Martti Korpioksa

Cooperation between Unity and PLC

Comparison of different PLCs and OPC-servers

Thesis
Autumn 2014
School of engineering
Electric automation

Seinäjoen ammattikorkeakoulu
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikan Yksikkö

Koulutusohjelma: Automaatio

Suuntautumisvaihtoehto: Sähkö automaatio

Tekijä: Martti Korpioksa

Työn nimi: **Cooperation between Unity and PLC**

Ohjaaja: Petteri Mäkelä

Vuosi: 2014 Sivumäärä: Liitteiden lukumäärä:

Tässä opinnäytetyössä vertaillaan erillaisten ohjelmoitavien logiikoiden ja OPC-palvelimien toimintaa. Porausasemasta on olemassa 3D-malli, joka on tehty Unity-ohjelmalla. Tälle mallille lähetetään komentoja ohjelmoitavalla logiikalla. Nämä komennot siirtyvät Unityyn OPC-palvelimen ja soketti-palvelimen kautta. Tällainen järjestelmä on tässä työssä rakennettu 3 kertaa eri ohjelmoitavilla logiikoilla ja OPC-palvelimilla. Käytettävät logiikat ovat Omronin CPM1, CJ1M ja Beckhoffin TwinCAT. Käytetyt OPC-palvelimet ovat PLC data gateway ja Kepwaren KEPServerEX5. Kun kaikki kolme järjestelmää oli rakennettu, niiltä mitattiin vaste-ajat, eli kuinka kauan kestää signaalin kulku Unitysta logiikalle ja takaisin.

Avainsanat: ohjelmoitavat logiikat, 3D-mallinnus

Avainsanat: ohjelmoitavat logiikat, 3D-mallinnus

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Engineering

Degree programme: Automation Engineering

Specialisation: Electric Automation

Author: Martti Korpioksa

Title of thesis: **Cooperation between Unity and PLC**

Supervisor: Petteri Mäkelä

Year: 2014 Number of pages: Number of appendices:

In this thesis operations of different programmable logic controllers and OPCs are compared. In this project there is a 3D-model of a drilling station in Unity, which receives commands from a programmable logic controller. These commands are then transferred to Unity via an OPC server and a socket server.

These kinds of setups are built three times with different programmable logic controllers and OPC servers. The logics used are Omron's CPM1, CJ1M and Beckhoff's twincat. The OPC-servers used were PLC data gateway and Kepware's KEPServerEX5. When all these three setups had been built, their response times were measured. In other words, it was studied how long it takes from the signal to travel from Unity to logic and back.

Keywords: Programmable logic controller, 3D modelling

CONTENT

Tables and figures

1 Introduction

There have not been any major studies about the cooperation between Unity and programmable logic controller (PLC). This might also be why Unity does not have any PLC add-ons like for example the Visual Components' 3DCreate. Unlike 3DCreate, Unity has a built in real-time physics engine which would make Unity a lot more useful than 3DCreate. Unity's market share is also increasing, so it will probably be a more popular software in the future. Unity is used mainly in video game industry to develop different games. Despite that Unity has become quite popular among independent game developers, but not among major gaming studios.

In this thesis a test environment was built where a virtual drilling workstation is controlled by a PLC. The virtual drilling workstation was modelled by Unity. The PLC controls this 3D-model just like the real workstation. The data will be transferred from PLC to Unity with OPC server. However, because Unity is not capable of receiving data directly from the OPC server, there will also be an additional server using TCP/IP socket communication. The socket server is a program which receives the data from the OPC server and then sends it to Unity.

Even though there are not any major studies about this subject, there are companies that have concentrated on building and designing virtual simulators. One example of these companies is Mevea from Lappeenranta Finland. Mevea was founded in 2005 and its main focus is dynamic simulation applications. (Mevea ltd. 2013)

Mevea's product repertoire also includes education simulators like mining simulators, forestry machine simulators, product development simulators, as well as modelling and simulation services. Other simulation services are Mevea cabin and Mevea Cave. (Mevea ltd. 2013)

1.1 Goals

In this thesis a virtual learning environment of a mechatronic laboratory device will be created with a Unity game engine. This learning environment gives a chance to research the possibilities to control virtual device with a PLC. The information from PLC to PC will be transferred with OPC. This whole setup would significantly ease the designing of production lines. With this designers are able to test the production lines before actually building them. Unity is equipped with a real time physics engine, so the designer would also be able to test different scenarios that could affect the production lines.

1.2 Thesis structure

Chapter 2 contains theory about PLCs and PLC programming, OPC servers, Unity and 3D-modelling. Chapter 3 reviews all software and hardware used in this project. Chapter 4 illustrates the work that was done and all different environments what were used. Chapter 5 presents the results, how the response time of different environments was measured and compared. Chapter 6 includes a summary of this project. In the end all sources and attachments are listed.

1.3 CAVE and Virtual-laboratory

There was a plan in the early 2000s to build a new technology center in Seinäjoki, where also the school of engineering would be placed. At that time also an idea about building CAVE was announced. By that time the only places to have similar virtual-laboratories in Finland were the University of Jyväskylä, Tampere University of Technology and Helsinki University of Technology. The technology center in Seinäjoki was ready in 2003, but CAVE needed two more years and its opening was on 10.February.2005. It was funded by Seinäjoki University of Applied Sciences, but some of the funding came from Western-Finland’s provincial government EAKR-project. Even today SeAMK’s CAVE is one the most advanced virtual rooms in Finland. (Hellman. 2014.)

CAVE or Cave Automatic Virtual Environment is a real-time interactive 3-dimensional computer graphics studio. In the CAVE a user can get the 3D-plans in natural scale and in the most realistic form. A real-time interactive environment is built around the user. This is done by scanning the location of the spectator's eyes and the picture is projected to each surface surrounding the spectator from all directions of the visual field. This will fully cover the spectator's visual range. (Hellman. 2014.)

CAVE and other equipment in the visual laboratory are used for education, research and thesis work. CAVE can also be used in product development, because with CAVE developed products can be kept in virtual form without any physical prototypes. In CAVE motion capturing is also possible because of optical localization. This data of motion capturing can be used to create character animations by recording motion captured data to the computer to create a virtual skeleton. This skeleton can then be utilized in animation, ergonomics research and in robotics. There is also a haptic gadget, which is a 3-dimensional drawing -and processing tool with a somatosensory system. This makes it possible to feel the surfaces of virtual 3D-models by simulating the touch of the surface, liquid's viscosity, gravity, spring strength or inertia. There are several other pieces of equipment in the virtual laboratory, for example Kinect-character sensing devices, data gloves and leap motion-controllers. (Hellman. 2014.)

2 Tools

This chapter introduces the tools which were used to build the virtual drilling workstation. Also some basic information about the tools is presented.

2.1 Programmable Logic Controllers

Programmable logic controllers (PLC) were originally designed for car industry. In the year 1968 General Motors gave five demands for PLCs: The device has to be programmable and reprogrammable. It has to work perfectly in different workshops. It must tolerate 120V voltage used in United States electrical grid. It must stand the load of the electrical motors in continuous use as well as in starting. Its price must be competitive as compared to solidly wired logics. The first PLCs started to come to the markets already in 1968 - 1969. (Keinänen, Kärkkäinen, Metso & Putkonen 2001, 241-242.)

Basically there are two different types of logics: Stepping logics and freely programmable logics. In stepping logics the hierarchy of automation is straightforward and it goes on step by step. The biggest difference between freely programmable logics and stepping logics is that in freely programmable logics it does not matter in which order the program is written. Nowadays most of the logics are freely programmable logics. In freely programmable logic or shortly in programmable logic, input ports are coupled with all plausible sensors and buttons. Everything that is wanted to be controlled by the logic is coupled with the outputs, like different motors or cylinders. The program is written into the PLC's memory that monitors the programs progress in real-time. Because of this, it does not matter in what order you write the program. (Keinänen, Kärkkäinen, Metso & Putkonen 2001, 243-244.)

PLC's hardware consists of six different parts: inputs, a central processing unit, outputs, a programming device, a program memory and a power input. All signals that come from devices, like sensors, buttons and limit switches, are coupled into the inputs. Central processing unit (CPU) executes the program which is written to

the PLC. Usually microprocessors are used as central processing units, because then PLCs are able to do arithmetic calculations. Outputs control the actual device. These outputs send signals to the device's motors, cylinders, indicator lights and all components that move the device. Programming device is the device, which is used to write the program to the PLC. Almost all programs are made with PCs, but in the old times special programming devices were used. These somewhat resembled a calculator. Program memory is a part of the PLC this is where the actual program is stored, the CPU reads the program from there. Nowadays there are basically three different memory types in use: CMOS-RAM, EPROM and EEPROM. (Keinänen, Kärkkäinen, Metso & Putkonen 2001, 245-248.)

PLCs also contain other functions, like auxiliary memory bits, timers, counters, shift registers, pulse functions and the main control functions. Auxiliary memory bits are normally used to save data. They have two states, 0=not in use and 1=in use. Auxiliary memory bits can be used in several different options. For example, all requirements which are needed to start the program can be connected to one memory bit and then use this auxiliary memory bit in the actual program. (Keinänen, Kärkkäinen, Metso & Putkonen 2001, 248-251.)

Timers are meant to delay the device's work routine. Timers work on the principle that, timer starts with some input condition. Its output turns on when the timer's time reaches the time set in the timer. Counters can be used for example, to set an exact number of work routines for a device. Counter can also calculate the passing product flow. This is used for example in reverse vending machines. Counters usually have two input values: counter value and reset input. To the counter value are set all the commands that are going to increase the counter value. Reset input will reset the counter's counter value back to zero. Counter's output stays normally off until its value reaches the set value and then the counter's output turns on. (Keinänen, Kärkkäinen, Metso & Putkonen 2001, 248-251.)

There are four types of shift registers: SISO single input and single output, SIPO single input and multiple outputs, PISO multiple inputs single output and PIPO multiple inputs and multiple outputs. (Aalto-yliopisto 2003) Pulse of the pulse

function is very short and it is used in functions that need extreme speed. Main control function makes possible to stop the programs reading and by resetting main control function makes possible to continue programs reading at the exact point. (Keinänen, Kärkkäinen, Metso & Putkonen 2001, 251-255.)

Other common commands used in PLCs are for example LOAD, LOADNOT, AND, ANDNOT, OR, ORNOT, AND-LOAD, OR-LOAD, OUT, SET, RESET, JUMP, FUN, NOP and END. LOAD command is used to open the circuit, but for example Hitachi uses ORG command. NOP “No Operation” means empty row in program and END-command ends the program. In the most PLCs, commands are mostly same. The Biggest differences are in German Siemens’s STEP 6 command list, because they come from German words. Festo’s PLCs command lists also differ from others. They resemble more BASIC computer program. (Keinänen, Kärkkäinen, Metso & Putkonen 2001, 255-257.)

Programming languages that are approved by the IEC 61131-3 standard are ladder diagram, function block diagram, sequential function charts, structured text and instruction list. One PLC can support multiple different programming languages so the designer can choose which one to use. Ladder diagram is the most popular programming language when it comes to PLC. Ladder diagram resembles the actual hardware of the PLC. Ladder diagram has several rungs which are used to connect inputs to outputs. All PLC programs that are used in this project have been written in ladder diagram. (Kronotech, [Ref. 7.10.2014])

2.2 OPC

OPC is a way to transfer data created by OPC-foundation, which fulfills OPC data access specifications. Abbreviation OPC stands for OLE for process control, where OLE stands for Object Linking and Embedding. OLE is an older name for Microsoft’s COM data transfers. Originally OPC was meant to capitalize Microsoft’s component technology for the automation industry. First version of OPC came out in 1996. Most common OPC specifications are A&E (Alarms and

Events), HDA (Historical Data Access) and DA (Data Access). (Automaatioseura ry, [Ref. 16.9.2014])

Other specifications are, for example: Batch, Batch auto, Commands, Common, CPX (Complex Data), DX (Data eXchange), Security, UA (Unified Architecture and XMLDA (Honeywell international Inc., 2014). Data Access is meant for real-time process data transfer between control systems and process machinery. Alarms and Events are meant to transfer alarm and events data. For transferring historical data, Historical Data access is used. Data exchange is meant for data transfer between different OPC servers. XMLDA is similar to Data Access, but it uses Webservices and XML for its data transfer. (Automaatioseura ry, [Ref. 16.9.2014])

OPC Unified Architecture was first released in 2009, but some parts were published already in 2006 (OPCconnect.com, 2013). It was built so that, it would surpass all the previous OPC specifications. It was more extensive, when talking about hardware platforms and operating systems. Unified Architecture was compatible with following hardware platforms: PC hardware, cloud-based servers, PLCs and micro controllers. It was also compatible with these operating systems: Microsoft Windows, Apple OSX, Android and all distributions of Linux. Security was also a big concern when designing Unified Architecture. Its messages are sent in 128 or 256 bit encryption levels without corrupting original messages. It also uses sequencing to eliminate message replay attacks. Transport of the data can be OPC binary transport or SOAP-HTTPS, but also other options are available. Authentication is done by OpenSSL. In this OpenSSL all Unified Architecture servers and clients will be identified. This will control which applications and systems are allowed to connect with each other. All this can be done without having any problems with firewalls. (OPC foundation, 2014)

There were a few main reasons why OPC foundation started creating Unified Architecture. Microsoft’s COM and DCOM were becoming old and web services had risen to the main option for a data transfer between computers. In earlier OPC specification data models were different in every specification and there wasn’t any consistency between them. There also was not any backward compatibility between previous OPCs. (OPCconnect.com, 2013)

Unified Architecture differs from previous specifications by using IEC multipart specification and consists of twelve parts: Consepts, Address Space Model, Services, information model, service mappings and profiles. These six parts are core specifications and the other six parts are Access type specifications: Data Access, Alarms and Events, Commands, Historical Data, Batch and Data exchange. Unified architecture's architecture core consists among other things: object model, address space and profiles. In unified architecture they renewed object model, address space and semantic information model. In Unified Architecture the structure of address space was changed to be more versatile as compared to older specifications. (Automaatioseura ry, [Ref. 16.9.2014])

When it comes to performance Unified Architecture does not reach the same level as Data Access. This results from WebServices that are much heavier than DCOM, what Data Access uses. Computer capacity's rapid development will decrease this problem. OPC has created its own binary coding for the Unified Architecture, because binary coding xml would increase the performance of WebServices. It also increased the speed of a data transfer, because xml in text form wastes transfer resources. (Automaatioseura ry, [Ref. 16.9.2014])

The newest specification from OPC foundation is OPC.NET, which is based on framework of Microsoft's WCF.NET (Windows Communication Foundation). OPC .NET makes it possible to communicate easily through firewalls with quite a simplistic data model and removes the need for .NET and DCOM wrappers. OPC.NET enables access in both historical data and run-time data, events and alarms. OPC .NET's user interface is also designed so that user can do mapping to the OPC DA, HAD and A&E interfaces. For a comparison Unified Architecture is more complex and is created for communication between several different platforms. (OPC Training Institute, 2014)

OPC.Net has six goals:

- Security: all communication should be secure, but computers should also be accessible through firewalls.
- Simplicity: servers and clients are needed to be easy to implement, deploy and configure.

- Robustness: all communication is needed to be able to recover from errors.
- Backward compatibility: it is necessary to be able to connect previous OPC servers with .NET interface.
- Plug-and-Play: it is necessary to be able to find servers automatically.
- Transparency protocols are needed for proper communication between clients and servers.

Figure1 shows an example of the OPC test client, which uses .NET specification. (OPC Training Institute, 2014)

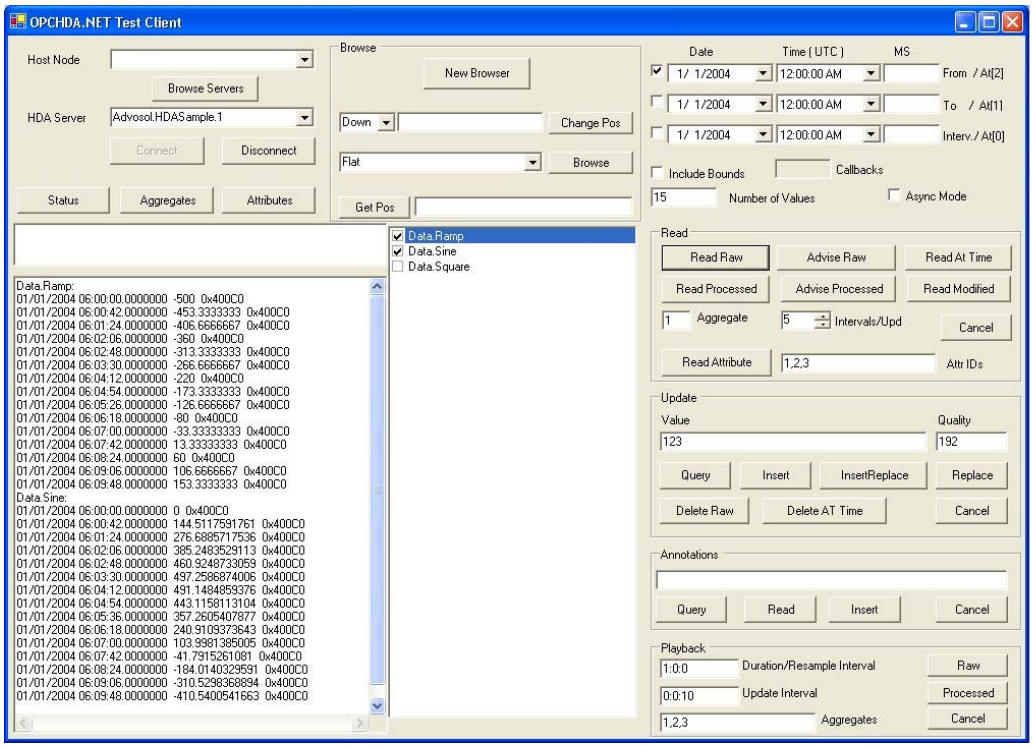

Figure 1. The interface of OPC test client, which uses .NET specification. (Advosol, [Ref 15.10.2014])

In Finland in the spring 2005 the OPC committee was founded as a part of Automaatioseura. OPC committee's goals were to advance Finnish automation education, research and entrepreneurship by sharing information about OPC foundation's activities and specifications. This was done by organizing education and events and also by taking part in creating OPC specifications. One of the reasons why the OPC committee was founded was the upcoming big specification called Unified architecture. (Automaatioseura ry, [Ref. 16.9.2014])

2.3 Unity

Unity (Unity technologies, 2014) is a multiplatform game-engine. It can be used to develop games for the following platforms:

- iOS and Mac
- Android
- Windows Phone, Windows and Windows store apps
- Blackberry 10
- Linux
- Web Player
- Playstation 3, 4, vita and mobile
- Xbox 360 and one
- Wii U.

Unity uses NVIDIA's PhysX physics engine, which is able to handle real-time physics (NVIDIA Corporation, 2014). The latest Unity version is Unity 4.5.3 which had several bugs fixed and it also contains enhanced 2d physics. A beta version of the Unity 4.6 is also available at the moment. Also Unity 5 has been announced. It is available for a pre-order, but its official release date has not been announced yet. In Unity 5 physics based shadings will be available also as a free version. Other improvements in Unity 5 are improved audio and a new 64-bit editor which will be beneficial when making large projects, a lighting system based on real-time physics and WEBGL, which makes it possible to take all the content to the server which uses WEBGL, without plugins. (Unity technologies, 2014)

Unity makes it possible to lay out levels and create menus. Also animating, making scripts and organizing projects is possible, which makes Unity fully 3d compatible. Unity's interface consists of four different panels: The project panel, hierarchy panel, inspector panel and scene panel. All the project's assets are stored in the project panel. All imported assets also appear there. In the hierarchy panel the assets of the scene can be arranged. In the inspector panel parameters for the assets can be adjusted. For example the assets position and ability to cast

shadows. The creation can be viewed from the scene panel. (Envato Pty Ltd. 2014)

Most of the assets like 3d models, textures, audio, scripts, fonts and materials, have to be imported to Unity. This results that Unity cannot create itself these assets, except from a few very basic models like spheres and cubes. Fortunately Unity is very open to different 3d-modelling programs and allows the transfer of files from other programs to Unity with all textures and materials intact. Unity supports all common file types like: PNG, JPEG, TIFF and PSD files from photoshop without any changes to the files. A list of all formats that Unity can import can be found from their homepage. (Envato Pty Ltd. 2014)

2.4 3D-modelling

3D-modelling means that products are designed in three dimensions, what happens by using x-, y-, and z- coordinates. So the designer can make the model look more like the final product. Real physical and mechanical properties can also be given to the 3D-model as in real life. x- , y- and z- coordinates are placed on the pc screen so that the x-axis is in line with the screens bottom edge, the y-axis is in line with the screens left edge and the z-axis points towards the designer. As in 2D-modelling it is also very important in 3D-modelling to which coordinates are positive and negative in direction. This information is needed to know in which direction will the product rotate. This is used when pictures are placed on the paper and when given assembly recommendations in degree form. (Tuhola & Viitanen 2008, 17-18)

All 3D-modelling programs assume that all degrees are given in positive forms, because programs will rotate the object in to a positive direction. The positive rotation direction of x- and y-axis is direction of positive z-axis, so towards the designer. The positive rotation direction of z-axis is negative y-axis so directly down on the pc screen. (Tuhola & Viitanen 2008, 18-19)

3D-model means a three dimensional product, which compares by look and properties to the final product. 3D-model can be examined in different ways in different programs. But most 3D-modelling programs use similar ways to examine products. (Tuhola & Viitanen 2008, 20)

Wireframe model means that only the edges of the model are displayed. The positive thing in this is that you can define points and edges through surfaces. Negative side in this model is that, it is hard to know which surfaces are at the back or at the front. It is difficult to know on which position the model is. Displaying holes and threads is difficult. it is also messy and unpractical. (Tuhola & Viitanen 2008, 20-21) This is usually used when 3D-models have to be transformed into 2D pictures (Tuhola & Viitanen 2008, 23).

3D-surface model displays only surfaces of the product. This is used usually only for casted and extruded products. In this model the product can be sculpted more freely than with the basic tools. However, it is possible to work only with visible surfaces. (Tuhola & Viitanen 2008, 21)

3D-model contains information of the models shape and also which parts of the model contain material. A good thing in 3D-model is that it is clear and easy to comprehend. It can also be examined to how it would be in real life. The disadvantages of this model are that it is not possible to choose surfaces that aren't visible or grab a surface through other surfaces. (Tuhola & Viitanen 2008, 22)

There are several different 3D modeling programs, but one of the most popular is the Blender. The Blender is a free 3D modeling program, which is being developed by volunteers. Blender makes possible to model, rig, simulate, animate, composite, render, and do motion tracking. Blender is a multiplatform program and it works for Linux', Mac's and windows' computers. (Blender, [Ref 22.10.2014])

3 Software and Hardware

In this project the Omron's Sysmac CPM1 (Figure 2), CJ1M (Figure 3) and Beckhoff's soft PLC were used as PLCs. Several PLCs were used to find out which PLC would work best. Omron PLCs were programmed by using a free trial version of CX-Programmer version 9.4 and Bechoff's soft PLC with TwinCAT3.

Figure 2. Omron's CPM1

Figure 3. Omron's CJ1M

In this project Omron's CPM1 and CJ1M were programmed by using a PC and a tool bus to connect the PC to the PLC. The ladder diagram was the programming

language used in this project. Unlike Omron's PLCs, the Beckhoff's PLC used in this project was not a physical PLC, it was only a software program inside the PC. The Beckhoff's PLC was also programmed using a ladder diagram, but TwinCat3 was used instead of the CX-programmer. There was also no need to create a connection between the PC and soft PLC, because TwinCAT3 made it automatically.

OPC Labs' QuickOPC 5.2 and PLC Data Gateway Developer Environment were used to create the OPC server for the first setup (Figure 4). Kepware's KEPServerEX 5 and OPC Quick Client were used to create the OPC server for the second and third setup. (Figure 5).

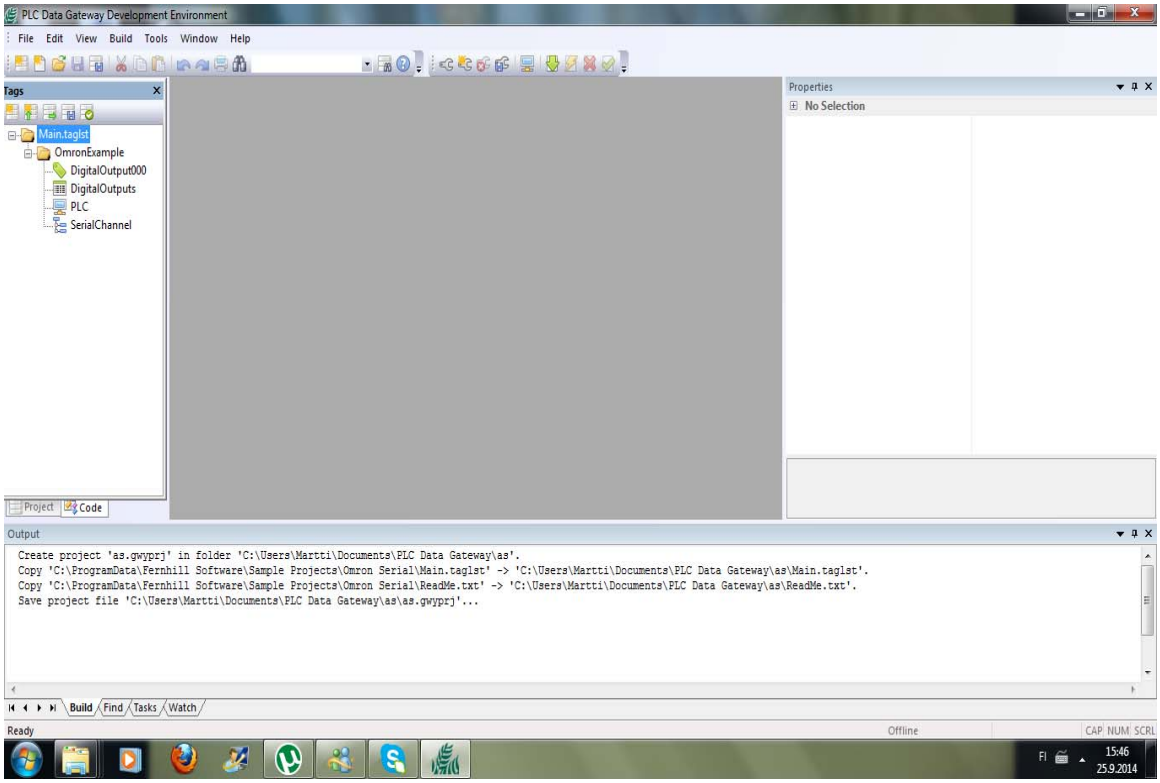

Figure 4 An Interface of PLC Data Gateway Developer Environment.

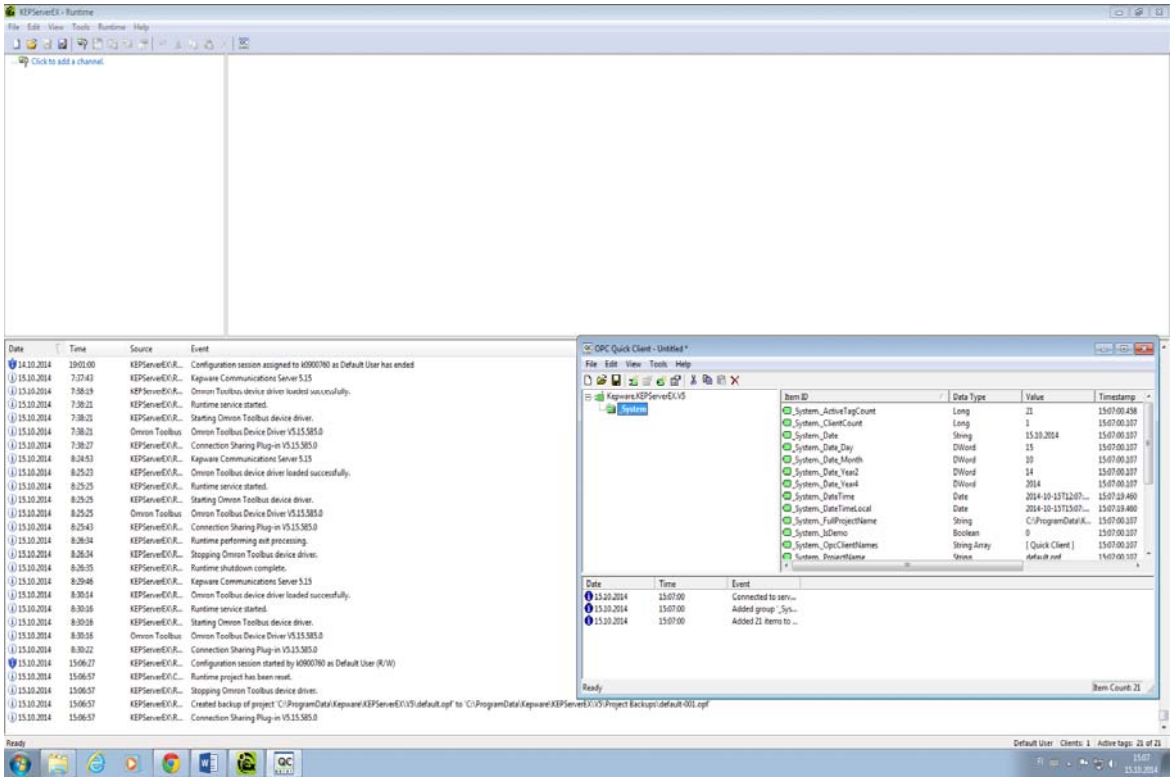

Figure 5. KEPServerEX and OPC quick client.

In the first setup a PLC Data Gateway was used as the OPC server with Omron's CPM1, but in the second setup, the OPC server had to be changed to KEPServerEX, because the PLC Data Gateway was not compatible with Omron's CJ1M. KEPServerEX was also used in the third setup with the Beckhoff's soft PLC.

Microsoft's Visual Studio Express 2013 for web was used for creating a socket server and scripts for Unity. In this project a free version of Unity's 4.5.4 was used (Figure 6). All models used in Unity were imported from other 3D-modelling programs, like Solid Edge or Blender. This means that complicated 3D-models cannot be created in Unity.

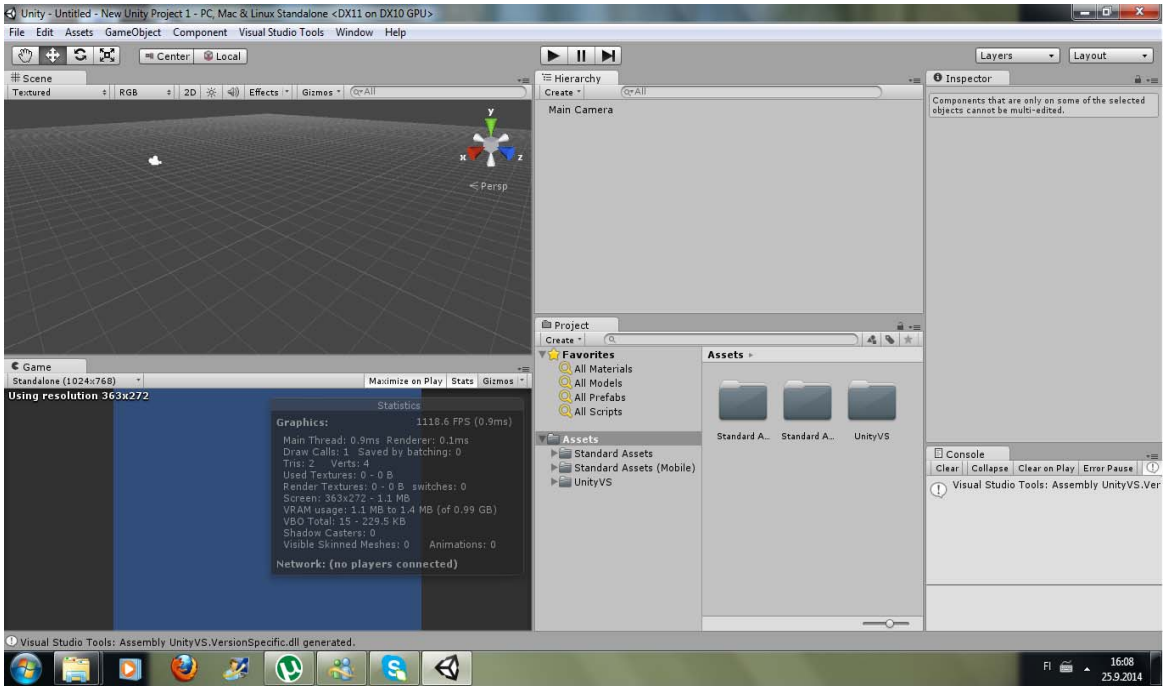

Figure 6. An interface of the Unity 3D

3.1 Changing Unity Script Editor

Unity has a built in script editor, MonoDevelop, but in this project it was changed to Microsoft's Visual studio Express 2013 for web. In this chapter it will be shown how this can be done. First a plugin for the Unity, Visual Studio 2013 Tools for Unity, needs to be downloaded. It can be downloaded from the web page: http://unityvs.com/. In the following way: First select preferences from edit tab (Figure 7). Then select external tools and browse from the script editor (Figure 8). From there choose: C:\Program Files (x86)\Microsoft Visual Studio12.0\Common7\IDE\VWDExpress (Figure 9). After this Visual Studio will be used automatically every time, when writing scripts in Unity. (Scott Richmond, 2013)

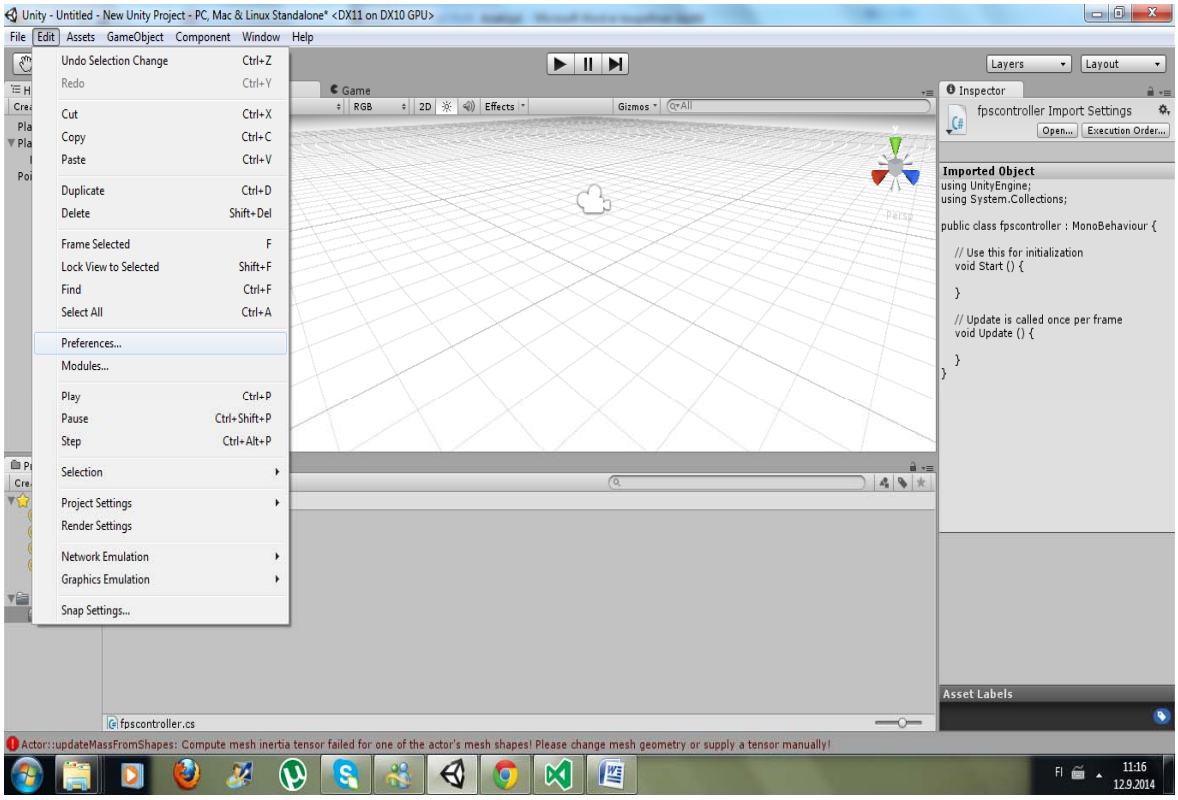

Figure 7. Unity interface with edit tab open.

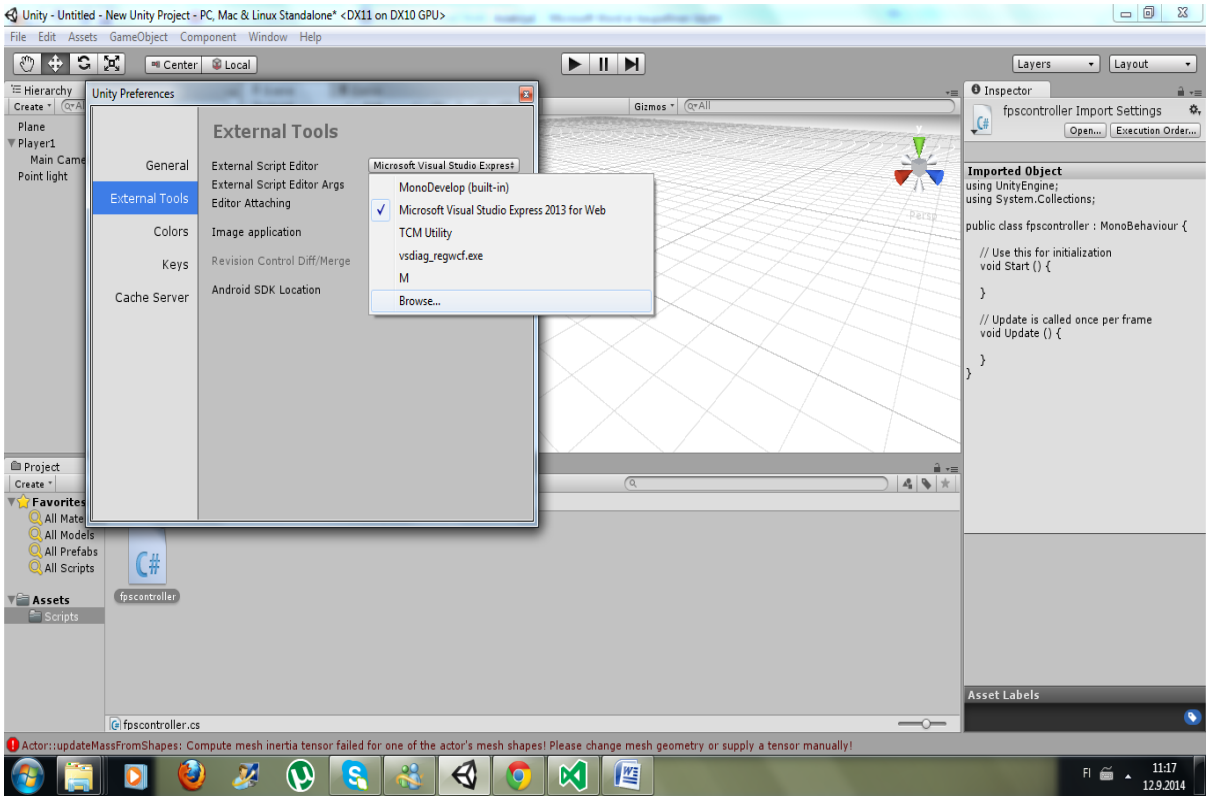

Figure 8. Unity interface, external tools.

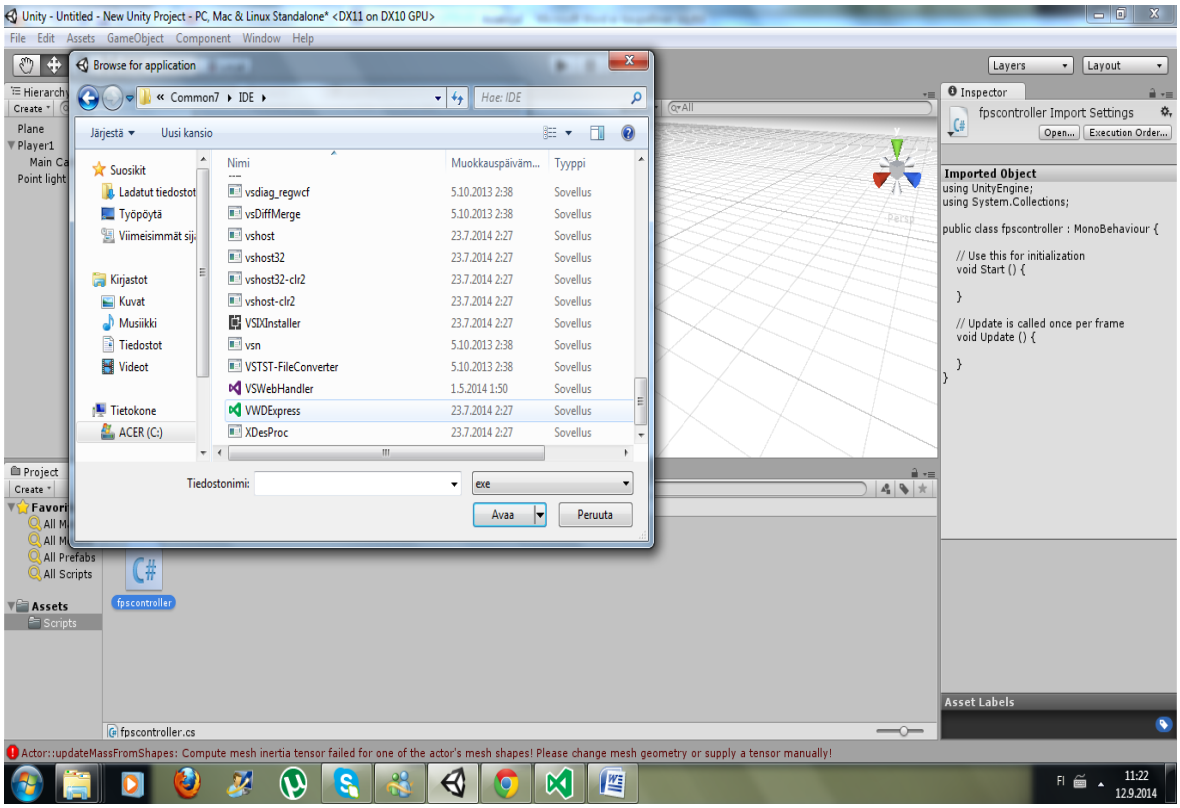

Figure 9. Choosing VWDExpress.

4 Test Setups

Three test setups with different PLC's and OPC servers were developed. In these setups there is a connection between PLC and OPC and between OPC and Unity (Figure 10). The socket server is a tool, which is used to transfer data from OPC to Unity, because Unity cannot receive data directly from the OPC server. A socket server might be integrated to Unity sometime in future. Data also flows backwards from Unity to OPC. This makes it possible to simulate sensors in Unity. Sensors in Unity send signals to PLC, which can be used in this program.

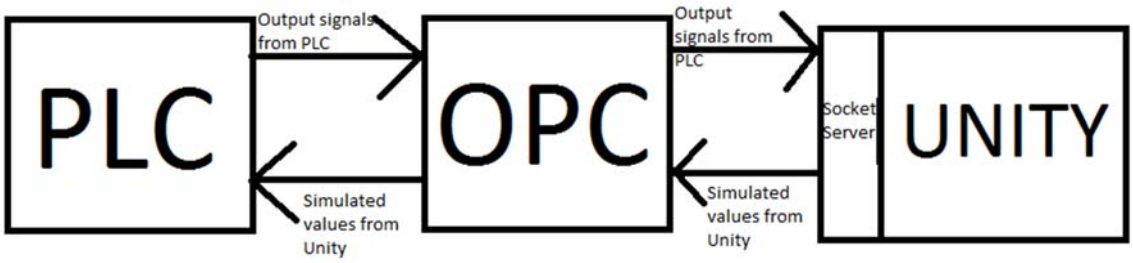

Figure 10. Connections between PLC, OPC, Socket Server and Unity.

4.1 First Setup

The drilling station (Figure 11), which is used to model the Unity model (Figure 12) is a basic workstation. It has a sledge which is able to move all horizontal directions. This sledge will carry an object which will be drilled. The drill is also a very basic, it just moves down and up. There are some differences between the actual drilling station and the Unity model. For example, the user interface is in different location.

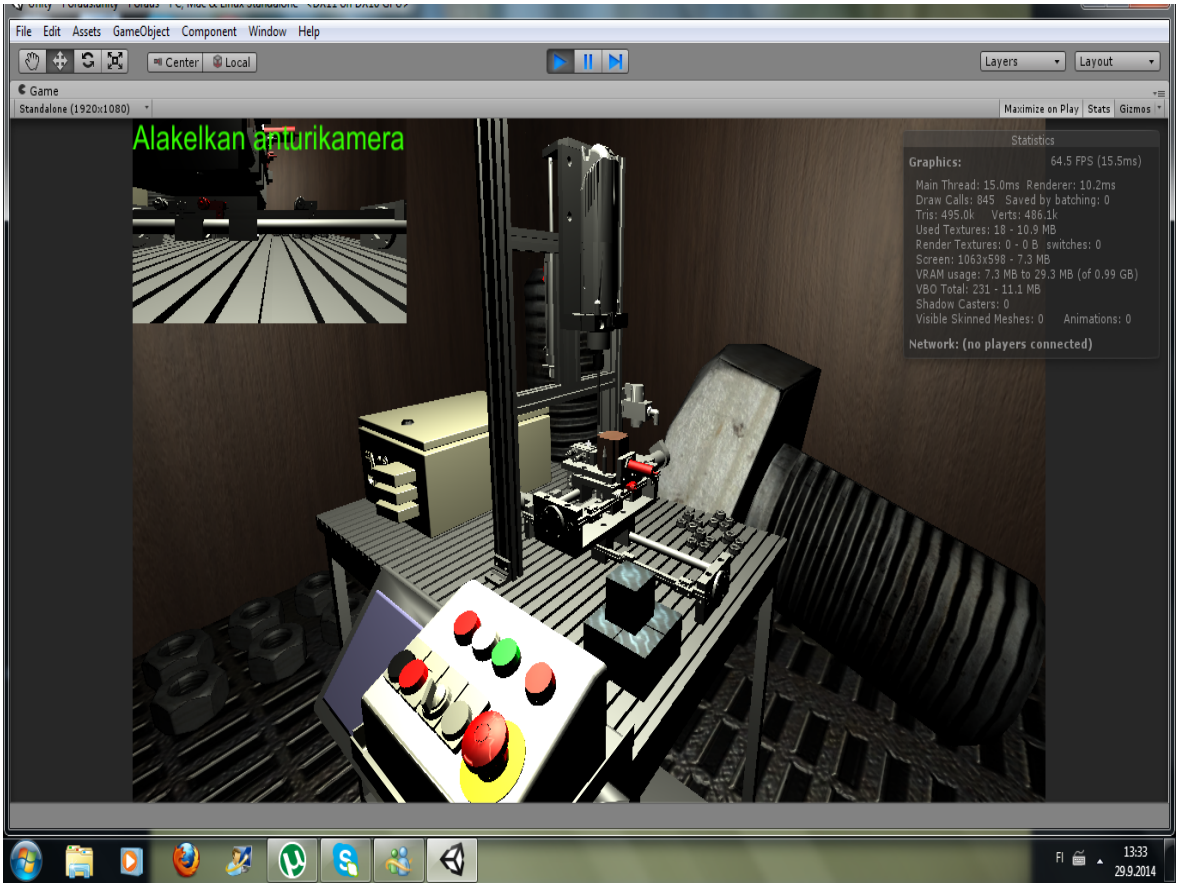

Figure 11. The drilling station in Unity

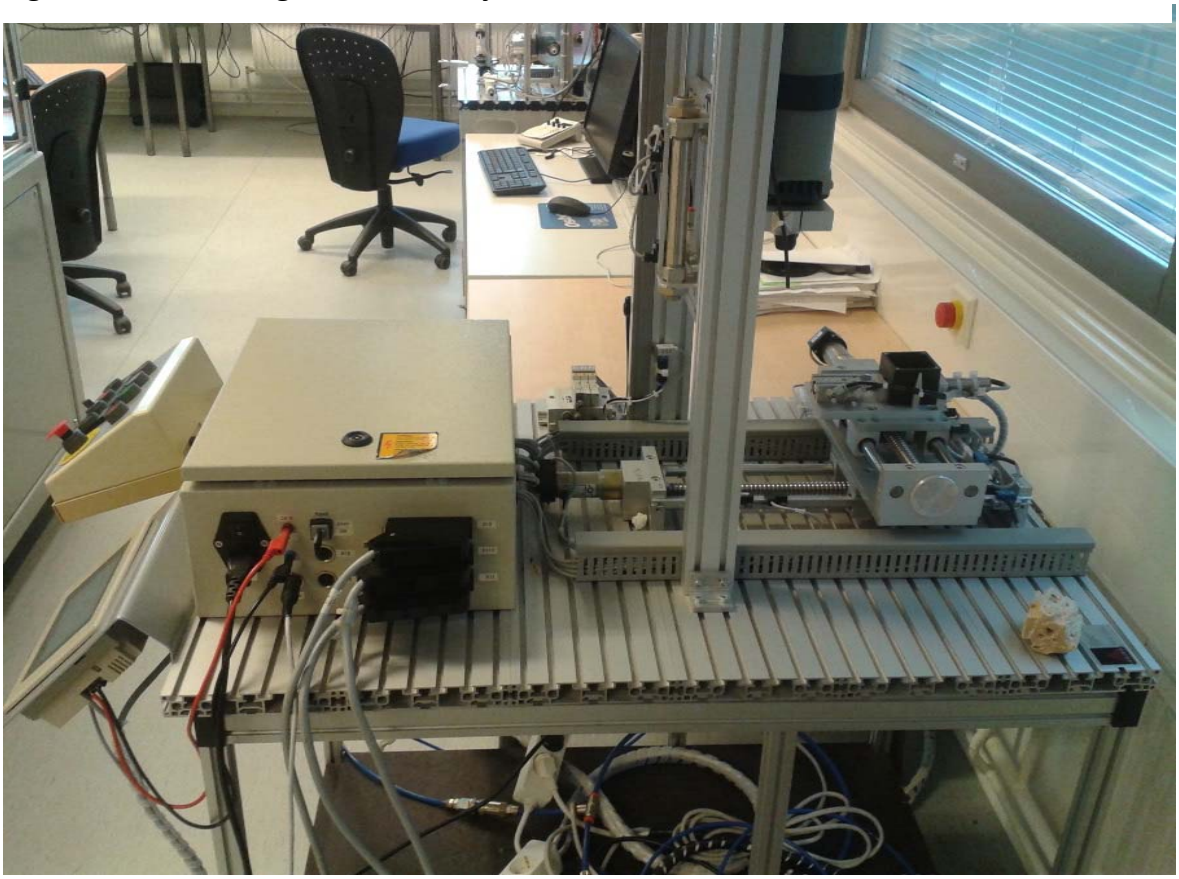

Figure 12. The actual drilling station

In the first setup of this project: there is a PLC connected to a laptop which has OPC server, socket server and Unity. The PLC will run the Unity simulation. All the PLC's signals will be transmitted to Unity with OPC and socket server. The Unity model is a drilling station, which is replicating a real drilling station, which is located in a laboratory. In this setup, PLC Data Gateway is used as an OPC server and PLC is Omron's CPM1.

Figure 13. A laptop is connected to PLC

4.1.1 PLC Program

In this project PLC has six inputs which are controlled with switches, in this program they are named Input00-Input05. Their addresses are 0.00-0.05. There are also six digital inputs, these are built inside the PLC and only one of them is used in this program, DigitalInput004, which moves the drill down. The others are not used in this program. Digital input names are DigitalInput000-DigitalInput005. Their addresses are 1.00-1.05. There are also four digital outputs these are used for moving the sledge of the drilling station. Their names are DigitalOutput000-DigitalOutput003 and addresses 10.00-10.03. (Figure 13)

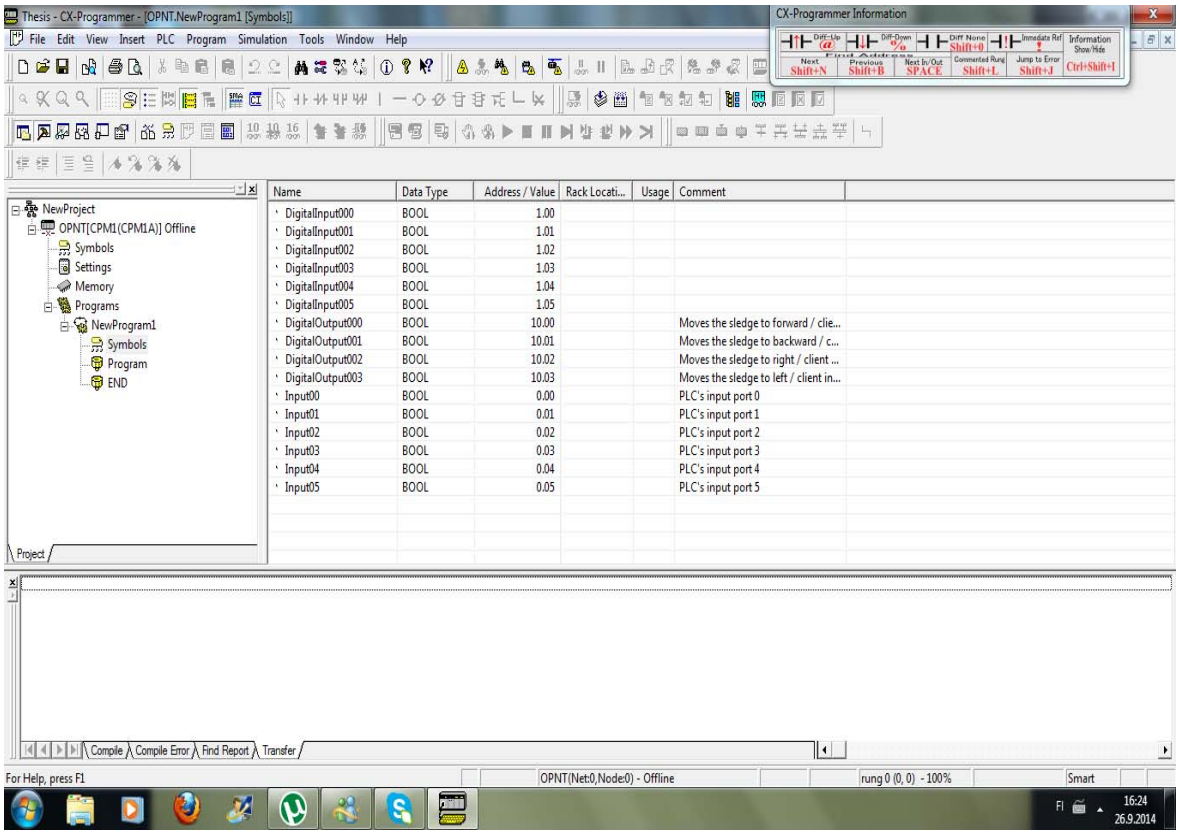

Figure 14. An I/O list of a PLC program.

The actual program is very simple (Figure 15). It is made so that Inputs00-03 move the sledge. It was developed so it is not possible to move the sledge forward and backward at the same time or left and right at the same time. It is also not possible to move the sledge when the drill is down and when Input05 is true, it is not possible to move the sledge or move the drill down. When connecting the CX-programmer to PLC, the OPC must be turned off. Otherwise you are not able to connect to the PLC with the CX-programmer.

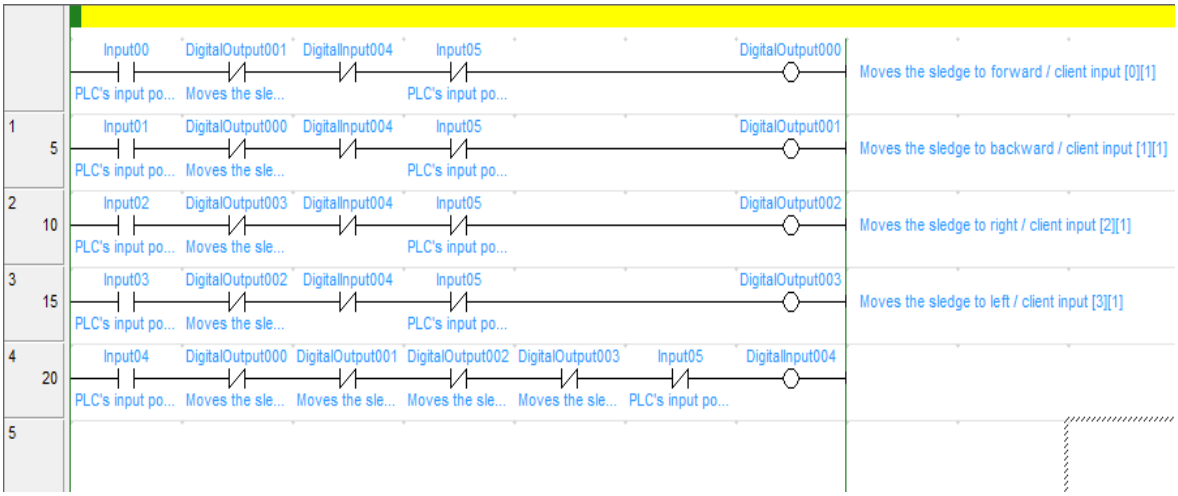

Figure 15. A program which is uploaded to the PLC.

4.1.2 OPC Server, PLC Data Gateway

Like the program for the PLC also the OPC server is very simple (Figure 16). Digital inputs and outputs in this server have the same names and addresses as the ones in the PLC program, so the OPC server is able to take those values from the PLC program and transfer them to Unity. All digital inputs are mapped to the digital inputs register block and all digital outputs are mapped to the digital outputs register block. Where they are given their address' start word. For digital inputs it is 1 and for digital outputs it is 10. To map a tag to register block it must be done individually for each tag. It can be done in properties section which is located at left side of the screen and in the bottom of the properties section is register block, there you need to write the address of the required register block. For example in this server digital inputs address is Main.OmronExample.Digital Inputs. Just above the register block it can set the bit off set and above that allow controls, which must be set to true. Every register block must be given the device's address and the device channel's address, these can be given in same section as the register block.

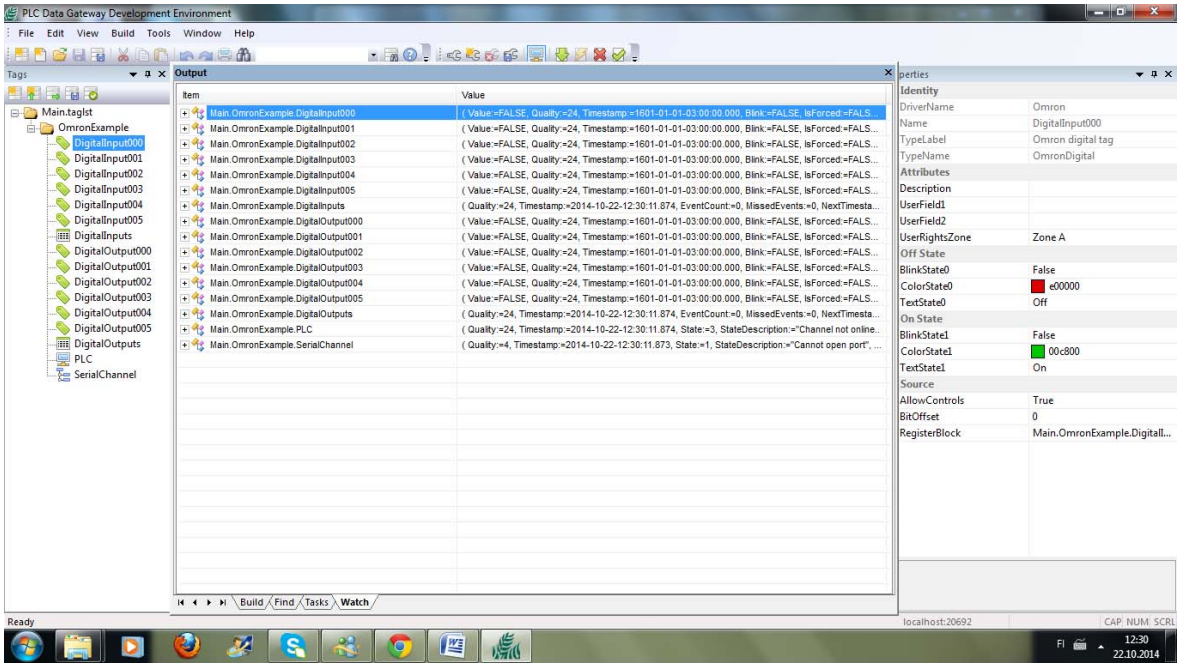

Figure 16. OPC server made with PLC Data Gateway development Environment.

4.1.3 Socket Server

Socket server is a program which captures the data from the OPC server and transfers it to Unity. In the future the socket server might be integrated to the OPC or to Unity and would not be needed any more. The socket server was created in Visual studio and is written in C#. Every time when PLC or OPC is changed, modifications to the socket servers needs to be done.

The Socket Server program consist of two modules main program and Reader class. First main program opens connection to OPC server and does all necessary initializations. (Figure 17). The whole socket server can be found in attachments.

```
Reader reader = new Reader();

IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
TcpListener tcpListener = new TcpListener(ipAddress, 8221);


OpcLabs.EasyOpc.DataAccess.EasyDAClient easyDAClient1 =
    new OpcLabs.EasyOpc.DataAccess.EasyDAClient();
easyDAClient1.ClientMode.AllowAsynchronousMethod = false;
easyDAClient1.ClientMode.AllowAsynchronousMethod = true;
easyDAClient1.ClientMode.DesiredMethod =
    OpcLabs.EasyOpc.DataAccess.DAReadWriteMethod.Synchronous;

//Wake up OPC client
easyDAClient1.ReadItemValue("", "Kepware.KEPServerEX.V5",
    "Channel1.PLC.POU_1.DigitalOutput000");

tcpListener.Start();
```

Figure 17. Socket server making a connection to OPC-server.

After that the main program waits for signals coming from the OPC server. When a signal arrives all input and output streams are opened. (Figure 18).

```
tcpListener.Start();

while (true)
{
    TcpClient tcpClient = tcpListener.AcceptTcpClient();

    NetworkStream ns = tcpClient.GetStream();
    StreamWriter sw = new StreamWriter(ns);
    StreamReader sr = new StreamReader(ns);

    sw.AutoFlush = true;

    bool stopped = false;

    while (!stopped)
    {
```

Figure 18. Socket server opening the streams.

Next the program reads the commands from the OPC server. After READ command all values from the PLC will be read. Write command means, that values are written to OPC which transfers the values to PLC. (Figure 19)

```
while (!stopped)
{
    if (ns.DataAvailable)
    {
        string command = sr.ReadLine();
        string answer;

        switch (command)
        {
            case "read DigitalOutput000":
                answer = inputs[0][1];
                break;
            // other read commands...

            case "write DigitalInput000 True":
                easyDAClient1.WriteItemValue("", "Kepware.KEPServerEX.V5",
                    "Channel1.PLC.POU_1.DigitalInput000", "True");
                answer = "";
                break;
            // other write commands

            case "quit":
                answer = "quit";
                stopped = true;
                break;
            default:
                answer = "default";
                break;
        }
        sw.WriteLine(answer);
    }
    Thread.Sleep(1);
}
```

Figure 19. Socket server command handling.

4.1.4 Unity Simulation

The Unity model that is being simulated is a drilling station which has a movable sledge (Figure 24). This sledge is moved by the PLC's input ports 0-3. This sledge is also carrying a brown cube. The drill is controlled with the PLC's input port4 and it doesn't do any actual drilling. When it comes to contact with another object during the simulation it just stops. The program inside the PLC does not allow movement and drilling at the same time.

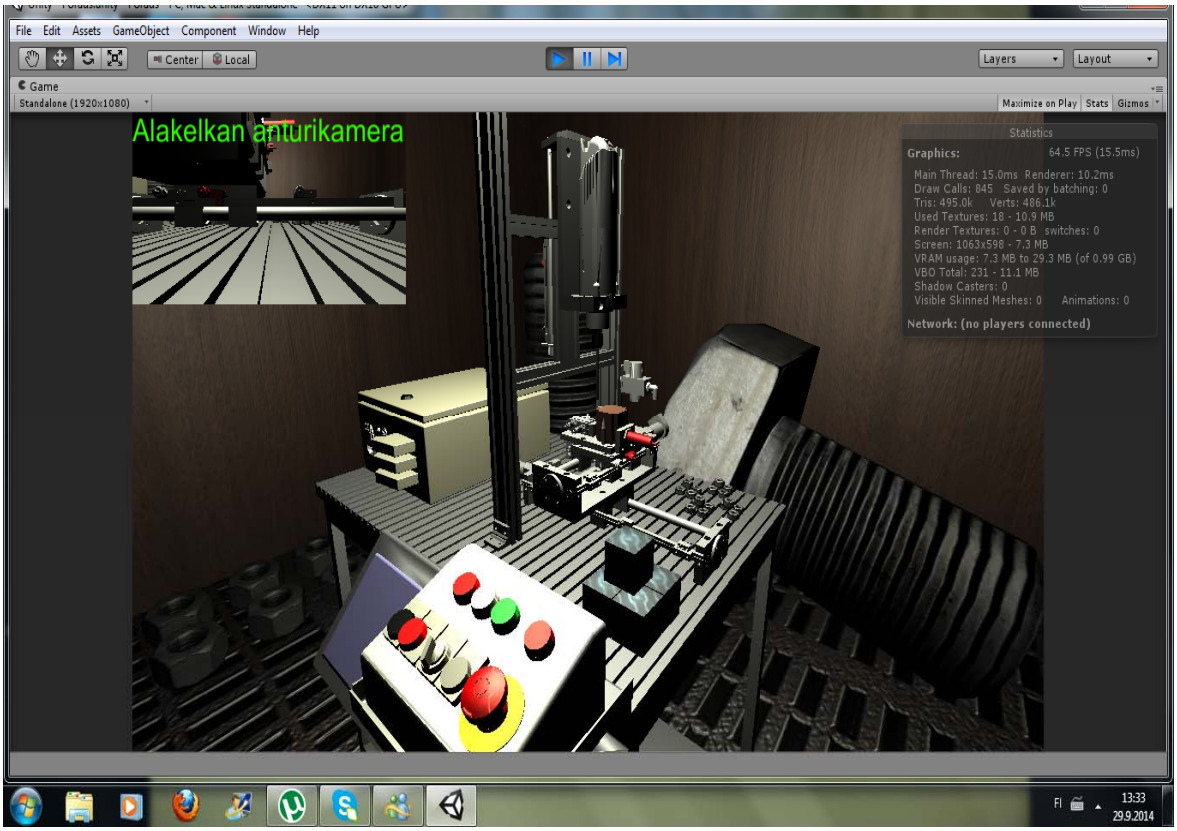

Figure 20. The 3D model in Unity.

Commands that come from PLC will be implemented to Unity simulation with a script. Also the values from Unity can be transferred to PLC with this same script. At this particular model it comes to a client script and from there these commands will be distributed to different parts of the simulation. The client opens streams and updates outputs and inputs. (Figure 21) The whole Unity client can be found in attachments.

```
    private static List<Sensori> sensorit = new List<Sensori>();


    public static string[][] inputs = new string[][]{
        new string[] {"DigitalOutput000", "False"},
        // Other commands
};

    private static bool DigitalInput000 = false;
        // Other commands

    public static void KytkeSensori(Sensori sensori)
    {
        sensorit.Add(sensori);
    }

    public static void Paivita()
    {
        TcpClient client = new TcpClient("localhost", 8221);

        // avataan streamit
        NetworkStream ns = client.GetStream();
        StreamWriter sw = new StreamWriter(ns);
        StreamReader sr = new StreamReader(ns);
        sw.AutoFlush = true;

        //päivitetään outputit
        foreach (Sensori sensori in sensorit)
        {
            switch (sensori.AnturinKytkenta)
            {
                case Sensori.InputNumber.Input_0:
                    if (sensori.Tila != DigitalInput000)
                    {
                        DigitalInput000 = sensori.Tila;
                        sw.WriteLine("write DigitalInput000 " + DigitalInput000.ToString());
                        sr.ReadLine();
                    }
                    break;
                // Other sensor inputs

        //päivitetään inputit
        for (int i = 0; i < inputs.Length; i++)
        {
            sw.WriteLine("read " + inputs[i][0]);
            inputs[i][1] = sr.ReadLine();
        }
```

Figure 21. Unity Client.

4.2 Second Setup

This second setup is almost similar to the first one, but in this setup the OPC server is Kepware's KEPServerEX5. Also the PLC is changed for this setup,

because Omron's CPM1 is not compatible with KEPServerEX5. In this setup Omron's CJ1M is used. The PLC's program and Unity model are very similar as in the first setup, but some small modifications are made, because CJ1M has more outputs available than CPM1. Also the socket server is very silmilar as in the first setup. The only thing that has to be changed is the OPC's address. In the first setup it was: (“”; “FernHillSoftware.PLCDataGateway”, “localhost.Main.Omronexample.DigitalOutput000”) for DigitalOutput000. In the second setup its address is: (“”, “Kepware.KEPServerEX.V5”, “Channel1.PLC.DigitalOutput000”).

4.2.1 OPC Server, Kepware

The interface of the KEPServerEX5 is similar to PLC Data Gateway Development environment (Figure 29). This server has five digitalinputs and five digitaloutputs, which are located in a device called PLC, which is connected to channel1.

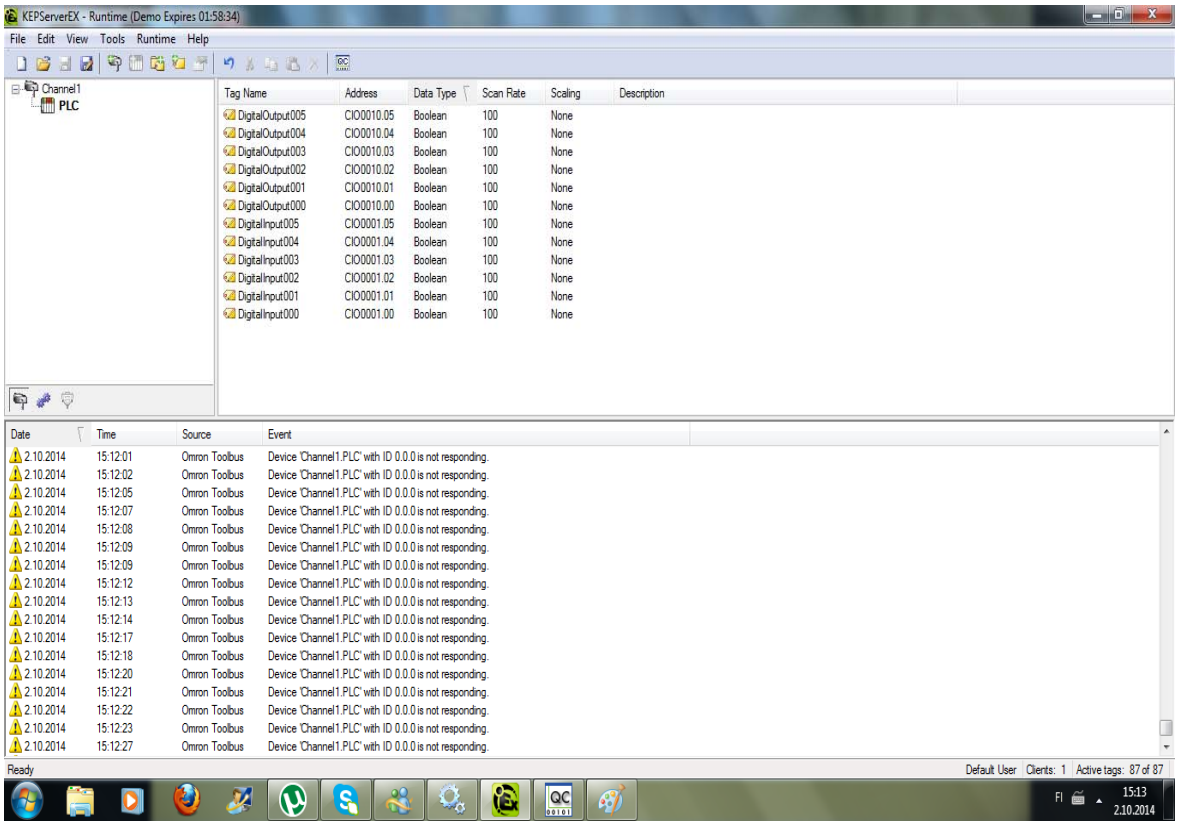

Figure 22. KEPServerEX.

In the tools tab there is “Launch OPC Quick Client” and by clicking this, OPC Quick Client can be started (Figure 30). All tag values can be monitored here. In this screen, tags’ connection quality can be checked: If it is bad, it can be improved in Channel Properties, by setting the right COM ID, baud rate, data bits, parity and stop bits. Also adjusting the request timeout in the device properties might help.

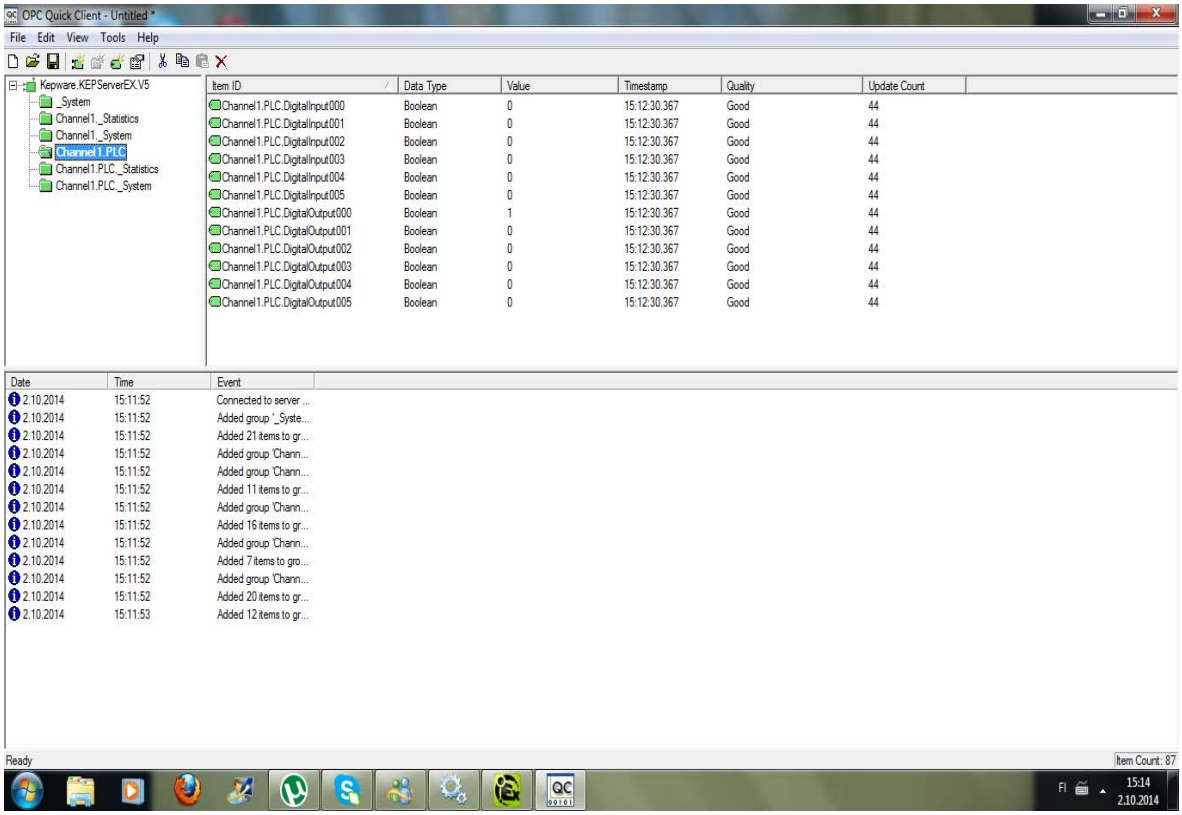

Figure 23. OPC quick client.

4.3 Third Setup

In this setup instead of using physical PLC a soft PLC was used. The soft PLC is just a software, but it has all the same functions and capabilities as the normal PLC. The soft PLC that was used was Beckhoff’s. It was programmed by using TwinCAT3 and was programmed by using ladder diagram.

```
PROGRAM POU_1
VAR
    Input00: BOOL;
    Input01: BOOL;
    Input02: BOOL;
    Input03: BOOL;
    Input04: BOOL;
    Input05: BOOL;
    DigitalOutput000: BOOL;
    DigitalOutput001: BOOL;
    DigitalOutput002: BOOL;
    DigitalOutput003: BOOL;
    DigitalOutput004: BOOL;
    DigitalOutput005: BOOL;
    DigitalInput000: BOOL;
    DigitalInput001: BOOL;
    DigitalInput002: BOOL;
    DigitalInput003: BOOL;
    DigitalInput004: BOOL;
    DigitalInput005: BOOL;
END_VAR
```

Figure 24. I/O list of the program.

Even though all these PLCs were programmed with same programming language, ladder diagram, the layout is still a little different. Especially with the I/O lists. The actual program is basically the same which was used also in Omron's PLCs.

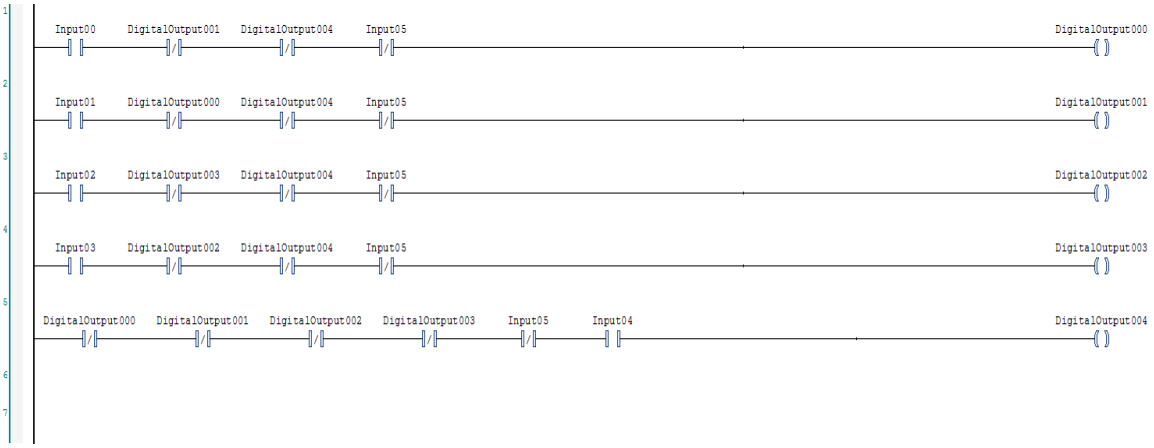

Figure 25. The program which was used in this setup.

The OPC server which was used in this setup was also Kepware's KepServerEX. The only major difference in this setup and in setup 2 is that, when using KepServerEX's Beckhoff TwinCAT driver tags cannot be manually created, they must be auto created. This can be done in device properties, database creation and auto create. All the settings must be correct otherwise this would not work.

5 Testing & Results

To determine which of these setups were most successful, some tests were done to find out which setup had the fastest response time. Setup 2 was not included in these, because it was not able to send signals from Unity back to OPC and PLC. So setup 1 and 3 were only used in these tests. The program in PLC was altered slightly for these experiments. The program was made so that it moves the sledge in the Unity model to the left until it hits a sensor. When this sensor is activated the sledge changes its direction. The picture below is the program made for the Beckhoff, but the program for the Omron is basically same.

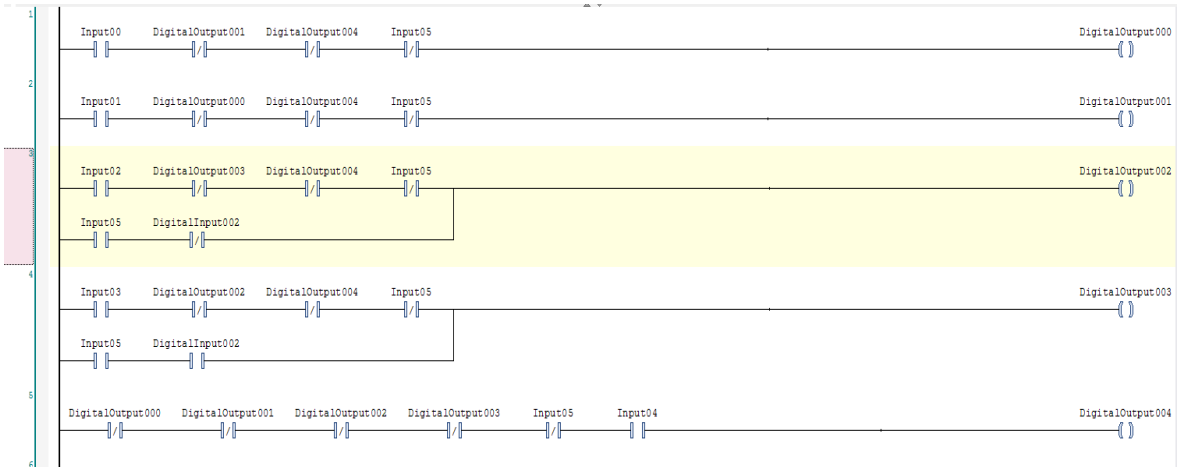

Figure 26. The program made for testing.

The interval between activating sensor and moving direction is measured in these experiments. The measurements were done in Unity, by adding Debug.Log twice in the script. So it gave system time with accuracy of a millisecond when sensor was activated and when it changed direction. Then the response time was manually calculated with these two times.

```
if (Input.GetAxis("KelkkaHorizontal") > 0 || Client.inputs[3][1].Equals("True"))

{

    movingState = MovingState.Left;

    DateTime now = DateTime.Now;

    string Time = now.Hour.ToString() + ":" + now.Minute.ToString() + ":" + now.Second.ToString() + ":" + now.Millisecond.ToString ();

    Debug.Log (Time);

}
```

Figure 27. A code inserted to ”kelkka ylempi” script, which gives a time when the sledge changes direction.

```
if (AnturinKytkenta == InputNumber.Input_2)

{

    DateTime now = DateTime.Now;

    string Time = now.Hour.ToString() + ":" + now.Minute.ToString() + ":" + now.Second.ToString() + ":" + now.Millisecond.ToString ();

    Debug.Log (Time);

}
```

Figure 28. A code inserted to ” Anturit” script, which gives a time when the sensor is activated.

These tests were performed twenty times for both setups. These setups were also made in different PCs, so the results will not be fully comparable, but they will be directional. The results are displayed in milliseconds. Also the first setup failed twice in changing direction when the sensor was activated. This resulted from the fact that, the first setup’s response time was so slow that the sledge was able to pass the sensor before the signal to change direction came to Unity.

PLC	BECKHOFF	OMRON
TEST 1	49	825
TEST 2	55	1122
TEST 3	49	622
TEST 4	49	801
TEST 5	65	639
TEST 6	33	534
TEST 7	49	465
TEST 8	33	935
TEST 9	49	638
TEST 10	66	699
TEST 11	66	1366
TEST 12	49	886
TEST 13	33	915
TEST 14	33	798
TEST 15	49	733
TEST 16	49	493
TEST 17	49	835
TEST 18	66	689
TEST 19	65	1182
TEST 20	65	784
Average	51,05	798,05
Highest	66	1366
Lowest	33	465

As the results show the third setup was almost 16 times faster than the first setup. The third setup’s average response time was about 50ms, when the first setup’s average response time was about 800ms. The big time difference most likely originates from the fact that in the third setup there was a soft PLC, which makes the response time so fast that it is possible to simulate pulse sensors in Unity.

6 Summary

The outcome of this project was somewhat surprising. It was expected, that the soft PLC would be much faster than Omron’s CPM1. The biggest surprise was how slow the normal PLC was. Its response time was about 800ms, when its response time was expected to be about 500ms. Also the response time of Beckhoff’s soft PLC was shorter than expected.

Unfortunately Omron’s CJ1M did not work properly. It would have been interesting to see how long its response time would have been. Most likely its response time would been somewhere between CPM1 and the soft PLC. Overall these tests gave a good knowledge about how much the response times depend on the PLC and OPC server type.

This subject was overall very interesting and most likely very beneficial. It is possible that these kinds of virtual models will become more popular in the near future. Also the use of Unity will most likely increase in the future. At the moment Unity is mainly used for making video games. Therefore, it has lots of potential if these kinds of automation applications become more popular. It is also possible that the upcoming Unity 5 has some functions which would ease the creating of these kinds of setups.

Sources

Aalto-yliopisto.2003 Siirtorekisterit (Web Page). Aalto-yliopisto Sähkötekniikan korkeakoulu. (Ref. 15.10.2014).Available: http://legacy.spa.aalto.fi/sig-legacy/digis/luento11/siirto.html

Advosol. No date. OPCHDA.NET. (Web Page). Advosol Inc. (Ref 15.10.2014). Available: https://advosol.com/pc-11-3-opchdanet.aspx

Blender. No date. About. (Web Page). Blender Foundation. (Ref 22.10.2014). Available: http://www.blender.org/about/

Envato Pty Ltd. 2014. Introduction to Unity 3D. (Web Page). Envato Pty Ltd. (Ref. 16.9.2014). Available: http://code.tutsplus.com/tutorials/introduction-to-unity3d--mobile-10752

Honeywell International inc.2014 Specification Downloads. (Web Page). Honeywell International inc. (Referenced16.9.2014).Available: http://www.matrikonopc.com/downloads/types/specifications/index.aspx

Keinänen, T. Kärkkäinen, P. Metso, T. & Putkonen, K. 2001 Logiikat ja ohjausjärjestelmät koneautomaatio 2. Porvoo: WSOY

Kronotech. No date. PLC languages. (Web Page). Kronotech. (Ref. 7.10.2014). Available: http://www.kronotech.com/PLC/Languages.htm

Mevea. 2013. Tuotteet ja Palvelut. (Web Page). Mevea ltd. (Ref. 15.10.2014). Available: http://www.mevea.com/fi/tuotteet_ja_palvelut

Mevea. 2013. Yrityksen tausta. (Web Page). Mevea ltd. (Ref. 16.10.2014). Available: http://www.mevea.com/fi/yritys/historia

NVIDIA Corporation. 2014. PhysX FAQ. (Web Page). NVIDIA Corporation. (Ref 16.9.2014) Available: http://www.nvidia.com/object/physx_faq.html

OPC foundation.2014 Unified Architecture. (Web Page). OPC Foundation (Ref 16.9.2014). Available: https://opcfoundation.org/about/opc-technologies/opc-ua/

OPC programmers' connection. 2013 OPC Unified Architecture. (Web Page). OPC Programmes' Connection. (Ref 16.9.2014) Available: http://www.opcconnect.com/ua.php

OPC Training Institute. 2007-2014 OPC .NET (OPC Xi) Specification. (Web Page). OPC Training Institute. (Ref 16.9.2014) Available: http://www.opcti.com/opc-xi-specification.aspx

Scott Richmond. 2013. Unity and Visual Studio 2012. (Web Page). Scott Richmond. (Ref. 26.9.2014). Available: http://www.strichnet.com/unity-4-and-visual-studio-2012/

Suomen Automaatioseura ry. No date. OPC:n uudet tuulet (PDF-file). Suomen Automaatioseura ry. (Ref. 16.9.2014) Available: http://www.automaatioseura.fi/index/tiedostot/OPC_Lisatieto.pdf

Tapio Hellman. 2014. CAVE visualisointiluola. (PDF-file). Seinäjoki University of Applied Sciences. (Ref. 17.9.2014). Available: http://www.mad.fi/mad/tilapaiset/ArchiMAD_3_2014_cave.pdf

Tuhola, E. & Viitanen, K. 2008 Logiikat 3D-mallintaminen suunnittelun apuvälineenä. Jyväskylä: Gummerus Kirjapaino Oy 16.9

Unity Technologies. 2014. EFFORTLESSLY UNLEASH YOUR GAME ON THE WORLD'S HOTTEST PLATFORMS. (Web Page). Unity Technologies. (Ref 16.9.2014). Available: http://unity3d.com/unity/multiplatform/

Unity Technologies. 2014. FREQUENTLY ASKED QUESTIONS. (Web Page). Unity Technologies. (Ref. 16.9.2014). Available: http://unity3d.com/unity/faq

Unity Technologies. 2014. Whats new. (Web Page). Unity technologies. (Ref 16.9.2014). Available: http://unity3d.com/unity/whats-new

Unity Technologies. Unity 5. (Web Page). Unity Technologies. (Ref. 16.9.2014). Available: http://unity3d.com/5

APPENDICES

APPENDIX 1. Unity client 1/4.

```
using UnityEngine;
using System.IO;
using System.Net.Sockets;
using System.Collections.Generic;

public static class Client
{
    private static List<Sensori> sensorit = new List<Sensori>();

    public static string[][] inputs = new string[][] {
        new string[] {"DigitalOutput000", "False"},
        new string[] {"DigitalOutput001", "False"},
        new string[] {"DigitalOutput002", "False"},
        new string[] {"DigitalOutput003", "False"},
        new string[] {"DigitalInput004", "False"},
        new string[] {"DigitalInput005", "False"},
    };

    private static bool DigitalInput000 = false;
    private static bool DigitalInput001 = false;
    private static bool DigitalInput002 = false;
    private static bool DigitalInput003 = false;
    private static bool DigitalInput004 = false;
    private static bool DigitalInput005 = false;


    public static void KytkeSensori(Sensori sensori)
    {
        sensorit.Add(sensori);
    }

    public static void Paivita()
    {
        TcpClient client = new TcpClient("localhost", 8221);
```

APPENDIX 2. Unity client 2/4.

```
TcpClient client = new TcpClient("localhost", 8221);

// avataan streamit
NetworkStream ns = client.GetStream();
StreamWriter sw = new StreamWriter(ns);
StreamReader sr = new StreamReader(ns);
sw.AutoFlush = true;

//päivitetään outputit
foreach (Sensori sensori in sensorit)
{
    switch (sensori.AnturinKytkenta)
    {
        case Sensori.InputNumber.Input_0:
            if (sensori.Tila != DigitalInput000)
            {
                DigitalInput000 = sensori.Tila;
                sw.WriteLine("write DigitalInput000 " + DigitalInput000.ToString());
                sr.ReadLine();
            }
            break;
        case Sensori.InputNumber.Input_1:
            if (sensori.Tila != DigitalInput001)
            {
                DigitalInput001 = sensori.Tila;
                sw.WriteLine("write DigitalInput001 " + DigitalInput001.ToString());
                sr.ReadLine();
            }
            break;
        case Sensori.InputNumber.Input_2:
            if (sensori.Tila != DigitalInput002)
            {
                DigitalInput002 = sensori.Tila;
                sw.WriteLine("write DigitalInput002 " + DigitalInput002.ToString());
```

APPENDIX 3. Unity client 3/4.

```
    if (sensori.Tila != DigitalInput002)
    {
        DigitalInput002 = sensori.Tila;
        sw.WriteLine("write DigitalInput002 " + DigitalInput002.ToString());
        sr.ReadLine();
    }
    break;
case Sensori.InputNumber.Input_3:
    if (sensori.Tila != DigitalInput003)
    {
        DigitalInput003 = sensori.Tila;
        sw.WriteLine("write DigitalInput003 " + DigitalInput003.ToString());
        sr.ReadLine();
    }
    break;
case Sensori.InputNumber.Input_4:
    if (sensori.Tila != DigitalInput004)
    {
        DigitalInput004 = sensori.Tila;
        sw.WriteLine("write DigitalInput004 " + DigitalInput004.ToString());
        sr.ReadLine();
    }
    break;
case Sensori.InputNumber.Input_5:
    if (sensori.Tila != DigitalInput005)
    {
        DigitalInput005 = sensori.Tila;
        sw.WriteLine("write DigitalInput005 " + DigitalInput005.ToString());
        sr.ReadLine();
    }

    break;
```

APPENDIX 4. Unity client 4/4.

```
                    }

                    break;
                }
            }

            //päivitetään inputit
            for (int i = 0; i < inputs.Length; i++)
            {
                sw.WriteLine("read " + inputs[i][0]);
                inputs[i][1] = sr.ReadLine();
            }

            // katkaistaan yhteys
            sw.WriteLine("quit");
            sr.ReadLine();

            // suljetaan streamit
            sr.Close();
            sw.Close();
            ns.Close();
            client.Close();
        }
    }
```

APPENDIX 5. Socket Server reader 1/2.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ConsoleApplication1
{
    public class Reader
    {
        public Reader()
        {
            Thread thread = new Thread(new ThreadStart(Paivita));
            thread.Start();
        }

        private void Paivita()
        {
            OpcLabs.EasyOpc.DataAccess.EasyDAClient easyDAClient1 = new OpcLabs.EasyOpc.DataAccess.EasyDAClient();
            easyDAClient1.ClientMode.AllowAsynchronousMethod = false; // these three lines might be unesessary.
            easyDAClient1.ClientMode.AllowAsynchronousMethod = true;
            easyDAClient1.ClientMode.DesiredMethod = OpcLabs.EasyOpc.DataAccess.DAReadWriteMethod.Synchronous;

            //Wake up OPC client
            easyDAClient1.ReadItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalOutput000");

            while (true)
            {
```

APPENDIX 6. Socket server reader 2/2.

```
public class Reader
{
    public Reader()
    {
        Thread thread = new Thread(new ThreadStart(Paivita));
        thread.Start();
    }

    private void Paivita()
    {
        OpcLabs.EasyOpc.DataAccess.EasyDAClient easyDAClient1 = new OpcLabs.EasyOpc.DataAccess.EasyDAClient();
        easyDAClient1.ClientMode.AllowAsynchronousMethod = false; // these three lines might be unesessary.
        easyDAClient1.ClientMode.AllowAsynchronousMethod = true;
        easyDAClient1.ClientMode.DesiredMethod = OpcLabs.EasyOpc.DataAccess.DAReadWriteMethod.Synchronous;

        //Wake up OPC client
        easyDAClient1.ReadItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalOutput000");

        while (true)
        {
            Program.inputs[0][1] = easyDAClient1.ReadItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalOutput000").ToString();
            Program.inputs[1][1] = easyDAClient1.ReadItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalOutput001").ToString();
            Program.inputs[2][1] = easyDAClient1.ReadItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalOutput002").ToString();
            Program.inputs[3][1] = easyDAClient1.ReadItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalOutput003").ToString();
            Program.inputs[4][1] = easyDAClient1.ReadItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalInput004").ToString();
            Program.inputs[5][1] = easyDAClient1.ReadItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalInput005").ToString();
        }
    }
}
}
```

APPENDIX 7. Socket Server program 1/5.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
    static class Program
    {
        public static string[][] inputs = new string[][] {
        new string[] {"DigitalOutput000", "False"},
        new string[] {"DigitalOutput001", "False"},
        new string[] {"DigitalOutput002", "False"},
        new string[] {"DigitalOutput003", "False"},
        new string[] {"DigitalInput004", "False"},
        new string[] {"DigitalInput005", "False"},
};

        static void Main(string[] args)
        {
            Reader reader = new Reader();

            IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
            TcpListener tcpListener = new TcpListener(ipAddress, 8221);

            OpcLabs.EasyOpc.DataAccess.EasyDAClient easyDAClient1 = new OpcLabs.EasyOpc.DataAccess.EasyDAClient();
```

APPENDIX 8. Socket Server program 2/5.

```
//Wake up OPC client
easyDAClient1.ReadItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalOutput000");

tcpListener.Start();

while (true)
{
    TcpClient tcpClient = tcpListener.AcceptTcpClient();

    NetworkStream ns = tcpClient.GetStream();
    StreamWriter sw = new StreamWriter(ns);
    StreamReader sr = new StreamReader(ns);

    sw.AutoFlush = true;

    bool stopped = false;

    while (!stopped)
    {
        if (ns.DataAvailable)
        {
            string command = sr.ReadLine();
            string answer;

            switch (command)
```

APPENDIX 9. Socket Server program 3/5.

```
switch (command)
{
    case "read DigitalOutput000":
        answer = inputs[0][1];
        break;
    case "read DigitalOutput001":
        answer = inputs[1][1];
        break;
    case "read DigitalOutput002":
        answer = inputs[2][1];
        break;
    case "read DigitalOutput003":
        answer = inputs[3][1];
        break;
    case "read DigitalInput004":
        answer = inputs[4][1];
        break;
    case "read DigitalInput005":
        answer = inputs[5][1];
        break;

    case "write DigitalInput000 True":
        easyDAClient1.WriteItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalInput000", "True");
        answer = "";
        break;
    case "write DigitalInput000 False":
        easyDAClient1.WriteItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalInput000", "False");
        answer = "";
        break;
    case "write DigitalInput001 True":
        easyDAClient1.WriteItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalInput001", "True");
```

APPENDIX 10. Socket Server program 4/5.

```
case "write DigitalInput001 True":
    easyDAClient1.WriteItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalInput001", "True");
    answer = "";
    break;
case "write DigitalInput001 False":
    easyDAClient1.WriteItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalInput001", "False");
    answer = "";
    break;
case "write DigitalInput002 True":
    easyDAClient1.WriteItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalInput002", "True");
    answer = "";
    break;
case "write DigitalInput002 False":
    easyDAClient1.WriteItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalInput002", "False");
    answer = "";
    break;
case "write DigitalInput003 True":
    easyDAClient1.WriteItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalInput003", "True");
    answer = "";
    break;
case "write DigitalInput003 False":
    easyDAClient1.WriteItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalInput003", "False");
    answer = "";
    break;
case "write DigitalInput004 True":
    easyDAClient1.WriteItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalInput004", "True");
    answer = "";
    break;
case "write DigitalInput004 False":
    easyDAClient1.WriteItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalInput004", "False");
    answer = "";
    break;
```

APPENDIX 11. Socket Server program 5/5.

```
                            answer = "";
                            break;
                        case "write DigitalInput005 True":
                            easyDAClient1.WriteItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalInput005", "True");
                            answer = "";
                            break;
                        case "write DigitalInput005 False":
                            easyDAClient1.WriteItemValue("", "FernhillSoftware.PLCDataGateway", "localhost.Main.OmronExample.DigitalInput005", "False");
                            answer = "";
                            break;

                        case "quit":
                            answer = "quit";
                            stopped = true;
                            break;
                        default:
                            answer = "default";
                            break;
                    }
                    sw.WriteLine(answer);
                }
                Thread.Sleep(1);
            }

            sr.Close();
            sw.Close();
            ns.Close();
            tcpClient.Close();
        }
    }
}
}
```