



Universidad Nacional de San Luis
Facultad de Ingeniería y Ciencias Agropecuarias

***DISEÑO Y CONSTRUCCIÓN DE UNA CELDA DE
FABRICACIÓN FLEXIBLE DIDÁCTICA***

Autor: Matias Iván Fajardo

Trabajo final de Ingeniería Mecatrónica

Director: Gabriel Iglesias
Codirector: Daniel Moran

Villa Mercedes, San Luis
Año 2025

DERECHO DE AUTOR

© 2024, Matias Iván Fajardo.

Se autoriza la reproducción total o parcial, con fines académicos, por cualquier medio o procedimiento, incluyendo la cita bibliográfica del documento.

DEDICATORIA

A mis padres, Carmen Rojas y Valentín Fajardo, por su amor incondicional, apoyo constante y sacrificios sin fin. Gracias por enseñarme el valor del esfuerzo y la importancia de la educación. Les debo todo lo que soy y todo lo que he logrado.

“Guarda, hijo mío, el mandamiento de tu padre, y no dejes la enseñanza de tu madre; átalos siempre en tu corazón, enlázalos a tu cuello. Té guiarán cuando andes; cuando duermas te guardarán; Hablarán contigo cuando despiertes”. (Proverbios 6: 20-22)

AGRADECIMIENTO

Este Trabajo final ha sido posible gracias a mis padres, mis hermanos y a mi novia. Quienes diariamente me motivan para alcanzar nuevas metas y me dan su apoyo en todo lo que emprendo.

Seguidamente quiero agradecer especialmente a los profesores con los que he trabajado dando clases en la facultad, por introducirme al mundo de la programación y por encender en mí una chispa de curiosidad y entusiasmo. Pude aprender muchísimo con sus experiencias.

Agradezco enormemente a la Universidad Nacional de San Luis y a todos los profesores que me formaron y compartieron sus conocimientos de manera amable y profesional.

RESUMEN

El presente trabajo final de grado tiene como objetivo el diseño, construcción e implementación de una Celda de Fabricación Flexible Didáctica (CFFD) destinada a la enseñanza; en el Laboratorio de Mecatrónica de la Facultad de Ingeniería y Ciencias Agropecuarias. La CFFD está compuesta por una cinta transportadora, una estación de almacenamiento, un robot manipulador (CXN2), que interactúa con las diferentes estaciones de trabajo, un centro de mecanizado el cual solo emula el proceso y una máquina de medición. Para coordinar las operaciones de la celda, se desarrolló un controlador adecuado para el proceso de eventos discretos. Además, se generó un sistema de monitoreo remoto, con almacenamiento en la nube, emulando la tecnología utilizada en la industria 4.0, recopilando y transfiriendo los datos en tiempo real. Se realizaron pruebas para validar la operatividad y cumplimiento de los objetivos. Los resultados obtenidos demostraron la correcta operación de la celda didáctica, permitiendo el transporte, manipulación y procesamiento de piezas, de manera autónoma. Estas experiencias validaron su funcionalidad como herramienta didáctica, proporcionando a los estudiantes experiencia práctica en automatización y manufactura flexible, abriendo la posibilidad de futuras ampliaciones y mejoras del sistema.

Palabras claves: Celda de Fabricación Flexible. Proceso de eventos discretos. Industria 4.0. Robótica.

ÍNDICE DE CONTENIDO

<u>CAPITULO 1: Propuesta</u>	11
1.1 <u>Introducción</u>	11
1.2 <u>Objetivos</u>	12
1.2.1 <u>Objetivo general</u>	12
1.2.2 <u>Objetivos específicos</u>	12
1.3 <u>Alcances y limitaciones</u>	13
1.4 <u>Marco teórico</u>	144
1.5 <u>Justificación</u>	21
<u>CAPITULO 2: Análisis y Diseño de la CFFD</u>	2222
2.1 <u>Sistema de Transporte</u>	22
2.2 <u>Estaciones de Trabajo</u>	31
2.2.1 <u>Centro de Mecanizado</u>	31
2.2.2 <u>Estación de Inspección</u>	37
2.3 <u>Estación de Almacenamiento</u>	13
2.4 <u>Robot CXN2</u>	¡Error! Marcador no definido.
2.5 <u>Layout de la CFFD</u>	¡Error! Marcador no definido.
<u>CAPITULO 3: Diseño del Controlador de la CFFD</u>	53
3.1 <u>Controlador de la CFFD</u>	53
3.2 <u>Comunicación de la CFFD con el Robot</u>	55
3.3 <u>Priorización y Gestión de Tareas de Robot</u>	57
<u>CAPITULO 4: Implementación de IoT para la Monitorización Remota de la CFFD</u>	59
4.1 <u>Protocolo MQTT</u>	59
4.2 <u>Adafruit IO</u>	61
4.3 <u>Transmisión de Datos de la CFFD a la Nube</u>	63
<u>CAPITULO 5: Recursos Utilizados y Evaluación de Factibilidad del Proyecto</u>	69

<u>CAPITULO 6: Conclusiones</u>	72
<u>Glosario</u>	73
<u>Referencias Bibliográficas</u>	74
<u>Anexo/s</u>	75
<u>Anexo A: Codigo del Controlador Mega</u>	75
<u>Anexo B: Codigo del Controlador Nano</u>	79
<u>Anexo C: Codigo del Controlador Python</u>	83
<u>Anexo D: Planos de las maquinas de la CFFD</u>	108
<u>Anexo E: Criterios de selección de sensores y actuadores</u>	113

ÍNDICE DE FIGURAS

<u>Figura 1. Celda de Fabricación Flexible.</u>	14
<u>Figura 2. Layout en Línea.</u>	15
<u>Figura 3. Layout de Lazo.</u>	¡Error! Marcador no definido.
<u>Figura 4. Layout de Escalera.</u>	¡Error! Marcador no definido.
<u>Figura 5. Layout de Campo Abierto.</u>	16
<u>Figura 6. Layout Centrado en Robot.</u>	¡Error! Marcador no definido.
<u>Figura 7. Representación de una Cola.</u>	¡Error! Marcador no definido.
<u>Figura 8. Ingreso de Elementos a la Cola.</u>	¡Error! Marcador no definido.
<u>Figura 9. Salida de Elementos de la Cola.</u>	¡Error! Marcador no definido.
<u>Figura 10. Diseño en 3D de la Cinta transportadora.</u>	2 ¡Error! Marcador no definido.
<u>Figura 11. Partes de la Cinta transportadora impresas en 3D.</u>	¡Error! Marcador no definido.
<u>Figura 12. Cinta transportadora ensamblada.</u>	23
<u>Figura 13. Motoreductor de DC usado en la Cinta.</u>	23
<u>Figura 14. Diseño del Sistema de Engranajes Rectos</u>	¡Error! Marcador no definido. 4
<u>Figura 15. Sistema de Engranajes Rectos de la Cinta transportadora.</u> ¡Error! Marcador no definido. 5	
<u>Figura 16. Arduino Mega 2560.R3</u>	¡Error! Marcador no definido. 5
<u>Figura 17. Driver L298N.</u>	¡Error! Marcador no definido. 6
<u>Figura 18. Conexiones al Driver L298N.</u>	¡Error! Marcador no definido. 6
<u>Figura 19. Sistema de Guias Laterales y Tope Fijo.</u>	¡Error! Marcador no definido. 7
<u>Figura 20. Sensor de Color TCs230.</u>	¡Error! Marcador no definido. 7
<u>Figura 21. Sensor de Color instalado en el Tope Fijo.</u>	¡Error! Marcador no definido. 8
<u>Figura 22. Sensor Ultrasonico HC-SR04.</u>	¡Error! Marcador no definido. 8
<u>Figura 23. Conexiones electrónica de la Cinta transportadora.</u>	¡Error! Marcador no definido. 9
<u>Figura 24. Diagrama de Flujo de la Cinta transportadora.</u>	¡Error! Marcador no definido. 9

<u>Figura 25. Diseño en 3D del Torno de la CFFD.</u>	30
<u>Figura 26. Modulo Infrarrojo Sensor Tcrt5000.</u>	31
<u>Figura 27. Led y Sensor Infrarrojo en el Torno.</u>	31
<u>Figura 28. Motor DC de 5V empleado en el Torno.</u>	31
<u>Figura 29. Torno de la CFFD.</u>	32
<u>Figura 30. Pines del Transistor 2N2222.</u>	32
<u>Figura 31. Circuito Electrónico de encendido del motor DC.</u>	34
<u>Figura 32. Tabla de arreglo de bytes del Display de 7 segmentos.</u>	34
<u>Figura 33. Circuito Electrónica del Centro de Mecanizado.</u>	35
<u>Figura 34. Circuito en PCB del Centro de Mecanizado.</u>	35
<u>Figura 35. Diagrama de Flujo del Funcionamiento del Torno.</u>	36
<u>Figura 36. Diseño en 3D de la máquina de Medir.</u>	36
<u>Figura 37. Diseño en 3D del sistema Piñon-Cremallera.</u>	37
<u>Figura 38. Sistema Piñon-Cremallera de la Maquina de Medir.</u>	38
<u>Figura 39. Mesa deslizante de la máquina de Medir.</u>	38
<u>Figura 40. Rodillos en cada extremo de la Mesa.</u>	39
<u>Figura 41. Interruptor Final de Carrera 1 de la máquina de Medir.</u>	39
<u>Figura 42. Sensor HC-RS04 utilizado para medir los productos.</u>	40
<u>Figura 43. Interruptor Final de Carrera 2 de la máquina de Medir</u>	40
<u>Figura 44. Shield Expansor de Arduino Nano</u>	40
<u>Figura 45. Circuito Electrónico de la Maquina de Medir</u>	41
<u>Figura 46. Diagrama de flujo de la Maquina de Medir</u>	41
<u>Figura 47. Diseño en 3D del Almacén del Plato Giratorio</u>	42
<u>Figura 48. Motor Paso a Paso NEMA 17</u>	42
<u>Figura 49. Base Impresa del Deposito con motor PAP</u>	43
<u>Figura 50. Driver A4988 POLOLUS</u>	43

<u>Figura 51. Valor Rs del Driver A4988</u>	44
<u>Figura 52. Sensor Optico Reflexivo instalado en la base.....</u>	45
<u>Figura 53. Plato Giratorio de la Estación de Almacenamiento</u>	46
<u>Figura 54. Rodillos instalados en la base del plato giratorio</u>	46
<u>Figura 55. Conexión Electronica del Plato Giratorio</u>	47
<u>Figura 56. Diagrama de Flujo del Plato Giratorio</u>	47
<u>Figura 57. Robot CXN2 de la CFFD</u>	48
<u>Figura 58. Controlador del Robot CXN2</u>	48
<u>Figura 59. Interfaz del Controlador del Robot</u>	49
<u>Figura 60. Layout centrado en un Robot</u>	50
<u>Figura 61. Layout centrado en un Robot de la CFFD</u>	50
<u>Figura 62. Base o Modulos de la CFFD.....</u>	52
<u>Figura 63. Uniones tipo eslabon de los modulos</u>	52
<u>Figura 64. Secuencia de Trabajo del Robot CXN2</u>	53
<u>Figura 65. Interfaz grafica del Controladro de la CFFD.....</u>	54
<u>Figura 66. Dashboard de comandos del Sistema</u>	55
<u>Figura 67. Indicadores del Sistema</u>	56
<u>Figura 68. Comunicación de la CFFD con el CXN2.....</u>	57
<u>Figura 69. Cola FIFO de Rutinas del Robot.....</u>	57
<u>Figura 70. Robot operando bajo el Metodo FIFO.....</u>	58
<u>Figura 71. Hilos Principales del controlador de la CFFD.....</u>	58
<u>Figura 72. Hilo encargado de gestionar el Arduino Mega</u>	58
<u>Figura 73. Hilo encargado de gestionar el Arduino Nano</u>	59
<u>Figura 74. Hilo encargado de Ejecutar Rutinas</u>	59
<u>Figura 75. Logo de MQTT</u>	60
<u>Figura 76. Arquitectura publicación/Suscripción de MQTT</u>	60

<u>Figura 77. Logo de la empresa Adafruit</u>	61
<u>Figura 78. Feeds creados en la plataforma de Adafruit IO</u>	61
<u>Figura 79. Bloques para crear Dashboard en Adafruit IO</u>	62
<u>Figura 80. Credencial proporcionada por Adafruit IO</u>	63
<u>Figura 81. Diagrama de Conexión de la CFFD con la Nube</u>	64
<u>Figura 82. Dashboard creada para la CFFD</u>	65
<u>Figura 83. Correo de Advertencia</u>	66

CAPITULO 1: Propuesta

1.1 Introducción

La presente tesis tiene como objetivo diseñar y construir una celda de fabricación flexible didáctica (CFFD) destinada a la enseñanza, en la Facultad de Ingeniería y Ciencias Agropecuarias, específicamente en el Laboratorio de Mecatrónica. La misma cuenta con una cinta transportadora, que provee la entrada de piezas, un robot manipulador, que actúa como recurso compartido entre las estaciones de trabajo, un centro de mecanizado, una máquina de medir y un depósito de salida de piezas; permitiendo la ejecución de los procesos necesarios de manera coordinada. Este sistema aporta al proceso de enseñanza aprendizaje de los conceptos de fabricación flexible e industria 4.0, ya que los estudiantes pueden observar y participar en todo el ciclo de fabricación, desde la entrada de las piezas hasta la finalización de las distintas etapas.

La implementación de esta CFFD no solo mejora la comprensión teórica de los estudiantes, sino que también les proporciona una valiosa experiencia práctica otorgando la posibilidad de ampliar e integrar la CFFD, en procesos más complejos. El trabajo aborda detalladamente cada tapa del proyecto, desde el diseño inicial, la fabricación y montaje, hasta la programación del sistema. Además, se incluyen pruebas y evaluaciones para asegurar que la CFFD funcione correctamente y cumple con los objetivos propuestos. La implementación de esta celda didáctica representa un aporte para la enseñanza de las temáticas abordadas.

1.2 Objetivos

1.2.1 Objetivo general

- Diseñar, fabricar e implementar, una celda de fabricación flexible didáctica (CFFD).

1.2.2 Objetivos específicos

- Diseñar mediante herramientas CAD, las partes que constituyen la celda: cinta transportadora, estación de almacenamiento de piezas y las estaciones de trabajo emuladas.
- Construir la Cinta transportadora, estación de almacenamiento de piezas y las estaciones de trabajo emuladas.
- Instalar sensores y aplicar la electrónica correspondiente para la puesta en marcha y el control de la CFFD.
- Programar: las Rutinas de trabajo del Robot CXN2, el controlador de la celda y realizar monitoreo de los datos en de un dispositivo

1.3 Alcances y limitaciones

El proyecto abarca las etapas del diseño, desarrollo y fabricación de la celda de fabricación flexible didáctica (CFFD), asegurando que se incluyan todos los componentes necesarios. La misma incluye la integración de una cinta transportadora, un robot (CXN2), y diversas estaciones de trabajo como el depósito de plato giratorio, emulador de centro de mecanizado y máquina de medir.

La CFFD está diseñada específicamente para el uso educativo en la Facultad de Ingeniería y Ciencias Agropecuarias, debido a que, para su implementación, se utilizaron sensores pertenecientes a la gama de opciones de Arduino y otras tecnologías educativas de bajo costo, lo que podría limitar su aplicabilidad en otros contextos como industriales.

1.4 Marco teórico

Celda de Fabricación Flexible:

Una celda de fabricación flexible (CFF) o también llamada comúnmente como celda de manufactura flexible (FMC, *Flexible Manufacturing Cell*), es un conjunto de máquinas interconectadas, que realizan una tarea específica o un paso dentro de un proceso de producción más amplio. Pueden operar de manera independiente o formar parte de un Sistema de Manufactura Flexible (FMS). Estas celdas constan de estaciones de trabajo conectadas entre sí, que emplean tecnologías de automatización y robótica para gestionar diversos procesos productivos. Algunas ventajas que ofrecen las CFF en comparación con los procesos de fabricación convencionales incluyen:

- Mejor adaptación a los cambios del producto y flexibilidad en la producción
- Disminución del stock
- Adaptabilidad a nuevas tecnologías
- Monitoreo continuo de la producción
- Ambiente de trabajo controlado

El aspecto flexible de una celda de manufactura indica que no se limita a un único tipo de pieza o proceso, sino que puede ajustarse con facilidad a diferentes diseños de producto, generalmente dentro de familias con propiedades físicas y dimensionales similares [1].

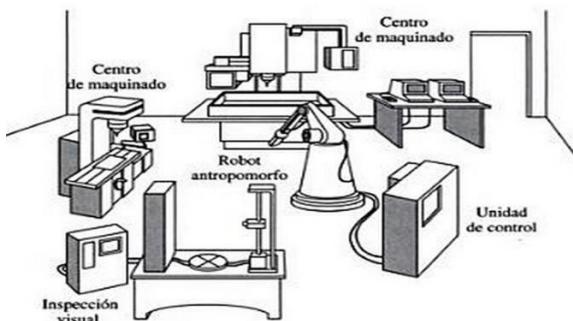


Figura 1. Celda de Fabricación Flexible
Fuente: tomado de [3]

Las celdas de fabricación flexible representan un cambio significativo en la manufactura moderna, proporcionando la capacidad de responder rápidamente a cambios en el diseño y la demanda del producto. La integración de maquinaria avanzada, sistemas de control automatizados, y software de gestión robusto, permite a estas celdas mantener una alta eficiencia y calidad en la producción, adaptándose a las necesidades dinámicas del mercado. Las CFF están compuestas por varios elementos, en forma generales son:

- **Estaciones de trabajo:** hace referencia a estaciones de carga/descargar, de mecanizado, de montaje o estaciones de inspección.
- **Sistemas de manipulación y transporte:** en este grupo se encuentran por ejemplo los robots, que son manipuladores multifuncionales reprogramables con varios grados de libertad y las cintas transportadoras que se encargan de transportar piezas a las diferentes estaciones de trabajo
- **Almacenamiento de materiales:** en este grupo se encuentran buffers intermedios, que almacenan temporalmente las piezas procesadas para evitar cuellos de botella, o las estanterías automatizadas, que organizan y almacenan las piezas antes y después del procesamiento.
- **Sistemas de control informático o controlador:** este puede ser un controlador lógico programable (PLC) o una computadora central que coordinan las operaciones de la celda.

Configuración de Layout de la CFF:

Existen diferentes configuraciones de layout, en función del sistema de manejo de materiales en una celda de fabricación flexible. Pueden distinguirse cinco tipos:

1. **Layout en línea**
2. **Layout de lazo**
3. **Layout de escalera**
4. **Layout de campo abierto**
5. **Layout centrado en un robot**

1. Layout en Línea:

El diseño en línea usa un sistema de transferencia lineal para mover las piezas entre las estaciones de procesamiento y las de carga/descarga. El sistema de transferencia en línea generalmente tiene capacidad de movimiento en dos direcciones (V); de lo contrario, el FMS opera en forma muy parecida a una línea de transferencia, y los diferentes estilos de piezas hechos en el sistema deben seguir la misma secuencia básica de procesamiento debido al flujo en una dirección.

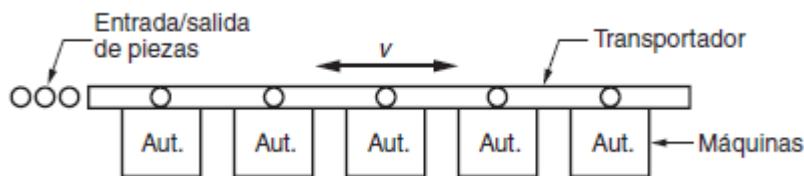


Figura 2. Layout en Línea

Fuente: tomado de [2]

2. Layout de Lazo:

Las piezas normalmente se mueven en una dirección alrededor del lazo, con la capacidad, parar y ser transferido a cualquier estación. La estación de carga y descargar (L/UL) suele estar situada en un extremo del lazo.

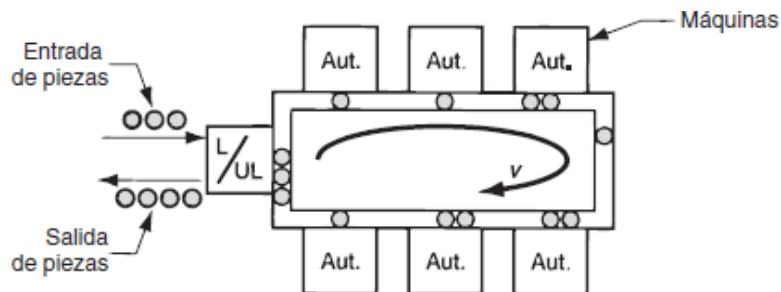


Figura 3. Layout de Lazo.

Fuente: tomado de [2]

3. Layout de Escalera:

En la distribución en Escalera, la estación de carga y descargar (L/UL) suele estar situada en el mismo extremo. La Secuencia de operación/transferencia de piezas de una máquina a otra tiene forma de escalera.

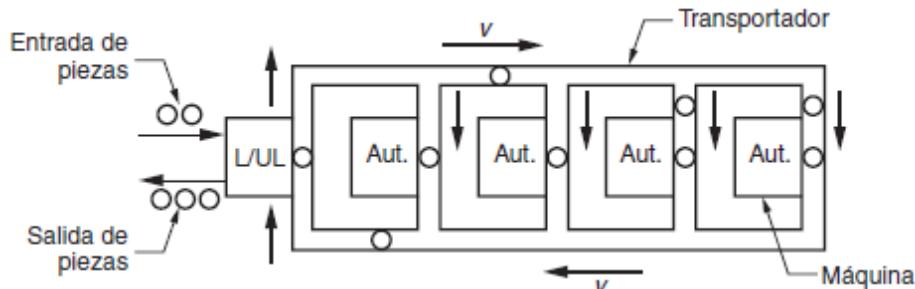


Figura 4. Layout de Escalera

Fuente: tomado de [2]

4. Layout de Campo Abierto:

Consiste en múltiples lazos y escaleras y pueden incluir caminos laterales. Este tipo de layout es normalmente apropiado para procesar grandes cantidades de familias de partes. El número de máquinas diferentes puede ser unas limitantes y las partes son ruteadas a diferentes estaciones de trabajo dependiendo de cual está disponible primero. Para el transportar los materiales, piezas o productos a las diferentes estaciones de trabajo se usan robots móviles autónomo o AGV (Vehículo de Guiado Automatizado).

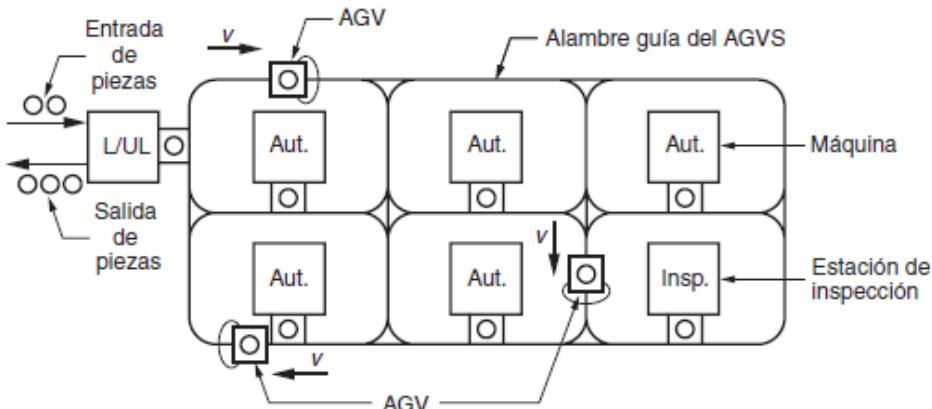


Figura 5. Layout de Campo Abierto

Fuente: tomado de [2]

5. Layout Centrado en Robot:

Una celda centrada en un robot consiste en un robot cuyo volumen de trabajo incluye las posiciones de carga/descarga de las máquinas en la celda. Esta configuración puede utilizar uno o más robots como sistemas de manipulación de materiales. Los robots industriales pueden equiparse con pinzas que los hacen muy adecuados para el manejo de partes rotacionales [2].

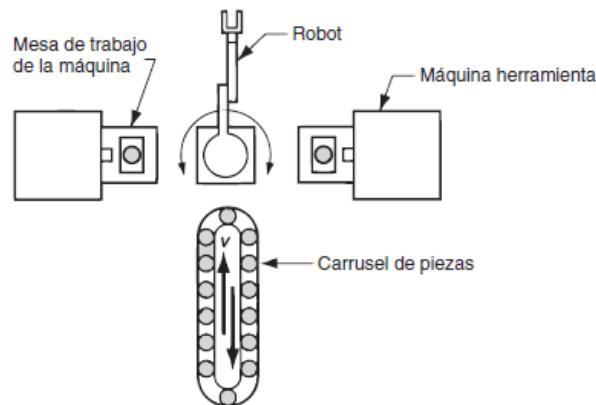


Figura 6. Layout Centrado en Robot

Fuente: tomado de [2]

Industria 4.0:

La Industria 4.0 describe la digitalización de sistemas y procesos industriales, y su interconexión mediante la internet de las cosas e internet de los servicios para conseguir una mayor flexibilidad e individualización de los procesos productivos. Es una visión de la fábrica del futuro o fábrica inteligente. La transformación digital de la industria y las empresas con la integración de las nuevas tecnologías disruptivas, el citado Internet de las Cosas, Big Data, la Nube (*Cloud Computing*) y la Ciberseguridad, todo ello enmarcado en las Ciudades Inteligentes (Smart Cities) está produciendo el advenimiento y despliegue de la Cuarta Revolución Industrial. La cuarta revolución industrial trae consigo una tendencia a la automatización total de la manufactura (fabricación).

La Industria 4.0 busca como objetivo principal la creación de fábricas inteligentes mediante la integración de sistemas de fabricación ciberfísicos (virtuales y físicos). La cuarta revolución industrial en sus orígenes consiste en la creación de máquinas y sistemas inteligentes conectados. La automatización se basa en los sistemas ciberfísicos facilitada por la Nube (cloud computing) y el Internet de las Cosas, con la ayuda indispensable de la fabricación aditiva mediante las impresoras 3D, y, además, el soporte fundamental de la inteligencia artificial y de big data, como tecnologías clave para la conversión de los grandes volúmenes de datos que se comienzan a generar en conocimiento y su uso eficiente en la toma de decisiones [4]. Los beneficios de adaptar una empresa a la Industria 4.0 son:

- Mejora la productividad y eficiencia en el uso de recursos.
- Genera información útil para la toma de decisiones en tiempo real y la planificación a mediano y largo plazo.
- Permite la creación de nuevos productos y servicios que mejoren la experiencia de los usuarios a partir de esta información recolectada.
- Integra de manera eficiente a todos los actores que intervienen en el proceso de fabricación.

Estructura Cola (FIFO)

Una cola es una estructura de datos cuyos elementos mantienen un cierto orden, de tal modo que sólo se pueden añadir elementos por un extremo, final de la cola, y eliminar o extraer por el otro extremo, llamado frente.

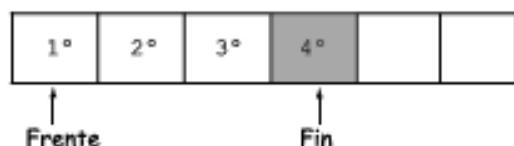


Figura 7. Representación de una Cola

Fuente: tomado de [5]

Para comprender este concepto, basta con imaginar la cola que se forma en la caja de un supermercado donde la cajera atiende a los clientes que forman la cola respetando el orden de llegada; por lo tanto, el primero que llegó también será el primero en ser atendido y en salir.

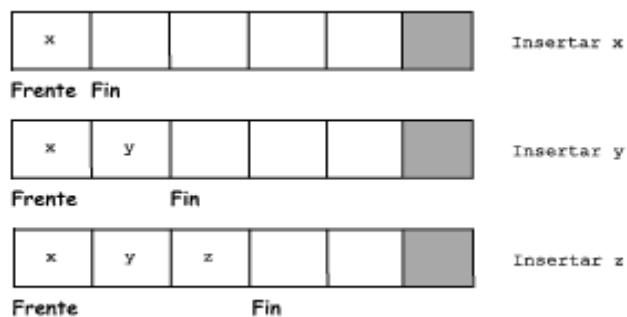


Figura 8. Ingreso de elementos a la Cola

Fuente: tomado de [5]

Las operaciones usuales en las colas son *Insertar* y *Quitar*. La operación **Insertar** añade un elemento por el extremo **final** de la cola, y la operación **Quitar** elimina o extrae un elemento por el extremo opuesto, el **frente** o primero de la cola. La organización de elementos en forma de cola asegura que el primero en entrar es el primero en salir. En una cola, al igual que en una pila, los datos se almacenan de un modo lineal y el acceso a los datos sólo está permitido en los extremos de la cola [5].

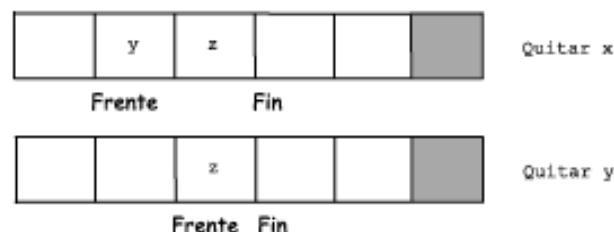


Figura 9. Salida de elementos de la Cola

Fuente: tomado de [5]

1.5 Justificación

Este trabajo final tiene como propuesta poder desarrollar e implementar una herramienta didáctica de Celda de Fabricación Flexible, para la enseñanza de Automatización y Manufactura en el Laboratorio de Mecatrónica (LABME) de la Universidad Nacional de San Luis, en la Facultad de Ingeniería y Ciencias Agropecuarias (FICA).

La CFFD no solo permite a los estudiantes aplicar conceptos fundamentales de Manufactura Flexible y Sistemas de Control, sino que también fomenta la integración de tecnologías que abarcan electrónica, mecánica, diseño y programación lo que permite abordar todos los conceptos que los estudiantes adquieren durante su formación en la carrera de Ingeniería Mecatrónica en una plataforma basada en principios de flexibilidad y adaptabilidad orientada a la Industria 4.0.

Esta herramienta didáctica permite que los estudiantes se enfrenten a escenarios de manufactura automatizada, la interacción con sistemas integrados de sensores, actuadores, robots y controladores les brindará habilidades técnicas que serán directamente aplicables en el campo laboral. Asimismo, esta celda flexible servirá como base para proyectos de investigación y experimentación futuros, promoviendo el desarrollo de nuevas tecnologías en automatización dentro del entorno académico.

CAPITULO 2: Análisis y Diseño de la CFFD

En este capítulo se presenta el análisis y diseño de las partes que conforman la Celda de Fabricación Flexible Didáctica (CFFD) como así también su organización. Cada componente de esta, están interrelacionadas y cumplen una función específica dentro de la Celda. A continuación, se describen en detalle todos sus elementos, como así también sus características principales y su función dentro del sistema.

2.1 Sistema de Transporte

El sistema de transporte de la CFFD es una cinta transportadora, que permite el ingreso de piezas a la celda. La misma cuenta con una estructura fabricada con tecnología 3D, un motor de Corriente Continua (DC) con caja reductora que genera el movimiento de la cinta y dos sensores de ultrasonido que detecta la presencia de las piezas en el extremo de la cinta y la detiene cuando sea necesario, garantizando así la sincronización con el sistema de manipulación.

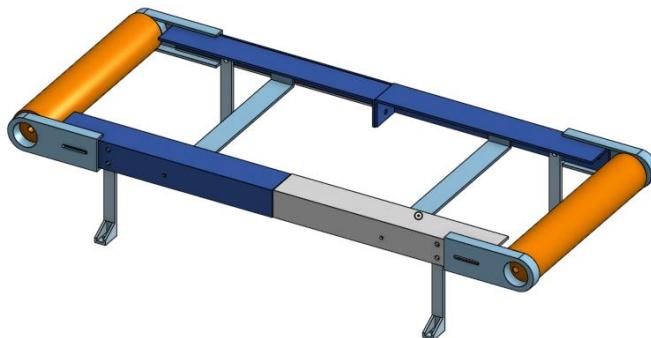


Figura 10. Diseño en 3D de la Cinta transportadora

El diseño de la cinta transportadora, así como la mayoría de los componentes de la celda, se llevó a cabo utilizando Onshape, el cual es un software CAD en línea. Es importante destacar que, durante el proceso de diseño, se consideró cuidadosamente las dimensiones de los componentes mecánicos, que integran la estructura, como lo son tornillos, rodamientos, etc., los cuales permiten la unión de las distintas piezas que conforman la cinta transportadora como así también los componentes electrónicos. A continuación, se muestra una lista con dichos elementos:

Tabla Nº1. Componentes Mecánicos de la cinta transportadora

Materiales	Descripción	Cant.
rodamientos	8x22x7 mm	4
tuercas autofrenantes	Ø 5 mm	4
Arandelas	Ø 5mm	10
Barrilla Roscada	Ø 5mm	2
Tornillos y tuercas	M3x6	20
Motor DC con caja Reductora	5V	1
Sensores de ultrasonido	HC-SR04	2
Sensor de Color	Tcs 3200	1

Tras realizar el diseño en 3D, cada pieza fue exportada como archivo STL y luego convertida a un archivo G-CODE, para su posterior impresión en una impresora 3D. Una vez impresas todas las partes, se procedió a ensamblarlas para construir la Cinta transportadora.



Figura 11. Partes de la Cinta transportadora impresas en 3D

Las dimensiones de la cinta transportadora una vez ensamblada son:

- **Largo:** 400.8 mm
- **Ancho:** 203.6 mm
- **Alto:** 90.32 mm



Figura 12. Cinta transportadora ensamblada

Para generar el movimiento de la cinta transportadora, se ha diseñado un sistema de engranajes rectos, basado en las características mecánicas del motor de corriente continua encargado de transmitir el movimiento rotatorio. Este motor, que cuenta con una caja reductora, tiene una velocidad nominal de 200 RPM sin carga y 152 RPM con carga, siendo esta última el valor considerado en el diseño de las ruedas dentadas.



Figura 13. Motorreductor de DC usado en la cinta transportadora

Se determinó una velocidad de giro de la corona o rueda conducida (ω_{corona}) que sea aproximadamente de 60 revoluciones por minuto de manera de obtener una velocidad adecuada para desplazar las piezas. Como la velocidad de giro del piñón ($\omega_{piñon}$) es igual a velocidad del motor DC con carga, entonces se tiene que la relación de transmisión (i), está dada por (1):

$$i = \frac{\omega_{piñon}}{\omega_{corona}} = \frac{152 \text{ RPM}}{60 \text{ RPM}} = 2,53 \quad (1)$$

Esto significa que la relación de transmisión es 2.53:1. Por cada vuelta completa del piñón, la corona girará 1/2.53 vueltas. El Piñón o Rueda conductora se

diseñó con 10 dientes (N_1) y de módulo 2. Usando la relación de transmisión (i) y el número de dientes del piñón se calculó la cantidad de dientes de la corona (N_2):

$$N_2 = i * N_1 = 2,53 * 10 = 25,3 \quad (2).$$

Este último valor se redondeó a 25 de manera que la rueda conducida se diseñó con 25 Dientes. El diámetro primitivo (D_p) y el paso (P) de la Corona se puede calcular de la siguiente manera:

$$D_p = M * N_2 = 2 * 25 = 50\text{mm} \quad (3).$$

$$P = \pi * M = 3,14 * 2\text{mm} = 6,28\text{mm} \quad (4).$$

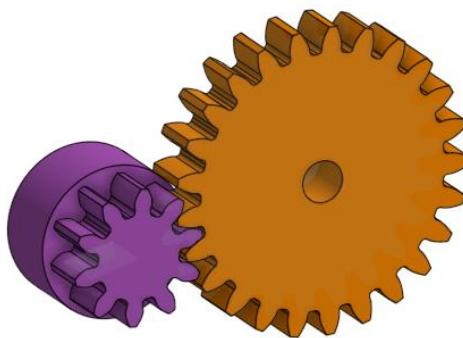


Figura 14. Diseño del sistema de Engranajes Rectos

Para obtener la velocidad lineal de la cinta transportadora, se ha convertido la velocidad de giro de la corona (ω) a radianes por segundo:

$$\omega = \frac{(60\text{RPM} * 2\pi)}{60} = 6,28\text{rad/s} \quad (5)$$

Como el diámetro primitivo es de 50 mm, entonces la velocidad lineal (v) del sistema de transporte está dado por:

$$v = \omega * r = 6,28 \frac{\text{rad}}{\text{s}} * \frac{5\text{cm}}{2} = 15,7 \text{ cm/s} \quad (6)$$

Esta velocidad obtenida es la velocidad máxima de la cinta, sin embargo, puede ajustarse para tener desplazamiento más adecuado, usando un drive.

Es importante señalar que, para el piñón, se diseñó un acople específico considerando las dimensiones del eje del motorreductor. De manera similar, se elaboró

un acople para la corona, tomando como referencia el diámetro de la varilla roscada (ver Tabla 1). Asimismo, se diseñó un soporte para el motor DC, ubicado en el extremo de la cinta transportadora, con el objetivo de mantenerlo fijo y asegurar que el engranaje conductor permanezca en contacto constante con el engranaje conducido.

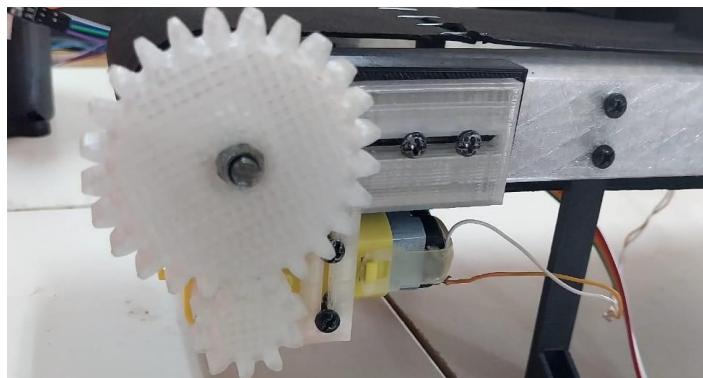


Figura 15. Sistema de Engranajes Rectos de la Cinta transportadora

El controlador de la cinta transportadora es un Arduino Mega, el cual tiene un microcontrolador *Atmega2560* y su voltaje de operación es de 5V, además cuenta con 54 pines digitales y 16 entradas analógicas. Por cada pin digital puede entregar como máximo 40mA de corriente.



Figura 16. Arduino Mega 2560 R3

Como el motorreductor consume una corriente de 250mA con carga, para suministrarle la corriente necesaria al motorreductor se utiliza un drive, es decir un circuito integrado diseñado específicamente para controlar motores, permitiendo manejar corrientes más altas y ofreciendo protección para el Arduino. En este caso se optó por

usar el *driver L298N* para la protección de los motores, de sobre corrientes y sobre calentamientos.

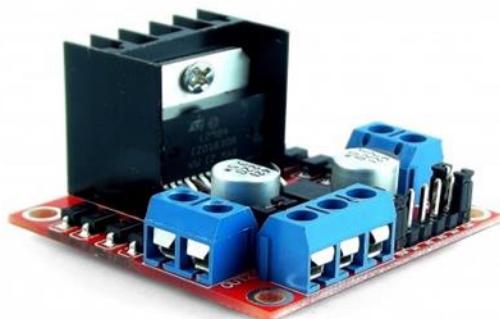


Figura 17. Driver L298N

El L298N es un puente H dual que permite controlar la velocidad y el giro de dos motores DC. Para el caso de la cinta transportadora, este drive se usa para suministrar la corriente necesaria al motorreductor y poder modificar su velocidad. En el siguiente esquema eléctrico se puede apreciar las conexiones de este:

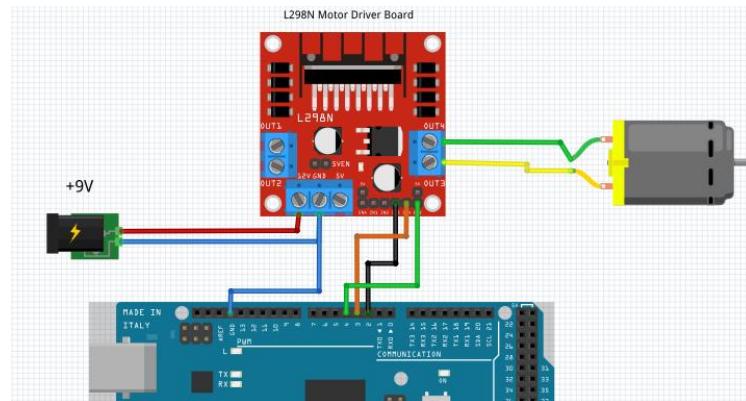


Figura 18. Conexiones al *Driver L298N*

Los pines *IN3* e *IN4* se usan para controlar la dirección del motor y con el Pin *ENB* se controla la velocidad del motor mediante *PWM*. Para girar el motor en un sentido se debe establecer en estado alto el pin *IN3* y el pin *IN4* en estado bajo, mientras que en el pin *ENB* se debe establecer un valor entre 0 y 255, siendo este último valor la máxima velocidad. Para detener el motor se debe establecer el pin *IN3* y *IN4* en estado bajo y el Pin *ENB* en 0.

Con el propósito de garantizar que la pieza se detenga en una misma posición al final del recorrido y que el robot manipulador pueda recogerla en un punto fijo, se diseñó

un sistema compuesto por guías laterales y un tope fijo. Las guías laterales tienen la función de mantener el material centrado en la banda durante todo su recorrido, mientras que el tope fijo, ubicado al final del trayecto, detiene la pieza en la posición requerida.

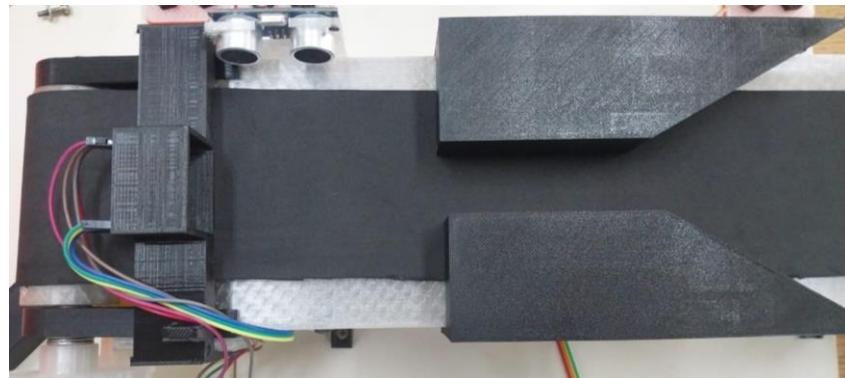


Figura 19. Sistema de guías Laterales y Tope fijo

Para dar flexibilidad a la celda de fabricación, se instaló un sensor de color TCS230 en el tope fijo. Este sensor, al momento de que la pieza se detiene en cierta posición, registra el color de esta, lo que permite identificar el tipo de producto presente. Con esta información, el sistema ajusta automáticamente el proceso de mecanizado de acuerdo con las especificaciones preestablecidas para cada tipo de producto.



Figura 20. Sensor de Color TCS230

El sensor de color fue instalado en la cinta transportadora, ya que por este medio ingresan las piezas al sistema. De esta manera, al realizar el escaneo de las piezas, el sensor proporciona información en tiempo real sobre el tipo de producto que ingresa a la celda. Esta capacidad de identificación permite que la celda pueda adaptarse de forma rápida a nuevos productos o cambios de procesos, facilitando así la incorporación de

nuevos productos o a modificaciones en los procesos existentes, respondiendo de forma flexible a los requerimientos de producción.



Figura 21. Sensor de Color instalado en el tope fijo.

Para la detección de las piezas que ingresan a la celda, se utiliza sensores ultrasónicos HC-SR04, los cuales son sensores que emiten sonido ultrasónico por uno de sus transductores, y espera a que el sonido rebote en algún objeto presente, el eco es captado por el segundo transductor. La distancia en la que detecto la pieza el sensor es proporcional tiempo que demora en llegar el eco.



Figura 22. Sensor ultrasónico HC-SR04

La cinta transportadora está equipada con dos sensores ubicados en sus extremos. El primer sensor detecta la entrada de la pieza, activando el motor para iniciar el movimiento de la cinta. El segundo sensor, situado en el extremo opuesto, identifica la llegada de la pieza al final del recorrido, deteniendo el movimiento de la cinta. Esto permite que el manipulador pueda recoger la pieza y transportarla a la estación de trabajo de manera coordinada. La conexión de todos los dispositivos al Arduino Mega se puede apreciar en el siguiente esquema:

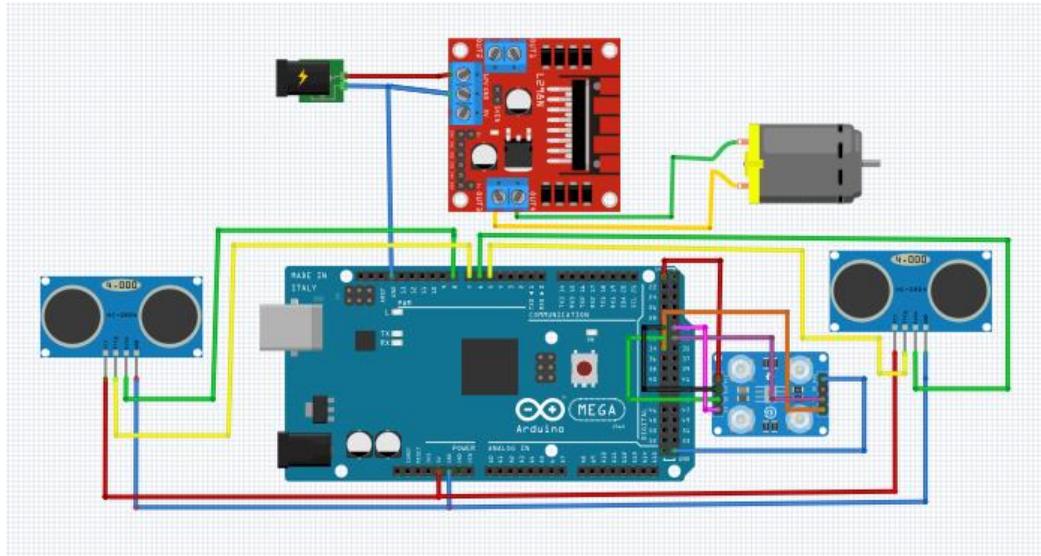


Figura 23. Conexión electrónica de la Cinta transportadora

Es importante remarcar que el Arduino Mega es el controlador de la cinta transportadora y una de las estaciones de trabajo. Esto es fundamental tenerlo en cuenta a la hora de realizar la programación de ambas maquinas, ya que se debe evitar el uso de bucles bloqueantes como lo son el “*While*” o el “*For*”, y esperas con “*Delays*”, debido a que estas funciones detienen el código y no permiten que Arduino realice múltiples tareas al mismo tiempo. Para evitar bucles bloqueantes y realizar múltiples tareas en simultaneo, una solución es usar los Hilos de ejecución y usar la función “*millis()*” en vez de “*delay()*”. El siguiente diagrama de flujo da una idea de la lógica de programación que contiene la cinta transportadora:

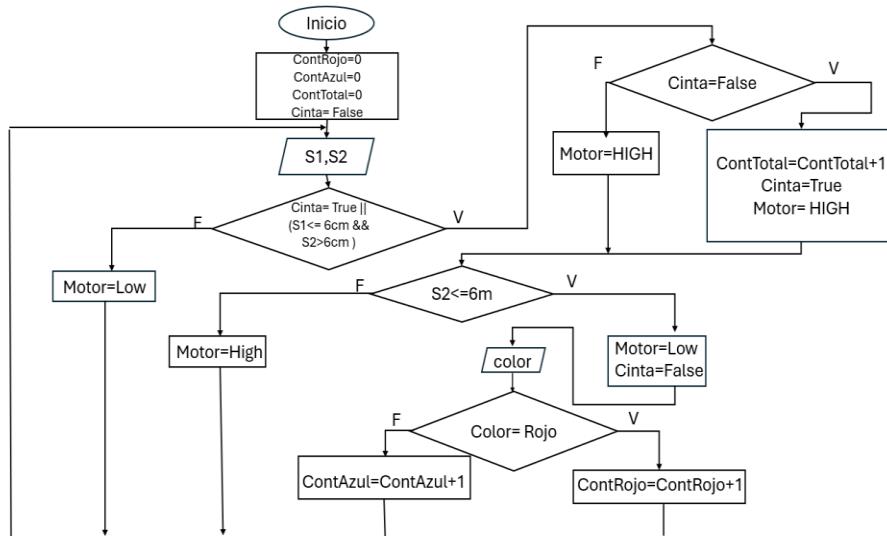


Figura 24. Diagrama de Flujo de la Cinta transportadora

2.2 Estaciones de Trabajo

Las estaciones de trabajo de la CFFD están conformadas por un centro de mecanizado y una estación de inspección, representada por una máquina de medición. Ambas estaciones realizan únicamente una simulación de sus respectivos procesos: en el caso del torno, se emula el proceso de torneado, mientras que la máquina de medir, la supervisión de las características del producto.

2.2.1 Centro de Mecanizado

Para el centro de mecanizado se diseñó un torno el cual emula el trabajo de mecanizado cuando la pieza es colocada en la misma. Durante su modelado en 3D, se consideraron las dimensiones de los componentes electrónicos necesarios para garantizar una representación funcional.

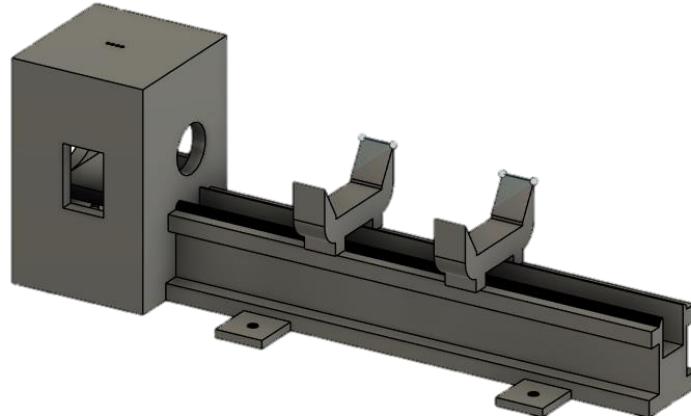


Figura 25. Diseño 3D del Torno de la CFFD

Para detectar cuando una pieza es colocada en el torno, se utiliza un sensor óptico reflectivo *TCRT5000*. Este dispositivo cuenta con un emisor que proyecta un haz de luz hacia un reflector, el cual devuelve el haz hacia un receptor integrado en el sensor. Cuando un objeto interrumpe el haz de luz, el receptor registra este cambio y activa una salida. Es importante señalar que este sensor únicamente detecta piezas de colores distintos al negro, ya que este último absorbe la luz, impidiendo que el sensor registre una variación en la señal.



Figura 26. Modulo Infrarrojo Sensor *Tcr5000*

Para indicar la puesta en marcha del torno, se instaló un diodo *LED* rojo en la parte superior de este, cuando dicho *LED* este encendido señala que el motor del torno se encuentra detenido. Al momento en que el sensor infrarrojo detecta que se depositó una pieza en el torno, el *LED* se apaga, indicando la puesta en marcha del torno. Una vez que el proceso de mecanizado finaliza, el *LED* vuelve a encenderse, indicando que el torno ha regresado a su estado de parada.

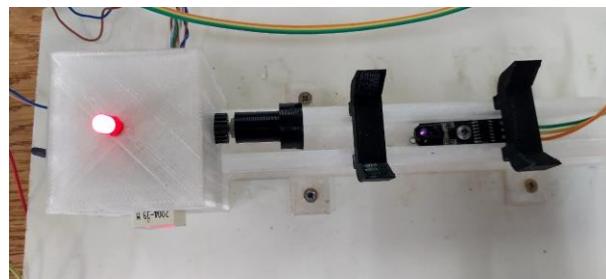


Figura 27. *LED* y Sensor Infrarrojo en el torno.

Para emular el proceso de mecanizado, se utilizó un motor de corriente continua (*DC*) instalado en el interior del torno. En el eje del motor se acopló una pieza diseñada en 3D con la forma del plato giratorio de un torno industrial. Cuando el sensor detecta la colocación de una pieza, el motor se pone en marcha, haciendo girar su eje y simulando el funcionamiento del proceso de torneado.

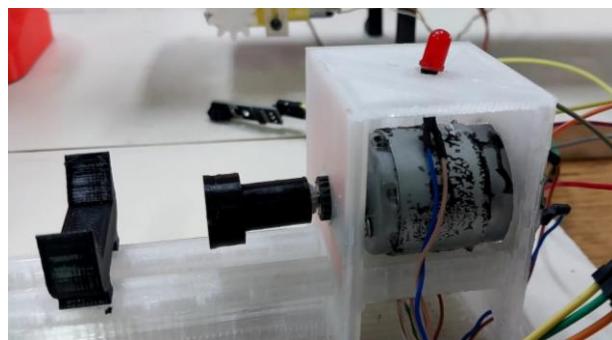


Figura 28. Motor DC de 5V empleado en el torno.

El tiempo de operación de mecanizado se visualiza en un *display* de siete segmentos el cual muestra un contador de forma descendente. Cuando dicho Contador llega a cero el motor DC se detiene y se activa el *LED*, indicando el fin del proceso.

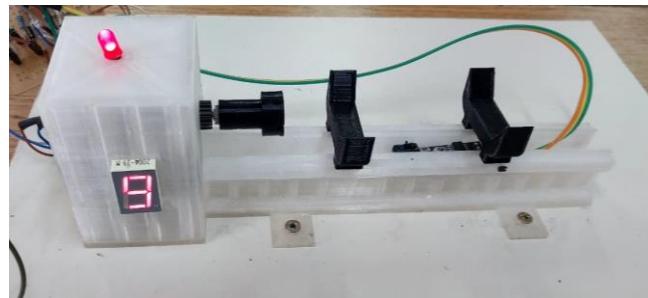


Figura 29. Torno de la CFFD.

Como tanto la cinta transportadora y el centro de mecanizado son controlados por un Arduino Mega, para el funcionamiento del motor DC, hay que tener en cuenta que el Arduino solo entrega como máximo 40 mA en cada pin digital y el motor DC consume sin carga a 5v, una corriente de 230mA. Entonces, el pin del Arduino no puede entregar suficiente corriente para hacer funcionar el motor, es por esta razón que se necesita un circuito adicional que permita manejar la corriente requerida por el motor.

Este circuito está compuesto por un transistor NPN 2N2222. Se utilizó este transistor debido a sus características de conmutación rápida y manejo de corriente media [6]. Este componente actúa como interruptor controlado por el pin del Arduino, operando en sus dos estados: **Corte** y **saturación**, permitiendo que una fuente externa de energía entregue la corriente necesaria al motor.

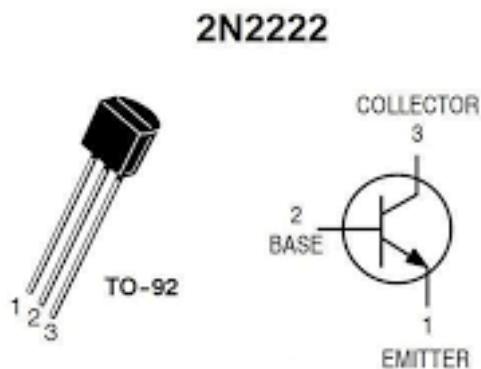


Figura 30. Pines del Transistor 2N2222

Fuente: tomado de [6]

En corte, no fluye corriente entre el colector y el emisor, es decir que el transistor está apagado, mientras que, en saturación, la corriente entre el colector y el emisor es máxima, es decir que el transistor se encuentra encendido. La base del transistor actúa como una llave que permite o bloquea el flujo de corriente entre el colector y el emisor. Cuando el pin del Arduino está en estado alto, se aplica voltaje a la base, lo que permite que fluya corriente entre el colector y el emisor. Cuando el pin de Arduino está en estado bajo, no hay voltaje en la base, y el transistor bloquea el flujo de corriente entre el colector y el emisor. La corriente que fluye hacia la base (I_b) controla una corriente mucho mayor entre el colector y el emisor (I_c), esto se debe a la ganancia de corriente (hFE o β) del transistor.

Para proteger el pin del Arduino, se coloca una resistencia en serie, a la base para limitar la corriente que fluye desde el pin hacia la base del transistor. Para calcular el valor del resistor de la base (R_b), primero se debe determinar la corriente del Colector la cual está dada por la siguiente ecuación (7):

$$I_b = \frac{I_c}{h_{fe}} \quad (7)$$

La corriente del colector en este caso será la corriente que consume el motor DC, la cual se nombró anteriormente su valor y el valor de ganancia (h_{fe}) medido con un multímetro fue de 18. Por lo que la corriente de base es:

$$I_b = \frac{I_c}{h_{fe}} = \frac{230mA}{18} = 12,77 mA \quad (8)$$

Para determinar el valor de resistencia que se colocara en la base, se debe aplicar la ley de ohm:

$$R_b = \frac{V_{out}-V_{be}}{I_b} \quad (9)$$

La tensión de salida (V_{out}) en este caso es de 5V y la tensión de base-emisor es de 0,7 voltios, entonces:

$$R_b = \frac{V_{out}-V_{be}}{I_b} = \frac{5V-0,7V}{0,01277A} = 336,72\Omega \quad (10)$$

El valor calculado es de 336,72 Ohm, pero los valores comerciales estándar más cercano es de **330Ω**.

Es importante señalar que se agregó al circuito, un diodo de protección el cual se coloca en paralelo con el motor para proteger al transistor de los picos de voltaje inverso generados cuando la corriente a través de la carga inductiva se interrumpe bruscamente. En este caso se utilizó un diodo 1N4001 y se conecta su cátodo al positivo de la fuente de alimentación y el ánodo conectado al colector del transistor. El circuito para encender el motor se muestra en el siguiente diagrama:

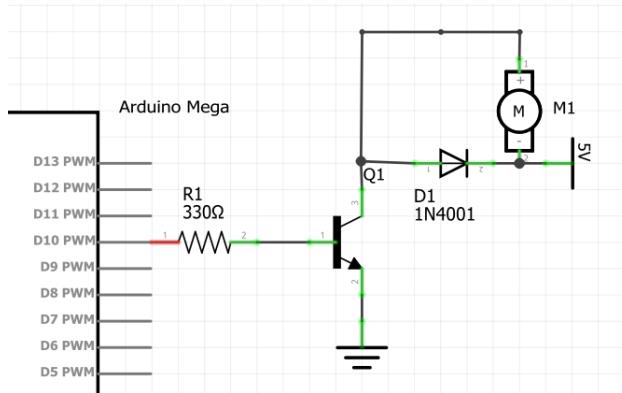


Figura 31. Circuito electrónico de encendido del motor DC

Para el *display* de siete segmentos el cual muestra el tiempo de operación del torno, contiene pines que corresponden a los segmentos A, B, C, D, E, F, G, el punto decimal (DP) y un pin de cátodo común debe conectarse a GND. En la siguiente figura muestra las distintas combinaciones para formar los números enteros entre 0 a 9.

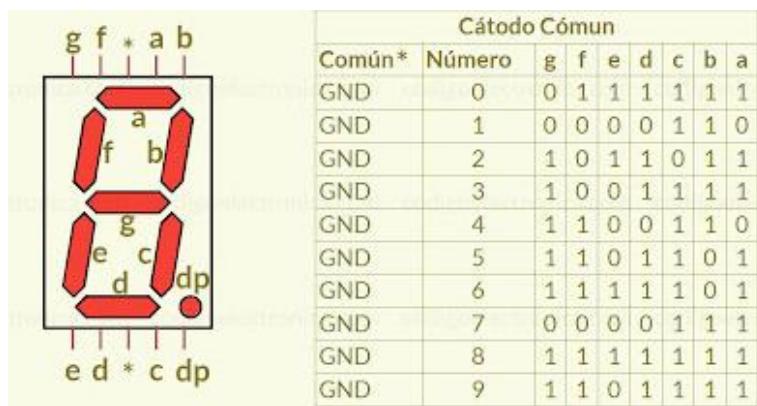


Figura 32. Tabla de arreglo de bytes del Display de 7 segmentos

Fuente: tomado de [7]

Cada segmento del *display* es un *LED*, por lo que se necesita una resistencia limitadora de corriente para evitar dañar el *LED* o el pin de Arduino, en este caso se usó

resistores de 220 Ohms. Los segmentos de “A” al “G” se conectaron en los pines digitales del 22 al 28 respectivamente. El circuito completo del centro de mecanizado se puede observar en la siguiente figura:

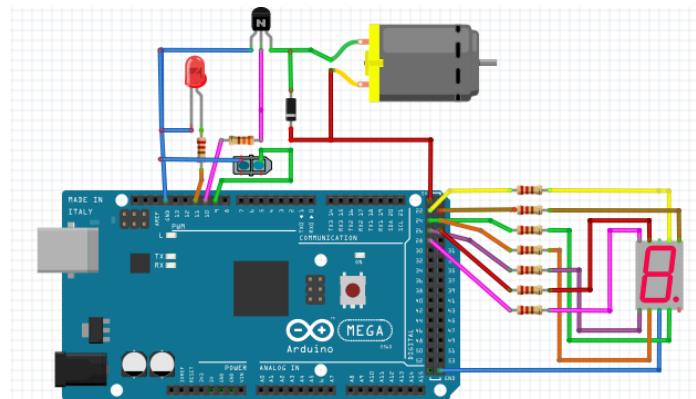


Figura 33. Circuito Electrónico del Centro de Mecanizado

En base a este circuito electrónico, se diseñó una placa de circuito impreso (*PCB*) con el objetivo de facilitar las conexiones y optimizar la organización del circuito del torno con el Arduino.

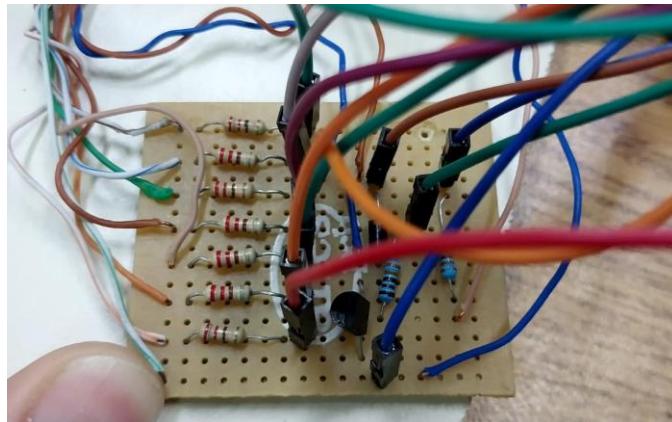


Figura 34. Circuito en PCB del Centro de Mecanizado

Para la lógica de programación del torno, se evitó usar bucles bloqueantes al igual que en la programación de la cinta transportadora, ya que el Arduino Mega es el controlador de ambas máquinas y el uso bucles o *delays* no permitirían que ambas maquinas trabajen simultáneamente. El siguiente diagrama de flujo muestra el funcionamiento del Torno:

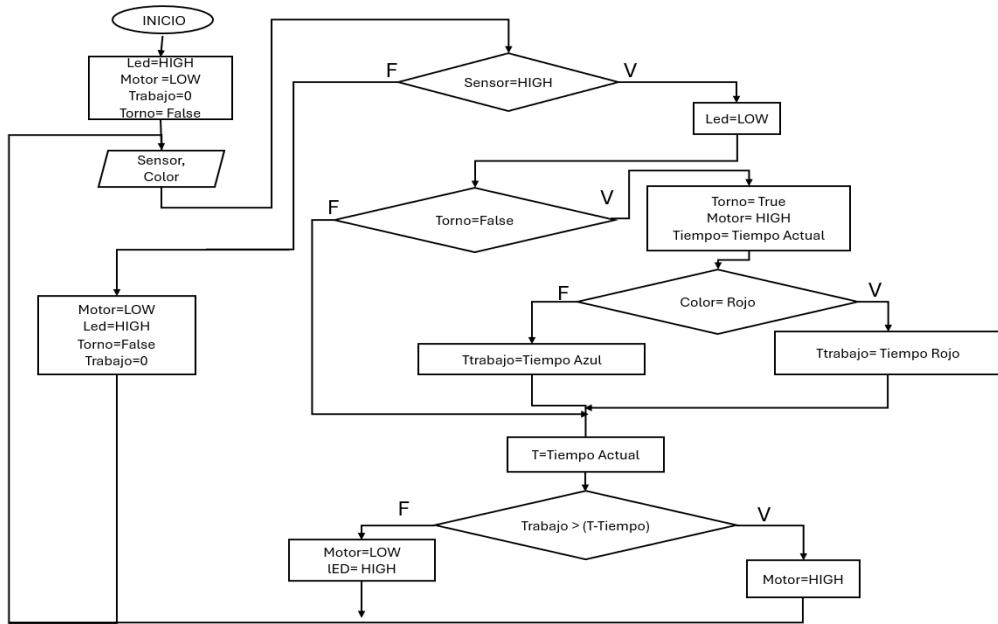


Figura 35. Diagrama de Flujo del funcionamiento del Torno

2.2. Estación de Inspección

La estación de Inspección cuenta con una máquina de medir, la cual determina si el largo de las piezas, cumple con las especificaciones técnicas establecidas. Durante su diseño, se consideraron las dimensiones del sensor encargado de realizar la medición del largo, así también, un sistema de desplazamiento rectilíneo. Este sistema incluye una mesa con un recipiente en el que el robot deposita la pieza tras completar el proceso de mecanizado.

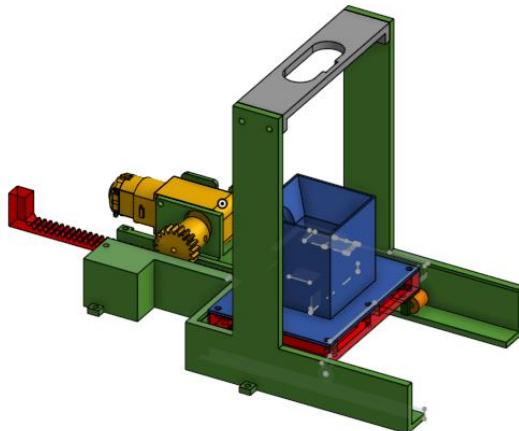


Figura 36. Diseño en 3D de la máquina de Medir

Se diseñó un sistema de piñón-cremallera que transforma el movimiento rotatorio del motor DC en movimiento rectilíneo, dicho mecanismo permite trasladar la pieza depositada en el contenedor hacia el punto de medición. El piñón, el cual se encuentra acoplado al eje del motor DC, se diseñó de modulo uno con un diámetro primitivo de veinte milímetros, con un ángulo de presión de veinte grados y de veinte dientes. Por su lado la cremallera se diseñó de igual Modulo y Angulo de presión.

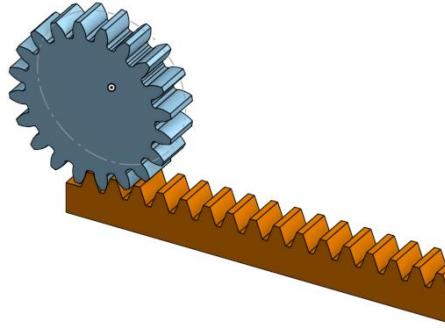


Figura 37. Diseño en 3D del sistema Piñón-Cremallera

Como el módulo del piñón (M) y de la cremallera son iguales, el paso (P) de la cremallera es:

$$P = \pi * M = 3,14 * 1\text{mm} = \mathbf{3,14\text{mm}} \quad (11)$$

El espesor o ancho efectivo (bp) y la altura (H) del Diente son:

$$bp = \frac{\pi * M}{2} = \frac{P}{2} = \frac{3,14\text{mm}}{2} = \mathbf{1,57\text{mm}} \quad (12)$$

$$H = 2,25 * M = 2,25 * 1\text{mm} = \mathbf{2,25\text{mm}} \quad (13).$$

Calculando la altura del pie del diente (hf) y sumarlo con la altura de la cabeza del diente ($ha=M$), el cual es igual al módulo, nos da la Altura total del diente de la cremallera:

$$hf = 1,25 * M = 1,25 * 1 = \mathbf{1,25\text{mm}} \quad (14).$$

$$H = hf + M = 1,25\text{mm} + 1\text{mm} = \mathbf{2,25\text{mm}} \quad (15)$$

El ancho del fondo del diente (T) se determina en función del paso y el Angulo de presión el cual es de veinte grados:

$$T = \frac{P - 4 \cdot M \cdot \operatorname{tg}(\alpha)}{2} = \frac{3,14 \text{ mm} - 4 \cdot (1) \cdot \operatorname{tg}(20^\circ)}{2} = \mathbf{0,84 \text{ mm}}. \quad (16)$$

Por último, el avance (*A*) del piñón, el cual es distancia que avanza la cremallera en una vuelta completa del piñón, este valor coincide con el diámetro primitivo (*D_p*) de la circunferencia:

$$A = \Pi * D_p = 3,14 * 20 \text{ mm} = \mathbf{62,8 \text{ mm}} \quad (17).$$

Este último dato nos dice que la cremallera avanza aproximadamente 62,8 mm por cada vuelta completa del piñón.

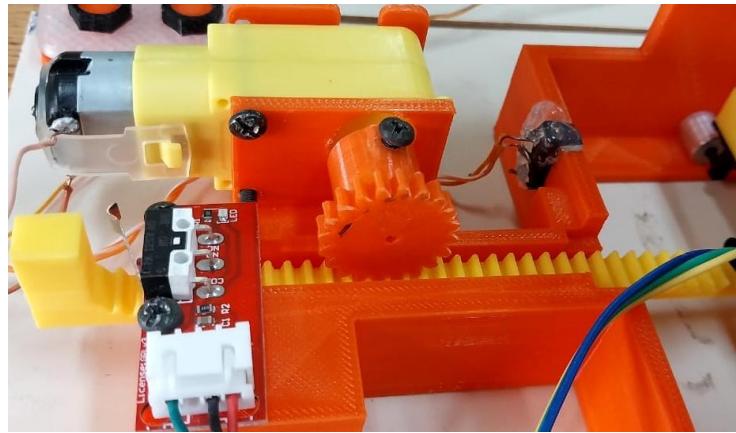


Figura 38. Sistema Piñón-Cremallera de la Maquina de Medir

La mesa donde se deposita la pieza fue diseñada con un recipiente que en su interior alberga un sensor óptico reflectivo *TCRT5000*. Cuando se coloca una pieza en el recipiente, el sensor la detecta y activa el motor *DC*, iniciando el desplazamiento hacia el punto de medición.

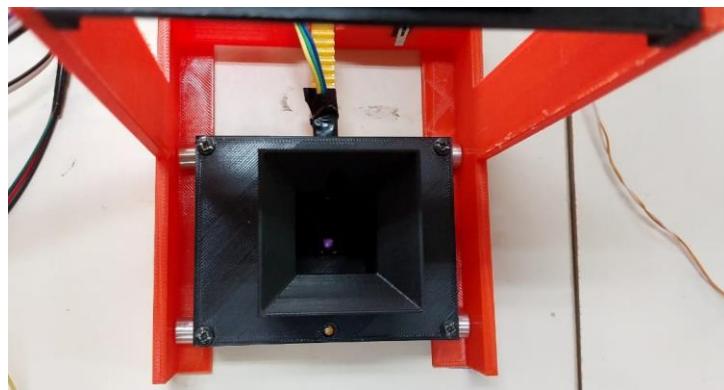


Figura 39. Mesa deslizante de la máquina de Medir

Para facilitar el desplazamiento rectilíneo de la mesa, se instalaron cuatro rodillos, cada uno se colocó en el extremo de la base, permitiendo que acompañen el movimiento de la mesa cuando el motor es activado.

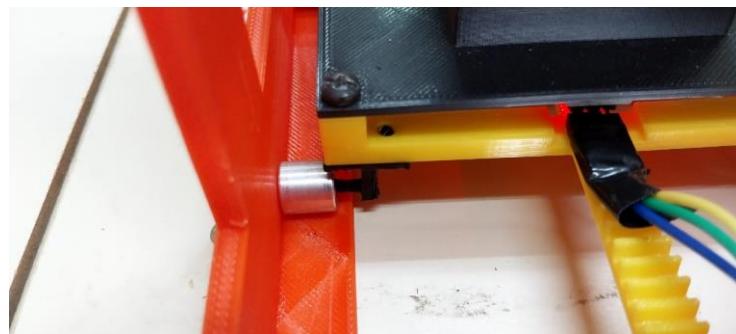


Figura 40. Rodillos en cada extremo de la mesa.

Cuando el sensor detecta la pieza y activa el motor, desplazando la mesa hacia el interior de la máquina, este se mantendrá activo hasta que la mesa establezca contacto con un interruptor final de carrera (Final de carrera 1), el cual al ser presionado enviará una señal deteniendo el motor *DC* en el punto de medición.

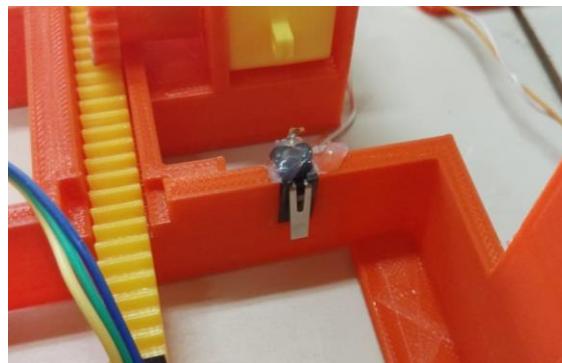


Figura 41. Interruptor Final de Carrera 1 de la máquina de Medir

Para medir la pieza, se utilizó un sensor ultrasónico *HC-SR04*, el cual se instaló en la parte superior de la máquina de medir para determinar el largo de la pieza. Una vez que la mesa con la pieza colocada en forma vertical se traslada al punto de medición, el sensor se activa y mide la distancia desde la superficie superior de la pieza hasta su ubicación. La diferencia entre esta distancia y la distancia de referencia, correspondiente a la medición desde el sensor hasta la mesa sin la pieza colocada (10,82 cm), permite calcular aproximadamente el largo de la pieza. Para medir el largo de los productos se utilizó un sensor ultrasónico *HC-SR04*, cuyo esquema se muestra en la Figura 42:

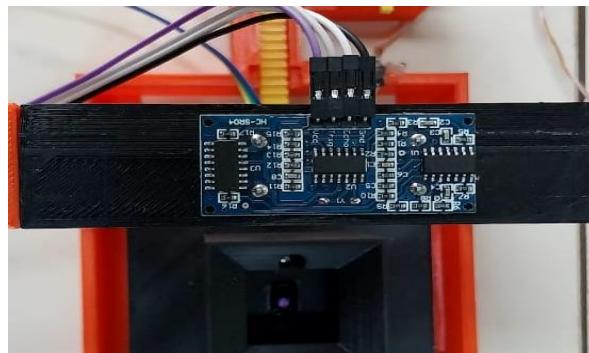


Figura 42. Sensor HC-RS04 utilizado para medir los productos

Luego de realizar la medición el motor se activa, pero girando en sentido contrario al de inicio, de manera que realiza el movimiento rectilíneo en dirección de fuera de la máquina y al entrar en contacto el tope de la cremallera con un segundo interruptor final de carrera (Final de Carrera 2), este último enviará una señal el cual detendrá el motor *DC*.

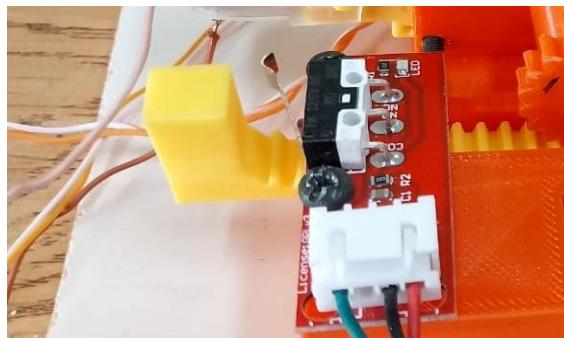


Figura 43. Interruptor Final de Carrera 2 de la máquina de Medir

El controlador de la máquina de medir en este caso es un Arduino Nano el cual se le agrego un *shield* expansor para facilitar la conexión entre el Arduino Nano y los distintos dispositivos proporcionando 14 pines de entrada y salida con *GND*, potencia y señal.

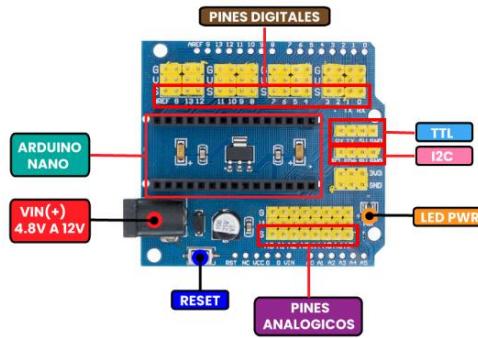


Figura 44. Shield Expansor de Arduino Nano

Fuente: tomado de [8]

Es importante decir que el Arduino Nano al igual que el Arduino Mega, controla la mitad de la Celda de fabricación, en este caso se encarga de controlar la máquina de medir y el depósito de plato giratorio. El circuito electrónico con la conexión de los diferentes dispositivos electrónicos de la máquina de medir se puede observar en el siguiente esquema:

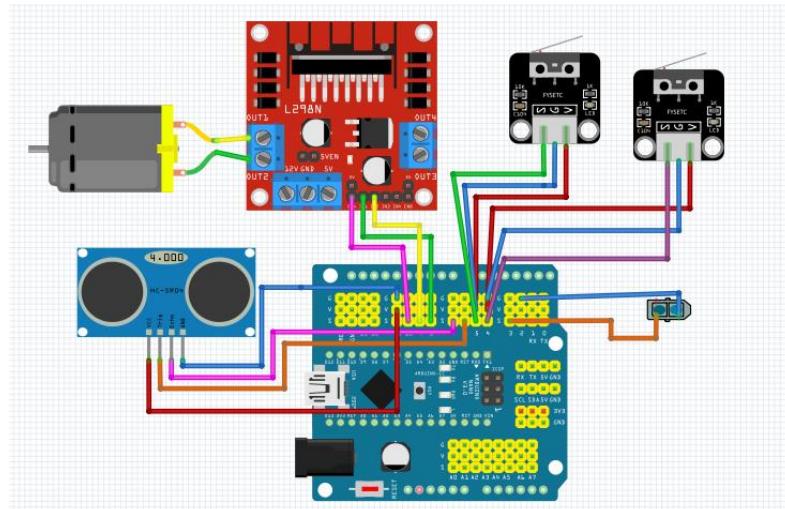


Figura 45. Circuito Electrónico de la Maquina de Medir

La lógica con la cual se programó el funcionamiento de la máquina de medir se puede observar el siguiente diagrama de Flujo:

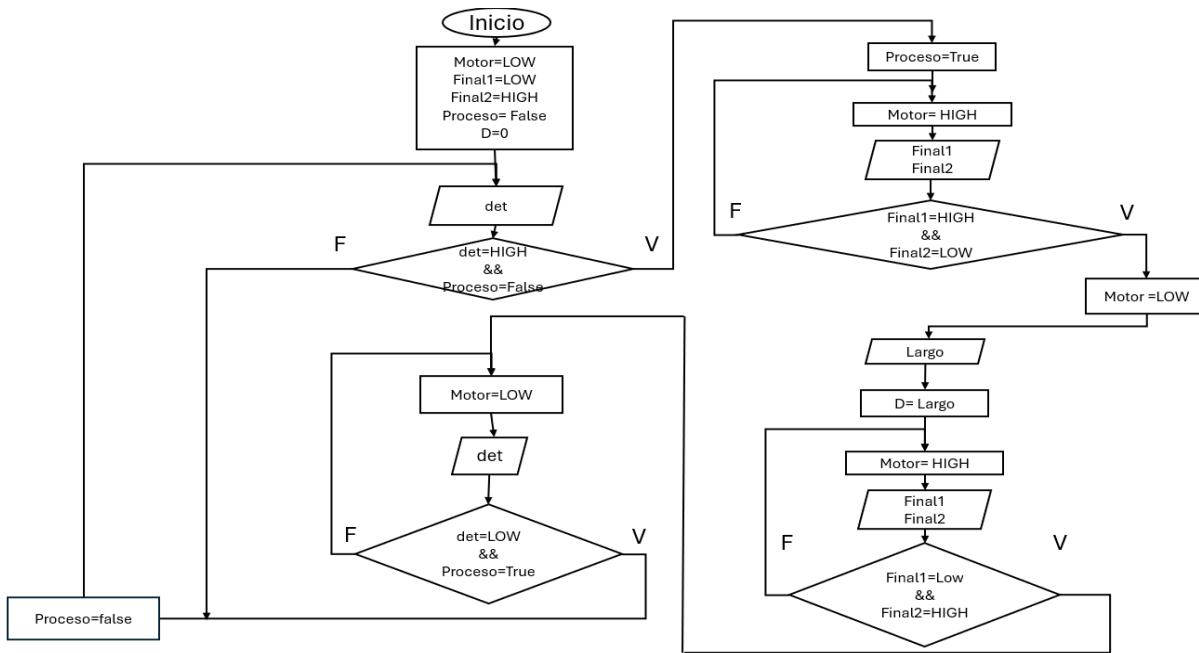


Figura 46. Diagrama de Flujo de la Maquina de Medir

2.3 Estación de Almacenamiento

La función de la estación de almacenamiento en la CFFD es almacenar los productos terminados de la celda o también facilitar el acceso rápido y organizado a las piezas necesarias para el sistema de manufactura. El almacén de piezas de la CFFD consiste en un plato giratorio capaz de almacenar hasta cuatro productos terminados.

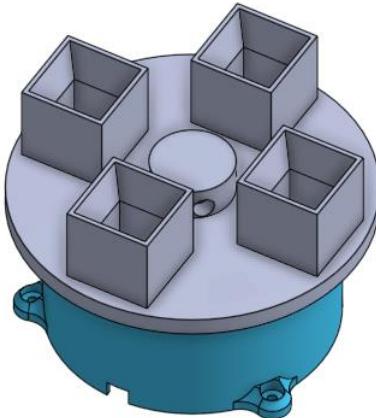


Figura 47. Diseño en 3D del Almacén de Plato Giratorio

Para el diseño de la base del almacén de piezas se tuvo en cuenta las dimensiones y características del motor encargado de girar el plato. En este caso se optó

por usar el motor paso a paso (*PAP*) Nema 17, debido a que permite movimientos precisos y repetitivos, además de que son fáciles de controlar y son compatibles con *drivers* comunes.



Figura 48. Motor Paso a Paso NEMA17

Este motor *PAP* se instala en el centro de la base, de manera que su eje se acopla al plato donde se depositarán los productos. Al colocar un producto terminado, el motor gira el plato un cierto ángulo, posicionándolo en el siguiente espacio para permitir la colocación del siguiente producto.

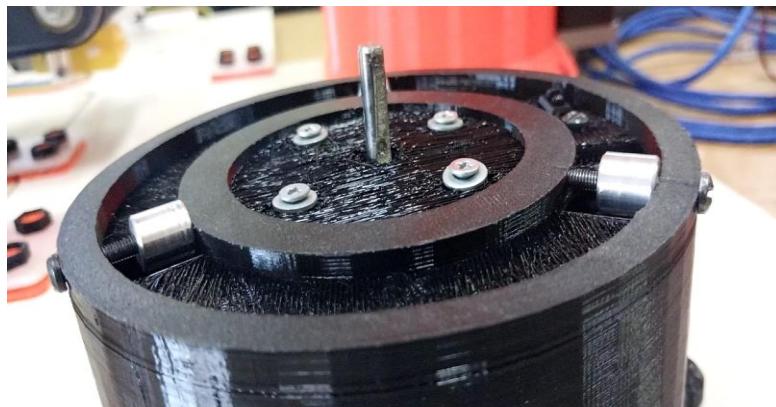


Figura 49. Base impresa del depósito con motor PAP

Para controlar el motor se optó por utilizar un *driver* A4988 conocido con el nombre de "*Pololus*". Este controlador permite controlar motores paso a paso bipolares de hasta 2 Amperes y posee protección contra sobre corriente. Trabaja con voltajes de alimentación entre 8V a 35V y puede suministrar 1A por bobina sin usar ventilación forzada o un disipador y soporta picos de corriente de hasta 2A.



Figura 50. Driver A4988 POLOLUS

Para manejar el controlador solo son necesarios 2 pines, uno para la dirección de giro (*DIR*) y otro para dar el paso (*STEP*). El pin *ENABLE* debe estar conectado a tierra (*GND*) para que el motor funcione. Sobre el circuito impreso del módulo tendremos un *preset* que nos permite establecer la corriente máxima por fase, esta medida de seguridad permite interrumpir la alimentación del motor al superar dicha corriente establecida.

Para establecer la corriente máxima, el fabricante del circuito integrado brinda una fórmula, la cual es la siguiente:

$$I_{max} = V_{ref}/(8 * R_s) \quad (18)$$

Donde *Vref* es la Tensión de Referencia y *R_s* es el valor de la resistencia de sensado o sensibilidad. Como la corriente máxima lo estableceremos en función de la corriente máxima por fase de nuestro motor *PAP*, en nuestro caso es de 1.5 A. Por otro lado, el valor de *R_s* de nuestro *driver* es de 0.1 ohm, este valor lo determinamos observando dicho Resistor:

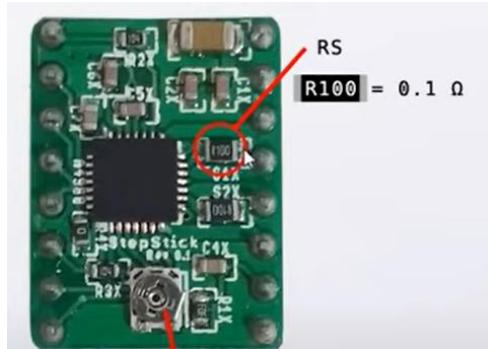


Figura 51. Valor R_s del Driver A4988

Entonces el valor que se debe determinar para ajustar con el *preset* es el valor de tensión de referencia:

$$V_{ref} = I_{max} * 8 * R_s = 1,5A * 8 * 0,1\Omega = 1.2V \quad (19)$$

Dado que el motor Nema17 tiene una resolución de 1.8 grados por paso, lo que significa 200 pasos por vuelta completa de 360°, podemos calcular cuántos grados debe girar el motor para mover el plato en función de la cantidad máxima de productos que se requiere almacenar. Como en este caso se requiere almacenar hasta 4 productos, entonces para distribuir uniformemente las piezas en el plato, el ángulo que el plato debe girar para colocar una pieza en la siguiente posición será:

$$\varnothing = \frac{360^\circ}{4} = 90^\circ \quad (20)$$

Esto significa que el plato debe girar 90 grados para mover de una pieza a la siguiente. Para calcular cuántos pasos debe dar el motor para girar esos 90 grados, se debe tener en cuenta que cada paso del motor mueve el eje 1.8 grados, por lo tanto:

$$Pasos = \frac{90^\circ}{1,8^\circ} = 50 \quad (21)$$

Entonces el motor realizará 50 pasos por cada 90° de movimiento. Por otro lado, el motor paso a paso puede iniciar desde cualquier ángulo, es decir, en una posición del plato giratorio que no necesariamente coincide con el punto donde el robot deposita los productos terminados. Es por esta razón que se diseñó un sistema de Home, el cual

permite que, al iniciar el programa, el plato giratorio se posicione correctamente en su punto de referencia, asegurando precisión en la operación. Este sistema se diseñó utilizando un sensor infrarrojo *Tcrt5000*, similar a los empleados en las máquinas anteriores. Dicho sensor se instaló debajo del plato giratorio de manera que, al iniciar el programa, si no detecta uno de los huecos de los recipientes donde se depositan los productos, el motor girará hasta que el sensor lo identifique. Una vez detectado, el motor se detendrá, posicionando el plato en su punto de inicio correcto.



Figura 52. Sensor óptico reflexivo instalado en la base

Para detectar cuándo el robot coloca un producto en el plato giratorio, se emplea el mismo sensor infrarrojo, ya que posteriormente de realizar el proceso de *Home*, el plato se posiciona de manera alineada con el punto de colocación del producto. Esto permite que el mismo sensor infrarrojo detecte la presencia del producto una vez depositado la pieza. Además, se establece un retraso de cuatro segundos tras la detección para garantizar que el manipulador tenga suficiente tiempo para completar la colocación. Posteriormente, se inicia el giro del motor, desplazando el plato 90° (anteriormente calculados), tras lo cual el sistema queda en espera hasta que se detecte la colocación de un nuevo producto.

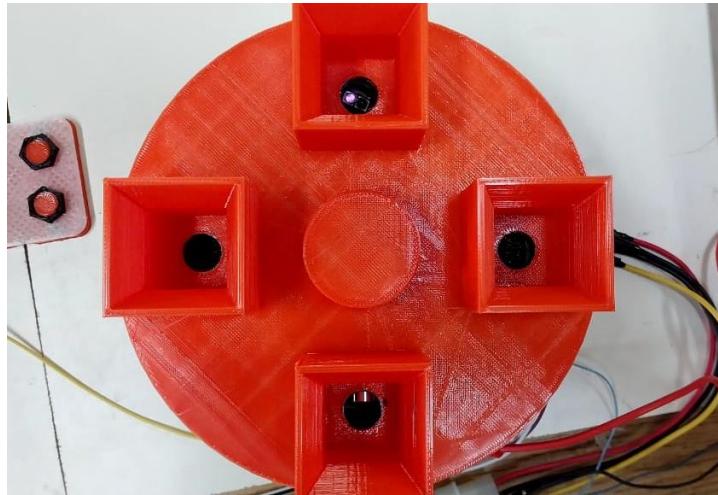


Figura 53. Plato giratorio de la Estación de almacenamiento

Para minimizar la fricción con la base y facilitar su movimiento rotatorio, se instalaron tres rodillos, los cuales acompañan el giro del plato. De este modo, el desplazamiento se realiza de manera más fluida y con menor resistencia mecánica.



Figura 54. Rodillos instalados en la base del plato giratorio

El circuito electrónico del depósito de plato giratorio se puede apreciar en el siguiente esquema, donde su controlador al igual que la máquina de medir es un Arduino Nano:

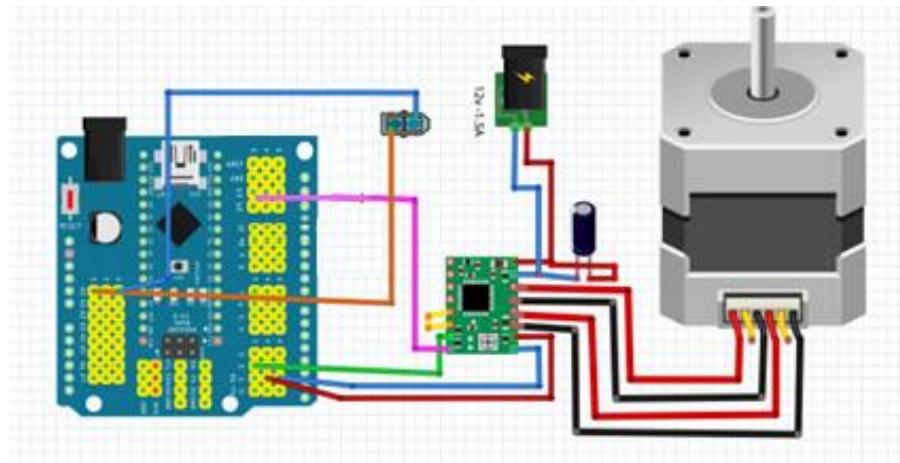


Figura 55. Conexiones Electrónicas del Plato Giratorio

Para la programación del plato giratorio, a diferencia de lo que ocurría en el controlador del Arduino Mega, es posible utilizar bucles bloqueantes, ya que nunca habrá piezas colocadas simultáneamente en la máquina de medir y en el plato giratorio. Esto se debe a que el sistema cuenta con un único robot manipulador, el cual realiza una transición a la vez. La lógica de programación del plato giratorio se puede observar en la figura 56:

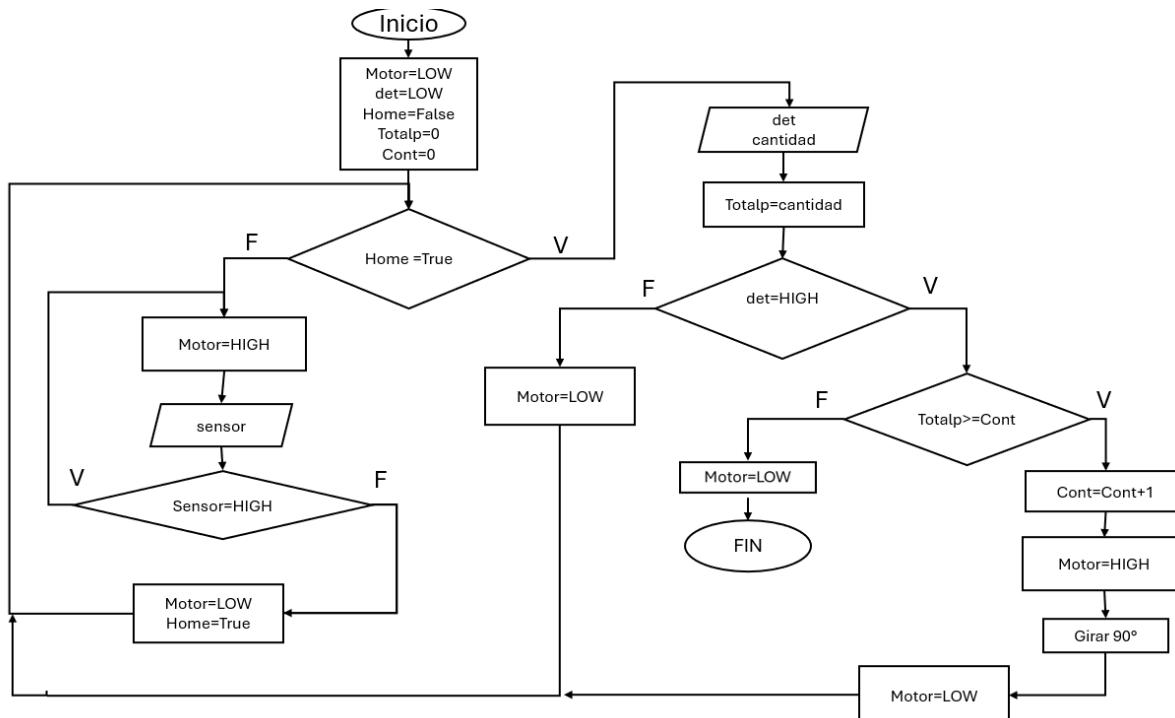


Figura 56. Diagrama de Flujo del Plato Giratorio

2.4 Robot CXN2

El sistema de manipulación de la CFFD es un Robot didáctico llamado CXN2, que fue desarrollado y construido en el Laboratorio de Mecatrónica, FICA-UNSL, para la enseñanza y experimentación en Robótica Industrial. El manipulador es de 6 grados de libertad y su función es la de mover piezas entre diferentes estaciones de trabajo dentro de la celda.



Figura 57. Robot CNX2 de la CFFD

El CXN-2, tiene un tamaño máximo extendido verticalmente de 730 mm y una capacidad aproximada de carga de 200 gr. Cuenta con una velocidad máxima aproximada de 90°/s (dependiendo la articulación) y puede tener un uso prolongado de actividad sin inconvenientes. El controlador electrónico del Robot está integrado por una placa Raspberry Pi 3B como comando principal y dos placas Arduino Mega de comando secundario.

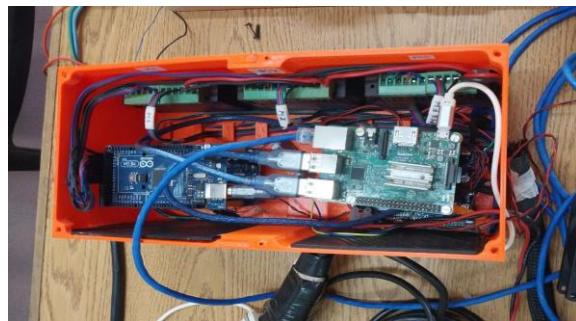


Figura 58. Controlador del Robot CNX2

La placa Raspberry es la encargada de realizar los cálculos necesarios para accionar los motores, permitiendo que el extremo del robot siga la trayectoria deseada. Cuando la Raspberry recibe una consigna de movimiento desde un dispositivo externo, genera, a través de las placas Arduino, los trenes de pulsos requeridos por los *drivers* de los motores paso a paso, así como los datos de movimiento que deben enviarse a los servomotores para ejecutar la acción solicitada.

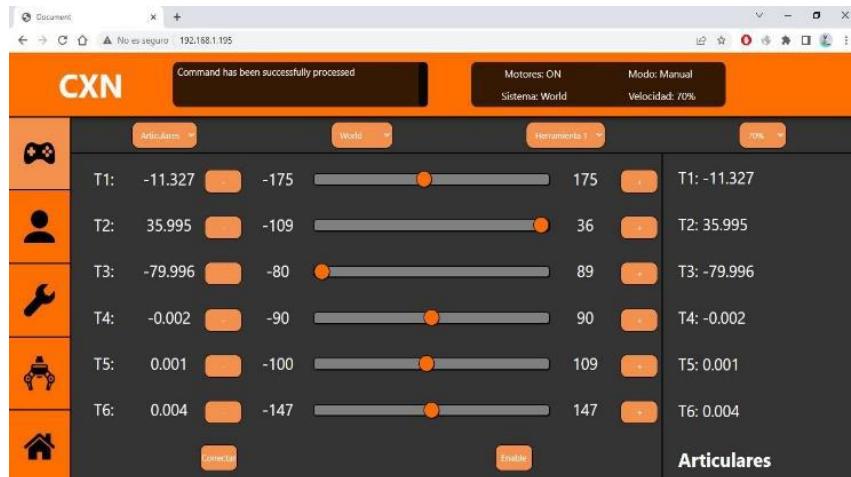


Figura 59. Interfaz del Controlador del Robot

El Robot cuenta con un software de control basado en *ROS (Robot Operating System)*, versión *kinetic*, cuya interfaz está basada en una aplicación web brindando un rápido acceso a la misma desde cualquier dispositivo que cuente con conexión a Internet. Esta aplicación web se encuentra alojada en la Raspberry Pi, la cual funciona como servidor web. Con este software podemos programar rutinas de trabajo para el robot, las cuales con enviar desde la celda el mensaje con el nombre de la rutina creada activaremos dicho proceso y daremos la orden al Robot para ejecutarlas. Antes de crear

las rutinas del robot es importante primero fijar las posiciones y la distribución de las distintas maquinas, es decir se debe tener un *layout* adecuado para la organizar la CFFD.

2.5 Layout de la CFFD

Existen diferentes configuraciones de *layout* de una celda de fabricación flexible, entre todas estas, la configuración que mejor se adapta a esta celda, teniendo en cuenta los elementos que contiene, es la configuración centrada en un Robot.

En este tipo de configuración el robot está estratégicamente ubicado de modo que su volumen de trabajo abarca todas las máquinas dentro de la celda. El robot manipula las piezas dentro de su área de alcance e interactúa con todas las máquinas necesarias, es decir que el robot es el recurso compartido de la operación actuando como un intermediario entre las diferentes estaciones de trabajo. La siguiente imagen muestra la configuración de layout de la CFFD centrada en un robot y la ubicación de cada máquina de la celda.

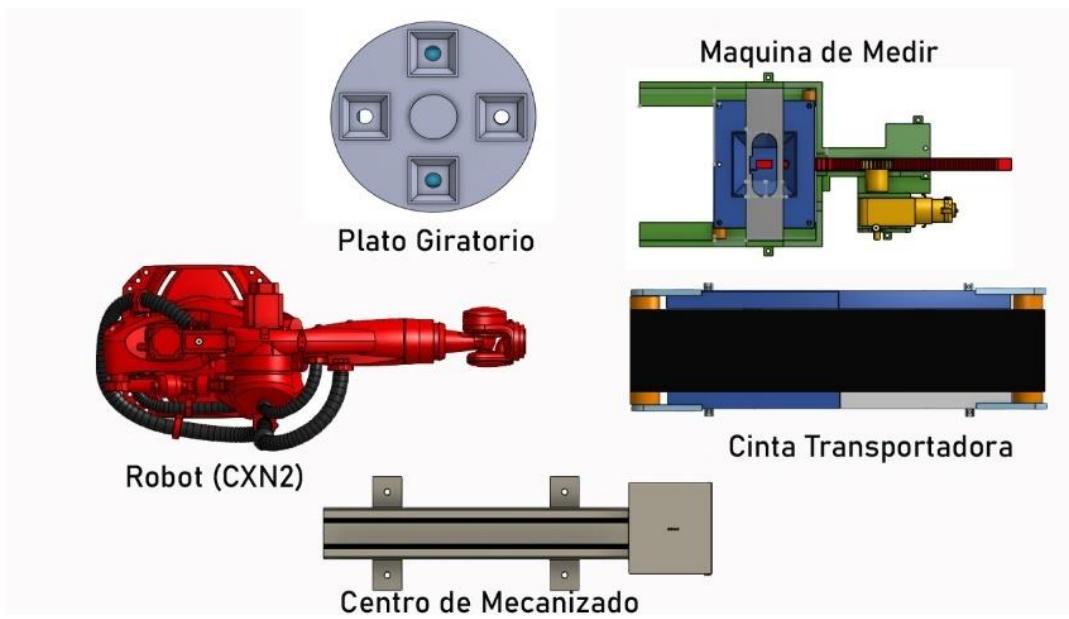


Figura 61. Layout Centrado en un Robot de la CFFD

Con la idea de que la CFFD sea desmontable y fácil de transportar, cada elemento de la celda se diseñó como un módulo o bloque independiente, colocándolo sobre una base rectangular de madera. Esta disposición permite una mejor estructuración del sistema, optimizando el cableado y la fijación de cada máquina.

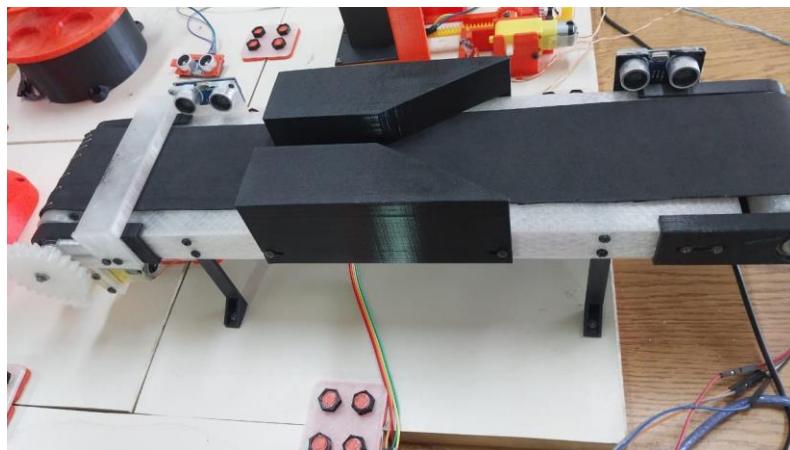


Figura 62. Base o módulos de la CFFD

Para garantizar la estabilidad y precisión en la operación del robot, se diseñaron uniones tipo eslabón en cada extremo de los módulos de la celda. Estas uniones aseguran que, al ensamblar los bloques, queden fijos y alineados, evitando desplazamientos que puedan generar errores en los puntos donde el robot recoge y deposita las piezas.

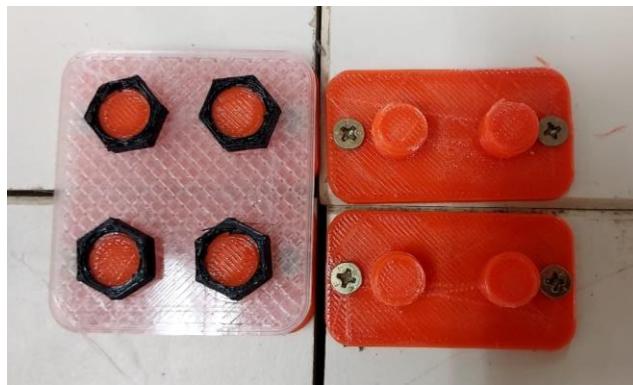


Figura 63. Uniones tipo eslabón de los módulos

Al fijar cada máquina en cada módulo se programó las rutinas o secuencias de trabajo para el manipulador. En este caso se crearon cuatro rutinas las cuales transportan las piezas de estación en el orden del proceso que se realiza en la celda. En la siguiente imagen se muestra la secuencia de las rutinas programadas para el Robot CXN2:

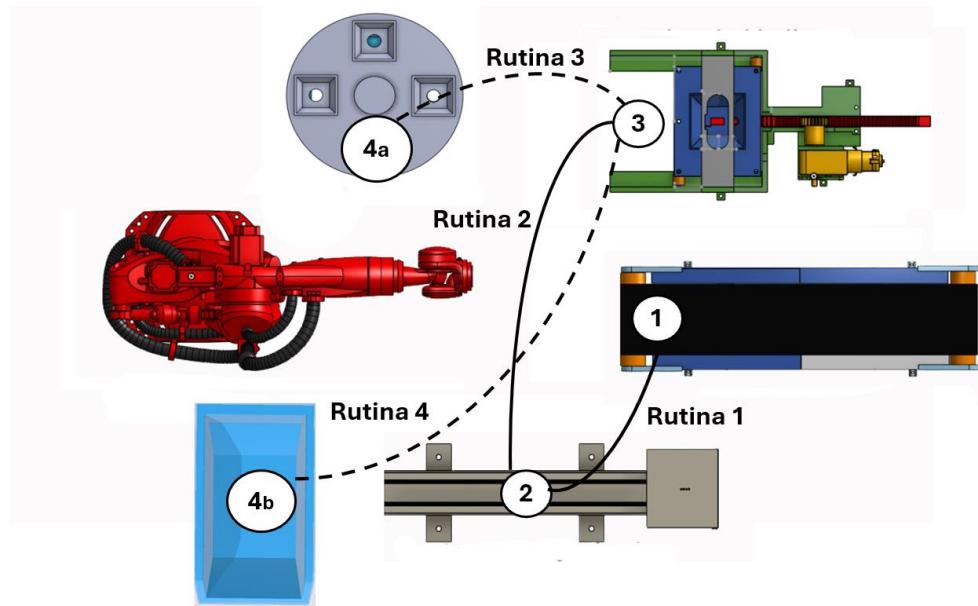


Figura 64. Secuencias de trabajo del Robot CXN2

La pieza ingresa por la cinta transportadora (1) y una vez detectada en la cinta, el controlador envía la orden al robot para ejecutar la Rutina 1, la cual consiste en llevar la pieza al Torno (2). Cuando termina el proceso de mecanizado, se ejecuta la siguiente tarea que es la Rutina 2. Esta consiste en transportar la pieza a la estación de inspección (3) donde se mide el largo de esta. Si el largo de la pieza cumple con las especificaciones de medidas previamente establecidas, se activará a la rutina 3, donde el robot transporta

la pieza al depósito de plato giratorio (4a). En caso de que la pieza no cumpla con las especificaciones de medida esta activará una rutina 4, por la cual la pieza es descartada en un contenedor (4b).

Los cuatro rutinas se definieron con un nombre específico de forma que cuando el controlador envía el nombre de dicha rutina al CXN2, este ejecuta la tarea programada con ese nombre:

- 1) "RUT_CINTA_TORNO" # Transporta la pieza de la cinta al torno
- 2) "RUT_TORNO_MEDIR" # Transporta pieza de torno a máquina de medir
- 3) "RUT_MEDIR_PLATO" # Transporta la pieza aceptada al plato giratorio
- 4) "RUT_MEDIR_DESCARTE" # Descarta pieza no válida

CAPITULO 3: Diseño del Controlador de la CFFD

En este capítulo se presenta el diseño del controlador principal de la celda de fabricación flexible didáctica (CFFD), el cual permite la interacción entre la interfaz tipo SCADA y los distintos elementos de la celda como así también establecer comunicación con el manipulador (CXN2).

3.1 Controlador de la CFFD

El controlador de la CFFD se encarga de gestionar la comunicación entre una computadora central y los dos Arduino (Mega y Nano) que controlan las diferentes máquinas de la Celda, conectados a ella a través del puerto serie, además este se encarga de recibir y subir los datos de la celda a la nube para su monitoreo remoto. Este controlador fue desarrollado en Python, dado que este lenguaje ofrece múltiples herramientas para la creación de interfaces gráficas y la comunicación con dispositivos externos. Se utilizó la librería *Tkinter* por su simplicidad, para el desarrollo de la interfaz gráfica del controlador logrando un sistema similar a un sistema SCADA.

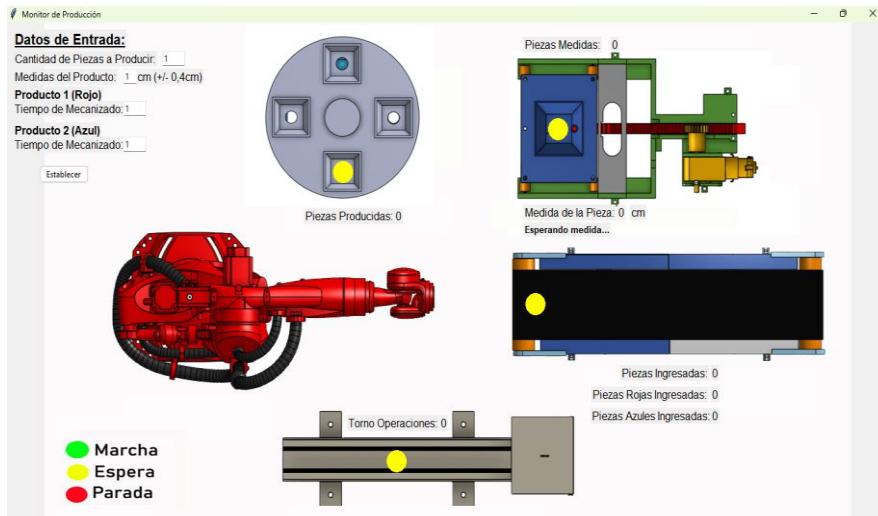


Figura N°65. Interfaz del controlador de la CFFD

El controlador cuenta con un panel de control donde se puede ingresar datos y dar a la celda órdenes de producción como: Cantidad de piezas que se quiere producir, la medida del producto final depositado en el almacén y tiempos de mecanizado para dos piezas diferentes (Producto 1: Rojas y Producto 2: Azules) que ingresan a la celda. Con fines didácticos, estos tiempos de mecanizado simulan el trabajo del torno sobre la pieza. Cuando se ingresan los datos y se selecciona el botón “Establecer”, el controlador envía estos datos a ambos Arduinos por puerto serie.

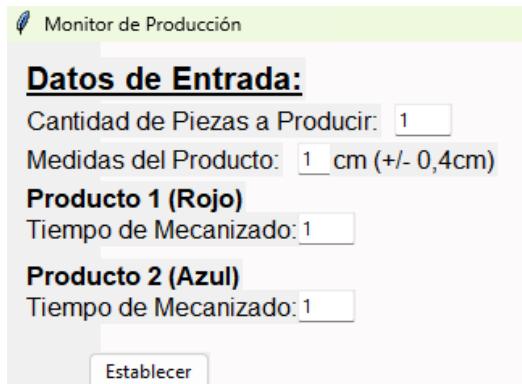


Figura N°66. Dashboard de comandos del Controlador

Cuando se ingresan piezas a la celda, en la interfaz del controlador es posible monitorear y ver donde se encuentran las piezas, como así también el estado de las máquinas. El sistema de monitoreo indica mediante luces circulares el estado de las máquinas de modo que, si se observa una luz verde, indica que la maquina se puso en marcha o está en estado operativo. Una luz roja indica que la maquina se detuvo o

finalizó su proceso y por último una luz amarilla, indica que está esperando que ingrese una pieza a la misma.

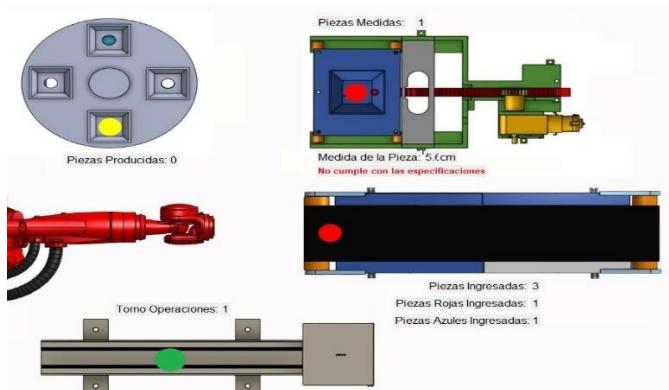


Figura N°67. Indicadores del Sistema

Además de las indicaciones con colores, cada vez que se detecta una pieza en una máquina el sistema muestra un registro de la cantidad de piezas producidas por la máquina, como así también las características de medidas en el caso de la máquina de medir.

3.2 Comunicación de la CFFD con el Robot

Para comunicar la celda con el manipulador CXN2, se utilizó un protocolo cliente-servidor basado en sockets TCP/IP, donde el controlador actúa como cliente y el robot como servidor. Para poder configurar el socket (cliente), se utilizó la librería “socket” en *Python* para crear un cliente TCP/IP con conexión por flujo y establecer la conexión al servidor del robot, la cual tiene la IP: **10.10.10.10** y escucha en el puerto **40001**:

```
> sock=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
> server_addres=('10.10.10.10',40001)
> print(sock.connect(server_addres))
```

Cuando se establece la comunicación de ambos dispositivos, la celda puede enviar y recibir datos al robot mediante mensajes textuales. De esta manera el robot puede enviar a la celda una cadena de texto: **“RUTINA:OK”** para indicar que está libre o finalizó una rutina. Para ello se creó una función que corre en un hilo paralelo llamada **“escuchar_robot”** y está constantemente esperando mensajes del robot. Para leer el

mensaje recibido se usa la función “**Socket.recv**”. El controlador recibe el mensaje del robot convirtiéndolo texto y lo guarda en la variable llamada “**mensajerobot**”.



Figura 68. Comunicación de la CFFD con el CXN2

El robot tiene programado cuatro rutinas para mover las piezas entre las diferentes estaciones de la celda. Para poder enviar la orden al CXN2, de ejecutar una rutina específica, se usó la función “**sock.sendall(rutina.encode())**”, donde **rutina** es una cadena de texto que contiene el nombre del comando que se quiere ejecutar: por ejemplo: “**RUTINA:RUT_CINTA_TORNO**”, es la rutina donde el robot transporta la pieza desde la cinta transportadora al torno. La función “**.encode()**” convierte esa cadena en una secuencia de bytes, debido a que los sockets solo pueden enviar datos binarios.

3.3 Priorización y Gestión de Tareas del Robot

En el controlador desarrollado, para la gestión de tareas del robot se implementó el método *FIFO* (*First In, First Out*), cuya función principal es asegurar que las acciones o rutinas del robot se ejecuten en el mismo orden en que las piezas ingresan a la celda de fabricación flexible. Este enfoque permite mantener la lógica de producción continua y ordenada, evitando conflictos en el uso de las máquinas y asegurando que se respete la secuencia de procesamiento.

Para lograr esta gestión secuencial, se utilizó la clase “**deque**” del módulo **collections** de *Python*, que facilita la implementación eficiente de una cola. Cada vez que se detecta una nueva pieza mediante los sensores conectados al Arduino Mega o Nano, se agrega una rutina específica a esta cola. Estas rutinas se están codificadas como comandos en formato: “**RUTINA:<nombre de la rutina>**” y corresponden a diferentes acciones que el robot debe ejecutar, tales como transportar una pieza desde la cinta al torno, del torno a la máquina de medición, o decidir entre enviar la pieza al plato giratorio

si cumple con la especificación de medida, o descartarla en un contenedor en caso de no cumplir con la medida establecida.

```
Cola actual: ['RUTINA:RUT_CINTA_TORNO']
Cola actual: ['RUTINA:RUT_TORNO_MEDIR']
Mensaje del robot: RUTINA:OK
Cola actual: ['RUTINA:RUT_TORNO_MEDIR']
Mensaje del robot: RUTINA:OK
Cola actual: ['RUTINA:RUT_TORNO_MEDIR', 'RUTINA:RUT_MEDIR_DESCARTE']
```

Figura 69. Cola FIFO de Rutinas del Robot

Para evitar colisiones o acciones simultáneas que comprometan la operación de la celda, se establecieron condiciones para ejecutar cualquier rutina del robot, considerando si la máquina de destino está libre u ocupada. Además, se verifica que, al momento de enviar una orden al robot, este no esté ejecutando otra tarea. Para ello, se utilizó una bandera mediante una variable que almacena el mensaje enviado por el robot (*mensajerobot*), el cual indica si se encuentra ocupado o disponible ("OCUPADO"/"RUTINA:OK"). De esta manera, cuando una pieza es detectada o finaliza su procesamiento en una máquina y debe ser transportada a la siguiente estación, si el robot ha enviado el mensaje "RUTINA:OK" y la máquina de destino está libre (indicada con el color de monitoreo amarillo), entonces se ejecuta la rutina correspondiente. Posteriormente, se elimina dicha rutina de la cola y se actualiza el estado del robot a "OCUPADO".

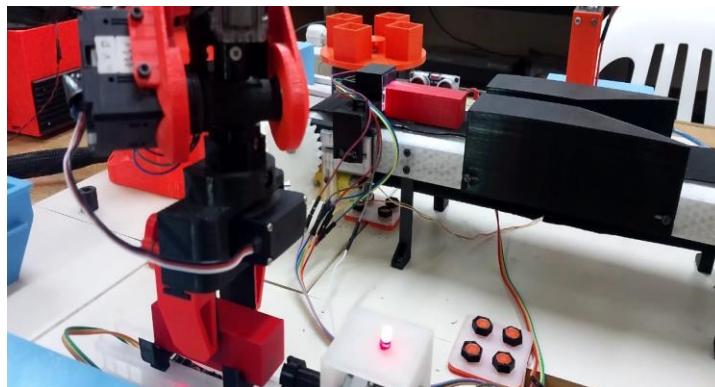


Figura 70. Robot operando bajo el método FIFO

Este enfoque de control por cola permite mantener la trazabilidad del flujo de piezas, asegurando un uso exclusivo de cada estación de trabajo, y facilitar la escalabilidad del sistema en caso de agregar nuevas rutinas o máquinas. Además, al

mantenerse la lógica *FIFO*, se asegura una gestión equitativa y ordenada para todas las piezas, minimizando posibles errores o sobreescrituras de información durante el procesamiento. Una vez alcanzada la cantidad de productos establecida para la producción, y estos son depositados en el sistema de recolección del plato giratorio, el sistema detiene el procesamiento de nuevas piezas. En consecuencia, la cola de rutinas se interrumpe y el robot deja de transportar piezas dentro de la celda.

Las pruebas y resultados obtenidos evidencian el correcto funcionamiento del controlador de la celda como así también la comunicación con los Controladores Arduino y el Robot CXN2. La lógica de control permite gestionar de forma eficiente las rutinas de operación de la celda. La implementación de una cola FIFO y la coordinación entre los distintos subsistemas demostraron ser elementos clave para lograr un control ordenado y fiable del flujo de piezas.

Para gestionar la comunicación con los dispositivos Arduino y el robot, y así poder leer datos y enviar órdenes, el controlador fue diseñado utilizando múltiples hilos de ejecución. Esta arquitectura permite organizar y controlar de forma paralela la celda. El controlador cuenta con tres hilos principales, los cuales son: Hilo Arduino Mega, Hilo Arduino Nano y Hilo Ejecutar_Rutinas.

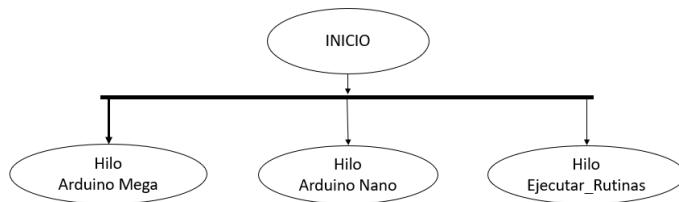


Figura 71. Hilos Principales del Controlador de la CFFD

El hilo encargado del Arduino Mega se ocupa de la comunicación con este dispositivo a través del puerto serie. Su función principal es enviar órdenes a las máquinas controladas por el Arduino Mega, así como recibir la información proveniente de estos equipos. Esta información es utilizada por el controlador para determinar la ubicación de las piezas dentro de la celda. La figura 72, se presenta el diagrama de flujo que describe su funcionamiento:

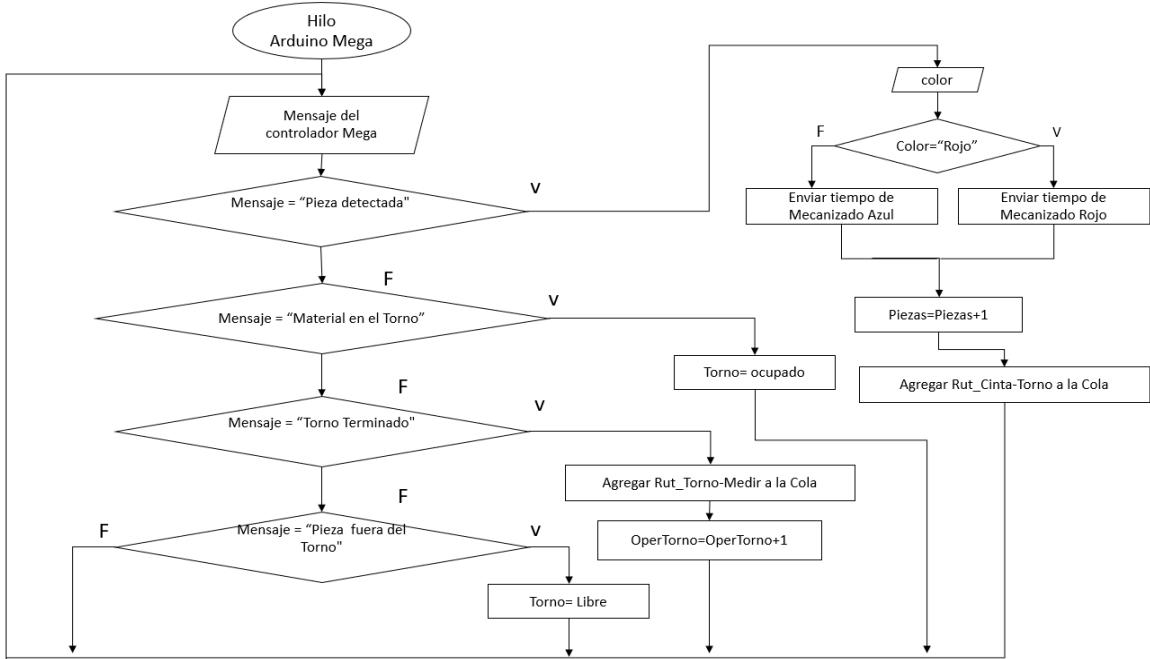


Figura 72. Hilo encargado de gestionar el Arduino Mega

De igual manera el hilo encargado del Arduino Nano, el cual se muestra en la figura 73, se ocupa de la comunicación con este dispositivo a través del puerto serie y se encarga de enviar información al controlador Nano como por ejemplo la medida que debe cumplir las piezas, como así también leer los datos enviados por el microcontrolador, en cuanto al largo de la pieza medida y la cantidad de piezas producidas.

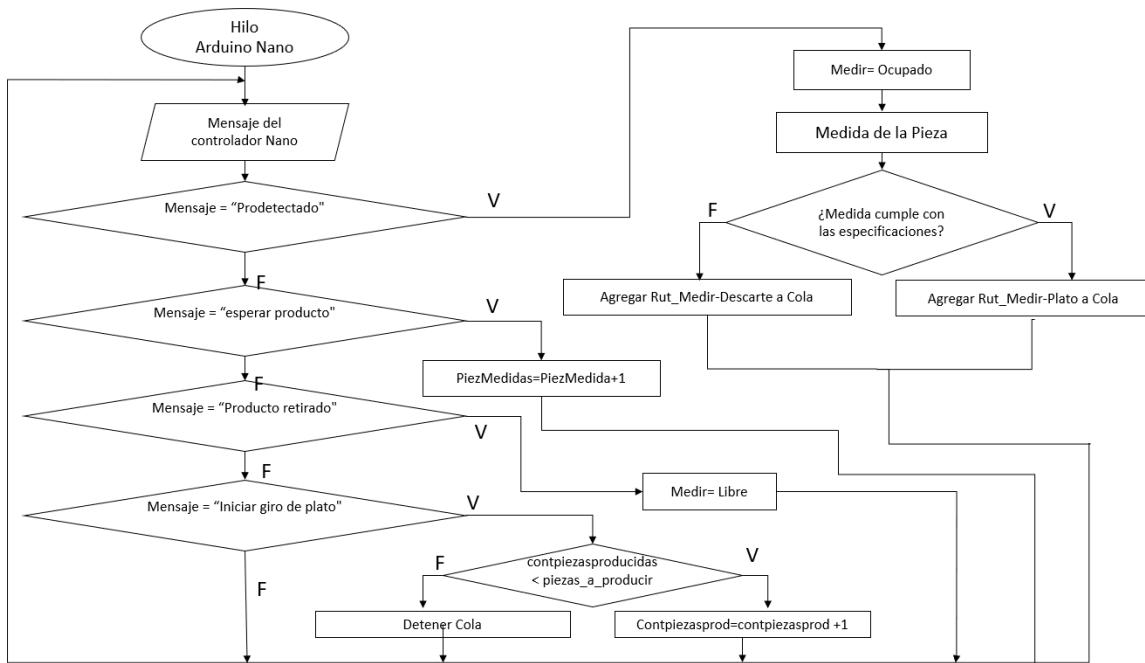


Figura 73. Hilo encargado de gestionar el Arduino Nano

Por último, el Hilo Ejecutar Rutinas se encarga de gestionar las tareas que el robot debe ejecutar en función del orden de rutinas almacenadas en la cola (FIFO).

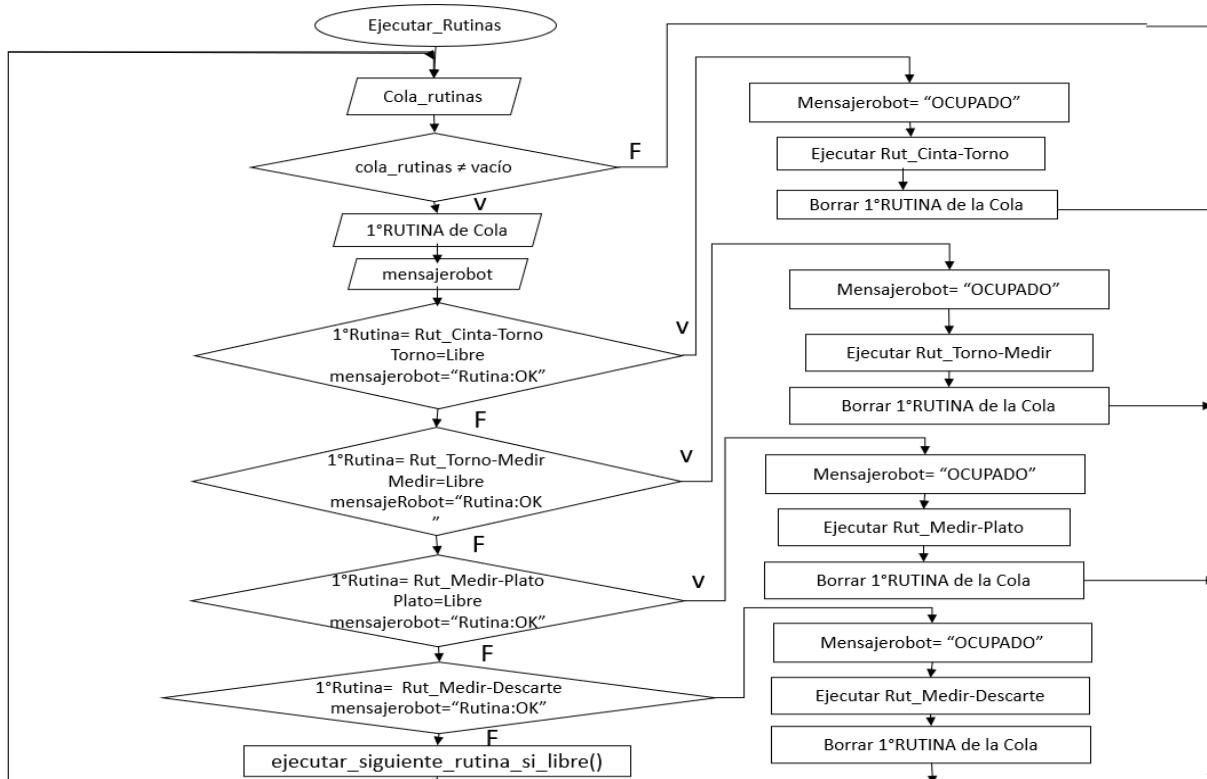


Figura 74. Hilo encargado de ejecutar Rutinas

CAPITULO 4: Implementación de IoT para la Monitorización Remota de la CFFD

En este capítulo se describe la implementación del Internet de las Cosas (IoT) para la monitorización remota de la Celda de Fabricación Flexible Didáctica. Para lograrlo, se empleó el protocolo de comunicación MQTT y la plataforma Adafruit IO, permitiendo el envío y visualización de datos en tiempo real desde los dispositivos utilizados en la celda. A continuación, se detallan los componentes, configuraciones y desarrollo del sistema de monitorización.

4.1 Protocolo MQTT

MQTT (*Message Queueing Telemetry Transport*) es un protocolo de mensajería de publicación-suscripción ligero y popular que es ideal para conectar dispositivos y aplicaciones de Internet de las cosas (IoT) o de máquina a máquina (M2M) a través de Internet. MQTT está diseñado para funcionar de manera eficiente en entornos de bajo ancho de banda o de bajo consumo, lo que lo convierte en una opción ideal para aplicaciones con una gran cantidad de clientes remotos. Se utiliza en una variedad de industrias, incluidas la electrónica de consumo, la automotriz, el transporte, la fabricación y la atención médica [7].



Figura 75. Logo del Protocolo MQTT

Fuente: Tomada de [9]

Algunas de las características importantes del protocolo MQTT, las cuales fundamenta su elección para el sistema, son las siguientes:

- Ligero: La implementación de MQTT en los dispositivos IoT requiere recursos mínimos, por lo que se puede usar incluso en pequeños microcontroladores.
- Fiable: Muchos dispositivos IoT se conectan a través de redes celulares. MQTT es un protocolo adecuado para redes de bajo ancho de banda que requieren mensajes

compactos que utilicen menos datos. Esto hace que MQTT sea más fiable, incluso cuando el ancho de banda de la red es limitado o inestable.

- Escalable: MQTT puede escalar para conectarse con millones de dispositivos IoT.
- Seguro: Los mensajes MQTT pueden cifrarse mediante el estándar de seguridad de la capa de transporte (TLS) y admiten el uso de credenciales para autenticación. Esto hace de MQTT un protocolo de mensajería seguro, ideal para aplicaciones de IoT que manejan información sensible.

Los dispositivos o aplicaciones conectados a través de MQTT se conocen como clientes, estos pueden publicar (enviar) y/o se suscribirse (recibir) a mensajes sobre un tema específico o sobre múltiples temas. Los clientes suscritos reciben todos los mensajes publicados en ese tema, lo que permite un intercambio de datos eficiente y tolerante a fallas entre muchos dispositivos y servicios.

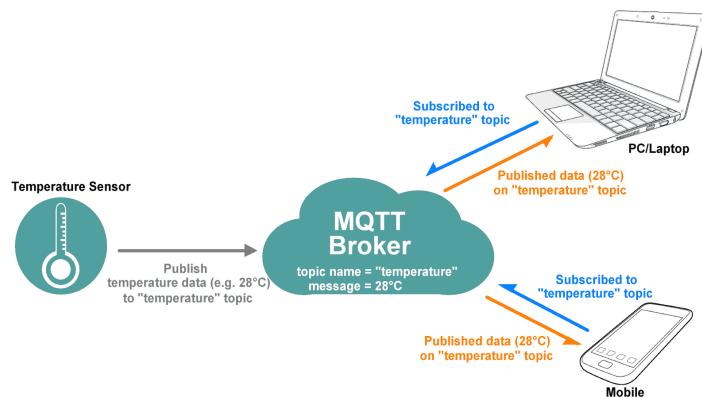


Figura 76. Arquitectura de publicación/suscripción de MQTT

Fuente: tomado de [10]

En el núcleo de la arquitectura MQTT se encuentra el *bróker*, un servidor encargado de gestionar las conexiones de los clientes, registrar los temas a los que están suscritos y dirigir los mensajes a los destinatarios correspondientes. Existen muchos *brokers* MQTT, tanto gratuitos como de pago, que pueden ejecutarse en la nube o de forma local. Algunos *brokers* populares son: *EMQX*, *SHIFTR*, *HIVEMQ*, *MOSQUITTO*, etc. Todos operan bajo el esquema cliente-servidor utilizando el modelo publicador-suscriptor como se mencionó anteriormente.

4.2 Adafruit IO

Adafruit IO es una plataforma en la nube diseñada para recopilar, almacenar y visualizar datos de sensores, dispositivos *IoT* y sistemas embebidos, esta desarrollada y mantenida por *Adafruit Industries*, una empresa de *hardware* de código abierto.



Figura 73. Logo de empresa Adafruit

Fuente: tomado de [11]

Algunas de las características por la cual se optó por usar esta plataforma son las siguientes:

- Es una plataforma gratuita
- Permite controlar y/o leer datos de sensores o dispositivos
- Se integra fácilmente con MQTT
- Muestra los datos en tiempo real y permite compartir esos datos
- Permite crear proyectos electrónicos sin código que se conecten a Internet.

Para poder visualizar los datos de la CFFD en la plataforma *Adafruit IO*, se deben crear los “feed” para cada dato que se quiere visualizar, como lo son: la cantidad de piezas que ingresan, las medidas de las piezas, etc. Los feed son básicamente canales de datos que almacenan y organizan la información que envía la celda. Estos funcionan como contenedores donde cada información que se transmite desde el programa de *Python* de control de la Celda se guarda de manera estructurada para su uso posterior.

Default			
Feed Name	Key	Last value	Recorded
especificaciones_medidas	especificaciones-medidas		4 days from now
estado_depieza	estado-depieza	Aceptado	5 days from now
medida_pieza	medida-pieza	6.89	5 days from now
piezas-ingresadas	piezas-ingresadas	8	5 days from now
piezas_azules	piezas-azules	4	5 days from now
piezas_producidas	piezas-producidas	6	5 days from now
piezas_rechazadas	piezas-rechazadas	1	5 days from now
piezas_rojas	piezas-rojas	4	5 days from now

Figura 74. Feeds creados en la plataforma Adafruit IO

La plataforma *Adafruit IO*, nos permite crear una Dashboard o paneles de trabajo basado en la web, esto permite visualizar los datos que envía la CFFD o interactuar con los dispositivos conectados a él. Sobre el *Dashboard* se pueden colocar los bloques o “widgets” que conforman nuestro proyecto, y que servirán para ver o transferir información entre la plataforma *Adafruit IO* y la Celda.

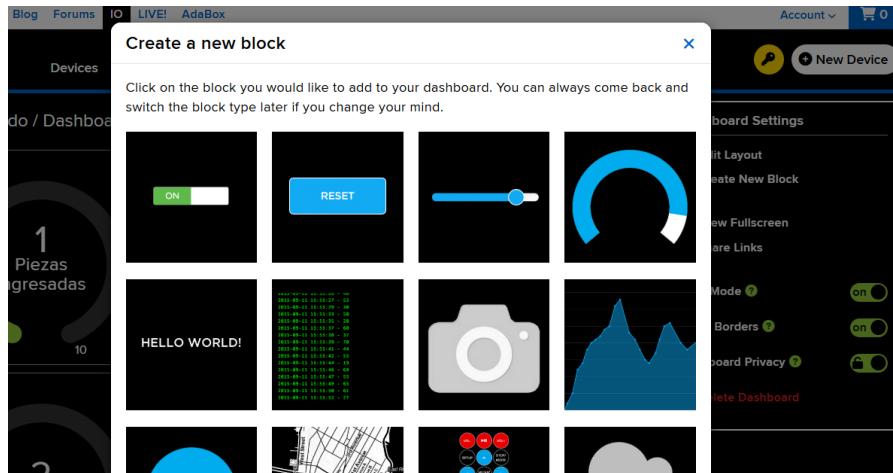


Figura 75. Bloques para crear una Dashboard en Adafruit IO

Es importante señalar que Adafruit IO ofrece una versión de pago y una versión de gratuita, siendo esta ultima la seleccionada para el presente trabajo. La cuenta gratuita en Adrafruit presenta ciertas limitaciones, las cuales son:

- Límite de 10 feeds
- Máximo de 5 Dashboard
- 30 días de almacenamiento de Datos
- Velocidad de transmisión restringida a 30 puntos de datos por minuto.

Sin embargo, estas restricciones no representan una limitación significativa para el alcance de este proyecto, ya que los requisitos de almacenamiento, visualización y frecuencia de transmisión de datos se ajustan dentro de los parámetros permitidos en la versión gratuita.

4.3 Transmisión de Datos de la CFFD a la Nube

Para establecer la comunicación entre la CFFD y la plataforma de *Adafruit IO*, se utilizó la librería “*Adafruit_IO*” en Python y se configuró un cliente MQTT con el nombre de usuario y la clave de acceso proporcionados por la plataforma.

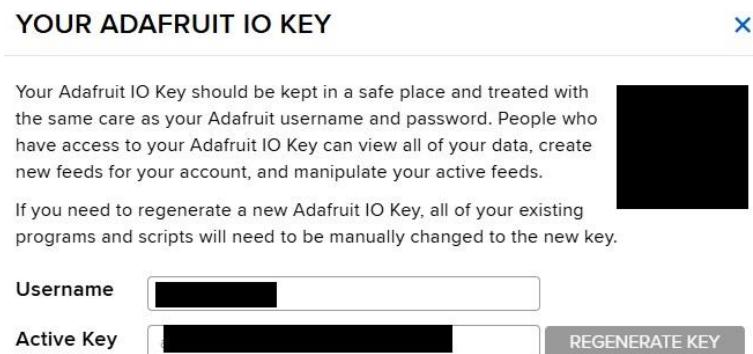


Figura 76. Credencial proporcionada por la Adafruit IO

Al crear una cuenta en *Adafruit IO*, la plataforma asigna al usuario una credencial de acceso compuesta por un *Username* (nombre de usuario) y una *Key* (clave). Esta clave actúa como contraseña para autenticarse y establecer la conexión con el *broker* MQTT de *Adafruit IO*, permitiendo así la transmisión segura de datos hacia la nube. En el programa de control de Python se estableció las siguientes líneas de código para comunicar la Celda con la plataforma y configurar el cliente MQTT con credencial proporcionada por la plataforma:

```
> from Adafruit_IO import MQTTClient  
> ADAFRUIT_IO_USERNAME = "#####"  
> ADAFRUIT_IO_KEY = "*****"
```

```

> def connected(client):
>     print("Conectado a Adafruit IO")
>
> def send_data(feed, value):
>     client.publish(feed, value)
>
> client = MQTTClient(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)
> client.connect()
> client.loop_background()

```

Se definió una función de conexión para verificar que la comunicación con *Adafruit IO* fuera exitosa y se activó el bucle en segundo plano para gestionar la comunicación de manera continua. A medida que se procesan las piezas, los microcontroladores envían datos a la computadora mediante la comunicación serial, estos datos son procesados y enviados a los distintos *feeds* configurados en *Adafruit IO*. Posteriormente en la *Dashboard*, los bloques creados se suscriben a los diferentes *feeds*, que almacenan la información que se quiere mostrar para ver en tiempo real los datos de producción y lograr el monitoreo de la CFFD.

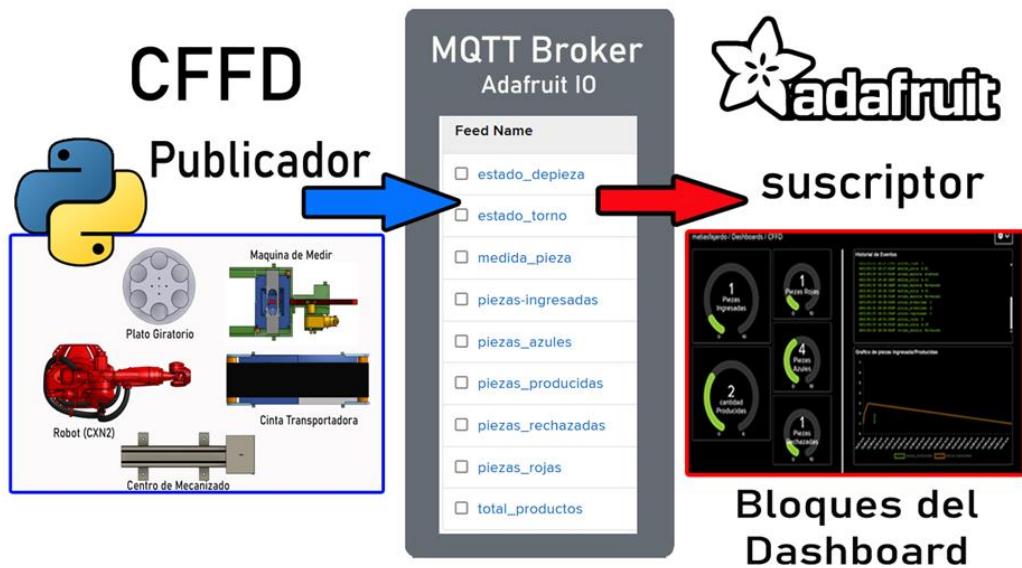


Figura 77. Diagrama de conexión de la CFFD con la nube.

Para enviar los datos de la celda, se definió una función para publicar los datos en los *feeds* correspondientes:

```

> def send_data(feed, value):

```

```
➤     client.publish(feed, value)
```

De esta manera, por ejemplo, si queremos ver los datos de número de piezas que ingresaron a la Celda, se creó un *Feed* llamado: “piezas_ingeradas”, en la cual cada vez que se detecta el ingreso de una pieza al sistema, este publica el dato a dicho *feed*:

```
if "Pieza detectada" in linea:  
    contador_piezas += 1  
    send_data("piezas_ingeradas", contador_piezas)
```

Cada *feed* creado en la plataforma, almacena un dato específico de la celda, en este caso se crearon 9 *feeds*, los cuales son:

- **Piezas_ingeradas**: almacena cuantas piezas ingresaron a la celda.
- **Piezas_rojas**: almacena cuantas piezas rojas ingresaron.
- **Piezas_azules**: almacena cuantas piezas azules ingresaron.
- **Estado_torno**: almacena el estado operativo del torno (Marcha o Detenido)
- **Medida_pieza**: almacena la medida de la pieza medida en la máquina de Medir.
- **Estado_depieza**: almacena la validación de la pieza medida (Aceptado o Rechazado).
- **Piezas_rechazadas**: almacena el número de piezas rechazadas.
- **Piezas_producidas**: almacena cuantos productos se produjeron.
- **Piezas_totales**: almacena cuantas piezas se deben producir

Cada uno de estos *feeds* son asociados a un bloque adecuado en el *dashboard* para poder mostrar de forma rápida y clara los datos de la Celda. A continuación, se muestra el *dashboard* desarrollado para la CFFD: el cual cuenta con visualizadores con contadores de piezas ingresadas, egresada y rechazadas, un historial de eventos y un gráfico de cantidad ingresada/egresada en función del tiempo.

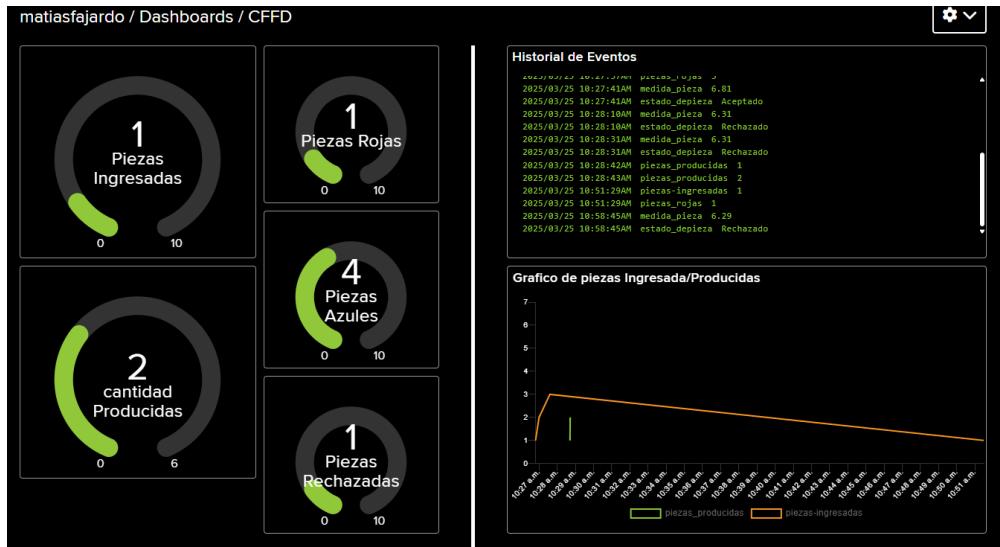


Figura 78. Dashboard creada para la CFFD

Además, *Adafruit IO* nos permite programar acciones como por ejemplo enviar un correo cuando ocurra algún evento específico. Para la CFFD se añadió la función de enviar un correo de Alerta cada vez que se logre producir la cantidad de piezas deseadas de tal forma que cuando se el proceso de la celda finaliza envía un aviso a nuestro correo informando de dicho evento.

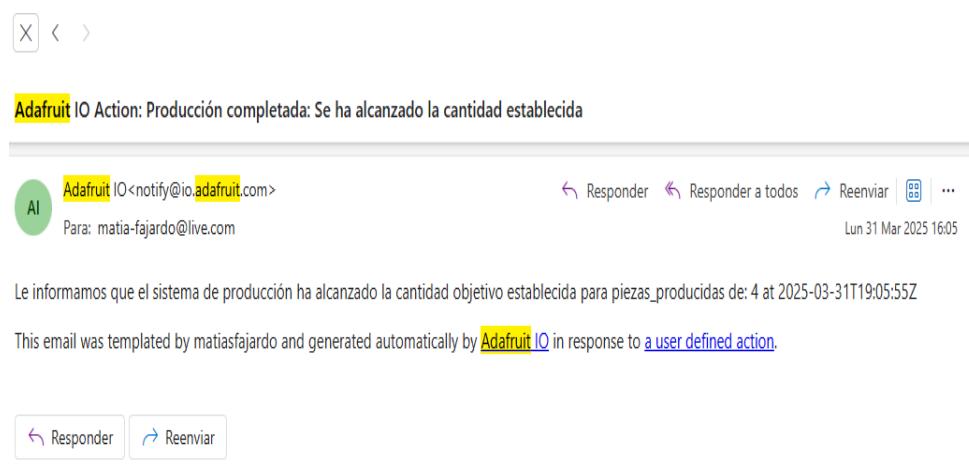


Figura 79. Correo de Advertencia

El desarrollo presentado en este capítulo evidenció la incorporación exitosa de tecnologías IoT para la monitorización remota de la Celda de Fabricación Flexible Didáctica. A través del uso de protocolos como MQTT y la plataforma en la nube como *Adafruit IO*, se logró establecer un entorno de supervisión remota funcional, accesible y

adaptable. Esta integración permite visualizar en tiempo real el estado de las máquinas y recibir alertas desde cualquier ubicación con acceso a internet. Además, se demostró que es posible construir soluciones de Industria 4.0 con recursos de bajo costo, manteniendo la escalabilidad y el enfoque educativo. En conjunto, esta implementación sienta las bases para futuras mejoras orientadas al control remoto completo y a la interoperabilidad con otros sistemas inteligentes.

CAPITULO 5: Recursos Utilizados y Evaluación de Factibilidad del Proyecto

En este capítulo se describen los recursos y materiales utilizados, junto con una evaluación de la factibilidad de implementación de la CFFD, teniendo en cuenta aspectos técnicos, económicos y organizativos que formaron parte del desarrollo del proyecto. Los valores expresados en dólares (USD) han sido tomados en referencia al valor del dólar oficial en Argentina, correspondiente a 1200 ARS el día 30/06/2025.

Los recursos físicos y tecnológicos fundamentales para el funcionamiento de la CFFD, los cuales fueron empleados tanto en la etapa de desarrollo como en la operación de la celda. Se muestran en el siguiente cuadro:

Tabla Nº2. Recursos Físicos empleados en el desarrollo de la CFFD

Recurso Físico	Descripción/uso
Notebook (Personal)	Control central de la celda, programación, Diseño CAD y documentación.
Impresoras 3D (personal/LABME)	Fabricación de componentes mecánicos en PLA.
Fuente de alimentación.	Alimentación de los módulos electrónicos de la celda.
Multímetro digital	Verificación de conexiones eléctricas y pruebas de continuidad.

Los softwares utilizados en el desarrollo de la Celda son de uso libre y gratuito. Sin embargo, muchas de las funcionalidades interesantes para proyectos interconectados, como son la visualización de datos y el almacenamiento en la nube más complejos, requieren de un mayor desarrollo o de la incorporación de servicios pagos.

Tabla Nº3. Softwares usados en el desarrollo de la CFFD

Item	Descripción	Costo
Arduino IDE	Programación de los controladores Arduinos.	Gratis
Onshape	Diseño de las estaciones de trabajo en 3D.	Licencia gratuita
Python	Entorno de Programación del Controlador de la CFFD.	Gratis
Client MQTT	Broker MQTT de Adafruit IO Gratis en la Nube	Gratis
Adafruit IO	Plataforma en la Nube donde se aloja y visualiza los datos	Licencia gratuita

Los costos de fabricación correspondientes a la cantidad de material PLA y el tiempo de impresión de las distintas estaciones de trabajo de la CFFD y carcasas de los controladores Arduino se muestran en el siguiente cuadro:

Tabla Nº4. Cantidad de PLA utilizado en las estaciones de Trabajo.

Maquina	Filamentos PLA (g)	Hs de Impresión	Costo (\$ ARS)	Costo (USD)
Cinta transportadora	317 gramos	36hs	\$18.392	15,27 USD
Torno	80 gramos	8hs	\$4.633	3,85 USD
Máquina de Medir	154 gramos	20hs	\$8.953	7,43 USD
Plato Giratorio	210 gramos	24hs	\$12.185	10,12 USD
Uniones Tipo Eslabón	144 gramos	16hs	\$8.352	6,93 USD
Carcasa Arduino Mega	82 gramos	8hs	\$4.748	3,94 USD
Carcasa Arduino Nano	69 gramos	6hs	\$3.990	3,31 USD
Carcaza Fuente	41 gramos	5hs	\$2.381	1,98 USD
Total	1,097 Kg	123 hs	\$63.634	52,83 USD

Para el funcionamiento de los distintos módulos de la celda, se utilizaron diversos componentes electrónicos, seleccionados por su disponibilidad y bajo costo. A continuación, se detallan los principales materiales electrónicos empleados en el desarrollo del sistema:

Tabla Nº5. Elementos eléctricos y Electrónicos utilizados.

Componente	Cantidad	Costo (\$ ARS)	Costo (USD)
Arduino Mega 2560	1	\$27.500	23 USD
Arduino Nano	1	\$11.000	9,20 USD
Sensor ultrasónico HC-SR04	3	\$21.000	17,5 USD
Sensor de color TCS3200	1	\$12.600	10,54 USD
Sensor infrarrojo TCRT5000	3	\$15.000	12,5 USD
Motor paso a paso Nema 17	1	\$21.700	18,15 USD
Motor DC 5V	3	\$18.000	15 USD
driver A4988	1	\$5.000	4,18 USD
Driver L298N	1	\$8.000	6,7 USD
Display de 7 segmentos	1	\$4.000	3,34 USD
Finales de Carrera	2	\$8.000	6,7 USD
Cable Canal 14X7 mm	1	\$2.000	1,67 USD
Curva plana	6	\$1.200	1 USD
TOTAL		\$155.000	129,48 USD

En base al análisis de los recursos utilizados y las tareas realizadas, se concluye que el desarrollo de la Celda de Fabricación Flexible Didáctica (CFFD) fue técnica y económicamente factible, considerando su objetivo principal: brindar una herramienta funcional y accesible para la enseñanza de automatización e Industria 4.0. La utilización de tecnologías de bajo costo, como plataformas Arduino, sensores estándar y componentes impresos en 3D, permitió alcanzar los objetivos del proyecto sin requerir inversiones elevadas. Además, el empleo de software libre y herramientas ya disponibles en el laboratorio redujo significativamente los costos asociados.

CAPITULO 6: Conclusiones

El presente trabajo logró con éxito el diseño, construcción e implementación de una Celda de Fabricación Flexible Didáctica (CFFD) orientada a la enseñanza de tecnologías de Industria 4.0 en el ámbito universitario. La CFFD desarrollada integra mecánica, electrónica, informática y control, permitiendo emular un entorno de producción automatizado de forma accesible y segura para los estudiantes.

Se cumplió con los objetivos planteados, desde la elaboración del diseño CAD y la impresión 3D de componentes, hasta la programación de los sistemas de control y la incorporación de tecnologías emergentes como la comunicación en la nube mediante MQTT y la plataforma Adafruit IO. La implementación de un sistema SCADA en Python, junto con la gestión eficiente de tareas del robot mediante una estructura FIFO, permitió una coordinación precisa de los procesos, asegurando un flujo de trabajo ordenado, sin colisiones ni interferencias entre estaciones.

Las pruebas realizadas demostraron la operatividad confiable del sistema, validando su utilidad como herramienta didáctica. La CFFD no solo permitirá ejecutar tareas de manufactura flexible de manera automatizada, sino que también servirá como una plataforma educativa para que los estudiantes experimenten conceptos clave de la mecatrónica moderna: sensado, control, programación de robots, monitoreo en tiempo real, la lógica de eventos discretos y la implementación de industria 4.0.

Este proyecto no solo representa una solución funcional para la enseñanza, sino también una base sólida para futuras ampliaciones y desarrollos, tales como la integración de inteligencia y visión artificial. La CFFD se posiciona, así como un recurso valioso para la formación de ingenieros mecatrónicos capacitados en los desafíos de la industria digital.

Es importante señalar que en caso de que una estación se detenga inesperadamente, el sistema actual no cuenta con una recuperación automática. Este comportamiento constituye una limitación del sistema, que podría abordarse en trabajos futuros mediante la incorporación de lógica de detección de errores y reinicio automático, o sistemas *watchdog*.

Glosario

Arduino	Plataforma de hardware libre basada en una placa con microcontrolador y un entorno de desarrollo para la creación de proyectos electrónicos.
CAD	El diseño asistido por ordenador (CAD) consiste en el uso de programas de ordenador para crear, modificar, analizar y documentar representaciones gráficas bidimensionales o tridimensionales (2D o 3D) de objetos físicos como una alternativa a los borradores manuales y a los prototipos de producto.
Dashboard	También conocido como cuadro de mando o tablero de control, es una interfaz visual que presenta información clave de manera resumida, permitiendo una rápida comprensión de datos importantes para un negocio o proceso.
FIFO	método de gestión de tareas o datos donde el primer en ingresar es el primero en salir
IOT	Internet de las Cosas, se refiere a la red de objetos físicos conectados a Internet, como dispositivos, sensores y software, que permiten la recopilación, transmisión y análisis de datos. En esencia, es la capacidad de conectar y comunicar objetos cotidianos a través de Internet.
Layout	Se refiere a la disposición o distribución física de elementos dentro de un espacio
PWM	PWM son las siglas de “Pulse Width Modulation”, que en español significa modulación por ancho de pulsos. Es una técnica que se utiliza para controlar la cantidad de energía que se suministra a una carga eléctrica.
TCP/IP	Son las siglas de Protocolo de Control de Transmisión/Protocolo de Internet. Es un conjunto de protocolos que permiten la comunicación entre dispositivos de red.

Referencias Bibliográficas

- [1] J. M. Ortiz, *Fundamentos de ingeniería industrial: Sistemas de manufactura*. Ciudad de México: Colofón; Universidad Autónoma de Tamaulipas, 2019.
- [2] M. P. Groover, *Fundamentos de manufactura moderna: Materiales, procesos y sistemas*, 3^a ed. Madrid, España: Pearson Educación, 2007.
- [3] S. Kalpakjian y S. R. Schmid, *Manufactura, ingeniería y tecnología*, 7^a ed. México: Pearson Education, 2014.
- [4] L. Joyanes Aguilar, *Industria 4.0: La Cuarta Revolución Industrial*, 1^a ed. Madrid: Alfaomega, 2017.
- [5] L. Joyanes Aguilar, *Estructuras de Datos en Java*. Madrid: McGraw-Hill, 2003.
- [6] FCI, *Data Sheet: 2N2222A NPN General Purpose Transistor*, Alldatasheet. [En línea]. Disponible: <https://www.alldatasheet.com/datasheet-pdf/pdf/38963/FCI/2N2222A.html>. [Accedido: 03-Abr-2024].
- [7] Código Electrónica, "Arduino - Display 7 segmentos ánodo común", Código Electrónica, [En línea]. Disponible: <http://www.codigoelectronica.com/blog/arduino-display-7-segmentos-anodo-comun>. [Accedido: 02-Abr-2024].
- [8] uElectronics, "Shield para Arduino Nano Expansion I/O," *uElectronics.com*. [En línea]. Disponible: <https://uelectronics.com/producto/shield-para-arduino-nano-expansion-i-o/>. [Accedido: 06-May-2024].. [Accedido: 02-Abr-2024].
- [8] uElectronics, "Shield para Arduino Nano Expansion I/O," *uElectronics.com*. [En línea]. Disponible: <https://uelectronics.com/producto/shield-para-arduino-nano-expansion-i-o/>. [Accedido: 06-May-2024].. [Accedido: 02-Abr-2024].
- [9] F5 Networks, "¿Qué es MQTT?", *F5 Glossary*, 2024. [Online]. Available: https://www.f5.com/es_es/glossary/mqtt. [Accedido: 15-Mar-2024].
- [10] MQTT, *MQTT Specification*, 2024. [Online]. Available: <https://mqtt.org/mqtt-specification/>. [Accedido: 13-Mar-2025].
- [11] Electrónica Embajadores, "Logo de Adafruit", Electrónica Embajadores. [En línea]. Disponible: <https://www.electronicaembajadores.com/datos/fotos/articulos/medianas/marcas/adafuit.png>. [Accedido: 25-Mar-2025].

Anexo/s

- **Anexo A:** Códigos del Controlador Mega:

```
///////////  
  
#include <ArduinoQueue.h>  
ArduinoQueue<String> colaColores(5); // Cola de Strings  
  
/////////// Cinta transportadora ///////////  
int IN3 = 2;  
int IN4 = 3;  
int ENB = 4;  
  
const int Trigger1 = 5;  
const int Echo1 = 6;  
long d1;  
  
const int Trigger2 = 7;  
const int Echo2 = 8;  
long d2; // distancia en centímetros  
  
unsigned long tiempoDeteccion = 0;  
bool esperandoDetencion = false;  
  
/////////// Sensor de Color ///////////  
#define S0 34  
#define S1 33  
#define S2 32  
#define S3 31  
#define salidaTCS 30  
  
bool tiempocolor = false;  
bool estadocolor = false;  
unsigned long previousMillisColor = 0;  
const long intervalColor = 200;  
  
int rojo = 0;  
int verde = 0;  
int azul = 0;
```

```

////////// TORNO ///////////
int sensor = 9;
int estado;

unsigned long tiemptorno1;
unsigned long tiemptorno2;

bool piezacontada = false;

const int segmentPins[7] = {22, 23, 24, 25, 26, 27, 28};
unsigned long tiempo;
unsigned long tiempo2 = 0;
int digito = 9;

enum ColorPieza { NINGUNO, ROJO, AZUL };
ColorPieza colorPiezaActual = NINGUNO;
enum EstadoColor { ESTADO_ROJO, ESTADO_VERDE, ESTADO_AZUL, ESTADO_ESPERAR };
EstadoColor estadoActualColor = ESTADO_ROJO;

int tiempoMecanizadoRojo = digito;
int tiempoMecanizadoAzul = digito;

const int digits[10][7] = {
    {1, 1, 1, 1, 1, 1, 0}, // 0
    {0, 1, 1, 0, 0, 0, 0}, // 1
    {1, 1, 0, 1, 1, 0, 1}, // 2
    {1, 1, 1, 1, 0, 0, 1}, // 3
    {0, 1, 1, 0, 0, 1, 1}, // 4
    {1, 0, 1, 1, 0, 1, 1}, // 5
    {1, 0, 1, 1, 1, 1, 1}, // 6
    {1, 1, 1, 0, 0, 0, 0}, // 7
    {1, 1, 1, 1, 1, 1, 1}, // 8
    {1, 1, 1, 1, 0, 1, 1} // 9
};

int motor = 10;
int LED = 11;

unsigned long previousMillisTorno = 0;
unsigned long intervalTorno = 1000;
bool tornoActivo = false;
bool cintaActiva = false;
bool contadorActivo = false;

void setup() {

```

```

Serial.begin(9600);

// Configuración de los pines del motor
pinMode(ENB, OUTPUT);
pinMode(IN3, OUTPUT);
pinMode(IN4, OUTPUT);

// Configuramos los pines del sensor 1
pinMode(Trigger1, OUTPUT); // Pin como salida
pinMode(Echo1, INPUT); // Pin como entrada
digitalWrite(Trigger1, LOW); // Inicializamos el pin con 0

// Configuramos los pines del sensor 2
pinMode(Trigger2, OUTPUT); // Pin como salida
pinMode(Echo2, INPUT); // Pin como entrada
digitalWrite(Trigger2, LOW); // Inicializamos el pin con 0

// Configuración del torno
pinMode(sensor, INPUT);
pinMode(motor, OUTPUT);
pinMode(LED, OUTPUT);

// Configuración de los pines del display de 7 segmentos
for (int i = 0; i < 7; i++) {
    pinMode(segmentPins[i], OUTPUT);
}
digitalWrite(LED, HIGH);

// Configuración del sensor de color
pinMode(S0, OUTPUT);
pinMode(S1, OUTPUT);
pinMode(S2, OUTPUT);
pinMode(S3, OUTPUT);
pinMode(salidaTCS, INPUT);
digitalWrite(S0, HIGH);
digitalWrite(S1, LOW);
}

void loop() {

    recibirTiemposMecanizado();

    gestionarCintaTransportadora();

    gestionarTorno();
}

```

```

// Lógica para leer el color
if (estadocolor) {
    leerColor();
}

}

/////////// FUNCIO para recibir el tiempo de Mecanizado ///////////
void recibirTiemposMecanizado() {
    if (Serial.available() > 0) {
        String mensaje = Serial.readStringUntil('\n');
        mensaje.trim();

        if (mensaje.startsWith("TIEMPO_ROJO:")) {
            tiempoMecanizadoRojo = mensaje.substring(12).toInt();
            Serial.print("Tiempo Rojo actualizado: ");
            Serial.println(tiempoMecanizadoRojo);
        } else if (mensaje.startsWith("TIEMPO_AZUL:")) {
            tiempoMecanizadoAzul = mensaje.substring(12).toInt();
            Serial.print("Tiempo Azul actualizado: ");
            Serial.println(tiempoMecanizadoAzul);
        }
    }
}

// Función para controlar la cinta transportadora
void gestionarCintaTransportadora() {
    d1 = medirDistancia(Trigger1, Echo1);
    d2 = medirDistancia(Trigger2, Echo2);
    if (d2 <= 6 && !esperandoDetencion) {
        tiempoDeteccion = millis();
        esperandoDetencion = true;
    }
    if (esperandoDetencion) {
        if (millis() - tiempoDeteccion >= 200) {
            digitalWrite(ENB, 0);
            digitalWrite(IN3, LOW);
            digitalWrite(IN4, LOW);
            cintaActiva = false;
            esperandoDetencion = false;

            if (!piezacontada) {
                Serial.println("Pieza detectada");
                piezacontada = true;
            }
        }
    }
}

```

```

        estadocolor = true;
        estadoActualColor = ESTADO_ROJO;
        previousMillisColor = millis();
    }
}
} else if (d1 <= 6 && !cintaActiva) {
    cintaActiva = true;
    piezacontada = false;
    digitalWrite(IN3, HIGH);
    digitalWrite(IN4, LOW);
    analogWrite(ENB, 190);
    Serial.println("cinta activada");
} else {
    Serial.println("Pieza retirada");
    estadocolor = false;
}
}

// Función para controlar el torno
void gestionarTorno() {
    estado = digitalRead(sensor);

    if (estado == HIGH && !tornoActivo) {
        tiempotorno1 = millis();
        Serial.println("material en el torno");
        if (tiempotorno2 == 0) {
            tiempotorno2 = tiempotorno1;
            Serial.println("material en el torno");
        }
        if (tiempotorno1 - tiempotorno2 >= 2000 && estado == HIGH) {
            tornoActivo = true;
            contadorActivo = true;
            String colorPieza = colaColores.dequeue();
            if (colorPieza == "ROJO") {
                digito = tiempoMecanizadoRojo;
                displayDigit(digito);
            } else if (colorPieza == "AZUL") {
                digito = tiempoMecanizadoAzul;
                displayDigit(digito);
            }
            previousMillisTorno = millis();
            digitalWrite(motor, HIGH);
            digitalWrite(LED, LOW);
            Serial.println("Pieza colocada en el torno");
        }
    }
}

```

```

    }

    if (tornoActivo && contadorActivo) {
        unsigned long currentMillis = millis();
        if (currentMillis - previousMillisTorno >= intervalTorno) {
            previousMillisTorno = currentMillis;
            displayDigit(dígito);
            dígito--;
        }

        if (dígito < 0) {
            dígito = 0;
            contadorActivo = false;
            digitalWrite(motor, LOW);
            digitalWrite(LED, HIGH);
            Serial.println("Torno terminado");
        }
    }
}

// Si la pieza se retira mientras el torno está funcionando, reiniciar todo
if (estado == LOW && tornoActivo) {
    tornoActivo = false;
    contadorActivo = false;
    digitalWrite(motor, LOW);
    digitalWrite(LED, HIGH);
    dígito = 0;
    displayDigit(dígito);
    Serial.println("Pieza fuera del torno");
    tiempotorno2 = 0;
}
if (estado == LOW && !tornoActivo) {
    Serial.println("No hay pieza en el torno");
    dígito = 0;
    displayDigit(dígito);
}
}

// Función para medir la distancia usando el sensor de ultrasonido
long medirDistancia(int triggerPin, int echoPin) {
    digitalWrite(triggerPin, LOW);
    delayMicroseconds(2);
    digitalWrite(triggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(triggerPin, LOW);

    long t = pulseIn(echoPin, HIGH);
}

```

```

long d = t / 59;
return d;
}

// Función para mostrar un dígito en el display de siete segmentos
void displayDigit(int digit) {
    for (int i = 0; i < 7; i++) {
        digitalWrite(segmentPins[i], digits[digit][i]);
    }
}

// Función para leer el color usando el sensor de color
void leerColor() {
    unsigned long currentMillis = millis();

    switch (estadoActualColor) {
        case ESTADO_ROJO:
            if (currentMillis - previousMillisColor >= intervalColor) {
                digitalWrite(S2, LOW);
                digitalWrite(S3, LOW);
                rojo = pulseIn(salidaTCS, LOW);
                estadoActualColor = ESTADO_VERDE;
                previousMillisColor = currentMillis;
            }
            break;

        case ESTADO_VERDE:
            if (currentMillis - previousMillisColor >= intervalColor) {
                digitalWrite(S2, HIGH);
                digitalWrite(S3, HIGH);
                verde = pulseIn(salidaTCS, LOW);
                estadoActualColor = ESTADO_AZUL;
                previousMillisColor = currentMillis;
            }
            break;

        case ESTADO_AZUL:
            if (currentMillis - previousMillisColor >= intervalColor) {
                digitalWrite(S2, LOW);
                digitalWrite(S3, HIGH);
                azul = pulseIn(salidaTCS, LOW);
                estadoActualColor = ESTADO_ESPERAR;
                previousMillisColor = currentMillis;
            }
    }
}

```

```

        if (rojo < 300 && verde > 250 && azul > 250){
            Serial.println("ROJO");
            colorPiezaActual = ROJO;
            colaColores.enqueue("ROJO");

        } else{
            Serial.println("AZUL");
            colorPiezaActual = AZUL;
            colaColores.enqueue("AZUL");
        }
        estadocolor = false;
    }
    break;

case ESTADO_ESPERAR:
    break;
}
///////////

```

- **Anexo B:** Códigos del Controlador Nano:

```

///////////

int detector = 3;
int det;
unsigned long tiempoDetectado = 0;
unsigned long tiempoInicio = 0;
bool objetoDetectado = false;
bool procesoIniciado = false;
bool regresoFinalizado = false;
bool medicionenviada = false;

/////////// Motor de la máquina de medir ///////////
int IN1 = 8;
int IN2 = 9;
int ENA = 10;

/////////// Final de Carrera ///////////
int Final2 = 4;
int Final1 = 5;
int carrera2;
int carrera1;
bool medirObjeto = false;

```

```

/////////// Pines del sensor de ultrasonido ///////////////
const int Trigger1 = 11;
const int Echo1 = 12;
long t1;
long d1;

unsigned long esperaMedicion = 0;
unsigned long tiempoMedicion = 2000;

int pro = 1;

/////////// Plato Giratorio /////////////
#define STEP 2
#define DIR 13
#define STEP_INTERVAL 10
#define SENSOR_PIN 6
bool home;

bool sensorDetectado = false;
unsigned long ultimoPasoTiempo = 0;
unsigned long tiempoFinGiro = 0;
int pasosRealizados = 0;
bool platoGirando = false;

int valor;
int contador = 0;
int contadorMaximo = 0;
int procesoPlato = 1;
unsigned long tiempoesperaPlato = 0;
bool mensajeEnviadoPlato = false;

void setup() {
  Serial.begin(9600);

  // Configuración de pines para la máquina de medir
  pinMode(detector, INPUT);
  pinMode(ENA, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);

  pinMode(Final2, INPUT);
  pinMode(Final1, INPUT_PULLUP);

```

```

pinMode(Trigger1, OUTPUT);
pinMode(Echo1, INPUT);
digitalWrite(Trigger1, LOW);
home = false;

// Configuración de pines para el plato giratorio
pinMode(SENSOR_PIN, INPUT);
pinMode(STEP, OUTPUT);
pinMode(DIR, OUTPUT);
digitalWrite(DIR, HIGH);
}

void loop() {
    recibirCantidadDePiezas();
    maquinaDeMedir();
    platoGiratorio();
}

void recibirCantidadDePiezas() {
    if (Serial.available() > 0) {
        int cantidadRecibida = Serial.parseInt();
        if (cantidadRecibida >= 1 && cantidadRecibida <= 4) {
            contadorMaximo = cantidadRecibida;
            contador = 0;
            procesoPlato = 1;
            Serial.print("Cantidad de piezas a producir actualizada: ");
            Serial.println(contadorMaximo);
        }
    }
}

// Función para el proceso de la máquina de medir
void maquinaDeMedir() {
    carrera2 = digitalRead(Final2);
    carrera1 = digitalRead(Final1);
    det = digitalRead(detector);

    if (det == HIGH && !procesoIniciado && !objetoDetectado && pro == 1) {
        objetoDetectado = true;
        tiempoDetectado = millis();
        Serial.println("prodetectado");
    }

    if (objetoDetectado==true && millis() - tiempoDetectado >= 4000 &&
!procesoIniciado && pro == 1){

```

```

Serial.println("ingresar producto");
digitalWrite(IN1, HIGH);
digitalWrite(IN2, LOW);
analogWrite(ENA, 180);
procesoIniciado = true;
}

if (procesoIniciado== true && carrera1 == LOW && !medirObjeto && pro == 1) {
  Serial.println("detener motor");
  analogWrite(ENA, 0);
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  medirObjeto = true;
  esperaMedicion = millis();
}

if (medirObjeto== true && millis() - esperaMedicion >= tiempoMedicion && pro ==
1) {
  float distancia = medirDistancia(Trigger1, Echo1);
  float largo = (10.82-distancia);

  Serial.print("Distancia: ");
  Serial.print(distancia);
  Serial.println(" cm");

  Serial.print("Largo: ");
  Serial.print(largo);
  Serial.println(" cm");

  if (medicionenviada == false) {
    Serial.print("Largo del Objeto: ");
    Serial.print(largo);
    Serial.println(" cm");
    medicionenviada = true;
  }

  medirObjeto = false;

  Serial.println("Iniciando el regreso a la posición original...");
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  analogWrite(ENA, 170);
}

if (carrera2 == HIGH && procesoIniciado && !medirObjeto && pro == 1) {

```

```

Serial.println("esperar producto");
digitalWrite(IN1, HIGH);
digitalWrite(IN2, LOW);
analogWrite(ENA, 150);
digitalWrite(IN1, HIGH);
digitalWrite(IN2, LOW);
analogWrite(ENA, 150);
analogWrite(ENA, 0);
digitalWrite(IN1, LOW);
digitalWrite(IN2, LOW);
procesoIniciado = false;
objetoDetectado = false;
medicionenviada = false;
regresoFinalizado = true;
pro = 0;
}

if (regresoFinalizado && det == LOW) {
  Serial.println("producto retirado");
  regresoFinalizado = false;
  pro = 1;
}
if (det == LOW && !procesoIniciado && !objetoDetectado && pro == 1) {
  Serial.println("producto no detectado");
}
}

// Función para el proceso del plato giratorio
void platoGiratorio(){
  sensorDetectado = digitalRead(SENSOR_PIN);
  if (!home) {
    Serial.println("no poner nada");
    if (sensorDetectado == HIGH) {
      digitalWrite(STEP, HIGH);
      delayMicroseconds(8000);
      digitalWrite(STEP, LOW);
      delayMicroseconds(8000);
    } else {
      digitalWrite(STEP, HIGH);
      delayMicroseconds(8000);
      digitalWrite(STEP, LOW);
      delayMicroseconds(8000);
      digitalWrite(STEP, HIGH);
      delayMicroseconds(8000);
      digitalWrite(STEP, LOW);
    }
  }
}

```

```

        home = true;
        Serial.println("Plato en posicion home");
    }
}

if (home == true){
    sensorDetectado = digitalRead(SENSOR_PIN);

    if (sensorDetectado == HIGH && contador < contadorMaximo && procesoPlato ==
1) {
        if (!mensajeEnviadoPlato) {
            Serial.println("objeto detectado en el plato giratorio");
            mensajeEnviadoPlato = true;
            tiempoesperaPlato = millis();
        }

        if (millis() - tiempoesperaPlato >= 3000 && !platoGirando) {
            Serial.println("Iniciando giro del plato");
            digitalWrite(DIR, HIGH);
            pasosRealizados = 0;
            platoGirando = true
        }
    } else if (sensorDetectado == LOW){
        Serial.println("No se detecta objeto en el plato giratorio");
        procesoPlato = 1;
        mensajeEnviadoPlato = false;
    }
}

if (platoGirando==true) {
    if (millis() - ultimoPasoTiempo >= STEP_INTERVAL) {
        ultimoPasoTiempo = millis();
        digitalWrite(STEP, HIGH);
        delayMicroseconds(1);
        digitalWrite(STEP, LOW);

        pasosRealizados++;

        if (pasosRealizados >= 50) {
            platoGirando = false;
            contador++;
            Serial.println("Giro completado para una pieza.");
        }
    }
}
if (contador >= contadorMaximo) {

```

```

    procesoPlato = 0;
    Serial.println("cantidad maxima de piezas alcanzada");
}
}

float medirDistancia(int triggerPin, int echoPin) {
    digitalWrite(triggerPin, LOW);
    delayMicroseconds(2);
    digitalWrite(triggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(triggerPin, LOW);

    long duracion = pulseIn(echoPin, HIGH);
    float distancia = duracion * 0.0343 / 2;

    return distancia;
}
/////////////////////////////////////////////////////////////////

```

- **ANEXO C:** Código del Controlador en Python

```

/////////////////////////////////////////////////////////////////

import threading
import time
import serial
import socket
import tkinter as tk
from collections import deque
from tkinter import ttk
from tkinter import font
from PIL import Image, ImageTk, ImageDraw
from tkinter import messagebox
from Adafruit_IO import MQTTClient

#Configuración de Socket
sock=socket.socket(socket.AF_INET,socket.SOCK_STREAM)

mensajerobot = "RUTINA:OK"

```

```

server_addr=('10.10.10.10',40001)
print(sock.connect(server_addr))

ADAFRUIT_IO_USERNAME = "ID-DEL-USUARIO"
ADAFRUIT_IO_KEY = "*****"

def connected(client):
    print("Conectado a Adafruit IO")

def send_data(feed, value):
    client.publish(feed, value)

client = MQTTClient(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

client.connect()
client.loop_background()

# Reiniciar contadores en la nube al iniciar el programa

send_data("Piezas_producidas", 0)
send_data("piezas_rechazadas", 0)
send_data("piezas_rojas", 0)
send_data("piezas_azules", 0)
send_data("medida_pieza", 0)
send_data("estado_depieza", "Reiniciado")

# Configuración de la comunicación serial para ambos Arduinos
Nano = serial.Serial("COM12", 9600, timeout=0.1)
Mega = serial.Serial("COM8", 9600, timeout=0.1)

# Variables de contador de piezas y otras mediciones

```

```
contpiezasmedidas = 0  
contpiezasproducidas = 0  
contador_piezas = 0  
contador_torno = 0  
medida_pieza = 0  
piezas_a_producir = 1  
contpiezasrojas = 0  
contpiezasazul = 0  
contpiezasrechazadas=0  
tiempo_de_espera=0
```

```
# Definir una cola FIFO para las piezas  
cola_rutinas = deque()
```

```
# Estados de las máquinas  
estado_maquina_medicion = "libre"  
estado_plato_giratorio = "libre"  
rutina_en_proceso = None
```

```
# Cargar la imagen
```

```
imagen_base = Image.open(r"C:\Users\matia\Documents\Tesis\Layout2.jpg")  
imagen_modificada = imagen_base.resize((1250, 720))
```

```
# Estados iniciales de color
```

```
color_medir = "yellow"  
color_plato_giratorio = "yellow"  
color_cinta = "yellow"  
color_torno = "yellow"
```

```
# Función para modificar el color de un círculo en la interfaz
```

```
def modificar_circulo(circulo, color):
```

```

global imagen_modificada
draw = ImageDraw.Draw(imagen_modificada)

# Cambiar colores de los elementos según su ubicación en la imagen

coordenadas = {

    'Maquina_de_medir': (780, 145, 810, 175),

    'Plato_giratorio': (447, 205, 477, 235),

    'cinta': (745, 390, 775, 420),

    'torno': (530, 610, 560, 640),

}

if circulo in coordenadas:

    draw.ellipse(coordenadas[circulo], fill=color)

actualizar_imagen()

# Función para actualizar la imagen en la interfaz

def actualizar_imagen():

    img_tk = ImageTk.PhotoImage(imagen_modificada)

    imagen_label.config(image=img_tk)

    imagen_label.image = img_tk

# Función para leer lo que envia el robot

def escuchar_robot():

    global mensajerobot

```

```

while True:

    try:

        data = sock.recv(2048)

        if data:

            mensajerobot = data.decode().strip()

            print(f"Mensaje del robot: {mensajerobot}")

    except Exception as e:

        print(f"Error al recibir datos del robot: {e}")

        break


# Loop para que se ejecute constamemte ejecución rutinas

def loop_ejecucion_rutinas():

    ejecutar_siguiente_rutina_si_libre()

    sc.after(200, loop_ejecucion_rutinas) # Llama a esta función cada 200 ms

# Función para agregar una rutina a la cola

def agregar_rutina_a_cola(rutina):

    cola_rutinas.append(rutina)

    print("Cola actual:", list(cola_rutinas))

# Función para ejecutar la siguiente rutina si la máquina está libre

def ejecutar_siguiente_rutina_si_libre():

    global mensajerobot

```

```
for rutina in list(cola_rutinas):

    if rutina == "RUTINA:RUT_CINTA_TORNO" and color_torno == "yellow" and
mensajerobot=="RUTINA:OK":

        mensajerobot="OCUPADO"

        sock.sendall(rutina.encode())

        cola_rutinas.remove(rutina)

        break

    elif rutina == "RUTINA:RUT_TORNO_MEDIR" and color_medir == "yellow" and
mensajerobot=="RUTINA:OK":

        mensajerobot="OCUPADO"

        sock.sendall(rutina.encode())

        cola_rutinas.remove(rutina)

        break

    elif rutina == "RUTINA:RUT_MEDIR_PLATO" and color_plato_giratorio == "yellow" and
mensajerobot=="RUTINA:OK":

        mensajerobot="OCUPADO"

        sock.sendall(rutina.encode())

        cola_rutinas.remove(rutina)

        break
```

```

elif rutina == "RUTINA:RUT_MEDIR_DESCARTE" and mensajerobot=="RUTINA:OK":
    mensajerobot="OCUPADO"
    sock.sendall(rutina.encode())
    cola_rutinas.remove(rutina)
    break

# Función para verificar si la medida cumple con las especificaciones

def verificar_medida():

    global medida_pieza,contpiezasrechazadas

    try:

        medida_ingresada = float(entry_Medproducto.get())

        # Comparar con la medida recibida del Arduino

        margen = 0.4

        if medida_ingresada - margen <= medida_pieza <= medida_ingresada + margen:
            lbl_verificacion.config(text="Sí cumple con las especificaciones", foreground="green")
            send_data("estado_depieza", "Aceptado")
            agregar_rutina_a_cola("RUTINA:RUT_MEDIR_PLATO")
            message=b'RUTINA:RUT_MEDIR_PLATO'
            sock.sendall(message)
            sc.after(0, ejecutar_siguiente_rutina_si_libre)

    else:

```

```

contpiezasrechazadas += 1

lbl_verificacion.config(text="No cumple con las especificaciones", foreground="red")

send_data("estado_depieza", "Rechazado")

send_data("piezas_rechazadas", contpiezasrechazadas)

agregar_rutina_a_cola("RUTINA:RUT_MEDIR_DESCARTE")

message=b'RUTINA:RUT_MEDIR_DESCARTE'

sock.sendall(message)

sc.after(0, ejecutar_siguiente_rutina_si_libre)

except ValueError:

    lbl_verificacion.config(text="Medida inválida", foreground="orange")

def actualizar_contadores():

    global color_medir, contpiezasmedidas, medida_pieza,
color_plato_giratorio,contpiezasrechazadas

    global color_cinta, color_torno, contador_piezas, contador_torno,
contpiezasproducidas,contpiezasrojas,contpiezasazul

# Función para leer datos del Arduino Nano

def leer_nano():

    global medida_pieza, contpiezasmedidas, contpiezasproducidas, color_medir,
color_plato_giratorio

```

```

while True:

    if Nano.in_waiting > 0:

        linea = Nano.readline().decode('utf-8', errors='ignore').strip()

        if "producto retirado" in linea:

            color_medir = "yellow"

            sc.after(0, lambda: modificar_circulo('Maquina_de_medir', color_medir))

        elif "esperar producto" in linea:

            contpiezasmedidas += 1

            color_medir = "red"

            sc.after(0, lambda: modificar_circulo('Maquina_de_medir', color_medir))

        elif "prodetectado" in linea:

            color_medir = "green"

            sc.after(0, lambda: modificar_circulo('Maquina_de_medir', color_medir))

        elif "Largo del Objeto:" in linea:

            try:

                medida_str = linea.split(":")[1].strip().replace("cm", "").strip()

                medida_pieza = float(medida_str)

                sc.after(0, lambda: medida_pieza_lbl.config(text=str(medida_pieza)))

                sc.after(0, verificar_medida)

            except ValueError:

```

```

        sc.after(0, lambda: lbl_verificacion.config(text="Medida inválida",
foreground="orange"))

    elif "Plato en posicion home" in linea:

        color_plato_giratorio = "yellow"

        sc.after(0, lambda: modificar_circulo('Plato_giratorio', color_plato_giratorio))

    elif "objeto detectado en el plato giratorio" in linea:

        color_plato_giratorio = "green"

        sc.after(0, lambda: modificar_circulo('Plato_giratorio', color_plato_giratorio))

    elif "Iniciando giro del plato" in linea:

        if contpiezasproducidas < piezas_a_producir:

            contpiezasproducidas += 1

            color_plato_giratorio = "green"

            sc.after(0, lambda: modificar_circulo('Plato_giratorio', color_plato_giratorio))

            sc.after(0, ejecutar_siguiente_rutina_si_libre)

        else:

            # Se alcanzó el número de piezas a producir, detener todo

            cola_rutinas.clear() # Limpiar la cola

            print("Producción finalizada. Se alcanzó el número de piezas deseadas.")

    elif "No se detecta objeto en el plato giratorio" in linea:

        color_plato_giratorio = "yellow"

        modificar_circulo('Plato_giratorio', color_plato_giratorio)

```

```

sc.after(0, lambda: contador_piezas_medidas_lbl.config(text=contpiezasmedidas))

sc.after(0, lambda: contador_piezas_producidas_lbl.config(text=contpiezasproducidas))

# Función para leer datos del Arduino Mega

def leer_mega():

    global contador_piezas, contador_torno, color_cinta, color_torno, contpiezasrojas,
    contpiezasazul

    while True:

        if Mega.in_waiting > 0:

            linea = Mega.readline().decode('utf-8', errors='ignore').strip()

            if "Pieza detectada" in linea:

                contador_piezas += 1

                color_cinta = "red"

                sc.after(0, lambda: modificar_circulo('cinta', color_cinta))

                message=b'RUTINA:RUT_CINTA_TORNO'

                sock.sendall(message)

                agregar_rutina_a_colas("RUTINA:RUT_CINTA_TORNO")

                sc.after(0, ejecutar_siguiente_rutina_si_libre)

```

elif "cinta activada" in linea:

color_cinta = "green"

```
sc.after(0, lambda: modificar_circulo('cinta', color_cinta))
```

elif "ROJO" in linea:

contpiezasrojas += 1

```
message=b'RUTINA:RUT_CINTA_TORNO'
```

```
sock.sendall(message)
```

```
sc.after(0, lambda: contador_piezasrojas_lbl.config(text=contpiezasrojas))
```

elif "AZUL" in linea:

cont piezas azul += 1

```
message=b'RUTINA:RUT_CINTA_TORNO'
```

```
sock.sendall(message)
```

```
sc.after(0, lambda: contador_piezasazul_lbl.config(text=contpiezasazul))
```

elif "Pieza retirada" in linea:

```
if color_cinta != "yellow": # Cambiar solo si no está ya en blanco
```

color_cinta = "yellow"

```
modificar_circulo('cinta', color_cinta)
```

```

elif "material en el torno" in linea:

    if color_torno != "green": # Cambiar solo si no está ya en verde

        color_torno = "green"

        modificar_circulo('torno', color_torno)

        send_data("estado_torno", "En Marcha")

        sc.after(0, ejecutar_siguiente_rutina_si_libre)

elif "Torno terminado" in linea:

    contador_torno += 1

    color_torno = "red"

    agregar_rutina_a_colas("RUTINA:RUT_TORNO_MEDIR")

    sc.after(0, ejecutar_siguiente_rutina_si_libre)

    message=b'RUTINA:RUT_TORNO_MEDIR'

    sock.sendall(message)

    sc.after(0, lambda: modificar_circulo('torno', color_torno))

elif "Pieza fuera del torno" in linea or "No hay pieza en el torno" in linea:

    color_torno = "yellow"

    sc.after(0, lambda: modificar_circulo('torno', color_torno))

    sc.after(0, lambda: contador_piezas_lbl.config(text=contador_piezas))

    sc.after(0, lambda: contador_torno_lbl.config(text=contador_torno))

# Iniciar los hilos para la lectura en paralelo

```

```

hilo_nano = threading.Thread(target=leer_nano, daemon=True)

hilo_mega = threading.Thread(target=leer_mega, daemon=True)

hilo_escucha = threading.Thread(target=escuchar_robot, daemon=True)

hilo_escucha.start()

hilo_nano.start()

hilo_mega.start()

# Función para enviar el número de piezas a producir

def enviar_piezas_a_producir():

    global piezas_a_producir, contpiezasproducidas

    try:

        n = int(entry_piezas.get())

        tiempo_rojo = int(entry_Tiemporojo.get())

        tiempo_azul = int(entry_Tiempoazul.get())

        if 1 <= n <= 4:

            piezas_a_producir = n

            contpiezasproducidas = 0

            contador_piezas_producidas_lbl.config(text=contpiezasproducidas)

            Nano.write(f"PIEZAS:{piezas_a_producir}\n".encode())

            send_data("total_productos", n)

            # Enviar los tiempos de mecanizado al Arduino Mega

            Mega.write(f"TIEMPO_ROJO:{tiempo_rojo}\n".encode())

```

```
Mega.write(f"TIEMPO_AZUL:{tiempo_azul}\n".encode())

else:

    messagebox.showerror("Entrada inválida", "Se puede almacenar hasta 4 Productos.")

except ValueError:

    messagebox.showerror("Entrada inválida. Se puede almacenar hasta 4 Productos.")

# Crear la interfaz unificada con Tkinter

sc = tk.Tk()

sc.title("Monitor de Producción")

sc.config(width=800, height=700)

# Mostrar la imagen en la interfaz

imagen_label = tk.Label(sc)

imagen_label.pack(side='bottom')

actualizar_imagen()

modificar_circulo('Maquina_de_medir', color_medir)

modificar_circulo('cinta', color_cinta)

modificar_circulo('Plato_giratorio', color_plato_giratorio)

modificar_circulo('torno', color_torno)
```

```
# Configuración de widgets

fuente_subrayada = font.Font(family="Arial", size=15, weight="bold", underline=1)

lbl_dato = ttk.Label(sc, text="Datos de Entrada:", font=fuente_subrayada)

lbl_dato.place(x=10, y=10)

lbl_producir = ttk.Label(sc, text="Cantidad de Piezas a Producir:", font=("Arial", 12))

lbl_producir.place(x=10, y=40)

entry_piezas = ttk.Entry(sc, width=5)

entry_piezas.place(x=240, y=40)

entry_piezas.insert(0, "1")

btn_enviar = ttk.Button(sc, text="Establecer", command=enviar_piezas_a_producir)

btn_enviar.place(x=50, y=200)

# Etiquetas de productos en la interfaz

lbl_Medidaproducto= ttk.Label(sc, text="Medidas del Producto:", font=("Arial", 12))

lbl_Medidaproducto.place(x=10, y=65)

entry_Medproducto = ttk.Entry(sc, width=5)

entry_Medproducto.place(x=180, y=65)

entry_Medproducto.insert(0, "1")

lbl_cm1 = ttk.Label(sc, text="Cm", font=("Arial", 10))

lbl_cm1.place(x=202, y=65)
```

```
lbl_producto1 = ttk.Label(sc, text="Producto 1 (Rojo)", font=("Arial", 12, "bold"))

lbl_producto1.place(x=10, y=90)
```

```
lbl_Mecanizadorojo= ttk.Label(sc, text="Tiempo de Mecanizado:",font=("Arial",12))
```

```
lbl_Mecanizadorojo.place(x=10, y=110)
```

```
entry_Tiemporojo = ttk.Entry(sc, width=5)
```

```
entry_Tiemporojo.place(x=180, y=110)
```

```
entry_Tiemporojo.insert(0, "1")
```

```
lbl_seg1 = ttk.Label(sc, text="Seg.", font=("Arial", 10))
```

```
lbl_seg1.place(x=199, y=110)
```

```
lbl_producto2 = ttk.Label(sc, text="Producto 2 (Azul)", font=("Arial", 12, "bold"))
```

```
lbl_producto2.place(x=10, y=140)
```

```
lbl_Mecanizadoazul= ttk.Label(sc, text="Tiempo de Mecanizado:",font=("Arial",12))
```

```
lbl_Mecanizadoazul.place(x=10, y=160)
```

```
entry_Tiempoazul = ttk.Entry(sc, width=5)
```

```
entry_Tiempoazul.place(x=180, y=160)
```

```
entry_Tiempoazul.insert(0, "1")
```

```
lbl_seg2 = ttk.Label(sc, text="Seg.", font=("Arial", 10))
```

```
lbl_seg2.place(x=200, y=160)

lbl_verificacion = ttk.Label(sc, text="Esperando medida...", font=("Arial", 10, "bold"))

lbl_verificacion.place(x=800, y=280)

# Etiquetas para el Nano

lbl_piezas_medidas = ttk.Label(sc, text="Piezas Medidas:", font=("Arial", 12))

lbl_piezas_medidas.place(x=800, y=20)

contador_piezas_medidas_lbl = ttk.Label(sc, text=contpiezasmedidas, font=("Arial", 12))

contador_piezas_medidas_lbl.place(x=935, y=20)

lbl_piezas_producidas = ttk.Label(sc, text="Piezas Producidas:", font=("Arial", 12))

lbl_piezas_producidas.place(x=460, y=260)

contador_piezas_producidas_lbl = ttk.Label(sc, text=contpiezasproducidas, font=("Arial", 12))

contador_piezas_producidas_lbl.place(x=600, y=260)

lbl_medida_pieza = ttk.Label(sc, text="Medida de la Pieza:", font=("Arial", 12))

lbl_medida_pieza.place(x=800, y=255)

medida_pieza_lbl = ttk.Label(sc, text=medida_pieza, font=("Arial", 12))

medida_pieza_lbl.place(x=945, y=255)

lbl_cm = ttk.Label(sc, text="cm", font=("Arial", 12))
```

```
lbl_cm.place(x=965, y=255)

# Etiquetas para el Mega

lbl_piezas = ttk.Label(sc, text="Piezas Ingresadas:", font=("Arial", 12))

lbl_piezas.place(x=950, y=480)

contador_piezas_lbl = ttk.Label(sc, text=contador_piezas , font=("Arial", 12))

contador_piezas_lbl.place(x=1090, y=480)

lbl_contadorrojos = ttk.Label(sc, text="Piezas Rojas Ingresadas:", font=("Arial", 12))

lbl_contadorrojos.place(x=905, y=510)

contador_piezasrojas_lbl = ttk.Label(sc, text=contpiezasrojas, font=("Arial", 12))

contador_piezasrojas_lbl.place(x=1090, y=510)

lbl_contadorrojos = ttk.Label(sc, text="Piezas Azules Ingresadas:", font=("Arial", 12))

lbl_contadorrojos.place(x=904, y=540)

contador_piezasazul_lbl = ttk.Label(sc, text=contpiezasazul, font=("Arial", 12))

contador_piezasazul_lbl.place(x=1090, y=540)

lbl_torno = ttk.Label(sc, text="Torno Operaciones:", font=("Arial", 12))

lbl_torno.place(x=527, y=520)

contador_torno_lbl = ttk.Label(sc, text=contador_torno, font=("Arial", 12))
```

```
contador_torno_lbl.place(x=670, y=520)
```

```
# Actualizar contadores
```

```
actualizar_contadores()
```

```
loop_ejecucion_rutinas()
```

```
sc.mainloop()
```

```
# Iniciar el bucle principal
```

```
sc.mainloop()
```

```
#Cerrar MQTT y serial al finalizar
```

```
client.disconnect()
```

```
# Cerrar comunicación serial al finalizar
```

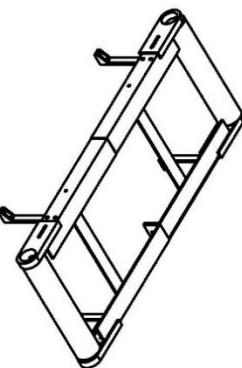
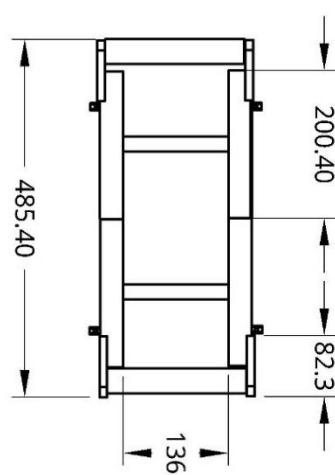
```
Nano.close()
```

```
Mega.close()
```

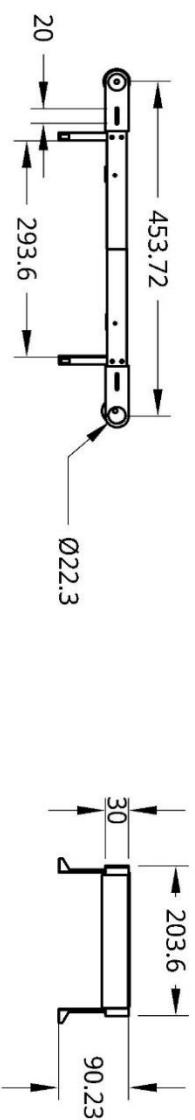
2

1

B



A



2

1

A

B

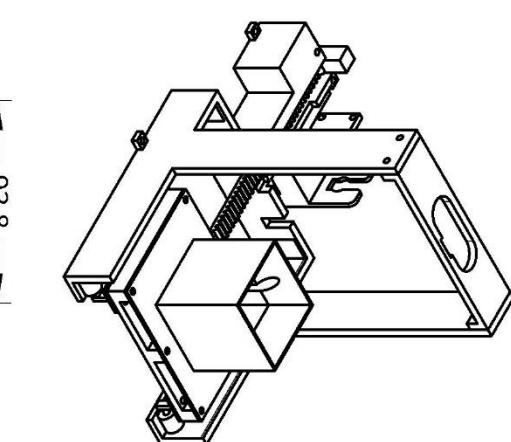
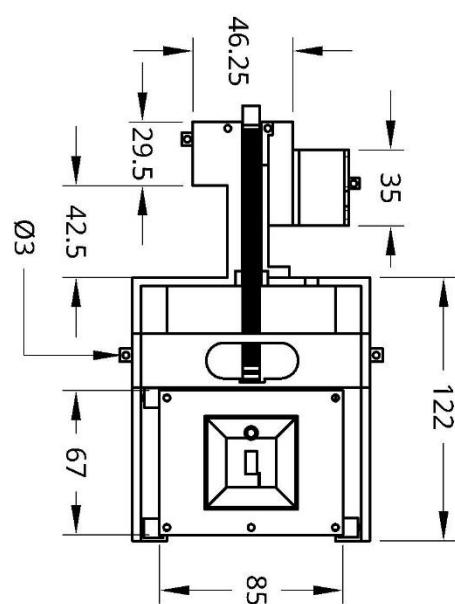
- **ANEXO D: PLANO DE LAS MAQUINAS DE LA CFFD**
Página 110 de 116

UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN MILLIMETRE .XX = ±0. .XXX = ±0.00 ANGULAR: ± FRACTIONAL: ± SURFACE FINISH: ✓					
DRAWN BY	NAME MATEO PABLO	DATE 27/12/2024	TITLE Cinta Transportadora de la Celdula de Fabricación Flexible		
CHECKED APPROVED			SIZE A	DWG NO.	REV.
BREAK ALL SHARP EDGES AND REMOVE BURRS	MATERIAL PLA	FINISH	SIZE A 1:8	WEIGHT 1 of 1	REV.
THIRD ANGLE PROJECTION					

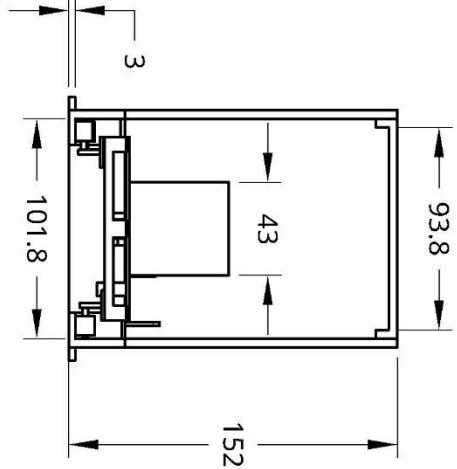
2

1

B



A

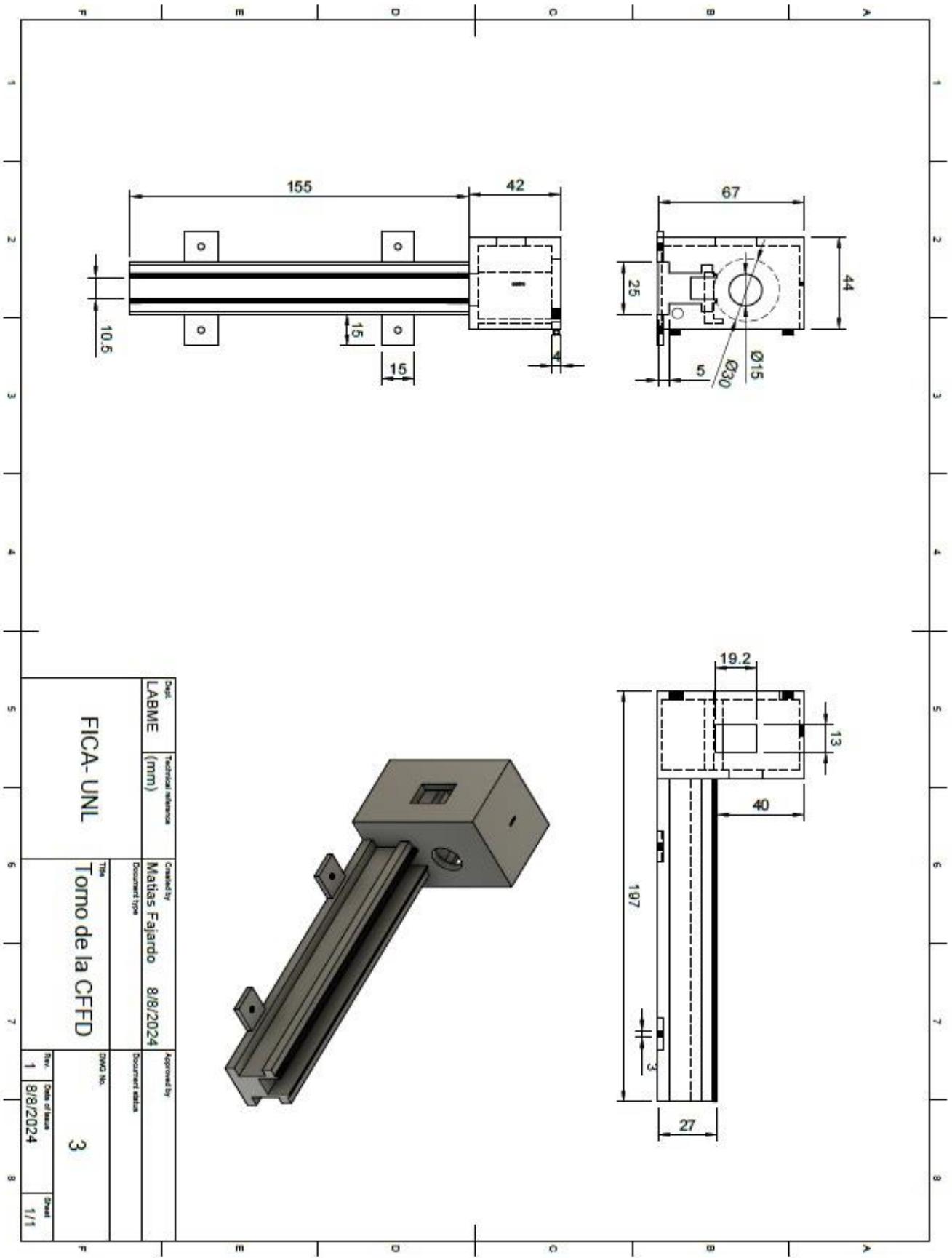


A

UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN INCHES XXX = ±0.05 ANGULAR = ° XXXX = ±0.000 FRACTIONAL = 1/16		
DRAWN	NAME	DATE
MATIAS FAJARDO		04/22/2025
CHECKED		
	TITLE	
	Maquina de Medir	
APPROVED		
DO NOT SCALE DRAWING		
BREAK ALL SHARP EDGES AND REMOVE BURRS		
THROUGH PROJECTION		
	MATERIAL	FINISH
	PLA	
SHEET	2	REV
SCALE	1:4	WEIGHT
		SHEET 1 of 1

2

1

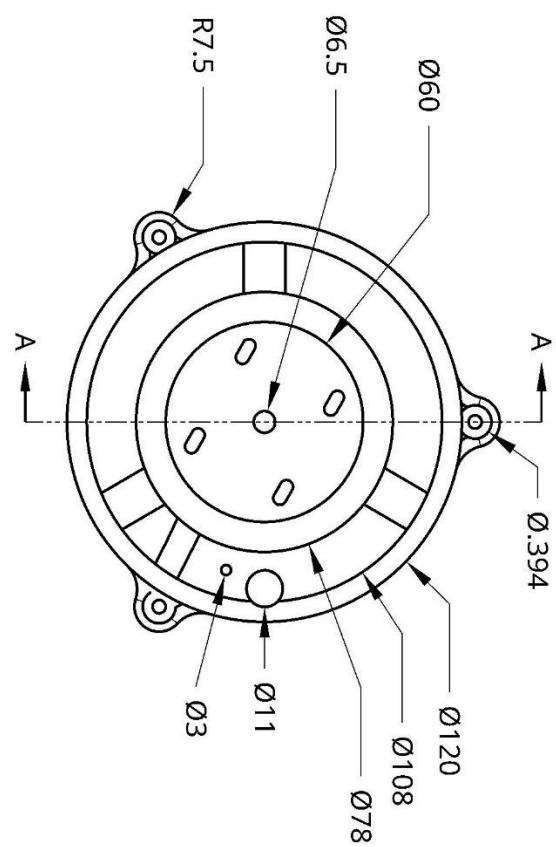


2

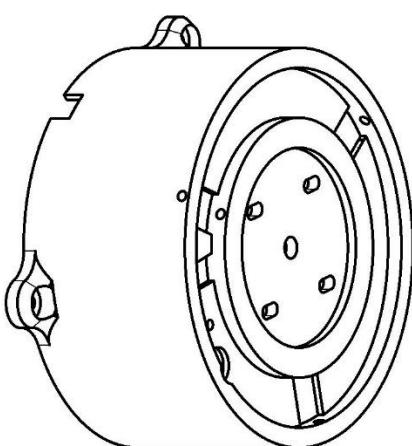
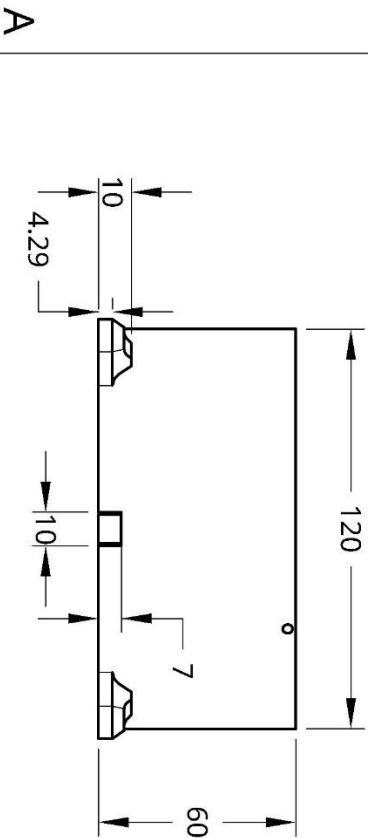
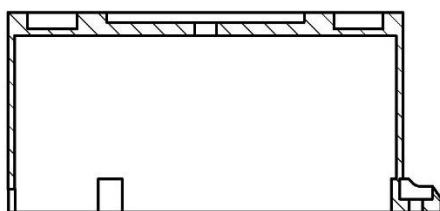
1

B

B



A

SECTION A - A
SCALE 1:2

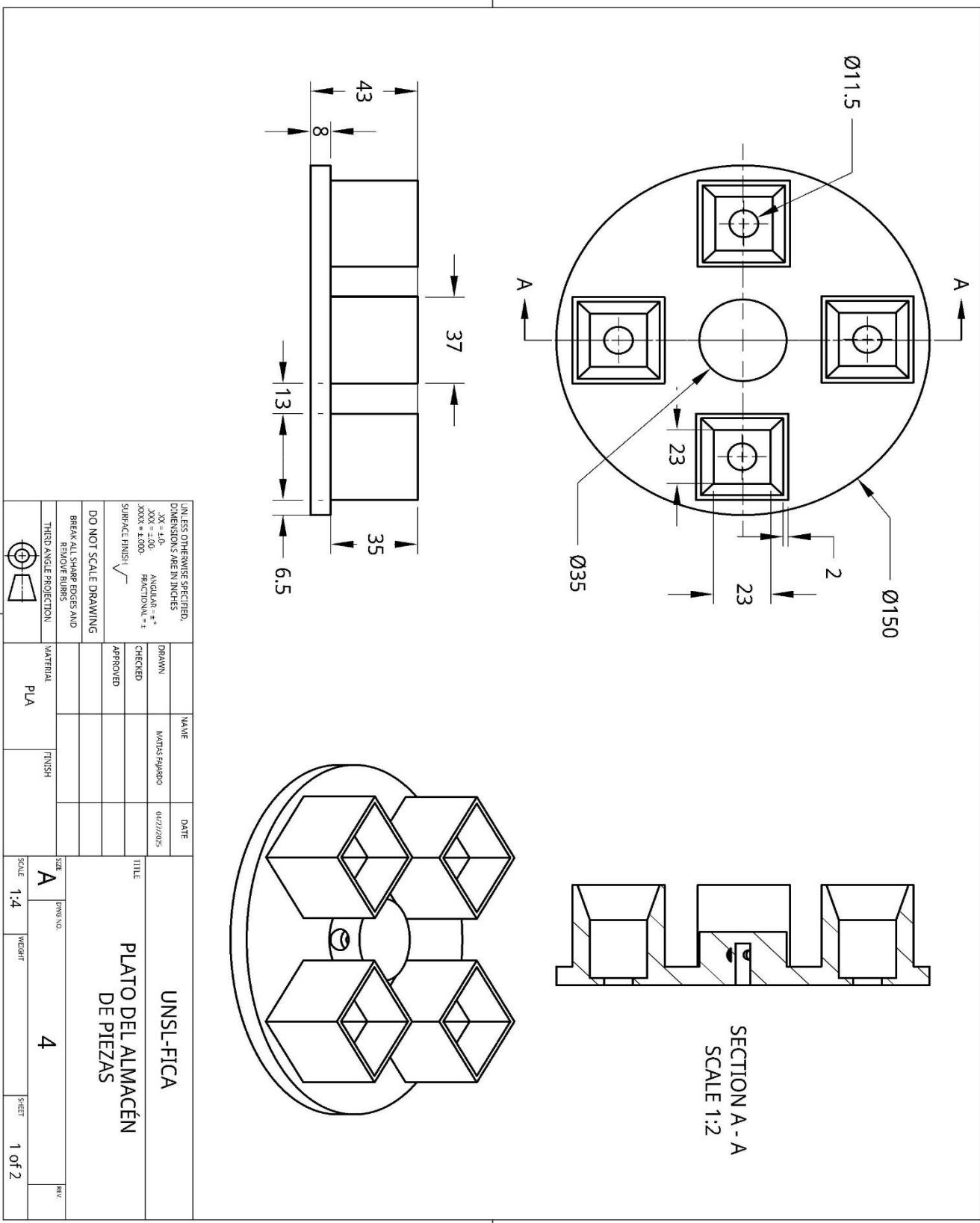
A

A

UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN INCHES XX = ±0.00; XXXX = ±0.000; ANGULAR ± °; FRACTIONAL ± †			NAME _____ DATE _____		
DRAWN BY MATIAS AGUARO 04/26/2025			TITLE _____		
SURFACE FINISH ✓			Base del Almacén de Piezas		
APPROVED _____			REV. _____		
DO NOT SCALE DRAWING BREAK ALL SHARP EDGES AND REMOVE BURRS	MATERIAL PLA	FINISH	SIZE A	DWG NO. 4	SCALE 1:3 WEIGHT REV. _____ SHEET 1 of 1

2

1



- **ANEXO E:** Criterios de Selección de sensores y actuadores

Consideraciones sobre el reemplazo o variación del β :

En el diseño del circuito del motor del torno del capítulo 2, se utilizó un transistor BJT, NPN modelo 2N2222, controlado desde un pin digital del Arduino Mega. Este transistor funciona como interruptor operando en corte y saturación. El mismo se midió experimentalmente la ganancia de corriente (hFE o β) del dispositivo, obteniendo un valor igual a 18, con lo cual se determinó una corriente de base (I_b) de 13 mA, ya que la corriente de colector es la consumida por el motor DC (230 mA), y se seleccionó una resistencia de 330 Ω como valor comercial más cercano.

$$I_b = \frac{V_{out} - V_{be}}{R_b} = \frac{5V - 0,7V}{330\Omega} = 13 \text{ mA} \quad (17)$$

El valor de corriente de base calculado resulta seguro para el pin digital del Arduino Mega utilizado en el circuito, dado que se encuentra por debajo del límite máximo permitido de 20 mA por pin. Sin embargo, es importante remarcar que el valor de β puede variar considerablemente entre dispositivos del mismo modelo, y también con la temperatura. Por esta razón, para futuros diseños o en caso de reemplazo del transistor, se deberá verificar que la corriente de base exigida no exceda la capacidad de entrega del pin del microcontrolador, y que el transistor seleccionado posea una ganancia mínima suficiente para asegurar su operación en saturación con una corriente de colector de 230 mA.

$$\beta = \frac{I_c}{I_b} = \frac{230 \text{ mA}}{13 \text{ mA}} \approx 18 \quad (18)$$

Esta ganancia mínima de transistor debe ser un $\beta \geq 18$ en saturación ya que funcionara sin modificar la resistencia de base. En caso contrarios se deberá ajustar la resistencia de base manteniendo un valor por debajo de la corriente máxima del Arduino.

Criterios de selección de sensores:

Todos los sensores utilizados para el desarrollo de las diferentes estaciones de trabajo de la celda, fueron seleccionados por su disponibilidad en el laboratorio (LABME) y considerando criterios como disponibilidad comercial (en caso de reemplazos), costo y facilidad de integración.

Tabla Nº6. Elementos Electrónicos utilizados

Sensor / Actuador	Función	Criterio de Selección
Sensor Ultrasonido HC-SR04	Detección de piezas en cinta/máquina de medir.	Alcance adecuado (2-400 cm), económico, fácil de usar con Arduino.
Sensor Color TCS230	Identificación de tipo de pieza.	Compatible con Arduino, lectura RGB, bajo costo.
Sensor Infrarrojo TCRT5000	Detección de presencia (pieza o home).	Respuesta rápida, fácil de instalar, funcionamiento fiable en cercanía.
Motor PAP NEMA17	Movimiento del plato giratorio.	Precisión de pasos, torque suficiente, disponibilidad local.
Motor DC 5V	Movimiento de la cinta / torno / cremallera.	Bajo consumo, fácil de controlar con transistores o drivers, tamaño compacto, costo accesible.