIO (Analog Input/Output), que permiten lecturas o salidas analógicas, aunque no son comúnmente usados en aplicaciones típicas. También existen pines relacionados con la interfaz SPI (SPI_CLK, SPI_MISO, SPI_MOSI, SPI_CSN), que se utilizan principalmente durante la programación y configuración avanzada del módulo. En la mayoría de las aplicaciones estándar, se utilizan únicamente los pines VCC, GND, TXD, RXD y ocasionalmente el

pin EN para la configuración mediante comandos AT, mientras que los otros pines suelen estar presentes en el módulo pero no se exponen en las versiones más comunes.

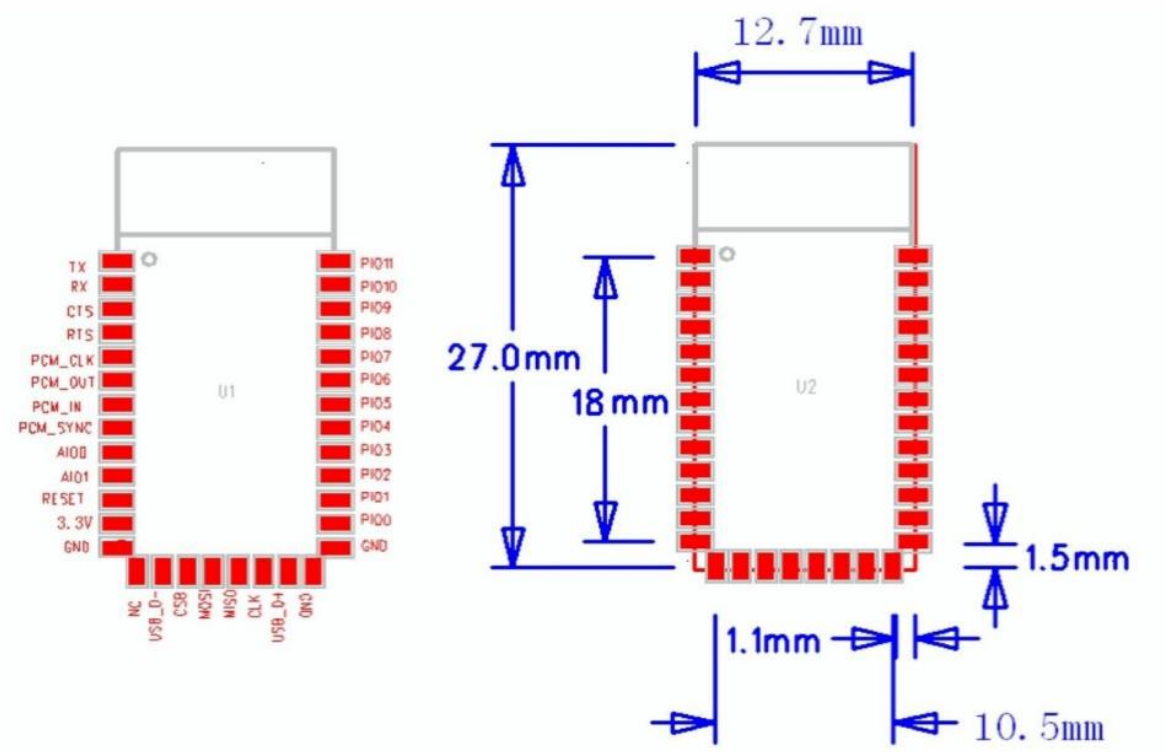


Ilustración 10 Esquema de Pines de HC-05

PIN Name	PIN #	Pad type	Description	Note
GND	13 21 22	VSS	Ground pot	
3.3 VCC	12	3.3V	Integrated 3.3V (+) supply with On-chip linear regulator output within 3.15-3.3V	
AIO0	9	Bi-Directional	Programmable input/output line	
AIO1	10	Bi-Directional	Programmable input/output line	
PIO0	23	Bi-Directional RX EN	Programmable input/output line, control output for LNA(if fitted)	
PIO1	24	Bi-Directional TX EN	Programmable input/output line, control output for PA(if fitted)	

PIO2	25	Bi-Directional	Programmable input/output line	
PIO3	26	Bi-Directional	Programmable input/output line	
PIO4	27	Bi-Directional	Programmable input/output line	
PIO5	28	Bi-Directional	Programmable input/output line	
PIO6	29	Bi-Directional	Programmable input/output line	
PIO7	30	Bi-Directional	Programmable input/output line	
PIO8	31	Bi-Directional	Programmable input/output line	
PIO9	32	Bi-Directional	Programmable input/output line	
PIO10	33	Bi-Directional	Programmable input/output line	
PIO11	34	Bi-Directional	Programmable input/output line	
RESETB	11	CMOS input with weak internal pull-up	Reset if low.input debouncde so must be low for >5MS to cause a reset	
UART_RTS	4	CMOS output, tri-stable with weak internal pull-up	UART request to send, active low	
UART_CTS	3	CMOS input with weak internal pull-down	UART clear to send, active low	
UART_RX	2	CMOS input with weak internal pull-down	UART Data input	
UART_TX	1	CMOS output, Tri-stable with weak internal pull-up	UART Data output	
SPI_MOSI	17	CMOS input with weak internal pull-down	Serial peripheral interface data input	
SPI_CSB	16	CMOS input with weak internal pull-up	Chip select for serial peripheral interface, active low	
SPI_CLK	19	CMOS input with weak internal pull-down	Serial peripheral interface clock	
SPI_MISO	18	CMOS input with weak internal pull-down	Serial peripheral interface data Output	
USB_-	15	Bi-Directional		
USB_+	20	Bi-Directional		
NC	14			
PCM_CLK	5	Bi-Directional	Synchronous PCM data clock	
PCM_OUT	6	CMOS output	Synchronous PCM data output	
PCM_IN	7	CMOS Input	Synchronous PCM data input	
PCM_SYNC	8	Bi-Directional	Synchronous PCM data strobe	

Ilustración 11 Tabla de Pines del Módulo HC-05

En el trabajo los pines utilizados son los presentes en la siguiente figura donde también se puede observar cómo es el esquema de conexiones al Arduino.

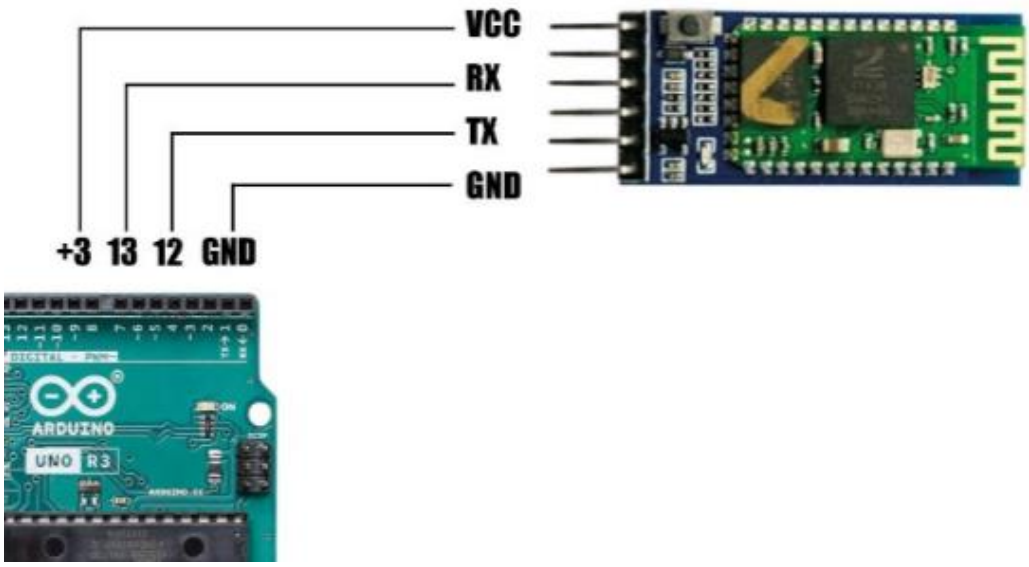


Ilustración 12 Conexiones HC-05

El módulo HC-05 ha sido elegido sobre otros dispositivos similares debido a su capacidad para permitir la comunicación bidireccional, lo que facilita un intercambio de datos eficiente entre dispositivos. Su excelente tiempo de transmisión, bajo coste, y accesibilidad lo hacen especialmente. Además, su compatibilidad con como Arduino, junto con su tamaño versátil, lo convierte en una opción ideal para el proyecto. Además, en comparación con el diseño anterior, que utilizaba el software CoolTerm, permitiendo aumentar la portabilidad, la accesibilidad y eliminar los cables en el proceso de impresión. Con el anterior Software era necesario tener conectado el prototipo a un ordenador mediante un cable que enviaba el Gcode de la imagen a imprimir. El cambio de la forma de trasmisión de datos a una de bluetooth es lo que permite la opción de programar una aplicación para un smartphone así mejorando la portabilidad y la accesibilidad, pues se pone a disposición un equipo tan común como un teléfono móvil.

3.2 Fuente de alimentación

Para la alimentación del proyecto se ha reutilizado una batería QTshine que suministra energía directa a cuatro motores paso a paso y al microcontrolador Arduino. Esta elección se fundamenta, en que es reutilizada lo que barata el coste y en sus

características que la hacen muy compatibles con el proyecto, además destacar que es recargable lo que mejora el coste del producto y lo hace más sostenible. Las características en las que destaca, es en las cuatro salidas independientes y en la cantidad de voltios y amperios que suministra cada una de estas.

La batería externa QTshine cuenta con una capacidad ultra grande de 36800 mAh. Esta capacidad es crucial para mantener la operatividad de los motores y otros componentes del robot, eliminando preocupaciones sobre la baja duración de la batería o la falta de potencia suministrada. Además, este cargador portátil incluye cuatro puertos de salida: un puerto USB-C (5 V/3A) y tres puertos micro USB (5 V/2,1 A), además de dos entradas (Micro USB y USB-C). Esta configuración permite la carga simultánea de hasta cuatro dispositivos.

Ilustración 13 Batería QTshine

La batería QTshine está equipada con un sistema de seguridad múltiple que ofrece protección contra sobrecargas, sobre descargas, sobretensiones, sobre corrientes y cortocircuitos. Esta protección integral es esencial para mantener la seguridad y la fiabilidad del sistema, especialmente durante operaciones prolongadas. Además, cuenta con cuatro indicadores LED azules que muestran la potencia restante en intervalos de 25%, 50%, 75% y 100%. Esto facilita el monitoreo del estado de carga de la batería.

Se ha seleccionado esta batería frente a las 4 pilas de 5V propuestas en el primer diseño, por su mayor potencia y las tres salidas que presenta, estas salidas son necesarias, pues al conectar a una sola fuente con una sola salida a todos los componentes la intensidad no era suficiente para hacer funcionar a todos los componentes. Además, cabe destacar su capacidad de recarga que aunque aumente

un poco el coste en un inicio, acabará siendo rentable después de que se necesite una recarga del dispositivo.

3.3 Microprocesador

El microprocesador de este proyecto es el Arduino UNO R3, por diversos motivos. El principal es su capacidad para controlar los motores y a la vez de programación para interpretar el Gcode, además el código original del proyecto está escrito en este lenguaje y para dicho microprocesador utilizando los repositorios concretos de este. La versatilidad y la cantidad de accesorios con los que cuenta y con los que es compatible, también lo hace ideal para esta función. El modelo Arduino Uno también tiene un tamaño y una cantidad de salidas adecuadas para este proyecto, hay versiones de Arduino que tiene menos salidas como el Arduino NANO, pero este no tiene suficientes para ser compatible con el proyecto. O que por el contrario que tiene más, como el Arduino MEGA, teniendo en cuenta que este trabajo trata de abaratar costes y la portabilidad, la elección de un microprocesador más grande no nos traería ningún beneficio, además este componente aumentaría el coste del prototipo, ya que es más caro que el Arduino

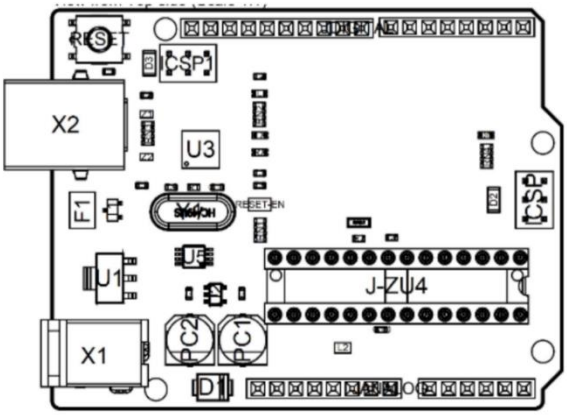


Ilustración 14 Esquema de Arduino UNO

UNO. También es más grande así que afectaría al tamaño del prototipo haciendo menos compacto.

El Arduino UNO es una placa microcontroladora basada en el ATmega328P. Tiene 14 pines digitales de entrada/salida (de los cuales 6 se pueden utilizar como salidas PWM), 6 entradas analógicas, un resonador cerámico de 16 MHz, una conexión USB, un conector de alimentación, un cabezal ICSP y un botón de reinicio.

Los pines del Arduino UNO tienen diversas funciones y características que permiten la conexión y el control de diferentes componentes electrónicos. La placa cuenta con un

total de 14 pines digitales, numerados del 0 al 13, que pueden configurarse como entradas o salidas mediante programación. Estos pines operan a 5V y pueden proporcionar o recibir un máximo de 40 mA por pin, con una resistencia interna de pull-up de 20-50 kOhms que se puede activar mediante software.

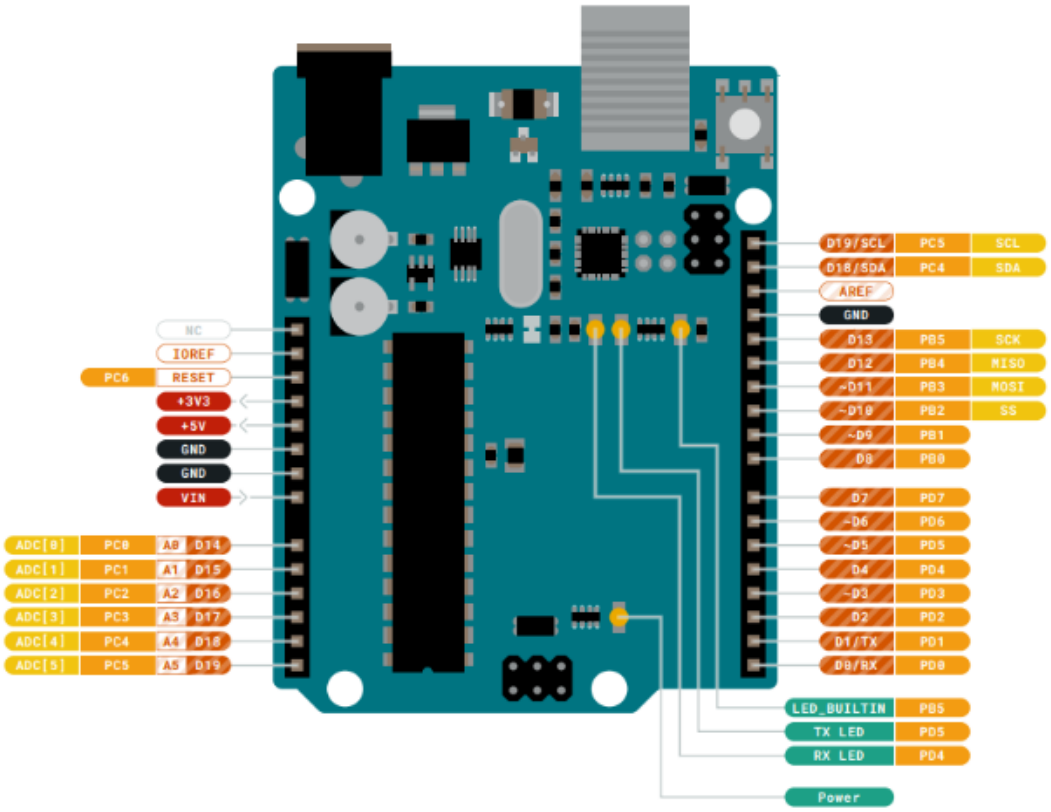


Ilustración 15 Pinout Arduino UNO

Además, de los 14 pines digitales, seis de ellos (3, 5, 6, 9, 10 y 11) tienen capacidades PWM (Modulación por Ancho de Pulso), lo que permite generar señales analógicas simuladas para el control de dispositivos como servomotores y luces LED. La placa también incluye seis entradas analógicas, numeradas de A0 a A5, que permiten medir voltajes variables entre 0 y 5V. Cada una de estas entradas analógicas tiene una resolución de 10 bits, lo que significa que puede representar el voltaje de entrada en 1024 niveles discretos.

El Arduino UNO dispone de pines de alimentación específicos, incluidos el pin de 5V, el pin de 3.3V y los pines GND (tierra). Estos pines se utilizan para suministrar energía a la placa y a los componentes conectados. Adicionalmente, el pin VIN permite suministrar un voltaje de entrada de 7-12V, que es regulado internamente para proporcionar la alimentación adecuada a la placa.

5.1 JANALOG

Pin	Function	Type	Description
1	NC	NC	Not connected
2	IOREF	IOREF	Reference for digital logic V - connected to 5V
3	Reset	Reset	Reset
4	+3V3	Power	+3V3 Power Rail
5	+5V	Power	+5V Power Rail
6	GND	Power	Ground
7	GND	Power	Ground
8	VIN	Power	Voltage Input
9	A0	Analog/GPIO	Analog input 0 /GPIO
10	A1	Analog/GPIO	Analog input 1 /GPIO
11	A2	Analog/GPIO	Analog input 2 /GPIO
12	A3	Analog/GPIO	Analog input 3 /GPIO
13	A4/SDA	Analog input/I2C	Analog input 4/I2C Data line
14	A5/SCL	Analog input/I2C	Analog input 5/I2C Clock line

5.2 JDIGITAL

Pin	Function	Type	Description
1	D0	Digital/GPIO	Digital pin 0/GPIO
2	D1	Digital/GPIO	Digital pin 1/GPIO
3	D2	Digital/GPIO	Digital pin 2/GPIO
4	D3	Digital/GPIO	Digital pin 3/GPIO
5	D4	Digital/GPIO	Digital pin 4/GPIO
6	D5	Digital/GPIO	Digital pin 5/GPIO
7	D6	Digital/GPIO	Digital pin 6/GPIO
8	D7	Digital/GPIO	Digital pin 7/GPIO
9	D8	Digital/GPIO	Digital pin 8/GPIO
10	D9	Digital/GPIO	Digital pin 9/GPIO
11	SS	Digital	SPI Chip Select
12	MOSI	Digital	SPI1 Main Out Secondary In
13	MISO	Digital	SPI Main In Secondary Out
14	SCK	Digital	SPI serial clock output
15	GND	Power	Ground
16	AREF	Digital	Analog reference voltage
17	A4/SD4	Digital	Analog input 4/I2C Data line (duplicated)
18	A5/SD5	Digital	Analog input 5/I2C Clock line (duplicated)

Ilustración 16 Tabla de Pines del Arduino UNO

El pin de reset del Arduino UNO se utiliza para reiniciar el microcontrolador, y hay un pin AREF (Referencia Analógica) que puede ser usado para establecer un voltaje de referencia externo para las entradas analógicas, lo que mejora la precisión de las mediciones analógicas en ciertos casos.

3.4 Ruedas del robot

Se trata de elementos adaptados para mover o girar en cualquier dirección paquetes cuya superficie sea suficientemente lisa, rígida y plana, empleadas esencialmente en los cruces de enlaces en los transportadores por gravedad o en las estanterías de trabajo y montaje en máquinas y transferencias.

La estructura, rotante por rozamiento en el eje principal, presenta una serie de pequeños rodillos con rotulas locas, montados en pivotes en acero inoxidable, desplazados entre ellos para garantizar la continuidad del contacto con la superficie de los paquetes en movimiento.

Construidos en Poliamida, resistentes a la corrosión y la abrasión, son muy ligeros, versátiles y fáciles de montar; además los tipos con hueco hexagonal permiten la motorización en instalaciones automáticas. Pueden ser admitidas cargas mayores de lo indicado para los diferentes tipos, no obstante para los paquetes con superficies rígidas (no cartón) el traslado se vuelve difícil.

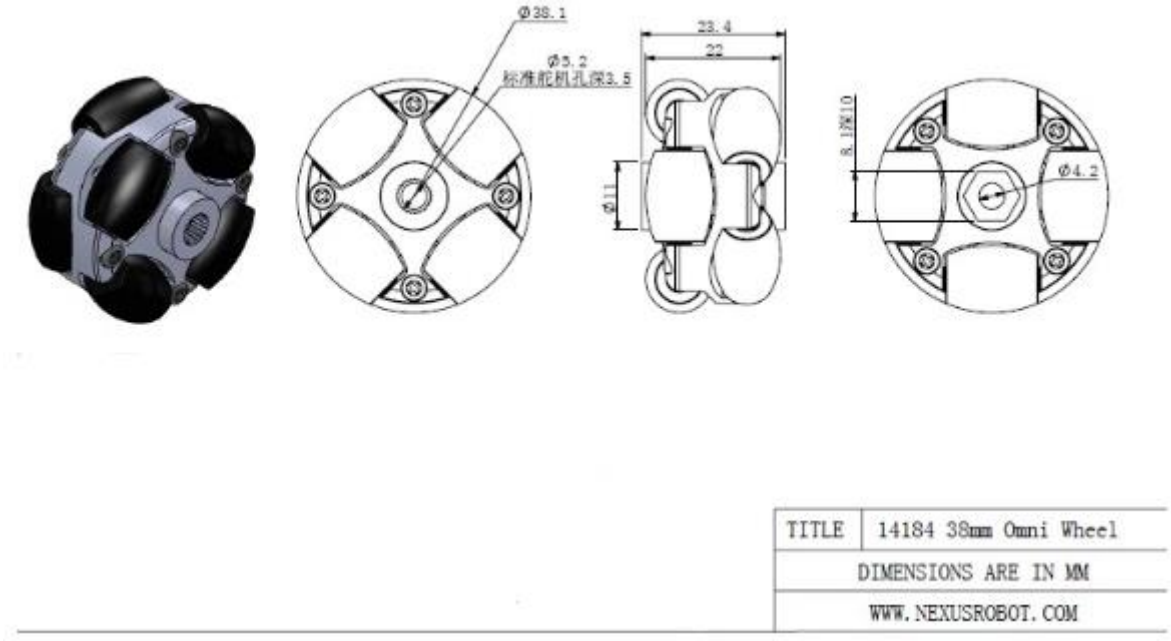


Ilustración 17 Planos de ruedas Robot

La rueda omnidireccional de plástico de 38mm (1,5 pulgadas) es la rueda omnidireccional más pequeña de todas las ruedas omnidireccionales Nexus. Es única, ya que pueden rodar libremente en dos direcciones. Y esta rueda Omni de 38mm tiene 8 Rodillos de PU. Sus rodillos son más duraderos, robustos y flexibles para proporcionar a la rueda un movimiento de 360 grados fácilmente. Esta rueda Omni de 38mm se aplica a los pequeños coches robot, plataformas robot, juguetes pequeños, etc.

Estas ruedas irán conectadas al step motor, tendremos 4 en el proyecto y son una gran ventaja a la hora de los movimientos que realiza el robot, pues no será necesario ruedas articuladas facilitando la programación del dispositivo, además de permitir movimientos que de otra forma serían mucho más complicados. Estas ruedas se han mantenido con

respecto al diseño orinal ya que, al ser tan especiales no hay muchas alternativas, ya que además el código está preparado para tratar con unas ruedas de este diámetro.

3.5 Servomotor

El servomotor es integrado en un mecanismo, es el encargado de bajar o levantar el útil de escritura, en otras palabras, es el encargado de controlar el grado de libertad del eje z.

Un servomotor es un motor eléctrico que incorpora un juego de engranes junto con un circuito electrónico de control, con la finalidad de poder controlar el giro de dicho motor y así poder elegir siempre la posición deseada. Normalmente estos dispositivos tienen un rango de operación que está limitado a la hora de su construcción, por lo general trabajan con 180° a 360º.

La mayoría de los servomotores están compuestos por cuatro partes. Un motor de corriente continua es él que proporciona el par y el movimiento. Los engranajes es una parte que encarga de regular la velocidad de giro del DC motor, además así también la relación de engranajes aumenta el par. Cuenta con un sensor de desplazamiento, para el que se utiliza un potenciómetro que se conecta al eje del motor para que gire junto con él y medir su cambio de resistencia para así poder calcular el giro. Para el control de todo se hace de un circuito de control, que es diseñado con una retroalimentación para poder controlar deseado el giro del motor.

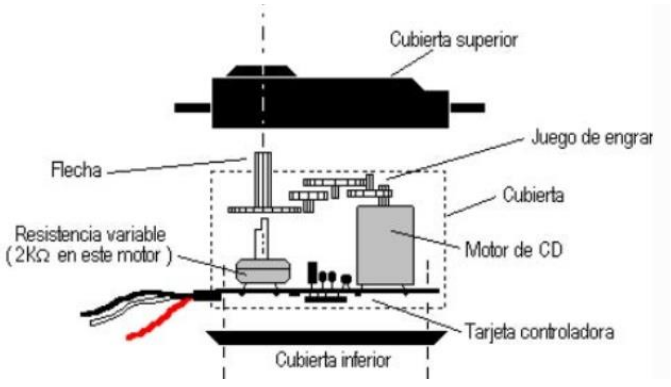


Ilustración 18 Componentes del Servo

La posición que toma el eje de los servomotores se controla mediante una señal eléctrica conocida como Pulse Wide Modulation (PWM). La mayoría de los servomotores cuentan con tres cables, uno para voltaje, otro para tierra y el tercero

se utiliza para recibir la señal PWM, que controla el giro. El control de estos motores es sencillo, pues funciona por norma general con ciclos de 50Hz, es decir, 20 milisegundos para mandar la instrucción. La posición del servomotor depende del ancho de pulso de la PWM, el control de esta señal suele venir dado por un microprocesador. En el caso del trabajo viene dado por un el Arduino UNO.

El servomotor utilizado para este trabajo es el Servomotor SG90, las características de tamaño compacto, disponibilidad y bajo precio han sido determinantes en la elección de este servomotor. El control en el eje Z por cómo está diseñado el robot, no tiene que ser muy preciso, ni mover muchos

Ilustración 19 Servomotor SG90

Las características principales de este servomotor se reflejan en la siguiente tabla.

Peso	9g
Dimensiones	22,2x11,8x31mm
Velocidad de trabajo	0,1s/60º
Tensión de trabajo	4,8~5V

Ilustración 20 Tabla de Características SG90

3.6 Drivers de los motores.

Los drivers del motor son un componente complementario a los motores paso a paso componente 3.8, que da el giro a las ruedas.

Un driver o controlador de motor es un dispositivo electrónico que se encarga de controlar la energía que se suministra al motor paso a paso. Su principal función es amplificar la señal de control de baja corriente proveniente de un microcontrolador o un

sistema de control, y convertirla en una señal de corriente más alta que pueda conducir el motor. En otras palabras, el controlador es el encargado de suministrar la corriente necesaria para que el motor se mueva según las instrucciones recibidas.

Los drivers para motores paso a paso son esenciales para garantizar un funcionamiento adecuado de los motores, ya que proporcionan una corriente estable y controlada, evitando fluctuaciones y protegiendo tanto al motor como al sistema en general.

El módulo controlador utilizado en este trabajo es ULN2003, este controlador es ampliamente utilizado y acostumbra a ir de la mano con el modelo de los motores paso a paso, que utiliza este trabajo. El ULN2003 es un circuito integrado muy versátil, compuesto internamente por 7 drivers idénticos e independientes entre sí. Cada driver está constituido por dos transistores en configuración Darlington. La configuración Darlington, ideada por Sidney Darlington en 1953, consiste en conectar dos transistores bipolares en cascada obteniendo así, una ganancia muy elevada ya que produce un efecto que multiplica la ganancia de cada uno de los dos transistores. Esta configuración permite controlar cargas de una cierta potencia con corrientes de entrada muy pequeñas.

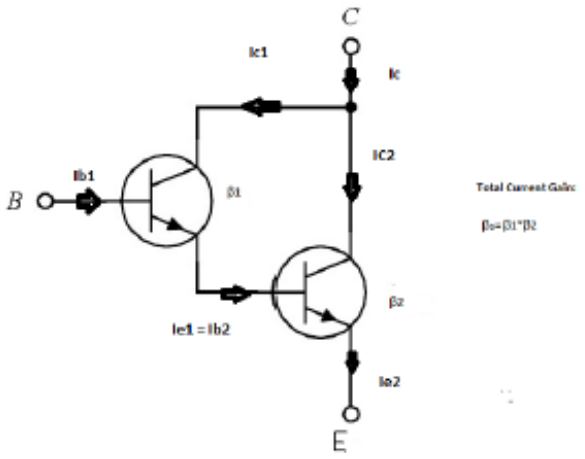


Ilustración 21 Configuración en Darlington

Las características principales del ULN2003 quedan recogidas en la siguiente ilustración:

Symbol	Parameter	Value	Unit
V_O	Output voltage	50	V
V_I	Input voltage (for ULN2002A/D - 2003A/D - 2004A/D)	30	V
I_C	Continuous collector current	500	mA
I_B	Continuous base current	25	mA
T_A	Operating ambient temperature range	- 40 to 85	°C
T_{STG}	Storage temperature range	- 55 to 150	°C
T_J	Junction temperature	150	°C

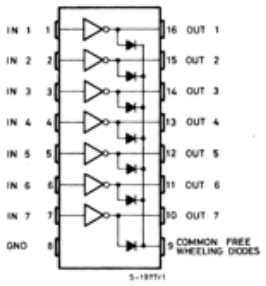


Ilustración 22 Tabla de características y pines del ULN2003

Como se puede ver en la siguiente ilustración, los motores de cada eje están conectados a los mismos pines de la placa Arduino porque reciben la misma señal. Pero si se mira más en detalle se puede apreciar que están conectados a modo espejo, es decir con las conexiones invertidas. Esto es porque al estar enfrentadas las ruedas, los motores deben girar en sentidos contrarios para que el robot avance hacia la dirección deseada, de lo contrario se produciría rotación sobre el eje z.

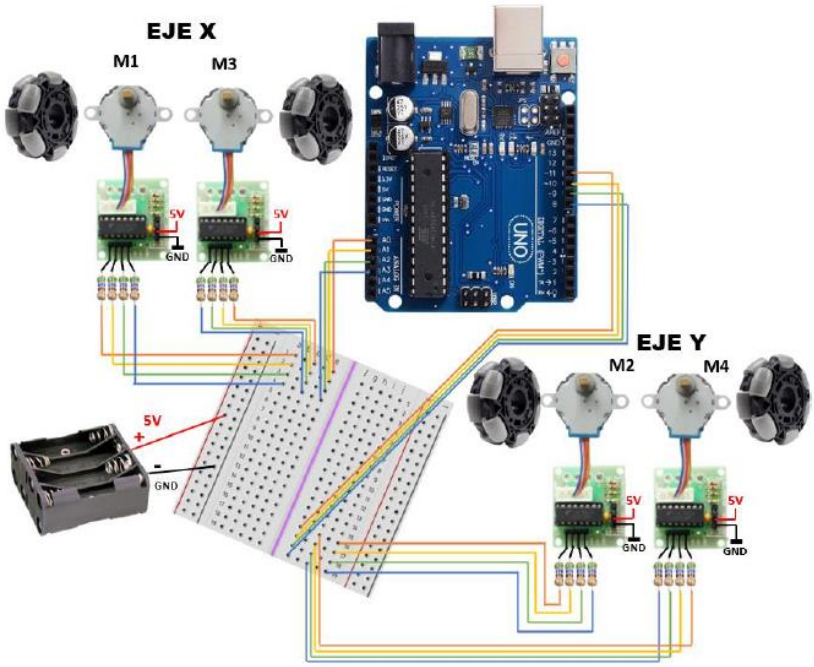


Ilustración 23 Conexionado de los Servos con el Arduino

3.7 Chasis Robot

El chasis del robot compuesto por 8 piezas atornilladas entre ellas, que han sido modificadas respecto al diseño inicial. Estas piezas han sido diseñadas en concreto para este robot teniendo en cuenta para los componentes que se van a integrar en este y

buscando el mayor ahorro de material. Cabe destacar que estas piezas han sido diseñadas en el software gratuito de diseño 3D FreeCad.

3.7.1 Diseño mecánico

Para el rediseño de las partes físicas y del robot se han tenido en cuenta los componentes finales escogidos para la implementación del prototipo y la reducción del coste de fabricación del prototipo reduciendo la cantidad de material utilizado para el prototipo.

El robot necesita 4 GDL, debe tener movilidad dentro del eje Z, para poder subir o bajar el útil de escritura, además de poder ir a cualquier punto del plano XY. Para conseguir estos movimientos, además de alojar en su estructura el resto de los componentes, se ha rediseñado el chasis del motor, al que se le han acoplado una serie de piezas auxiliares acopladas a la estructura principal, estas piezas han sido diseñadas también para paliar limitaciones de tamaño de las piezas en el proceso de impresión por el modelo de la impresora 3D, que se ha utilizado para imprimirlas.

Como se ha tratado en puntos anteriores, los materiales para imprimir piezas en 3D para prototipos son muy variados y la elección de estos dependen del tipo de prototipo y el uso que se le vaya a dar. Además, se debe tener en cuenta el modelo de impresora, y sus filamentos compatibles. En el anterior TFG se propuso el uso de la impresión por deposición fundida, método que se ha mantenido en este TFG, por sus características, además de ser el método que utiliza la impresora 3D disponible modelo Artillery Genuous**.**

Layer Resolution	0.1mm	Maximum Print Speed	150 mm/s
Frame	Aluminum Extrusion & ABS	Maximum Travel Speed	250mm/s
XYZ Positioning Accuracy	0.05mm, 0.05mm, 0.1mm	Build Volume	220 x 220 x 250 mm
Printing Filament	PLA, ABS, TPU, Flexible Materials	Extruder Type	Titan Extruder (Direct Drive)
Filament Diameter	1.75mm	Maximum Build Plate Temperature	130C
Nozzle Diameter	0.4mm		
Machine Dimensions	430 x 390 x 510 mm	Power Requirement	110V / 220V
	430 x 390 x 590 mm (with spool holder)		600W max (with heated bed turned on)
Shipping Box Dimensions	540 x 440 x 320 mm	Connectivity	USB, TF Card, USB Stick
Machine Weight	8.9 kg	Control Board	MKS Gen L
Shipping Weight	10.5 kg	Nozzle Type	Volcano

Ilustración 24 Características Impresora 3D Artillery Genious

El material utilizado para la impresión es el PLA de la marca Elegoo color gris, este producto en concreto está fabricado con un material termoplástico de alta calidad, lo que facilita su uso debido a su baja temperatura de fusión, baja deformación, sin olor durante la impresión y un acabado brillante. Ofrece precisión y consistencia dimensional con una tolerancia de +/- 0,02 mm y se presenta en bobinas de 1 kg. Su bobinado mecánico y examen manual aseguran un filamento ordenado y sin enredos, lo que evita roturas. Además, está diseñado para una extrusión suave y sin obstrucciones, secado y sellado al vacío para protegerlo de la humedad. Su compatibilidad es universal con la mayoría de las impresoras 3D FDM de 1,75 mm en el mercado.

Ilustración 25 Filamento Elegoo gris

3.7.1.1 Chasis

El chasis es la pieza principal del robot, sobre la cual van atornilladas el resto de las piezas y componentes. Esta es una de las partes que más cambio se puede observar respecto al robot inicial, esto se debe a la elección definitiva de los componentes que constituyen el robot, la búsqueda de ahorro de material y el hacer un diseño más compacto para mejorar la portabilidad.

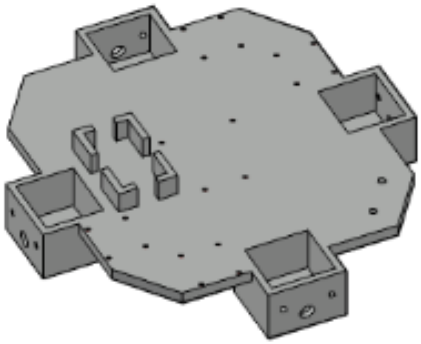


Ilustración 26 Chasis anterior diseño

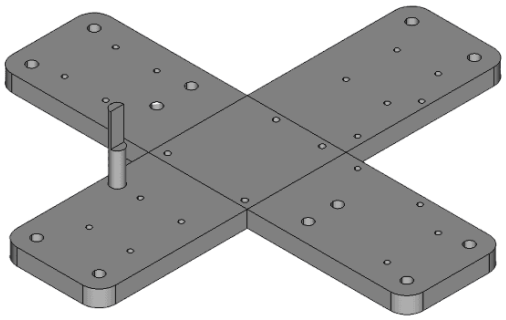


Ilustración 27 Chasis nuevo diseño

La pieza cuenta con varias perforaciones donde se van a anclar los componentes electrónicos del robot. También se acoplarán a las que están en los extremos las 4 piezas que alojan a los motores, también se fijará a esta la pieza inferior que conforma el mecanismo para el útil de escritura y por último la pieza que actúa como soporte para la batería.

Dentro del cuadrado central, hay 4 perforaciones donde se alojará el Arduino UNO, a en la mitad de los extremos hay perforaciones que sirven para acoplar los drivers de los motores. Cuenta también con una pieza cilíndrica saliente en forma de antena, que es parte de la estructura del chasis donde se acoplará el módulo HC-05.

En esta pieza, todas las fijaciones sobre las perforaciones se hacen con tornillos metálicos, y en aquellas perforaciones que son pasantes se utiliza una tuerca para hacer el tope y asegurar el elemento.

3.7.1.2 Soporte de escritura fijo

El soporte de escritura fijo es otra de las piezas que presenta gran cambio respecto al anterior diseño. Para empezar, se ha simplificado el sistema de alza y hay una reducción del uso de material. Ahora la fijación del rotulador es mayor y hay menos problemas de fallos en la impresión a causa de pequeñas vibraciones en el útil de escritura.

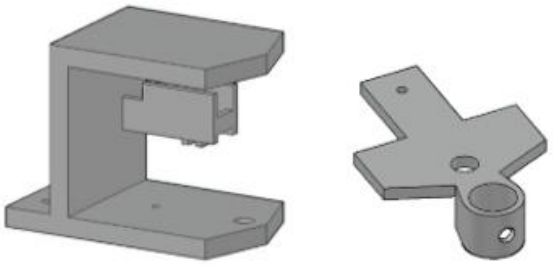


Ilustración 28 Piezas que componían el anterior soporte de escritura

El nuevo soporte de escritura se compone de tres partes. La parte principal del útil es la que se fija al chasis y sobre la que se coloca atornillado a los soportes centrales de la base del servomotor. El servomotor cuando sea accionado hará que la segunda pieza, que compone el soporte, donde está fijado el boli se mueva en el eje Z, recorriendo unos clavos que actúan como carriles. Estos clavos están, insertados a la primera pieza mediante unas extensiones perforadas que sobresalen de la misma con un diámetro lo suficientemente estrecho para que queden fijados por fricción. A la pieza superior se fija el útil de escritura mediante un tornillo y una tuerca, el tornillo empujará el útil contra la pared del cilindro haciendo que así quede fijado.

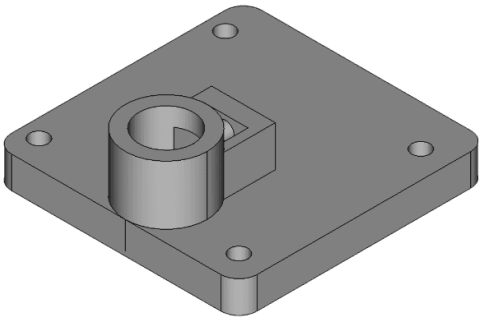


Ilustración 29 Parte superior del soporte

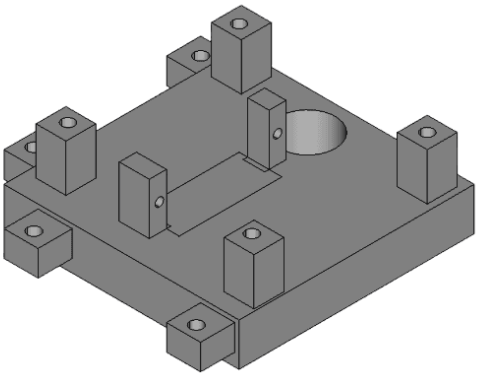


Ilustración 30 parte inferior del soporte

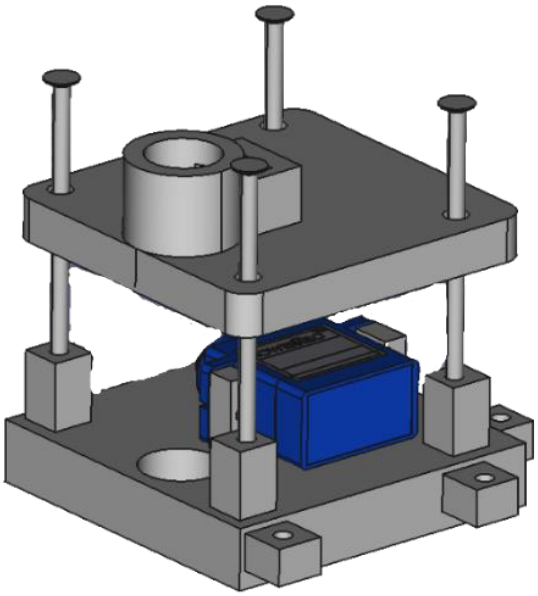


Ilustración 31 Soporte de escritura fijo

3.7.1.3 Anclaje de la batería

Esta es una de las piezas que han sido desarrolladas en concreto para este diseño. Esta pieza es la encargada de acoplar la batería en la parte inferior del robot. La batería será acoplada en la parte baja del robot, pues así tendrá un diseño más compacto al añadir una altura al prototipo. La batería se colocará en la parte central de la pieza, a esta irá acoplada un velcro para así obtener una mayor sujeción sin perder la posibilidad de ser retirada cómodamente. Esta práctica de añadir un velcro a las baterías para lograr una mayor sujeción y resistencia a las vibraciones se puede observar también en las bicicletas eléctricas de montaña, donde suele ser habitual ver este elemento utilizado gracias a su versatilidad y bajo costo. Esto nos indica que, aunque sea un material vulgar, de uso común, ha sido sometido a pruebas para comprobar su versatilidad en entornos mucho más hostiles de los que sufrirá el robot. Además, esto mejora la fiabilidad al trasportar el robot en trayectos mayores donde se puede ver afectado por algún golpe o vibración.

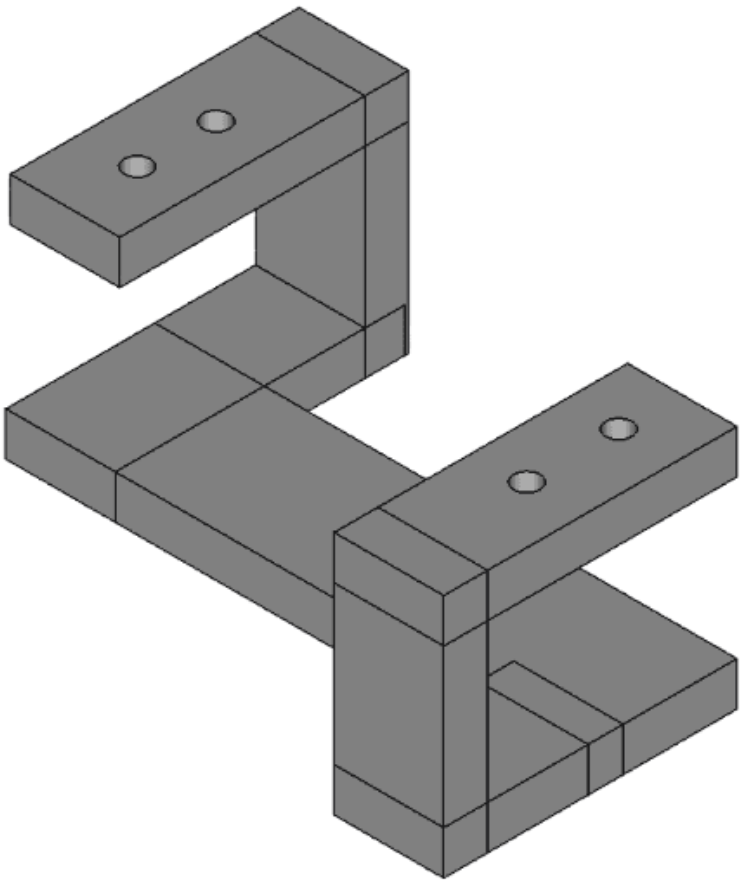


Ilustración 32 Anclaje de la batería

3.7.1.4 Soporte de los motores

Estas cuatro piezas han sido añadidas al diseño final para hacer más robusto el soporte donde se acoplan los motores que impulsan las ruedas. Esta estructura está conformada por un cilindro agujereado donde se insertará el step motor y este se apoyará en las paredes interiores del cilindro y será acoplado a esta pieza mediante unos tornillos con tuerca, la tuerca se acoplará al tornillo haciendo tope con las dos inserciones rectangulares que tiene la pieza, colocadas de forma simétrica una en frente de la otra. El atornillado se hará aprovechando las aletas agujereadas que ya vienen por defecto en el modelo del motor, puestas por el fabricante para facilitar el acople de su motor paso a paso. De esta forma lograremos un acople seguro y robusto para el motor, sin incurrir en mucho más material que los acoples del anterior motor.

La pieza cuenta con una base rectangular con dos agujeros, esta base está en contacto con el chasis del robot y es usada para atornillar a este la pieza. Además, la pieza, cuenta con unas hendiduras laterales en el cilindro principal para así facilitar la colocación de las tuercas anteriormente comentadas.

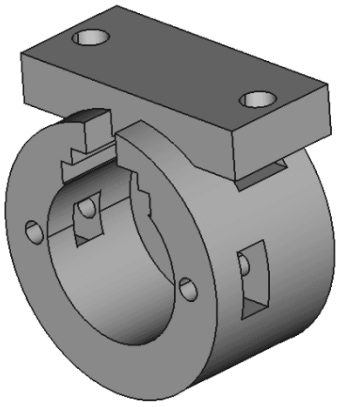


Ilustración 33 Soporte de los motores

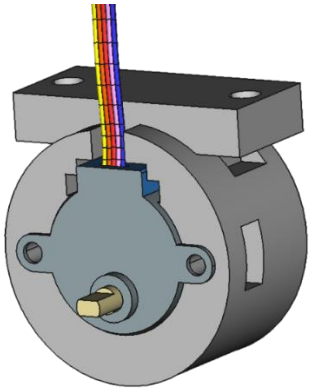


Ilustración 34 Soporte de los motores con motor PaP

3.8 Motores

El robot se desplaza por el Plano XY, para cumplir con esta trayectoria de forma estable utilizamos cuatro motores. Por los requerimientos de este robot, necesitamos que los motores sean precisos en su movimiento, por encima de rápidos o potentes. La mejor opción para esta tarea, como se remarca en el anterior diseño, son los motores paso a paso.

Un motor PaP se puede definir como un conversor electromagnético incremental que transforma pulsos eléctricos en movimientos angulares de un eje. Este movimiento angular, se repite exactamente con cada pulso sucesivo que el circuito de control inyecta al motor. Estos poseen una elevada capacidad de posicionamiento, el error que presentan es muy bajo y además no es acumulativo, y esta característica los hace ideales para sistemas que requieran un control exacto de dirección, velocidad y posición de un movimiento.

Estos motores cuentan con 3 secuencias de giro. La secuencia de excitación por bobina doble que provee mayor torque que la de bobina simple, resultando ambas en un movimiento angular igual al paso del motor. La secuencia de medios pasos alterna las secuencias de excitación por bobina simple y por bobina doble. De esta manera, se obtiene mayor precisión en los movimientos ya que permite movimientos cada medio ángulo de paso. En sistemas donde se necesite todavía mayor precisión, se recurre a una técnica denominada micro pasos, que consiste en que el motor alcance posiciones intermedias entre un paso y un medio paso.

De esta manera, en un motor de 1,8° por paso, por ejemplo, realizando 8 micro pasos por paso se podría obtener, movimientos cada 0,225°. La técnica de micro pasos trae aparejado otras ventajas tales como reducción de los problemas de resonancia y mejora en la velocidad de rotación.

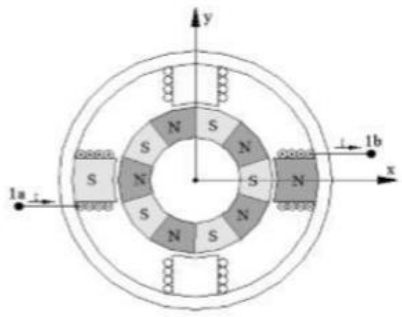


Ilustración 35 Motor paso a paso de dos fases excitado por bobina en posición de equilibrio a 0º

Existen dos tipos de motores paso a paso, de reluctancia auto conmutado y de reluctancia conmutado. El motor de reluctancia auto conmutado es un accionamiento de corriente continua sin escobillas y sin imanes permanentes, que está constituido por una estructura magnética con polos salientes tanto en el estator como en el rotor. Donde en los polos del estator se ubican bobinas concentradas y conectadas entre sí en serie, a pares diametralmente opuestos, forman do las fases del motor. La conmutación de las corrientes en las fases se realiza mediante un convertidor estático de potencia, en el que la secuencia de conmutación de los interruptores de estado sólido que lo componen se controla por la posición del rotor.

En base a los tipos de motores paso a paso que hay, elegiremos uno unipolar, por facilitar el desarrollo. En base a estas dos premisas se elige el motor paso a paso 28BYJ-48 del fabricante Kiatronics que se puede ver en la siguiente ilustración.

Ilustración 36 Motor paso a paso 28BYJ-48

La versión elegida en el primer diseño, entre las dos que hay de 5V y 12V, es la de 5V y la que también se mantendrá en este trabajo. Los motivos son que este componente cumple con los requisitos de coste y accesibilidad, además que requiere una fuente de alimentación de menor potencia. En la siguiente ilustración se muestran las principales características del motor, todas ellas obtenidas de su datasheet.

Modelo	28BYJT-48 – 5V (Hoja de datos)
Tensión nominal	5VDC
Nº de fases	4
Ángulo de paso	5,625º/64
Resistencia DC	50Ω±7%(25ºC)
Torque con tracción	34.3mN.m(120Hz)
Torque de arrastre	300 gf.cm

Ilustración 37 Características principales 28BYJT-48 5V

Para el dimensionado de la fuente de alimentación es necesario conocer el valor de la intensidad soportada por los motores, para este proceso es usada la ley de Ohm. De esta forma conocemos el valor de la intensidad soportada por fase es de 0.1 A.

El ángulo de paso de este motor es de 5.625º, pero para aumentar la resolución hace uso de una reductora de 1/64, que gracias a esta el motor tendrá una resolución de 0.88º. En el diseño anterior se usaba la configuración de medios pasos, que escita a dos bobinas al mismo tiempo, proporcionando mayor precisión y fluidez. Este método hace que se multiplique por dos el número de pasos para dar una vuelta, es decir, para dar una vuelta se toman 8 pasos.

Nº de paso	Bobina A	Bobina B	Bobina C	Bobina D
1	HIGH	LOW	LOW	LOW
2	HIGH	HIGH	LOW	LOW
3	LOW	HIGH	LOW	LOW
4	LOW	HIGH	HIGH	LOW
5	LOW	LOW	HIGH	LOW
6	LOW	LOW	HIGH	HIGH
7	LOW	LOW	LOW	HIGH
8	HIGH	LOW	LOW	HIGH

Ilustración 38 Encendido y apagado de pibes con medios pasos

Para la alimentación de estos motores se hará uso del módulo controlador UNL2003, como se explicó anteriormente en el apartado 3.6.

3.9 Cableado

Para las conexiones de los componentes electrónicos del motor se han empleado distintos tipos de cables, con conectores distintos, dependiendo del uso y los dispositivos que conecten.

3.9.1 Cableado alimentación

Para las conexiones de la alimentación se han tenido que usar cables con conectores tipo USB, pues son los que requiere la batería. Para el Arduino se ha utilizado el propio cable que viene por defecto con este y que también sirve para programarle, el cable USB to USB tipo B. Para la alimentación de los drivers se ha adaptado dos cables, cada uno cuenta con una salida tipo USB, compatible con la batería, y se le ha empalmado cuatro cables de puente hembra, dos al negativo y dos al positivo. De esta forma serán alimentados satisfactoriamente los cuatro drivers del motor con dos cables de este tipo.

Ilustración 39 Cable USB to USB tipo B

Ilustración 40 Cable adaptado USB to puente hembra

Para la alimentación del servo y del módulo bluetooth HC-05, que no se conectan directamente a la alimentación si no que se alimentan gracias a los puertos de alimentación presentes en el Arduino, requieren otro tipo de entradas. Para el servo, que se conecta a los pines de los 5 V y GND se hace uso de un cable tipo puente de macho a macho. Para la conexión del módulo Bluetooth que se conecta a los pines de 3V y GND mediante un cable Dupont tipo macho a hembra.

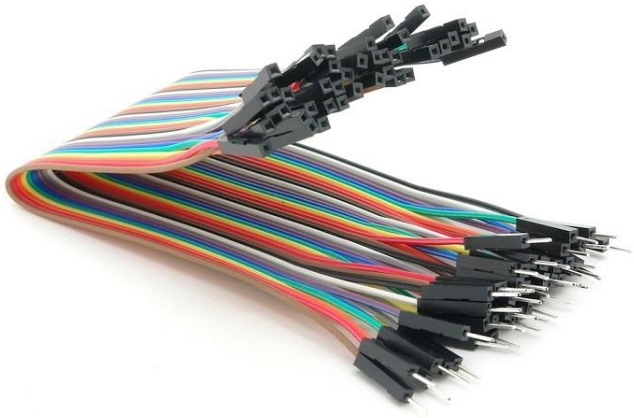


Ilustración 41 Cables Dupont hembra – macho

Ilustración 42 Cable puente macho – macho

3.9.2 Cables de conexión

Los cables que han sido utilizados para establecer las conexiones de comunicación entres lo dispositivos son cables tipo puente, comúnmente utilizados para los proyectos de Arduino. Para la conexión del Arduino a los drivers de los motores, que aumentarán las señales y la mandarán a las fases del motor es nuevamente mediante cables puente.

de macho a macho. El número de los cables empleados para esto son ocho, cuatro para cada fase, como se vio anteriormente, mediante cuatro cables se envía señales a dos de los drives, de forma inversa. Estos están conectados a una mini breadboard adhesiva.

Ilustración 43 Mini BreadBoard

De esta Breadboard saldrán cuatro cables Dupont macho a hembra con cuatro conexiones cada uno, una por cada fase del motor, dando así un total de dieciséis conexiones. Para la conexión de los pines RX y TX del Módulo HC-05 se utilizarán los cables puentes de macho a hembra. Finalmente, para las conexiones de la PWM del servomotor se hace uso de un cable puente macho a macho.

Ilustración 44 Cable puente hembra macho

Los motores ya vienen con un cable con una entrada, preparada para los drivers de este trabajo.

Para el cableado del motor se han utilizado en total, 5 cables Dupont de cuatro conexiones, 11 cables puente, además 2 cables puente hembra – hembra adicionales para la adaptación de los dos cables de USB, y un cable USB to USB tipo B.

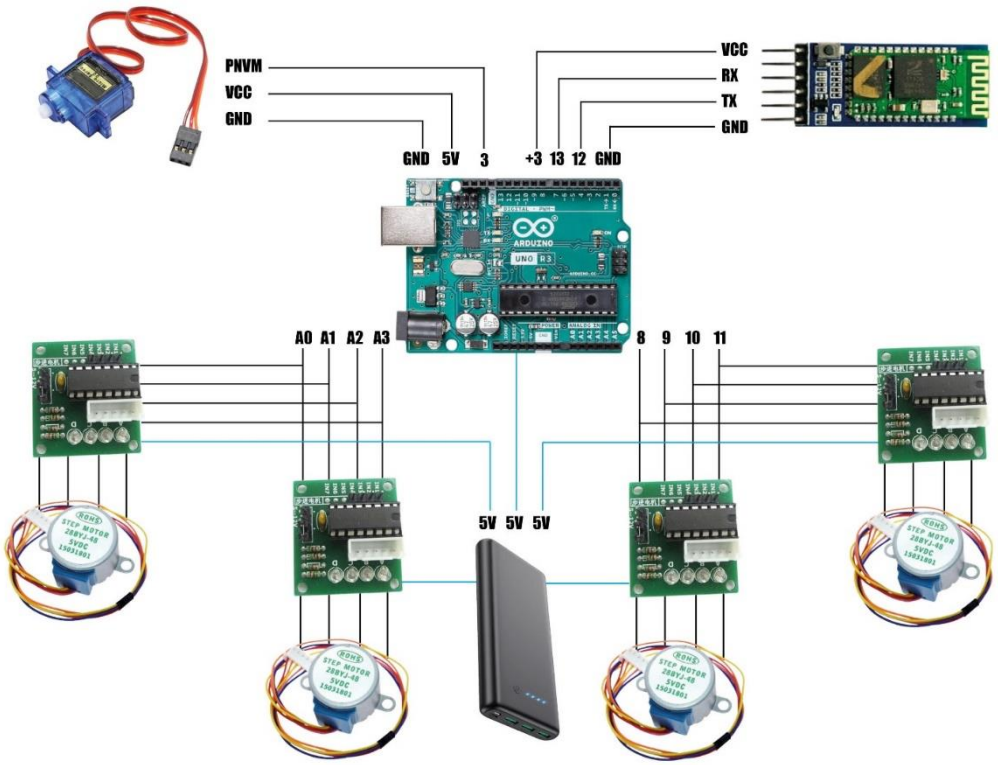


Ilustración 45 Esquema Conexiones del Robot

3.10 Montaje del robot completo

Antes de comenzar el montaje, una vez impresas las piezas habrá que revisar los elementos, por si hiciera falta algún pulido o limados necesarios antes de ensamblar el robot, puesto, aunque se haya tenido en cuenta en la hora del diseño las tolerancias de la máquina, las impresiones 3D no son perfectas y tienen unos márgenes de exactitud.

Para el montaje del robot recomendado por este trabajo, primero se debe comenzar atornillando los componentes electrónicos a su lugar correspondiente, seguido atornillar la base inferior de escritura y soporte de la batería.

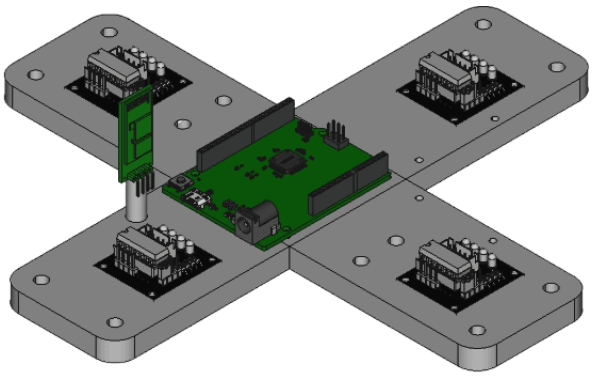


Ilustración 46 Paso 1

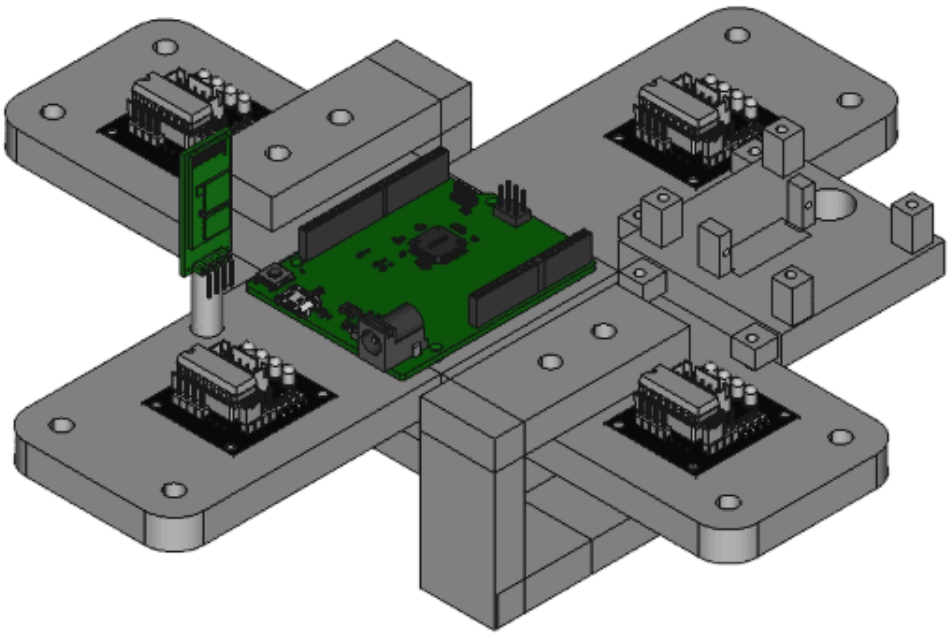


Ilustración 47 Paso 2

Para luego insertar la batería en su anclaje y atornillar el servomotor en su lugar. Mientras se debe atornillar uno por uno los motores al soporte de estos, como no hay cambios en el diseño de cada una de dichas piezas, es indiferente a qué soporte sea atornillado cada motor.

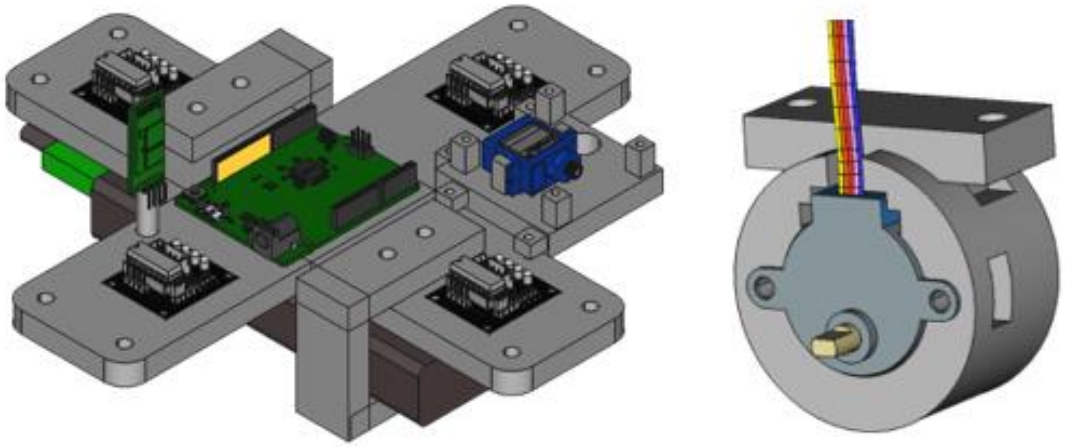


Ilustración 48 Paso 3

En el paso cuatro se atornillarán al chasis los cuatro soportes de los motores PaP. A continuación, se colocará la parte superior de escritura de forma que los cuatro agujeros de la base inferior y superior queden alineados, después aplicando algo de fuerza se colocarán los clavos que actúan como los carriles del mecanismo, completando así el montaje de la parte de escritura.

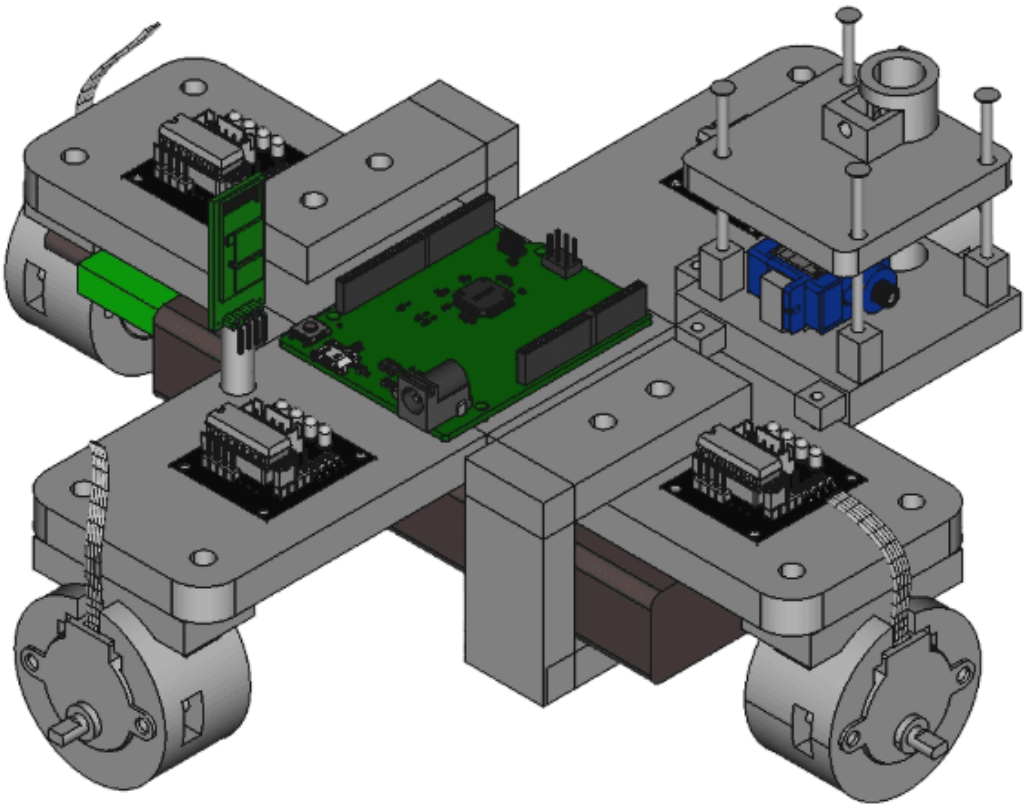


Ilustración 49 Paso 4

El quinto y último paso será colocar el útil de escritura deseado, después insertar las ruedas omnidireccionales en cada uno de los ejes del motor, quedando así montado el cuerpo completo del robot. Se puede ver una evolución de los componentes cuando este se compara con el del trabajo anterior, teniendo el nuevo un aspecto más robusto, con menos uso de material, más compacto, y elaborado.

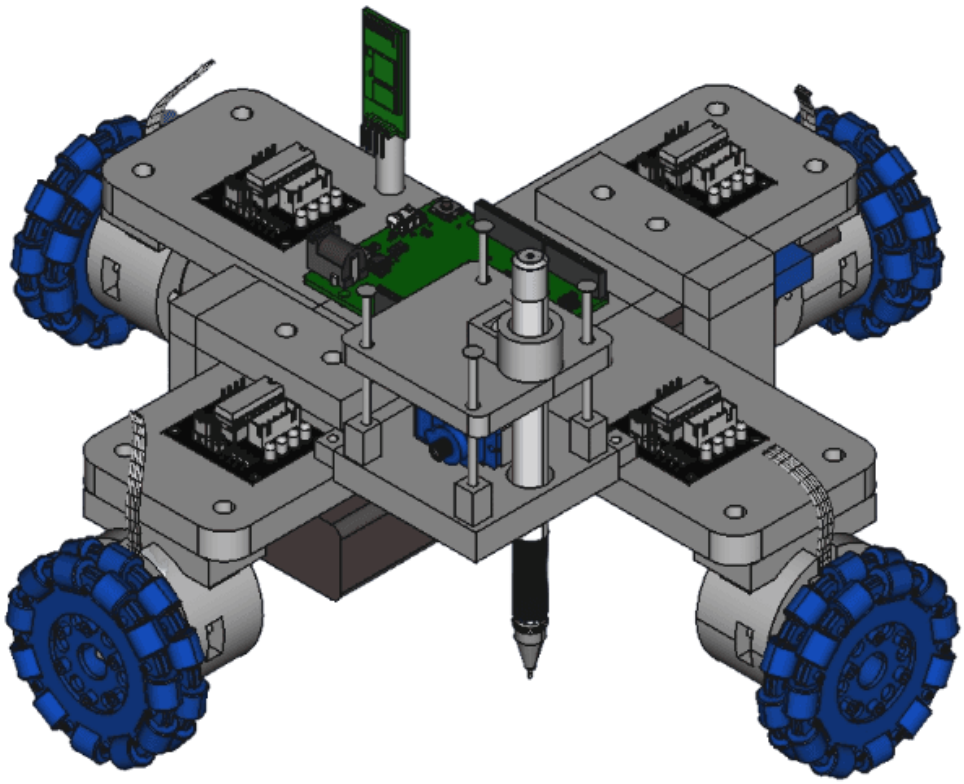


Ilustración 50 Paso 5 Robot completo

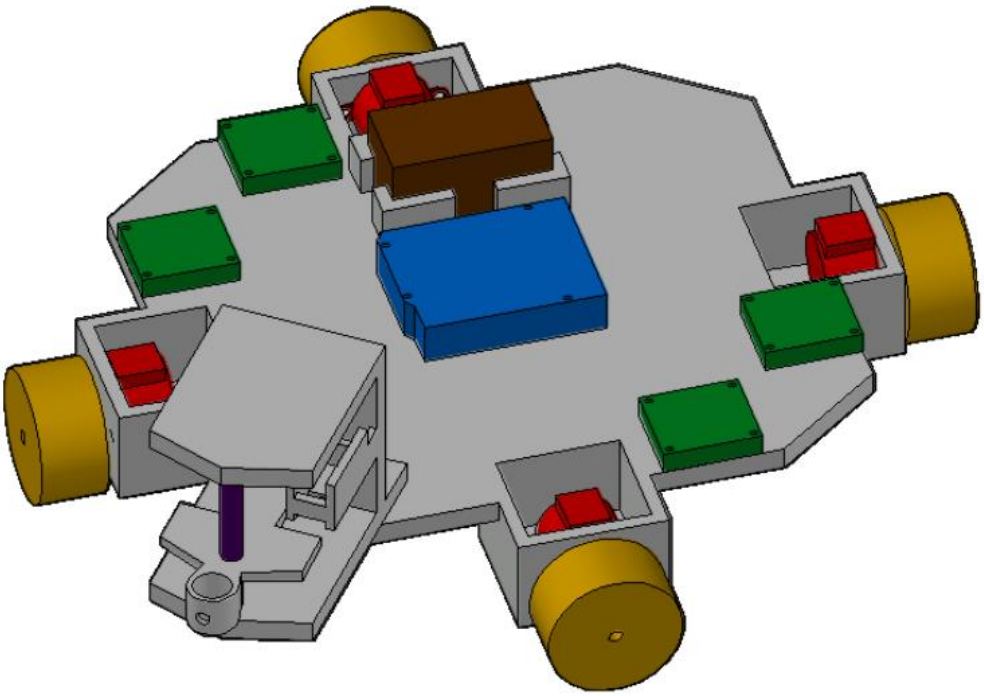


Ilustración 51 Robot completo diseño anterior

4. SOFTWARE Y PROGRAMACIÓN

Para llevar a cabo este proyecto se ha hecho uso de múltiples softwares, con los cuales se han realizados labores de diseño y programación entre otras. En cuanto a la parte de trasmisión es la que más ha cambiado, pues se ha establecido otro medio distinto para llevarlo a acabo, además de la programación de una aplicación que sirva como software de comunicación, siendo en esta en la que se sube el archivo de impresión, y mediante bluetooth se realiza la comunicación con el Arduino. Todos los programas utilizados han sido de licencia libre. Podemos ver que en la siguiente ilustración el flujo de proceso se ha mantenido igual que en el anterior trabajo.

Ilustración 52 Flujo de proceso

4.1 Generación del G-code

La generación de los datos necesarios para realizar la impresión se ha mantenido en la misma línea, pues no hay razón para cambiarlo. El software utilizado es Inkscape, que es gratuito y de código abierto, con este generaremos el G-code.

El G-code es el código que compone el archivo de control para una impresora 3D. El G-Code consta de comandos G y M, cada uno de ellos con un movimiento o acción asignado. La combinación de estos comandos permitirá a la impresora 3D entender qué patrón seguir con el fin de crear la pieza final.

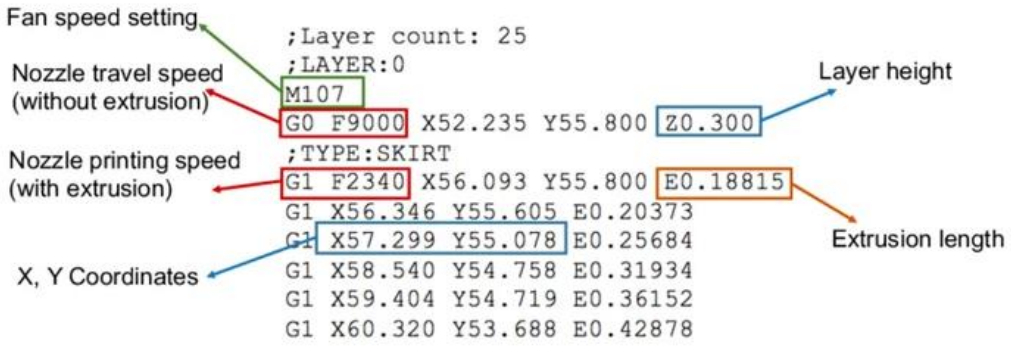


Ilustración 53 Ejemplo de comandos para impresión 3D

El software Inkscape puede ser utilizado para la generación de Gcode para impresoras, precisamente porque es capaz de convertir imágenes vectoriales en instrucciones que una impresora, máquina de control numérico por computadora (CNC) o el robot de este trabajo puede procesar.

Para la transmisión, una vez obtenido el archivo G-code, que puede ser también guardado como un archivo de texto, es decir, de extensión.txt, utilizaremos una aplicación desarrollada especialmente para este trabajo, que hará la función de software de trasmisión de datos.

4.2 Trasmisión del G-code

Para el desarrollo de la aplicación se ha utilizado el software gratuito de desarrollo de aplicaciones Android Studio, versión Flamingo, desarrollando el código en del main de la aplicación en el lenguaje Java, un lenguaje de programación orientado a objetos, multiplataforma y muy utilizado para aplicaciones web, móviles y empresariales.

Android Studio es el entorno de desarrollo integrado (IDE) oficial para la creación de aplicaciones Android, desarrollado por Google. Basado en IntelliJ IDEA, ofrece un editor

de código avanzado con autocompletado y refactorización, un diseñador de interfaces visual, y un emulador para probar aplicaciones en un entorno virtual. Se integra con herramientas de compilación como Gradle y sistemas de control de versiones como Git. Incluye perfiladores para analizar el rendimiento y soporta el desarrollo multiplataforma a través de Flutter. Con extensa documentación y una gran comunidad, Android Studio facilita el desarrollo eficiente de aplicaciones Android.

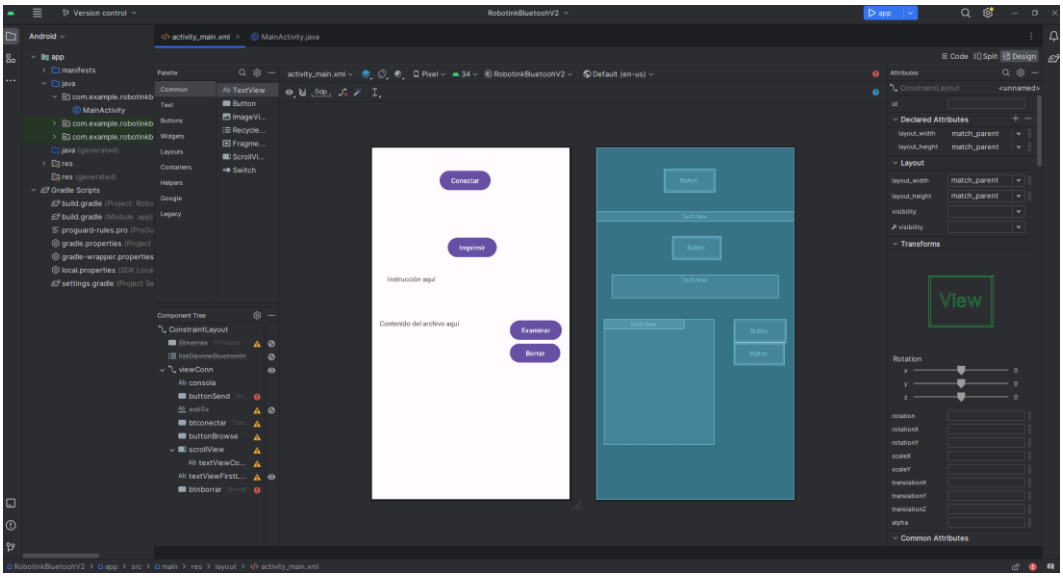


Ilustración 54 Entorno de programación de Android Studio

4.2.1 Interacción con la aplicación a nivel usuario

La aplicación desarrollada permite una interacción directa con el robot móvil impresora, ofreciendo control y monitoreo en tiempo real durante la impresión. Además, al basarse en el código de Arduino del proyecto anterior y utilizar el protocolo XON/XOFF para la transmisión, garantiza una comunicación eficiente y estable entre la aplicación y el robot. La ventaja principal de contar con una aplicación propia radica en la adaptabilidad y personalización a las necesidades específicas del proyecto, proporcionando un control más preciso y una experiencia de usuario mejorada en comparación con soluciones genéricas. Esta integración software-hardware optimizada contribuye significativamente al éxito y la eficacia del sistema global.

La aplicación a nivel usuario funciona de la siguiente manera. Al entrar por primera vez a la aplicación, esta pedirá permiso al usuario, que le dé permisos para poder acceder a as funciones de Bluetooth, además de no tener activada la antena Bluetooth del dispositivo, esta le avisará al usuario y a continuación le pedirá permiso al usuario para

activarlo. Después de activar el Bluetooth en el dispositivo pasará a mostrar la interfaz principal.

Ilustración 55 Interfaz de la aplicación pidiendo permisos de activación de Bluetooth

La aplicación consta de dos pantallas. Una pantalla principal donde se harán las funciones de comunicación y otra donde se muestran los dispositivos para conectar.

En la pantalla principal se muestra de abajo hacia arriba un botón “Conectar”, este cambia la pantalla y muestra la lista de dispositivos a conectar, el mismo botón cuando está conectado servirá como botón de desconexión y aparecerá escrito en él “Desconectar”. Para hacer más visual el apartado de la conexión del dispositivo hay una pequeña línea, que se muestra de color blanco cuando no hay conexión y de color verde cuando sí la hay.

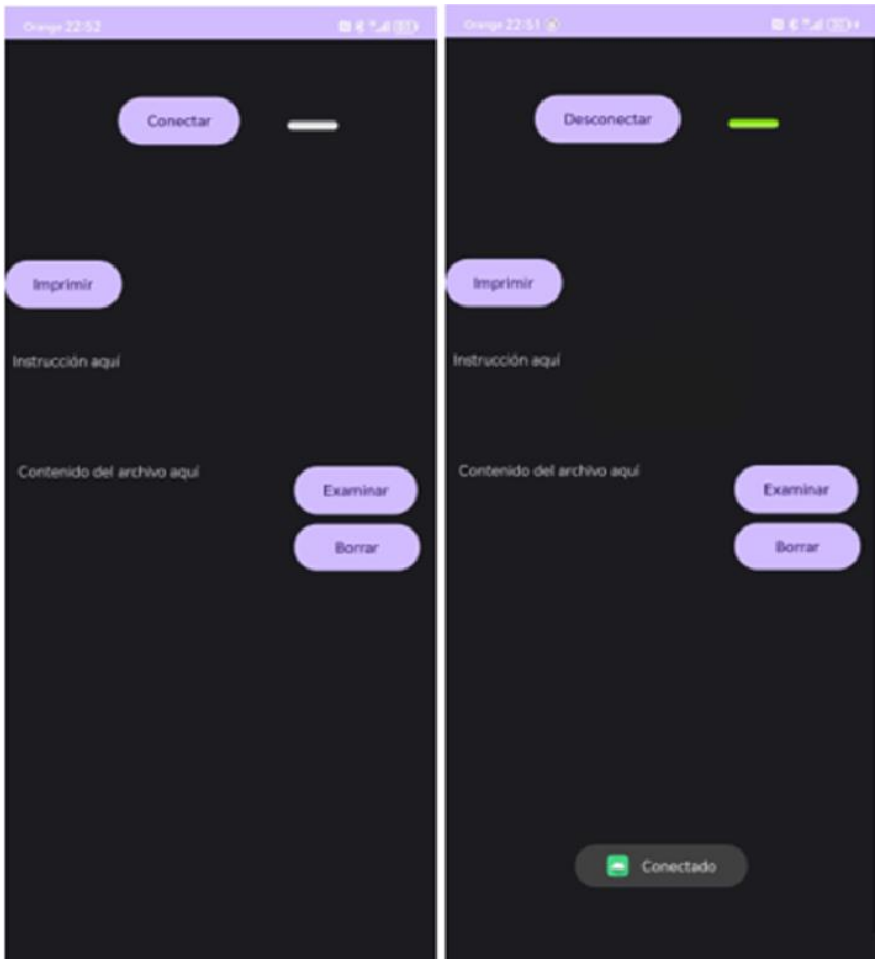


Ilustración 56 Pantalla principal en estado de desconexión y en estado de conexión

Más abajo se muestra el botón Imprimir el cual, tiene la funcionalidad de que cuando esté el dispositivo conectado al robot, la aplicación comenzará la comunicación. Este botón está preparado para que, si el dispositivo no está enlazado con el robot y este es pulsado, se mostrará por pantalla un mensaje emergente avisando al usuario de que no hay conexión. Donde aparece el texto donde dice “Instrucción aquí” es donde se mostrarán las instrucciones recibidas por el robot, esto es usado para así poder monitorear de una manera visual y precisa si la comunicación entre el robot y la aplicación es correcta, es decir, muestra por pantalla los comandos enviados al robot. Debajo de esta, aparece escrito “Contenido del archivo aquí”, aquí aparecerá el código una vez subido de forma que se pueda ver. Cómo el código tiene bastantes líneas esta permite “scrollear” el contenido del archivo. Para subir el Archivo a la aplicación se encuentra en la mitad derecha de la pantalla principal, un botón que dice Examinar. Este botón abre en el teléfono el explorador de archivos y deja seleccionar el archivo donde se encuentra el G-code.

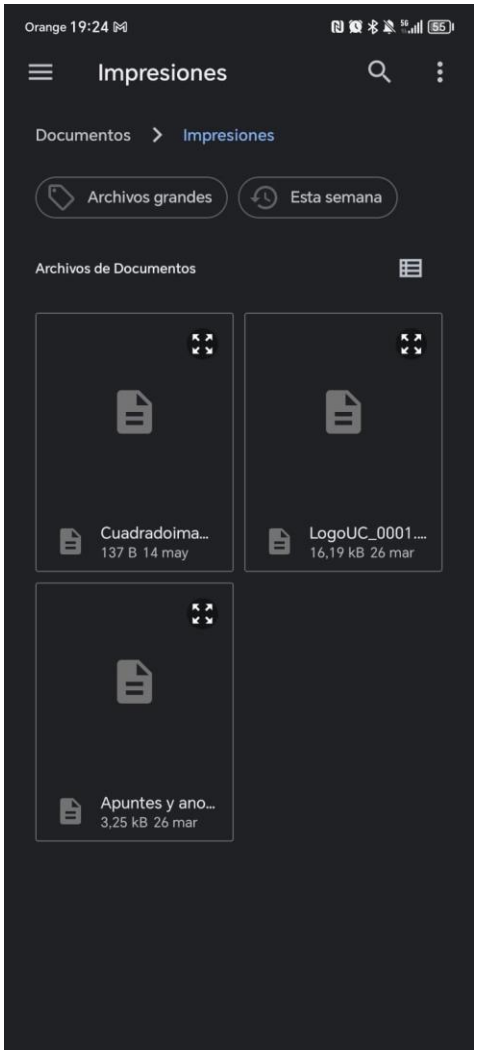


Ilustración 57 Explorador de archivos

Seleccionado el archivo lo muestra por pantalla, de este archivo es de donde obtendremos las instrucciones que el robot interpretará para poder realizar los movimientos correspondientes para imprimir la imagen. Una vez se pulse imprimir la aplicación y el robot irán tratando el código línea por línea interactuando entre ellos, realizando una comunicación bidireccional.

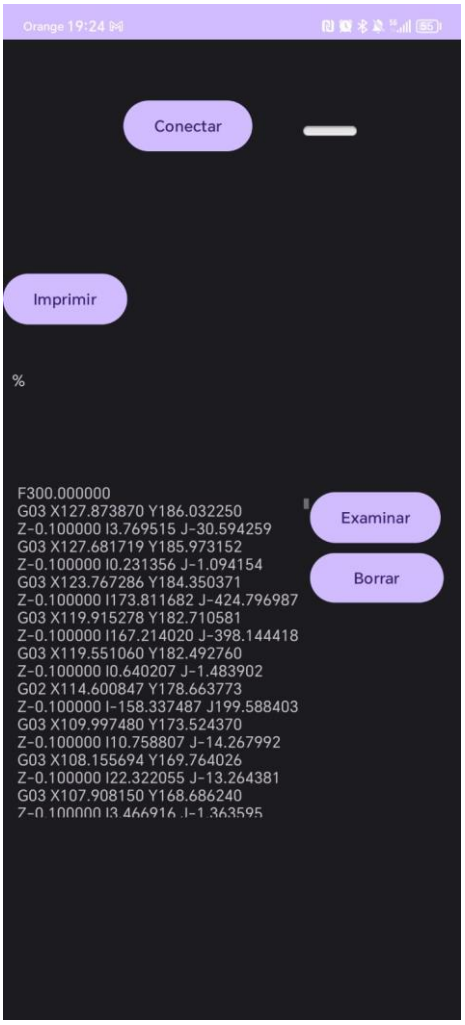


Ilustración 58 Pantalla de la aplicación donde se muestra el Archivo de G-Code cargado

Debajo del botón de conexión se encuentra el botón de borrar que limpiará en la pantalla, los comandos que aparezcan en pantalla, los enviados por la aplicación y lo recibidos, además también eliminará la visión del archivo de la aplicación, permitiendo así subir otros archivos.

El botón conectar en la parte superior cambia de pantalla en la aplicación mostrando una lista “scrolleable” de los dispositivos enlazados con el teléfono móvil en el que esté descargada la aplicación. Si seleccionamos uno de los dispositivos que aparecen, aparecerá un mensaje emergente en la parte inferior de la pantalla que dirá “conectando”, desaparecerá pasados unos segundos. Sí la conexión es exitosa aparecerá otro mensaje del mismo tipo que dirá “Conectado” y automáticamente se nos mostrará la pantalla principal. Del contrario si la conexión no es exitosa, aparecerá otro mensaje que dirá “False”. Estos mensajes son conocidos como “Toast”, mensajes emergentes de corta duración dejados por el programador de la app para mandar al

usuario un mensaje puntual que considere necesario. En esta pantalla también se puede ver un botón que pone principal que permite al usuario volver a la pantalla inicial.

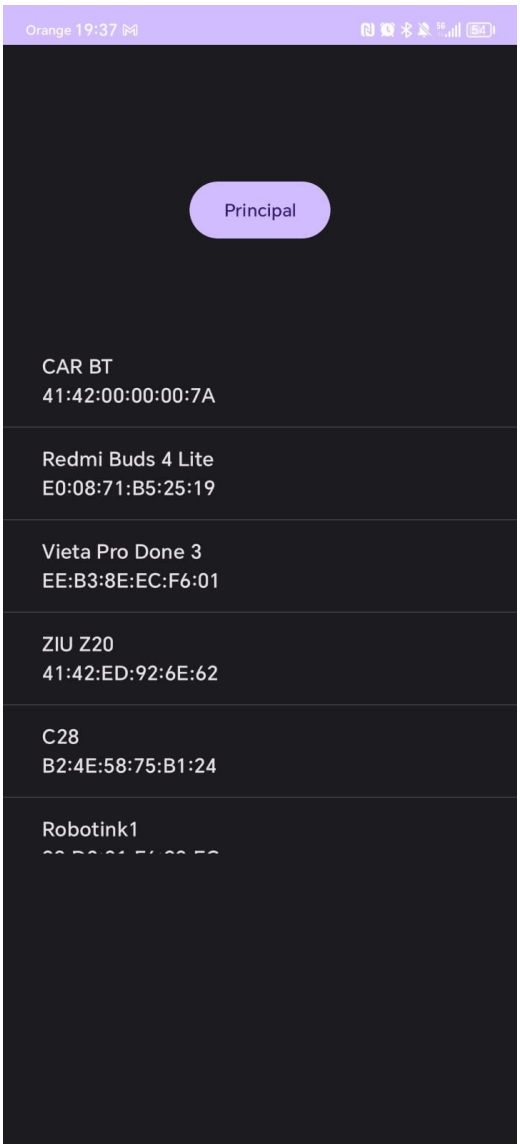


Ilustración 59 Pantalla que muestra los dispositivos disponibles

4.2.2 Programación de la app

En el entorno de programación de Android Studio se pueden diferenciar dos partes principales de la programación de una aplicación. La programación de la interfaz visual, es decir cómo están colocados los objetos, que colores tienen y demás características, en resumen, como percibe el usuario la aplicación. En esta parte es muy importante declarar correctamente los bloques y dejar los objetos visuales definidos, pues de no ser así, además de no que la aplicación no copilará, no habrá una buena sinergia con la programación de la lógica de la aplicación, la otra parte principal de la aplicación. En esta parte se programa cuáles son los efectos de la interacción con objetos y la función

que cumple la aplicación. La programación lógica es crucial, ya que determina el comportamiento y la funcionalidad de la aplicación, asegurando que todas las acciones del usuario se traduzcan correctamente en respuestas del sistema.

4.2.2.1 Programación visual de la app

Android Studio permite programar la aplicación tanto como con un código xml, como de una forma más interactiva usando un editor de diseño visual. Este editor permite a los desarrolladores colocar objetos de manera sencilla e interactiva arrastrándolos y soltándolos en la interfaz de usuario. Esta funcionalidad no solo agiliza el proceso de diseño, sino que también ofrece una vista previa en tiempo real de cómo se verá la aplicación en un dispositivo real.

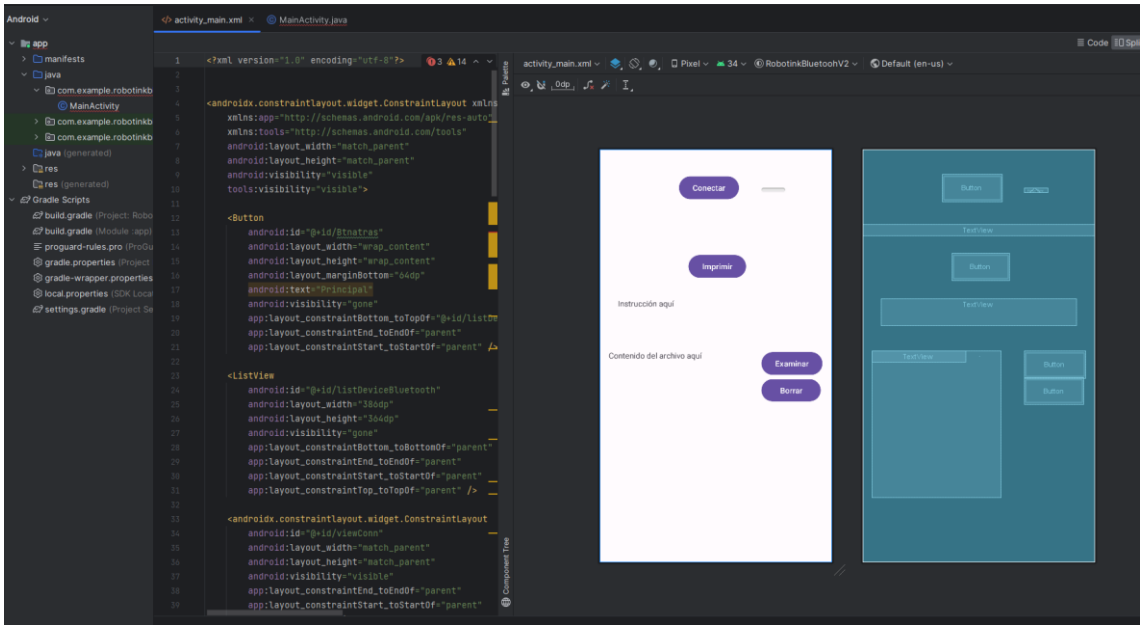


Ilustración 60 Split de la interfaz de programación visual

A continuación, el código XML de la app comentado.

```
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:visibility="visible"
    tools:visibility="visible">

    <Button
        android:id="@+id/Btnatras"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="64dp"
```

```
        android:text="Principal"
        android:visibility="gone"
        app:layout_constraintBottom_toTopOf="@+id/listDeviceBluetooth"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />
```

En el código XML anterior, se define un ConstraintLayout como el contenedor principal de la interfaz. Este contenedor permite una disposición flexible de los elementos. Dentro de él, se encuentra un botón con el ID Btnatras, que está inicialmente oculto (android:visibility="gone") y se posiciona en el centro horizontal de la pantalla, justo encima de una lista de dispositivos Bluetooth.

```
<ListView
        android:id="@+id/listDeviceBluetooth"
        android:layout_width="386dp"
        android:layout_height="364dp"
        android:visibility="gone"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

El ListView con el ID listDeviceBluetooth está destinado a mostrar una lista de dispositivos Bluetooth. Al igual que el botón anterior, está inicialmente oculto y ocupa una posición central dentro del ConstraintLayout. Esta parte del código corresponde a la pantalla de conexión de, qué en vez de crear lo que se conoce en Android studio como distintas activities, se limita a ocultar y mostrar los objetos en la misma pantalla. El motivo de esto es la simplificación del código, ya que a estos niveles la aplicación no se verá limitada en rendimiento por tener un solo activity.

```
<androidx.constraintlayout.widget.ConstraintLayout
        android:id="@+id/viewConn"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:visibility="visible"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <TextView
            android:id="@+id/consola"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="40dp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@+id/btconectar"></TextView>

        <Button
```

```
            android:id="@+id/buttonSend"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_marginTop="32dp"
            android:text="Imprimir"
            app:layout_constraintTop_toBottomOf="@+id/consola"
            tools:layout_editor_absoluteX="158dp"></Button>

        <EditText
            android:id="@+id/edtTx"
            android:layout_width="243dp"
            android:layout_height="54dp"
            android:layout_marginTop="40dp"
            android:layout_marginEnd="32dp"
            android:visibility="gone"
            app:layout_constraintEnd_toStartOf="@+id/buttonSend"

app:layout_constraintTop_toBottomOf="@+id/consola"></EditText>

        <Button
            android:id="@+id/btconectar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="44dp"
            android:layout_marginEnd="164dp"
            android:text="Conectar"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <Button
            android:id="@+id/buttonBrowse"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerInParent="true"
            android:layout_marginEnd="16dp"
            android:text="Examinar"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toTopOf="@+id/scrollView" />

        <ScrollView
            android:id="@+id/scrollView"
            android:layout_width="229dp"
            android:layout_height="260dp"
            android:layout_below="@id/buttonBrowse"
            android:layout_marginTop="204dp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.087"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/edtTx">

            <TextView
                android:id="@+id/textViewContent"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
```

```
                android:text="Contenido del archivo aquí" />

        </ScrollView>

        <TextView
            android:id="@+id/textViewFirstLine"
            android:layout_width="346dp"
            android:layout_height="47dp"
            android:layout_marginBottom="40dp"
            android:text="Instrucción aquí"
            android:visibility="visible"
            app:layout_constraintBottom_toTopOf="@+id/scrollView"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.492"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/buttonSend"
            app:layout_constraintVertical_bias="0.855" />

        <Button
            android:id="@+id/btnborrar"
            android:layout_width="105dp"
            android:layout_height="47dp"
            android:text="Borrar"
            app:layout_constraintStart_toStartOf="@+id/buttonBrowse"
            app:layout_constraintTop_toBottomOf="@+id/buttonBrowse" />

        <ImageView
            android:id="@+id/Noconectado"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="24dp"
            android:visibility="visible"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.327"
            app:layout_constraintStart_toEndOf="@+id/btconectar"
            app:layout_constraintTop_toTopOf="@+id/btconectar"

app:srcCompat="@android:drawable/button_onoff_indicator_off" />

        <ImageView
            android:id="@+id/Conectado"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="24dp"
            android:visibility="gone"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.327"
            app:layout_constraintStart_toEndOf="@+id/btconectar"
            app:layout_constraintTop_toTopOf="@+id/btconectar"

app:srcCompat="@android:drawable/button_onoff_indicator_on" />

    </androidx.constraintlayout.widget.ConstraintLayout>
```

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

Dentro del ConstraintLayout principal, se define otro ConstraintLayout con el ID viewConn, que contiene varios elementos interactivos como TextView, Button, EditText, ScrollView, y ImageView. Estos elementos están organizados para facilitar la interacción del usuario con la aplicación.

El TextView con el ID consola se utiliza para mostrar mensajes o datos relevantes para el usuario, mientras que los botones como buttonSend, btconectar, buttonBrowse, y btnborrar permiten realizar acciones específicas como enviar datos, conectar el dispositivo, examinar archivos y borrar contenido.

El ScrollView con el ID scrollView contiene otro TextView que muestra el contenido de un archivo, proporcionando una manera fácil de ver información extensa. Finalmente, las ImageView con los ID Noconectado y Conectado indican el estado de la conexión, cambiando su visibilidad según sea necesario.

A través de los comentarios anteriores, se ha detallado cómo se configuran botones, listas, vistas de texto y otros componentes clave, así como su interactividad y visibilidad.

4.2.2.*2* Programación lógica de la app

Ahora que se ha explicado la estructura y disposición de los elementos visuales de la aplicación en Android Studio, pasaremos a detallar la parte lógica de la aplicación. En esta sección, se abordarán los métodos y funciones que permiten la interacción con los componentes visuales y la ejecución de las acciones definidas. Para facilitar la implementación de los comandos Bluetooth, se ha utilizado la biblioteca BlueJHR, desarrollada por José Hidalgo Rodríguez y disponible de forma gratuita en su repositorio de GitHub. Esta biblioteca simplifica significativamente la gestión de la comunicación Bluetooth, permitiendo un desarrollo más eficiente y enfocado en las funcionalidades específicas de la aplicación.

```
package com.example.robotinkbluetoohv2;

// Bibliotecas blueooth

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;

import android.content.Intent;
import android.os.Build;
import android.os.Bundle;
```

```
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import com.ingenieriajhr.blujhr.BluJhr;

import java.util.ArrayList;
import java.util.List;

//Bibliotecas .text

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
//Bibliotecas  para leer las lineas del fichero
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
```

El código comienza declarando el paquete de la aplicación y las importaciones de bibliotecas esenciales para el desarrollo de aplicaciones Android. Estas importaciones abarcan tanto componentes relacionados con la interfaz de usuario como funcionalidades específicas, como la gestión de Bluetooth y el manejo de archivos.

Las importaciones relacionadas con la interfaz de usuario, como AppCompatActivity, Button, EditText, TextView, ListView, y ConstraintLayout, proporcionan las herramientas necesarias para construir la apariencia y la interactividad de la aplicación.

Por otro lado, las importaciones relacionadas con Bluetooth, como BluJhr, facilitan la comunicación inalámbrica entre la aplicación y dispositivos externos compatibles con esta tecnología. Estas bibliotecas simplifican la configuración y la gestión de código.

Además, las importaciones relacionadas con el manejo de archivos, como BufferedReader, InputStream, e InputStreamReader, brindan la capacidad de leer y procesar archivos almacenados localmente en el dispositivo del usuario. Parte importante para seleccionar el archivo de impresión donde se encuentra el G-Code.

```
public class MainActivity extends AppCompatActivity {
    //Tema .txt

    private static final int READ_REQUEST_CODE = 42;
    private TextView textViewContent;

    private TextView textViewFirstLine;

    //Bluetooth
    BluJhr blue;
    List<String> requiredPermissions;
    ArrayList<String> devicesBluetooth = new ArrayList<String>();
    ConstraintLayout viewConn;
    ListView listDeviceBluetooth;
    Button buttonSend;
    Button BtnConectar;
    Button Btnatras;
    Button btnborrar;
    TextView consola;
    EditText edtTx;
    ImageView conectado;
    ImageView noconectado;
```

En esta declaración de la clase MainActivity, se definen varios atributos que son fundamentales para el funcionamiento de la aplicación, estos van desde elementos visuales a otros más específicos de una biblioteca concreta como es 'BluJhr' y las listas de dispositivos Bluetooth.

La variable textViewContent es un componente de texto utilizado para mostrar el contenido completo de un archivo de texto seleccionado por el usuario. Por otro lado, textViewFirstLine se encarga de mostrar la primera línea del archivo seleccionado. Esta será enviada por Bluetooth al robot.

En cuanto a los componentes de Bluetooth, blue es una instancia de la clase BluJhr, que facilita la gestión de la comunicación Bluetooth en la aplicación, será utilizado para, por así decirlo, llamar a la conexión bluetooth. Las listas devicesBluetooth y listDeviceBluetooth almacenan los dispositivos Bluetooth disponibles y proporcionan una interfaz para que el usuario seleccione uno de ellos.

Los botones buttonSend, BtnConectar, Btnatras, y btnborrar permiten al usuario interactuar con la aplicación enviando datos por Bluetooth, conectándose o desconectándose de dispositivos Bluetooth, navegando entre vistas, y borrando la consola de comandos, respectivamente.

Finalmente, los componentes edtTx, noconectado y conectado son utilizados para mostrar información al usuario y permitirle ingresar datos en la aplicación. En conjunto,

estos atributos representan los elementos clave que forman la interfaz y la lógica de funcionamiento de la aplicación.

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Varioabels paa  la lógica de  la app
        final int[] conexion = {0}; // Inicialmente no conectado; 0
para no conexion 1 para conexion
        conexion[0]=0;


        //Variables bluetooth
        blue = new BluJhr(this);
        blue.onBluetooth();
        listDeviceBluetooth = findViewById(R.id.listDeviceBluetooth);
        viewConn = findViewById(R.id.viewConn);
        buttonSend = findViewById(R.id.buttonSend);
        BtnConectar = findViewById(R.id.btconectar);
        Btnatras = findViewById(R.id.Btnatras);
        consola = findViewById(R.id.consola);
        edtTx = findViewById(R.id.edtTx);
        btnborrar = findViewById(R.id.btnborrar);
        noconectado =  findViewById(R.id.Noconectado);
        conectado =  findViewById(R.id.Conectado);

        //Boton examniar
        Button buttonBrowse = findViewById(R.id.buttonBrowse);
        textViewContent = findViewById(R.id.textViewContent);
        // Listener para el botón de examinar archivos
        buttonBrowse.setOnClickListener(v -> openFile());
        // TextView para mostrar la primera línea del archivo
        textViewFirstLine = findViewById(R.id.textViewFirstLine);
```

En el método onCreate de la actividad MainActivity, se inicializan y configuran varios elementos esenciales para el funcionamiento de la aplicación.

Primero, se establece el diseño de la actividad mediante setContentView, que carga la interfaz de usuario desde el archivo XML activity_main.

Luego, se inicializan las variables y componentes necesarios para la lógica de la aplicación y la comunicación Bluetooth. Esto incluye la creación de una instancia de BluJhr, la activación del Bluetooth mediante onBluetooth, y la vinculación de los componentes visuales a las variables Java correspondientes mediante findViewById.

Además, se asignan acciones a los botones mediante setOnClickListener, lo que permite al usuario interactuar con la aplicación. Por ejemplo, el botón buttonBrowse se

utiliza para abrir un archivo de texto, y el botón BtnConectar se emplea para conectar o desconectar dispositivos Bluetooth.

Finalmente, se asignan referencias a los elementos visuales, como textViewContent y textViewFirstLine, que se utilizarán para mostrar el contenido del archivo de texto y su primera línea respectivamente. Estos elementos son esenciales para proporcionar retroalimentación visual al usuario durante la interacción con la aplicación.

```
// Lista bluetooth
listDeviceBluetooth.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
@Override
public void onItemClick(AdapterView<?> adapterView, View view, int i,
long l) {
if (!devicesBluetooth.isEmpty()){
blue.connect(devicesBluetooth.get(i));
blue.setDataLoadFinishedListener(new BluJhr.ConnectedBluetooth() {
@Override
public void onConnectState(@NonNull BluJhr.Connected connected) {
if (connected == BluJhr.Connected.True){
Toast.makeText(getApplicationContext(),"Conectado",Toast.LENGTH_SHORT)
.show();
listDeviceBluetooth.setVisibility(View.GONE);
viewConn.setVisibility(View.VISIBLE);
conectado.setVisibility(View.VISIBLE);
noconectado.setVisibility(View.GONE);
rxReceived();
BtnConectar.setText("Desconectar");
conexion[0] = 1;


}else{
if (connected == BluJhr.Connected.Pending){
Toast.makeText(getApplicationContext(),"Conectando",Toast.LENGTH_SHORT
).show();
}else{
if (connected == BluJhr.Connected.False){
Toast.makeText(getApplicationContext(),"No se pudo
conectar",Toast.LENGTH_SHORT).show();
}else{
if (connected == BluJhr.Connected.Disconnect){
Toast.makeText(getApplicationContext(),"Disconnect",Toast.LENGTH_SHORT
).show();
viewConn.setVisibility(View.VISIBLE);
conectado.setVisibility(View.GONE);
noconectado.setVisibility(View.VISIBLE);
BtnConectar.setText("Conectar");
conexion[0] = 0; // Modificación del valor de conexion

}
}
}
}
```

```
}
});
}
}
});
```

La parte donde se gestiona la biblioteca Bluetooth es crucial para la parte de comunicación. Esta función ha sido originalmente creada por Jose Hidalgo Rodriguez y modificada para este trabajo, adaptándola a cumplir las funcionalidades de la aplicación. La lista permite al usuario seleccionar un dispositivo Bluetooth para conectarse. Si un dispositivo Bluetooth está disponible, al hacer clic en él, se intenta establecer una conexión utilizando la biblioteca BluJhr, y modificada para adaptarse a las necesidades específicas de esta aplicación.

En la parte modificada de esta función, se gestionan las acciones visuales y de lógica de la aplicación en respuesta al estado de la conexión. Por ejemplo, cuando se establece una conexión con éxito (connected == BluJhr.Connected.True), se ocultan la lista de dispositivos Bluetooth y otros elementos visuales relacionados con la conexión, y se muestra un mensaje de "Conectado". Además, el botón de conexión se actualiza para mostrar la opción "Desconectar".

En caso de que la conexión falle (connected == BluJhr.Connected.False), se muestra un mensaje de error indicando que no se pudo establecer la conexión. Y cuando se desconecta (connected == BluJhr.Connected.Disconnect), se restablecen los elementos visuales y el botón de conexión a su estado inicial.

```
//Cuando pulsas  el botn conectar muestra la lista de dispositvos
        BtnConectar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if(conexion[0] ==  0) {
                    listDeviceBluetooth.setVisibility(View.VISIBLE);
                    Btnatras.setVisibility(View.VISIBLE);
                    viewConn.setVisibility(View.GONE);
                }else{
                    blue.closeConnection();
                }
            }
        });
```

Este fragmento de código controla el comportamiento del botón "Conectar" en la interfaz de usuario. Cuando el botón se pulsa, sucede una de dos cosas dependiendo del estado de conexión actual.

Si conexion[0] es igual a 0, lo que indica que no hay una conexión activa, el código muestra la lista de dispositivos Bluetooth disponibles (listDeviceBluetooth) y otros elementos visuales relacionados con la conexión, como el botón "Atrás" (Btnatras), mientras oculta la vista de conexión (viewConn). Esto permite al usuario seleccionar un dispositivo para conectarse.

Por otro lado, si conexion[0] no es igual a 0, lo que significa que hay una conexión activa, se llama al método closeConnection() de la instancia blue de la clase BluJhr para cerrar la conexión Bluetooth existente. Esto se hace para permitir al usuario cerrar la conexión Bluetooth activa.

```
//Volver a antes boton atrás
        Btnatras.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                listDeviceBluetooth.setVisibility(View.GONE);
                viewConn.setVisibility(View.VISIBLE);
                Btnatras.setVisibility(View.GONE);
            }
        });
```

Este bloque de código maneja el comportamiento del botón "Atrás" en la interfaz de usuario. Cuando el botón es presionado, se ejecuta una acción que revierte el estado de la interfaz a la configuración anterior, restaurando la visibilidad de los elementos relevantes.

En concreto, cuando se hace clic en el botón "Atrás", se establece la visibilidad de listDeviceBluetooth en View.GONE, lo que oculta la lista de dispositivos Bluetooth. Al mismo tiempo, se establece la visibilidad de viewConn en View.VISIBLE, lo que vuelve a mostrar la vista de conexión. Además, el botón "Atrás" (Btnatras) se oculta también al establecer su visibilidad en View.GONE.

Esta funcionalidad permite al usuario retroceder desde la selección de dispositivos Bluetooth hacia la vista de conexión, proporcionando una experiencia de navegación coherente y facilitando la interacción con la aplicación.

```
    //Cuando pulsas  el botn conectar muestra la lista de dispositvos
        BtnConectar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if(conexion[0] ==  0) {
                    listDeviceBluetooth.setVisibility(View.VISIBLE);
                    Btnatras.setVisibility(View.VISIBLE);
                    viewConn.setVisibility(View.GONE);
                }else{
                    blue.closeConnection();
                }
```

```
        }
    });
```

Este fragmento de código gestiona la funcionalidad del botón "Conectar" en la interfaz de usuario. Cuando se hace clic en este botón, se ejecuta un evento que realiza dos acciones según el estado de la conexión:

En el caso de que no haya conexión (conexion[0] == 0), se muestran la lista de dispositivos Bluetooth disponibles (listDeviceBluetooth), el botón "Atrás" (Btnatras), y se oculta la vista de conexión (viewConn). En el caso de que haya una conexión establecida (conexion[0] != 0), se cierra la conexión Bluetooth activa mediante el método closeConnection() del objeto blue.

```
        buttonSend.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                if(conexion[0] ==  0) {
                  Toast.makeText(getApplicationContext(),"No
connect",Toast.LENGTH_SHORT).show();
                }else{
                    //blue.bluTx(edtTx.getText().toString() + "\n");
                    blue.bluTx("Imprimir \n");

                }
            }
        });
```

Este bloque es un código residual, de las pruebas de la app, se ha dejado pues es probable que para mejorar la app se vuelva a utilizar. Lo que hace es configurar un listener para el botón "buttonSend", lo que significa que se ejecutará cierto código cuando el botón sea presionado. Dentro de la función onClick, primero se verifica el estado de conexión representado por la variable conexion[0]. Si esta variable es igual a 0, se muestra un mensaje corto ("Toast") indicando que no hay conexión. Si conexion[0] no es igual a 0, se llama al método bluTx del objeto blue para enviar la cadena "Imprimir \n" a través de la conexión Bluetooth.

```
//Boton que borra las consolas de muestra de comandos
        btnborrar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                consola.setText("");
                textViewContent.setText("");
                textViewFirstLine.setText("");

            }
        });
```

```
    }
```

Este fragmento de código crea un listener para el botón "btnborrar" que, al hacer clic en él, borra el contenido de tres TextViews: "consola", "textViewContent" y "textViewFirstLine".

```
    private void openFile() {
        Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
        intent.addCategory(Intent.CATEGORY_OPENABLE);
        intent.setType("text/plain");
        startActivityForResult(intent, READ_REQUEST_CODE);
    }
```

Este método openFile() se encarga de abrir un archivo de texto plano mediante el uso de la acción Intent.ACTION_OPEN_DOCUMENT. Esto lanzará una actividad que permitirá al usuario seleccionar un archivo de texto desde una aplicación de gestión de archivos o almacenamiento del dispositivo movil.

El intent se configura para filtrar solo archivos que sean de tipo text/plain, lo que significa que solo se mostrarán archivos de texto. Luego, se inicia la actividad con startActivityForResult(), lo que indica que esperamos un resultado de esta actividad. Cuando el usuario selecciona un archivo, el resultado se manejará en el método onActivityResult() de la actividad actual.

```
private void rxReceived() {

blue.loadDateRx(new BluJhr.ReceivedData() {
@Override
public void rxDate(@NonNull String receivedData) {
// Verificar si el dato recibido es "XON"
if (receivedData.equals("XON")) {
// Si es "XON", leer la siguiente línea del archivo
// y mostrarla en el TextView correspondiente
leerSiguienteLineaArchivo();
} else {
// Borrar la línea anterior del consola textViewFirstLine
consola.setText("");
// Si no es "XON", simplemente mostrar el dato en el TextView consola
consola.setText(consola.getText().toString() + receivedData + "\n");

}
}
});

}
```

Este método rxReceived() se encarga de recibir datos desde una conexión Bluetooth utilizando la biblioteca BluJhr. Dentro de este método, se llama al método loadDateRx() de la instancia blue de BluJhr, pasando como argumento una instancia anónima de la

interfaz BluJhr.ReceivedData. Esta interfaz proporciona un método rxDate() que se invoca cada vez que se recibe un dato a través de la conexión Bluetooth.

Dentro del método rxDate(), se verifica si el dato recibido del robot es igual a "XON", entendiendo así que está listo para la siguiente línea de código. En caso afirmativo, se llama al método leerSiguienteLineaArchivo() para leer la siguiente línea del archivo y mostrarla en un TextView correspondiente. Si el dato no es "XON", se muestra en un TextView denominado consola para ser monitoreado por el usuario, esto nos indica que el robot no está listo para imprimir y debe esperar a mandar el siguiente código.

```
// Variable para mantener el índice de la línea actual
private int currentIndex = 0;

// Método para leer, mostrar y enviar la siguiente línea del archivo
private void leerSiguienteLineaArchivo() {
// Obtener el contenido actual del TextView textViewContent
String contenido = textViewContent.getText().toString();
// Dividir el contenido en líneas
String[] lineas = contenido.split("\n");
// Verificar si el índice actual es válido
if (currentIndex < lineas.length - 1) {
// Incrementar el índice para apuntar a la siguiente línea
currentIndex++;
// Obtener la siguiente línea del archivo
String siguienteLinea = lineas[currentIndex];
// Mostrar la siguiente línea en el TextView correspondiente
textViewFirstLine.setText(siguienteLinea);
//ENVIO DE LINBEA POR BLUETOOTH
blue.bluTx(siguienteLinea + "\n");

} else {
// Si no hay más líneas disponibles, mostrar un mensaje o realizar alguna
acción
// Esto puede variar según tus necesidades específicas

int contafin  = 0;
contafin++;
Toast.makeText(this,     "No     hay     más     líneas     disponibles",
Toast.LENGTH_SHORT).show();
if(contafin == 3){
consola.setText("");
textViewContent.setText("");
textViewFirstLine.setText("");
blue.closeConnection();
contafin=0;

}
}
}
```

La variable currentIndex que se utiliza para mantener el índice de la línea actual mientras se lee un archivo de texto.

El método llamado leerSiguienteLineaArchivo() se encarga de leer, mostrar y enviar la siguiente línea del archivo.

En primer lugar, el método obtiene el contenido actual del TextView textViewContent y lo divide en líneas utilizando el carácter de nueva línea como delimitador. Luego, verifica si el índice actual (currentIndex) es válido, es decir, si es menor que la longitud total de las líneas menos uno. Si es así, incrementa el índice para apuntar a la siguiente línea, obtiene la siguiente línea del archivo, la muestra en el TextView correspondiente (textViewFirstLine), facilitando la monitorización de usuario de lo que se está enviando, y la envía por Bluetooth utilizando el objeto blue, así le enviará el comando al robot.

En caso de que no haya más líneas disponibles, se muestra un mensaje de aviso mediante un Toast y se realiza alguna acción específica, como borrar los contenidos de ciertos TextView y cerrar la conexión Bluetooth.

```
    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull
String[] permissions, @NonNull int[] grantResults) {
        if (blue.checkPermissions(requestCode,grantResults)){
            Toast.makeText(this, "Exit", Toast.LENGTH_SHORT).show();
            blue.initializeBluetooth();
        }else{
            if(Build.VERSION.SDK_INT < Build.VERSION_CODES.S){
                blue.initializeBluetooth();
            }else{
                Toast.makeText(this,       "Algo       salio       mal",
Toast.LENGTH_SHORT).show();
            }
        }

        super.onRequestPermissionsResult(requestCode,      permissions,
grantResults);
    }
```

Este método onRequestPermissionsResult se encarga de manejar la respuesta del usuario a la solicitud de permisos de la aplicación. Cuando el usuario otorga o niega permisos, este método es llamado automáticamente por el sistema operativo.

Dentro del método, se verifica si los permisos fueron concedidos utilizando el método checkPermissions del objeto blue. Si los permisos fueron concedidos, se muestra un mensaje indicando que la operación fue exitosa y se inicializa la conexión Bluetooth mediante initializeBluetooth.

En caso contrario, se verifica si la versión del sistema operativo es menor que Android 12 (código de versión Build.VERSION_CODES.S). Si es así, se intenta inicializar la conexión Bluetooth nuevamente. De lo contrario, se muestra un mensaje de error.

```
@Override
    protected void onActivityResult(int requestCode, int resultCode,
@Nullable Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        // Tu lógica actual para manejar el resultado de la actividad
de Bluetooth
        if (!blue.stateBluetoooth() && requestCode == 100){
            blue.initializeBluetooth();
        } else {
            if (requestCode == 100){
                devicesBluetooth = blue.deviceBluetooth();
                if (!devicesBluetooth.isEmpty()){
                    ArrayAdapter adapter = new
ArrayAdapter(this,android.R.layout.simple_expandable_list_item_1,devic
esBluetooth);
                    listDeviceBluetooth.setAdapter(adapter);
                } else {
                    Toast.makeText(this, "No tienes vinculados
dispositivos", Toast.LENGTH_SHORT).show();
                }
            }
        }

        // Nueva lógica para abrir y mostrar el contenido de un
archivo de texto
        // Verificar si la solicitud de selección de archivos fue
exitosa
        if (requestCode == READ_REQUEST_CODE && resultCode ==
RESULT_OK) {
            if (data != null) {
                Uri uri = data.getData();
                if (uri != null) {
                    try {
                        // Obtener un flujo de entrada para leer el
archivo seleccionado
                        InputStream inputStream =
getContentResolver().openInputStream(uri);
                        if (inputStream != null) {
                            // Crear un lector de flujo para leer el
archivo línea por línea
                            BufferedReader reader = new
BufferedReader(new InputStreamReader(inputStream));
                            // StringBuilder para almacenar el
contenido completo del archivo
                            StringBuilder stringBuilder = new
StringBuilder();
```

```
                            // Leer la primera línea del archivo y
mostrarla en textViewFirstLine
                            String line;
                            if ((line = reader.readLine()) != null) {
                                textViewFirstLine.setText(line);

stringBuilder.append(line).append("\n");
                            }
                            // Leer el resto del archivo y agregarlo
al StringBuilder
                            while ((line = reader.readLine()) != null)
{

stringBuilder.append(line).append("\n");
                            }
                            // Cerrar el flujo de entrada y el lector
                            inputStream.close();
                            reader.close();
                            // Mostrar el contenido completo del
archivo en textViewContent

textViewContent.setText(stringBuilder.toString());
                        }
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
    }

}
```

En este método onActivityResult, se manejan los resultados de actividades que se lanzan desde esta actividad, como por ejemplo, la selección de un archivo de texto o la activación del Bluetooth.

Primero, se realiza una comprobación para determinar si la actividad de Bluetooth ha sido iniciada y si el Bluetooth está activado. Si no está activado, se inicializa el Bluetooth. Luego, si la solicitud se realizó para obtener dispositivos Bluetooth y la lista de dispositivos no está vacía, se crea un adaptador de arreglo para mostrar los dispositivos Bluetooth disponibles en una lista desplegable en la interfaz de usuario.

Después, se maneja la lógica para abrir y mostrar el contenido de un archivo de texto seleccionado. Se verifica si la solicitud de selección de archivos fue exitosa y si se seleccionó un archivo. Si es así, se obtiene un flujo de entrada para leer el contenido del archivo seleccionado. Luego, se lee el contenido del archivo línea por línea y se agrega al StringBuilder para construir el contenido completo del archivo. Finalmente, se muestra el contenido del archivo en los elementos de la interfaz de usuario textViewFirstLine y textViewContent.

Este método es crucial para manejar la interacción con otras actividades y procesar los datos obtenidos de ellas, lo que contribuye significativamente a la funcionalidad y usabilidad de la aplicación.

En resumen, la aplicación Android combina funcionalidades de Bluetooth y manejo de archivos de texto. Define elementos de interfaz y lógica para conectar dispositivos Bluetooth, enviar y recibir datos, y abrir y mostrar archivos de texto seleccionados por el usuario, para así lograr una comunicación bidireccional con el robot. También gestiona permisos y eventos de interacción con el usuario.

4.3 Interpretación del G-code e Impresión

La interpretación del G-code se realiza en el código integrado en el Arduino. Este código está basado en el realizado por Javier López Martínez autor del anterior proyecto, que a su vez usó la programación de un plotter XY ya existente. En este apartado se verá de forma general la programación utilizada y se profundizará en las partes modificadas del código de partida. En esencia la parte modificada para este proyecto trata la adaptación de parte de la comunicación con el robot. El código está programado como un plotter XY por lo que la base del algoritmo es controlada en base al algoritmo de Bresenham.

El algoritmo de Bresenham se utiliza para dibujar líneas rectas en una cuadrícula de píxeles de manera eficiente y precisa. Se basa en decidir en cada paso si el siguiente píxel de la línea debe estar en la misma fila o en la fila siguiente, minimizando así el error de aproximación. En un plotter XY, este algoritmo traduce las coordenadas de la línea a una serie de movimientos discretos, garantizando que la trayectoria sea lo más recta posible y se aproxime al camino ideal, moviendo el cabezal de dibujo paso a paso en la dirección correcta.

El algoritmo trata o siguiente. Partiendo de una recta trazada entre dos coordenadas dentro del plano XY (x1, y1) y (x2, y2) , el algoritmo utilizará la siguiente ecuación para calcular el error acumulativo en cada paso.

$$Error = 2 * dy - dx$$

Donde dx es la variación en el eje X y dy es la variación en el eje Y.

$$dx = x2 - x1$$

$$dy = y2 - y1$$

La demostración matemática del algoritmo de Bresenham se basa en la relación entre el error acumulativo y la distancia perpendicular entre el píxel actual y la trayectoria deseada. Si el error es positivo, esto indica que el píxel actual está más cerca de la línea superior; si el error es negativo, el píxel está más cerca de la línea inferior. Al seleccionar el píxel más cercano en cada paso, se logra aproximar la trayectoria real a la trayectoria deseada.

El algoritmo comienza en el punto de inicio (x0, y0) y dibuja el primer píxel. En cada paso del eje x, decide si moverse hacia arriba o hacia abajo en el eje y basado en el valor del error acumulativo. Si el error es positivo o cero, incrementa el valor de y; si el error es negativo, mantiene el valor de y.

En cuanto a la programación, se ha configurado y definido todas las variables antes del bucle principal y de las funciones utilizadas. Además, se ha adaptado el resto del código para nuestro robot, incluyendo una reedición de las variables locales y de las propias funciones, con el objetivo de interpretar el código y asegurar el correcto funcionamiento del robot. En la siguiente ilustración se muestra un flujograma de la lógica utilizada para interpretar el G-code.

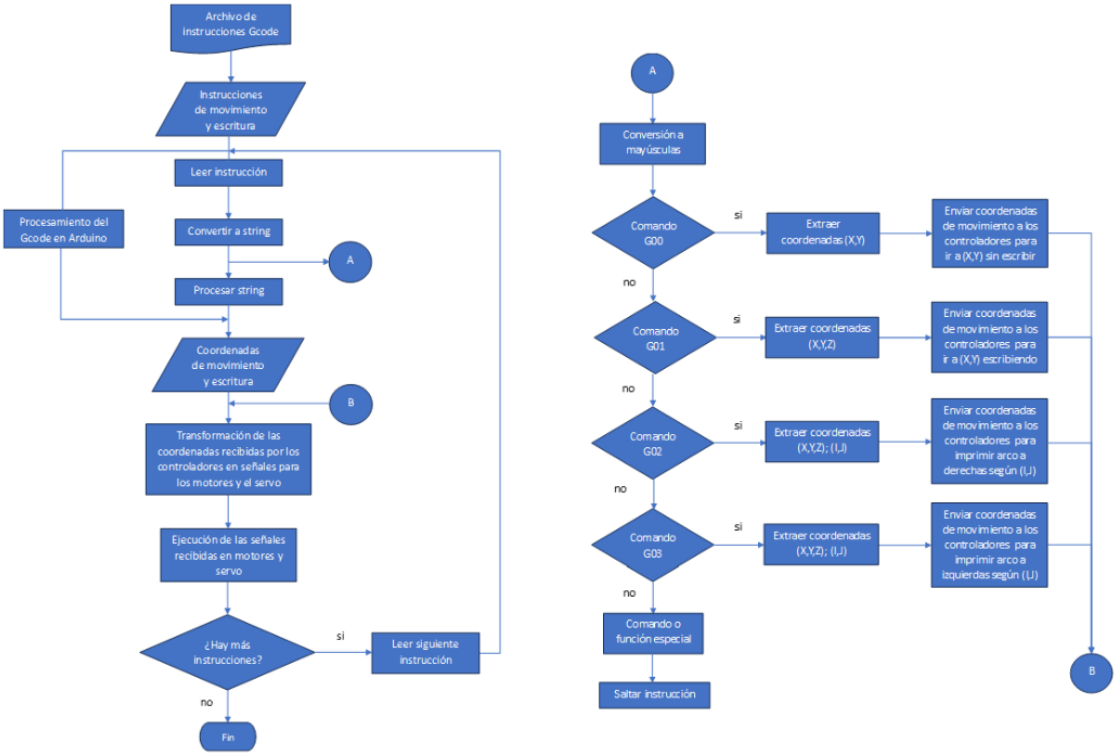


Ilustración 61 Flujograma de la lógica de interpretación de G-code

A continuación, se explica de forma resumida el código implementado en Arduino, indicandodo en las partes donde se ha cambiado el código original para adaptarlo a esta versión.

```
/Biblioteca bluetooth::::::::
#include<SoftwareSerial.h>
//::::::::::::::::::::::::::::::::
// --------------------
// VARIABLES GLOBALES
// --------------------
// Constantes
#define PI 3.14159265358979323846264338327795
#define DOS_PI 6.28318530717958647692528676655 9
// Macros de bit set/clear/check/toggle
#define SET(x,y) (x |=(1<<y))
#define CLR(x,y) (x &= (~(1<<y)))
#define CHK(x,y) (x & (1<<y))
#define TOG(x,y) (x^=(1<<y))
// ConfiguraciÛn del plotter
#define BAUDRATE 9600
#define ANCHO_MAX_PAPEL 420
#define ALTO_MAX_PAPEL 420
#define BOLI 3
#define XOFF 0x13
#define XON 0x11
//INICIALIZACION BLUETOOTH::::::
SoftwareSerial BT(12,13);
//::::::::::::::::::::::::::::::
bool DEBUG = false;
float
FACTOR_ESCALA = 1.0,
ARCO_MAX = 1.0;
int
VALOR_X = 0,
VALOR_Y = 0,
ANTERIOR_X = 0,
ANTERIOR_Y = 0;
// ConfiguraciÛn del Gcode
#define STRING_SIZE 128
char
 BUFFER[STRING_SIZE + 1],
 INPUT_CHAR;
String
 INPUT_STRING,
 SUB_STRING;
int
 POS = 0,
 INICIO,
 FIN;
float
 X,
 Y,
 I,
 J;
```

```
// Entradas de los controladores de los motores
#define IN1_X A0
#define IN2_X A1
#define IN3_X A2
#define IN4_X A3

#define IN1_Y 8
#define IN2_Y 9
#define IN3_Y 10
#define IN4_Y 11
// DefiniciÛn secuencia de medios pasos del motor
byte Motor[8] =
{ B00000001,
 B00000011,
 B00000010,
 B00000110,
 B00000100,
 B00001100,
 B00001000,
 B00001001
};
int p = 0;
byte Sec = 0;
#define PASOS_POR_MM 4096/(PI*48)
#define AVANCE PASOS_POR_MM*5
unsigned long DELAY = 1;
int contar_X = 0;
int contar_X_max = PASOS_POR_MM * ANCHO_MAX_PAPEL;
byte X_Sec;
int contar_Y = 0;
int contar_Y_max = PASOS_POR_MM * ALTO_MAX_PAPEL;
byte Y_Sec;
```

El código comienza en una de las primeras modificaciones, incluyendo la biblioteca SoftwareSerial para manejar la comunicación Bluetooth. Luego, define varias constantes y macros para configuración y control, como las constantes matemáticas PI y DOS_PI, y las macros para manipulación de bits. Se configuran parámetros específicos del plotter, como la tasa de baudios para la comunicación serial (9600), otra modificación, el tamaño máximo del papel (420x420), y los comandos para controlar el flujo de datos (XON y XOFF).

La inicialización del módulo Bluetooth se realiza con los pines 12 y 13 del Arduino. Hay una serie de variables globales definidas para manejar el estado y la configuración del plotter, como FACTOR_ESCALA y ARCO_MAX para la escala y el tamaño de los arcos, y las coordenadas VALOR_X y VALOR_Y para la posición actual del plotter.

Para la configuración del G-code, se define un tamaño de cadena de entrada y se crean buffers y variables para manejar los comandos entrantes. También se especifican los pines para los controladores de los motores en los ejes X e Y.

Se define la secuencia de pasos para controlar los motores de paso a paso, usando un array de bytes que especifica la secuencia de medios pasos. Además, se establecen constantes para calcular los pasos por milímetro y la distancia de avance. Se inicializan variables para contar los pasos en los ejes X e Y, y se configuran los límites máximos de pasos basados en el tamaño del papel.

Finalmente, se establece un retraso mínimo para la operación del motor, y se definen contadores y variables de estado para rastrear el progreso del plotter en ambos ejes.

```
void setup()
{
 // ConfiguraciÛn inicial del motor X
 pinMode(IN1_X, OUTPUT);
 pinMode(IN2_X, OUTPUT);
 pinMode(IN3_X, OUTPUT);
 pinMode(IN4_X, OUTPUT);
 digitalWrite(IN1_X, LOW);
 digitalWrite(IN2_X, LOW);
 digitalWrite(IN3_X, LOW);
 digitalWrite(IN4_X, LOW);
 // ConfiguraciÛn inicial del motor Y
 pinMode(IN1_Y, OUTPUT);
 pinMode(IN2_Y, OUTPUT);
 pinMode(IN3_Y, OUTPUT);
 pinMode(IN4_Y, OUTPUT);
 digitalWrite(IN1_Y, LOW);
 digitalWrite(IN2_Y, LOW);
 digitalWrite(IN3_Y, LOW);
 digitalWrite(IN4_Y, LOW);
 // ConfiguraciÛn del sistema de escritura
 pinMode(BOLI, OUTPUT);
 TCCR2A = _BV(COM2B1) | _BV(COM2B0) | _BV(WGM20);
 TCCR2B = _BV(WGM22) | _BV(CS22) | _BV(CS21) | _BV(CS20);

  OCR2A = 156;
 OCR2B = 148;

 // ConfiguraciÛn del plotter
 memset(BUFFER, '\0', sizeof(BUFFER));
 INPUT_STRING.reserve(STRING_SIZE);
 INPUT_STRING = "";
 // Establecer conexiÛn serie
 Serial.begin(BAUDRATE);
//ESTABLECER cONEXION Bluetooth::::::::::::
 BT.begin(BAUDRATE);
 // Vaciar los buffers
 Serial.flush();
```

```
 //:::::::::::::::::::::::::::::::::::::::::
 BT.flush();

 while (Serial.available())
Serial.read();//:::::::::::::::::::::::::::::Prestar atrencion
 // Comandos mostrados por pantalla
 opc();
 Serial.write(XON);//::::::::::
 //BT.write(XON);
}
```

El código dentro de la función setup() se encarga de configurar los pines y establecer las conexiones necesarias para que el plotter funcione correctamente. Primero, se configuran los pines de salida para los motores de los ejes X e Y, y se aseguran de que todos los pines empiecen en un estado bajo. Luego, se configura el sistema de escritura, incluyendo la configuración del temporizador del Arduino para controlar el motor que mueve el útil de escritura.

Se inicializan varios parámetros del plotter, como el buffer de entrada para el G-code, y se establece la conexión serial a la velocidad definida por la constante BAUDRATE. Se añade otra modificación a este código que establece la conexión Bluetooth utilizando los mismos parámetros de velocidad. Se vacían los buffers de la conexión serial y Bluetooth para asegurarse de que no haya datos residuales.

Finalmente, se muestra un menú de opciones en la pantalla serial, termina la parte del setup() con la última modificación en esta función para el código bluetooth que envía un comando XON para iniciar la comunicación con el plotter, permitiendo que se comience a recibir comandos.

```
//----------------------------------------
// MAIN LOOP
//----------------------------------------
void loop(){
  // Leer instrucción desde el módulo Bluetooth
  if(BT.available()){

    // Leer un carácter del módulo Bluetooth
    INPUT_CHAR = (char)BT.read();
    // Escribir el carácter en el puerto serial
    Serial.write(INPUT_CHAR);
    // Almacenar el carácter en el buffer
    BUFFER[POS++] = INPUT_CHAR;
    // Si se recibe un salto de línea
    if (INPUT_CHAR == '\n'){
      // Vaciar el búfer del puerto serial
      Serial.flush();
      // Enviar el carácter "XOFF" al dispositivo conectado al puerto
serial
      Serial.write("XOFF\n");
```

```
      // Vaciar el búfer del módulo Bluetooth
      BT.flush();
      // Enviar el carácter "XOFF" al dispositivo Bluetooth
      BT.write("XOFF\n");
      // Convertir el contenido del buffer en un String
      INPUT_STRING = BUFFER;
      // Procesar el String
      process(INPUT_STRING);
      // Limpiar el buffer
      memset(BUFFER, '\0', sizeof(BUFFER));
      // Restablecer la posición del buffer a cero
      POS = 0;
      // Restablecer el String de entrada a vacío
      INPUT_STRING = "";
      //Esperar un segundo
      delay(500);//Tambien funciona con 1000
      // Vaciar el búfer del puerto serial
      Serial.flush();
      // Enviar el carácter "XON" al dispositivo conectado al puerto
serial
      Serial.write("XON\n");
      // Vaciar el búfer del módulo Bluetooth
      BT.flush();
      BT.write("XON\n");
      // Enviar el carácter "XON" al dispositivo Bluetooth

    }
  }
  if(Serial.available()){
    BT.write(Serial.read());
  }

}
```

La función loop ha sido completamente reescrita pues es la parte que se encarga de la comunicación, esta función especial de Arduino ejecuta el código de forma constante en función de la lógica del código que esta tenga. El código se encarga de leer instrucciones del módulo Bluetooth que llegan desde la app y procesarlas. Primero, verifica si hay datos disponibles en el módulo Bluetooth. Si es así, lee un carácter y lo escribe en el puerto serial, almacenándolo también en un buffer. Si se recibe un salto de línea, el buffer se procesa como una cadena completa. Esto implica vaciar los buffers del puerto serial y Bluetooth, enviar señales de control ("XOFF" y "XON") para manejar el flujo de datos, convertir el buffer en un String y procesarlo con la función process(). Luego, el buffer se limpia y se reinicia para la siguiente instrucción. Además, el código también envía cualquier dato disponible del puerto serial al módulo Bluetooth, asegurando la comunicación bidireccional entre ambos y una monitorización de esto para un mejor control para el usuario.

```
//----------------------
// OPCIONES ADICIONALES
//----------------------
void opc() {
 Serial.println(F(" "));
 Serial.println(F(" OPCIONES ADICIONALES"));
 Serial.println(F(" "));
 Serial.println(F(" G00 X## Y## ........ mover a (X,Y)
(Boliarriba)"));
 Serial.println(F(" G01 X## Y## ........ mover a (X,Y) (Boliabajo)"));
 Serial.println(F(" C1 S##.## .......... fijar escala deescritura
(1=100%)"));
 Serial.println(F(" C2 ................. adelante boli"));
 Serial.println(F(" C3 ................. atras boli"));
 Serial.println(F(" "));
}
```

En la función opc() se utiliza para la para la confirmación de la funcionalidad del motor, de la que usuario final no hará uso. En esta se definen las opciones adicionales del sistema, las cuales se imprimen en el monitor serial para informar al programador sobre los comandos disponibles. Esta función despliega un conjunto de comandos G-code, que permiten controlar movimientos y configuraciones del plotter. Los comandos incluyen movimientos a coordenadas específicas con el bolígrafo arriba o abajo, ajustes de escala de escritura, y controles para mover el bolígrafo hacia adelante o atrás. Estos mensajes se imprimen utilizando Serial.println para facilitar la interacción con el usuario, proporcionando una guía clara de los comandos disponibles para controlar el plotter.

```
void process(String string)
{
 if (DEBUG)
 {
 Serial.println(string);
 }

 // Convertir string a mayusculas
 INPUT_STRING = string;
 INPUT_STRING.toUpperCase();
 // -------------------------------------
 // G00 - Movimiento lineal sin imprimir
 // -------------------------------------
 if (INPUT_STRING.startsWith("G00"))
 {
 // extraer coordenada X
 INICIO = INPUT_STRING.indexOf('X');
 if (!(INICIO < 0))
 {
 FIN = INICIO + 8;
 SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 1);
 X = SUB_STRING.toFloat();
 }
```

```
// extraer coordenada Y
INICIO = INPUT_STRING.indexOf('Y');
if (!(INICIO < 0))
{
FIN = INICIO + 8;
SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 1);
Y = SUB_STRING.toFloat();
}
subir_boli();
ir_a(X, Y);
}
// ------------------------------------
// G01 - Movimiento lineal imprimiendo
// ------------------------------------
if (INPUT_STRING.startsWith("G01"))
{
// extraer coordenada X
INICIO = INPUT_STRING.indexOf('X');
if (!(INICIO < 0))
{
FIN = INICIO + 8;
SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 1);
X = SUB_STRING.toFloat();
}
// extraer coordenada Y
INICIO = INPUT_STRING.indexOf('Y');

if (!(INICIO < 0))
{
FIN = INICIO + 8;
SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 1);
Y = SUB_STRING.toFloat();
}
bajar_boli();
ir_a(X, Y);
}
// ---------------------------------------
// G02 - Imprimir arco en sentido horario
// ---------------------------------------
if (INPUT_STRING.startsWith("G02"))
{
// extraer coordenada X
INICIO = INPUT_STRING.indexOf('X');
if (!(INICIO < 0))
{
FIN = INPUT_STRING.indexOf('.', INPUT_STRING.indexOf('X'));
SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 7);
X = SUB_STRING.toFloat();
}
// extraer coordenada Y
INICIO = INPUT_STRING.indexOf('Y');
if (!(INICIO < 0))
{
FIN = INPUT_STRING.indexOf('.', INPUT_STRING.indexOf('Y'));
SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 7);
```

```
 Y = SUB_STRING.toFloat();
 }
 // extraer coordenada I
 INICIO = INPUT_STRING.indexOf('I');
 if (!(INICIO < 0))
 {
 FIN = INPUT_STRING.indexOf('.', INPUT_STRING.indexOf('I'));
 SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 7);
 I = SUB_STRING.toFloat();
 }
 // extraer coordenada J
 INICIO = INPUT_STRING.indexOf('J');
 if (!(INICIO < 0))
 {
 FIN = INPUT_STRING.indexOf('.', INPUT_STRING.indexOf('J'));
 SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 7);
 J = SUB_STRING.toFloat();
 }
 bajar_boli();
 arco_horario(X, Y, I, J);
 }
// ------------------------------------------------
 // G03 - Imprimir arco en sentido antihorario
 // ------------------------------------------------
 if (INPUT_STRING.startsWith("G03"))
 {
// extraer coordenada X
 INICIO = INPUT_STRING.indexOf('X');
 if (!(INICIO < 0))
 {
 FIN = INPUT_STRING.indexOf('.', INPUT_STRING.indexOf('X'));
 SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 7);
 X = SUB_STRING.toFloat();
 }
 // extraer coordenada Y
 INICIO = INPUT_STRING.indexOf('Y');
 if (!(INICIO < 0))
 {
 FIN = INPUT_STRING.indexOf('.', INPUT_STRING.indexOf('Y'));
 SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 7);
 Y = SUB_STRING.toFloat();
 }
 // extraer coordenada I
 INICIO = INPUT_STRING.indexOf('I');
 if (!(INICIO < 0))
 {
 FIN = INPUT_STRING.indexOf('.', INPUT_STRING.indexOf('I'));
 SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 7);
 I = SUB_STRING.toFloat();
 }
 // extraer coordenada J
 INICIO = INPUT_STRING.indexOf('J');
 if (!(INICIO < 0))
 {
 FIN = INPUT_STRING.indexOf('.', INPUT_STRING.indexOf('J'));
```

```
 SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 7);
 J = SUB_STRING.toFloat();
 }
 bajar_boli();
 arco_antihorario(X, Y, I, J);
 }
 // ---------
 // OPCIONES
 // ---------
 if (INPUT_STRING.startsWith("opc"))
 {
 opc();
 }
 // ----------------------------
 // C1 - Fijar factor de escala
 // ----------------------------
 if (INPUT_STRING.startsWith("C1"))
 {
 INICIO = INPUT_STRING.indexOf('S');
 if (!(INICIO < 0))
 {
 FIN = INICIO + 6;
 SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN);
 FACTOR_ESCALA = SUB_STRING.toFloat();
 Serial.print(F("Dibujando ")); Serial.print(FACTOR_ESCALA *
100); Serial.println(F("%"));

}
 else
 {
 Serial.println(F("Factor de escala no v·lido!!"));
 }
 }
 // ---------------------------
 // C2 - Escritura desactivada
 // ---------------------------
 if (INPUT_STRING.startsWith("C2"))
 {
 subir_boli();
 }
 // ------------------------
 // C3 - Escritura activada
 // ------------------------
 if (INPUT_STRING.startsWith("C3"))
 {
 bajar_boli();
 }
}
```

La función process(String string) se encarga de interpretar y ejecutar comandos G-code recibidos en forma de cadena. Primero, convierte la cadena a mayúsculas para uniformar el procesamiento. Luego, evalúa diferentes comandos G-code, como G00

para movimientos sin imprimir, G01 para movimientos imprimiendo, y G02 para imprimir arcos en sentido horario.

Para los comandos G00 y G01, extrae las coordenadas X e Y de la cadena, convierte estas subcadenas a valores de punto flotante y llama a funciones específicas (subir_boli() y bajar_boli()) para controlar el bolígrafo, seguido de ir_a(X, Y) para mover a las coordenadas especificadas.

El comando G02, además de las coordenadas X e Y, también extrae los parámetros I y J, necesarios para definir el centro del arco, y luego llama a arco_horario(X, Y, I, J) para dibujar el arco correspondiente.

El bloque if (DEBUG) imprime la cadena de entrada en el monitor serial si la depuración está activada. Esto proporciona retroalimentación útil durante el desarrollo y la resolución de problemas.

La función sigue interpretando y ejecutando los comandos G-code. Para el comando G03, que imprime un arco en sentido antihorario, extrae las coordenadas X e Y, así como los parámetros I y J para definir el centro del arco. Luego llama a bajar_boli() para activar la escritura y a arco_antihorario(X, Y, I, J) para trazar el arco.

Para comandos adicionales, como "opc", se llama a la función opc() que imprime las opciones disponibles.

El comando C1 permite fijar el factor de escala. Extrae el valor especificado con 'S', lo convierte a punto flotante y ajusta FACTOR_ESCALA. Si no se proporciona un valor válido, se muestra un mensaje de error en el monitor serial.

Los comandos C2 y C3 controlan la escritura. C2 llama a subir_boli() para desactivar la escritura, mientras que C3 llama a bajar_boli() para activarla. Estos comandos permiten cambiar el estado del bolígrafo según las instrucciones recibidas.

```
// FUNCION DE MOVIMIENTO
void ir_a(float x, float y)
{
 // Aplicar factor de escala
 VALOR_X = round(x * PASOS_POR_MM * FACTOR_ESCALA);
 VALOR_Y = round(y * PASOS_POR_MM * FACTOR_ESCALA);
 // Dibujar linea entre coordenadas escaladas
 traza_linea(ANTERIOR_X, ANTERIOR_Y, VALOR_X, VALOR_Y);
 // ----- ⁄ltimas coordinadas escaldas
 ANTERIOR_X = VALOR_X;
 ANTERIOR_Y = VALOR_Y;
}
```

La función ir_a(float x, float y) maneja el movimiento del plotter hacia las coordenadas especificadas. Primero, ajusta las coordenadas x e y aplicando el factor de escala y convirtiéndolas en valores de pasos por milímetro. Luego, llama a traza_linea(ANTERIOR_X, ANTERIOR_Y, VALOR_X, VALOR_Y) para dibujar una línea desde las coordenadas previas hasta las nuevas coordenadas escaladas. Finalmente, actualiza las variables ANTERIOR_X y ANTERIOR_Y con las últimas coordenadas escaladas para utilizarlas en futuros movimientos.

```
// --------------------
// DIBUJAR UNA LINEA
// --------------------
void traza_linea(int x1, int y1, int x2, int y2)
{
 // Variables locales
 int
 x = x1,
 y = y1,
 dy,
 dx,
 pendiente,
 largo,
 corto,
 maximo,
 error,
 margen;
 // Definir eje largo y eje corto
 dy = y2 - y1;

 dx = x2 - x1;
 largo = max(abs(dy), abs(dx));
 corto = min(abs(dy), abs(dx));
 // Escalar los valores de Bresenham por 2*largo
 error = -largo;
 margen = 0;
 maximo = (largo << 1);
 pendiente = (corto << 1);
 // Inicializar el cambio de flanco
 bool cambioXY = true;
 if (abs(dx) >= abs(dy)) cambioXY = false;
 // Simular que siempre estamos en el octante 0
 for (int i = 0; i < largo; i++)
 {
 // Movimiento a izquierda/derecha sobre eje X
 if (cambioXY)
 {
 if (dy < 0)
 {
 y--;
 atras();
 }
 else
 {
```

```
 y++;
 adelante();
 }
 }
 else
 {
 if (dx < 0)
 {
 x--;
 izquierda();
 }
 else
 {
 x++;
 derecha();
 }
 }
 // Movimiento arriba/abajo en eje Y
 error += pendiente;
 if (error > margen)
 {
 error -= maximo;
 // Movimiento arriba/abajo sobre eje Y
 if (cambioXY)
 {
 if (dx < 0)
 {
 x--;
 izquierda();
 }
 else
 {

x++;
 derecha();
 }
 }
 else
 {
 if (dy < 0)
 {
 y--;
 atras();
 }
 else
 {
 y++;
 adelante();
 }
 }
 }
 }
}
```

La función traza_linea(int x1, int y1, int x2, int y2) implementa el algoritmo de Bresenham para dibujar una línea entre dos puntos en el sistema de coordenadas del plotter. Se inicia definiendo y calculando variables locales para manejar las diferencias en las coordenadas y determinar los ejes largo y corto de la línea a trazar.

Primero, calcula la diferencia en x (dx) y en y (dy), y determina cuál es el eje largo y cuál es el corto. Luego, escala los valores de Bresenham y establece los errores iniciales y máximos para la trayectoria.

La variable cambioXY se utiliza para determinar si el eje principal de movimiento es x o y. Durante la iteración, la función mueve el plotter paso a paso en el eje correspondiente. Si el error acumulado supera un cierto margen, realiza un ajuste adicional en el eje secundario para mantener la línea lo más cercana posible a la trayectoria deseada.

Dependiendo de la dirección del movimiento, la función llama a adelante(), atras(), izquierda(), o derecha() para mover el plotter en la dirección correcta, ya sea en el eje x o y. Esta lógica asegura que el plotter trace una línea recta de manera precisa entre los dos puntos dados, ajustando continuamente su posición en función del error acumulado.

```
//--------------------------------
// Mover el boli un paso hacia x-
//--------- ----------------------
void izquierda()
{
 // Rotar motor en sentido antihorario
 contar_X--;
 if (contar_X >= 0)
 {
 p = contar_X % 8;
 Sec = PORTC;
 Sec = Sec & B11110000;
 Sec = Sec | Motor[p];
 PORTC = Sec;
 delay(DELAY);
 }
}
//--------------------------------
// Mover el boli un paso hacia x+
//--------- ----------------------
void derecha()
{
 // Rotar motor en sentido horario
 contar_X++;
 if (contar_X < contar_X_max)
 {
 p = contar_X % 8;
 Sec = PORTC;
 Sec = Sec & B11110000;
 Sec = Sec | Motor[p];
 PORTC = Sec;
```

```
 delay(DELAY);
 }
}
//--------------------------------
// Mover el boli un paso hacia y+
//--------- ----------------------
void adelante()

{
 // Rotar motor en sentido horario
 contar_Y++;
 if (contar_Y < contar_Y_max)
 {
 p = contar_Y % 8;
 Sec = PORTB;
 Sec = Sec & B11110000;
 Sec = Sec | Motor[p];
 PORTB = Sec;
 delay(DELAY);
 }
}
//--------------------------------
// Mover el boli un paso hacia y-
//--------------------------------
void atras()
{
 // Rotar motor sentido antihorario
 contar_Y--;
 if (contar_Y >= 0)
 {
 p = contar_Y % 8;
 Sec = PORTB;
 Sec = Sec & B11110000;
 Sec = Sec | Motor[p];
 PORTB = Sec;
 delay(DELAY);
 }
}
```

En este bloque se encuentran las funciones específicas para mover el boli del plotter en las cuatro direcciones cardinales: izquierda, derecha, adelante y atrás, implementando así movimientos lineales precisos.

La función izquierda() decrementa el contador contar_X y rota el motor en sentido antihorario si el contador está dentro de los límites, actualizando los pines correspondientes en el puerto PORTC para realizar el movimiento.

La función derecha() incrementa el contador contar_X y rota el motor en sentido horario, siempre que el contador no exceda el máximo permitido (contar_X_max). Similar a la función de moverse hacia la izquierda, se actualizan los pines en el puerto PORTC.

La función adelante() incrementa contar_Y y rota el motor en sentido horario para mover el boli hacia adelante, actualizando el puerto PORTB con el patrón adecuado de la secuencia del motor.

Finalmente, la función atras() decrementa contar_Y y rota el motor en sentido antihorario, moviendo el boli hacia atrás si el contador contar_Y está dentro de los límites, y actualizando también el puerto PORTB.

Cada una de estas funciones asegura que el motor paso a paso realice el movimiento deseado mediante un ajuste preciso de los pines y una breve pausa (delay(DELAY)) para estabilizar la operación.

```
//--------------------------------------
// Dibujar arco en sentido horario (G02)
//--------------------------------------
void arco_horario(float x, float y, float i, float j)
{
 if ((i < -420) || (i > 420) || (j < -420) || (j > 420))
 {
 ir_a(x, y);
 }
 else
 {
 // Variables locales
 float
 valorX = ANTERIOR_X / (PASOS_POR_MM * FACTOR_ESCALA),
 valorY = ANTERIOR_Y / (PASOS_POR_MM * FACTOR_ESCALA),
 valorsigX = x,
 valorsigY = y,
 nuevaX,
 nuevaY,
 I = i,
 J = j,
 circX = valorX + I,
 circY = valorY + J,
 delta_x,
 delta_y,
 distancia,
 radio,
 alpha,
 beta,

 arco,
 angulo,
 sig_angulo;
 // Calcular arcoo
 delta_x = valorX - valorsigX;
 delta_y = valorY - valorsigY;
 distancia = sqrt(delta_x * delta_x + delta_y * delta_y);
 radio = sqrt(I * I + J * J);
 alpha = 2 * asin(distancia / (2 * radio));
 arco = alpha * radio;
```

```
 // Dividir ·ngulo en partes
 int segmentos = 1;
 if (arco > ARCO_MAX)
 {
 segmentos = (int)(arco / ARCO_MAX);
 beta = alpha / segmentos;
 }
 else
 {
 beta = alpha;
 }
 // Calcular ·ngulo actual
 angulo = atan2(-J, -I);
 if (angulo <= 0) angulo += 2 * PI;
 // Coordenadas intermedias sentido horario
 sig_angulo = angulo;
 for (int segmento = 1; segmento < segmentos; segmento++)
 {
 sig_angulo -= beta;
 if (sig_angulo < 0) sig_angulo += 2 * PI;
 nuevaX = circX + radio * cos(sig_angulo);
 nuevaY = circY + radio * sin(sig_angulo);
 ir_a(nuevaX, nuevaY);
 }
 // Dibujar final de linea
 ir_a(valorsigX, valorsigY);
 }
}
//--------------------------------------
// Dibujar arco en sentido antihorario
//--------------------------------------
void arco_antihorario(float x, float y, float i, float j)
{
 if ((i < -420) || (i > 420) || (j < -420) || (j > 420))
 {
 ir_a(x, y);
 }
 else
 {
 // Variables locales
 float
 valorX = ANTERIOR_X / FACTOR_ESCALA,
 valorY = ANTERIOR_Y / FACTOR_ESCALA,
 valorsigX = x,
 valorsigY = y,

 nuevaX,
 nuevaY,
 I = i,
 J = j,
 circX = valorX + I,
 circY = valorY + J,
 delta_x,
 delta_y,
 distancia,
```

```
 radio,
 alpha,
 beta,
 arco,
 angulo,
 sig_angulo;
 // Calcular arco
 delta_x = valorX - valorsigX;
 delta_y = valorY - valorsigY;
 distancia = sqrt(delta_x * delta_x + delta_y * delta_y);
 radio = sqrt(I * I + J * J);
 alpha = 2 * asin(distancia / (2 * radio));
 arco = alpha * radio;
 // Dividir ·ngulo en partes
 int segmentos = 1;
 if (arco > ARCO_MAX)
 {
 segmentos = (int)(arco / ARCO_MAX);
 beta = alpha / segmentos;
 }
 else
 {
 beta = alpha;
 }
 // Calcular ·ngulo actual
 angulo = atan2(-J, -I);
 if (angulo <= 0) angulo += 2 * PI;
 // Coordenadas intermedias sentido antihorario
 sig_angulo = angulo;
 for (int segmento = 1; segmento < segmentos; segmento++)
 {
 sig_angulo += beta;
 if (sig_angulo > 2 * PI) sig_angulo -= 2 * PI;
 nuevaX = circX + radio * cos(sig_angulo);
 nuevaY = circY + radio * sin(sig_angulo);
 ir_a(nuevaX, nuevaY);
 }
 // Dibujar final de linea
 ir_a(valorsigX, valorsigY);
 }
}
```

Aquí se presentan dos funciones esenciales para el movimiento circular del plotter, permitiendo dibujar arcos tanto en sentido horario como antihorario, utilizando las coordenadas proporcionadas junto con los desplazamientos i y j.

La función arco_horario(float x, float y, float i, float j) se encarga de dibujar un arco en el sentido horario. Primero, verifica si los valores de i y j están dentro de un rango aceptable (-420 a 420). Si no lo están, simplemente mueve el boli a las coordenadas x y y. Si están dentro del rango, calcula las variables locales necesarias como las coordenadas iniciales, los desplazamientos, la distancia y el radio del arco, y el ángulo del arco.

Luego, el arco se divide en segmentos pequeños si es necesario para asegurar un movimiento suave. Para cada segmento, calcula las nuevas coordenadas y llama a la función ir_a para mover el boli a esas coordenadas.

De manera similar, la función arco_antihorario(float x, float y, float i, float j) se utiliza para dibujar arcos en sentido antihorario. Esta función sigue la misma lógica que la de sentido horario, pero ajusta los cálculos del ángulo para moverse en la dirección opuesta. Al igual que la función anterior, verifica los valores de i y j, calcula las variables locales, divide el arco en segmentos si es necesario y mueve el boli a las nuevas coordenadas en cada segmento.

Ambas funciones aseguran que los movimientos del boli sean precisos y suaves, respetando las coordenadas y desplazamientos proporcionados, y utilizando el algoritmo de Bresenham para dibujar los arcos de manera eficiente.

```
//--------------------------------
// Levantar el ˙til de escritura
//--------------------------------
void subir_boli()
{

OCR2B = 148;
 delay(250);
}
//-----------------------------
// Bajar el ˙til de escritura
//-----------------------------
void bajar_boli()
{
 OCR2B = 140;
 delay(250);
}
```

Estas dos funciones finales controlan el movimiento del útil de escritura, levantándolo y bajándolo según sea necesario.

La función subir_boli() se encarga de levantar el útil de escritura. Simplemente modifica el valor de OCR2B, lo que controla la altura del útil. Luego, agrega un pequeño retraso de 250 milisegundos para asegurar que el útil se levante completamente antes de continuar.

Por otro lado, la función bajar_boli() se utiliza para bajar el útil de escritura. Al igual que la función anterior, ajusta el valor de OCR2B para controlar la altura del útil y luego espera 250 milisegundos para permitir que el útil descienda completamente.

Estas funciones aseguran que el útil de escritura esté en la posición adecuada antes de iniciar o finalizar un trazo, garantizando así un dibujo preciso y consistente.

5. Futuros Desarrollos y mejoras

Se sugiere una revisión y mejora del diseño de la interfaz de usuario de la aplicación móvil. Actualmente, la interfaz es muy básica y podría beneficiarse de una actualización que la haga más intuitiva y atractiva visualmente. Una interfaz de usuario más amigable y moderna no solo mejorará la experiencia del usuario, sino que también aumentará la funcionalidad y eficiencia de la aplicación.

Para mejorar el transporte y la robustez del robot, se propone el diseño de una placa base que integre todos los componentes electrónicos. Esto permitirá una mejor organización interna y facilitará el mantenimiento y las reparaciones.

Junto con la placa base, se recomienda el diseño de una carcasa que cubra completamente el robot. Esta carcasa no solo mejorará la apariencia estética del dispositivo, sino que también protegerá los componentes internos de daños físicos y del polvo, aumentando así la durabilidad y fiabilidad del robot.

Implementar estas mejoras hará que el proyecto sea más profesional y funcional, aumentando su viabilidad para aplicaciones prácticas y su atractivo para potenciales usuarios o inversores.

6.Bibliografía

Accedido el día 20/05/2024

- **TFG de Javier Lopez Martinez DISEÑO DE UN ROBOT MÓVIL PARA IMPRESIÓN DE DOCUMENTOS firigido por Carlos torre**
    - https://digibuo.uniovi.es/dspace/bitstream/handle/10651/47853/?sequence=1
- **Piezas 3D**
    - https://grabcad.com
- **Definición de impresión**
    - https://definicion.de/impresion/#google_vignette
- **Tipos de impresoras**

    - https://www.podiprint.com/impresion-bajo-demanda/impresion-digital-principales-metodos-tipos/

- **Flexografía**
    - https://www.esagraf.com/caracteristicas-impresion-flexografica/
- **Sublimación**
    - https://www.artlex.com/es/blog/que-es-la-impresion-por-sublimacion/

Accedido el día 21/05/2024

- **Impresión 3D**
    - https://www.adslzone.net/reportajes/tecnologia/impresion-3d/
    - https://blog.igus.es/knowledge-base/que-es-el-metodo-fdm/
    - https://xometry.eu/es/impresion-3d/
    - https://es.3dsystems.com/

Accedido el día 22/05/2024

- **Robótica**
    - https://fi.ort.edu.uy/blog/que-es-la-robotica-y-cuales-son-sus-usos
- **Métodos de comunicación inalámbrica de corto alcance**
    - https://www.mokosmart.com/es/short-range-wireless-communication-technology-vs-long-range-wireless-communication-technology/
- **Información sobre RoboThespian de Engineered Arts**
    - https://www.engineeredarts.co.uk/es/inicio/
- **El robot ANYmal de ANYbotics**
    - https://www.anybotics.com/robotics/anymal/

Accedido el día 25/05/2024

- **Información sobre tecnologías inalámbricas de corto alcance**
    - Artículo de IEEE "An Overview of Bluetooth Wireless Technology"
        - https://www.ieee.org/

Accedido el día 27/05/2024

- **Para tecnología wifi directTrabajo fin de master "EVALUACIÓN DE LA TECNOLOGÍA WI-FI DIRECT EN ESCENARIOS DE MOVILIDAD" por MIRAVALLES FUENTES, DANIEL**
    - https://digibuo.uniovi.es/dspace/bitstream/handle/10651/47853/?sequence=1

- **Tecnología NFC**
    - https://www.dipolerfid.es/blog-rfid/que-es-nfc#:~:text=%C2%BFC%C3%B3mo%20funciona%20la%20tecnolog%C3%ADa%20NFC,campo%20electromagn%C3%A9tico%20de%20baja%20frecuencia.
- **Transmisión con infrarrojos**
    - https://www.digikey.es/en/maker/tutorials/2021/understanding-the-basics-of-infrared-communications

Accedido el día 28/05/2024

- **HC-05**
    - Tutorial completo sobre el módulo Bluetooth HC-05 y su conexión con Arduino
        - https://www.makerguides.com/es/arduino-and-hc-05-bluetooth-module-complete-tutorial/#Hardware_Components
- **Datasheet HC05**
    - https://components101.com/sites/default/files/component_datasheet/HC-05%20Datasheet.pdf
- **Batería**
    - https://www.amazon.es/QTshine-Entradas-Cargador-Port%C3%A1til-Compatible/dp/B0CKLBW6QT/ref=sr_1_22?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=27XEYHOH74O85&dib=eyJ2IjoiMSJ9.6_ZphnoTKIinUiRR5ikkg3Hl6FdKvZOr_EdiMp_BwOz3SALislVEL28358qvyMA00dd7vUuDy4B-OKEm8gggRwisLKa0s_-Br4J4pxz_MSSraNSlm7uztMkHfCDBSd2qV28jcJVoxE6OEFTLBADJs47-r2VXr5ND4OG9m1TYimv-S0LMthrBr5h8LFZukyKr0dDk9meSxuJFIqbkQl9XHy26ZGWPGQZk6PUihrP85pGkeGe1Zcb67cN3imQ2Kj3BTTVjYBVowvL547ytjm7t6EoR17AN8fj3RW5hT6xjhfw.gHGjgnmEdW9tNoJPSReLlS27SnevCFiaD7ysMsBhoPA&dib_tag=se&keywords=bateria+carga+3+salidas&qid=1716892330&sprefix=bateria+carga+3+salid%2Caps%2C162&sr=8-22
- **Página oficial de Arduino**
    - https://docs.arduino.cc/hardware/uno-rev3/

Accedido el día 29/05/2024

- **Ruedas Omnidireccionales**
    - https://es.aliexpress.com/item/32798917452.html?spm=a2g0o.productlist.main.71.4a88XtUUXtUUkr&algo_pvid=e8e9849a-f973-4790-aea0-0bf8f954f0aa&algo_exp_id=e8e9849a-f973-4790-aea0-0bf8f954f0aa-35&pdp_npi=4%40dis%21EUR%212.46%212.34%21%21%212.61%212.48%21%40211b612817169633802902162e32a5%211200002054755

1042%21sea%21ES%210%21AB&curPageLogUid=1lBnVNvB0JAz&utparam-url=scene%3Asearch%7Cquery_from%3A

- https://www.rulmeca.com/es/productos_unit/catalogue/2/transporte_industrial/8/componentes_para_transportadores_por_gravedad/39/ruedas_ow

- **Servomotores**
  - https://www.ingmecafenix.com/electronica/motores-electronicos/servomotor/
- **Driver Motores paso a paso**
  - https://www.cbjmotor.es/drivers-para-motores-paso-a-paso/#:~:text=Los%20drivers%20para%20motores%20paso%20a%20paso%20son,eficiencia%20energ%C3%A9tica%2C%20control%20flexible%20y%20protecci%C3%B3n%20del%20motor.
  - https://www.inventable.eu/2018/02/09/uln2003-driver-salida-microcontroladores/
  - https://octopart.com/datasheet/uln2003-stmicroelectronics-3748191?msclkid=dc89291323511b0074c74f440ed93fbc&utm_source=bing&utm_medium=cpc&utm_campaign=b_cpc_emea-es_search_dsa_english_en_usd_all-categories&utm_term=semiconductors&utm_content=Discrete%20Semiconductors%20DSA

Accedido el día 30/05/2024

- **Motores paso a paso**
  - Libro "Motores paso a paso" por Francisco Conti (2005)
- **Revista de la ciencia aplicada**
  - http://investigacion.utc.edu.ec/index.php/ciya/article/view/375/355

Accedido el día 2/06/2024

- **Impresora 3D información**
  - https://www.google.com/imgres?imgurl=https://tresdenou.com/1966-medium_default/artillery-genius.jpg&tbnid=sqgcAWJQoRls4M&vet=1&imgrefurl=https://tresdenou.com/fr/741-artillery-genius.html&docid=pbyvCrLaej7DUM&w=370&h=423&hl=es-ES&source=sh/x/im/m6/4&kgs=d6ef31630f0cfcda&shem=abme,ssim,ssimpx,trie
- **PLA**
  - https://eu.elegoo.com/en-es/products/elegoo-pla-filament-1-75mm-3d-printer-filament-1kg-spool-2-2-lbs?variant=45521201004820

- **Mini breadboard adhesiva**
    - https://tienda.bricogeek.com/herramientas-de-prototipado/211-mini-breadboard-adhesiva.html?gad_source=1&gclid=Cj0KCQjwsPCyBhD4ARIsAPaaRf3WawgP4Igr47pYAY5rFnlswxg77ZGkTS9LJL-OopPdUrtGWEDi-gkaAhfiEALw_wcB

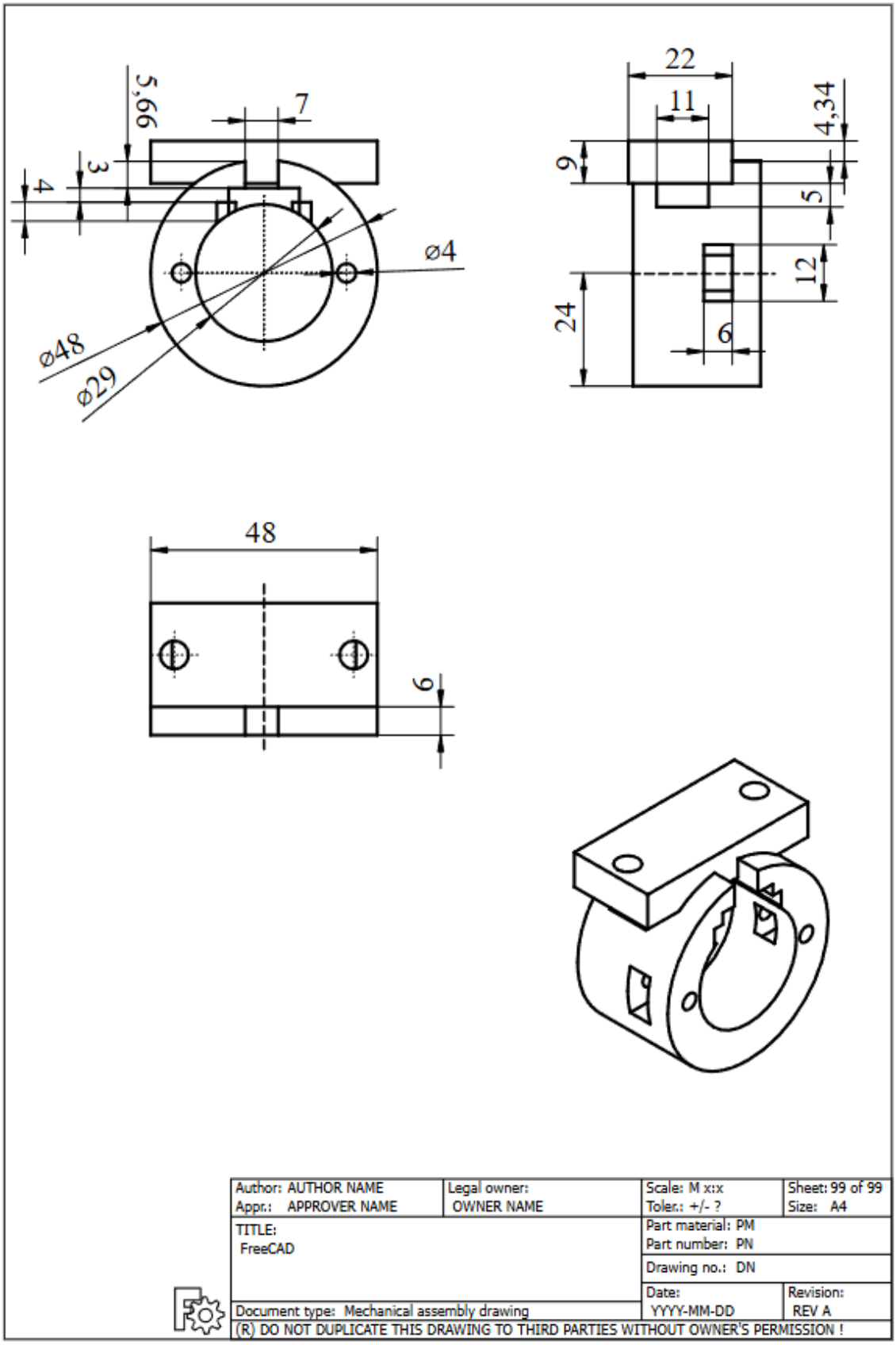
Accedido el día 03/06/2024

- **Gcode**
    - https://www.3dnatives.com/es/g-code-proceso-impresion-3d-230920212/

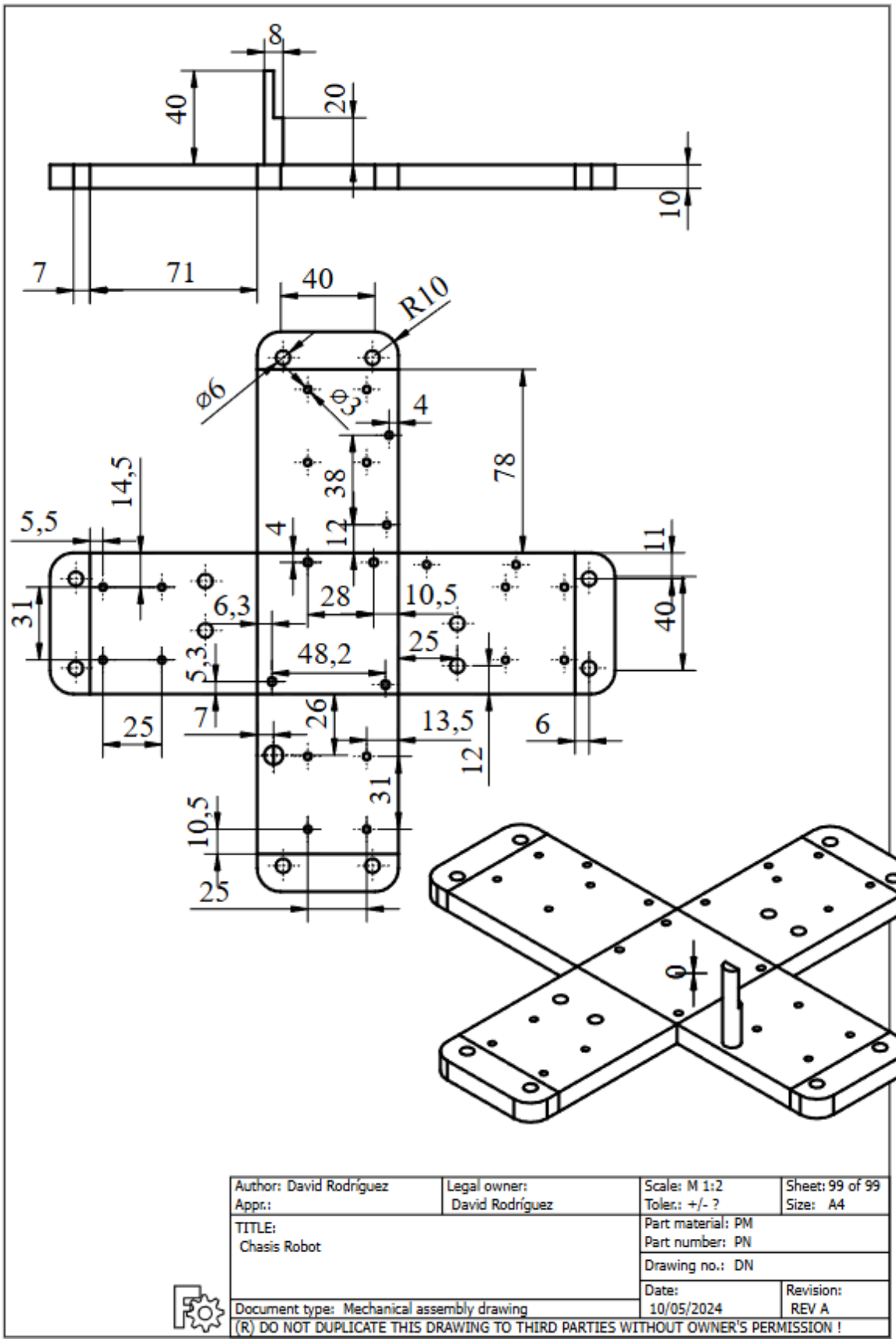
Accedido durante la realización

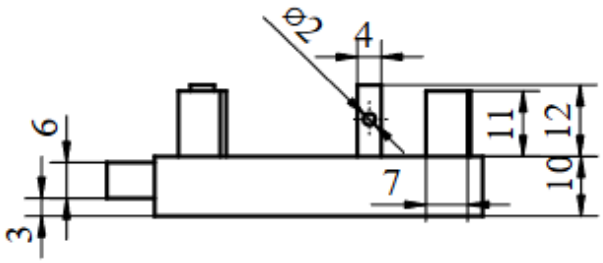
- **TFG de Javier Lopez Martinez DISEÑO DE UN ROBOT MÓVIL PARA IMPRESIÓN DE DOCUMENTOS firigido por Carlos torre**

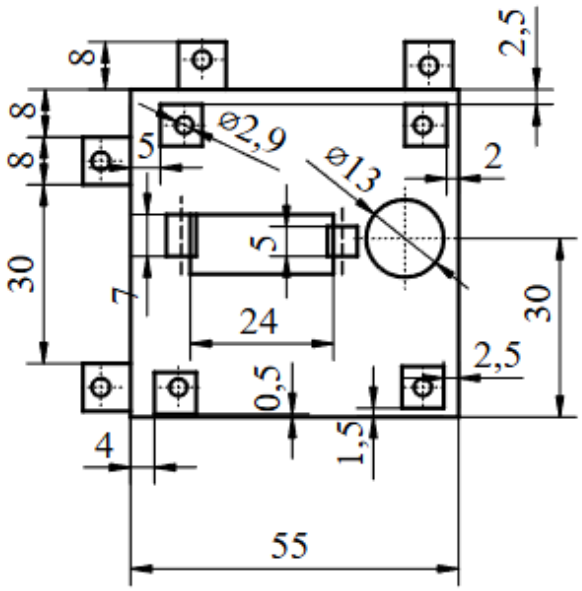
Documento 2: Planos

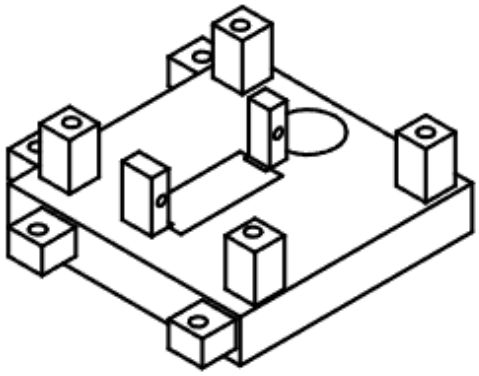


Author: AUTHOR NAME Appr.: APPROVER NAME	Legal owner: OWNER NAME	Scale: M x:x Toler.: +/- ?	Sheet: 99 of 99 Size: A4
TITLE: FreeCAD		Part material: PM Part number: PN	
		Drawing no.: DN	
Document type: Mechanical assembly drawing		Date: YYYY-MM-DD	Revision: REV A
(R) DO NOT DUPLICATE THIS DRAWING TO THIRD PARTIES WITHOUT OWNER'S PERMISSION !			

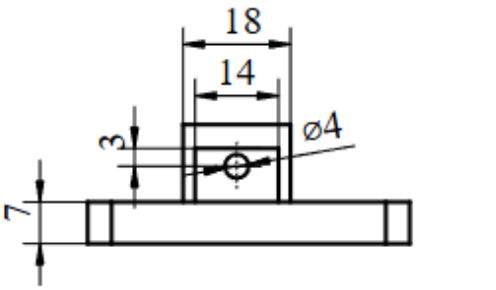

8
40
20
10
7
71
40
R10
ø6
ø3
4
78
38
14,5
5,5
4
12
11
31
6,3
28
10,5
40
5,3
48,2
25
25
7
26
13,5
6
12
31
10,5
25
Author: David Rodríguez
Appr.:
Legal owner:
David Rodríguez
Scale: M 1:2
Toler.: +/- ?
Sheet: 99 of 99
Size: A4
TITLE:
Chasis Robot
Part material: PM
Part number: PN
Drawing no.: DN
Date:
10/05/2024
Revision:
REV A
Document type: Mechanical assembly drawing
(R) DO NOT DUPLICATE THIS DRAWING TO THIRD PARTIES WITHOUT OWNER'S PERMISSION !

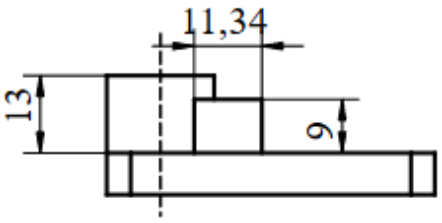


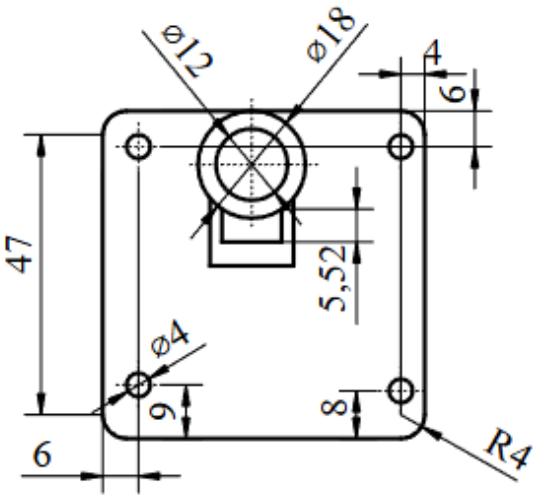


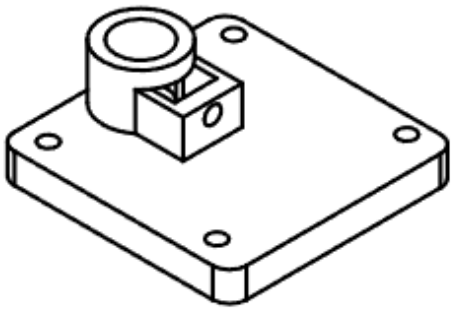


Author: David Rodríguez Appr.:	Legal owner: David Rodriguez	Scale: 1:1 Toler.: +/- ?	Sheet: 99 of 99 Size: A4
TITLE: Soporte Inferior Escritura		Part material: PM Part number: PN	
		Drawing no.: DN	
Document type: Mechanical assembly drawing		Date: 10/06/2024	Revision: REV A

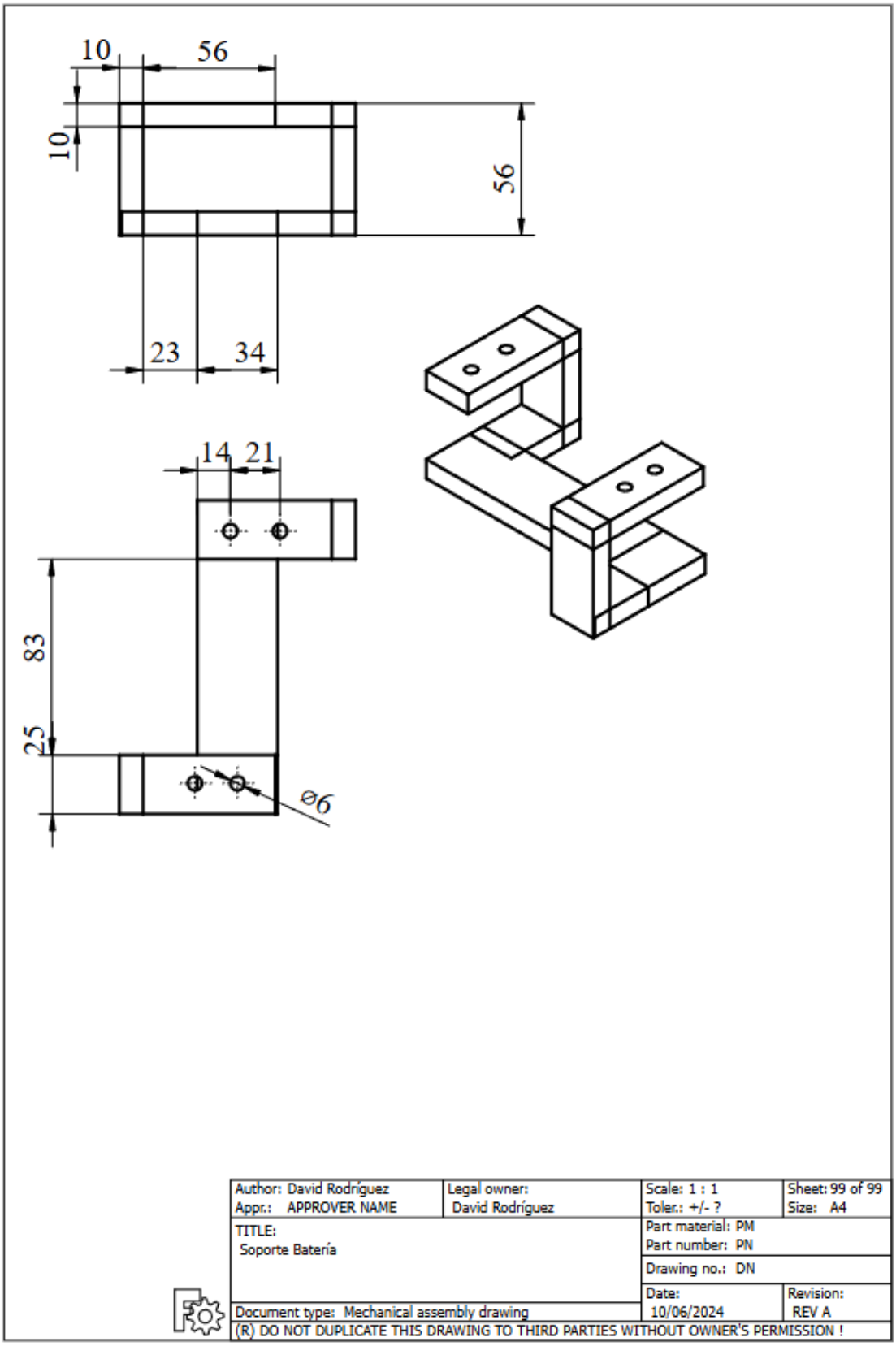








Author: David Rodríguez Appr.:	Legal owner: David Rodríguez	Scale: 1:1 Toler.: +/- ?	Sheet: 99 of 99 Size: A4
TITLE: Soporte Superior Escritura		Part material: PM Part number: PN	
		Drawing no.: DN	
Document type: Mechanical assembly drawing		Date: 10/06/2024	Revision: REV A
(R) DO NOT DUPLICATE THIS DRAWING TO THIRD PARTIES WITHOUT OWNER'S PERMISSION !			



Author: David Rodríguez Appr.: APPROVER NAME	Legal owner: David Rodríguez	Scale: 1 : 1 Toler.: +/- ?	Sheet: 99 of 99 Size: A4
TITLE: Soporte Batería		Part material: PM Part number: PN	
		Drawing no.: DN	
Document type: Mechanical assembly drawing		Date: 10/06/2024	Revision: REV A
(R) DO NOT DUPLICATE THIS DRAWING TO THIRD PARTIES WITHOUT OWNER'S PERMISSION !			

Documento 3: Anexos

1. Aplicación Bluetooth

1.1 MainActivity.java

```
package com.example.robotinkbluetoohv2;


// Bibliotecas blueooth

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;

import android.content.Intent;
import android.os.Build;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import com.ingenieriajhr.blujhr.BluJhr;

import java.util.ArrayList;
import java.util.List;

//Bibliotecas .text

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;
//Bibliotecas  para leer las lineas del fichero
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;



public class MainActivity extends AppCompatActivity {

    //Tema .txt
```

```
    private static final int READ_REQUEST_CODE = 42;
    private TextView textViewContent;

    private TextView textViewFirstLine;

    //Bluetooth
    BluJhr blue;
    List<String> requiredPermissions;
    ArrayList<String> devicesBluetooth = new ArrayList<String>();
    ConstraintLayout viewConn;
    ListView listDeviceBluetooth;
    Button buttonSend;
    Button BtnConectar;
    Button Btnatras;
    Button btnborrar;
    TextView consola;
    EditText edtTx;
    ImageView conectado;
    ImageView noconectado;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Varioabels paa  la lógica de  la app
        final int[] conexion = {0}; // Inicialmente no conectado; 0
para no conexion 1 para conexion
        conexion[0]=0;


        //Variables bluetooth
        blue = new BluJhr(this);
        blue.onBluetooth();
        listDeviceBluetooth = findViewById(R.id.listDeviceBluetooth);
        viewConn = findViewById(R.id.viewConn);
        buttonSend = findViewById(R.id.buttonSend);
        BtnConectar = findViewById(R.id.btconectar);
        Btnatras = findViewById(R.id.Btnatras);
        consola = findViewById(R.id.consola);
        edtTx = findViewById(R.id.edtTx);
        btnborrar = findViewById(R.id.btnborrar);
        noconectado =  findViewById(R.id.Noconectado);
        conectado =  findViewById(R.id.Conectado);

        //Boton examniar
        Button buttonBrowse = findViewById(R.id.buttonBrowse);
        textViewContent = findViewById(R.id.textViewContent);
        // Listener para el botón de examinar archivos
        buttonBrowse.setOnClickListener(v -> openFile());
        // TextView para mostrar la primera línea del archivo
        textViewFirstLine = findViewById(R.id.textViewFirstLine);

        // Lista bluetooth
```

```
        listDeviceBluetooth.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> adapterView, View
view, int i, long l) {
                if (!devicesBluetooth.isEmpty()){
                    blue.connect(devicesBluetooth.get(i));
                    blue.setDataLoadFinishedListener(new
BluJhr.ConnectedBluetooth() {
                        @Override
                        public void onConnectState(@NonNull
BluJhr.Connected connected) {
                            if (connected == BluJhr.Connected.True){

Toast.makeText(getApplicationContext(),"Conectado",Toast.LENGTH_SHORT)
.show();

listDeviceBluetooth.setVisibility(View.GONE);
                                viewConn.setVisibility(View.VISIBLE);
                                conectado.setVisibility(View.VISIBLE);
                                noconectado.setVisibility(View.GONE);
                                rxReceived();
                                BtnConectar.setText("Desconectar");
                                conexion[0] = 1;


                            }else{
                                if (connected ==
BluJhr.Connected.Pending){

Toast.makeText(getApplicationContext(),"Conectando",Toast.LENGTH_SHORT
).show();
                                }else{
                                    if (connected ==
BluJhr.Connected.False){

Toast.makeText(getApplicationContext(),"No se pudo
conectar",Toast.LENGTH_SHORT).show();
                                    }else{
                                        if (connected ==
BluJhr.Connected.Disconnect){

Toast.makeText(getApplicationContext(),"Disconnect",Toast.LENGTH_SHORT
).show();

viewConn.setVisibility(View.VISIBLE);

conectado.setVisibility(View.GONE);

noconectado.setVisibility(View.VISIBLE);

BtnConectar.setText("Conectar");
                                            conexion[0] = 0; //
Modificación del valor de conexion
```

```
                                        }
                                    }
                                }
                            }
                        }
                    });
                }
            }
        });
        //Cuando pulsas  el botn conectar muestra la lista de
dispositvos
        BtnConectar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if(conexion[0] ==  0) {
                    listDeviceBluetooth.setVisibility(View.VISIBLE);
                    Btnatras.setVisibility(View.VISIBLE);
                    viewConn.setVisibility(View.GONE);
                }else{
                    blue.closeConnection();
                }
            }
        });

        //Volver a antes boton atrás
        Btnatras.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                listDeviceBluetooth.setVisibility(View.GONE);
                viewConn.setVisibility(View.VISIBLE);
                Btnatras.setVisibility(View.GONE);
            }
        });

        //Cuando pulsas  el botn conectar muestra la lista de
dispositvos
        BtnConectar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if(conexion[0] ==  0) {
                    listDeviceBluetooth.setVisibility(View.VISIBLE);
                    Btnatras.setVisibility(View.VISIBLE);
                    viewConn.setVisibility(View.GONE);
                }else{
                    blue.closeConnection();
                }
            }
        });

        buttonSend.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {

                if(conexion[0] ==  0) {
```

```
                    Toast.makeText(getApplicationContext(),"No
connect",Toast.LENGTH_SHORT).show();
                }else{
                    //blue.bluTx(edtTx.getText().toString() + "\n");
                    blue.bluTx("Imprimir \n");

                }
            }
        });

        //Boton que borra las consolas de muestra de comandos
        btnborrar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                consola.setText("");
                textViewContent.setText("");
                textViewFirstLine.setText("");

            }
        });

    }


    private void openFile() {
        Intent intent = new Intent(Intent.ACTION_OPEN_DOCUMENT);
        intent.addCategory(Intent.CATEGORY_OPENABLE);
        intent.setType("text/plain");
        startActivityForResult(intent, READ_REQUEST_CODE);
    }

    private void rxReceived() {

        blue.loadDateRx(new BluJhr.ReceivedData() {
            @Override
            public void rxDate(@NonNull String receivedData) {
                // Verificar si el dato recibido es "XON"
                if (receivedData.equals("XON")) {
                    // Si es "XON", leer la siguiente línea del
archivo
                    // y mostrarla en el TextView correspondiente
                    leerSiguienteLineaArchivo();
                } else {
                    // Borrar la línea anterior del consola
textViewFirstLine
                    consola.setText("");
                    // Si no es "XON", simplemente mostrar el dato en
el TextView consola
                    consola.setText(consola.getText().toString() +
receivedData + "\n");

                }
            }
        });
```

```
    }

    // Variable para mantener el índice de la línea actual
    private int currentIndex = 0;

    // Método para leer, mostrar y enviar la siguiente línea del
archivo
    private void leerSiguienteLineaArchivo() {
        // Obtener el contenido actual del TextView textViewContent
        String contenido = textViewContent.getText().toString();
        // Dividir el contenido en líneas
        String[] lineas = contenido.split("\n");
        // Verificar si el índice actual es válido
        if (currentIndex < lineas.length - 1) {
            // Incrementar el índice para apuntar a la siguiente línea
            currentIndex++;
            // Obtener la siguiente línea del archivo
            String siguienteLinea = lineas[currentIndex];
            // Mostrar la siguiente línea en el TextView
correspondiente
            textViewFirstLine.setText(siguienteLinea);
            //ENVIO DE LINBEA POR BLUETOOTH
            blue.bluTx(siguienteLinea + "\n");

        } else {
            // Si no hay más líneas disponibles, mostrar un mensaje o
realizar alguna acción
            // Esto puede variar según tus necesidades específicas

            int contafin  = 0;
            contafin++;
            Toast.makeText(this, "No hay más líneas disponibles",
Toast.LENGTH_SHORT).show();
            if(contafin == 3){
                consola.setText("");
                textViewContent.setText("");
                textViewFirstLine.setText("");
                blue.closeConnection();
                contafin=0;

            }
        }
    }



    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull
String[] permissions, @NonNull int[] grantResults) {
        if (blue.checkPermissions(requestCode,grantResults)){
            Toast.makeText(this, "Exit", Toast.LENGTH_SHORT).show();
            blue.initializeBluetooth();
        }else{
            if(Build.VERSION.SDK_INT < Build.VERSION_CODES.S){
                blue.initializeBluetooth();
```

```
            }else{
                Toast.makeText(this, "Algo salio mal",
Toast.LENGTH_SHORT).show();
            }
        }

        super.onRequestPermissionsResult(requestCode, permissions,
grantResults);
    }


    @Override
    protected void onActivityResult(int requestCode, int resultCode,
@Nullable Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        // Tu lógica actual para manejar el resultado de la actividad
de Bluetooth
        if (!blue.stateBluetoooth() && requestCode == 100){
            blue.initializeBluetooth();
        } else {
            if (requestCode == 100){
                devicesBluetooth = blue.deviceBluetooth();
                if (!devicesBluetooth.isEmpty()){
                    ArrayAdapter adapter = new
ArrayAdapter(this,android.R.layout.simple_expandable_list_item_1,devic
esBluetooth);
                    listDeviceBluetooth.setAdapter(adapter);
                } else {
                    Toast.makeText(this, "No tienes vinculados
dispositivos", Toast.LENGTH_SHORT).show();
                }
            }
        }

        // Nueva lógica para abrir y mostrar el contenido de un
archivo de texto
        // Verificar si la solicitud de selección de archivos fue
exitosa
        if (requestCode == READ_REQUEST_CODE && resultCode ==
RESULT_OK) {
            if (data != null) {
                Uri uri = data.getData();
                if (uri != null) {
                    try {
                        // Obtener un flujo de entrada para leer el
archivo seleccionado
                        InputStream inputStream =
getContentResolver().openInputStream(uri);
                        if (inputStream != null) {
                            // Crear un lector de flujo para leer el
archivo línea por línea
                            BufferedReader reader = new
BufferedReader(new InputStreamReader(inputStream));
```

```
                            // StringBuilder para almacenar el
contenido completo del archivo
                            StringBuilder stringBuilder = new
StringBuilder();
                            // Leer la primera línea del archivo y
mostrarla en textViewFirstLine
                            String line;
                            if ((line = reader.readLine()) != null) {
                                textViewFirstLine.setText(line);

stringBuilder.append(line).append("\n");
                            }
                            // Leer el resto del archivo y agregarlo
al StringBuilder
                            while ((line = reader.readLine()) != null)
{

stringBuilder.append(line).append("\n");
                            }
                            // Cerrar el flujo de entrada y el lector
                            inputStream.close();
                            reader.close();
                            // Mostrar el contenido completo del
archivo en textViewContent

textViewContent.setText(stringBuilder.toString());
                        }
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
    }

}
```

1.2 MainActivity.xml

```
<?xml version="1.0" encoding="utf-8"?>


<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:visibility="visible"
    tools:visibility="visible">

    <Button
        android:id="@+id/Btnatras"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="64dp"
        android:text="Principal"
        android:visibility="gone"
        app:layout_constraintBottom_toTopOf="@+id/listDeviceBluetooth"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <ListView
        android:id="@+id/listDeviceBluetooth"
        android:layout_width="386dp"
        android:layout_height="364dp"
        android:visibility="gone"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <androidx.constraintlayout.widget.ConstraintLayout
        android:id="@+id/viewConn"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:visibility="visible"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <TextView
            android:id="@+id/consola"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="40dp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@+id/btconectar"></TextView>

        <Button
```

```
            android:id="@+id/buttonSend"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_marginTop="32dp"
            android:text="Imprimir"
            app:layout_constraintTop_toBottomOf="@+id/consola"
            tools:layout_editor_absoluteX="158dp"></Button>

        <EditText
            android:id="@+id/edtTx"
            android:layout_width="243dp"
            android:layout_height="54dp"
            android:layout_marginTop="40dp"
            android:layout_marginEnd="32dp"
            android:visibility="gone"
            app:layout_constraintEnd_toStartOf="@+id/buttonSend"

app:layout_constraintTop_toBottomOf="@+id/consola"></EditText>

        <Button
            android:id="@+id/btconectar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="44dp"
            android:layout_marginEnd="164dp"
            android:text="Conectar"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <Button
            android:id="@+id/buttonBrowse"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerInParent="true"
            android:layout_marginEnd="16dp"
            android:text="Examinar"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toTopOf="@+id/scrollView" />

        <ScrollView
            android:id="@+id/scrollView"
            android:layout_width="229dp"
            android:layout_height="260dp"
            android:layout_below="@id/buttonBrowse"
            android:layout_marginTop="204dp"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.087"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/edtTx">

            <TextView
                android:id="@+id/textViewContent"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
```

```
                android:text="Contenido del archivo aquí" />

        </ScrollView>

        <TextView
            android:id="@+id/textViewFirstLine"
            android:layout_width="346dp"
            android:layout_height="47dp"
            android:layout_marginBottom="40dp"
            android:text="Instrucción aquí"
            android:visibility="visible"
            app:layout_constraintBottom_toTopOf="@+id/scrollView"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.492"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toBottomOf="@+id/buttonSend"
            app:layout_constraintVertical_bias="0.855" />

        <Button
            android:id="@+id/btnborrar"
            android:layout_width="105dp"
            android:layout_height="47dp"
            android:text="Borrar"
            app:layout_constraintStart_toStartOf="@+id/buttonBrowse"
            app:layout_constraintTop_toBottomOf="@+id/buttonBrowse" />

        <ImageView
            android:id="@+id/Noconectado"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="24dp"
            android:visibility="visible"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.327"
            app:layout_constraintStart_toEndOf="@+id/btconectar"
            app:layout_constraintTop_toTopOf="@+id/btconectar"

app:srcCompat="@android:drawable/button_onoff_indicator_off" />

        <ImageView
            android:id="@+id/Conectado"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="24dp"
            android:visibility="gone"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintHorizontal_bias="0.327"
            app:layout_constraintStart_toEndOf="@+id/btconectar"
            app:layout_constraintTop_toTopOf="@+id/btconectar"

app:srcCompat="@android:drawable/button_onoff_indicator_on" />

    </androidx.constraintlayout.widget.ConstraintLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
```

2. Código Arduino

```
//Biblioteca bluetooth::::::::
#include<SoftwareSerial.h>
//::::::::::::::::::::::::::::::::
// --------------------
// VARIABLES GLOBALES
// --------------------
// Constantes
#define PI 3.14159265358979323846264338327795
#define DOS_PI 6.28318530717958647692528676655 9
// Macros de bit set/clear/check/toggle
#define SET(x,y) (x |=(1<<y))
#define CLR(x,y) (x &= (~(1<<y)))
#define CHK(x,y) (x & (1<<y))
#define TOG(x,y) (x^=(1<<y))
// ConfiguraciÛn del plotter
#define BAUDRATE 9600
#define ANCHO_MAX_PAPEL 420
#define ALTO_MAX_PAPEL 420
#define BOLI 3
#define XOFF 0x13
#define XON 0x11
//INICIALIZACION BLUETOOTH::::::
SoftwareSerial BT(12,13);
//::::::::::::::::::::::::::::::
bool DEBUG = false;
float
FACTOR_ESCALA = 1.0,
ARCO_MAX = 1.0;
int
VALOR_X = 0,
VALOR_Y = 0,
ANTERIOR_X = 0,
ANTERIOR_Y = 0;
// ConfiguraciÛn del Gcode
#define STRING_SIZE 128
char
 BUFFER[STRING_SIZE + 1],
 INPUT_CHAR;
String
 INPUT_STRING,
 SUB_STRING;
int
 POS = 0,
 INICIO,
 FIN;
float
 X,
 Y,
 I,
 J;
// Entradas de los controladores de los motores
#define IN1_X A0
```

```
#define IN2_X A1
#define IN3_X A2
#define IN4_X A3

#define IN1_Y 8
#define IN2_Y 9
#define IN3_Y 10
#define IN4_Y 11
// DefiniciÛn secuencia de medios pasos del motor
byte Motor[8] =
{ B00000001,
 B00000011,
 B00000010,
 B00000110,
 B00000100,
 B00001100,
 B00001000,
 B00001001
};
int p = 0;
byte Sec = 0;
#define PASOS_POR_MM 4096/(PI*48)
#define AVANCE PASOS_POR_MM*5
unsigned long DELAY = 1;
int contar_X = 0;
int contar_X_max = PASOS_POR_MM * ANCHO_MAX_PAPEL;
byte X_Sec;
int contar_Y = 0;
int contar_Y_max = PASOS_POR_MM * ALTO_MAX_PAPEL;
byte Y_Sec;
// ----------------------------
// CONFIGURACI”N DE ACTUADORES
// ----------------------------
void setup()
{
 // ConfiguraciÛn inicial del motor X
 pinMode(IN1_X, OUTPUT);
 pinMode(IN2_X, OUTPUT);
 pinMode(IN3_X, OUTPUT);
 pinMode(IN4_X, OUTPUT);
 digitalWrite(IN1_X, LOW);
 digitalWrite(IN2_X, LOW);
 digitalWrite(IN3_X, LOW);
 digitalWrite(IN4_X, LOW);
 // ConfiguraciÛn inicial del motor Y
 pinMode(IN1_Y, OUTPUT);
 pinMode(IN2_Y, OUTPUT);
 pinMode(IN3_Y, OUTPUT);
 pinMode(IN4_Y, OUTPUT);
 digitalWrite(IN1_Y, LOW);
 digitalWrite(IN2_Y, LOW);
 digitalWrite(IN3_Y, LOW);
 digitalWrite(IN4_Y, LOW);
 // ConfiguraciÛn del sistema de escritura
 pinMode(BOLI, OUTPUT);
```

```
 TCCR2A = _BV(COM2B1) | _BV(COM2B0) | _BV(WGM20);
 TCCR2B = _BV(WGM22) | _BV(CS22) | _BV(CS21) | _BV(CS20);

  OCR2A = 156;
 OCR2B = 148;

 // ConfiguraciÛn del plotter
 memset(BUFFER, '\0', sizeof(BUFFER));
 INPUT_STRING.reserve(STRING_SIZE);
 INPUT_STRING = "";
 // Establecer conexiÛn serie
 Serial.begin(BAUDRATE);
//  eSTABLECER cONEXION Bluetooth::::::::::::
 BT.begin(BAUDRATE);
 // Vaciar los buffers
 Serial.flush();
 //::::::::::::::::::::::::::::::::::::::::
 BT.flush();

 while (Serial.available())
Serial.read();//:::::::::::::::::::::::::::Prestar atrencion
 // Comandos mostrados por pantalla
 opc();
 Serial.write(XON);//::::::::::
 //BT.write(XON);
}
//----------------------------------------
// MAIN LOOP
//----------------------------------------
void loop(){
  // Leer instrucción desde el módulo Bluetooth
  if(BT.available()){

    // Leer un carácter del módulo Bluetooth
    INPUT_CHAR = (char)BT.read();
    // Escribir el carácter en el puerto serial
    Serial.write(INPUT_CHAR);
    // Almacenar el carácter en el buffer
    BUFFER[POS++] = INPUT_CHAR;
    // Si se recibe un salto de línea
    if (INPUT_CHAR == '\n'){
      // Vaciar el búfer del puerto serial
      Serial.flush();
      // Enviar el carácter "XOFF" al dispositivo conectado al puerto
serial
      Serial.write("XOFF\n");
      // Vaciar el búfer del módulo Bluetooth
      BT.flush();
      // Enviar el carácter "XOFF" al dispositivo Bluetooth
      BT.write("XOFF\n");
      // Convertir el contenido del buffer en un String
      INPUT_STRING = BUFFER;
      // Procesar el String
      process(INPUT_STRING);
      // Limpiar el buffer
```

```
      memset(BUFFER, '\0', sizeof(BUFFER));
      // Restablecer la posición del buffer a cero
      POS = 0;
      // Restablecer el String de entrada a vacío
      INPUT_STRING = "";
      //Esperar un segundo
      delay(500);//Tambien funciona con 1000
      // Vaciar el búfer del puerto serial
      Serial.flush();
      // Enviar el carácter "XON" al dispositivo conectado al puerto
serial
      Serial.write("XON\n");
      // Vaciar el búfer del módulo Bluetooth
      BT.flush();
      BT.write("XON\n");
      // Enviar el carácter "XON" al dispositivo Bluetooth

    }
  }
  if(Serial.available()){
    BT.write(Serial.read());
  }

}
//----------------------
// OPCIONES ADICIONALES
//----------------------
void opc() {
 Serial.println(F(" "));
 Serial.println(F(" OPCIONES ADICIONALES"));
 Serial.println(F(" "));
 Serial.println(F(" G00 X## Y## ........ mover a (X,Y)
(Boliarriba)"));
 Serial.println(F(" G01 X## Y## ........ mover a (X,Y) (Boliabajo)"));
 Serial.println(F(" C1 S##.## .......... fijar escala deescritura
(1=100%)"));
 Serial.println(F(" C2 .................. adelante boli"));
 Serial.println(F(" C3 .................. atras boli"));
 Serial.println(F(" "));
}
//--------------------------
// INTERPRETACI”N DEL GCODE
//--------------------------
void process(String string)
{
 if (DEBUG)
 {
 Serial.println(string);
 }

 // Convertir string a mayusculas
 INPUT_STRING = string;
 INPUT_STRING.toUpperCase();
 // -------------------------------------
 // G00 - Movimiento lineal sin imprimir
```

```
// ------------------------------------
if (INPUT_STRING.startsWith("G00"))
{
// extraer coordenada X
INICIO = INPUT_STRING.indexOf('X');
if (!(INICIO < 0))
{
FIN = INICIO + 8;
SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 1);
X = SUB_STRING.toFloat();
}
// extraer coordenada Y
INICIO = INPUT_STRING.indexOf('Y');
if (!(INICIO < 0))
{
FIN = INICIO + 8;
SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 1);
Y = SUB_STRING.toFloat();
}
subir_boli();
ir_a(X, Y);
}
// ------------------------------------
// G01 - Movimiento lineal imprimiendo
// ------------------------------------
if (INPUT_STRING.startsWith("G01"))
{
// extraer coordenada X
INICIO = INPUT_STRING.indexOf('X');
if (!(INICIO < 0))
{
FIN = INICIO + 8;
SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 1);
X = SUB_STRING.toFloat();
}
// extraer coordenada Y
INICIO = INPUT_STRING.indexOf('Y');

if (!(INICIO < 0))
{
FIN = INICIO + 8;
SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 1);
Y = SUB_STRING.toFloat();
}
bajar_boli();
ir_a(X, Y);
}
// ---------------------------------------
// G02 - Imprimir arco en sentido horario
// ---------------------------------------
if (INPUT_STRING.startsWith("G02"))
{
// extraer coordenada X
INICIO = INPUT_STRING.indexOf('X');
if (!(INICIO < 0))
```

```
{
FIN = INPUT_STRING.indexOf('.', INPUT_STRING.indexOf('X'));
SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 7);
X = SUB_STRING.toFloat();
}
// extraer coordenada Y
INICIO = INPUT_STRING.indexOf('Y');
if (!(INICIO < 0))
{
FIN = INPUT_STRING.indexOf('.', INPUT_STRING.indexOf('Y'));
SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 7);
Y = SUB_STRING.toFloat();
}
// extraer coordenada I
INICIO = INPUT_STRING.indexOf('I');
if (!(INICIO < 0))
{
FIN = INPUT_STRING.indexOf('.', INPUT_STRING.indexOf('I'));
SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 7);
I = SUB_STRING.toFloat();
}
// extraer coordenada J
INICIO = INPUT_STRING.indexOf('J');
if (!(INICIO < 0))
{
FIN = INPUT_STRING.indexOf('.', INPUT_STRING.indexOf('J'));
SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 7);
J = SUB_STRING.toFloat();
}
bajar_boli();
arco_horario(X, Y, I, J);
}
// ------------------------------------------------
// G03 - Imprimir arco en sentido antihorario
// ------------------------------------------------
if (INPUT_STRING.startsWith("G03"))
{
// extraer coordenada X
INICIO = INPUT_STRING.indexOf('X');
if (!(INICIO < 0))
{
FIN = INPUT_STRING.indexOf('.', INPUT_STRING.indexOf('X'));
SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 7);
X = SUB_STRING.toFloat();
}
// extraer coordenada Y
INICIO = INPUT_STRING.indexOf('Y');
if (!(INICIO < 0))
{
FIN = INPUT_STRING.indexOf('.', INPUT_STRING.indexOf('Y'));
SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 7);
Y = SUB_STRING.toFloat();
}
// extraer coordenada I
INICIO = INPUT_STRING.indexOf('I');
```

```
 if (!(INICIO < 0))
 {
 FIN = INPUT_STRING.indexOf('.', INPUT_STRING.indexOf('I'));
 SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 7);
 I = SUB_STRING.toFloat();
 }
 // extraer coordenada J
 INICIO = INPUT_STRING.indexOf('J');
 if (!(INICIO < 0))
 {
 FIN = INPUT_STRING.indexOf('.', INPUT_STRING.indexOf('J'));
 SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN + 7);
 J = SUB_STRING.toFloat();
 }
 bajar_boli();
 arco_antihorario(X, Y, I, J);
 }
 // ---------
 // OPCIONES
 // ---------
 if (INPUT_STRING.startsWith("opc"))
 {
 opc();
 }
 // ----------------------------
 // C1 - Fijar factor de escala
 // ----------------------------
 if (INPUT_STRING.startsWith("C1"))
 {
 INICIO = INPUT_STRING.indexOf('S');
 if (!(INICIO < 0))
 {
 FIN = INICIO + 6;
 SUB_STRING = INPUT_STRING.substring(INICIO + 1, FIN);
 FACTOR_ESCALA = SUB_STRING.toFloat();
 Serial.print(F("Dibujando ")); Serial.print(FACTOR_ESCALA *
100); Serial.println(F("%"));

}
 else
 {
 Serial.println(F("Factor de escala no v·lido!!"));
 }
 }
 // ----------------------------
 // C2 - Escritura desactivada
 // ----------------------------
 if (INPUT_STRING.startsWith("C2"))
 {
 subir_boli();
 }
 // ------------------------
 // C3 - Escritura activada
 // ------------------------
 if (INPUT_STRING.startsWith("C3"))
```

```
 {
 bajar_boli();
 }
}
// ----------------------
// FUNCI”N DE MOVIMIENTO
// ----------------------
void ir_a(float x, float y)
{
 // Aplicar factor de escala
 VALOR_X = round(x * PASOS_POR_MM * FACTOR_ESCALA);
 VALOR_Y = round(y * PASOS_POR_MM * FACTOR_ESCALA);
 // Dibujar linea entre coordenadas escaladas
 traza_linea(ANTERIOR_X, ANTERIOR_Y, VALOR_X, VALOR_Y);
 // ----- ⁄ltimas coordinadas escaldas
 ANTERIOR_X = VALOR_X;
 ANTERIOR_Y = VALOR_Y;
}
// -------------------
// DIBUJAR UNA LINEA
// -------------------
void traza_linea(int x1, int y1, int x2, int y2)
{
 // Variables locales
 int
 x = x1,
 y = y1,
 dy,
 dx,
 pendiente,
 largo,
 corto,
 maximo,
 error,
 margen;
 // Definir eje largo y eje corto
 dy = y2 - y1;

 dx = x2 - x1;
 largo = max(abs(dy), abs(dx));
 corto = min(abs(dy), abs(dx));
 // Escalar los valores de Bresenham por 2*largo
 error = -largo;
 margen = 0;
 maximo = (largo << 1);
 pendiente = (corto << 1);
 // Inicializar el cambio de flanco
 bool cambioXY = true;
 if (abs(dx) >= abs(dy)) cambioXY = false;
 // Simular que siempre estamos en el octante 0
 for (int i = 0; i < largo; i++)
 {
 // Movimiento a izquierda/derecha sobre eje X
 if (cambioXY)
 {
```

```
 if (dy < 0)
 {
 y--;
 atras();
 }
 else
 {
 y++;
 adelante();
 }
 }
 else
 {
 if (dx < 0)
 {
 x--;
 izquierda();
 }
 else
 {
 x++;
 derecha();
 }
 }
 // Movimiento arriba/abajo en eje Y
 error += pendiente;
 if (error > margen)
 {
 error -= maximo;
 // Movimiento arriba/abajo sobre eje Y
 if (cambioXY)
 {
 if (dx < 0)
 {
 x--;
 izquierda();
 }
 else
 {

x++;
 derecha();
 }
 }
 else
 {
 if (dy < 0)
 {
 y--;
 atras();
 }
 else
 {
 y++;
 adelante();
```

```
 }
 }
 }
 }
}
//--------------------------------
// Mover el boli un paso hacia x-
//--------- ----------------------
void izquierda()
{
 // Rotar motor en sentido antihorario
 contar_X--;
 if (contar_X >= 0)
 {
 p = contar_X % 8;
 Sec = PORTC;
 Sec = Sec & B11110000;
 Sec = Sec | Motor[p];
 PORTC = Sec;
 delay(DELAY);
 }
}
//--------------------------------
// Mover el boli un paso hacia x+
//--------- ----------------------
void derecha()
{
 // Rotar motor en sentido horario
 contar_X++;
 if (contar_X < contar_X_max)
 {
 p = contar_X % 8;
 Sec = PORTC;
 Sec = Sec & B11110000;
 Sec = Sec | Motor[p];
 PORTC = Sec;
 delay(DELAY);
 }
}
//--------------------------------
// Mover el boli un paso hacia y+
//--------- ----------------------
void adelante()

{
 // Rotar motor en sentido horario
 contar_Y++;
 if (contar_Y < contar_Y_max)
 {
 p = contar_Y % 8;
 Sec = PORTB;
 Sec = Sec & B11110000;
 Sec = Sec | Motor[p];
 PORTB = Sec;
 delay(DELAY);
```

```
 }
}
//--------------------------------
// Mover el boli un paso hacia y-
//--------------------------------
void atras()
{
 // Rotar motor sentido antihorario
 contar_Y--;
 if (contar_Y >= 0)
 {
 p = contar_Y % 8;
 Sec = PORTB;
 Sec = Sec & B11110000;
 Sec = Sec | Motor[p];
 PORTB = Sec;
 delay(DELAY);
 }
}
//---------------------------------------
// Dibujar arco en sentido horario (G02)
//---------------------------------------
void arco_horario(float x, float y, float i, float j)
{
 if ((i < -420) || (i > 420) || (j < -420) || (j > 420))
 {
 ir_a(x, y);
 }
 else
 {
 // Variables locales
 float
 valorX = ANTERIOR_X / (PASOS_POR_MM * FACTOR_ESCALA),
 valorY = ANTERIOR_Y / (PASOS_POR_MM * FACTOR_ESCALA),
 valorsigX = x,
 valorsigY = y,
 nuevaX,
 nuevaY,
 I = i,
 J = j,
 circX = valorX + I,
 circY = valorY + J,
 delta_x,
 delta_y,
 distancia,
 radio,
 alpha,
 beta,

 arco,
 angulo,
 sig_angulo;
 // Calcular arcoo
 delta_x = valorX - valorsigX;
 delta_y = valorY - valorsigY;
```

```
 distancia = sqrt(delta_x * delta_x + delta_y * delta_y);
 radio = sqrt(I * I + J * J);
 alpha = 2 * asin(distancia / (2 * radio));
 arco = alpha * radio;
 // Dividir ·ngulo en partes
 int segmentos = 1;
 if (arco > ARCO_MAX)
 {
 segmentos = (int)(arco / ARCO_MAX);
 beta = alpha / segmentos;
 }
 else
 {
 beta = alpha;
 }
 // Calcular ·ngulo actual
 angulo = atan2(-J, -I);
 if (angulo <= 0) angulo += 2 * PI;
 // Coordenadas intermedias sentido horario
 sig_angulo = angulo;
 for (int segmento = 1; segmento < segmentos; segmento++)
 {
 sig_angulo -= beta;
 if (sig_angulo < 0) sig_angulo += 2 * PI;
 nuevaX = circX + radio * cos(sig_angulo);
 nuevaY = circY + radio * sin(sig_angulo);
 ir_a(nuevaX, nuevaY);
 }
 // Dibujar final de linea
 ir_a(valorsigX, valorsigY);
 }
}
//---------------------------------------
// Dibujar arco en sentido antihorario
//---------------------------------------
void arco_antihorario(float x, float y, float i, float j)
{
 if ((i < -420) || (i > 420) || (j < -420) || (j > 420))
 {
 ir_a(x, y);
 }
 else
 {
 // Variables locales
 float
 valorX = ANTERIOR_X / FACTOR_ESCALA,
 valorY = ANTERIOR_Y / FACTOR_ESCALA,
 valorsigX = x,
 valorsigY = y,

 nuevaX,
 nuevaY,
 I = i,
 J = j,
 circX = valorX + I,
```

```
 circY = valorY + J,
 delta_x,
 delta_y,
 distancia,
 radio,
 alpha,
 beta,
 arco,
 angulo,
 sig_angulo;
 // Calcular arco
 delta_x = valorX - valorsigX;
 delta_y = valorY - valorsigY;
 distancia = sqrt(delta_x * delta_x + delta_y * delta_y);
 radio = sqrt(I * I + J * J);
 alpha = 2 * asin(distancia / (2 * radio));
 arco = alpha * radio;
 // Dividir ·ngulo en partes
 int segmentos = 1;
 if (arco > ARCO_MAX)
 {
 segmentos = (int)(arco / ARCO_MAX);
 beta = alpha / segmentos;
 }
 else
 {
 beta = alpha;
 }
 // Calcular ·ngulo actual
 angulo = atan2(-J, -I);
 if (angulo <= 0) angulo += 2 * PI;
 // Coordenadas intermedias sentido antihorario
 sig_angulo = angulo;
 for (int segmento = 1; segmento < segmentos; segmento++)
 {
 sig_angulo += beta;
 if (sig_angulo > 2 * PI) sig_angulo -= 2 * PI;
 nuevaX = circX + radio * cos(sig_angulo);
 nuevaY = circY + radio * sin(sig_angulo);
 ir_a(nuevaX, nuevaY);
 }
 // Dibujar final de linea
 ir_a(valorsigX, valorsigY);
 }
}
//--------------------------------
// Levantar el ˙til de escritura
//--------------------------------
void subir_boli()
{

OCR2B = 148;
 delay(250);
}
//----------------------------
```

```
// Bajar el ˙til de escritura
//----------------------------
void bajar_boli()
{
 OCR2B = 140;
 delay(250);
}
```

Documento 4: Pliego de Condiciones

Índice general

1. Pliego de Condiciones Generales

1.1 Documentos del proyecto

La documentación del proyecto se compone de varios documentos esenciales, que deben presentarse y mantenerse actualizados a lo largo del ciclo de vida del proyecto. Los documentos requeridos son los siguientes:

- Documento nº 1: Memoria.

La Memoria proporciona una visión general del proyecto, incluyendo los objetivos, antecedentes, justificación, y alcance. Describe las metodologías y técnicas a utilizar, los recursos necesarios y los resultados esperados.

- Documento nº 2: Anexos.

Los Anexos incluyen información complementaria que respalda la Memoria y otros documentos del proyecto. Pueden contener estudios técnicos, datos específicos, cálculos, tablas, y gráficos relevantes.

- Documento nº 3: Planos

Los Planos son representaciones gráficas del proyecto. Incluyen planos de situación, emplazamiento, distribución, y detalle, elaborados conforme a las normativas técnicas aplicables.

- Documento n° 4: Pliego de Condiciones.

El Pliego de Condiciones establece los requisitos, términos y condiciones para la ejecución del proyecto. Incluye especificaciones técnicas, cláusulas administrativas, criterios de evaluación y selección, y responsabilidades de las partes.

- Documento nº 5: Presupuesto.

El Presupuesto detalla los costos asociados a la ejecución del proyecto, desglosando los gastos por partidas, tales como materiales, mano de obra, equipos, y otros gastos necesarios.

1.2 Normativa del aplicable

El presente proyecto debe cumplir con las siguientes normativas y estándares aplicables para asegurar la seguridad, calidad y desempeño del robot móvil omnidireccional:

1. **UNE-EN ISO 10218-1:2011**
    - **Título:** Robots y dispositivos robóticos. Requisitos de seguridad para robots industriales. Parte 1: Robots.
    - **Descripción:** Establece los requisitos de seguridad para el diseño, fabricación y uso de robots industriales, asegurando que operen de manera segura y eficiente.
2. **UNE-EN ISO 12100:2012**
    - **Título:** Seguridad de las máquinas. Principios generales de diseño. Evaluación y reducción de riesgos.
    - **Descripción:** Proporciona directrices para la evaluación y reducción de riesgos en el diseño de máquinas, incluyendo robots, asegurando que se tomen en cuenta todos los aspectos de seguridad durante el desarrollo del proyecto.
3. **UNE-EN 60204-1:2018**
    - **Título:** Equipamiento eléctrico de máquinas. Parte 1: Requisitos generales.
    - **Descripción:** Especifica los requisitos de seguridad y recomendaciones para el diseño y la construcción del equipamiento eléctrico de máquinas, incluyendo sistemas de control eléctricos para robots.

Consideraciones Específicas para el Proyecto

- **Control y Automatización:** La normativa IEC 61131-3:2013 (Programación de controladores lógicos programables) relevante para el software de control.
- **Seguridad en Robots Móviles:** La normativa ISO 13482:2014 (Robots y dispositivos robóticos. Requisitos de seguridad para robots personales de asistencia) puede ofrecer pautas adicionales aplicables a robots móviles en entornos no industriales.

1.3 Equipos para la ejecución de los trabajos

El personal encargado de la ejecución de este proyecto deberá atenerse a todas las indicaciones especificadas en relación con los materiales y equipos necesarios para llevar a cabo los trabajos correspondientes. Estos materiales y equipos deberán cumplir los requisitos de la normativa aplicable, garantizando como mínimo una calidad estándar. Si en algún momento de este proyecto se selecciona un determinado producto, tipo o modelo para su utilización, y alguno de los equipos o materiales no cumple con las especificaciones determinadas, la entidad ejecutora tendrá el deber de sustituirlos por otros que sí cumplan, asegurándose siempre de que tengan características idénticas o muy similares, y con la aprobación del ingeniero proyectista.

1.4 Otras consideraciones que cumplir por los materiales y equipos

Los materiales y equipos utilizados deben estar en conformidad con las normas y reglamentos correspondientes. Si se elige un producto, tipo o modelo específico para usar en cualquier parte de este proyecto, debe ser sustituido por un producto que cumpla con los mismos estándares de calidad. Las piezas del equipo o del conjunto pueden ser intercambiadas por otras, siempre y cuando sean compatibles de características iguales o muy parecidas y siempre con la aprobación del ingeniero de diseño.

1.5 Otras consideraciones que cumplir por los materiales y equipos

Los planos y especificaciones tienen por objeto mostrar al cliente la forma, tamaño, calidad y cantidad de las piezas y sistemas que deben fabricarse, así como su disposición relativa durante el montaje y la instalación. También detallan la mano de obra utilizada, el equipo y las instalaciones de montaje necesarios para llevar a cabo el proyecto, a menos que el ingeniero de diseño indique lo contrario.

El Promotor realizará todos los trabajos descritos en los planos y especificados en el presupuesto, o cualquier trabajo adicional que se considere necesario para completar la compilación de forma aceptable.

1.6 Programa de ejecución y plazos

Para la ejecución de este proyecto se ha estimado una duración de aproximadamente 9 días.

- Comienzo del proyecto: Obtención de los materiales y modelos 3D obtenidos de los planos presentes en el documento para su impresión (3 Días aprox).
- Continuación del proyecto: Impresión de las piezas con su posterior ajuste y tratado (3 Días aprox).
- Final Montaje del proyecto: Instalación de componentes en la estructura del Robot (1 Dia aprox).
- Finalización de instalaciones: Descarga de software para la comunicación y obtención de imagen en G-code. (1 Día aprox).
- Finalización del proyecto: pruebas y puesta en marcha definitiva (1 Día arpox).

El autor del proyecto puede ordenar cambios en cualquier unidad planificada y presentar los planos finales al contratista, quien se integrará al proyecto. Los ajustes se registrarán en el libro de órdenes y se devolverán al mercado al finalizar el trabajo. Si los ajustes implican la sustitución de una unidad por otra con características similares, no se modificará el precio unitario del proyecto.

2. Pliego de Condiciones Técnicas

2.1 Robot Móvil Impresora

El modelo utilizado será el desarrollado en el anterior TFG con las modificaciones correspondientes presentes en este Documento.

El robot tiene 4 grados de libertad diseñado para la impresión de documentos en 2D en una superficie plana. El diseño del robot es complementado con el diseño de una aplicación desarrollada para el entorno de Android con la que recibirá mediante tecnología Bluetooth los documentos a imprimir.

2.1.1 Masa Robot

La estructura del robot tendrá un peso máximo de 2kg una vez ensamblados todos los elementos mecánicos, y la electrónica del robot (incluidas las ruedas) tendrá un peso máximo de 1,5kg.

2.1.2 Planos y dimensiones del Robot

Los planos necesarios para la elaboración de este proyecto se encuentran en este documento. Estos planos deben cumplir con la siguiente normativa.

- UNE-EN ISO 5457:2000/A1:2010.
- UNE-EN ISO 7200:2004.
- UNE-EN ISO 5455:1996.
- UNE-EN ISO 5845-1:2000.

2.2 Normas y Seguridad

2.2.1 Normas

El robot cumple con la siguiente normativa:

- UNE-EN ISO 12100:2012.
- IEC 60204-1:2005.
- UNE-EN ISO 13849-1:2016.
- IEC 62061:2012.

2.2.2 Seguridad

El robot está concebido para ser completamente seguro. Solo operará cuando sea manejado por una persona que cargue un documento, y la velocidad máxima que alcanzará será de 2.5 mm/s.

2.3 Normas y Seguridad

2.3.1 Consideraciones generales

El robot está concebido con una configuración específica de los ejes. No obstante, se puede adoptar una configuración diferente para el plano XY modificando la conexión de los motores, aunque esto solo podrá realizarse con la aprobación del ingeniero de diseño.

2.3.2 Requisitos de Funcionamiento

El robot podrá ser operado y almacenado en entornos con baja humedad, por debajo del 60%. Asimismo, no debe estar expuesto a cambios bruscos de temperatura. Puede soportar temperaturas constantes entre 0ºC y 25ºC durante periodos prolongados y puede tolerar temperaturas de -5ºC a 45ºC durante cortos periodos de uso o almacenamiento.

2.3.3 Requisitos de Funcionamiento

Para mantener el robot en óptimas condiciones, se requiere un mantenimiento regular que garantice su seguridad y funcionamiento adecuado a lo largo de su vida útil. Este mantenimiento puede ser tanto preventivo como correctivo, abordando cualquier problema que pueda surgir.

El diseño del robot ha sido pensado para facilitar el mantenimiento, ofreciendo una estructura que permite realizar ajustes con facilidad:

- Los motores y el servo son de corriente continua, lo que significa que no necesitan mantenimiento. Además, su accesibilidad simplifica su sustitución en caso necesario.
- Las piezas están diseñadas para ser reemplazadas con facilidad en caso de rotura o deterioro.
- La fuente de alimentación consta de cuatro celdas electrolíticas (pilas), lo que facilita su reemplazo y evita posibles errores en caso de fallo de alguna de ellas.

Se recomienda realizar el mantenimiento según el uso del robot y las condiciones de operación y almacenamiento. Es aconsejable:

- Cada 4 meses, verificar la tensión nominal de la fuente de alimentación y sustituirla si es necesario.
- Cada 2 meses, revisar el estado de las resistencias de protección de los controladores.

2.3.4 Movimiento del Robot

Durante una sola impresión, el robot puede desplazarse de manera continua a lo largo del eje X y del eje Y, cubriendo una distancia de 420 mm en ambos ejes. Esto posibilita la impresión en un área cuadrada con estas dimensiones, compatible con papel estándar de hasta tamaño DIN A3, ya sea en posición vertical u horizontal.

2.3.5 Velocidad

El robot puede moverse a una velocidad máxima de 2,5 mm/s en los ejes X e Y, mientras que en el eje Z, el servo mantendrá una velocidad de rotación de 0,1 s/60º.

2.4 Útil de Escritura y Espacio de Trabajo

2.4.1 Útil de Escritura

El dispositivo de impresión utilizado deberá ser un útil de escritura estándar con un diámetro máximo de su cuerpo principal de 12 mm y un diámetro máximo de su extremo efector de 1,5 mm.

2.4.2 Entorno de trabajo

El robot debe operarse en una superficie horizontal plana y paralela al suelo que tenga una extensión mínima de 1,5876 m², lo que equivale al 300% del espacio máximo de trabajo para el que está limitado el robot en este trabajo, que es de 420x420 mm. Pero de hacer un cambio de esta opción en el código de Arduino podría funcionar sin un límite de espacio.

3. Pliego de Especificaciones de Ejecución

3.1 Proceso

La función principal del robot es imprimir en un papel o superficie designada el archivo que reciba mediante Bluetooth usando la aplicación desarrollada para su impresión.

3.2 Colocación del Robot y Referencia de Impresión

El robot se posicionará con los motores que controlan el movimiento en el eje X alineados de manera paralela al eje Y. Los motores que controlan el movimiento en el eje Y se colocarán perpendicularmente a los motores del eje X. El punto de inicio para la impresión será la esquina inferior izquierda del área de impresión seleccionada. Se recomienda asegurar ligeramente las esquinas del área de impresión a la superficie para prevenir el desplazamiento del papel durante el proceso de impresión.

Documento 5: Presupuesto

Índice general

1. Presupuesto Horas Hombre

Los tiempos de este apartado han sido estimados en función del tiempo aproximado que se ha tomado en realizar cada parte del trabajo. El precio de la hora hombre se tomará del salario medio estimado para cada profesional en España.

Ítem	Operación	Tiempo (h)	Precio Unitario (€/h)	Precio Total (€)
1	Diseño de la aplicación	35 h	20 €	700 €
2	Programación Robot	30 h	20 €	600 €
3	Diseño estructura Robot	20 h	25 €	500 €
4	Desarrollo de planos	10 h	25 €	250 €
5	Pruebas y ajustes	10 h	20 €	200 €
			Total Horas Hombre	**2250 €**

Presupuesto estimado sin tener en cuenta los impuestos del país del proyecto.

2. Costes Materiales

Los costes de materiales se van a dividir por un lado en los componentes electrónicos que conforman el robot, obteniendo los precios de mercado. Por otro lado, se hará dos propuestas para la impresión de la estructura del robot. La primera propuesta será enfocada a reproducir el prototipo para unas pocas unidades, la otra propuesta trata de obtener un presupuesto de los materiales necesarios para un número más grande de unidades.

2.1 Presupuesto componentes

Ítem	Operación	Cantidad ud	Precio Unitario (€)	Precio Total (€)
1	Arduino Uno	1	13.75€	13.75 €
2	Motor 28BYJ-48	4	2.30 €	9.20 €

3	Módulo controlador ULN2003	4	1.20 €	4.50 €
4	Micro servomotor SG90	1	3.60 €	3.60 €
5	Cable USB tipo B a tipo A	1	2.38 €	2.38 €
6	Cable USB	2	0.85 €	1.70
7	Cables Interconexionado	42	0.05 €	2.1 €
8	Betería recargable	1	30.59 €	30.59 €
9	HC-05	1	7.25 €	7.25 €
10	Ruedas Omnidirecionales	4	7.56 €	30.24 €
11	Tornillo 12x3mm	26	0.16 €	4.16 €
12	Tornillo 10x4mm	12	0.10 €	1.20 €
13	Tuerca M4	12	0.14 €	16.8 €
14	Tuerca M3	26	0.12 €	3.12 €
15	Rotulaor	1	1.10 €	1.10 €
			Total Componentes	131.69 €

2.2 Presupuesto estructura

2.2.1 Primer Presupuesto estructura

El primer presupuesto es para pocas unidades en el que se externaliza la producción. Para este apartado se ha consultado 2 proveedores eligiendo los mejores precios en cada caso. Incluimos el costo del envío de las piezas en las mismas

Ítem	**Operación**	**Cantidad ud**	**Precio Unitario (€)**	**Precio Total (€)**
1	Chasis	1	36 €	36 €
2	Sujeción batería	1	12 €	12 €
3	Soporte escritura (Integro)	1	17 €	17 €
4	Soporte Motores	4	13€	52 €

2.2.2 Segundo Presupuesto estructura

Para esta parte se ha estimado el precio con los datos que se han obtenido de la impresión 3D de este propio proyecto. Para ello se refleja que para la creación de todo el robot ha sido necesario 0.5kg de PLA y tendremos en cuenta el precio de la impresora empleada además del rollo de material utilizado.

Item	Operación	Cantidad	Precio Unitario (€)	Precio Total (€)
1	PLA	1 kg	18 €	18€
2	Impreosra d	1 ud	200€	200€
3	Cuerpo completo Robot	0.5 kg		

2.1.2 Comparación Presupuesto estructura

Para la comparación se va a hacer uso del punto de equilibrio. El punto de equilibrio es el nivel de producción o ventas en el cual los ingresos son iguales a los costos, es decir, no hay ni ganancias ni pérdidas. En el contexto de este proyecto, el punto de equilibrio es la cantidad de unidades a producir internamente (utilizando la impresora 3D) para que el costo total sea igual al costo de externalizar la producción. Para encontrar este punto de equilibrio, se debe comparar los costos fijos y variables de ambas opciones.

$$Punto\ Equilibrio = \frac{Costo\ fijo}{Costo\ Variablke\ por\ unidad - Costo\ Variable\ por\ unidad\ interna}$$

- Costo Fijo: Inversión en la impresora 3D. 200€
- Costo Variable por unidad externa: Costo de externalizar cada unidad. 117 €
- Costo Variable por unidad interna: Costo de imprimir cada unidad internamente. 9 € (0.5 kg de PLA por 18 €/kg)

$$Punto\ Equilibrio = \frac{200€}{117€ - 9€} = 1.89$$

Esto significa que, a partir de producir aproximadamente 2 unidades, la inversión en una impresora 3D empieza a ser más rentable. Para una producción superior a 2 unidades, la opción de invertir en una impresora 3D puede resultar más económica, considerando que ya se ha invertido en la impresora y solo es necesario comprar más PLA.

3. Presupuesto Total

Concepto	**Precio (€)**
Total Horas Hombre	2250 €
Total Componentes	131.39 €
Total Externalización	117 €
Total Presupuesto	2498.69 €

El presupuesto global para la producción de una unidad del robot omnidireccional es de 2498.69 €**,** sin incluir impuestos.

Comparamos ahora el presupuesto total para producir 10 unidades, invirtiendo en la impresora 3D.

Concepto	**Precio (€)**
Total Horas Hombre	2250 €
Total Componentes	1168.70 €
Total Externalización	290 €
Total Presupuesto	3708.70 €

El presupuesto global para la producción de 10 unidades del robot omnidireccional es de 3708.70 €, sin incluir impuestos.

El análisis financiero revela que, para la producción de un único prototipo de robot, externalizar la fabricación de la estructura es más rentable. Sin embargo, a partir de dos unidades, la inversión en una impresora 3D interna se amortiza, convirtiéndose en la opción más económica. Con un presupuesto total de 3708.70 € para 10 unidades, queda claro que la producción a mayor escala reduce significativamente los costos unitarios, beneficiando el proyecto y asegurando su viabilidad económica.