

ESTUDIO

INDICE

ESTUDIO DEL FUNCIONAMIENTO DE UN PLC.....	3
INTRODUCCIÓN.....	3
ARQUITECTURA.....	3
LENGUAJES DE PROGRAMACIÓN (CX-PROGRAMMER).....	4
LÓGICA LADDER.....	4
DIAGRAMAS DE BLOQUE DE FUNCIONES (SFD).....	5
DIAGRAMAS DE FUNCIONES SECUENCIALES (SFC).....	5
SISTEMAS SCADA.....	6
SOBRE EL PLC FÍSICO DEL LABORATORIO.....	7
CARACTERÍSTICAS.....	7
RECONOCIMIENTO DEL CUADRO ELECTRÓNICO.....	11
IMAGEN GENERAL.....	11
EXPLICACIÓN DE CADA COMPONENTE.....	13
MODO DE OPERACIÓN.....	18
ESTABLECIMIENTO DE CONECTIVIDAD CON PLCS VÍA RS232 - USB.....	19
RS232 INTRODUCCIÓN.....	19
CONEXIÓN CON PLC CJ2M-CPU31.....	20
ESTUDIO DEL FUNCIONAMIENTO DE LA WWTP EN SIMULINK.....	21
WWTP INTRODUCCIÓN.....	21
DISEÑO DEL BANCO DE PRUEBAS Y PRUEBAS DE ATAQUE.....	24
ESTUDIO DE LA TECNOLOGÍA DOCKER.....	27
DOCKER BEGINNER TUTORIAL.....	27
DOCKER COMPOSE TUTORIAL.....	30
CREACIÓN DOCKERFILE.....	31
CONSTRUIR, EJECUTAR Y EMPUJAR LA IMAGEN.....	31
TERMINOLOGÍA.....	32
DIRECTORIO.....	33
ESTUDIO DEL FUNCIONAMIENTO DEL PLC VIRTUAL.....	34
HARDWARE.....	34
SOFTWARE.....	35
COMUNICACIÓN MODBUS.....	35
BOARD COMUNICACIÓN.....	35

ESTUDIO DEL PROTOCOLO MODBUS.....	36
INTRODUCCIÓN AL PROTOCOLO MODBUS.....	36
DESCRIPCIÓN DEL PROTOCOLO.....	37
MODELO DE DATOS MODBUS.....	38
TRANSACCIONES EN MODBUS.....	39
MODELO MODBUS TCP/IP.....	39
Modelo Cliente/Servidor:.....	39
Descripción del Protocolo.....	40
MODELO DE ARQUITECTURA.....	41
ESTUDIO DEL MÉTODO DE TESTEO HARDWARE IN THE LOOP.....	42
INTRODUCCIÓN.....	42
ESTUDIO DE MÉTRICAS DE RENDIMIENTO.....	43
MÉTRICAS DE PFGs.....	43
PROTOCOLO MODBUS.....	43
OPENPLC.....	46
HARDWARE IN THE LOOP.....	46
MÉTRICAS A ANALIZAR.....	50
Network Performance: (Rendimiento de la red).....	50
Latency: (Latencia).....	50
Throughput: (Rendimiento).....	50
Reliability: (Fiabilidad).....	50
Availability: (Disponibilidad de red).....	50
MÉTRICAS DE PAPERS.....	50
OpenPLC and lib61850 Smart Grid Testbed: Performance Evaluation and Analysis of GOOSE Communication.....	50
Testbed Design.....	51
Configuración y metodología del experimento.....	51
Resultados.....	52

ESTUDIO DEL FUNCIONAMIENTO DE UN PLC

INTRODUCCIÓN

El PLC, o Controlador Lógico Programable, se encarga de controlar y supervisar el funcionamiento de maquinarias y procesos, lo que lo convierte en una herramienta fundamental para la eficiencia y seguridad en la producción industrial. Es un dispositivo electrónico utilizado en la automatización industrial para controlar y monitorear procesos en tiempo real. Su función principal es recibir información de los sensores y actuadores, procesarla y enviar señales de control a los actuadores para que realicen las acciones necesarias.

ARQUITECTURA

El PLC consta de:

- Una unidad procesadora (CPU) que interpreta las entradas, ejecuta el programa de control almacenado en la memoria y envía señales de salida.
- Una fuente de alimentación que convierte la tensión alterna en continua.
- Una unidad de memoria que almacena los datos de las entradas y el programa que debe ejecutar el procesador.
- Módulos de entrada y salida, en la que el controlador recibe y envía datos desde/hacia dispositivos externos.
- Una interfaz de comunicaciones para recibir y transmitir datos en redes de comunicación desde/hacia PLC remotos.

Los PLC requieren un dispositivo de programación que se utiliza para desarrollar y posteriormente descargar el programa creado en la memoria del controlador

Nuestro PLC es modular y ofrece espacio para módulos con distintas funciones, como la fuente de alimentación, el procesador, la selección de módulos de E/S y las interfaces de comunicación, que pueden personalizarse para cada aplicación concreta.

Las señales discretas (digitales) sólo pueden tener valor de encendido o apagado (1 o 0, verdadero o falso).

Las señales analógicas pueden utilizar voltaje o corriente proporcional al tamaño de la variable monitorizada y pueden tomar cualquier valor dentro de su escala. La presión, la temperatura, el caudal y el peso suelen representarse mediante señales analógicas. El PLC tomará este valor y lo transpondrá a las unidades deseadas del proceso para que el operador o el programa puedan leerlo.

LENGUAJES DE PROGRAMACIÓN (CX-PROGRAMMER)

La programación de un PLC se realiza mediante un software específico que permite la creación de programas lógicos en lenguajes de programación. Una vez programado, el PLC puede funcionar de forma autónoma, sin necesidad de una supervisión constante. Para ello, se utilizan las siguientes técnicas:

Manual de diagrama de bloques (FBD) y structured test (ST) para cx programmer:

https://assets.omron.eu/downloads/manual/en/v6/w447_cx-programmer_fb_st_operation_manual_en.pdf

LÓGICA LADDER

Es el lenguaje más utilizado y consiste en una estructura de escalera por donde todo funciona a partir de dos carriles verticales, uno izquierdo (L1), el que recibe el voltaje (entrada) y deja pasar la energía al riel derecho (L2) que actúa como tierra (salida). Su lectura siempre sigue el mismo patrón, de izquierda a derecha y de arriba hacia abajo. En su interior se colocan los símbolos de contactos, relés, funciones de bloque y operaciones lógicas.

Basic instructions		
Symbol	List code	Action
- -	LD	Evaluates to true if the physical contact it represents is closed or on.
-	LDB	Opposite to LD.
-○-	OUT	Evaluates to true, and thus energises a normally open coil, if there is a continuous "true" path from the left-hand side of the ladder to it.
-○-	OUTB	Opposite to OUT

If the data table bit is	The status of the instruction is		
	XIC EXAMINE IF CLOSED	XIO EXAMINE IF OPEN	OTE OUTPUT ENERGIZE
Logic 0	False	True	False
Logic 1	True	False	True

Más detallado en:

https://www.tud.ttu.ee/im/Andres.Rahni/Automaatika%20alused/77511_05_LAD_and_FB_D.pdf

DIAGRAMAS DE BLOQUE DE FUNCIONES (SFD)

Todas las funciones se colocan en bloques de funciones. Todos tienen una o más entradas y salidas. El bloque de función se ilustra con un recuadro. En el centro del recuadro suele haber un símbolo o un texto. Este símbolo representa la funcionalidad real del bloque de función. Dependiendo de la función, puede haber un número indefinido de entradas y salidas. Se puede conectar la salida de un bloque de función a la entrada de otro. De este modo se crea un diagrama de bloques de función.

Existen muchos bloques de función estándar, pero también se pueden crear bloques de función propios. Por ejemplo, a veces habrá que usar el mismo fragmento varias veces en el programa y para ello se pueden crear bloques de función específicos y utilizarlos varias veces.

Visión general de los bloques más importantes:

https://www.plcacademy.com/function-block-diagram-programming/?utm_content=cmp-true

- Operaciones Lógicas (OR, AND, NOR, NAND, XOR)
- Operación de asignación (=)
- Set/Reset o Reset/Set
- Timer: TP, TON, TOF
- Counter: CTU (Up counter), CTD (Down counter), CTUD (Up Down counter)
- Edge detection: R_Trig (Rising edge), F_Trig (Falling edge)
- Equality, inequality, less than, greater than
- ...

DIAGRAMAS DE FUNCIONES SECUENCIALES (SFC)

Es una representación de secuencias de control en un programa en el que se pueden organizar subrutinas o etapas que van afectando el producto de las funciones posteriores. La energía fluye de un punto a otro siempre y cuando se haya cumplido una condición. Este lenguaje proviene del estándar GRAFCET que también utiliza etapas, transiciones y acciones para su funcionamiento.

Las secuencias SFC se representan por cajas rectangulares que contienen las etapas que están conectadas por líneas verticales llamadas transiciones, por último están las condiciones (verdadero o falso) que desbloquean la acción para seguir con las funciones siguientes.

SISTEMAS SCADA

El control de supervisión y adquisición de datos (SCADA) es una arquitectura de sistema de control que comprende ordenadores, comunicaciones de datos en red e interfaces gráficas de usuario para la supervisión de alto nivel de máquinas y procesos. También incluye sensores y otros dispositivos, como controladores lógicos programables, que interactúan con la maquinaria o las instalaciones de proceso.

Un sistema SCADA suele constar de los siguientes elementos principales:

- Ordenadores de supervisión:

Es el núcleo del sistema SCADA, ya que recopila datos sobre el proceso y envía órdenes de control a los dispositivos conectados sobre el terreno. Se refiere al ordenador y al software responsables de la comunicación con los controladores de conexión de campo, que son las RTU y los PLC, e incluye el software HMI que se ejecuta en las estaciones de trabajo de los operadores.

- Unidades Terminales Remotas (RTU):

Las RTU se conectan a los sensores y actuadores del proceso y se conectan en red al sistema informático de supervisión. Tienen funciones de control integradas y a menudo cumplen la norma IEC 61131-3 de programación y admiten la automatización mediante lógica de escalera, un diagrama de bloques de funciones o una variedad de otros lenguajes.

- Controladores Lógicos Programabables (PLC):

Los PLC se conectan a los sensores y actuadores del proceso y se conectan en red al sistema de supervisión.

- Infraestructuras de comunicación:

Conecta el sistema informático de supervisión con las RTU y los PLC, y puede utilizar protocolos estándar de la industria o propios del fabricante. Tanto las RTU como los PLC operan de forma autónoma en el control casi en tiempo real del proceso, utilizando la última orden dada desde el sistema de supervisión. El fallo de la red de comunicaciones no detiene necesariamente los controles de proceso de la planta, y al reanudarse las comunicaciones, el operador puede continuar con la supervisión y el control. Algunos sistemas críticos tendrán autopistas de datos redundantes dobles, a menudo cableadas a través de diversas rutas.

- Interfaz Hombre-Máquina (HMI):

Las HMI son las ventanas del operador del sistema de supervisión. Presenta la información de la planta al personal de operación gráficamente en forma de diagramas mímicos, que son una representación esquemática de la planta que se está controlando. La HMI está conectada al ordenador de supervisión SCADA para proporcionar datos en tiempo real. En muchas instalaciones, la HMI es la interfaz gráfica de usuario para el operador, recoge todos los datos de los dispositivos externos, crea informes, realiza alarmas, envía notificaciones, etc.

SOBRE EL PLC FÍSICO DEL LABORATORIO

OMRON SYSMAC CJ2M CPU31 PROGRAMMABLE CONTROLLER:

https://assets.omron.eu/downloads/latest/datasheet/en/p059_cj2-series_programmable_controller_datasheet_en.pdf?v=7

Program capacity	25K steps (creo que es 5K)
Data memory capacity	64 K words
Logic execution time	0.04 µs
Max. number of expansion units	40
Max. number of local I/O points	2560
Number of built-in digital I/Os	0
Communication port(s)	EtherNet/IP, Ethernet TCP/IP, USB

[CJ1W-PA202](#) / [CJ2M CPU31](#) / [CJ1W-PRM21](#) / [CJ1W-ID211](#) / [CJ1W-OD212](#) / [CJ1W-MAD42](#)

CARACTERÍSTICAS

OMRON SYSMAC CJ2M CPU31 PROGRAMMABLE CONTROLLER:

Dirección Mac: 0000 0A B5 7F 67

CJ1W-PA202: Fuente de alimentación 100 a 240Vca 5Vcc 2,8A Relé

<https://industrial.omron.es/es/products/CJ1W-PA202>

Type of module	Power supply
Type of voltage (input voltage)	AC
Power output	14 W
Input voltage at AC 50 Hz	85-264 V

CJ1W-PRM21: Módulo maestro Profibus DP

<https://industrial.omron.es/es/products/CJ1W-PRM21>

I/O system	CJ I/O Bus
Expansion unit type	CPU Bus Unit
Type of module	Communication
IO-Link master	No
Communication port(s)	PROFIBUS DP Master
Interfaz Profibus	Conecotor sub D hembra de 9 patillas equivalente al puerto RS-232C

CJ1W-ID211: Basic I/O Units (Input Units)<https://industrial.omron.es/es/products/CJ1W-ID211>https://assets.omron.eu/downloads/latest/datasheet/en/cj1w-id_ia_datasheet_en.pdf?v=6

I/O system	CJ I/O Bus
Expansion unit type	Basic I/O Unit
Type of module	Digital I/O
Number of digital inputs	16
Digital input type	PNP/NPN
Permitted voltage at input	20.4-26.4 V
OFF/ON delay	0-32 ms
Number of digital outputs	0
I/O connection type	Screw

CJ1W-OD212: Módulo 16 Salidas PNP Term.<https://industrial.omron.es/es/products/CJ1W-OD212>https://assets.omron.eu/downloads/latest/datasheet/en/cj1w-oc_oa_od_datasheet_en.pdf?v=6

I/O system	CJ I/O Bus
Expansion unit type	Basic I/O Unit
Type of module	Digital I/O
Number of digital inputs	0
OFF/ON delay	0-32 ms
Number of digital outputs	16
Digital output type	PNP
Permitted voltage at output	20.4-26.4 V
Output current	0.5 A
Short-circuit protected outputs	SI
I/O connection type	Screw
Number of I/O connectors	1

Detachable I/O connector	Si
Suitable for safety functions	NO
Number of IOV (V+) terminals	1
Number of IOG (V-) terminals	1
Number of COM terminals	0

CJ1W-MAD42: Módulo analógico mixto 4 entradas, 2 salidas

<https://industrial.omron.es/es/products/CJ1W-MAD42>

I/O system	CJ I/O Bus
Expansion unit type	Special I/O Unit
Type of module	Analog I/O
Number of analog inputs	4
Linear analog input type	-10 to 10 V, 0 to 10 V, 0 to 5 V, 1 to 5 V, 4 to 20 mA
Resolution of the analog inputs	13 Bit
Number of analog outputs	2
Linear analog output type	-10 to 10 V, 0 to 10 V, 0 to 5 V, 1 to 5 V, 4 to 20 mA
Resolution of the analog outputs	13 Bit
Conversion time	4000 ms/unit
I/O connection type	Screw
Number of I/O connectors	1
Detachable I/O connector	Si

CJ1W-MAD42: Módulo analógico mixto 4 entradas, 2 salidas

<https://industrial.omron.es/es/products/CJ1W-ETN21>

■ Ethernet Unit

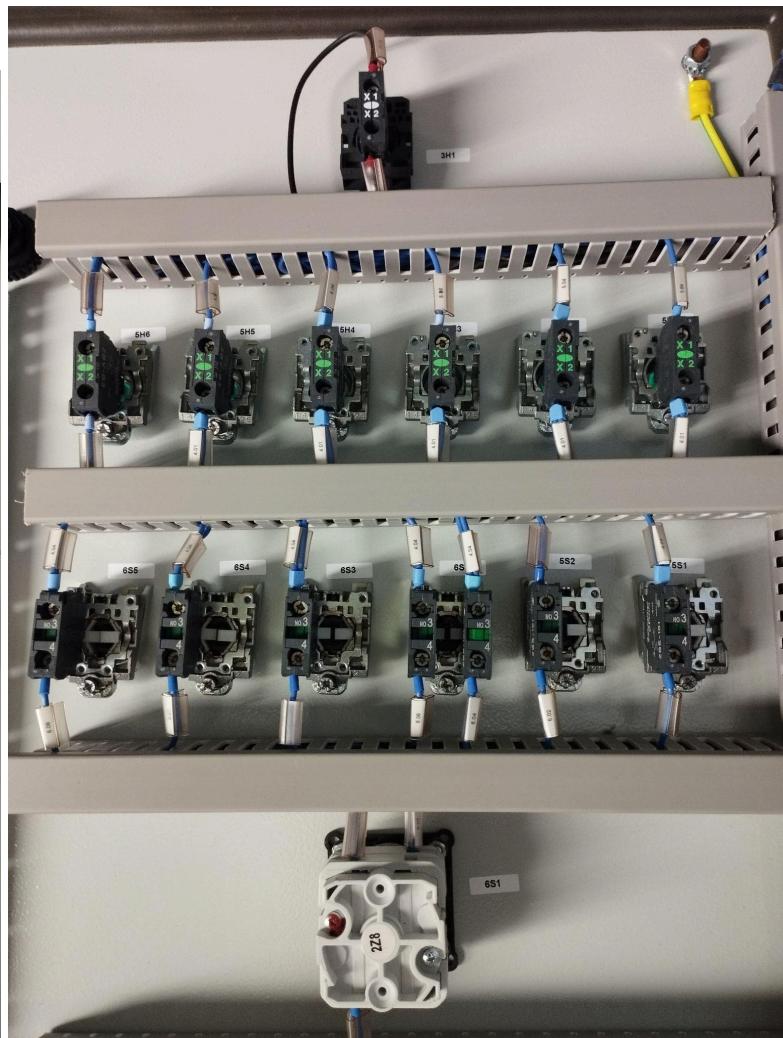
Unit classification	Product name	Specifications			No. of unit numbers allocated	Current consumption (A)		Model	Standards
		Communications cable	Communications functions	Max.Units mountable per CPU Unit		5 V	24 V		
CJ1 CPU Bus Unit	Ethernet Unit 	100Base-TX	FINS communications service (TCP/IP, UDP/IP), FTP server functions, socket services, mail transmission service, mail reception (remote command receive), automatic adjustment of PLC's built-in clock, server/host name specifications	4	1	0.37	---	CJ1W-ETN21	UC1, N, L, CE

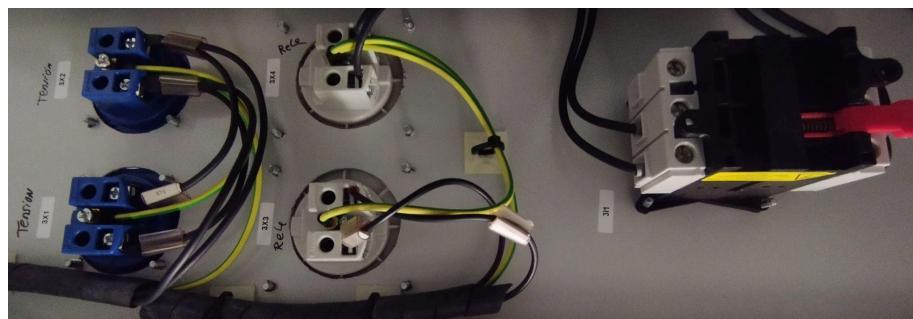
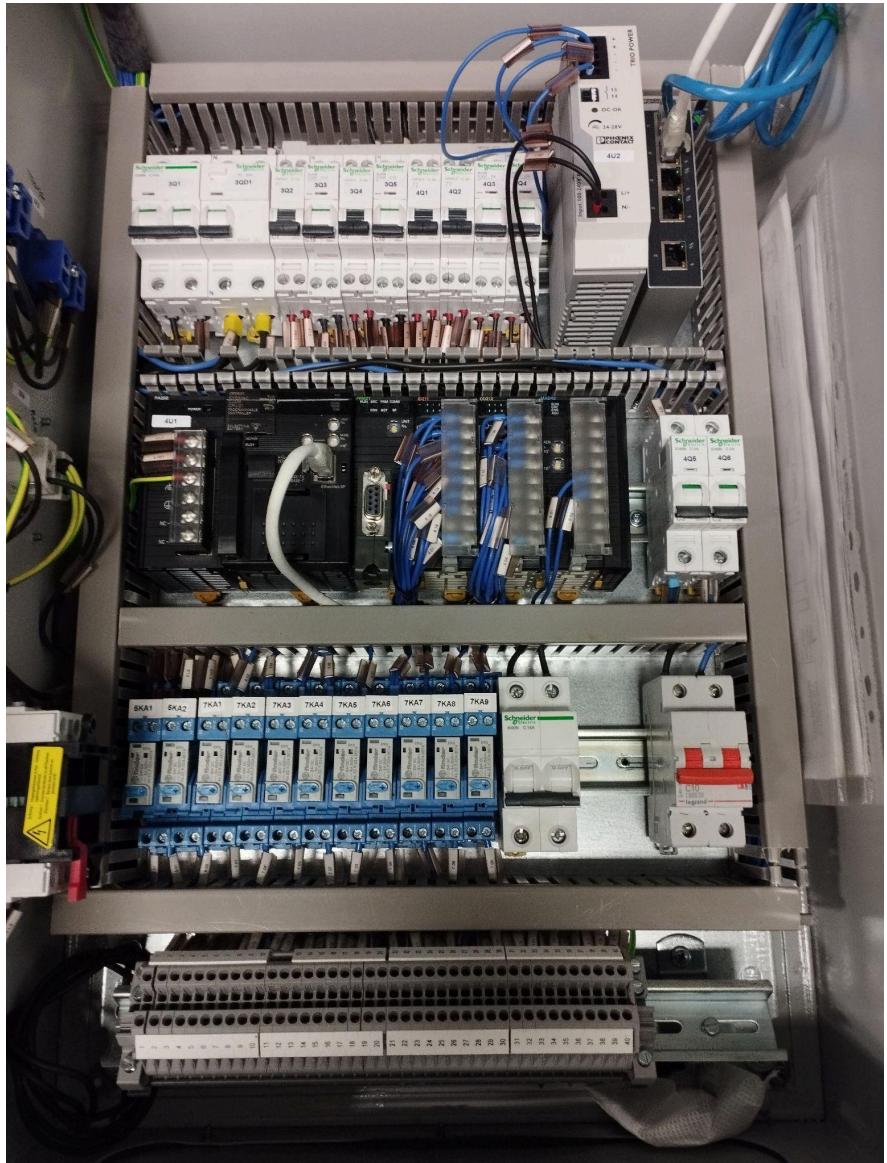
TIPO DE PROGRAMACIÓN

Por ahora entiendo que usaré CX-programmer por ser el que previamente he usado, es conocido por mucha gente y es creado por OMRON que es el fabricante también del PLC que usaremos.

RECONOCIMIENTO DEL CUADRO ELECTRÓNICO

IMAGEN GENERAL





EXPLICACIÓN DE CADA COMPONENTE

Simbología:

H → Dispositivos de señalización

KA → Contactos auxiliares

Q → Interruptores de potencia

U → Convertidores (AC/DC)



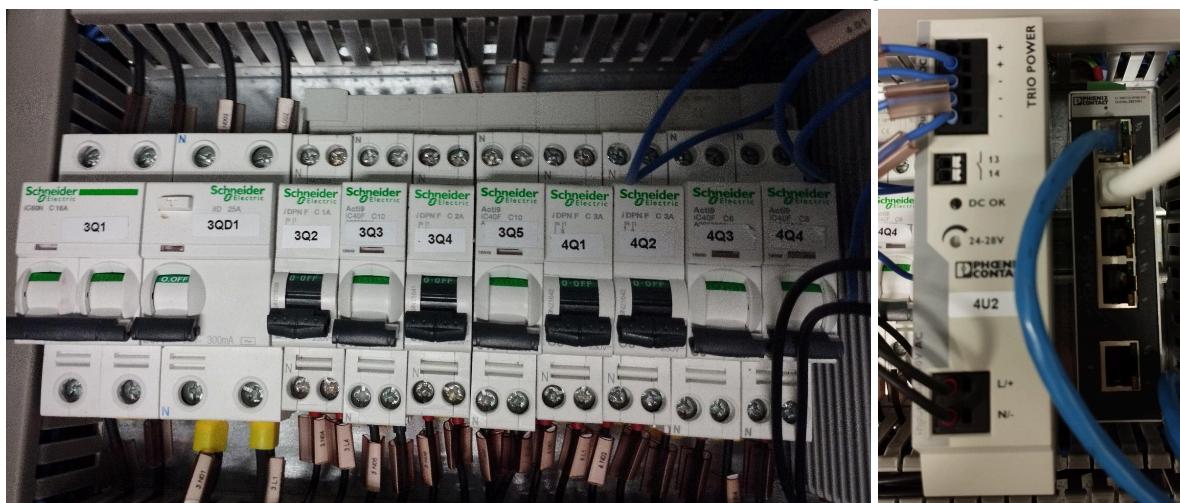
Puente de Graetz o puente rectificador de doble onda con 4 diodos.

X → Bornas de conexión

S → Dispositivos de mando

Explicación:

Interruptores automáticos, fuente de alimentación conmutada y switch:



3Q1, 4Q5, 4Q6: iC60N C 16A: es un interruptor automático en miniatura (MCB) de baja tensión con dos polos. Protege los circuitos contra cortocircuitos y sobrecargas. 16A significa su corriente nominal.

3QD1: iID: Es un interruptor diferencial. Protege contra descargas eléctricas por contacto directo o indirecto y peligros de incendios. Realiza la función de desconexión de circuitos eléctricos en caso de fallo a tierra.

3Q2, 3Q4, 4Q1, 4Q2: iDPN: interruptor automático en miniatura

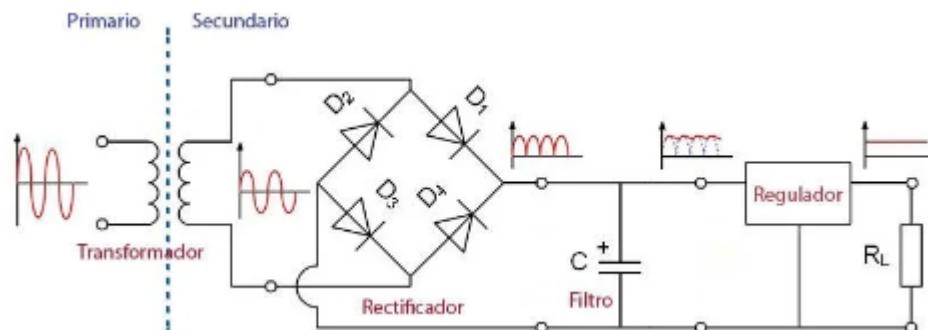
3Q3, 3Q5, 4Q3, 4Q4: iC40F: Este interruptor en miniatura combina funciones de protección de circuitos contra cortocircuito y sobrecarga de corriente, control y aislamiento

4U2: TRIO-PS-2G/1AC/24DC/5: Fuente de alimentación conmutada en primario. Entrada: monofásica, salida: 24 V DC/5 A.



Puente de Graetz o puente rectificador de doble onda con 4 diodos:

Se usa cuando se necesita convertir la corriente alterna (AC) en corriente continua (DC).



En principio, entiendo que será así.

Datos de entrada:

- Funcionamiento en AC:
 - Estructura de la red → en estrella
 - Margen de tensión nominal → 100V AC ... 240V AC
 - Rango de tensión → 100V AC ... 240V AC -15%... +10%
 - Margen de tensión → 85V AC ... 264V AC
 - Tensión de red típica del país → 120V AC / 230V AC
 - Tipo de tensión de la alimentación → AC/DC

- Funcionamiento en DC:
 - Margen de tensión nominal → 110V DC ... 250V DC
 - Rango de tensión → 99V DC ... 275V DC
 - Tensión de funcionamiento → ≥ 88 V DC
 - Tensión de desconexión → < 60 V DC
 - Tipo de tensión de la alimentación → AC/DC

Datos de salida:

- Rendimiento → > 90 % (con 230 V AC y valores nominales)
- Tensión nominal → 24 V DC ±1 %
- Corriente nominal → 5A
- Posibilidad de conexión en paralelo para redundancia y aumento de potencia
- Posibilidad de conexión en serie

Phoenix contact FL Switch SFNB 5TX: Red Interruptor - sin administrar 5 puertos IP20

PLC (OMROM SYSMAC CJ2M CPU31) :



[CJ1W-PA202](#) / [CJ2M CPU31](#) / [CJ1W-PRM21](#) / [CJ1W-ID211](#) / [CJ1W-OD212](#) / [CJ1W-MAD42](#)

- **CJ1W-PA202 (4U1)**: Fuente de alimentación 100 a 240Vca 5Vcc 2,8A Relé
 - **CJ1W-PRM21**: Módulo maestro Profibus DP

Es un sistema maestro. Intercambia datos de E/S e información de comunicaciones y estado con la CPU del PLC. Para configurar el dispositivo CS1W-PRM21 se puede utilizar un puerto serie de la CPU. Pero, como la configuración se realiza a través de comunicación FINS, se puede usar prácticamente cualquier punto de acceso de la red de comunicaciones existente: Controller Link, Ethernet, intercambia datos y comandos con las estaciones PROFIBUS-DP esclavas a través de la red PROFIBUS.

- **CJ1W-ID211:** Basic I/O Units (Input Units)

Sus unidades E/S funcionan únicamente como entrada

- **CJ1W-OD212:** Módulo 16 Salidas PNP Term.

Sus unidades E/S funcionan únicamente como salida

- **CJ1W-MAD42:**

4 inputs (1 to 5 V, 4 to 20 mA, etc.) 2 outputs (1 to 5 V, 4 to 20 mA, etc.)

● Analog I/O Units



46.52.9.024.0074 finder 24V DC 8A 250V

5KA1, 5KA2, 7KA1, ..., 7KA9: Relé electromagnéticos: La punta de plástico (situada justo debajo del botón de prueba) permanece intacta. En este caso, cuando se pulsa el botón de prueba, los contactos funcionan. Cuando se suelta el botón los contactos vuelven a su estado anterior.

La configuración de los contactos es DPDT(Double Pole Double Throw)

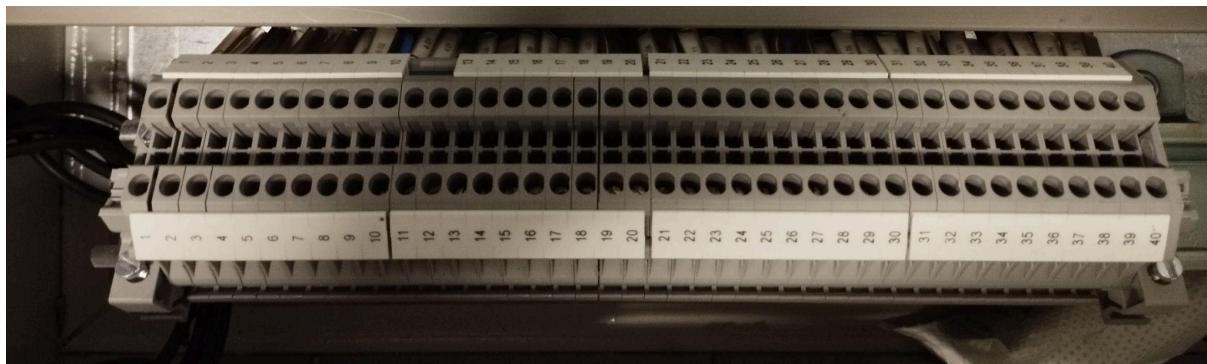
Otro **Schneider**: iK60N: Este interruptor en miniatura combina funciones de protección de circuitos contra cortocircuito y sobrecarga de corriente (sin función por ahora)



Magnetotérmico RX³ - 2P - 230/400 V~ - 10A - curva C

Interruptor magnetotérmico para proteger las instalaciones eléctricas contra sobrecargas y cortocircuitos. También se puede utilizar para controlar y aislar la instalación. (sin función por ahora)

UT 2,5 - Borne de paso: (tipo de conexión: Conexión por tornillo)



<https://www.phoenixcontact.com/es-es/productos/borne-de-paso-ut-25-3044076>

Borna de paso o bloques terminales: es un componente con un marco aislado que tiene el único propósito de fijar dos o más cables juntos. Los bloques de terminales ofrecen flexibilidad. Las modificaciones del cableado son sencillas porque los cables se pueden quitar o añadir rápidamente. El cableado del bloque de terminales es limpio y ordenado, lo que permite una rápida identificación y facilita las modificaciones y la localización de averías.

Método de conexión de cables utilizado: "Atornillado". Son aquellos que utilizan tornillos como método de sujeción de los cables. ¡Atención! Un apriete excesivo de los tornillos puede dañar los cables.

Otros:

Circuitos de Maniobra: Los circuitos de maniobra son los encargados de alimentar a los sensores y captadores, encargados de suministrar información en forma de señales eléctricas o digitales a los sistemas de Lógica Cableada (Contactores, relés, etc.) o Lógica Programada (Relés programables o autómatas programables). Estos circuitos son normalmente alimentados con tensiones de Muy Baja Tensión (MBT) o de Seguridad a 24V.

Dispositivos de señalización



En tensión (3H1) se enciende cuando activas 3Q2.

EV1 entrada(5H1)/salida(5H2) se encienden cuando el relé **7KA2** está activado.

EV2 entrada(5H3)/salida(5H4) se encienden cuando el relé **7KA4** está activado.

EV3 entrada(5H5)/salida(5H6) se encienden cuando el relé **7KA6** está activado.

Conexiones con el conmutador de dos o tres posiciones con el selector



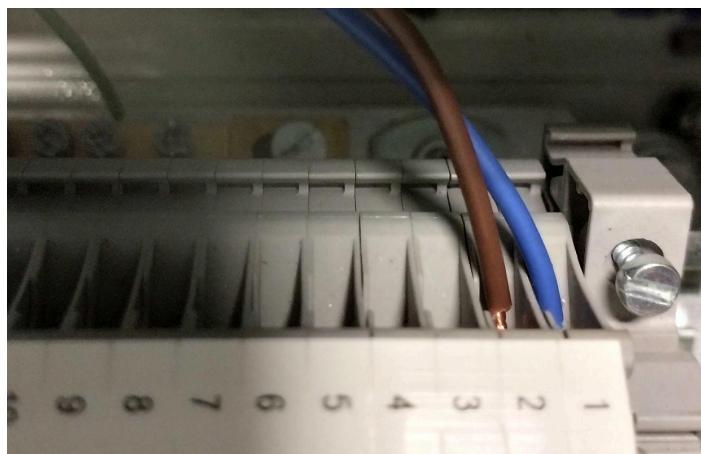
Dispositivos de maniobra

Son dispositivos dirigidos a la implementación del proyecto anterior.

MODO DE OPERACIÓN

1. Alimentación del panel electrónico.

Como se muestra en la imagen, el neutro (azul) al borne 1, la línea (marrón) al borne 2 y la tierra (verde y amarillo) a la conjunción de tierra.



2. Interruptores a accionar

3Q3 → ENCHUFE AZUL (3X1) O (3X2)

5KA1 y 3Q4 → ENCHUFE GRIS (3x3) 5KA2 y 3Q5 → (3X4)

3Q2 → Para encender luz de tensión (3H1) muestra presencia de la tensión

4U1 + 4Q1 → Alimentación de la CPU

4U2 + 4Q2 + 4Q5 → Alimentación maniobra

4U2 + 4Q2 + 4Q6 → Alimentación módulos PLC y switch

El manual del cuadro electrónico muestra de una manera visual las conexiones entre los componentes.

ESTABLECIMIENTO DE CONECTIVIDAD CON PLCS VÍA RS232 - USB

RS232 INTRODUCCIÓN

RS-232 (*Recommended Standard 232*), también conocido como EIA/TIA RS-232C, es una interfaz que designa una norma para el intercambio de datos binarios serie entre un DTE (*Data Terminal Equipment*, "Equipo Terminal de Datos"), como por ejemplo una computadora, y un DCE (*Data Communication Equipment*, "Equipo de Comunicación de Datos"), por ejemplo un módem.

La interfaz RS-232 está diseñada para imprimir documentos para distancias cortas, de hasta 15 metros según la norma, y para velocidades de comunicación bajas, de no más de 20 kbps. A pesar de esto, muchas veces se utiliza a mayores velocidades con un resultado aceptable. La interfaz puede trabajar en comunicación asíncrona o síncrona y tipos de canal *simplex*, *half duplex* o *full duplex*. En un canal *simplex* los datos siempre viajarán en una dirección, por ejemplo desde DCE a DTE. En un canal *half duplex*, los datos pueden viajar en una u otra dirección, pero solo durante un determinado periodo de tiempo; luego la línea debe ser comutada antes que los datos puedan viajar en la otra dirección. En un canal *full duplex*, los datos pueden viajar en ambos sentidos simultáneamente. Las líneas de *handshaking* de la RS-232 se usan para resolver los problemas asociados con este modo de operación, tal como en qué dirección los datos deben viajar en un instante determinado.

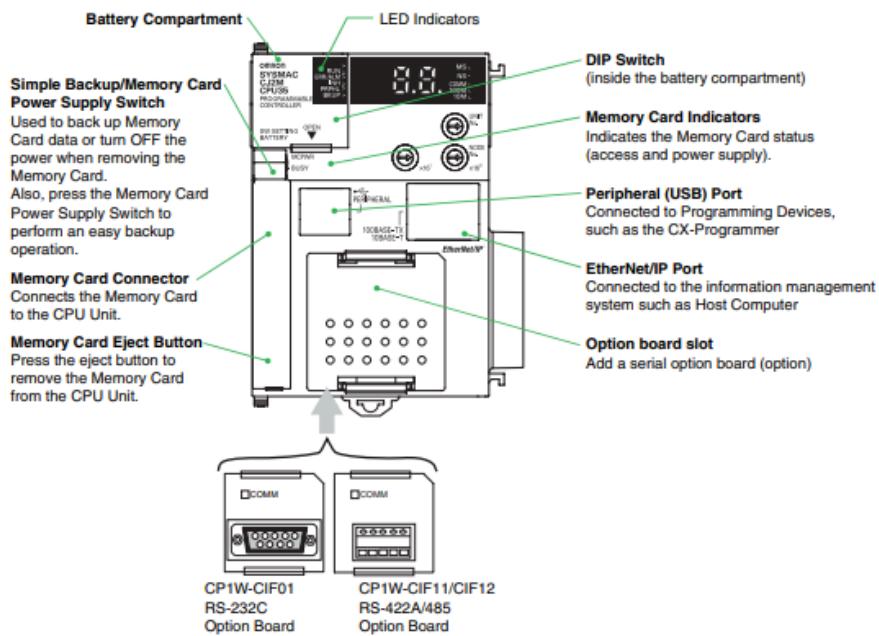
Si un dispositivo de los que están conectados a una interfaz RS-232 procesa los datos a una velocidad menor de la que los recibe deben de conectarse las líneas *handshaking* que permiten realizar un control de flujo tal que al dispositivo más lento le de tiempo de procesar la información. Las líneas de *hand shaking* que permiten hacer este control de flujo son las líneas RTS y CTS.

CONEXIÓN CON PLC CJ2M-CPU31

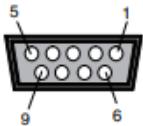
El CJ2M-CPU31 no tiene puerto RS-232 incorporado, pero tiene una ranura opcional para la comunicación en serie. La referencia para la tarjeta opcional RS-232C es CP1W-CIF01:

CJ2M-CPU3□ (CJ2M with Built-in EtherNet/IP)

A CJ2M-CPU3□ provides two communications ports for external interfaces: a peripheral (USB) port and an EtherNet/IP port. Serial ports can be added by mounting a Serial Communications Option Board (sold separately) in an option slot.



●RS-232C Connector



Pin No.	Signal	Name	Direction
1	FG	Protection earth	-
2	SD (TXD)	Send data	Output
3	RD (RXD)	Receive data	Input
4	RS (RTS)	Request to send	Output
5	CS (CTS)	Clear to send	Input
6	5 V	Power supply	-
7	DR (DSR)	Data set ready	Input
8	ER (DTR)	Data terminal ready	Output
9	SG (0 V)	Signal ground	-
Connector hood	FG	Protection earth	-

Note: Do not use the 5-V power from pin 6 of the RS-232C port for anything but CJ1W-CIF11 RS-422A Conversion Adapter, NT-AL001 RS-232C/RS-422A Conversion Adapter and NV3W-M_20L Programmable Terminal. The external device or the CPU Unit may be damaged.

De todas formas, en nuestro caso tenemos un módulo maestro con puerto RS-232 incorporado. Tenemos un problema con este módulo. Aparece el LED ERH encendido siendo esta una luz roja. Creemos que ese puede ser el problema por el cual no conseguimos la conectividad del PLC con el PC vía RS-232-USB. Hemos probado con varios cables diferentes, pero ninguno ha dado resultado desde el módulo maestro.

Al final, hemos conseguido la conectividad del PLC con el PC vía USB tipo B 2.0 - USB tipo A conectando directamente desde la CPU del PLC.

Conclusiones:

- Creemos que el módulo maestro está averiado.
- Si hay conectividad desde la CPU del CJ2M-CPU31, probablemente añadiendo directamente el CP1W-CIF01 se consiga conectar también.

ESTUDIO DEL FUNCIONAMIENTO DE LA WWTP EN SIMULINK

WWTP INTRODUCCIÓN

Una WWTP es una instalación en la que una combinación de varias unidades de tratamiento (físico, químico y biológico) trabajan juntas para reducir los contaminantes de las aguas residuales a niveles que garanticen la preservación de las masas de agua receptoras. La unidad más crítica de una WWTP es el tratamiento secundario, en el que se eliminan biológicamente la materia orgánica, el nitrógeno y el fósforo.

A) Level 0: Wastewater Treatment Plant (WWTP)

La figura 1 alberga en el nivel 0 un simulador del tratamiento secundario, como parte del proceso físico. Una configuración típica de planta para las unidades de tratamiento secundario es el proceso de pre-desnitrificación (proceso DN) (Figura 2).

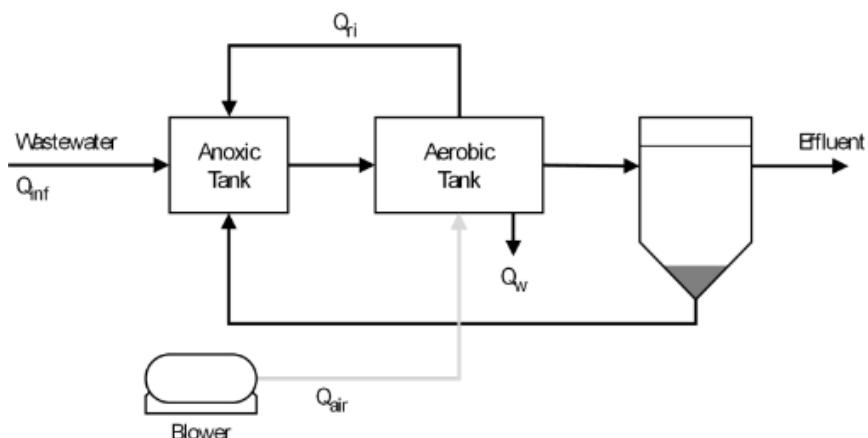


Figure 2. Layout of the DN process.

Variable	Símbolo	Variables	
		I/O	Valor
Caudal de Purga	Qw	Input	95145
Nitrato	NO	Input	1.3
Amonio	NH4	Input	6.85
Oxígeno disuelto	O	Input	0.0254
Sólido Suspenido	SOL	Input	3138.87
Aireación	AIR	Input	3
Temperatura	T	Input	15
Caudal de Aire	Qair	Output	30 - 200
Caudal de recirculación interna	Qri	Output	57087

ARQUITECTURA

Trabajamos en un WWTP, que en vez de ser un proceso físico real, se representa mediante un modelo creado en Matlab/Simulink. Simulink se conecta mediante comunicación UDP con el OpenPLC o con un PLC real que todavía no sabemos cómo lo implementaremos en el sistema. Por otro lado, Modbus TCP/IP es la comunicación de la conexión realizada entre el OpenPLC y el SCADA-LTS. Matlab/Simulink, OpenPLC y Scada-LTS están siendo soportados por Docker.

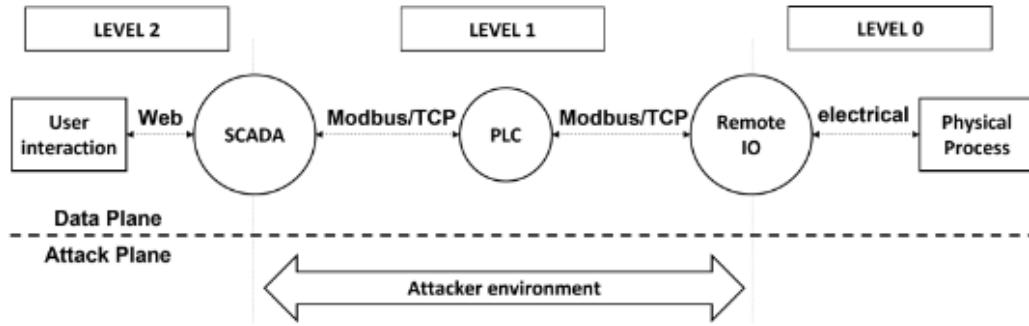


Figure 1. Overall Virtual Testbed architecture and attacker environment.

B) Level 1: OpenPLC

C) Level 2: Scada-LTS

Los SCADAS son sistemas de elementos de software y hardware que permiten a las organizaciones industriales: (1) controlar procesos industriales localmente o en ubicaciones remotas; (2) monitorear, recolectar y procesar datos en tiempo real; (3) interactuar con dispositivos tales como sensores, válvulas, bombas, motores y más a través de software HMI. SCADA-LTS proporciona Fuentes de Datos y Puntos de Datos como forma de organizar los datos procedentes de los dispositivos remotos

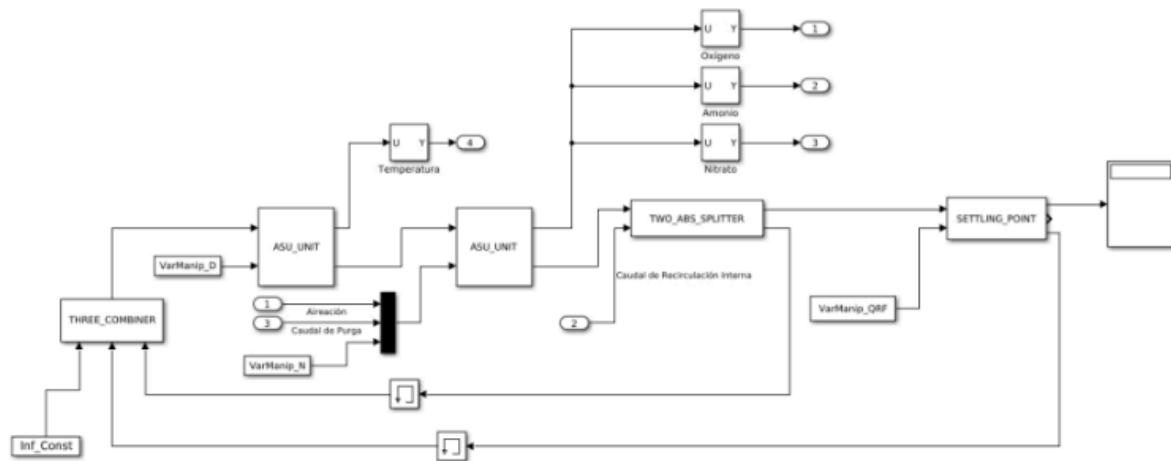


Figure 3. WWTP modeled in Simulink.

D) *Comunicaciones: Modbus/TCP*

Modbus permite el intercambio de datos entre diferentes partes del proceso industrial. A este respecto, la Figura 1 muestra cómo Modbus soporta todo el flujo de comunicación.

E) *Atacante. ModTester*

El plano de ataque soporta todas las pruebas de seguridad a realizar sobre el protocolo de comunicación Modbus TCP. Modtester es un marco unificado de pentesting para el protocolo Modbus.

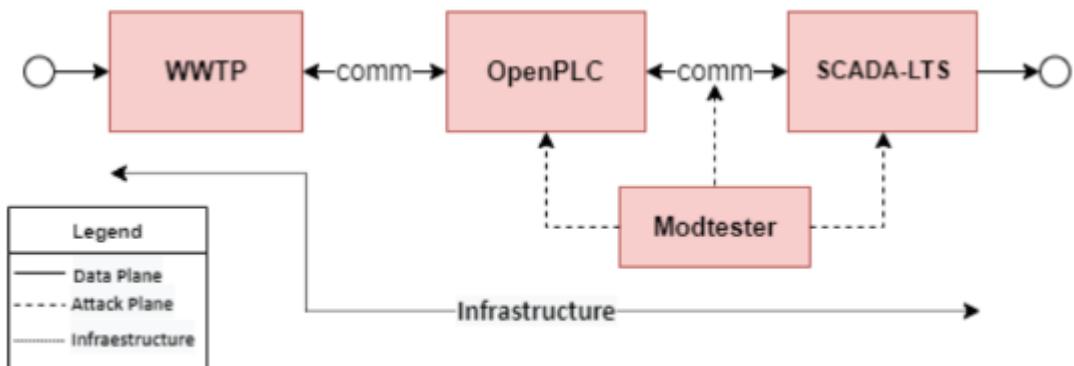


Figure 6. Data and attack plane and infrastructure.

DISEÑO DEL BANCO DE PRUEBAS Y PRUEBAS DE ATAQUE

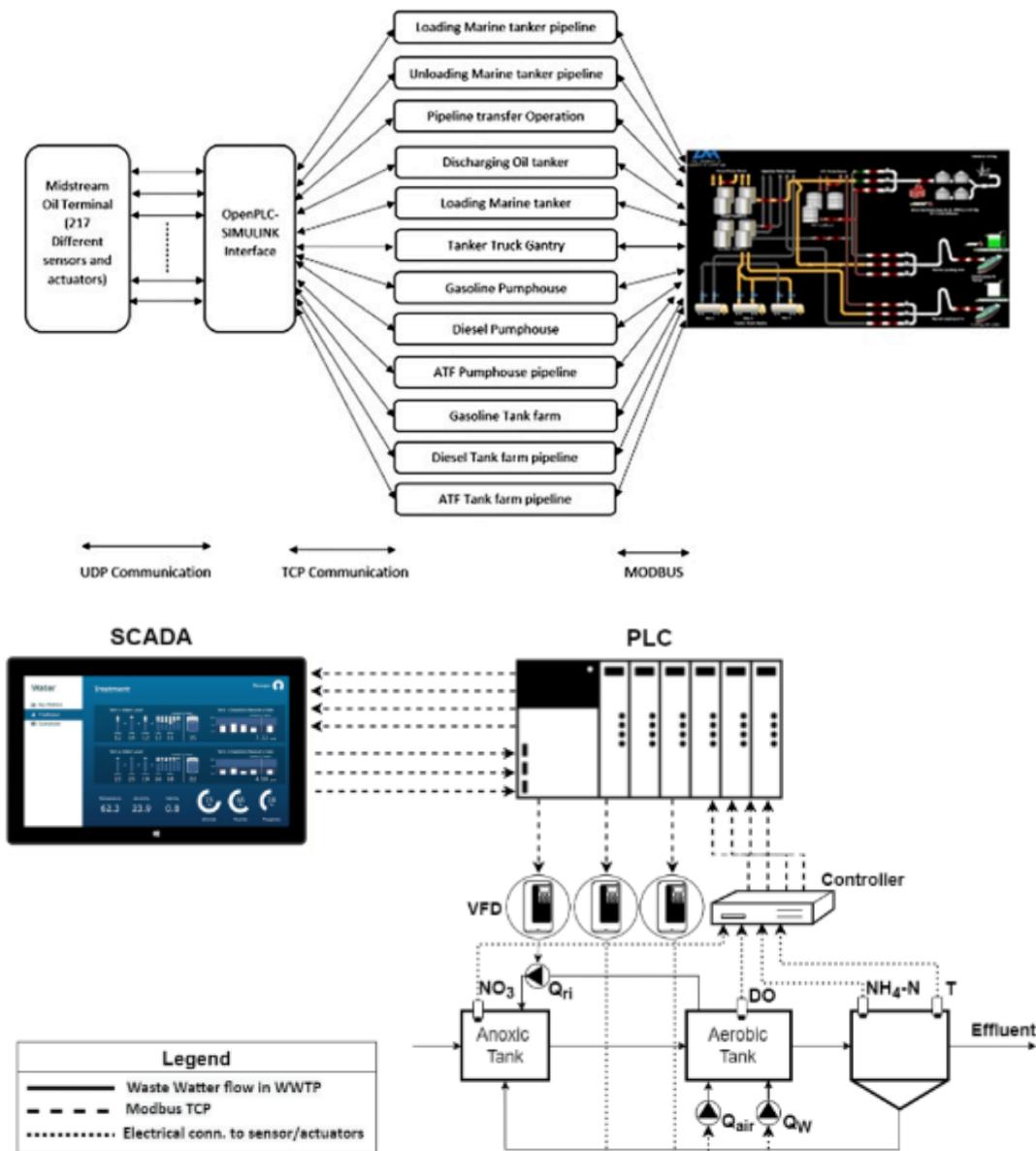


Figure 7. Interactions logical diagram in the *data plane*.

- La configuración adoptada en el PLC es de lazo abierto, compuesta por un conjunto de sensores y actuadores de la WWTP totalmente mapeados en el SCADA. Las señales salen del SCADA vía Modbus TCP, pasan por el PLC y llegan a los Variadores de Frecuencia (VFDs), también vía Modbus TCP.
- ModTester integra un conjunto de herramientas definidas de la siguiente manera:
- Nmap: herramienta para conocer la topología de la red.

- Hping: una herramienta de línea de comandos que nos permite crear y analizar paquetes TCP/IP (pruebas de cortafuegos, escaneo de puertos, escaneo de red. Puede provocar un ataque SYN Flood a través de DoS. Técnicas: Puerto Comúnmente Utilizado y Mensaje de Comando No Autorizado. (Commonly Used Port and Unauthorized Command Message.)
- Modpoll: herramienta de comunicación con dispositivos Modbus. Técnicas: Mensaje de Comando No Autorizado y Protocolo Estándar de Capa de Aplicación. (Unauthorized Command Message and Standard Application Layer Protocol)

Esta tabla muestra los ataques que define ModTester:

- Reconnaissance: Nmap
- Command Injection: Modpoll
- DoS: Modpoll and Hping

MODTESTER ATTACKS DEFINITION

Attack	Code	Purpose
Recce.	REC-01	SCADA network IP range quick scan.
	REC-02	SCADA network single IP intense scan; device identification scans.
	REC-03	Modbus address and function code scan.
	REC-04	PLC memory full scan.
Command Injection	CI-01	The payload size is right while the payload it is filled with all '0' or all '1' or all 'F' or random bits.
	CI-02	Incorrect payload size.
	CI-03	Out of bound value input (The pump speed is 200/-200).
	CI-04	Regular pump speed without stop.
	CI-05	Slow pump speed without stop.
	CI-05	Change the pump speed repetitively which keeps the pump working.
DoS	DOS-01	Massive Modbus scans to make PLC enter denial-of-service.
	DOS-02	Massive Modbus instruction with incorrect CRC to make PLC enter DoS.
	DOS-03	Classic DoS Attack over PLC.

Matlab está siendo soportado recientemente por Docker. Dado que WWTP se ha creado íntegramente sobre Matlab/Simulink, la virtualización de WWTP ha permitido su integración en el plano de datos.

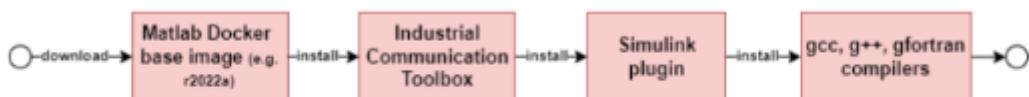


Figure 8. Virtualized WWTP set-up.

El conector Modbus TCP permite la comunicación entre la WWTP y el PLC. La parte izquierda de la Figura 9 muestra las variables de entrada de la WWTP, correspondientes a los actuadores y la parte derecha muestra las variables de salida de la WWTP, correspondientes a los sensores.

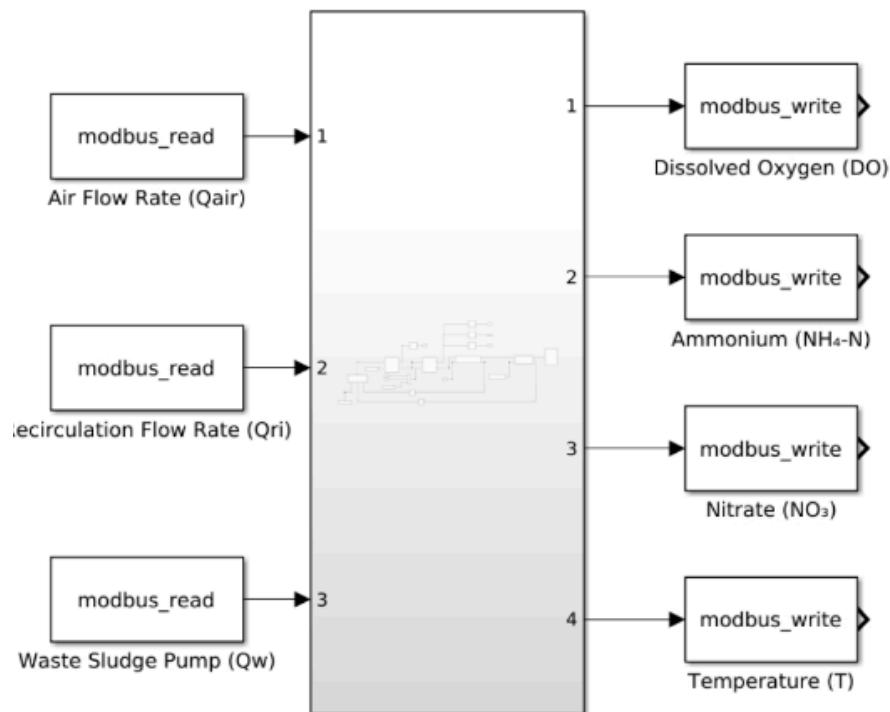
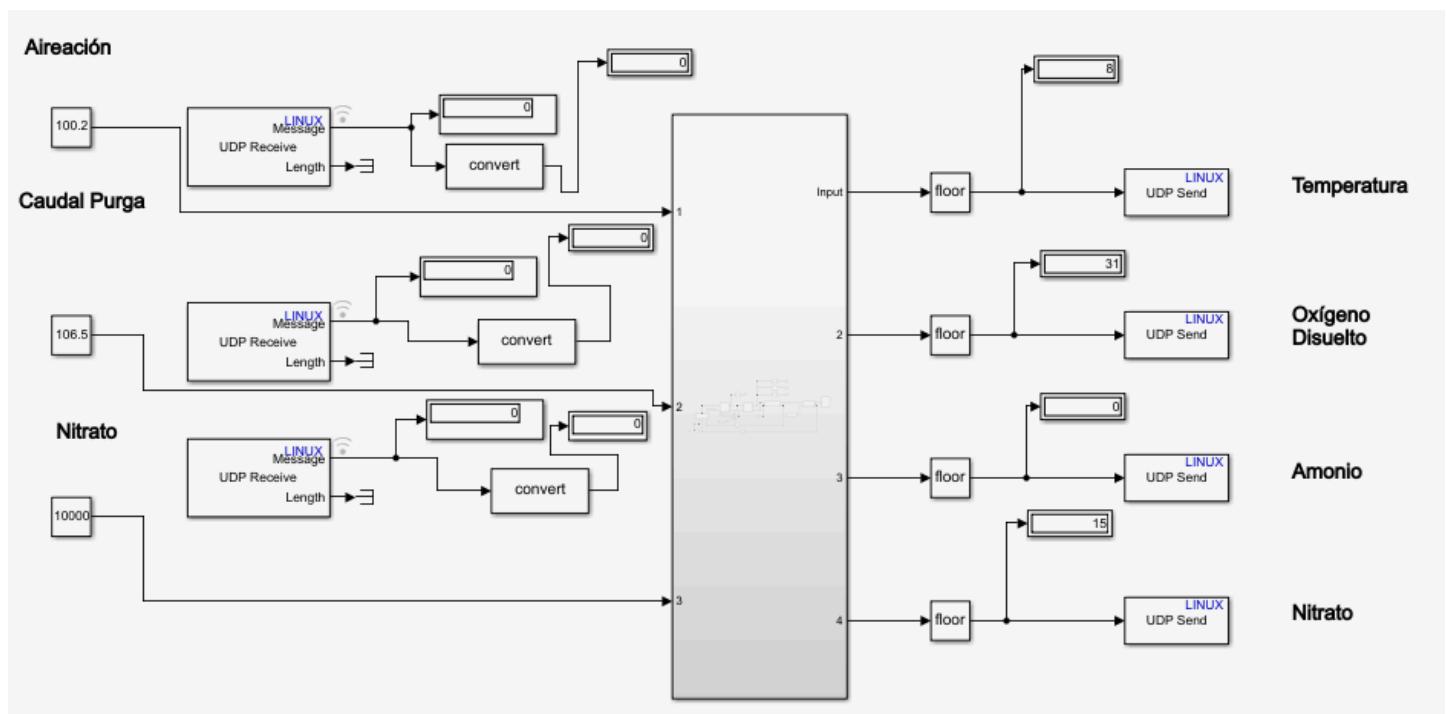


Figure 9. WWTP Modbus connector.

ESTUDIO DE LA TECNOLOGÍA DOCKER

DOCKER BEGINNER TUTORIAL

- Setup:
- Running your first container:
 - **docker pull *imagen***: (\$ docker pull alpine) obtiene la imagen, en este caso alpine, del registro Docker y la guarda en nuestro sistema
 - **docker images**: para ver una lista de todas las imágenes en tu sistema.

```
oihan@oihan:~$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
alpine          latest   f8c20f8bbcb6  12 days ago  7.38MB
hello-world     latest   d2c94e258dcb  7 months ago  13.3kB
```

- **docker run *imagen* ls -l**:

```
oihan@oihan:~$ docker run alpine ls -l
total 56
drwxr-xr-x  2 root    root        4096 Dec  7 09:43 bin
drwxr-xr-x  5 root    root        340 Dec 20 10:32 dev
drwxr-xr-x  1 root    root        4096 Dec 20 10:32 etc
drwxr-xr-x  2 root    root        4096 Dec  7 09:43 home
drwxr-xr-x  7 root    root        4096 Dec  7 09:43 lib
drwxr-xr-x  5 root    root        4096 Dec  7 09:43 media
drwxr-xr-x  2 root    root        4096 Dec  7 09:43 mnt
drwxr-xr-x  2 root    root        4096 Dec  7 09:43 opt
dr-xr-xr-x  180 root   root        0 Dec 20 10:32 proc
drwx-----  2 root    root        4096 Dec  7 09:43 root
drwxr-xr-x  2 root    root        4096 Dec  7 09:43 run
drwxr-xr-x  2 root    root        4096 Dec  7 09:43 sbin
drwxr-xr-x  2 root    root        4096 Dec  7 09:43 srv
dr-xr-xr-x  13 root   root        0 Dec 20 10:32 sys
drwxrwxrwt  2 root    root        4096 Dec  7 09:43 tmp
drwxr-xr-x  7 root    root        4096 Dec  7 09:43 usr
drwxr-xr-x  12 root   root        4096 Dec  7 09:43 var
```

- run: ejecuta el comando
- ls -l: es el comando. ls → listar; -l → long format

- **docker run *imagen* echo "hello from alpine"**:

```
oihan@oihan:~$ docker run alpine echo "hello from alpine"
hello from alpine
```

- run: ejecuta el comando
- echo: repite lo pedido

- **docker run *imagen* /bin/sh**:

```
oihan@oihan:~$ docker run alpine /bin/sh
oihan@oihan:~$ docker run -it alpine /bin/sh
/ # ls -l
total 56
drwxr-xr-x  2 root    root        4096 Dec  7 09:43 bin
drwxr-xr-x  5 root    root        360 Dec 20 10:23 dev
drwxr-xr-x  1 root    root        4096 Dec 20 10:23 etc
drwxr-xr-x  2 root    root        4096 Dec  7 09:43 home
drwxr-xr-x  7 root    root        4096 Dec  7 09:43 lib
drwxr-xr-x  5 root    root        4096 Dec  7 09:43 media
drwxr-xr-x  2 root    root        4096 Dec  7 09:43 mnt
drwxr-xr-x  2 root    root        4096 Dec  7 09:43 opt
dr-xr-xr-x  175 root   root        0 Dec 20 10:23 proc
drwx-----  1 root    root        4096 Dec 20 10:24 root
drwxr-xr-x  2 root    root        4096 Dec  7 09:43 run
drwxr-xr-x  2 root    root        4096 Dec  7 09:43 sbin
drwxr-xr-x  2 root    root        4096 Dec  7 09:43 srv
dr-xr-xr-x  13 root   root        0 Dec 20 10:23 sys
drwxrwxrwt  2 root    root        4096 Dec  7 09:43 tmp
drwxr-xr-x  7 root    root        4096 Dec  7 09:43 usr
drwxr-xr-x  12 root   root        4096 Dec  7 09:43 var
/ # uname -a
Linux 302bc20458e6 5.15.0-91-generic #101-Ubuntu SMP Tue Nov 14 13:30:08 UTC 2023 x86_64 Linux
/ # exit
```

estos shells interactivos se saldrán (exit) después de ejecutar cualquier comando de script, a menos que se ejecuten en un terminal interactivo

→ **docker run -it imagen /bin/sh:**

- -it: interactive terminal
- **/bin/sh:** Lo más común sería `/bin/bash` que se encuentra en la mayoría de imágenes, pero justamente Alpine no viene con ella instalada.

El directorio `/bin` es una ubicación estándar del sistema de archivos que contiene archivos binarios ejecutables, y `/bin/bash` se refiere al intérprete de comandos Bash. Cuando ves `/bin/bash` en un contexto de Docker, generalmente significa que estás especificando un contenedor para ejecutar un shell Bash como su proceso principal.

- directorio `ls -l` y `uname -a`
- exit: para salir del it

○ **docker ps -a:**

```
oihan@oihan:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1b0b880330e6	alpine	"ls -l"	26 minutes ago	Exited (0) 26 minutes ago		MES
302bc20458e6	alpine	"/bin/sh"	34 minutes ago	Exited (0) 32 minutes ago		actical_mcclintock
dest_lovelace	alpine	"/bin/sh"	34 minutes ago	Exited (0) 34 minutes ago		ving_kapitsa
66cec358db5b	alpine	"/bin/sh"	34 minutes ago	Exited (0) 34 minutes ago		hcfc7042aeeff
	alpine	"-it /bin/sh"	35 minutes ago	Created		

- **ps:** muestra todos los contenedores que se están ejecutando actualmente
- **-a:** todos

● Webapps with docker:

- **docker run -d dockersamples/static-site:** para ejecutar la imagen en un contenedor
- **docker ps:** para ver los contenedores en ejecución
 - container ID: para ver el ID del contenedor

```
oihan@oihan:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
712051f23aae        dockersamples/s
```

- docker **stop Container ID:** para detenerlo
- docker **rm Container ID:** para removerlo

- **docker run --name static-site -e AUTHOR="Your Name" -d -P dockersamples/static-site**

para iniciar un contenedor en modo independiente

- **-d:** para crear un contenedor con el proceso separado en nuestra terminal
- **-P:** publicará todos los puertos de contenedores expuestos en puertos aleatorios en el host Docker.
- **-e:** para pasar las variables del entorno al contenedor.
- **--name:** permite especificar un nombre al contenedor.

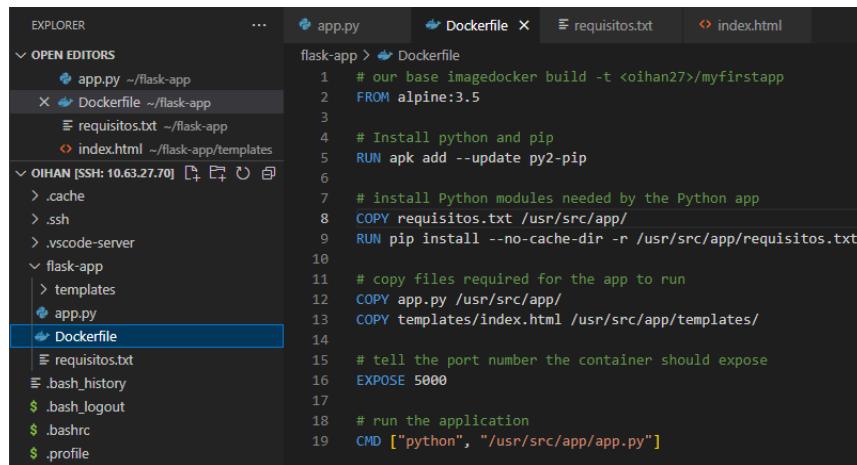
- **AUTHOR:** es el nombre de la variable del entorno y **Your Name** es el valor que puede pasar
- **docker port static site:** para ver los puertos. con docker ps, también se ven.
- Imágenes:
 - Imágenes base: son imágenes que no tienen imágenes principales, generalmente imágenes con un sistema operativo como ubuntu, alpine o debian.
 - Imágenes secundarias: son imágenes que se basan en imágenes base y agrega funcionalidad adicional.
 - Imágenes oficiales: son imágenes aprobadas por docker.
https://hub.docker.com/search?q=&type=image&image_filter=official
 - Imágenes de usuario: son imágenes creadas y compartidas por usuarios
- Crear mi propia imagen:
 - **mkdir “flask-app”:** para crear un directorio llamado “flask-app”
 - **ls:** para asegurarte de que se ha creado

```
oihan@oihan:~$ mkdir flask-app
oihan@oihan:~$ ls
flask-app
```

 - **rm -r nombre del directorio:** para remover el directorio y su contenido de forma permanente.
 - **cd “flask-app”:** para entrar en el directorio.
 - **..** para volver al directorio anterior
 - **pwd:** para verificar en qué directorio se encuentra.

```
oihan@oihan:~$ cd flask-app
oihan@oihan:~/flask-app$ pwd
/home/oihan/flask-app
```

 - **crear archivos en visual studio**
 - **crear Dockerfile**



The screenshot shows a terminal window with the following content:

```

EXPLORER          ...   app.py    Dockerfile X  requisitos.txt  index.html
OPEN EDITORS
  app.py ~/flask-app
  Dockerfile ~/flask-app
  requisitos.txt ~/flask-app
  index.html ~/flask-app/templates
OIHAN [SSH: 10.63.27.70]  ⌂ ⌂ ⌂
  .cache
  .ssh
  .vscode-server
  flask-app
    templates
    app.py
  Dockerfile
  requisitos.txt
  .bash_history
  .bash_logout
  .bashrc
  .profile
Dockerfile
  # our base image
  FROM alpine:3.5
  # Install python and pip
  RUN apk add --update py2-pip
  # install Python modules needed by the Python app
  COPY requisitos.txt /usr/src/app/
  RUN pip install --no-cache-dir -r /usr/src/app/requisitos.txt
  # copy files required for the app to run
  COPY app.py /usr/src/app/
  COPY templates/index.html /usr/src/app/templates/
  # tell the port number the container should expose
  EXPOSE 5000
  # run the application
  CMD ["python", "/usr/src/app/app.py"]

```

- **construir la imagen**
docker build -t oihan27/myfirstapp flask-app/ para construir la imagen desde cualquier lado
~/flask-app\$ docker build -t oihan27/myfirstapp . para construir la imagen desde el mismo repositorio

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
oihan27/myfirstapp	latest	f4c169c91a9f	43 seconds ago	56.8MB
alpine	latest	f8c20f8bbcb6	12 days ago	7.38MB
hello-world	latest	d2c94e258dcb	7 months ago	13.3kB
alpine	3.5	f80194ae2e0c	4 years ago	4MB
ubuntu	12.04	5b117edd0b76	6 years ago	104MB
dockersamples/static-site	latest	f589ccde7957	7 years ago	191MB

- ejecutar la imagen

docker run -p 8888:5000 --name myfirstapp oihan27/myfirstapp

se necesita crear un puente para poder entrar al localhost de la máquina virtual

Para ello escribir en la máquina virtual:

ssh -N -D 995 oihan@10.63.27.70

para abrir el localhost escribir en la barra de búsqueda de firefox:

<http://10.63.27.70:8888/>

- empujar la imagen

docker login → Username: oihan27 Password: la primera

docker push oihan27/myfirstapp

para eliminar

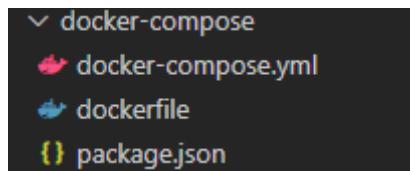
docker stop myfirstapp

docker rm myfirstapp

DOCKER COMPOSE TUTORIAL

Docker Compose es una herramienta desarrollada para ayudar a definir y compartir aplicaciones de múltiples contenedores. Con Compose, podemos crear un archivo YAML para definir los servicios y, con un solo comando, podemos activar o desactivar todo. La gran ventaja de usar Compose es que puede definir la pila de su aplicación en un archivo, mantenerla en la raíz del repositorio de su proyecto (ahora tiene control de versión) y permitir fácilmente que otra persona contribuya a su proyecto.

- Crear un nuevo directorio junto con los siguientes files:



El archivo package.json contiene información importante sobre la aplicación, como su nombre, versión, dependencias, scripts de ejecución, entre otros.

- añadir dentro del docker compose lo siguiente:

```

docker-compose > docker-compose.yml
  1 services:
  2   app:
  3     image: node:18-alpine
  4     command: sh -c "yarn install && yarn run dev"
  5     ports:
  6       | - 3000:3000
  7     working_dir: /app
  8     volumes:
  9       | - ./app
 10      environment:
 11        MYSQL_HOST: mysql
 12        MYSQL_USER: root
 13        MYSQL_PASSWORD: secret
 14        MYSQL_DB: todos
 15
 16 mysql:
 17   image: mysql:8.0
 18   volumes:
 19     | - todo-mysql-data:/var/lib/mysql
 20   environment:
 21     MYSQL_ROOT_PASSWORD: secret
 22     MYSQL_DATABASE: todos
 23
 24 volumes:
 25   todo-mysql-data:

```

- **docker-compose up -d:**

- docker-compose up para iniciar una pila de aplicaciones
- -d para ejecutar todo en segundo plano
- podemos comprobar que se ha creado con docker ps

```

● oihan@oihan:~/docker-compose$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
8b460237476a mysql:8.0 "docker-entrypoint.s..." 3 minutes ago Up 3 minutes 3306/tcp, 33060/tcp docker-compose_mysql_1
● oihan@oihan:~/docker-compose$ docker-compose ps
          Name           Command    State    Ports
----- 
docker-compose_app_1   docker-entrypoint.sh sh -c ...   Exit 1
docker-compose_mysql_1 docker-entrypoint.sh mysqld   Up      3306/tcp, 33060/tcp

```

- **docker-compose down:** para derribarlo todo

- si se quiere eliminar los volúmenes añadir --volumes

CREACIÓN DOCKERFILE

FROM - inicia el Dockerfile. Es un requisito que el Dockerfile debe comenzar con el comando FROM . Las imágenes se crean en capas, lo que significa que puedes usar otra imagen como imagen base para la tuya. El comando FROM define su capa base. Como argumentos toma el nombre de la imagen.

RUN - se utiliza para construir la imagen que estás creando. Para cada comando, Docker ejecutará el comando y luego creará una nueva capa de la imagen. De esta manera puedes revertir tu imagen a estados anteriores fácilmente.

COPY - copia archivos locales en el contenedor

CMD - define los comandos que se ejecutarán en la imagen al inicio. A diferencia de RUN, esto no crea una nueva capa para la imagen, sino que simplemente ejecuta el comando. Solo puede haber uno CMD por cada Dockerfile/Imagen. Si necesita ejecutar varios comandos, la mejor manera de hacerlo es ejecutar CMD un script.

EXPOSE - Crea una pista para los usuarios de una imagen sobre qué puertos brindan servicios.

CONSTRUIR, EJECUTAR Y EMPUJAR LA IMAGEN

```

docker build -t oihan27/myfirstapp flask-app/
docker run -p 8888:5000 --name myfirstapp oihan27/myfirstapp
docker login → docker push oihan27/myfirstapp

```

TERMINOLOGÍA

images - El sistema de archivos y la configuración de nuestra aplicación que se utilizan para crear contenedores

containers - Ejecución de instancias de imágenes Docker - los contenedores ejecutan las aplicaciones reales. Un contenedor incluye una aplicación y todas sus dependencias. Comparte el kernel con otros contenedores, y se ejecuta como un proceso aislado en el espacio de usuario en el sistema operativo anfitrión. Crea un contenedor usando docker run, lo cual hiciste usando la imagen alpine que descargaste. Se puede ver una lista de los contenedores en ejecución utilizando el comando docker ps.

Docker daemon - El servicio en segundo plano que se ejecuta en el host y que gestiona la creación, ejecución y distribución de contenedores Docker.

Docker client - La herramienta de línea de comandos que permite al usuario interactuar con el Docker daemon.

Docker store - Un registro de imágenes Docker, donde puedes encontrar contenedores de confianza y listos para la empresa, plugins y ediciones Docker.

Un **Dockerfile** es un archivo de texto que contiene una lista de comandos que el Docker daemon llama mientras crea una imagen. Contiene toda la información que Docker necesita saber para ejecutar la aplicación.

DIRECTORIO

Para más ayuda –help

run:

```
oihan@oihan:~$ docker run --help
```

Comando **images**:

```
Commands:
  build      Build an image from a Dockerfile
  history    Show the history of an image
  import     Import the contents from a tarball to create a filesystem image
  inspect    Display detailed information on one or more images
  load       Load an image from a tar archive or STDIN
  ls         List images
  prune     Remove unused images
  pull       Download an image from a registry
  push       Upload an image to a registry
  rm        Remove one or more images
  save      Save one or more images to a tar archive (streamed to STDOUT by default)
  tag       Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
```

Directorio **ls**: enlista con los parámetros seleccionados

```
List directory contents

  -1      One column output
  -a      Include names starting with .
  -A      Like -a, but exclude . and ..
  -x      List by lines
  -d      List directory names, not contents
  -L      Follow symlinks
  -H      Follow symlinks on command line
  -R      Recurse
  -p      Append / to directory names
  -F      Append indicator (one of */=@|) to names
  -l      Long format
  -i      List inode numbers
  -n      List numeric UIDs and GIDs instead of names
  -s      List allocated blocks
  -lc     List ctime
  -lu     List atime
  --full-time   List full date/time
  -h      Human readable sizes (1K 243M 2G)
  --group-directories-first
  -S      Sort by size
  -X      Sort by extension
  -v      Sort by version
  -t      Sort by mtime
  -tc     Sort by ctime
  -tu     Sort by atime
  -r      Reverse sort order
  -w N    Format N columns wide
  --color[={always,never,auto}]
```

uname es un comando utilizado en sistemas operativos tipo Unix (como Linux) para obtener información sobre el sistema. Su nombre proviene de "UNIX Name" o "Kernel Name":

```
oihan@oihan:~$ uname --help
Usage: uname [OPTION]...
Print certain system information. With no OPTION, same as -s.

  -a, --all           print all information, in the following order,
                     except omit -p and -i if unknown:
  -s, --kernel-name  print the kernel name
  -n, --nodename     print the network node hostname
  -r, --kernel-release  print the kernel release
  -v, --kernel-version  print the kernel version
  -m, --machine      print the machine hardware name
  -p, --processor    print the processor type (non-portable)
  -i, --hardware-platform  print the hardware platform (non-portable)
  -o, --operating-system  print the operating system
  --help            display this help and exit
  --version          output version information and exit
```

ESTUDIO DEL FUNCIONAMIENTO DEL PLC VIRTUAL

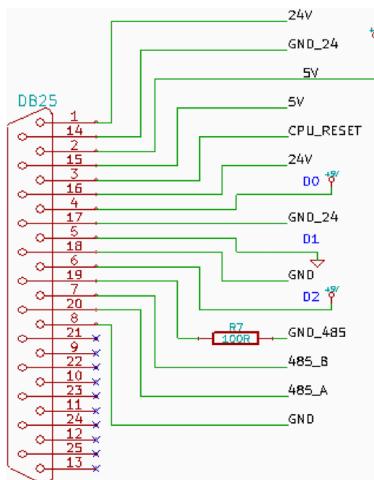
La comunicación MODBUS-TCP se probó utilizando software SCADA de distintos proveedores. Fue posible leer entradas y salidas y forzar salidas como sería en cualquier otro PLC.

HARDWARE

El primer prototipo de OpenPLC tenía 4 placas:

Bus Board:

Actúa como un bastidor con una fuente de alimentación de 5VDC integrada. Cada módulo se conecta a la placa de bus a través de un conector DB-25. La comunicación entre módulos se realiza a través de un bus RS-485.



Es el cerebro del OpenPLC. El procesador es el AVR ATmega2560. También incluye otro circuito integrado (IC), el Wiznet W5100, responsable de la comunicación con el ethernet. El Wiznet W5100 soporta cableados de protocolo TCP/IP como TCP, UDP y Ethernet, entre otros, tiene 16 KB de memoria interna para búferes Tx/Rx y acepta interfaz serie (sobre SPI) o paralela.

Input card:

La tarjeta de entrada es un módulo de entrada digital para el OpenPLC. Se procesan las entradas digitales leídas por el circuito de señal de acondicionamiento y se envían a la tarjeta CPU. La tarjeta de entrada tiene 8 circuitos de entrada aislados, de modo que cada módulo puede leer hasta 8 señales digitales al mismo tiempo. El estado de cada entrada se envía a la tarjeta CPU, a través del bus RS-485, para ser procesado según el Ladder Logic.

Output card:

Cada tarjeta de salida tiene 8 salidas basadas en relés que controlan hasta 8 cargas al mismo tiempo.

SOFTWARE

La norma IEC 61131-3 define cinco lenguajes en los que se pueden programar los PLC: FBD (Function Block Diagram), Ladder Diagram, Structured Text, Instruction List and SFC (Sequential Function Chart).

El más conocido es la lógica Ladder. El OpenPLC puede realizar muchas otras funciones, como la comunicación a través de Ethernet para sistemas de supervisión MODBUS-TCP, RS-485 y USB, el control de módulos individuales, la generación de mensajes de error, etc.

El resultado final contiene la lógica Ladder y las opciones del firmware del OpenPLC

COMUNICACIÓN MODBUS

Como el OpenPLC tiene Ethernet sobre TCP-IP, se implementó soporte para el protocolo MODBUS-TCP. Sólo se implementaron las funciones más utilizadas del protocolo, como se muestra a continuación:

- | | |
|--------------------------------------|---|
| - FC01 → Leer Estado de Bobina | - FC02 - Lectura del estado de las entradas |
| - FC03 → Leer Registros de Retención | - FC05 - Forzar una bobina |
| - FC15 → Forzar varias bobinas | |

BOARD COMUNICACIÓN

Para que cada módulo del OpenPLC tenga forma de comunicarse con la CPU crearon un protocolo en la capa de aplicación para estandarizar los mensajes enviados y recibidos: Protocolo OPLC. Se trata de un protocolo sencillo que encapsula cada mensaje enviado o recibido con información sobre el destino, el tamaño del mensaje y la función que debe ejecutarse.

Start	Size	Function	Address	Data
1 Byte	1 Byte	1 Byte	1 Byte	n Bytes

Cada mensaje comienza con un byte de “Start”, que siempre es 0x7E. El receptor sólo procesa el mensaje después de recibir el byte de inicio. El campo “Size” debe contener únicamente el tamaño (en bytes) del campo “Data”. El campo “Function” está relacionado con el campo “Data”. Significa que lo que el receptor hará con los datos recibidos depende de la función. Para el protocolo OPLC se han implementado cinco funciones:

1. 0x01 - Pregunta por el tipo de tarjeta
2. 0x02 - Cambia la dirección lógica de la tarjeta
3. 0x03 - Lectura de discrete inputs
4. 0x04 - Establecer discrete outputs
5. 0x05 - Error message

El campo de “Address” puede tener la dirección lógica o física de la tarjeta, según la función solicitada.

ESTUDIO DEL PROTOCOLO MODBUS

INTRODUCCIÓN AL PROTOCOLO MODBUS

MODBUS es un protocolo de mensajería de capa de aplicación, situado en el nivel 7 del modelo OSI, que proporciona comunicación cliente/servidor entre dispositivos conectados en diferentes tipos de buses o redes. Existen varias versiones del protocolo Modbus para el puerto serie y ethernet: Modbus RTU, Modbus ASCII, Modbus TCP y Modbus Plus.

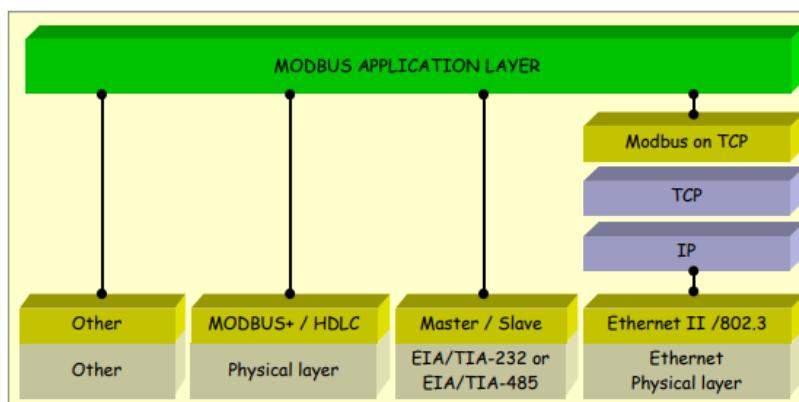


Figure 1: MODBUS communication stack

La comunicación entre los nodos Modbus se realiza mediante mensajes del tipo enviar solicitud y leer respuesta. Modbus es un estándar abierto que describe el diálogo de comunicación de mensajes. Modbus se comunica a través de varios tipos de medios físicos, como rs-232 serie, ... y a través de ethernet. En interfaces simples, como rs-232, los mensajes Modbus se envían en formato plano a través de la red y ésta se dedica únicamente a la comunicación Modbus. Si la red requiere múltiples dispositivos heterogéneos utilizando un sistema de red más versátil como TCP/IP sobre Ethernet, los mensajes Modbus se incrustan en paquetes Ethernet con un formato prescrito para esta interfaz física.

Así, en este caso, Modbus y otros tipos de protocolos mixtos pueden coexistir en la misma interfaz física al mismo tiempo. Los dispositivos Modbus se comunican utilizando una técnica de cliente servidor maestro/esclavo para Ethernet en la que sólo un dispositivo puede iniciar transacciones denominadas consultas. Los otros dispositivos responden suministrando los datos solicitados al maestro o realizando la acción solicitada en una consulta. Un esclavo es cualquier dispositivo periférico, como un transductor de E/S, una válvula, una unidad de red o dispositivos de medición, que procesa información y envía su mensaje de respuesta al maestro mediante Modbus.

DESCRIPCIÓN DEL PROTOCOLO

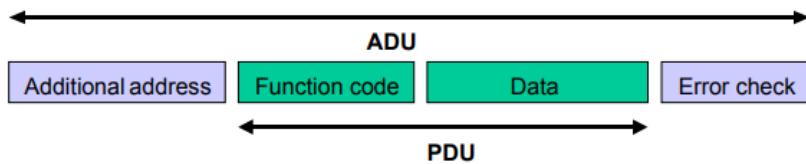


Figure 2: General MODBUS frame

El protocolo MODBUS define una unidad de datos de protocolo (**PDU**) sencilla e independiente de las capas de comunicación subyacentes. El mapeo del protocolo MODBUS en buses o redes específicas puede introducir algunos campos adicionales en la Unidad de Datos de Aplicación (**ADU**). Cada mensaje Modbus tiene la misma estructura. En cada mensaje se presentan cuatro elementos básicos:

- *Dirección esclavo*: se utiliza para definir qué dispositivo esclavo debe responder a un mensaje.
- *Código de función*: en qué grupo de datos de registro lee o escribe.

El campo de código de función de una unidad de datos MODBUS se codifica en un byte. Los códigos válidos están en el rango de 1 ... 255 decimal (el rango 128 - 255 está reservado y se utiliza para respuestas de excepción). Cuando se envía un mensaje desde un dispositivo Cliente a un dispositivo Servidor, el campo de código de función indica al servidor qué tipo de acción debe realizar. El código de función "0" no es válido.

- *Datos*: Número de bytes de datos solicitados por el maestro. Proporciona al esclavo cualquier información adicional que éste necesite para completar la acción especificada por el código de función.

El campo de datos de los mensajes enviados desde un cliente a los dispositivos del servidor contiene información adicional que el servidor utiliza para realizar la acción definida por el código de función. Puede incluir elementos como direcciones discretas y de registro, la cantidad de elementos que deben manejarse y el recuento de bytes de datos reales en el campo. El campo de datos puede ser inexistente (de longitud cero) en ciertos tipos de peticiones, en este caso el servidor no requiere ninguna información adicional y sólo el código de función especifica la acción.

- Comprobación de errores CRC:

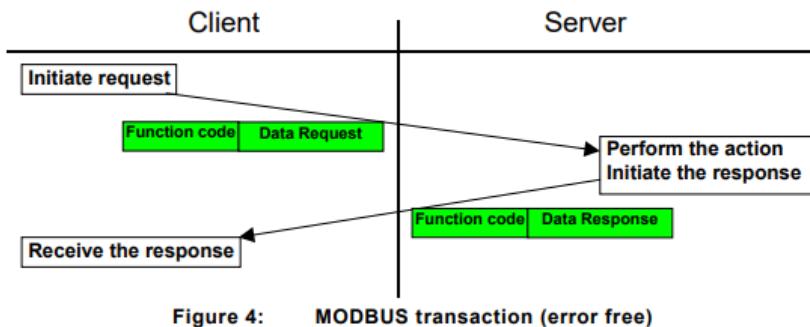


Figure 4: MODBUS transaction (error free)

Si no se produce ningún error relacionado con la función MODBUS solicitada en un MODBUS ADU, el campo de datos de una respuesta de un servidor a un cliente contiene los datos solicitados.

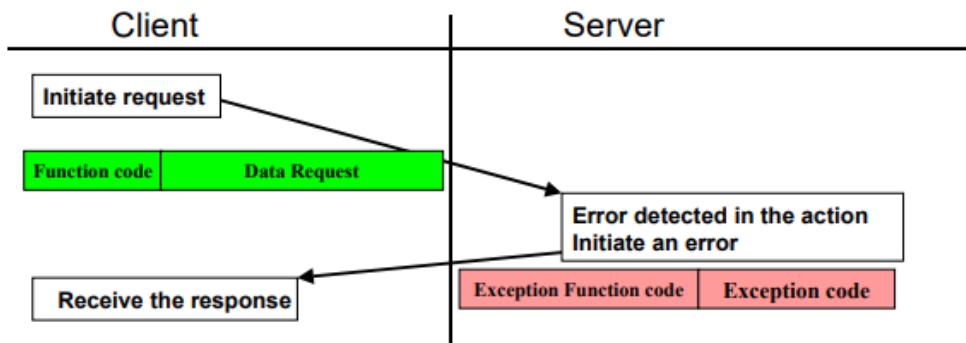


Figure 5: MODBUS transaction (exception response)

Si se produce un error relacionado con la función MODBUS solicitada, el campo de datos de una respuesta de un servidor a un cliente contiene un código de excepción que la aplicación del servidor puede utilizar para determinar la siguiente acción a realizar.

MODELO DE DATOS MODBUS

Primary tables	Object type	Type of	Comments
Discretes Input	Single bit	Read-Only	This type of data can be provided by an I/O system.
Coils	Single bit	Read-Write	This type of data can be alterable by an application program.
Input Registers	16-bit word	Read-Only	This type of data can be provided by an I/O system
Holding Registers	16-bit word	Read-Write	This type of data can be alterable by an application program.

TRANSACCIONES EN MODBUS

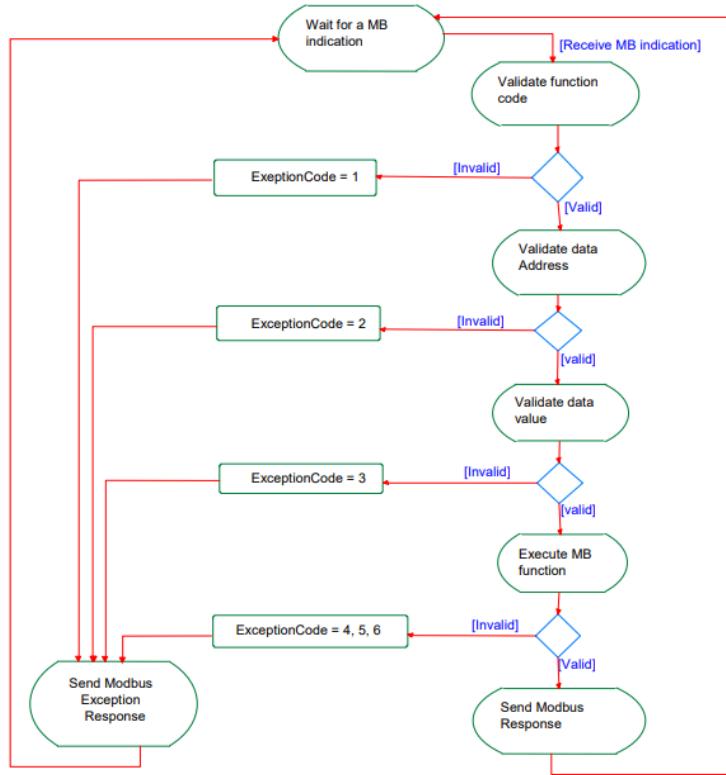
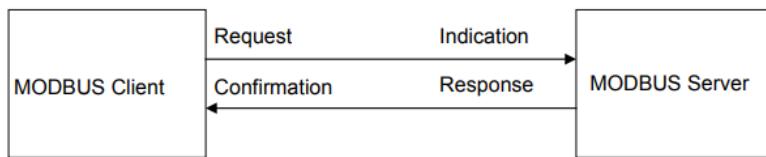


Figure 9 MODBUS Transaction state diagram

MODELO MODBUS TCP/IP

Modelo Cliente/Servidor:

El servicio de mensajería MODBUS proporciona una comunicación Cliente/Servidor entre dispositivos conectados en una red Ethernet TCP/IP.



“Request” es el mensaje enviado en la red por el Cliente para iniciar una transacción,
“Indication” es el mensaje de Solicitud recibido en el lado del Servidor,
“Response” es el mensaje de Respuesta enviado por el Servidor,
“Confirmation” es el mensaje de respuesta recibido en el lado del cliente.

Los servicios de mensajería MODBUS (Modelo Cliente / Servidor) se utilizan para el intercambio de información en tiempo real entre dos aplicaciones de dispositivo, entre aplicación de dispositivo y otro dispositivo, entre aplicaciones HMI/SCADA y otros dispositivos y entre un PC y un programa de dispositivos que proporciona servicios en línea.

Descripción del Protocolo

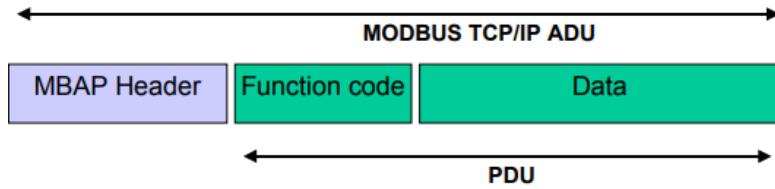


Figure 3: MODBUS request/response over TCP/IP

En TCP/IP se usa una cabecera específica, llamada MBAP header (MODBUS Application Protocol header), para identificar la ADU. Esta cabecera presenta algunas diferencias:

- El campo 'slave address' MODBUS que se utiliza habitualmente en la línea serie MODBUS se sustituye por un 'Unit Identifier' de un byte dentro de la cabecera MBAP. El 'Unit Identifier' se utiliza para comunicarse a través de dispositivos tales como puentes, routers y pasarelas que utilizan una única dirección IP para soportar múltiples unidades finales MODBUS independientes.
- Todas las peticiones y respuestas MODBUS están diseñadas de tal manera que el destinatario pueda verificar que un mensaje ha finalizado.
- Cuando MODBUS se transmite a través de TCP, la información de longitud adicional se transmite en el encabezado MBAP para permitir que el destinatario reconozca el mensaje, incluso si el mensaje se ha dividido en varios paquetes para su transmisión. La existencia de reglas de longitud explícitas e implícitas, y el uso de un código de comprobación de errores CRC-32 (en Ethernet) da como resultado una posibilidad infinitesimal de corrupción no detectada en un mensaje de solicitud o respuesta.

La cabecera tiene una longitud de 7 Bytes:

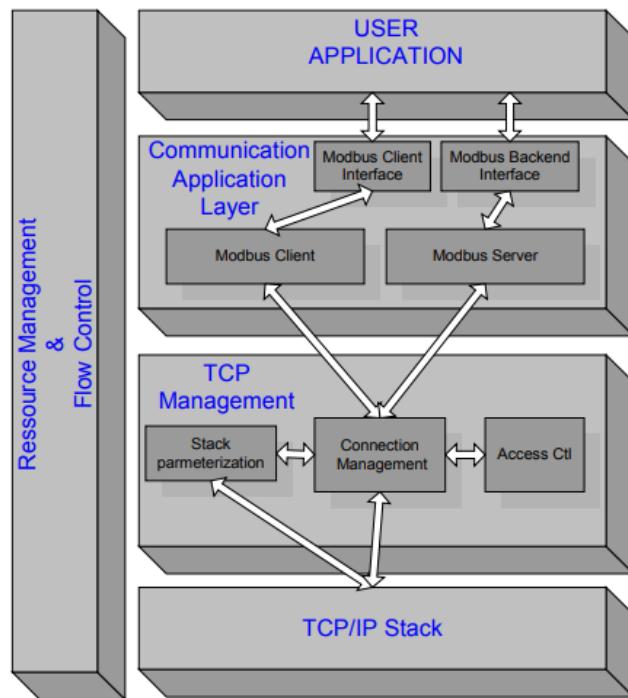
Fields	Length	Description -	Client	Server
Transaction Identifier	2 Bytes	Identification of a MODBUS Request / Response transaction.	Initialized by the client	Recopied by the server from the received request
Protocol Identifier	2 Bytes	0 = MODBUS protocol	Initialized by the client	Recopied by the server from the received request
Length	2 Bytes	Number of following bytes	Initialized by the client (request)	Initialized by the server (Response)
Unit Identifier	1 Byte	Identification of a remote slave connected on a serial line or on other buses.	Initialized by the client	Recopied by the server from the received request

- *Transaction identifier*: Se utiliza para el emparejamiento de transacciones, el servidor MODBUS copia en la respuesta el identificador de transacción de la solicitud.

- *Protocol Identifier*: Se utiliza para la multiplexación intrasistema. El protocolo MODBUS se identifica por el valor 0.
- *Length*: El campo de longitud es un recuento en bytes de los siguientes campos, incluidos los campos Unidad y los campos de datos.
- *Unit Identifier*: Este campo se utiliza para el enrutamiento dentro del sistema. Este campo es MODBUS en la solicitud y debe ser devuelto con el mismo valor en la respuesta del servidor. la respuesta del servidor.

Todas las ADU MODBUS/TCP se envían a través de TCP en el puerto registrado 502.

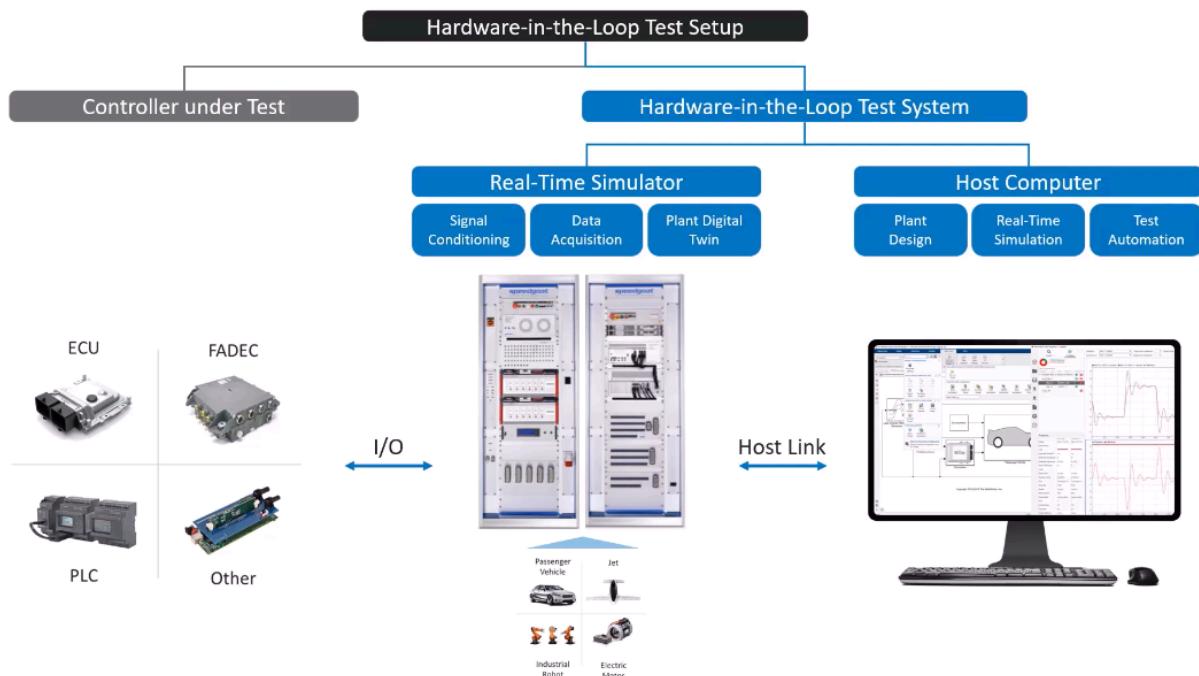
MODELO DE ARQUITECTURA



ESTUDIO DEL MÉTODO DE TESTEO HARDWARE IN THE LOOP

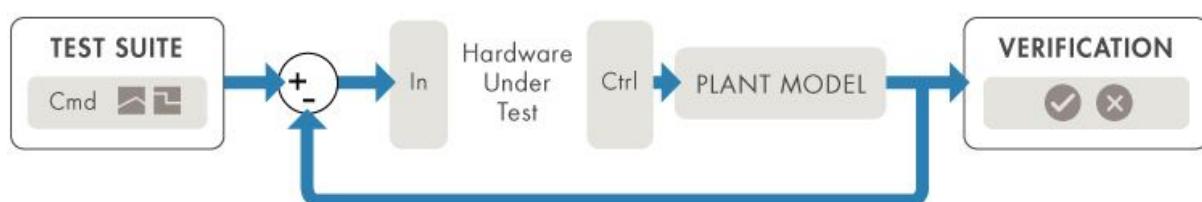
INTRODUCCIÓN

La simulación hardware-in-the-loop (HIL) es una técnica para validar su algoritmo de control mediante la creación de un entorno virtual en tiempo real que representa su sistema físico a controlar. Ayuda a probar el comportamiento de los algoritmos de control sin necesidad de prototipos físicos. Se pueden ejecutar fácilmente miles de escenarios posibles para poner en práctica a su controlador sin el costo ni el tiempo asociados con las pruebas físicas de la actualidad.



¿Cómo funciona la simulación HIL?

1. Se crea y simula una implementación virtual en tiempo real de componentes físicos como una planta, sensores y actuadores en un ordenador de destino en tiempo real.
2. El algoritmo de control se ejecuta en un controlador integrado y el modelo de la planta o el entorno se ejecuta en tiempo real en un ordenador de destino conectado al controlador. El controlador embebido interactúa con la simulación del modelo de planta a través de varios canales de E/S.
3. Se perfeccionan las representaciones de software de los componentes y se sustituyen gradualmente las partes del entorno del sistema por los componentes de hardware reales.



Esta imagen muestra una simulación HIL en la que el Hardware Under Test es un controlador embebido y el PLANT MODEL es una representación de un sistema físico.

La simulación HIL es especialmente útil cuando probar un algoritmo de control en el sistema físico real es costoso o peligroso. Se utiliza ampliamente en los sectores de automoción, aeroespacial y defensa, y de maquinaria y automatización industrial para probar diseños integrados.

ESTUDIO DE MÉTRICAS DE RENDIMIENTO

MÉTRICAS DE PFGs

PROTOCOLO MODBUS

Introducción

La intención es probar las capacidades de una red modbus. Para ello, se ha generado una red de pruebas la cual será la base de un cyber range. Dentro de la red modbus ha creado un conjunto de contenedores que funcionan como clientes y otros como servidores modbus. Para llevar a cabo el despliegue virtualizado se ha usado la herramienta Docker.

Adicionalmente, como parte de la red Modbus, también se han desplegado otros contenedores con la función de monitorización y control especializadas en la captura de métricas y monitorización.

Al despliegue de dicha red se han ejecutado y procesado distintos escenarios con un servidor y variando el número de clientes (n)

Implementación

Determinamos el número máximo de clientes Modbus TCP que aceptan estos dispositivos, el cual, normalmente llega hasta 32. Dado que la arquitectura diseñada es de n:1, se procede a realizar experimentos variando el número de clientes a 1,4,8,16 y 32. El tiempo de recolección de datos es de 30 minutos en cada caso, desde el lanzamiento del Docker-Compose, para finalmente comparar los resultados.

Links para encontrar los programas que usa. Los suyos ya no sirven.

<https://github.com/eterey/pymodbus3/blob/master/examples/common/synchronous-server.py>
<https://github.com/eterey/pymodbus3/blob/master/examples/common/performance.py>

Como nosotros tenemos python 3.10 y él usa 3.7 tenemos que tener en cuenta a la hora de escribir un dockerfile y que ya no existen unos directorios para modificarlos. e.g. del primero al segundo

```
from pymodbus.client.sync import ModbusTcpClient  
from pymodbus.client.tcp import ModbusTcpClient
```

Resultados

Red MODBUS TCP

La carga de memoria de los clientes se mantiene estable sobre unos 10MB por cada cliente mientras que la memoria del servidor aumenta de forma lineal según el tiempo que lleve la red activa.

El valor inicial de la memoria requerida por el servidor es inicialmente igual o similar para todos los casos, sin embargo, la pendiente del crecimiento inicial varía según el número de clientes conectados al servidor. A mayor número de clientes, mayor es la velocidad con la que el servidor aumenta su uso de memoria.

No. Clientes	Mínimo	Media	Máximo	Crecimiento
1 Cliente	8,18MB	23,39MB	38,25MB	30,07MB
4 Clientes	12,50MB	31,83MB	49,15MB	36,65MB
8 Clientes	14,23MB	34,91MB	54,09MB	39,86MB
16 Clientes	16,33MB	52,72MB	83,86MB	67,53MB
32 Clientes	21,70MB	61,69MB	99,32MB	77,62MB

Tabla 2 Crecimiento de memoria en el servidor TCP

El comportamiento de la memoria debido a la optimización de Docker es inicialmente mejor que en los dispositivos reales, ya que con 32 cliente la primera media hora utiliza 61MB de memoria de los 256MB disponibles, cuando un dispositivo real estaría ya muy cerca de su límite de memoria, el aumento lineal de memoria que nos da Docker significa que con 32 clientes el servidor colapsaría por memoria en menos de una hora y media, lo cual no debería de pasar en un dispositivo real.

	Mínimo	Máximo
1 Cliente	1428pet./s	2010pet./s
4 Clientes	132pet./s	510pet./s
8 Clientes	66pet./s	286pet./s
16 Clientes	29pet./s	98pet./s
32 Clientes	38pet./s	44pet./s

Tabla 3 Peticiones por segundo en TCP

Red MODBUS TLS

El comportamiento de la memoria, para un número de clientes bajo, sigue siendo parecido al obtenido en TCP con la diferencia de que el valor inicial, algo más que el doble que en TCP, lo cual como ya hemos mencionado era lo esperado

Ningún cliente parece usar más memoria que un TCP, con la diferencia que el valor inicial es algo más que el doble que en TCP, lo cual era lo esperado.

	Mínimo	Media	Máximo	Crecimiento
1 Cliente	19,77MB	32,00MB	43,84MB	30,07MB
4 Clientes	27,40MB	57,24MB	90,63MB	36,65MB
8 Clientes	26,25MB	46,32MB	63,83MB	39,86MB
16 Clientes	29,03MB	45,32MB	62,56MB	67,53MB
32 Clientes	36,90MB	54,69MB	99,32MB	69,58MB

Tabla 4 Crecimiento de memoria en el servidor TLS

La caída del rendimiento del servidor en el número de peticiones es aplicable a los dispositivos reales.

	Mínimo	Máximo
1 Cliente	839pet./s	922pet./s
4 Clientes	369pet./s	437pet./s
8 Clientes	60pet./s	191pet./s
16 Clientes	22pet./s	83pet./s
32 Clientes	10pet./s	38pet./s

Se aprecia un rendimiento menor que en TCP junto con una mayor estabilidad entre máximos y mínimos. Además, en el caso de 32 clientes, se llega a apreciar la sobrecarga del servidor durante un momento, donde no es capaz de aceptar una conexión de un cliente.

```
[DEBUG/MainProcess] 35 requests/second
[DEBUG/MainProcess] time taken to complete 1000 cycle by 10 workers is 28.26242383600038 seconds
[DEBUG/MainProcess] 12 requests/second
[DEBUG/MainProcess] time taken to complete 1000 cycle by 10 workers is 79.63126860699958 seconds
[DEBUG/MainProcess] 10 requests/second
[DEBUG/MainProcess] time taken to complete 1000 cycle by 10 workers is 91.94805216700115 seconds
[DEBUG/MainProcess] 14 requests/second
[DEBUG/MainProcess] time taken to complete 1000 cycle by 10 workers is 67.59918666399972 seconds
[DEBUG/MainProcess] 38 requests/second
[DEBUG/MainProcess] time taken to complete 1000 cycle by 10 workers is 26.236887884999305 seconds
[ERROR/MainProcess] failed to run test successfully
  traceback (most recent call last):
    File "performanceTLS.py", line 74, in single_client_test
      client.read_holding_registers(10, 1, unit=1) #.registers[0] da error por algun motivo(se supone que no tiene atributo registers)
    File "/usr/local/lib/python3.8/dist-packages/pymodbus/client/common.py", line 114, in read_holding_registers
      return self.execute(request)
    File "/usr/local/lib/python3.8/dist-packages/pymodbus/client/sync.py", line 107, in execute
      raise ConnectionException("Failed to connect[%s]" % (self.__str__()))
pymodbus.exceptions.ConnectionException: Modbus Error: [Connection] Failed to connect[ModbusTlsClient(autoserver:8020)]
[DEBUG/MainProcess] 36 requests/second
[DEBUG/MainProcess] time taken to complete 1000 cycle by 10 workers is 27.777717125998606 seconds
[DEBUG/MainProcess] 13 requests/second
[DEBUG/MainProcess] time taken to complete 1000 cycle by 10 workers is 72.06350807799936 seconds
[DEBUG/MainProcess] 10 requests/second
[DEBUG/MainProcess] time taken to complete 1000 cycle by 10 workers is 93.74398726599975 seconds
[DEBUG/MainProcess] 13 requests/second
[DEBUG/MainProcess] time taken to complete 1000 cycle by 10 workers is 72.9443238040003 seconds
[DEBUG/MainProcess] 38 requests/second
[DEBUG/MainProcess] time taken to complete 1000 cycle by 10 workers is 26.0810097829999 seconds
[DEBUG/MainProcess] 36 requests/second
[DEBUG/MainProcess] time taken to complete 1000 cycle by 10 workers is 27.136822796999695 seconds
```

A pesar de los resultados de memoria obtenidos mediante Dockprom en Grafana, este resultado es realmente fiel a un dispositivo real, ya que el servidor (limitado a 256MB de memoria RAM) va perdiendo rendimiento según el número de clientes crece hasta dar el primer fallo en 32 clientes, precisamente el límite de clientes máximos aceptados por los dispositivos reales habituales.

OPENPLC

Es nuestro trabajo pero en lugar de usar el SCADA-LTS nosotros variamos el número de clientes para medir el rendimiento y latencia del OpenPLC

HARDWARE IN THE LOOP

Introducción

La intención es obtener un software capaz de comunicar de manera eficiente y en tiempo real la simulación del modelo de cualquier sistema con algún programa SCADA y un controlador, con el fin de poder realizar el control de este modelo de la misma manera en la que se podría controlar un sistema de producción real. El tipo de conexión utilizado ha sido una comunicación vía Ethernet a través de un servidor de OPC (OPC no nos interesa). En definitiva, conexión entre un autómata PLC, servidor OPC y cliente OPC de Matlab/Simulink.

Se ha escogido un sistema sencillo para testear el controlador. El sistema en lazo cerrado se representaría de la siguiente manera:

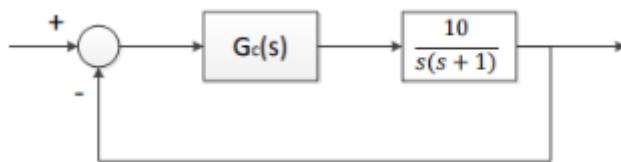


Figura 1 Sistema en lazo cerrado

Trabajo realizado

1. Alternativas para interconectar aplicaciones externas con un autómata
2. Configuración de una red local

El ordenador y el PLC deben tener la misma máscara subred

Configura la máscara del autómata en CX-Programmer

Dos opciones para conectarse al autómata vía ethernet. Utilizar un cable y conectarlo directamente o conectar el cable a un switch y conectar al autómata a través de un switch

3. Configurar tablas de I/O del PLC
4. Real-Time Windows Target

Se trata de una herramienta de Matlab/Simulink que permite ejecutar los modelos y los bloques conectados a tablas de entrada y salida en tiempo real. Permite al usuario crear y controlar sistemas en tiempo real para "hardware-in-the-loop simulations".

5. OPC que no interesa y la combinación del Real-Time con el OPC
6. Configuración DCOM para habilitar la comunicación

DCOM es una arquitectura basada en la estructura cliente-servidor para permitir la comunicación entre dos aplicaciones que se ejecutan en equipos distintos.

7. SCADA.

La razón por la que se ha decidido incluir una interfaz operario/máquina (HMI) en este proyecto ha sido la necesidad de facilitar la utilización del software al usuario.

8. Simulación teórica

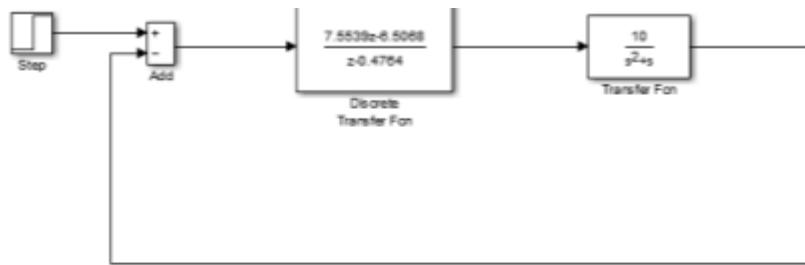


Figura 20 Lazo de control en modo discreto

9. Simulación HIL

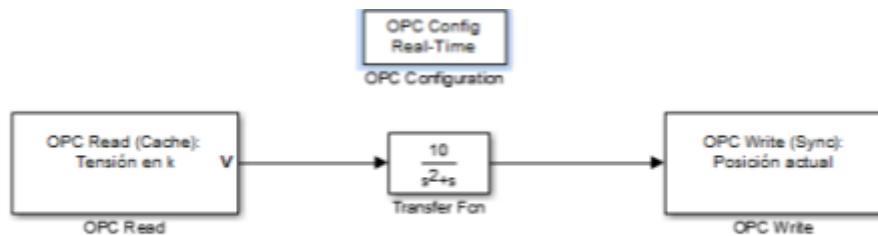
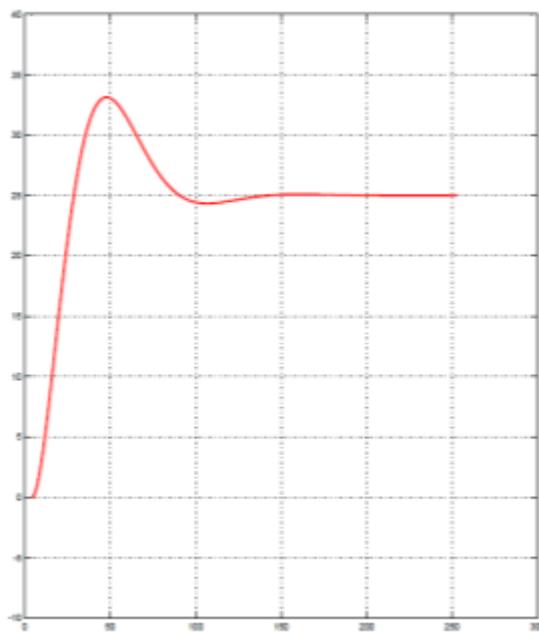


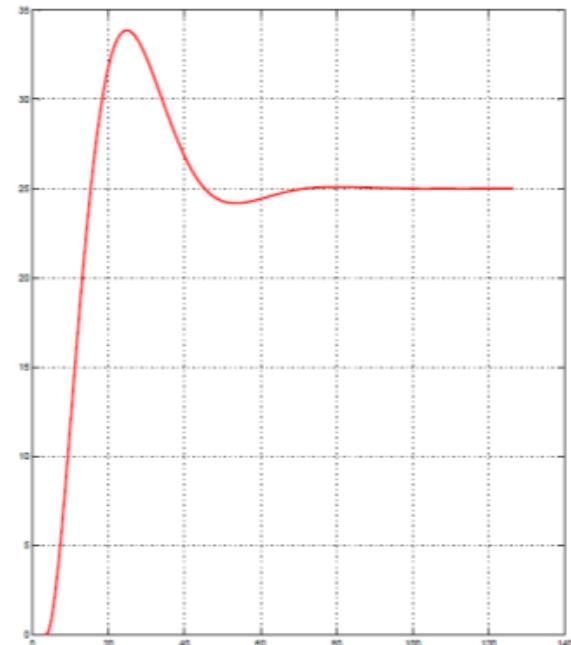
Figura 22 Simulación con "hardware-in-the-loop"

Se realiza un cambio de variable donde la variable t representa el tiempo real y la t' representa la variable en el modelo ralentizado.

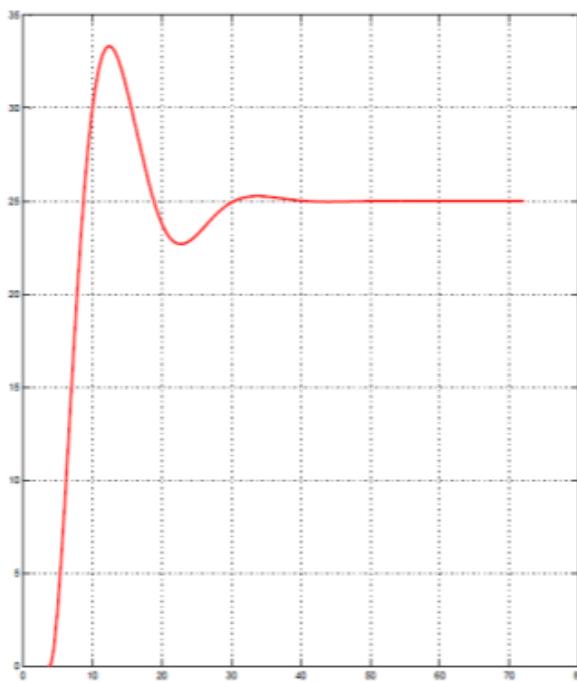
Simulación para $t'=100t$



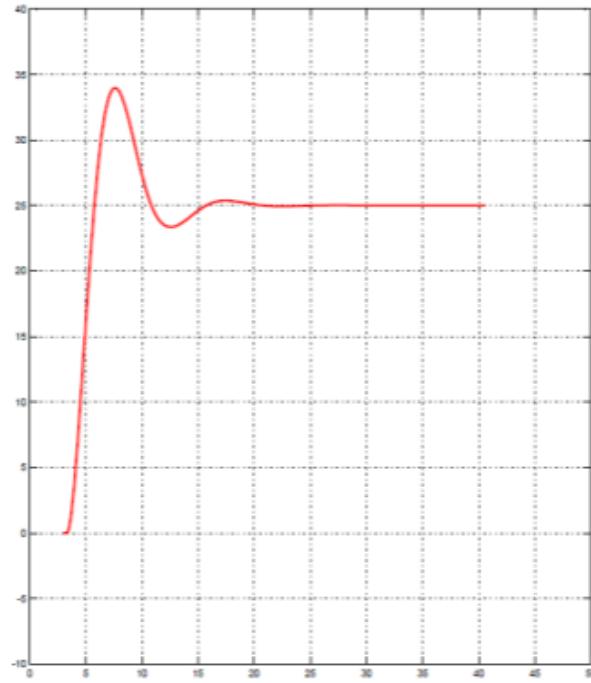
Simulación para $t'=50t$



Simulación para $t'=20t$



Simulación para $t'=10t$



Simulación para $t'=2t$



El último caso es totalmente crítico, ya que nuestro controlador es incapaz de controlar el sistema. Con esta velocidad impuesta nuestro sistema se convierte en un sistema inestable y totalmente aleatorio.

Conclusiones

Por tanto, se llega a la conclusión de que una ralentización es necesaria para que nuestro sistema funcione correctamente, pero hay que tener cuidado con esa ralentización ya que si es muy pequeña el sistema se vuelve inestable.

A medida que se incrementa la velocidad de simulación, se llega antes a la estabilidad pero nuestro controlador tiene mayores dificultades para controlar el sistema. Así, si se desea obtener unos resultados buenos y certeros, es conveniente ralentizar la simulación lo máximo posible. Cuanto más ralentizamos nuestro sistema, más se parecerán nuestros valores obtenidos a los teóricos.

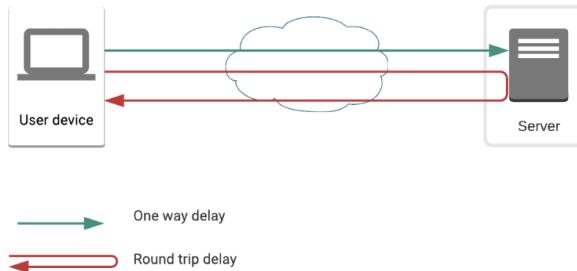
MÉTRICAS A ANALIZAR

Network Performance: (Rendimiento de la red)

Latency: (Latencia)

La latencia es el tiempo que tarda un paquete en viajar por la red de un punto a otro. En la mayoría de los casos, nos referiremos a la latencia de red como el tiempo necesario para que un paquete se mueva desde un cliente/usuario a un servidor a través de la red.

Formas de medir la latencia de la red:



Throughput: (Rendimiento)

El rendimiento es la cantidad de datos enviados/recibidos por unidad de tiempo.

Reliability: (Fiabilidad)

La fiabilidad de la red se refiere a la capacidad de una red informática para proporcionar conectividad y servicios a los usuarios de forma constante y fiable. Una red fiable es aquella que funciona sin problemas, sin interrupciones frecuentes, tiempos de inactividad o fallos.

Availability: (Disponibilidad de red)

La disponibilidad de la red es la cantidad de tiempo de actividad de un sistema de red en un intervalo de tiempo específico. El tiempo de actividad se refiere a la cantidad de tiempo que una red está plenamente operativa. La disponibilidad de la red se mide en porcentaje y se supervisa para garantizar que la red funcione de forma constante para los usuarios finales.

$$\text{Network availability} = \frac{\text{Uptime}}{\text{Total time}} = \frac{\text{Uptime}}{(\text{Uptime} + \text{downtime})}$$

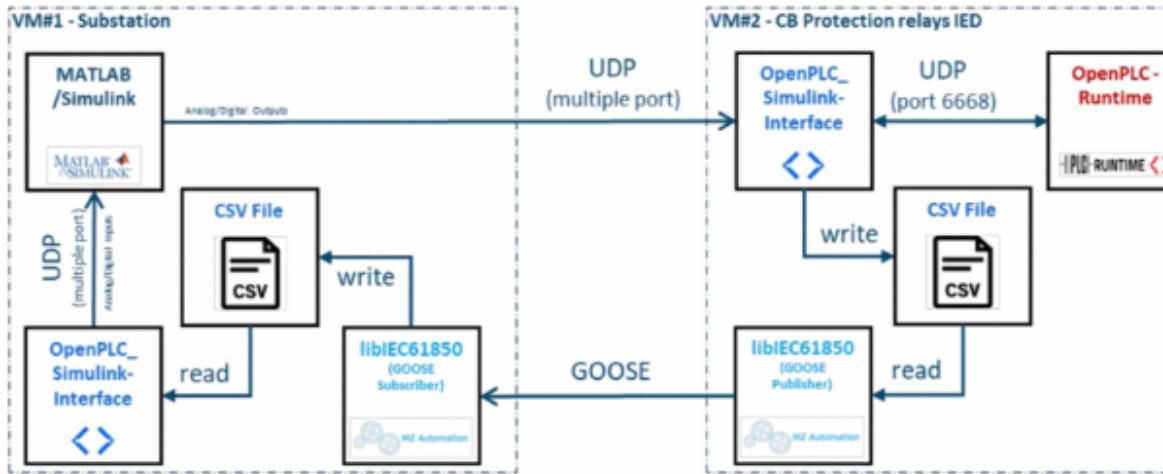
MÉTRICAS DE PAPERS

OpenPLC and lib61850 Smart Grid Testbed: Performance Evaluation and Analysis of GOOSE Communication

El artículo detalla el diseño y la implementación de un banco de pruebas de simulación utilizando MATLAB/Simulink, OpenPLC y la biblioteca lib61850. Este banco de pruebas simula un sistema de red inteligente y genera mensajes IEC61850 GOOSE para la creación de conjuntos de datos.

Testbed Design

Utilizamos el software MATLAB/Simulink para la simulación del modelo de red inteligente. Un componente crucial en nuestra configuración es el OpenPLC_Simulink-Interface (OS-Interface), un script en lenguaje Simulink C proporcionado por Thiago Alves. Esta interfaz facilita la comunicación entre la aplicación MATLAB /Simulink y el entorno OpenPLC Runtime, que empleamos para simular los dispositivos IED.



A. Testbed System Architecture and Setup

El banco de pruebas consta de dos máquinas virtuales (VM) basadas en Linux. La primera VM aloja el software MATLAB/Simulink, el OS-Interface y la librería de suscriptores lib61850. La segunda VM, aloja el OpenPLC-Runtime, el OS-Interface y la librería lib61850 publisher.

B. Integration and Communication Setup

Los datos se transmiten utilizando el protocolo UDP, con puertos predefinidos en la aplicación MATLAB /Simulink utilizando los bloques UDP Send y Receive.

Configuración y metodología del experimento

Empezamos por simular una situación de fallo en el bus de 33 KV utilizando el bloque "Fallo 33_bus" en el modelo de simulación, lo que provocó un consumo y una fuga de corriente adicionales en el bus de 33 KV.

La siguiente imagen muestra: Mensaje de MATLAB/simulink udp con valores de corriente de falla

Arrival Time	Source	Destination	Protocol	Info	Data
06:03:58.498560	23.23.23.254	23.23.23.255	UDP	43542 → 30122 Len=8	0000000000000000
06:03:58.534892	23.23.23.254	23.23.23.255	UDP	43542 → 30122 Len=8	9ee63a758d42a43e
06:04:05.468931	23.23.23.254	23.23.23.255	UDP	43542 → 30122 Len=8	6123935b4b77403e
06:04:05.470068	23.23.23.254	23.23.23.255	UDP	43542 → 30122 Len=8	0000000000000000
06:04:09.894531	23.23.23.254	23.23.23.255	UDP	43542 → 30111 Len=8	0000000000000000
06:04:09.960892	23.23.23.254	23.23.23.255	UDP	43542 → 30111 Len=8	3e2c0c70bd20fa3c
06:04:18.401253	23.23.23.254	23.23.23.255	UDP	43542 → 30111 Len=8	12b6f1012d003a3e
06:04:18.405045	23.23.23.254	23.23.23.255	UDP	43542 → 30111 Len=8	0000000000000000
06:04:22.147907	23.23.23.254	23.23.23.255	UDP	43542 → 30100 Len=8	0000000000000000
06:04:22.152112	23.23.23.254	23.23.23.255	UDP	43542 → 30100 Len=8	4e36e9b941a1e33c
06:04:26.066102	23.23.23.254	23.23.23.255	UDP	43542 → 30100 Len=8	de93418a01ae403e
06:04:26.076376	23.23.23.254	23.23.23.255	UDP	43542 → 30100 Len=8	0000000000000000

Resultados

El tiempo promedio que tardaron los respectivos suscriptores en transmitir y recibir los mensajes GOOSE fue de alrededor de 9 milisegundos. La detección y respuesta rápida de fallas son fundamentales para garantizar la estabilidad del sistema y evitar daños al equipo. También medimos el tiempo promedio desde el momento en que comenzamos a simular la falla en MATLAB/Simulink hasta el momento en que recibimos el comando de disparo en MATLAB/Simulink. Nuestras mediciones indicaron un tiempo de respuesta promedio de aproximadamente 3,5 segundos. Esta latencia se mantiene dentro de límites aceptables