



Desarrollo de un prototipo de sistema mecatrónico para el sector manufacturero de industria 4.0  
en el departamento del huila

Diseño e integración de automatismos mecatrónicos

Ficha: 2449131

Instructor Líder: Cesar Eduardo Guevara

Centro de la industria, la empresa y servicio Sede industria

Neiva – Huila

2023

## CONTENIDO

<b>I.</b>	<b>ILUSTRACIONES .....</b>	<b>3</b>
<b>II.</b>	<b>TABLAS .....</b>	<b>4</b>
<b>III.</b>	<b>INTRODUCCION .....</b>	<b>5</b>
<b>IV.</b>	<b>OBJETIVO GENERAL.....</b>	<b>5</b>
<b>V.</b>	<b>OBJETIVOS ESPECÍFICOS .....</b>	<b>5</b>
<b>VI.</b>	<b>HISTORIA .....</b>	<b>6</b>
<b>VII.</b>	<b>ESPECIFICACIÓN TÉCNICA PUNZADORA CON CINTA TRANSPORTADORA 9V7</b>	
	Motor XS.....	7
	Interruptor de límite .....	8
	Fototransistor.....	9
	Barrera de luz led .....	10
<b>VIII.</b>	<b>MODELADO MÁQUINA PUNZADORA CON BANDA TRANSPORTADORA</b>	
	MARCA FISHERTECHNIK .....	11
<b>IX.</b>	<b>FUNCIONAMIENTO PUNZONADORA.....</b>	<b>17</b>
	¿Qué es una banda transportadora? .....	17
	¿Cómo funciona una banda transportadora? .....	17
	¿Como aplicar esto en la vida cotidiana? .....	18
<b>X.</b>	<b>MODULO .....</b>	<b>18</b>
<b>XI.</b>	<b>PROGRAMACIÓN .....</b>	<b>20</b>
	Codigo Python Thonny Lcd .....	21

Codigo Python Thonny Blink .....	26
Codigo Python Thonny librería I2c_lcd.....	32
Codigo Python Thonny IOT-Banda transportadora .....	34
<b>XII. INTERFAZ DE USUARIO BLINK .....</b>	<b>38</b>
<b>XIII. DIAGRAMA DE FLUJO FUNCIONAMIENTO .....</b>	<b>39</b>
<b>XIV. CIRCUITO DE FUNCIONAMIENTO .....</b>	<b>40</b>
<b>XV. RESULTADOS .....</b>	<b>40</b>
<b>XVI. CONCLUSIONES .....</b>	<b>43</b>
<b>I. ILUSTRACIONES</b>	
Ilustración 1. Motor XS .....	7
Ilustración 2. Mini Interruptor .....	8
Ilustración 3. Señal interruptor .....	9
Ilustración 4. Fototransistor .....	9
Ilustración 5. Barrera de luz led.....	10
Ilustración 6.Punzadora vista lateral izquierda .....	11
Ilustración 7.Punzadora vista lateral Derecha.....	12
Ilustración 8.Punzadora vista inferior .....	12
Ilustración 9.Punzadora vista frontal .....	13
Ilustración 10. Plano de vistas Punzadora.....	13
Ilustración 11.Punzadora con banda transportadora vista frontal .....	14
Ilustración 12.Punzadora con banda transportadora vista lateral.....	14

Ilustración 13. Punzadora con banda transportadora vista posterior .....	15
Ilustración 14. Vistas generales punzadora con banda transportadora.....	16
Ilustración 15 Tarjeta de desarrollo Esp32.....	19
Ilustración 16. Interfaz blink en pc. ....	38
Ilustración 17. Interfaz blink en celular .....	38
Ilustración 18. Circuito del prototipo.....	40
Ilustración 19. Prototipo final .....	40
Ilustración 20. Prototipo final .....	41
Ilustración 21. Prototipo final .....	41
Ilustración 22. Prototipo final .....	42
Ilustración 23. Prototipo final .....	42

## II. TABLAS

Tabla 1. Especificaciones motor XS .....	8
Tabla 2. Especificaciones interruptor de limite.....	8
Tabla 3. Especificaciones fototransistor .....	10
Tabla 4. Especificaciones barra de luz led .....	11

### **III. INTRODUCCION**

El proyecto "Desarrollo de un prototipo de sistema mecatrónico para el sector manufacturero de industria 4.0 en el departamento del Huila" representa un importante paso hacia la modernización y automatización de los procesos de fabricación en la región. En el contexto de la industria 4.0, la integración de tecnologías avanzadas se ha vuelto crucial para mejorar la eficiencia y la precisión en la producción industrial. En este sentido, este proyecto ha buscado diseñar e implementar un prototipo de máquina punzadora con cinta transportadora, utilizando componentes mecatrónicos y aprovechando las ventajas de la tecnología de vanguardia. Esta introducción clave establece la relevancia y el propósito del proyecto, así como el enfoque en la industria 4.0 y la necesidad de impulsar el desarrollo del sector manufacturero en el departamento del Huila.

### **IV. OBJETIVO GENERAL**

Desarrollar un prototipo de sistema mecatrónico para el sector manufacturero de industria 4.0 en el departamento del huila

### **V. OBJETIVOS ESPECÍFICOS**

- Elaborar los planos técnicos del prototipo utilizando el software de diseño asistido por computadora SolidWorks, asegurando una representación precisa y detallada de todas las partes y componentes involucrados.
- Desarrollar y programar los algoritmos necesarios para la operación eficiente y sincronizada de todos los elementos del sistema de una banda transportadora con punzador utilizando la plataforma ESP32.
- Implementar los códigos desarrollado en la plataforma de Internet de las Cosas (IOT), asegurando una correcta comunicación y control remoto de la banda transportadora con punzador a través de la conexión a Internet.

- Realizar las conexiones eléctricas y electrónicas de los sistemas de la banda transportadora con punzador, siguiendo las normas y estándares pertinentes para garantizar una comunicación confiable y segura entre los componentes y dispositivos involucrados.

## **VI. HISTORIA**

Las primeras cintas transportadoras que se conocieron fueron empleadas para el transporte de carbón y materiales de la industria minera. El transporte de material mediante cintas transportadoras data de aproximadamente el año 1795. La mayoría de estas tempranas instalaciones se realizaban sobre terrenos relativamente planos, así como en cortas distancias.

El primer sistema de cinta transportadora era muy primitivo y consistía en una cinta de cuero, lona, o cinta de goma que se deslizaba por una tabla plana o cóncava. Este tipo de sistema no fue calificado como exitoso, pero proporcionó un incentivo a los ingenieros para considerar los transportadores como un rápido, económico y seguro método para mover grandes volúmenes de material de un lugar a otro.

Durante los años 20, las instalaciones de la compañía Frick & Co, fundada por el empresario H. C. Frick junto con algunos socios, demostraron que los transportadores de cinta podían trabajar sin ningún problema en largas distancias. Estas instalaciones se realizaron bajo tierra, desde una mina recorriendo casi 8 kilómetros. La cinta transportadora consistía en múltiples pliegues de algodón de pato recubierta de goma natural, que eran los únicos materiales utilizados en esos tiempos para su fabricación. En 1913, Henry Ford introdujo la cadena de montaje basada en cintas transportadoras en las fábricas de producción de la Ford Motor Company.

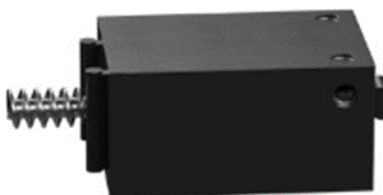
Durante la Segunda Guerra Mundial, los componentes naturales de los transportadores se volvieron muy escasos, permitiendo que la industria de goma se volcara en crear materiales sintéticos que reemplazaran a los naturales. Desde entonces se han desarrollado muchos materiales para aplicaciones muy concretas dentro de la industria, como las bandas con aditivos antimicrobianos para la industria de la alimentación o las bandas con características resistentes para altas temperaturas

Las cintas transportadoras han sido usadas desde el siglo XIX. En 1901, Sandvik inventó y comenzó la producción de cintas transportadoras de acero.

## VII. ESPECIFICACIÓN TÉCNICA PUNZADORA CON CINTA TRANSPORTADORA 9V

La punzadora con cinta transportadora simula el transporte y punzonado de piezas de trabajo. El modelo se puede combinar idealmente con el modelo de entrenamiento "Robot de 3 ejes con pinza". Para realizar su funcionamiento el dispositivo cuenta con 2 motores XS (DC), 2 botones (Interruptor de límite), 2 Fototransistores y 2 barreras de LED.

### Motor XS



*Ilustración 1. Motor XS*

El motor XS es un motor eléctrico largo y alto. Además, es muy ligero. Esto permite instalarlo en lugares donde no hay espacio para los grandes motores. El motor XS está diseñado para una tensión de alimentación de 9 voltios y un consumo máximo de energía de 0,3 amperios.

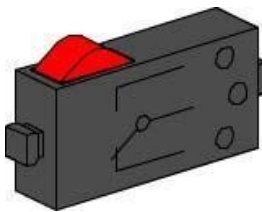
Color	Negro
-------	-------

Alimentación	9VCC
Velocidad	5995 rpm
Torque	1,52 mNm
Corriente	265 mA

*Tabla 1. Especificaciones motor XS*

### **Interruptor de límite**

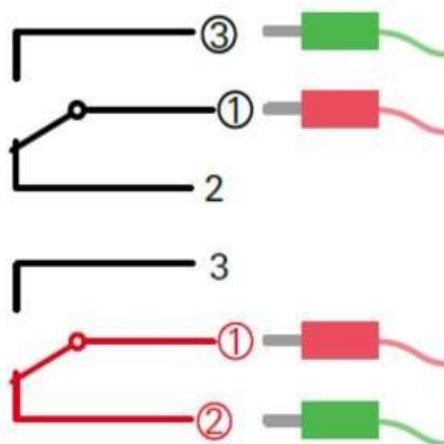
Los pulsadores cuentan como sensores de contacto. Al activar el botón rojo, se mueve de manera mecánica un contacto en la carcasa y fluye corriente entre las conexiones.

Art.-Nr.	parte.	37783	
No.			
Nombre		Mini interruptor	
Dimensiones		30x15x7,5mm	<i>Ilustración 2. Mini Interruptor</i>
Peso		3,4g	
Potencia de conmutación:		máx. 2A, 50V	

*Tabla 2. Especificaciones interruptor de límite*

Señal: Interruptor digital 0 /1, se puede utilizar "normalmente abierto" o "normalmente cerrado"

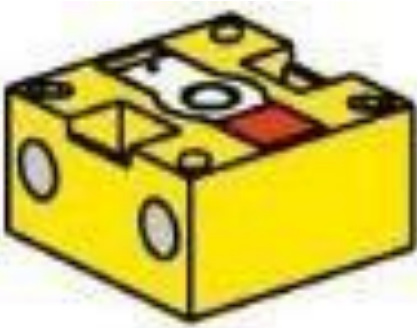




*Ilustración 3. Señal interruptor*

### Fototransistor

El fototransistor también se conoce como "sensor de brillo". Este es un "sensor" que reacciona al brillo. En el caso de una barrera de luz, forma la contrapartida de la lámpara de lente. Con mucha luz, es decir, cuando el transistor está iluminado por la lámpara de la lente, conduce la electricidad. Si se interrumpe el haz de luz, el transistor no conduce corriente.

Art. Nr parte.No.	36134	
Nombre	Fototransistor	
Dimensiones	15x15x7,5mm	
Peso	1,7g	

*Ilustración 4. Fototransistor*

Tipo:	Fototransistor de silicio NPN para fotocélula (con fotocélulas LED)
Tensión colector-emisor VCE	35V
Corriente del colector IC	15mA
Corriente de sobretensión del colector ICS	75mA

*Tabla 3. Especificaciones fototransistor*

### **Barrera de luz led**

Un diodo emisor de luz es un elemento semiconductor, que emite luz. Se lo denomina «LED». La abreviatura proviene del inglés «light-emitting diode». Si fluye corriente a través del diodo, el mismo emite luz. La longitud de onda (color de la luz) depende del material semiconductor y del dopaje. El LED emite luz blanca. La barrera de luz LED se utiliza como transmisor para una barrera de luz.

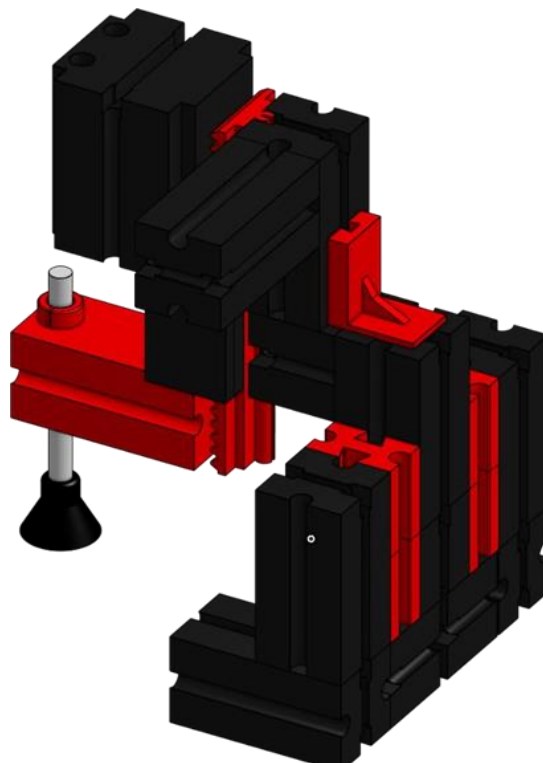


*Ilustración 5. Barrera de luz led*

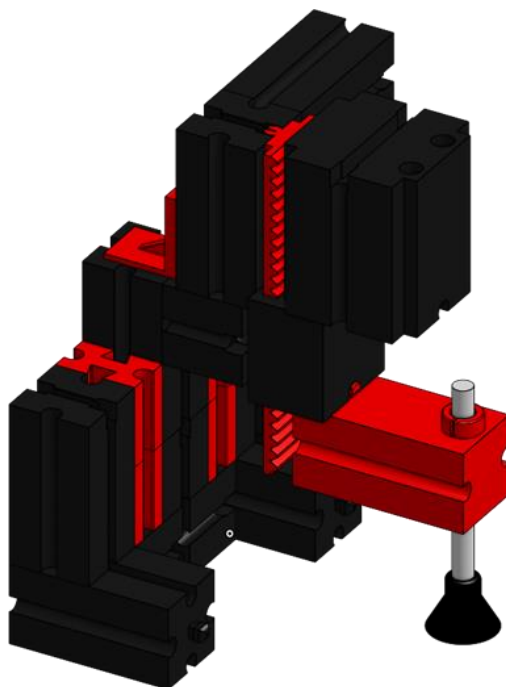
Alimentación (Voltaje)	9 VDC
Corriente	0.02 A

*Tabla 4. Especificaciones barra de luz led*

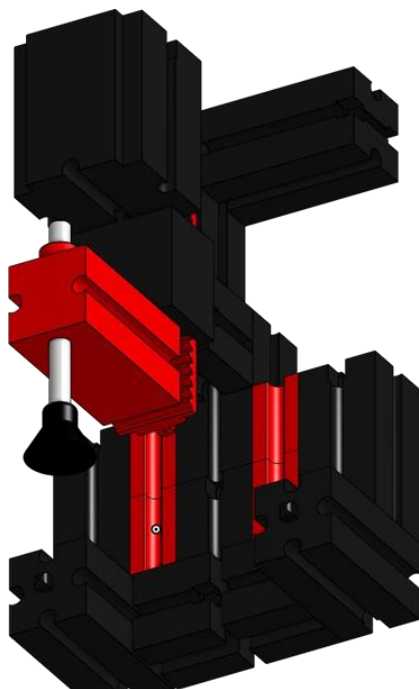
## VIII. MODELADO MÁQUINA PUNZADORA CON BANDA TRANSPORTADORA MARCA FISHERTECHNIK



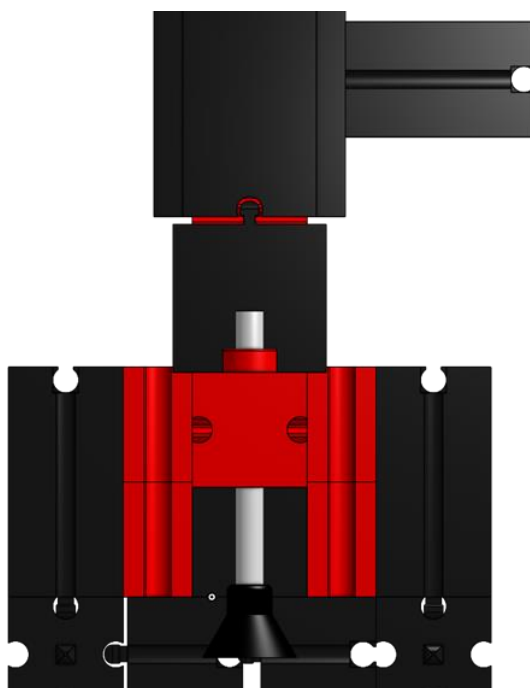
*Ilustración 6. Punzadora vista lateral izquierda*



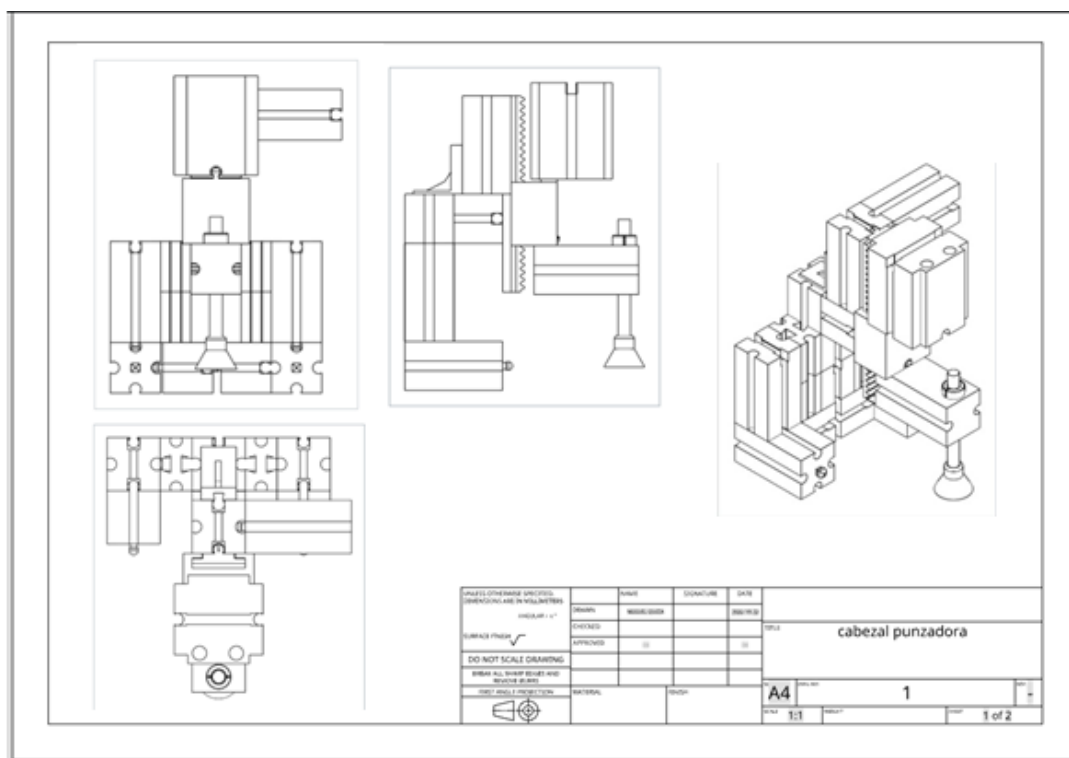
*Ilustración 7. Punzadora vista lateral Derecha*



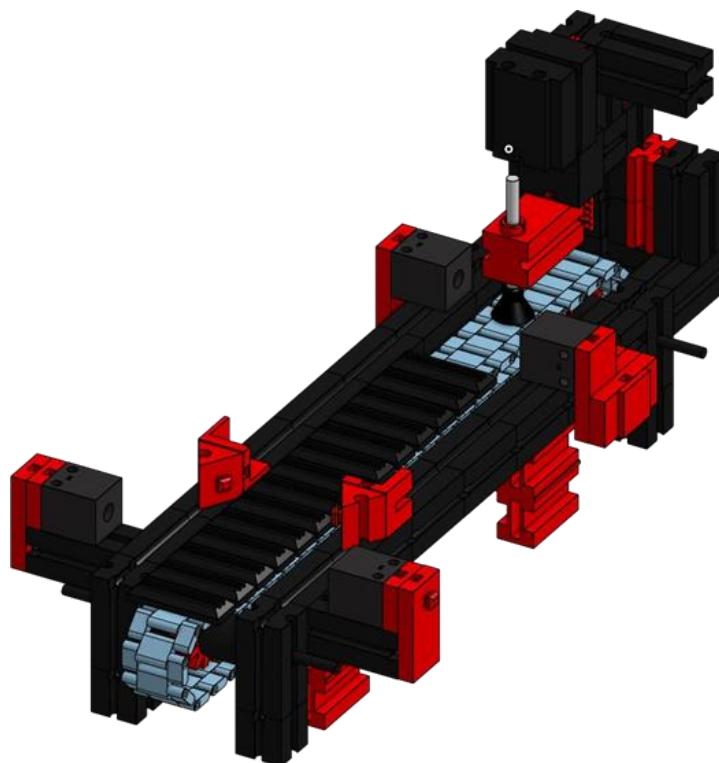
*Ilustración 8. Punzadora vista inferior*



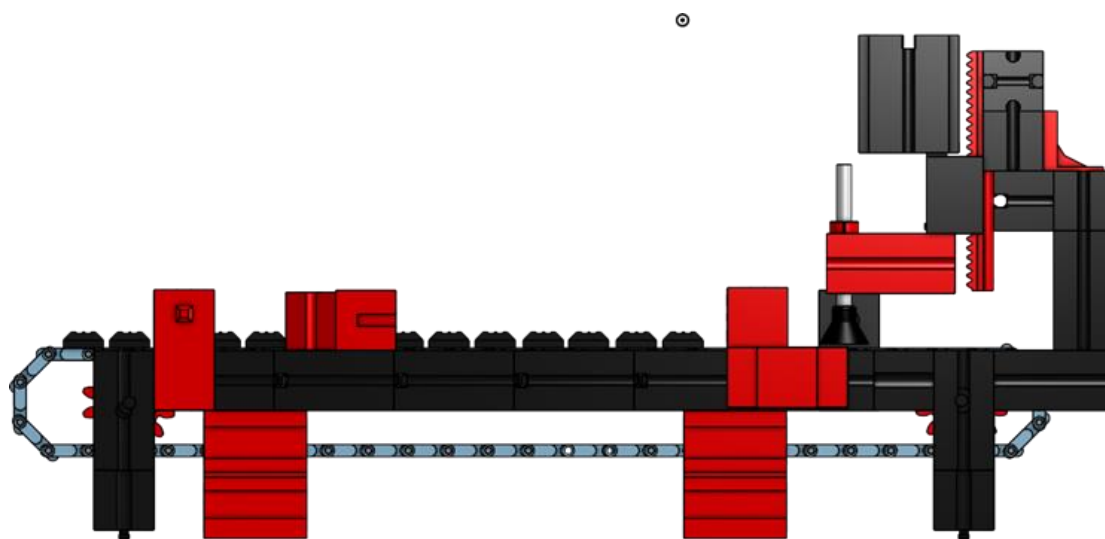
*Ilustración 9. Punzadora vista frontal*



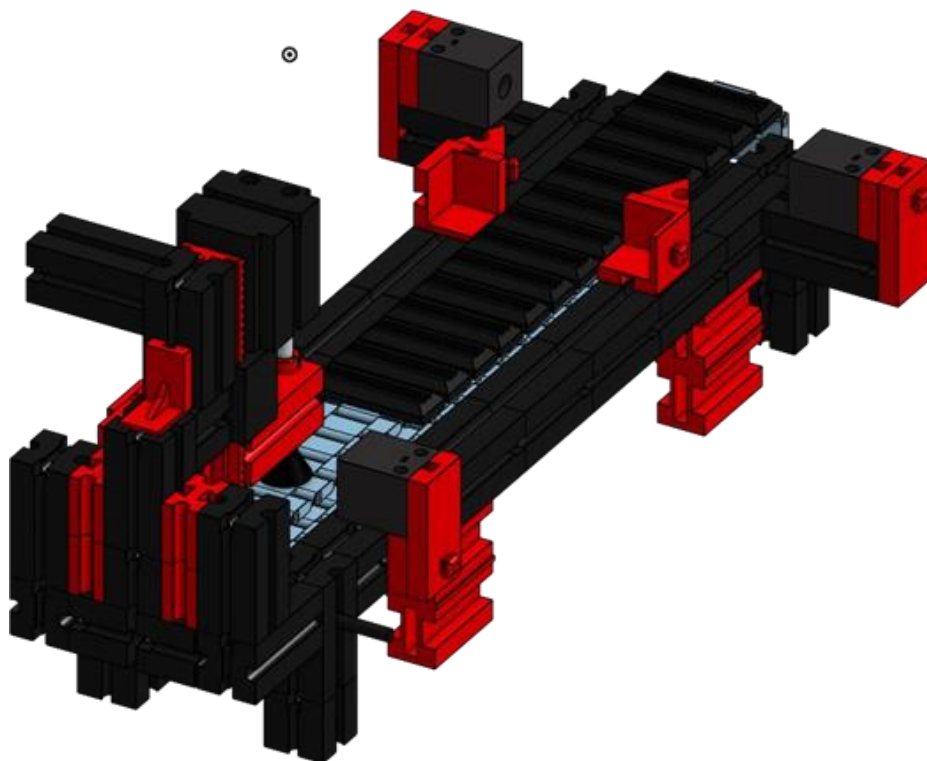
*Ilustración 10. Plano de vistas Punzadora*



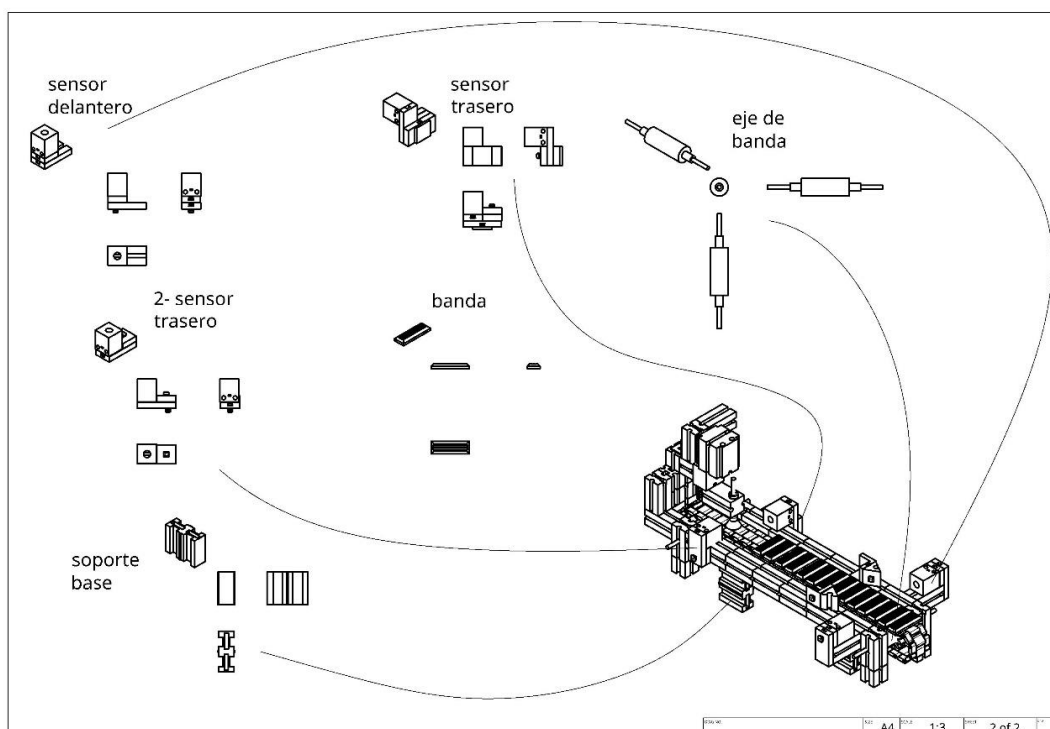
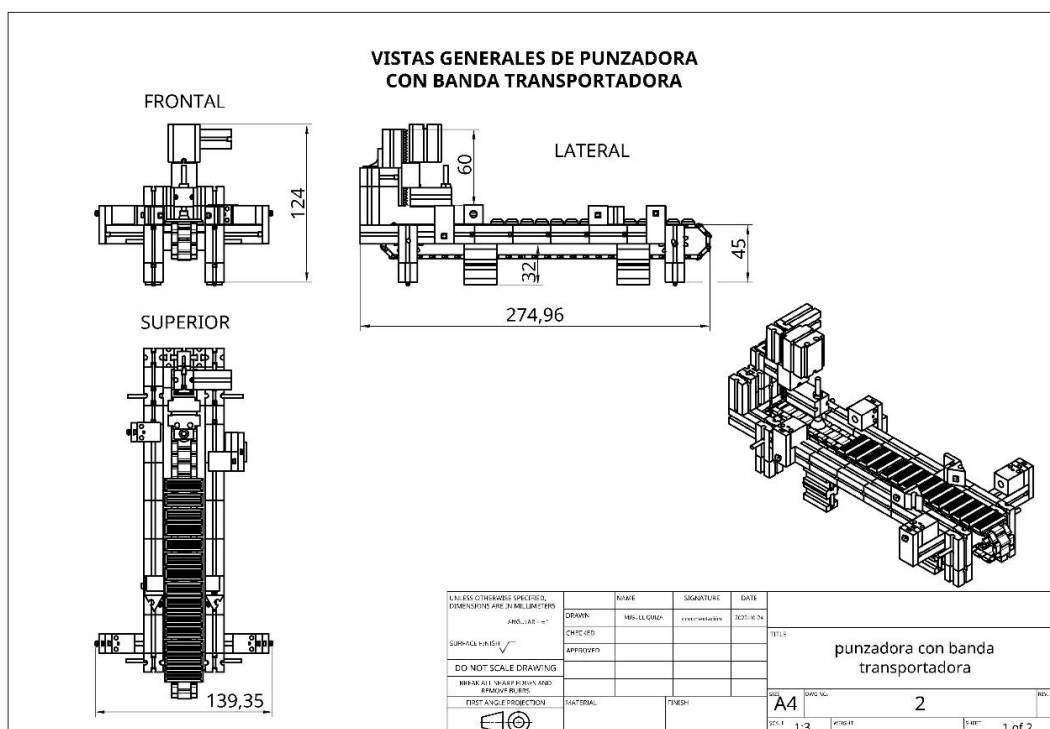
*Ilustración 11. Punzadora con banda transportadora vista frontal*



*Ilustración 12. Punzadora con banda transportadora vista lateral*



*Ilustración 13. Punzadora con banda transportadora vista posterior*



*Ilustración 14. Vistas generales punzadora con banda transportadora*



## **IX. FUNCIONAMIENTO PUNZONADORA**

### **¿Qué es una banda transportadora?**

La banda o cinta transportadora es un componente incorporado mayormente en la industria, su función es mover o trasladar de un lugar a otro los materiales utilizados en el procedimiento de fabricación de un producto, haciendo que el ritmo de trabajo avance de manera rápida y no se vea perjudicado el costo de operaciones debido al tiempo de más invertido en diferentes actividades.

El primer registro que se tiene del uso de las bandas transportadoras se remonta al año de 1795 en la industria minera, donde materiales como el carbón eran trasladados en una primera versión de estas. Estaban hechas a base de madera y con una cinta de lona o cuero, su tamaño era bastante reducido por lo que no eran utilizadas en muchos procedimientos; no sería hasta principios del siglo XX que comenzaron a fabricarse y emplearse como son conocidas hasta el día de hoy.

### **¿Cómo funciona una banda transportadora?**

Su funcionamiento es a través de la cinta, que funge como un soporte físico de forma continua gracias a una polea matriz motorizada, permitiéndole el movimiento constante hacia adelante. Se encuentra montada en una superficie normalmente metálica y cuyas dimensiones varían, al igual que los componentes que la conforman, para definir la carga que es capaz de soportar y la velocidad que tendrá para realizar el traslado de los materiales.

Las bandas transportadoras se mueven con rodillos o tambores y dos o más bidones que transportan los materiales; para mejorar sus capacidades pueden contar con barandillas, guardas laterales, ruedas o cualquier otro accesorio mecánico dependiendo del procedimiento y la materia que desee realizar la industria en cuestión.

### **¿Como aplicar esto en la vida cotidiana?**

Este prototipo es utilizado para cortar materiales como acero al carbono, aluminio y acero inoxidable. Una vez que el proceso de punzonado ha terminado, el material cortado se expulsa de manera rápida y conveniente.

La punzonadora funciona de manera similar a una perforadora de papel. El punzón presiona el material contra una matriz y finalmente se forma una abertura redonda.

El proceso de punzonado consta de cuatro fases: en primer lugar, el punzón entra en contacto con el material y lo deforma. Luego, se corta el material a lo largo del contorno del corte debido a la alta tensión interna. El trozo de material cortado, llamado trozo de punzón se expulsa hacia abajo. Al volver a subir, el punzón puede arrastrar el material, por lo que se utiliza un extractor para liberarlo.

Una ventaja importante de la punzonadora es su versatilidad en el punzonado y perforación de diferentes materiales, sin límites que abarcan desde acero inoxidable hasta plástico y madera. El grosor del material que se puede mecanizar varía entre 0,5 mm y 6,0 mm.

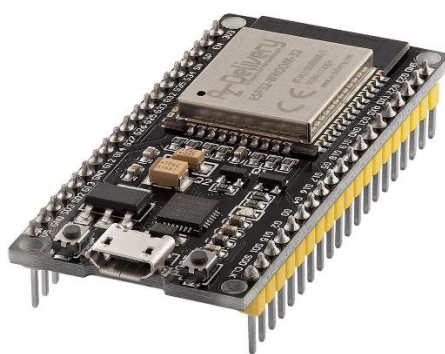
Sin embargo, es necesario capacitar a los operadores en el uso correcto de estas máquinas para evitar peligros. El acceso a la máquina debe estar restringido durante el movimiento entre el punzón y la matriz para evitar atrapamientos. Además, se deben utilizar equipos de protección para prevenir cortes causados por las piezas recién perforadas. Dado que las piezas suelen tener dimensiones grandes, es importante mantener el área de la punzonadora libre de personas para evitar golpes accidentales.

## **X. MODULO**

En este prototipo se ha seleccionado el módulo ESP32 debido a su idoneidad para aplicaciones de la industria 4.0. Esta elección se basa en la necesidad de contar con un módulo

que cumpla con especificaciones claves, como conectividad Wi-Fi, lo que permite establecer comunicación con un servidor remoto. El microprocesador con el que cuenta el módulo ESP32 es el Xtensa Dual-Core 32-bit LX6, este es un componente de alto rendimiento utilizado en diversos dispositivos electrónicos. Cuenta con dos núcleos de procesamiento, lo que le permite realizar múltiples tareas simultáneamente y ejecutar aplicaciones de forma eficiente.

Cada núcleo del microprocesador opera a una velocidad de hasta 600 DMIPS (Dhrystone Million Instructions Per Second), lo que indica su capacidad para ejecutar millones de instrucciones por segundo. Esta alta potencia de procesamiento permite manejar cargas de trabajo exigentes y realizar cálculos complejos de manera rápida y precisa. Esta funcionalidad resulta fundamental para lograr un control remoto del proceso del prototipo a través de dispositivos móviles u otros dispositivos conectados a la red.



*Ilustración 15 Tarjeta de desarrollo Esp32*

La integración del módulo ESP32 en este prototipo proporciona una solución eficiente y confiable para la comunicación inalámbrica. La conectividad Wi-Fi permite la transmisión de datos en tiempo real y la posibilidad de enviar comandos y recibir retroalimentación desde un

servidor centralizado. Esto permite supervisar y controlar el proceso del prototipo de forma remota, sin importar la ubicación física del usuario.

Además de la conectividad Wi-Fi, el ESP32 ofrece otras características importantes para este proyecto de la industria 4.0. Su procesador dual-core de 32 bits y su velocidad de reloj de hasta 240 MHz proporcionan un rendimiento de procesamiento potente y eficiente. Esto permite la ejecución de algoritmos complejos y el manejo de grandes volúmenes de datos en tiempo real; el ESP32 cuenta con interfaces y periféricos versátiles, como puertos GPIO (34), puertos UART (2), puertos I2C (2) y puertos SPI (4), que permiten la conexión con una amplia gama de sensores, actuadores y otros dispositivos externos. Esto facilita la integración del prototipo con otros sistemas y componentes de la industria 4.0, permitiendo una mayor automatización y control en el proceso.

## **XI. PROGRAMACIÓN**

La programación del microcontrolador para el proceso de la punzadora con banda transportadora implica el desarrollo de cuatro códigos en MicroPython. Estos códigos son esenciales para el funcionamiento completo del prototipo y permiten establecer una conexión en Internet de las cosas (IOT).

La elección del lenguaje de programación MicroPython se basa en su compatibilidad con el microcontrolador utilizado. Se ha seleccionado este lenguaje debido a que es uno de los aceptados por el microcontrolador y permite una implementación eficiente del código.

Además, se ha incorporado una pantalla LCD 16x2 I2C al prototipo con el objetivo de mejorar la experiencia del usuario. Esta pantalla proporciona una interfaz visual donde se pueden mostrar información relevante sobre el funcionamiento de la punzadora con banda transportadora, lo que facilita la interacción y el monitoreo del sistema en tiempo real.

## Código Python Thonny Lcd

```

"""Provides an API for talking to HD44780 compatible character LCDs."""

import time

class LcdApi:
    """Implements the API for talking with HD44780 compatible character LCDs.
    This class only knows what commands to send to the LCD, and not how to get
    them to the LCD.

    It is expected that a derived class will implement the hal_xxx functions.
    """

    # The following constant names were lifted from the avrlib lcd.h
    # header file, however, I changed the definitions from bit numbers
    # to bit masks.
    #
    # HD44780 LCD controller command set

    LCD_CLR = 0x01          # DB0: clear display
    LCD_HOME = 0x02         # DB1: return to home position

    LCD_ENTRY_MODE = 0x04   # DB2: set entry mode
    LCD_ENTRY_INC = 0x02    # --DB1: increment
    LCD_ENTRY_SHIFT = 0x01  # --DB0: shift

    LCD_ON_CTRL = 0x08      # DB3: turn lcd/cursor on
    LCD_ON_DISPLAY = 0x04   # --DB2: turn display on
    LCD_ON_CURSOR = 0x02    # --DB1: turn cursor on
    LCD_ON_BLINK = 0x01     # --DB0: blinking cursor

    LCD_MOVE = 0x10         # DB4: move cursor/display
    LCD_MOVE_DISP = 0x08    # --DB3: move display (0-> move cursor)
    LCD_MOVE_RIGHT = 0x04   # --DB2: move right (0-> left)

    LCD_FUNCTION = 0x20     # DB5: function set
    LCD_FUNCTION_8BIT = 0x10 # --DB4: set 8BIT mode (0->4BIT mode)
    LCD_FUNCTION_2LINES = 0x08 # --DB3: two lines (0->one line)
    LCD_FUNCTION_10DOTS = 0x04 # --DB2: 5x10 font (0->5x7 font)
    LCD_FUNCTION_RESET = 0x30 # See "Initializing by Instruction" section

    LCD_CGRAM = 0x40        # DB6: set CG RAM address
    LCD_DDRAM = 0x80        # DB7: set DD RAM address

    LCD_RS_CMD = 0
    LCD_RS_DATA = 1

```

```

LCD_RW_WRITE = 0
LCD_RW_READ = 1

def __init__(self, num_lines, num_columns):
    self.num_lines = num_lines
    if self.num_lines > 4:
        self.num_lines = 4
    self.num_columns = num_columns
    if self.num_columns > 40:
        self.num_columns = 40
    self.cursor_x = 0
    self.cursor_y = 0
    self.implicit_newline = False
    self.backlight = True
    self.display_off()
    self.backlight_on()
    self.clear()
    self.hal_write_command(self.LCD_ENTRY_MODE | self.LCD_ENTRY_INC)
    self.hide_cursor()
    self.display_on()

def clear(self):
    """Clears the LCD display and moves the cursor to the top left
    corner.
    """
    self.hal_write_command(self.LCD_CLR)
    self.hal_write_command(self.LCD_HOME)
    self.cursor_x = 0
    self.cursor_y = 0

def show_cursor(self):
    """Causes the cursor to be made visible."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
                           self.LCD_ON_CURSOR)

def hide_cursor(self):
    """Causes the cursor to be hidden."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY)

def blink_cursor_on(self):
    """Turns on the cursor, and makes it blink."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
                           self.LCD_ON_CURSOR | self.LCD_ON_BLINK)

```

```
def blink_cursor_off(self):
    """Turns on the cursor, and makes it no blink (i.e. be solid)."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY |
                           self.LCD_ON_CURSOR)

def display_on(self):
    """Turns on (i.e. unblanks) the LCD."""
    self.hal_write_command(self.LCD_ON_CTRL | self.LCD_ON_DISPLAY)

def display_off(self):
    """Turns off (i.e. blanks) the LCD."""
    self.hal_write_command(self.LCD_ON_CTRL)

def backlight_on(self):
    """Turns the backlight on.

    This isn't really an LCD command, but some modules have backlight
    controls, so this allows the hal to pass through the command.
    """
    self.backlight = True
    self.hal_backlight_on()

def backlight_off(self):
    """Turns the backlight off.

    This isn't really an LCD command, but some modules have backlight
    controls, so this allows the hal to pass through the command.
    """
    self.backlight = False
    self.hal_backlight_off()

def move_to(self, cursor_x, cursor_y):
    """Moves the cursor position to the indicated position. The cursor
    position is zero based (i.e. cursor_x == 0 indicates first column).
    """
    self.cursor_x = cursor_x
    self.cursor_y = cursor_y
    addr = cursor_x & 0x3f
    if cursor_y & 1:
        addr += 0x40 # Lines 1 & 3 add 0x40
    if cursor_y & 2: # Lines 2 & 3 add number of columns
        addr += self.num_columns
    self.hal_write_command(self.LCD_DDRAM | addr)
```

```

def putchar(self, char):
    """Writes the indicated character to the LCD at the current cursor
    position, and advances the cursor by one position.
    """
    if char == '\n':
        if self.implicit_newline:
            # self.implicit_newline means we advanced due to a wraparound,
            # so if we get a newline right after that we ignore it.
            pass
        else:
            self.cursor_x = self.num_columns
    else:
        self.hal_write_data(ord(char))
        self.cursor_x += 1
    if self.cursor_x >= self.num_columns:
        self.cursor_x = 0
        self.cursor_y += 1
        self.implicit_newline = (char != '\n')
    if self.cursor_y >= self.num_lines:
        self.cursor_y = 0
    self.move_to(self.cursor_x, self.cursor_y)

def putstr(self, string):
    """Write the indicated string to the LCD at the current cursor
    position and advances the cursor position appropriately.
    """
    for char in string:
        self.putchar(char)

def custom_char(self, location, charmap):
    """Write a character to one of the 8 CGRAM locations, available
    as chr(0) through chr(7).
    """
    location &= 0x7
    self.hal_write_command(self.LCD_CGRAM | (location << 3))
    self.hal_sleep_us(40)
    for i in range(8):
        self.hal_write_data(charmap[i])
        self.hal_sleep_us(40)
    self.move_to(self.cursor_x, self.cursor_y)

def hal_backlight_on(self):
    """Allows the hal layer to turn the backlight on.

    If desired, a derived HAL class will implement this function.
    """
    pass

```



```
def hal_backlight_off(self):
    """Allows the hal layer to turn the backlight off.

    If desired, a derived HAL class will implement this function.
    """
    pass

def hal_write_command(self, cmd):
    """Write a command to the LCD.

    It is expected that a derived HAL class will implement this
    function.
    """
    raise NotImplementedError

def hal_write_data(self, data):
    """Write data to the LCD.

    It is expected that a derived HAL class will implement this
    function.
    """
    raise NotImplementedError

def hal_sleep_us(self, usecs):
    """Sleep for some time (given in microseconds)."""
    time.sleep_us(usecs)
```

## Código Python Thonny Blink

# Copyright (c) 2015-2019 Volodymyr Shymanskyy. See the file LICENSE for copying permission.

```
__version__ = "1.0.0"

import struct
import time
import sys
import os

try:
    import machine
    gettime = lambda: time.ticks_ms()
    SOCK_TIMEOUT = 0
except ImportError:
    const = lambda x: x
    gettime = lambda: int(time.time() * 1000)
    SOCK_TIMEOUT = 0.05

def dummy(*args):
    pass

MSG_RSP = const(0)
MSG_LOGIN = const(2)
MSG_PING = const(6)

MSG_TWEET = const(12)
MSG_NOTIFY = const(14)
MSG_BRIDGE = const(15)
MSG_HW_SYNC = const(16)
MSG_INTERNAL = const(17)
MSG_PROPERTY = const(19)
MSG_HW = const(20)
MSG_HW_LOGIN = const(29)
MSG_EVENT_LOG = const(64)

MSG_REDIRECT = const(41) # TODO: not implemented
MSG_DBG_PRINT = const(55) # TODO: not implemented

STA_SUCCESS = const(200)
STA_INVALID_TOKEN = const(9)

DISCONNECTED = const(0)
CONNECTING = const(1)
CONNECTED = const(2)
```

[illegible]

```

class EventEmitter:
    def __init__(self):
        self._cbks = {}

    def on(self, evt, f=None):
        if f:
            self._cbks[evt] = f
        else:
            def D(f):
                self._cbks[evt] = f
                return f
            return D

    def emit(self, evt, *a, **kv):
        if evt in self._cbks:
            self._cbks[evt](*a, **kv)

class BlynkProtocol(EventEmitter):
    def __init__(self, auth, tmpl_id=None, fw_ver=None, heartbeat=50, buffin=1024, log=None):
        EventEmitter.__init__(self)
        self.heartbeat = heartbeat*1000
        self.buffin = buffin
        self.log = log or dummy
        self.auth = auth
        self.tmpl_id = tmpl_id
        self.fw_ver = fw_ver
        self.state = DISCONNECTED
        self.connect()

    def virtual_write(self, pin, *val):
        self._send(MSG_HW, 'vw', pin, *val)

    def send_internal(self, pin, *val):
        self._send(MSG_INTERNAL, pin, *val)

    def set_property(self, pin, prop, *val):
        self._send(MSG_PROPERTY, pin, prop, *val)

    def sync_virtual(self, *pins):
        self._send(MSG_HW_SYNC, 'vr', *pins)

    def log_event(self, *val):
        self._send(MSG_EVENT_LOG, *val)

```

```

def _send(self, cmd, *args, **kwargs):
    if 'id' in kwargs:
        id = kwargs.get('id')
    else:
        id = self.msg_id
        self.msg_id += 1
        if self.msg_id > 0xFFFF:
            self.msg_id = 1

    if cmd == MSG_RSP:
        data = b''
        dlen = args[0]
    else:
        data = ('\0'.join(map(str, args))).encode('utf8')
        dlen = len(data)

    self.log('<', cmd, id, '|', *args)
    msg = struct.pack("!BHH", cmd, id, dlen) + data
    self.lastSend = gettime()
    self._write(msg)

def connect(self):
    if self.state != DISCONNECTED: return
    self.msg_id = 1
    (self.lastRecv, self.lastSend, self.lastPing) = (gettime(), 0, 0)
    self.bin = b""
    self.state = CONNECTING
    self._send(MSG_HW_LOGIN, self.auth)

def disconnect(self):
    if self.state == DISCONNECTED: return
    self.bin = b""
    self.state = DISCONNECTED
    self.emit('disconnected')

def process(self, data=None):
    if not (self.state == CONNECTING or self.state == CONNECTED): return
    now = gettime()
    if now - self.lastRecv > self.heartbeat+(self.heartbeat//2):
        return self.disconnect()
    if (now - self.lastPing > self.heartbeat//10 and
        (now - self.lastSend > self.heartbeat or
         now - self.lastRecv > self.heartbeat)):
        self._send(MSG_PING)
        self.lastPing = now

    if data != None and len(data):
        self.bin += data

```

```

while True:
    if len(self.bin) < 5:
        break

    cmd, i, dlen = struct.unpack("!BHH", self.bin[:5])
    if i == 0: return self.disconnect()

    self.lastRecv = now
    if cmd == MSG_RSP:
        self.bin = self.bin[5:]

        self.log('>', cmd, i, '|', dlen)
        if self.state == CONNECTING and i == 1:
            if dlen == STA_SUCCESS:
                self.state = CONNECTED
                dt = now - self.lastSend
                info = ['ver', __version__, 'h-beat', self.heartbeat//1000, 'buff-in', self.buffin, 'dev', sys.platform+'-py']
                if self.tmpl_id:
                    info.extend(['tmpl', self.tmpl_id])
                    info.extend(['fw-type', self.tmpl_id])
                if self.fw_ver:
                    info.extend(['fw', self.fw_ver])
                self._send(MSG_INTERNAL, *info)
                try:
                    self.emit('connected', ping=dt)
                except TypeError:
                    self.emit('connected')
            else:
                if dlen == STA_INVALID_TOKEN:
                    self.emit("invalid_auth")
                    print("Invalid auth token")
                    return self.disconnect()
        else:
            if dlen >= self.buffin:
                print("Cmd too big: ", dlen)
                return self.disconnect()

            if len(self.bin) < 5+dlen:
                break

            data = self.bin[5:5+dlen]
            self.bin = self.bin[5+dlen:]

            args = list(map(lambda x: x.decode('utf8'), data.split(b'\0')))

```

```

        self.log('>', cmd, i, '|', '|'.join(args))
        if cmd == MSG_PING:
            self._send(MSG_RSP, STA_SUCCESS, id=i)
        elif cmd == MSG_HW or cmd == MSG_BRIDGE:
            if args[0] == 'vw':
                self.emit("V"+args[1], args[2:])
                self.emit("V*", args[1], args[2:])
            elif cmd == MSG_INTERNAL:
                self.emit("internal:"+args[0], args[1:])
            elif cmd == MSG_REDIRECT:
                self.emit("Redirect", args[0], int(args[1]))
            else:
                print("Unexpected command: ", cmd)
                return self.disconnect()

import socket

class Blynk(BlynkProtocol):
    def __init__(self, auth, **kwargs):
        self.insecure = kwargs.pop('insecure', False)
        self.server = kwargs.pop('server', 'blynk.cloud')
        self.port = kwargs.pop('port', 80 if self.insecure else 443)
        BlynkProtocol.__init__(self, auth, **kwargs)
        self.on('redirect', self.redirect)

    def redirect(self, server, port):
        self.server = server
        self.port = port
        self.disconnect()
        self.connect()

    def connect(self):
        print('Connecting to %s:%d...' % (self.server, self.port))
        s = socket.socket()
        s.connect(socket.getaddrinfo(self.server, self.port)[0][-1])
        try:
            s.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)
        except:
            pass
        if self.insecure:
            self.conn = s
        else:
            try:
                import ssl
                ssl_context = ssl
            except ImportError:
                import ssl
                ssl_context = ssl.create_default_context()
            self.conn = ssl_context.wrap_socket(s, server_hostname=self.server)

```

```
try:
    self.conn.settimeout(SOCK_TIMEOUT)
except:
    s.settimeout(SOCK_TIMEOUT)
BlynkProtocol.connect(self)

def _write(self, data):
    #print('<', data)
    self.conn.write(data)
    # TODO: handle disconnect

def run(self):
    data = b''
    try:
        data = self.conn.read(self.buffer)
        #print('>', data)
    except KeyboardInterrupt:
        raise
    except socket.timeout:
        # No data received, call process to send ping messages when needed
        pass
    except: # TODO: handle disconnect
        return
    self.process(data)
```

## Código Python Thonny librería I2c\_lcd

```
import utime
import gc

from lcd_api import LcdApi
from machine import I2C

# PCF8574 pin definitions
MASK_RS = 0x01      # P0
MASK_RW = 0x02      # P1
MASK_E  = 0x04      # P2

SHIFT_BACKLIGHT = 3 # P3
SHIFT_DATA      = 4 # P4-P7

class I2cLcd(LcdApi):

    #Implements a HD44780 character LCD connected via PCF8574 on I2C

    def __init__(self, i2c, i2c_addr, num_lines, num_columns):
        self.i2c = i2c
        self.i2c_addr = i2c_addr
        self.i2c.writeto(self.i2c_addr, bytes([0]))
        utime.sleep_ms(20) # Allow LCD time to powerup
        # Send reset 3 times
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        utime.sleep_ms(5) # Need to delay at least 4.1 msec
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        utime.sleep_ms(1)
        self.hal_write_init_nibble(self.LCD_FUNCTION_RESET)
        utime.sleep_ms(1)
        # Put LCD into 4-bit mode
        self.hal_write_init_nibble(self.LCD_FUNCTION)
        utime.sleep_ms(1)
        LcdApi.__init__(self, num_lines, num_columns)
        cmd = self.LCD_FUNCTION
        if num_lines > 1:
            cmd |= self.LCD_FUNCTION_2LINES
        self.hal_write_command(cmd)
        gc.collect()

    def hal_write_init_nibble(self, nibble):
        # Writes an initialization nibble to the LCD.
        # This particular function is only used during initialization.
        byte = ((nibble >> 4) & 0x0f) << SHIFT_DATA
        self.i2c.writeto(self.i2c_addr, bytes([byte | MASK_E]))
        self.i2c.writeto(self.i2c_addr, bytes([byte]))
        gc.collect()
```



```

def hal_backlight_on(self):
    # Allows the hal layer to turn the backlight on
    self.i2c.writeto(self.i2c_addr, bytes([1 << SHIFT_BACKLIGHT]))
    gc.collect()

def hal_backlight_off(self):
    #Allows the hal layer to turn the backlight off
    self.i2c.writeto(self.i2c_addr, bytes([0]))
    gc.collect()

def hal_write_command(self, cmd):
    # Write a command to the LCD. Data is latched on the falling edge of E.
    byte = ((self.backlight << SHIFT_BACKLIGHT) |
            ((cmd >> 4) & 0x0f) << SHIFT_DATA))
    self.i2c.writeto(self.i2c_addr, bytes([byte | MASK_E]))
    self.i2c.writeto(self.i2c_addr, bytes([byte]))
    byte = ((self.backlight << SHIFT_BACKLIGHT) |
            ((cmd & 0x0f) << SHIFT_DATA))
    self.i2c.writeto(self.i2c_addr, bytes([byte | MASK_E]))
    self.i2c.writeto(self.i2c_addr, bytes([byte]))
    if cmd <= 3:
        # The home and clear commands require a worst case delay of 4.1 msec
        utime.sleep_ms(5)
    gc.collect()

def hal_write_data(self, data):
    # Write data to the LCD. Data is latched on the falling edge of E.
    byte = (MASK_RS |
            (self.backlight << SHIFT_BACKLIGHT) |
            (((data >> 4) & 0x0f) << SHIFT_DATA))
    self.i2c.writeto(self.i2c_addr, bytes([byte | MASK_E]))
    self.i2c.writeto(self.i2c_addr, bytes([byte]))
    byte = (MASK_RS |
            (self.backlight << SHIFT_BACKLIGHT) |
            ((data & 0x0f) << SHIFT_DATA))
    self.i2c.writeto(self.i2c_addr, bytes([byte | MASK_E]))
    self.i2c.writeto(self.i2c_addr, bytes([byte]))
    gc.collect()

```

## Código Python Thonny IOT-Banda transportadora

```
# Import required modules
#define BLYNK_TEMPLATE_ID "TMPL23-29bHyA"
# define BLYNK_TEMPLATE_NAME "IOT"

import network
from machine import Pin, PWM, SoftI2C
import BlynkLib
import uasyncio as asyncio
import time
from lcd_api import LcdApi
from i2c_lcd import I2cLcd

# Connect to Wi-Fi network
wifi_ssid = "Mi 10"
wifi_password = "Diego15122001"

sta_if = network.WLAN(network.STA_IF)
sta_if.active(True)
sta_if.connect(wifi_ssid, wifi_password)

while not sta_if.isconnected():
    pass

print("Wi-Fi connected:", sta_if.ifconfig())

# Connect to Blynk server
auth_token = "_QD1n6H6Dr5CSuehtobiz69VCfRx1a6T"

blynk = BlynkLib.Blynk(auth_token)

# Define pin numbers
led_pin = 15
# Initialize LED pin
led = Pin(led_pin, Pin.OUT)

#LCD
I2C_ADDR = 0x27 #the address of your LCD I2C device
#in our case, we are using a LDC 20x2
totalRows = 2
totalColumns = 20

#initialize the SoftI2C method for ESP32 by giving it three arguments.
i2c = SoftI2C(scl=Pin(22), sda=Pin(21), freq=10000)
```

```
#This line is used to initialize the I2C connection for the library by creating an 'lcd' object.
lcd = I2CLcd(i2c, I2C_ADDR, totalRows, totalColumns)

# Sensor pins
s1 = Pin(27, Pin.IN, Pin.PULL_DOWN) # Sensor 1
s2 = Pin(35, Pin.IN, Pin.PULL_DOWN) # Sensor 2
s3 = Pin(36, Pin.IN, Pin.PULL_DOWN) # Sensor 3 - ALTO
s4 = Pin(39, Pin.IN, Pin.PULL_DOWN) # Sensor 4 - BAJO

# Motor pins
frequency = 240
duty = 600
duty2 = 600

motorA = PWM(Pin(18, Pin.OUT, Pin.PULL_DOWN), frequency) # Banda
motorB = PWM(Pin(19, Pin.OUT, Pin.PULL_DOWN), frequency) # Banda
motorC = PWM(Pin(25, Pin.OUT, Pin.PULL_DOWN), frequency) # Trocleadora
motorD = PWM(Pin(26, Pin.OUT, Pin.PULL_DOWN), frequency) # Trocleadora

def motor(on, off):
    motorA.duty(on)
    motorB.duty(off)

def motor_2(on, off):
    motorC.duty(on)
    motorD.duty(off)

async def trocleadora(s3, s4):
    contador = 0
    while contador < 20:
        #Sensor 1 - ALTO
        right_value = s3.value() # Obtener el valor del sensor derecho IR (0 o 1)
        await asyncio.sleep_ms(100)
        if right_value == 1:
            motor_2(duty2, 0)
            contador += 1
            if contador == 20:
                motor_2(0, duty)
                await asyncio.sleep_ms(500)
                motor_2(0, 0)
```

```

# Sensor 2 - BAJO
left_value = s4.value() # Obtener el valor del sensor izquierdo IR (0 o 1)
await asyncio.sleep_ms(100)
if left_value == 1:
    motor_2(0, duty2)
    contador += 1
    if contador == 20:
        motor_2(duty, 0)
        await asyncio.sleep_ms(500)
        motor_2(0, 0)

await asyncio.sleep_ms(100) # Esperar 100 milisegundos

async def banda(s1, s2):
    activo = True # Variable bandera para controlar el bucle while True
    right_value = s1.value()
    while activo: # Mientras la bandera esté activa
        right_value = s1.value()

        if right_value == 1:
            motor(0, 0)
            print(s1.value())
        else:
            while right_value == 0:
                motor(duty, 0)
                print(s1.value())
                left_value = s2.value()

            if left_value == 0:
                print(s2.value())
                await asyncio.sleep_ms(1250)
                motor(0, 0) # Detener la banda transportadora
                activo = False # Desactivar la bandera para salir del bucle while True
                break

        await asyncio.sleep_ms(100)
    #break # Romper el ciclo después de esperar 100 ms

async def banda2(s1):
    activo = True # Variable bandera para controlar el bucle while True
    right_value = s1.value()
    while activo:
        motor(0, duty)
        right_value = s1.value()
        if right_value == 0:
            await asyncio.sleep_ms(1250)
            motor(0, 0)
            activo = False # Desactivar la bandera para salir del bucle while True
            break

```

```

async def troceladora_p(s3,s4):
    activo2 = True
    while activo2:
        motor(0,0)
        left_value = s4.value()
        await asyncio.sleep_ms(100)
        if left_value == 1:
            motor_2(0, duty)
            right_value = s3.value()
            await asyncio.sleep_ms(100)
            if right_value == 1:
                motor_2(0, 0)
                activo2 = False
                break
            right_value = s3.value()
            if right_value == 1:
                motor_2(0, 0)
                activo2 = False
                break
        else:
            break

# Define Blynk virtual pin handlers
@blynk.on("V0")
def v0_handler(value):
    lcd.clear() #Erase all characters or anything written on the screen
    lcd.move_to(0,0) #Move to position based on row and col values
    lcd.putstr("MECATRONICA-SENA") # Send a string of characters to the screen
    lcd.move_to(0,1) #Move to position based on row and col values
    lcd.putstr("FICHA-2449131") # Send a string of characters to the screen
    if int(value[0]) == 1:
        async def run_tasks():
            lcd.clear() #Erase all characters or anything written on the screen
            lcd.move_to(0,0) #Move to position based on row and col values
            lcd.putstr("BANDA TRANSPORTADORA") # Send a string of characters to the screen
            lcd.move_to(0,1)
            lcd.putstr("EN FUNCIONAMIENTO")
            time.sleep(0.5) #time to show
            await banda(s1,s2)
            lcd.clear() #Erase all characters or anything written on the screen
            lcd.move_to(0,0) #Move to position based on row and col values
            lcd.putstr("TROQUELADORA") # Send a string of characters to the screen
            lcd.move_to(0,1)
            lcd.putstr("EN FUNCIONAMIENTO")
            time.sleep(0.5) #time to show
            await trocleadora(s3,s4)
            lcd.clear() #Erase all characters or anything written on the screen
            lcd.move_to(0,0) #Move to position based on row and col values
            lcd.putstr("CONTROL CALIDAD") # Send a string of characters to the screen
            lcd.move_to(0,1)
            lcd.putstr("EN FUNCIONAMIENTO")
            await banda2(s1)
        loop = asyncio.get_event_loop()
        loop.create_task(run_tasks())
        loop.run_forever()
    else:
        led.value(0)

# Start Blynk loop
while True:
    blynk.run()

```

## XII. INTERFAZ DE USUARIO BLINK

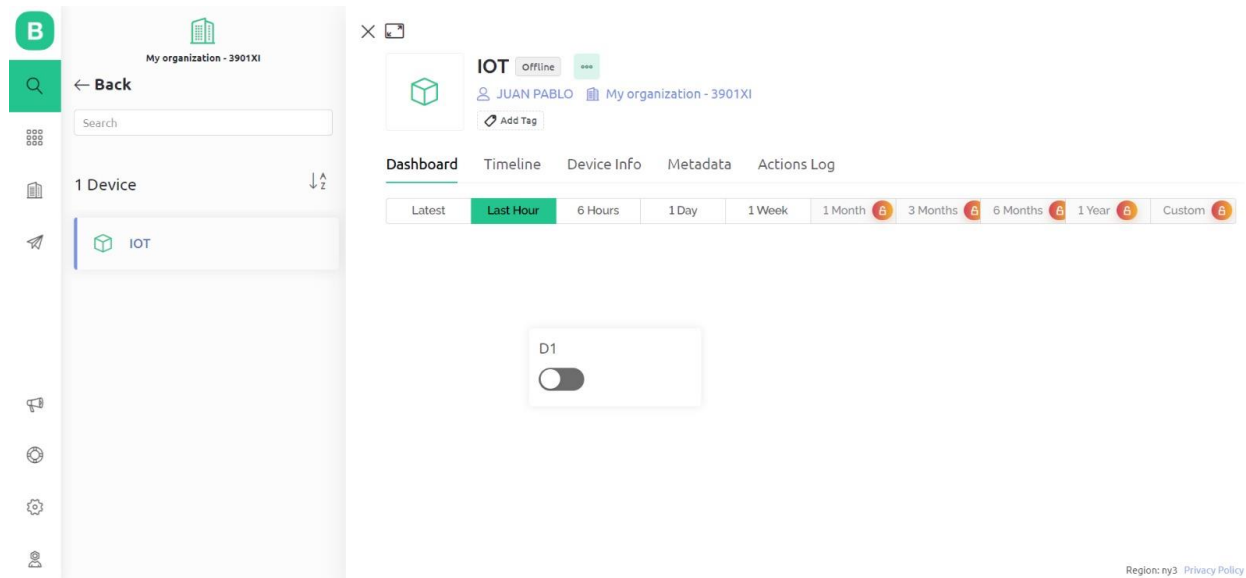


Ilustración 16. Interfaz blink en pc.

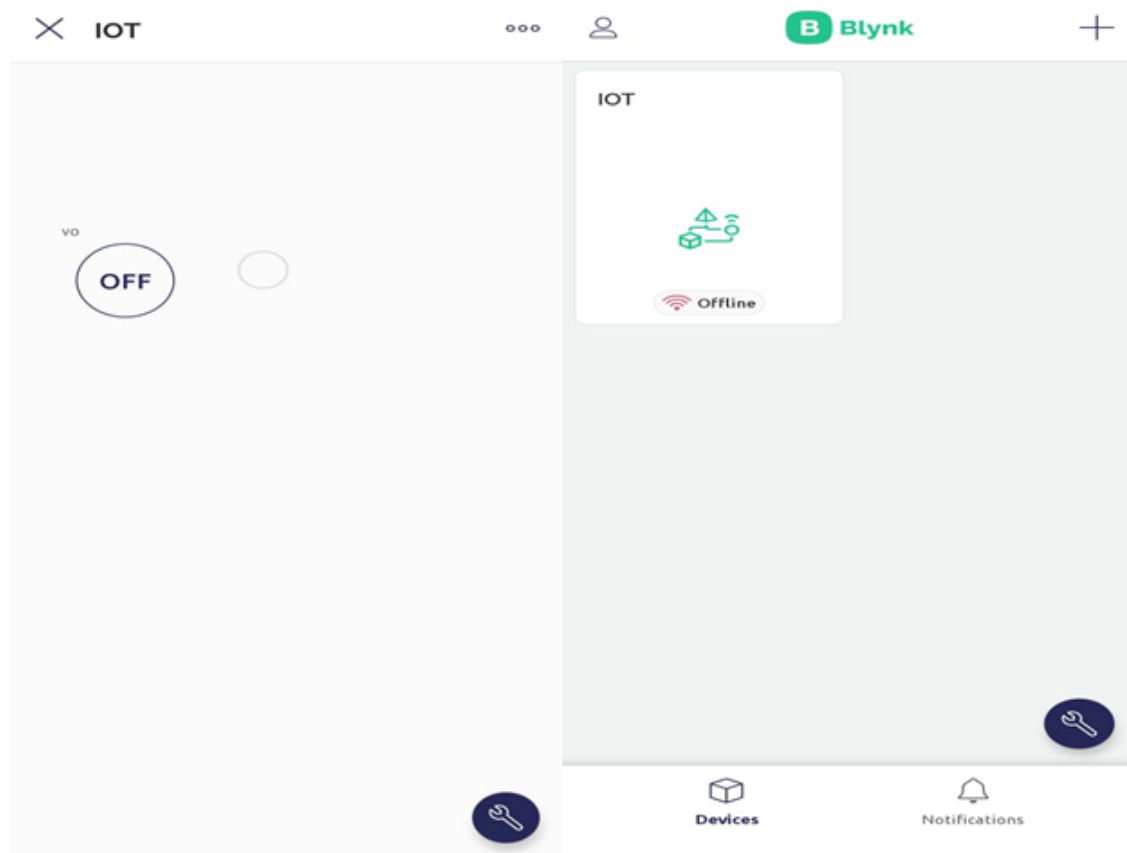
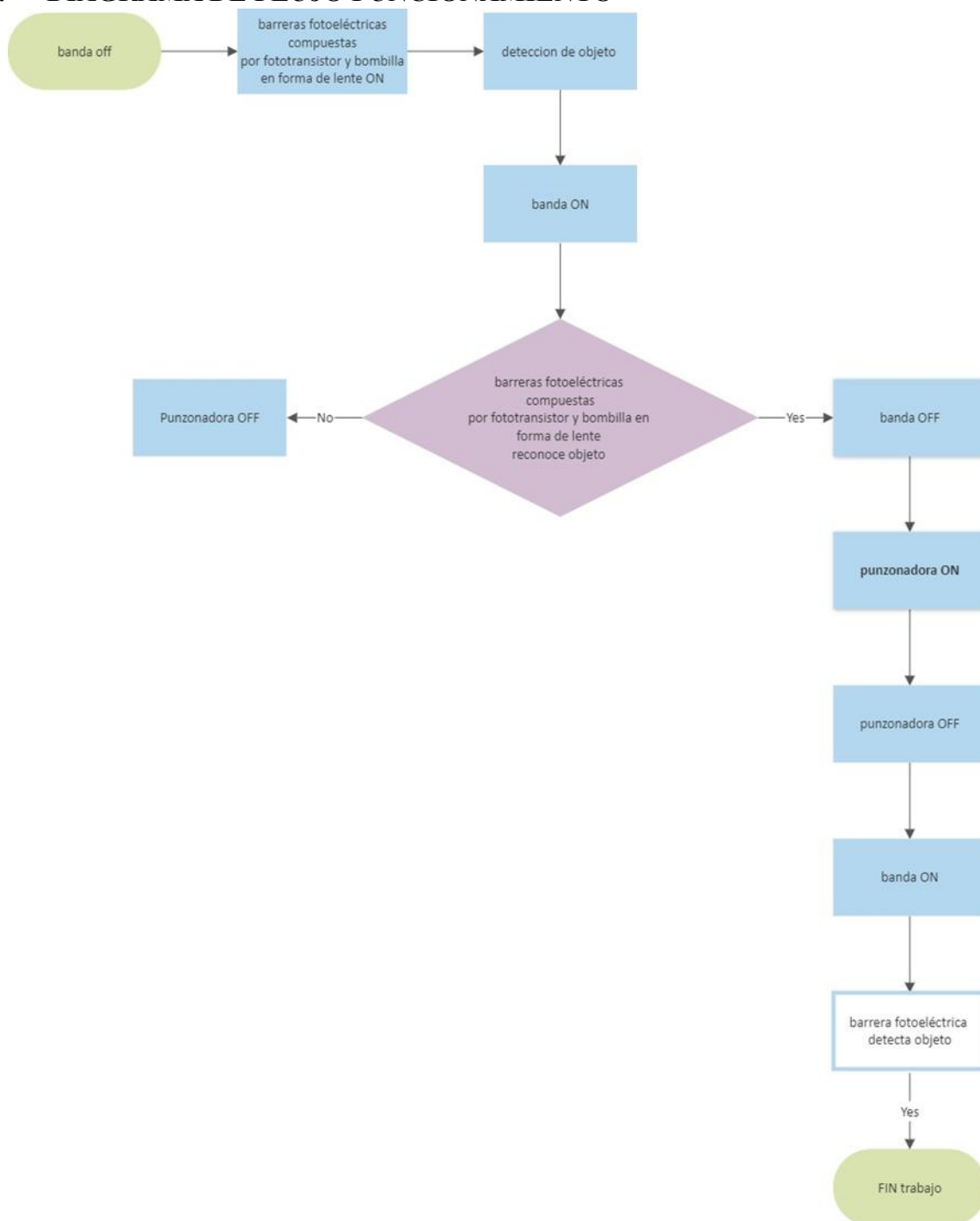
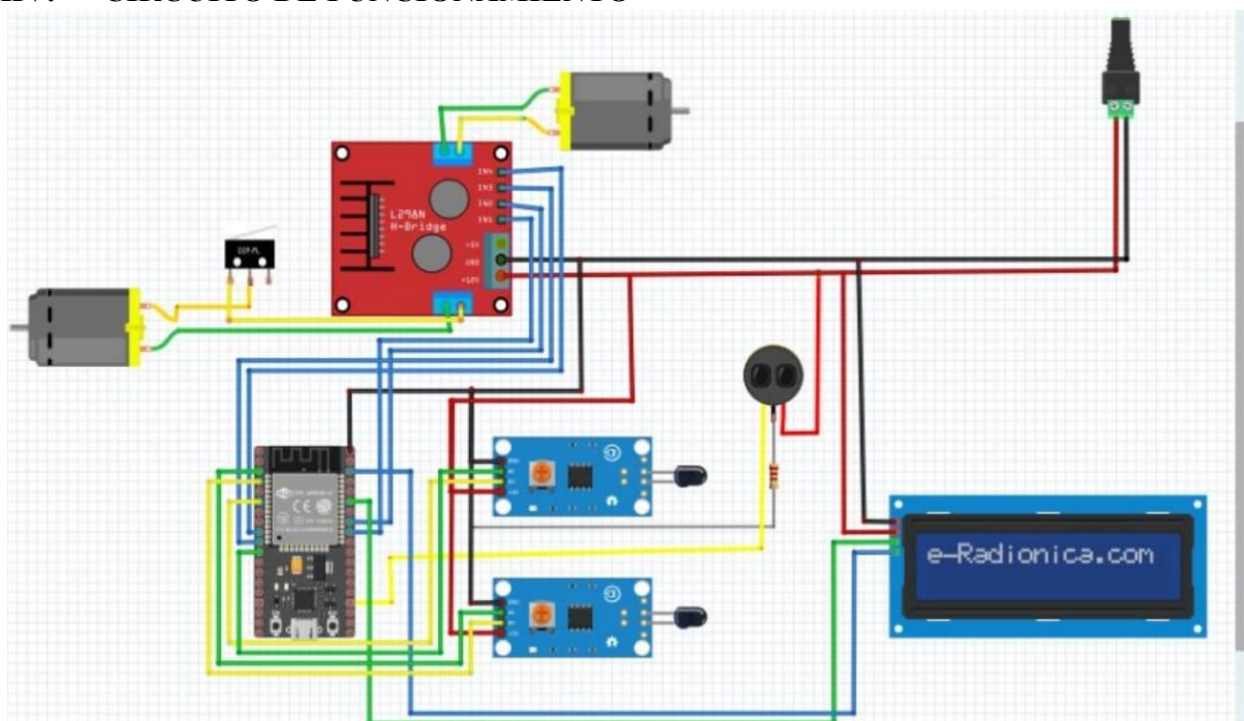


Ilustración 17. Interfaz blink en celular

### XIII. DIAGRAMA DE FLUJO FUNCIONAMIENTO

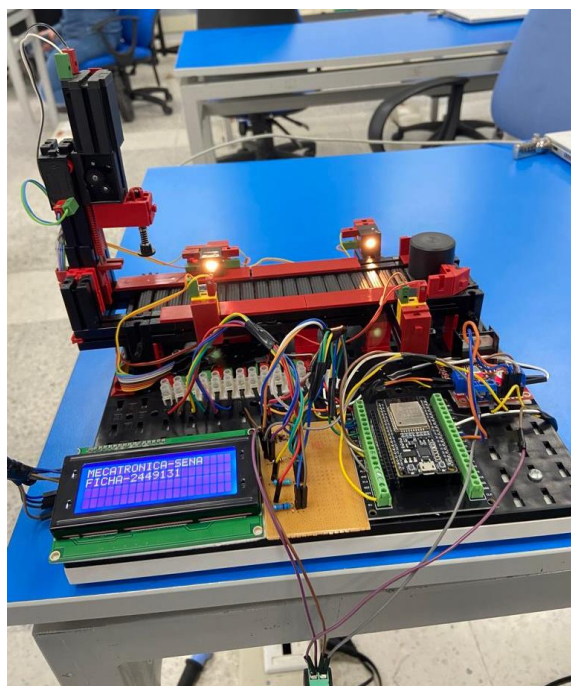


#### XIV. CIRCUITO DE FUNCIONAMIENTO



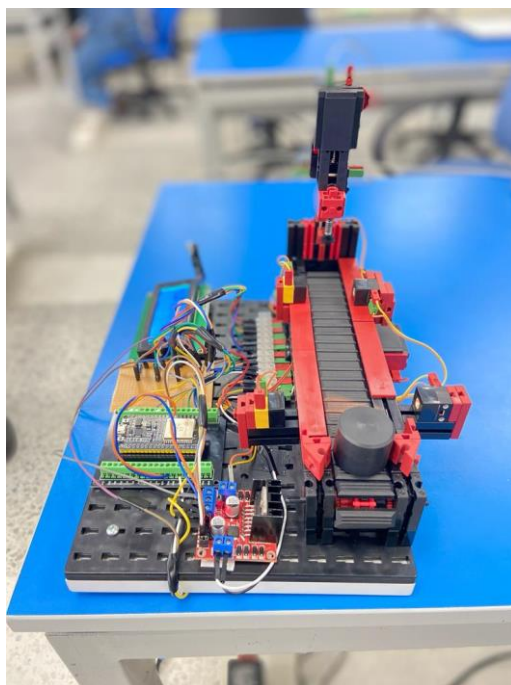
*Ilustración 18. Circuito del prototipo*

#### XV. RESULTADOS

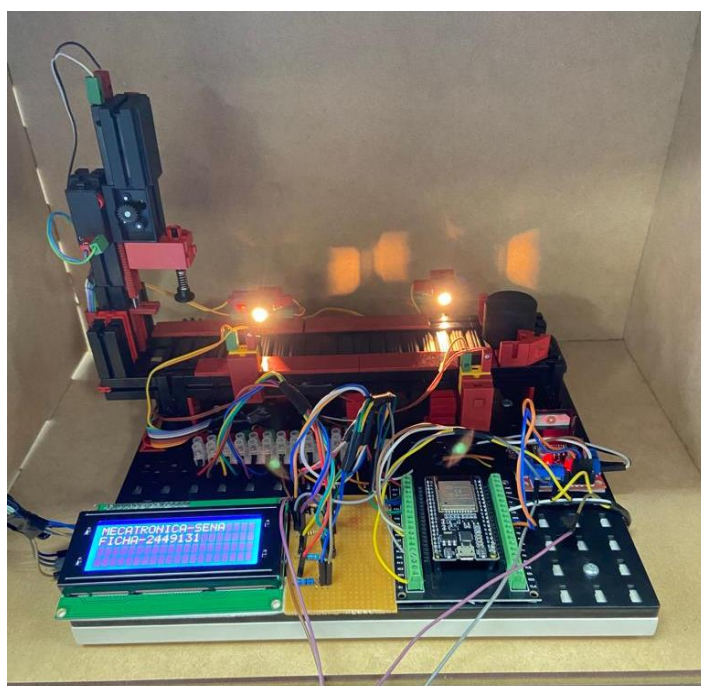


*Ilustración 19. Prototipo final*





*Ilustración 20. Prototipo final*



*Ilustración 21. Prototipo final*



Ilustración 22. Prototipo final

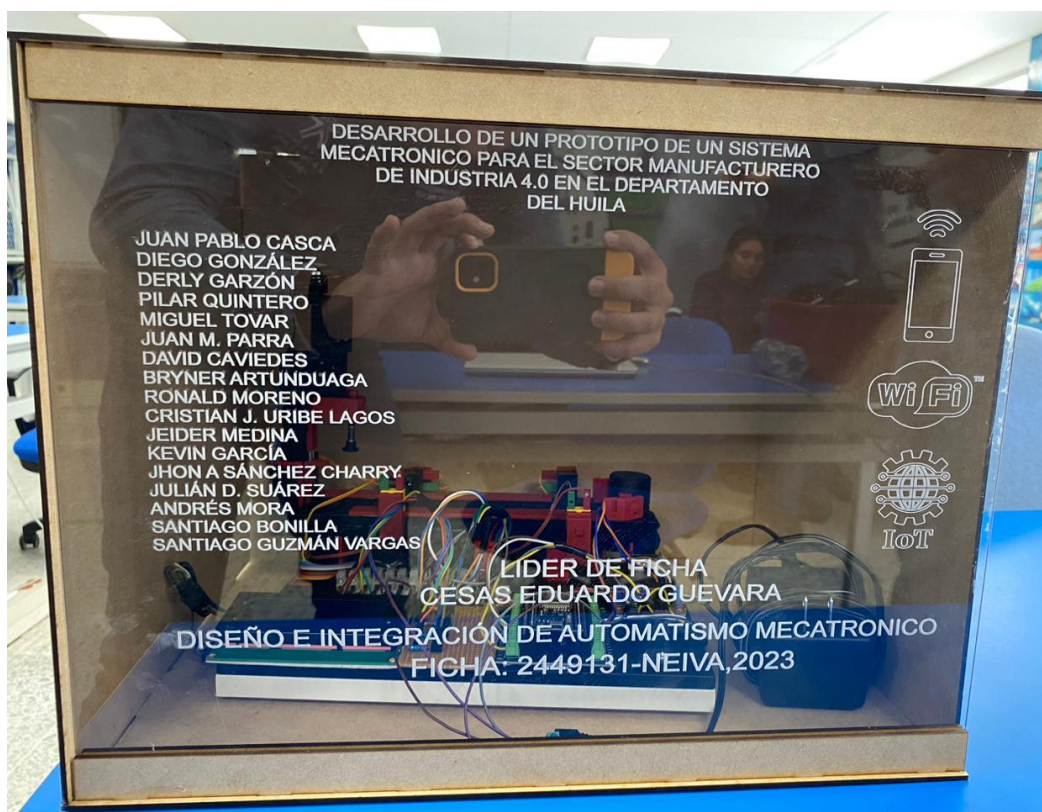


Ilustración 23. Prototipo final

[https://drive.google.com/file/d/1EHui3Dm-DL56sqeFga3E6Jic\\_SmFEo6j/view?usp=sharing](https://drive.google.com/file/d/1EHui3Dm-DL56sqeFga3E6Jic_SmFEo6j/view?usp=sharing)

## **XVI. CONCLUSIONES**

En conclusión, la ejecución exitosa de estos objetivos ha tenido un impacto significativo en el proyecto de la banda transportadora con punzador controlada por Internet de las Cosas (IOT). A través del uso de SolidWorks, se logró una representación precisa y detallada de los planos técnicos, sentando las bases para la fabricación y montaje del sistema con garantías de funcionamiento correcto.

El desarrollo y programación de los algoritmos en la plataforma ESP32 permitió alcanzar un control preciso y coordinado de todos los elementos de la banda transportadora, mejorando su rendimiento y asegurando un funcionamiento fluido y sin contratiempos.

La implementación de los códigos en la plataforma de IOT fue exitosa, facilitando una comunicación efectiva y un control remoto óptimo de la banda transportadora a través de la conexión a Internet. Esta integración de IOT brindó flexibilidad y accesibilidad en la supervisión y control del sistema, optimizando su eficiencia operativa y permitiendo una mayor interacción con el entorno digital.

Se ha prestado especial atención a las conexiones eléctricas y electrónicas de acuerdo con los estándares y normas relevantes. Esto ha asegurado una comunicación confiable y segura entre los diversos componentes y dispositivos del sistema, garantizando la integridad de las operaciones en su conjunto.

En resumen, la consecución de estos objetivos ha demostrado la viabilidad y el éxito del proyecto, sentando las bases para futuros avances en el campo de las bandas transportadoras con control IOT. Estos logros técnicos y tecnológicos representan un paso significativo hacia la mejora de los procesos industriales y la optimización de las operaciones logísticas en general. Además, han abierto nuevas posibilidades para la implementación de soluciones IOT en entornos

industriales, mejorando la eficiencia y la conectividad de los sistemas. El uso del software SolidWorks ha sido fundamental para el estudio, diseño y programación del sistema mecatrónico. Gracias a este programa, se ha logrado modelar y representar de manera precisa la máquina punzadora con cinta transportadora, considerando distintas vistas y planos.

En resumen, el prototipo desarrollado en este proyecto demuestra el potencial de la mecatrónica y la integración de tecnologías avanzadas en el sector manufacturero. Los resultados obtenidos son prometedores y abren oportunidades para futuras implementaciones y mejoras en los procesos de fabricación, contribuyendo así al avance de la industria 4.0 en el departamento del Huila.