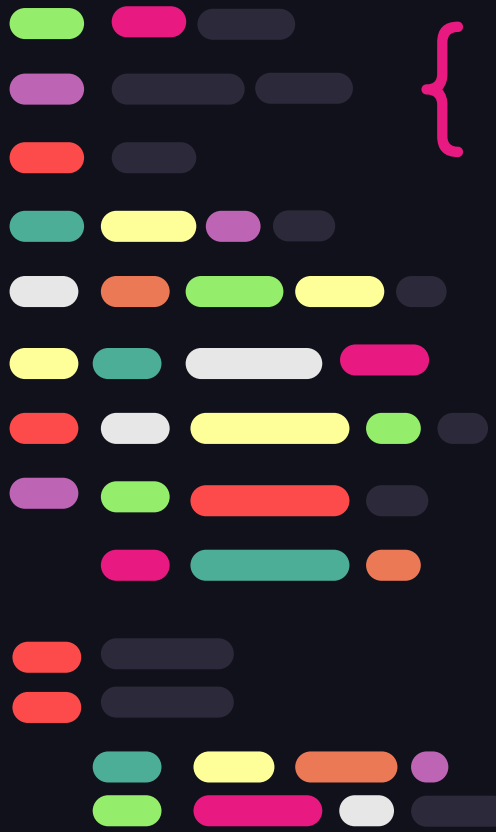


Modbus Protocol & Hardware in the Loop

< Izzy Scanlan >

Table of Contents

01 { ..

Introduction

} ..

Critical Infrastructures

{ ..

- Communications
- Water
- Energy
- Data & Cloud
- Health

} ..

Programmable Logic Controller

Role in Critical Interfaces:

- Control & Automate Essential Processes
- Ensure Safety, Reliability, & Efficiency
- Cybersecurity

02 { .. Background

} ..

Tools & Technologies

PLC

Manages data

Simulink

Simulates a wastewater treatment plan (WWTP)

pymodbusTCP

Python library to create clients & servers

Fabric

Python library to run multiple processes at once

Tools & Technologies

Hardware in the Loop

Testing physical components in a virtual environment

Modbus Protocol

Call & response loop between a client and server

Docker

Manages applications with containers

Tools & Technologies

Latency

Amount of time for a package to travel from point to point

Vertical Scalability

Increasing a load to assess program degradation

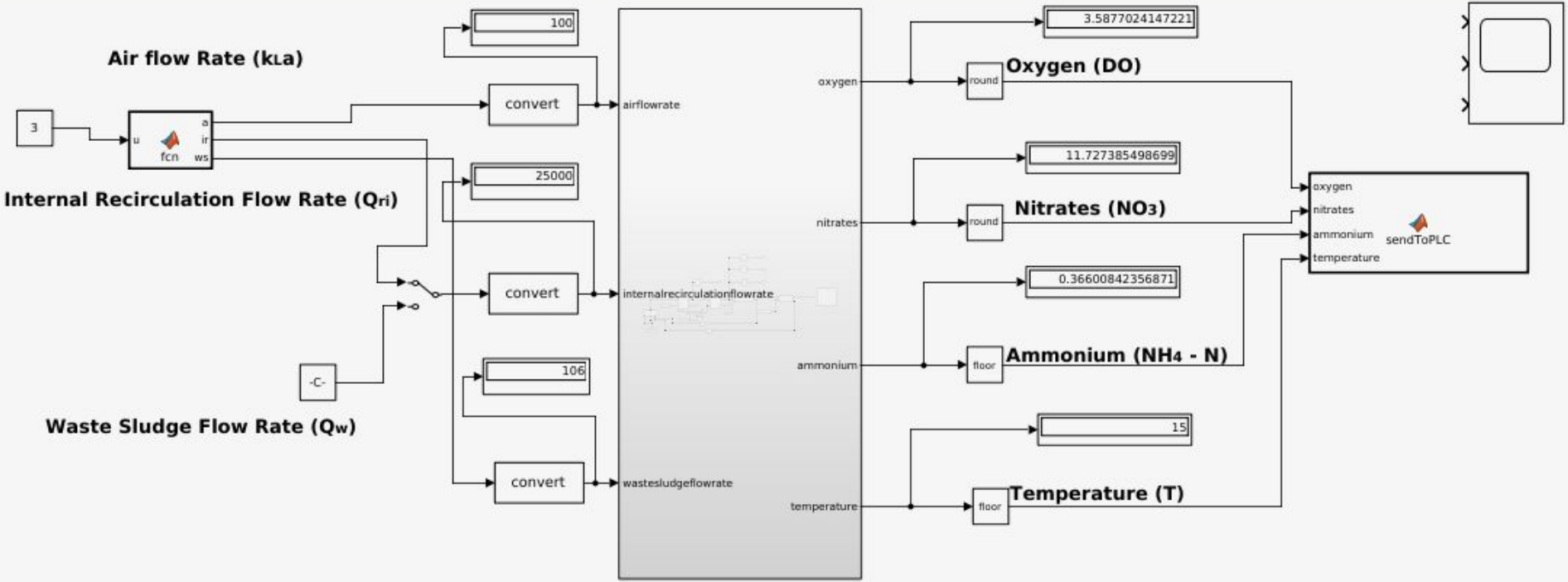
03 { .. Design

} ..

PLC-WWTP Interface

Interface

WWTP


Air flow Rate (kLa)
3
u
fcn
a
ir
ws
100
convert
airflowrate
Internal Recirculation Flow Rate (Qri)
25000
convert
internalrecirculationflowrate
-C-
Waste Sludge Flow Rate (Qw)
106
convert
wastesludgeflowrate
oxygen
nitrates
ammonium
temperature
3.5877024147221
round
Oxygen (DO)
11.727385498699
round
Nitrates (NO3)
0.36600842356871
floor
Ammonium (NH4 - N)
15
floor
Temperature (T)
oxygen
nitrates
ammonium
temperature
sendToPLC

04 { ..

Implementation

} ..

Code Overview

Server.py

Creates a Modbus server, initializes holding registers, receives data from WWTP

Plcsend.py

Creates 2 Modbus clients that connect to PLC & server, writes data to PLC & server

Fabfile.py

Runs server.py & plcsend.py concurrently

Server.py

```
server = ModbusServer("0.0.0.0", port=5020, no_block=True)
server.start()
server.data_bank.set_holding_registers(0,[0,0,0,0,0,0,0])
try:
  while True:

    hrv = server.data_bank.get_holding_registers(0,8)
    print("Receiving Sensors and Actuator Data", hrv)
    print(f" Oxygen:  {hrv[0]}\n Nitrates:  {hrv[1]}\n Ammonium:  {hrv[2]}\n Temperature:  {hrv[3]}")
    print(f" Aeration:  {hrv[5]}\n Internal Recirculation:  {hrv[6]}\n Waste Sludge Flow:  {hrv[7]}\n")
    time.sleep(.1)
```

Server.py

```
Receiving Sensors and Actuator Data [4, 12, 0, 15, 0, 25000, 106, 100]
 Oxygen:  4
 Nitrates:  12
 Ammonium:  0
 Temperature:  15
 Aeration:  25000
 Internal Recirculation:  106
 Waste Sludge Flow:  100
```

Plcsend.py

```python
# client connects to virtual server
vs = ModbusClient(host='10.63.28.53', port=5020, auto_open=True, debug=False)

# client connects to  real PLC
plc = ModbusClient(host='10.63.28.65', port=502, auto_open=True, debug=False)
```

```
Write Actuator values to virtual server: [25000, 106, 100]
Write Sensor Data to PLC: [4, 12, 0, 15]
Write Actuator values to virtual server: [25000, 106, 100]
Write Sensor Data to PLC: [4, 12, 0, 15]
Write Actuator values to virtual server: [25000, 106, 100]
Write Sensor Data to PLC: [4, 12, 0, 15]
```

Fabfile.py

```
@task
def run_concurrent(c):
    server_thread = threading.Thread(target=run_server)
    client_thread = threading.Thread(target=run_client)

    server_thread.start()
    sleep(2)
    client_thread.start()

    server_thread.join()
    client_thread.join()
```

Fabfile.py

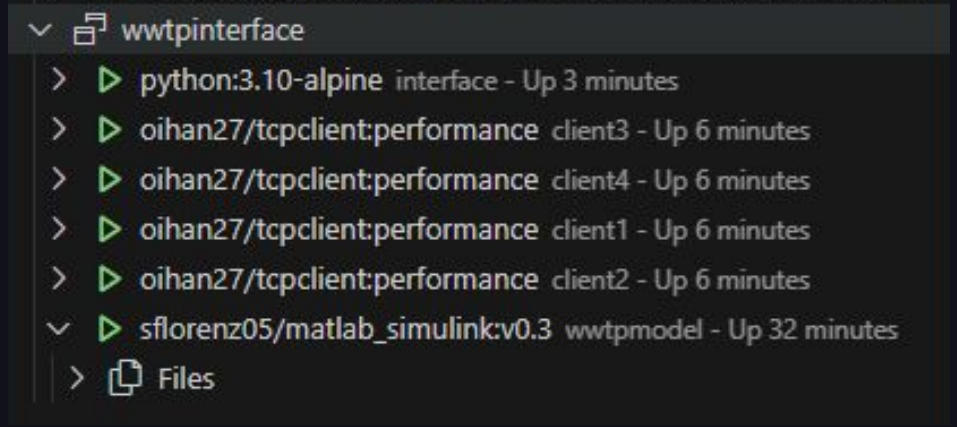
```
Write Sensor Data to PLC: [4, 12, 0, 15]
Receiving Sensors and Actuator Data [4, 12, 0, 15, 0, 25000, 106, 100]
 Oxygen:  4
 Nitrates:  12
 Ammonium:  0
 Temperature:  15
 Aeration:  25000
 Internal Recirculation:  106
 Waste Sludge Flow:  100

Write Actuator Data to virtual server: [25000, 106, 100]
Receiving Sensors and Actuator Data [4, 12, 0, 15, 0, 25000, 106, 100]
 Oxygen:  4
 Nitrates:  12
 Ammonium:  0
 Temperature:  15
 Aeration:  25000
 Internal Recirculation:  106
 Waste Sludge Flow:  100
```

Docker Integration

Docker Compose:

- Interface (python:3.10-alpine)
- MATLAB Simulink
- Clients to test system performance



```yaml
version: '3'

networks:
  ics:
    name: ics
    driver: bridge
    ipam:
      config:
        - subnet: 192.168.192.0/24

services:

  interface:
      image: python:3.10-alpine
      container_name: interface
      command: sh -c "pip install fabric pymodbustcp && cd /Interface && fab run-concurrent && tail -f /dev/null" #
      volumes:
        - ./pyinterface:/Interface
      restart: unless-stopped
      ports:
        - "5020:5020"
      networks:
        ics:
          ipv4_address: 192.168.192.2
```

```yaml
wwtpmodel: #Contiene el programa de Matlab/Simulink
  image: sflorenz05/matlab_simulink:v0.3
  container_name: wwtpmodel
  shm_size: 512M
  ports:
    - "5901:5901"
    - "6080:6080"
  command: -vnc
  volumes:
    - ./wwtpmodelmodbus:/home/matlab/Documents/MATLAB/wwtp
  restart: unless-stopped
  networks:
    ics:
      ipv4_address: 192.168.192.4
```

```yaml
client1:
  image: oihan27/tcpclient:performance #Ejecuta performance.py
  container_name: client1
  build:
    context: ./Clients
  restart: unless-stopped
  networks:
    ics:
      ipv4_address: 192.168.192.5
  environment:
    - AUTO_SERVER_HOST=10.63.28.53
```

Interface Deployment

```
Write Sensor Data to PLC: [4, 12, 0, 15]
Receiving Sensors and Actuator Data [4, 12, 0, 15, 0, 25000, 106, 100]
 Oxygen:  4
 Nitrates:  12
 Ammonium:  0
 Temperature:  15
 Aeration:  25000
 Internal Recirculation:  106
 Waste Sludge Flow:  100

Write Actuator Data to virtual server: [25000, 106, 100]
Receiving Sensors and Actuator Data [4, 12, 0, 15, 0, 25000, 106, 100]
 Oxygen:  4
 Nitrates:  12
 Ammonium:  0
 Temperature:  15
 Aeration:  25000
 Internal Recirculation:  106
 Waste Sludge Flow:  100
```

05 { ..

Results

} ..

Network Performance Analysis

Goals:

- Calculate latency
  - Low latency = higher efficiency
  - Delays/spikes can indicate a cyberattack
- Scalability
  - Test program’s ability to withstand increasing loads

Network Performance Analysis

```
[DEBUG/MainProcess] 451101 requests/second
[DEBUG/MainProcess] time taken to complete 1000 cycle by 1 workers is 0.0022167970018927 seconds
[DEBUG/MainProcess] Maximun time taken by 1000 cycles is 3.0064757250074763 seconds
[DEBUG/MainProcess] Minimun time taken by 1000 cycles is 0.0 seconds
[DEBUG/MainProcess] Average time of all 1000 cycles is 0.72460620131678884 seconds
```

Results

Client	Time Taken to Complete 1000 Cycles by One Client (s)	Minimum Time to Complete 1000 Cycles by One Client (s)	Maximum Time to Complete 1000 Cycles by One Client (s)	Average (s)
1	0.00242	0.0	0.00978	0.00236
2	0.00190	0.0	0.00956	0.00237
3	0.00212	0.0	0.00980	0.00234
4	0.00194	0.0	0.01044	0.00220

06 { .. Conclusion & Future Work

} ..

Conclusion & Future Work

Based on Results...

- The interface can handle multiple concurrent Modbus clients
- Consistent latency

Future Work

- Increase the number of clients deployed to find max
- Evaluate more performance metrics

References