Model Based Design

Graduation

How to visualize a digital twin in Unity, based upon data derived from 20-sim?

Bas Gunnink

Saxion University of Applied Sciences

September 28, 2018

Author Note

Bas Gunnink, Department Game Design and Production, Saxion University of Applied Sciences.

Correspondence concerning this article should be addressed to Bas Gunnink, Saxion University of Applied Sciences, Antwerpenstraat 30, 7543ZT Enschede, the Netherlands.
Contact: BGunnink@hotmail.com

Justification

Title:	How to visualize a digital twin in Unity, based upon data derived from 20-sim?
Pages:	28
Commissioner:	Controllab Products B.V.
Author:	Bas Gunnink
Saxion Coach:	Taco van Loon
Company Coach I:	Tom Bokhove – *Unity*
Company Coach II:	Frank Groen – *20-sim*
Company Coach III:	René Valenzuela – *Marketing*
Document Type:	Report
Status:	Final
Date:	September 28, 2018
Week:	20

Controllab Products B.V.

Hengelosestraat 500

7521 AN Enschede, The Netherlands

Tel:	085-7731872
E-mail:	info@controllab.nl
Web:	www.controllab.nl

Table of Contents

Abstract

This report explores what the best way is to visualize the modelling workflow of a digital twin. The digital twin is a digital replica of a physical plant and can be beneficial for building, maintaining, operating and training. The visualization helps to convince parties such as the management department of a company, which often lacks engineering insights. In addition, this research revealed that visualizations are also ideal for debugging. It becomes possible to instantly examine mistakes, for example in the control system, from a visual perspective instead of going through many lines of code. In my particular case, data from the digital twin has been visualized in the gaming engine *Unity*. This data was sent from the digital twin to *Unity* via the XML-RPC protocol. Components that were applicable to be visualized were identified and combined as a Unity Package, including generic code and custom interfaces for the developer. This is beneficial for future projects and speeds up the process for the developer. During the phase of prototyping, it became clear that maintaining a clean overview is important. Throughout an iterative and incremental process, each component is designed to be opened and closed by the user. Multiple data sources can be plotted into a single graphical chart, and the method of interaction has evolved into a minimal effort approach for the end-user.

Glossary

An alphabetical list of words relating to a specific subject, text, or dialect, with explanations; a brief dictionary.

Binary-encoded	A representation of symbols in a source alphabet by strings of binary digits.
Communication-layer	A layer that transfers data between two software packages.
Computer-Aided Design	A method to create precision drawings or technical illustrations.
Control System	A system that manages and commands the behaviour of other devices.
Digital Twin	A digital replica of a physical plant that can be used for various purposes.
Encoder	An electro-mechanical device that converts the angular position to analog or digital output signal.
Extended Reality	A form of reality such as virtual reality, mixed reality or augmented reality.
Function Mock-up Interface	A standardized interface to be used in co-simulations, which are simulations between multiple different computer tools.
Haptics	A form of feedback that recreates the sense of touch by applying forces or vibrations to the user.
Model-Based Design	A process to quickly and efficiently developing dynamical systems. For example system for controls, signal processing or communication.
Numerical Data	A measurable form of data that looks at amounts or quantities.
Object-Oriented Programming	A programming style based on "objects" that interact with each other, rather than sequential programming.
Physical Plant	A mechanical or industrial plant.
Predictive Maintenance	A process with the aim to predict when equipment failure might occur.
Programmable Logic Controller	A controller that is responsible for the manufacturing processes.
Proportional-Integral-Derivative Controller	A controller that corrects its itself based on a calculated error value.
Simulation	A replica of a real-world process or system.
Unified Modeling Language	A modeling language that helps with constructing and documenting systems.
Unity Package	A collection of Unity assets combined as an importable package.
XML-RPC	A protocol that that uses HTTP to transport its XML-encoded data.

1. Introduction

Controllab is an innovative company that specializes in model-based engineering. They are a spin-off of the University of Twente. Due to the high academic level, the given assignments can become challenging. This allows me to broaden my current expertise. One of many skill that I learned during my previous assignments at Controllab was to learn basic programming in C#. This is considered to be a valuable skill that fits with the gaming education I received and enables me to work more flexibly in the future. Over the last years, I found out that the serious gaming sector is more appealing to me than the regular gaming sector. I have the feeling that I can make an important contribution to this sector, with my knowledge of gaming techniques.

1.1. Background

The client, Controllab, works with model-based design, and performs numerical simulations that include a lot of data such as physics, controller design, and boolean logic. All this data is connected and simulated in their own developed software package called 20-sim. "*20-sim is a commercial modelling and simulation program for multi-domain dynamic systems. It is widely used for modelling complex multi-domain systems and development of control systems*".[01]

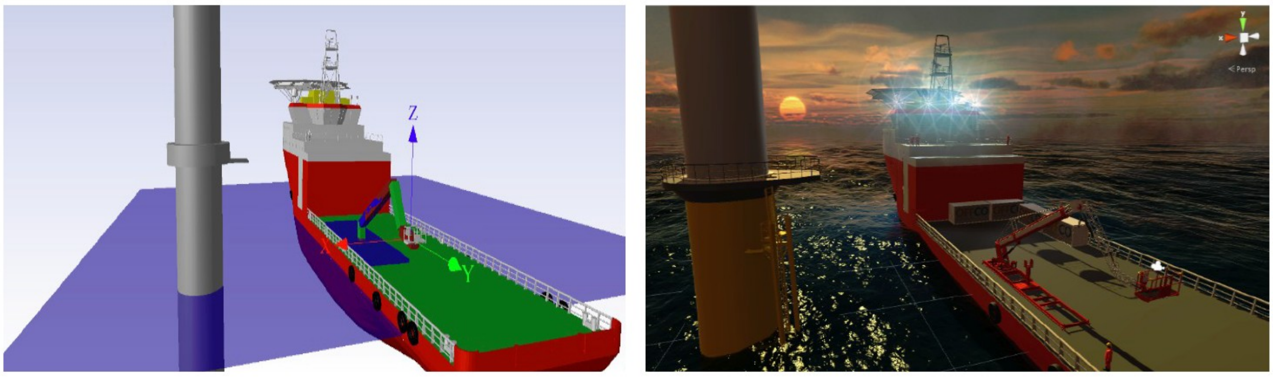


Figure 1: Scene rendered in 20-sim versus Unity.

One of the tools in 20-sim is the 3D animation toolbox. This toolbox makes it possible to create a three dimensional representation of your 20-sim model. However, 20-sim was never meant for complex visualization, but rather to give another view on the physical system that is being built. Therefore, to visualize numerical data, the gaming engine Unity will be used, as by request of Controllab. This is primarily due to its royalties, but also due to the powerful graphical capabilities of the engine and rapidity in supporting innovative developments from the gaming industry like virtual and augmented reality. Figure 1 shows an example of the graphical differences between a scene rendered in 20-sim and rendered in Unity.

1.2. Problem Analysis

Controllab creates control systems with model-based design. This is a technique that is used for the development and testing of systems. Specifications, operational procedures and performance descriptions are replaced by models and simulations. However, this service can be difficult to explain and potentially sell to customers. Most companies avoid investing in simulations due to high costs, lack of knowledge in this particular field and the feeling that nothing physical is being built while money has been spent. However, investing in simulations can improve control system. It helps to avoid future mistakes that can be critical and therefore more expensive at later stages of development. In brief, the main problem arises when having to convince parties such as the management department of a company, which often lacks engineering insights, to make them buy a controller based on a model-based approach.

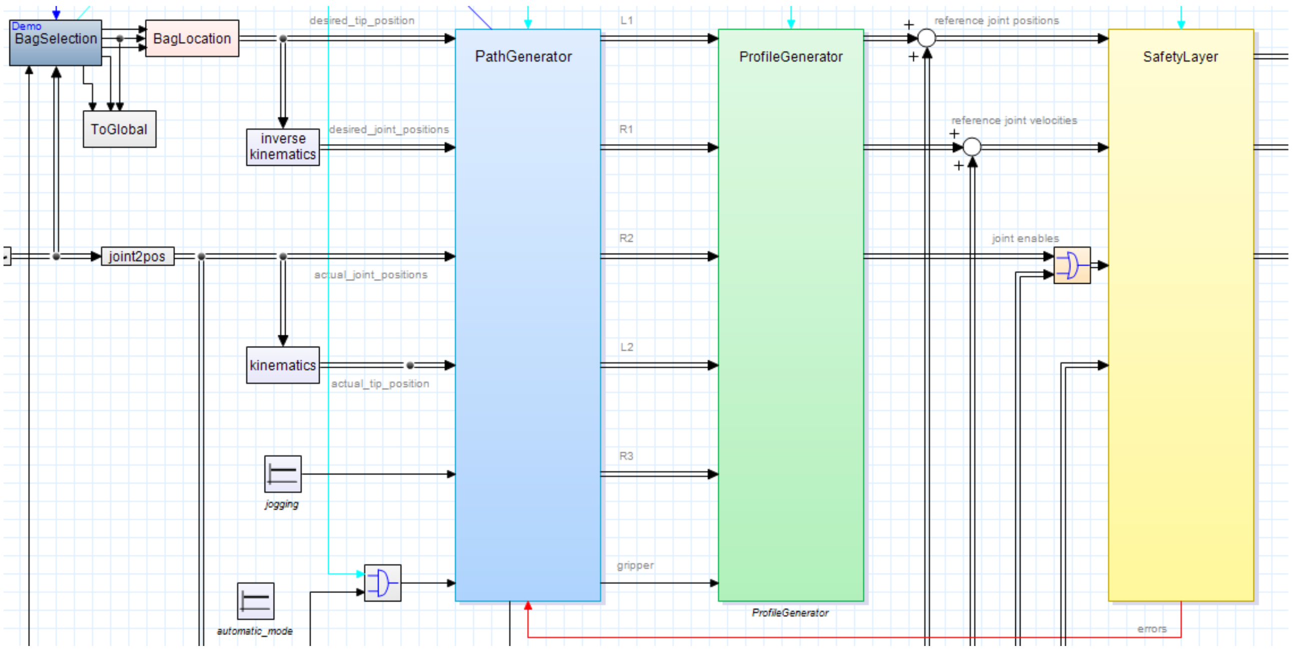


Figure 2: Graphical overview of the 20-sim model.

Figure 2 shows one of many graphical overviews that are present inside the 20-sim model. The solid displayed blocks are referred to as sub-models and contain equations or another layer of a graphical model. For example, the yellow sub-model (on the right side) is called "SafetyLayer" and is one of the relevant components that is necessary to built a valid control system. Building all these components in the proper order and to the necessary level of details is referred to as the modelling workflow. The 20-sim model can be exported to C-code, which can be used for several controller hardware targets, like the Programmable Logic Controllers (PLC) for robust, commercial projects, but also for targets like the raspberry pi, which serve a more educational market. This code includes handling the communication, reading and handling sensor data, monitoring the health of the system etc.

The goal throughout this research is to visualize the modelling workflow in an intuitive fashion. This makes it possible for Controllab to present their 20-sim models in a way that is more clear to people not directly concerned with the engineering aspects of a project. Besides making it easier to explain the ideas behind a 20-sim model, it can also be used to analyse the behaviour of the model. It is for example possible to reuse a visualization that shows the boundaries of your physical system (the working envelope) to see if your physical system stays within these boundaries at all times. Based on the problems above, a main research question has been formulated: *"How to visualize the modelling workflow of a digital twin?".*

This main research question has been divided into several sub-questions:

- What is a digital twin and how will data visualization help to improve this digital twin?
- How to transfer data between 20-sim and Unity?
- Which components from the modelling workflow are most applicable to visualize?
    - 3D-representation of the physical plant to observe it in virtual-space.
    - Finite state machines that express the current state of the physical plant.
    - Path planning that predicts and analyses the behaviour of the translational and rotational joints.
    - Workspace limitations that display the acceptable areas, limitations and identify collision detections.
    - Graphs to provide numerical data of the translational and rotational joints in runtime.

1.3. Conditions of Satisfaction

The following conditions should be met during the period of this research. First of all, a functional communication-layer between 20-sim and Unity is required. This makes it possible to obtain the results from the simulation. The obtained variables from 20-sim are used to update the components in Unity. Therefore, the speed of transferring variables should be sufficient. This is important for when, for example, the application makes use of virtual-reality, where a low amount of frames can result into motion sickness.

Based on Controllab's use case, applicable components from the modelling workflow in 20-sim will be selected and visualized in Unity. Preferably, the components have to be generic so that they can be re-used for future projects. All the components have to be designed so that they are user-friendly. This means that the components should have custom interfaces that are designed to be minimalistic and intuitive for the developers. The overview of the application should be kept clean. This means no unnecessary dialogues or duplicated components in the scene. Finally, all components will be combined as a Unity package for easy reuse by others. However, analyzing, designing, building and testing all components remains an ambitious act. Therefore, it is important to setup a planning to avoid a collection of unfinished components.

2. Theoretical Framework

This graduation will be at the boundary between two different domains. The world of Game Design that contains unlimited possibilities versus the world of Model-Based Design that should adhere to the rules of physics. Models that can predict the behaviour of certain systems in a specific domain of physics to such an extent that it gets very close to the reality are known as *Digital Twins*.

2.1. Digital Twin

The concept of the digital twin is not new. It dates back to a presentation given at the University of Michigan, given by Michael Grieves, who came up with the concept of a digital twin in 2002.[02] It refers to a digital replica of physical assets, processes and systems that can be used for various purposes.[03] *"The ultimate vision for the digital twin is to create, test and build our equipment in a virtual environment,"* says John Vickers, NASA's leading manufacturing expert and manager of NASA's National Center for Advanced Manufacturing.[04] Subsequently, it appears to be an exact predictor of system failure in the virtual environment that can be applied to the physical world. Digital twins are also referred to as the bridge between the physical and digital world. The virtual environment represents traditional physical machines with coupled measuring equipments such as pressure gauges, pressure valves and so on.

Research firm Gartner predicts a glorious future for the digital twins.[05, 06] Their group of analysts predict that by 2020 more than 21 billion connected sensors, endpoints and digital twins exist. Companies will use the digital twins for repairing and maintaining machines efficiently, plan manufacturing processes, and run entire factories effectively and efficiently to improve the development of new products.

Digital twins are perfect in combination with virtual, augmented and mixed realities. One element of the digital twin concept is already present in 3D-models created by computer-aided design (CAD). This is an accurate digital representation that can be used to ensure different parts fit together both statically and dynamically. The digital twin can then be overload on real-time data feeds from sensors in a physical operating asset which knows the exact state and condition of an operating-asset (product).[07] Similar to this concept, Controllab does not capture real-time data feeds from sensors but simulates the behaviour of the physical plant based on model-based design. This includes physics, dynamics and control engineering. The most common use case for digital twin technology today is monitoring equipment health in real-time. This is also called predictive maintenance where the aim is to predict when equipment failure might occur.[08]

Visualizing a digital twin aid the following processes of a product or machine:

1. **Build:** Operators use a digital replica to capture the asset's history, combined with intelligence, to provide unprecedented knowledge and insights.
2. **Operate:** Operators use a digital replica to capture history, combined with operating intelligence, to provide insights which predict performance.
3. **Monitor:** Digital twins offer key performance indicators (present data) and insights (future data) about an asset or system, from design and build to operation and maintenance.
4. **Training:** Operators use a digital replica, such as a training simulator, to gain skill, experience and insight of operating the physical plant.

This research is not about using digital twins to directly increase companies business and economic models such as revenue, profits, return on investment (ROI) or other cost optimizations. Instead, it researches the value of visualizing physical data used in digital twins. Visualizing a digital twin provides visual feedback that can help during debug-sessions of the model. With this visual feedback, the rate of development could be increased. Furthermore, the usage of extended reality (XR)[09] could increase the sense of reality, which makes it possible to give better verification of the plant that is visualized. Finally, visualizing a digital twin makes it easier to demonstrate the physical plant, which can be shown to others that are not directly part of a project, such that they can get an insightful overview of the results that were obtained.

2.2. Data Visualization

"Data visualization is both art and science".[10] A quote that is often used to refer to infographics that explore new ways of displaying information such as mathematics quickly and effectively. It sums up the core of this graduation assignment; the thin line between the two diverse worlds of reality and virtuality.

Data visualization is a powerful way to simplify the complexity in our data and present it in a form which is comprehensible, insightful and actionable.[11] It already started around 40.000 years ago when the first-man made two-dimensional symbols on cave walls. These are among the earliest known visual expressions of human ideas.[12] To communicate information clearly and efficiently, data visualization uses statistical graphics and plots. Numerical data may be encoded using dots, lines or bars to visually communicate a quantitative message. This helps the user analyze and reason about data and evidence. Visualization makes complex data more accessible, understandable and usable.[13]

The most common methods of visualizing data are pictorial or via graphs. Businesses can see large amounts of data in clear, cohesive ways and draw their conclusions. This is significantly faster to analyze information as opposed to looking at spreadsheets. It could be possible to create infographics in Unity. Relevant variables from 20-sim could be plotted against the time on a two-dimensional infographic.

2.3. XML-RPC

The Unity engine supports one programming and one scripting language. The programming language is called C# and supports all features in Unity. The scripting language is called UnityScript that is similar to JavaScript and is used for simplicity. Fortunately, C# is a compiled, statically typed and strongly OOP (Object-Oriented Programming) language that supports the .NET features. These features are required to interact with the .NET frameworks in-depth.[14, 15]

Sending and receiving data between 20-sim and Unity can be implemented using different network interfaces such as REST (Representational State Transfer) or RPC (Remote Procedure Call).[16, 17] Both styles primarily use a HTTP communication and support the XML-format. XML is preferred due to its simplicity, accessibility and standardization.[18] Within the world of XML, there are two main ways to implement a RPC: SOAP and XML-RPC. SOAP is an acronym for Simple Object Access Protocol that is a messaging protocol specification for exchanging structured information. The protocol uses XML as message format and consists of the following.

- A defined framework for describing what is in a message and how to process it.
- A set of encoding rules for expressing instances of application-defined data types.
- A convention for representing remote procedure calls and responses.[19]

Though SOAP is widely known and being used, it is also widely criticized for its design complexity.[20] The other network-based approach is XML-RPC. This includes a set of implementations that allow software running on different operating systems, running in different environments to make procedure calls. XML-RPC is able to use HTTP as transport and XML as encoding. It is designed to be as simple as possible, while allowing complex data structures to be transmitted, processed and returned.[21] Furthermore, XML-RPC has a simpler architecture compared to SOAP.[22]

3. Design

3.1. XML-RPC

All the data will come from 20-sim that is transferred by a communication layer to Unity. The speed of transferring data is an important aspect to update the location and orientation of the Unity components and provide real-time visual feedback. Unity sceneries can for example contain machines, robots or cranes. These objects often contain separate parts that are designed, controlled and simulated in 20-sim. Each part can contain translational and/or rotational joints. These data values need to be transferred from 20-sim to Unity. The minimum requirement is to be able to request at least 20 individual variables in a single call and update each variable 60 times per second. This should make it possible to obtain enough data from 20-sim in order to update the transformations of 3D-objects and update relevant infographics in real-time. 20-sim has a scripting API based on XML-RPC, with which you can perform certain actions within the tool. Furthermore, Controllab also developed another communication layer that was specifically designed not to interfere with the frame rate of Unity.

There are two functions available that request the value of an individual variable in 20-sim via their scripting interface. This can also be a matrix or an array. The first function is called *"GetVariables"* and is able to request variables at any time. However, it pauses the simulation of 20-sim to obtain these values from the internals of 20-sim. The other function is called *"GetMonitorValues"* and simply puts all the requested variables in a lock-free data buffer. This means that the 20-sim simulation will not block when putting the variables in this buffer. The to-be-monitored variables then have to be set prior to simulation via the *"SetMonitorValues"* function.

The other communication method is based on the FMU[23] principle. Unity counts as the server, containing a lock-free data buffer that enables clients to transfer data to. This can be beneficial since Unity can read at an extremely quick speed from this data buffer. However, 20-sim is not able to receive data back since it currently is a one-way communication.

3.2. Workflow

The visualization components will be built in Unity and arranged together as a *Unity package*. The intention is to design a generic package. This means that the package will be applicable for future projects. However, designing each component in a generic fashion will be more time-consuming than designing it for one particular use case. Furthermore, the perspective of a developer using the Unity package and the perspective of the user using the final application have to be kept in mind. The available options of each component should be limited to a certain extent, while still offering all the most important options to a user.

The Unity package that will be created can be imported into any Unity project. Custom editor scripts allow an improved version of the user-interface. This makes it easy for the developer to include the *POIInterface* of the package into the hierarchy of the Unity Editor. Most relevant aspects of the Unity package are labelled with a *POI* (point of interest) prefix. The included *POIInterface* is able to generate *POIObjects* which are the key-components for the actual user to interact with. The *POIControls* allows to change the control inputs to mouse, keyboard or touch controls. Once the user interacts with a *POIObject*, a menu component will open to display any related *POIItems* such as graphs, range indicators or other visual components. There will be a line between these *POIItems* and the corresponding *POIObject* to help the user oversee from which joint the data comes from.

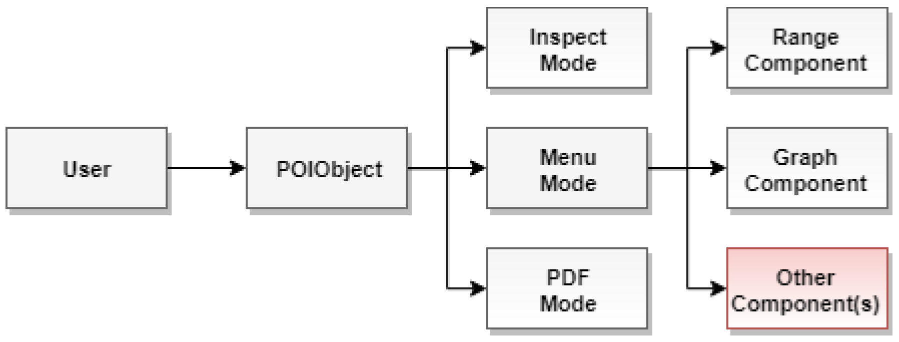


Figure 3: User perspective: Interacting with the POIObject(s).

Figure 3 shows an overview of the user's perspective to interact with a *POIObject*. Some *POIObjects* contain inspect mode. This brings all the attention to a certain 3D-model by pulling it to the foreground while blurring out the background. This allows the user to explore the details of the 3D-model. The other mode, PDF, opens external existing data such as numerical information of the 3D-model or CAD data. As final, the menu mode opens a display that provides all the available components that are related to that specific *POIObject*.

3.3. 3D-Representation

The project mentioned above contains certain 3D-models that have been created before the graduation. This was presented at the Virtual (R)evolution 2018 trade-fair in Tilburg. There is a pipeline that contains the tools to transform the CAD data to a 3D-representation. It is an algorithm that converts NURBS to a polygonal-format such as STL. However, the conversion usually creates an overhead of triangles that influences the rate of frames in Unity. Therefore, remeshing processes such as retopologizing and reduction are applied in dedicated software packages such as MeshLab and Autodesk Maya. Furthermore, software packages such as Substance Painter create Physical Based Rendering (PBR) materials as textures. Finally, the 3D-model, including texture is ready to be imported into Unity. It is necessary to maintain the original joints of the machine from the CAD data through this process due to the movement of the machine. This 3D-model might seem trivial from a gaming perspective, however it contains important parts of the 3D-structure of your plant. Examples are dimensions of the machine and locations of the joints. These are all part of the 20-sim model, and are therefore part of the modelling workflow.

3.4. Finite State Machine

"Finite State Machine (FSM) is an abstract machine that can be exactly one of a finite number of states at any given time. The change from one state to another is called a transition. An FSM is defined by a list of its states, its initial state and the conditions for each transition."[24] These are requirements for the FSM component in Unity. 20-sim uses an FSM for the project to indicate the current state of the controller. It is an important component to include to the Unity package since it provides an overview to the user of all the states, the current state and its transitions. An example of the FSM in 20-sim of the machine can be found in Appendix A.2. Building a duplicate of this in Unity should be pretty straightforward. However, the difficulty lies in making the classes generic to allow Unity to build any kind of 20-sim FSM in runtime. In order for this to succeed, the requirements mentioned above need to be communicated for Unity to initialize the FSM component:

- Unity needs to know how many states exist in the FSM in 20-sim.
- Unity needs to know all available transitions that exist between the states.
- Unity needs to know the initial and active state.

3.5. Path Planning

The movement of the machine is important for the simulation result. A 20-sim user can receive feedback by plotting relevant variables in graphs and analysing a 3D view. There are different types of path planning that give different results. Path planning using feed-forward control contains a predictive path for the machine. This information could be plotted in a graph against the actual movement of the machine to see the differences of the ideal movement versus the actual movement. Another type would be path planning using feedback control, which shows a trail of the movement of individual parts through world-space. These trails can provide information about how well the PID controller (Proportional-Integral-Derivative controller) performs its task.[25] A PID controller continuously calculates an error value as the difference between a desired setpoint and a measured point (the actual movement of the 3D-model). The path planning component will be a mixture of 3D-objects moving through virtual-space using particle system(s), trail or line-renderer components.[26-28]

3.6. Workspace Limitation

When you are working with workspaces, you should be aware of known limitations. The workspace refers to the area in which the physical plant is allowed to move in. The limitation of the workspace is composed of the boundaries of this area and is known as the safety-layer component in 20-sim. The safety-layer keeps track of each individual part of the model. In case any part of the physical plant exceeds the limitation of the workspace, the safety-layer activates and disables the engines of the physical plant. This should avoid the physical plant from endangering any surrounding entities or itself, as long as the margins of the limitations are set up properly. Colliding with other objects could be detected (collision detection) and could be an interesting component to visualize in Unity to increase visual feedback. This could help for example a trainee that is being trained in a training simulator to maintain the controls of a machine.

There are also other forms of feedback that unfortunately will not be covered during this research such as audio feedback or haptic feedback, often referred to as simply "haptics". When referring to mobile phones and similar devices, this generally means the use of vibrations from the device's vibration alarm to denote that a touchscreen button has been pressed.[29]

3.7. Infographics

There are many types of infographics that can be used to plot data, ranging from 2D bar charts to 3D pie charts. For this research, one infographic style was chosen. Others can be done in a similar fashion. The X-axis will be used to plot the simulation time of the 20-sim model and the Y-axis will be used to plot numerical data. This enables the user to visually see the requested variables such as the change in translations or rotations versus the simulated time. In addition, other relevant variables that are available in 20-sim such as speed, force or torques could be selected and visualized.

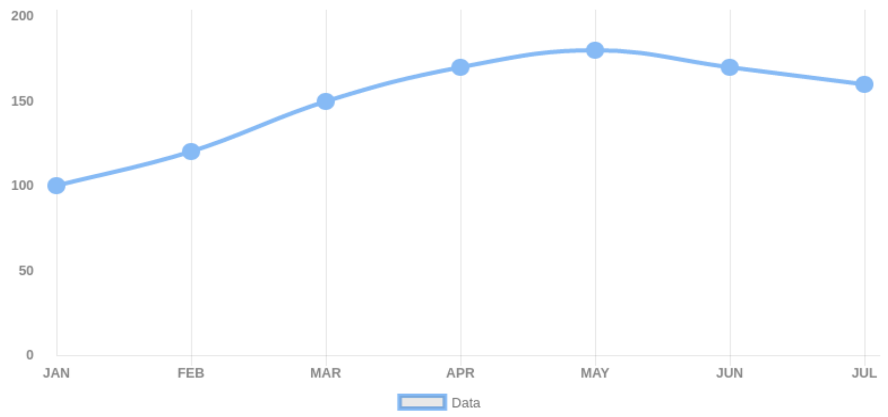


Figure 4: Example of a 2D chart that uses months as time variable (x-axis).

4. Implementation & Results

4.1. XML-RPC

Available clients can communicate via XML-RPC with the 20-sim scripting API are Octave and Python.[30, 31] A new XML-RPC wrapper has been created for C#, because Unity uses C# as its main language. First of all, an existing XML-RPC library called *"CookComputing.dll"* was found.[32] Subsequently, based on the Python client, most functions of 20-sim have been rewritten for C#.

In order to visualize the digital twin from 20-sim, a communication-layer needs to be built that sends data towards the Unity engine. It is a *request-response* method in which Unity will be requesting data from 20-sim. This is referred as a Client-Server Model.[33]

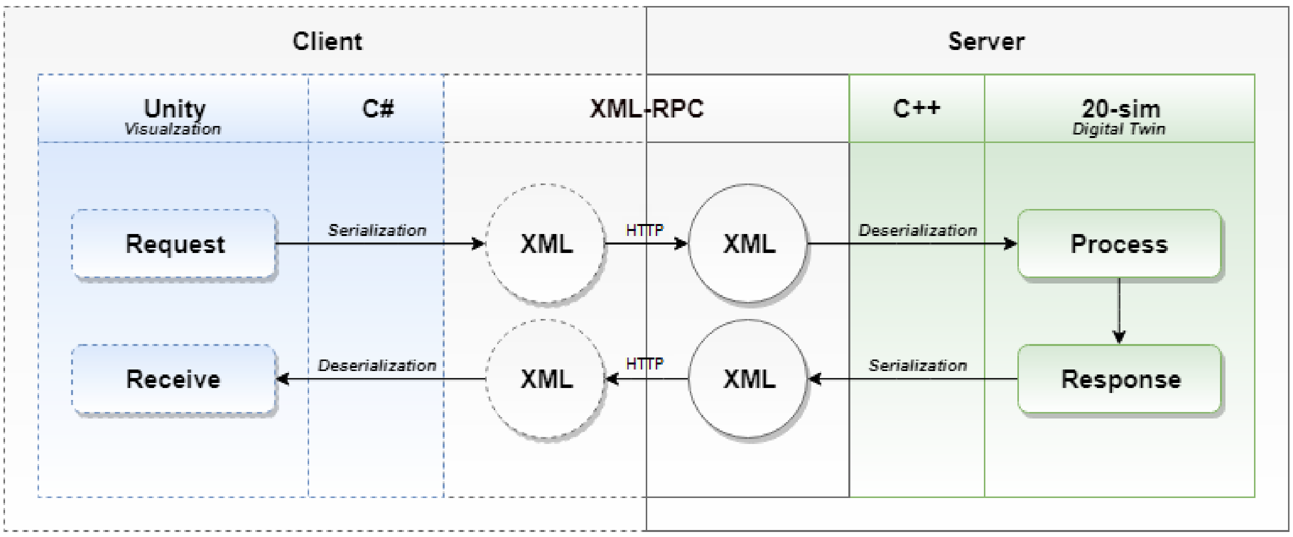


Figure 5: Client-Server Model: Server (20-sim) Client (Unity).

Figure 5 shows the implementation of the XML-RPC wrapper. These are the processes of requesting and receiving data between a client and server via the XML-RPC protocol. Unity sends a requests of certain variables from 20-sim in the form of XML over an HTTP protocol. A TCP protocol that is able to transfer data in binary format would be preferred. This would increase the rate of transferring data a lot. However, the current C# XML-RPC library does not support this functionality, because this is an extension designed by Controllab, which is not part of the XML-RPC protocol itself.

Unity has vSync enabled by default. *"Sceneries can have various update cycles that do not update continuously but at regular intervals. Without synchronised updates, it is possible for unity to issue a new frame while the display is still rendering the previous one. This will result in a visual artefact called "tearing" at the position onscreen where the frame change occurs."*[34] However, we will not be using any visual effects and purely benchmark the speed of transferring variables. Therefore, vSync will be turned off.

As explained in Chapter 3.1, there are two available functions within 20-sim to receive the data from variables. Both functions have been tested in-depth. C# requested 1000 times the amount of 1, 5, 10, 20, 50 or 100 variable(s) from 20-sim and successfully received it. The time of the actual request and response has been measured and can be found in the *'Time'* column. The '%' column is based on the *'Calls per Second'* column. The requirement is 60 Hertz which is equal to 100%. The Table below shows the results.

Table 1: Benchmark Results: XML-RPC Calls

Method	Amount of Variables	Time (sec)	Calls per Second (Hz)	% (60 calls = 100%)
GetVariables	*1*	*19.347*	*52*	*86.66*
	5	*21.159*	*47*	*78.33*
	10	*39.569*	*25*	*41.66*
	20	***42.015***	***24***	***40.00***
	50	*71.128*	*14*	*23.33*
	100	*137.677*	*7*	*11.66*
GetMonitorValues	*1*	*8.051*	*124*	*206.66*
	5	*8.610*	*116*	*193.33*
	10	*10.608*	*94*	*156.66*
	20	***13.541***	***74***	***123.33***
	50	*21.009*	*48*	*80.00*
	100	*38.436*	*26*	*43.33*

A graphical overview of the results can be found in Appendix A.1.

The *GetVariables* function is not applicable. A single requested variable can only be updated 52 times per second. This is below the requirement of 60 times per second. The reason of the minor delay is due to the function pausing the simulation of 20-sim before transferring the requested data. In comparison, the *GetMonitorValues* shows a better result. It updates a single variable 124 times per second and is able to update 20 variables at 74 times per second. It is important to remember that the requested variables needs to be set in 20-sim before being requested. This can be done with the *SetMonitorValues* function.

However, after testing the communication-layer in a project that is filled with assets, the results were deficient. This is because purely the speed of transferring data (in Hertz) was tested without the render-loop from Unity. Even the *GetMonitorValues* function does not succeed to update the variables 60 times per second. The transferring speed is interrupted by the visual aspects of Unity's render-loop. These are aspects such as vertex processing, memory bandwidth, batching, fill rates, etc. However, in order to maintain an appropriate transferring speed, the FMU package of Controllab has been used. This is, as explained in Chapter 3.1, a one-way communication between 20-sim and Unity. The FMU package encodes the XML-code in binary-format before transferring the data. Therefore, Unity receives the data much quicker in comparison to the XML-RPC layer.

4.2. Finite State Machine

How to built a finite state machine in Unity quick and efficient? At first, the state-prefabs were made out of small 2D-sprites that contained a few attributes such as a transition entry (to draw a line between states) and the name of the state. However, this became a lot of manual work to build a complex finite state machine. Also, the Unity hierarchy became chaotic due to being overloaded with GameObjects.[35] This was clearly not the most efficient way to create finite state machines and a different approach was necessary.

Unity has its own state machine built in their engine. The module, also referred to as Animator[36], is originally designed to handle key frame-cycles for animations. "*If you want to use State Machines without animation, look into State Machine Behaviours on empty states. While State Machines were designed with animation in mind, they extend surprisingly well to general state machines.*" says Catherine Proulx.[37] Most programmers like control and flexibility and would create their own state machines. The animator seems like a "black box" because you don't really know what it does in the background and it can take time to transition from one state to another.

However, for designers, it can be a handy tool to achieve your goal in a hacky way. The animator makes it possible to setup a multi-layered state machine including many states and transitions within minutes. The only problem remains, it is all created in the edit mode of the Unity engine. The data needs to be obtained from the state machine and used to convert it into *GameObjects* to world-space.

Lucky enough, Unity is an open-source engine that support a wide API with a lot of documentation. The file in which the state machine is being built is called an *animation controller*. This is a Unity standard component which is C# supported. Therefore, it was possible to obtain all the data from the states, such as the positions in 2D-space, names and transitions. Subsequently, the obtained data is stored into member variables and is used to instantiate prefabs in runtime. This makes it possible to build a complete functional finite state machine with just one button.

The multi-layered Finite State Machine is instantiated before runtime. Multi-layered means that a state can have sub-states. These states will be instantiated behind the main-state. In case a sub-state is active, it will also lit up the main-state at top-level. Each transition between two states contains an arrow that indicates the direction of the next state.

4.3. 3D-Representation

The Unity Package that includes components to visualize data has been used in two separate projects. Figure 6 shows an overview of the interaction methods being used in two different projects. The left side contains the pick and place machine and uses the POIObjects (spheres) located at the joints of the machine. The user can interact with the spheres to open relevant components. However, the spheres can take up a lot of unnecessary screen-space and may be hard to interact with if they are connected to moving joints. Therefore, the newest project, displayed on the right side, uses a different approach for highlighting the POIObjects. It uses an outline shader that only displays when the user hovers its mouse over a certain interactable 3D-object. Removing all previous style of POIObjects remains a clean overview at all times.

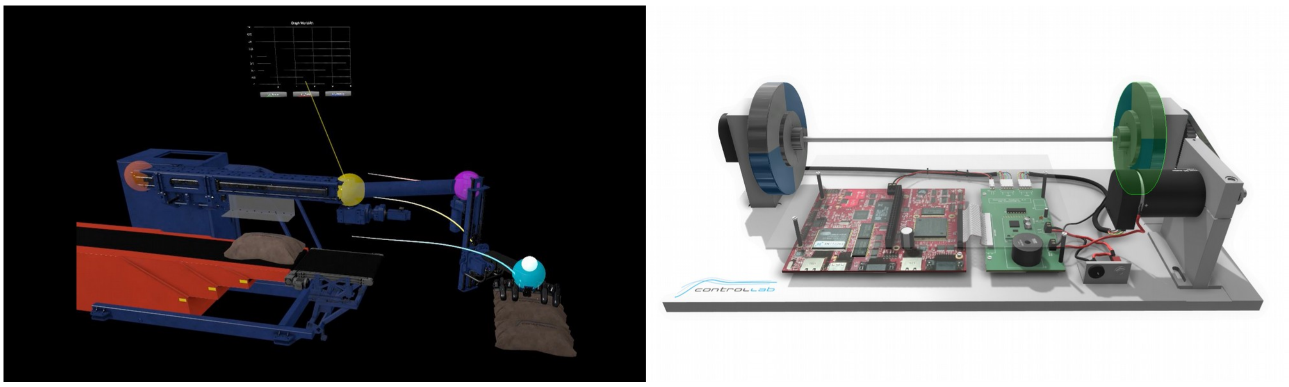


Figure 6: Different interaction methods: POIObjects vs Outline Shaders.

4.4. Workspace Limitation

With 20-sim processing the complex physics, Unity has to support the visualization of these components, but also of its physical boundaries. This could be as simple as communicating a single boolean to Unity that indicates if the individual part of the model is outside the defined boundary or not. The visualization should indicate clearly when the safety-layer is activated and whether collision has been made. The Unity package should include a component that is able to visualize a specific part of a 3D-model.

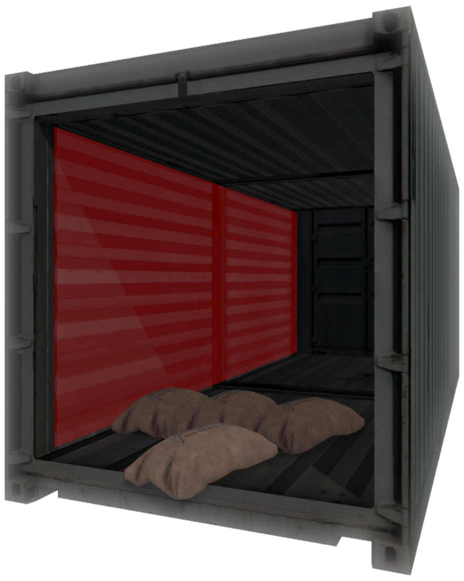


Figure 7: Safety-layer displaying the collided area in Unity.

Figure 7 shows an example of the left side of the container side being lit up. Signal processing and signal interpretation is a big topic on its own. In brief, the component should be able to flash 3D-models in Unity at a certain speed (Hz) and it should use certain color(s).

First of all, flashing signals. These are flashing beacons that are created with strobe lights and often indicate warnings. The speed of most strobe lights are factory-limited to about 10-12 Hz (10-12 flashes per second) in their internal oscillators. Studies have shown that people that are susceptible to the flashing effects can have symptoms such as epilepsy. *"An infamous event took place in 1997 in Japan when an episode of the Pokémon featured a scene that depicted a huge explosion using flashing red and blue lights, causing about 685 of the viewing children to be sent to hospitals."*[38] The flashing explosion was bright, including multiple colors with a flashing effect at about 12 Hz. The average strobe effect of flashing beacons at schools, hospitals or stadiums flash at a rate of 1 Hz. Therefore, the same rate is being used for the detection of collision.

"Color is a great way to impart vitality, provide visual continuity, communicate status information, give feedback in response to user actions, and help people visualize data."[39] As for signal processing and interpretation, coloring your objects is also a big topic on its own. Due to the scope of this research it should be sufficient, to use simple, common and understandable colors were used throughout this research for the components.

Another form of visualizing the safety-layer can be established with a range indicator. The translational and rotational limitations are also forms of workspace limitations. Each individual joint of the physical plant contains limitations that could be visualized in Unity. The radial indicator shows the acceptable angle of a rotational joint. This means that the actual indication of the rotation should be within a certain minimum and maximum range. A complete rotation contains 360° degrees. The Unity component contains an area-prefab that is a 2D-sprite of 1° degree. In case the range between the minimum and the maximum angle is 120° degrees, the area-prefab will be instantiated 120 times, with each iteration rotating 1° degree on the local Z-axis. This way, the radial indicator can be dynamically built in Unity's runtime. Subsequently, the minimum and maximum values are printed on each corner, visualizing a generic radial indicator that supports all possible angles.

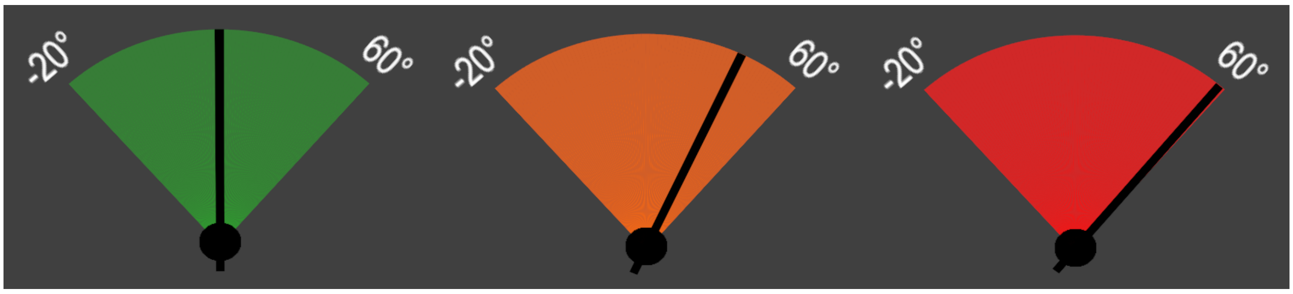
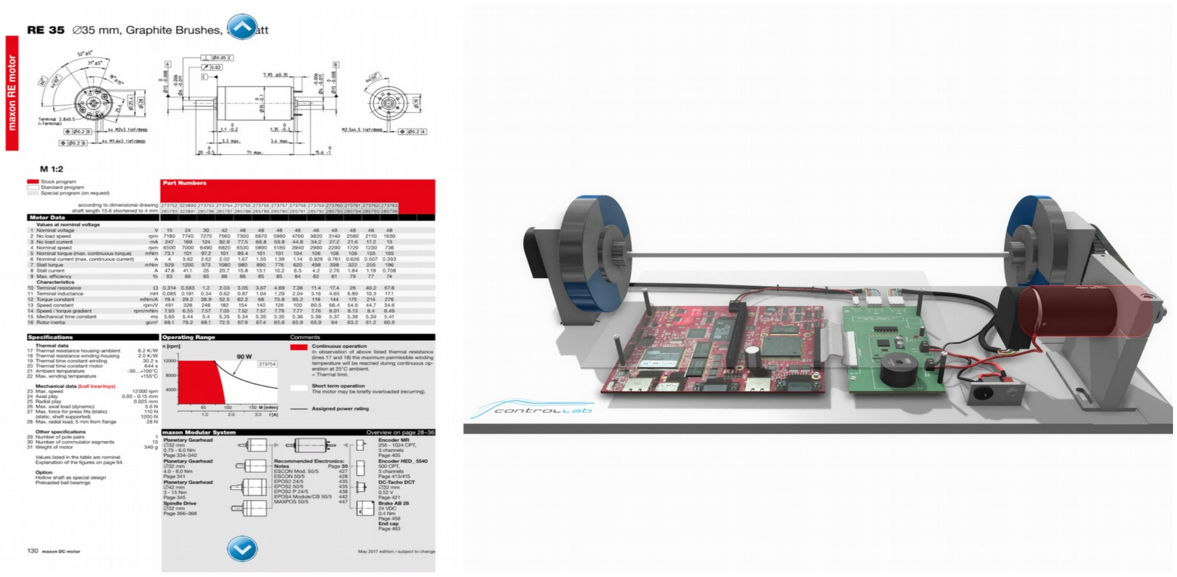


Figure 8: Radial Indicator shifting color based on the indicator value.

Figure 8 shows the three stages of the radial indicator with an angle from -20° to 60° degrees. The option to apply warning and error zones is supported. The developer is able to provide an angle to the warning zone (orange) and an angle to the error zone (red). If the actual indicator comes in range of those angles, the color of the range indicator changes. However, the current value of the indicator is still missing and needs to be supported before the release of the Unity package.

4.5. PDF-Reader

Many of the 3D-models are known objects that may contain existing data that can be red from PDF-files. This can be any kind of information such as CAD- or other general data written on PDF-format. The PDF-module is placed at the left side of the screen divided by a 3rd of the resolution's width ratio. For example, running the application at a resolution of 1920×1080 pixels (FullHD) creates a PDF-module of 640×1080 pixels. This does not cost a lot of screen-space and it allows the PDF-module to render the font sharp for the user to read clearly.


RE 35 ∅35 mm, Graphite Brushes, att

maxon RE motor

M 1:2

Part Numbers

Motor Data

Specifications

Operating Range

90 W

maxon Modular System

130 maxon DC motor

Figure 9: Example of the PDF-module opening a datasheet.

Figure 9 shows the result of the PDF-module that opens when a user interacts with a highlighted object in the scene, explained in Chapter 4.3. PDF-files containing multiple pages are allowed thanks to the supported navigation arrows displayed at the top and bottom of the PDF-module. When the module is closed, the 3D-view goes back to its original state, displaying at fullscreen. This remains a clean view for the user at all times.

4.6. Infographics

The graph component is useful to plot numerical data from 20-sim. The chart scrolls data horizontally, only showing the last 5 seconds of the simulation. The X-axis shows the simulation time while the Y-axis shows any selected 20-sim variable. Each variable is plotted 25 times per second (25Hz). Therefore, the graph can only support a maximum of 3 variables to remain above 60 frames per second. Plotting over 3 variables in a single graph decreases the rate of frames in Unity heavily. The developer has the option to initialize the chart with 3 variables from 20-sim. The user has the option to choose which variables need to be toggled at runtime. Each variable has its own category and color to separate itself from the other variables. The component supports more options for the developer such as to instantiate the graph-object in local or world-space, have the graph-object look at the camera or have the graph-object fixed at a certain position and rotation.

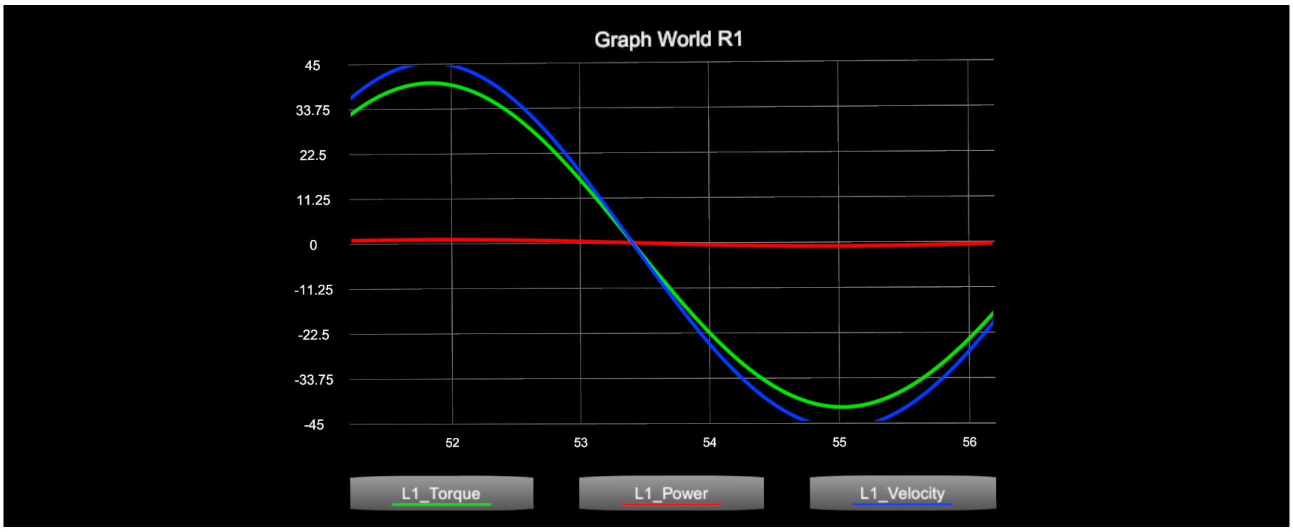


Figure 10: Graph component showing three variables at simulation time.

Figure 10 shows the current state of the graph component. Similar to the range indicator, this component also misses the current values of the categories. This makes it difficult for the user to analyse the actual values. One more remaining issue is the range of the Y-axis. Once a very high or low value is being plotted, the Y-axis ranges automatically to that value. However, it does not automatically range back to the previous highest value once that specific value has disappeared from the chart.

5. Discussion

5.1. XML-RPC

The speed of transferring data in the current XML-RPC communication-layer was too slow to use it in the render-loop of Unity. Therefore, the FMU package, that has a communication encoded in binary-format instead of XML-format, has been used. It provides the demanding speed that is required to update the components in Unity in real-time without harming the frame-rate of the application. However, the trade-off is that the FMU package has no communication back towards 20-sim. This means that the FMU package is, in contrast to the XML-RPC communication, not able to change parameters in 20-sim in runtime. This could, for example, be beneficial for tuning parameters or variables of the 20-sim model in runtime.

5.2. Unity Package

It was a good decision to combine all created components as a Unity package. From the start, user-friendliness was kept in mind, for both developers of a Unity demo and end-users of that demo. The Unity package makes it more intuitive for developers to choose applicable components to add to their project. Making the interface more intuitive was an iterative and incremental process. Unity allows custom interfaces inside Unity's editor environment.[40] The custom interface became more efficient after removing all the unnecessary actions that were required to implement and modify components.

Besides user-friendliness and building an efficient interface, it was also important to create a minimalistic interaction with the components. *"A minimal, simple design can have maximum impact visually and can improve the user experience"*.[41] The first method of interaction between the user and the components contained basic 3D-meshes, so called primitives. Eventually, many primitives and related components were present in world-space and the overview became very disorganized. Therefore, the method of interaction was replaced by an outline shader, only highlighting the outline of the actual 3D-mesh when the user hovers its mouse over it. This minimalistic approach has also been applied to other components. For example, instead of multiple graphs in world-space that each display their own data, is it changed to one graph that can display any data depending on what the user selected last. Furthermore, all components can be removed by the user. These decisions were made with the idea of maintaining a clean overview at all times.

Most classes and functions that are created during this research were designed to be generic. This means to me that the Unity Package contains components that can be used in future projects without the user having to adjust source code. This is a time-consuming process for a programmer that is inexperienced in object-oriented programming (OOP). One thing that I would do different the next time, is to start with a class diagram in the unified modeling language (UML). This would help to think about the structure of the code before starting to write it. Some of the problems, like several iterations of user-friendliness improvements or making the code generic, might have been avoided by doing so.

5.3. Components

It was an ambitious task to design and build this amount of components in Unity. It might be that focussing on a few of them might have given me components that were more user-friendly and stable. Nevertheless, seeing the overall results of the final components is satisfying. In particular the graph component is suitable for any kind of project. There is quite often a desire to plot numerical data in real-time. However, the component still contains a memory leak. It saves all data that has ever been plotted. Therefore, the memory buffer increases which eventually causes the frame-rate to drop. This issue has been analysed and debugged, but unfortunately not solved due to lack of time.

Building a finite state machine from scratch by using the Animator in the Unity Editor is a huge time-saver. With the single press of a button, the state machine gets converted to GameObjects into the hierarchy of the Unity scene. However, the state machine should not have been made in 3D, such that the layers of the state machine were visible at once. This made it quite chaotic and therefore difficult to identify which sub-state is active. It would be more intuitive to show one layer of the state machine at a time.

The radial indicator builds itself in run-time and is completely generic. It is a decent component for providing data in real-time to the user. Its simplicity makes the component self-evident. This makes it easier to examine the visualized data and is considered to be a strong benefit. However, based on feedback, the option to see the actual, current value of the radial indicator is missing.

The PDF-module is useful to include external data to the project such as CAD data or other informatics. The first iteration used a resolution that was too small to be able to read the font of the PDF-sheet. Subsequently during prototyping, I found out to use ⅓ of the screen to display the module. This allows to render the font sharp for the user to read clearly.

6. Conclusion

The digital twin is a digital replica of a physical plant and can be beneficial for building, maintaining, operating and training. In combination with simulation software such as 20-sim, flaws in the control systems can easily be detected and improved. Visualizations give us an abstract overview of complex data and helps us to understand the data from a different perspective. It makes it possible, for example, to see within a second if an object is rotating in a positive or negative direction, without having to debug through many lines of code. This increases the easiness of development. Besides improving debug-sessions, visualizing 20-sim components (the modelling workflow) will also help Controllab to explain and potentially sell their services.

Before all else, the limitations of the 3D animation features of 20-sim have been analysed and the gaming engine Unity has been used to overcome these limitations. The transferring speed of the communication-layer between the engines was critical. The XML-RPC communication-layer that has been created sends over plain text instead of a version that is binary-encoded. The latter is a lot faster. This affects Unity's frames-per-second and is unacceptable for applications that, for example, make use of virtual-reality, where a low amount of frames can result into motion sickness.

Furthermore, the pick and place use case contained a modelling workflow that contained many controller aspects that could be shown in a virtual representation of the physical system. Each selected component from this use case has its own set of classes that are mostly generic. This is a very time-consuming job for an inexperienced programmer that especially lacks the knowledge about how to structure code. The collection of components are designed to be user-friendly and combined as a Unity package. This includes custom intuitive interfaces for easy usage for the developer that implements the components.

Maintaining a clean overview of the visualization is essential. Therefore, all components take this into account. For example, all components are designed to be opened and closed. Specific components such as the graph components are designed to plot multiple data sources into a single chart instead of opening a chart for each data source. Furthermore, methods such as the interaction between the user and the components are also improved. For example, the outline shader that highlights the actual 3D objects helps to maintain a clean overview at all times.

"How to visualize the modelling workflow of a digital twin?". Based on all sub-questions that are answered above, it is possible to visualize a digital twin. Throughout this research, two examples of such visualizations have been shown. It showed that you should look at the project at hand to determine which aspects of the digital twin are worth visualizing.

7. Recommendations

One interesting question came up during feedback throughout this research: *"Could data visualization be used for another purpose than explaining the modelling workflow to others?"*. It is possible to modify the digital twin throughout the visualization in real-time. This is called run-time tuning.

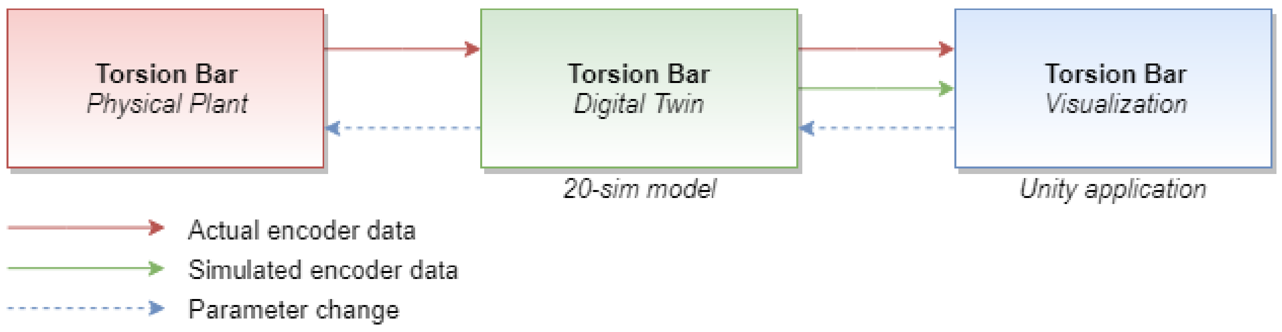


Figure 11: Run-time tuning.

Figure 11 shows a similar setup with the Torsion Bar that has been used throughout this research. A representation of the Torsion bar can be found in Figure 9. The setup contains an engine that visualizes a digital twin of a physical plant. Both the physical plant and the digital twin use the same control system. This means that their behaviour should be identical. The physical plant sends its data of the angles (encoder data) towards the digital twin. The 20-sim model redirects this encoder data towards the visualization, including its own simulated encoder data. In the visualization application, the data can be plotted using the graph component. Furthermore, in this particular case, the user is able to change certain parameters in the graph component that will affect the simulated digital twin. The physical plant remains untouched and continues to plot its original values to the graph component. However, the digital twin's control system has been changed and will plot diverse values. This allows the user to easily observe the changes of the simulated version, and how these parameters would impact the physical plant. These run-time changes, for example with virtual-reality, could be really immersive and time-efficient for the user.

Chapter 3.6 shortly discussed other existing forms of feedback such as haptics or audio feedback. Not every aspect of a digital twin's representation has to be visualized. In fact, supporting audio and haptics could be more immersive for the user in some cases. Interfacing is another aspect that could be improved. It could be beneficial to have an interface that is accessible with virtual-reality. For example, an interface that supports the workspace limitations such as collision detection and path planning.

8. References

[01] 20-sim. (2014, September 23). Retrieved May 15, 2018, from https://en.wikipedia.org/wiki/20-sim

[02] Grieves, M., & Vickers, J. (2016). Origins of the Digital Twin Concept.

[03] Digital twin. (2018, March 27). Retrieved May 15, 2018, from https://en.wikipedia.org/wiki/Digital_twin

[04] Marr, B. (2017, March 06). What Is Digital Twin Technology – And Why Is It So Important? Retrieved May 15, 2018, from https://www.forbes.com/sites/bernardmarr/2017/03/06/what-is-digital-twin-technology-and-why-is-it-so-important/#480a1cbf2e2a

[05] Pettey, C. (2017, October 18). Prepare for the Impact of Digital Twins. Retrieved May 15, 2018, from https://www.gartner.com/smarterwithgartner/prepare-for-the-impact-of-digital-twins/

[06] Digital twins: Investeer vooral in simpele modellen. (2017, June 21). Retrieved May 15, 2018, from https://itexecutive.nl/technologie/digital-twins-investeer-vooral-simpele-modellen/

[07] Industry 4.0, Digital Twin and IoT. (n.d.). Retrieved May 15, 2018, from http://www.theorem.com/Digital-Realities/Using-Augmented-or-Mixed-Reality-enables-you-to-visualize-a-digital-twin.htm

[08] Predictive Maintenance: What is PdM? (n.d.). Retrieved May 15, 2018, from https://www.fiixsoftware.com/maintenance-strategies/predictive-maintenance/

[09] Irvine, K. (2017, November 17). XR: VR, AR, MR-What's the Difference? | Viget. Retrieved May 15, 2018, from https://www.viget.com/articles/xr-vr-ar-mr-whats-the-difference/

[10] Manuela Aparicio and Carlos J. Costa (November 2014). Data visualization. *Communication Design Quarterly Review. 3*

[11] Guerrero, J. (2015). Visual storytelling and data visualization in numerical simulations. Retrieved May 15, 2018, from https://www.researchgate.net/publication/286624924_Visual_storytelling_and_data_visualization_in_numerical_simulations

[12] Schöpke, A. (n.d.). Data Visualization – Art or Science? Retrieved May 15, 2018, from http://ib5k.com/blog/posts/data-visualization-art-or-science

[13] Data visualization. (2018, May 12). Retrieved May 15, 2018, from https://en.wikipedia.org/wiki/Data_visualization

[14] Hemmendinger, D. (2018, May 03). Object-oriented programming. Retrieved May 15, 2018 from https://www.britannica.com/technology/object-oriented-programming

[15] Glenn, W. (2016, May 10). What Is the Microsoft .NET Framework. Retrieved May 15, 2018, from https://www.howtogeek.com/253588/what-is-the-microsoft-net-framework-and-why-is-it-installed-on-my-pc/

[16] What is REST? (n.d.). Retrieved May 15, 2018, from https://www.codecademy.com/articles/what-is-rest

[17] Remote procedure call. (2018, May 12). Retrieved May 15, 2018, from https://en.wikipedia.org/wiki/Remote_procedure_call

[18] Ferrara, D. (n.d.). Here Is a List of 5 Good Reasons to Use XML on Your Website. Retrieved May 15, 2018, from https://www.lifewire.com/reasons-to-use-xml-3471386

[19] SOAP. (2018, May 07). Retrieved May 15, 2018, from https://en.wikipedia.org/wiki/SOAP

[20] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielson, H. F., . . . Winer, D. (2000). Simple Object Access Protocol (SOAP). Retrieved May 15, 2018, from https://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[21] What is XML-RPC? (n.d.). Retrieved May 15, 2018, from http://xmlrpc.scripting.com/

[22] Ivak, M. (n.d.). REST vs XML-RPC vs SOAP – pros and cons. Retrieved May 15, 2018, from https://maxivak.com/rest-vs-xml-rpc-vs-soap/

[23] Functional Mock-up Interface (2018, June 10). Retrieved June 15, 2018, from https://en.wikipedia.org/wiki/Functional_Mock-up_Interface

[24] Finite-state machine. (2018, July 31). Retrieved June 15, 2018, from https://en.wikipedia.org/wiki/Finite-state_machine

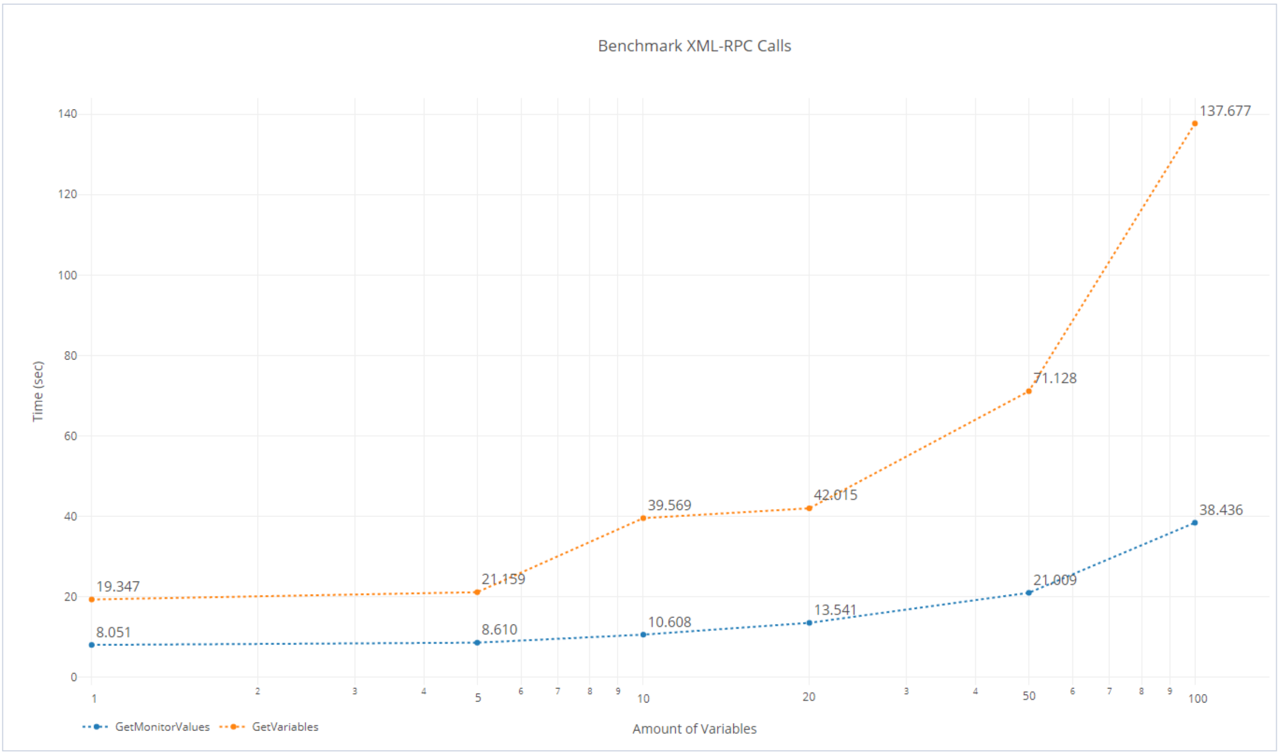
[25] PID controller. (2018, August 16). Retrieved June 15, 2018, from https://en.wikipedia.org/wiki/PID_controller

[26] Technologies, U. (n.d.). ParticleSystem. Retrieved June 15, 2018, from https://docs.unity3d.com/ScriptReference/ParticleSystem.html

[27] Technologies, U. (n.d.). Trail Renderer. Retrieved June 15, 2018, from https://docs.unity3d.com/Manual/class-TrailRenderer.html
[28] Technologies, U. (n.d.). Line Renderer. Retrieved June 15, 2018 from https://docs.unity3d.com/Manual/class-LineRenderer.html
[29] Introduction to Haptic Feedback. (n.d.). Retrieved June 15, 2018 from https://www.precisionmicrodrives.com/haptic-feedback/introduction-to-haptic-feedback/
[30] GNU Octave (n.d.). Retrieved June 15, 2018, from https://www.gnu.org/software/octave/
[31] Python (n.d.). Retrieved June 15, 2018, from https://www.python.org/
[32] Cook Computing (n.d.). Retrieved June 15, 2018, from http://www.cookcomputing.com/blog/index.html
[33] What is the Client-Server Model? - Definition from Techopedia. (n.d.). Retrieved May 15, 2018, from https://www.techopedia.com/definition/18321/client-server-model
[34] Technologies, U (n.d.). Quality Settings. Retrieved June 15, 2018, from https://docs.unity3d.com/Manual/class-QualitySettings.html
[35] Technologies, U. (n.d.). GameObject. Retrieved June 15, 2018 from https://docs.unity3d.com/560/Documentation/Manual/class-GameObject.html
[36] Technologies, U. (n.d.). The Animator Controller Asset. Retrieved June 15, 2018 from https://docs.unity3d.com/Manual/Animator.html
[37] State Machine Transition interruptions – Unity Blog. (n.d.). Retrieved June 15, 2018 from https://blogs.unity3d.com/2016/07/13/wait-ive-changed-my-mind-state-machine-transition-interruptions/
[38] Strobe light. (2018, August 16). Retrieved June 15, 2018, from https://en.wikipedia.org/wiki/Strobe_light
[39] Apple Inc. (n.d.). Color – Visual Design – iOS – Human Interface Guidelines. Retrieved June 15, 2018 from https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/color/
[40] Technologies, U. (n.d.). MonoBehaviour.runInEditMode. Retrieved June 15, 2018 from https://docs.unity3d.com/ScriptReference/MonoBehaviour-runInEditMode.html
[41] Minimum UI for Maximum UX. (n.d.). Retrieved June 15, 2018, from https://blog.prototypr.io/minimum-ui-for-maximum-ux-b3495f669314
[42] Redmine. (n.d.). Retrieved May 15, 2018, from https://www.redmine.org/

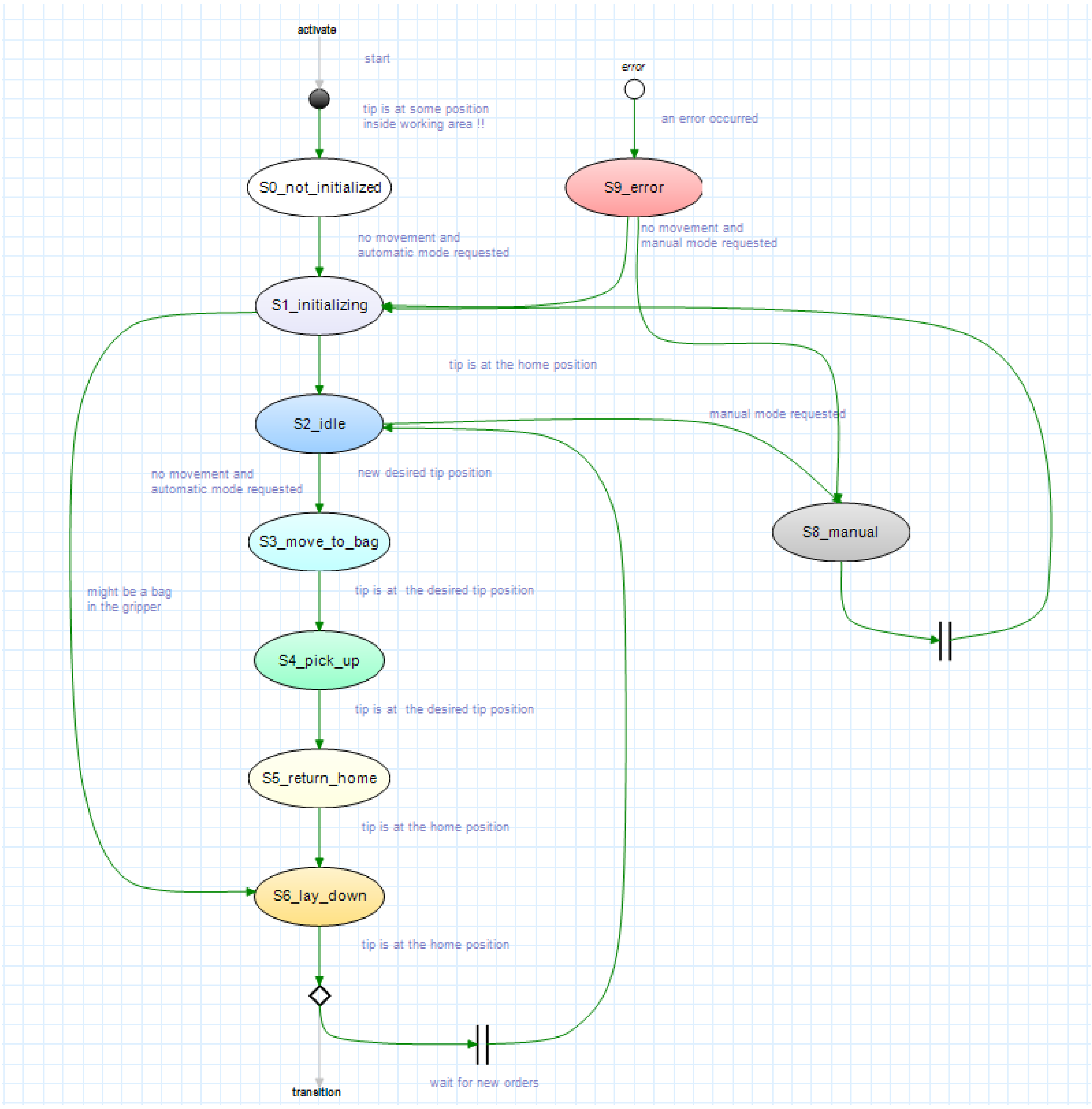
A . Appendices

A.1 Benchmark XML-RPC Result



The X-axis (horizontal) shows the amount of variables being requested from Unity and transferred from 20-sim. The amounts 1, 5, 10, 20, 50 and 100 have each been tested separately with the length of 1000 calls. The Y-axis (vertical) shows the time in seconds needed to make 1000 calls for the amount of variables at the X-axis. The timestamps are also displayed at the points in the graph. The *GetMonitorValues* function clearly transfers data quicker than the *GetVariables* function.

A.2 Finite State Machine of the 20-sim model


activate
start
tip is at some position inside working area !!
error
an error occurred
S0_not_initialized
S9_error
no movement and automatic mode requested
no movement and manual mode requested
S1_initializing
tip is at the home position
S2_idle
manual mode requested
no movement and automatic mode requested
new desired tip position
S3_move_to_bag
S8_manual
might be a bag in the gripper
tip is at the desired tip position
S4_pick_up
tip is at the desired tip position
S5_return_home
tip is at the home position
S6_lay_down
tip is at the home position
transition
wait for new orders