Assignment 3 in IIA2017

Torstein Solheim Ølberg | 263054

February 24, 2024

Contents

1 Introduction

In answer to the competition listed by Not At All Made Up Industrial Production Company [1], MyCompany AS has decided to compete in this contest. This article will outline MyCompany AS's results from designing an answer to this competition. This task of the competition is to design a prof of concept for a data acquisition application set up to communicate with a server, and log the data to a file, using OPC interface. In this article, four different applications will be presented, with different software languages and types of OPC being represented. First, the different applications developed will be explained, then the results of each software tested with different servers. Afterwards, a test of the applications over a network using two different devices, and finally, a discussion of and a conclusion on the results.

2 Results

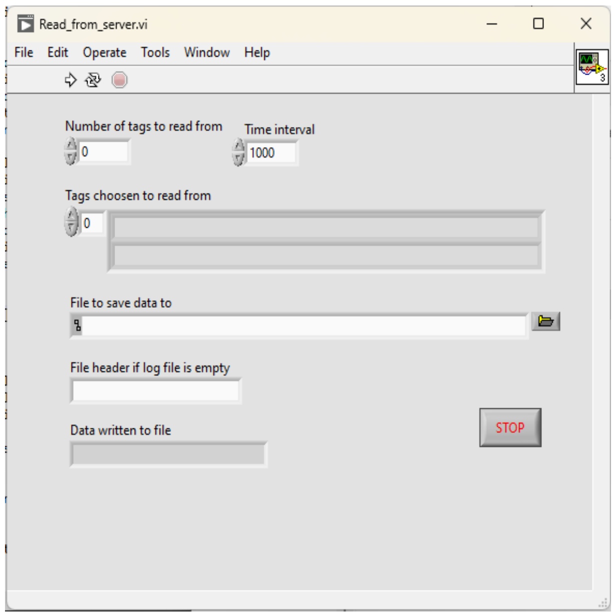
2.1 Application Information

Four applications where developed. Two using LabView, a software development platform from NI [2], one using the programming language C# and the development platform Visual Studio from Microsoft [3], and one using the programming language python. The two LabView applications use OPC DA and OPC UA respectively as communication interfaces, and the other applications use UA only.

2.1.1 LabView DA Application

The LabView OPC DA application is made using LabView Q1 2024. The program consists of two application, one for writing two and one for reading from the server. The Data Socket Pallet is used for connecting to, reading from, writing to and disconnecting from the server. The applications first prompt the user to choose an available server and tag and then continuously, with the use of a while loop, writes data to or reads data from the chosen server tag with a fixed time interval. The reading application also saves the data automatically to a .csv file for later use. The user can choose how many tags to interact with, and also display the latest data on the server.

In figure 1 it is shown that the reading application consists of a simple GUI where the user can choose the number of tags to be read and time interval of reading. The file to be written to can also be chosen, and if that file is empty a possible header for the file can be given.



Number of tags to read from 0

Time interval 1000

Tags choosen to read from 0

File to save data to

File header if log file is empty

Data written to file

Figure 1: A screenshot of the LabView OPC DA server reader application. the user can decide different inputs, and recieve information back from the server.

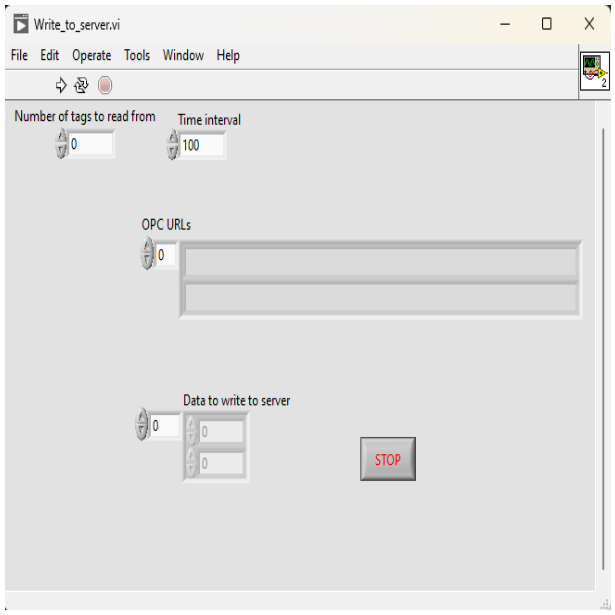


Figure 2: A screenshot of the LabView OPC DA server writer application. It is show that the design is similar to the reader, with different input being given and the Server URL being displayed.

In figure 2 it is possible to see that the writing application is similar to the reading application, but with fewer elements. The number of tags, time interval and the display for OPC URL are the same, just with different names. However there is an input element where the user can write what data they want to write to the server, which replaces the file selection and output element of the reader.

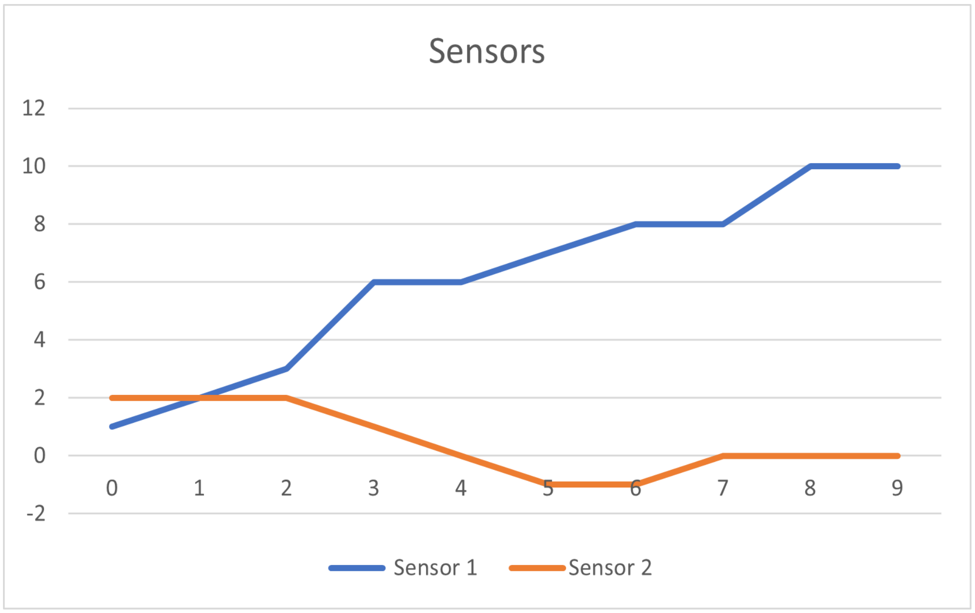
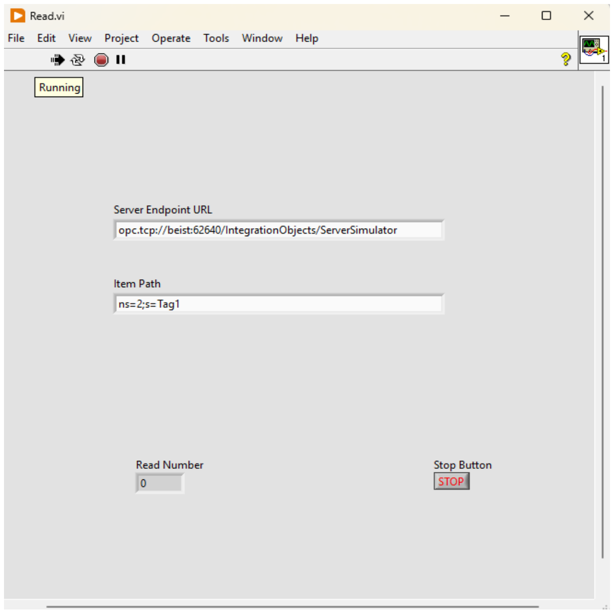


Figure 3: A plot of the data produced from running the application with two tags.

The resulting data read from a single running of the program, with two tags, and plotted using Microsoft Excel can be seen in figure 3.

2.1.2 LabView UA Application

As for the LabView UA application, this also uses the Q1 2024 version of LabView, but with the added package LabView OPC UA Toolkit. This application includes three parts. The clients, which read and write from the server, each connect to the sever using a string input from the user and the Connect.vi. Then they enter a while loop and interact with the server using either a Multiple Read- or Multiple Write.vi continuously with fixed time intervals. Finally, they disconnect from the server and handle any problems. The reader displays the value of a single tag read from the server as seen in figure 4, while the writer gets a random value generated, to simulate a sensor input, and displays this value as seen in figure 5, while writing it to a single tag. The final part of the application is a server, set up with a single tag, which is capable of being read from and written to. The server, seen in figure 6 starts by opening a server using the Create.vi. Then it creates a folder with the name Data and an item with the name Temperature, type

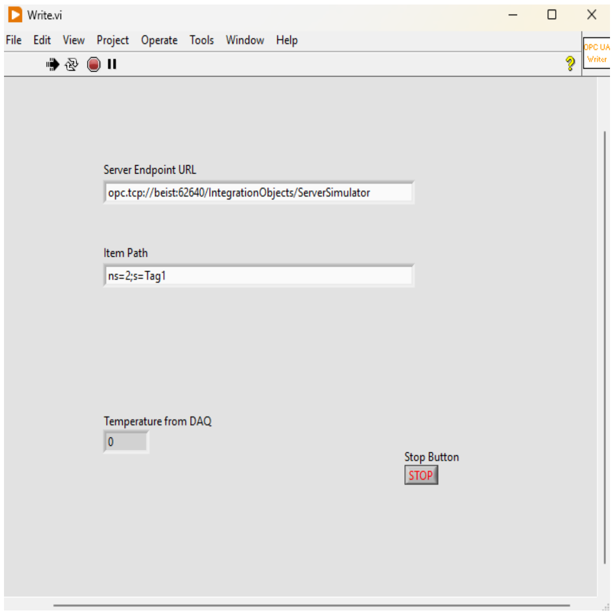


Server Endpoint URL: opc.tcp://beist:62640/IntegrationObjects/ServerSimulator

Item Path: ns=2;s=Tag1

Read Number: 0

Figure 4: A screenshot of the LabView OPC UA server reader application. The design is similar to the DA reader, but it only displays the data read and doesn't allow for multiple tags to be read.



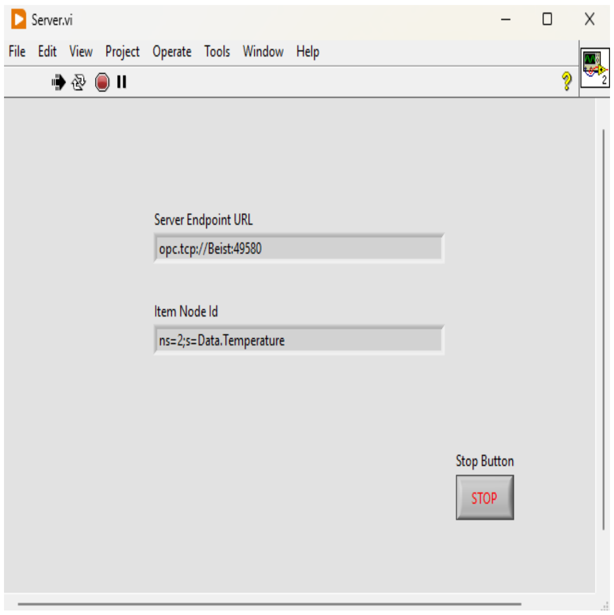
Server Endpoint URL
opc.tcp://beist:62640/IntegrationObjects/ServerSimulator

Item Path
ns=2;s=Tag1

Temperature from DAQ
0

Stop Button

Figure 5: A screenshot of the LabView OPC UA server writer application. Almost identical to the reader except the indicator for data is what is being written to the server and not what has been read.



Server Endpoint URL
opc.tcp://Beist:49580

Item Node Id
ns=2;s=Data.Temperature

Figure 6: A screenshot of the LabView OPC UA server application. Almost identical to the reader and writer, except it lacks an indicator for data.

double and both read and write access. Finally, it starts the server with Start.vi and enters a while loop and when exiting the while loop it stops and closes the server.

2.1.3 C# Application

The C# application is a single program, as seen in figure 7, which can both read and write itself. It was built using Visual Studio and the NuGet Package Opc UaFx Client version 2.41.1. The program connects to the server each time it is prompted to do so by the user, either to read from or write to the server. The user can specify a number to be written to the server by filling in the input box to the right of the write button.

2.1.4 Python Application

The finale application was created using the programming language Python, and the package Opcua. The application consists of a reading and a writing program, each which connects to the server using Opcua and then enter a while loop where they perform their task. At some point, the user interrupts

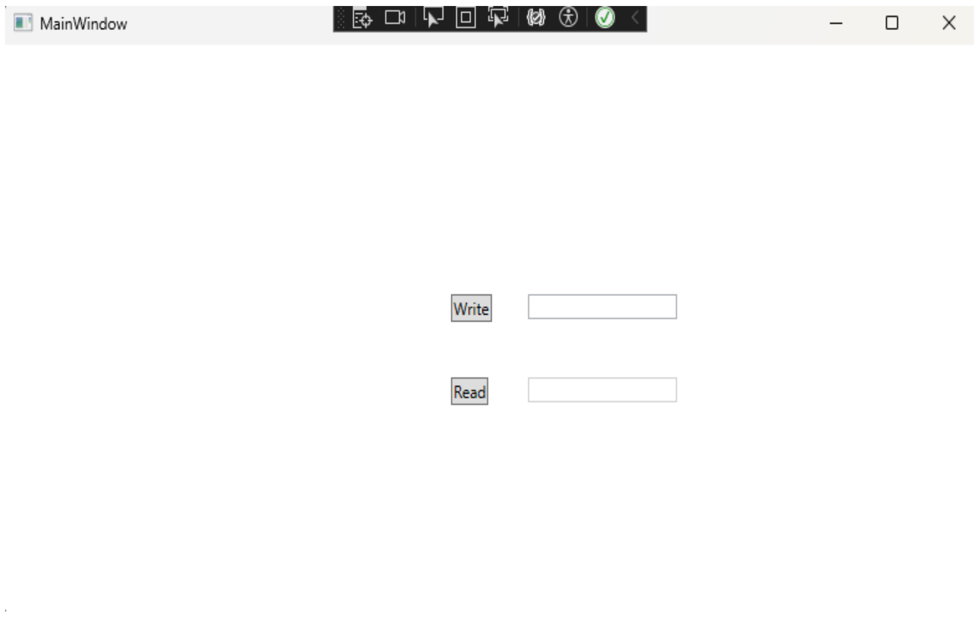
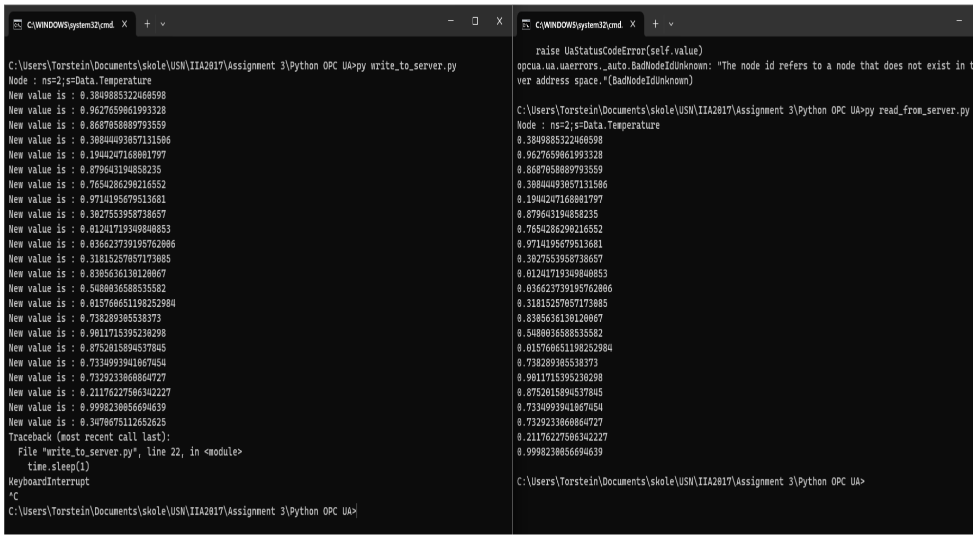


Figure 7: A screenshot of the C# OPC UA application built from debugging in Visual Studio. There is a window for input to what should be written to the server and two buttons. One button writes the user input to the server and one reads from server.



```
C:\Users\Torstein\Documents\skole\USN\IIA2017\Assignment 3\Python OPC UA>py write_to_server.py
Node : ns=2;s=Data.Temperature
New value is : 0.3849885322460598
New value is : 0.9627659061993328
New value is : 0.8687058089793559
New value is : 0.30844493057131506
New value is : 0.1944247168001797
New value is : 0.879643194858235
New value is : 0.7654286290216552
New value is : 0.9714195679513681
New value is : 0.3027553958738657
New value is : 0.012417193498408533
New value is : 0.036623739195762006
New value is : 0.31815257057173085
New value is : 0.8305636130120067
New value is : 0.5480036588535582
New value is : 0.015760651198252984
New value is : 0.738289305538373
New value is : 0.9011715395230298
New value is : 0.8752015894537845
New value is : 0.7334993941067454
New value is : 0.7329233060864727
New value is : 0.21176227506342227
New value is : 0.9998230056694639
New value is : 0.3470675112652625
Traceback (most recent call last):
  File "write_to_server.py", line 22, in <module>
    time.sleep(1)
KeyboardInterrupt
^C
C:\Users\Torstein\Documents\skole\USN\IIA2017\Assignment 3\Python OPC UA>
```

```
    raise UaStatusCodeError(self.value)
opcua.ua.uaerrors._auto.BadNodeIdUnknown: "The node id refers to a node that does not exist in t
ver address space."(BadNodeIdUnknown)

C:\Users\Torstein\Documents\skole\USN\IIA2017\Assignment 3\Python OPC UA>py read_from_server.py
Node : ns=2;s=Data.Temperature
0.3849885322460598
0.9627659061993328
0.8687058089793559
0.30844493057131506
0.1944247168001797
0.879643194858235
0.7654286290216552
0.9714195679513681
0.3027553958738657
0.012417193498408533
0.036623739195762006
0.31815257057173085
0.8305636130120067
0.5480036588535582
0.015760651198252984
0.738289305538373
0.9011715395230298
0.8752015894537845
0.7334993941067454
0.7329233060864727
0.21176227506342227
0.9998230056694639

C:\Users\Torstein\Documents\skole\USN\IIA2017\Assignment 3\Python OPC UA>
```

Figure 8: A screenshot of the Python OPC UA reader and writer applications ran using CMD. These programs reads and writes a tags value and prints the value to the terminal.

the code with a keyboard interrupt, and the programs perform any last tasks, disconnects from the server and ends. The reader, as seen in figure 8 reads the value of a tag from the server and prints it to the terminal as well as a .csv file. The writer, as seen in figure 8, gets a random number from the random package and writes this to the server.

2.2 Communication Test

Each application where tested by running the program up against their partner. Meaning that the reader programs and writer programs from each application where tested together. They where also tested using different servers, and the results of the tests can be seen in table 1 The table shows that all the applications work at that the server made in LabView also works. There are two server simulators used as well, one from Matrikon and one from Integration Objects.

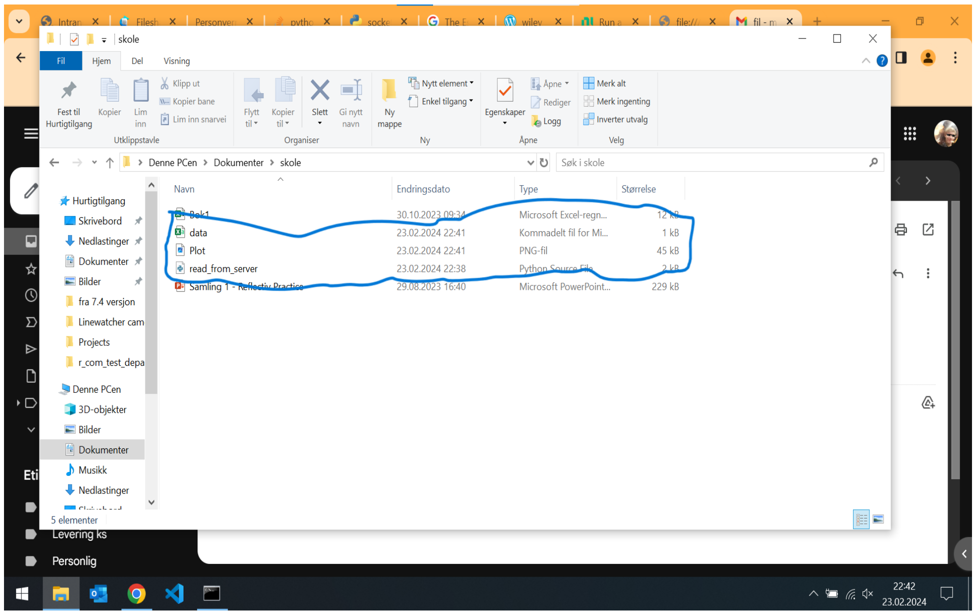
2.3 Network Communication Test

The network communication test was performed only on the python application, and a screenshot of the resulting datafiles and a plot of the data

Application	Server	Result
LabView OPC DA	Matrikon OPC Simulation Server [4]	Worked
LabView OPC UA	Selfmade LabView OPC UA Server	Worked
LabView OPC DA	OPC UA Server Simulator [5]	Worked
LabView OPC DA	Selfmade LabView OPC UA Server	Worked

Table 1: The result from testing each of the applications. All the applications work.

produced are shown in figure 9 and 10 respectively.


Denne PCen › Dokumenter › skole

Navn	Endringsdato	Type	Størrelse
Bok1	30.10.2023 09:34	Microsoft Excel-regn...	12 kB
data	23.02.2024 22:41	Kommadelt fil for Mi...	1 kB
Plot	23.02.2024 22:41	PNG-fil	45 kB
read_from_server	23.02.2024 22:38	Python Source File	2 kB
Samling 1 - Reflectiv Practice	29.08.2023 16:40	Microsoft PowerPoint...	229 kB

5 elementer

Figure 9: A screenshot of the C# OPC UA application built from debugging in Visual Studio. There is a window for input to what should be written to the server and two buttons. One button writes the user input to the server and one reads from server.

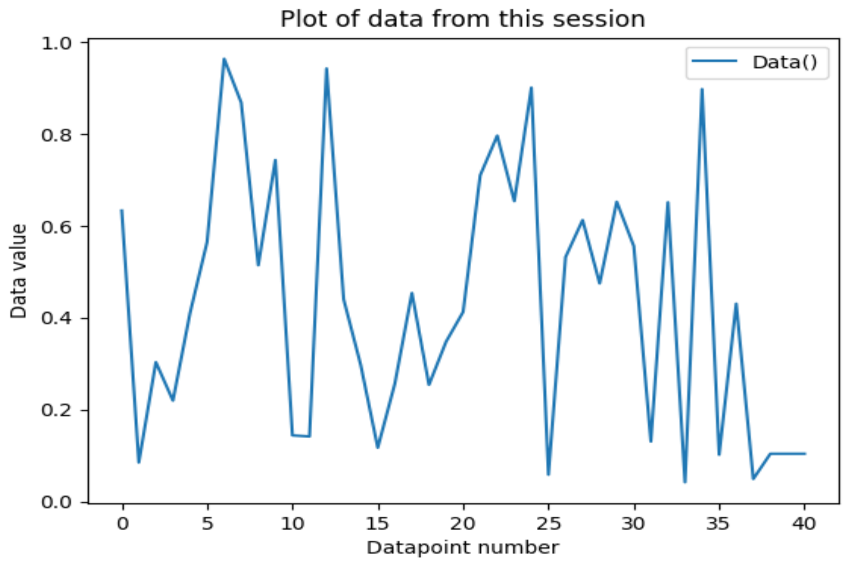


Figure 10: A screenshot of the C# OPC UA application built from debugging in Visual Studio. There is a window for input to what should be written to the server and two buttons. One button writes the user input to the server and one reads from server.

The figures show that the reader was run from a different pc and that two files where created as expected. They also show that the data read was not empty or a single value, but changing throughout the running.

3 Discussion

Each of the applications work and can perform the task of writing data to an OPC server and reading it from the server. Some of the applications can save the data to files and one of them can also plot the data as well. The applications answer the task outlined in the competition, but do little more. The combination of all the different applications into a single one with the capabilities off reading, writing saving to file and plotting, as well as prompting the user to choose the server and tag, and how many tags to choose would have been optimal.
The applications should also be tested more thoroughly, not just up against their own partners, but also each others. They should also have been tested up against different servers. If more time had been available this would have

been the next step. Furthermore, as there was only one other computer available during the production of these applications, and this computer did not have administrative rights, the only application which could be tested over a network was the one which that computer already had the software for. This was the Python application, and with more funds It would have been possible to rent another device and test the other applications as well. This was also the reason why Python was chosen in stead of MATLAB which the contest listed as preferred, but since Python was the only software possible to test over a network, and the contest said other software could be used, Python was chosen.
Finally, a test where actual sensors and not just simulated sensors where used would have been beneficial, so as to make sure this would not produce complications, but as with the network test, there where no funds for such a test and this was cut from the scope.
If this article should recommend the use of one of these applications, it would be the LabView OPC DA application, as it has the most features and could be used with little extra development.

4 Conclusion

The contest, posted by Not At All Made Up Industrial Production Company, has been answered, and four OPC server interaction applications have been produced. The are all capable of reading from and writing data to an OPC server, but lack some testing and should possibly have been combined into a single application with the abilities all of them. However, the best one of the four is most likely the LabView OPC DA application as it has the most features.

References

[1] H. P. Halvorsen. "Daq and opc system." (2024), [Online]. Available: https://www.halvorsen.blog/documents/teaching/courses/industrialit/lab_assignment/OPC%20Lab.pdf. (accessed: 22.02.2024).

[2] NI. "Labview." (2024), [Online]. Available: https://www.ni.com/en/support/downloads/software-products/download.labview.html#521715. (accessed: 24.02.2024).

[3] Microsoft. "Visual studio." (2022), [Online]. Available: https://visualstudio.microsoft.com. (accessed: 24.02.2024).

[4] Matrikon. "Matrikonopc simulation server." (2024), [Online]. Available: https://www.matrikonopc.com/products/opc-drivers/opc-simulation-server.aspx. (accessed: 24.02.2024).

[5] I. Objects. "Opc ua server simulator." (2020), [Online]. Available: https://integrationobjects.com/sioth-opc/sioth-opc-unified-architecture/opc-ua-server-simulator/. (accessed: 24.02.2024).

[6] T. S. Olberg. "Assignment 3." (2024), [Online]. Available: https://github.com/MrTorstein/IIA2017/tree/main/Assignment%203. (accessed: 24.02.2024).

5 Appendix A

Here you can find the documentation for the applications:

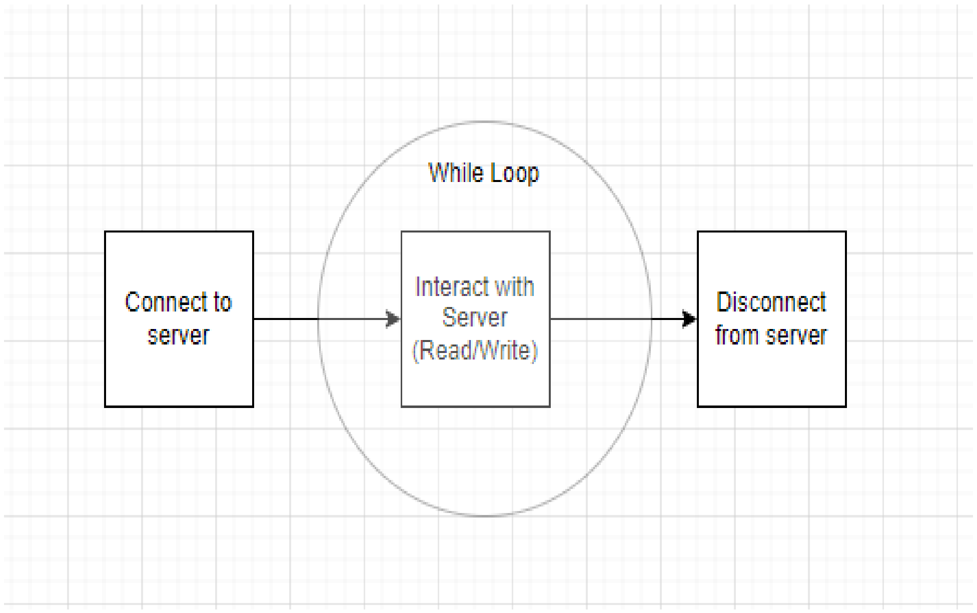


Figure 11: Flowchart depicting the general flow of all the applications. The applications all connect to a server, enter a while loop where they interact with the server, and finally, exit the loop and disconnect from the server.

Figure 11 shows the general build up of all the applications. All applications are build up in the same way, connection to the server, interacting with it inside a while loop, and finally disconnecting from the server when exiting the while loop.

6 Appendix B

All code can be found on the projects GitHub repository [6]. As listing the code in this rapport would be long and hard thing to read, this way has been chosen.