# Cuadernillo del Taller de Automatización Industrial - Tomo II

*Tec. Juan Pedro Lardet*

Año: 2026

Autor: Juan Pedro Lardet

Año: 2026

 DOI: 10.5281/zenodo.18711683 (V1)

# Índice general

# Prólogo

Este segundo tomo nace de la necesidad de profundizar en los conceptos que quedaron esbozados en el primer cuadernillo. Si en el Tomo I aprendimos a caminar, aquí daremos pasos más largos y complejos: nos adentraremos en el mundo de las señales analógicas, los tipos de datos avanzados, los algoritmos estructurados y la simulación realista de procesos industriales mediante Factory I/O y PLCSIM.

El objetivo es que el estudiante sea capaz de abordar problemas de automatización de mediana complejidad, integrando sensores continuos, actuadores proporcionales y lógica de control más sofisticada. Todo ello con un enfoque práctico y apoyado en herramientas profesionales.

Agradezco a todos los estudiantes y colegas que, con sus inquietudes y desafíos, han impulsado la creación de este material. Espero que sea de utilidad en su formación y en su futura labor profesional.

Juan Pedro Lardet
2026

# Introducción General

En el Tomo I nos familiarizamos con los fundamentos: lógica binaria, temporizadores, contadores, y la programación básica de PLCs. Ahora es momento de dar el salto a los sistemas que manejan variables continuas, como presión, temperatura, nivel o caudal. Estos sistemas requieren un tratamiento diferente: ya no basta con encender o apagar; hay que medir, comparar, calcular y actuar de forma proporcional.

Además, exploraremos lenguajes de programación más potentes como SCL (Structured Control Language) y técnicas de estructuración de código que nos permitirán abordar proyectos más grandes sin perder claridad.

Finalmente, la simulación 3D con Factory I/O nos brindará un entorno realista para probar nuestros algoritmos antes de llevarlos a un PLC real.

# Parte I

# Fundamentos de la Programación Avanzada

# 1 Repaso de Conceptos Básicos y Entorno TIA Portal

## 1.1. El ciclo de escaneo y su impacto en la programación

Recordemos que un PLC ejecuta su programa de forma cíclica: lectura de entradas, ejecución del programa y escritura de salidas. Este ciclo, conocido como *scan cycle*, tiene una duración típica de entre 1 ms y 150 ms. En aplicaciones sencillas este tiempo no es crítico, pero cuando trabajamos con señales analógicas o lazos de control rápidos, debemos considerar el retardo introducido por el ciclo.

Por ejemplo, si necesitamos leer una señal analógica y actuar sobre una salida en el mismo ciclo, el tiempo de respuesta será al menos un ciclo completo. En aplicaciones con realimentación, esto puede afectar la estabilidad del sistema.

## 1.2. Tipos de bloques en TIA Portal: OB, FB, FC, DB (repaso)

En TIA Portal, el código se organiza en bloques:

- **OB (Organization Block)**: Controlan la ejecución del programa. El más importante es el OB1 (ciclo principal). También existen OBs de arranque, de alarma, de error, etc.
- **FB (Function Block)**: Bloques con memoria. Tienen una instancia asociada en un bloque de datos (DB) que guarda los valores de sus variables internas entre llamadas. Ideales para máquinas de estado, temporizadores, etc.
- **FC (Function)**: Bloques sin memoria. Todas las variables locales se pierden al finalizar la ejecución. Útiles para operaciones matemáticas o lógicas reutilizables.
- **DB (Data Block)**: Almacenan datos globales o de instancia. Pueden ser globales (accesibles desde cualquier bloque) o de instancia (asociados a un FB).

## 1.3. Lenguajes de programación: LADDER, FBD, SCL (introducción)

TIA Portal soporta varios lenguajes de programación:

- **LADDER (KOP)**: Basado en diagramas de contactos. Es el más intuitivo para técnicos electricistas.

- **FBD (Function Block Diagram)**: Utiliza bloques funcionales interconectados. Muy útil para procesamiento de señales analógicas.
- **SCL (Structured Control Language)**: Lenguaje de alto nivel similar a Pascal. Permite implementar algoritmos complejos de forma compacta y estructurada.

En este tomo utilizaremos principalmente LADDER y SCL, y ocasionalmente FBD para operaciones analógicas.

## 1.4. Herramientas de depuración: tablas de vigilancia, referencias cruzadas

Para depurar programas, TIA Portal ofrece:

- **Tablas de vigilancia (watch tables)**: Permiten monitorizar y forzar variables en tiempo real.
- **Referencias cruzadas**: Muestran dónde se utiliza cada variable, ayudando a localizar errores.
- **Lista de errores**: Muestra los errores de compilación y advertencias.
- **Modo de simulación (PLCSIM)**: Permite probar el programa sin hardware real.

### 1.4.1. Ejercicios del capítulo 1

1. Explique por qué el tiempo de ciclo de escaneo puede afectar el control de un lazo analógico.
2. Diferencie entre un FB y un FC. ¿Cuándo usaría cada uno? Enumere los lenguajes de programación disponibles en TIA Portal y mencione una aplicación típica para cada uno.
3. Investigue cómo crear una tabla de vigilancia en TIA Portal y describa su utilidad.

# 2 Tipos de Datos en Automatización

## 2.1. Datos elementales: BOOL, BYTE, WORD, DWORD, INT, DINT, REAL

En la programación de PLCs, las variables tienen un tipo de datos que determina su tamaño y el rango de valores que pueden almacenar. Los más comunes son:

| Tipo | Descripción | Tamaño |
|---|---|---|
| BOOL | Booleano (verdadero/falso) | 1 bit |
| BYTE | Número sin signo de 8 bits | 1 byte |
| WORD | Número sin signo de 16 bits | 2 bytes |
| DWORD | Número sin signo de 32 bits | 4 bytes |
| INT | Entero con signo de 16 bits | 2 bytes |
| DINT | Entero con signo de 32 bits | 4 bytes |
| REAL | Número en coma flotante (32 bits) | 4 bytes |

Cuadro 2.1: Tipos de datos elementales en TIA Portal.

## 2.2. Rango y representación: enteros con signo, coma flotante

Es fundamental conocer el rango de cada tipo para evitar desbordamientos:

- **INT**: de -32 768 a 32 767.
- **DINT**: de -2 147 483 648 a 2 147 483 647.
- **REAL**: aproximadamente ±1.18e-38 a ±3.40e38, con 6-7 dígitos de precisión decimal.
- **BYTE, WORD, DWORD**: sin signo, respectivamente 0 a 255, 0 a 65 535, 0 a 4 294 967 295.

## 2.3. Constantes y literales

En los programas podemos usar constantes directamente:

- Booleanas: `true`, `false`
- Enteros: `5`, `-10`, `16FF` (hexadecimal)
- Reales: `3.1416`, `1.2e3`

## 2.4. Variables globales y locales: tablas de tags y bloques de datos

En TIA Portal, las variables se pueden definir en:

- **Tablas de tags (PLC tags)**: Variables globales accesibles desde cualquier bloque. Se asignan a direcciones físicas ( %I, %Q, %M) o simbólicas.
- **Bloques de datos (DB)**: Pueden contener variables globales (DB global) o variables de instancia (asociadas a un FB).
- **Variables temporales (Temp)**: Se definen dentro de bloques y solo existen durante la ejecución del bloque.

## 2.5. Conversión entre tipos (CONV)

A menudo es necesario convertir un tipo a otro. TIA Portal proporciona la instrucción `CONV`. Por ejemplo, convertir un entero a real:

$$\#realVar := CONV(INT\_TO\_REAL(\#intVar))$$

También se pueden usar operadores de conversión implícita en algunos casos, pero es recomendable hacerlas explícitas para evitar errores.

### 2.5.1. Ejercicios del capítulo 2

1. Declare una variable de tipo DINT y asígnele el valor máximo posible. ¿Qué ocurre si intenta sumarle 1?
2. ¿Cuál es la diferencia entre una variable global definida en una tabla de tags y una variable en un DB global?
3. Escriba en TIA Portal un segmento que convierta un valor entero (INT) en un real (REAL) y lo multiplique por 2.5.
4. Investigue qué es el *endianness* y cómo afecta a la comunicación entre PLCs.

# Parte II

# Procesamiento de Señales Analógicas

# 3 Adquisición de Señales Analógicas

## 3.1. Sensores analógicos: tipos y conexión al PLC

En la industria, muchas variables son de naturaleza continua: presión, temperatura, nivel, caudal, velocidad, etc. Para capturar estas magnitudes se utilizan **sensores analógicos**, que entregan una señal proporcional a la variable medida. Las señales más comunes son:

- **4-20 mA** (corriente): Muy utilizada por su inmunidad al ruido y capacidad de detectar rotura de cable (si la corriente es 0 mA hay fallo). Rango típico: 4 mA = valor mínimo, 20 mA = valor máximo.
- **0-10 V** (tensión): Simple pero más susceptible a caídas de tensión y ruido. Se usa en distancias cortas.

La conexión al PLC se realiza mediante **módulos de entrada analógica** (AI). Estos módulos convierten la señal física a un valor digital que la CPU puede procesar.

## 3.2. Módulos de entrada analógica: resolución, rangos, configuración en TIA Portal

Los módulos analógicos se caracterizan por:

- **Número de canales**: 2, 4, 8, etc.
- **Resolución**: típicamente 12, 14 o 16 bits. Determina la precisión de la conversión. Por ejemplo, un módulo de 16 bits divide el rango en 65536 niveles.
- **Rango de entrada**: Configurable (0-10V, 4-20mA, ±10V, etc.).
- **Tiempo de conversión**: Tiempo que tarda en digitalizar una señal.

En TIA Portal, para configurar un módulo analógico:

1. Agregar el módulo en la configuración del hardware (ej. SM 1231 AI 4x13 bits).
2. En las propiedades del módulo, seleccionar el rango de cada canal (por ejemplo, “4-20 mA”).
3. Asignar las direcciones de E/S (por defecto se asignan automáticamente).

## 3.3. Lectura de valores analógicos: direccionamiento (IW, PIW)

Una vez configurado, el valor digitalizado se almacena en una dirección de entrada. Por ejemplo, si el módulo ocupa las direcciones `IW64` (palabra de entrada), podemos leerlo con:

- `IW64` (WORD) o `PIW64` (periferia, lectura directa sin imagen de proceso). Es más común usar `IW` para trabajar con la imagen de proceso.

El valor leído es un número entero sin signo (WORD) que varía entre 0 y 27648 (valor típico para señales de 0-10V o 4-20 mA en la mayoría de módulos SIEMENS). Algunos módulos usan 0-27648 como rango normalizado.

## 3.4. Ejemplo práctico: lectura de un potenciómetro o sensor de temperatura

Supongamos que tenemos un sensor de temperatura 4-20 mA conectado al canal 0 de un módulo analógico. La dirección asignada es `IW64`. Queremos leer ese valor y almacenarlo en una variable `temp_raw` de tipo INT.

En LADDER, podemos usar una simple asignación (MOVE):

Listing 3.1: Lectura de valor analógico en LADDER

```
1    MOVE
2    EN ENO
3    IW64 IN OUT temp_raw
```

En SCL sería:

Listing 3.2: Lectura de valor analógico en SCL

```
1 #temp_raw := IW64;
```

## 3.5. Ejercicios del capítulo 3

1. Investigue las diferencias entre una señal de corriente 4-20 mA y una de tensión 0-10 V. ¿En qué casos es preferible usar corriente?

2. Configure un módulo analógico SM 1231 AI 4x13 bits en TIA Portal para un sensor 4-20 mA en el canal 0. Anote la dirección asignada.

3. Escriba un programa en LADDER que lea el valor analógico del canal 0 y lo copie a un DB global.

4. ¿Qué significa que el valor máximo normalizado sea 27648 y no 65535? Investigue la razón.

# 4 Escalado y Normalización de Señales

## 4.1. Necesidad de escalar: de valores crudos a magnitudes físicas

El valor leído (por ejemplo, 0 a 27648) no tiene significado físico directo. Para obtener la temperatura real (en °C), presión (en bar), etc., necesitamos **escalar** ese valor. La relación suele ser lineal:

$$\text{Valor físico} = \text{Valor mínimo del rango físico} + \left( \frac{\text{Valor crudo} - \text{Crudo mínimo}}{\text{Crudo máximo} - \text{Crudo mínimo}} \right) \times (\text{Rango físico})$$

Para un sensor 4-20 mA que mide temperatura de 0 a 100 °C, el valor crudo 0 corresponde a 0 °C y 27648 a 100 °C. Pero si el sensor tiene offset, por ejemplo 4 mA = 0 °C, entonces el valor crudo mínimo no es 0 sino el correspondiente a 4 mA. En módulos SIEMENS, 4 mA suele corresponder a 5530 (aproximadamente, porque 0-20 mA = 0-27648, entonces 4 mA = 27648 * 4/20 = 5530). Esto es importante: el crudo mínimo es 5530 y el máximo 27648.

## 4.2. Instrucciones de escalado: NORM_X y SCALE_X en TIA Portal

TIA Portal ofrece instrucciones específicas para normalizar y escalar:

- **NORM_X**: Normaliza un valor entero a un valor real entre 0.0 y 1.0. Se le proporciona el valor, el mínimo y el máximo.
- **SCALE_X**: Escala un valor normalizado (0.0 a 1.0) a un rango físico (mínimo y máximo) y lo convierte al tipo deseado (INT, REAL, etc.).

Ambas suelen usarse juntas:

Listing 4.1: Escalado con NORM_X y SCALE_X en LADDER

```
1   // Normalizar el valor crudo a un real entre 0 y 1
2   NORM_X INT TO REAL
3   EN
4   VALUE := IW64
5   MIN := 5530   // valor para 4 mA
6   MAX := 27648  // valor para 20 mA
7   OUT := #norm
8
9   // Escalar a temperatura real (0 a 100  C )
10  SCALE_X REAL TO REAL
11  EN
12  VALUE := #norm
```

```
    MIN := 0.0
    MAX := 100.0
    OUT := #temp_real
```

En SCL es más directo:

Listing 4.2: Escalado manual en SCL

```
#temp_real := INT_TO_REAL(IW64 - 5530) / INT_TO_REAL(27648 - 5530)
   * 100.0;
```

**Nota:** El valor 27648 es el límite superior estándar para señales analógicas en los módulos de SIEMENS (corresponde a 20 mA o 10 V). Algunos módulos permiten configurar otros rangos, pero 0-27648 es el más común.

## 4.3. Cálculo manual de escalado lineal: fórmula y ejemplos

Si no se desea usar las instrucciones, se puede calcular directamente:

$$\text{Valor físico} = \frac{(\text{Valor crudo} - \text{Crudo mínimo}) \times (\text{Rango físico})}{\text{Crudo máximo} - \text{Crudo mínimo}} + \text{Físico mínimo}$$

### 4.3.1. Ejemplo adicional en LADDER: escalado con comprobación de rango

En la práctica, los valores crudos pueden estar fuera del rango esperado (por ejemplo, 0 mA por rotura de cable). Es recomendable detectar esta situación y tomar acciones, como activar una alarma o utilizar el último valor válido. El siguiente programa en LADDER lee el valor analógico de un sensor de presión (0-10 bar, 4-20 mA), verifica que esté dentro del rango válido (5530 a 27648) y, en caso contrario, fuerza la presión a un valor de seguridad (0 bar) y activa una alarma.

Listing 4.3: Escalado con comprobación de rango en LADDER

```
// Red 1: Leer valor crudo
MOVE
EN
IW64 IN OUT #raw_value

// Red 2: Verificar si est  dentro de rango
+---[CMP >= INT]---+---[CMP <= INT]---+---( )--- #rango_ok
|   IN1: #raw_value|   IN1: #raw_value|
|   IN2: 5530      |   IN2: 27648     |
|   OUT: #mayor_5530|   OUT: #menor_27648|
+------------------+------------------+

// Red 3: Si est  fuera de rango, activar alarma y usar valor
   m nimo
+---[NOT #rango_ok]---+---( S )--- #alarma_sensor
|
```

```
+---( MOVE )--- 5530 -> #raw_value_ajustado

// Red 4: Si est  dentro, usar el valor crudo directamente
+---[ #rango_ok ]---+---( MOVE )--- #raw_value -> #
   raw_value_ajustado

// Red 5: Normalizar el valor ajustado (0-1)
NORM_X INT TO REAL
EN
VALUE := #raw_value_ajustado
MIN := 5530
MAX := 27648
OUT => #norm

// Red 6: Escalar a presi n (0-10 bar)
SCALE_X REAL TO REAL
EN
VALUE := #norm
MIN := 0.0
MAX := 10.0
OUT => #presion_bar

// Red 7: Asignar salida de alarma (por ejemplo, Q0.2)
Q0.2 := #alarma_sensor;
```

En este ejemplo se utiliza un valor ajustado (`raw_value_ajustado`) que se fuerza al mínimo si el sensor falla. La alarma se activa y permanece enclavada (podría resetearse con un botón externo). Esta técnica evita que el programa utilice valores erróneos que podrían causar comportamientos peligrosos.

## 4.4. Ejercicios del capítulo 4

1. Un sensor de presión 4-20 mA mide de 0 a 10 bar. El valor crudo leído es 12000. Calcule la presión real (suponiendo crudo mínimo 5530, máximo 27648).

2. Implemente en LADDER el escalado para el sensor de presión anterior, usando NORM_X y SCALE_X.

3. ¿Qué ocurre si el valor crudo está fuera del rango (por ejemplo, menor que 5530)? ¿Cómo se puede manejar esta situación?

4. Repita el ejercicio 1 usando SCL con una fórmula directa.

# 5 Comparadores y Operaciones Aritméticas

## 5.1. Comparadores: CMP ==, <>, >, <, >=, <= en LADDER

Para tomar decisiones basadas en valores analógicos, se utilizan comparadores. En LADDER, los comparadores se encuentran en la paleta como "CMP". Por ejemplo, para comparar si la temperatura es mayor a 50 °C:

Listing 5.1: Comparador en LADDER

```
    CMP > REAL
    #temp_real IN1
    50.0 IN2
    OUT -> #alta_temperatura
```

También se pueden usar contactos especiales como "==" en algunos PLC, pero lo más común es usar la instrucción CMP.

En SCL, se usan operadores relacionales normales:

Listing 5.2: Comparador en SCL

```
IF #temp_real > 50.0 THEN
    #alta_temperatura := TRUE;
ELSE
    #alta_temperatura := FALSE;
END_IF;
```

## 5.2. Operaciones aritméticas: suma, resta, multiplicación, división (ADD, SUB, MUL, DIV)

En LADDER, las operaciones aritméticas se realizan con bloques como ADD, SUB, MUL, DIV. Por ejemplo, para sumar dos valores reales:

Listing 5.3: Suma en LADDER

```
    ADD REAL
    EN
    IN1 := #valor1
    IN2 := #valor2
    OUT => #resultado
```

En SCL es más natural:

Listing 5.4: Suma en SCL

```
#resultado := #valor1 + #valor2;
```

## 5.3. Uso combinado para control por umbrales y lazos sencillos

Podemos combinar comparadores y operaciones aritméticas para implementar lógica de control como histéresis. Por ejemplo, control de temperatura con histéresis: encender un ventilador cuando la temperatura supere 50 °C y apagarlo cuando baje de 45 °C.

Listing 5.5: Control con histéresis en SCL

```
IF NOT #ventilador AND #temp_real > 50.0 THEN
    #ventilador := TRUE;
ELSIF #ventilador AND #temp_real < 45.0 THEN
    #ventilador := FALSE;
END_IF;
```

## 5.4. Ejemplo: control de temperatura con histéresis

Implementaremos un sistema completo: un sensor de temperatura analógico, un ventilador digital y una alarma si la temperatura supera 60 °C.

Listing 5.6: Programa completo en SCL

```
// Lectura y escalado
#temp_raw := IW64;
#temp_real := INT_TO_REAL(#temp_raw - 5530) / INT_TO_REAL(27648 -
   5530) * 100.0;

// Hist resis
IF NOT #ventilador AND #temp_real > 50.0 THEN
    #ventilador := TRUE;
ELSIF #ventilador AND #temp_real < 45.0 THEN
    #ventilador := FALSE;
END_IF;

// Alarma
IF #temp_real > 60.0 THEN
    #alarma := TRUE;
ELSE
    #alarma := FALSE;
END_IF;

// Salidas
Q0.0 := #ventilador;
Q0.1 := #alarma;
```

A continuación se muestra la misma lógica implementada en LADDER utilizando bobinas de Set y Reset. La ventaja de este enfoque es que las condiciones de encendido y apagado son explícitas y la memoria se gestiona mediante el biestable.

Listing 5.7: Control con histéresis en LADDER

```
// Red 1: Condici n de encendido (Set)
```

```
+---[CMP > REAL]---+---[/ "ventilador" ]---+---( S )--- "ventilador
   "
|   IN1: "temp_real"|                      |
|   IN2: 50.0       |                      |
|   OUT: #mayor_50  |                      |
+-------------------+----------------------+

// Red 2: Condici n de apagado (Reset)
+---[CMP < REAL]---+---[ "ventilador" ]---+---( R )--- "ventilador"
|   IN1: "temp_real"|                     |
|   IN2: 45.0       |                     |
|   OUT: #menor_45  |                     |
+-------------------+---------------------+

// Red 3: Alarma si temperatura > 60 C
+---[CMP > REAL]---+---( )--- "alarma"
|   IN1: "temp_real"|
|   IN2: 60.0       |
|   OUT: #mayor_60  |
+-------------------+
```

Observa que se han utilizado bobinas de Set (S) y Reset (R) para manejar la variable `ventilador`. La condición de Set se activa cuando la temperatura supera 50°C **y** el ventilador está apagado (contacto cerrado), aunque esto último no es estrictamente necesario si se usa un biestable, pero ayuda a evitar re-disparos innecesarios. La condición de Reset se activa cuando la temperatura baja de 45°C **y** el ventilador está encendido. Finalmente, la alarma se activa directamente al superar los 60°C.

## 5.5. Ejercicios del capítulo 5

1. Diseñe un control de nivel con histéresis para un tanque: la bomba de llenado se activa cuando el nivel baja de 2 metros y se desactiva cuando supera 8 metros (rango total 0-10 m). Use un sensor analógico.

2. Implemente en LADDER la comparación de dos valores analógicos y active una salida si el primero es mayor que el segundo.

3. Escriba un programa que calcule el promedio de 4 lecturas analógicas y lo almacene.

4. ¿Qué precauciones deben tomarse al dividir números enteros? (pista: división entera vs. real)

### Ejemplo resuelto: Promedio de 4 lecturas analógicas en LADDER

Supongamos que tenemos cuatro sensores analógicos conectados a las direcciones `IW64`, `IW66`, `IW68` e `IW70`. Queremos calcular el promedio de estos cuatro valores y almacenarlo en una variable `promedio` de tipo REAL. La siguiente implementación en LADDER realiza la suma total y luego divide entre 4.

Listing 5.8: Cálculo de promedio en LADDER

```
// Red 1: Sumar los dos primeros valores (IW64 + IW66)
ADD INT
EN
IN1 := IW64
IN2 := IW66
OUT => #suma1

// Red 2: Sumar los dos siguientes (IW68 + IW70)
ADD INT
EN
IN1 := IW68
IN2 := IW70
OUT => #suma2

// Red 3: Suma total (suma1 + suma2)
ADD INT
EN
IN1 := #suma1
IN2 := #suma2
OUT => #suma_total

// Red 4: Convertir suma_total a REAL
CONV INT TO REAL
EN
IN := #suma_total
OUT => #suma_real

// Red 5: Dividir entre 4 para obtener el promedio
DIV REAL
EN
IN1 := #suma_real
IN2 := 4.0
OUT => #promedio
```

Las variables `suma1`, `suma2`, `suma_total` son de tipo INT, mientras que `suma_real` y `promedio` son REAL. Este ejemplo puede adaptarse fácilmente para más lecturas o para trabajar directamente con valores escalados.

# 6 Control PID Básico (Introducción)

## 6.1. Concepto de lazo cerrado: setpoint, variable de proceso, salida de control

El control PID (Proporcional-Integral-Derivativo) es la técnica más extendida para controlar variables continuas en lazo cerrado. Sus componentes:

- **Setpoint (SP)**: Valor deseado de la variable (consigna).
- **Variable de proceso (PV)**: Valor medido actual.
- **Error**: Diferencia SP - PV.
- **Salida de control (Output)**: Señal que actúa sobre el actuador (ej. velocidad de bomba, apertura de válvula).

El PID calcula la salida como:

$$\text{Output} = K_p \cdot e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}$$

## 6.2. El bloque PID Compact en TIA Portal: configuración básica

TIA Portal dispone del bloque `PID_Compact` (para S7-1200/1500). Para usarlo:

1. Agregar un FB "PID_Compact" desde la librería.
2. Configurar los parámetros básicos en el bloque de datos de instancia:
   - Setpoint: variable donde se escribe la consigna.
   - Input: variable de proceso (real).
   - Output: variable de salida (real) que luego se puede escalar a 0-27648 para una salida analógica.
3. Ajustar los parámetros PID (ganancia, tiempo integral, tiempo derivativo). Inicialmente se puede usar autosintonía (pretuning).

## 6.3. Ajuste de parámetros (P, I, D) de forma empírica

Para sintonizar un PID manualmente:

- **P** (proporcional): Aumentar hasta que el sistema oscile. Luego reducir a la mitad aproximadamente.

- **I** (integral): Disminuir el tiempo integral (aumentar acción integral) para eliminar el error en estado estacionario, pero con cuidado de no causar oscilaciones.
- **D** (derivativo): Ayuda a anticipar cambios, pero es sensible al ruido. A menudo se deja en cero inicialmente.

El bloque PID_Compact incluye una función de autosintonía que puede facilitar el ajuste.

## 6.4. Ejemplo: control de nivel o temperatura con simulación

Supongamos un tanque con bomba de velocidad variable (salida analógica) y sensor de nivel analógico. Queremos mantener el nivel en 5 m (setpoint).

Listing 6.1: Configuración básica de PID en SCL

```
// Llamada al PID_Compact (suponiendo que ya se ha creado la instancia)
#PID_Instance(Setpoint := #nivel_sp,
              Input := #nivel_pv,
              Output => #bomba_speed_raw);

// Escalar la salida del PID (0-100%) a valor anal gico (0-27648)
#bomba_analog := REAL_TO_INT(#bomba_speed_raw * 276.48);
QW64 := #bomba_analog;   // Salida anal gica a bomba
```

A continuación se muestra la misma configuración implementada en LADDER. Primero se debe insertar el bloque `PID_Compact` desde la librería y crear su DB de instancia (por ejemplo, `DB_PID_Nivel`). Luego, en el programa principal (OB1), se llama al bloque y se escalan sus salidas.

Listing 6.2: Llamada al PID_Compact en LADDER

```
// Red 1: Llamada al bloque PID
+---[ CALL "PID_Compact", DB1 ]---+
|   Setpoint := #nivel_sp         |
|   Input := #nivel_pv            |
|   Output => #bomba_speed_raw    |
+---------------------------------+

// Red 2: Escalar la salida del PID (0-100%) a valor anal gico (0-27648)
MUL REAL
EN
IN1 := #bomba_speed_raw
IN2 := 276.48
OUT => #bomba_speed_scaled

// Red 3: Convertir a entero y enviar a la salida anal gica
CONV REAL TO INT
EN
```

```
IN := #bomba_speed_scaled
OUT => #bomba_speed_int

MOVE
EN
#bomba_speed_int IN OUT QW64
```

En este ejemplo, `bomba_speed_raw` es la salida del PID en tanto por uno (0.0 a 1.0). Multiplicando por 276.48 se obtiene el valor escalado a 0-27648. Luego se convierte a entero y se asigna a la salida analógica `QW64` que controla el variador de velocidad.

## 6.5. Ejercicios del capítulo 6

1. Explique la diferencia entre un control todo-nada (con histéresis) y un control PID. ¿Cuándo es preferible cada uno?

2. Configure un PID_Compact en TIA Portal para controlar la temperatura de un horno. Indique los parámetros necesarios.

3. Simule un lazo de control de nivel con Factory I/O y un PID. Ajuste los parámetros hasta obtener una respuesta estable.

4. ¿Qué es el "windup" integral y cómo se evita?

### 6.5.1. Protección contra windup integral

El bloque `PID_Compact` incorpora internamente mecanismos anti-windup. Sin embargo, si se implementa un PID manual, es necesario limitar la integral cuando la salida está saturada para evitar un sobrepaso excesivo. Esta es una de las ventajas de utilizar el bloque estándar.

# Parte III

# Algoritmos Complejos y Estructuras de Programación

# 7 Máquinas de Estados (Step-by-Step)

## 7.1. Concepto de máquina de estados: diagrama de flujo y transiciones

Una máquina de estados es un modelo de comportamiento donde el sistema se encuentra en un estado a la vez, y puede cambiar a otro estado cuando se cumplen ciertas condiciones (transiciones). Es ideal para procesos secuenciales (ej. una línea de envasado, un brazo robótico).

Los elementos clave:

- **Estados**: Situaciones discretas (ej. reposo, avanzando, girando, etc.).
- **Transiciones**: Condiciones que provocan el cambio de estado.
- **Acciones**: Actividades que se realizan en cada estado.

## 7.2. Implementación con registros de paso (INT) y comparadores

Una forma sencilla de implementar una máquina de estados en PLC es usar un registro de paso (variable INT) y una estructura CASE (en SCL) o comparadores en LADDER.

En SCL:

Listing 7.1: Máquina de estados simple en SCL

```
CASE #estado OF
    0: // Reposo
        Q0.0 := FALSE;
        Q0.1 := FALSE;
        IF #start THEN
            #estado := 10;
        END_IF;
    10: // Avanzar cilindro
        Q0.0 := TRUE;   // V lvula de avance
        IF #fin_carrera_avance THEN
            #estado := 20;
        END_IF;
    20: // Retroceder
        Q0.0 := FALSE;
        Q0.1 := TRUE;   // V lvula de retroceso
        IF #fin_carrera_retroceso THEN
            #estado := 0;
        END_IF;
END_CASE;
```

### 7.2.1. Ejemplo resuelto en LADDER: semáforo peatonal

Implementaremos un semáforo peatonal con dos estados: ROJO (peatones detenidos) y VERDE (peatones pueden cruzar). Un pulsador de petición (`I0.0`) cambia el estado a VERDE durante 10 segundos, luego vuelve a ROJO. El semáforo permanece en ROJO hasta que se presiona el pulsador.

Utilizaremos un registro de paso (`estado`) de tipo INT y comparadores para gestionar las transiciones.

Listing 7.2: Máquina de estados en LADDER para semáforo peatonal

```
// Red 1: Inicializaci n al arranque (primer ciclo)
+---[ "FirstScan" ]---+---( MOVE )--- 0 -> #estado

// Red 2: Estado 0 - ROJO
+---[CMP == INT]---+---[ "pulsador" ]---+---( MOVE )--- 1 -> #
   estado
|   IN1: #estado   |                     |
|   IN2: 0         |                     |
+------------------+---------------------+

// Red 3: Estado 1 - VERDE (se activa la salida y temporizador)
+---[CMP == INT]---+---( )--- Q0.0  // Luz verde
|   IN1: #estado   |
|   IN2: 1         |
+------------------+
|
+---( TON )--- #temporizador
    IN := TRUE
    PT := T#10s

// Red 4: Transici n de VERDE a ROJO al cumplirse el tiempo
+---[CMP == INT]---+---[ "temporizador".Q ]---+---( MOVE )--- 0 ->
   #estado
|   IN1: #estado   |                           |
|   IN2: 1         |                           |
+------------------+---------------------------+

// Red 5: Salida luz roja (activa cuando NO est  en verde)
+---[CMP <> INT]---+---( )--- Q0.1  // Luz roja
    IN1: #estado
    IN2: 1
```

**Explicación:**

- `FirstScan` es un bit especial que se activa solo en el primer ciclo del PLC; se usa para inicializar el estado a 0.
- En estado 0, si se presiona el pulsador, se pasa al estado 1.
- En estado 1, se activa la salida verde y se inicia un temporizador TON de 10 segundos.
- Cuando el temporizador finaliza, se vuelve al estado 0.

- La luz roja se activa en cualquier estado que no sea el 1 (es decir, en reposo).

## 7.3. Uso de bloques FB para encapsular la lógica

Para máquinas más complejas, conviene encapsular la máquina de estados en un FB. Así se pueden reutilizar y mantener más fácilmente. El FB tendrá una variable interna para el estado y métodos para las transiciones.

## 7.4. Ejemplo: secuencia de un brazo robótico con múltiples pasos

Diseñaremos un brazo con 3 movimientos (avanzar, girar, sujetar) en secuencia, con parada de emergencia.

Listing 7.3: FB de máquina de estados para brazo

```
FUNCTION_BLOCK FB_Brazo
VAR
    estado : INT := 0;
END_VAR

CASE estado OF
    0: // Reposo
        Q_avance := FALSE;
        Q_giro := FALSE;
        Q_sujetar := FALSE;
        luz_activo := FALSE;
        IF start THEN
            estado := 10;
            luz_activo := TRUE;
        END_IF;
    10: // Avanzar
        Q_avance := TRUE;
        IF sensor_avance THEN
            estado := 20;
        END_IF;
    20: // Girar
        Q_avance := FALSE;
        Q_giro := TRUE;
        IF sensor_giro THEN
            estado := 30;
        END_IF;
    30: // Sujetar
        Q_giro := FALSE;
        Q_sujetar := TRUE;
        temporizador(IN:=TRUE, PT:=T#2s);
        IF temporizador.Q THEN
            estado := 40;
        END_IF;
    40: // Retorno (simplificado)
```

```
        Q_sujetar := FALSE;
        // ... movimientos de retorno ...
        IF fin_ciclo THEN
            estado := 0;
            luz_activo := FALSE;
        END_IF;
END_CASE;

// Parada de emergencia
IF emergencia THEN
    estado := 0;
    luz_activo := FALSE;
    Q_avance := FALSE;
    Q_giro := FALSE;
    Q_sujetar := FALSE;
END_IF;
```

## 7.5. Ejercicios del capítulo 7

1. Dibuje el diagrama de estados para un semáforo peatonal (rojo/verde con pulsador).
2. Implemente en SCL la máquina de estados del semáforo.
3. ¿Por qué es importante incluir un estado de reposo o inicial?
4. Modifique el ejemplo del brazo para incluir un reset que vuelva al estado inicial en cualquier momento.

# 8 Manejo de Arrays y Estructuras de Datos

## 8.1. Declaración de arrays en DB y su uso

Un array es una colección de elementos del mismo tipo, accesibles por índice. En TIA Portal, se declaran en DB o en variables temporales. Ejemplo en un DB global:

Listing 8.1: Declaración de array en DB

```
    miArray : ARRAY[1..10] OF INT;
```

Para acceder: `miArray[3]`.

## 8.2. Acceso a elementos mediante índices

Podemos usar índices variables. Por ejemplo, para sumar todos los elementos:

Listing 8.2: Suma de array en SCL

```
#suma := 0;
FOR #i := 1 TO 10 DO
    #suma := #suma + #miArray[#i];
END_FOR;
```

### 8.2.1. Ejemplo en LADDER: Acceso a array con índice variable

En LADDER, para acceder a un elemento de un array usando un índice que puede variar (por ejemplo, un contador), se puede utilizar la instrucción `MOVE` junto con operaciones aritméticas. Sin embargo, TIA Portal permite direccionamiento indexado directamente en las variables, siempre que el índice sea una variable del tipo adecuado.

Supongamos que tenemos un array de 10 enteros llamado `miArray` en un DB global (por ejemplo, `DB_Global.miArray[1..10]`). Queremos escribir en la posición indicada por un contador `indice` el valor leído de una entrada analógica `IW64`. El siguiente programa en LADDER realiza esta tarea, asegurándose de que el índice esté dentro del rango (1 a 10).

Listing 8.3: Escritura en array con índice variable en LADDER

```
// Red 1: Incrementar contador (ejemplo: cada vez que se detecta un
    flanco)
+---[ "sensor" ]---+---( P )--- #flanco
+---[ #flanco ]---+---( ADD )--- #indice := #indice + 1

// Red 2: Limitar  ndice  entre 1 y 10
+---[CMP > INT]---+---( MOVE )--- 10 -> #indice
|   IN1: #indice  |
|   IN2: 10       |
```

```
+------------------+
+---[CMP < INT]---+---( MOVE )--- 1 -> #indice
    IN1: #indice
    IN2: 1

// Red 3: Escribir el valor anal gico en la posici n del array
// Nota: Se utiliza el operador de indexaci n directa: DB_Global.
   miArray[#indice]
MOVE
EN
IW64 IN OUT "DB_Global".miArray[#indice]
```

En este ejemplo, el contador `indice` se incrementa con cada flanco del sensor, pero se limita a 1-10. Luego se escribe el valor de `IW64` en la posición correspondiente del array. Es importante que el DB global esté correctamente definido y que la variable `miArray` sea de tipo `ARRAY[1..10] OF INT`.

## 8.3. Estructuras (STRUCT) y UDT (User Defined Types)

Una estructura agrupa variables de diferentes tipos. Se puede definir un tipo de datos propio (UDT) para reutilizarlo. Ejemplo:

Listing 8.4: Definición de UDT en TIA Portal

```
TYPE MotorData :
STRUCT
    nombre : STRING[20];
    velocidad_actual : INT;
    temperatura : REAL;
    en_marcha : BOOL;
END_STRUCT;
END_TYPE
```

Luego se puede usar en DB: `motor1 : MotorData;`

### 8.3.1. Ejemplo en LADDER: Acceso a campos de una UDT

Una vez definida una UDT (por ejemplo, `MotorData`) y creada una variable de ese tipo en un DB (por ejemplo, `motor1`), podemos acceder a sus campos directamente en LADDER. Supongamos la siguiente UDT:

Listing 8.5: Definición de UDT MotorData

```
TYPE MotorData :
STRUCT
    nombre : STRING[20];
    velocidad_actual : INT;
    temperatura : REAL;
    en_marcha : BOOL;
END_STRUCT;
END_TYPE
```

Y en un DB global declaramos:

```
motor1 : MotorData;
```

Ahora, en LADDER podemos leer y escribir cada campo. Por ejemplo, para actualizar la velocidad actual y la temperatura desde entradas analógicas:

Listing 8.6: Acceso a campos de UDT en LADDER

```
// Red 1: Leer velocidad desde IW64 y asignar a motor1.velocidad_actual
MOVE
EN
IW64 IN OUT "DB_Global".motor1.velocidad_actual

// Red 2: Leer temperatura desde IW66, escalar y asignar a motor1.temperatura
// Primero normalizamos y escalamos (similar a cap tulos anteriores)
NORM_X INT TO REAL
EN
VALUE := IW66
MIN := 5530
MAX := 27648
OUT => #temp_norm

SCALE_X REAL TO REAL
EN
VALUE := #temp_norm
MIN := 0.0
MAX := 100.0
OUT => #temp_real

MOVE
EN
#temp_real IN OUT "DB_Global".motor1.temperatura

// Red 3: Activar motor si temperatura < 80 C  y velocidad < 1000
+---[CMP < REAL]---+---[CMP < INT]---+---( )--- "DB_Global".motor1.en_marcha
|   IN1: #temp_real|   IN1: "DB_Global".motor1.velocidad_actual|
|   IN2: 80.0      |   IN2: 1000      |
+------------------+------------------+
```

Observa que los campos de la UDT se referencian con la notación de punto: `"DB_Global".motor1.vel` Es importante que el nombre del DB y la variable no contengan espacios (o usar comillas si los tienen).

## 8.4. Ejemplo: almacenar recetas con parámetros

Podemos crear un array de estructuras para guardar varias recetas:

Listing 8.7: Array de recetas

```
TYPE Receta :
STRUCT
    temperatura_sp : REAL;
    tiempo_coccion : TIME;
    agitacion : BOOL;
END_STRUCT;
END_TYPE

VAR_GLOBAL
    recetas : ARRAY[1..5] OF Receta;
END_VAR
```

## 8.5. Ejercicios del capítulo 8

1. Declare un array de 10 reales en un DB y escriba una función que calcule el promedio.
2. Defina una UDT para un sensor que incluya: tipo (STRING), valor$_actual(REAL), unidad(STRIN$

2. Cree un DB con 3 instancias de esa UDT y asígneles valores.
3. ¿Qué ventajas tiene usar UDT frente a variables sueltas?

### 8.5.1. Ejemplo resuelto: Promedio de un array en LADDER

Supongamos que tenemos un array `datos` de 10 elementos de tipo REAL en un DB global. Queremos calcular su promedio y almacenarlo en una variable `promedio`. En LADDER podemos hacerlo utilizando un bucle (no hay bucles en LADDER, pero podemos usar un contador y sumas repetitivas). Una forma eficiente es utilizar un bloque de función o programar en SCL, pero también es posible implementarlo con un contador y saltos (aunque no es lo más común). A continuación se muestra una implementación utilizando un contador y un acumulador, que se ejecuta en cada ciclo. Nota: esto no es un bucle, sino que se suma un elemento por ciclo, pero para 10 elementos se necesitarían 10 ciclos para obtener el promedio. Otra opción es usar una FC en SCL. Para mantener la simplicidad, aquí presentamos una solución en SCL (ya que es más adecuada), pero se menciona la alternativa.

Listing 8.8: Función en SCL para calcular promedio de un array

```
FUNCTION "PromedioArray" : REAL
VAR_INPUT
    datos : ARRAY[1..10] OF REAL;
END_VAR
VAR_TEMP
    i : INT;
    suma : REAL;
END_VAR
BEGIN
    suma := 0.0;
```

```
    FOR i := 1 TO 10 DO
        suma := suma + datos[i];
    END_FOR;
    PromedioArray := suma / 10.0;
END_FUNCTION
```

Esta función puede ser llamada desde LADDER usando el bloque `CALL`. De esta manera combinamos la potencia de SCL con la interfaz LADDER.

Si se desea una implementación puramente en LADDER, se puede usar un contador y sumas parciales, pero requiere más redes y no es tan elegante. Se recomienda usar SCL para operaciones repetitivas.

# 9 Programación con SCL (Structured Control Language)

## 9.1. Introducción a SCL: ventajas frente a LADDER

SCL es un lenguaje de alto nivel (similar a Pascal) que permite:

- Algoritmos complejos de forma compacta.
- Estructuras de control como IF, CASE, FOR, WHILE.
- Mejor legibilidad para tareas matemáticas o de manejo de datos.
- Reutilización de código mediante funciones y bloques.

## 9.2. Sintaxis básica: variables, asignaciones, condicionales (IF, CASE), bucles (FOR, WHILE)

- **Asignación**: `variable := expresion;`
- **IF**:

```
IF condicion THEN
    ...
ELSIF otra THEN
    ...
ELSE
    ...
END_IF;
```

- **CASE**:

```
CASE variable OF
    1: ...
    2: ...
    ELSE ...
END_CASE;
```

- **FOR**:

```
FOR i := 1 TO 10 DO
    ...
END_FOR;
```

- **WHILE**:

```
WHILE condicion DO
    ...
END_WHILE;
```

## 9.3. Implementación de funciones matemáticas y lógicas

En SCL podemos usar funciones matemáticas estándar: `SQRT, SIN, COS, ABS, etc.`. Ejemplo:

```
#raiz := SQRT(#valor);
```

## 9.4. Ejemplo: convertir un algoritmo LADDER a SCL

Supongamos un circuito de enclavamiento en LADDER. En SCL sería:

```
IF (marcha AND NOT paro) OR (motor AND NOT paro) THEN
    motor := TRUE;
ELSE
    motor := FALSE;
END_IF;
```

## 9.5. Ejercicios del capítulo 9

1. Re-escriba el programa de histéresis del capítulo 5 en SCL.
2. Implemente un bucle FOR que calcule el factorial de un número (usando DINT).
3. ¿Cómo se declara una función (FC) en SCL? Dé un ejemplo.
4. Convierta a SCL el ejemplo de máquina de estados del capítulo 7.

# Parte IV

# Simulación Avanzada con Factory I/O y PLCSIM

# 10 Integración Profunda entre TIA Portal, PLCSIM y Factory I/O

## 10.1. Configuración de la comunicación: driver Siemens PLCSIM

Factory I/O puede comunicarse con PLCSIM mediante el driver “Siemens PLCSIM”. Pasos:

1. En TIA Portal, compilar y cargar el programa en PLCSIM (iniciar simulación).
2. En Factory I/O, abrir el escenario deseado.
3. Ir a `File >Drivers` y seleccionar `Siemens PLCSIM`.
4. Asegurarse de que PLCSIM esté en ejecución y el PLC en RUN.

## 10.2. Asignación de direcciones de E/S en Factory I/O

Cada elemento del escenario (botones, sensores, luces, motores) tiene una dirección asignable. En Factory I/O, se puede configurar qué entrada/salida del PLC controla cada elemento. Por ejemplo, asignar el botón START a I0.0, la luz piloto a Q0.0, etc.

## 10.3. Simulación de escenas complejas: sensores analógicos y actuadores

Factory I/O incluye sensores analógicos (ultrasonidos, barreras, etc.) y actuadores analógicos (variadores de velocidad, posicionadores). Para usarlos, hay que configurar la dirección analógica correspondiente (por ejemplo, IW64 para un sensor de distancia) y luego escalar en el PLC.

## 10.4. Ejemplo completo: Control de una cinta transportadora con Factory I/O

A continuación se describe paso a paso cómo crear un proyecto en TIA Portal, programar un control básico de una cinta transportadora y simularlo en Factory I/O con el escenario "Basic - Start/Stop Conveyor".

### 10.4.1. Paso 1: Crear proyecto en TIA Portal

1. Abrir TIA Portal V17 y seleccionar Çrear nuevo proyecto".

2. Asignar nombre (por ejemplo, Çinta_FactoryIO") y ubicación.

3. En la vista de proyecto, hacer clic en .ªgregar dispositivoz seleccionar un PLC, por ejemplo, CPU 1214C DC/DC/DC.

4. Aceptar la configuración por defecto.

### 10.4.2. Paso 2: Configurar tabla de variables

Crear una tabla de variables (PLC tags) con las siguientes entradas y salidas:

| Nombre | Tipo | Dirección |
|---|---|---|
| Start_Button | Bool | %I0.0 |
| Stop_Button | Bool | %I0.1 |
| Motor_Conveyor | Bool | %Q0.0 |

Cuadro 10.1: Tabla de variables para el ejemplo

### 10.4.3. Paso 3: Programación en LADDER

En el OB1, insertar el siguiente programa de arranque/parada con enclavamiento:

Listing 10.1: Programa para cinta transportadora

```
// Red 1: Marcha/Parada con autoretencion
+---[ "Start_Button" ]---+---[/ "Stop_Button" ]---+---( )--- "
   Motor_Conveyor"
|                         |                          |
|   +---[ "Motor_Conveyor" ]---+                     |
+---+--------------------------+--------------------+
```

Compilar el programa (accesos directos: Ctrl+B).

### 10.4.4. Paso 4: Configurar PLCSIM

1. Hacer clic en Ïniciar simulación"(icono de computadora con engranaje).

2. En la ventana de PLCSIM, cargar el programa (si no se carga automáticamente).

3. Poner el PLC en RUN.

### 10.4.5. Paso 5: Configurar Factory I/O

1. Abrir Factory I/O.

2. Seleccionar el escenario "Basic - Start/Stop Conveyor"(se encuentra en la categoría "Basic").

3. Ir a `File >Drivers` y seleccionar "Siemens PLCSIM".

4. En la ventana de asignación de E/S, vincular los elementos del escenario con las direcciones del PLC:

- Hacer clic en el botón verde "Start.[en] la escena, y en la ventana de propiedades asignarlo a la entrada `I0.0`.
- Hacer clic en el botón rojo "Stopz asignarlo a `I0.1`.
- Hacer clic en el motor de la cinta y asignarlo a la salida `Q0.0`.

5. Guardar la configuración (opcional).

### 10.4.6. Paso 6: Simulación

1. En Factory I/O, presionar el botón "Play"para iniciar la simulación.
2. Presionar el botón Start en la escena. La cinta debería comenzar a moverse.
3. Presionar Stop para detenerla.
4. Verificar que el comportamiento es el esperado.

### 10.4.7. Ampliación: Añadir un sensor

Para practicar, podemos añadir un sensor inductivo en la escena que detenga la cinta cuando detecte una pieza. Los pasos serían:

1. En Factory I/O, agregar un sensor inductivo desde la biblioteca y colocarlo sobre la cinta.
2. Asignar su salida a una entrada del PLC, por ejemplo `I0.2`.
3. Modificar el programa en TIA Portal para que cuando el sensor se active, el motor se detenga (por ejemplo, añadiendo un contacto NC del sensor en serie con la bobina).
4. Recargar el programa en PLCSIM y probar.

## 10.5. Depuración: uso de tablas de vigilancia y forzado

Mientras se simula, se pueden usar tablas de vigilancia en TIA Portal para monitorear y forzar variables, y ver cómo reacciona el escenario.

## 10.6. Ejercicios del capítulo 10

1. Configure la comunicación entre TIA Portal, PLCSIM y Factory I/O para el escenario “Basic - Start/Stop Conveyor”. Verifique que el motor arranca y para.
2. Agregue un sensor inductivo en la escena y asígnelo a una entrada. Programe que cuando el sensor detecte una pieza, se detenga la cinta.
3. ¿Qué ventajas ofrece la simulación 3D frente a la simulación con PLCSIM solo?

# 11 Escenarios Analógicos en Factory I/O

## 11.1. Sensores analógicos en Factory I/O: distancia, nivel, presión

Factory I/O dispone de sensores con salida analógica:

- Sensor de distancia (ultrasonidos): entrega un valor proporcional a la distancia de un objeto.
- Sensor de nivel: en tanques, mide el nivel de líquido.
- Sensor de presión: mide presión en circuitos neumáticos o hidráulicos.

Estos sensores se configuran en una dirección de entrada analógica (ej. IW64) y el rango de valores (0-27648) corresponde al rango físico configurable en el escenario.

## 11.2. Lectura de valores analógicos en el PLC y escalado

Se procede igual que con señales reales: leer la palabra de entrada y escalar a unidades físicas según el sensor. Por ejemplo, un sensor de nivel configurado de 0 a 10 m.

## 11.3. Actuadores analógicos: variadores de velocidad, posicionadores

También hay actuadores analógicos:

- Variador de velocidad: se controla con una salida analógica (QW) que determina la velocidad (0-27648 = 0-100 %).
- Posicionador: para cilindros con control proporcional.

## 11.4. Ejemplo: control de nivel de un tanque con bomba de velocidad variable

Escenario: tanque con sensor de nivel (IW64) y bomba con variador (QW64). Queremos mantener nivel en 5 m.

Programa en SCL:

```
#nivel_raw := IW64;
#nivel_real := INT_TO_REAL(#nivel_raw) / 27648.0 * 10.0;  // 0-10 m
#error := 5.0 - #nivel_real;

```

```
// P simple
#velocidad := #Kp * #error;
IF #velocidad < 0.0 THEN #velocidad := 0.0; END_IF;
IF #velocidad > 100.0 THEN #velocidad := 100.0; END_IF;

QW64 := REAL_TO_INT(#velocidad * 276.48);
```

### 11.4.1. Configuración detallada de sensores analógicos en Factory I/O

Para utilizar un sensor analógico en Factory I/O, sigue estos pasos:

1. En el escenario, selecciona el sensor deseado (por ejemplo, `Ultrasonic Sensor` o `Level Sensor`).

2. En el panel de propiedades del sensor, busca la opción `Address` y asigna una dirección de entrada analógica, como `IW64`.

3. Configura el rango físico del sensor en `Range Min` y `Range Max`. Por ejemplo, para un sensor de nivel de 0 a 10 metros, establece `Min=0`, `Max=10`.

4. Asegúrate de que el tipo de señal sea el adecuado (por defecto, Factory I/O utiliza 0-10V, pero internamente convierte a valores 0-27648 para el PLC).

### 11.4.2. Ejemplo avanzado: control PID de nivel con LADDER

Retomemos el ejemplo del tanque con sensor de nivel y bomba de velocidad variable. Ahora implementaremos un control PID completo utilizando el bloque `PID_Compact` de TIA Portal, pero esta vez en LADDER. Además, añadiremos una alarma por fallo de sensor.

#### Configuración en TIA Portal

1. Crea un nuevo FB llamando al bloque `PID_Compact` desde la librería. Esto generará un DB de instancia (por ejemplo, `DB_PID_Nivel`).

2. En la tabla de variables, define las siguientes:

   - `nivel_raw` (INT) – lectura del sensor (IW64).
   - `nivel_real` (REAL) – nivel escalado en metros.
   - `nivel_sp` (REAL) – setpoint de nivel (por ejemplo, 5.0 m).
   - `bomba_speed` (REAL) – salida del PID (0-100 %).
   - `bomba_analog` (INT) – valor para la salida analógica (QW64).
   - `alarma_sensor` (BOOL) – alarma por fallo de sensor.
   - `raw_min` (INT) – constante 5530 (valor para 4 mA).
   - `raw_max` (INT) – constante 27648.

**Programa en LADDER**

Listing 11.1: Control PID de nivel con alarma en LADDER

```
// Red 1: Lectura del sensor
MOVE
EN
IW64 IN OUT #nivel_raw

// Red 2: Comprobaci n de rango v lido
+---[CMP >= INT]---+---[CMP <= INT]---+---( )--- #rango_ok
|   IN1: #nivel_raw|   IN1: #nivel_raw|
|   IN2: #raw_min  |   IN2: #raw_max  |
|   OUT: #mayor_min|   OUT: #menor_max|
+------------------+------------------+

// Red 3: Si fuera de rango, activar alarma y usar  ltimo  valor
   v lido (opcional)
+---[NOT #rango_ok]---+---( S )--- #alarma_sensor
|                     |
|                     +---( MOVE )--- #ultimo_valor -> #
   nivel_raw_ajustado
+---------------------+

// Red 4: Si est  dentro de rango, desactivar alarma y usar valor
   actual
+---[ #rango_ok ]---+---( R )--- #alarma_sensor
|                   |
|                   +---( MOVE )--- #nivel_raw -> #
   nivel_raw_ajustado
|                   |
|                   +---( MOVE )--- #nivel_raw -> #ultimo_valor
+-------------------+

// Red 5: Escalado del valor ajustado a nivel real (0-10 m)
NORM_X INT TO REAL
EN
VALUE := #nivel_raw_ajustado
MIN := #raw_min
MAX := #raw_max
OUT => #norm

SCALE_X REAL TO REAL
EN
VALUE := #norm
MIN := 0.0
MAX := 10.0
OUT => #nivel_real

// Red 6: Llamada al bloque PID_Compact
+---[ CALL "PID_Compact", "DB_PID_Nivel" ]---+
|   Setpoint := #nivel_sp                    |
```

```
45 |   Input := #nivel_real                          |
46 |   Output => #bomba_speed                         |
47 +-------------------------------------------------+
48
49 // Red 7: Escalar salida del PID (0-100%) a valor anal gico
      (0-27648)
50 MUL REAL
51 EN
52 IN1 := #bomba_speed
53 IN2 := 276.48
54 OUT => #bomba_scaled
55
56 CONV REAL TO INT
57 EN
58 IN := #bomba_scaled
59 OUT => #bomba_analog
60
61 MOVE
62 EN
63 #bomba_analog IN OUT QW64
64
65 // Red 8: Salida de alarma (por ejemplo, una luz en Q0.3)
66 Q0.3 := #alarma_sensor;
```

**Explicación del programa**

- Las redes 2 a 4 implementan una comprobación de rango: si el valor crudo está entre 5530 y 27648, se considera válido. En caso contrario, se activa la alarma y se utiliza el último valor válido almacenado en `ultimo_valor` (para evitar cambios bruscos). Si se prefiere, se puede forzar a 0 o a un valor seguro.
- La red 5 realiza el escalado habitual.
- La red 6 llama al bloque PID, que debe estar previamente configurado con los parámetros adecuados (por ejemplo, ganancia proporcional 2.0, tiempo integral 10 s, etc.).
- La red 7 convierte la salida del PID (0.0 a 1.0) a un valor entero para la salida analógica.
- La red 8 asigna la alarma a una salida digital.

### 11.4.3. Simulación del fallo de sensor en Factory I/O

Para probar la robustez del programa, podemos simular un fallo de sensor en Factory I/O. Por ejemplo:

- Desconecta virtualmente el sensor (en Factory I/O, puedes deshabilitar la salida del sensor o cambiar su dirección).
- Observa cómo se activa la alarma y el sistema mantiene la última velocidad de bomba válida (o se detiene, según la lógica implementada).

Esta práctica ayuda a entender la importancia de la detección de fallos en entornos industriales reales.

## 11.5. Ejercicios del capítulo 11

1. En Factory I/O, seleccione el escenario “Tank”. Configure el sensor de nivel y la bomba con variador. Programe un control proporcional para mantener el nivel en 50 %.

2. Agregue histéresis al control para evitar oscilaciones.

3. Implemente un control PID completo y compare la respuesta.

# 12 Simulación de Procesos Complejos

## 12.1. Escenario "Sorting by Height.º similar: detección y clasificación

El escenario “Sorting by Height” (o “Sorting by Size”) de Factory I/O presenta una cinta con piezas de diferentes alturas, un sensor de altura (analógico) y expulsores neumáticos. Es ideal para practicar clasificación.

## 12.2. Programación con múltiples sensores y actuadores

Se deben leer sensores (altura, presencia) y controlar actuadores (cinta, expulsores). Usaremos máquina de estados y arrays para llevar el conteo.

## 12.3. Uso de temporizadores, contadores y comparadores

Para clasificar, se puede usar un contador para cada categoría, y temporizadores para controlar el tiempo de expulsión.

## 12.4. Ejemplo: clasificación de piezas por altura con expulsor

Supongamos que queremos clasificar piezas en dos categorías: bajas (<5 cm) y altas (>=5 cm). Al llegar al sensor de altura, se determina la categoría y cuando la pieza llegue al expulsor correspondiente, se activa.

Programa:

```
// Detecci n de pieza en sensor de entrada
IF #sensor_entrada AND NOT #prev_entrada THEN
    #altura := IW64;  // leer altura
    IF #altura < 5.0 * 2764.8 THEN  // 5 cm escalado a crudo
        #categoria := 0;  // baja
    ELSE
        #categoria := 1;  // alta
    END_IF;
    // Guardar en cola (array FIFO)
    // ...
END_IF;
#prev_entrada := #sensor_entrada;

```

```
// Cuando la pieza llega al expulsor, activar seg n categor a
IF #sensor_expulsor_baja AND #categoria_actual = 0 THEN
    Q_expulsor_baja := TRUE;
    TON_expulsor(IN:=TRUE, PT:=T#500ms);
    IF TON_expulsor.Q THEN
        Q_expulsor_baja := FALSE;
    END_IF;
END_IF;
```

### 12.4.1. Ejemplo avanzado en LADDER: clasificador con cola FIFO

En sistemas de clasificación reales, las piezas no están en el lugar del expulsor en el mismo ciclo en que se mide su altura. Por ello, es necesario almacenar la categoría de cada pieza en una cola (FIFO) mientras viaja por la cinta. La longitud de la cola depende de la distancia entre el sensor de altura y los expulsores, y de la resolución deseada.

Supongamos que la distancia entre el sensor de altura y el primer expulsor equivale a 10 pulsos de encoder (o 10 ciclos de scan). Implementaremos una cola circular de 10 posiciones utilizando un array y un puntero de escritura/lectura.

Listing 12.1: Clasificador con cola FIFO en LADDER (fragmento)

```
// Declaraciones en DB (ejemplo)
// cola_categorias : ARRAY[0..9] OF INT;
// puntero_escritura : INT := 0;
// puntero_lectura : INT := 0;
// contador_piezas : INT := 0;

// Red 1: Detecci n de nueva pieza en sensor de entrada (flanco
   ascendente)
+---[ "sensor_entrada" ]---+---[NOT "prev_entrada"]---+---( )--- #
   flanco_entrada
+---[ "flanco_entrada" ]---+---( MOVE )--- #altura_actual -> "
   cola_categorias"[puntero_escritura]
|                          |
|                          +---( ADD )--- "puntero_escritura" + 1 ->
    "puntero_escritura"
|                          |
|                          +---( )--- #nueva_pieza
+--------------------------+

// Red 2: Ajuste circular del puntero de escritura
+---[CMP >= INT]---+---( SUB )--- "puntero_escritura" - 10 -> "
   puntero_escritura"
    IN1: "puntero_escritura"
    IN2: 10

// Red 3: Avance de la cinta (por ejemplo, cada pulso de encoder)
+---[ "pulso_encoder" ]---+---( )--- #avance_cinta
|                          |
```

```
|                          +---( MOVE )--- "cola_categorias"[
   puntero_lectura] -> #categoria_actual
|                          |
|                          +---( ADD )--- "puntero_lectura" + 1 -> "
   puntero_lectura"
+--------------------------+

// Red 4: Ajuste circular del puntero de lectura
+---[CMP >= INT]---+---( SUB )--- "puntero_lectura" - 10 -> "
   puntero_lectura"
    IN1: "puntero_lectura"
    IN2: 10

// Red 5: Actuaci n sobre expulsores seg n categor a
+---[CMP == INT]---+---[ "sensor_expulsor1" ]---+---( S )--- "
   expulsor1"
|   IN1: #categoria_actual|                          |
|   IN2: 0 (baja)         |                          |
+-------------------------+--------------------------+

+---[CMP == INT]---+---[ "sensor_expulsor2" ]---+---( S )--- "
   expulsor2"
|   IN1: #categoria_actual|                          |
|   IN2: 1 (mediana)      |                          |
+-------------------------+--------------------------+

// Red 6: Temporizadores para desactivar expulsores despu s de un
   tiempo
+---[ "expulsor1" ]---+---( TON )--- #TON_exp1 (PT:=T#500ms)
+---[ #TON_exp1.Q ]---+---( R )--- "expulsor1"

+---[ "expulsor2" ]---+---( TON )--- #TON_exp2 (PT:=T#500ms)
+---[ #TON_exp2.Q ]---+---( R )--- "expulsor2"
```

**Explicación:**

- La cola se implementa con un array de 10 enteros. Los punteros de escritura y lectura avanzan circularmente.

- Cuando se detecta una nueva pieza (flanco ascendente en el sensor de entrada), se lee su altura (analógica) y se determina la categoría (0, 1, 2). Este valor se guarda en la posición del puntero de escritura, y el puntero avanza.

- Cada pulso de encoder (o cada ciclo de scan, si la cinta es síncrona) se considera un avance de la cinta. Se lee la categoría de la posición del puntero de lectura y ese valor se convierte en la `categoria_actual` para los expulsores.

- Cuando una pieza llega al sensor del expulsor correspondiente, se activa el expulsor durante 500 ms.

## 12.5. Ejercicios del capítulo 12

1. Implemente en Factory I/O y TIA Portal un clasificador de 3 categorías.
2. Añada un contador de piezas totales y por categoría, mostrados en una HMI virtual.
3. Simule fallos (atascos) y programe una alarma.

### 12.5.1. Detección de atascos (watchdog)

En procesos con cintas transportadoras, es común que las piezas se atasquen, bloqueando el flujo. Podemos detectar esta situación mediante temporizadores que vigilen el tiempo entre detecciones sucesivas.

Listing 12.2: Temporizador watchdog para detección de atasco

```
// Red 1: Reiniciar temporizador cada vez que se detecta una pieza
+---[ "sensor_entrada" ]---+---( R )--- #watchdog_timer
|                          |
|                          +---( )--- #reset_watchdog
+--------------------------+

// Red 2: Temporizador watchdog (se activa si no hay piezas en 10
   segundos)
+---[NOT "sensor_entrada"]---+---( TON )--- #watchdog_timer (PT:=T
   #10s)

// Red 3: Alarma de atasco
+---[ #watchdog_timer.Q ]---+---( )--- "alarma_atasco"
```

Esta lógica activa una alarma si el sensor de entrada no detecta ninguna pieza durante 10 segundos, indicando un posible atasco o fin de suministro. La alarma puede detener la cinta y notificar al operador.

# 13 Depuración y Puesta a Punto en Simulación

## 13.1. Herramientas de diagnóstico en TIA Portal

Repaso de herramientas: referencias cruzadas, lista de errores, tablas de vigilancia, gráficas de tendencias (para señales analógicas).

### 13.1.1. Uso de gráficas de tendencias para señales analógicas

TIA Portal dispone de una herramienta de gráficas de tendencias que permite visualizar la evolución de variables en tiempo real. Es especialmente útil para ajustar controladores PID o para observar el comportamiento de señales ruidosas.

**Pasos para configurar una gráfica:**

1. Abre una tabla de vigilancia (watch table).
2. Agrega las variables que deseas monitorear (por ejemplo, `nivel_real` y `bomba_speed`).
3. Haz clic en el icono de gráfica (en la parte superior derecha de la tabla de vigilancia) para abrir la ventana de tendencias.
4. Selecciona las variables y configura el tiempo de muestreo (por ejemplo, 1 segundo).
5. Inicia la simulación y observa cómo evolucionan las señales.

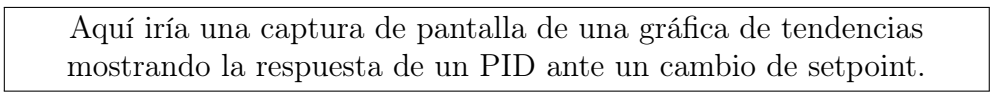


Figura 13.1: Ejemplo de gráfica de tendencias en TIA Portal.

Las gráficas de tendencias permiten:

- Visualizar la estabilidad del sistema.
- Detectar oscilaciones o sobrepasos excesivos.
- Comparar el valor real con el setpoint.
- Exportar los datos para análisis posterior.

## 13.2. Simulación de fallos en Factory I/O

Factory I/O permite inyectar fallos: sensores atascados, objetos que no se detectan, etc. Útil para probar la robustez del programa.

## 13.3. Estrategias para probar el programa paso a paso

- Probar cada sensor por separado forzando entradas.
- Usar breakpoints en SCL (si el entorno lo permite).
- Probar la máquina de estados con todas las transiciones.
- Verificar los límites de escalado.

### 13.3.1. Checklist para pruebas sistemáticas de un sistema automatizado

Antes de dar por finalizado un proyecto, es recomendable seguir una lista de verificación que asegure que todos los aspectos críticos han sido probados.

| Ítem | Descripción | OK |
|---|---|---|
| 1 | Verificar que todas las entradas digitales responden correctamente (forzando sensores). | |
| 2 | Verificar que todas las salidas digitales activan los actuadores correspondientes. | |
| 3 | Probar la parada de emergencia en cualquier estado del proceso. | |
| 4 | Comprobar el escalado de señales analógicas con valores conocidos (ej. 4 mA, 20 mA). | |
| 5 | Verificar la respuesta del PID ante cambios de setpoint (observar sobrepaso, tiempo de establecimiento). | |
| 6 | Simular fallos de sensor (cable roto, valor fuera de rango) y comprobar alarmas. | |
| 7 | Probar todas las transiciones de la máquina de estados (incluyendo condiciones límite). | |
| 8 | Verificar que los temporizadores y contadores funcionan con la base de tiempo correcta. | |
| 9 | Comprobar que el sistema vuelve a un estado seguro ante pérdida y recuperación de alimentación. | |
| 10 | Documentar los resultados de las pruebas y cualquier incidencia. | |

Cuadro 13.1: Checklist de pruebas para sistemas automatizados.

Esta lista puede adaptarse a cada proyecto concreto, pero cubre los aspectos fundamentales que deben validarse antes de la puesta en marcha real.

## 13.4. Ejercicios del capítulo 13

1. Tome un programa previo y use la tabla de vigilancia para forzar entradas y verificar salidas.
2. Simule un fallo de sensor en Factory I/O (por ejemplo, sensor de altura siempre en cero) y observe cómo reacciona el programa. Mejore el programa para detectar esta condición.

3. Realice un test completo de un sistema de clasificación, documentando los casos de prueba.

# Parte V

# Proyectos Integradores

# 14 Proyecto Integrador Nivel 1 - Control de una Prensa con Señal Analógica

## 14.1. Consigna

Diseñar un sistema para una prensa neumática con control de presión. El sistema cuenta con: - Un cilindro de doble efecto con válvula 5/2. - Un sensor de presión analógico en el circuito neumático (0-10 bar, 4-20 mA). - Pulsadores de marcha (I0.0) y parada (I0.1). - Parada de emergencia (I0.2, NC). - Sensor de posición retraído (I0.3) y extendido (I0.4). - Salidas: Q0.0 (avance), Q0.1 (retroceso), Q0.2 (luz verde activo).

## 14.2. Requisitos funcionales

1. Al presionar marcha, si el cilindro está retraído y no hay emergencia, se inicia el ciclo. 2. El cilindro avanza hasta llegar al final de carrera extendido. 3. Una vez extendido, debe alcanzar una presión de 8 bar (medida por el sensor analógico) y mantenerla durante 3 segundos. 4. Luego retrocede hasta el final de carrera retraído. 5. El ciclo se detiene y queda listo para una nueva marcha. 6. Si la presión supera 9 bar en cualquier momento, debe abortar el ciclo y retroceder inmediatamente, activando una alarma (luz roja Q0.3). 7. La parada de emergencia detiene todo inmediatamente y requiere un reset (pulsador de reset o nueva marcha) para reanudar.

## 14.3. Guía de desarrollo

- Escalar la señal analógica de presión. - Implementar máquina de estados. - Incluir enclavamientos y condiciones de emergencia.

## 14.4. Solución guiada

A continuación se presenta una posible implementación del sistema de prensa. Se utiliza una máquina de estados con 4 estados: REPOSO, AVANZAR, PRESIONAR, RETROCEDER. La presión se lee mediante un sensor analógico (4-20 mA) y se escala a bares.

### 14.4.1. Asignación de E/S (tags)

| Nombre | Tipo | Dirección | Descripción |
|---|---|---|---|
| marcha | BOOL | I0.0 | Pulsador NA de marcha |
| parada | BOOL | I0.1 | Pulsador NA de parada |
| emergencia | BOOL | I0.2 | Parada de emergencia (NC) |
| retraido | BOOL | I0.3 | Sensor de cilindro retraído |
| extendido | BOOL | I0.4 | Sensor de cilindro extendido |
| presion_raw | INT | IW64 | Valor crudo del sensor de presión (4-20 mA) |
| avance | BOOL | Q0.0 | Válvula de avance |
| retroceso | BOOL | Q0.1 | Válvula de retroceso |
| luz_activo | BOOL | Q0.2 | Luz verde de sistema activo |
| luz_alarma | BOOL | Q0.3 | Luz roja de alarma |

Cuadro 14.1: Tabla de variables para la prensa

### 14.4.2. Constantes y variables internas

- `estado` (INT): 0=REPOSO, 10=AVANZAR, 20=PRESIONAR, 30=RETROCEDER.
- `presion_bar` (REAL): presión escalada.
- `timer_presion` (TON): temporizador para mantener presión.
- `raw_min` (INT): 5530 (4 mA).
- `raw_max` (INT): 27648 (20 mA).

### 14.4.3. Diagrama de estados

[node distance=2cm, auto] [state, initial] (reposo) REPOSO; [state, right of=reposo] (avanzar) AVANZAR; [state, below of=avanzar] (presionar) PRESIONAR; [state, left of=presionar] (retroceder) RETROCEDER;
`->](reposo)edgenode{marchayretraÃŋdoynoemergencia}(avanzar)(avanzar)`
`edgenode{extendido}(presionar)(presionar)edgenode{presiÃşn>`
`=8barytemporizadorcumplido}(retroceder)(retroceder)`
`edgenode{retraÃŋdo}(reposo)(presionar)edge`loop below] node presión <8 bar o temporizando () (avanzar) edge[loop above] node no extendido ();

Figura 14.1: Diagrama de estados de la prensa

### 14.4.4. Programa en LADDER

A continuación se muestran las redes principales. Se asume que se ha creado un bloque de datos (DB) o variables estáticas en el OB1 para mantener el estado.

Listing 14.1: Red 1: Lectura y escalado de presión

```
1 // Red 1: Leer valor crudo y escalar a bares (0-10 bar)
```

```
MOVE
EN
IW64 IN OUT #presion_raw

NORM_X INT TO REAL
EN
VALUE := #presion_raw
MIN := 5530
MAX := 27648
OUT => #norm

SCALE_X REAL TO REAL
EN
VALUE := #norm
MIN := 0.0
MAX := 10.0
OUT => #presion_bar
```

Listing 14.2: Red 2: Máquina de estados - transiciones y acciones

```
// Red 2: Estado REPOSO (0)
+---[CMP == INT]---+---[ "marcha" ]---+---[ "retraido" ]---+---[NOT "emergencia" ]---+---( MOVE )--- 10 -> #estado
|   IN1: #estado   |                   |                     |
|   IN2: 0         |                   |                     |
+------------------+-------------------+---------------------+

// Red 3: Estado AVANZAR (10)
+---[CMP == INT]---+---( )--- "avance"
|   IN1: #estado   |
|   IN2: 10        |
+------------------+
|
+---[ "extendido" ]---+---( MOVE )--- 20 -> #estado

// Red 4: Estado PRESIONAR (20)
+---[CMP == INT]---+---( )--- "avance"   (mantener extendido)
|   IN1: #estado   |
|   IN2: 20        |
+------------------+
|
+---[ "parada" ]---+---( MOVE )--- 0 -> #estado   // parada manual
|
+---[CMP >= REAL]---+---( TON )--- #timer_presion (PT:=T#3s)
    IN1: #presion_bar
    IN2: 8.0
|
+---[ #timer_presion.Q ]---+---( MOVE )--- 30 -> #estado

```

```
// Red 5: Estado RETROCEDER (30)
+---[CMP == INT]---+---( )--- "retroceso"
|   IN1: #estado   |
|   IN2: 30        |
+------------------+
|
+---[ "retraido" ]---+---( MOVE )--- 0 -> #estado

// Red 6: Alarma por sobrepresi n
+---[CMP > REAL]---+---( )--- "luz_alarma"
|   IN1: #presion_bar|              // y tambi n forzar retroceso
|   IN2: 9.0         |
+--------------------+
|
+---[ "luz_alarma" ]---+---( MOVE )--- 30 -> #estado   // abortar a
    retroceso
```

Listing 14.3: Red 7: Luz de sistema activo

```
// Red 7: Luz verde activo cuando no est  en reposo
+---[CMP <> INT]---+---( )--- "luz_activo"
    IN1: #estado
    IN2: 0
```

Listing 14.4: Red 8: Parada de emergencia (reset)

```
// Red 8: Emergencia - forzar estado 0 y desactivar salidas
+---[ "emergencia" ]---+---( MOVE )--- 0 -> #estado
|                      |
|                      +---( R )--- "avance"
|                      |
|                      +---( R )--- "retroceso"
|                      |
|                      +---( R )--- "luz_activo"
|                      |
|                      +---( )--- "luz_alarma"   (opcional)
+----------------------+
```

### 14.4.5. Explicación adicional

- La máquina de estados se implementa mediante comparadores del valor de `estado` con constantes. Cada estado activa las salidas correspondientes.
- En el estado PRESIONAR, se espera a que la presión alcance 8 bar y luego se inicia un temporizador de 3 segundos. Si durante la espera se pulsa parada (I0.1), se vuelve a reposo.
- La alarma por sobrepresión (>9 bar) fuerza la transición al estado RETROCEDER y activa la luz roja.
- La parada de emergencia (I0.2, normalmente cerrado) se programa como contacto normalmente abierto en el programa (ya que si se pulsa, la entrada se desactiva). En

la red 8, se utiliza el contacto directo de `emergencia` (que será TRUE cuando no está pulsado, por ser NC). Para detectar la emergencia, debemos usar `NOT emergencia`? Cuidado: La lógica depende del cableado. Si la entrada es NC, cuando no hay emergencia, la entrada está activa (TRUE). Al pulsar emergencia, la entrada se desactiva (FALSE). Por tanto, la condición de emergencia es `NOT emergencia`. En el listing hemos usado `.emergencia"` directamente, pero debería ser `NOT .emergencia"`. Ajustaremos la red 8:

Listing 14.5: Red 8 corregida: Parada de emergencia

```
// Red 8: Emergencia (entrada NC) - forzar estado 0 y desactivar
   salidas
+---[NOT "emergencia" ]---+---( MOVE )--- 0 -> #estado
|                         |
|                         +---( R )--- "avance"
|                         |
|                         +---( R )--- "retroceso"
|                         |
|                         +---( R )--- "luz_activo"
|                         |
|                         +---( S )--- "luz_alarma"
+-------------------------+
```

### 14.4.6. Ejercicios adicionales propuestos

1. Modificar el programa para que, al producirse una alarma por sobrepresión, sea necesario un reset explícito (por ejemplo, pulsador de reset) antes de poder reanudar.
2. Añadir un contador de ciclos realizados.
3. Implementar la misma lógica en SCL y comparar.

# 15 Proyecto Integrador Nivel 2 - Línea de Empaquetado con Clasificación

## 15.1. Consigna

Diseñar una línea de empaquetado que clasifica piezas por altura y las cuenta en cajas. Elementos: - Cinta transportadora principal con motor (Q0.0). - Sensor de entrada (I0.0) para detectar piezas. - Sensor de altura analógico (IW64) que mide altura de 0 a 10 cm. - Dos expulsores neumáticos (Q0.1 y Q0.2) para desviar piezas a dos bandas laterales. - Sensores de final de carrera de los expulsores (I0.1, I0.2) para confirmar movimiento. - Contadores: por cada categoría, mostrar en una HMI virtual (o en variables).

## 15.2. Requisitos funcionales

1. Clasificar piezas en tres categorías: pequeñas (<3 cm), medianas (3-7 cm), grandes (>7 cm). 2. Las pequeñas se expulsan con el primer expulsor, las medianas con el segundo, las grandes siguen por la cinta principal (fin de línea). 3. Llevar un conteo de piezas por categoría. 4. Si una categoría llega a 10 piezas, detener la cinta principal y encender una luz indicadora de "lote completo". Reanudar con un botón de reanudar. 5. Incluir parada de emergencia.

## 15.3. Solución guiada

Para la línea de empaquetado, implementaremos una máquina de estados simple combinada con una cola FIFO (similar a la del capítulo 12) para recordar la categoría de cada pieza. Se utilizarán contadores para llevar el conteo por categoría.

### 15.3.1. Asignación de E/S (tags)

| Nombre | Tipo | Dirección | Descripción |
|---|---|---|---|
| $\text{sensor}_{e}ntrada$ | BOOL | I0.0 | Sensor de entrada (detección de pieza) |
| $\text{sensor}_{a}ltura_{r}aw$ | INT | IW64 | Sensor analógico de altura (0-10 cm) |
| $\text{sensor}_{e}xp_{b}aja$ | BOOL | I0.1 | Sensor en expulsor de piezas pequeñas |
| $\text{sensor}_{e}xp_{m}ediana$ | BOOL | I0.2 | Sensor en expulsor de piezas medianas |
| emergencia | BOOL | I0.3 | Parada de emergencia (NC) |
| reanudar | BOOL | I0.4 | Botón para reanudar tras lote completo |
| $\text{motor}_{c}inta$ | BOOL | Q0.0 | Motor de la cinta principal |
| $\text{expulsor}_{b}aja$ | BOOL | Q0.1 | Expulsor para piezas pequeñas |
| $\text{expulsor}_{m}ediana$ | BOOL | Q0.2 | Expulsor para piezas medianas |
| $\text{lote}_{c}ompleto$ | BOOL | Q0.3 | Luz indicadora de lote completo |

Cuadro 15.1: Tabla de variables para la línea de empaquetado

### 15.3.2. Variables internas y constantes

- `altura_cm` (REAL): altura escalada.
- `categoria` (INT): 0=pequeña, 1=mediana, 2=grande.
- `cola[0..9]` (ARRAY[0..9] OF INT): cola FIFO circular.
- `ptr_escritura` (INT), `ptr_lectura` (INT).
- `cont_peq, cont_med, cont_grand` (INT): contadores.
- `estado` (INT): 0=NORMAL, 10=LOTE$_{C}OMPLETO$.
- Constantes: `raw_min`=0, `raw_max`=27648 (para altura).

### 15.3.3. Programa en LADDER

#### Escalado de altura

Listing 15.1: Escalado de altura

```
MOVE
EN
IW64 IN OUT #altura_raw

NORM_X INT TO REAL
EN
VALUE := #altura_raw
MIN := 0
MAX := 27648
OUT => #norm

SCALE_X REAL TO REAL
EN
VALUE := #norm
MIN := 0.0
MAX := 10.0
OUT => #altura_cm
```

#### Clasificación y cola FIFO

Listing 15.2: Detección de nueva pieza y almacenamiento en cola

```
// Red: Flanco ascendente en sensor_entrada
+---[ "sensor_entrada" ]---+---[NOT "prev_entrada"]---+---( )--- #
   flanco
+---[ #flanco ]---+---( MOVE )--- #categoria -> "cola"[
   ptr_escritura]
|                 |
|                 +---( ADD )--- "ptr_escritura" + 1 -> "
   ptr_escritura"
|                 |
```

```
|                  +---( )--- #nueva_pieza
+------------------+

// Red: Determinar categor a (se puede hacer con comparadores)
+---[CMP < REAL]---+---( MOVE )--- 0 -> #categoria   // peque a
|   IN1: #altura_cm |
|   IN2: 3.0        |
+-------------------+

+---[CMP >= REAL]---+---[CMP <= REAL]---+---( MOVE )--- 1 -> #
   categoria   // mediana
|   IN1: #altura_cm |   IN1: #altura_cm |
|   IN2: 3.0        |   IN2: 7.0        |
+-------------------+-------------------+

+---[CMP > REAL]---+---( MOVE )--- 2 -> #categoria   // grande
    IN1: #altura_cm
    IN2: 7.0
```

### Avance de la cinta y lectura de cola

Se puede usar un temporizador cíclico para simular el avance de la cinta, o un encoder. Aquí simplificamos usando un impulso cada cierto tiempo.

Listing 15.3: Avance de cinta y actualización de categoría actual

```
// Red: Temporizador para avance (ej. cada 1 segundo)
+---( TON )--- #timer_avance (PT:=T#1s)
    IN := TRUE   // siempre activo, modo astable

+---[ #timer_avance.Q ]---+---( )--- #pulso_avance
+---[ #pulso_avance ]---+---( MOVE )--- "cola"[ptr_lectura] -> #
   cat_actual
|                       |
|                       +---( ADD )--- "ptr_lectura" + 1 -> "
   ptr_lectura"
+-----------------------+

// Ajuste circular de punteros (similar al cap tulo 12)
+---[CMP >= INT]---+---( SUB )--- "ptr_escritura" - 10 -> "
   ptr_escritura"
    IN1: "ptr_escritura"
    IN2: 10
... (igual para lectura)
```

### Expulsores y contadores

Listing 15.4: Activación de expulsores según categoría actual

```
// Expulsor para peque as
+---[CMP == INT]---+---[ "sensor_exp_baja" ]---+---( S )--- "
   expulsor_baja"
```

```
|   IN1: #cat_actual|                              |
|   IN2: 0          |                              |
+-------------------+------------------------------+

// Expulsor para medianas
+---[CMP == INT]---+---[ "sensor_exp_mediana" ]---+---( S )--- "
   expulsor_mediana"
|   IN1: #cat_actual|                                  |
|   IN2: 1          |                                  |
+-------------------+----------------------------------+

// Temporizadores para desactivar expulsores (500 ms)
+---[ "expulsor_baja" ]---+---( TON )--- #T_baja (PT:=T#500ms)
+---[ #T_baja.Q ]---+---( R )--- "expulsor_baja"

+---[ "expulsor_mediana" ]---+---( TON )--- #T_med (PT:=T#500ms)
+---[ #T_med.Q ]---+---( R )--- "expulsor_mediana"

// Contadores por categor a (incrementar cuando se expulsa o al
   final)
+---[ #flanco ]---+---( ADD )--- "cont_peq" + 1 -> "cont_peq"   (si
    categor a 0)
+---[ #flanco ]---+---( ADD )--- "cont_med" + 1 -> "cont_med"   (si
    categor a 1)
+---[ #flanco ]---+---( ADD )--- "cont_grand" + 1 -> "cont_grand" (
   si categor a 2)
```

### Gestión de lote completo

Listing 15.5: Detección de lote completo y parada de cinta

```
// Red: Detectar si alg n contador llega a 10
+---[CMP >= INT]---+---[CMP >= INT]---+---[CMP >= INT]---+---( )---
    #lote_ok
|   IN1: "cont_peq"|   IN1: "cont_med"|   IN1: "cont_grand"|
|   IN2: 10        |   IN2: 10        |   IN2: 10         |
+------------------+------------------+-------------------+

// Red: M quina de estados para lote completo
// Estado 0 = normal, Estado 10 = lote completo
+---[CMP == INT]---+---[ #lote_ok ]---+---( MOVE )--- 10 -> #
   estado_lote
|   IN1: #estado_lote|                    |
|   IN2: 0           |                    |
+--------------------+--------------------+

+---[CMP == INT]---+---( )--- "lote_completo"
|   IN1: #estado_lote|
|   IN2: 10          |
+--------------------+
|
```

```
+---( R )--- "motor_cinta"   // detener cinta

// Red: Reanudar con bot n
+---[CMP == INT]---+---[ "reanudar" ]---+---( MOVE )--- 0 -> #estado_lote
|   IN1: #estado_lote|                  |                  // reset contadores?
|   IN2: 10          |                  |
+--------------------+------------------+

// Al reanudar, se podr an resetear los contadores (opcional)
+---[ "reanudar" ]---+---( MOVE )--- 0 -> "cont_peq"
|                    |
|                    +---( MOVE )--- 0 -> "cont_med"
|                    |
|                    +---( MOVE )--- 0 -> "cont_grand"
+--------------------+
```

### 15.3.4. Consideraciones de seguridad

- La parada de emergencia debe detener la cinta y los expulsores inmediatamente. Incluir una red similar a la del proyecto anterior.
- Asegurar que los expulsores no se activen simultáneamente (por diseño, la categoría es única).

# 16 Proyecto Final - Estación de Mezclado con Control PID

## 16.1. Consigna

Diseñar una estación de mezclado con los siguientes componentes: - Tanque con agitador (motor Q0.0). - Válvula de llenado (Q0.1) y válvula de vaciado (Q0.2). - Sensor de nivel analógico (IW64, 0-10 m). - Sensor de temperatura analógico (IW66, 0-100 °C). - Bomba de recirculación con velocidad variable (salida analógica QW64) para control de temperatura mediante intercambiador. - HMI virtual para ingresar setpoint de nivel y temperatura, y visualizar variables.

## 16.2. Requisitos funcionales

1. Ciclo de llenado: abrir válvula de llenado hasta alcanzar el setpoint de nivel (por ejemplo, 5 m). Cerrar válvula. 2. Activar agitador y control de temperatura: mediante PID, regular la velocidad de la bomba para mantener la temperatura en el setpoint (ej. 50 °C). Usar el sensor de temperatura como PV. 3. Una vez transcurrido un tiempo de mezclado (parametrizable), abrir válvula de vaciado hasta que el nivel sea cero. 4. El ciclo puede repetirse automáticamente o esperar una nueva orden. 5. Alarmas: temperatura >90 °C, nivel >9 m, fallo de sensor (señal fuera de rango).

## 16.3. Guía de desarrollo

- Crear UDT para parámetros de receta (setpoint nivel, setpoint temp, tiempo mezclado). - Implementar máquina de estados. - Configurar PID para temperatura. - Usar arrays o estructuras para manejo de alarmas.

## 16.4. Solución guiada

La estación de mezclado combina control de nivel (todo-nada con histéresis o PID) y control de temperatura (PID). Se propone una máquina de estados con los siguientes estados: REPOSO, LLENADO, MEZCLADO, VACIADO. Se utilizará un UDT para almacenar los parámetros de receta.

### 16.4.1. Asignación de E/S (tags)

| Nombre | Tipo | Dirección | Descripción |
|---|---|---|---|
| inicio | BOOL | I0.0 | Botón de inicio de ciclo |
| emergencia | BOOL | I0.1 | Parada de emergencia (NC) |
| nivel$_raw$ | INT | IW64 | Sensor de nivel (0-10 m) |
| temp$_raw$ | INT | IW66 | Sensor de temperatura (0-100 °C) |
| agitador | BOOL | Q0.0 | Motor del agitador |
| valv$_llenado$ | BOOL | Q0.1 | Válvula de llenado |
| valv$_vaciado$ | BOOL | Q0.2 | Válvula de vaciado |
| bomba$_raw$ | INT | QW64 | Salida analógica a bomba de recirculación |
| luz$_alarma$ | BOOL | Q0.3 | Alarma general |

Cuadro 16.1: Tabla de variables para la estación de mezclado

### 16.4.2. UDT para recetas

Listing 16.1: Definición de UDT Receta

```
TYPE Receta :
STRUCT
    nivel_sp : REAL;      // Setpoint de nivel (m)
    temp_sp : REAL;       // Setpoint de temperatura ( C )
    tiempo_mezcla : TIME; // Tiempo de mezclado
END_STRUCT;
END_TYPE
```

Se puede crear un array de recetas en un DB global:

```
VAR_GLOBAL
    recetas : ARRAY[1..3] OF Receta;
END_VAR
```

Y una variable `receta_actual` (INT) para seleccionar la receta.

### 16.4.3. Variables internas y constantes

- `nivel_real, temp_real` (REAL): valores escalados.
- `estado` (INT): 0=REPOSO, 10=LLENADO, 20=MEZCLADO, 30=VACIADO.
- `timer_mezcla` (TON).
- `bomba_speed` (REAL): salida del PID de temperatura.
- `PID_Temp`: instancia del bloque PID_Compact.

### 16.4.4. Escalado de señales

Listing 16.2: Escalado de nivel y temperatura

```
// Nivel: 0-10 m, sensor 4-20 mA (raw_min=5530, raw_max=27648)
MOVE
EN
IW64 IN OUT #nivel_raw
... (similar a cap tulos anteriores)

// Temperatura: 0-100  C , asumimos sensor 0-10V (raw_min=0,
   raw_max=27648)
MOVE
EN
IW66 IN OUT #temp_raw

NORM_X INT TO REAL
EN
VALUE := #temp_raw
MIN := 0
MAX := 27648
OUT => #norm_temp

SCALE_X REAL TO REAL
EN
VALUE := #norm_temp
MIN := 0.0
MAX := 100.0
OUT => #temp_real
```

### 16.4.5. Máquina de estados

Listing 16.3: Máquina de estados en LADDER

```
// Red: Estado REPOSO (0)
+---[CMP == INT]---+---[ "inicio" ]---+---[NOT "emergencia"
   ]---+---( MOVE )--- 10 -> #estado
|   IN1: #estado   |                   |                          |
|   IN2: 0         |                   |                          |
+------------------+-------------------+--------------------------+

// Red: Estado LLENADO (10)
+---[CMP == INT]---+---( )--- "valv_llenado"
|   IN1: #estado   |
|   IN2: 10        |
+------------------+
|
+---[CMP >= REAL]---+---( MOVE )--- 20 -> #estado   // nivel
   alcanzado
    IN1: #nivel_real
    IN2: recetas[receta_actual].nivel_sp

// Red: Estado MEZCLADO (20)
+---[CMP == INT]---+---( )--- "agitador"
```

```
|   IN1: #estado   |
|   IN2: 20        |
+------------------+
|
+---( TON )--- #timer_mezcla (PT:=recetas[receta_actual].
   tiempo_mezcla)
    IN := TRUE
|
+---[ #timer_mezcla.Q ]---+---( MOVE )--- 30 -> #estado

// Red: Estado VACIADO (30)
+---[CMP == INT]---+---( )--- "valv_vaciado"
|   IN1: #estado   |
|   IN2: 30        |
+------------------+
|
+---[CMP <= REAL]---+---( MOVE )--- 0 -> #estado   // nivel m nimo
    (por ejemplo, <0.1 m)
    IN1: #nivel_real
    IN2: 0.1
```

### 16.4.6. Control PID de temperatura

El PID debe actuar durante el estado MEZCLADO. Se puede llamar al bloque PID_Compact en una red condicionada.

Listing 16.4: Llamada al PID de temperatura

```
// Red: Llamada al PID solo en estado MEZCLADO
+---[CMP == INT]---+---( CALL "PID_Compact", "DB_PID_Temp" )---+
|   IN1: #estado   |   Setpoint := recetas[receta_actual].temp_sp|
|   IN2: 20        |   Input := #temp_real                       |
+------------------+   Output => #bomba_speed                    +
                   |                                              |
                   +----------------------------------------------+

// Red: Escalar salida del PID (0-1) a valor anal gico (0-27648)
MUL REAL
EN
IN1 := #bomba_speed
IN2 := 276.48
OUT => #bomba_scaled

CONV REAL TO INT
EN
IN := #bomba_scaled
OUT => #bomba_int

MOVE
EN
#bomba_int IN OUT QW64
```

### 16.4.7. Alarmas

Listing 16.5: Detección de alarmas

```
// Alarma por temperatura alta
+---[CMP > REAL]---+---( S )--- #alarma_temp
|   IN1: #temp_real |
|   IN2: 90.0       |
+-------------------+

// Alarma por nivel alto
+---[CMP > REAL]---+---( S )--- #alarma_nivel
|   IN1: #nivel_real|
|   IN2: 9.0        |
+-------------------+

// Alarma por fallo de sensor (valor fuera de rango)
+---[CMP < INT]---+---[CMP > INT]---+---( S )--- #alarma_sensor
|   IN1: #nivel_raw|   IN1: #nivel_raw|
|   IN2: 5530      |   IN2: 27648     |
+------------------+------------------+

// Salida de alarma general (OR de las alarmas)
+---[ "alarma_temp" ]---+---( )--- "luz_alarma"
+---[ "alarma_nivel" ]---+
+---[ "alarma_sensor" ]---+

// Reset de alarmas con bot n (no implementado)
```

### 16.4.8. Parada de emergencia

Similar a proyectos anteriores, la emergencia debe forzar estado 0 y desactivar todas las salidas.

### 16.4.9. Consideraciones finales

- Se puede añadir una HMI virtual para seleccionar receta y visualizar variables.
- El PID debe ser parametrizado adecuadamente (valores de ganancia, tiempo integral, etc.) mediante autosintonía o ajuste manual.
- La lógica de vaciado podría incluir un tiempo de espera para asegurar que el tanque se vacía completamente.

# 17 Control de Variadores de Frecuencia y Actuadores Analógicos

## 17.1. Introducción a los Variadores de Frecuencia (VFD)

Los variadores de frecuencia (Variable Frequency Drives, VFD) son dispositivos electrónicos diseñados para controlar la velocidad y el par de motores eléctricos de corriente alterna, variando la frecuencia y tensión suministradas. En la automatización industrial moderna, los VFD son fundamentales para procesos que requieren:

- Control preciso de velocidad en bombas, ventiladores y cintas transportadoras.
- Ahorro energético mediante la regulación de caudal en lugar de válvulas o dampers.
- Arranques y paradas suaves que reducen el estrés mecánico y eléctrico.
- Integración con lazos de control PID para mantener variables como presión, caudal o nivel.

Un variador de frecuencia típico recibe una consigna de velocidad (setpoint) desde un PLC y, basándose en esa consigna, genera la tensión y frecuencia adecuadas para el motor. Además, puede enviar información de retorno al PLC, como velocidad real, corriente, temperatura o estado de alarmas [citation:3].

## 17.2. Métodos de conexión entre PLC S7-1200 y variadores

Existen dos enfoques principales para conectar un PLC S7-1200 con un variador de frecuencia:

### 17.2.1. Conexión mediante salida analógica

Es el método más simple y universal. El PLC envía una señal analógica (típicamente 0-10 V o 4-20 mA) que el variador interpreta como consigna de velocidad. La principal ventaja es su simplicidad y compatibilidad con cualquier variador que acepte entradas analógicas. Sin embargo, presenta limitaciones:

- Una sola señal por cable (solo consigna, sin retorno de información).
- Susceptibilidad al ruido eléctrico en entornos industriales.
- Resolución limitada por la conversión A/D del variador.

### 17.2.2. Conexión mediante bus de campo (comunicación digital)

Este método utiliza protocolos de comunicación digital como USS, Modbus RTU o PROFINET. El PLC y el variador intercambian telegramas de datos que pueden incluir consignas, parámetros y lecturas de estado. Las ventajas son:

- Comunicación bidireccional: el PLC puede leer velocidad real, corriente, temperatura y alarmas.
- Múltiples variadores en un mismo bus (hasta 16 con USS o 32 con Modbus).
- Mayor inmunidad al ruido.
- Posibilidad de leer y modificar parámetros internos del variador [citation:6].

Siemens ofrece bibliotecas específicas para cada protocolo:

- **USS**: Para comunicación serie RS-485 con variadores como SINAMICS G110, G120, V20 [citation:7].
- **Modbus RTU**: Para variadores de terceros o cuando se requiere este estándar [citation:10].
- **PROFINET**: Para integración nativa con variadores SINAMICS usando telegramas estándar [citation:1].

## 17.3. Configuración de hardware en TIA Portal

### 17.3.1. Para salida analógica

1. En la configuración del PLC, agregar un módulo de salida analógica (por ejemplo, SM 1232 AQ 2x14 bits) o utilizar la Signal Board integrada en algunos modelos.
2. Anotar la dirección asignada (por ejemplo, `QW64` para el primer canal).
3. Verificar el rango configurado (0-10V o 4-20mA) según la entrada del variador.

### 17.3.2. Para comunicación USS

1. Agregar un módulo de comunicación serie CM 1241 RS485 (o utilizar CB 1241 en CPUs con conector integrado).
2. Configurar la velocidad (baud rate) y otros parámetros del puerto.
3. Obtener el hardware identifier del puerto (disponible en las constantes del sistema) [citation:6].

### 17.3.3. Para comunicación PROFINET

1. Agregar el variador como dispositivo en la red PROFINET (por ejemplo, SINAMICS G120).
2. Asignar una dirección IP y un nombre de dispositivo.
3. Configurar el telegrama de comunicación (estándar 1, 20, 352, etc.) según los datos que se deseen intercambiar.

## 17.4. Programación en LADDER y SCL

### 17.4.1. Control mediante salida analógica

El método más directo consiste en generar una consigna de velocidad (por ejemplo, de 0 a 100 %), escalarla al rango de la salida analógica (0-27648) y enviarla al variador.

Listing 17.1: Control de variador con salida analógica en LADDER

```
// Red 1: Definir setpoint de velocidad desde HMI o variable
   interna
MOVE
EN
50.0 IN OUT #setpoint_porcentaje   // 50% de velocidad

// Red 2: Limitar setpoint entre 0 y 100
LIMIT REAL
EN
MN := 0.0
IN := #setpoint_porcentaje
MX := 100.0
OUT => #setpoint_limitado

// Red 3: Escalar a valor anal gico (0-27648)
NORM_X REAL TO REAL
EN
VALUE := #setpoint_limitado
MIN := 0.0
MAX := 100.0
OUT => #norm

SCALE_X REAL TO INT
EN
VALUE := #norm
MIN := 0
MAX := 27648
OUT => #salida_analogica

// Red 4: Enviar a la salida f sica
MOVE
EN
#salida_analogica IN OUT QW64
```

En SCL, el mismo programa se simplifica:

Listing 17.2: Control de variador en SCL

```
#setpoint_limitado := LIMIT(0.0, #setpoint_porcentaje, 100.0);
#salida_analogica := REAL_TO_INT(#setpoint_limitado / 100.0 *
   27648.0);
QW64 := #salida_analogica;
```

### 17.4.2. Control mediante comunicación USS

La biblioteca USS en TIA Portal proporciona bloques específicos para la comunicación con variadores [citation:6][citation:8]:

- **USS_Port_Scan**: Se encarga de la comunicación a nivel de puerto. Debe ejecutarse periódicamente (por ejemplo, en un OB de ciclo de 30-50 ms).
- **USS_Drive_Control**: Controla un variador individual. Se necesita una instancia por cada variador en el bus.

Listing 17.3: Configuración de comunicación USS en LADDER

```
// Red 1: Llamada a USS_Port_Scan en OB30 (ciclo de 30ms)
CALL "USS_Port_Scan"
PORT := 269            // Hardware identifier del CM1241
BAUD := 9600           // Misma velocidad configurada en variadores
USS_DB := "USS_Drive_DB"  // DB de instancia compartido
ERROR => #port_error
STATUS => #port_status

// Red 2: Control del variador 1 (en OB1 principal)
CALL "USS_Drive_Control", "VFD1_DB"
RUN := #marcha_VFD1
OFF2 := TRUE           // Habilitar paro natural
OFF3 := TRUE           // Habilitar paro r pido
F_ACK := #reset_fallo
DIR := #direccion_VFD1
DRIVE := 1             // Direcci n USS del variador
PZD_LEN := 2           // Longitud de datos PZD (palabras)
SPEED_SP := #setpoint_VFD1
NDR => #datos_nuevos
ERROR => #error_VFD1
STATUS => #status_VFD1
RUN_EN => #vfd1_en_marcha
D_DIR => #vfd1_direccion
INHIBIT => #vfd1_inhibido
FAULT => #vfd1_fallo
SPEED => #velocidad_real_VFD1
```

La variable `SPEED` en la salida del bloque proporciona la velocidad real del motor en porcentaje, lo que permite cerrar el lazo de control o simplemente monitorizar el proceso.

### 17.4.3. Control mediante PROFINET con SINA_SPEED

Para variadores SINAMICS conectados vía PROFINET, Siemens proporciona bloques de librería como `SINA_SPEED` [citation:1].

Listing 17.4: Control con SINA_SPEED en SCL

```
// Instanciar bloque SINA_SPEED para un variador
#sina_speed_inst(
    axisName := "VFD1",           // Nombre configurado en hardware
    modePos := 0,                  // 0: speed control
    enableAxis := #habilitar,
    speedSp := #setpoint_rpm,      // Consigna en RPM
    configEPos := 0,
    position := 0,
    velocity := 0,
    overstore := 0,
    cancelTraversing := 0,
    intermediateStop := 0,
    fixedStop := 0,
    syncMode := 0,
    axisDisabled => #eje_deshabilitado,
    axisEnabled => #eje_habilitado,
    speedAct => #velocidad_actual,
    statusPos => #estatus_palabra,
    error => #error_sina,
    status => #status_sina
);
```

## 17.5. Ejemplo integrador: control de caudal con PID y variador

Consideremos un sistema de bombeo donde se desea mantener un caudal constante. El sistema consta de:

- Bomba centrífuga accionada por motor trifásico con variador SINAMICS G120C.
- Sensor de caudal analógico (4-20 mA) conectado a entrada analógica del PLC.
- PLC S7-1200 con comunicación PROFINET al variador.

El lazo de control funcionará de la siguiente manera:

1. Se lee el caudal actual del sensor y se escala a unidades de ingeniería (L/min).
2. Un bloque PID_Compact compara el caudal real con el setpoint deseado.
3. La salida del PID (0-100 %) se envía como consigna de velocidad al variador mediante PROFINET.
4. El variador ajusta la velocidad de la bomba para mantener el caudal en el valor deseado.

Listing 17.5: Lazo PID para control de caudal en SCL

```
// Lectura y escalado de caudal
#caudal_raw := IW64;
#caudal_real := INT_TO_REAL(#caudal_raw - 5530) / INT_TO_REAL(27648
    - 5530) * 100.0; // 0-100 L/min

// Llamada al PID_Compact
#PID_Instance(Setpoint := #caudal_sp,
              Input := #caudal_real,
              Output => #velocidad_consigna);

// Enviar consigna al variador mediante PROFINET
#sina_speed_inst(
    axisName := "BOMBA",
    modePos := 0,
    enableAxis := #bomba_habilitada,
    speedSp := #velocidad_consigna * 1400.0 / 100.0, // Convertir %
        a RPM (base 1400 RPM)
    speedAct => #velocidad_actual
);

// Alarmas y protecciones
IF #caudal_real > 95.0 THEN
    #alarma_caudal_alto := TRUE;
ELSIF #caudal_real < 5.0 THEN
    #alarma_caudal_bajo := TRUE;
ELSE
    #alarma_caudal_alto := FALSE;
    #alarma_caudal_bajo := FALSE;
END_IF;
```

## 17.6. Lectura de parámetros y diagnóstico

Una de las grandes ventajas de la comunicación digital es la posibilidad de leer parámetros internos del variador, como corriente, temperatura o estado de alarmas.

Para USS, el bloque `USS_Drive_Control` proporciona acceso a los parámetros mediante el mecanismo PKW (Parameter ID/Value). Para PROFINET, se pueden configurar telegramas más largos que incluyan estas variables.

Listing 17.6: Lectura de corriente del motor en USS

```
// Configurar par metro a leer (ej. r0009: corriente real)
#param_request := 9;      // N mero de par metro
#param_index := 0;        //  ndice  (para arrays)
#param_value := 0;

// Enviar petici n mediante USS (requiere configuraci n adicional
   )
// ...
```

## 17.7. Ejercicios propuestos

1. **Control analógico básico**: Configure una salida analógica del S7-1200 para controlar un variador. Programe en LADDER una rampa de aceleración que lleve el motor de 0 a 100 % en 10 segundos al presionar un botón de marcha, y lo detenga en rampa de 5 segundos al presionar parada.

2. **Comunicación USS**: Implemente un sistema con dos variadores SINAMICS V20 conectados mediante RS485. Desde una HMI, permita seleccionar qué variador controlar, ajustar su velocidad (0-100 %) y visualizar su velocidad real y estado de fallo. Incluya un botón de reset de fallos.

3. **Control PID con variador**: Utilizando el escenario "Tank"de Factory I/O, configure un variador para controlar la velocidad de la bomba de llenado. Implemente un control PID que mantenga el nivel en 5 metros. Ajuste los parámetros del PID para obtener una respuesta estable sin sobrepaso excesivo. Observe el comportamiento con y sin acción integral.

4. **Integración PROFINET**: Si dispone de un variador SINAMICS G120 real o simulado, configure la comunicación PROFINET y programe un bloque `SINA_SPEED`. Lea la velocidad real y la corriente del motor, y muéstrelas en una pantalla HMI.

5. **Proyecto integrador**: Diseñe una estación de bombeo con tres bombas controladas por variadores, donde se mantenga la presión de una tubería constante mediante un PID. Implemente un algoritmo de alternancia de bombas para igualar horas de funcionamiento y garantizar redundancia.

## 17.8. Referencias y recursos adicionales

- Siemens Industry Online Support: *USS communication with SINAMICS V20* [citation:6]
- Siemens Industry Online Support: *Configuring SINA_SPEED for SINAMICS drives* [citation:1]
- Siemens *S7-1200 System Manual* - Comunicación con variadores
- Repositorio USM: Diseño de sistema de control S7-1200 con SINAMICS G120C [citation:1]
- Electrotec: Configuración de salida analógica S7-1200 [citation:9]

# A Anexo A: Tabla de Tipos de Datos en TIA Portal

| **Tipo** | **Descripción** | **Rango** |
|---|---|---|
| BOOL | Booleano | TRUE/FALSE |
| BYTE | Entero sin signo 8 bits | 0..255 |
| WORD | Entero sin signo 16 bits | 0..65535 |
| DWORD | Entero sin signo 32 bits | 0..4294967295 |
| INT | Entero con signo 16 bits | -32768..32767 |
| DINT | Entero con signo 32 bits | -2147483648..2147483647 |
| REAL | Coma flotante 32 bits | ±1.18e-38 a ±3.40e38 |
| S5TIME | Duración en formato S5 | 0..9990 ms, etc. |
| TIME | Duración en ms | -24d20h31m23s647ms a +24d20h31m23s647ms |
| STRING | Cadena de caracteres | hasta 254 caracteres |

# B Anexo B: Resumen de Instrucciones de Escalado y Comparación

- **NORM_X**: Normaliza un valor entero a real entre 0.0 y 1.0.
- **SCALE_X**: Convierte un valor normalizado a un rango físico.
- **CMP**: Compara dos valores (EQ, NE, GT, LT, GE, LE).
- **LIMIT**: Limita un valor entre un mínimo y un máximo.
- **SEL**: Selecciona entre dos valores según una condición booleana.
- **MAX/MIN**: Devuelve el máximo/mínimo de dos valores.

# C Anexo C: Guía Rápida de SCL

```
// Declaraci n de variables en FB/FC
VAR_INPUT
    in1 : INT;
END_VAR
VAR_OUTPUT
    out1 : REAL;
END_VAR
VAR_TEMP
    temp : BOOL;
END_VAR

// Asignaci n
out1 := INT_TO_REAL(in1) * 2.5;

// Condicional
IF temp THEN
    out1 := 0.0;
END_IF;

// Bucle
FOR i := 1 TO 10 DO
    array[i] := i * 2;
END_FOR;
```

# D Anexo D: Configuración de Escenarios en Factory I/O para Simulación Analógica

Pasos para usar un sensor analógico: 1. En Factory I/O, seleccionar un sensor con salida analógica (ej. Ultrasonic Sensor). 2. En sus propiedades, asignar la dirección de entrada (ej. IW64). 3. Configurar el rango físico del sensor (ej. 0-100 cm). 4. En el PLC, leer IW64 y escalar según el rango configurado.

# E Anexo E: Glosario de Términos Avanzados

**SCL** Structured Control Language.

**PID** Proporcional-Integral-Derivativo.

**SP** Setpoint.

**PV** Process Variable.

**UDT** User Defined Type.

**FIFO** First In First Out (cola).

**Windup** Acumulación excesiva del término integral.

**Histeresis** Diferencia entre umbral de activación y desactivación.