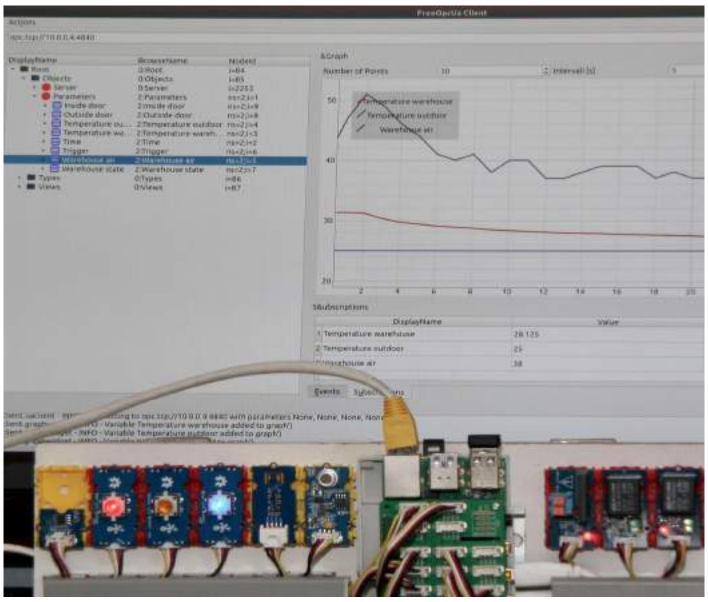
Smart Industry – Zelf aan de Slag (SIZAS Pi-VNC)



Summary need devices, software & network..........2
Before you start with the assignments..........3
Assignment 1 Get remote access to your Pi and write a simple Python program..........4
Assignment 2 Data collection & remote control using OPC-UA..........7
Assignment 3: OPC-UA server behind a firewall and client outside..........12

VNC version

This version of the “Smart Industry cyber security - Zelf Aan de Slag” (SIZAS-VNC) course is meant for courses with a Pi without a monitor/keyboard/mouse. You will use a remote MS-windows interface using VNC and connect your PC to the Pi of your group using the Wi-Fi “smart” network.

The work was made possible by Min EZK, provincie NoordBrabant and TNO.
This material is planned to submitted by the Smart Industry program (www.smartindustry.nl) into open source.

Summary need devices, software & nework

Notebook with MS-Windows (or MacOS or Ubuntu)

The participants notebook will be running the *vncviewer* and you need to install *Python3* & the *opcua* library. Note that in case you have a company notebook, you must be allowed as user to install software. If not get (travel) notebook from your company on which you can install own software or use your private machine.

The Raspberry Pi's are provided by the course. It is a standard Pi 3+ (Model B) with SEEED 'Grove Hat board for Pi' with 3 LED switches, 3 relays and a high accuracy temperature sensor. (www.seeedstudio.com with info at the seeed wiki forum)

Wi-Fi network name: smart password: industry (a WPA2 password)
10.0.0.0/24 (address range 10.0.0.0-10.0.0.255, (8 bits (24=32 bit IP number – 8 bits))
10.0.0.1 is gateway router
10.0.0.2 is RevPi (core 1)
10.0.0.4 is Pi-4, the demo course Pi (model 4 with 2GB RAM) with Grove I/O + OPC-server
10.0.0.x (with x 11-16) IP nr of your Pi: for nr see sticker at back/bottom of Pi
10.0.0.x with 20 < x <100 is the your computer's IP number provided by DHCP.
10.0.0.253 firewall 2 (5-port gray hEXs) with 8 port ethernet switch for extra
10.0.0.254 firewall 1 (5-port white hEX next to RevPi-3)

Some Unix command (usable in command line mode or in a terminal box)

sudo poweroff	to poweroff a Pi (and wait till lights do not flicker anymore)
ip addr	to see the IP number of the Pi, e.g. 10.0.0.11
startx	to switch from command line mode to Raspbian Windows
ls -als	to show the directory listing
passwd	to change your password

Download or copy from USB stick or email with the surf transfer location

- python3 from www.python.org NOTE: **click on checkbox to set PATH** immediately during the first installation screen
- python library opcua using pip/pip3 (pip is part of the Python environment)
- vncviewer.exe from www.realvnc.com
- (opt.) openssh on windows 10 or downloadputty.exe from www.putty.org
- https://github.com/ejsol/Smart-Industry-zelf-aan-de-slag.git
- (opt.) UaExpert.exe from www.united-automation.com for observing OPC-UA

Before you start with the assignments

The software should be downloaded from Internet and installed (opt. from USB) before you switch to the workshop Wi-Fi smart. The Wi-Fi 10.0.0.0/24 network is not connected to the Internet, so once on smart you can't download software any more. Most programs will also be provided during the workshop on an USB stick and are in the SIZAS directory of the Pi's (where you can copy them on your own USB). For one piece (the opcua Python library, see below) you need internet access. The best moment to do is to preinstall software in advance at home.

To install the free opcua on your notebook you must first install a Python environment. (See annex if you want to install WinPython instead of Python). Download Python from www.python.org/downloads and select the latest version (*python-3.7.3-amd6.exe* or higher for MS-windows on AMD or Intel processors). You get the IDLE for Python. IDLE is the integrated development environment for Python. **Notice that you must click ON to include python in your path (bottom menu in opening screen).**

For our assignments we make use of the opcua library package which you need to install in the Python environment. Open a terminal window with the command prompt and type: *pip3 install opcua –user* Or *C:\Program Files\Python37\Scripts\pip3 install opcua --user*

Background information:

> Pip is the Python Installation of Packages. In general Python packages come in python source code, interesting at debugging. The pypi.org is the best place to find python packages and their info. There are many opcua packages, but we use the free opcua package (now 0.98.7) from oroulet (Olivier Roulet-Dubonnet).

While preparing for the course, there is more software you might install in advance to.

Each Raspbian/Pi has a built-in VNC server. You need the free VNC viewer on your PC to access it. Install *vncviewer* (www.realvnc.com (of use the USB stick to install it)). VNC is a windows interface to a Pi running Raspbian-windows. If you want a command line interface (prompt) to your Pi you can use the Unix SSH (secure shield) or the MS-windows PuTTy to access your Pi. Since 2018 *ssh* is part of MS-windows 10, but you need to active it. Use setting, apps and feature, optional features and then OpenSSH. In that case PuTTy is not needed, else you need to download & install PuTTY from putty.org.

Finally, download the Python software source programs for the workshop on your computer using GitHub. Go to https://github.com and do a search for ejsol/Smart-Industry-zelf-aan-de-slag. Download the ZIP and extract all. If already interested, have a look at the Python code that will be running on the Pi's (and the demo with industrial PLC Pi, the RevPi).

Optional is the program opcua-client (a pip python library same as the opcua library, but read the annex) and/or UaExpert.exe from www.united-automation.com for observing OPC-UA. With opcua-client you have to start the program with *python3 app.py.* You can download uaexpert after registration from their website or copy it from the USB stick.

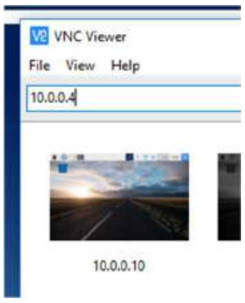
Assignment 1 Get remote access to your Pi and write a simple Python program

You can use a Pi with a monitor/keyboard/mouse and work in command line mode (terminal mode also called CLI command line interface of prompt) or startup the Pi in window mode (also called Desktop mode) with *$ startx*. However, you can also use your own computer and access the Pi from remote in terminal or in windows mode.
[In this version of the document (SIZAS-VNC) we assume this last (remote/windows-VNC) option]

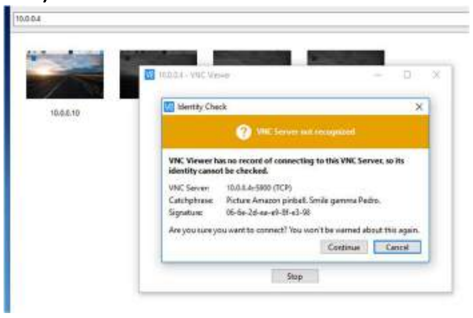
In this (remote) assignment you need to know the IP number of your Pi (10.0.0.x x=11-16).

Open your VNC viewer on your PC and connect to your Pi

1)



2)

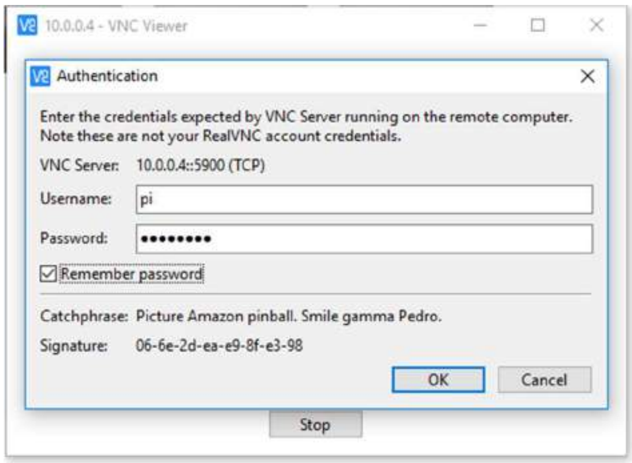


VNC Viewer has no record of connecting to this VNC Server, so its identity cannot be checked.

VNC Server: 10.0.0.4::5900 (TCP)
Catchphrase: Picture Amazon pinball. Smile gamma Pedro.
Signature: 06-6e-2d-ea-e9-8f-e3-98

Are you sure you want to connect? You won't be warned about this again.

Now you can login with the username: Pi and password: industry



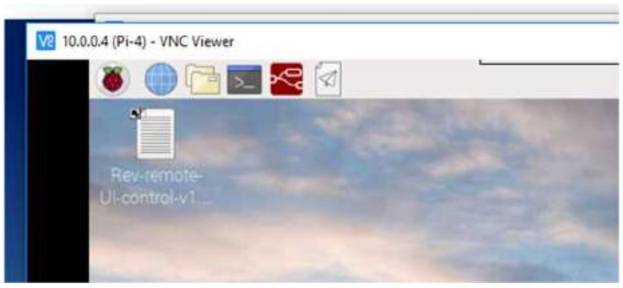
Enter the credentials expected by VNC Server running on the remote computer.
Note these are not your RealVNC account credentials.

VNC Server: 10.0.0.4::5900 (TCP)
Username: pi
Password: ••••••••
☑ Remember password

Catchphrase: Picture Amazon pinball. Smile gamma Pedro.
Signature: 06-6e-2d-ea-e9-8f-e3-98

If it goes right, you get a window with Raspbian, the Unix-windows version on the Pi, if not then the Pi has not booted in windows mode. See end note how to solve this situation.[1]

[1] If VNCview replies that your Pi is not in windows mode, than start a terminal window (command prompt mode of windows), type *ssh pi@10.0.0.x* to the Pi, login (password industry) and type *sudo raspi-config* and select menu option 3 boot mode and select B1 Desktop/CLI (windows/command line mode), then select B3 Desktop, tab to finish and reboot.



If the screen is very small, select the Raspberry icon, go to Preferences (NL: Voorkeuren) and then to appearance settings and select default for small screens.

Now you can select the menu options.

Some general instruction regarding Unix systems: do not turn-off the power supply of any Unix system (and this Pi) by just pulling the plug out. If possible, issue a power-off in a controlled way and wait till the disk stops (at the Pi the yellow light stops blinking).

You can power-off with the raspberry menu (the first) and then select the last option.
You can also use the terminal command prompt $ *sudo poweroff*
Wait a few seconds till the yellow light stops blinking (no SD-card access anymore) and then unplug the power supply

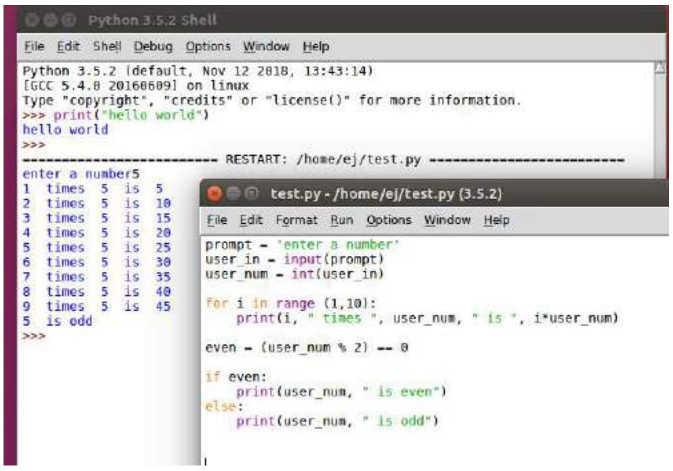
---

Open the terminal command prompt the black icon with blue top bar and the >_

You are now at the command prompt such as Pi-11$
Enter e.g. after the $ *ip addr* (to see your IP number 10.0.0.x)
$ *ls -als* (for a file directory listing)
And the command $ *sudo poweroff*

Now go to the raspberry icon and select programming and then select Python3 IDLE

And enter at the >>> print (“Hello world”)


Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
test.py - /home/ej/test.py (3.5.2)
File Edit Format Run Options Window Help

```
Python 3.5.2 (default, Nov 12 2018, 13:43:14)
[GCC 5.4.0 20160609] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print("hello world")
hello world
>>>
======================== RESTART: /home/ej/test.py ========================
enter a number5
1  times  5  is  5
2  times  5  is  10
3  times  5  is  15
4  times  5  is  20
5  times  5  is  25
6  times  5  is  30
7  times  5  is  35
8  times  5  is  40
9  times  5  is  45
5  is odd
>>>
```

```
prompt = 'enter a number'
user_in = input(prompt)
user_num = int(user_in)

for i in range (1,10):
    print(i, " times ", user_num, " is ", i*user_num)

even = (user_num % 2) == 0

if even:
    print(user_num, " is even")
else:
    print(user_num, " is odd")
```

Select at the Python shell the File option, open a new file and start to enter the above Python program test.py.

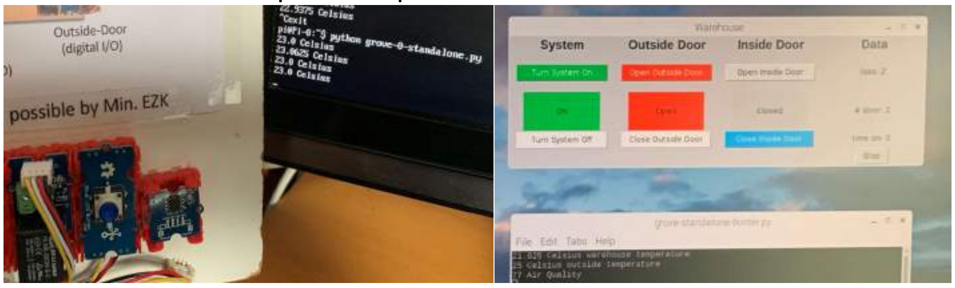
```
prompt = 'enter a number'
user_in = input(prompt)
user_num = int(user_in)

for i in range (1,10):
   print(i, " times ", user_num, " is ", i*user_num)

even = (user_num % 2) == 0

if even:
   print(user_num, " is even")
else:
   print(user_num, " is odd")
```

Once ready, select the 4th menu option Run and see what happens.

It is possible to write and run the same program with Python on your own computer.
Just start the Python IDLE and enter the same program, ;-)

In case you are ready and waiting for the course to continue:

Using the VNCviewer you are working already on the Pi. Check if the Pi/0-grove-cli.py (the command line cli) and the 1-grove-windows.py are on the /home/pi/Desktop or /home/pi/SIZAS/Pi subdirectory. If not, copied them from the USB stick or use the VNC send file function in the top bar dropdown menu.



```
22.9375 Celsius
^Cexit
pi@Pi-0:~$ python grove-0-standalone.py
23.0 Celsius
23.0625 Celsius
23.0 Celsius
23.0 Celsius
```

```
Celsius warehouse temperature
Celsius outside temperature
Air Quality
```

Figure 1 0-grove and 1-grove..py output screens

You can execute the 0-grove-cli.py and the 1-grove-windows.py by clicking on the icons at the desktop or bij entering after the command prompt: …

$ *cd /home/pi/Desktop*   [or shorter *cd Desktop* if you at the home directory of pi]
$ *python3 grove-0-cli.py* and see what happens.

On the terminal you see the temperature value appear, but you with 2-grove… can now click on the buttons to see the logic control in operation. If selecting your program in the Raspbian windows environment by clicking on the icon, then select open (the third option). Now you are in debugger mode.

Assignment 2 Data collection & remote control using OPC-UA

Install (copy from USB or use VNC file transfer) the 2x-grove-….py programs to your Pi and execute them on the Pi (in the windows interface, open a terminal and type after the $ prompt: *python3 0-grove-cli.py* and *python3 1-grove-windows.py.* If nothing shows, you are probably not in the /home/pi/Desktop directory (you need to type *cd /home/pi/Desktop* first)

*(*If you want you can make the programs self-executable: use the file manager, select file, go to properties and make them executable. (In command prompt *chmod +x file.py* , you can check it using the command *ls -als).* The programs on the Pi are already executable and clicking on them results in windows with several options: select the open button.)

Notice that in the 1-grove-windows program there is already some data collection and analysis. Under the last column, named Data, you see how often the door is opened, but also how long it is kept open, the longer it is, the more energy you lose, but with each time a maximum. So, if you keep a door open for a long time, the energy loss is caped.

You have seen the program in operation, running on the Pi using VNC from your notebook/PC. Now we go over to using OPC where the control remains at the Pi, the server Python OPC program and the OPC-client is your user interfacing at your notebook.

As OPC server Pi with the logic control and the data collection we use the the program *20-grove-opc-server.py*. The best way to start this program is to start it in a terminal box and at the prompt type $ *python3 20-grove-opc-server.py*

In the client file, you need to change the url from 0.0.0.0 (localhost) into your Pi-1x number, e.g. Pi-11 becomes 10.0.0.11. You need to do this with the client file in the Python IDLE environment at your notebook, the client. (However, there is a little nicety you can use. The 21-grove.. client program can run on your Pi as local client too. Once the servers is running in it own terminal mode, click on the 21-grove… icon and open it. Now go back to your own PC and edit the Pi-1x IP number in the opc.tcp url line)

Start the server program on your Pi-1x first. (using the remote VNC session running on the Pi from your own computer). It will take some time before the server is ready and the first sensor data is printed. The client program *grove-2-opc-client.py* is kept as simple as possible, but it can be extended with storing the data collection, visualizing it and analyzing it. Now it only dumps every 5 seconds the sensor data in a data-base ready format. More notebook PCs from your group can start the client version and each can monitor what the server is outputting.

Instead of asking the OPC server running on the Pi with our own (very, very simple) client program, one can also use OPC-viewers as UaExpert.exe from www.unified-automation.com. (These programs are writing and compiled in C or C++). With UaExpert as client one can dive into the whole object structure of an OPC server. (Or see annex).

Abstract from 0-grove -cli.py:

```python
#!/usr/bin/python3
# -*- coding: utf-8 -*-
#
# (c) EJSol 12 sept 2019 freeware for use in Smart Industry - Zelf Aan de Slag workshop (SIZAS)
#
# standalone, very simple, no user interface,
# start from command line, control the doors and log the accurate warehouse temperature
# this version -1x- is a little different in amount of grove sensors (only one high accuracy temperature, not three)

import time
from datetime import datetime
from grove.button import Button
from grove.factory import Factory
from grove.temperature import Temper
from grove.gpio import GPIO

class GroveRelay(GPIO):
    def __init__(self, pin):
        super(GroveRelay, self).__init__(pin, GPIO.OUT)

    def on(self):
        self.write(1)

    def off(self):
        self.write(0)

class GroveLedButton(object):
    def __init__(self, pin):
        # Low = pressed
        self.__led = Factory.getOneLed("GPIO-HIGH", pin)
        self.__btn = Factory.getButton("GPIO-LOW", pin + 1)
        self.__led.light(False)
        self.__on_release = None
        self.__on_press = None
        self.__btn.on_event(self, GroveLedButton.__handle_event)

    @property
    def on_press(self):
        return self.__on_press

    @on_press.setter
    def on_press(self, callback):
        if not callable(callback):
            return
        self.__on_press = callback

    def __handle_event(self, evt):

        self.__led.brightness = self.__led.MAX_BRIGHT

        if evt["code"] == Button.EV_LEVEL_CHANGED:
            if evt["pressed"]:
                if callable(self.__on_press):
                    self.__on_press()

    def led_on(self):
        self.__led.light(True)

    def led_off(self):
        self.__led.light(False)


Grove = GroveLedButton

main_state = False
door_1_state = False
door_2_state = False

def main():
    global main_state, door_1_state, door_2_state

    main_button = GroveLedButton(5)
    door_1_button = GroveLedButton(18)
    door_2_button = GroveLedButton(16)

    main_relay = GroveRelay(22)
    door_1_relay = GroveRelay(26)
    door_2_relay = GroveRelay(24)

    sensor = Factory.getTemper("MCP9808-I2C")
    sensor.resolution(Temper.RES_1_16_CELSIUS)

    print('Time Temperature (C)')

    def on_press_main():
        global main_state, door_1_state, door_2_state
        if main_state:
            main_state = False
            door_1_state = False
            door_2_state = False
            main_relay.off()
            door_1_relay.off()
            door_2_relay.off()
            main_button.led_off()
            door_1_button.led_off()
            door_2_button.led_off()
        else:
            main_state = True
            main_relay.on()
            main_button.led_on()

    def on_press_1():
        global main_state, door_1_state
        if main_state:
            if door_1_state:
                door_1_state = False
                door_1_relay.off()
                door_1_button.led_off()
            else:
                if not door_2_state:
                    door_1_state = True
                    door_1_relay.on()
                    door_1_button.led_on()

    def on_press_2():
        global main_state, door_2_state
        if main_state:
            if door_2_state:
                door_2_state = False
                door_2_relay.off()
                door_2_button.led_off()
            else:
                if not door_1_state:
                    door_2_state = True
                    door_2_relay.on()
                    door_2_button.led_on()

    main_button.on_press = on_press_main
    door_1_button.on_press = on_press_1
    door_2_button.on_press = on_press_2

    while True:
        try:
            time.sleep(1)
            print('{} '.format(datetime.now().strftime("%X")), '{}'.format(sensor.temperature))
        except KeyboardInterrupt:
            main_relay.off()
            door_1_relay.off()
            door_2_relay.off()
            main_button.led_off()
            door_1_button.led_off()
            door_2_button.led_off()
            print("exit")
            exit(1)

if __name__ == '__main__':
    main()
```

Abstract from 20-grove-opc-server.py:

(note: the first line from opcua import Client, this import is the reason why you needed to install that library on the Python environment on your PC. On the Pi the opcua and the grove.py libraries are needed. The library is in source code.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
```

```
#
# (c) EJSol 12 sept 2019 freeware for use in Smart Industry - Zelf Aan de Slag workshop (SIZAS)
#
# app running control I/O for doors and collecting temperature data,
# start from command line and use from other computer opc client to read data from this opc server
#
# the source is for the Pi-4 (10.0.0.4) with three sensors,
# in this version the outside temperature and air quality are not used
# as it is meant for the instruction Pi-11 Pi-16 with only digital I/O and 1 high accuracy temperature sensor
#
import time
from datetime import datetime

from opcua import Server

… as above .. (imports and GroveRelay and GroveLEDButton) …

class MyGroveOpcTerminalApp:

    def __init__(self):
        self.warehouse_state = False
        self.door_outside_state = False
        self.door_inside_state = False

        self.warehouse_button = GroveLedButton(5)
        self.door_outside_button = GroveLedButton(18)
        self.door_inside_button = GroveLedButton(16)

        self.warehouse_button.on_press = self.on_press_main
        self.door_outside_button.on_press = self.on_press_door_outside
        self.door_inside_button.on_press = self.on_press_door_inside

        self.warehouse_relay = GroveRelay(22)
        self.door_outside_relay = GroveRelay(26)
        self.door_inside_relay = GroveRelay(24)

        self.time_stamp = datetime.now()
        self.temperature_warehouse = Factory.getTemper("MCP9808-I2C")
        self.temperature_warehouse.resolution(Temper.RES_1_16_CELSIUS)

        print('starting OPC server ')
        self.opc_server = Server()
        self.opc_url = "opc.tcp://0.0.0.0:4840"
        self.opc_server.set_endpoint(self.opc_url)

        print('starting OPC server ..')
        self.opc_name = "Grove-opcua-server"
        self.addspace = self.opc_server.register_namespace(self.opc_name)
        print('starting OPC server ...')

        self.opc_node = self.opc_server.get_objects_node()
        self.param = self.opc_node.add_object(self.addspace, "Parameters")

        self.opc_time = self.param.add_variable(self.addspace, "Time", 0)
        self.opc_trigger = self.param.add_variable(self.addspace, "Trigger", 0)
        self.opc_warehouse_state = self.param.add_variable(self.addspace, "Warehouse state", 0)
        self.opc_door_outside = self.param.add_variable(self.addspace, "Outside door", 0)
        self.opc_door_inside = self.param.add_variable(self.addspace, "Inside door", 0)
        self.opc_temperature_w = self.param.add_variable(self.addspace, "Temperature warehouse", 0.0)

        self.opc_time.set_writable()
        self.opc_trigger.set_writable()
        self.opc_warehouse_state.set_writable()
        self.opc_door_outside.set_writable()
        self.opc_door_inside.set_writable()
        self.opc_temperature_w.set_writable()

        print('starting OPC server .....')
        self.opc_server.start()
        print("OPC UA Server started at {}".format(self.opc_url))
        print("time    Celsius warehouse")

    def closeapp(self):
        self.warehouse_relay.off()
        self.door_outside_relay.off()
        self.door_inside_relay.off()
        self.warehouse_button.led_off()
        self.door_outside_button.led_off()
        self.door_inside_button.led_off()
        self.opc_server.stop()
        print("exit")
        # self.master.destroy()
        exit(1)

    def update_opc(self, trigger):
        self.time_stamp = datetime.now()
        self.opc_time.set_value(self.time_stamp)
        self.opc_temperature_w.set_value(self.temperature_warehouse.temperature)
        print('{} '.format(self.time_stamp.strftime("%X")), '{}'.format(self.temperature_warehouse.temperature))
        self.opc_trigger.set_value(trigger)
        self.opc_warehouse_state.set_value(self.warehouse_state)
        self.opc_door_outside.set_value(self.door_outside_state)
        self.opc_door_inside.set_value(self.door_inside_state)

    def on_press_main(self):
        if self.warehouse_state:
            self.warehouse_state = False
            self.door_outside_state = False
            self.door_inside_state = False
            self.warehouse_relay.off()
            self.door_outside_relay.off()
            self.door_inside_relay.off()
            self.warehouse_button.led_off()
            self.door_outside_button.led_off()
            self.door_inside_button.led_off()
        else:
            self.warehouse_state = True
            self.warehouse_relay.on()
            self.warehouse_button.led_on()
        self.update_opc(1)

    def on_press_door_outside(self):
        if self.warehouse_state:
            if self.door_outside_state:
                self.door_outside_state = False
                self.door_outside_relay.off()
                self.door_outside_button.led_off()
            else:
                if not self.door_inside_state:
                    self.door_outside_state = True
                    self.door_outside_relay.on()
                    self.door_outside_button.led_on()
        self.update_opc(2)

    def on_press_door_inside(self):
        if self.warehouse_state:
            if self.door_inside_state:
                self.door_inside_state = False
                self.door_inside_relay.off()
                self.door_inside_button.led_off()
            else:
                if not self.door_outside_state:
                    self.door_inside_state = True
                    self.door_inside_relay.on()
                    self.door_inside_button.led_on()
        self.update_opc(3)

    # start functie is voor terminal-mode, niet voor windows mode
    def start(self):
        """Start event system and own cyclic loop."""

        while True:
            try:
                # dirty temp (client changes every 5 sec opc_warehouse_state)
                if self.opc_warehouse_state.get_value():
                    self.warehouse_button.led_on()
                else:
                    self.warehouse_button.led_off()
                time.sleep(5)
                self.update_opc(0)
            except KeyboardInterrupt:
                self.closeapp()


if __name__ == '__main__':
    myapp = MyGroveOpcTerminalApp()
    myapp.start()
```

and the client program 21-grove-opc-client.py

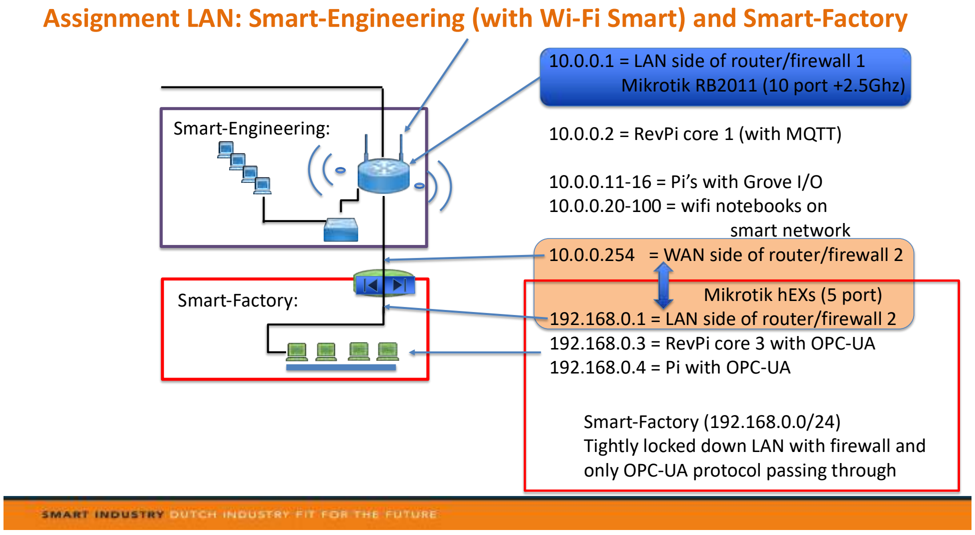
```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
#
# (c) EJSol 12 sept 2019 freeware for use in Smart Industry - Zelf Aan de Slag workshop (SIZAS)
#
# the source is for the Pi-4  with three sensors,
# in this version for the instruction Pi-11 till Pi-16 the outside temperature and air quality are not used
# as the Pi-11 Pi-16 have only digital I/O and 1 high accuracy temperature sensor
#
# for the instruction Pi you need to change in line 16 the IP number from 10.0.0.10 to 10.0.0.1x with 1<x<6
# and if Pi is behind firewall then call 10.0.0.254:54844 in case of Pi-4 (of else 54840+1x)

from opcua import Client
import time

url = "opc.tcp://localhost:4840"
# option 1: localhost if run on the Pi itself, you can write opc.tcp://localhost:4840"
# option 2: 192-net url= "opc.tcp://192.168.0.4 if the Pi=192.168.0.4 and the client is on another node in 192.168.0.0/24
# option 3: 10-net with firewall url ="opc.tcp://10.0.0.254:54840+your pi nr, e.g. Pi-11 goes to 54851
# which is translated into 192.168.0.4:4840 by the firewall/router DSTNAT

client = Client(url)
client.connect()
print("Client is connected")
print("                                    T = temperature in Celsius")
print("time     trigger  warehouse-state outside-door  inside-door T-warehouse")

while True:
   try:
      temperature_time = client.get_node("ns=2;i=2")
      trigger = client.get_node("ns=2;i=3")
      warehouse_state = client.get_node("ns=2;i=4")
      door_outside = client.get_node("ns=2;i=5")
      door_inside = client.get_node("ns=2;i=6")
      temperature_warehouse = client.get_node("ns=2;i=7")

      print('{} '.format(temperature_time.get_value().strftime("%X")), "  ", trigger.get_value(), "     ",
          int(warehouse_state.get_value()), "          ", int(door_outside.get_value()), "           ",
          int(door_inside.get_value()), "         ", temperature_warehouse.get_value())
      time.sleep(5)
   except KeyboardInterrupt:
      client.disconnect()
      exit(1)
```

Assignment 3: OPC-UA server behind a firewall and client outside

The OPC-UA server will often run on a machine in a highly secured own network segment, after a tightly locked down firewall e.g. Smart-Factory in the figure below. The client could be running outside this network, as in the Smart-Engineering network from the figure below. E.g. the Pi with the Groove I/O and the OPC-UA server run in their own smart-factory e.g. 192.168.0.0/24 network wired to the router/firewall on the LAN site and on the WAN site the router/firewall is in a e.g. 10.0.0.0/24 smart-engineering office network (with Wi-Fi to notebooks). Now the client cannot access the OPC-UA server.

In this assignment no Python program is written, but we need to configure the firewall and add specific parameters in the Python client program and change the line with url = “ocp.tcp://…..”. From the client side the server is not visible, only the WAN port of the firewall e.g. 10.0.0.253. And if you have multiple OPC-UA servers (more Pi’s e.g. 192.168.0.4 & 192.168.0.11) how to access them if you only have one access point?

The common way of working is to use so-called Destination NAT (network address translation). Inside the LAN part a client e.g. on a notebook wired to the router and having an 192.168.0.2x IP address can use the IP addresses and the OPC-UA port of 4840. Now the firewall-LAN side, e.g. 192.168.0.1 can also act as client to the two Pi's. In the firewall-LAN DstNAT is now configured such that if a message from received on IP nr (firewall) 10.0.0.254:54844 (or any other portnumber above 1024 and below 64K, but preferable between 50K-64K)) is mapped to 192.168.0.4:4840 and similar to port 54844 is mapped to 192.168.0.11:4840. This way a request from the client is translated by the firewall and forwarded to the proper Pi and during the answer from the Pi, the firewall knows to with client the answer should be sent.


Assignment LAN: Smart-Engineering (with Wi-Fi Smart) and Smart-Factory
10.0.0.1 = LAN side of router/firewall 1
Mikrotik RB2011 (10 port +2.5Ghz)
Smart-Engineering:
10.0.0.2 = RevPi core 1 (with MQTT)
10.0.0.11-16 = Pi's with Grove I/O
10.0.0.20-100 = wifi notebooks on smart network
10.0.0.254 = WAN side of router/firewall 2
Mikrotik hEXs (5 port)
Smart-Factory:
192.168.0.1 = LAN side of router/firewall 2
192.168.0.3 = RevPi core 3 with OPC-UA
192.168.0.4 = Pi with OPC-UA
Smart-Factory (192.168.0.0/24)
Tightly locked down LAN with firewall and only OPC-UA protocol passing through
SMART INDUSTRY DUTCH INDUSTRY FIT FOR THE FUTURE

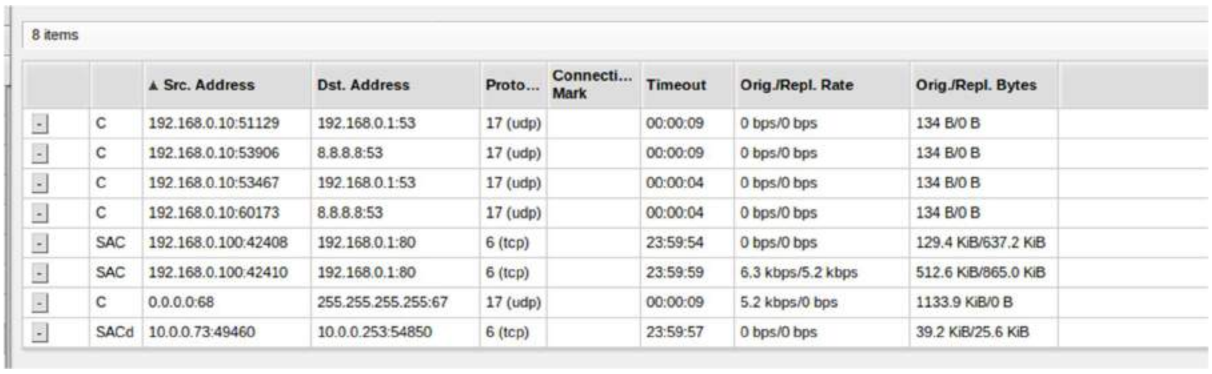
Once you move your 10.0.0.1x Pi-1x to the 192.168.0.0/24 network disable Wi-Fi and wired the UTP/ethernet port of your Pi into a four ports ether2-ether5 of the router (hEXs). The server program creates a server at 192.168.0.1x and that line in the code need to be changed to 192.168.0.1x. In the client program uses the 10.0.0.1x address, but that now need to change to (10.0.0.254 if on firewall 1 else firewall 2) at 10.0.0.253:548yx with yx 40+1x (51-56).

It is a common way of addressing a device behind a firewall also used with webservers behind firewall. The only thing to do is to configure the firewall to accept DstNAT and enter the translations in the firewall rules set of the firewall. (see training slides for more details too). Notice that we disabled ScrNAT (so no traffic can go outside the LAN)

```
/ip firewall filter
# chain input (to router itself)
add action=drop     chain=input comment="drop invalid to firewall router at 192.168.0.1/24"  connection-state=invalid
add action=accept   chain=input comment="allow established connections to firewall router " connection-state=established

# allow connection to firewall router from local network (!ether1 implies ether2-5 (=LAN) as ether1 is WAN),
# so next rules state accept all local LAN traffic, drop all remain WAN traffic
add action=accept   chain=input comment="" in-interface=!ether1 src-address=192.168.0.0/24
add action=drop     chain=input comment="drop all to firewall router not coming from LAN" in-interface=ether1

# chain forward from WAN (ether port 1) to LAN (ether port 2-5) or vice versa
add action=drop     chain=forward comment="drop invalid" connection-state=invalid
add action=accept   chain=forward comment="accept established and related" connection-state=established,related
add action=drop     chain=forward comment="drop all from WAN not DSTNATed" connection-nat-state=!dstnat
connection-state=new in-interface-list=WAN
add action=drop     chain=forward comment="drop everything else " disabled=yes
```

```
/ip firewall nat
# scrnat (source network address translation is e.g. web request (port 80) from a PC to external webserver,
# dstnat is request from outside via firewall to internal device, here OPC-server
add action=masquerade chain=srcnat comment="masquerade" disabled=yes ipsec-policy=out,none out-interface-list=WAN
add action=dst-nat  chain=dstnat dst-address=10.0.0.254 dst-port=54844 log=yes protocol=tcp to-addresses=192.168.0.4
to-ports=4840

# in LAN everyone can call OPC server at ocp:tcp://192.168.0.4:4840 (port 4840).
From outside call the OPC server via router 10.0.0.254 at port 54844
```

P.s with more OPC-UA servers inside the LAN, more dst-nat rules are to be entered per server with other port numbers.



8 items

		▲ Src. Address	Dst. Address	Proto...	Connecti... Mark	Timeout	Orig./Repl. Rate	Orig./Repl. Bytes
-	C	192.168.0.10:51129	192.168.0.1:53	17 (udp)		00:00:09	0 bps/0 bps	134 B/0 B
-	C	192.168.0.10:53906	8.8.8.8:53	17 (udp)		00:00:09	0 bps/0 bps	134 B/0 B
-	C	192.168.0.10:53467	192.168.0.1:53	17 (udp)		00:00:04	0 bps/0 bps	134 B/0 B
-	C	192.168.0.10:60173	8.8.8.8:53	17 (udp)		00:00:04	0 bps/0 bps	134 B/0 B
-	SAC	192.168.0.100:42408	192.168.0.1:80	6 (tcp)		23:59:54	0 bps/0 bps	129.4 KiB/637.2 KiB
-	SAC	192.168.0.100:42410	192.168.0.1:80	6 (tcp)		23:59:59	6.3 kbps/5.2 kbps	512.6 KiB/865.0 KiB
-	C	0.0.0.0:68	255.255.255.255:67	17 (udp)		00:00:09	5.2 kbps/0 bps	1133.9 KiB/0 B
-	SACd	10.0.0.73:49460	10.0.0.253:54850	6 (tcp)		23:59:57	0 bps/0 bps	39.2 KiB/25.6 KiB

Figure 2 The end result (see last line)

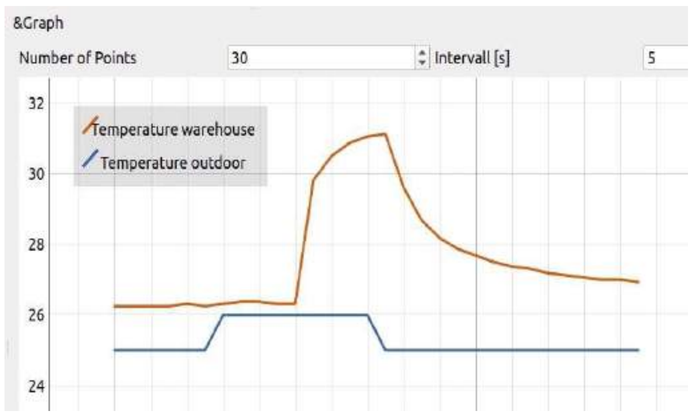
Annex: opcua-client (and with the graph outputs)

Instead of the standard Python environment, on might want to install WinPython in the windows environment. The reason is that Winpython comes with many pre-installed Python libraries, not only for drawing graphs, but also usable for numerical computing, AI usages etc.

If you search for opcua-client and enter the site it mentions that opcua-client can run on unix systems, but also on windows. However, the author suggests to use the Winpython environment on a windows pc, not the regular python, because of all the libraries it needs. It works quiet well, but the installation takes a long time. However if you use the standard Python environment and have opcua installed with pip3 install opcua, installing the opcua-client goes similar. It might be that additional libraries are needed and not yet automatically installed with opcua and/or opcua-client. In particular PyQt5 library might be needed. Just install that one with $*pip3 install pyqt5*. Notice that you need to be connected to the Internet.

We didn't start with it in the course as it takes a long time to install. I mix and match Python between Ubuntu, MacOS and Windows all with PyCharm and github (embedded with Pycharm) and prefer the standard Python distribution (as suggested above). But if you work a lot with Windows, it might be sensible to use this Python distribution and then install the opcua-client and play around with the client access the Pi-server and create the interesting graphs as below.

It is up to the reader to decide self. If you do the extra effort, graphs are possible, but it is a little more complex. The pictures in this course are down on an Ubuntu system.


&Graph
Number of Points
30
Intervall [s]
5
32
30
28
26
24
Temperature warehouse
Temperature outdoor