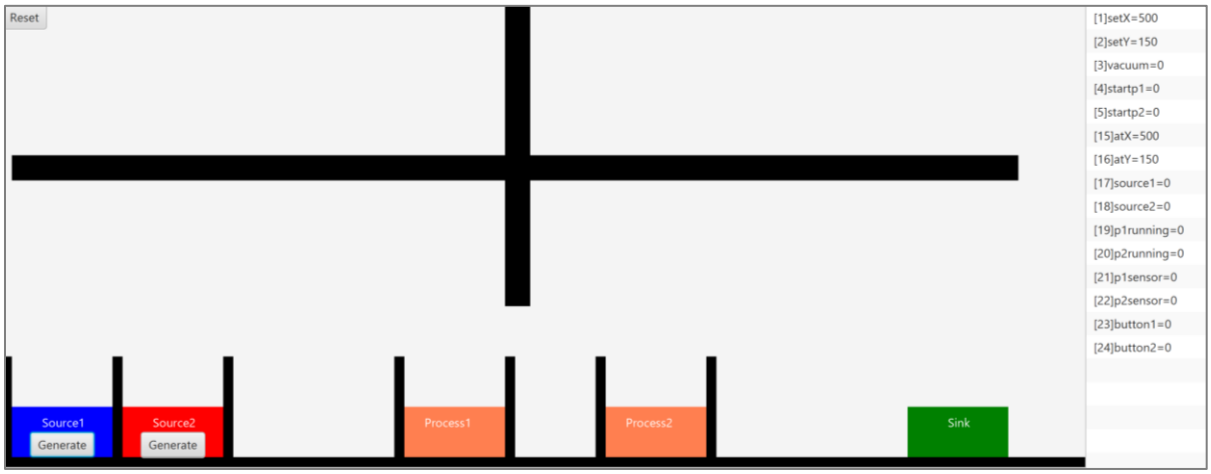
Automation Project

In the context of the project, we will engage with the crane simulation, visually represented in the figure below. The simulation incorporates two source stations for adding products to the system, a pair of processing units, a sink for products to leave the system, and a crane for moving products. To the right of the simulation visualization, a panel detailing variables is presented.



Installing Required Libraries

You'll need the pymodbus library for Modbus communication and Pandas for JSON file.

Open your command prompt or terminal and run:

pip install pymodbus

pip install pandas

Start the simulation

Download the CraneSimulation.zip and unzip the folder. Locate and run the file simulation.exe. We have only tested this on Windows 11. However, if you have another operative system, you may try the jar file instead.

If you run on Mac you can also access Windows 11 computers online at University West website:

https://www.hv.se/en/student/it-services-and-support/virtual-computer-rooms/

You will be randomly assigned to a computer so don't save your files on them, save it on network drive or cloud.

Setting Up the Modbus TCP Client

Create a new Python script and start by importing the necessary libraries and setting up the Modbus client.

```
import json
import pandas as pd
from pymodbus.client import ModbusTcpClient
from pymodbus.exceptions import ModbusException
import time
```

Then you need to connect to the Modbus client:

```
client = ModbusTcpClient('127.0.0.1')
```

Defining Modbus Addresses and Variables

You will interact with various resources in the simulation, each identified by specific Modbus addresses.

Reading from and writing to Modbus Registers

Create functions to read from and write to Modbus registers.

For example, the following function can read inputs from a certain address:

```
def read_input(address):
    result = client.read_holding_registers(address, 1)
    return result.registers[0]
```

The following function can write to an address with a certain value:

```
def write_output(address, value):
    result = client.write_register(address, value)
    print(f"Successfully wrote {value} to address {address}")
```

Modbus addresses

The following table shows all Modbus addresses in the simulation.

Resource	Signal Name	Address	I/O
Source 1	sensor	17	Input
Source 2	sensor	18	Input
Process 1	run	4	Output
	sensor	21	Input
	isRunning	19	Input
	reset	0	Output
Process 2	run	5	Output
	sensor	22	Input
	isRunning	20	Input
	reset	0	Output
Crane	setX	1	Output

Resource	Signal Name	Address	I/O
	setY	2	Output
	posX	15	Input
	posY	16	Input
	vacuum	3	Output

Modbus client connect and close

You must connect and close the Modbus connection. Consider that you have a function `execute_commands_from_json` to take the instructions from JSON and send them to the simulation via Modbus, then you can do it like the following example to properly open and close the connection:

```
if __name__ == "__main__":
    client.connect()
    json_file = 'crane_commands.json'
    execute_commands_from_json(json_file)
    client.close()
```

Controlling the Crane

You should write code to control the crane's movements by setting its X and Y positions and activating the vacuum.

Considering the table, the following would turn on the vacuum, meaning the part will attach to the crane:

```
write_output(3, 1)
```

And the following will release the part by turning off the vacuum:

```
write_output(3, 0)
```

You can also tell the crane to move to for example the location x=55 and y=44 with the following code:

```
write_output(1, 55)
write_output(2, 44)
```

where address 1 is “setX” and address 2 is “setY”.

To read the position of the crane we can use the “posX” and “posY” with the following code:

```
crane_atX = read_input(15)
crane_atY = read_input(16)
```

All other resources can be controlled with similar commands by looking in the table of addresses.

Reading the JSON with crane instructions

Implement the functionality to read commands from the JSON file. Parse the JSON data to extract commands and execute them using the functions you've created.

You can import the data from the file with this command:

```
def execute_commands_from_json(json_file):
    with open(json_file, 'r') as file:
        data = json.load(file)
    df = pd.DataFrame(data['actions'])
```

Here you need to create code logic to find the values of x, y, vacuum and send them via Modbus to the simulation of the crane.

Project requirements

Now, extend the function execute_commands_from_json and write code to iterate over the data from the JSON file and find the “setX” and “setY” together with vacuum state for each set of actions.

The behaviour of the system should be:

- You press the *Generate* button on *Source 1* in the simulation
- You start your Python code
- The crane reads the JSON file and executes all commands.

Note that not all variables exist on each group in the JSON so you might need to check if it exists, or you might get an error if trying to use it. For example, in the first group, only vacuum=0 exists, no variable setX and setY exist.

Also, remember to convert x, y, vacuum to integers after importing from file to get them in the correct type.

If you send a position setX and setY when the crane is running, it will interrupt the crane motion and go to the new location directly. This can cause collisions since the correct path is not taken. To solve this problem, you can use the posX and posY variables to check if the crane has arrived at the last position you sent. Another less professional approach is to add a simple delay to ensure that the crane always has time to go to positions. However, this will slow down the manufacturing system and it will not work if the crane path in the future will change to be longer or if the crane run slower, since that might risk the delay to not be long enough.

JSON file with instructions for crane

Name of the file is “crane_commands.json”.

This file instructs your code to:

- Turn off vacuum
- Move to location above Source 1
- Move down to Source 1
- Turn on vacuum
- Move up
- Move to location above Sink
- Move down to Sink
- Turn off vacuum
- Move up

The file content is the following:


```
{
 "actions": [
  {
   "vacuum": 0
  },
  {
   "setX": 55,
   "setY": 200
  },
  {
   "setX": 55,
   "setY": 82
  },
  {
   "vacuum": 1
  },
  {
   "setX": 55,
   "setY": 200
  },
  {
   "setX": 945,
   "setY": 200
  },
  {
   "setX": 945,
   "setY": 82
 },
 {
   "vacuum": 0
  },
  {
   "setX": 945,
   "setY": 200
  }
 ]
}
```