BACHELOR

Hardware-in-the-loop simulation using a digital twin

van Eijk, J.H.M. (Yann)

Award date:
2022

Department of Mechanical Engineering
Control Systems Technology

Hardware-in-the-loop simulation using a digital twin

Bachelor End Project Report

Author: J.H.M. van Eijk
Student-ID: 1454447

Supervisors:
dr. ir. J.M. van de Mortel-Fronczak
ir. L. Moormann
prof. dr. ir. J.E. Rooda

June 30, 2022

Abstract

In collaboration with Rijkswaterstaat, the Control Systems Technology group of Eindhoven University of Technology is investigating the applicability of synthesis-based engineering in the design process of supervisory controllers for bridges, waterways and tunnels. The task of a supervisory controller is to control the different components and subsystems such that the entire system, called the plant, behaves as intended. The synthesis-based engineering method uses simulations with models of the plant. One of these simulations is Hardware-in-the-loop simulation. Hardware-in-the-loop validates the realized and implemented controller on a model of the plant.

The Swalmen tunnel, which is a road tunnel, is part of the A73 highway network and is in use since 2008. In a previous project, a digital twin of the Swalmen tunnel has been developed. A digital twin is a virtual copy of a plant that can be used for simulations. The Control Systems Technology group wants to perform Hardware-in-the-loop simulations using the digital twin of the Swalmen tunnel, to provide a better understanding of the Swalmen tunnel and its control system. Moreover, using a digital twin in Hardware-in-the-loop simulations makes it possible to perform operator training in an early stage within the design process. This report describes the methods used to perform Hardware-in-the-loop simulations using a digital twin, specifically the digital twin of the Swalmen tunnel. Additionally, adaptations are made to the Swalmen tunnel digital twin to allow for early stage operator training.

The digital twin is developed using Unity, which is a 3D game engine. The Prespective plugin is used within Unity for signal communication with the Ignition OPC UA server. In general, the Ignition OPC UA server connects the different components needed for Hardware-in-the-loop simulation (Graphical User Interface, digital twin and supervisory controller). The supervisory controller runs on a Programmable Logic Controller, which is a highly robust and reliable industrial computer. The Graphical User Interface has been made using the Ignition software in earlier work. This report describes the method used to connect the digital twin to the Ignition OPC UA server, and the method used to exchange data between the supervisory controller and the digital twin using OPC tags. In addition, a method is described to adapt the components of the Swalmen tunnel digital twin to perform Hardware-in-the-loop simulations. These methods have been validated using a validation test plan.

Some adaptations are made to the Swalmen tunnel digital twin to allow for operator training. Some of these adaptations have been made to improve the performance of the digital twin, measured in frames per second. Other adaptations have been made to improve the engagement of the operator training. The methods of developing these adaptations are described in this report.

The method of adding the Swalmen tunnel digital twin to the Hardware-in-the-loop setup is applicable for all digital twins made within Unity. The only limitation of the method is that it only works for the PCs provided by Rijkswaterstaat. The limiting factor that prevents an arbitrary PC to be added to the Hardware-in-the-loop setup is an error related to the creation of client certificates within Unity. The digital twin of the Swalmen tunnel is not ready for operator training. More adaptations are needed to prepare the digital twin of the Swalmen tunnel for operator training.

Contents

Chapter 1

Introduction

In collaboration with Rijkswaterstaat (RWS), the Control Systems Technology (CST) group in the department of Mechanical Engineering of the Eindhoven University of Technology is researching supervisory controller design. The task of a supervisory controller is to control the different components and subsystems such that the entire system, called the plant, behaves as intended [1]. RWS, which is the Department of Waterways and Public Works, will be renovating or replacing many parts of its infrastructure in the upcoming years. RWS is seeking methods for modularization and standardization to increase quality and evolvability, decrease life-cycle costs, and enable so-called smart mobility [1]. With the so-called synthesis-based engineering method, it is possible to automatically synthesize supervisors that limit the behavior of a plant such that the given requirements are fulfilled. Since this process automatically synthesizes the supervisor based on requirements, the need for manually creating a supervisor is nonexistent. This decreases the chance of general coding mistakes and thus increases the quality of the supervisor. In addition, the process is highly standardized, being the same regardless of the type of plant.

The CST group researches the implementation of synthesis-based controller design technology in the design of supervisory controllers for bridges, waterway locks, and tunnels [1]. A part of the synthesis-based engineering method is to validate the supervisory controller using Hardware-in-the-loop (HIL) simulations, in which a realized and implemented controller is used to control a virtual system of a plant. A plant is a system, usually having actuators and sensors, that is being controlled [2]. With HIL simulations, it becomes possible to validate the realized controller without the need of a physical plant. In this project, the focus lies on performing HIL simulations using a digital twin (DT), also known as a virtual twin. A DT is the virtual representation of a physical system [3]. According to [4], the benefits of using DTs in the design process are increased product understanding, performance, and reduced design cycle time and cost. A DT in general can thus be used within the design process to achieve the results desired by RWS if applied properly. Using a DT in the design process of supervisory controllers can lead to increased quality and evolvability as well as to decreased life-cycle costs. When a DT is placed within a HIL simulation, the possibility of validating the realized controller is present. DTs and HIL simulations both have the capacity to provide a better understanding of the product when still in the design phase. For RWS and the CST group this could provide a better understanding of the plant as a whole using the DT, and a better understanding of the supervisory controller using the DT in a HIL simulation. In addition, using the DT in a HIL simulation enables the possibility for early stage operator training. To allow for this, the DT should be adjusted to improve the quality of the operator training. This project also touches on how to adjust an existing DT to allow for operator training.

Figure 1.1: Birds-eye view of the Swalmen tunnel [5].

As mentioned before, this project looks into performing HIL simulations using a DT, as well as improving an existing DT for operator training. In order to do so, the DT of the Swalmen tunnel (Swalmen road tunnel) is used. This DT has been developed in a previous project, described in [5]. The Swalmen tunnel is a road tunnel near the village of Swalmen in the Netherlands. It is part of the A73 highway network and is in use since 2008. Its renovation process started in 2022 [6]. The tunnel consists of 2 traffic tubes, each being approximately 400 meters long and having 4.5 meters of headroom [6]. This tunnel is under the supervision of RWS [7], and because of its planned renovation and the existence of a DT, it is suitable to use as a case study. Figure 1.1 shows a birds-eye view of the Swalmen tunnel, which has been taken from [5].

The Swalmen tunnel case is part of a larger research that the CST group is conducting in collaboration with RWS. A lot of work already has been done for this collaborative project, but the most relevant work for this project is with regards to the Swalmen tunnel itself. In previous work, a DT of the Swalmen tunnel has been successfully built to validate the supervisory controller for the Swalmen tunnel using model simulation [5]. The supervisory controller has been synthesized as part of an internship for RWS [8], and thus is already present for this project. [5] describes the method of developing a DT, the relevance of using a DT for controller validation, and shows the first validation stage of the full system. In this project, this model is adapted to fit the requirements set for operator training and to be able to perform HIL simulations for the second validation stage.

At the start of this project a HIL setup was already present for 2D simulations of the Swalmen tunnel using Ignition [9], which is a software platform for creating custom Graphical User Interface (GUI) applications and supervisory control and data acquisition (SCADA) applications. This means that a basis was already present. The initial HIL setup does not allow for the use of DTs. The main challenge of this project is thus to connect a DT to the existing HIL setup.

1.1 Research goals

The main goal of this project is to adapt the initial HIL setup and Swalmen tunnel DT developed in [5], in order to perform HIL simulations with the Swalmen tunnel DT. In addition, the Swalmen tunnel DT needs to be developed further in order to be used for operator training. HIL simulations are needed to validate the supervisory controller and can eventually be used for operator training. In order to make this process repeatable for future endeavors, it is important that the documentation on how to adapt the initial HIL setup to allow for HIL simulations using DTs is well documented, as well as the process of developing the DT to allow for operator training. Short descriptions of the research goals to achieve the main goal are given below.

1. Describe how to adjust the existing HIL setup to allow for HIL simulations using a DT.
2. Perform HIL simulations with the DT of the Swalmen tunnel.
3. Adjust the Swalmen tunnel DT such that it is suitable for operator training, and describe the methods used to adjust the Swalmen tunnel DT.

The first research goal is evaluated by the existence of a step-by-step guide. It is important to document how to adjust the existing HIL setup to allow for HIL simulations using a DT, since RWS and the CST group might want to perform HIL simulations using different HIL setups than the one used in this project. To improve the clarity of this documentation, a clear step-by-step guide is desired.

The second research goal is to perform the steps from the guide made for the first research goal. This is to validate the correctness of the step-by-step guide, and to proof that HIL simulations using a DT is possible using the methods described in this report.

The third research goal is to propose adjustments to be made to the Swalmen tunnel DT to allow for operator training. It is important to describe the methods used to make it possible to implement these changes in other DTs.

1.2 Report structure

In order to reach the set research goals, it is important to understand the background of the project. Therefore, the report starts with a background chapter, after which the research goals are worked out. Finally, conclusions and recommendations are given based on the results of the research goals.

Chapter 2 provides background information on supervisory control and the engineering method behind a supervisory controller called synthesis-based engineering. Furthermore, the applications, limitations and design of DTs are discussed as well as HIL simulations and PLCs. In addition, the hardware and software components used during this project are discussed in this chapter.

After giving insight in the relevant background topic in Chapter 2, the development process of creating a HIL setup for a Unity DT is discussed in Chapter 3. This is done by giving an overview of the initial HIL setup used for the 2D simulation of the Swalmen tunnel and discussing the expected challenges when modifying the initial setup to allow for HIL using a Unity DT . After this, the problems encountered with connecting the Unity DT to the Ignition OPC UA server and their proposed solutions are discussed. In addition the method of exchanging data between the ABB PLC and the Unity DT is explained, as well as a not functioning alternative to this method. Finally, a conclusion is given on the process of adapting the HIL setup and the method used to exchange data between the ABB PLC and the Unity DT.

Chapter 4 discusses the adjustments made to the Swalmen tunnel DT to allow for HIL simulations. This is done by explaining the techniques used in [5] to develop controllable components of the DT and the adjustments needed and made to allow for HIL simulations. Afterwards, the method used to adapt the controllable components in [5] is explained with the example of a boom barrier. A validation of and a conclusion on this method is given in the same chapter.

Chapter 5 discusses the adaptation made to the Swalmen tunnel DT to allow for operator training. These adaptations are related to the CCTV system, the use of 2 monitors during operator training and dynamic traffic. Each of these adaptations is explained on why they were implemented and how they were implemented. A short conclusion is given on the process of adapting the DT for operator training.

The final chapter, Chapter 6 gives final conclusions and recommendations based on the results obtained in the previous chapters. This chapter offers a critical reflection on the project as well as suggestions for future work.

Chapter 2

Preliminaries

In this chapter, an introduction is given to basic concepts of supervisory control and associated engineering methods as well as HIL simulations. This chapter contains relevant information on HIL simulations with respect to synthesis-based controller design techniques. A section is also dedicated to discussing the use of DTs, and another section is dedicated to introducing Programmable Logic Controllers (PLCs). In the end, the hardware and software used for this report are discussed.

2.1 Supervisory control

Since the controller used in the HIL simulation is a supervisory controller, some background information is given on supervisory control. As mentioned in the introduction, the task of a supervisory controller is to control the different components and subsystems such that the entire system, called the plant, behaves as intended [1]. Figure 2.1 gives a schematic overview of an infrastructural control system. Here, an operator interacts with a (GUI) to control actuators via a supervisory controller. The GUI in turn visualizes the status of the plant based on data from the supervisory controller. As Figure 2.1 demonstrates, a supervisory controller is the layer between the actuators and sensors and the GUI [1].

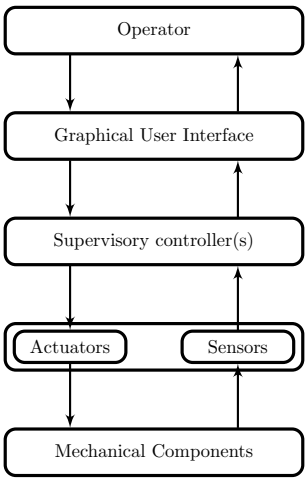

Figure 2.1: Schematic view of a control system structure of an infrastructural system [1].

A supervisory controller is able to block signals to the actuator to prevent violating the controller requirements, these signals are controllable. In contrast, signals from sensors are uncontrollable and are always able to occur [1]. A supervisory controller is thus able to prevent unwanted behavior of the plant. Unwanted behavior can be characterized as unsafe, uncontrollable, or system-blocking behavior. E.g. a boom barrier should not be allowed to close when an object is present in its path.

2.2 Engineering methods

The traditional practice of engineering supervisory controllers is to code them manually, based on control requirements [10]. A schematic overview of the traditional engineering method can be found in Figure 2.2. Here, each step is denoted with an arrow and each symbol represents the product of that step. A document icon indicates a documented product, and a square indicates a realized product [11].

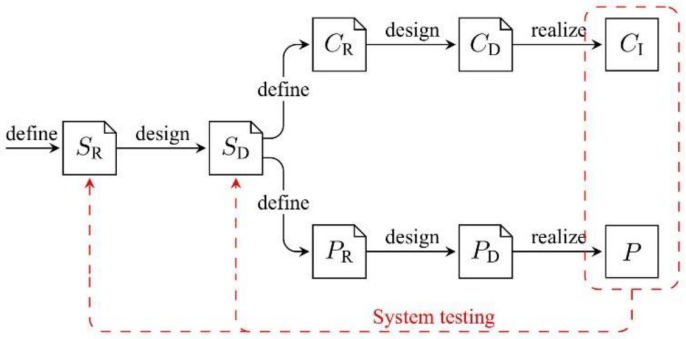

Figure 2.2: Schematic overview of the traditional engineering method [11].

First, a set of system requirements is defined at the beginning of the process (S_R) after which a document-based design is made (S_D). Now, the method splits the system into 2 parts, the controller (C) and the plant (P). These both follow a similar path in their design method. First, the controller and plant requirements are set (C_R and P_R respectively), after which a document-based design is made (C_D and P_D) and finally, both the controller and the plant are realized (C_I and P) [11]. It is only at the final step that system testing can be performed, and flaws in the design can be caught. This usually leads to either a system redesign or even a refinement of the systems requirements. It is apparent that this process can be time consuming and error prone due to the lack of testing options.

Synthesis-based engineering is able to solve the main problem relevant for this report of the traditional engineering method, which is the fact that the plant cannot be tested before realization. A schematic overview of synthesis-based engineering can be found in Figure 2.3.

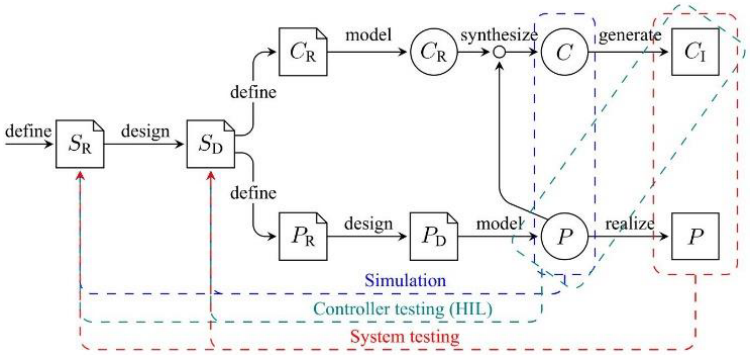

Figure 2.3: Schematic overview of the synthesis-based engineering method [11]

As one can see, the traditional method is expanded with models which are indicated by circles. Having these models present allow for testing before realization of the plant. After forming requirements for the controller (C_R) and plant (P_R), a design is made for the plant (P_D). The next step is to create mathematical models for both the controller requirements (C_R) and the plant (P), from which a supervisory controller C is synthesized. In order to perform simulation, a hybrid plant (P) is needed that allows for continuous-time behavior [12]. This hybrid plant could, for example, be a DT. At this stage, the synthesized controller and

the hybrid plant can be used as a first stage to test the controller. If inaccuracies are found within this simulation, it is possible to move back to an earlier stage to counter them. This already is a big advantage over the traditional engineering method. After the model simulation, the controller can be generated and implemented on stand-alone hardware. The model is now ready for the second stage of validation, the HIL simulation. With HIL, the realized controller can be tested on the hybrid plant to validate the controller once more. This is needed since small changes might be present in the realized controller compared to the controller model. This might be due to a difference between the software used to run the controller model and the realized controller. If there are no flaws detected in this stage, the plant can be realized and the last validation stage, called system testing, can take place. This is the same as the last step in Figure 2.2, but with the synthesis-based engineering method, the chances of flaws being present in the final supervisory controller and plant are greatly reduced.

2.3 Digital twin applications

This project revolves around the use of a DT of a road tunnel for HIL simulations and operator training. A DT is described as *a "Virtual and computerized counterpart of a physical system that can be used to simulate it for various purposes"* [13]. In other words, a DT is a virtual copy of a real-life system that can be used as its simulation model. In the case of this project, the DT is used as a plant simulation model in the HIL set-up. This allows for controller validation and operator training.

The Centrum voor Ondergrond Bouwen (COB) is a network organization aimed at collecting, developing and opening up knowledge about and related to the use of underground space. They have published a manual on their website aimed at explaining the use of digital validation and verification [14]. This section discusses the points raised by this document regarding DTs and their applications.

There are several reasons to use DTs in a design process, some of which are interconnected with the reasons to use HIL simulations as described in Section 2.2. For example, using a DT allows to validate and verify the product in an early stage of the design process. HIL simulation, although being somewhat later in the design process, is such a stage. DTs have been used to validate and verify systems before, but the potential for validation and verification becomes much larger when using a synchronized system, like HIL, along with DTs [15]. Using DTs also makes the predictability of the engineering process better because of a better insight in sensor and actuator data with respect to its real-life counterpart, giving more insight into the final result. DTs are also useful for other applications. For example, informing stakeholders in a project can be made more accessible using a 3D virtual version of the end product or system [16]. DTs can also be used for operator training in an early stage [14], which will be used in this project. By enabling operator training at an early stage it is better ensured that the operation, control, and monitoring match the needs and expectations of the operators. It also incorporates operators in an earlier stage, preventing that operator training becomes an afterthought, due to a hectic schedule [14].

While DTs are useful to validate and verify many requirements in an early stage, they are unable to fully replace the real life plant. Certain dynamic behavior still needs to be tested in real life [14]. This underlines the supporting role a DT can play in a design process. Validation and verification of the system should still be done, even when a DT was used in the design process. Designing a DT can also be very tedious and time-consuming. This is because of the considerations that need to be made while designing a DT.

Putting care into designing a DT is very important for the design process, especially when the DT is used for simulation purposes. In the case of this report, the DT is used in HIL simulations to validate a supervisory controller. This means that the dynamic behavior of the DT needs to closely match the dynamic behavior of the real life plant. The DT is limited in that sense by the software it is made in, and thus the software should be chosen accordingly for the purpose of the DT. This refers to the model quality, which defines how realistically a digital twin represents its physical product [17].

The DT of the Swalmentunnel is developed using Unity [5], which is able to model dynamic behavior [14]. This enables the possibility to validate the supervisory controller on the DT for dynamic behavior.

2.4 HIL preliminaries

With HIL simulation, the control system of a plant or physical part of a system is connected to a simulation of the plant [18]. The control system is able to control the actuators of the simulation model and the simulation model feeds sensor data back to the control system. Figure 2.4 gives a schematic overview of the principle of HIL simulations.

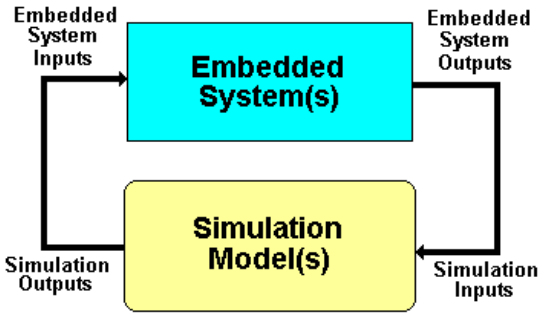

Figure 2.4: HIL simulation principle [18].

Within synthesis-based engineering, HIL is used to validate the realized controller on a simulation of the plant. This report focuses on using DTs within HIL simulations as the simulation of the plant. This allows DTs to be incorporated in the synthesis-based engineering method. This section discusses the type of control system used in the HIL simulation of this report and a short recap is given for the relevancy of HIL simulation to this report.

The control system used for this report is a PLC. A PLC is a highly robust and reliable industrial computer that has many control applications in the industry, such as manufacturing [19]. In the case of this report, the supervisory controller discussed in Section 2.1 runs on the PLC. In previous work, the supervisory controller ran on the same PC that ran the DT using TwinCAT, instead of a PLC. With a HIL setup, the supervisory controller runs on an actual PLC, while the DT and GUI run on separate PCs.

Synthesis-based engineering has been proved promising for providing a supervisory controller for road tunnels [11] [12]. It has the potential to reduce the variability of the time-to-market for road tunnels and improve evolvability [11]. This elucidates the reason why this report explores the possibilities of using HIL simulation for the Swalmen tunnel. In the case of the Swalmen tunnel, the plant is already realized but will be renovated in the upcoming years [6]. The revised controller can be validated on the hybrid plant (see Figure 2.3) using its realized version. This is used to reach one of the research goals of this report. Additionally, HIL simulation can be used for operator training. This allows the operator-trainee to train with the realized controller and its GUI before the tunnel is built or renovated.

2.5 Hardware and Software

This section gives a short introduction to the hardware and software components used for this project.

All three PCs used in this project are identical and are the HP Pavilion 15-CS3846ND. This PC has 8GB of RAM memory and an Intel Core i5-1035G1 CPU which has four cores. Each PC has its own separate task which is discussed in Chapter 3. The PLC is of the ABB brand, type PM866A with firmware 6.x. The Netgear Switch is used to connect the PCs to the PLC. The connection between these parts is done via Ethernet cables. Figure 2.5 shows a picture of one of the PCs and PLCs used for this project.

Figure 2.5: HP Pavilion (left) and ABB PLCs (right).

For this project, a lot of different software is used. A quick overview of the software used and its purpose can be found below. The usage of this software is elaborated on later in this report.

ABB Compact Control Builder
The Compact Control Builder is used to be able to upload the supervisory controller to the PLC hardware. In theory, this only needs to be done once, after which the PLC is active forever as long as it is provided with power. Practice shows that sometimes it helps to do a cold restart and re-upload the supervisory controller when facing connectivity issues.

ABB OPC DA server
The ABB OPC DA server enables a connection from the outside to the variables present on the ABB PLC. This server is already set up correctly and thus is largely left alone for this project. However, this is a vital part of the resulting setup.

Ignition server
The Ignition server enables connectivity between ignition clients and hosts the Ignition OPC UA server.

Ignition designer launcher
The Ignition designer launcher, also referred to as Ignition client, is used to design and run the GUI and the 2D plant. The Ignition designer launcher is able to connect to the Ignition server to share data.

Ignition OPC UA server
The Ignition OPC UA server runs within the Ignition server and allows for data sharing between the Ignition clients and the PLC. OPC, or *'Open Platform Communications'*, is an industry-wide standard that can be used for the secure and reliable exchange of data in the industrial automation space or in other industries [20].

Unity
Unity is well known as a game engine, but can also be used to create and simulate DTs for the industry [21]. Unity has previously been used to develop the DT for the Swalmen tunnel [5], and thus Unity is needed within this project to adjust the DT. Additionally, Unity is used to run the DT.

Prespective
Prespective is a plug-in for Unity that allows the Unity client to connect to an OPC server [22].

Wireshark
Wireshark is a network protocol analyzer, which allows the user to see what is happening on the network [23]. This supports the user in identifying if communication between 2 devices is present.

UaExpert
UaExpert is a separate OPC client, which enables the user to connect to an OPC server [24]. This may help with troubleshooting to check if a connection with an OPC server is possible in the first place.

OPC UA ANSI C Demo Server
The OPC UA ANSI C Demo Server is a simplified third-party OPC UA server that can be used to test whether the Unity client UaExpert is able to connect to a certain OPC UA server on a certain location.

Chapter 3

OPC UA connection for Unity

In this chapter, a theoretical background is given on how a connection is made from a Unity client to the Ignition OPC UA server. Additionaly, the method of how Unity and the PLC exchange data is discussed. This part of the project has been done in collaboration with E. Hoogstrate [25].

First, an overview is given of the HIL setup used for 2D simulations of the plant that was present from the start, after which the challenges that came along with adjusting the setup are discussed. Next, the problems encountered while adjusting the setup and their solutions are discussed. In addition, a method to link the Unity variables to the PLC variables is given, as well as an unsuccessful alternative. In the end, the full method is validated and a short conclusion is given of the whole process, a step-by-step guide is given in Appendix A.

3.1 Initial HIL setup

In this subsection, the initial setup of the HIL is discussed. This is the setup that was present for 2D simulations of the plant at the start of the project. Understanding the initial setup is important to understand the changes needed to be made to the initial setup to allow for a Unity DT. To facilitate this, a network topology as well as the data network of the setup are given.

As mentioned before, a previous setup for HIL simulations is modified to allow for a Unity DT. The initial setup uses the network topology described in Figure 3.1. Network topology is the arrangement of the elements of a communication network.

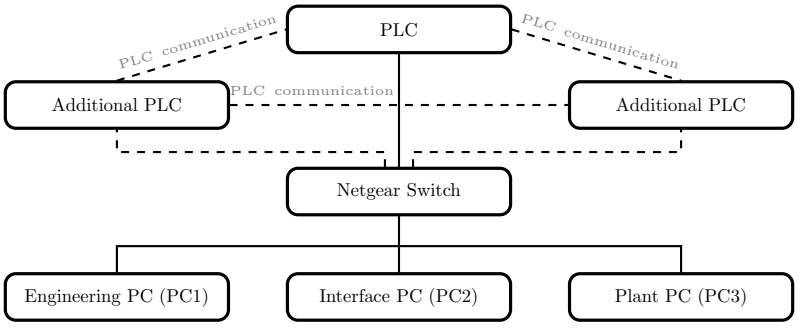

Figure 3.1: Network topology, adapted from [26].

The setup has the ability to host up to three PLCs, but for now, only one has been used for simplicity. Using more than one PLC requires the user to split the supervisory controller over three PLCs, which is not part of this project. Three PCs are present in the setup, the Engineering PC (PC1), Interface PC (PC2), and the Plant PC (PC3). The Engineering PC uploads the supervisory controller to the PLC, the Interface PC

hosts the GUI, and the Plant PC hosts the 2D simulation of the plant. Optionally, only one or two PCs can also be used, since the only requirement for the PLC to be active is to have an active connection with PC1. The GUI could for example also run on PC1, rendering PC2 redundant. The network topology is the same for both the initial setup as the setup adjusted to allow for a Unity DT. PC3 runs the Unity DT for the adjusted HIL setup instead of the 2D simulation of the plant for the initial HIL setup

The initial HIL setup, used for the 2D simulation of the plant, uses the data network in Figure 3.2. A data network is a system designed to transfer data between e.g. servers and clients. PC1 runs the ABB OPC DA server and the Ignition OPC UA server. The ABB OPC DA server interacts with the ABB PLC network as well as with the Ignition OPC UA server. The Ignition OPC UA server in turn communicates with the Ignition clients on PC2 and PC3 [26].

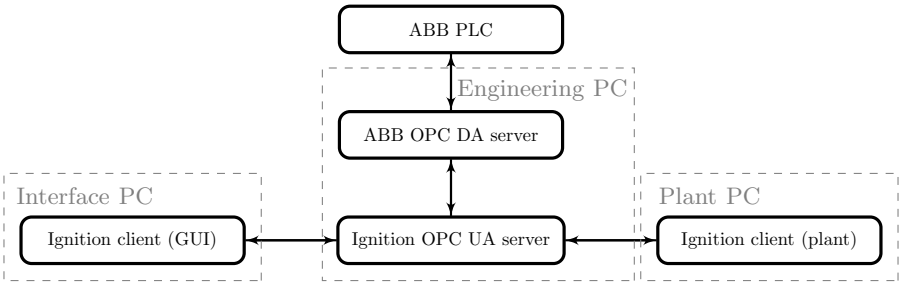

Figure 3.2: Data network, adapted from [26].

Figure 3.2 is vital to ultimately understand what needs to be done to incorporate the DT into the HIL setup. Understanding how the HIL setup works helps with understanding the steps taken to incorporate the Unity DT into the HIL setup, which is discussed in the next section.

3.2 DT HIL adaptations

In this section, the process of connecting a Unity DT to the Ignition OPC UA server is layed out. The order of these steps is not necessarily linear with time, in an attempt to keep the overview clear. It needs to be kept in mind that these steps are based on an already existing HIL setup, and thus some components were already correctly set up. For example, the connection between the Ignition OPC UA server and the PLC was already correctly set up, and thus only a network cable needed to be plugged in to initiate the connection. This section does not cover those details.

In Appendix A, a step-by-step guide can be found to connect a Unity DT to the Ignition OPC UA server. This section gives more elaboration on the challenges faced and the decisions made to get to the final result in Appendix A. First, the main challenge for the connection is discussed after which the initial steps taken are discussed. Since these steps were not sufficient to get to the desired end result, the problems and their solutions are discussed.

3.2.1 DT HIL challenges

The main challenge associated with the DT HIL setup has to do with the data network presented in Figure 3.2. This data network has to be adapted to allow for a Unity DT to connect to the Ignition OPC UA server. The desired data network is shown in Figure 3.3, where the green box indicates the difference to Figure 3.2.

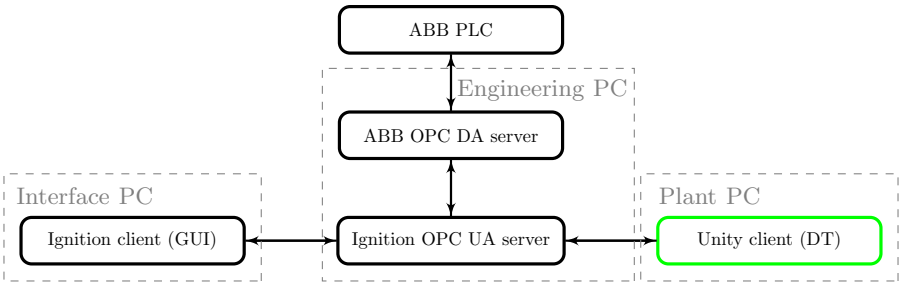

Figure 3.3: Desired data network for the DT HIL setup.

Although the visual representation of the desired data network shown in Figure 3.3 seems straightforward, the Ignition client cannot be simply replaced for the Unity client. First, Unity does not have the properties to connect to any OPC UA server, since it is not an OPC UA client. To solve this problem, it is decided to use the Prespective plug-in for Unity. This plug-in already has been used in the previous project involving the Swalmen tunnel to connect the DT to the supervisory controller running on TwinCAT [5]. Since this plug-in is already properly installed on the DT of the Swalmen tunnel, it is straightforward to also use it for the OPC UA connection. Alternatives to Prespective are available, like the OPCUA4UNITY tool by in2sight [27], but these alternatives are costly and require a new installation process. The Prespective plug-in should allow the Unity DT to connect to the Ignition OPC UA server according to Figure 3.3.

3.2.2 HIL initial steps

To get started with the HIL setup, the hardware components (PLC, PCs, and Netgear switch) need to be properly linked using Ethernet cables. This can be done using the network topology described in Section 3.1. This allows the hardware components to interact with each other and share data. Next, it is important to make sure the PLC hardware component runs the correct supervisory controller. This can be done using the ABB Compact Control Builder. A short description on how to use the ABB Compact Control Builder can be found in the manual of ABB [28]. For this step, the engineering PC needs to be active. When the engineering PC is started, the Ignition Gateway is made active which activates the OPC server. It should now be possible to connect to the OPC server using an OPC client.

It is assumed that Unity and Prespective are installed on one of the PCs. In the case of this project, a *Pre Logic Simulator* was already present within the DT of the Swalmen tunnel. A *Pre Logic Simulator* is a Logic Simulator which enables Prespective to connect to external logic controllers like PLCs [29]. If a *Pre Logic Simulator* is not yet present, the Prespective documentation can be found in [29], or Appendix A can be consulted on how to add a *Pre Logic Simulator* to an Unity project. If the *Pre Logic Simulator* is present in Unity, the OPC UA adapter in Figure 3.4 should be visible.

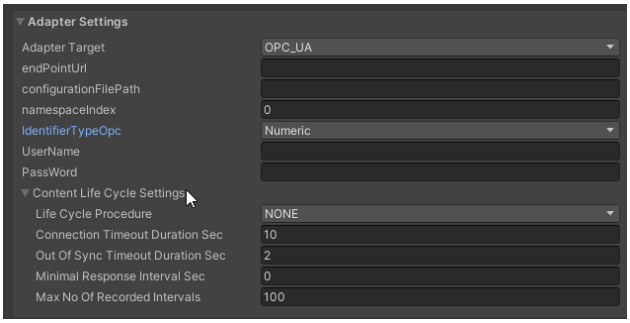

Figure 3.4: OPC UA adapter from Prespective [30].

Documentation on this adapter and its settings can be found in [30]. The important settings to make a connection to an OPC UA server are `EndPointUrl`, `UserName`, `PassWord` and `configurationFilePath`. The

`EndPointUrl` is the location of the OPC UA server, and optionally `UserName` and `PassWord` are the username and password to access the OPC UA server. The `EndPointUrl` can usually be found in the settings of the OPC UA server. In the case of the Ignition OPC UA server, the `EndPointUrl` can be found by typing *https://localhost:8088* in the search bar of a browser on the engineering PC. This gives access to the Ignition gateway running on the engineering PC. The `EndPointUrl` can be found in the Ignition gateway under `Config > opc client > opc connections > more > endpoint`. This can be copied into the `EndPointUrl` box in the OPC UA adapter in Unity. Optionally, the `UserName` and `PassWord` boxes can be filled in if necessary. The `` `configurationFilePath' `` should already be filled in correctly, if this is not the case, the `` `configurationFilePath' `` should be the path to the configuration file that contains the configuration XML named `Opc.Ua.MonoSampleClient.Config.xml` [30]. This configuration file generates the certificates needed for an OPC UA connection. If this is done, the button *'Export policy'* needs to be pressed. This generates the policy file that is used for connecting to the OPC server, and the mapping of variables.

After these steps, a connection to the OPC server should be possible according to the initial plan for the HIL setup. However, there are several potential problems that prevent a connection being established between the Unity client and the Ignition OPC UA server. The next sections discusses some of the potential problems and their solutions, as well as whether it was considered a problem in hindsight.

3.2.3 Checking the *'EndPointUrl'*

A potential problem when an OPC UA client is unable to connect to an OPC UA server is the possibility of the *'EndPointUrl'* being incorrect. This could be due to the uncertainty regarding the port of the Ignition OPC UA server.

Although an *'EndPointUrl'* is provided by the Ignition gateway, another port was tested to validate the *'EndPointUrl'* given by the Ignition gateway. The *'EndPointUrl'* given by the Ignition gateway ends on port *:62541*. It was noticed that the Ignition gateway ran on port *:8088*. The idea is to connect to the Ignition OPC UA server via the Ignition gateway, on which the Ignition OPC UA server runs. Using Wireshark (see Section 2.5), it could be seen that trying to connect to port *:62541* gave no response (see Figure 3.5), while trying to connect to port *:8088* did get a response (see Figure 3.6).

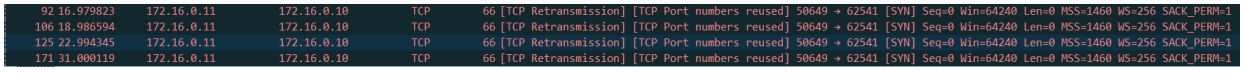

Figure 3.5: Wireshark responses for port *:62541*.

Figure 3.6: Wireshark responses for port *:8088*.

Getting the response from port *:8088* in Figure 3.6 is perceived as undesired, since it ends with a break message. This indicates that while port *:8088* is discoverable, it is not possible to connect to port *:8088*, meaning that port *:8088* is not the correct port to connect to. Port *:62541* did not get a response, which is also not desired since a response is necessary to be able to connect to the port. However, not getting a response indicates that port *:62541* is not discoverable, but does not indicate that the port is incorrect. Most likely, not getting a response has to due with the remote discoverability of the Ignition OPC UA server. This is discussed in the next section.

From this small validation, it can be concluded that port *:62541* is most likely correct since it does not give a break message. The fact that Unity is unable to reach the Ignition OPC UA server has no direct correlation with port *:62541*. Thus, it is assumed that the *'EndPointUrl'* given by the Ignition gateway is correct.

3.2.4 Remote versus local connection

From the previous subsection, it can be concluded that the Unity client was unable to reach the Ignition OPC UA server. This is unexpected since the Ignition OPC UA server is reachable for Ignition clients. This most likely has to do with Ignition clients first connecting to the overarching Ignition server before connecting to the Ignition OPC UA server. The Unity client is unable to reach the overarching Ignition server via the Ignition gateway as discussed in the previous section.

To see whether the Ignition OPC UA server is reachable for a third-party OPC client, UaExpert (see Section 2.5) has been used. UaExpert is a more complete OPC UA client than the Unity client with the Prespective plug-in, which allows for troubleshooting. Additionally, the OPC UA ANSI C Demo Server (see Section 2.5) has been used to determine whether the connectivity problem is on the client side or the server side. A simple connectivity test has been conducted using these software programs, and the test and its results can be seen in Table 3.1. Table 3.1 tests whether UaExpert is able to reach either the OPC UA ANSI C Demo Server or the Ignition OPC UA server. Each row represents a single test, and each cell indicates on which PC the program, displayed in each column, ran. For example, the first row shows that the attempt to connect UaExpert, running on the Plant PC, to the Ignition OPC UA server, running on the Engineering PC, was unsuccessful.

Nr.	UaExpert	OPC UA ANSI C Demo Server	Ignition OPC UA server	Successful connection
1.	Plant PC	-	Engineering PC	✗
2.	Plant PC	Engineer PC	-	☑
3.	Engineering PC	Plant PC	-	☑
4.	Engineering PC	-	Engineering PC	☑

Table 3.1: Set of connection tests done to determine the connectivity of Ignition OPC UA server.

From the results of Table 3.1, it can be concluded that the Ignition OPC UA server is not reachable for any OPC UA client except for when it runs on the Engineering PC. Having the OPC UA server and OPC UA client connect to each other is called a local connection, while connecting them between 2 separate PCs is called a remote connection. In other words, the Ignition OPC UA server did not allow for remote connections. This likely has nothing to do with the Engineering PC itself since the OPC UA ANSI C Demo Server is remotely reachable as test number 2 in Table 3.1 suggests. It is, however, possible to connect to the Ignition OPC UA server with an OPC UA client as test number 4 in Table 3.1 suggests, which confirms that the Ignition OPC UA server allows for third party OPC UA clients. One solution is to move the Unity client to the Engineering PC, and locally connect the client to the server. Figure 3.7 visualizes the data network for a local connection.

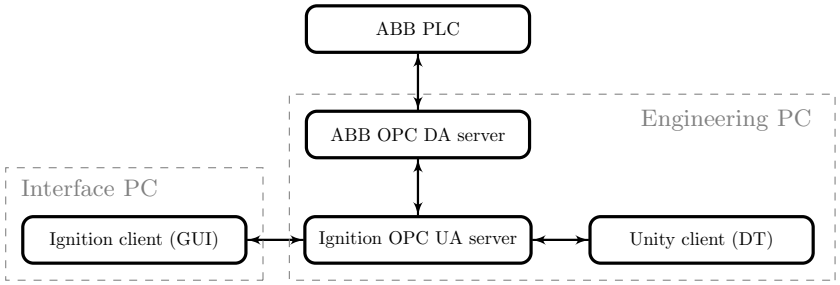

Figure 3.7: Adapted data network for local connection.

Moving the Unity client to the Engineering PC results in Prespective giving the following error:
`ServiceResultException: Error establishing a connection: Error received from remote host: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification`

`path to requested target`. This error demonstrates that the *remote host*, which is the Ignition OPC UA server, is requesting a trusted certificate. This indicates that the Ignition OPC UA server is communicating with the Unity client, and thus moving the Unity client to allow for a local connection is a suitable solution.

A local connection is a suitable solution as discussed before. However, having a remote connection is desirable since it has the potential to let any PC run the DT for HIL simulations. The hardware of the PC running the DT should be improved from the hardware of the current PCs, since the PCs provided by RWS are not well suited to run a DT. This remote setup is conform the data network shown in Figure 3.2. The rest of this section describes how to achieve a remote connection.

As mentioned in [31], *"The default configuration [of the Ignition OPC UA server] allows only clients on the same machine to connect [to the Ignition OPC UA server]"*. This indicates that the Ignition OPC UA server is only set up by default for a local connection. This explains why port *:62541* was not responding as per Figure 3.5, since this port has not been setup to allow for remote connections.

To set up the Ignition OPC UA server for remote connection, a few additional steps need to be taken. The full guide on how to do this can be found in Appendix A. The main steps are to change the *Bind Addresses*, *endPoint URL*, and to allow for remote connections to the Ignition OPC UA server's port in the firewall.

1. The bind addresses settings tell the Ignition OPC UA server which network interfaces to attempt to register on [32]. This is `localhost` by default, which means that only a local client is able to bind to the server. Changing this to the IPv4 Address used by the Ethernet port enables remote clients trying to connect over Ethernet to bind to the Ignition OPC UA server.
2. The *endPoint URL* is used to specify what the Ignition OPC server allows as acceptable addresses [32]. By default this is set to an endpoint only including localhost. This means that even though a remote client is allowed to bind to the server, it is not allowed to connect to the server. This subtle difference in binding and connecting means that both the *Bind Addresses* and *endPoint URL* need to be changed to the IPv4 address of the host PC.
3. The last change is in regards to the firewall of the Engineering PC. When a remote client wants to establish a connection with the Ignition OPC UA server, a initial foreign request signal is sent to the Ignition OPC UA server. Since this request is foreign to the engineer PC, the firewall of the engineer PC blocks the request. To allow the request signal to reach the Ignition OPC UA server, the port of the Ignition OPC UA server should accept all incoming TCP signals, which is the protocol used by OPC UA.

How to perform the changes above can be found in Appendix A. These steps lead to the same error as mentioned before, where the *remote host* requests a trusted certificate.

The changes mentioned above result in a successful remote connection for the PCs provided by RWS. However, the potential of a remote connection lies in the fact that any PC should be able to connect to the Ignition OPC UA server to perform HIL simulations. Unfortunately, the Plant PC cannot be directly switched out for another arbitrary PC. The main issue associated with this is the fact that all PCs in the RWS HIL setup have custom IP addresses over Ethernet. These IP addresses have the format `172.16.0.1x` with subnet mask `255.255.252.0`, where x is unique for each PC. Two devices connecting over Ethernet should have a similar IP address since IP addresses are location based [33]. The IP address on any arbitrary PC is generated based on location, while the PCs in the RWS setup have custom IP addresses. Since these IP addresses do not match, the RWS PCs think that any arbitrary PC is not in the same location and thus do not trust the connection. To solve this, the IP address for an arbitrary PC should be set to the same format `172.16.0.xxx` with subnet mask `255.255.252.0`, where xxx is a combination of numbers not used by either the PLCs or PCs. How to do this can be found in Appendix A. This solution enables UaExpert on any arbitrary PC to connect to the Ignition OPC UA server. However, Unity is not able to connect to the Ignition OPC UA server.

When running Unity, an error with format `ArgumentException: IP Address is invalid | Parameter name: name | Org.BouncyCastle.Asn1.X509.GeneralName..ctor` is given. The full error, available in Appendix A.7, indicates that something goes wrong when creating the certificates for Unity. When taking a look at the `Opc.Ua.MonoSampleClient.Config.xml` file, which is the file in which the Unity certificate is generated, no

discrepancies can be found. The peculiar part is that the exact same file is able to create a Unity certificate on the Plant PC provided by RWS. In addition, UaExpert is able to connect to the Ignition OPC UA server. It is therefore difficult to pinpoint the origin of the problem. Since the connection between Unity and the Ignition OPC UA server is present for the RWS PCs, the decision is made to let this problem be. A remote connection in which any PC can be used is desirable, but should be further investigated.

3.2.5 Certificate Accepting

As mentioned in the previous section, the *remote hosts* request trusted certificates to allow for a connection between the Unity client and the Ignition OPC UA server. A certificate is a file the Ignition OPC UA server and the Unity client use to ensure the legitimacy of the OPC UA connection [34]. The server and the client both have a unique certificate that needs to be trusted by each other to allow for a connection. Accepting certificates is crucial for any OPC UA connection, as the legitimacy of the connection needs to be able to be checked by both the OPC UA server as well as the OPC UA client.

Accepting a client certificate on the Ignition OPC UA server can be done via the Ignition gateway. This can be done trough `Config > opc client > security > server`, after which the client certificate can be trusted on the Ignition OPC UA server. Accepting a server certificate on the Unity client is unfortunately not as simple. Normally, certificate accepting is done in the OPC UA client, which is facilitated by the Prespective plug-in, but no option to accept certificates is present within the Prespective plug-in [30]. The documentation of Prespective suggests to use UaExpert to accept the server certificates for the Unitt client [30]. As discussed in Section 2.5, UaExpert is an OPC client that can be used to check whether a connection with an OPC server is possible. UaExpert also enables the user to trust server certificates on their device. UaExpert thus is able to trust the Ignition OPC UA server certificates for the Unity client. How to do this exactly can be found in Appendix A.

After accepting the certificates for both the Ignition OPC UA server and the Unity client, the error where the *remote host* request for trusted certificates is solved. This underlines the importance of certificate accepting to the process of connecting a Unity DT to the existing HIL setup.

3.2.6 Username and password

After accepting the certificates on both the server and client side, a new error was given by Prespective in the Unity client relating to the username and password of the Ignition OPC UA server. While this error was new, the notion that something was wrong with the username and password was discovered earlier in the process while still trying to connect remotely to the Ignition OPC UA server. This was discovered by taking a look in the only Prespective support file that is not encrypted by Prespective, the policy file. This policy file should contain the settings like `EndPointUrl`, `UserName` and `PassWord` to properly connect to an OPC UA server. However, when examining the policy file, it became clear that the values for `UserName` and `PassWord` were not present in the policy file as can be seen in Figure 3.8.

```
<ControllerSpecification>
  <ActiveController>
    <OPC_UA>
      <Endpoint>opc.tcp://172.16.0.10:62541</Endpoint>
      <NamespaceIndex>0</NamespaceIndex>
      <IdentifierType>i=</IdentifierType>
    </OPC_UA>
```

Figure 3.8: A snippet of the policy code.

Figure 3.8 shows a small piece of the policy code. The lines in between the `<OPC_UA>` and `</OPC_UA>` lines should contain the values for `UserName` and `PassWord`. It is unclear why Prespective does not save these values, or whether these values are encrypted in another Prespective support file. From previous connections with the Ignition OPC UA server using UaExpert, it is known that access to the Ignition OPC UA server requires a username and a password, and what the correct username and password Ignition OPC UA server are. However, entering the correct username and password in the OPC UA adapter in Unity yielded the

previously mentioned error. This, in addition to the observations made earlier in the policy file, indicates that Prespective has trouble with passing the username and password to the Ignition OPC UA server. One solution that was tried was manually adding the password and username in the policy file, like in Figure 3.9. This however obtained no result and was eventually left out of the policy file.

```
<ControllerSpecification>
  <ActiveController>
    <OPC_UA>
      <Endpoint>opc.tcp://localhost:62541/discovery</Endpoint>
      <NamespaceIndex>2</NamespaceIndex>
      <IdentifierType>s=</IdentifierType>
      <UserName>opcuauser</UserName>
      <PassWord>password</PassWord>
    </OPC_UA>
```

Figure 3.9: A snippet of the policy code.

Another solution to the problem was to remove the need for a username and password from the Ignition OPC UA server. To achieve this, the Ignition Gateway needs to be able to allow for anonymous access. In theory, only changing this setting should be sufficient, but it was discovered that this setting alone was not sufficient. Therefore, it was decided to also leave out the standard username and password of the Ignition OPC UA server. This is possible within the Ignition gateway under `Config > opc client > opc connections > Ignition OPC UA Server > edit`. Leave the username and password empty, and save the changes. Next to this, Allowing for anonymous access can be done via `Config > OPC UA > Server Settings > Anonymous Access Allowed` and toggling it to true.

After this, a connection to the Ignition OPC UA server was established from the Unity client. It was therefore useful to remove the need for a username and password from the Ignition OPC UA server. One might argue that this is not desired or sub-optimal since a level of protection is removed from the Ignition OPC UA server. This is certainly true and in a perfect world, the username and password would still be present. However, since trusted certificates are needed to connect to the Ignition OPC UA server, the current level of protection is deemed sufficient for HIL simulations. Additionally, the HIL setup runs on a closed network for which a physical connection is needed.

3.2.7 Contacting Prespective

As can be read in previous sections, some irregularities were found while using the Prespective software. The two main problems encountered were the accepting of certificates in the Unity client and the username and password not being passed to the Ignition OPC UA server. The certificate problem was addressed by Prespective in an update of their documentation [30]. The username and password problem was never addressed. Since these irregularities were found, the decision was made to contact Prespective themselves on several occasions. Unfortunately, Prespective was not able or willing to help solve the problems. This is something that is kept in mind when making a final conclusion on establishing a connection to the Ignition OPC UA server using the Unity client and Prespective. Prespective was chosen because it was already integrated into the DT of the Swalmen tunnel, the CST group already had experience working with the Prespective software, and the CST group already had contacts at Prespective. However, having unsolved problems leads to the question if alternatives are present.

Since many issues were found to be present within the Prespective software, it can be concluded that other software is needed to provide more user-friendly connectivity between the Unity client and the Ignition OPC UA server. One such alternative is the OPCUA4UNITY tool by in2sight [27]. This asset provides a way to connect to the OPC server from the Unity client, much like the Prespective plug-in does. One advantage of the OPCUA4UNITY tool is the fact that it allows for certificate accepting within the Unity client, contrary to Prespective. It might also be possible to work with a username and password using this tool, although this has not been tested. Prespective has been used in previous projects of the CST group and RWS, therefore it might be desirable to work with the same software. Additionally, the OPCUA4UNITY tool costs around 225 euros, whereas Prespective is free to use.

3.3 PLC control

After a connection is present for the Unity client to the Ignition OPC UA server, the next step is to exchange data between the Unity client and the PLC. This is necessary such that the supervisory controller on the PLC can control the DT on the Unity client. The PLC is connected to the ABB OPC server, as can be recalled from Figure 3.3. The ABB OPC server uses the DA (Data Access) protocol, which allows connected servers or devices to access the current value of a variable [35]. The Unity client is only able to communicate via UA (Unified Architecture) protocol, which allows the user to see for example the name of the variable and the timestamp of the last variable change [35]. Additionally, Unity is connected to the Ignition OPC UA server as can be seen in Figure 3.3, instead of the ABB OPC server. Thus, the variables in Unity cannot directly access the variables of the PLC. Therefore, an intermediate step needs to be taken to allow the PLC to control the DT. Fortunately, it is possible to connect an OPC Ignition tag to a PLC variable. These tags are saved in the Ignition tag database which runs on the Ignition OPC UA server. This method had been used previously for the HIL simulations of the 2D simulation of the plant and the GUI. Unity is then able to connect to those Ignition OPC tags via the Ignition OPC UA server. This results in the variable linkage that can be seen in Figure 3.10, where arrows pointing towards the supervisory controller indicate inputs, and arrows pointing away from the supervisory controller indicate outputs. The actuators and sensors in the Unity DT are able to communicate with the OPC tags using logic components, which are discussed in Section 3.3.2. This solution allows the PLC to communicate with the DT using a similar method used previously to set up the 2D simulation of the plant and GUI. This section discusses how to obtain the variable linkage in Figure 3.10.

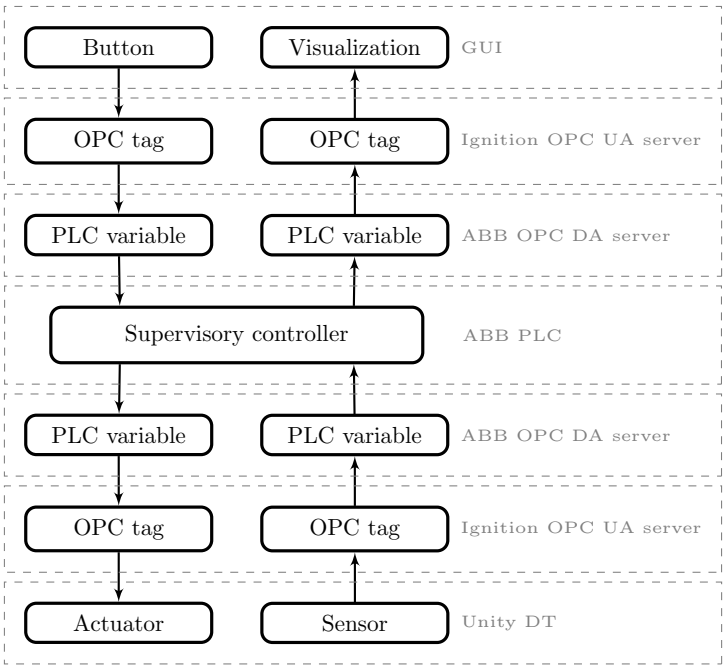

Figure 3.10: Variable linkage.

3.3.1 Ignition OPC tags

Before Unity is able to discover the Ignition OPC tags, a setting has to be changed within the Ignition OPC-UA server. This can be changed in the Ignition Gateway under `config > Security > Show advanced ... properties > Expose Tag Providers`. This is set to false on default, but should be set to true to enable Unity to discover the Ignition OPC tags. When applied and saved, an extra map should be visible in the OPC quick client in the Ignition Gateway called 'OPC tag providers'. If OPC tags are already added they should become visible either as a child or grandchild of this folder.

Adding Ignition OPC tags can be done via the Ignition designer launcher. This is the program that allows the user to edit and run the 2D simulation of the plant and the GUI. To add a tag, the following steps need to be taken.

1. Open an arbitrary project in the Ignition designer launcher.
2. A tag can be added via `Tag Browser > Plus sign > New Standard Tag > OPC Tag`, see Figure 3.11.
3. Set the *Target server* to the ABB OPC server.
4. In the tag path, choose the PLC variable to which the OPC tag should be connected. The data type of the Ignition OPC tag should be set to the same data type of the PLC variable. Choose a suitable name for the tag, as this name is used later on within Unity.
5. Save the tag by clicking *Apply*.

After saving the newly made tag it should pop up in the Tag Browser within the Ignition designer launcher. If done correctly, the tag is also visible in the OPC quick client of the Ignition gateway. Unity can now be adapted to read or write to the Ignition OPC tags.

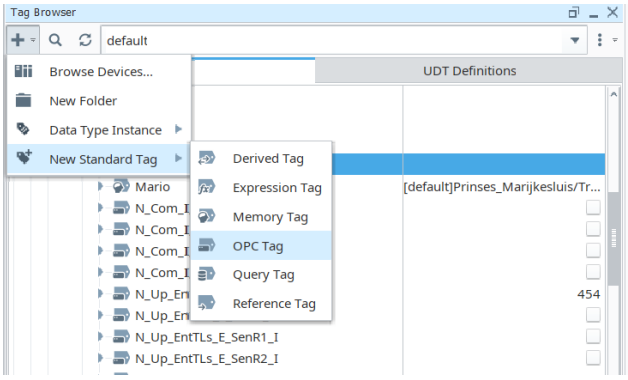

Figure 3.11: Add Ignition OPC tags.

Every added Ignition OPC tag has its own unique node ID. This is needed later for Unity to find the correct Ignition OPC tag to refer to. A node ID contains three components, the NamespaceIndex, the IdentifierType and the Identifier [36]. The NamespaceIndex is the index into a namespace table managed by the OPC-UA server [37]. The IdentifierType indicates the type of the Identifier, this can either be numeric, string, globally unique identifier (GUID) or opaque [36] depending on the OPC-UA server. The Identifier contains the unique path to the Ignition OPC tag and uses the type defined by the Identifer type. An example of a node ID can be *ns=1;s=SomeNodePath*, where *ns=1* is the NamespaceIndex, *s=* is the IdentifierType which is a string in this case, and *SomeNodePath* is the Identifier.

The full node ID of an Ignition OPC tag can be found using UaExpert. When connected to the Ignition OPC-UA server, the *Tag Providers* folder should become visible in the *Address Space* within UaExpert. Opening this folder should show multiple folders of which one is named *default*. This folder or one of its subfolders should contain the Ignition OPC tag. Clicking on this tag should enable the *Attributes* tab, in which the node ID is visible. To ensure that connecting the tags to the logic components in Unity is possible, the NamespaceIndex and IdentifierType settings in the Gateway Settings tab in Unity should be equal to the ones in the node ID. The Identifier is discussed later in the next section.

3.3.2 Logic components

In the DT in Unity, components (e.g. a boom barrier) should be able to react to output signals from, and to send input signals to the PLC. This is done using logic component GameObjects as described in [5], of which an example is shown in Figure 3.12. These GameObjects can be seen as data types that are used for communication with the Ignition OPC tags. These GameObjects can either be inputs or outputs.

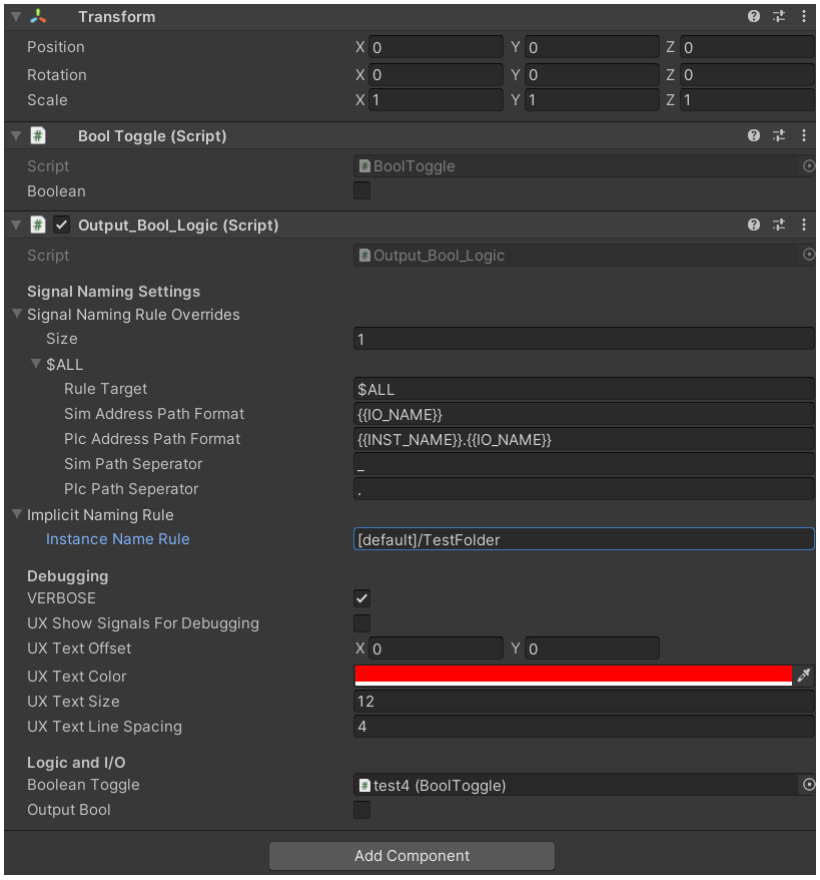

Figure 3.12: Example of an output boolean.

As seen in Figure 3.12, a logic component holds 2 scripts. One logic script and one toggle script. The logic script is used for communication with the Ignition OPC tag, whereas the toggle script is used for communication with Unity components. In the case of the Swalmen tunnel, the logic script can be of 4 types, either input or output and either of type Boolean or Integer. The type of the logic script is indicated in its name. The logic script compares the values of the toggle script and of the Ignition OPC tag. If these values are unequal, the logic script changes the value of the Ignition OPC tag to the value of the toggle script if the logic script is of type input. If the logic script is of type output, the value of the toggle script will be changed to the value of the Ignition OPC tag if the values are unequal. The toggle script can be of 2 types, either Integer or Boolean. It is important to match the right toggle script type to the right logic script type, so Boolean to Boolean and Integer to Integer. The toggle scripts can be referred to by other Unity scripts. These scripts can either change the value of the toggle script in case of an input, or read the value in case of an output. The toggle scripts have been set up such that they can be used as input and output. The fact that a toggle script is paired with either an input or output logic script determines whether the toggle script is an input or output. The old DT already contained logic and toggle scripts of the Boolean type. The reason for this is that the TwinCAT supervisory controller used at the time only contained Booleans [5]. The supervisory controller present on the PLC uses Integers as well as Booleans. Appendix A.6 contains the newly added scripts to allow for Integer values. Along with the previous created logic components prefab for Booleans in [5], and the newly added logic component prefabs for Integers, all relevant Ignition OPC tags can be added to the Unity project.

In order to link the logic component script to Ignition OPC tags, a few key steps need to be taken. It is important to distinguish the *Sim Address* and *Plc Address* in Figure 3.12. Both these settings have an effect on the policy file, of which a small part is available in Listing 3.1. The *Sim Address Path format* in Figure 3.12 determines the name of the output (line 4 in Listing 3.1). This name needs to be unique in order to be

correctly referred to later. Duplicate names for the *Sim Address* between logic components causes Unity to link a single Ignition OPC tag to multiple logic component script, which is undesired. Naming can be done in several ways, the first being naming each variable by hand. This is quite tedious, and thus Prespective has some settings available, called naming rules, to make it more modular. The two most commonly used naming rules are {{*IO_NAME*}} and {{*INST_NAME*}}. {{*IO_NAME*}} is replaced in the xml file with the name of the GameObject to which the logic script is attached, and {{*INST_NAME*}} is replaced with the *Instance Name Rule*, which is highlighted in Figure 3.12. These naming rules can be used both for the *Sim Address Path format* as well as the *Plc Address Path format*. The *Plc Address Path format* is the important setting to correctly link the logic component to an Ignition OPC tag. This is linked to line 5 in Listing 3.1. This *Address* line should contain the Identifier of the Ignition OPC tag to be connected to. The Identifier can be found via the Ignition Designer Launcher, by right clicking the tag and clicking *Copy Path*.

Listing 3.1: Part of XML policy file.

```
1     <Outputs>
2       <!--OUTPUTS 'AfsluitbomenBewegingOp' of type: Output_Bool_Logic-->
3       <Output>
4         <Name>AfsluitbomenBewegingOp_1</Name>
5         <Address>[default]Swalmentunnel_Wodes/Sim_tube_1/AfsluitbomenBewegingOp</Address>
6         <Type>BOOL</Type>
7         <Description>DEF: Value</Description>
8         <Initvalue>False</Initvalue>
9         <Priority>0</Priority>
10        <Fidelity>-1</Fidelity>
11      </Output>
12    </Outputs>
```

For the DT of the Swalmentunnel, the decision was made to set the *Instance Name Rule* equal to the folder that contains the Ignition OPC tag to which the logic component should be connected to. The names of these folders can be found in Appendix B.2. The GameObject to which the Logic script is connected to is then equal to the name of the Ignition OPC tag. The names of these Ignition OPC tags can be found in Appendix B.1. This means that the *Plc Address Path format* could be written as {{*INST_NAME*}}/{{*IO_NAME*}}, which is equal to the identifier of the Ignition OPC tag. The *Sim Address Path format* can be set equal to {{*IO_NAME*}} for all unique Ignition OPC tags. For the Swalmentunnel it was not sufficient to keep the *Sim Address Path format* equal to {{*IO_NAME*}} for tags connected to traffic tube 1 and 2, since similar names were used. This was solved by setting the *Sim Address Path format* equal to {{*IO_NAME*}}*_1* for traffic tube 1 and {{*IO_NAME*}}*_2* for traffic tube 2. This ensured unique names for the *Sim Address*.

Checking whether the logic components are correctly connected to the Ignition OPC tags can be done by clicking play within Unity. If a warning shows up similar to *Warning:* ***Identifier*** *path could not be found*, the *Address* in line 5 of Listing 3.1 is most likely wrong. If this warning does not show up, and no other errors occur, the connection can be tested.

For inputs, a change of the toggle script should result in a change of the PLC variable. This can be tested by opening the PLC variable list, for the ABB PLC this list can be found in the ABB compact control builder. When changing the *Toggle* in the inspector view, a change should be visible in the PLC variable list for the connected variable. For outputs this process is reversed, meaning that a change of the variable in the PLC variable list should result in a change of the Toggle in the inspector view of the connected logic component. This small scale validation was done for every added logic component to ensure that everything was connected properly. This was done before using them for Unity components which is discussed in Chapter 4.

3.3.3 OPC-COM Tunneller module Alternative

The method described in the sections above is a working solution to link PLC variables to logic components. Along with the described method, another method was tried to directly link the logic components to the PLC variables, without having the needs for Ignition OPC tags. This method did not work for the Swalmen tunnel DT, but might be more desirable when the described method is not available due to the absence of Ignition. This section generally discusses what steps were taken and what issues arose working out this method.

Figure 3.13 shows the intended variable linkage using the OPC-COM Tunneller module. The OPC-COM Tunneller module exposes OPC-COM connections to Ignition's OPC-UA server, allowing to easily access them over the network [38]. The ABB OPC-DA server is a OPC-COM connection, and thus can be exposed to the Ignition OPC-UA server. This should allow for direct access to the PLC variables. The OPC-COM Tunneller module can be downloaded from the download page of Inductive Automation [39] and installed in the Ignition gateway under `Config > System > Modules`. The OPC-COM tunneller can then be added as a new device to the Ignition OPC UA server [40]. This should result in an added [OPC-COM Tunneller] directive in the Ignition OPC UA server, which can be checked using the OPC UA Quick client feature in the Ignition gateway.

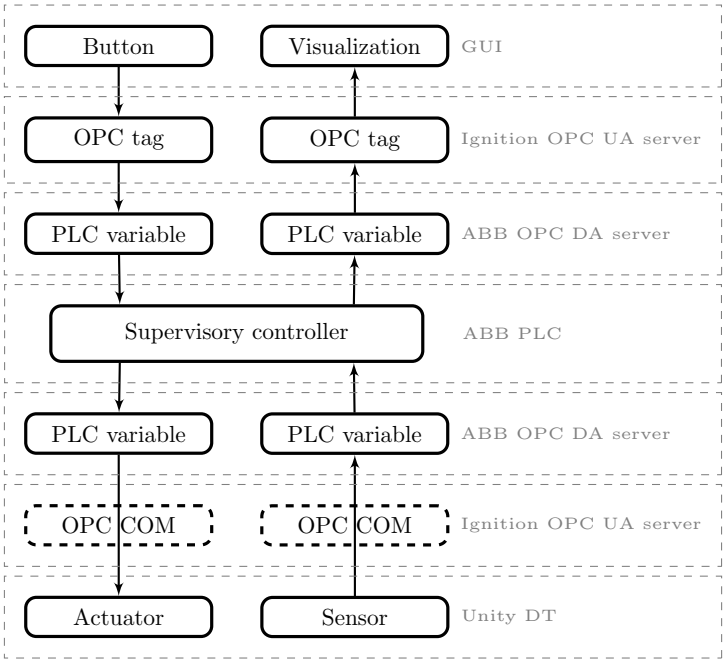

Figure 3.13: Variable linkage for the OPC-COM Tunneller alternative.

As discussed in Section 3.3.1, every tag has its own unique node ID. The identifiers of the node ID for the tags found using the OPC-COM tunneller module are different to the ones found using the method described in the previous sections, of which an overview is present in Appendix B.2. The main difference between the identifiers is that the OPC-COM Tunneller module identifiers use dots as separations instead of forward slashes. This should not be a problem since the supervisory controller for TwinCAT also uses dots as separations for its variable names, so Prespective should be able to handle this.

As mentioned before, the OPC-COM Tunneller module did not lead to a successful variable linkage. In this section, the problem is explained to the best knowledge available. When using UaExpert instead of the Unity client along with the OPC-COM Tunneller module, it is possible to change the PLC variables by changing the variables in UaExpert when connected to the Ignition OPC UA server. However, when swapping out UaExpert for the Unity client in a similar fashion, the Unity client fails to respond. After exactly 10 minutes of leaving Unity open to run, a warning message is given saying the variable cannot be found. This can be

seen in Figure 3.14, where 1 indicates the info message associated with a successful connection with an OPC UA server, and 2 indicates the previously mentioned warning. These messages are given exactly ten minutes apart as can be seen by their timestamps.

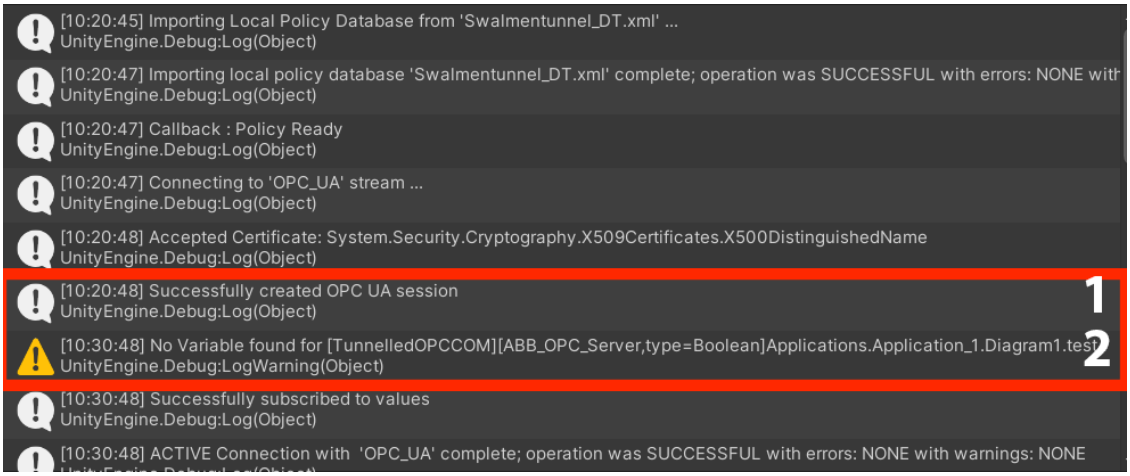

Figure 3.14: Info and warning messages in Unity when using the OPC-COM Tunneller module.

Seeing the messages being thrown exactly might indicate that either Prespective or the Ignition OPC UA server returns a *bad timeout* error. A *bad timeout* error usually means that a timely response was not given between the client and the server. This makes sense, as the Unity client is not able to respond for the full 10 minutes, leading to the variable linkage being broken off. Why the Unity client is unable to respond is unknown, as UaExpert is able to interact with the PLC using this method.

3.4 Conclusion

Using the step-by-step guide in Appendix A, it is possible to establish a connection between a Unity client and the Ignition OPC UA server. Section 3.2 explains the challenges faced while establishing a connection and the decisions made to achieve the end result. The main problems found are the remote accessibility of the OPC UA server and the Prespective software. The solutions found to the problems are feasible and lead to a successful connection being established between a Unity client and the Ignition OPC UA server, allowing for HIL simulations using a DT.

With the method described in Section 3.3 it is possible to link PLC variables to the variables used by the DT as described by Figure 3.10. This allows the DT to connect to the data of the PLC. Ignition OPC tags need to be added if they are not already present, and need to be exposed on the Ignition OPC UA server. Unity logic components can then connect to the Ignition OPC tags via the unique node ID of the Ignition OPC tags. This leads to an interchange of data between the DT and the PLC. Section 3.3.3 describes an alternative to this method, although the alternative is invalid for an Unity DT.

Chapter 4

Controllable components

As described in [5], components are added to the DT that are able to react to changes in readable variables if the component has actuators, and are able to change writable objects if the component has sensors. Such a component could for example be a boom barrier. These components are seen as controllable, since the supervisory controller running on the PLC can control them. These components were added in [5] for the supervisory controller running in TwinCAT. This is unfortunately not suitable for the HIL simulation using the ABB PLC. The logic components for the actuators and sensors are added and connected to the PLC as described in the previous section. This is different from the logic components added using the TwinCAT method used in [5]. Controllable components use logic components within their IO-scripts to model their own behavior. An IO-script handles the logic components for a controllable component [5]. The IO-scripts of the controllable components need to be adjusted so the controllable components can be used in HIL simulations.

This section discusses how the IO-scripts and their supporting scripts are adapted to use the HIL logic components described in the previous section. First the main differences between the TwinCAT logic components and HIL logic components are discussed, after which the changes made are explained. The IO-script for a boom barrier is used as an example.

4.1 Twincat and HIL logic components

As described in the previous section, the toggle scripts are used to communicate with Unity components in the DT. This allows for actuators and sensors in the DT to communicate with the PLC, either via an input or output. This can be done using an IO-script, which is a script that handles the input and output variables for a component such as a boom barrier [5]. In the original DT, these IO-scripts were already present. However, the original IO-scripts could only handle the structure used for TwinCAT logic components. The main issue lies within the *Static_Functions.cs* script, that tries to separate inputs and outputs by the name of the GameObject to which the logic script is connected and find their respective toggle script. In TwinCAT, outputs are denoted in their name by a leading *dvar_* and inputs are denoted by a leading *ivar_*. Since the Ignition OPC tags do not use this naming convention, this script is not usable for HIL logic components. Since the logic scripts distinguish between inputs and outputs, it is unnecessary to also make this difference in the IO-scripts. Additionally, this only worked for toggle scripts of the Boolean data type. It was, therefore, decided that the *Static_Functions.cs* needed to be rewritten in order to remove the need to check the name of a logic component, and to allow for Integer toggle scripts. This is discussed in Section 4.2.2.

Next to the logic components differences, the supervisory controller for the HIL is also different to the one used within TwinCAT. One of the differences is already discussed, being the fact that the current supervisory controller also uses Integers instead of only Booleans. Next to that, the supervisory controller used for HIL simulations uses less variables than the supervisory controller used for TwinCAT. This is partially because of the use of Integers, which for example eliminates the need to add a Boolean for every state of a traffic light. It was also discovered that some variables were redundant. For example, the boom barriers originally contained a Boolean output for *No_choice*, meaning that the supervisory controller does not make a decision

on what the boom barrier should do. This is redundant, since the PLC should always choose between moving up, moving down, or not moving at all. This meant that the IO-scripts needed to be adjusted such that the change in variable usages is correctly implemented. Section 4.3 discusses the changes and decisions made to resolve these problems.

4.2 Static functions

In this section, the adjustments made to the *Static_Functions.cs* are discussed. First, the original script is discussed after which the decisions made to adjust the original scripts are discussed.

4.2.1 Original script

The *Static_Functions.cs* script contains a referable function called *FindBoolToggle*, which searches for the toggle scripts in the logic components. The IO-script calls for this function to access the toggle script in a logic component. The way this was done originally was by entering the name of the logic component in the inspector view of the IO-script. If this name would start with *ivar*, the *Static_Functions.cs* script searches in the *ActuatorLogicComponents* GameObject. This GameObject contains all TwinCAT output logic components. The *Static_Functions.cs* script searches in this GameObject for a logic component that matches the name that was used as an input. If the input string starts with *dvar*, the script searches in the *SensorLogicComponents* GameObject. Figure 4.1 shows the string inputs that are given to the IO-script, and Figure 4.2 showsn of how the *Static_Functions.cs* script searches. Listing 4.1 shows the original *Static_Functions.cs* script.

Listing 4.1: C# code for the *FindBoolToggle* function.

```
1  public static BoolToggle FindBoolToggle(string IOName)
2      {
3          if (IOName.StartsWith("dvar"))
4          {
5              // The variable is an output
6              Transform actuatorParent = GameObject.Find("ActuatorLogicComponents").transform;
7              return actuatorParent.Find(IOName).GetComponent<BoolToggle>();
8          }
9          else if (IOName.StartsWith("ivar"))
10         {
11             // The variable is an input
12                 // The variable is a sensor input
13             Transform sensorParent = GameObject.Find("SensorLogicComponents").transform;
14             return sensorParent.Find(IOName).GetComponent<BoolToggle>();
15         }
16         else
17         {
18             Debug.LogWarning(IOName + " is not an input or output variable");
19             return null;
20         }
21     }
```

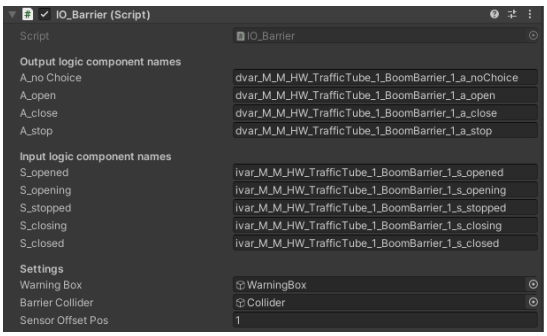

Figure 4.1: Inspector view of *IO_Barrier.cs*.

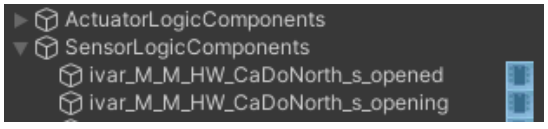

Figure 4.2: Original logic component structure.

As discussed before, this script did not work for the HIL logic components since these do not use the naming convention used by the TwinCAT logic components. The next section discusses the actions taken to make the referring of the HIL logic components possible.

4.2.2 Adjusted script

To make it easier to compare the two scripts, a new script is defined that is similar to the original one, called *Static_Functions_PLC.cs*. To start with, a style decision was made that makes it easier to add logic components to IO-scripts. Instead of using strings as an input for the *FindBoolToggle* script, GameObjects were used. Now, the *Static_Functions_PLC.cs* can directly access this GameObject and search for the toggle script using the code in Listing 4.2. This eliminates the need to precede the variable with either *ivar* or *dvar*, and the need to use the structure used in Figure 4.2. The structure used for the Swalmen DT can be seen in Figure 4.4. Now the GameObject containing the correct toggle script can be added using a drag and drop system visualized in Figure 4.3. A similar function was added called *FindIntToggle* to find the toggle scripts of type Integer. The full code can be found in Appendix A.6.

Listing 4.2: C# code for the *FindIntToggle* function.

```
1   public static BoolToggle FindBoolToggle(GameObject IOName)
2       {
3           if (IOName.GetComponent<BoolToggle>())
4           {
5               // The variable is an output
6               return IOName.GetComponent<BoolToggle>();
7           }
8           if (IOName.GetComponent<IntToggle>())
9           {
10              // The variable is an output
11              Debug.LogWarning(IOName + " is an Integer, rather than a Boolean. Try ...
                    FindIntToggle instead.");
12              return null;
13          }
14          else
15          {
16              Debug.LogWarning(IOName + " is not an input or output variable");
17              return null;
18          }
19      }
```

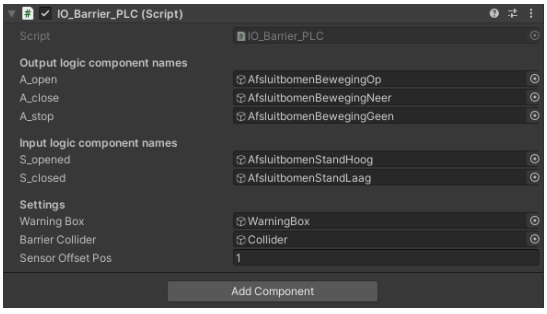

Figure 4.3: Inspector view of *IO_Barrier_PLC.cs*.

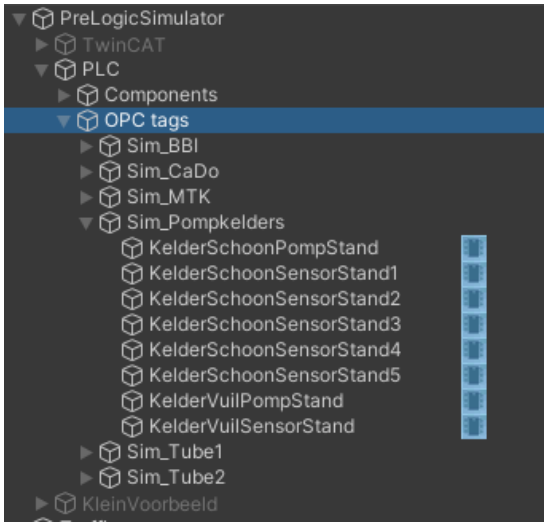

Figure 4.4: Logic component structure used for the Swalmentunnel DT.

These adjustments make it possible to add any logic component regardless of their naming convention, and to search for toggles of the type Boolean and Integer. The next section discusses how this was used inside an IO-script.

4.3 IO-scripts

As discussed previously, IO-scripts are scripts that can handle the input and output variables for a component such as a boom barrier [5], and thus model the behavior of a component. This effectively means that the IO-script makes the boom barrier move or not move on the PLC's requests. Since the new supervisory controller uses fewer variables, the IO-script for each component needs to be checked and adjusted where needed. This section discusses some of the changes made using the boom barrier IO-script as an example. These steps cover all changes made to the IO-scripts, since they are similar in structure.

This section discusses the *IO_Barrier_PLC.cs* script step by step. All relevant changes made to the original script are discussed. The full script for the boom barrier can be found in Appendix A.6.3. Additionally, all components that are adjusted and their input and output variables can be found in Appendix B.1

Listing 4.3: Input and output variables

```
1     [Header("Output logic component names")]
2          [SerializeField, Tooltip("Open the barrier")]
3          private GameObject a_open;
4
5      [Header("Input logic component names")]
6          [SerializeField, Tooltip("Opening the barrier")]
7          private GameObject s_opening;
8          [SerializeField, Tooltip("Closing the barrier")]
9          private GameObject s_closing;
10         [SerializeField, Tooltip("no movement")]
11         private GameObject s_stop;
12         [SerializeField, Tooltip("The barrier is fully opened")]
13         private GameObject s_opened;
14         [SerializeField, Tooltip("The barrier is fully closed")]
15         private GameObject s_closed;
```

Listing 4.3 displays how the input and output variables are added. This can be done by dragging and dropping a GameObject containing the logic component to the Inspector view of the *IO_Barrier_PLC.cs* script like in Figure 4.3. These GameObjects are referred to later as *a_open*, *s_opening*, *s_closing*, *s_stop*, *s_opened* and *s_closed*. Where *a_* indicates an output and *s_* indicates an input. The main differences with the original IO-script are the number of input and outputs of the supervisory controller, and the usage of GameObjects instead of strings.

Listing 4.4: Dictionary

```
1      [HideInInspector]
2      public Dictionary<string, BoolToggle> IO_dict_Bool = new Dictionary<string, BoolToggle>();
3      public Dictionary<string, IntToggle> IO_dict_Int = new Dictionary<string, IntToggle>();
```

Listing 4.4 shows the usage of two dictionaries called *IO_dict_Bool* and *IO_dict_Int*. A dictionary is a Unity type that uses a key to indicate a value. In this case a string is used to indicate a BoolToggle for *IO_dict_Bool* and an Integer for *IO_dict_Int*. The original IO-script for the boom barrier also uses a dictionary, but the original TwinCAT supervisory controller did not include Integers. Unfortunately, Unity dictionaries are unable to hold 2 different types of values. This is the reason that two dictionaries are used. It is possible to make IO-scripts without dictionaries, but since dictionaries were used in every IO-script made in [5], it was decided to keep them for a modular change to the existing IO-scripts. When only Booleans or Integers are present as inputs and outputs, it is possible to use a single dictionary.

Listing 4.5: Adding dictionary entries

```
1  private void Start()
2      {
3          // Filling the I/O dictionary
4
5              // Actuators
6              IO_dict_Int.Add("a_open", Static_Functions_PLC.FindIntToggle(a_open));
7              // Sensors
8              IO_dict_Bool.Add("s_opening", Static_Functions_PLC.FindBoolToggle(s_opening));
9              IO_dict_Bool.Add("s_closing", Static_Functions_PLC.FindBoolToggle(s_closing));
10             IO_dict_Bool.Add("s_stop", Static_Functions_PLC.FindBoolToggle(s_stop));
11             IO_dict_Bool.Add("s_opened", Static_Functions_PLC.FindBoolToggle(s_opened));
12             IO_dict_Bool.Add("s_closed", Static_Functions_PLC.FindBoolToggle(s_closed));
13     }
```

Listing 4.5 indicates how dictionary entries are added to the dictionaries. The key to the dictionaries is a string, which in this case can either be *a_open*, *s_opening*, *s_closing*, *s_stop*, *s_opened* or *s_closed*. These strings are linked to the BoolToggle or IntToggle scripts of the logic components added in Listing 4.3. This is done using the new *FindBoolToggle* and *FindIntToggle* functions using the new *Static_Functions_PLC.cs* script which is based on the old *Static_Functions.cs* script, as can be read in Section 4.2. When calling the dictionary key, the BoolToggle or IntToggle of the logic component is returned, depending on the data type of the logic component. The new IO-script thus uses the new *Static_Functions_PLC.cs* script to find the BoolToggles and the single IntToggle, instead of the old *Static_Functions.cs* script.

Listing 4.6: behavior modelling

```
1  private void Update()
2      {
3         if (IO_dict_Int["a_open"].Integer % 10 == 0)
4          {
5              motor.RotationDirection = 1;
6          }
7          else if (IO_dict_Int["a_open"].Integer % 10 == 1)
8          {
9              motor.RotationDirection = -1;
10         }
11         else if (IO_dict_Int["a_open"].Integer % 10 == 2)
12         {
13             motor.RotationDirection = 0;
14         }
15         else if (IO_dict_Int["a_open"].Integer % 10 == 3)
16         {
17             motor.RotationDirection = -1;
18         }
19         else
20         {
21             motor.RotationDirection = 0;
22         }
23
24         // --- Set the sensor values based on states of the components ---
25         IO_dict_Bool["s_opened"].Boolean =
26             Mathf.Abs(motor.OpenRotation - motor.currentRotation) < SensorOffsetPos;
27         IO_dict_Bool["s_closed"].Boolean =
28             Mathf.Abs(motor.ClosedRotation - motor.currentRotation) < SensorOffsetPos;
29
30         IO_dict_Bool["s_opening"].Boolean =
31             motor.RotationDirection == 1 && !IO_dict_Bool["s_opened"].Boolean;
32         IO_dict_Bool["s_closing"].Boolean =
33             motor.RotationDirection == -1 && !IO_dict_Bool["s_closed"].Boolean;
34         IO_dict_Bool["s_stop"].Boolean =
35             motor.RotationDirection == 0;
36
37         // Move the collider out of the way when the barrier is opened
38         if (IO_dict_Bool["s_opened"].Boolean)
```

```
39          {
40              BarrierCollider.transform.position = Vector3.Lerp(transform.position, ...
                    colliderStartPosition + Vector3.up * 20, Time.ΔTime * 100);
41          }
42          else
43          {
44              BarrierCollider.transform.position = Vector3.Lerp(transform.position, ...
                    colliderStartPosition, Time.ΔTime * 100);
45          }
46      }
```

The IO-script is now able to call the correct BoolToggles and IntToggle of the logic components. Listing 4.6 shows how these values are used to model the boom barrier behavior. First, in lines 3 to 23, the output behavior is modeled. For example, when the PLC wants to open the boom barrier, it sets the value of the IntToggle of *a_open* to an integer ending with 0, like 100. The if-statement in line 3 evaluates to true in that case, since the IO-script uses the modulus function on the Integer *a_open*. This is indicated within the IO-script with n % d, where n is the numerator and d the denominator. It divides the given numerator by the denominator to find a result. In simpler words, it produces a remainder of the Integer division. If 10 is used as the denominator, the modulus function returns 0 for the input 100 % 10, since 100 is divisible by 10, evaluating the if-statement in line 3 to true. Line 5 is then called, rotating the boom barrier upwards. Lines 19-22 are added for the case that the output value does not comply with any of the statements in lines 3,7,11 or 15. The supervisory controller should never allow this to happen, so lines 19-22 can be seen as redundant. Lines 25-35 set the Booltoggles for the input values to true if the conditions behind the dictionary entries are met. Lines 38-45 are used to move the collider of the boom barrier, which is used to stop the traffic. The main difference with the original IO-script lies within lines 3-35, since the correct action needs to happen using fewer output and input variables.

4.4 Validation

The adaptation of the IO-scripts is done for most of the IO-scripts from the original DT [5]. Appendix B.1 contains the input and output variables used for each IO-script added. The only IO-scripts missing are the one for broadcast messages, since this function of the DT first needs to be properly added, and the IO-script for the CCTV system, since it was not present in the original DT. This section discusses how this method of adjusting IO-scripts is validated and what the results are of the validation tests.

In order to validate the correct linkage between the PLC variables and the actions within the DT, short validation tests have been written that can be seen in Appendix B.3. Not all components of DT were correctly added at this point, and thus only the added components are tested. From this test, it can be concluded that all IO-scripts are properly adjusted. It can thus be concluded that the method described in this chapter is valid to model the behavior of controllable components. To validate the controller, it is important that all components are correctly added. Optionally, it is possible to adjust the IO-script such that a warning message is given when undesired behavior is detected in a component.

4.5 Conclusion

With the methods described in this chapter it is possible to either adjust or make new IO-scripts to model the behavior of controllable components. In this project, the IO-scripts only needed to be adjusted. The variables can be added via a drag and drop system, to easily visualize which variable has been added. The usage of a dictionary has been adopted from [5], but the toggle scripts are now found using an adjusted support script. Examples are provided on how to deal with Integers and Booleans.

The method described has been validated only for the correct linkage between PLC variables and actions within the DT. For future work, it is important to also validate the supervisory controller present on the PLC. To do this, it has to be known in which state each variable should be in a specific situation. The validation tests described in Appendix B.3 are not comprehensive enough for this, but can be used as a stepping stone for controller validation. The IO-scripts can be adjusted to check whether the incoming output variable corresponds with the given input variable.

Chapter 5

DT adaptations for operator training

As mentioned in Chapter 1, one of the goals of this report is to improve the Swalmen tunnel DT for operator training. This chapter discusses the improvements made to the Swalmen tunnel DT to allow for operator training. There are two types of improvements made to the Swalmen tunnel DT, the first one being performance-based improvements. These improvements are implemented to increase the frame rate, measured in frames per second (FPS), which is a direct measurement of the performance of a DT. If the frame rate is high, the DT runs more smoothly than when the frame rate is low. A DT that runs more smoothly is desired in order to improve operator training, since this enhances the level of engagement. Section 5.1 and Section 5.2 discuss improvements made in respect to the performance of the Swalmen tunnel DT.

In addition to improved performance, optimizations to the Swalmen tunnel DT need to be made to enable the Swalmen tunnel DT for operator training. These improvements are made to either enhance the engagement of operator training or to enable basic functions needed for operator training. Some of the performance optimizations coincide with the optimizations for operator training, such as Section 5.1. Section 5.1 describes both a basic function as engagement enhancer for operator training. Having the CCTV of the DT as similar as possible to the CCTV of the real life tunnel is important to successfully train an operator and engages the operator in the training due to the realism. Section 5.3 describes a basic function for operator training, as both the operator trainer (the person conducting the training) and the operator trainee (the person undergoing the training) need separate monitors to perform the operator training. Section 5.4 describes an improvement meant to enhance the level of engagement of the operator training, as having dynamic traffic is not necessary for operator training. This chapter discusses the optimizations made in more detail, and gives a short conclusion on the improvements made at the end.

Unity provides a tool to check the performance of the CPU on the host PC with Unity Profiler [41]. This tool allows the user to get inside where the loss in frame rate is coming from. This tool has been used throughout this project to identify potential losses of frame rate in the DT. If needed, more documentation on the profiler tool can be found in [41].

5.1 CCTV performance

In [5], it was already concluded that the CCTV cameras negatively impact the CPU performance of the host PC. In [5], a solution was proposed that would allow the environment to be rendered only once per instance. This should reduce the number of parts that needed to be rendered and thus increase performance. This base solution has some downsides, like decreasing the CCTV frame rate when more cameras are added, and still decreasing the overall frame rate of the DT. This section improves this solution by proposing an adaptation that can be implemented alongside with the original solution.

For operator training, it is important to have a multiple CCTV camera feed. This allows for better operator training since it emulates real life better. In order to do so, the CCTV cameras have to be batched in batches of multiple cameras, depending on the requirements set for operator training. This can be done within Unity by placing multiple cameras under the same *'GameObject'*. Listing 5.1 shows the C# code that is used for

camera switching. In here, it can be seen that cameras are batched under *'CCTV_set1'* and *'CCTV_set2'*. Important to note is that in order for this to work, cameras from different batches should render to the same texture. Listing 5.1 ensures that only 1 camera batch is active at the time, and thus only 1 texture is rendered at a given moment to the monitor in the DT. This code also allows displaying text in the DT to indicate the current active camera batch. This only works for 2 batches, but it can be expanded to include multiple batches. This solution allows for a better frame rate when used in combination with the solution of [5], both for the CCTV feed and for the DT itself.

Listing 5.1: C# code that switches between 2 sets of CCTV feeds.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class CCTVSwitching : MonoBehaviour
{
    public GameObject CCTV_set1;
    public GameObject CCTV_set2;
    public Text txt;
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.RightArrow))
        {
            CCTV_set1.SetActive(true);
            CCTV_set2.SetActive(false);
            txt.text = "CCTV_set1";
        }
        if (IInput.GetKeyDown(KeyCode.LeftArrow))
        {
            CCTV_set1.SetActive(false);
            CCTV_set2.SetActive(true);
            txt.text = "CCTV_set2";
        }
    }
}
```

For the DT of the Swalmen tunnel, it was chosen to use batches of 2. This emulates the CCTV feed the operator gets from the real tunnel, where each CCTV feed is from a single traffic tube. The switching of the batches is regulated by the PLC code, which provides an integer to the DT corresponding to a certain batch. The linking of the CCTV system to the PLC has not been done due to time constraints, and the uncertainty of which PLC variables are used for the switching of the CCTV feed.

There is the possibility to change the resolution of the render textures. Currently, the resolution is 1280x720 (720p) HD, but it can be up or down scaled fairly easily within the render texture under the size tab, see Figure 5.1. Since the DT of the Swalmen tunnel only uses 2 camera sets, the resolution was found not to have a big impact on performance. However, this might be different in the case of more cameras.

Figure 5.1: A snippet of *'Camera1'* render texture.

5.2 Frustum culling and occlusion culling

Figure 5.2 shows the difference in frame rate between two locations. The first location, *'Overzicht noord'* or *'Northern overview'* in English, is a top-down view of the northern entrance and exit of the tunnel (see Figure 5.3). The second location *'In-uitgang noord'* or *'Northern Entrance and Exit* in English, is an

isometric view of the northern entrance and exit of the tunnel (see Figure 5.4). The white line in Figure 5.2 shows the moment in time of switching between these two locations.

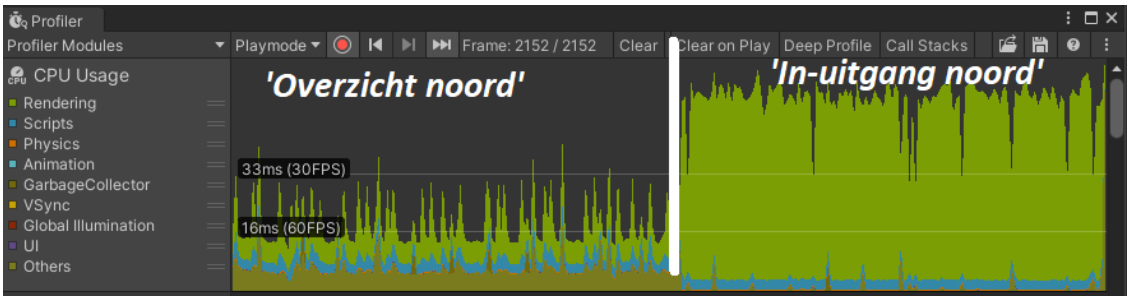

Figure 5.2: Frame rate in different locations.

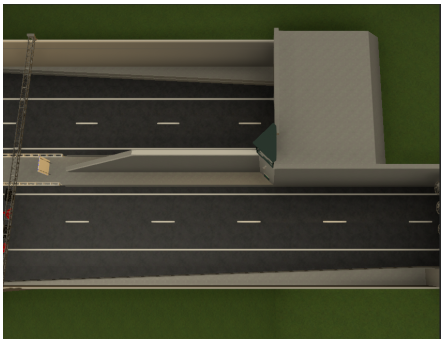

Figure 5.3: *'Overzicht noord'* location.

Figure 5.4: *'In-uitgang noord'* location.

From Figure 5.2, it becomes apparent that there is a loss in frame rate for the *'In-uitgang noord'* location. The Unity profiler module indicates that this is due to an increase in objects needed to be rendered. This is most likely due to the fact that Unity uses frustum culling by default. A frustum in this case refers to the frame of the camera. Frustum culling ensures that objects outside of the frustum are not displayed [42]. This greatly reduces the number of textures that need to be rendered. This means that objects that are obstructed by other objects but are within the frustum are displayed. When looking at Figures 5.3 and 5.4, it can be seen that for Figure 5.3 there are fewer objects present in the frustum, since the camera points downwards. This, therefore, leads to fewer textures that need to be rendered, and a higher frame rate. Figure 5.2 demonstrates this by giving an indication of the frame rate on the left hand side of the graph. The lower peaks for *'Overzicht noord'* indicate a higher frame rate, while the higher peaks for *'In-uigang noord'* indicate a lower frame rate.

Occlusion culling is similar to frustum culling, but now, objects that are obscured by other objects do not render, even if they are present in the frustum. This could increase the performance of the DT, however, in this case, it does not increase the performance since the tunnel is modeled as one big entity for the most part. When enabling occlusion culling, the frame rate stayed roughly the same, so for this project, another solution had to be found. If the tunnel was modeled as smaller modular sections instead of 1 big entity, occlusion culling could become more effective.

The proposed solution uses the knowledge of how frustum culling and occlusion culling work to increase the performance of the DT. In order to minimize the objects rendered to improve the single CCTV performance of the DT, a camera angle must be chosen such that the least amount of objects are rendered within the frustum. When implementing this change, the relevance of the camera angle to the location should be kept in mind, so the relevant information for that location is still displayed. Figure 5.5 still represents the same relevant information about the location *In-uitgang noord* as Figure 5.4, but needs less rendering of objects,

thus leading to a better frame rate. Figure 5.6 shows a comparison in frame rate of this part of the solution for the adjusted camera angle.

Figure 5.5: New *In-uitgang noord* location.

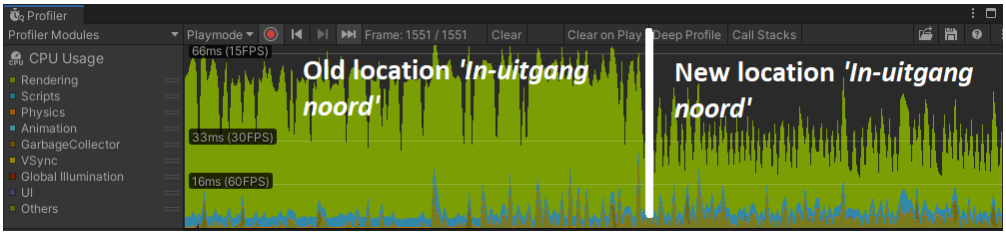

Figure 5.6: Frame rate difference between the old and adjusted camera angle for *'In-uitgang Noord.*

5.3 Multi-windowed display

In order to improve the DT for the purpose operator training, it is needed to run the DT in a two-windowed display. On the first main window, the DT would work as described in the previous project [5]. On the secondary window, only the CCTV room would be visible and no interaction would be possible with the DT via this window. This should allow the operator trainee to get the relevant information from the DT from the CCTV cameras, while the operator trainer could interact with the DT to start events. Unfortunately, Unity has not yet developed its support for multi-windowed display well [43], and thus the end result is sub-optimal. In the current case, displayed in Figure 5.7, the operator trainee uses the second display in full screen to see the CCTV camera feed, as is desired. The second display is the monitor to the right in Figure 5.7 Unfortunately, the operator trainer has a windowed screen without a toolbar, that does not fill the whole screen. This can be seen on the left display in Figure 5.7. Fortunately, the taskbar is still visible, so the DT can easily be closed. The operator trainer is, most likely, not bothered by this, so this solution is usable.

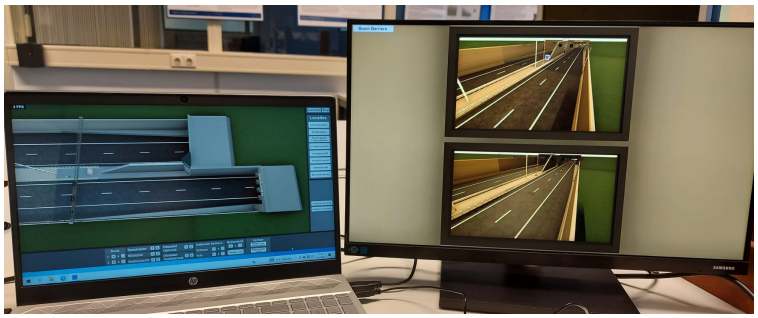

Figure 5.7: Dual monitor setup

In order to create 2 displays, it is important to have 2 cameras dedicated to their own displays. In this case, the *'FreeCam'* [5] was used to render to display 1, while a second camera called *'OperatorView'* was added in front of the CCTV screen. This camera has its target display set to display 2, this setting can easily be changed in the camera properties window. Unity does not make the second display active by default, therefore it is needed to add a small script to the *'FreeCam'* camera.

Listing 5.2: C# code that activates the second display

```
1  using UnityEngine;
2
3  public class ActivateAllDisplays : MonoBehaviour
4  {
5      void Awake()
6      {
7          if (Display.displays.Length > 1)
8          {
9              Screen.fullScreen = false;
10             Display.displays[1].Activate();
11         }
12     }
13 }
```

Listing 5.2 shows the code for activating the second display. The code activates the second display if a second display is detected. This only runs once on startup.

5.4 Dynamic traffic

In the original DT of the Swalmen tunnel, vehicles are spawned at semi-regular intervals and move in a straight line until they despawn [5]. To add a level of realism, dynamic traffic can be added to simulate real-life traffic. The Unity asset store provides some options for adding dynamic traffic, like *Simple Traffic System* [44]. A positive point of this asset is that it is very well worked out, and adding a dynamic traffic system can be done in a matter of minutes. The downside to this asset is that it is unclear from the description if this asset is suitable to use for a highway setting. This asset also includes more functions than desired, which is not needed for this project. Therefore it was attempted to include dynamic traffic with handwritten scripts. The assumption is that adding dynamic traffic is possible but decreases the performance of the DT. It is, therefore, needed that a balance is struck between realism and performance. The next subsections describes the process of attempting to add dynamic traffic.

The goal is to achieve a dynamic traffic level where vehicles are able to switch between two lanes or adjust speed based on their surroundings. This is all done in a highway-like setting, thus speeds are high but all vehicles move in the same direction. It is important to make sure cars do not slow down too much to create a traffic jam or do slow down too little and create an accident. Creating an accident in the tunnel is a different objective for operator training which is not covered in this project.

5.4.1 Raycasting

In order for a vehicle to perceive its surroundings, the method of raycasting is used. Raycasting is a technique available in Unity where the object shoots out a *'ray'* from a certain point for a certain distance and notes when an object hits the *'ray'*. Figure 5.8 gives a visual representation of the raycasts used. The white lines in front, to the left, and to the right of the vehicle are the *'rays'*.

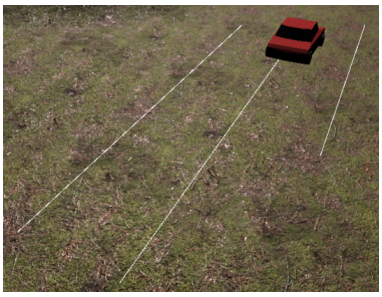

Figure 5.8: Visual representation of the raycasts.

This configuration was chosen so the vehicle always knows if there is a vehicle blocking one of its paths. For example, if a vehicle is in front of the vehicle the raycast in front of the vehicle detects this.

Listing 5.3: C# code for raycasts

```
1  private void FixedUpdate()
2  {
3      // Determine Raycasts
4      Rayorigin = transform.position + offset;
5      RaycastHit hit1, hit2, hit3;
6      Ray Forward = new Ray(Rayorigin, transform.TransformDirection(Vector3.forward));
7      // Draw Raycast (for debugging)
8      if (raycastDebug)
9      {
10         Debug.DrawRay(Rayorigin, transform.TransformDirection(Vector3.forward) * Raydistance);
11     }
12     // Detect hit on raycasts
13     if (Physics.Raycast(Forward, out hit1, Raydistance))
14     {
15         if (hit1.collider.CompareTag(VehicleTag))
16         {
17             ObstacleForward = true;
18         }
19     }
20     else
21     {
22         ObstacleForward = false;
23     }
24 }
```

The code in Listing 5.3 shows the code used to determine if an obstacle is present. First, in line 4 the origin of the raycast is determined. This is done using a certain offset that is unique for every vehicle prefab. The vehicle prefabs are the same as the ones used [5]. In lines 5 and 6 the raycast is made using an origin and a direction, and the information of the raycast is stored in a *'RaycastHit'* variable. Lines 8-11 only work when debugging is enabled, line 10 draws the raycast in the scene view. Lines 12-13 condense the information of the raycast into a simple Boolean. First, it is checked whether the raycast has detected a hit. Next, the object the raycast has hit is checked for its tag. If the tag is that of another vehicle, the Boolean *'ObstacleForward'* is set to true. If none of this is the case, the Boolean is set to false. Similar lines of code are used for the raycasts to the left and to the right of the vehicle, storing their information in single Booleans as well called *'ObstacleRight'* and *'ObstacleLeft'* respectively. This information can later be used to code the decision-making behavior of the vehicle. Lines 48-110 in Listing C.3 in Appendix C shows the total code for this.

5.4.2 Destination

In order to move the vehicle, a waypoint-like system is used. In this system, a destination can be set for the vehicle, and the vehicle keeps moving until that destination has been reached.

Listing 5.4: C# code for moving the vehicle

```
1  private void FixedUpdate()
2  {
3      // Keep going forward
4      if (reachedDestination)
5      {
6          Destination(0);
7      }
8
9      // If the destination is not reached, move towards the destination until you're in a ...
           certain range
10     if (transform.position != destination)
11     {
12         Vector3 destinationDirection = destination - transform.position;
13         destinationDirection.y = 0;
14
15         float destinationDistance = destinationDirection.magnitude;
16
17         if (destinationDistance ≥ stopDistance)
18         {
19             reachedDestination = false;
20             Quaternion targetRotation = Quaternion.LookRotation(destinationDirection);
21             transform.rotation = Quaternion.RotateTowards(transform.rotation, ...
                   targetRotation, rotationSpeed * Time.ΔTime);
22             transform.Translate(Vector3.forward * movementSpeed * Time.ΔTime);
23         }
24         else
25         {
26             reachedDestination = true;
27         }
28     }
29 }
30 void Destination(int dest)
31 {
32     destination = transform.position + new Vector3(dest, 0, 20);
33 }
```

Listing 5.4 shows the C# code for moving the vehicle. Lines 4-7 and 30-33 set the destination to 20 world units [45] forwards. World units is a unit of length specifically used by Unity [45]. Lines 10-28 make sure that the vehicle rotates and translates with a certain speed towards the set destination. Using this base code in addition to the raycasts enables the vehicle to make lane switching decisions and move towards the decision made. Lines 193-212 and 237-248 in Listing C.3 in Appendix C show this within the total code.

5.4.3 Lane switching and speed decisions

The vehicles in the tunnel need to individually and autonomously decide whether they need to switch between lanes, slow down, speed up or stop driving entirely. To achieve this, a set of decisions is proposed that can be translated into C# code to let the vehicles make these decisions themselves. This chapter discusses these sets of decisions.

To simplify the problem, the fact that the Swalmen tunnel only has 2 traffic lanes per traffic tube has been taken into account. The decisions the vehicle has to make can be split into decisions made driving on the left lane, and decisions made driving on the right lane. Lines 114-152 in Listing C.3 in Appendix C show the coding of the considerations made for lane switching while driving on the right lane, and lines 155-191 show the considerations made for lane switching while driving on the left lane. This section discusses the considerations made in the right lane as well as in the left lane without discussing the code in Appendix C.

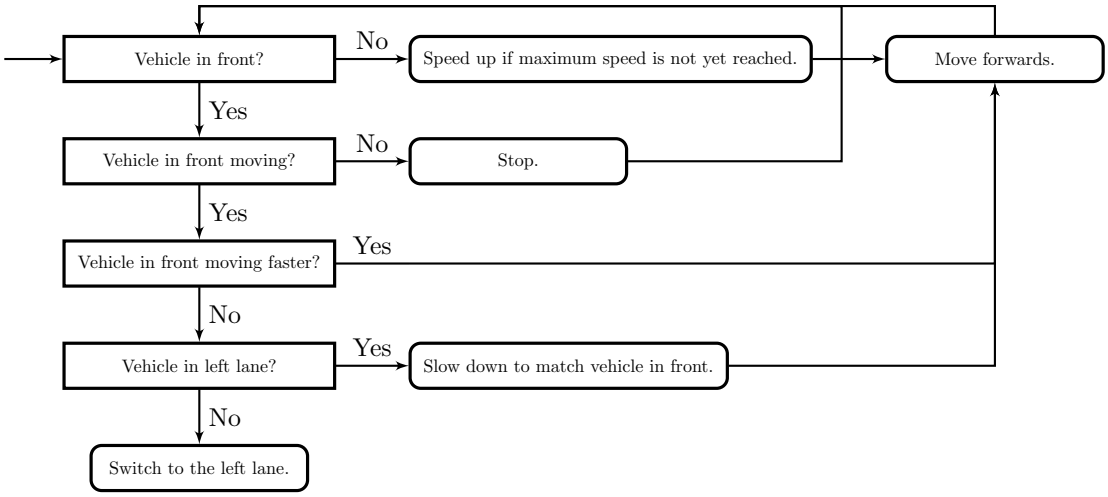

Figure 5.9: Decision table of the right lane.

Figure 5.9 shows the decision table for the vehicle while it is driving in the right lane. This roughly shows the decisions the code makes before undertaking action. First, the code checks whether there is a vehicle in front. If this is not the case, the vehicle can speed up until its maximum speed and keep on moving forward in the right lane, and the process repeats. If there is a vehicle in front, the code moves on to the next decision. Next, the code checks if the vehicle in front is moving. It can do this by reading out the current speed of the vehicle in front. If the vehicle in front is not moving, the vehicle stops immediately, and the process repeats. The next consideration questions whether the vehicle in front is moving faster than the vehicle it self. If this is the case, chances are that the vehicle in front just switched back from the left lane, so no action is needed and the vehicle can keep moving forwards. If this is not the case, the vehicle checks whether there is another vehicle blocking the left lane. If this is the case, the vehicle slows down to match the speed of the vehicle in front and keeps moving forwards. If this is not the case, the vehicle can switch to the left lane.

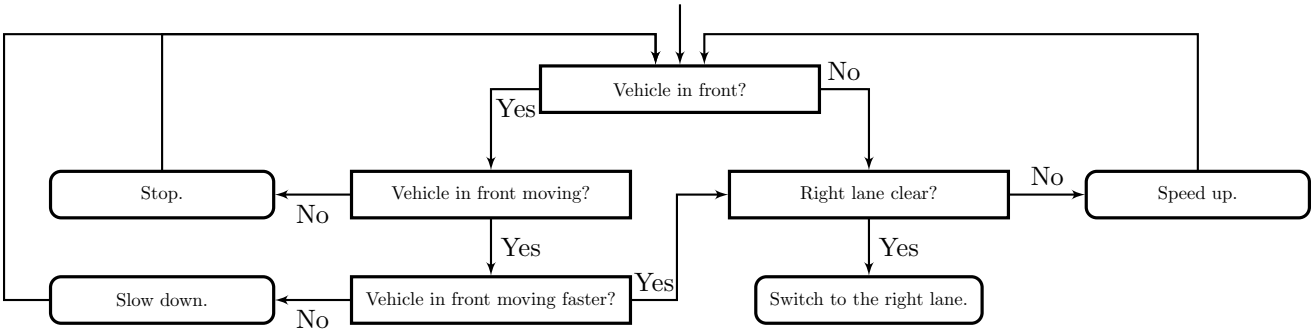

Figure 5.10: Decision table of the left lane.

The decision table in Figure 5.10 starts with a rather peculiar consideration. Instead of checking the right lane first, the left lane is scanned to check for vehicles in front. This is done to ensure that vehicles with the highest maximum speed keep using the left lane, instead of constantly switching lanes and risking overtakes on the right hand side. If another vehicle is in front of the vehicle, it is checked whether this vehicle is moving. If this is not the case, the vehicle stops and the process repeats. If this is the case, it is checked whether the vehicle in front is moving faster than the vehicle itself. If this is not the case, the speed is adjusted to the vehicle in front and the process repeats. If this is the case, the right lane can be checked to see whether it is clear. If the right lane is clear the vehicle can switch back to the right lane, if not the vehicle can speed up to the maximum speed and repeat the process.

These processes can occur each frame per vehicle, which means quite some processing power is needed to handle this. However, when running this dynamic traffic the frame rate did not rapidly decrease. The decisions explained above are what the code should do for each vehicle, but unfortunately some discrepancies are still present. The biggest one has to do with long trucks arbitrarily stopping when a vehicle switches back to the right lane right before the truck. This most likely is a bug within the code which has not been solved yet, but should be relatively easy to fix if given the time. All in all more time is needed to integrate the code for the vehicle behavior within the Swalmen tunnel DT to fix the current problem.

5.4.4 Spawner and destroyer

In order to gain a continuous flow of cars flowing into the tunnel, spawners and destroyers need to be used. In [5], a base spawner and destroyer had already been developed. The spawner has been modified to fit the purpose of this project, while the destroyer has been kept the same. The spawner can now be set to either the right or left lane via a Boolean, passing the information on to the spawned vehicles so that they know what lane they are on. It is also possible to easily change the range of speed and acceleration of the spawned vehicles in the spawner itself. This allows the user to make minor changes easily. Figure 5.11 shows the variables that can be changed within the spawner. The code for the spawner can be found in Appendix C.

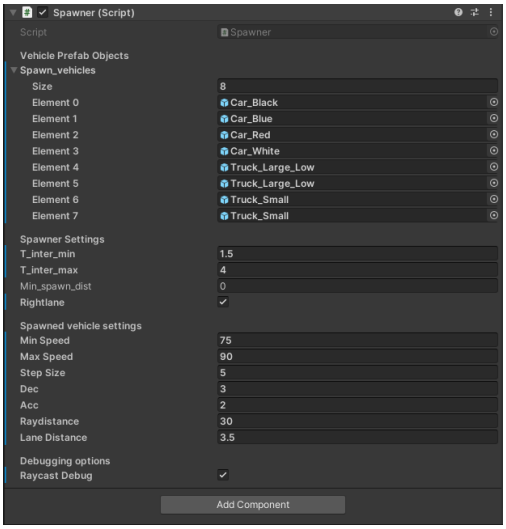

Figure 5.11: Variables that can be changed within the spawner.

5.5 Conclusion

This chapter discusses the adaptations made to the Swalmen tunnel DT to allow for operator training. Some of these changes have been made to improve the frame rate of the Swalmen tunnel DT, while others have been made to improve the level of engagement of the Swalmen tunnel DT for operator training. All the changes discussed in this chapter reach one of these goals. All of the changes made have been fully implemented in the Swalmen tunnel DT, except for the dynamic traffic. Due to time constraints, the final implementation has not been fully done.

In general, this part of the project has not been given enough time to be fully worked out. This means that the Swalmen tunnel DT is not ready for operator training. It is important to identify the most important missing components of the Swalmen tunnel DT before the Swalmen tunnel DT can be used for operator training.

Chapter 6

Conclusion and recommendations

In this section, a conclusion is drawn from the process of adding a DT to a HIL setup, and from the improvements made for the Swalmen tunnel DT for operator training. This is done by discussing the research goals stated in Section 1.1 and drawing conclusions from the findings of previous chapters. Afterwards, the decisions made in this project are evaluated and recommendations are given for future work.

6.1 Conclusion

The first research objective is in respect to adding a DT to the HIL setup.

1. Describe how to adjust the existing HIL setup to allow for HIL simulations using a DT.

Chapter 3 discuss how to link a DT to a HIL setup and how to exchange data between the DT and the supervisory controller running on the PLC. Important steps to setup the Ignition OPC UA server are to allow for remote connection and to remove the need for a username and password. Data exchange between the PLC and the DT can be achieved using Ignition OPC tags. Logic components can be added within the DT to communicate with these OPC tags. In addition, Chapter 4 describes how to adjust the controllable components within the Swalmen tunnel DT in order to perform HIL simulations. The main changes made in respect to the original Swalmen tunnel DT is the adaptations made to the *FindBoolToggle* to allow for HIL logic components, and the addition of Integers as usable data type. Moreover, a step-by-step guide is available in Appendix A that describes these steps. It is possible to perform HIL simulations using the Swalmen tunnel DT with the provided PCs from RWS, but it is not possible to perform HIL simulations using any arbitrary PC. As of yet, the error in Appendix A.7 is the only limitation for any arbitrary PC to connect to the HIL setup for a DT.

The second research objective is in respect to performing HIL simulations.

2. Perform HIL simulations with the DT of the Swalmen tunnel.

The HIL simulations with the DT of the Swalmen tunnel have been successfully performed. Appendix B.3 provides a validation test plan performed on the adjusted DT and its results. These results validate the step-by-step guide given in Appendix A and support the conclusion of the first research objective. The validation test plan in Appendix B.3 is not a validation test plan for the supervisory controller of the Swalmen tunnel. Performing HIL simulations using a DT can be done by following the steps in Chapter 3 and Chapter 4, or the step-by-step guide in Appendix A.

The third research objective is in respect to the continued development of the Swalmen tunnel DT.

3. Adjust the Swalmen tunnel DT such that it is suitable for operator training, and describe the methods used to adjust the Swalmen tunnel DT.

Adaptations have been made to the original Swalmen tunnel DT that are needed for operator training as described in Chapter 5. Two types of adaptations have been made. the first type of adaptation has been made to improve the performance of the Swalmen tunnel DT, measured in frames per second. The second type of adaptation has been made to enhance the engagement of operator training. The adaptations made have been in respect to the CCTV system, the usage of two displays for operator training and traffic flow. The first steps have been taken towards operator training using the Swalmen tunnel DT, however, the DT is not ready for operator training, since more changes are needed and the components described in Chapter 5 need to be correctly added to the Swalmen tunnel DT used for HIL simulations. This research objective has thus not been fully met. The main reason why this research objective has not been fully met is due to the allocation of time towards adjusting the DT for operator training during this project.

6.2 Recommendations

There are a few recommendations that can be made based on the findings of this report. The first has to do with the use of the Prespective software. As discussed in Chapter 3, many issues arose using the Prespective software. It could be that newer versions of Prespective solve these issues. An alternative to Prespective, like OPCUA4UNITY, can be tried along with the step-by-step guide in Appendix A to circumvent these issues.

As discussed in Chapter 3, HIL simulations can only be performed using the PCs provided by RWS. The PCs of RWS, however, are not suitable to run DTs because of their low performance. These PCs are not recommended to use for operator training. For future work, the current error displayed in Appendix A.7 can be solved and the possibility to connect any arbitrary PC to the existing HIL setup can be worked out. Alternatively, a new HIL setup can be made that uses a Plant PC with better hardware specifications. Before doing so, it should be investigated what the required hardware specifications are.

As discussed in Chapter 4, not all components of the Swalmen tunnel have been added to the DT for HIL simulations. The CCTV camera system and the broadcasting system both need to be added to the DT for HIL simulations using the methods described in Chapter 4. For the CCTV camera system, some base code is already provided in Chapter 5. The broadcasting system however needs to be made from scratch. The emergency exit broadcast has been implemented, and can be used as a template. Live broadcasting, which is a part of the broadcasting system, needs to be researched for its viability, since it is unclear whether it is possible to play a live message within an Unity DT.

To successfully adapt the DT for operator training, it is needed to determine what components need to be either added or adjusted. This has not been done within this project. A component that needs to be added is causing an accident. This is one example of an event which an operator should be able to train with. More of these training components need to be thought out and implemented. The dynamic traffic mentioned in Chapter 5 needs to be fully debugged and implemented in the DT before it can be used, after which causing an accident can be added. Alternatively, the *Simple Traffic System* can be used to create dynamic traffic. Before doing so, it should be investigated whether this Unity asset can be used in a highway-like setting, and if it can be adjusted to add the event of causing an accident.

Lastly, it is important to merge the Swalmen tunnel DT used for HIL simulations with the Swalmen tunnel DT used for operator training. As of now, these are 2 separate Unity projects. It is recommended to implement the changes made for operator training into the Swalmen tunnel DT used for HIL simulations. This is because the DT used for HIL simulations has more newly added components and adjusted old components. All the changes made for operator training can be added to the DT for HIL simulations in a modular fashion.

Bibliography

[1] M. A. Goorden, L. Moormann, F. F. H. Reijnen, J. J. Verbakel, D. A. van Beek, A. T. Hofkamp, J. M. van de Mortel-Fronczak, M. A. Reniers, W. J. Fokkink, J. E. Rooda, and L. F. P. Etman, "The road ahead for supervisor synthesis," in *Dependable Software Engineering. Theories, Tools, and Applications*, J. Pang and L. Zhang, Eds. Cham: Springer International Publishing, 2020, pp. 1–16.

[2] T. Wescott, *Applied Control Theory for Embedded Systems*. Elsevier, Newnes, 2006.

[3] J. Trauer, S. Schweigert-Recksiek, C. Engel, K. Spreitzer, and M. Zimmermann, "What is a digital twin? - Definitions and Insights from an Industrial Case Study in Technical Product Development," *Proceedings of the Design Society: DESIGN Conference*, vol. 1, no. May, pp. 757–766, 2020.

[4] D. E. Jones, C. Snider, L. Kent, and B. Hicks, "Early stage digital twins for early stage engineering design," *Proceedings of the International Conference on Engineering Design, ICED*, vol. 2019-August, no. AUGUST, pp. 2557–2566, 2019.

[5] J. van Hegelsom, J. M. van de Mortel-Fronczak, L. Moormann, D. A. van Beek, and J. E. Rooda, "Development of a 3D Digital Twin of the Swalmen Tunnel in the Rijkswaterstaat Project," *CoRR*, vol. abs/2107.12108, 2021. [Online]. Available: https://arxiv.org/abs/2107.12108

[6] C. O. Bouwen, "Tunneloverzicht," Jun 2021. [Online]. Available: https://www.cob.nl/tunneloverzicht/

[7] Ministerie van Infrastructuur en Waterstaat, Rijkswaterstaat, "Tunnels," Dec 2021. [Online]. Available: https://www.rijkswaterstaat.nl/wegen/wegbeheer/tunnels

[8] J. Schokking, "Supervisory control model for the Swalmentunnel," Jan 2021, internship Report; Eindhoven University of Technology.

[9] Inductive Automation, "Industrial Automation Software Solutions by Inductive Automation," 2022. [Online]. Available: https://inductiveautomation.com/

[10] R. R. H. Schiffelers, R. J. M. Theunissen, D. A. V. Beek, and J. E. Rooda, "Model-Based Engineering of Supervisory Controllers using CIF," *Communications*, vol. 21, no. January, 2009.

[11] L. Moormann, P. Maessen, M.A. Goorden, J.M. van de Mortel-Fronczak, and J.E. Rooda, "Design of a Tunnel Supervisory Controller using Synthesis-Based Engineering," in *ITA-AITES World Tunnel Congress, WTC2020 and 46th General Assembly, Proceedings*. ITA Library, Sep. 2020, pp. 573–578, iTA-AITES World Tunnel Congress (WTC 2020).

[12] L. Moormann, J. van Hegelsom, P. Maessen, W.J. Fokkink, J.M. van de Mortel-Fronczak, and J.E. Rooda, "Digital twins for the validation of road tunnel controllers," in *ITA-AITES World Tunnel Congress, WTC2022 and 47th General Assembly, Proceedings*. ITA Library, Sep. 2022, iTA-AITES World Tunnel Congress (WTC 2022).

[13] E. Negri, L. Fumagalli, and M. Macchi, "A Review of the Roles of Digital Twin in CPS-based Production Systems," *Procedia Manufacturing*, vol. 11, pp. 939–948, 2017, 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy.

[14] COB, "Groeiboek - Digitaal aantonen - COB." [Online]. Available: https://www.cob.nl/wat-doet-het-cob/groeiboek/digitaal-aantonen/

[15] A. Löcklin, M. Müller, T. Jung, N. Jazdi, D. White, and M. Weyrich, "Digital Twin for Verification and Validation of Industrial Automation Systems - A Survey," *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, vol. 2020-September, pp. 851–858, sep 2020.

[16] V. Stojanovic, M. Trapp, R. Richter, B. Hagedorn, and J. Döllner, "Towards the Generation of Digital Twins for Facility Management Based on 3D Point Clouds," pp. 270–279, 2018.

[17] F. Leitenberger, T. Gwosch, and S. Matthiesen, "Architecture of the digital twin in product validation for the application in virtual-physical testing to investigate system reliability," *Proceedings of the 32nd Symposium Design for X, DFX 2021*, pp. 1–9, 2021.

[18] M. Short and M. Pont, "Assessment of high-integrity embedded automotive control systems using hardware in the loop simulation," *Journal of Systems and Software*, vol. 81, pp. 1163–1183, 07 2008.

[19] M. J. Walker, "The programmable logic controller : Its prehistory, emergence and application," Sep 2012. [Online]. Available: https://doi.org/10.21954/ou.ro.0000d59f

[20] OPC foundation, "What is OPC? - OPC Foundation," 2022. [Online]. Available: https://opcfoundation.org/about/what-is-opc/

[21] Unity, "Unity homepage," 2022. [Online]. Available: https://unity.com/

[22] Prespective, "Prespective homepage," 2022. [Online]. Available: https://prespective-software.com

[23] Wireshark, "Wireshark · Go Deep." 2022. [Online]. Available: https://www.wireshark.org/

[24] Unified-Automation, "UaExpert "UA Reference Client" - Unified Automation," 2022. [Online]. Available: https://www.unified-automation.com/products/development-tools/uaexpert.html

[25] E. Hoogstrate, "HIL simulation with waterway locks digital twins," *BSc Thesis, Eindhoven University of Technology*, 2022.

[26] AT-Automation, "Documentatie testopstelling," Feb 2021, classified document used in the Rijkswaterstaat project.

[27] In2sight, "OPCUA4Unity — Utilities Tools — Unity Asset Store," 2022. [Online]. Available: https://assetstore.unity.com/packages/tools/utilities/opcua4unity-143980

[28] ABB, "Compact Control Builder AC 800M - Getting Started," 2016. [Online]. Available: https://library.e.abb.com/public/8353699805e94e68b0466316736bd522/3BSE041584-600_B_en_Compact_Control_Builder_AC_800M_6.0_Getting_Started.pdf

[29] Prespective, "2022.1.1148.0 Logic Simulator," 2022. [Online]. Available: https://unit040.atlassian.net/wiki/spaces/PUD/pages/2200699042/2022.1.1148.0+Logic+Simulator

[30] ——, "2022.1.1148.0 OPC UA," 2022. [Online]. Available: https://unit040.atlassian.net/wiki/spaces/PUD/pages/2200732373/2022.1.1148.0+OPC+UA

[31] Inductive Automation, "Connecting to Ignition's OPC UA server - Ignition - Inductive Automation Forum," 2019. [Online]. Available: https://forum.inductiveautomation.com/t/connecting-to-ignitions-opc-ua-server/26137

[32] ——, "Endpoint configuration in Ignition 8 - Ignition - Inductive Automation Forum," 2019. [Online]. Available: https://forum.inductiveautomation.com/t/endpoint-configuration-in-ignition-8/28628

[33] Norton, "What does an IP address tell you and how it can put you at risk," 2021. [Online]. Available: https://us.norton.com/internetsecurity-privacy-what-does-an-ip-address-tell-you.html

[34] Sectigo, "What Is SSL Server Certificate and How Does It Protect Websites?" 2022. [Online]. Available: https://sectigostore.com/page/what-is-ssl-server-certificate/

[35] The Automization, "OPC-UA vs DA - The Automization," 2019. [Online]. Available: https://theautomization.com/opc-ua-vs-da/#What_are_OPC_UA_and_OPC_DA

[36] OPC Labs, "OPC UA Node IDs." [Online]. Available: http://opclabs.doc-that.com/files/onlinedocs/QuickOpc/Latest/User%27sGuideandReference-QuickOPC/OPCUANodeIDs.html

[37] Unified-Automation, "OPC UA Namespaces." [Online]. Available: https://documentation.unified-automation.com/uasdkcpp/1.7.4/html/L2UaAdrSpaceConceptNamespaces.html

[38] Inductive Automation, "OPC COM Tunneller Module - Ignition User Manual 7.9 - Ignition Documentation." [Online]. Available: https://docs.inductiveautomation.com/display/DOC79/OPC+COM+Tunneller+Module

[39] ——, "Current Ignition Release — Inductive Automation." [Online]. Available: https://inductiveautomation.com/downloads/ignition/8.1.17

[40] D. Fogle, "A Primer to the OPC-COM Tunneler Module – Inductive Automation Help Center," 2022. [Online]. Available: https://support.inductiveautomation.com/hc/en-us/articles/360047615731-A-Primer-to-the-OPC-COM-Tunneler-Module

[41] Unity, "Profiler overview," 2022. [Online]. Available: https://docs.unity3d.com/Manual/Profiler.html

[42] C. Cookie, "Maximizing your unity game's performance," Apr 2017. [Online]. Available: https://cgcookie.com/articles/maximizing-your-unity-games-performance

[43] K. Jones, "Multi-Display Windowed Mode - Unity Forum," 2016. [Online]. Available: https://forum.unity.com/threads/multi-display-windowed-mode.429757/

[44] TurnTheGameOn, "Simple Traffic System — AI — Unity Asset Store," Apr 2021. [Online]. Available: https://assetstore.unity.com/packages/tools/ai/simple-traffic-system-159402

[45] Unity, "Preparing Assets for Unity," 2019. [Online]. Available: https://docs.unity3d.com/2019.3/Documentation/Manual/BestPracticeMakingBelievableVisuals1.html

Appendix A

Step-by-step HIL plan

This appendix contains the step-by-step guide that explains how to obtain a connection between the Unity client and the Ignition OPC UA server, as well as an explanation on how to connect the PLC variables to the Unity variables used in the DT. This appendix has been made in collaboration with E. Hoogstrate [25]. In addition, this appendix contains the logic components scripts needed to exchange data with the PLC. Appendix A.7 contains the error encountered when trying to connect an arbitrary PC which is not provided by RWS to the Ignition OPC UA server.

A.1 Set up of software and hardware

This section explains what software and hardware are needed before starting the steps.

A.1.1 Unity and Prespective

It is assumed that Unity has been downloaded and installed. The versions of Unity can be found on the website of Unity[1]. Unity version 2019.4.19f1 was used as proof of concept, but any stable newer version should also work. In addition to Unity, the Prespective asset needs to be added and licensed. The Prespective asset can be downloaded and installed from the Unity asset store[2], and the license can be obtained via the Prespective website[3].

A.1.2 Ignition and PLC

It is assumed that the PLC used has PLC code active and is connected to an OPC DA server, which in turn is able to communicate with the Ignition OPC UA server. One way to validate whether this has been done correctly is to see whether an Ignition Designer Launcher can access the variables of the OPC DA server by adding an Ignition OPC tag described in Appendix A.4. Alternatively, the connections can be checked within the Ignition gateway under `Config > OPC Client > OPC Quck Client`. This should show a similar structure as in Figure A.1, where the OPC DA server is indicated with *ABB OPC Server*.

[1] https://unity3d.com/get-unity/download/archive
[2] https://assetstore.unity.com/packages/tools/utilities/prespective-digital-twin-software-161664#description
[3] https://prespective-software.com/prespective-shop/

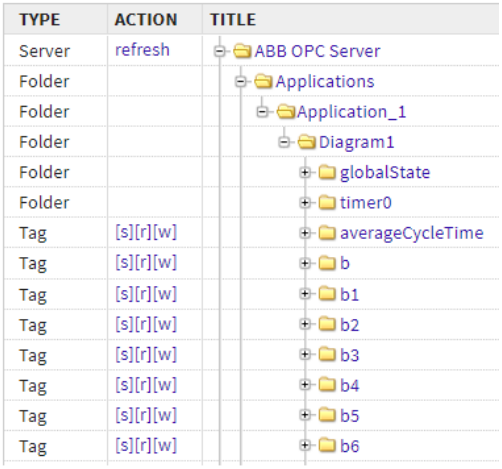

Figure A.1: OPC Quick Client initial view

A.1.3 UaExpert

UaExpert is a program that acts as an OPC UA client. UaExpert is mainly used to accept certificates and to troubleshoot the connection. UaExpert can be downloaded from the download page of Unified Automation[4]. Documentation on how to use UaExpert can be found on the Unified Automation website[5]. The relevant functions of UaExpert are discussed later on.

A.1.4 Ignition Designer Launcher

The Ignition Design Launcher is part of Ignition. This program is used to create OPC tags which are linked to PLC variables. These tags are used for the connection between Unity and the PLC.

A.1.5 ABB Compact Control Builder

ABB Compact Control Builder is a piece of software which is used to upload the PLC code on the PLC. This program is used to test the connection between the PLC and Unity once a connection is set up and the variables of the PLC and Unity are linked.

A.2 Configuring the Ignition OPC UA server

This section gives a step-by-step guide on how to configure the Ignition OPC UA server to allow for local and remote connection for a Unity digital twin. Step 1 is necessary for both local and remote connection, while steps 2-5 are only necessary for remote connection.

1. It is important to remove the need for a username and password for the Ignition OPC UA Server, and to allow for anonymous access. Removing the username and password can be done in the Ignition Gateway under `Config > opc client > opc connections > Ignition OPC UA Server > edit`. Leave the username and password empty, and save the changes. Allowing for anonymous access can be done via `Config > OPC UA > Server Settings > Anonymous Access Allowed` and toggling it to true.

The following steps configure the Ignition OPC UA server for remote connection.

2. Go to `Config > OPC UA > Server Settings`. Change the *Bind Addresses* to the IPv4-adres of the PC which hosts the Ignition OPC UA server, e.g. `172.16.0.10`. Change the *Endpoint Addresses* to: "`<IPv4-adres>, <localhost>`", where `<IPv4-adres>` is the same as the *Bind Addresses*.

3. The endPoint URL of the Ignition OPC UA server needs to be updated. Go to `Config > opc client > opc connections > more > endpoint`. Change the endPoint URL from `opc.tcp://localhost:62541/discovery` to `opc.tcp://<IPv4-adres>:62541/discovery`, where

[4]https://www.unified-automation.com/downloads/opc-ua-clients.html
[5]https://documentation.unified-automation.com/uaexpert/1.4.2/html/first_steps.html

`<IPv4-adres>` is the same IPv4-adres used in step 2. Continue clicking next to get to the overview page where the settings are saved.

4. The firewall of the PC which hosts the Ignition OPC UA server interferes with the remote connection. This problem is solved by going to the firewall of the PC which hosts the Ignition OPC UA server, by typing "Firewall" in the search box of the Taskbar. Next go to `Advanced settings > Inbound Rules`. Here, click `New Rule > Port > TCP > Specific local ports > 62541`. Select *Allow the connection* and apply to all networks.

5. The Ignition Gateway first needs to be restarted for the changes to take effect. This can be done by either restarting the PC or by opening Command Prompt as an administrator and typing `net stop ignition` in the Command Prompt window, waiting until the messages say that Ignition is stopped, followed by `net start ignition`.

A.3 Connection between Unity and OPC UA server

This section gives a step-by-step guide on how to establish a connection with the Ignition OPC UA Server using the Unity client.

1. First, a logic simulator needs to be added to Unity. This can be done by adding an empty GameObject in a Unity project and adding the *Pre Logic Simulator* script to this GameObject. More information about logic simulators can be found in Chapter 3.

2. The next steps relate to the Gateway settings tab within the Pre Logic Simulator. See Figure A.2 for a visual indication of the settings within the Pre Logic Simulator.

   (a) The *Adapter Target* should be set to OPC_UA via the drop-down menu.

   (b) The *XmlFilePath* needs to be named. This will create a file containing the policy of the Adapter Settings. A standard name that can be used for this is *policy.xml*. When another name is used, make sure it has the *.xml* extension.

   (c) The *EndPointUrl* should contain the Endpoint URL of the Ignition OPC UA server. This can be found in Ignition under `Config > opc client > opc connections > more > endpoint`.

   (d) The *ConfigurationFilePath* contains a path to the Configuration file of Prespective, this should be correct by default.

   (e) In the case of connecting to the Ignition OPC UA server, the IdentifierTypeOpc should be set to *String*.

   (f) The other three settings can be left alone for now. More information on the Gateway Settings can be found in the Prespective documentation of OPC UA [6].

   (g) To save the Gateway settings, the *Export Policy to XML* button needs to be pressed.

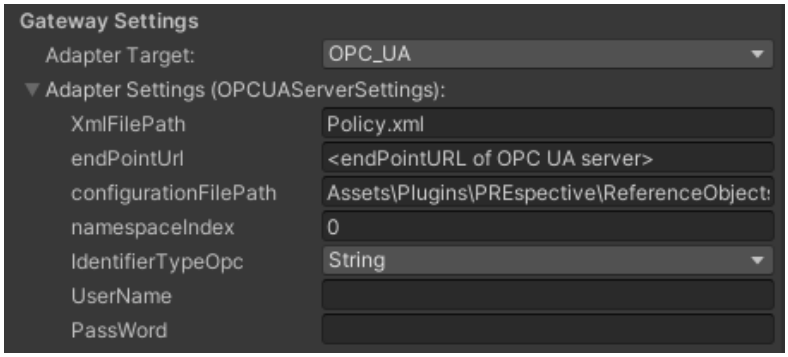

Figure A.2: Gateway Settings in Unity

[6] https://unit040.atlassian.net/wiki/spaces/PUD/pages/1212979579/2020.1.60.3+OPC+UA

3. Click play within Unity, the error *ServiceResultException: Error establishing a connection: Error received from remote host: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target* should pop up, since certificates need to be validated to establish the legitimacy of the connection.

4. The next steps relate to using UaExpert.

   (a) Open UaExpert and click the plus button in the task bar (see number 1 in Figure A.3).

   (b) In *Custom Discovery*, double click to add a server. The URL of the server should be the same as the endpoint used for Unity.

   (c) If done correctly, the tree can be extended, and the *Basic256Sha256* security level can be chosen, click *OK*. The server should now be visible under the Servers tab (left-hand side of the window).

   (d) Click on the plug connector in the taskbar to connect to the server (see number 2 in Figure A.3). A pop-up should now show up similar to Figure A.4. On this pop-up click *Trust Server Certificate*, this accepts the certificate of the server on the client side.

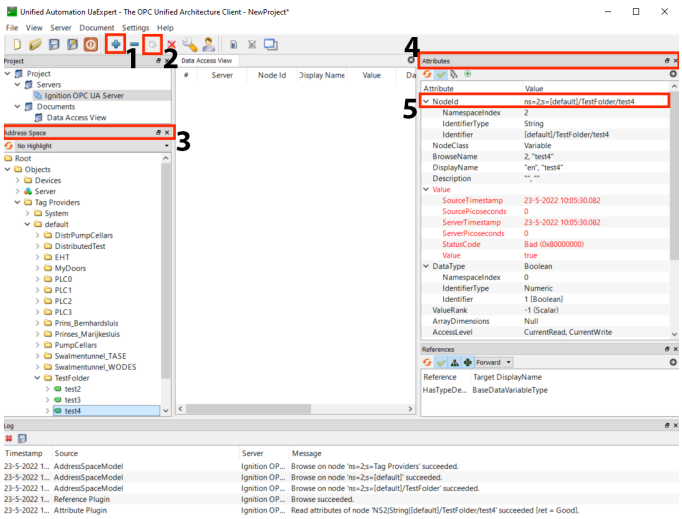

Figure A.3: Overview of UaExpert, with important parts highlighted.

Figure A.4: Certificate validation[7]

5. The certificates on the server side also need to be accepted, to do this, go to `Config > opc client > security > server`. Trust the untrusted certificates of UaExpert and Unity.

6. Click the plug connector in the taskbar of UaExpert once more (see number 2 in Figure A.3), if done correctly, UaExpert should now be able to connect to the Ignition OPC UA server.

7. Click play within Unity, if done correctly, info messages similar to the ones in Figure A.5 should be seen, indicating a successful connection to the OPC UA server of Ignition.

[7] https://documentation.unified-automation.com/uaexpert/1.4.2/html/first_steps.html

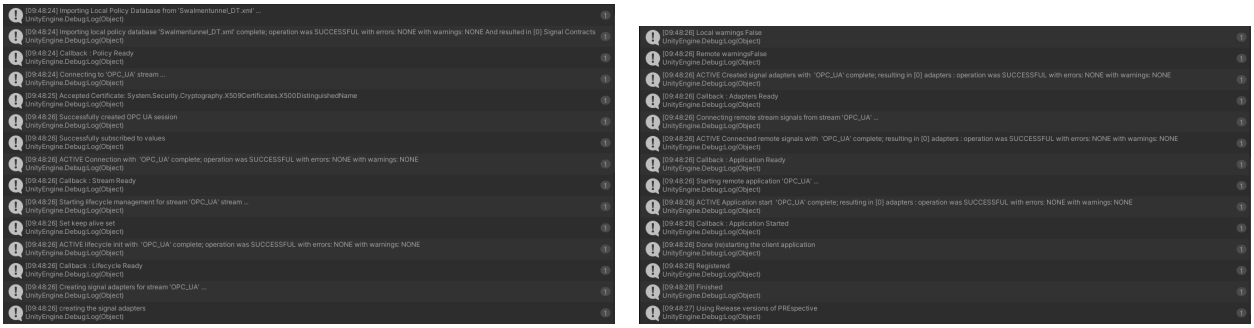

Figure A.5: Succesful connection info messages

A.4 Linking PLC variables to Unity variables

After establishing connection to the OPC UA server, the next step is to read or write variables from the PLC in Unity. The main challenge is that Unity is connected to the OPC UA server of Ignition, while the desired variables are on the OPC DA server of the PLC. To solve this, it is possible to link the Unity variables to OPC UA tags, which in turn are connected with the PLC variables. This process is visualized in Figure A.6. This section describes how to setup the required variables required for data interchange.

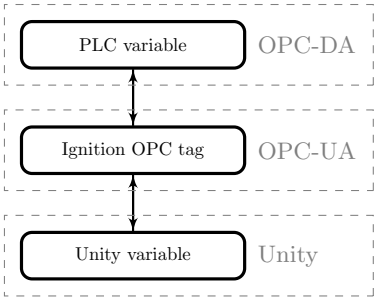

Figure A.6: Variable linkage

1. First, make sure the tag providers are exposed. This can be done in the Ignition gateway via `Config > OPC UA > Server Settings > Show advanced properties > Expose Tag Providers`. If done correctly, the Tag providers folder should be visible in the OPC quick client in the Ignition gateway (`Config > OPC Client > OPC Quick Client`). Alternatively, this can be checked via UaExpert by looking in the Address Space tab (see number 3 in Figure A.3) and opening the *Tag Providers* folder while connected to the Ignition OPC UA server.

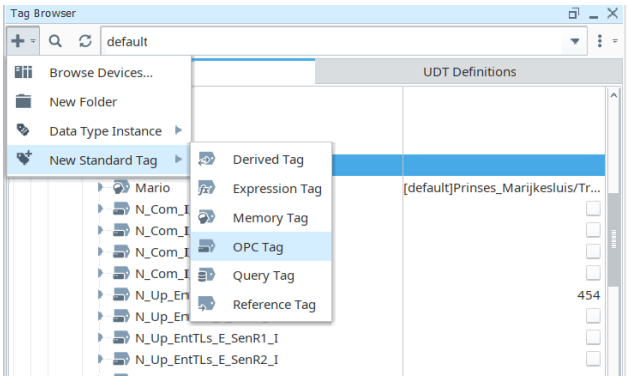

Figure A.7: Design launcher adding tag

2. Add an Ignition OPC tag to the Ignition OPC UA server. This can be done by opening the `Ignition Designer Launcher> Opening an arbitrary project` once inside the project a tag can be added by going to `Tag Browser > Plus sign > New Standard Tag > OPC Tag`, as in Figure A.7. Make sure the target server is set to the ABB OPC server. In the tag path, choose the PLC variable to which the OPC tag should be connected. Verify that the *Data type* of the OPC tag is the same as the date type of the PLC variable. Give the tag a clear name, as this will be needed in Unity. Figure A.8 shows the settings page of a tag. Here the tag can be given a *Name* under `Basic Properties`, the *Data type* can be adjusted under `Value` settings, the *OPC Server* setting needs to be set to "ABB OPC server" and *OPC Item Path* should point to the path of a PLC variable.

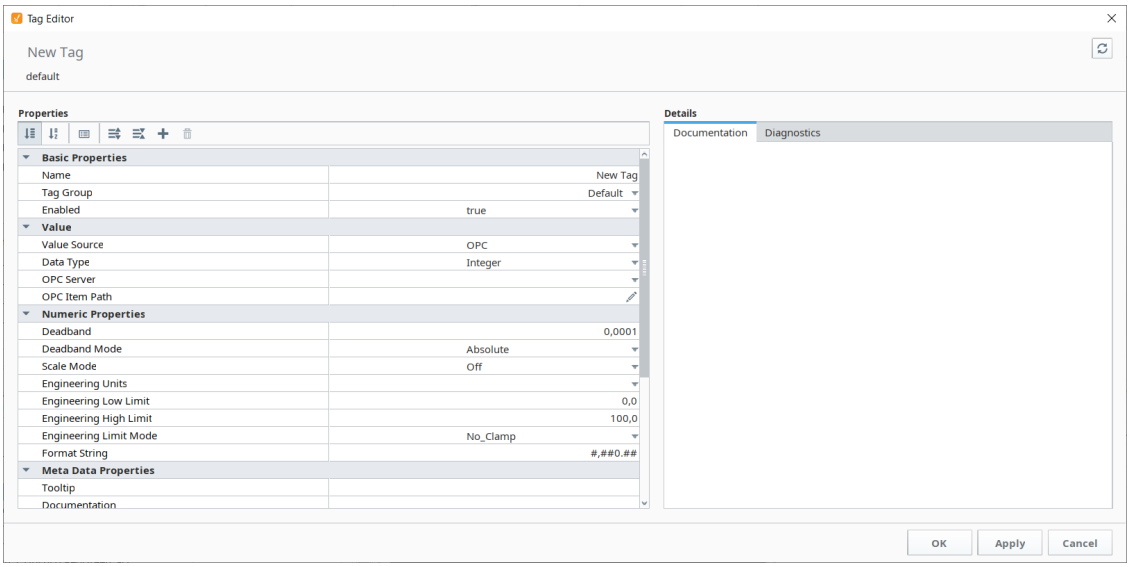

Figure A.8: Design launcher editing tag

3. Within Unity, variables need to be created. These variables are connected to the tags on the OPC UA server which allows the variables in Unity to read and write signals to the PLC. To achieve this, a script needs to be added to an empty GameObject, depending on the type of the variable and whether it needs to read a PLC variable (output) or write a PLC variable (input). Appendix A.6 provides an example of an Output Boolean variable. Next to this, the corresponding toggle file needs to be added, depending on the type of the variable, an example of a toggle file can also be found in Appendix A.6. The toggle file contains the variable which can be written or read by other Unity GameObjects. Figure A.9 provides an example of a GameObject with a (bool) toggle script.

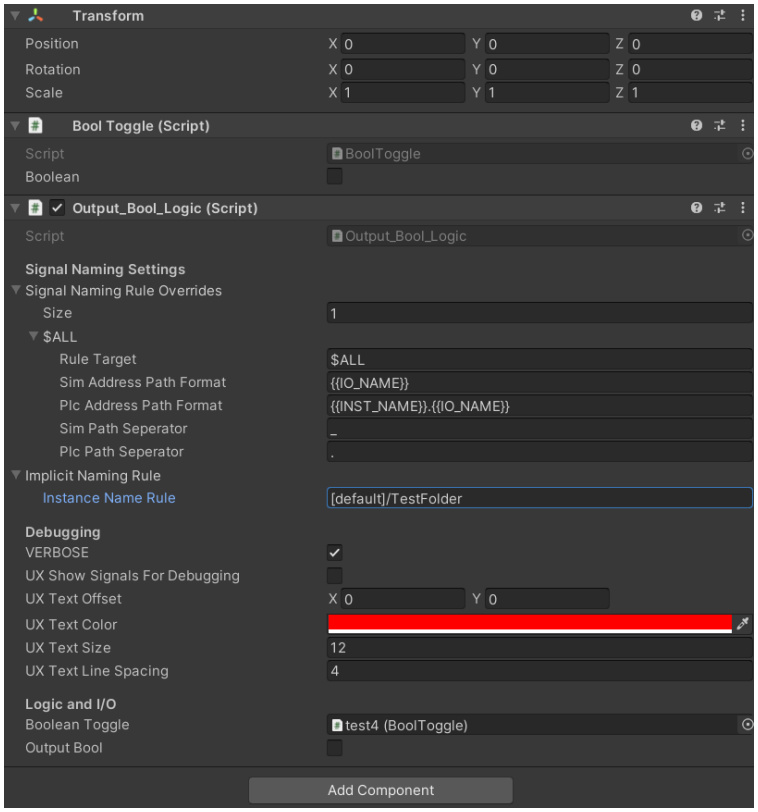

Figure A.9: Example of Output_Bool_Logic script

4. Figure A.9 shows an example of the *Output_Bool_Logic* script. To link an Unity variable to the created Ignition OPC tag, the following steps need to be taken.

   (a) Change the GameObject name to the name of the Ignition OPC tag. By right-clicking on the GameObject in the hierarchy window and selecting `Rename` or by changing the name at the top of the inspector window after selecting the GameObject.

   (b) Change the *Sim Address Path Format* under `Output_Bool_Logic > signal Naming Settings > Signal Naming Rule Overrides > $ALL` to {{*IO_NAME*}}.

   (c) Change the *Plc Address Path Format* under `Output_Bool_Logic > signal Naming Settings > Signal Naming Rule Overrides > $ALL` to {{*INST_NAME*}}/{{*IO_NAME*}}.

   (d) Change the *Instance Name Rule* under `Output_Bool_Logic > signal Naming Settings > Signal Naming Rule Overrides > Implicit Naming rule` to the Identifier of the OPC tag, minus the tag name and leading forward slash. This can either be found via the Ignition Designer Launcher, by right clicking the tag and clicking *Copy Path*, or via UaExpert, by selecting the tag in the Address Space and copying the Identifier from the Attributes tab (see numbers 3&4 in Figure A.3). Both options will give the following result for the path of tag test4: `[default]/TestFolder/test4`, in Unity the following needs to be filled in for the *Instance Name Rule*: `[default]/TestFolder`.

   (e) Additionally, the namespaceIndex setting and the IdentifierTypeOpc need to be changed in the Gateway Settings of the Prelogic Simlator to match the NodeID. This can be found in UaExpert in the Attributes tab of the tag (see numbers 4 and 5 in Figure A.3). These settings should be the same for all OPC tags.

   (f) To save these settings, the *Export Policy to XML* button needs to be pressed in the Gateway Settings tab.

5. If done correctly, a change of a writable variable in Unity should result in a change of the corresponding Ignition OPC tag value, and the corresponding PLC value. The Ignition OPC tag can be checked in the Ignition Designer Launcher, and the PLC variable changing can be checked in the ABB Compact Control Builder. The change of a readable variable in Unity should happen after changing the corresponding PLC variable value in the ABB Compact Control Builder.

The PLC address in the xml policy file from Appendix A.4, which can be opened by clicking "Open Policy File" in the Gateway settings, see Figure A.2, should contain the same information as the Identifier of the OPC tag for each variable. The Unity variable can now be used within Unity for e.g. an IO-script as described in a previous report [5].

A.5 Troubleshooting the connection

It can be difficult to pinpoint the cause of a problem when connecting fails. It is important to note that this appendix section assumes that the connection from Unity to the Ignition OPC UA server is done locally. When trying to connect remotely, these steps will likely not result in a successful connection. A helpful software program that may help troubleshooting is the OPC UA ANSI C Demo Server [8]. This demo server sets up a very simple and bare boned OPC UA server, allowing the user to connect to this server via either UaExpert or via Unity. Remember that connecting via Unity first requires a connection via UaExpert in order to accept the certificates. Using the Demo Server can thus help with identifying whether the problem lies within Unity or the Ignition Gateway. Below are a few errors and problems defined with their solutions.

- **ServiceResultException: Error establishing a connection: Error received from remote host: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target**

  The origin of this error has to do with certificates. To solve this problem the certificates need to be accepted locally as well as on the OPC UA server. See step 4 and 5 of the step-by-step guide of Appendix A.3.

- **Unable to access the Ignition Design Launcher on a remote laptop**

  This can be solved by opening the port of the Ignition Gateway (8088 by default). To do this, first search for `Firewall & Network protection`. In this window, choose for `Advanced settings` and `Inbound Rules` or `Outbound Rules`. It is recommended to do the following steps for both the `Inbound Rules` and `Outbound Rules`. Click `New Rule > Port > TCP > Specific local ports`. On this screen the port number can be added (8088 by default). Select `Allow the connection` and apply to all networks.

- **FormatException: Input string was not in a correct format**

  Set the *IdentifierTypeOpc* in the Gateway Settings of the Pre Logic Simulator to *String*. This step can also be found back in step 2e of Appendix A.3.

- **ServiceResultException: Error establishing a connection: BadNotConnected**

  This error can have multiple sources. One source is that a previous connection has not been terminated properly. The main source is probably caused by trying to use a remote connection which is not yet possible. A solution for this problem is to continue locally. Meaning, using Unity on the same PC that hosts the OPC server.

- **ServiceResultException: Error establishing a connection: BadChannelSecureClosed**

  Most likely a wrong port is used within Unity for the *EndPointUrl*. Check if the correct *EndPointUrl* is used by going to `Config > opc client > opc connections > more > endpoint`.

[8] https://www.unified-automation.com/downloads/opc-ua-servers.html

A.6 Logic component script

A.6.1 Logic and toggle scripts

Listing A.1: C# code of IntToggle

```
using UnityEngine;

public class IntToggle : MonoBehaviour
{
    [Tooltip("Integer Value of the toggle")]
    public int Integer = 0;

    public void Toggle(int oToggle)
    {
        Integer = oToggle;
    }
}
```

Listing A.2: C# code of Input_Int_Logic

```
using System;
using System.Collections.Generic;
using u040.prespective.prelogic;
using u040.prespective.prelogic.component;
using u040.prespective.prelogic.signal;
using UnityEngine;

public class Input_Int_Logic : PreLogicComponent
{
    #if UNITY_EDITOR || UNITY_EDITOR_BETA
        [HideInInspector] public int toolbarTab;
    #endif

    [Header("Logic and I/O")]

        [Tooltip("BooleanToggle component for the in or output")]
        public IntToggle IntegerToggle;
        [Tooltip("Input Boolean (used for input to the PLC)")]
        public int InputInt;

    #region <<PLC Signals>>
    #region <<Signal Definitions>>
    /// <summary>
    /// Declare the IO signals
    /// </summary>
    ///

    public override List<SignalDefinition> SignalDefinitions
    {
        get
        {
            return new List<SignalDefinition>
            {
                new SignalDefinition(gameObject.name, PLCSignalDirection.INPUT, ...
                    SupportedSignalType.REAL32, "", gameObject.name, null, null, 0f),
            };
        }
    }
    #endregion
    #endregion

    #region <<Update>>
    /// <summary>
    /// update the simulation component
```

```
44      /// </summary>
45      /// <param name="_simFrame">the current frame since start</param>
46      /// <param name="_ΔTime">the time since last frame</param>
47      /// <param name="_totalSimRunTime">total run time of the simulation</param>
48      /// <param name="_simStart">the time the simulation started</param>
49      protected override void onSimulatorUpdated(int _simFrame, float _ΔTime, float ...
            _totalSimRunTime, DateTime _simStart)
50      {
51          readComponent();
52      }
53      void readComponent()
54      {
55          if (IntegerToggle.Integer != InputInt)
56          {
57              InputInt = IntegerToggle.Integer;
58              WriteValue(gameObject.name, InputInt);
59          }
60      }
61  }
62  #endregion
```

Listing A.3: C# code of Output_Int_Logic

```
1    using   System;
2   using System.Collections.Generic;
3   using System.Reflection;
4   using u040.prespective.prelogic;
5   using u040.prespective.prelogic.component;
6   using u040.prespective.prelogic.signal;
7   using UnityEngine;
8
9
10  public class Output_Int_Logic : PreLogicComponent
11  {
12  #if UNITY_EDITOR || UNITY_EDITOR_BETA
13      [HideInInspector] public int toolbarTab;
14  #endif
15
16      [Header("Logic and I/O")]
17
18          [Tooltip("IntegerToggle component for the in or output")]
19          public IntToggle IntegerToggle;
20          [Tooltip("Input Integer (used for input to the PLC)")]
21          public int OutputInt;
22
23      #region <<PLC Signals>>
24      #region <<Signal Definitions>>
25      /// <summary>
26      /// Declare the IO signals
27      /// </summary>
28      ///
29
30      public override List<SignalDefinition> SignalDefinitions
31      {
32          get
33          {
34              return new List<SignalDefinition>
35              {
36                  new SignalDefinition(gameObject.name, PLCSignalDirection.OUTPUT, ...
                        SupportedSignalType.REAL32, "", "Value", onSignalChanged, null, 0f),
37              };
38          }
39      }
40      #endregion
41      #region <<PLC Outputs>>
42      /// <summary>
```

```
43      /// General callback for the IOs
44      /// </summary>
45      /// <param name="_signal">the signal that has changed</param>
46      /// <param name="_newValue">the new value</param>
47      /// <param name="_newValueReceived">the time of the value change</param>
48      /// <param name="_oldValue">the old value</param>
49      /// <param name="_oldValueReceived">the time of the old value change</param>
50      void onSignalChanged(SignalInstance _signal, object _newValue, DateTime ...
            _newValueReceived, object _oldValue, DateTime _oldValueReceived)
51      {
52          if (_signal.definition.defaultSignalName == gameObject.name)
53          {
54              OutputInt = (int)_newValue;
55              if (IntegerToggle.Integer != OutputInt)
56              {
57                  IntegerToggle.Integer = OutputInt;
58              }
59          }
60          else
61          {
62              Debug.LogWarning("Unknown Signal received:" + ...
                    _signal.definition.defaultSignalName);
63          }
64      }
65      #endregion
66  }
67  #endregion
```

A.6.2 Static functions

Listing A.4: C# code of Static_Functions_PLC

```
1    using   System.Collections.Generic;
2   using UnityEngine;
3
4   public class Static_Functions_PLC : MonoBehaviour
5   {
6       public static BoolToggle FindBoolToggle(GameObject IOName)
7       {
8           if (IOName.GetComponent<BoolToggle>())
9           {
10              // The variable is an output
11              return IOName.GetComponent<BoolToggle>();
12          }
13          if (IOName.GetComponent<IntToggle>())
14          {
15              // The variable is an output
16              Debug.LogWarning(IOName + " is an integer, rather than a boolean. Try ...
                    FindIntToggle instead.");
17              return null;
18          }
19          else
20          {
21              Debug.LogWarning(IOName + " is not an input or output variable");
22              return null;
23          }
24      }
25
26      public static IntToggle FindIntToggle(GameObject IOName)
27      {
28          if (IOName.GetComponent<IntToggle>())
29          {
30              // The variable is an output
31              return IOName.GetComponent<IntToggle>();
32          }
33          if (IOName.GetComponent<BoolToggle>())
```

```
34          {
35              // The variable is an output
36              Debug.LogWarning(IOName + " is a boolean, rather than an integer. Try ...
                    FindBoolToggle instead.");
37              return null;
38          }
39          else
40          {
41              Debug.LogWarning(IOName + " is not an input or output variable");
42              return null;
43          }
44      }
45  }
```

A.6.3 IO script

Listing A.5: C# code of IO_Barrier_PLC

```
1   using System.Collections.Generic;
2  using UnityEngine;
3
4  public class IO_Barrier_PLC : MonoBehaviour
5  {
6      [Header("Output logic component names")]
7
8          [SerializeField, Tooltip("Open the barrier")]
9          private GameObject a_open;
10
11     [Header("Input logic component names")]
12
13         [SerializeField, Tooltip("Opening the barrier")]
14         private GameObject s_opening;
15
16         [SerializeField, Tooltip("Closing the barrier")]
17         private GameObject s_closing;
18
19         [SerializeField, Tooltip("no movement")]
20         private GameObject s_stop;
21
22         [SerializeField, Tooltip("The barrier is fully opened")]
23         private GameObject s_opened;
24
25         [SerializeField, Tooltip("The barrier is fully closed")]
26         private GameObject s_closed;
27
28     [Header("Settings")]
29
30         [SerializeField, Tooltip("Warning box that shows when multiple actuators are ...
               turned on")]
31         private GameObject WarningBox;
32
33         [SerializeField, Tooltip("Collider that prevents vehicles from driving through the ...
               barrier")]
34         private GameObject BarrierCollider;
35
36         [SerializeField, Tooltip("Accepted offset of the motor position sensing [deg]")]
37         private float SensorOffsetPos;
38
39     private Actuator_RotationalMotor motor;
40     private Vector3 colliderStartPosition;
41
42     [HideInInspector]
43     public Dictionary<string, BoolToggle> IO_dict_Bool = new Dictionary<string, BoolToggle>();
44     public Dictionary<string, IntToggle> IO_dict_Int = new Dictionary<string, IntToggle>();
45
46     private void Start()
```

```
47      {
48          colliderStartPosition = BarrierCollider.transform.position;
49
50          // Components to communicate
51          motor = GetComponentInChildren<Actuator_RotationalMotor>();
52
53
54          // Filling the I/O dictionary
55
56              // Actuators
57              IO_dict_Int.Add("a_open", Static_Functions_PLC.FindIntToggle(a_open));
58
59              // Sensors
60              IO_dict_Bool.Add("s_opening", Static_Functions_PLC.FindBoolToggle(s_opening));
61              IO_dict_Bool.Add("s_closing", Static_Functions_PLC.FindBoolToggle(s_closing));
62              IO_dict_Bool.Add("s_stop", Static_Functions_PLC.FindBoolToggle(s_stop));
63              IO_dict_Bool.Add("s_opened", Static_Functions_PLC.FindBoolToggle(s_opened));
64              IO_dict_Bool.Add("s_closed", Static_Functions_PLC.FindBoolToggle(s_closed));
65      }
66
67      private void Update()
68      {
69          // Check whether multiple actuators are on and activate a red box signal
70          // if (Static_Functions.CountTrueActuators(IO_dict) > 1)
71          // {
72          //     WarningBox.SetActive(true);
73          // }
74
75          // --- Enable events based on the actuator states ---
76          if (IO_dict_Int["a_open"].Integer % 10 == 0)
77          {
78              motor.RotationDirection = 1;
79          }
80          else if (IO_dict_Int["a_open"].Integer % 10 == 1)
81          {
82              motor.RotationDirection = -1;
83          }
84          else if (IO_dict_Int["a_open"].Integer % 10 == 2)
85          {
86              motor.RotationDirection = 0;
87          }
88          else if (IO_dict_Int["a_open"].Integer % 10 == 3)
89          {
90              motor.RotationDirection = -1;
91          }
92          else
93          {
94              motor.RotationDirection = 0;
95          }
96
97          // --- Set the sensor values based on states of the components ---
98          IO_dict_Bool["s_opened"].Boolean =
99              Mathf.Abs(motor.OpenRotation - motor.currentRotation) < SensorOffsetPos;
100         IO_dict_Bool["s_closed"].Boolean =
101             Mathf.Abs(motor.ClosedRotation - motor.currentRotation) < SensorOffsetPos;
102
103         IO_dict_Bool["s_opening"].Boolean =
104             motor.RotationDirection == 1 && !IO_dict_Bool["s_opened"].Boolean;
105         IO_dict_Bool["s_closing"].Boolean =
106             motor.RotationDirection == -1 && !IO_dict_Bool["s_closed"].Boolean;
107         IO_dict_Bool["s_stop"].Boolean =
108             motor.RotationDirection == 0;
109
110         // Move the collider out of the way when the barrier is opened
111         if (IO_dict_Bool["s_opened"].Boolean)
112         {
113             BarrierCollider.transform.position = Vector3.Lerp(transform.position, ...
                    colliderStartPosition + Vector3.up * 20, Time.ΔTime * 100);
```

```
114             }
115             else
116             {
117                 BarrierCollider.transform.position = Vector3.Lerp(transform.position, ...
                        colliderStartPosition, Time.ΔTime * 100);
118             }
119 
120             // Debug.Log((Mathf.Abs(motor.ClosedRotation - motor.currentRotation)).ToString() ...
                    + " closed " + gameObject.name);
121         }
122     }
```

A.7 IP address error

```
ArgumentException: IP Address is invalid
Parameter name: name
Org.BouncyCastle.Asn1.X509.GeneralName..ctor (System.Int32 tag, System.String name) (at ...
    <4b876a5efd1b42a095fbfedce2ffc961>:0)
CertificateFactory.CreateSubjectAlternateNameDomains ...
    (System.Collections.Generic.IList`1[T] domainNames) (at ...
    <dc6e3f13df7d40c7a65f31d64b38a0b7>:0)
CertificateFactory.CreateCertificate (System.String storeType, System.String storePath, ...
    System.String password, System.String applicationUri, System.String applicationName, ...
    System.String subjectName, System.Collections.Generic.IList`1[T] domainNames, ...
    System.UInt16 keySize, System.DateTime startTime, System.UInt16 lifetimeInMonths, ...
    System.UInt16 hashSizeInBits, System.Boolean isCA, ...
    System.Security.Cryptography.X509Certificates.X509Certificate2 issuerCAKeyCert, ...
    System.Byte[] publicKey, System.Int32 pathLengthConstraint) (at ...
    <dc6e3f13df7d40c7a65f31d64b38a0b7>:0)
Opc.Ua.Configuration.ApplicationInstance+<CreateApplicationInstanceCertificate>d__55.MoveNext ...
    () (at <f0c8c481657f471c9bea6b518609558a>:0)
--- End of stack trace from previous location where exception was thrown ---
System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw () (at ...
    <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess ...
    (System.Threading.Tasks.Task task) (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification ...
    (System.Threading.Tasks.Task task) (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.ValidateEnd (System.Threading.Tasks.Task task) ...
    (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter`1[TResult].GetResult () (at ...
    <9577ac7a62ef43179789031239ba8798>:0)
Opc.Ua.Configuration.ApplicationInstance+<CheckApplicationInstanceCertificate>d__51.MoveNext ...
    () (at <f0c8c481657f471c9bea6b518609558a>:0)
--- End of stack trace from previous location where exception was thrown ---
System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw () (at ...
    <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess ...
    (System.Threading.Tasks.Task task) (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification ...
    (System.Threading.Tasks.Task task) (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.ValidateEnd (System.Threading.Tasks.Task task) ...
    (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter`1[TResult].GetResult () (at ...
    <9577ac7a62ef43179789031239ba8798>:0)
u040.prespective.prelogic.adapters.opcua.OPCUAClient+<CreateSession>d__21.MoveNext () (at ...
    <8676f7efb13e4f2193727a070f7c44cb>:0)
--- End of stack trace from previous location where exception was thrown ---
System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw () (at ...
    <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess ...
    (System.Threading.Tasks.Task task) (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification ...
    (System.Threading.Tasks.Task task) (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.ValidateEnd (System.Threading.Tasks.Task task) ...
    (at <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.TaskAwaiter.GetResult () (at ...
    <9577ac7a62ef43179789031239ba8798>:0)
u040.prespective.prelogic.adapters.opcua.OPCUAStreamAdapter+<Connect>d__13.MoveNext () (at ...
    <8676f7efb13e4f2193727a070f7c44cb>:0)
--- End of stack trace from previous location where exception was thrown ---
System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw () (at ...
    <9577ac7a62ef43179789031239ba8798>:0)
System.Runtime.CompilerServices.AsyncMethodBuilderCore+<>c.<ThrowAsync>b__6_0 ...
    (System.Object state) (at <9577ac7a62ef43179789031239ba8798>:0)
UnityEngine.UnitySynchronizationContext+WorkRequest.Invoke () (at ...
    <a555715ef291462eb190c2c939543f55>:0)
UnityEngine.UnitySynchronizationContext:ExecuteTasks()
```

Appendix B

Overview of Inputs and Outputs and Validation tests

This section provides the Inputs and Outputs used for the performed HIL simulation, as well as their Identifiers. Appendix B.2 provides the common part of the Identifiers for each main component, after which the x's need to be replaced with the variable names in Appendix B.1 to obtain the full Identifier of each individual variable. Appendix B.1 also shows which variable was used for which component and whether it was an Integer (i) or Boolean (b).

B.1 Overview Inputs and Outputs

Component	Input Variables	Output Variables
Aid cabinet A	HulppostkastA1Open (b) HulppostkastA1Noodtelefoon (b) HulppostkastA1Handblusser (b) HulppostkastA1Brandslang (b)	
Aid cabinet C	HulppostkastC1Open (b) HulppostkastC1Noodtelefoon (b) HulppostkastC1Handblusser (b)	
Boom barrier	AfsluitbomenBewegingGeen (b) AfsluitbomenBewegingNeer (b) AfsluitbomenBewegingOp (b) AfsluitbomenStandHoog (b) AfsluitbomenStandLaag (b) AfsluitbomenObstakeldetectie (b)	AfsluitboomCommando(i)
Emergency Exit	VluchtdeurOpenSensor (b)	VluchtdeurContourverlichtingStand (i) VluchtdeurGeluidsbakenStand (i)
Height detection	HoogteDetectie*XXX*Sensor (b)	HoogteDetectieToeritLampen (i)
Lighting		VerlichtingStand (i)
Lighting sensor	LichtsensorStand0 (b) LichtsensorStand1 (b) LichtsensorStand2 (b) LichtsensorStand3 (b) LichtsensorStand4 (b) LichtsensorStand5 (b) LichtsensorStand6 (b) LichtsensorStand7 (b) LichtsensorStand8 (b)	
Ventilation		VentilatieStand (i) VentilatieRichting (i)
Smoke detection	RookdetectieStand0 (b) RookdetectieStand1 (b) RookdetectieStand2 (b) RookdetectieStand3 (b) RookdetectieStand4 (b) RookdetectieStand5 (b) RookdetectieStand6 (b) RookdetectieStand7 (b) RookdetectieStand8 (b)	
SOS system	SOS (b)	
J32 system		J32 (i)
Traffic light		VRIstand (i)

Table B.1: List of input and output variables used for a traffic tube.

***XXX* can be replaced with either**
Advies, Waarschuwing* or *Rood

Component	Input Variables	Output Variables
Dynamic escape route indication system		DVIstand (i)
Lighting		VerlichtingStand (i)

Table B.2: List of input and output variables used for the central corridor.

Component	Input Variables	Output Variables
Calamity Passage North	CaDoNoordBewegingGeen (b) CaDoNoordBewegingNeer (b) CaDoNoordBewegingOp (b) CaDoNoordStandHoog (b) CaDoNoordStandLaag (b)	NoordCommando (i)
Calamity Passage South	CaDoZuidBewegingGeen (b) CaDoZuidBewegingNeer (b) CaDoZuidBewegingOp (b) CaDoZuidStandHoog (b) CaDoZuidStandLaag (b)	ZuidCommando (i)

Table B.3: List of input and output variables used for the emergency passages.

Component	Input Variables	Output Variables
Fire Extinguishing System	SensorsStandHoog (b) SensorsStandLaag (b)	PompStand (i)
Water Clean	KelderSchoonSensorStand1 (b) KelderSchoonSensorStand2 (b) KelderSchoonSensorStand3 (b) KelderSchoonSensorStand4 (b) KelderSchoonSensorStand4 (b)	KelderSchoonPompStand (i)
Water Dirty	KelderVuilSensorStand (b)	KelderVuilPompStand (i)

Table B.4: List of input and output variables used for the pomp cellars and fire extinguisher system.

B.2 Main parts Identifiers

Main component	Identifier
Traffictube 1	[default]Swalmentunnel_WODES/Sim_Tube1/xxx
Traffictube 2	[default]Swalmentunnel_WODES/Sim_Tube2/xxx
Pump cellars	[default]Swalmentunnel_WODES/Sim_Pompkelders/xxx
Central corridor	[default]Swalmentunnel_WODES/Sim_MTK/xxx
Emergency passage	[default]Swalmentunnel_WODES/Sim_CaDo/xxx
Fire extinguisher system	[default]Swalmentunnel_WODES/Sim_BBI/xxx

Table B.5: List of main components and their Identifiers.

B.3 Validation test

This appendix section contains the validation tests done for the components of the DT using HIL simulation. The supervisory controller and DT are reset every time a new test is performed. Each test is done using a step-by-step approach.

B.3.1 Traffic tube validation

Nr.	Action	Expected reaction	Result		Notes
			OK	NOK	
1.	GUI: [Press close button]	Traffic lights go trough states: *flashing, yellow and red*, J32 warning matrix turns on and boom barriers (long and short) close.	☑	☐	-
2.	GUI: [Press open button]	Traffic lights go trough states: *flashing, off*, J32 warning matrix turns off and boom barriers (long and short) open.	☑	☐	-
3.	DT: [Increase smoke level to 4]	Smoke level is increased at entrance of traffic tube, ventilation setting is automatically set to match the smoke level. State is set to Standby.	☑	☐	-
4.	DT: [Increase light level to 6]	Light level of the outside environment is increased, light level in the traffic tube is automatically set to match the light level of the environment.	☑	☐	-
5.	DT: [Decrease light level to 4]	Light level of the outside environment is decreased, light level in the traffic tube is automatically set to match the light level of the environment.	☑	☐	-
6.	DT: [Open aid cabinet] [Turn fire hose on] [Spawn stationary vehicle]	Traffic tube declares an emergency, other traffic tube becomes supporting, both traffic tubes close (see 1). Ventilation setting is increased to 8.	☑	☐	-
7.	GUI: [Set state to evacuation]	Emergency exit's contour lights and sound beacon turn on.	☑	☐	-
8.	GUI: [Set state to emergency]	Emergency exit's contour lights and sound beacon turn off.	☑	☐	-
9.	DT: [Remove traffic] [Turn off fire hose] [Close aid cabinet] [Decrease smoke level to 0]	Smoke level is non-existent at entrance of traffic tube, stationary vehicle is removed.	☑	☐	-
10.	GUI: [Set state to recovery] [Set state to operational]	Traffic tubes are opened (see 2) and ventilation setting is set to 2 on recovery, and to 0 on operational.	☑	☐	-
11.	**Exclusive traffic tube 1** DT: [Spawn extra high vehicle]	Extra high vehicle is being detected by *advice, warning and red* in that order. The height detection lights start to flicker in the same order. *Advice and warning* stop automatically, *red* does not stop automatically. On detection *red*, the traffic lights go trough states: *flashing, yellow and red* and J32 warning matrix turns on. Spawned vehicle stops before entrance tunnel.	☑	☐	-
12.	DT: [Remove traffic] GUI: [Manually turn traffic light off] [Press HDR reset button]	Traffic light switches to state *off*, J32 matrix turns off and the *red* height detection lights are turned off.	☑	☐	-

Table B.6: Validation test performed for traffic tube 1 and traffic tube 2

B.3.2 Emergency passage validation

The test described in Table B.7 have been performed for both Emergency passage north as south.

Nr.	Action	Expected reaction	Result		Notes
			OK	NOK	
1.	GUI: [Press open button]	Corresponding Emergency passage opens	☑	☐	-
2.	GUI: [Press close button]	Corresponding Emergency passage closes	☑	☐	-

Table B.7: Validation test performed for Emergency passage north and south

B.3.3 Pumping cellars and fire extinguisher system validation

Nr.	Action	Expected reaction	Result		Notes
			OK	NOK	
1.	DT: [Turn on a fire extinguisher]	Water level in the *Bluswater opslag* goes down. After a while the PLC turns on the pump to add more water. When the water level gets to high the pump is turned off.	☑	☐	-
2.	DT: [Turn off fire extinguisher]	Water level in the *Bluswater opslag* stays equal.	☑	☐	-
3.	DT: [Increase influx of clean water to 1]	Water level of the clean water storage increases, until the second height sensor turns on after which the pump is turned on. The water level is than reduced until the first two sensors are turned off. This process repeats.	☑	☐	-
4.	GUI: [Set control clean mode to manual] [Press store]	Water level of the clean water storage increases, until the fifth height sensor turns on after which the pump is turned on. The water level is than reduced until the last two sensors are turned off. This process repeats.	☑	☐	-
5.	GUI: [Press off]	Water level of the clean water storage increases, until capacity of the storage is reached.	☑	☐	-
6.	DT: [Increase influx of dirty water to 1]	Water level of the dirty water storage increases, until the maximum height is reached.	☑	☐	-
7.	DT: [Increase influx of dirty water to 1]	Water level of the dirty water increases until the sensor is turned on, after which the pump keeps the water level equal.	☑	☐	-
8.	GUI: [Set control dirty mode to manual] [Press off]	Water level of the dirty water storage increases until the capacity is reached.	☑	☐	-
9.	DT: [Decrease influx of dirty water to 0]	Water level of the dirty water stays level	☑	☐	-
10.	GUI: [Set control dirty mode to manual] [Press empty]	Water level of the dirty water storage decreases until the sensor turns off.	☑	☐	-

Table B.8: Validation test performed for the pumping cellars and fire extinguisher system

B.3.4 Central corridor validation

Nr.	Action	Expected reaction	Result		Notes
			OK	NOK	
1.	GUI: [Set route indication control mode to manual] [Press upward]	Upward indication light turns on.	☑	☐	-
2.	GUI: [Press downward]	Upward indication light turns off, Downward indication light turns on.	☑	☐	-
3.	GUI: [Press off]	Both indication lights turn off.	☑	☐	-
4.	GUI: [Set escape route lighting control mode to manual] [Press on]	The lighting in the central corridor turns on	☑	☐	-
5.	GUI: [Press on]	The lighting in the central corridor turns off	☑	☐	-
6.	GUI: [Set pressure system control mode to manual] [Press right]	The right pressure system indicator is turned on	☑	☐	-
7.	GUI: [Press left]	The left pressure system indicator is turned on	☑	☐	-

Table B.9: Validation test performed for the central corridor

B.3.5 Validation test videos

Some of the validation tests are recorded as proof that the results of the validation tests are true. Unfortunately, the recordings of the validation tests have been corrupted. Two videos could be recovered to give an indication of the validity of the results. The videos and their missing parts are discussed in this appendix section.

Unfortunately, due to corrupted video files, only part of the validation of traffic tube 1 is available. This video can be found on Youtube via this link: https://youtu.be/9rsGIvGpkng.

Due to the corrupted files, the emergency passage validation video is not available and only an old validation video for the pumping cellars is available. In this old video, the dirty water did not have a minimum height and thus was able to flow under the dirty water cellar's floor. This should be fixed and the video can be found on Youtube via this link: https://youtu.be/xK1b99Bi4Tw.

Appendix C

Code

This appendix includes the C# code used for the components added for operator training in Chapter 5.

Listing C.1: C# code of CCTVSwitching.cs

```
1   using   System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5
6  public class CCTVSwitching : MonoBehaviour
7  {
8      public GameObject BoomBarriers;
9      public GameObject EntranceTunnel;
10     public GameObject ExitTunnel;
11     public GameObject HeightDetection;
12     public GameObject EmergencyExit;
13     public GameObject VentillationPositions;
14     public GameObject WaterSuplyCameras;
15     public Text txt;
16     bool JustSwitched = false;
17     int t = 0;
18     private void Start()
19     {
20         BoomBarriers.SetActive(true);
21         EntranceTunnel.SetActive(false);
22         ExitTunnel.SetActive(false);
23         HeightDetection.SetActive(false);
24         EmergencyExit.SetActive(false);
25         VentillationPositions.SetActive(false);
26         WaterSuplyCameras.SetActive(false);
27         txt.text = "Boom Barriers";
28     }
29     void FixedUpdate()
30     {
31         if (JustSwitched)
32         {
33             t ++;
34             if (t > 25)
35             {
36                 JustSwitched = false;
37                 t = 0;
38             }
39         }
40         else
41         {
42             if (Input.GetKeyDown(KeyCode.RightArrow))
43             {
```

```
                if (BoomBarriers.activeSelf && !JustSwitched)
                {
                    BoomBarriers.SetActive(false);
                    EntranceTunnel.SetActive(true);
                    txt.text = "Tunnel Entrance";
                    JustSwitched = true;
                }
                if (EntranceTunnel.activeSelf && !JustSwitched)
                {
                    EntranceTunnel.SetActive(false);
                    ExitTunnel.SetActive(true);
                    txt.text = "Tunnel Exit";
                    JustSwitched = true;
                }
                if (ExitTunnel.activeSelf && !JustSwitched)
                {
                    ExitTunnel.SetActive(false);
                    HeightDetection.SetActive(true);
                    txt.text = "Height Detecion";
                    JustSwitched = true;
                }
                if (HeightDetection.activeSelf && !JustSwitched)
                {
                    HeightDetection.SetActive(false);
                    EmergencyExit.SetActive(true);
                    txt.text = "Emergency Exit";
                    JustSwitched = true;
                }
                if (EmergencyExit.activeSelf && !JustSwitched)
                {
                    EmergencyExit.SetActive(false);
                    VentillationPositions.SetActive(true);
                    txt.text = "Ventillation";
                    JustSwitched = true;
                }
                if (VentillationPositions.activeSelf && !JustSwitched)
                {
                    VentillationPositions.SetActive(false);
                    WaterSuplyCameras.SetActive(true);
                    txt.text = "Water Supply";
                    JustSwitched = true;
                }
                if (WaterSuplyCameras.activeSelf && !JustSwitched)
                {
                    WaterSuplyCameras.SetActive(false);
                    BoomBarriers.SetActive(true);
                    txt.text = "Boom Barriers";
                    JustSwitched = true;
                }
            }
        }
    }
}
```

Listing C.2: C# code of ActivateAllDisplays.cs

```
using UnityEngine;

public class ActivateAllDisplays : MonoBehaviour
{
    void Awake()
    {
        if (Display.displays.Length > 1)
        {
            Screen.fullScreen = false;
            Display.displays[1].Activate();
```

```
        }
    }
}
```

Listing C.3: C# code of CarControl.cs

```
 using  System.Collections;
using UnityEngine;

public class CarControl : MonoBehaviour
{
    public Rigidbody rb;
    public Transform car;
    public Vector3 offset;
    public float Raydistance;
    private Vector3 destination;
    private bool reachedDestination = true;
    private float stopDistance = 1;
    private float rotationSpeed = 120;
    public int acc = 1;
    public int dec = 1;
    public int minSpeed, maxSpeed;
    public float laneDistance;
    private float MaxSpeed;
    private float movementSpeed = 0;
    private bool ObstacleForward = false;
    private bool ObstacleRight = false;
    private bool ObstacleLeft = false;
    [SerializeField, Tooltip("Set right lane or left lane")]
    public bool RightLane = true;
    private bool JustSwitched = true;
    private string DestroyTag = "vehicle_destroy";
    private string VehicleTag = "Vehicle_driving";
    private string stopTag = "vehicle_stop";
    public int stepSize = 5;
    private float otherVehicleSpeed = 0;
    private float otherVehicleMaxSpeed = 0;
    private float Distance;
    private bool Driving = true;
    private Vector3 Rayorigin;
    [SerializeField, Tooltip("Toggle raycasting debugging")]
    public bool raycastDebug = true;
    private bool Switching = false;

    private void Start()
    {
        MaxSpeed = getSpeed(minSpeed, maxSpeed);
        destination = transform.position;
        movementSpeed = MaxSpeed / 2;
    }
    private void FixedUpdate()
    {
        // Determine Raycasts
        Rayorigin = transform.position + offset;
        RaycastHit hit1, hit2, hit3;
        Ray Forward = new Ray(Rayorigin, transform.TransformDirection(Vector3.forward));
        Ray parralelRight = new Ray(Rayorigin + new Vector3(laneDistance, 0, -20), ...
            transform.TransformDirection(Vector3.forward));
        Ray parralelLeft = new Ray(Rayorigin + new Vector3(-laneDistance, 0, 15), ...
            transform.TransformDirection(Vector3.back));
        // Draw Raycast (for debugging)
        if (raycastDebug)
        {
            Debug.DrawRay(Rayorigin, transform.TransformDirection(Vector3.forward) * ...
                Raydistance);
            Debug.DrawRay(Rayorigin + new Vector3(laneDistance, 0, -25), ...
```

```
                transform.TransformDirection(Vector3.forward) * Raydistance * 2);
            Debug.DrawRay(Rayorigin + new Vector3(-laneDistance, 0, 20), ...
                transform.TransformDirection(Vector3.back) * Raydistance * 1.5f);
        }
        // Detect hit on raycasts
        if (Physics.Raycast(Forward, out hit1, Raydistance))
        {
            if (hit1.collider.CompareTag(VehicleTag))
            {
                ObstacleForward = true;
                otherVehicleSpeed = hit1.transform.GetComponent<CarControl>().movementSpeed;
                otherVehicleMaxSpeed = hit1.transform.GetComponent<CarControl>().MaxSpeed;
                Distance = hit1.distance;
                Driving = true;
            }
        }
        else
        {
            ObstacleForward = false;
            otherVehicleSpeed = 0;
            Driving = true;
            gameObject.tag = VehicleTag;
        }
        if (Physics.Raycast(parralelRight, out hit2, Raydistance*2))
        {
            if (hit2.collider.CompareTag(VehicleTag) || hit2.collider.CompareTag(stopTag))
            {
                ObstacleRight = true;
            }
        }
        else
        {
            ObstacleRight = false;
        }
        if (Physics.Raycast(parralelLeft, out hit3, Raydistance*1.5f))
        {
            if (hit3.collider.CompareTag(VehicleTag) || hit3.collider.CompareTag(stopTag))
            {
                ObstacleLeft = true;
            }
        }
        else
        {
            ObstacleLeft = false;
        }

        if (Physics.Raycast(Forward, out hit1, Raydistance / 2))
        {
            if (hit1.collider.CompareTag(stopTag))
            {
                Driving = false;
                gameObject.tag = stopTag;
                movementSpeed = MaxSpeed / 2;
            }
        }

        // Determine destination
        // Right lane decisions
        if (RightLane && Driving)
        {
            // If no obstacles, keep going
            if (reachedDestination)
            {
                Destination(0);
            }
            // If Obstacle is in front of car, move to the leftlane if clear
            if (!Switching && ObstacleForward && !ObstacleLeft && otherVehicleSpeed > 0 && ...
                otherVehicleMaxSpeed < MaxSpeed - stepSize)
```

```
123             {
124                 Destination(-laneDistance);
125                 RightLane = false;
126                 JustSwitched = true;
127             }
128             else if (ObstacleForward && !ObstacleLeft && otherVehicleSpeed > 0 && ...
                    otherVehicleMaxSpeed > MaxSpeed - stepSize)
129             {
130                 if (movementSpeed > otherVehicleSpeed)
131                 {
132                     movementSpeed -= dec / 3.6f * Time.fixedDeltaTime;
133                 }
134                 else if (movementSpeed < maxSpeed && Distance > Raydistance / 2)
135                 {
136                     movementSpeed += acc / 3.6f * Time.fixedDeltaTime;
137                 }
138             }
139             // If Obstacle in front of car and left lane is not clear, deaccelate
140             else if (ObstacleForward && ObstacleLeft)
141             {
142                 Destination(0);
143                 if (movementSpeed > otherVehicleSpeed)
144                 {
145                     movementSpeed -= dec / 3.6f * Time.fixedDeltaTime;
146                 }
147                 else if (movementSpeed < maxSpeed && Distance > Raydistance / 2)
148                 {
149                     movementSpeed += acc / 3.6f * Time.fixedDeltaTime;
150                 }
151             }
152         }
153 
154         //Left lane decisions
155         if (!RightLane && Driving)
156         {
157             // Go straight for the first step
158             if (JustSwitched && reachedDestination)
159             {
160                 Destination(0);
161                 JustSwitched = false;
162                 reachedDestination = false;
163             }
164             // Turn back to right lane if everything is clear
165             if (!ObstacleRight && reachedDestination && !JustSwitched && !ObstacleForward)
166             {
167                 Destination(laneDistance);
168                 RightLane = true;
169             }
170             else if (reachedDestination)
171             {
172                 Destination(0);
173             }
174 
175             if (ObstacleForward && movementSpeed > otherVehicleSpeed)
176             {
177                 movementSpeed -= dec / 3.6f * Time.fixedDeltaTime;
178             }
179             else if (ObstacleForward && movementSpeed < MaxSpeed && Distance > Raydistance ...
                    / 2)
180             {
181                 movementSpeed += acc / 3.6f * Time.fixedDeltaTime;
182             }
183         }
184 
185         if (!ObstacleForward && Driving)
186         {
187             if (movementSpeed < MaxSpeed)
188             {
```

```
189                 movementSpeed += acc / 3.6f * Time.fixedDeltaTime;
190             }
191         }
192 
193         // If the destination is not reached, move towards the destination until you're in ...
                a certain range
194         if (transform.position != destination && Driving)
195         {
196             Vector3 destinationDirection = destination - transform.position;
197             destinationDirection.y = 0;
198 
199             float destinationDistance = destinationDirection.magnitude;
200 
201             if (destinationDistance ≥ stopDistance)
202             {
203                 reachedDestination = false;
204                 Quaternion targetRotation = Quaternion.LookRotation(destinationDirection);
205                 transform.rotation = Quaternion.RotateTowards(transform.rotation, ...
                        targetRotation, rotationSpeed * Time.ΔTime);
206                 transform.Translate(Vector3.forward * movementSpeed * Time.ΔTime);
207             }
208             else
209             {
210                 reachedDestination = true;
211             }
212         }
213     }
214     private void OnTriggerEnter(Collider other)
215     {
216         if (other.gameObject.CompareTag(DestroyTag))
217         {
218             // Warp the vehicle such that it exits all triggers before being destroyed
219             StartCoroutine(WarpAndDestroy());
220         }
221 
222     }
223 
224     public void WarpAndDestroyVehicle()
225     {
226         // Warp the vehicle such that it exits all triggers before being destroyed
227         StartCoroutine(WarpAndDestroy());
228     }
229 
230     IEnumerator WarpAndDestroy()
231     {
232         movementSpeed = 10000;
233         yield return new WaitForSeconds(0.5f);
234         Destroy(this.gameObject);
235     }
236 
237     void Destination(float dest)
238     {
239         destination = transform.position + new Vector3(dest, 0, Raydistance);
240         if (dest != 0)
241         {
242             Switching = true;
243         }
244         else
245         {
246             Switching = false;
247         }
248     }
249 
250     int getSpeed(int minspeed, int maxspeed)
251     {
252         int randomSpeed = Random.Range(minspeed, maxspeed);
253         int numSteps = randomSpeed / stepSize;
254         int adjustedSpeed = (int)(Mathf.Round(numSteps) * stepSize);
```

```
255             return adjustedSpeed;
256         }
257     }
```

Listing C.4: C# code of Spawner.cs

```
1   using  UnityEngine;
2
3   public class Spawner : MonoBehaviour
4   {
5       [Header("Vehicle Prefab Objects")]
6
7       [SerializeField, Tooltip("Array of vehicles that this spawner generates")]
8       private GameObject[] spawn_vehicles;
9
10      [Header("Spawner Settings")]
11
12      [SerializeField, Tooltip("Minimum time between spawining vehicles [s]")]
13      private float t_inter_min = 2;
14
15      [SerializeField, Tooltip("Maximum time between spawining vehicles [s]")]
16      private float t_inter_max = 6;
17
18      [SerializeField, Tooltip("Unoccupied distance in front of the spawner needed for ...
            spawning a new vehicle [m]")]
19      private float min_spawn_dist;
20
21      [SerializeField, Tooltip("Set right lane or left lane")]
22      private bool Rightlane = true;
23
24      [Header("Spawned vehicle settings")]
25
26      [SerializeField, Tooltip("Set minimum speed of spawned vehicles")]
27      private int MinSpeed;
28
29      [SerializeField, Tooltip("Set maximum speed of spawned vehicles")]
30      private int MaxSpeed;
31
32      [SerializeField, Tooltip("Set size of steps between speed")]
33      private int stepSize;
34
35      [SerializeField, Tooltip("Set deceleration and acceleration rate of spawned vehicles")]
36      private int dec, acc;
37
38      [SerializeField, Tooltip("Set distance of raycast of vehicles")]
39      private int Raydistance;
40
41      [SerializeField, Tooltip("Set distance between center of lanes")]
42      private float laneDistance;
43
44      [Header("Debugging options")]
45
46      [SerializeField, Tooltip("Set raycast lines visible or invisible")]
47      private bool raycastDebug;
48
49      private float timer;
50      private GameObject new_vehicle;
51      private float last_dist = 50;
52      [HideInInspector]
53      public bool SpawnerOn = false;
54      // Start is called before the first frame update
55      private void Start()
56      {
57          // Set the timer to a random value in the bounds
58          timer = Random.Range(t_inter_min, t_inter_max);
59          SpawnerOn = false;
```

```
60      }
61
62      private void Update()
63      {
64          timer -= Time.ΔTime;
65          // Determine whether to spawn a new gameobject
66          if (timer < 0 && last_dist > min_spawn_dist && SpawnerOn)
67          {
68              timer = Random.Range(t_inter_min, t_inter_max);
69              Spawn_vehicle();
70          }
71
72          // Obtain the distance from the spawner to the vehicle that was spawned last
73          // If the spawner is off, set the last distance to enable spawning when it starts ...
                again
74          if (SpawnerOn)
75          {
76              last_dist = Vector3.Magnitude(transform.position - ...
                    new_vehicle.transform.position);
77          }
78          else
79          {
80              last_dist = min_spawn_dist + 1;
81          }
82      }
83
84      private void Spawn_vehicle()
85      {
86          // Spawn a random vehicle from the vehicles array
87          new_vehicle = Instantiate(spawn_vehicles[Random.Range(0, spawn_vehicles.Length)], ...
                transform.position - new Vector3(0,0.5f,0), transform.rotation);
88          new_vehicle.transform.parent = transform.parent;
89          new_vehicle.GetComponent<CarControl>().RightLane = Rightlane;
90          new_vehicle.GetComponent<CarControl>().minSpeed = MinSpeed;
91          new_vehicle.GetComponent<CarControl>().maxSpeed = MaxSpeed;
92          new_vehicle.GetComponent<CarControl>().acc = acc;
93          new_vehicle.GetComponent<CarControl>().dec = dec;
94          new_vehicle.GetComponent<CarControl>().Raydistance = Raydistance;
95          new_vehicle.GetComponent<CarControl>().laneDistance = laneDistance;
96          new_vehicle.GetComponent<CarControl>().stepSize = stepSize;
97          new_vehicle.GetComponent<CarControl>().raycastDebug = raycastDebug;
98      }
99  }
```