LABORATORIO 3: PROGRAMACIÓN DE CELDAS MECATRÓNICAS FISCHERTECHNIK CON LA PLATAFORMA ILYZAELLE

---

1. Introducción:

En el presente laboratorio se abordará la programación y puesta en marcha de dos celdas mecatrónicas de Fischertechnik. La primera celda está diseñada para la clasificación de piezas según su color, empleando sensores ópticos que permiten identificar y dirigir cada objeto al contenedor correspondiente. La segunda celda realiza el transporte de piezas mediante un brazo accionado neumáticamente, lo cual implica el control de actuadores y electroválvulas.

---

2. Objetivos:

1. Aprender a programar en Izyaelle usando JavaScript.
2. Conectar las dos celdas con los controladores Xelorium.
3. Implementar la operación conjunta y sincronizada de ambas celdas de forma automática.

---

3. Requisitos

3.1 Requisitos Software:

1. Node.js (versión 20.x.x)
2. Arduino IDE (versión 1.18.x.x)
3. Python (versión 3.10.x.x)
4. Microsoft C++ Build Tools
5. ilyzaelleGatewayManager_0.1.bat

3.2 Requisitos Hardware:

1. 2x Controladores Xelorium
2. 1x Computadora Windows 11
3. 2x Celdas mecatrónicas **Fischertechnik** (clasificación por color y brazo neumático)

---

4. Marco Teorico:

4.1 Definiciones claves:

Celdas mecatrónicas didácticas:
Estaciones modulares utilizadas en la enseñanza técnica que simulan procesos industriales reales, permitiendo experimentar con sensores, actuadores y sistemas de control (Groover, 2007).

Controlador Xelorium:
Es un Arduino PLC educativo desarrollado para este proyecto, que permite la integración y control de sensores y actuadores. Está diseñado para facilitar la programación y monitoreo del sistema mediante entradas y salidas digitales/analógicas, y permite la comunicación entre distintas celdas del proceso automatizado.

JavaScript aplicado a automatización:
Lenguaje de programación flexible y dinámico que puede ser usado en entornos no web, como en sistemas educativos y de automatización, para manejar eventos, condiciones y flujos lógicos (Flanagan, 2020).

Plataforma Ilyzaelle:
Entorno de desarrollo creado para el presente proyecto, orientado al aprendizaje de la automatización mediante la programación de celdas mecatrónicas. Permite trabajar con lógica programada en JavaScript y conectarse a controladores como Xelorium.

4.2 Preguntas claves:

¿Qué es una celda mecatrónica y para qué sirve? Es una estación modular que simula un proceso industrial (como clasificar o mover piezas). Sirve para aprender control y automatización de forma práctica (Groover, 2007).

5. Procedimiento:

5.1. Configuración de la plataforma Ilyzaelle:

Para establecer la conexión entre los dos controladores, se sigue el mismo procedimiento utilizado en el Laboratorio 1: Introducción a Ilyzaelle. En esta etapa se configura tanto la plataforma como los dos controladores Xelorium que se emplearán en la práctica.

5.2. Programación de las celdas mecatronicas.

5.2.1 Declaración de Variables.

En la plataforma Ilyzaelle existen dos tipos de variables que se pueden crear:

- **Variables globales:** Son compartidas por todo el proyecto y pueden ser accedidas desde cualquier controlador conectado.

```
//VARIABLES GLOBALES
gVar[project].fototransistor = 0
```

Donde:

- **gVar["project"]** indica que la variable será global en todo el proyecto.
- **fototransistor** es el nombre de la variable.
- **= 0** establece su valor inicial.

- **Variables locales:** Sólo existen dentro del controlador donde fueron creadas y no pueden ser accedidas desde otros dispositivos.

```
// VARIABLES LOCALES
let voltage = 0
```

- **let:** Palabra clave que se usa para declarar una variable con un alcance (scope) local dentro del bloque donde se define. Permite que el valor de la variable pueda cambiar durante la ejecución del programa.
- **voltage:** Nombre de la variable que identifica el dato que se va a almacenar. En este caso, representa un valor de voltaje.

- **0:** Valor inicial que se asigna a la variable voltage. Aquí se inicia en cero.

5.2.2 Declaración de pines de Arduino

Para usar sensores y actuadores con Arduino, es importante saber en qué pin físico están conectados en la placa. Esto permite al programa identificar y controlar cada dispositivo correctamente.

```
// DECLARACION PINES ARDUINO
const fototransistor = 30;
```

- **const:** Palabra clave que declara una constante, es decir, un valor que no cambiará durante la ejecución del programa.
- **fototransistor:** Nombre que damos al pin donde está conectado el sensor, para poder referirnos a él fácilmente en el código.
- **= 30:** Asignación del número del pin físico (en este caso, el pin número 30) al nombre fototransistor.

2.3 Declaración tipo de pin

En este apartado declaramos el tipo de pin, que puede ser entrada o salida digital, entrada analógica o salida analógica (que en Arduino se implementa como PWM).

Dependiendo el tipo se declara:

```
board.pinMode(finalCarrera, board.MODES.INPUT);    //Entrada Digital
board.pinMode(sensorColor, board.MODES.ANALOG);    //Entrada Analoga
//motor1
board.pinMode(bandaL, board.MODES.OUTPUT);   //Salida Digital
board.pinMode(bandaR, board.MODES.OUTPUT);
board.pinMode(bandaPwm, board.MODES.PWM);    //Salida Analoga PWM
```

Procedimiento.

Para leer el estado de un sensor digital (como un final de carrera), se usa la función digitalRead de la siguiente forma:

Pegar el código *CODIGOLECTURADIGITAL.txt en el controlador de la Banda Transportadora*.

Ir al Dashboard y ver las variables globales.

Explicación del código:

- **board.digitalRead(...)** Esta función lee el valor (0 o 1) del pin definido como finalCarrera. Se ejecuta cada vez que el pin cambia su estado.
- **(value) => { ... }** Es una función flecha que se ejecuta automáticamente cuando se detecta un cambio en el pin. El valor leído (0 o 1) se recibe como value.
- **gVar["project"].finalCarrera = value;** Guarda el valor leído en una variable global llamada finalCarrera, para que pueda ser usada en cualquier parte del proyecto.

Para leer un valor analógico desde un sensor (como un sensor de color), se utiliza la función `analogRead` de la siguiente forma:

Pegar el código *CODIGOLECTURAANALOGO.txt en el controlador de la Banda Transportadora*.

Ir al Dashboard y ver las variables globales.

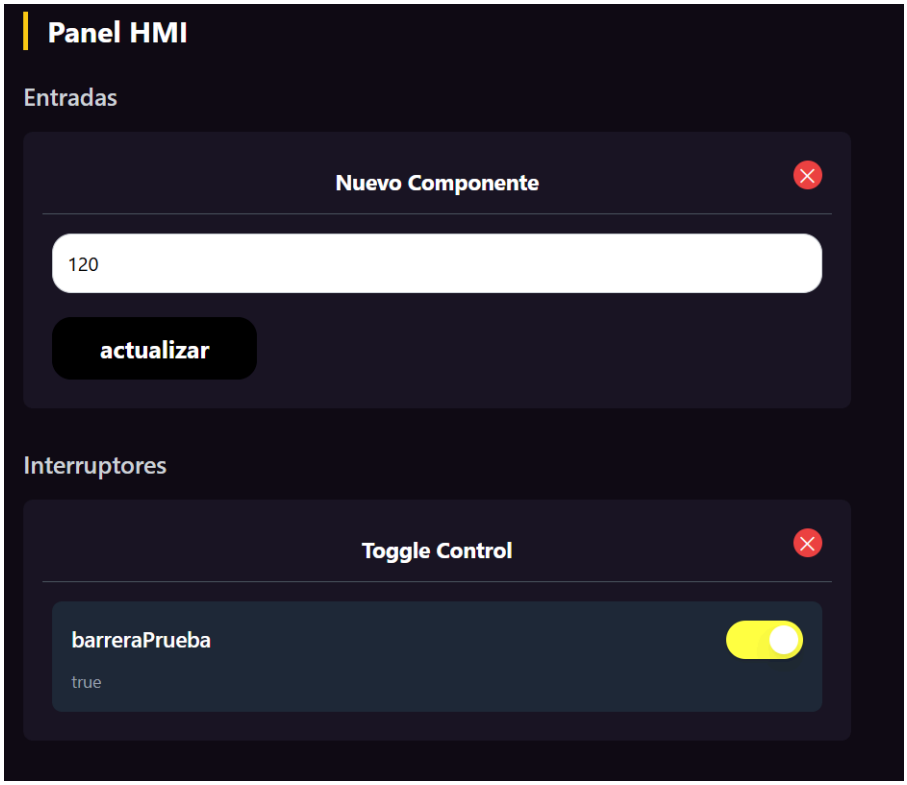
Explicación del código:

- **board.analogRead(...)**:
  Lee un valor analógico desde el pin conectado al sensor. Este valor varía entre 0 y 1023 dependiendo de la señal analógica que reciba (por ejemplo, de un sensor de color o luz).
- **sensorColor**:
  Es el nombre de la constante que representa el pin donde está conectado el sensor.
- **(value) => { ... }**:
  Función flecha que recibe el valor leído y realiza operaciones con él.
- **value.toFixed(2)**:
  Redondea el valor leído a dos decimales para mayor precisión en la conversión.
- **(value / 1023) * 10**:
  Convierte el valor leído al rango de voltaje (0–5 V), ajustado al factor de escala usado en el proyecto. *(Nota: comúnmente sería * 5, pero si estás usando * 10, asegúrate de que sea por diseño de tu sistema).*
- **gVar["project"].sensorColor = voltage;**:
  Guarda el valor del voltaje en una variable global, para que se pueda usar en cualquier parte del programa.

Cuando se desea mover un motor, se pueden usar tres pines conectados a un **puente H**: dos pines digitales para controlar el sentido de giro, y un pin PWM (salida analógica) para regular la velocidad.

Copiar el código del puente h *CODIGOPUENTEH.txt en el controlador de la BandaTransportadora.*

Desde el panel del Dashboard, se deben agregar al HMI las variables globales necesarias. En este caso, se inserta un input para la variable bandaVelocidad, que permitirá modificar la velocidad de la banda transportadora, y un toggle para la variable barreraPrueba, que habilita activar o desactivar la barrera según se requiera.



Explicación:

- **bandaL y bandaR**: Son los pines digitales que controlan la dirección del motor.
- **bpwm**: Es el valor PWM (de 0 a 255) que define la velocidad del motor.
- **digitalWrite(pin, valor)**: Escribe un 0 (apagado) o 1 (encendido) en un pin digital.
- **analogWrite(pin, valor)**: Envía una señal PWM al pin correspondiente, lo que permite regular la potencia (velocidad del motor).

Importante:

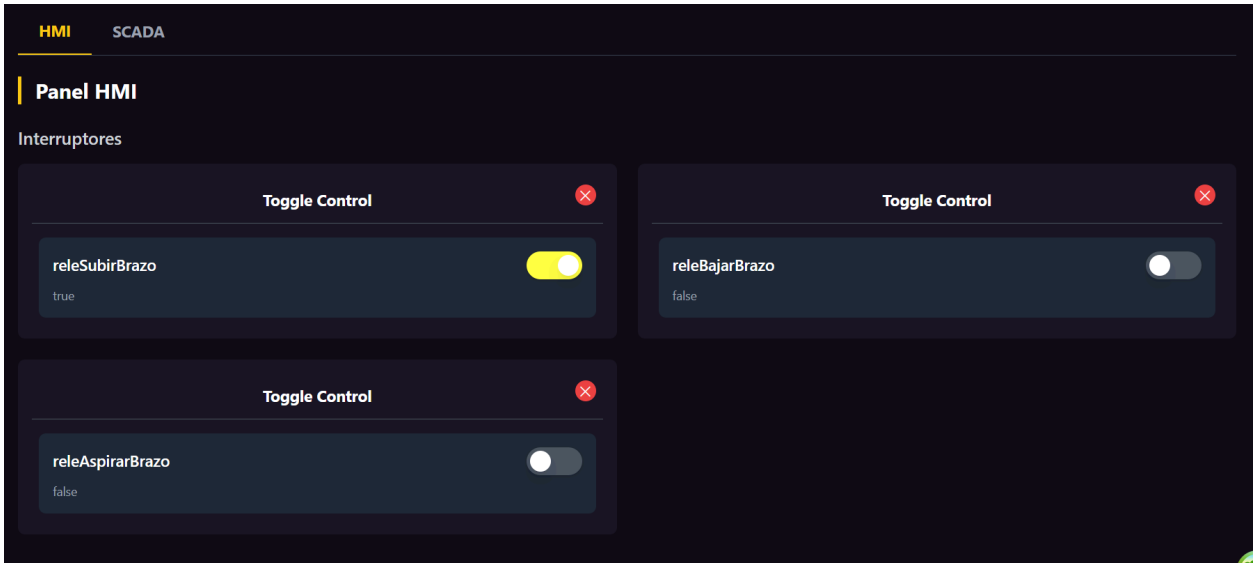
Nunca se deben activar ambos pines digitales (bandaL y bandaR) al mismo tiempo con valor 1. Esto podría generar un **cortocircuito** en el puente H y dañar el componente o el microcontrolador.

Para controlar un **relé**, se utiliza la función de escritura digital en el pin al que está conectado. Esta escritura define si el relé permite o no el paso de corriente:

Pegar Código de Relés *CODIGORELES.txt en el controlador del Brazo Neumático.*

Desde el Dashboard, se deben incorporar al HMI las variables globales correspondientes a los tres relés del sistema, utilizando el componente **toggle** para cada uno.

La activación o desactivación de estos toggles permitirá controlar directamente las electroválvulas asociadas, permitiendo así gestionar el funcionamiento neumático del sistema según las necesidades del proceso.



Explicación:

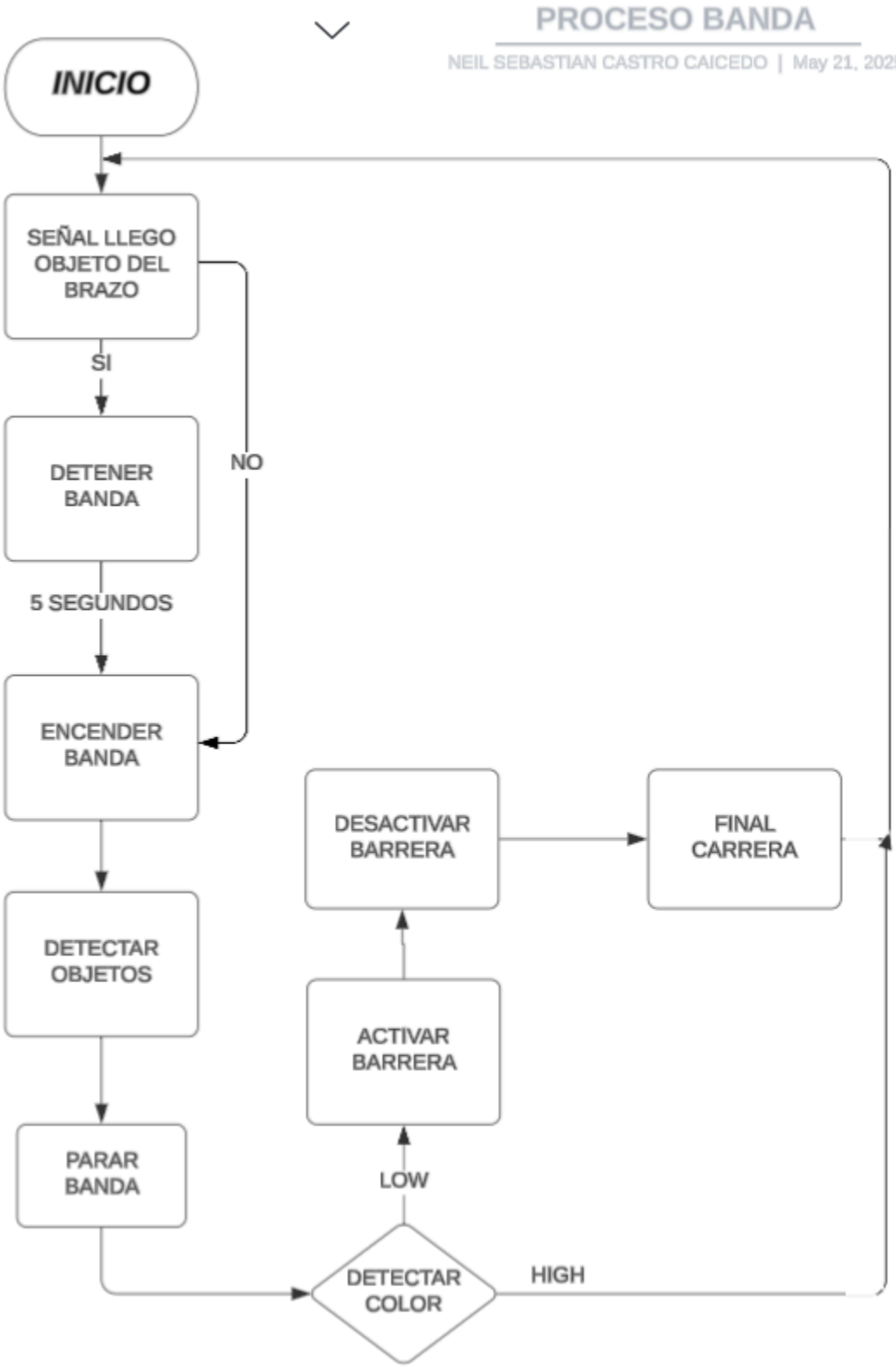
- Un valor **1** en el pin digital activa el relé y permite que el voltaje pase.
- Un valor **0** desactiva el relé y corta el paso del voltaje.

Importante:

No debes activar continuamente el relé si ya está encendido. Repetir la activación innecesariamente puede causar desgaste prematuro del componente o errores en el sistema.

Proceso General de las celdas Mecatrónica FischerTechnik.

Proceso General Banda Transportadora.



Explicación del código general de la banda transportadora.

VARIABLES GLOBALES

```
//VARIABLES GLOBALES
gVar[project].fototransistor = 0
gVar[project].finalCarrera = 0
gVar[project].sensorColor = 0
gVar[project].bandaVelocidad = 120
gVar[project].banderaBanda = 0
gVar[project].contBlanco = 0
gVar[project].contNegro = 0
```

VARIABLES LOCALES

```
// VARIABLES LOCALES
let voltage = 0
let isFototransistor = true
let isBarrera = false
let isBlanco = false
let isNegro = false
let contadorBlanco = 0;
let contadorNegro = 0;
```

DECLARACIÓN PINES

```
// DECLARACION PINES ARDUINO
const fototransistor = 30;
const finalCarrera = 32;
const sensorColor = 0;
const bandaL = 35;
const bandaR = 37;
const bandaPwm = 6;
const clasificadorL = 31;
const clasificadorR = 33;
const clasificadorPwm = 5;
```

TIPO DE PIN

```
// DECLARACION TIPO DE PIN
//sensores
board.pinMode(fototransistor, board.MODES.INPUT);
board.pinMode(finalCarrera, board.MODES.INPUT);    //Entrada Digital
board.pinMode(sensorColor, board.MODES.ANALOG);    //Entrada Analoga
//motor1
board.pinMode(bandaL, board.MODES.OUTPUT);   //salida Digital
board.pinMode(bandaR, board.MODES.OUTPUT);
board.pinMode(bandaPwm, board.MODES.PWM);    //Salida Analoga PWM
//motor2
board.pinMode(clasificadorL, board.MODES.OUTPUT);
board.pinMode(clasificadorR, board.MODES.OUTPUT);
board.pinMode(clasificadorPwm, board.MODES.PWM);
```

LECTURA DE SENSORES

```
// CODIGO LECTURA PINES
//sensores
board.digitalRead(finalCarrera, (value) => {
  gVar[project].finalCarrera = value;
});
board.digitalRead(fototransistor, (value) => {
  gVar[project].fototransistor = value;
});

board.analogRead(sensorColor, (value) => {
  voltage = (value.toFixed(2) / 1024) * 10; // Conversión a rango 0 5V
  gVar[project].sensorColor = voltage;
  isBlanco = (gVar[project].sensorColor >= 2.8 && gVar[project].sensorColor < 6.7) ? true : false;
  isNegro = (gVar[project].sensorColor >= 6.71 && gVar[project].sensorColor <= 9) ? true : false;
});
//motores
moverBarrera(0, 1, 0)
```

FUNCIONES PARA MOTORES

```
// CODIGO
function moverMotor(bpwm) {
  board.digitalWrite(bandaL, 0);
  board.digitalWrite(bandaR, 1);
  board.analogWrite(bandaPwm, bpwm);
  // Código para mover el motor
}
function moverBarrera(barreraL, barreraR, barreraPwm) {
  board.digitalWrite(clasificadorL, barreraL);
  board.digitalWrite(clasificadorR, barreraR);
  board.analogWrite(clasificadorPwm, barreraPwm);

}
```

La banda se desactiva cuando una pieza se aproxima desde el brazo.

```
setInterval(() => {
  moverMotor(gVar[project].bandaVelocidad)
  if (gVar[project].banderaBrazoaBanda === true) {
    gVar[project].bandaVelocidad = 0
    gVar[project].banderaBrazoaBanda = false
    setTimeout(() => {
      gVar[project].bandaVelocidad = 120
    }, 4000, { _id });
    return
  }
```

Al detectar un objeto, se detiene la banda, se identifica el color del objeto y se clasifica en función del mismo.

```
if (gVar[project].fototransistor === 0 && isFototransistor === true && gVar[project].banderaBrazoaBanda === false) {
  isFototransistor = false
  gVar[project].bandaVelocidad = 0
  console.log("OBJETO")
  console.log(gVar[project].sensorColor)

  if (isBlanco === true) {
    console.log("hola")
    isBlanco = false
    setTimeout(() => {
      moverBarrera(1, 0, 85)
      gVar[project].bandaVelocidad = 120
      isBarrera = true
      console.log("holAA")
      contadorBlanco++
      gVar[project].contBlanco = contadorBlanco
    }, 1300, { _id });
    return
  }
  if (isNegro === true) {
    console.log("adios")
    isNegro = false
    setTimeout(() => {
      isBarrera = true
      gVar[project].bandaVelocidad = 120
      moverBarrera(0, 1, 85)
      contadorNegro++
      gVar[project].contNegro = contadorNegro
    }, 1300, { _id });
    return
  }
```

Se reinician las banderas una vez finalizado el proceso principal.

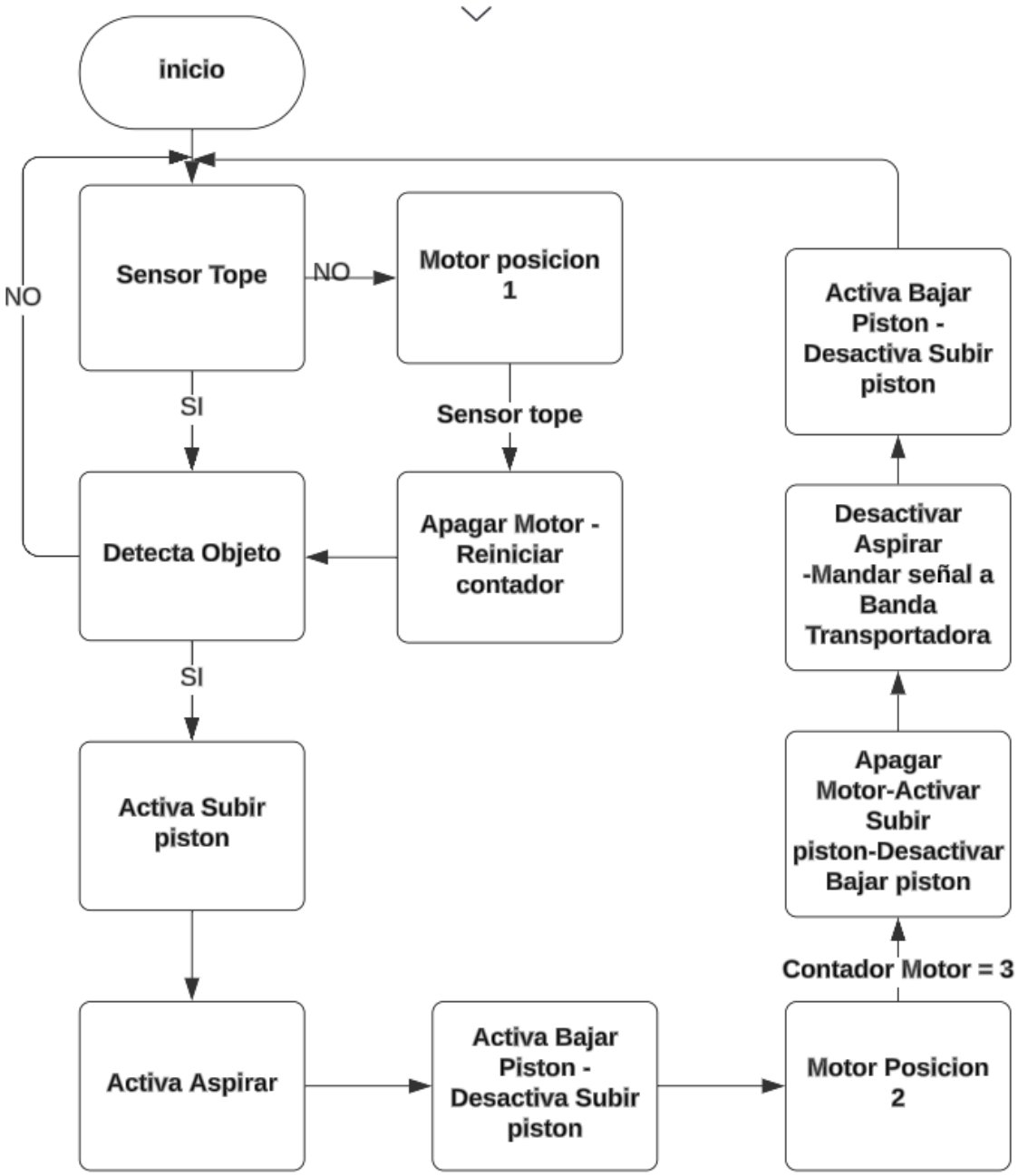
```
  if (isBarrera === true) {
    isBarrera = false
    setTimeout(() => {
      isFototransistor = true
      moverBarrera(1, 0, 0)
    }, 1500, { _id });
    return

  }

}, 100, { _id });
```

Copiar código del CODIGOBANDA.txt y pegar en el controlador de la banda transportadora.

Proceso General Brazo Neumático.



Explicación del código general de la banda transportadora.

VARIABLES GLOBALES

```
//Variables Globales
gVar[project].sensorMotor = 0
gVar[project].sensorTope = 0
gVar[project].fototransistorBrazo = 0
gVar[project].contadorPiezas = 0
gVar[project].arraycontadorPiezas = []
gVar[project].banderaBrazoaBanda = false
//Banderas
gVar[project].rtFototransistorBrazo = false
gVar[project].rtPiezaNueva = false
gVar[project].rtreleAspirar = false
gVar[project].rtreleBajar = false
gVar[project].rtreleSubir = false
```

VARIABLES LOCALES

```
//Variables
let banderaPrueba = true; // BANDERA REINICIO
let isCero = false;
let isUno = false;
let isDos = false;
let isTres = false;
let isCuatro = false;
let isCinco = false;
let isSeix = false;
let isSiete = false;
let estadoAnterior = 0; // Estado anterior del interruptor
let contador = 0; // Contador de cambios de 0 a 1
let contadoPiezas = 0;
```

DECLARACIÓN PINES

```
//declarar pines
//Reles
const releAspirar = 39;
const releSube = 41;
const releBaja = 43;
//Sensores 24v
const sensorMotor = 32;
const sensorTope = 30;
const fototransistor = 28;
// motor
const bandaL = 35;
const bandaR = 37;
const bandaPwm = 6;
```

TIPO DE PIN

```
//DEFINIPINES
// modo pines
board.pinMode(releAspirar, board.MODES.OUTPUT);
board.pinMode(releSube, board.MODES.OUTPUT);
board.pinMode(releBaja, board.MODES.OUTPUT);
board.pinMode(sensorMotor, board.MODES.INPUT);
board.pinMode(sensorTope, board.MODES.INPUT);
board.pinMode(fototransistor, board.MODES.INPUT);
//motor1
board.pinMode(bandaL, board.MODES.OUTPUT);
board.pinMode(bandaR, board.MODES.OUTPUT);
board.pinMode(bandaPwm, board.MODES.PWM);
```

LECTURA DE SENSORES

```
//Codigo Inicial
//sensores
board.digitalRead(sensorMotor, (value) => {
  gVar[project].sensorMotor = value;
});
board.digitalRead(sensorTope, (value) => {
  gVar[project].sensorTope = value;
});
board.digitalRead(fototransistor, (value) => {
  gVar[project].fototransistorBrazo = value;
});
```

FUNCIONES PARA ACTUADORES

```
// CODIGO
//INICIAR
function moverMotorBrazo(brazoL, brazoR, brazoPwm) {
  board.digitalWrite(bandaL, brazoL);
  board.digitalWrite(bandaR, brazoR);
  board.analogWrite(bandaPwm, brazoPwm);
}
function relesSB(rSube, rBaja) {
  board.digitalWrite(releSube, rSube);
  board.digitalWrite(releBaja, rBaja);
  gVar[project].rtreleBajar = (rBaja) ? true : false;
  gVar[project].rtreleSubir = (rSube) ? true : false;
}
function relesA(rAspira) {
  gVar[project].rtreleAspirar = (rAspira) ? true : false;
  board.digitalWrite(releAspirar, rAspira);
}
```

Realiza el conteo del sensor del motor y lleva el brazo a su posición inicial

```
//MAIN
setInterval(() => {

  if (estadoAnterior === 1 && gVar[project].sensorMotor === 0) {
    contador++;
    console.log(`Cambio detectado de 0 a 1. Contador: ${contador}`);
  }
  // Actualizar el estado anterior
  estadoAnterior = gVar[project].sensorMotor;

  if (gVar[project].sensorTope === 0 && banderaPrueba === true) {
    moverMotorBrazo(1, 0, 200)
  } else if (gVar[project].sensorTope === 1 && banderaPrueba === true) {
    banderaPrueba = false
    relesSB(0, 0)
    moverMotorBrazo(1, 0, 0)
  }
```

Al detectar un objeto, se inicia el proceso de traslado del punto A al punto B.

```
if (gVar[project].fototransistorBrazo === 0 && isCero === false && gVar[project].sensorTope === 1) {
  contador = 0
  relesSB(1, 0)
  isUno = true;
  isCero = true;
  console.log("pieza")
  return
}
if (isUno === true) {
  isUno = false
  setTimeout(() => {
    relesA(1)
    isDos = true;
    console.log("primer paso")
  }, 1000, { _id });
}
if (isDos === true) {
  isDos = false;
  setTimeout(() => {
    isTres = true
    relesSB(0, 1)
    console.log("segundo paso")
    return
  }, 1000, { _id });
}
```

Banderas para el SCADA y grafica de piezas en el tiempo.

```
setInterval(() => {

  if (gVar[project].fototransistorBrazo === 0) {
    gVar[project].rtFototransistorBrazo = true
  } else {
    gVar[project].rtFototransistorBrazo = false
  }
  if (gVar[project].banderaBrazoaBanda === true) {
    gVar[project].rtPiezaNueva = true
  } else {
    gVar[project].rtPiezaNueva = false
  }
}, 40, { _id });
setInterval(() => {
  gVar[project].arraycontadorPiezas.push(contadoPiezas)
}, 400, { _id });
```

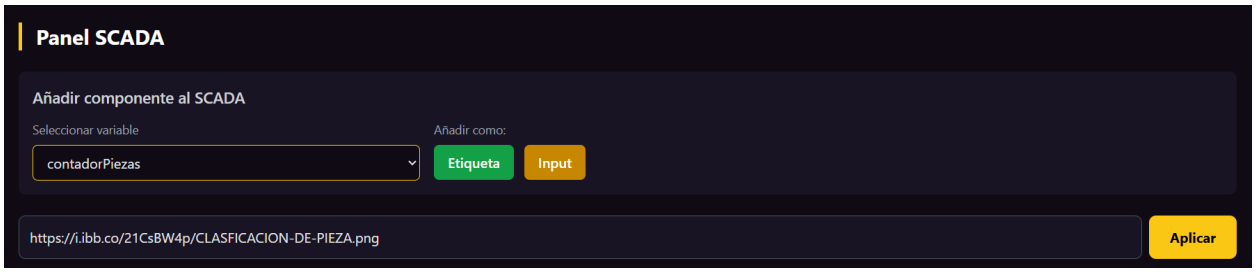
Copiar código de CODIGOBRAZOTRANSPORTADOR.txt y pegar en el controlador del Brazo Neumático.

SCADA DEL PROCESO

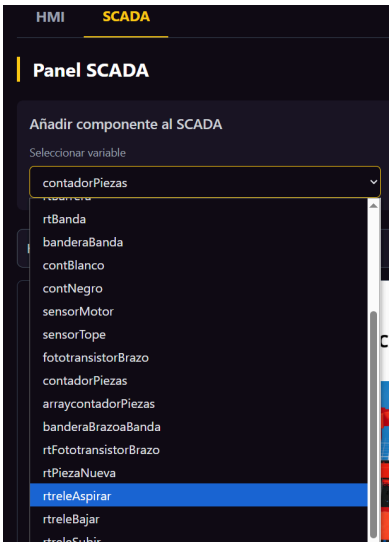
Para monitorear en tiempo real los sensores y actuadores, se decidió implementar un sistema SCADA que represente el proceso general. Este sistema abarca tanto el transporte como la clasificación de piezas. Para lograr un seguimiento adecuado de cada etapa del proceso, se utilizaron variables que actúan como banderas, permitiendo identificar el estado de cada parte del sistema.

```
//Banderas
gVar[project].rtFototransistorBrazo = false
gVar[project].rtPiezaNueva = false
gVar[project].rtreleAspirar = false
gVar[project].rtreleBajar = false
gVar[project].rtreleSubir = false
```

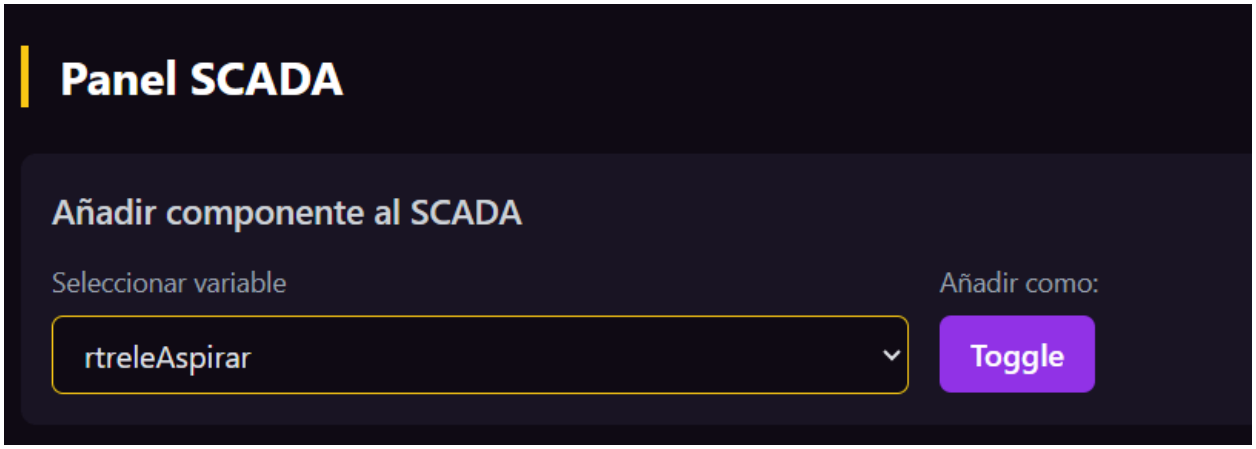
Para cargar una imagen en el SCADA, ubicado en el panel del Dashboard, es necesario copiar la dirección (URL) de la imagen. En este caso, se insertará la imagen del proceso general. Luego, se hace clic en Aplicar para guardar los cambios, como se muestra en la siguiente imagen:



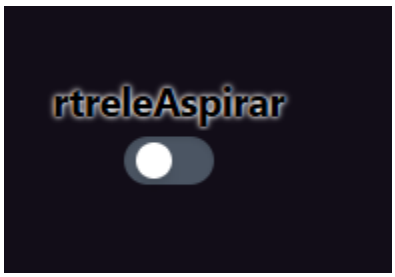
Para configurar la visualización en el sistema SCADA, nos dirigimos al panel ubicado en el Dashboard. Allí seleccionamos las variables que deseamos mostrar; en este caso, se eligió la variable rtReleAspirar, la cual indica el estado del relé encargado del sistema de aspiración.



Dependiendo del tipo de variable seleccionada, aparecerán diferentes opciones para añadirla al SCADA. En este caso, utilizaremos la opción "Añadir como Toggle", que permite ver si esta activa o desactiva una variable.

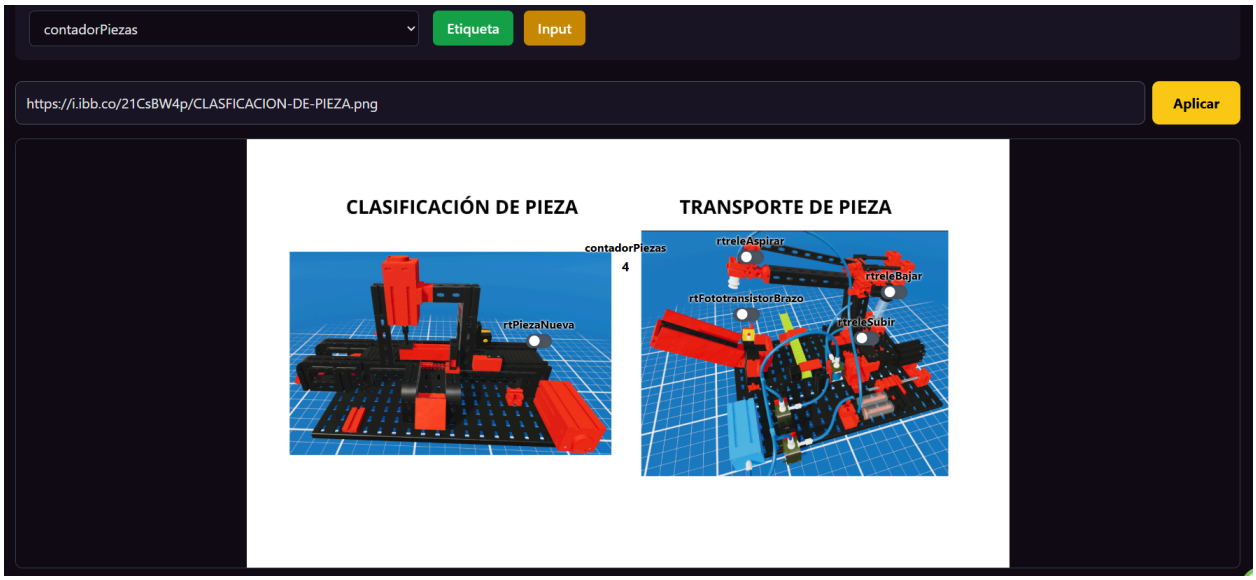


Al añadirlo como Toggle, este aparecerá en el panel del SCADA, donde podremos ajustar su tamaño, moverlo libremente o eliminarlo según sea necesario para organizar la interfaz de manera clara y funcional.



Lo siguiente es arrastrarlo hasta la ubicación correspondiente dentro del proceso. En este caso, se optó por colocarlo directamente sobre el componente que representa, facilitando así la visualización y el control del sistema en tiempo real.

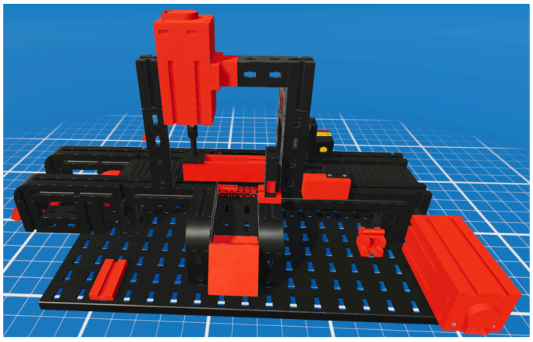
Se realizó un ejemplo utilizando el apartado de transporte de piezas, donde es posible observar el funcionamiento de los sensores y actuadores principales. Además, se implementó un contador llamado contadorPiezas, el cual incrementa su valor cada vez que una pieza es transportada exitosamente, permitiendo así un seguimiento en tiempo real del proceso.



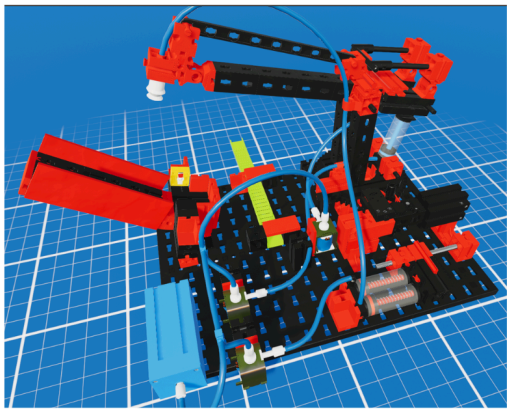
6. Actividad:

- Usando la función SCADA de la plataforma, implementa un panel que utilice las banderas y contadores necesarios para supervisar el correcto funcionamiento de la banda transportadora. El resultado debe ser similar al ejemplo presentado durante el desarrollo del laboratorio. Usa como fondo la imagen del proceso general disponible en el siguiente enlace: https://i.ibb.co/21CsBW4p/CLASFICACION-DE-PIEZA.png

CLASIFICACIÓN DE PIEZA



TRANSPORTE DE PIEZA



-

---

7. Bibliografía:

- Bolton, W. (2015). *Mecatrónica: sistemas de control electrónico en la ingeniería mecánica* (5.ª ed.). Pearson Educación.
- Groover, M. P. (2007). *Automatización, producción y manufactura* (3.ª ed.). Pearson Educación.
- Flanagan, D. (2020). *JavaScript: The Definitive Guide* (7.ª ed.). O'Reilly Media.
- Nise, N. S. (2015). *Control Systems Engineering* (7.ª ed.). Wiley.