

# Natural Language Industrial Control PLC Raspberry

Fabio Hernán Realpe Martínez  
Carlos Felipe Rengifo  
Victor Hugo Mosquera

UNIVERSIDAD DEL CAUCA  
INGENIERIA EN AUTOMATICA INDUSTRIAL  
POPAYAN - 2026



# Índice general

<b>1. Instalación de sistema operativo para PLCRaspberry de Industrial Shield</b>	<b>8</b>
1.1. Características del Plc . . . . .	8
1.2. Preparación Software del Plc . . . . .	9
1.2.1. Instalación del sistema operativo . . . . .	9
1.3. Configuración de Red . . . . .	12
1.4. Configuracion de SSH . . . . .	13
<b>2. Configuración e instalación de librerías</b>	<b>15</b>
2.1. Instalación de dependencias . . . . .	15
2.1.1. Instalación de NVM, Node.js y Yarn . . . . .	15
2.2. Descarga de paquetes y configuración . . . . .	16
2.2.1. Librerías librpiplc . . . . .	16
2.2.2. Compilación de programas en C . . . . .	18
2.2.2.1. Programas C . . . . .	18
2.2.3. Compilación de programas en Node . . . . .	24
2.2.3.1. Programas en Node.js . . . . .	24
2.2.4. Compilación de programas en Python . . . . .	31
2.2.4.1. Programas en Python . . . . .	31
2.3. Dependencias y paquetes de terceros. . . . .	36
2.3.1. Guía de instalación y configuración de Mosquitto MQTT Broker . . . . .	36
<b>3. Comparativa funcional entre control clásico y control IA</b>	<b>39</b>
3.1. Esquema para la implementación del controlador . . . . .	42
3.1.1. La arquitectura está compuesta por cinco capas funcionales: . . . . .	45
3.1.1.1. Capa de Usuario - Interfaz Web React . . . . .	45
3.1.1.2. Capa de Servidor y Agente Central . . . . .	46
<b>4. Capa Plc Físico</b>	<b>50</b>
4.1. IA: Caracterización . . . . .	50
4.1.1. Agente de Caracterización en la Capa Física del PLC . . . . .	51
4.1.2. Resultados Agente Caracterización . . . . .	56

4.1.3.	Promt: Caracteriza y utiliza como entrada a la planta una secuencia binaria aleatoria que oscile entre el 45 % y el 55 % de la señal de entrada -Captura 1000 muestras. . . . .	56
4.1.4.	Promt: Usa una señal entre 4 V y 8 V para caracterizar la planta, con 1000 muestras y un tiempo de muestreo 100 ms. . . . .	58
4.1.5.	Promt: Excita la planta con una secuencia que oscile entre el 10 % y el 20 %. . . . .	60
4.2.	Conclusiones del capítulo . . . . .	62
<b>5.</b>	<b>Identificación</b>	<b>64</b>
5.1.	IA: Identificación . . . . .	67
5.2.	Etapas del proceso de identificación de sistemas asistida por IA . . . . .	68
5.3.	Resultados Agente de Identificación . . . . .	79
5.3.1.	Promt: Identifica la planta . . . . .	79
5.3.2.	Promt: Identifica la planta tomando de la base de datos el segundo id (orden 2) . . . . .	80
5.3.3.	Promt: Identifica la planta tomando de la base de datos el segundo id (orden 3) IA . . . . .	82
<b>6.</b>	<b>Controlador</b>	<b>85</b>
6.1.	Resultados Agente de ControlIA . . . . .	91
6.1.1.	Promt: Controlador IA con set point de 6v y 60 s. . . . .	91
6.1.2.	Promt: Controlador IA con set point de 6v y 60 s, disturbio. . . . .	92
6.2.	Conclusiones . . . . .	93
6.3.	Trabajos Futuros . . . . .	93

# Índice de figuras

1.1.	Raspberry PLC 58 Industrial Shields . . . . .	9
1.2.	Sistemas operativo para Plc Raspberry . . . . .	10
1.3.	Software para grabar el sistema operativo en Sd Card . . . . .	10
1.4.	Seleccionar imagen . . . . .	11
1.5.	Volumen seleccionado . . . . .	11
1.6.	Herramientas de configuración . . . . .	12
1.7.	Habilitación de red inalámbrica . . . . .	13
1.8.	Ssh . . . . .	13
2.1.	Planta para controlador PI . . . . .	35
3.1.	Diagrama general de la arquitectura. . . . .	44
3.2.	Interfaz de usuario. . . . .	45
3.3.	Agente IA . . . . .	48
4.1.	IA caracterización . . . . .	52
4.2.	Agente de caracterización . . . . .	53
4.3.	Caracterización 5000 muestras, secuencia binaria aleatoria entre el 45 % y el 55 % de la señal de entrada, tiempo de muestreo 50 ms. . . . .	58
4.4.	Caracterización 1000 muestras, señal entre 4 V y 8 V tiempo de muestreo 100 ms. . . . .	60
4.5.	Excitación de la planta con una secuencia que oscila entre el 10 % y el 20 %. . . . .	62
5.1.	Modelo agente IA . . . . .	66
5.2.	Natural language interpretation pipeline for plant identification . . . . .	69
5.3.	Preprocesamiento de datos y agente IA Plc . . . . .	71
5.4.	Modelos Candidatos . . . . .	76
5.5.	Diagrama de Flujo para la Ejecución de Pruebas de Comparación. . . . .	78
5.6.	Comparación modelo planta y modelo de orden . . . . .	80
5.7.	Comparacion modelo planta y modelo de orden 2 . . . . .	82
5.8.	Comparacion modelo planta y modelo de orden 3 . . . . .	84
6.1.	Control Orchestration Workflow of the AI-Assisted Edge Controller. . . . .	87

6.2. Time-Domain Response of the System under AI-Assisted PI Control . . . . .	90
6.3. Closed-Loop Response with AI-Assisted PI Controller (Setpoint = 6 V, No Disturbance) . . . . .	91
6.4. Disturbance Rejection Test with AI-Assisted PI Controller (Setpoint = 6 V) . . . . .	92

# Algoritmos

2.1.	Gpio.cpp . . . . .	20
2.2.	AdcPwm.cpp . . . . .	21
2.3.	ControladorPI.cpp . . . . .	23
2.4.	binding.gyp . . . . .	25
2.5.	rpiplc_wrapper.cpp . . . . .	26
2.6.	Gpio.js . . . . .	28
2.7.	Pwm_adc.js . . . . .	29
2.8.	ControlPI.js . . . . .	30
2.9.	Gpio.py . . . . .	32
2.10.	Pwm_adc.py . . . . .	33
2.11.	ControlPi.py . . . . .	34
3.1.	Algoritmo Servidor IA. . . . .	49
4.1.	Algoritmo Caracterización IA. . . . .	55
5.1.	Data Preprocessing. . . . .	72
5.2.	Mathematical ARX Identification (Fallback). . . . .	73
5.3.	IA identificación. . . . .	75
5.4.	Algoritmo de Comparación Planta-Modelo . . . . .	78
5.5.	Identificación con modelo Orden 1. . . . .	79
5.6.	Identificación con modelo Orden 2. . . . .	81
5.7.	Identificación con modelo Orden 3. . . . .	83
6.1.	AI-Orchestrated Control Execution Pipeline. . . . .	88
6.2.	AI-Orchestrated Control Execution Pipeline. . . . .	89

# Capítulo 1

## Instalación de sistema operativo para PLCRaspberry de Industrial Shield

### 1.1. Características del Plc

El PLC utilizado para el desarrollo de la presente guía es el Raspberry PLC 58 v6 de Industrial Shields, cuyo hardware está basado en una Raspberry Pi 4B (2 GB RAM). Puede consultar el equipo en: IndustrialShield.

Las características principales del PLC son las siguientes:

- Entradas:
  - 39 entradas en total
  - 18 entradas analógico/digital
  - 21 entradas digitales aisladas (6 con interrupción)
- Salidas:
  - 24 salidas en total
  - 15 salidas digitales aisladas
  - 9 salidas convertibles (digital aislada / PWM / analógica)
- Comunicaciones y módulos:
  - I2C, RS485 (FD), Ethernet (x2), WiFi, BLE
  - Serial TTL, SPI
  - UPS, RTC, ranura para tarjeta microSD
  - Compatibilidad opcional: LoRa, DALI, GPRS-4G, CAN, GPS

Gracias a que este PLC está basado en una Raspberry Pi, utiliza como sistema operativo Linux (Raspberry Pi OS). Esto permite aprovechar todas las capacidades de una micro-computadora, facilitando la integración de herramientas avanzadas como sistemas de inteligencia artificial, servidores, bases de datos, interfaces gráficas y comunicación industrial ver figura 1.1.



Figura 1.1: Raspberry PLC 58 Industrial Shields

## 1.2. Preparación Software del Plc

En esta sección se describirá el procedimiento completo para la instalación de las herramientas necesarias sobre el Plc, esta descripción está basada en pasos secuenciales necesarios para tener un correcto funcionamiento de los entornos de desarrollo.

### 1.2.1. Instalación del sistema operativo

Como se ha descrito anteriormente la Raspberry utiliza un sistema operativo basado en la distribución de (Raspberry Pi OS) las imágenes disponibles para dicho Plc se encuentra en el siguiente repositorio RaspberryIOS, ver figura 1.2.

<a href="#">File Name</a>	<a href="#">File Size</a>	<a href="#">Date</a>
Old/	-	-
2022-01-28-raspios-bullseye-armhf-lite-raspberry-plc-v4-20230505150414.zip	573.2 MiB	2024-Aug-29 12:25
2024-03-15-raspios-bookworm-arm64-lite-raspberry-plc-v4-20240827131336.zip	839.0 MiB	2023-May-05 13:35
2024-03-15-raspios-bookworm-arm64-lite-raspberry-plc-v6-20240606100822.zip	821.7 MiB	2024-Aug-29 12:19
2024-07-04-raspios-bookworm-arm64-desktop-raspberry-plc-v6-20240916163624.zip	1.9 GiB	2024-Aug-01 11:51
		2024-Sep-17 10:06

Figura 1.2: Sistemas operativo para Plc Raspberry

Para nuestro propósito se utilizará la imagen Raspberry Pi OS Lite (821.7 MB), debido a que es mucho más liviana que la imagen completa de 1.9 GB, la cual incluye un entorno de escritorio recomendado para usuarios sin experiencia en Linux.

Es importante aclarar que la tarjeta microSD debe ser de al menos 8 GB y clase 10. En nuestro caso, debido a que se implementarán varias aplicaciones que requieren un mayor consumo de recursos, se utilizará una tarjeta de 32 GB clase 10.<sup>1</sup>.

Lo primero es dirigirse a Raspberry Imager y descargar para el sistema operativo anfitrión la versión correspondiente, para nuestro caso se selecciona la versión de Linux debido a que el sistema operativo es Ubuntu, ver 1.3



Figura 1.3: Software para grabar el sistema operativo en Sd Card

Se debe seleccionar Choose Device y esto lo llevara a un menú como se muestra en la siguiente figura ver 1.4.

---

<sup>1</sup>clase 10 escribe datos a una velocidad mínima de 10 MB/s (megabytes por segundo).

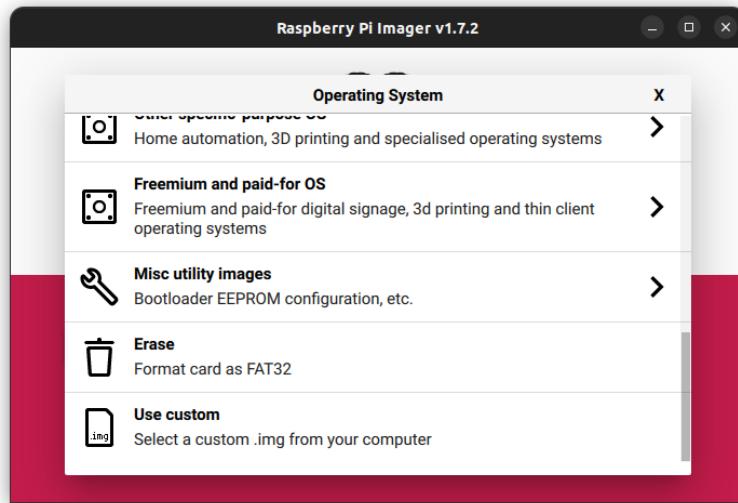


Figura 1.4: Seleccionar imagen

Una vez seleccionada la imagen damos clic en el ícono Operating System y se selecciona la sd card previamente insertada, ver 1.5.

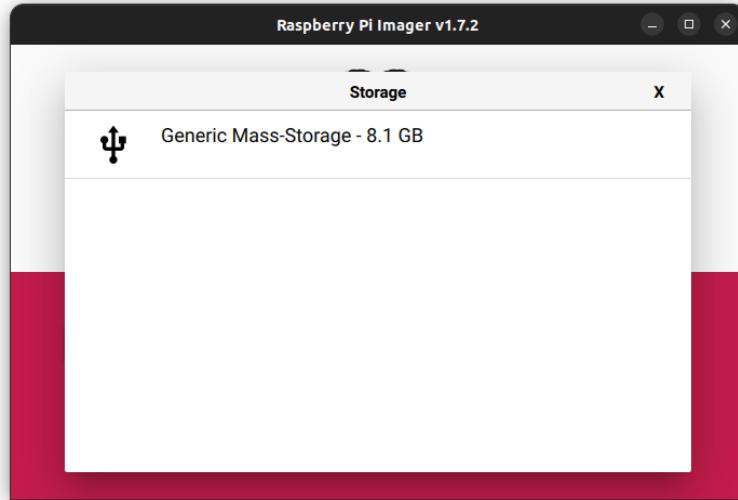


Figura 1.5: Volumen seleccionado

Una vez seleccionada la opción Write, comenzará el proceso de escritura de la imagen en la tarjeta SD. Al completar la grabación, será necesario conectar el PLC a un monitor HDMI, y disponer de un teclado y mouse, con el fin de llevar a cabo las configuraciones iniciales del entorno de trabajo.

### 1.3. Configuración de Red

Antes de iniciar con la configuración de red, es importante aclarar que el PLC requiere una alimentación externa de 12 a 24 V con una capacidad mínima de 2.0 Amp para poder arrancar correctamente. Aunque la tarjeta cuenta con un puerto USB Tipo-C de 5 V, este no es suficiente para iniciar el sistema operativo ni para garantizar el funcionamiento estable del equipo. Por lo tanto, la alimentación debe realizarse obligatoriamente a través de la entrada de voltaje principal.

Al iniciar por primera vez nos encontraremos con una sesión. El usuario y contraseña son por defecto los dados por la imagen del sistema operativo

```
login: pi  
password: raspberry  
pi@raspberrypi:~ $
```

Un vez iniciada la sesión procederemos a ejecutar el siguiente comando

```
pi@raspberrypi:~ $ sudo raspi-config
```

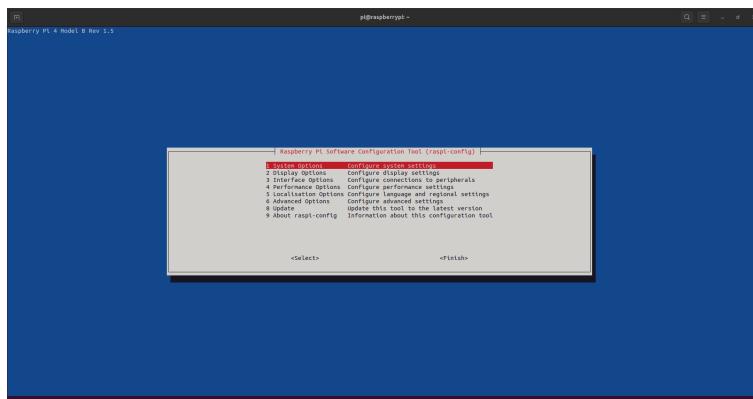


Figura 1.6: Herramientas de configuración

En el menú seleccionamos System Options y, posteriormente, configuramos el nombre de la red (SSID) a la cual deseamos conectarnos. De igual manera, el sistema solicitará la contraseña de la red para completar la conexión. Véase la figura 1.7.

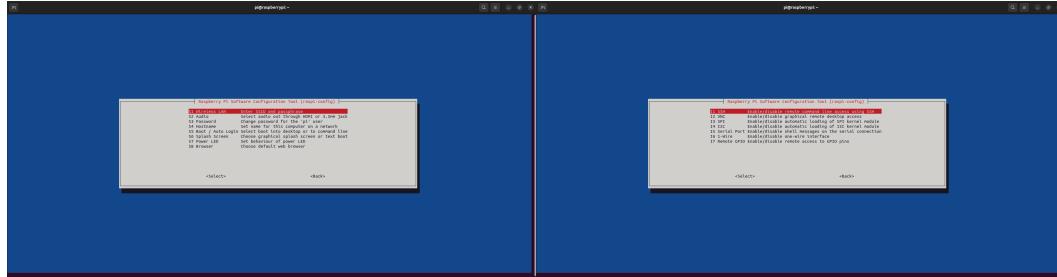


Figura 1.7: Habilitación de red inalámbrica

## 1.4. Configuración de SSH

La habilitación del servicio SSH permite acceder al PLC de manera remota desde otro equipo a través de la red, utilizando una terminal o cliente SSH. Para activarlo, ingresamos al menú de configuración del sistema, seleccionamos Interface Options y luego SSH, donde confirmamos su habilitación. Una vez activo, el PLC aceptará conexiones remotas utilizando su dirección IP, lo que facilita la administración y el control del dispositivo sin necesidad de mantener teclado, mouse o monitor conectados físicamente.

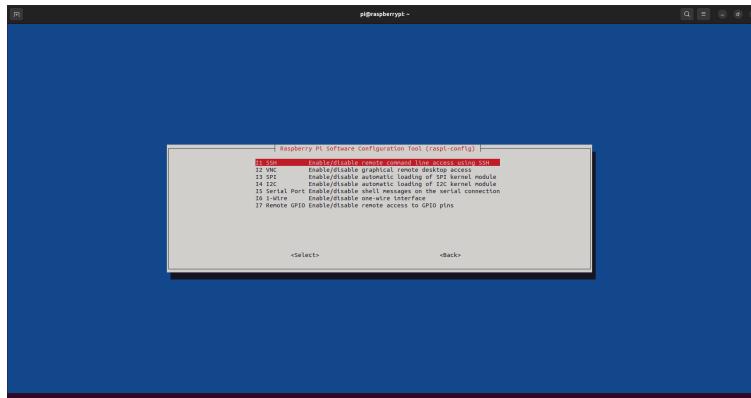


Figura 1.8: Ssh

Ahora vamos a ejecutar el comando

```
pi@raspberrypi:~ $ sudo hostname -I
10.10.10.20 10.10.11.20 172.24.36.193 172.22.0.1 172.17.0.1
```

El PLC se encuentra conectado a la red Wi-Fi y ha recibido la dirección IP 172.24.36.193. Si el computador desde el cual se desea acceder se encuentra conectado a la misma red inalámbrica, es posible establecer la conexión remota mediante SSH.

En sistemas Linux, la conexión puede realizarse directamente desde la terminal, utilizando el siguiente comando:

```
fabio@nicolas:~$ ssh pi@172.24.36.193
pi@172.24.36.193's password: raspberry
pi@raspberrypi:~ $
```

## Conclusión:

- Con los pasos realizados, se preparó el PLC para su funcionamiento inicial: se grabó correctamente la imagen del sistema operativo en la tarjeta SD, se realizó la primera puesta en marcha con la alimentación adecuada y se configuraron los parámetros básicos del sistema, incluyendo la conexión a la red y la habilitación del acceso remoto mediante SSH. A partir de este momento, el PLC puede ser administrado de manera remota sin necesidad de periféricos adicionales, lo que facilita su operación y mantenimiento.
- Con el PLC ya accesible y conectado correctamente a la red, proseguiremos con la instalación de los paquetes necesarios para su configuración, desarrollo y puesta en marcha de las aplicaciones requeridas en el proyecto.

## Capítulo 2

# Configuración e instalación de librerías

### 2.1. Instalación de dependencias

Para el desarrollo y ejecución de aplicaciones en el PLC, especialmente aquellas que requieren comunicación con servicios externos, manejo de interfaces web o interacción con APIs, es necesario contar con un entorno basado en Node.js, el cual permite ejecutar JavaScript del lado del servidor de forma eficiente. Sin embargo, diferentes proyectos pueden requerir versiones distintas de Node.js, por lo que se utiliza NVM (Node Version Manager), una herramienta que facilita la instalación, actualización y cambio entre versiones de Node sin afectar el sistema base. Esto nos proporciona un entorno flexible, estable y fácil de mantener durante la implementación y evolución del sistema.

#### 2.1.1. Instalación de NVM, Node.js y Yarn

Para gestionar diferentes versiones de Node.js dentro del sistema, utilizamos NVM (Node Version Manager). Este permite instalar, actualizar o cambiar versiones de Node sin afectar al sistema base.

1. Descargamos e instalamos NVM usando el siguiente comando:

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.sh | bash
```

2. Cargamos NVM en la sesión actual, indicando a la terminal dónde se encuentra instalado:

```
export NVM_DIR="$HOME/.nvm" [ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
```

3. Actualizamos la configuración de la terminal para que NVM esté disponible en cada inicio:

```
source ~/.bashrc
```

4. Verificamos que la instalación se realizó correctamente:

```
nvm --version
```

5. Finalmente, instalamos Yarn, un gestor de paquetes alternativo a npm, utilizado frecuentemente en proyectos modernos:

```
npm install --global yarn
```

Con estos pasos, el sistema queda preparado para manejar versiones de Node.js y utilizar Yarn en la instalación y administración de dependencias dentro de los proyectos que se desarrollarán en el PLC.

## 2.2. Descarga de paquetes y configuración

Para mantener una organización adecuada de los archivos y facilitar el acceso a los recursos del proyecto, se crearán dos carpetas principales. La primera carpeta se llamará Plc y estará ubicada en el directorio Home, donde se almacenarán los scripts, configuraciones y archivos propios del desarrollo. La segunda carpeta será Download, ubicada en /home/pi/Download, la cual servirá como espacio temporal para almacenar descargas, instaladores y archivos de apoyo antes de ser procesados o movidos a su ubicación definitiva. Esta estructura permitirá mantener el entorno de trabajo ordenado y diferenciar claramente el contenido permanente del contenido temporal.

```
mkdir ~/home/Plc  
mkdir ~/home/pi/Download
```

### 2.2.1. Librerías librpiplc

#### Preparar el sistema

Actualiza índices e instala las herramientas necesarias para compilar (git, cmake y toolchain de C/C++).

```
sudo apt update  
sudo apt install git cmake build-essential -y
```

- git: clona el repositorio.
- cmake: genera los archivos de compilación.
- build-essential: incluye gcc, g++, make, headers, etc.

## Clonar la librería

Descarga el código fuente ibrpiplc desde GitHub en la carpeta Downloads.

```
git clone https://github.com/Industrial-Shields/ibrpiplc.git  
cd ibrpiplc
```

## Configurar la compilación (CMake)

Indica tu versión y modelo de PLC (en tu caso, RPIPLC\_V6 y RPIPLC\_58) y genera la carpeta de build.

```
cmake -B build/ -DPLC_VERSION=RPIPLC_V6 -DPLC_MODEL=RPIPLC_58
```

- -B build/: crea/usa el directorio build para los artefactos.
- -DPLC\_VERSION=... y -DPLC\_MODEL=...: activan los pines/definiciones correctas para tu hardware.
- Si tuvieras otro modelo, ajusta PLC\_MODEL y/o PLC\_VERSION según corresponda.

## Compilar e instalar

Compila con todos los núcleos disponibles e instala en el sistema.

```
cmake --build build/ -- -j"${nproc}"  
sudo cmake --install build/
```

- El primer comando compila (equivalente a make -j).
- El segundo instala headers y librerías (por defecto en /usr/local/include y /usr/local/lib).

## Actualizar la caché de librerías

Para que el sistema encuentre ibrpiplc al enlazar/ejecutar:

```
sudo ldconfig
```

## 2.2.2. Compilación de programas en C

Con el fin de mantener una estructura organizada dentro del directorio de trabajo, dentro de la carpeta Plc crearemos una nueva carpeta llamada Programas. Esta carpeta contendrá tres subcarpetas: Programas\_N, Programas\_C y Programas\_P, destinadas a almacenar, respectivamente, los programas desarrollados en Node.js, C/C++ y Python. Esta clasificación permitirá mantener un orden adecuado según el lenguaje utilizado en cada implementación. Para las pruebas iniciales, trabajaremos en la carpeta Programas\_C, donde se desarrollarán y ejecutarán los programas en lenguaje C.

```
mkdir ~/Plc/Programas  
mkdir ~/Plc/Programas/Programas_N  
mkdir ~/Plc/Programas/Programas_C  
mkdir ~/Plc/Programas/Programas_P
```

### 2.2.2.1. Programas C

Dentro de la carpeta Programas\_C se crearán tres subcarpetas que agruparán los diferentes módulos de prueba y control del PLC. Estas subcarpetas serán:

1. Pines/ → Contendrá los programas relacionados con la configuración y manipulación de pines digitales (entradas y salidas).
2. Adc/ → Almacenará los programas destinados a la lectura y procesamiento de señales analógicas a través del conversor ADC.
3. Pwm/ → Incluirá los programas encargados de la generación y control de señales PWM para actuadores u otras aplicaciones que requieran modulación por pulso.

Comenzaremos por explorar el funcionamiento de los pines digitales del PLC Raspberry de Industrial Shields. La primera configuración consistirá en desarrollar un programa sencillo que permita encender un LED conectado a una de las salidas digitales del equipo.

Este ejercicio será nuestro punto de partida, ya que nos permitirá:

- Verificar que las librerías del PLC han sido instaladas correctamente.
- Confirmar que el entorno de trabajo permite la compilación y ejecución de programas.
- Comprobar que la comunicación con el hardware se realiza de manera correcta y estable.
- Una vez validado el funcionamiento básico, podremos avanzar a configuraciones más complejas como lectura de entradas, manejo del ADC y generación de señales PWM.

Una vez realizada la instalación de las librerías de Industrial Shields para el PLC Raspberry, se crea automáticamente una carpeta en la ruta:

/home/pi/common/RPIPLC58

La ruta de instalación puede variar ligeramente según el modelo específico del PLC; en este caso se exemplifica utilizando el modelo RPIPLC-56. Dentro de la carpeta correspondiente se encuentran dos archivos fundamentales que permiten la correcta identificación y uso de los pines del PLC durante la programación:

- common.h Este archivo contiene la declaración interna de los arreglos que agrupan las entradas digitales, entradas analógicas, salidas digitales y salidas analógicas disponibles en el dispositivo. Dichas agrupaciones están organizadas en función del modelo del PLC y son utilizadas directamente por la librería para gestionar el acceso y funcionamiento del hardware.
- pins\_reference.h Este archivo proporciona una tabla de equivalencias legible por el usuario, en la cual se asocia el nombre lógico del pin tal como aparece en el PLC (por ejemplo, Q0.0, I1.2) con su representación interna (Q0\_0, I1\_2). Gracias a esto, el programador puede manipular los pines de manera clara y directa dentro del código en C, evitando confusiones y facilitando el desarrollo de aplicaciones.

### Gpio I/O\_C:

Este programa realiza un parpadeo (encendido y apagado) continuo en la primera salida digital del PLC Raspberry PLC 58, específicamente en el pin Q0.0. Primero, las directivas `#define RPIPLC_V6` y `#define RPIPLC_58` indican a la librería librpiplc qué versión y modelo de hardware se está utilizando, lo que permite que el archivo common.h cargue correctamente la distribución de pines del PLC. Luego se llama a `initExpandedGPIO(false)` para inicializar los pines de entrada/salida sin módulos de expansión externos. Se selecciona la primera salida digital del arreglo `digitalOutputs[]` y se configura como salida utilizando `pinMode()`. Dentro del ciclo infinito `while(1)`, la variable `value` alterna entre 0 y 1, logrando activar y desactivar la salida digital en cada iteración. Esto se materializa en el hardware mediante `digitalWrite(digitalOutputs[0], value)`. El estado actual se imprime por consola con `printf`, y el programa espera un segundo entre cada cambio gracias a `sleep(1)`.

---

**Algoritmo 2.1** Gpio.cpp

---

```
#define RPIPLC_V6 // Define la versión de tu PLC
#define RPIPLC_58 // Define el modelo 58
#include <stdio.h>
#include <unistd.h>
#include <librpiplc/rpiplc.h>
#include "common.h" // Trae digitalOutputs[] y
                  numDigitalOutputs
int main() {
    // Inicializa los GPIO sin expansiones externas
    initExpandedGPIO(false);
    // Configura la primera salida (Q0.0) como salida digital
    pinMode(digitalOutputs[0], OUTPUT);
    int value = 0;
    while (1) {
        value = !value; // Alterna entre 0 y 1
        digitalWrite(digitalOutputs[0], value);
        printf("Q0.0 -> %d\n", value);
        fflush(stdout);
        sleep(1); // Espera 1 segundo
    }
    return 0;
}
```

---

Para compilar el programa se debe ejecutar la siguiente instrucción:

■ `sudo g++ -o Gpio Gpio.cpp -I/usr/local/include/librpiplc -L/usr/local/lib -lrpiplc`

**Adc y Pwm\_C:**

El código presentado permite leer un valor analógico desde una entrada del PLC Raspberry PLC 58 y utilizar ese valor para controlar una salida analógica tipo PWM en el mismo dispositivo. Primero se incluyen las definiciones RPIPLC\_V6 y RPIPLC\_58, necesarias para que la librería librpiplc cargue correctamente la configuración de hardware correspondiente a este modelo. Luego se inicializan los pines y se verifica que el PLC disponga de al menos una entrada y una salida analógica. Dentro del ciclo principal (while(1)), el programa realiza la lectura del canal analógico seleccionado mediante analogRead, obteniendo un valor de 12 bits entre 0 y 4095. Este valor se utiliza como ciclo de trabajo de la señal PWM y se envía a la salida con analogWrite. Finalmente, se imprime en pantalla tanto el valor leído como el valor aplicado a la salida, repitiendo el proceso cada 500 ms. En términos prácticos, este código permite convertir una señal analógica de entrada, como la proveniente de un sensor, en una señal modulada que puede ser enviada a actuadores como ventiladores, válvulas proporcionales o controladores de velocidad.

---

**Algoritmo 2.2** AdcPwm.cpp

---

```
#define RPIPLC_V6 // Necesario para librpiplc
#define RPIPLC_58 // Necesario para que common.h cargue
    los pines de este modelo
#include <stdio.h>
#include <unistd.h>
#include <librpiplc/rpiplc.h>
#include "common.h"
int main() { // Inicializa GPIO initExpandedGPIO(false);
// Verifica si hay al menos 1 entrada analógica y 1
salida analógica
if (numAnalogInputs > 0 && numAnalogOutputs > 0)
{ uint32_t pinIn = analogInputs[0]; // Primer canal
análogo (ej: I0_7)
uint32_t pinOut = analogOutputs[0]; // Primer canal PWM (
ej: A0_5)
pinMode(pinIn, INPUT); pinMode(pinOut, OUTPUT);
while (1) { // Leer ADC (12 bits normalmente 04095)
uint16_t value = analogRead(pinIn);
// Escalar a PWM (4095 duty cycle por ejemplo)
uint16_t pwmValue = (value * 4095) / 4095;
// Enviar a salida PWM analogWrite(pinOut, pwmValue);
printf("AI[0]=%u->PWM=%u\n", value, pwmValue);
usleep(500 * 1000); // 500 ms }
else {
printf("_Este.modelo_no_tiene_entradas_o_salidas_
análogicas_definidas.\n");
}
return 0; }
```

---

```
sudo g++ -o Adcpwm Adcpwm.cpp -I/usr/local/include/librpiplc -L/usr/local/lib -lrpiplc
```

**Controlador PI\_C**

Este programa implementa un controlador PI (Proporcional–Integral) para regular una señal analógica en el PLC Raspberry PLC 58. Primero se definen los parámetros del controlador y la constante de muestreo  $T_s = 0.1s$ . La función `pi_controller()` calcula la acción de control a partir del error entre el valor deseado (set point) y el valor medido, acumulando el error en el término integral para mejorar la estabilidad y eliminar el error permanente. En el programa principal, se inicializan las entradas y salidas analógicas y se toma el primer canal de entrada y salida definidos en `common.h`. Dentro del ciclo continuo `while(1)`, se lee el valor analógico del sensor, convirtiéndolo a voltaje real en el rango de 0–10V. Luego se define un punto de consigna de 5V, se calcula el error y se

evalúa el controlador PI para obtener el voltaje de control. Este valor se escala nuevamente al rango digital (0–4095) y se envía mediante analogWrite() a la salida analógica tipo PWM. Finalmente, el programa imprime el voltaje medido y el valor aplicado a la salida, repitiendo el proceso cada 100 ms. Con esto, el sistema ajusta automáticamente la salida para mantener la señal cercana al valor deseado, compensando variaciones o perturbaciones externas. Este código está basado en las notas de clase de sistemas de control en tiempo discreto Carlos Felipe

---

**Algoritmo 2.3** ControladorPI.cpp

---

```
#define RPIPLC_V6
#define RPIPLC_58
#include <stdio.h>
#include <unistd.h>
#include <librpiplc/rpiplc.h>
#include "common.h"
#define Ts 0.1f
// Funcion de Control PI
float pi_controller(float error) {
    const float sampling_time = Ts;
    const float kp = 1.2f;
    const float Ti = 0.5049f;
    static float integral_error = 0.0f;
    float control;
    integral_error += sampling_time * error;
    control = kp * (error + (1.0f / Ti) * integral_error);
    return control; }
// Programa principal
int main() {
    initExpandedGPIO(false);
    float voltage;
    float set_point_volts;
    float error;
    float control_volts;
    if (numAnalogInputs > 0 && numAnalogOutputs > 0)
    {
        uint32_t pinIn = analogInputs[0];
        uint32_t pinOut = analogOutputs[0];
        pinMode(pinIn, INPUT);
        pinMode(pinOut, OUTPUT);
        while (1) {
            uint16_t value = analogRead(pinIn);
            voltage = 10.0f * value / 4095.0f;
            set_point_volts = 5.0f;
            error = set_point_volts - voltage;
            control_volts = pi_controller(error);
            uint16_t pwmValue = ((control_volts * 4095.0f) / 10.0f);
            analogWrite(pinOut, pwmValue);
            printf("AI[0] = %.4f V -> PWM = %u\n", voltage, pwmValue);
            usleep(Ts * 1000000.0f); }
    }
    else {
        printf("Este modelo no tiene entradas o salidas "
               "analogicas definidas.\n");
    }
    return 0;
}
```

---

```
sudo g++ -o ControladorPI ControladorPI.cpp -I/usr/local/include/librpiplc -L/usr/local/lib -lrpiplc
```

## 2.2.3. Compilación de programas en Node

### 2.2.3.1. Programas en Node.js

La traducción de las rutinas escritas originalmente en C hacia Node.js se realiza mediante el uso de Node Addons, una técnica que permite crear módulos nativos en C/C++ que pueden ser invocados directamente desde JavaScript. Para ello, se emplea la librería node-addon-api, la cual ofrece una interfaz sencilla y segura para conectar funciones en C con el entorno de ejecución de Node.

```
sudo apt install -y build-essential python3-dev python3-pip git  
sudo apt install -y make g++ python3  
npm install -g node-gyp  
npm install node-addon-api
```

En este proceso, se debe compilar el código C utilizando node-gyp, herramienta que genera un módulo binario compatible con Node. En el archivo binding.gyp se especifican las rutas de las librerías del PLC (como librpiplc) y se definen constantes que habilitan la configuración correcta del hardware (RPIPLC\_V6 y RPIPLC\_58).

---

**Algoritmo 2.4** binding.gyp

---

```
{  
  "targets": [  
    {  
      "target_name": "rpiplc",  
      "sources": [ "rpiplc_wrapper.cpp" ],  
      "include_dirs": [  
        "<!(node -p \\\"require('node-addon-api').include\\\")",  
        "/usr/local/include/librpiplc"  
      ],  
      "libraries": [  
        "-lrpiplc"  
      ],  
      "cflags": [ "-Wall" ],  
      "cflags_cc": [ "-std=c++17",  
        "-fexceptions" ],  
      "defines": [  
        "NAPI_CPP_EXCEPTIONS",  
        "RPIPLC_V6",  
        "RPIPLC_58"  
      ]  
    }  
  ]  
}
```

---

Se crea un archivo

```
sudo nano rpiplc_wrapper.cpp
```

---

**Algoritmo 2.5** rpiplc\_wrapper.cpp

---

```
#define RPIPLC_V6
#define RPIPLC_58
#include <napi.h>
#include <librpiplc/expanded-gpio.h>
#include <librpiplc/rpiplc.h>
#include "common.h"

// _____ Digital _____
Napi::Number WriteDigital(const Napi::CallbackInfo& info) {
    Napi::Env env = info.Env();
    if (info.Length() < 2) return Napi::Number::New(env, -1);
    uint32_t pin = info[0].As<Napi::Number>().Uint32Value();
    uint8_t value = info[1].As<Napi::Number>().Uint32Value();
    pinMode(pin, OUTPUT); int res = digitalWrite(pin, value);
    return Napi::Number::New(env, res);
}

Napi::Number ReadDigital(const Napi::CallbackInfo& info) {
    Napi::Env env = info.Env();
    if (info.Length() < 1) return Napi::Number::New(env, -1);
    uint32_t pin = info[0].As<Napi::Number>().Uint32Value(); int res =
        digitalRead(pin);
    return Napi::Number::New(env, res);
}

// _____ ADC _____
Napi::Number ReadADC(const Napi::CallbackInfo& info) {
    Napi::Env env = info.Env();
    if (numAnalogInputs == 0) return Napi::Number::New(env, -1);
    int channel = 0; // por defecto canal 0
    if (info.Length() >= 1) {
        channel = info[0].As<Napi::Number>().Int32Value();
        if (channel < 0 || channel >= numAnalogInputs) {
            return Napi::Number::New(env, -2); // canal inválido
        }
    }
    uint16_t value = analogRead(analogInputs[channel]);
    return Napi::Number::New(env, value);
}

// _____ PWM (12 bits) _____
Napi::Number WritePWM(const Napi::CallbackInfo& info) {
    Napi::Env env = info.Env();
    if (numAnalogOutputs == 0 || info.Length() < 2) {
        return Napi::Number::New(env, -1); // error: sin salidas o sin args
    }
    int channel = info[0].As<Napi::Number>().Int32Value();
    if (channel < 0 || channel >= numAnalogOutputs) {
        return Napi::Number::New(env, -2); // error: canal inválido
    }
    // Usar uint16_t para PWM de 12 bits (0-4095)
    uint16_t duty = info[1].As<Napi::Number>().Uint32Value();
    if (duty > 4095) duty = 4095; // limitar a 12 bits
    analogWrite(analogOutputs[channel], duty);
    return Napi::Number::New(env, 0); // ok
}

// _____ Init _____
Napi::Object Init(Napi::Env env, Napi::Object exports) {
    initExpandedGPIO(false);
    exports.Set("writeDigital", Napi::Function::New(env, WriteDigital));
    exports.Set("readDigital", Napi::Function::New(env, ReadDigital));
    exports.Set("readADC", Napi::Function::New(env, ReadADC));
    exports.Set("writePWM", Napi::Function::New(env, WritePWM));
    return exports;
}
NODE_API_MODULE(rpiplc, Init)
```

---

Este código implementa un módulo nativo para Node.js que permite controlar el GPIO del Raspberry Pi.

lar directamente el hardware del PLC Raspberry PLC 58 utilizando JavaScript. Para lograrlo, se utilizan las librerías librpiplc y node-addon-api, que actúan como puente entre el entorno C/C++ y el motor de ejecución de Node. El archivo define funciones que encapsulan operaciones fundamentales del PLC: escritura y lectura de salidas digitales (writeDigital y readDigital), lectura de entradas analógicas (readADC) y control de salidas analógicas PWM de 12 bits (writePWM). Cada función recibe parámetros desde Node.js, valida los argumentos y ejecuta las funciones nativas correspondientes del PLC. Finalmente, la función Init() registra estas funciones como métodos exportados, permitiendo su uso en aplicaciones JavaScript mediante require(). De esta manera, el código logra integrar el control de hardware de bajo nivel dentro de aplicaciones de más alto nivel ejecutadas en Node, facilitando el desarrollo de interfaces web, dashboards o servidores de automatización sin abandonar el entorno JavaScript.

Se hace necesario compilar la solución

```
npm install  
node-gyp clean  
node-gyp configure  
node-gyp build
```

- npm install Instala las dependencias del proyecto definidas en package.json. En este caso, asegura que node-addon-api y cualquier otra librería necesaria estén disponibles antes de compilar.
- node-gyp clean Limpia archivos de compilación previos. Elimina carpetas como build/ para garantizar que la compilación se realice desde cero, evitando errores por archivos antiguos.
- node-gyp configure Genera los archivos necesarios para la compilación en base al archivo binding.gyp. Crea los scripts y configuraciones que el compilador necesitará dependiendo del sistema operativo y versión de Node.
- node-gyp build Compila el código C/C++ y genera el módulo nativo final (rpiplc.node) dentro de la carpeta build/Release/. Este archivo es el que podrás importar desde JavaScript usando require().

### Gpio I/O\_N:

El archivo Gpio.js se creó con el propósito de simplificar el control de salidas digitales del PLC desde aplicaciones desarrolladas en Node.js. A través de una clase orientada a objetos, se encapsula la lógica necesaria para encender y apagar un pin digital específico utilizando las funciones del módulo nativo previamente compilado. De esta manera, el programador únicamente debe instanciar la clase indicando el número del pin y llamar a los métodos on() u off() según sea necesario. Esta abstracción facilita la reutilización del código y permite integrar el control del hardware directamente en aplicaciones de mayor nivel, como servidores web, paneles de monitoreo, sistemas de automatización o interfaces gráficas, sin necesidad de trabajar directamente con la API de bajo nivel del PLC.

---

**Algoritmo 2.6 Gpio.js**

---

```
const plc = require ('./ build/Release/rpiplc');
const Q0_0_PIN = 20971532; // Reemplaza por el valor correcto de tu common.h>
console.log("Encendiendo Q0.0... ");
plc.writeDigital(Q0_0_PIN, 1);
setTimeout(() => { console.log("Apagando Q0.0... ");
plc.writeDigital(Q0_0_PIN, 0);
}, 2000);
```

---

Para correr el programa sólo se hace necesario ejecutar la instrucción

- node Gpio.js

**Adc y Pwm\_N:**

Este script en Node.js permite realizar una prueba combinada de lectura analógica (ADC) y control de salida PWM en el PLC Raspberry PLC. Primero se define el canal de entrada analógica a leer y el canal de salida PWM a controlar. El programa ejecuta un ciclo periódico cada 100 ms, donde se obtiene el valor analógico desde el canal seleccionado mediante rpiplc.readADC(). Luego, ese valor se escala proporcionalmente para ajustarse al rango de salida PWM de 12 bits (0 a 4095), y se envía al canal PWM correspondiente usando rpiplc.writePWM(). Durante la ejecución, el programa muestra en consola el valor leído y el valor aplicado en la salida, permitiendo observar el comportamiento en tiempo real. El proceso continúa durante un tiempo determinado (20 segundos en este ejemplo), tras lo cual se detiene automáticamente. Esta rutina es útil para probar sensores analógicos y verificar control proporcional de actuadores, tales como válvulas, motores o módulos de señal proporcional.

---

**Algoritmo 2.7 Pwm\_adc.js**

---

```
const rpiplc = require ('./ build/Release/rpiplc');
const canalADC = 0; // Canal ADC que quieres leer
const canalPWM = 0; // Canal PWM a controlar
const duracionSegundos = 20;
const intervaloMs = 100;
let tiempoTranscurrido = 0;
const interval = setInterval(() => { let valorADC = rpiplc.readADC(canalADC); ////
if (valorADC < 0) {
  valorADC = 0;
}
const valorPWM = Math.round ((valorADC / 4095) * 4095);
rpiplc.writePWM(canalPWM, valorPWM);
tiempoTranscurrido += intervaloMs / 1000;
if (tiempoTranscurrido >= duracionSegundos) {
  clearInterval(interval); console.log(' Proceso finalizado . ');
}
}, intervaloMs );
```

---

■  
node Pwm\_adc.js

**Controlador PI.js**

Este programa implementa un controlador PI para el PLC Raspberry PLC 58 utilizando Node.js como entorno de ejecución. El algoritmo realiza la lectura de una señal analógica proveniente de un sensor, la convierte a su equivalente en voltaje y la compara contra un valor de referencia o consigna. A partir de este error, la función PI calcula una señal de control que se ajusta en el tiempo gracias al término integral, lo cual permite eliminar errores persistentes y mejorar la estabilidad del sistema. La señal de control se escala y se envía a una salida PWM de 12 bits, logrando así modular la respuesta de un actuador. El ciclo de control se ejecuta con un periodo fijo definido por el usuario, permitiendo mantener actualizada la acción de control frente a cambios en la señal medida. Este enfoque demuestra cómo conceptos clásicos de control automático pueden integrarse con tecnologías modernas basadas en JavaScript, habilitando aplicaciones de control industrial en plataformas IoT y sistemas distribuidos. Este código está basado en las notas de clase de sistemas de control en tiempo discreto Carlos Felipe.

---

**Algoritmo 2.8** ControlPI.js

---

```
const rpiplc = require ('./ build/Release/rpiplc ');
const canalADC = 0; // Primer canal ADC
const canalPWM = 0; // Primer canal PWM
const duracionSegundos = 60; // Duración total
const Ts = 0.1; // Tiempo de muestreo en segundos (100 ms)
const intervaloMs = Ts * 1000; // Convertir a ms para setInterval
// Parámetros PI
const kp = 1.2;
const Ti = 0.5049;
let integralError = 0.0;
let tiempoTranscurrido = 0;
const piController = (error) => {
  integralError += Ts * error; let control = kp * (error + (1 / Ti) * integralError);
  if (control < 0) control = 0;
  if (control > 10) control = 10;
  return control;
};
const interval = setInterval(() => {
  let valorADC = rpiplc.readADC(canalADC);
  if (valorADC < 0) valorADC = 0;
  // Convertir a voltaje 0–10V
  const voltage = (10.0 * valorADC) / 4095;
  // Calcular error
  const setPoint = 5.0; // Setpoint 5V
  const error = setPoint - voltage;
  // Calcular control PI
  const controlVoltage = piController(error);
  // Escalar a PWM 12-bit
  const valorPWM = Math.round((controlVoltage / 10) * 4095);
  rpiplc.writePWM(canalPWM, valorPWM);
  tiempoTranscurrido += Ts;
  if (tiempoTranscurrido >= duracionSegundos) {
    clearInterval(interval);
  }
}, intervaloMs);
```

---

node ControlPI.js

## 2.2.4. Compilación de programas en Python

Para instalar y probar la librería rpiplc-lib en un PLC Raspberry Pi versión V6, primero es necesario identificar la versión del sistema operativo ejecutando:

```
cat /etc/os-release
```

Si el sistema corresponde a Raspberry Pi OS Bookworm, se debe añadir el repositorio oficial de Industrial Shields con los siguientes comandos:

```
sudo tee /etc/apt/sources.list.d/industrialshields.list > /dev/null <<EOF
deb https://apps.industrialshields.com/main/DebRepo/ .
EOF
```

Luego se importa la clave pública y se actualiza la lista de paquetes:

```
wget -O - https://apps.industrialshields.com/main/DebRepo/PublicKey.gpg | gpg --dearmor | \ sudo tee /etc/apt/trusted.gpg.d/IndustrialShieldsDebian.gpg > /dev/null
sudo apt update
```

A continuación, se instala la librería correspondiente:

```
sudo apt install rpiplc-lib
```

Para verificar la instalación, se utiliza:

```
python3 -m pip show rpiplc-lib
```

El sistema debe mostrar la versión 4.0.1 instalada en la ruta /usr/local/lib/python3.xx/dist-packages/. Finalmente, para comprobar el correcto funcionamiento del PLC y la comunicación con sus módulos, se ejecuta un programa de prueba:

### 2.2.4.1. Programas en Python

En condiciones normales, las librerías del PLC (como rpiplc-lib) no requieren mantenerse dentro de la carpeta del proyecto, ya que se instalan en el sistema de forma global y quedan disponibles para cualquier script de Python sin necesidad de incluir dependencias adicionales en el mismo directorio. Sin embargo, cuando se trabaja con entornos virtuales, estos no tienen acceso por defecto a las librerías del sistema. Para conservar la compatibilidad y permitir que el entorno utilice las librerías ya instaladas, es necesario crear el entorno virtual con acceso explícito al espacio de paquetes del sistema mediante el parámetro --system-site-packages. Por ejemplo:

```
python3 -m venv opc_plc_env --system-site-packages
```

De esta manera, el entorno virtual opc\_plc\_env podrá utilizar correctamente las librerías preinstaladas del PLC sin necesidad de reinstalarlas o duplicarlas dentro del proyecto.

### **Gpio I/O \_py:**

El archivo gpio.py fue creado con el propósito de simplificar el control de salidas digitales del PLC desde aplicaciones desarrolladas en Python. A través de una clase orientada a objetos, se encapsula la lógica necesaria para activar y desactivar un pin digital específico utilizando las funciones proporcionadas por la librería rpiplc. De esta manera, el programador únicamente debe instanciar la clase indicando el nombre del pin (por ejemplo, "Q0.0") y llamar a los métodos on(1) u off(0) según sea necesario. Esta abstracción facilita la reutilización del código y permite integrar el control del hardware directamente en aplicaciones de mayor nivel, como servicios de automatización, interfaces gráficas en PyQt/PySide, scripts de monitoreo o controladores industriales, sin la necesidad de interactuar directamente con la API de bajo nivel del PLC.

---

#### **Algoritmo 2.9 Gpio.py**

---

```
from librpiplc import rpiplc
rpiplc.init("RPIPLC_V6", "RPIPLC_58", restart=True)
rpiplc.pin_mode("Q0.0", rpiplc.OUTPUT)
for i in range(5):
    rpiplc.digital_write("Q0.0", rpiplc.HIGH)
    rpiplc.delay(500)
    rpiplc.digital_write("Q0.0", rpiplc.LOW)
    rpiplc.delay(500)
rpiplc.deinit(restart=True)
print("Blink test terminado")
```

---

python3 Gpio.py

### **Adc y Pwm \_py:**

Este script en Python permite realizar una prueba combinada de lectura analógica (ADC) y control de salida PWM en el PLC Raspberry PLC utilizando la librería rpiplc. Primero se configura el canal de entrada analógica a leer y el canal de salida PWM a controlar. El programa ejecuta un ciclo periódico cada 100 ms, donde obtiene el valor analógico desde el canal seleccionado mediante rpiplc.analogRead(). Posteriormente, ese valor se escala proporcionalmente al rango de salida PWM de 12 bits (0 a 4095) y se envía al canal PWM correspondiente utilizando rpiplc.pwmWrite(). Durante el proceso, el script imprime en consola tanto el valor leído como el valor aplicado en la salida, lo que permite observar el comportamiento en tiempo real. La ejecución se mantiene durante un tiempo determinado (por ejemplo, 20 segundos) y luego se detiene automáticamente. Esta rutina es especialmente útil para la verificación de sen-

sores analógicos y el control proporcional de actuadores como válvulas, motores o módulos de señal 0–10 V basados en modulación por ancho de pulso (PWM).

---

**Algoritmo 2.10** Pwm\_adc.py

---

```
import time
import rpiplc
# Configurar el modelo del PLC
rpiplc.config(rpiplc.RPIPLC_V6, rpiplc.RPIPLC_58)
canalADC = "ADC0" # Canal analógico
canalPWM = "PWM0" # Canal PWM
duracion = 20 # Segundos
Ts = 0.1 # Período de muestreo (100 ms)
print("Iniciando prueba ADC -> PWM... ")
t = 0
while t < duracion:
    valorADC = rpiplc.analogRead(canalADC)
    if valorADC < 0:
        valorADC = 0
    # Escalar a rango PWM (0-4095)
    valorPWM = int((valorADC / 4095.0) * 4095)
    rpiplc.pwmWrite(canalPWM, valorPWM)
    print(f"t={t:.1f} s | ADC={valorADC} -> PWM={valorPWM}")
    time.sleep(Ts)
    t += Ts
print("Prueba finalizada.")
```

---

**Controlador PI.py**

Este programa implementa un controlador PI para el PLC Raspberry PLC 58 utilizando Python como entorno de ejecución. El algoritmo realiza la lectura de una señal analógica proveniente de un sensor, la convierte a su equivalente en voltaje y la compara con un valor de referencia o consigna. A partir de esta diferencia, conocida como error, el controlador PI calcula una señal de control que combina una acción proporcional y una acción integral acumulada en el tiempo, lo cual permite reducir errores persistentes y mejorar la estabilidad del sistema ante perturbaciones. La señal de control resultante es posteriormente escalada y enviada a una salida PWM de 12 bits, permitiendo modular la respuesta de un actuador que opere en el rango 0–10 V. El ciclo de control se ejecuta de manera periódica con un intervalo de muestreo fijo, garantizando la actualización continua de la acción de control frente a cambios en la señal medida. Este enfoque muestra cómo los fundamentos de control automático en tiempo discreto pueden trasladarse eficientemente a plataformas modernas basadas en Python, facilitando su integración en aplicaciones industriales, sistemas embebidos e infraestructura IoT. Este código se basa en los principios estudiados en las notas

de clase del curso de Sistemas de Control en Tiempo Discreto Carlos Felipe y se emplea aquí para su implementación práctica en hardware real.

---

**Algoritmo 2.11** ControlPi.py

---

```
import time
import rpiplc
rpiplc.config(rpiplc.RPIPLC_V6, rpiplc.RPIPLC_58)
# Parámetros de hardware
canalADC = "ADC0" # Primer canal ADC
canalPWM = "PWM0" # Primer canal PWM
duracionSegundos = 60
Ts = 0.1 # Tiempo de muestreo en segundos
# Parámetros del controlador PI
kp = 1.2
Ti = 0.5049
integralError = 0.0
tiempoTranscurrido = 0.0
def piController(error):
    global integralError
    integralError += Ts * error
    control = kp * (error + (1 / Ti) * integralError)
    # Limitar control entre 0 y 10 V
    if control < 0: control = 0
    if control > 10: control = 10
    return control
while tiempoTranscurrido <= duracionSegundos:
    # Leer ADC (0-4095)
    valorADC = rpiplc.analogRead(canalADC)
    if valorADC < 0:
        valorADC = 0
    # Convertir a voltaje 0-10V
    voltage = (10.0 * valorADC) / 4095.0
    # Setpoint y error
    setPoint = 5.0
    error = setPoint - voltage
    # Calcular control PI
    controlVoltage = piController(error)
    # Convertir a PWM (0-4095)
    valorPWM = round((controlVoltage / 10.0) * 4095.0)
    rpiplc.pwmWrite(canalPWM, valorPWM)
    tiempoTranscurrido += Ts
    time.sleep(Ts)
print("Control PI finalizado.")
```

---

## Notas:

Todos los programas de control desarrollados en C, Node.js y Python, tanto para las rutinas ADC→PWM como para el lazo PI, se fundamentan en la planta mostrada en la figura 2.1. Aunque el circuito físico corresponde a un sistema de *tercer orden* (una cadena de etapas RC), para el diseño, sintonía e implementación práctica se adopta una *aproximación de primer orden* que representa la dinámica dominante del sistema. Esta reducción de orden se justifica por la separación entre los tiempos de los polos y la similitud con la respuesta medida, obteniéndose ya sea mediante identificación experimental (respuesta al escalón) o a través del ajuste de un modelo equivalente  $G(s) = \frac{K}{\tau_s + 1}$ . El modelo continuo se discretiza mediante retención de orden cero (ZOH) con un período de muestreo  $T_s$ , donde la variable de proceso se mide con el conversor **ADC** (0–4095 → 0–10 V), y la acción de control se aplica mediante el módulo **PWM** (0–4095, equivalente a 0–10 V). De esta forma, el software implementa un lazo cerrado coherente con la teoría del control en tiempo discreto, manteniendo simplicidad computacional sin comprometer el desempeño dinámico del sistema.

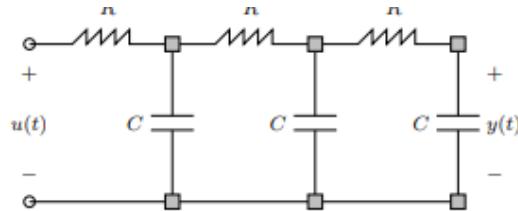


Figura 2.1: Planta para controlador PI.

- El circuito RC no es solo un ejemplo, sino el modelo base sobre el cual se fundamenta:
  - El diseño y análisis en tiempo discreto
  - La identificación y obtención del modelo matemático
  - El diseño del controlador PI
  - La implementación del lazo cerrado real
  - En otras palabras, todo el control implementado en el PLC replica exactamente la lógica estudiada en clase para el sistema RC en tiempo discreto.

## **2.3. Dependencias y paquetes de terceros.**

### **2.3.1. Guía de instalación y configuración de Mosquitto MQTT Broker**

#### **Introducción**

El protocolo MQTT (Message Queuing Telemetry Transport) es un estándar ligero de mensajería basado en el modelo publicador/suscriptor, ideal para la comunicación entre dispositivos IoT, como el Raspberry PLC-58 y servidores o aplicaciones industriales.

El broker MQTT actúa como intermediario entre los clientes que publican mensajes (publishers) y los que los reciben (subscribers). En este caso, el broker será el PC y el cliente será el PLC.

#### **1. Actualizar el sistema**

```
sudo apt update
```

#### **2. Instalar Mosquitto y las herramientas cliente**

```
sudo apt install mosquitto mosquitto-clients -y
```

Estas herramientas instalan tanto el servidor (broker) como los comandos de prueba mosquitto\_pub y mosquitto\_sub.

#### **3. Iniciar y habilitar el servicio**

Habilitar el servicio para que inicie automáticamente al arrancar:

```
sudo systemctl enable mosquitto
```

Ejecución directa con salida en consola (modo verbose):

```
mosquitto -v
```

#### **4. Verificación de funcionamiento**

Para comprobar que el servicio escucha en el puerto 1883 (por defecto):

```
sudo lsof -i :1883
```

Si se desea usar otro puerto:

```
mosquitto -p 1884 -v
```

#### **5. Permitir conexiones externas**

Por defecto, Mosquitto puede estar configurado solo para conexiones locales. Para habilitar conexiones externas (por ejemplo, desde el PLC):

```
sudo nano /etc/mosquitto/mosquitto.conf
```

```

listener 1883 0.0.0.0
protocol mqtt
listener 8083 0.0.0.0
protocol websocket
# — Seguridad
allow_anonymous false
password_file /etc/mosquitto/passwd
acl_file /etc/mosquitto/aclfile
# — Logs
connection_messages true
log_type all
log_dest syslog
log_dest stdout

```

Guarda con Ctrl + O, luego Enter y sal con Ctrl + X. Después, reinicia el servicio:

```
sudo systemctl restart mosquitto
```

## 6. Configuración del acceso seguro (usuarios y permisos)

Crear un usuario y contraseña. Ejemplo: crear usuario plcuser con contraseña plc:

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd plcuser
```

Para añadir más usuarios (sin sobrescribir el archivo):

```
sudo mosquitto_passwd /etc/mosquitto/passwd otroUser
```

Crear archivo de control de acceso (ACL). Define los permisos de publicación y suscripción por tema (topic).

```
sudo nano /etc/mosquitto/aclfile
```

Contenido recomendado:

```

# — 1. CONFIGURACIÓN
user plcuser
#Estos se mantienen para la autonomía interna y la comunicación funcional
topic readwrite Plc/Adc
topic readwrite Plc/Ia
topic readwrite Plc/Pwm
topic readwrite Plc/Timer
topic readwrite Plc/Control
topic readwrite Plc/Respuesta
topic readwrite Plc/Supervisor
topic readwrite Plc/Caracterizacion
topic readwrite Plc/Identificacion
topic readwrite Plc/Petri

```

```
topic readwrite Plc/Comparacion  
topic readwrite Plc/ControlIA  
topic readwrite Plc/Reset
```

Guarda los cambios y reinicia el servicio:

```
sudo systemctl restart mosquitto
```

Verifica nuevamente:

```
sudo systemctl status mosquitto sudo lsof -i :1883
```

## 7. Pruebas de comunicación

- Suscriptor (en el BROKER / PC)

Escucha todos los tópicos que empiecen con Plc/:

```
mosquitto_sub -h localhost -t "Plc/#" -u plcuser -P "plc"
```

- Publicador (en el CLIENTE / PLC)

Envía un mensaje de prueba:

```
mosquitto_pub -h <IP_DEL_BROKER> -t "Plc/Adc" -m "123" -u plcuser -P "plc"
```

Ejemplo: Si el PC tiene IP 192.168.1.100, entonces:

```
mosquitto_pub -h 192.168.1.100 -t "Plc/Adc" -m "123" -u plcuser -P "plc"
```

## 8. Recomendaciones de seguridad

Deshabilita allow\_anonymous true una vez configuradas las contraseñas.

Usa un puerto diferente de 1883 (por ejemplo, 1884 o 1885) para entornos con múltiples brokers.

Si el PLC se conecta desde otra red, abre el puerto en el firewall:

```
sudo ufw allow 1883/tcp
```

## Capítulo 3

# Comparativa funcional entre control clásico y control IA

El método de identificación empleado hasta ahora es un proceso manual y secuencial, basado en la experimentación directa sobre el sistema físico ver 2.1. Cada etapa desde la adquisición de datos hasta la implementación del controlador requiere intervención humana, decisiones técnicas, y validaciones parciales, lo que incrementa significativamente la complejidad operativa. Además, el procedimiento involucra el uso simultáneo de diversas herramientas de desarrollo, como Arduino, MATLAB y lenguajes de programación de bajo nivel, que deben integrarse de forma manual para completar el ciclo de identificación y control.

### 1. Fase de caracterización:

En esta primera etapa, el operador debe configurar el entorno experimental: establecer las conexiones eléctricas, definir el rango de operación de las señales analógicas, y ajustar los sensores. El éxito depende de la experiencia del usuario, pues cualquier desviación en los niveles de referencia, ruido o tiempo de muestreo puede alterar el comportamiento del sistema. Además, los tiempos de configuración y las pruebas previas incrementan la carga de trabajo antes incluso de iniciar la identificación.

### 2. Fase de identificación:

Esta fase representa el núcleo del procedimiento y también su punto más complejo. Requiere seleccionar un modelo matemático adecuado (por ejemplo, de primer, segundo o tercer orden), capturar datos experimentales de entrada y salida, y aplicar herramientas de ajuste o estimación de parámetros. El problema radica en que este proceso es iterativo: los datos deben limpiarse, filtrarse, y validarse antes de obtener un modelo fiable. Además, se necesita conocimiento especializado en técnicas de ajuste, análisis de error y validación de modelos. Por ello, la identificación no solo es laboriosa, sino que también depende del criterio y la interpretación del analista.

### 3. Fase de simulación:

Una vez identificado el modelo, es necesario simularlo para evaluar su precisión y estabilidad. Esto implica definir condiciones iniciales, configurar entornos de simulación (como MATLAB o Simulink), e interpretar gráficas de respuesta temporal. Aunque conceptualmente sencilla, esta etapa requiere atención al detalle y conocimientos de control digital para garantizar que las simulaciones representen con fidelidad el comportamiento físico del sistema.

### 4. Fase de control:

Finalmente, el modelo obtenido se utiliza para diseñar un controlador. Aquí, nuevamente, el usuario debe seleccionar la estrategia (PI, PID, o más avanzada), discretizar el controlador, ajustar los parámetros de ganancia, y validar la respuesta experimental. Cada ajuste demanda nueva compilación, carga del programa en el microcontrolador y medición de la respuesta real, repitiendo un ciclo de prueba–error que consume tiempo y recursos.

Para superar estas limitaciones, se propone reemplazar el flujo manual por un agente inteligente de identificación y control, que gestione automáticamente las fases del proceso:

#### 1. Caracterización automática:

En esta nueva arquitectura, el usuario únicamente debe definir los parámetros de operación de la planta, tales como el rango de voltaje, la frecuencia de muestreo, el tiempo de duración de la prueba, los límites de voltaje y el intervalo de muestreo. A partir de esta información inicial, el agente autónomo se encarga de ejecutar de manera automática las instrucciones necesarias, activando los canales de entrada y salida del PLC para llevar a cabo la adquisición y estimulación del sistema físico sin requerir intervención manual.

En caso de que el usuario no introduzca alguno de estos parámetros, el sistema cuenta con valores de caracterización por defecto, preestablecidos para garantizar la correcta ejecución del proceso. Estos parámetros incluyen configuraciones seguras de tensión, límites de voltaje máximos permitidos por el hardware y una frecuencia de muestreo óptima basada en la respuesta dinámica típica de la planta 2.1. De esta forma, el agente puede iniciar el procedimiento de caracterización incluso sin configuración previa, asegurando la operación autónoma y la reproducibilidad de las pruebas.

Durante esta fase, el agente:

- a) Inicializa los canales ADC (entradas analógicas) y PWM/DAC (salidas analógicas o moduladas).
- b) Verifica el estado de conexión y ajustar las señales dentro de los límites definidos por el usuario.

- c) Ajusta el tiempo de muestreo y la resolución de lectura según la dinámica estimada del sistema.
- d) Registra los datos de entrada y salida necesarios para las etapas posteriores de identificación y simulación.
- e) Este proceso elimina la necesidad de configuración directa en MATLAB o el entorno del PLC, ya que el agente controla directamente la comunicación con el hardware, garantizando que las pruebas se realicen dentro de condiciones seguras y reproducibles.

2. Identificación adaptativa:

Durante la fase de caracterización, toda la información adquirida de voltaje, señales PWM, tiempos de muestreo y condiciones experimentales se almacena de forma estructurada en una base de datos PostgreSQL. Esta arquitectura de almacenamiento permite registrar, versionar y reutilizar los ensayos realizados, facilitando el acceso a series históricas de datos y la trazabilidad de cada prueba. Gracias a esta integración inteligente, el agente puede consultar los registros históricos almacenados en la base de datos y seleccionar automáticamente la prueba de caracterización más representativa para el proceso de identificación del modelo. En caso de que el usuario desee intervenir, puede elegir manualmente una prueba específica; de lo contrario, el agente selecciona por defecto la prueba más reciente y completa disponible, garantizando así la continuidad y coherencia del proceso de identificación sin requerir supervisión directa.

En la etapa de identificación adaptativa, el agente analiza los datos almacenados, selecciona el modelo más adecuado y ajusta sus parámetros dinámicos. El sistema evalúa de forma autónoma diferentes estructuras de modelos (por ejemplo, de primer o segundo orden) y elige la que mejor describe el comportamiento observado.

- 3. Posteriormente, en la fase de simulación interna, el modelo identificado se valida comparando su respuesta con los datos experimentales almacenados. Garantizando que la representación obtenida sea precisa y estable.
- 4. Finalmente, en la fase de diseño y ajuste de control, el agente genera y evalúa de forma automática el controlador óptimo para nuestro caso (PI), simulando la respuesta antes de aplicarla al sistema físico. Esta secuencia integrada permite cerrar el ciclo completo desde la adquisición y modelado hasta el control sin intervención del usuario, aprovechando los datos históricos registrados en la base de datos para acelerar y mejorar el proceso de identificación. y posteriormente probar el controlador con la planta 2.1.

### **3.1. Esquema para la implementación del controlador**

La arquitectura general del sistema inteligente se encuentra definida bajo un enfoque multiagente, donde cada componente cumple un rol específico dentro de un entorno cooperativo orientado a la automatización total del ciclo de caracterización, identificación, simulación, comparación y control.

En este modelo, los agentes no son simples módulos de software, sino entidades autónomas, comunicantes y coordinadas, capaces de percibir información, procesarla, tomar decisiones y actuar sobre el entorno físico o virtual según su función asignada.

El sistema está compuesto por dos niveles jerárquicos de agentes:

#### **Agente Central (Servidor – OpenAI)**

Funciona como agente cognitivo de coordinación, responsable de interpretar las instrucciones emitidas en lenguaje natural por el usuario.

- Analiza la intención de la solicitud, determina la tarea requerida y la distribuye a los agentes especializados mediante el sistema de mensajería (broker MQTT).
- Supervisa la ejecución global del sistema y mantiene la coherencia entre las fases del proceso, asegurando la trazabilidad y la secuencia lógica de caracterización → identificación → simulación → comparación → control.

#### **Agentes Locales (IAs embebidas en el PLC Raspberry PLC-58)**

- Actúan como agentes de campo o agentes de ejecución, con capacidad de percepción, acción y aprendizaje en el entorno físico.
- Cada uno de estos agentes ejecuta de forma autónoma una etapa específica:
- Agente de Caracterización: realiza pruebas experimentales y registra la respuesta de la planta.
- Agente de Identificación: analiza los datos para generar el modelo matemático del sistema.
- Agente de Simulación: valida el modelo y compara comportamiento real vs. estimado.
- Agente de Comparación: evalúa el rendimiento entre diferentes estrategias de control.
- Agente de Control Inteligente: integra la información de los demás agentes para generar la acción óptima sobre la planta.

Cada agente local cuenta con acceso directo a la base de datos PostgreSQL, donde almacena sus resultados y consulta registros históricos, permitiendo aprendizaje incremental y retroalimentación continua.

## **Comunicación y coordinación**

La interacción entre agentes se desarrolla mediante una infraestructura de mensajería basada en MQTT, que actúa como medio de comunicación asíncrono y descentralizado. El servidor (agente central) publica las instrucciones en los tópicos correspondientes, mientras que los agentes locales (en el PLC) se suscriben a dichos tópicos y ejecutan las acciones asociadas. De forma inversa, los agentes locales también publican los resultados de su ejecución en tópicos de retroalimentación, que son recibidos por el servidor y reenviados a la interfaz web en tiempo real.

## **Características del sistema multiagente**

- Autonomía: cada agente decide y actúa sin intervención externa.
- Comunicación: los agentes intercambian información mediante protocolos estandarizados.
- Cooperación: los agentes colaboran para alcanzar un objetivo común (optimización del control).
- Aprendizaje: el conocimiento adquirido se retroalimenta en los modelos de decisión.
- Escalabilidad: se pueden incorporar nuevos agentes (por ejemplo, de predicción o diagnóstico) sin modificar la arquitectura base.

En conjunto, esta estructura multiagente permite que el sistema evolucione dinámicamente, coordinando acciones entre el nivel cognitivo (servidor OpenAI) y el nivel operativo (PLC con IAs locales), logrando así un control inteligente distribuido, autónomo y adaptable a las condiciones del entorno ver figura 3.1

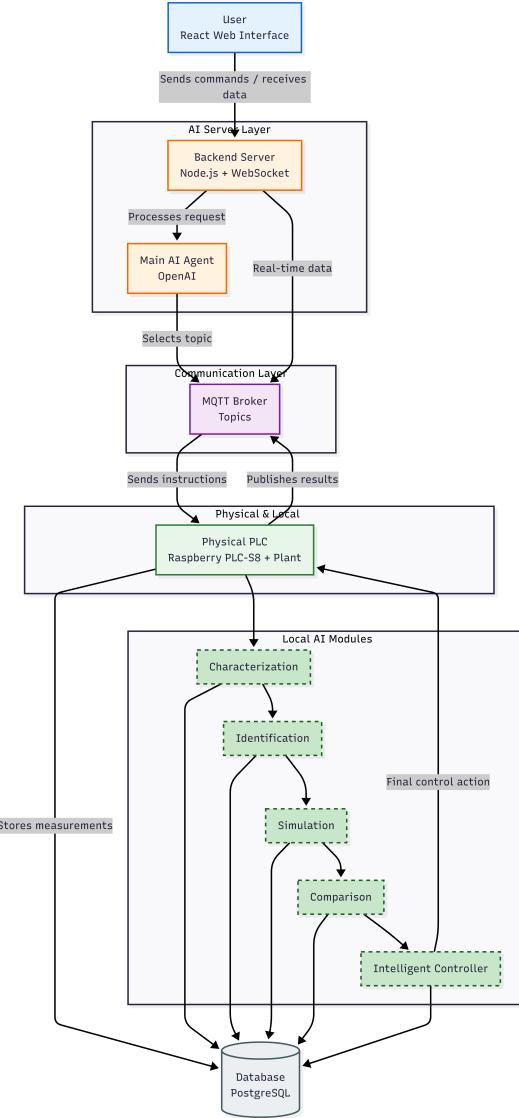


Figura 3.1: Diagrama general de la arquitectura.

El sistema 3.1 propuesto implementa un enfoque integral de automatización inteligente para identificación, simulación y control de procesos físicos, combinando tecnologías de inteligencia artificial, comunicación en tiempo real y ejecución embebida sobre un PLC industrial (Raspberry PLC-58). El objetivo principal es cerrar el ciclo completo de modelado y control sin intervención manual, partiendo de la interacción de un usuario mediante lenguaje natural.

### 3.1.1. La arquitectura está compuesta por cinco capas funcionales:

1. Capa de Usuario (Interfaz Web React).
2. Capa de Servidor y Agente Central (Node.js + OpenAI).
3. Capa de Comunicación (Broker MQTT).
4. Capa Física e Inteligencia Local (PLC con IAs embebidas).
5. Capa de Persistencia (Base de Datos PostgreSQL).

Cada capa cumple funciones específicas dentro del flujo general de información, permitiendo la interacción sincrónica entre el usuario, el agente inteligente y el sistema físico.

#### 3.1.1.1. Capa de Usuario - Interfaz Web React

Esta capa constituye el punto de interacción directa entre el operador y el sistema. Su función principal es capturar instrucciones en lenguaje natural y mostrar los resultados procesados por las IAs en tiempo real ver 3.2.

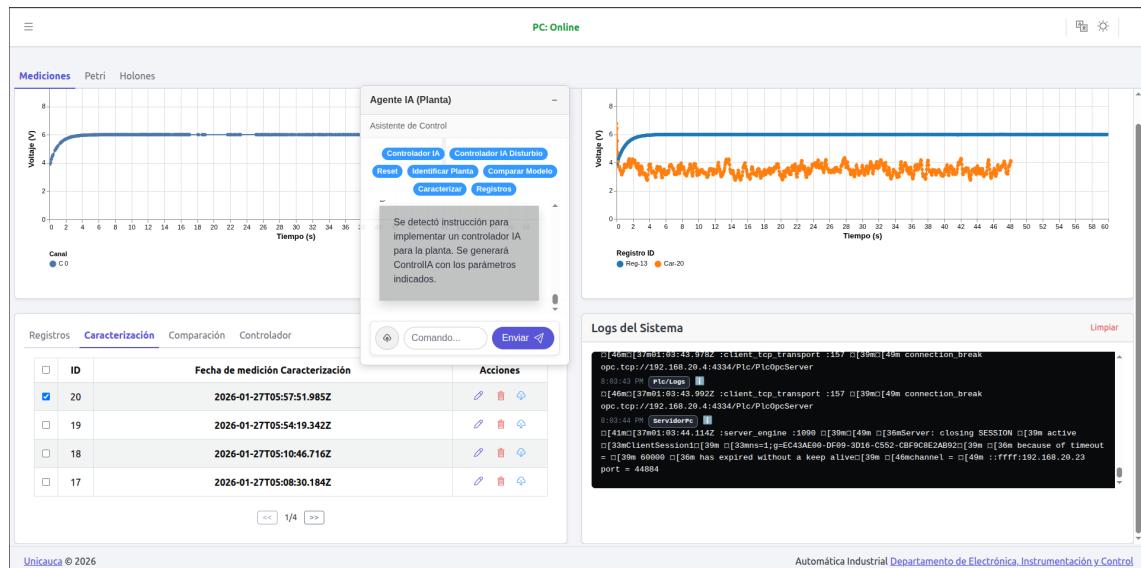


Figura 3.2: Interfaz de usuario.

#### Características principales

La interfaz gráfica está desarrollada en React.js, con soporte nativo para WebSocket, permitiendo una visualización dinámica y actualizaciones en tiempo real.

Sus principales capacidades incluyen:

- Envío de comandos textuales o estructurados hacia el sistema.
- Formularios contextuales configurables (voltajes, tiempos de muestreo, tipo de modelo, parámetros del controlador, etc.).
- Integración de gráficas temporales, indicadores de estado, métricas de desempeño, resultados de simulación y control.
- Componentes adaptativos que muestran únicamente las opciones relevantes según la acción que el usuario desea ejecutar.

Una característica clave es que no se requieren conocimientos técnicos sobre identificación, simulación o control automático.

Ejemplos de instrucciones que el usuario puede emitir directamente:

- “Ejecuta una nueva prueba de caracterización durante 60 segundos.”
- “Simula el modelo identificado con un escalón de entrada.”
- “Compara el control clásico con el control inteligente.”

La capa de usuario funciona como un panel intuitivo que abstrae la complejidad técnica del resto del sistema.

### **3.1.1.2. Capa de Servidor y Agente Central**

El servidor, implementado en Node.js, constituye el núcleo lógico del sistema. Este módulo gestiona:

- Las solicitudes provenientes de la interfaz web.
- Las conexiones bidireccionales vía WebSocket.
- La comunicación con el broker MQTT.
- La ejecución del Agente Principal de OpenAI, responsable de interpretar instrucciones en lenguaje natural.

#### **Funciones principales del Servidor**

- Recepción y parsing del mensaje enviado desde la interfaz React.
- Invocación del Agente OpenAI, que analiza la intención del usuario y determina la acción técnica correspondiente.
- Enrutamiento inteligente hacia MQTT según el tópico asociado:
  - /caracterizacion
  - /identificacion

- /simulacion
- /comparacion
- /controlador\_inteligente

- Gestión de resultados:

- El servidor se suscribe a los tópicos /resultados/\*.
- Recibe las respuestas generadas por las IAs locales o por el PLC.
- Retransmite los datos hacia el frontend mediante WebSocket, garantizando actualización en tiempo real.
- El servidor opera como un bus lógico central, coordinando a todos los agentes del ecosistema.
- El agente se implementa como un modelo de razonamiento natural (LLM) cuyo propósito es interpretar de forma semántica las instrucciones del usuario, identificar la intención subyacente y extraer los parámetros técnicos necesarios. A partir de este análisis, el agente genera un comando estructurado en formato SQL o como instrucción PLC que puede ser procesado directamente por los módulos operativos del sistema.
- El diagrama de flujo 3.3 asociado representa de manera concisa el funcionamiento lógico del servidor, mostrando cómo el agente recibe el prompt, determina el tipo de instrucción y dirige la petición hacia el módulo correspondiente.

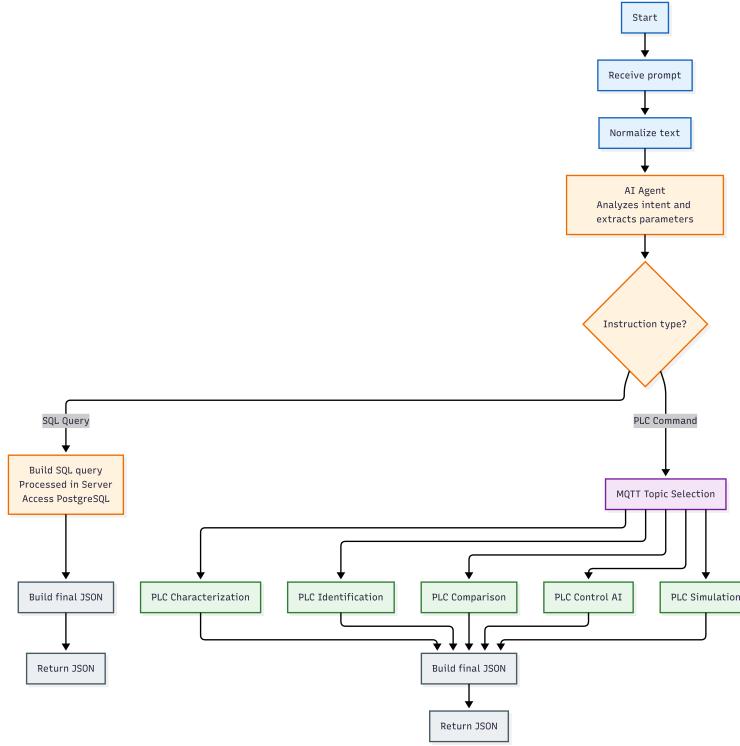


Figura 3.3: Agente IA

El algoritmo 3.1 propuesto constituye el núcleo lógico del sistema inteligente y define el proceso mediante el cual una instrucción en lenguaje natural es transformada en una acción técnica válida para los módulos inferiores. A partir del texto ingresado por el usuario, el algoritmo ejecuta una serie de etapas que incluyen la validación de la entrada, la normalización del contenido y la identificación de palabras clave que permiten clasificar la solicitud en uno de dos dominios operativos: consultas SQL hacia la base de datos PostgreSQL o instrucciones PLC destinadas a los agentes especializados de control, caracterización, identificación o comparación. Con base en esta clasificación, el algoritmo construye dinámicamente la respuesta adecuada, ya sea generando una consulta SQL completa o seleccionando el tópico MQTT correspondiente para el PLC. El resultado final es un objeto JSON limpio y estructurado que estandariza la comunicación entre el agente cognitivo y los subsistemas técnicos, garantizando coherencia, trazabilidad y ejecución autónoma de las tareas solicitadas.

---

### Algoritmo 3.1 Algoritmo Servidor IA.

---

```
1. Verificar validez:  
    Si prompt.text no es una cadena, retornar un JSON de error.  
2. Convertir texto a minusculas:  
    T = prompt.text en minusculas.  
3. Determinar modo SQL o PLC segun reglas generales:  
    Si T contiene alguna de:  
        "select", "tabla", "caracterizacion", "datalogger",  
        "consulta", "sql"  
        modo = "SQL"  
    Si T contiene alguna de:  
        "control", "simulacion", "canal", "adc",  
        "salida", "modelo"  
        modo = "PLC"  
4. Si modo = "SQL":  
    4.1 Construir estructura base:  
        json.conversacion = texto tecnico  
        json.resultado = []  
    4.2 Seleccionar tabla:  
        Si T contiene "caracterizacion":  
            tabla = "caracterizacion"  
        Si T contiene "datalogger":  
            tabla = "datalogger"  
        Si no:  
            Determinar contexto:  
            - si se pide identificar planta: tabla = "  
                caracterizacion"  
            - si se pide ultima medicion: tabla = "datalogger"  
    4.3 Generar consulta SQL:  
        Si T contiene "ultima":  
            Si tabla = "datalogger":  
                sql = "SELECT_*_FROM_datalogger_ORDER_BY_  
                    prueba_DESC LIMIT_1;"  
            Si tabla = "caracterizacion":  
                sql = "SELECT_*_FROM_caracterizacion ORDER  
                    _BY_prueba DESC LIMIT_1;"  
        Si T contiene "contar":  
            sql = "SELECT_COUNT(prueba)_FROM_" + tabla + ";"  
        Si T contiene "tiempo" o "voltaje" o "pwm":  
            sql = "SELECT_json_array_elements(resultado)_FROM_  
                " + tabla + ";"  
        Si T contiene "identificar_planta":  
            sql = "SELECT_*_FROM_caracterizacion ORDER_BY_prueba DESC LIMIT_1;"  
        Si no aplica ninguna regla anterior:  
            sql = "SELECT_*_FROM_" + tabla + ";"  
    4.4 Construir JSON final:  
        json.resultado = [  
            { "sql": sql, "prueba": tabla }  
        ]  
        Retornar json.  
5. Si modo = "PLC":  
    5.1 Construir estructura base:  
        json.conversacion = descripcion clara  
        json.resultado = []  
    5.2 Seleccionar topico MQTT:  
        Si T contiene "canal" o "adc" o "lee":  
            topic = "Plc/Adc"  
        Si T contiene "salida" o "q":  
            topic = "Plc/Ia"  
        Si T contiene "control" o "planta" o "simulacion":  
            topic = "Plc/Control"  
        Si T contiene "informe" o "reporte":  
            topic = "Plc/Supervisor"  
        Si T contiene "caracterizacion" o "llevar":  
            topic = "Plc/Caracterizacion"  
        Si T contiene "identifica" o "modelo":  
            topic = "Plc/Identificacion"  
        Si T contiene "compara_el_modelo":  
            topic = "Plc/Comparacion"  
        Si no coincide ninguna regla:  
            topic = "Plc/Otros"  
    5.3 Construir JSON:  
        json.resultado = [  
            { "topic": topic, "mensaje": prompt.text }  
        ]  
        Retornar json.  
Fin del algoritmo.
```

---

El código puede ser consultado en el repositorio ServidorIA

# Capítulo 4

## Capa Plc Físico

La capa del PLC constituye el nivel operativo donde se ejecutan las acciones físicas y los procesos autónomos de análisis local. En este nivel, el Raspberry PLC-58 se integra con un conjunto de módulos de inteligencia individual, cada uno especializado en una tarea específica del ciclo de identificación y control. Estas IAs locales incluyen los agentes de Caracterización, Identificación, Simulación, Comparación y Control Inteligente, los cuales procesan directamente las señales del sistema, ejecutan algoritmos embebidos y generan respuestas en tiempo real. Cada agente opera de forma independiente pero coordinada, respondiendo a los tópicos MQTT correspondientes y realizando funciones como adquisición automática de datos, estimación de modelos dinámicos, generación de escenarios de simulación, evaluación de desempeño entre planta real y modelo y ajuste adaptativo del controlador. De esta manera, la capa del PLC no solo actúa como interfaz con la planta física, sino que también incorpora capacidad de decisión distribuida, permitiendo un procesamiento descentralizado, rápido y eficiente que complementa la lógica cognitiva del servidor central.

### 4.1. IA: Caracterización

La primera inteligencia local implementada en la arquitectura del PLC es la IA de Caracterización, responsable de ejecutar las pruebas experimentales necesarias para obtener la respuesta dinámica de la planta física. Su función principal es aplicar señales de excitación controladas que permitan estimular adecuadamente el sistema sin generar perturbaciones excesivas.

- Entre estas señales, se utiliza una excitación compuesta por:
  - Un voltaje base constante de 5 V,
  - Más una componente aleatoria de  $\pm 0.5$  V, con probabilidad equiproporcional del 50 %,
  - Actualizada cada 50 milisegundos.

Esta estrategia produce un estímulo persistente y suficientemente rico en contenido frecuencial, adecuado para excitar diversas dinámicas del sistema y garantizar datos de calidad para el proceso de identificación.

Durante la prueba, la IA registra en tiempo real las variables relevantes del proceso —principalmente voltaje y otras magnitudes medidas por el PLC, procesa las señales adquiridas, organiza los datos en estructuras normalizadas y almacena los resultados en la base de datos. Esta etapa representa el punto de partida del ciclo completo del sistema inteligente: la precisión de la caracterización determina la calidad de los modelos en la fase de identificación y la validez de las simulaciones en etapas posteriores. Por esto, la IA de Caracterización se convierte en un componente esencial dentro de la plataforma.

#### **4.1.1. Agente de Caracterización en la Capa Física del PLC**

El agente de caracterización implementado directamente en el PLC tiene como propósito traducir instrucciones en lenguaje natural en un conjunto estructurado de parámetros numéricos necesarios para realizar la prueba de caracterización.

El módulo interpreta el texto proporcionado por el usuario, ejecuta un modelo LLM especializado y genera un objeto JSON normalizado con los siguientes parámetros:

- N: número de muestras o duración de la prueba.
  - PwmPin: canal PWM que genera la señal de excitación.
  - AdcPin: canal ADC encargado de la lectura.
  - Ts: tiempo de muestreo.
  - Offset: nivel base del voltaje.
  - Amplitud: variación máxima alrededor del offset.

El agente incorpora reglas explícitas para interpretar comandos simples del tipo “caracterizar la planta”, así como expresiones más complejas que incluyan rangos porcentuales o valores mínimo/máximo. A partir de estos rangos, el módulo calcula automáticamente los valores de offset y amplitud, asegurando consistencia y coherencia.

Además, el agente cuenta con mecanismos de:

- Corrección y normalización de claves,
  - Adaptación de porcentajes a valores físicos,
  - Asignación de valores por defecto cuando la instrucción es incompleta,
  - Validación de parámetros numéricos.

De esta forma, convierte descripciones verbales en instrucciones discretas listas para ser ejecutadas por el PLC, garantizando reproducibilidad, estandarización y robustez en el proceso de caracterización ver 4.1.

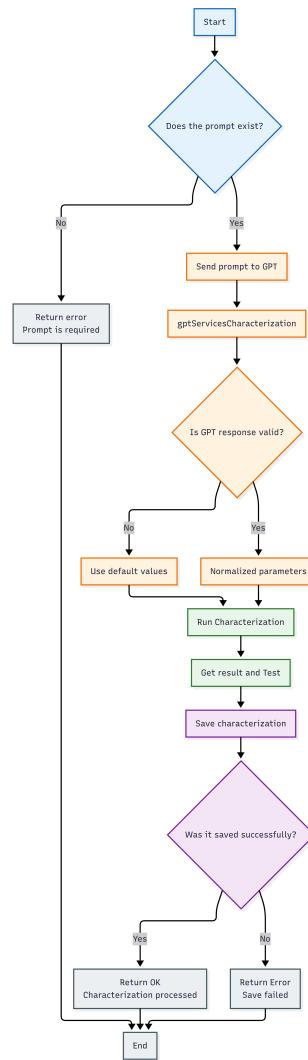


Figura 4.1: IA caracterización

El diagrama de bloques se articula con el algoritmo central del sistema, el cual establece la secuencia lógica que permite transformar instrucciones en

lenguaje natural en acciones formales y ejecutables. El proceso inicia con la captura y normalización del mensaje del usuario, tras lo cual un agente de IA analiza la intención ver 4.2, extrae parámetros relevantes y determina el tipo de instrucción.

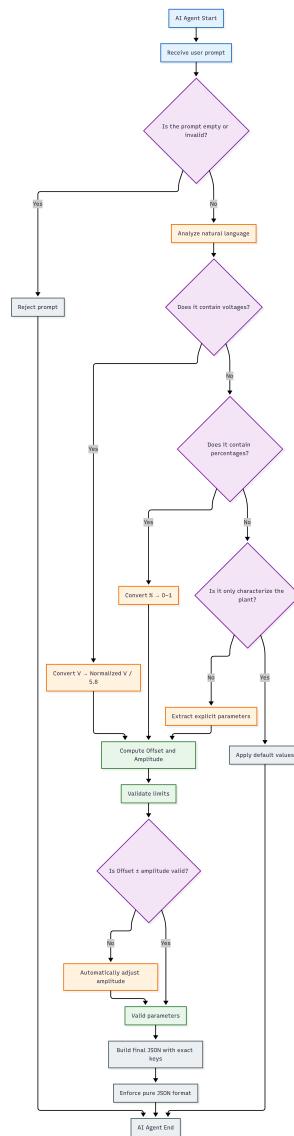


Figura 4.2: Agente de caracterización

A partir de esta decisión, el algoritmo se bifurca en dos rutas: en la primera, orientada a consultas SQL, se construye automáticamente la sentencia optimizada para PostgreSQL y se encapsula en un JSON estructurado; en la segunda, destinada a la operación del PLC, se selecciona el tópico MQTT correspondiente (caracterización, identificación, comparación, control o simulación) y se genera un JSON con los parámetros numéricos requeridos para la ejecución física de la tarea. Este mecanismo garantiza que el sistema convierta peticiones verbales en instrucciones precisas, reproducibles y estandarizadas, integrando en un mismo flujo la comprensión semántica, la traducción de parámetros y la activación de procesos tanto digitales como experimentales ver 4.1.

---

**Algoritmo 4.1** Algoritmo Caracterización IA.

---

Input :  
    prompt : string

Output :  
    JSON { N, PwmPin, AdcPin, Ts, Offset , amplitud }

1. if prompt is null or empty then  
2.     throw "Invalid prompt"  
3. end if  
4. Send prompt to LLM with instructions:  
    – Output only valid JSON  
    – Allowed keys: N, PwmPin, AdcPin, Ts, Offset , amplitud  
    – Values normalized in [0,1]  
    – If prompt indicates characterization only, use default values

5. rawJson = receive response from LLM  
6. try parse rawJson as JSON  
7. if parsing fails then  
8.     clean rawJson (remove extra text, fix syntax)  
9.     retry JSON parsing  
10.    if parsing fails again then  
11.       rawJson = default JSON  
12.    end if  
13. end if  
14. Normalize JSON keys:  
    – Map synonyms to standard keys  
    – Remove unknown keys

15. Normalize JSON values:  
    – Convert percentages to [0,1]  
    – Convert numeric strings to numbers  
    – Round Offset and amplitud to 3 decimals

16. Search prompt for range patterns "X% to Y%"  
17. if range is detected then  
18.     min = X / 100  
19.     max = Y / 100  
20.     Offset = (min + max) / 2  
        amplitud = (max - min) / 2

21. end if  
22. Fill missing keys with default values:  
    N           = 1000  
    PwmPin     = 0  
    AdcPin     = 0  
    Ts          = 50  
    Offset     = 0.5  
    amplitud = 0.1

23. Validate physical limits:  
24. if Offset + amplitud > 1 then  
25.     amplitud = 1 - Offset  
26. end if  
27. if Offset - amplitud < 0 then  
28.     amplitud = Offset  
29. end if  
30. return final validated JSON

---

#### **4.1.2. Resultados Agente Caracterización**

En este apartado se presentan los resultados generados por la IA de Caracterización, cuyo funcionamiento se basa en el envío de un prompt desde la interfaz gráfica del sistema. Dicho mensaje es recibido y procesado por el servidor, que se encarga de interpretar la instrucción, extraer los parámetros pertinentes y construir el conjunto de valores normalizados necesarios para ejecutar la prueba experimental en el PLC. De esta manera, se establece un flujo integrado entre la interacción del usuario, el procesamiento semántico realizado por la IA y la ejecución física de la señal de excitación, permitiendo validar la correcta traducción de comandos en lenguaje natural hacia acciones técnicas reproducibles en la planta.

#### **4.1.3. Promt: Caracteriza y utiliza como entrada a la planta una secuencia binaria aleatoria que oscile entre el 45 % y el 55 % de la señal de entrada -Captura 1000 muestras.**

Interpretación:

- tipo: Señal binaria aleatoria
  - min: Valor mínimo en porcentaje (45 %)
  - max: Valor máximo en porcentaje (55 %)
  - muestras: Número total de muestras → N = 1000
  - ts\_ms: Tiempo entre muestras → Ts = 50 ms

Respuestas del Agente: Para realizar la caracterización de la planta con una secuencia binaria aleatoria que oscile entre el 45 % y el 55 % de la señal de entrada, se debe configurar el PLC para generar dicha secuencia y aplicarla a la planta. Además, se capturarán 5000 muestras con un tiempo de muestreo de 50 ms, lo que permitirá analizar la respuesta dinámica bajo estas condiciones.

- Según las reglas del agente:
  - Normalizar los porcentajes (min y max)
    - $minNorm = 45\% = 0,45$ .
    - $maxNorm = 55\% = 0,55$ .
  - Cálculo del Offset y la amplitud (regla corregida)
    - $Offset = \frac{0,45+0,55}{2} = 0,50$
    - $Amplitud = \frac{0,55-0,45}{2} = 0,05$

- Esto asegura que la señal binaria oscile entre 0.45 y 0.55 (es decir, 45 % y 55 % del rango 0–8.8 V), como lo pide el comando json generado es:

```
{
  "N": "5000",
  "PwmPin": "0",
  "AdcPin": "0",
  "Ts": "0.05",
  "Offset": "0.50",
  "amplitud": "0.05"
}
```

Basada en

```
const duty = offset + amplitud * signo;
```

Dos posibles valores:

- $duty = 0.50 + 0.05 = 0.55$ .
- $duty = 0.50 - 0.05 = 0.45$ .

En voltaje real (0–10V):

- $V = duty \times 8.8 \rightarrow 4.84 \text{ V o } 3.96 \text{ V}$ .

En PWM 12 bits:

- $0.55 \rightarrow 2252$
- $0.45 \rightarrow 1843$

Duración total de la prueba:

- $N = 1000, T_s = 50ms$
- $1000 * 50ms = 50000ms = 00 : 50s$ .

El resultado de la prueba se puede ver en la figura 4.3.

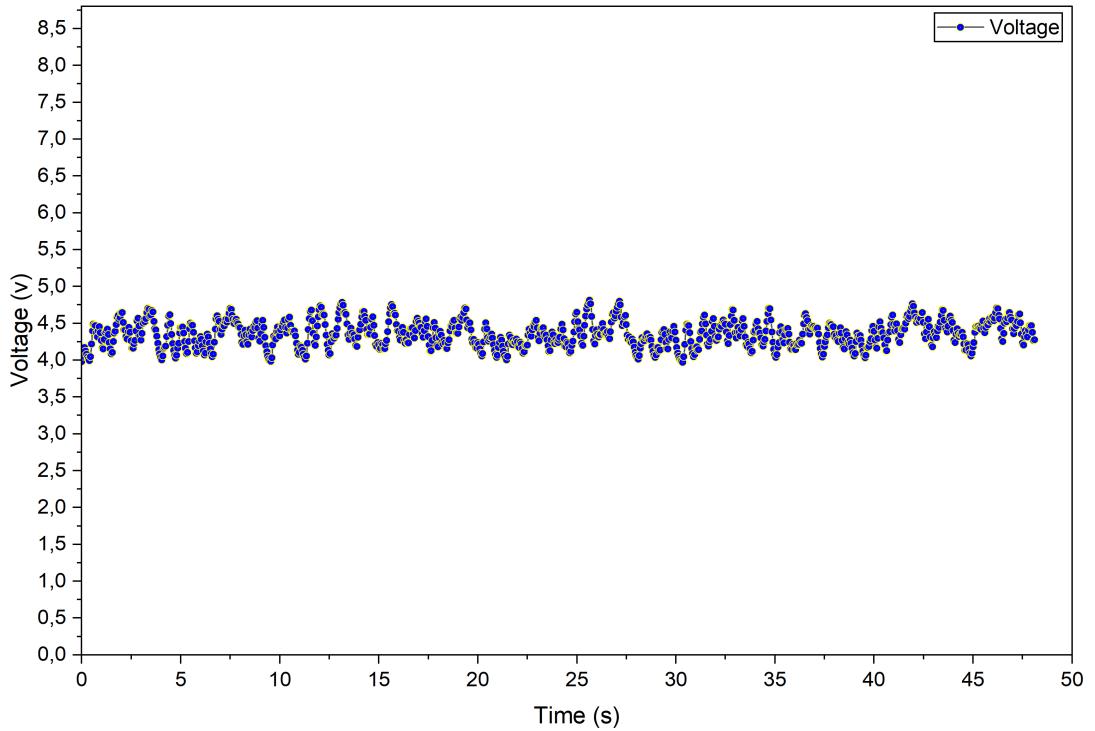


Figura 4.3: Caracterización 5000 muestras, secuencia binaria aleatoria entre el 45 % y el 55 % de la señal de entrada, tiempo de muestreo 50 ms.

**4.1.4. Promt:** Usa una señal entre 4 V y 8 V para caracterizar la planta, con 1000 muestras y un tiempo de muestreo 100 ms.

Interpretación:

- tipo: Señal binaria aleatoria
  - min: Valor mínimo 4V.
  - max: Valor máximo 8V.
  - muestras: Número total de muestras →  $N = 1000$
  - ts\_ms: Tiempo entre muestras →  $T_s = 100$  ms

Respuestas del Agente: Para caracterizar la planta con una señal entre 4 V y 8 V, tomando 1000 muestras y un tiempo de muestreo de 100 ms, se debe

enviar una instrucción al PLC para iniciar el proceso de caracterización bajo estos parámetros. El PLC generará la señal, recogerá los datos y almacenará la información para su posterior análisis.

- Según las reglas del agente:

- Normalizar los porcentajes (min y max)

- $minNorm = \frac{4}{8,8} = 0,455$ .

- $maxNorm = \frac{8}{8,8} = 0,909$ .

- Cálculo del Offset y la amplitud (regla corregida)

- $Offset = \frac{0,455+0,909}{2} = 0,682$

- $Amplitud = \frac{0,909-0,455}{2} = 0,227$

- Esto significa que la señal binaria va a oscilar simétricamente alrededor de 0.682, con una amplitud de 0.227 → niveles 0.40 y 0.80.

```
{
"N": "1000",
"PwmPin": "0",
"AdcPin": "0",
"Ts": "100",
"Offset": "0.681",
"amplitud": "0.227"
}
```

Basada en

```
const duty = offset + amplitud * signo;
```

Dos posibles valores:

- $duty = 0.682 + 0.227 = 0.909$ .
- $duty = 0.682 - 0.227 = 0.455$ .

En voltaje real (0–8.8V):

- $V = duty \times 8.8 \rightarrow 8 \text{ V o } 4 \text{ V}$ .

En PWM 12 bits:

- $0.909 \rightarrow 3723$
- $0.455 \rightarrow 1863$

Duración total de la prueba:

- $N = 1000, T_s = 100ms$
- $1000 * 100ms = 100s = 1 : 40m$ .

El resultado de la prueba se puede ver en la figura 4.4.

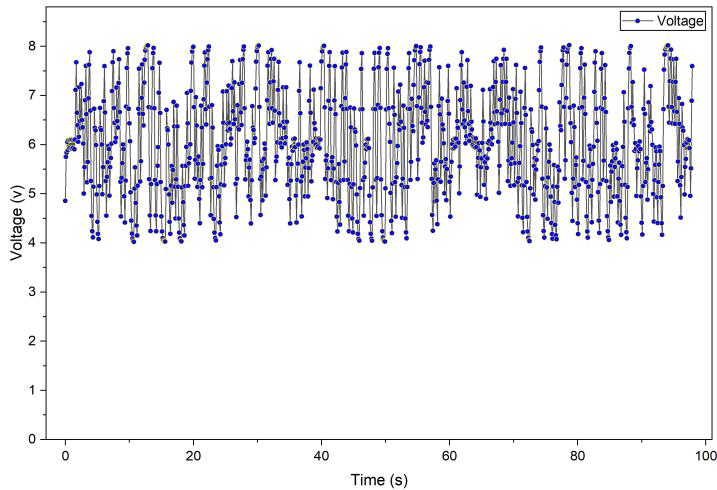


Figura 4.4: Caracterización 1000 muestras, señal entre 4 V y 8 V tiempo de muestreo 100 ms.

#### 4.1.5. Promt:Excita la planta con una secuencia que oscile entre el 10 % y el 20 %.

Dado que la instrucción no especifica ni el número de muestras ni el tiempo de muestreo, el sistema aplica automáticamente los valores por defecto, que son: N = 1000 muestras y Ts = 50 ms.

Interpretación:

- tipo: Señal binaria aleatoria
  - min: Valor mínimo 0.88V.
  - max: Valor máximo 1.76V.
  - muestras: Número total de muestras → N = 1000
  - ts\_ms: Tiempo entre muestras → Ts = 50 ms

Respuestas del Agente: Para excitar la planta con una secuencia que oscile entre el 10 % y el 20 % del rango de la señal de control, se debe enviar una instrucción al PLC especificando estos valores de PWM. Esto permitirá realizar pruebas de caracterización en ese rango de operación.

Se debe tener

- Según las reglas del agente:
  - Normalizar los porcentajes (min y max)
    - $minNorm = \frac{1}{8,8} = 0,114$ .
    - $maxNorm = \frac{2}{8,8} = 0,227$ .
  - Cálculo del Offset y la amplitud (regla corregida)
    - $Offset = \frac{0,114+0,227}{2} = 0,1705$
    - $Amplitud = \frac{0,227-0,114}{2} = 0,0565$
- Esto significa que la señal binaria va a oscilar simétricamente alrededor de 0.65, con una amplitud de 0.25 → niveles 0.40 y 0.90.

```
{
  "N": "1000",
  "PwmPin": "0",
  "AdcPin": "0",
  "Ts": "50",
  "Offset": "0.15",
  "amplitud": "0.05"
}
```

Basada en

```
const duty = offset + amplitud * signo;
```

Dos posibles valores:

- $duty = 0.1705 + 0.0565 = 0.227$ .
- $duty = 0.1705 - 0.0565 = 0.114$ .

En voltaje real (0–8.8V):

- $V = duty \times 8.8 \rightarrow 2 \text{ V o } 1 \text{ V}$ .

En PWM 12 bits:

- $0.2 \rightarrow 930$
- $0.1 \rightarrow 467$

Duración total de la prueba:

- $N = 1000, T_s = 50ms$
- $1000 * 50ms = 50s = 0 : 50m$ .

El resultado de la prueba se pude ver en la figura 4.5.

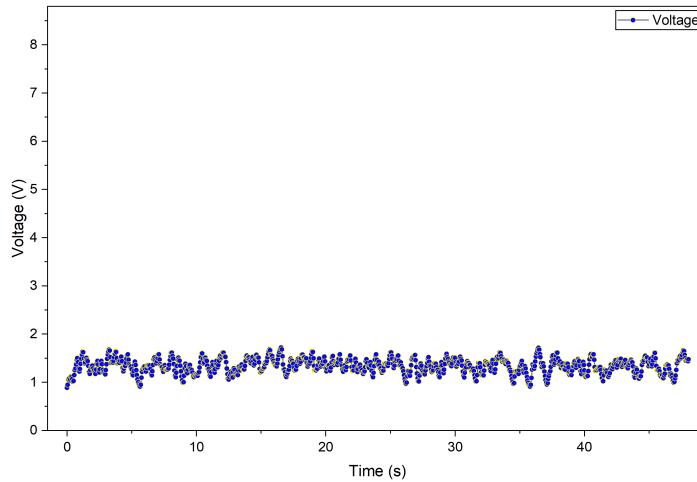


Figura 4.5: Excitación de la planta con una secuencia que oscila entre el 10 % y el 20 %.

## 4.2. Conclusiones del capítulo

- En este capítulo se desarrolló y validó el proceso de caracterización de la planta mediante una señal binaria aleatoria, integrando la interpretación de instrucciones en lenguaje natural con la generación automática de parámetros de excitación adecuados para el sistema físico.
- Se estableció un mecanismo robusto de interpretación que permite al agente identificar correctamente los rangos de voltaje solicitados por el usuario, incluso cuando la instrucción no especifica explícitamente parámetros como el número de muestras o el tiempo de muestreo, aplicando en estos casos valores por defecto coherentes con el sistema.
- La normalización de la señal se realizó considerando el voltaje máximo real del sistema (8.8 V), lo que garantiza que los valores de duty cycle generados correspondan fielmente a los niveles de voltaje físicos aplicados a la planta. A partir de esta normalización, se definieron correctamente el offset y la amplitud, asegurando una excitación simétrica dentro del rango operativo deseado.
- Asimismo, se verificó la correcta conversión entre los distintos dominios de representación de la señal —voltaje real, duty cycle normalizado y PWM de 12 bits— garantizando la consistencia matemática y física del proceso

de caracterización. La duración total del experimento fue determinada de forma precisa a partir del número de muestras y el tiempo de muestreo, permitiendo un control temporal adecuado de la prueba.

- En conjunto, los resultados obtenidos demuestran que el enfoque propuesto permite automatizar de manera confiable la generación de señales de caracterización, reduciendo errores de configuración, mejorando la reproducibilidad de los ensayos y sentando una base sólida para etapas posteriores de identificación y modelado de la planta

# Capítulo 5

## Identificación

La identificación de sistemas es un área fundamental de la ingeniería de control cuyo objetivo es obtener modelos matemáticos que representen el comportamiento dinámico de un proceso físico a partir de datos experimentales. Estos modelos constituyen la base para tareas de análisis, simulación, diseño de controladores y validación de estrategias de automatización en sistemas reales.

Entre los enfoques clásicos de identificación, los modelos del tipo ARX estimados mediante técnicas de mínimos cuadrados ofrecen una formulación matemática bien establecida, con propiedades de optimalidad, reproducibilidad e interpretabilidad ampliamente documentadas en la literatura. En este contexto, la estimación de los coeficientes del modelo se realiza de manera determinista y directamente a partir de las señales medidas del sistema, garantizando trazabilidad y consistencia en los resultados obtenidos.

No obstante, una de las principales dificultades del proceso de identificación radica en la selección del modelo más adecuado cuando se dispone de múltiples candidatos con diferentes órdenes y métricas de desempeño similares. Esta etapa implica un compromiso entre precisión, estabilidad y complejidad estructural, cuya evaluación tradicionalmente depende del criterio y la experiencia del ingeniero encargado del análisis.

Con el fin de abordar esta problemática, el presente capítulo propone un esquema de identificación híbrido en el cual el cálculo del modelo se mantiene estrictamente dentro del marco de la identificación matemática clásica, mientras que un agente de inteligencia artificial generativa actúa como evaluador experto del conjunto de modelos candidatos. En este enfoque, la inteligencia artificial no interviene en la estimación de los parámetros del modelo, sino que analiza los resultados obtenidos para determinar la idoneidad de cada candidato de acuerdo con criterios técnicos propios de la ingeniería de control.

El agente inteligente evalúa aspectos como la estabilidad del modelo, el principio de parsimonia y la mejora relativa de las métricas de error, emulando el razonamiento de un especialista humano en identificación de sistemas. Esta separación explícita de responsabilidades preserva el rigor matemático del proceso de cálculo, al tiempo que introduce un mecanismo sistemático y reproducible

de toma de decisiones en la etapa de selección del modelo.

Finalmente, el procedimiento descrito culmina con la generación automática del modelo seleccionado en una forma directamente utilizable para simulación y control, cerrando el ciclo completo de identificación desde los datos experimentales hasta la obtención de un modelo operativo validado ver fig 5.1.

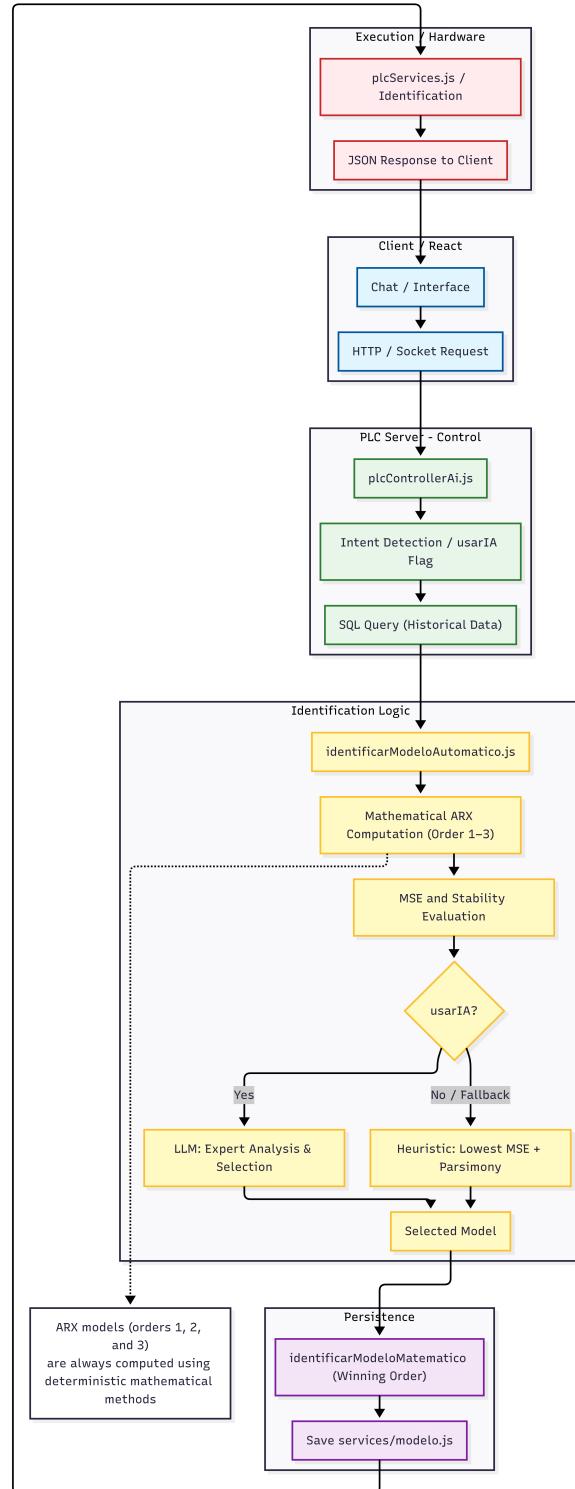


Figura 5.1: Modelo agente IA

## 5.1. IA: Identificación

Se ha implementado un método de identificación de sistemas asistido por inteligencia artificial cuyo objetivo es automatizar la selección y validación del modelo matemático que representa el comportamiento dinámico de una planta a partir de datos experimentales. El proceso de identificación se fundamenta en técnicas clásicas de identificación de sistemas, específicamente en la estimación de modelos ARX mediante métodos matemáticos deterministas basados en mínimos cuadrados, lo que garantiza la trazabilidad, reproducibilidad e interpretabilidad de los parámetros obtenidos.

Los datos experimentales recolectados desde la planta incluyen la señal de entrada aplicada al actuador (PWM), la señal de salida medida (voltaje) y el instante temporal asociado a cada muestra. Dichos registros son sometidos a un proceso de depuración, normalización y estimación del tiempo de muestreo, con el fin de asegurar condiciones adecuadas para la correcta identificación matemática del sistema.

A partir de estos datos preprocesados, se calculan de manera sistemática múltiples modelos ARX de diferente orden (primer, segundo y tercer orden), cuyos coeficientes son obtenidos exclusivamente mediante procedimientos analíticos. Para cada modelo candidato se evalúan métricas de desempeño tales como el error de predicción, la varianza residual y la estabilidad del sistema identificado, permitiendo una comparación objetiva entre las distintas alternativas.

En una etapa posterior, un agente inteligente basado en modelos de lenguaje avanzados (LLM) interviene únicamente como evaluador experto, analizando el conjunto de modelos previamente calculados mediante métodos matemáticos clásicos. Esta restricción es deliberada y responde a consideraciones fundamentales de rigor científico y confiabilidad del proceso de identificación. La estimación de los coeficientes de un modelo ARX requiere operaciones numéricas precisas, reproducibles y verificables —como la resolución de sistemas de ecuaciones por mínimos cuadrados— cuya correcta ejecución depende de la exactitud aritmética y de la estabilidad numérica del algoritmo empleado.

Los modelos de lenguaje, debido a su naturaleza probabilística y generativa, no garantizan consistencia numérica ni reproducibilidad exacta en cálculos matemáticos continuos, especialmente en procedimientos iterativos o sensibles al ruido presente en los datos experimentales. En consecuencia, delegar la estimación paramétrica a un LLM introduciría incertidumbre no cuantificable en los coeficientes del modelo, comprometiendo la validez técnica del resultado y dificultando su verificación posterior.

Por esta razón, el cálculo del modelo se mantiene estrictamente dentro de un marco matemático determinista, mientras que la intervención del LLM se limita a una etapa de análisis cualitativo y toma de decisiones. En este rol, el agente inteligente evalúa los modelos candidatos aplicando criterios técnicos como estabilidad, parsimonia estructural y mejora relativa del error, emulando el

razonamiento que tradicionalmente realiza un ingeniero en control al seleccionar el modelo más adecuado para representar la dinámica de la planta.

Esta separación funcional entre el cálculo matemático y la evaluación experta permite combinar las ventajas de ambos enfoques: por un lado, la precisión y robustez de los métodos clásicos de identificación de sistemas; por otro, la capacidad de razonamiento de alto nivel y síntesis de información compleja proporcionada por la inteligencia artificial. De este modo, el uso del LLM no sustituye el fundamento matemático del proceso, sino que lo complementa, proporcionando un mecanismo sistemático, reproducible y técnicamente justificable para la selección del modelo óptimo.

Finalmente, una vez seleccionado el modelo más adecuado, el sistema genera automáticamente un archivo de código JavaScript completamente funcional que contiene los coeficientes del modelo elegido y una función discreta capaz de simular la respuesta de la planta ante señales de entrada arbitrarias. Esta implementación permite integrar el proceso de identificación dentro de arquitecturas de software distribuidas, sin depender de herramientas especializadas como MATLAB o Python, facilitando su aplicación en entornos embebidos, sistemas en tiempo real y esquemas de actualización dinámica del modelo conforme se adquieran nuevos datos experimentales.

## 5.2. Etapas del proceso de identificación de sistemas asistida por IA

### Interpretación de la Solicitud de Identificación Servidor

El proceso inicia con la interpretación de la solicitud del usuario expresada en lenguaje natural. En esta etapa, un modelo de lenguaje analiza la intención semántica del comando para determinar que la acción requerida corresponde a un proceso de identificación de sistemas, diferenciándola de otras acciones como caracterización, control o comparación. El resultado de esta etapa es un comando estructurado en formato JSON el cual se encuentra en el servidor principal que especifica explícitamente que se debe ejecutar una identificación del sistema, incluyendo parámetros como el uso de inteligencia artificial y el orden del modelo ver fig 5.2.

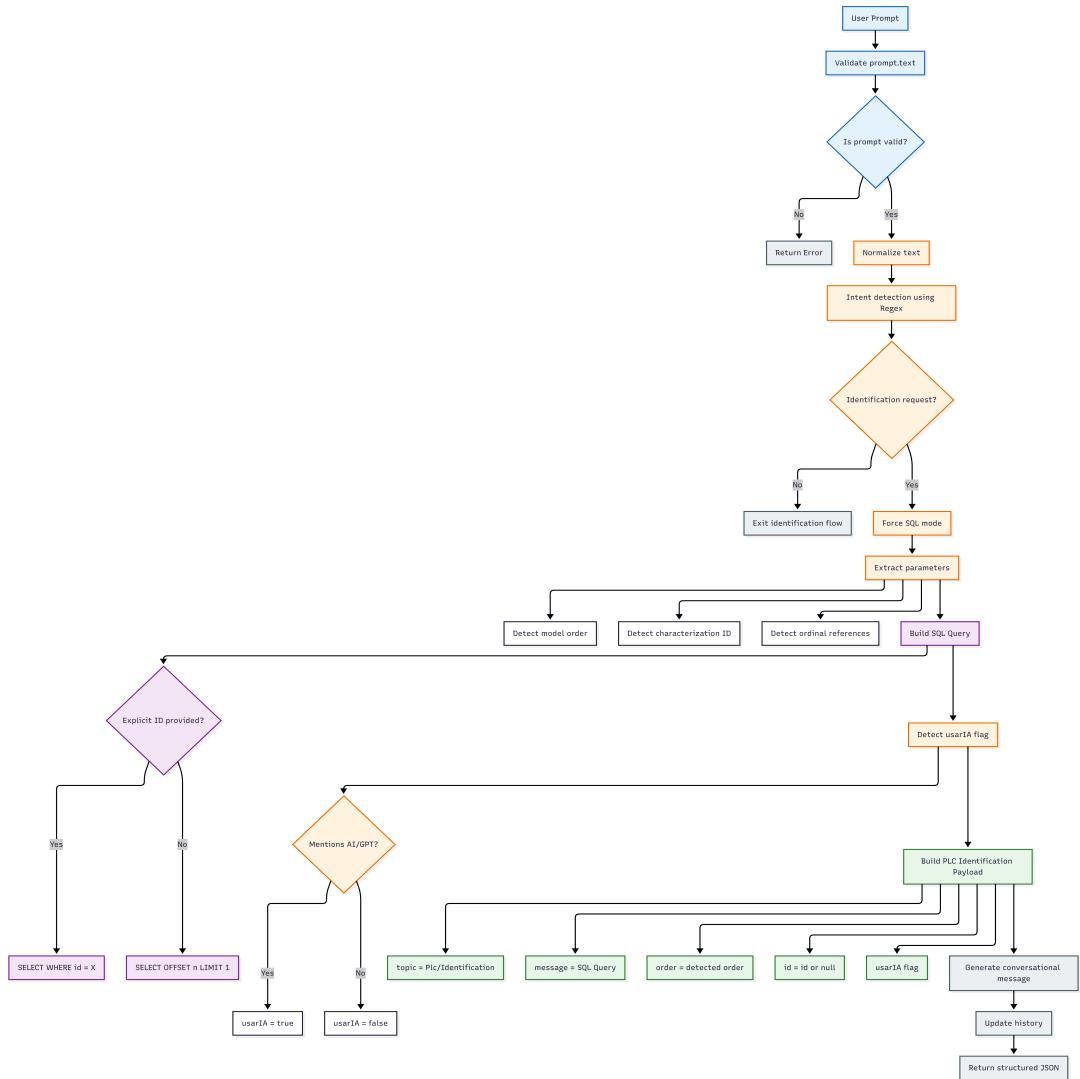


Figura 5.2: Natural language interpretation pipeline for plant identification

### Activación del Modo de Identificación Plc

Una vez generado el comando estructurado, el sistema activa el modo de identificación. Este modo define el flujo de procesamiento que se seguirá, estableciendo si la identificación se realizará mediante métodos matemáticos clásicos o mediante asistencia de inteligencia artificial generativa. En el caso de identificación asistida por IA, el sistema prepara los datos y los modelos candidatos

para ser evaluados por el modelo experto ver fig 5.3.

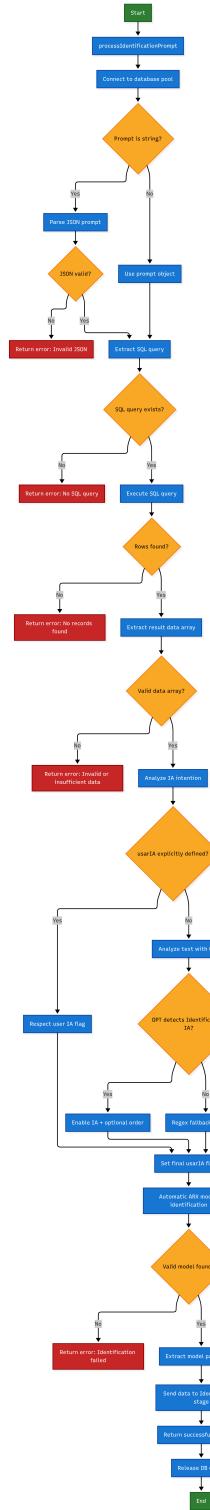


Figura 5.3: Preprocesamiento de datos y agente IA Plc

## Adquisición y Preprocesamiento de Datos

En esta etapa se recopilan los datos experimentales provenientes de la planta real o del proceso físico bajo estudio. Los datos incluyen señales de entrada (actuación) y salida (respuesta del sistema), así como marcas temporales ver . Posteriormente, los datos son preprocesados para ver alg 5.1:

- Eliminar valores inválidos o inconsistentes.
- Normalizar las señales de entrada y salida.
- Calcular el tiempo de muestreo promedio del sistema.
- Descartar transitorios iniciales que puedan sesgar la identificación.
- Este paso garantiza que los datos utilizados sean adecuados para el proceso de modelado.

---

### **Algoritmo 5.1 Data Preprocessing.**

---

```
ALGORITHM PreprocessData
INPUT:      data
OUTPUT:
           normalizedData , Ts
BEGIN
    Remove invalid samples (NaN)
    Compute average sampling time Ts
    FOR each sample DO
        u <- normalized input
        y <- normalized output
        Store (u, y)
    END FOR
    Return normalizedData and Ts
END
```

---

## Generación de Modelos Candidatos

Con los datos preprocesados, se generan múltiples modelos matemáticos candidatos del tipo ARX, cada uno con un orden diferente previamente definido. Cada modelo es calculado mediante técnicas de mínimos cuadrados ver 5.2, obteniendo:

- Coeficientes del modelo.

---

**Algoritmo 5.2** Mathematical ARX Identification (Fallback).

---

```
ALGORITHM IdentifyARXModelMathematical
INPUT:
    data : experimental samples
    order : ARX model order
OUTPUT:
    coefficients , Ts, variance
BEGIN
    Validate that order is 1, 2, or 3
    Validate that data contains at least 50 samples
    (normalizedData , Ts) <- PreprocessData(data)
    Remove initial transient (first 20 percent)
    Build regression matrices Phi and Y
    theta <- SolveLeastSquares(Phi, Y)
    coefficients <- MapThetaToARX(theta , order)
    errors <- ComputePredictionErrors(normalizedData , coefficients , order)
    (variance , autocorr) <- ComputeMetrics(errors)
    Generate model.js file
    Return results
END
```

---

- Métricas de error y varianza.
- Información de estabilidad y comportamiento dinámico.

El objetivo de esta etapa es producir un conjunto de modelos viables que representen el sistema desde diferentes niveles de complejidad. Evaluación Experta Mediante Inteligencia Artificial. Los modelos candidatos generados son enviados a un modelo de lenguaje especializado, el cual actúa como 5.4 experto en identificación de sistemas y control automático. El modelo de IA analiza cada candidato considerando criterios técnicos como:

- Estabilidad del modelo.
- Compromiso entre precisión y simplicidad (parsimonia).
- Mejora marginal del error frente al aumento del orden.
- Coherencia con el tiempo de muestreo del sistema.

Como resultado, la IA selecciona el modelo que ofrece el mejor equilibrio entre exactitud y robustez

## **Selección del Modelo Óptimo.**

A partir del análisis experto, se selecciona formalmente el modelo considerado óptimo. Esta selección se basa en una decisión explícita que identifica el orden del modelo y su posición dentro del conjunto de candidatos evaluados. El modelo seleccionado es validado internamente para asegurar consistencia estructural y compatibilidad con el sistema de simulación y control.

## **Generación del Modelo Operativo**

El modelo óptimo es transformado en una representación computacional ejecutable. Se genera una función dinámica que implementa el modelo ARX, incluyendo:

- Manejo de estados anteriores.
- Actualización de entradas y salidas.
- Coeficientes identificados.

Esta función queda lista para ser utilizada en simulación, validación o diseño de controladores ver 5.4.

---

**Algoritmo 5.3** IA identificación.

---

```
ALGORITHM SelectBestARXModelAI
INPUT:
    candidates : list of ARX models
    Ts          : sampling time
OUTPUT:
    selected model or error
BEGIN
    Build PROMPT with:
        - objective description
        - stability and parsimony criteria
        - list of candidate models
        - sampling time Ts
    Send PROMPT to AI model
    Receive response in JSON format

    IF response is invalid THEN
        return error
    END IF
    Extract selected model index or order
    Retrieve model from candidate list

    IF model does not exist THEN
        return error      END IF
    Generate model.js using:
        - model order
        - coefficients
        - Ts
        - variance
    Return selected model and AI analysis END
```

---

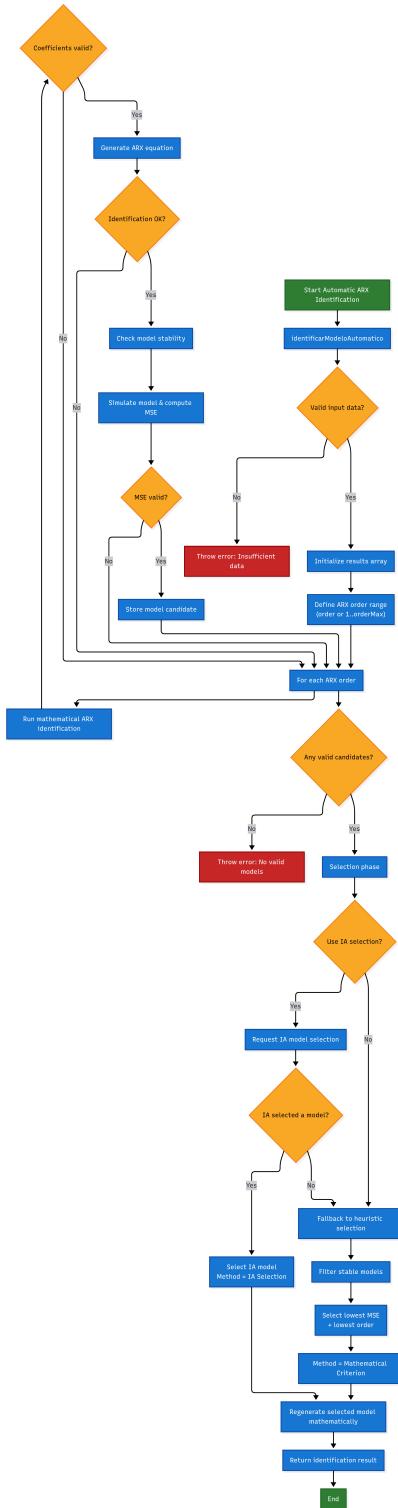


Figura 5.4: Modelos Candidatos

## Cierre del Ciclo de Identificación

Finalmente, el proceso de identificación concluye almacenando el modelo generado y sus parámetros asociados. El sistema queda preparado para etapas posteriores como validación, comparación con la planta real o integración en lazo cerrado.

Este cierre garantiza la trazabilidad del proceso y la reutilización del modelo identificado en ciclos futuros.

## Ciclo de comparación

Es importante destacar que cada vez que se ejecuta el agente de identificación, el usuario tiene la posibilidad de seleccionar previamente la prueba sobre la cual se realizará dicho proceso. Esta prueba proviene de un paso anterior denominado Caracterización IA, donde otro agente se encarga de generar la señal de excitación adecuada, ejecutar la prueba sobre la planta física y almacenar sus resultados en la base de datos. De esta forma, el usuario puede elegir desde la interfaz cuál de las pruebas almacenadas desea usar para la identificación del modelo. Una vez seleccionada, el agente de identificación procesa los datos asociados a esa prueba y genera automáticamente una función que representa el modelo dinámico del sistema, con una estructura flexible adaptada al orden solicitado (es decir, modelos de primer, segundo o tercer orden).

Esta capacidad de seleccionar pruebas y generar modelos bajo demanda facilita la actualización y la precisión del sistema de control en función de las condiciones reales de operación de la planta. Como parte del proceso de validación del modelo matemático identificado, se ha desarrollado un agente adicional denominado Agente de Comparación 5.5, cuyo propósito es automatizar la ejecución de pruebas controladas en la planta física para evaluar el desempeño del modelo frente a datos reales ver 5.4. Este agente está implementado en el servicio gtpServicesCompara, y se encarga de interpretar instrucciones humanas y traducirlas a comandos estructurados en formato JSON compatibles con el control del PLC. La funcionalidad principal de este agente consiste en diseñar secuencias de prueba en las cuales se aplica una entrada controlada al sistema por ejemplo, variaciones porcentuales de la señal PWM durante intervalos de tiempo definidos y luego comparar la respuesta medida con la respuesta simulada por el modelo identificado. Gracias a esta estructura flexible, el usuario puede generar pruebas de comparación personalizadas, que permiten evaluar visualmente y numéricamente la capacidad del modelo para reproducir el comportamiento dinámico de la planta bajo distintas condiciones de operación.

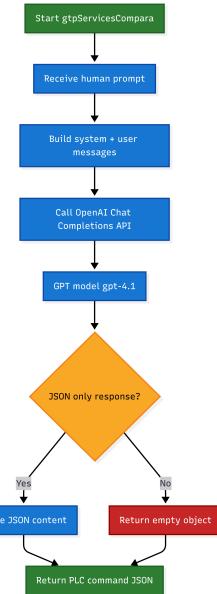


Figura 5.5: Diagrama de Flujo para la Ejecución de Pruebas de Comparación.

---

#### **Algoritmo 5.4** Algoritmo de Comparación Planta-Modelo

---

```

ALGORITHM GtpServicesCompara
INPUT:
    prompt
OUTPUT:
    plcCommandJSON
BEGIN
    Build system message with:
        - PLC command translation rules
        - Comparison command schema
        - Field normalization rules
    Build user message using prompt
    Send messages to GPT model (JSON-only mode)
    Receive model response
    IF response is empty THEN
        RETURN empty JSON
    END IF
    Parse response content as JSON
    RETURN plcCommandJSON
END

```

---

## 5.3. Resultados Agente de Identificación

Para asegurar la validez del modelo identificado y verificar que representa adecuadamente la dinámica de la planta física, se realizarán pruebas comparativas entre la respuesta simulada del modelo y los datos reales obtenidos durante la prueba seleccionada. Este proceso consiste en aplicar al modelo ARX identificado la misma señal de entrada utilizada originalmente (PWM) y comparar la salida generada por el modelo con la señal de voltaje medida experimentalmente. La evaluación incluirá métricas de desempeño como el error cuadrático medio (ECM) y análisis visual mediante la superposición de ambas señales en el dominio del tiempo. De esta forma, se podrá determinar el grado de precisión del modelo, identificar posibles desviaciones dinámicas y establecer si el modelo es apto para tareas de simulación o control predictivo dentro del sistema.

### 5.3.1. Promt:Identifica la planta

Respuestas del Agente: El modelo de orden 1 es estable, tiene un error cuadrático medio (MSE) bajo y satisface el criterio de parsimonia. Dado que no se presentan otros modelos, su simplicidad y adecuada precisión lo hacen la elección más razonable ver .

---

#### Algoritmo 5.5 Identificación con modelo Orden 1.

---

```
// Orden: 1 | Ts: 0.050000 s | Var: 5.961e-5
let y_1; let u_1;
y_1 = 0; u_1 = 0;
function modeloPlanta(u) {
    const y = (0.929461 * y_1) + (-0.003101 * u) + (0.073717 * u_1);
    y_1 = y;
    u_1 = u;
    return y;
}
const coeficientes = {
    "a1": 0.9294614887898411,
    "b0": -0.0031007564915783107,
    "b1": 0.07371733941180511
};
```

---

**Promt:Compara el modelo 15 % 5s, 20 % 5s, 25 % 15s ver figura 5.6**

La gráfica 5.6 representa la prueba desarrollada.

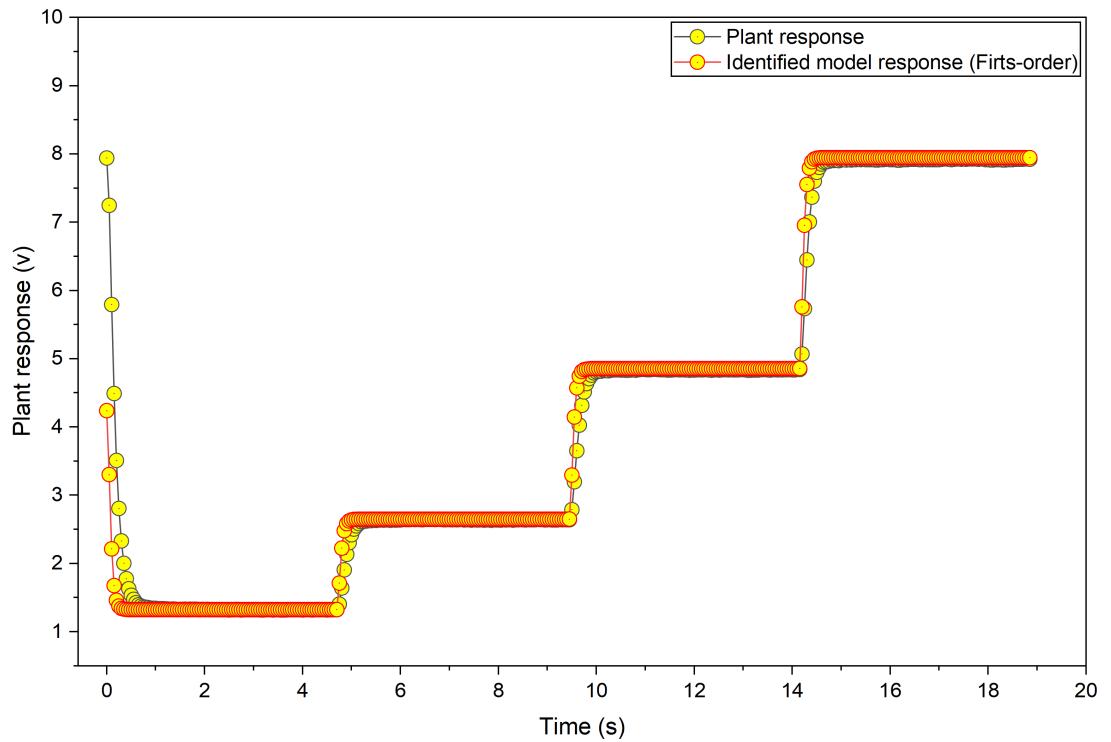


Figura 5.6: Comparación modelo planta y modelo de orden

### 5.3.2. Promt: Identifica la planta tomando de la base de datos el segundo id (orden 2)

Análisis IA: Dado que ambos módulos son menores que la unidad, se concluye que el sistema es estable en el dominio discreto. Esta evaluación confirma que el modelo cumple el criterio de estabilidad, validando que la condición estable = true es correcta.

---

**Algoritmo 5.6** Identificación con modelo Orden 2.

---

```
// Orden: 2 | Ts: 0.050000 s | Var: 3.951e-7
let y_1, y_2; let u_1, u_2;
y_1 = 0; y_2 = 0;
u_1 = 0; u_2 = 0;
function modeloPlanta(u) {
    const y = (1.180107 * y_1) + (-0.370723 * y_2) + (0.000306 * u)
               + (0.070544 * u_1) + (0.120089 * u_2);
    y_2 = y_1;    y_1 = y;
    u_2 = u_1;    u_1 = u;
    return y;
}
const coeficientes = {
    "a1": 1.180106937550394,
    "a2": -0.37072282937872236,
    "b0": 0.0003056924707942176,
    "b1": 0.07054447036386519,
    "b2": 0.12008947121337245
};
```

---

**Promt:Compara el modelo 15% 5s, 20% 5s, 25% 15s.**

La gráfica 5.7 representa la prueba desarrollada.

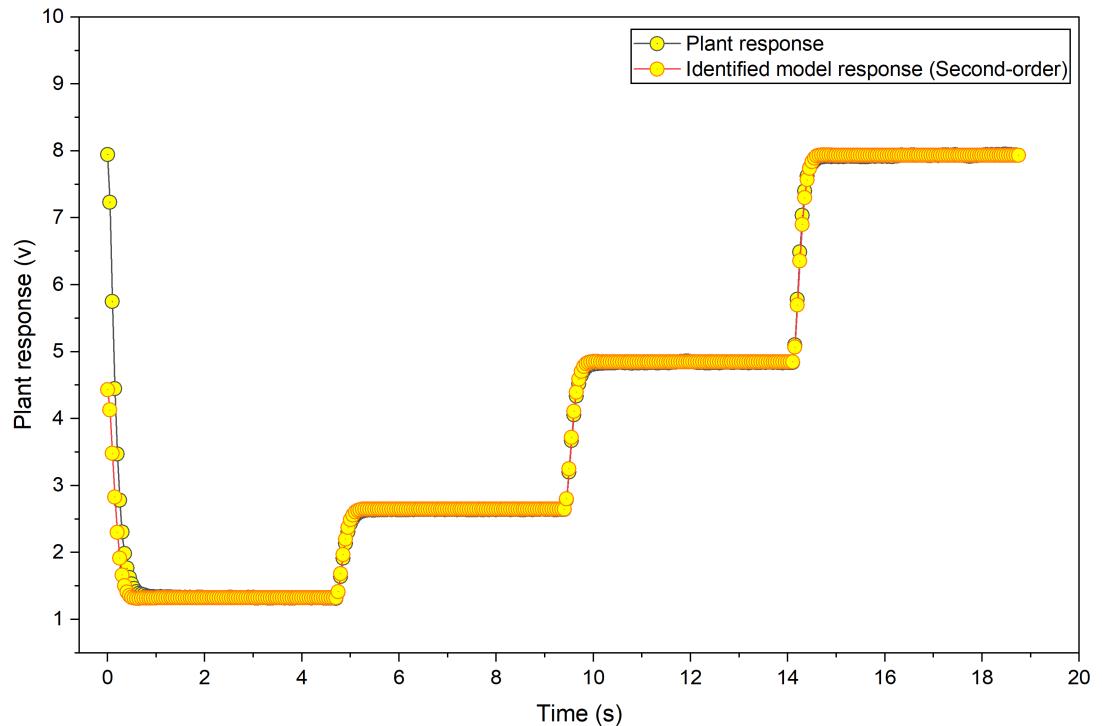


Figura 5.7: Comparacion modelo planta y modelo de orden 2

### 5.3.3. Promt: Identifica la planta tomando de la base de datos el segundo id (orden 3) IA

Análisis IA: El único modelo candidato es de orden 3, es estable y tiene un error cuadrático medio (MSE) de 0.0085. No hay otros modelos con los que comparar directamente. Aunque se busca simplicidad, la ausencia de modelos de orden inferior hace que esta elección sea inevitable. Dado que la estabilidad está garantizada y el error es aceptable, se selecciona este único modelo ver 5.7.

---

**Algoritmo 5.7** Identificación con modelo Orden 3.

---

```
// Orden: 3 | Ts: 0.100000 s | Var: 1.511e-5
let y_1, y_2, y_3;
let u_1, u_2, u_3;
y_1 = 0;
y_2 = 0;
y_3 = 0;
u_1 = 0;
u_2 = 0;
u_3 = 0;
function modeloPlanta(u) {
    const y = (0.238597 * y_1) + (0.092539 * y_2) + (-0.014429 * y_3)
        + (0.000492 * u) + (0.293197 * u_1)
        +(0.317410 * u_2) + (0.074456 * u_3);
    y_3 = y_2;
    y_2 = y_1;
    y_1 = y;
    u_3 = u_2;
    u_2 = u_1;
    u_1 = u;
    return y;
}
const coeficientes = {
    "a1": 0.23859660501588098,
    "a2": 0.09253921802860472,
    "a3": -0.01442905045677729,
    "b0": 0.0004919489060828885,
    "b1": 0.29319745897476895,
    "b2": 0.3174100991058665,
    "b3": 0.07445559846071487
};
```

---

**Promt:Compara el modelo 15% 5s, 20% 5s, 25% 15s**

La gráfica 5.8 representa la prueba desarrollada.

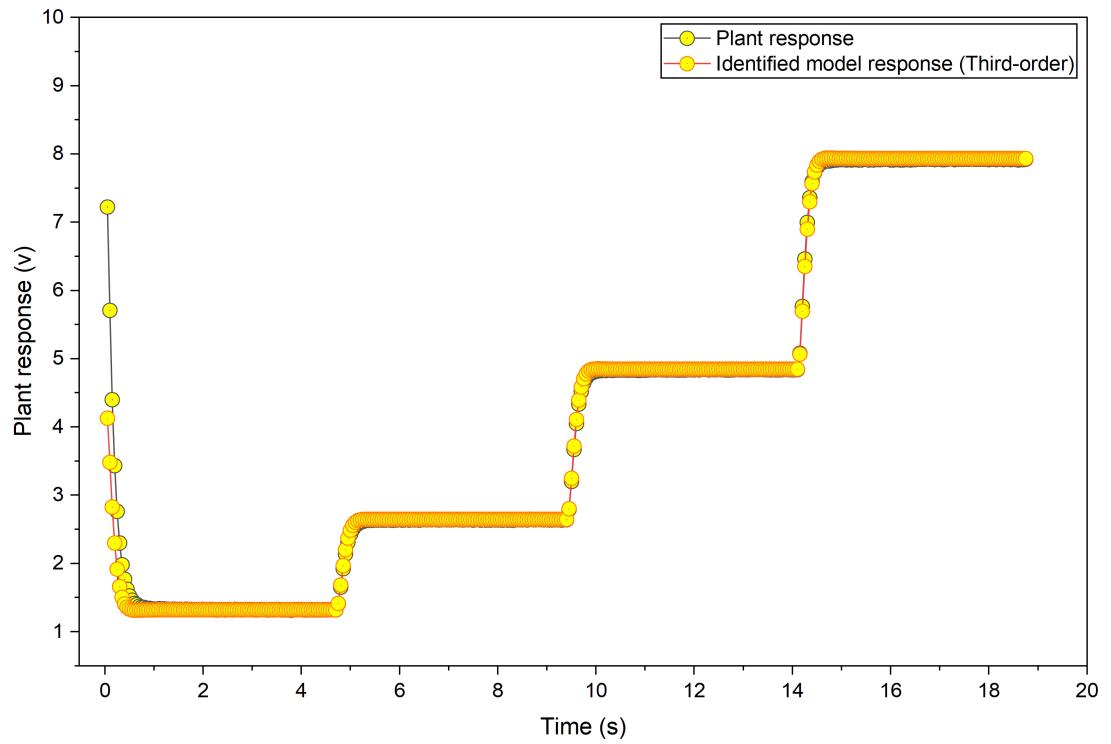


Figura 5.8: Comparacion modelo planta y modelo de orden 3

# Capítulo 6

## Controlador

Este capítulo consolida y articula los resultados obtenidos a lo largo del desarrollo de la plataforma de automatización industrial inteligente presentada en este documento. A partir de la implementación práctica de un controlador industrial asistido por inteligencia artificial, se demuestra que es posible integrar de manera efectiva técnicas clásicas de control automático, identificación de sistemas y modelos generativos de lenguaje, manteniendo criterios estrictos de seguridad, estabilidad y ejecución determinista.

El sistema desarrollado implementa un flujo completo y autónomo que inicia con la caracterización experimental de la planta, continúa con la identificación automática de modelos dinámicos de tipo ARX y culmina en la síntesis y ejecución de controladores PI discretos generados dinámicamente. Todo este proceso se realiza íntegramente en el borde (edge computing), eliminando la dependencia de procesamiento en la nube para la operación en tiempo real, reduciendo latencias y aumentando la confiabilidad operativa del sistema.

La arquitectura propuesta permite que el usuario interactúe con procesos industriales mediante instrucciones en lenguaje natural, las cuales son interpretadas por un módulo de inteligencia artificial generativa. Dicho módulo traduce las intenciones del usuario —como la regulación de un setpoint, la aceleración de la respuesta del sistema o la inyección controlada de perturbaciones en parámetros de control matemáticamente consistentes. Un aspecto central del enfoque es que la inteligencia artificial no actúa directamente sobre los actuadores físicos, sino que funciona como una capa de supervisión e interpretación, encargada de aplicar reglas de seguridad, restricciones físicas y fundamentos de teoría de control antes de que cualquier acción sea ejecutada sobre el hardware real.

La síntesis del controlador se basa en los modelos ARX identificados automáticamente a partir de datos reales del proceso, los cuales capturan la dinámica dominante de la planta y constituyen la base matemática del diseño del control. A partir de estos modelos, el sistema genera código ejecutable que implementa controladores PI discretos con mecanismos robustos, tales como protección anti-windup, saturación de la acción de control y manejo explícito de perturbaciones. Este código es generado y desplegado dinámicamente en tiempo de ejecución,

permitiendo adaptar el comportamiento del controlador a nuevas condiciones operativas sin necesidad de reprogramación manual ni reinicio del sistema.

La ejecución del lazo de control se mantiene completamente local y determinista, preservando los requisitos de tiempo real y desacoplando el razonamiento de alto nivel de la actuación física. Esta separación entre interpretación inteligente y ejecución determinista resulta clave para garantizar la estabilidad del sistema incluso en escenarios de prueba exigentes o ante perturbaciones externas.

En conjunto, este capítulo final demuestra que la convergencia entre control clásico, computación en el borde e inteligencia artificial interpretativa no solo es viable, sino altamente efectiva para el diseño de sistemas industriales modernos. Los resultados obtenidos validan la hipótesis de que es posible reducir significativamente la complejidad del comisionamiento, mejorar la interacción humano-máquina y avanzar hacia plataformas de automatización más flexibles, inteligentes y seguras, sentando así las bases para futuras aplicaciones de control industrial asistido por inteligencia artificial.

## Controlador IA

La función `procesarPromptControlIA` actúa como el orquestador principal del sistema de control asistido por inteligencia artificial, coordinando de manera secuencial y automática las etapas de caracterización, identificación y ejecución del control mediante la invocación explícita de los distintos agentes y módulos especializados del sistema. A partir de un prompt ingresado por el usuario, ya sea en forma de texto libre o como objeto estructurado, el sistema inicia un proceso integral que comienza con la caracterización experimental de la planta, haciendo uso del agente de caracterización para forzar la excitación del sistema y realizar un reinicio controlado del PLC, garantizando así condiciones iniciales consistentes. Posteriormente, la función evalúa si el flujo debe involucrar inteligencia artificial explícita, analiza parámetros como el orden del experimento y el desplazamiento de datos históricos, y construye dinámicamente la consulta utilizada por el agente de identificación. Con los coeficientes del modelo de la planta obtenidos a partir de este proceso, `procesarPromptControlIA` invoca el agente de síntesis de control basado en IA, encargado de interpretar la intención del usuario y generar parámetros de control matemáticamente válidos. Finalmente, el controlador resultante es cargado dinámicamente en tiempo de ejecución, ejecutado localmente en el dispositivo de borde y registrado mediante el sistema de adquisición de datos, cerrando así el ciclo completo desde la caracterización hasta la ejecución del control, y asegurando una operación determinista, segura y completamente trazable del lazo de control ver figura 6.1.

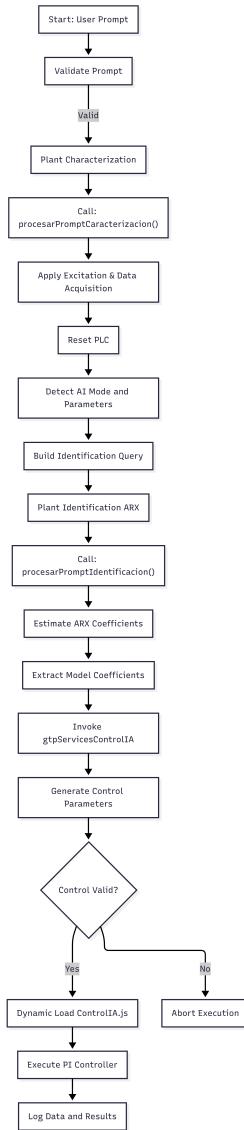


Figura 6.1: Control Orchestration Workflow of the AI-Assisted Edge Controller.

El pseudocódigo refleja el diseño modular y jerárquico del sistema. Cada bloque cumple una función bien definida: primero se garantiza la validez del prompt, luego se obtiene información experimental de la planta, se identifica su modelo dinámico y finalmente se sintetiza y ejecuta el controlador. La generación dinámica de código permite que cada ejecución del sistema se adapte a las condiciones reales del proceso, mientras que la separación entre identificación,

---

**Algoritmo 6.1** AI-Orchestrated Control Execution Pipeline.

---

```
FUNCTION procesarPromptControlIA(prompt):
    SI prompt es nulo
        RETORNAR error
    Guardar prompt original
    // Etapa 1: Caracterización
    Ejecutar caracterización automática de la planta
    Reiniciar PLC para condiciones iniciales limpias
    // Etapa 2: Análisis del prompt
    SI prompt es un objeto
        Extraer usarIA , orden y offset
    SINO
        Detectar uso de IA por palabras clave
        // Etapa 3: Identificación
        Construir consulta SQL de datos de caracterización
        Ejecutar identificación del modelo ARX
        Reiniciar PLC tras identificación
        // Etapa 4: Síntesis de control
        Obtener coeficientes del modelo identificado
        Definir tiempo de muestreo Ts
        Llamar a servicio de IA para generar parámetros de control
        SI la síntesis fue exitosa
            Invalidar caché del controlador previo
            Cargar nuevo controlador generado
            Ejecutar lazo de control PI
            Registrar resultados experimentales
    FIN
```

---

síntesis y ejecución asegura claridad, mantenibilidad y seguridad operacional ver algoritmo 6.1.

### Agente ControlIA

La función gtpServicesControlIA constituye el agente de síntesis de control basado en inteligencia artificial dentro de la arquitectura propuesta. Su responsabilidad principal es interpretar instrucciones de alto nivel proporcionadas por el usuario en lenguaje natural y traducirlas en parámetros de control matemáticamente consistentes, fundamentados en el modelo dinámico previamente identificado de la planta. A partir de los coeficientes ARX y del tiempo de muestreo, el módulo calcula los parámetros físicos del sistema, como la ganancia estática y la constante de tiempo dominante, y diseña un controlador PI discreto mediante la cancelación del polo más lento. La inteligencia artificial

---

**Algoritmo 6.2** AI-Orchestrated Control Execution Pipeline.

---

FUNCTION gtpServicesControlIA (coeficientes , Ts, promptUsuario):

    Analizar prompt del usuario  
    Detectar si se solicita perturbación (modo prueba)  
    Construir prompt de interpretación para IA  
    Solicitar a la IA extracción de parámetros en formato JSON  
    SI falla la interpretación  
        Usar parámetros por defecto seguros  
    Validar y limitar setpoint , alpha y perturbaciones  
    Calcular parámetros de la planta:  
        – Ganancia K  
        – Constante de tiempo dominante tau  
    Diseñar controlador PI:  
        – Ti = tau  
        – Kp = alpha / K  
        – Ki = Kp / Ti  
    Construir estructura de parámetros de control  
    Construir prompt de generación de código  
    Solicitar a la IA el código del controlador PI  
    Limpiar y validar el código generado  
    Guardar archivo ControlIA.js dinámicamente  
    Retornar parámetros de control y ruta del archivo

---

se emplea como una capa interpretativa que extrae parámetros como setpoint, duración de la simulación y condiciones de perturbación, aplicando reglas de seguridad y validación de rangos antes de cualquier ejecución. Finalmente, el módulo genera dinámicamente el código del controlador (ControlIA.js), el cual implementa un lazo de control determinista con mecanismos de anti-windup, saturación del actuador y registro de datos, garantizando una ejecución segura, trazable y coherente con los principios clásicos de control automático.

El pseudocódigo refleja el rol de gtpServicesControlIA como agente especializado en síntesis, encargado de transformar información semántica y modelos matemáticos en lógica de control ejecutable. La inteligencia artificial se utiliza exclusivamente para la interpretación de instrucciones y la generación de código, mientras que el diseño del controlador permanece estrictamente gobernado por ecuaciones clásicas de control. Este enfoque garantiza que, aun cuando se utilice IA generativa, el comportamiento del sistema sea determinista, estable y verificable, cumpliendo con los requisitos de control industrial.

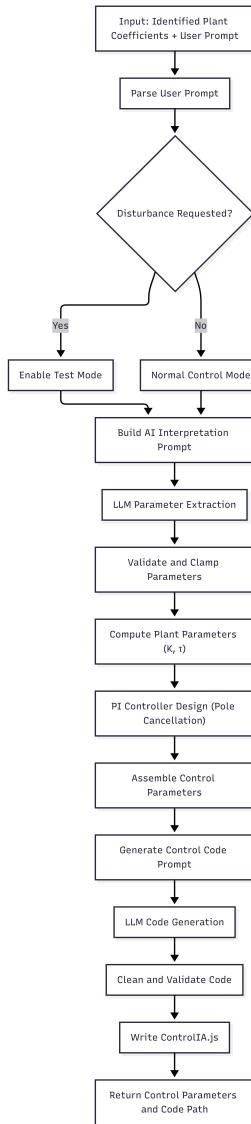


Figura 6.2: Time-Domain Response of the System under AI-Assisted PI Control.

## 6.1. Resultados Agente de ControlIA

### 6.1.1. Promt: Controlador IA con set point de 6v y 60 s.

Los resultados obtenidos en los dos escenarios evaluados confirman la eficacia y robustez del controlador PI asistido por inteligencia artificial. En el primer escenario, con un setpoint de 6 V y un tiempo de simulación de 60 s, el controlador alcanza la referencia de manera estable y progresiva, sin sobreimpulsos significativos y con un error en estado estacionario prácticamente nulo. La respuesta del sistema en lazo cerrado se mantiene estable durante todo el intervalo de simulación, lo que evidencia que los parámetros de control generados automáticamente permiten un adecuado compromiso entre rapidez, estabilidad y precisión.

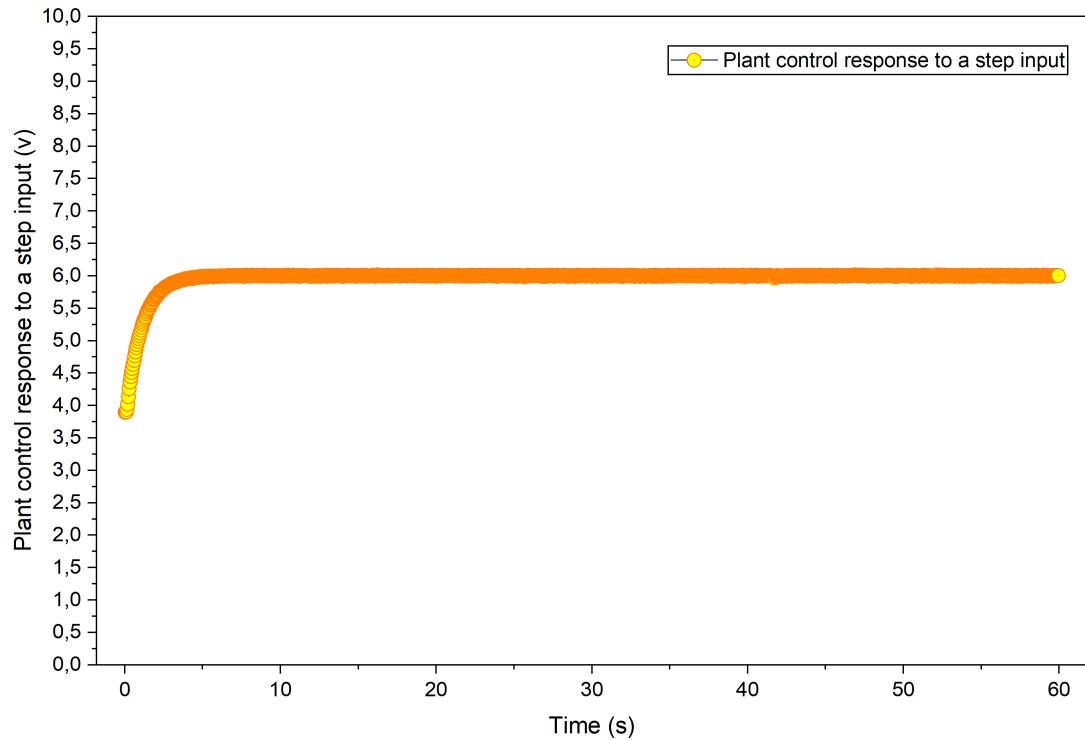


Figura 6.3: Closed-Loop Response with AI-Assisted PI Controller (Setpoint = 6 V, No Disturbance)

### 6.1.2. Promt: Controlador IA con set point de 6v y 60 s, disturbio.

En el segundo escenario, se introduce un disturbio de forma sostenida desde la mitad de la prueba y se mantiene activo hasta el final de la simulación, conservando el mismo setpoint de 6 V y una duración total de 60 s. Ante la aplicación del disturbio, la salida del sistema presenta una desviación inmediata respecto a la referencia; sin embargo, la acción integral del controlador compensa progresivamente su efecto, permitiendo recuperar el valor de consigna y mantenerlo dentro de márgenes aceptables durante el resto del experimento. Este comportamiento confirma la capacidad del controlador asistido por IA para rechazar perturbaciones persistentes y preservar la estabilidad del sistema, incluso bajo condiciones de operación adversas y prolongadas.

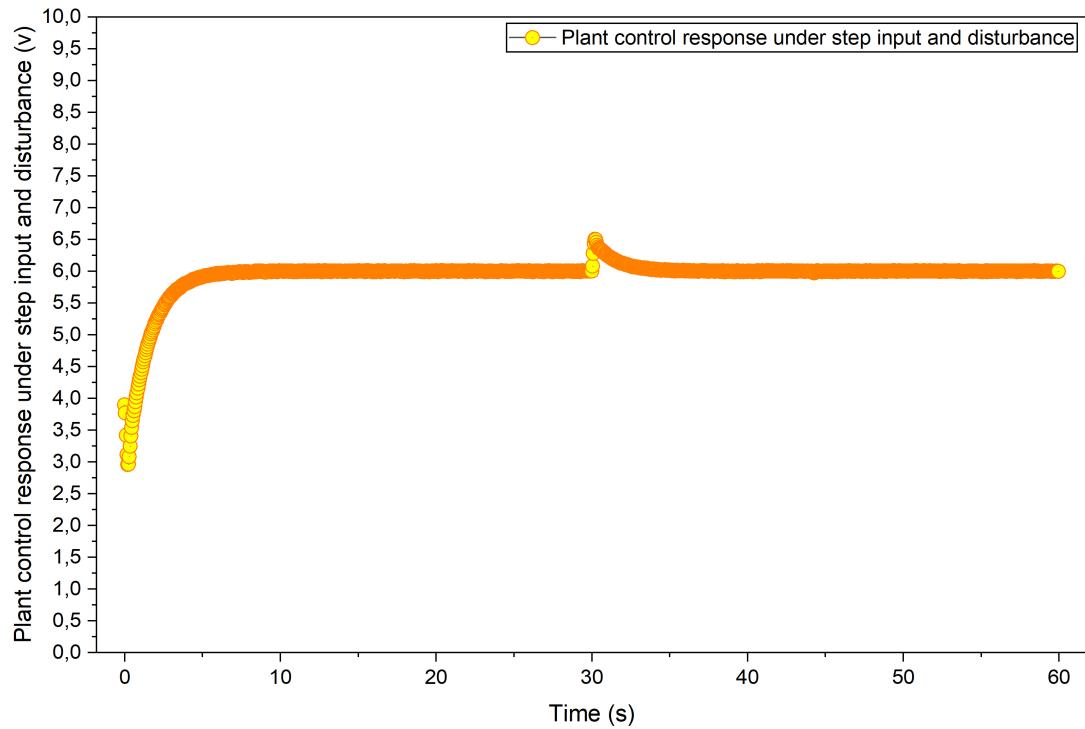


Figura 6.4: Disturbance Rejection Test with AI-Assisted PI Controller (Setpoint = 6 V)

## 6.2. Conclusiones

En este trabajo se presentó y validó una plataforma de automatización industrial inteligente que integra técnicas clásicas de control automático con mecanismos avanzados de inteligencia artificial interpretativa, ejecutados de forma segura en el borde. A lo largo del desarrollo se demostró que es posible automatizar de manera completa el ciclo de diseño de control desde la caracterización experimental de la planta, pasando por la identificación de modelos dinámicos tipo ARX, hasta la síntesis y ejecución de controladores PI discretos sin intervención manual directa del ingeniero durante la operación.

Los resultados obtenidos en los distintos escenarios de prueba evidencian que el controlador PI asistido por IA es capaz de alcanzar el setpoint deseado de manera estable, con tiempos de respuesta adecuados y sin comprometer la seguridad del sistema. En el escenario nominal, el sistema mostró un seguimiento preciso de la referencia, validando la correcta identificación del modelo y la sintonización automática del controlador. En el escenario con perturbación sostenida, aplicada a la mitad de la prueba y mantenida hasta el final de la simulación de 60 segundos, el sistema demostró capacidad de rechazo de perturbaciones, manteniendo la estabilidad del lazo cerrado y recuperando el desempeño deseado, lo que confirma la robustez del enfoque propuesto.

Un aporte clave del trabajo es la utilización de la inteligencia artificial como una capa de interpretación y supervisión, y no como un actuador directo. Este enfoque permite traducir instrucciones en lenguaje natural en parámetros de control matemáticamente consistentes, los cuales son posteriormente validados mediante reglas de seguridad, restricciones físicas y fundamentos de teoría de control antes de su ejecución. De esta forma, se preservan las propiedades deterministas y la confiabilidad requeridas en entornos industriales reales.

Asimismo, la arquitectura basada en agentes especializados y carga dinámica de código permite una alta flexibilidad y extensibilidad del sistema, reduciendo significativamente la complejidad del comisionamiento y facilitando la interacción humano-máquina. En conjunto, los resultados confirman la hipótesis inicial del trabajo: es viable y efectivo integrar control clásico, computación en el borde e inteligencia artificial para desarrollar plataformas de automatización industrial más inteligentes, seguras y adaptativas.

## 6.3. Trabajos Futuros

Como líneas de trabajo futuro, se identifican varias oportunidades de extensión y mejora de la plataforma desarrollada. En primer lugar, se propone ampliar el enfoque hacia controladores más avanzados, como PID adaptativos, control predictivo basado en modelos (MPC) o esquemas de control robusto, manteniendo la misma filosofía de síntesis automática asistida por IA.

En segundo lugar, resulta de interés incorporar mecanismos de identificación en línea y adaptación continua, que permitan al sistema actualizar el modelo de la planta y los parámetros del controlador frente a cambios dinámicos en

el proceso, envejecimiento de componentes o variaciones en las condiciones de operación.

Otra línea relevante es la integración de estrategias de detección y diagnóstico de fallas, utilizando los datos recolectados durante la operación para anticipar comportamientos anómalos y mejorar la confiabilidad del sistema. Asimismo, la expansión de la plataforma hacia arquitecturas IIoT más amplias, con múltiples nodos de control coordinados y comunicación segura entre capas IT/OT, permitiría su aplicación en sistemas industriales de mayor escala.