Simulation and Visualization of the Dutch High Voltage Power Grid

BSc Thesis

Rik Bieling 4955854
Joran Kroese 5292190

Date:
14 / 06 / 2024

DELFT UNIVERSITY OF TECHNOLOGY

FACULTY OF ELECTRICAL ENGINEERING, MATHEMATICS AND COMPUTER SCIENCE

ELECTRICAL ENGINEERING PROGRAMME

Abstract

This report explains the processes behind the design, construction and testing of a high-voltage power grid simulator and visualizer. For this, a system based on data found in a paper was created. This paper describes an aggregated grid for the Dutch high-voltage power grid in 2018.

The main use case of this system would be for educative purposes, where interested students and potential students can come, play and learn at the same time while making interactive decisions and seeing how their actions are influencing the system as a whole.

The subsystem in this report is tasked with the integration of the data from the forecasting & scenario module and the inputs from the hardware module, furthermore it calculates the power flow using a Gauss-Seidel algorithm and visualizes its results alongside the forecasting in Godot.

Preface

This thesis is written for the Bachelor Graduation Project of the Bachelor electrical engineering of the TU Delft. The project was initiated with the goal of taking the first steps in designing an interactive power grid simulation and visualization system that eventually can be placed in the *Digital Energy Studio*, which is, as of writing, still under construction and will be a novel maker space.

We would like to express our gratitude to our supervisor, Simon Tindemans, as well as Kutay Bölat and the rest of the team who helped us throughout the many meetings. Furthermore, we would like to thank Remko Koornneef and Martin Schumacher for allowing us to work in the EEE project room and opening it for us every morning. Finally, we would like to thank our colleagues, Johannes Ketelaars, Michiel De Rop, Bodhi van Dam and Björn Bucksch, for an enjoyable and constructive collaboration and the little breakfast club we made this quarter.

Rik Bieling & Joran Kroese
Delft, 2024

Contents

1 Introduction **1**

2 Program of Requirements **3**
2.1 Global Mandatory Requirements . . . 3
2.2 Subgroup Mandatory Requirements . . . 3
2.3 Subgroup Trade-off Requirements . . . 4

3 Deliberations **5**
3.1 Game engine & Coding language . . . 5
3.2 Power Flow in C++ . . . 5
3.3 Scenario Decisions . . . 6
3.4 Communication with hardware module . . . 6

4 Internal Communication **8**
4.1 Hardware Module . . . 8
4.2 Scenario & Forecasting Module . . . 10

5 Analysis of existing visualizers **12**

6 Simulation **15**
6.1 C++ Implementation . . . 15
6.1.1 Register Types . . . 16
6.1.2 Mapgenerator . . . 16
6.1.3 Tileset . . . 16
6.1.4 Powerline . . . 17
6.1.5 Loadflow . . . 17
6.1.6 Timebase . . . 20
6.1.7 Canvas Control . . . 21
6.1.8 Pop-up Box . . . 21
6.1.9 Collision Box . . . 22
6.2 Godot Project . . . 22

7 Visualization **24**
7.1 Visualisation design choices . . . 24
7.1.1 Color usage . . . 24
7.1.2 Iconography . . . 25
7.1.3 Legends . . . 25
7.1.4 Maps . . . 25
7.1.5 Node placement . . . 25

8 Conclusions **27**

9 Discussions & Suggestions for future groups **28**
9.1 Tips and retaining issues 28
9.2 Suggestions for progress 29

Appendices 29

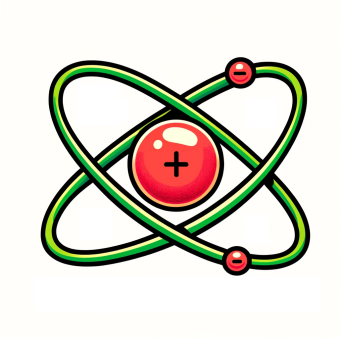
A Software installation **30**
A.1 Software installation 30
A.1.1 Prerequisites 30
A.1.2 Project file structure 32
A.1.3 Installation 33
A.1.4 Adding c++ code 33

B Godot workings and vocabulary **35**

Chapter 1

Introduction

As is explicitly stated throughout the scientific world, it is important to visualize concepts in order for people to understand and remember them better.[1] Therefore, visualization plays a crucial role in the transmission of knowledge to people who want to learn. For this, many systems for differing concepts exist, a couple of examples of these are the classical image of an atom via the Rutherford-Bohr model, as seen in figure 1.1a, and the plethora of different maps of the Earth, helping students understand topics such as topography, geography, politics and demographics. Figure 1.1b, for example, shows the population density of the world per country.



(a) Rutherford-Bohr model

(b) The world map scaled to population per country[2]

Figure 1.1: Two examples that show the importance of visualization in education

It is images like these that, while they are simplified versions of the concepts they represent, help construct a better image of the real thing in the head of the learner. Thus, showing that visualization is a great tool for conveying complex systems to students. The challenge for this project was then to try and portray the incredibly complex and delicate nature of a power grid to a person who is playing with the simulation.[3]

To achieve this, the project was divided into three subgroups that, when integrated, would provide a system that enables a user of the simulator to have some influence on the power grid and scenario that is visualized by going forward and backwards in time and increasing or decreasing wind and solar generation. With the user seeing a hardware module with buttons to press and knobs to turn. In accordance with its ultimate goal to get a better understanding of the inner workings of a power grid.

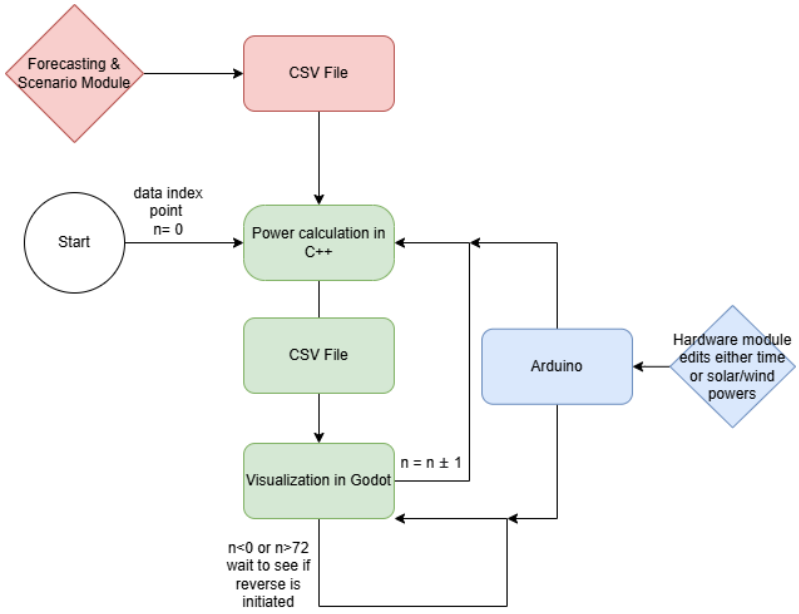


Figure 1.2: How the different modules work together in the project

The three subgroups for this project can be seen in figure 1.2. Where the hardware module is represented in red, the forecasting & scenario module in blue and finally, the visualisation & simulation module in green. In this report, the focus will be on the latter and its connections to the other modules. A thing that should be noted is that all inputs arrive from the hardware module, so the start, stop and reverse signals along side the wind and solar generative power. Furthermore, all input data, so the line input data, the bus input data and the forecasted data, come from the forecasting & scenario team. This leaves only the actual simulation and visualization for the module that is discussed in this report.

Chapter 2

Program of Requirements

In order to build this whole system some requirements were set up to measure the achieved progression of the project. These can be split into a multitude of different sets of requirements, one being global versus subgroup, thus stating the features that all modules together should have and the features that are simulation & visualization specific. Furthermore there are also mandatory and trade-off requirements, these indicate the necessity of the criteria being included in the final project versus the requirement for which it is preferable to comply with as much as possible. Finally, there are non-functional and functional requirements which indicate what functionalities the system must have and must do respectively.

2.1 Global Mandatory Requirements

1. The user must be able to interact with the simulated power grid and see how this affects the power grid dynamic.
2. The user must be able to see useful information about the grid, such as power generations and loads or line loadings.
3. The user must be able to access forecasted values of the grid, which can be useful for taking actions in the grid.
4. The user must be able to change the portrayed time in the simulation.
5. The modules must be able to send and receive data between one another in an agreed-upon format and form.
6. The system must use working solar and wind generators in real life that links to the scenario.

2.2 Subgroup Mandatory Requirements

Non-Functional Requirements

1. The simulator must only use open-source coding libraries to perform the power calculations and, turn out to be open-source itself.
2. The code should be built in such a manner that a future group can easily understand and build upon it, so with a lot of comments and perhaps even external documentation.
3. There shall be a clear manner in which the information behind the line loading colours, the tiles of the tilemap and the current date/time of the simulation are transferred to the user.
4. The visualization shall clearly relay the effect that hardware inputs have on the power grid that is being simulated via imagery on the screen, such as a stop or play picture.

Functional Requirements

1. The simulator must be able to calculate the power flows while running the simulation.
2. The visualization must convey the workings of a power grid in a manner in which the audience can comprehend the changes that occur in the simulation fully, so the simulation should not visualize faster than a human can comprehend the changes that are happening.
3. The visualization shall portray the power grid simulation in a way that is as accurate as possible to avoid creating misconceptions, in the sense that it will visualize an actual high-voltage system with realistic parameters.
4. The visualisation shall display predictions made by the forecasting module to the user.
5. The simulation must be able to calculate the power flows based on the combined information of the case and the hardware inputs via an algorithm which can work in Direct Current (DC).
6. The simulation must be able to visualize the line loading and the direction of power transfer based on the power flow calculations.
7. The simulation and visualisation must be able to handle a case that is dynamic and time-variant, and it must be possible to pause, reverse and fast-forward the simulation.

2.3 Subgroup Trade-off Requirements

1. The simulation could be able to calculate the power flows based on the combined information of the case and the hardware inputs via an algorithm which can work with multiple slack busses in Alternating Current (AC).
2. The simulation and visualization should be done as smoothly and fast as possible, so showing as little stutters or jittery behaviour as possible, preferably none.
3. The simulation should be able to run for as long as possible within the case.
4. The simulation and visualisation should be built in a manner so that a future group can make it into an augmented reality or projected reality game in a room in EWI.
5. The simulation could be implemented with a slow-motion feature by interpolating the power flow results between set time points from the data.
6. The visualization can have some degree of interactability via the computer/touchscreen by allowing the user to, for example, turn off nodes or lines or show the in-depth bus characteristics when hovering on it.
7. The visualization could come equipped with zoom and mouse drag functionalities.

Chapter 3

Deliberations

When getting started with the project, several initial decisions had to be made. This was done with all three subgroups together and with all the limitations of each individual subgroup in mind as well as the global program of requirements.

3.1 Game engine & Coding language

One of the first critical decisions that was faced was the choice of programming language and platform, as these are inherently intertwined. It was quickly realized that using a game engine would probably be a solid option to work in for this project. This is because game engines have become much more capable in simulating real world environments or creating digital twins over the years.[4] Another benefit of game engines is that they are also designed to work seamlessly with various customized controllers and inputs, facilitating easy integration with the hardware module. Additionally, game engines are well-known for their capability to produce visually appealing graphics and they are also capable of creating many unique or interesting time systems.[5]

Initially, Python was considered due to its readily available power flow libraries and its ability to easily implement forecasting algorithms. However, it was discovered that almost no game engines run on Python or support it via extensions. The two main exceptions, PyGame and Blender, did not meet the standards due to lackluster aesthetics and being overly complicated, respectively.

Consequently, it was agreed to switch to C++, as it is widely supported by most game engines, many of which are built in C++ themselves. With the decision for the programming language being set, the choice between different game engines was further narrowed down with the help of the assessments made in Sharif and Yousif Ameen [6]. In addition to these arguments a solid base for C++ was already established in a course that recently covered the main syntax and workings, and via it a solid e-book which gave all the information on syntax and other structures had already been found.[7]

This led to two primary options: Unity and Godot. Unity, while powerful, is complex and would require a significant learning curve. On the other hand, Godot is user-friendly and well-documented, and its open-source nature, combined with the C++ editing plugin, offers complete control over its entities and classes. A nice bonus next to this is that Godot has plenty of add-ons that allow the game to be projected or used in augmented reality, which was set as a goal for a future version of this project as it has been proven that augmented reality improves the educative process by increasing interest and interactability.[8]

3.2 Power Flow in C++

The simulation has to solve the power flow equations at every iteration. This is an issue as these are interconnected and thus a converging algorithm needs to be implemented. As there are a multitude of these, all coming with their own benefits and drawbacks.

When researching these algorithms the choice of a power flow algorithm for implementation in C++ was essentially narrowed to a choice between the Newton-Raphson (NR) algorithm and the Gauss-Seidel (GS) algorithm. While both algorithms are effective for solving power flow problems, they have distinct characteristics. The NR algorithm is known for its fast convergence but requires more computational effort per iteration. In contrast, the GS algorithm, although generally slower in convergence, is computationally simpler and faster per iteration.[9]

We opted for the GS algorithm to ensure the simulation runs smoothly. It is anticipated that the difference in convergence speed between the two algorithms will be negligible over 500 iterations. Additionally, achieving a perfect simulation of power flow is not within the scope of this project; the objective is to attain an accurate simulation. The GS algorithm provides sufficient accuracy for the purpose of this project while maintaining computational efficiency, aligning with the project's goals.

3.3 Scenario Decisions

The next decision that followed was what scenario the power system would represent, as power grids are everywhere and it is unrealistic to both represent low voltage (LV) distribution grids in California as well as high voltage (HV) transmission grids in another country.

As it was stated in the project proposal that this project should eventually be placed in a room in EWI to enthuse current and new students into learning more about grid management, it made sense to focus on a grid which is close to that group of people.[10] This resulted in the choice of the Dutch grid, and with that came the assumption of a 50 Hz system. After some research, it was decided that it would be more interesting to work with an AC HV transmission system as that would mean that the windmill and solar panel which the hardware module is building could represent an offshore wind park and a solar park, respectively. From this came the assumption that the scenario in which the work was going to be done would be a balanced transmission grid, meaning that single-phase calculations are able to estimate the three-phase high voltage system without too many inaccuracies.[11]

Soon after these decisions it followed that the model would be based on a paper that proposes an aggregated grid for the Dutch grid based on data from 2018.[12] This model works with multiple slack busses and models busses, lines and transformers between the 220 kV and 380 kV segments. Crucially, for the forecasting module it presents data on which forecasting can be run. From this, it followed that the base system, which is applied throughout every calculation in the power flow in the whole project, is done with a complex power base (S_B) of 100 MVA and a voltage base (V_B) of 220 or 380 kV respectively.[13]

3.4 Communication with hardware module

To make this hardware usable inside the Godot game engine and to enable the user to interact with the simulator communication between the two modules is crucial.

The first method consisted of downloading a library that would be included in the Godot project to create a new type of node inside the Godot project. The benefit of this method was that this was already used successfully by the team of this report to create a working project that could be programmed using C++ code and C++ libraries instead of the usual GDScript. This was implemented using a software construction tool called SCons and the Godot C++ repository.[14][15] So this proved that this was a possible method of implementation. For this implementation method the main files from the libraries needed to be cloned inside the project files of a Godot project and then compiled using the SCons tool.

The second method was using a C# library called System.IO.Ports and using the Script function of Godot to create different signals, which could then be emitted to other nodes in the Godot project.[16] The benefit of this method was that it used a coding library that was used in several projects already. So once again it was an already proven method of internal communication.

The third method was to make the Arduino micro-controller write keyboard inputs toward the computer enabling Godot to execute different commands depending on the keyboard inputs the Arduino would give the personal computer (PC). A disadvantage of this method is that the Arduino would take over the

keyboard so no inputs could be given anymore to the PC via the computer or via a touchscreen. As it is also a design requirement to work with a computer or a touchscreen it was decided that this would become the last resort option.

Firstly, it was decided to go with the first method, as some success with this method had already been found. This open-source serial communication library had a short installation manual on GitHub which at first made it look as a very appealing option.[17] However, due to the fact that the library was quite outdated, the installation kept failing. So this library quickly turned out to be unusable. There were several other open-source GitHub libraries but none were well-documented enough. So it was decided to go for the second method, which turned out to work just fine and further information can be found in section 4.1.

Chapter 4

Internal Communication

In this chapter the main criteria for designing the communication between the hardware and scenario & forecasting modules and the simulation and visualization module are discussed. These are based on the global program of requirements, design choices of other subsystems, and the limitations of Godot and C++ which concern the subgroup discussed in this report. Furthermore, it might be useful to read Appendix B before reading this chapter and chapter 6, as it explains the workings behind Godot and its vocabulary.

4.1 Hardware Module

The hardware and simulation module are heavily intertwined as the hardware can edit the simulation via data. This data is passed through an Arduino from the hardware side to the Godot Arduino plug-in. These Arduino commands then cause certain Godot node variables to change into new values and based on that the C++ code causes the simulation to respond accordingly.

For this an implementation of a serial connection was utilized using the System.IO.Ports library inside a C# script.[17] A C# code was developed that edited a rich-text node inside of the Godot project. The code made use of signals that could be read by all the other C++ files used for the visualization, which proved to be great for integration. The signals were created in the C# code and outputted using the “emit.signal” function of the library. After this, it was necessary to specify inside the functions of the simulation module how to call the correct signal that changes the values. The signals that were sent to Godot consisted of Strings. All the signals and their function in the simulator are written in the table bellow.

String	**Function**
-1	go backwards in simulation time
1	go forward in simulation time
2	play
3	pause
Sxxx	change solar to xxx
Wxxx	change wind to xxx

Wind & Solar values

For two busses within the scenario the generated power is artificially edited to the value that is present in the respective Godot variable of the Map Generator node.

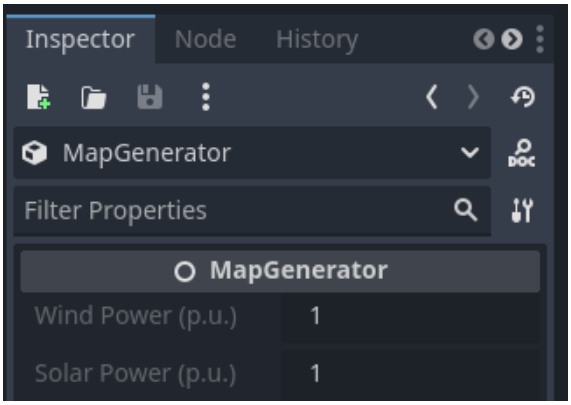


Figure 4.1: The Godot solar and wind generation variables of the map generator node

The signal used to change the wind generation is transmitted whenever the user changes the wind input with the sliding potentiometer. The signal is outputted and used in the simulation as a value for the power generated in a wind park, as a purely generative node.

The signal used to change the solar generation is gathered by the solar panel module of the hardware and is emitted to the simulation whenever the user presses a button. The value is of the solar panel is passed on to the simulation and represents the power generated in a solar park.

Time control

The signals were handled inside the Godot engine by the code which will be explained in section 6.1.6. There are four signals that control the simulation time. The go forward and go backward signal. These are a one and a minus one. They get emitted if the rotary encoder is turned in the hardware module. A one gets emitted if the encoder is turned clockwise, and a minus one is emitted if the encoder is turned counterclockwise. The pause or play signal, so a two for play and a 3 for pause gets emitted if the user presses the rotary encoder and it stops or starts the simulation time. The pause or play button gets emitted if the user presses the rotary encoder and the result of this is that it pauses or starts the simulation time, differing on the current state of the system.

Overloaded lines

On the hardware board there is a RGB LED which represents the amount of overloaded and almost overloaded lines, via emitted signals from the C++ code into the Godot engine, which works with C# code. When doing the line loading calculations in the c++ code, there is a counter that counts the amount of fully overloaded lines (more than 100% loaded) and the almost overloaded lines (between 95% and 100% loaded). To enable the hardware modules to handle signals sent from the simulation the micro-controller and the code controlling it had to be able to receive data. The serial write methods of the Serial.IO.Port C# library were used to write the data to the Arduino. The signals that were sent to the Arduino can be seen in the following table.

Signal	**Function**
1	Turn RGB LED green
2	Turn RGB LED yellow
3	Turn RGB LED red

The signals control the color of an RGB LED controlled by the micro-controller. The signal emitted to the Arduino was a three if a line exceeded its maximum rated capacity in the simulation. If the Arduino receives this signal the LED turns red. If this is not true then a check is run whether there is more than one line between 95 and 100 percent of their rated capacity the code writes a 2 to the Arduino and the RGB

LED turns yellow. If both of these conditions are not met a 1 gets emitted and the RGB LED turns green.

4.2 Scenario & Forecasting Module

The nature of communication with this module and the one in this report is based on writing to and reading from specified csv files that are sent before the simulation is ran, as this is done in python. The reasoning behind this is that after some headway was already made with Godot and c++, the scenario & forecasting team decided to do that part in python, as its libraries made the calculations much easier.

Scenario Part

For the scenario part there are two csv files which are relevant, both of which are read by the C++ code. The reason that csv files are chosen throughout this project is because it is a file type known for its their simplicity, compatibility, and efficiency. In addition to this it is in a human readable format while supporting excellent libraries that can read from and write to it in both C++ and Python.[18] The first file is ”busInputData.csv”, this file contains the initial values for each bus. The first thing it specifies is the type via a 1,2 or 3. One for the main slack node, a two for a generative bus (PV) and a three for a load bus (PQ), and the necessary information that should be initialized for that specific type of bus. Furthermore, the data for the next time step can be found directly below the data for all the busses. This entails that data initialization for a specific bus can be found at:

$$n = t_{step} \cdot N_{totalbusses} - 1 \tag{4.1}$$

	A	B	C	D	E	F	G	H	I	J	K
1	Type	PU Volt	Angle (Deg)	Gen MW	Gen Mvar	Load MW	Load Mvar	Gen Mvar(max)	Gen Mvar(min)	Slack yes,no	Slack Weight
2	1	1	0			2	1	4	-2.8	1	1
3	3					0.5	0.2	4	-2.8	0	0
4	3					2	1	4	-2.8	0	0
5	2	1.05		5	1	2	0.2	4	-2.8	0	0
6	1	1	0			2	1	4	-2.8	1	1
7	3					5	0.2	4	-2.8	0	0
8	3					1	1	4	-2.8	0	0
9	2	1.05		10	1	4	0.2	4	-2.8	0	0

Figure 4.2: Example format of a 4 bus system with 2 different defined time steps

The choice for this data structure was made based on the idea that data should be easily interchangeable between the forecasting & scenario module and the simulation & visualization module. This means that the data structure that the pandapower library outputs is the same as the data structure that the power flow calculations of the simulation take as input, or generate as output.[19]

The second csv file relates to the lines which connect the different busses. This file is thus fittingly named ”lineInputData.csv”. This file contains the p.u. values for the 4 line parameters resistance R, reactance X, conductance G, susceptance B and the maximum line capacity all provided by the scenario team. This file is not edited outside of the scenario so it does not need data for different time steps.

	A	B	C	D	E	F	G
1	From Bus	To Bus	R	X	G	B	Max MVA
2	44	27	0.005536	0.064356	0	0.000735	16
3	3	2	0.077372	0.853361	0	0.000955	16
4	4	8	0.106771	1.175719	0	0.00131	16
5	4	6	0.070347	0.773614	0	0.000866	16

Figure 4.3: Example format of 4 lines

Forecasting Part

The communication between the Forecasting module and this module is unilateral, just like it is for the scenario data. All the data that is forecasted is put into a single csv file called ”Forecast.csv” and is made available to this module.

It was agreed upon by both this subgroup and the forecasting subgroup that in this csv file the first column would include the node number and the first row the time step index for the forecasted value.

By ordering the file in this way it is easy to find specific forecast values for specific nodes. It was also decided that all predictions for a single time step would be grouped together. This creates groups of 28 nodes by 24 prediction values per time step. Each prediction per time step is placed below the last row of the previous time step.

	A	B	C	D	E	F	G	H	I	J	K
1		0	1	2	3	4	5	6	7	8	9
2	3	184.8496	185.3802	185.4144	186.3734	192.5285	201.5621	206.6579	209.3825	209.6447	209.5984
3	4	191.8669	186.0588	185.7242	188.6142	197.3084	246.6955	313.1868	388.1314	400.2298	405.4756
4	5	154.0227	154.4647	154.4932	155.2619	160.421	167.948	172.194	174.4642	174.6827	174.6441
5	6	64.58867	64.77403	64.78597	65.10832	67.27176	70.42819	72.20872	73.16073	73.25233	73.23615

Figure 4.4: Example format of the first 8 predicted values of 4 nodes

Chapter 5

Analysis of existing visualizers

Power flow visualization has been done in a bunch of different forms and even summarizing papers have been written, that analyse different forms of visualization and their successes in doing so.[20] So it is only logical that before the construction of the simulator and visualizer, several other models that are already available on the internet were analysed to see what features are already present and which ones of these are desirable for the model that is created in this report and which ones are not.

CREDC Illinois

The first model that was looked at was the model of CREDC Illinois.[21] This an applet which offers an interactive representation of a power grid. Users can explore various power generation sources, observe their outputs, and engage with interactive components to simulate grid operations. Such as increasing load and generation for some busses and removing some connections whenever they want to, to see its result. The things that are lacking in this model are the geographic realism, losses and scenarios that vary over time. Therefore, this model chooses to focus more on interactability and less on realism. Furthermore, it has to be said that the model looks quite unappealing due to the fact that some images are drawings some are pictures.

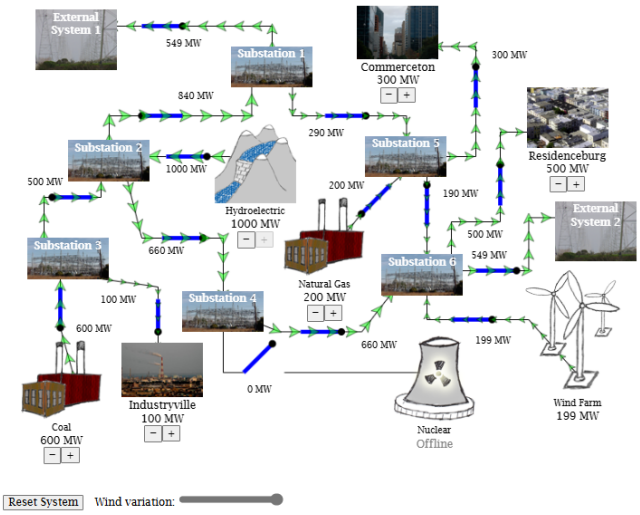


Figure 5.1: The visualization of the CREDC Illinois model

Power Factory

Secondly, the PowerFactory software was examined.[22] The PowerFactory software by DIgSILENT is a rather complex tool designed for the analysis, simulation, and modelling of power transmission systems. It is meant for transmission grid operators and also often used for serious research, as the software is highly regarded for its precision and efficiency.[23] Next to this the visualization is clean-looking, intuitive and attractive. It supports a wide range of operations and customization, but this comes paired with significant drawbacks in simplicity and understandability. The other major drawback of this system is the fact that it is not open-source and very expensive.

Figure 5.2: The visualization of the Powerfactory software

Tennet model

Lastly there is the model created by the company Tennet.[24] The TenneT Power Flow Simulator is also rather interactive and quickly proved to be the one of the three models that would be the model that most resembled the desired product. As it provides a very clean-looking detailed simulation of power flow within a grid, allowing users to visualize and analyze the effects of various operations that they can perform on the model. The model offers features for adjusting parameters and observing changes in power distribution. The model also allows the grid to be placed on a map, and thus, it allows for a geographically realistic model. Furthermore it also allows insight into the frequency and the sustainability index within the grid. Finally, what it does not have, is time variant scenarios in its simulation.

Figure 5.3: The visualization of the Tennet model

Conclusions

From this research it can be concluded that for the purposes stated in the program of requirements that some aspects from these models should be copied and merged into the model that will be created. From

the CREDC Illinois model the interactability is desirable, while the model that is created should look more appealing and be both time variant and geographically accurate to the Netherlands. From the Powerfactory software the calculations in the back-end should be taken, however they should be simpler and the data that is put into the simulations should be much less complex and therefore easier to understand for the players of the simulation. In addition to this, the software is not open source, something that is required for the model that will be created. Finally, the Tennet model has many characteristics that should be copied into the model, its clean look, interactability, informative legend and geographic accuracy. However, the one thing the Tennet model is lacking that should be added is time variant scenarios to show certain events happening in the grid over a set time period.

Chapter 6

Simulation

After having done all the research and taking into account the program of requirements, the deliberations and the internal agreements the simulator could be built. The first step in this process was the downloading of all relevant software and getting used to the new environment and its interaction with C++. This installation of the C++ plug-in for Godot follows the documentation, but since this is not a trivial installation, an installation manual with additional instructions has been put in Appendix A.

If any future groups were to follow in the footsteps of this project it is heavily recommended to take enough time for this and read every line carefully. All of the code described in this chapter can be found in its entirety and most updated version on the GitHub dedicated to this project, adequately still named BAP.PY because it was created when the language was still assumed to be python.[25]

6.1 C++ Implementation

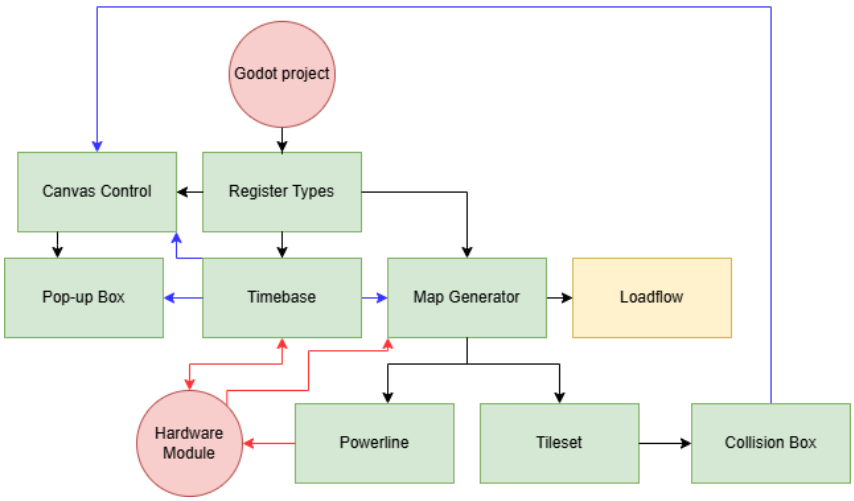


Figure 6.1: A visualization of the entanglement of the C++ code

The code that was written follows the path specified in figure 6.1, spanning several different .cpp and .h files, a single .hpp file and their internal and external signals. In this figure, the red boxes represent non-C++ entities, the green boxes represent .ccp and .h C++ files, and finally, the yellow box represents a .hpp file. The arrows also carry meaning because the black arrows represent #include statements, the blue arrows represent internal signals alongside #include statements, and the red arrows represent external signals. The functionalities of each of these files and their input/output behaviour will be discussed in this chapter.

6.1.1 Register Types

The register types files in a Godot project with custom C++ files is crucial for integrating custom classes into the Godot engine. It works via defining two main functions: initializeModule and uninitializeModule. These function are solely responsible for linking the Godot game engine to the custom C++ code that was written.

The initializeModule function is called during the engine's startup, and it registers custom classes using Godot's ClassDB::registerClass method, making them available to the engine and the editor. This means that the custom classes can now be placed inside of the game engine.

Conversely, the uninitializeModule function is called when the module is unloaded, allowing for cleanup of resources if necessary. Which it often is, as anything that is generated needs to be deconstructed, otherwise leakage will occur.

The main structure for this file was taken from the Godot C++ documentation, however it needed to be edited multiple times to add more custom nodes and functionalities. [14]

6.1.2 Mapgenerator

Mapgenerator is created as a node that extends a Godot node. In future C++ files it will become clear that some custom nodes that have been made for this project will at the start inherit preprogrammed behaviour from a certain Godot node that the C++ code builds on. In this case this is the Godot node node, which is the most basic form of node and therefore suitable for being the main custom node of the project, calling almost all of the other ones into action.

The MapGenerator class in the project is essential for managing and dynamically updating the grid with tiles and powerlines. Every time step it calls the powerline files to generate the new lines and arrows and at the beginning it is responsible for constructing the tilemap, which places the busses in their correct location on the map. Furthermore this file is also responsible for the creation of the Godot variables that represent the power generated by the solar and wind bus that is connected to the hardware module.

In the constructor the code initializes the values for wind and solar power attributes to 1 p.u. and sets pointers for tile and line nodes to null. The destructor ensures proper cleanup of all dynamically allocated memory, preventing leaks.

In the bind_methods function, the class registers methods and properties with Godot's ClassDB, making them accessible in the Godot Editor This includes methods like _on_timebase_time_index_signal for handling time-based updates and the get and set functions for wind and solar power, which are called by the hardware module.

The _ready method function, which is a function called directly after the constructor, sets up the node by allocating memory for tiles and powerlines, adding them as children to the main node. It connects the MapGenerator to the time‘_index_signal from the timebase node, ensuring ability to respond to a signal that would indicate the next time step.

If such a trigger would arrive from timebase at MapGenerator this would cause the MapGenerator to delete all current lines and it will then instruct the load flow to do all the new calculations and based on that the powerline and Tileset will generate the new map representing the state in the new time step.

This C++ code also has a lot of debugging features build in as well as highly documented code. An example of this are the print tree and print node tree functions which have proved to be great help in debugging by printing the scene tree, providing a clear overview of the node hierarchy and aiding in troubleshooting and target selection in different C++ files.

6.1.3 Tileset

The Tileset files define and implement a custom Tileset class that extends Godot Engine's TileMap class. Which is a class which allows for tile placement in a defined grid.

The Tileset defines the _create_map() method which populates the grid with different tiles each associated with specific type of bus like a solar panel for a solar park of a city for a bus located in the place of a city. This method also handles the instantiation of collision boxes for interactive game elements, ensuring that physical interactions within the game map are managed correctly.

The Tileset functionality is only called once, at the first running of the project. As there is no need as of now to edit the scenario, such as where the busses are located, as there is only one. However, the way this was built allows for easy addition of more scenarios and alternation between them during the simulation, should any future team want to work on that.

6.1.4 Powerline

The Powerline entity, defined inside of the Powerline files, is a custom class within the Godot game engine designed to manage the creation and destruction of powerline representations in the Godot project. The class builds on Godot's Node2D class.

Powerline creates powerline by the function _create_powerlines which reads data and uses this data to instantiate and configure Line2D and Polygon2D nodes, representing the powerlines and their directional arrowheads, respectively.

It reads the appropriate positions, directions, and colors for these graphical elements based on the output data of the loadflow.hpp which stores its results in a csv file. This means that in this code there is a redundant writing to and reading from a csv file. However, the choice to keep it in this manner was because no significant slowdown has occurred yet, showing that the processes are not too complex yet. Additionally, for debugging purposes this csv file was more than helpful, as it shows all results of the last loadflow. A significant number of bugs were filtered because of this file and it can almost be guaranteed that, if changes were made to the project, even more will be filtered because of this redundancy in the code.

Furthermore, the class emits signals, such as red_lines_signal and overloaded_lines_signal, to notify the hardware module of the system about overloaded or almost overloaded lines based on the powerline data. Finally, The _cleanup_powerlines method ensures that all dynamically created nodes are properly removed and freed, maintaining the integrity of the system's memory management and preventing leaks.

6.1.5 Loadflow

The loadflow.hpp file is the file which contains all the calculations of the power flow algorithm. This file is largely based on an open-source GitHub commit which contained power flow calculations via a Gauss-Seidel algorithm in C++.[26] However, it quickly turned out that the file which was found online did not contain all the functionalities which were desired to be in the project, so these were coded into the file.

The very first thing that this file does is define the total amount of busses it should iterate over and what the bus number of the solar and wind bus from the hardware is. In the scenario defined by the scenario module this is 47 busses and wind and solar are bus numbers 46 and 47 respectively.

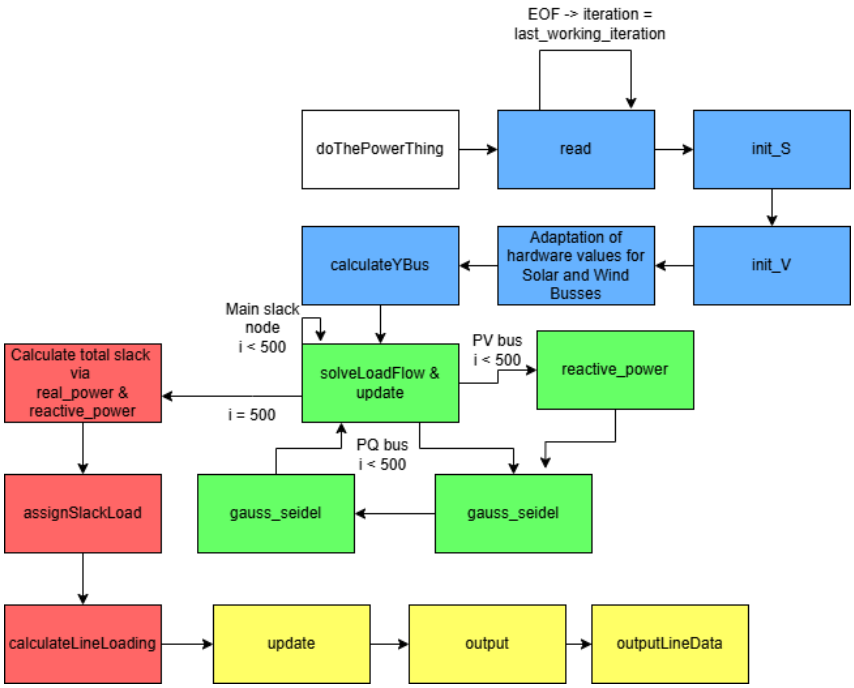


Figure 6.2: A visualization of the functionalities inside of the loadflow file

As one can see in figure 6.2 the functions can be split into four sections, with debugging functions being a fifth. The functions will be handled in order.

The function that can be called from the outside to run the whole system is called dothepowerthing and it is the main function that coordinates the power flow analysis. It calls the read function to load the input data, the init_S and the init_V functions after which the solar and wind values specified by the hardware module are taken for the power generated by the solar and wind busses. Then the Y bus is created via the calculateYBus function and eventually the power flow is solved via the solveLoadFlow function.

Initialization and data gathering

The read function is responsible for reading bus and line input data from the input CSV files. It begins by opening the bus input data file and reading each line into a string stream. Each value is converted to a double and stored in a vector. If conversion fails, the value is defaulted to zero. This vector is then appended to the bus data matrix. The process continues until all bus data has been read or the end of the file is reached. If the end of the file is reached without reading all data, the function recursively calls itself with the last successfully read iteration. After successfully reading the bus data, the function opens the line input data file and performs the same operations there to read and store line data.

The init_S function initializes the complex power S at each bus. This involves setting the initial values for generated real power P and generated reactive power Q based on the type for generators and loads. With the PV busses getting the values specified in the input file and PQ busses having both set to 0. The initialization is crucial for setting up the initial conditions for the power flow calculations, and wrong values can quickly lead to a non-converging system of equations.[27][28]

The init_V function initializes the voltage magnitudes and angles for each bus. This involves setting a flat start where all voltage magnitudes are set to 1.0 per unit and all angles are set to 0 radians except those which are specified in the businput data, PV busses. This provides a starting point for the iterative power flow solution.

The calculateYbus function constructs the admittance matrix, also called the Ybus, which represents the network's admittance between buses. The admittance ($Y = \frac{1}{R+jX}$) is computed for each line. Self-admittance (Y_{ii}) for each bus i is calculated as the sum of all line admittances connected to the bus, while

mutual admittance ($Y_{ij} = -Y_{line}$) is the negative of the line admittance between buses i and j.

Iterative processes

The solveloadflow function addresses the load flow problem using an iterative method, for determining the voltage magnitudes ($|V|$), phase angles (θ), and power flows in the network. The function employs the Gauss-Seidel method, updating the voltage at each bus i using the formula[27][28]:

$$V_i^{(k+1)} = \frac{1}{Y_{ii}} \left(\frac{P_i - jQ_i}{V_i^{(k)*}} - \sum_{j \neq i} Y_{ij} V_j^{(k)} \right) \tag{6.1}$$

where P_i and Q_i are the real and reactive power injections at bus i, Y_{ii} is the admittance of bus i, Y_{ij} is the admittance between buses i and j and V_i and V_j are the voltage magnitudes at buses i and j.

The iterative process continues, checking for convergence by evaluating power mismatches (ΔP and ΔQ). If mismatches fall within a specified tolerance, the iterations stop. Otherwise, they continue until the maximum number of iterations is reached, 500 in the case of this project. The function computes the final power flows and bus voltages upon convergence. It then updates the output bus and line data and outputs this to the csv files.

The gauss_seidel function implements the Gauss-Seidel iterative method for solving power flow equations. For each iteration, the function updates the voltage magnitude and angle for each bus based on the admittance matrix and the power mismatch. The iterative process continues until the changes in voltage magnitudes and angles between successive iterations are within an acceptable tolerance or the maximum iterations has been reached. The function ensures that voltage updates consider the constraints on reactive power generation limits for PV buses, which have been specified in the bus input csv file, changing their type if too much reactive power is produced. The final voltage values are then returned for further computations.

The real_power function calculates the real power injected at a bus. The function iterates over all buses and sums the contributions from each bus based on the admittance matrix and the current voltage magnitudes and angles. The real power P_i injected at bus i is given by[27][28]:

$$P_i = V_i \sum_{j=1}^{N} V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}) \tag{6.2}$$

where V_i and V_j are the voltage magnitudes at buses i and j, θ_{ij} is the voltage angle difference between buses i and j, and G_{ij} and B_{ij} are the conductance and susceptance between buses i and j.

The reactive_power function calculates the reactive power injected at a bus. Following a similar algorithm as the real power calculation, it iterates over all buses and sums the contributions from each bus. The reactive power Q_i injected at bus i is given by[27][28]:

$$Q_i = V_i \sum_{j=1}^{N} V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}) \tag{6.3}$$

where V_i and V_j are the voltage magnitudes at buses i and j, θ_{ij} is the voltage angle difference between buses i and j, and G_{ij} and B_{ij} are the conductance and susceptance between buses i and j.

The update function updates the bus data matrix after each iteration of the power flow in the solveLoadFlow function. It recalculates the power mismatch for each bus and updates the voltage magnitudes and angles. This function ensures that the latest values are used in subsequent iterations, improving the accuracy of the solution each iteration.

Final calculations

The final calculations, the calculations after the iterations are done start by running the real_power and reactive power functions to calculate the total slack.

This is followed by the function assignSlackLoad, which takes the total slack and assigns all the busses capable of taking slack a portion of the total slack corresponding to the share of their slack weight in the summation of all slack weights. It is here that it is important to look at the difference in functionality behind the main slack bus and busses that can handle slack. The main slack node is where the voltage angle 0 is defined and it has no load or generation other than the slack functionality. Meanwhile, other busses that can handle slack do not, necessarily, have the angle always at 0 and have the slack functionality added or subtracted from its normal operational generation or load as PV bus.

The calculate_line_loading function calculates the power flow S_{ij} from bus i to bus j for each power line. This function uses the voltage magnitudes and angles from the buses connected by the line and elements of the admittance matrix. The power flow is given by[27][28]:

$$S_{ij} = V_i V_j (G_{ij} - jB_{ij}) e^{j\theta_{ij}} \tag{6.4}$$

where V_i and V_j are the voltage magnitudes at buses i and j, θ_{ij} is the voltage angle difference between buses i and j, and G_{ij} and B_{ij} are the conductance and susceptance between buses i and j.

Output

After the final calculation is run, the update function is called for one last time in order to ensure that the voltage magnitudes and angles are updated to their final values.

The output function then writes the final bus data to a csv file. This includes the voltage magnitudes and angles, as well as the calculated real and reactive power injections at each bus.

The outputLineData function calculates and prints the line loading. It computes the power flow S_{ij} from bus i to bus j using the formula[27][28]:

$$S_{ij} = V_i V_j (G_{ij} - jB_{ij}) e^{j\theta_{ij}} \tag{6.5}$$

where V_i and V_j are the voltage magnitudes at buses i and j, θ_{ij} is the voltage angle difference between buses i and j, and G_{ij} and B_{ij} are the conductance and susceptance between buses i and j.

The function then writes the line loading data to csv file, which is useful for identifying overloaded lines and further visualization of the power system.

Debugging functions

The printBusData, printLineData and printYbus functions print their respective information the console. These functions are very useful for debugging and verifying the correctness of the power flow solution and inputs and can be called whenever and wherever issues are encountered if they are encountered.

6.1.6 Timebase

The Timebase class is a custom class that inherits its functionality from the Godot Timer class. This class functions as the time keeping part of the entire system.

In Godot timers are essentially a basic function that counts down from a certain start amount to 0, the timer is updated after every frame is rendered by the engine. This makes the timers frame rate dependent, which is not desirable but for time steps that are sufficiently large (>0.25 s) it has not caused any issues.

Upon reaching 0 a signal called ”timeout” is emitted by the timer, the timebase for the simulation is implemented by incrementing or decrementing a value called ”time index” every time the timeout signal is received. Combining the timeout signal of the timer with a direction signal and varying its speed makes it possible to control time fully.

The _on_timebase_timeout method is called on every timeout of the timer and updates the time index while also emitting the signals ”time_index_signal” and ”clock_tracker_signal”. These signals send the time index to the Pop-up Box class and the time index, pause status and time flow direction to the Canvas Control

class respectively.

The _on_direction_signal method is called every time the hardware is interacted with. Based on the data sent in this signal the timebase module will determine whether the simulation needs to revert, pause or forward.

6.1.7 Canvas Control

The Canvas Control class is a custom version of the standard Godot Control class. The main function of this class is to be the parent to all drawn text and popups.

The reason for this is the way Godot handles styling of these objects, when an object like a window or text label is created it will automatically inherit the theme of its parent node. By making all the popups and text a child of this control node the theme can be centrally controlled and edited.

Inside the _create_clock_labels and _update_clock_labels methods of this class the Canvas Control node creates two labels that show the user of the simulation the simulation time and status. Furthermore, within the _delete_popup and check_child_ready methods the popups for the forecasted values are deleted and created.

6.1.8 Pop-up Box

This class is a custom version of the Godot class PopupPanel, which can create popup windows on the screen. In the _create_line_legend and _create_icon_legend methods the window settings for the legends are set, by calling the _create_line_legend_entry and _create_icon_legend_entry respectively the data within the legends is created. Both legends are created at the beginning of the simulation and do not get updated while the simulation is running.

The Pop-up Box class is also responsible for creating the windows that show the forecasted values for each node when hovering the mouse on the node. This requires the simulation to find the correct data for the node and show it accordingly. To this end the following methods were implemented: _create_content, _get_node_index, _get_node_data, _popup_update and _update_label. In the _create_content method multiple container nodes are added as children of each other, these nodes determine the structure of the created popup. The first container is a MarginContainer, this container creates a space between the window border and the children of this container. To the MarginContainer a VBoxContainer is added, this container type structures all its children to be aligned vertically below each other. After these two containers are added labels are added as children to the VBoxContainer, making them appear below each other. These labels contain data that is found using _get_node_index and _get_node_data.

The _get_node_index function is used to find the node index number. This is done by using the x and y tile coordinates of the node that the mouse is on to find the corresponding bus index number.

After the bus index is found _get_node_data is called to fetch the data of the node. This function reads the ”Output.csv” and ”Forecast.csv” files based on the given time index and bus index. The values that are found are stored in a 25-number array of floats and are returned by reference.

When the user is hovering their mouse over the popup it should still be able to update itself, that is why _popup_update and _update_label were implemented.

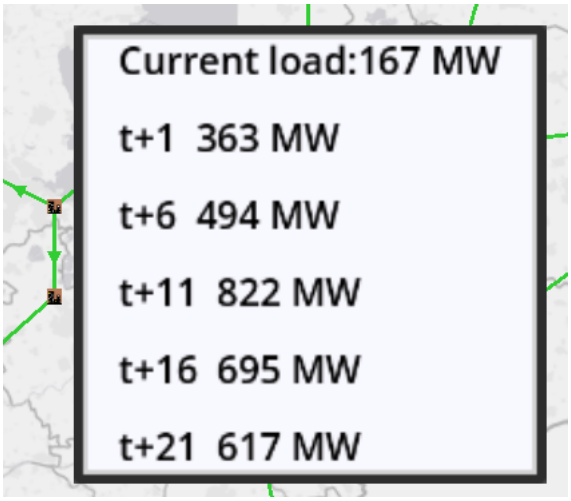


Figure 6.3: A popup window for forecast values

These two methods work in almost the same way as the _create_content method, the main difference is that instead of creating new containers and labels the existing ones are updated. This was implemented by introducing a labelnr variable which allows for the editing of a specific label within the popup.

6.1.9 Collision Box

The Collision Box class is of the type Area2D. This class creates an invisible area that can detect when a mouse enters or leaves the area. This is used to determine if the user is hovering their mouse over a node. These Collision Boxes are created on the location of every node during the placement of the node images on the map.

In the initialization of an instance of this class two other Godot classes are used, CollisionShape2D and RectangleShape2D. The RectangleShape2D is used to set the shape of the CollisionShape2D to a square which matches the size of the nodes drawn on the screen. The CollisionShape2D is set as a child of the Area2D and sends signals based on whether the mouse enters or exits the shape.

The _mouse_enter method is called when the CollisionShape2D has detected that the mouse has entered its area. This method navigates to the Canvas Control node and tells it to create a popup at the location of the node.

The _mouse_exit method uses the CollisonShape2D for the purpose of detecting when the mouse leaves the area specified by it. Once the mouse exits the area the Canvas Control node is told to remove the popup from the screen.

6.2 Godot Project

As shown in the previous segments, the coding only enhances existing Godot nodes and increases their functionalities. To construct the project all of these custom nodes still have to be added to a Node2D, which is the main scene. As one can see in figure 6.4, the three custom nodes which were written to Godot via the register types files are added to the scene. In addition to this the Camera2D node is added with the camera zoom set to 0.4 and position set to +500,0. This is required because when one runs the program the player will see what the camera encompasses.

Finally, the background map needs to be added to the scene, this can be any image file and the one used in this project is further discussed in section 7. The final thing which is crucial in the set-up of this project is to ensure that the layers are correctly assigned via either hard coding in the C++ code or in the project via the nodes that are added in the scene. The map should always be at the bottom, above it the lines and

polygons, on top of that the tiles and finally any legend or pop-up box generated by a C++ file should be at the very front.

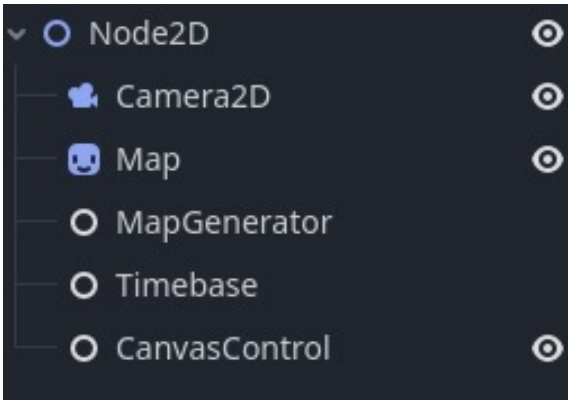


Figure 6.4: The structure of the Godot nodes inside of Godot

Chapter 7

Visualization

As shown in the introduction, chapter 1, of this thesis there are many different ways to show information in a visual way. Coming up with a way of showing the complex information of a power grid in a visually appealing way is not an easy task because it quickly gets convoluted.
In this section, the main design choices for the visualisation of the simulation are explained and elaborated upon.

Figure 7.1: A screenshot from the power grid simulator

7.1 Visualisation design choices

7.1.1 Color usage

For the color usage in the visualisation the choice for bright colors was made, this choice allows for the attention of the user to be directed to the most important aspects of the simulation.

Because humans have certain emotional associations with specific colours, the colours that most clearly portray good and bad scenarios are chosen to be black and red for bad situations and green and blue for good or acceptable situations. Research in this area has shown that these colours evoke negative and positive reactions respectively.[29] By using these colors in this way the user of the simulation will be able to quickly grasp the situation of the power grid being portrayed.

7.1.2 Iconography

For the visualization it should be created such that a new user can clearly understand the imagery of the simulation. To facilitate this some images were created to represent the different types of buses in the power grid, although the resolution was limited to 16 by 16 pixels, they are able to clearly portray what type of bus is shown.

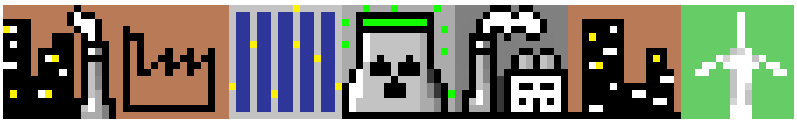
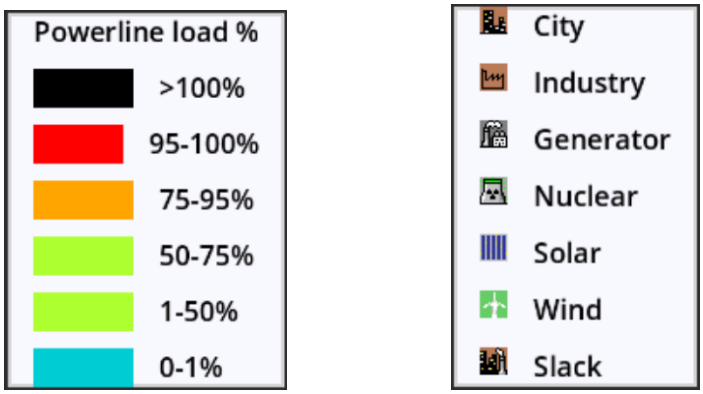


Figure 7.2: Icons used for visualization

7.1.3 Legends

When visualizing something it is important to effectively convey the meaning of different images and colors. That is why two legends were implemented, one for the icons of the buses that are present on the map and a legend for the % of line loading the colors of the lines present. Aside from the legends, the current time in the simulation, and the status of the simulation are also shown.



(a) Line loading % legend (b) Icon meaning legend

Figure 7.3: Legends used in the simulation window

7.1.4 Maps

Because the scenario considered in this thesis is that of the high voltage transmission grid in the Netherlands the map was chosen to correspond to this choice.

This map is easily interchangeable with that of other countries, making it possible to adapt the simulation to any scenario desired by the user. With the only real constraint being the availability of the data needed to simulate a power grid.

7.1.5 Node placement

Sometimes in the model of a power grid there are many buses placed near or on top of each other, when this is the case it is possible that the visualisation of that grid becomes very cluttered.

To this end, in the scenario that has been portrayed throughout this thesis, the choice was made to move some buses to different positions to avoid cluttering. While this makes the visual representation a bit less true to life the calculations are not altered, so the portrayed data is still accurate. The main issues for the scenario lied in Groningen (top right) where 7 nodes lied on top of one another. Figure 7.4 shows the nodes

how they would be, if their positions remained true, showing the nature of some nodes cluttered on top of one another.

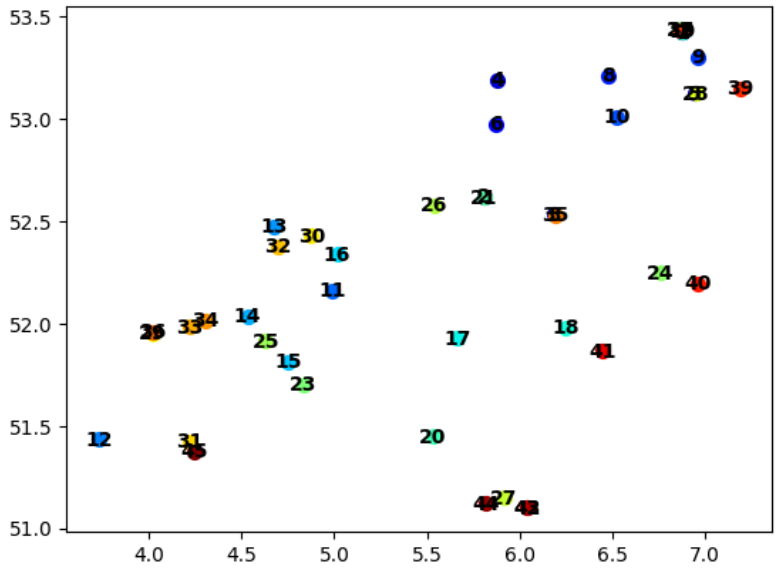


Figure 7.4: The true location of all the busses in the scenario of this report

The main motivation for this choice is that the trade-off between the intuitiveness and realism of the simulated power grid is worth it because a cluttered power grid quickly becomes very difficult to understand.

This design decision is supported by research that shows that when presented with well-focused and non-cluttered visualisations test subjects are more likely to recall them and have a better positive sentiment when compared to the response to cluttered or unfocused visualisations.[30]

Chapter 8

Conclusions

Reflecting back, this project aimed to create the base for an interactive power grid simulation and visualization system with specific requirements. The primary objectives included enabling user interaction with the simulated power grid, displaying grid information, incorporating forecasted values, and facilitating time manipulation via the hardware module within the simulation. These goals were set to ensure a base was built for other groups to build upon so that eventually it could become a permanent feature of a new maker space in the EEMCS building.

The project successfully achieved several requirements. The user can interact with the simulated power grid, influencing and observing the effects in real-time, allowing inputs such as play, pause, fast-forward, and reverse to manipulate the time of the simulation. In addition to this the user can change the amount of wind or solar power in the simulation in real-time by changing the amount of light or wind in real-life. These interactions are done together with a hardware module. The visualization then displays information about the grid, including power generation, loads, and line loadings, which help the user's understanding of the grid's operation. Additionally, the simulation includes the forecasted values of the forecasting module, showing how the grid is likely to change in its future states.

However, some aspects of the project were not fully realized. While the simulation can calculate power flows and visualize the grid's operation, certain trade-off requirements were only partially met. An example of this are the interactive features of turning off nodes or lines and adjusting load/generation directly through the visualization were not fully implemented, limiting the extent of user interaction. It would still require some time to complete these parts of the project, and therefore these trade of requirements that have not been reached are discussed further in chapter 9.

Overall, the project made progress that adequately satisfies all mandatory requirements as stated in chapter 2 and even most of the trade-off requirements, creating a functional power grid simulation and visualization system. The establishment of a solid foundation for future development via the documentation, the code commenting and the appendices make it a robust starting point. To realize the project's full potential and make it suitable for display in the maker space, further refinement and expansion are necessary. However, what exists now provides a solid foundation that future groups can easily build upon.

Chapter 9

Discussions & Suggestions for future groups

Since it was specified in the project proposal and in the project requirements in chapter 2 that the future of this project lies with another group and eventually a permanent workable model in a new maker space room in EEMCS, it feels logical to assume that there will be future groups working upon the foundation that was build.[10] To ensure that that process goes as smoothly as possible there was a significant amount of time spent on documenting code and increasing its legibility. Moreover, it felt necessary to outline the direction for the next steps. While coding and writing the report, this seems obvious, but it is often unclear to those who join a project that is already in progress.

9.1 Tips and retaining issues

First of all there is a strong recommendation to not underestimate the time to get all the software downloaded and running correctly. This took longer than expected, mainly because the Godot CPP documentation is written in unclear language and feels chaotic.[14] Furthermore, it was also experienced that on different PC's different issues arose with the installation. So even though it was attempted to write a more clear installation guide in appendix A, some troubleshooting might still be necessary.

The second major tip is that due to the nature of the C++ compiler, SCons, being outside of the Godot executable almost all debugging has to be done manually. This entails that it happens more often than desired that Godot simply instantly crashes after opening or running. For debugging these cases excellent knowledge of C++, Godot and the code is required. The main tip to deal with situations like these is opening Godot via the command prompt and using "cout" statements instead of Godot's "Utilityfunctions" so that it can be seen what the final message is before the program exits, as the command prompt stays open even after Godot has crashed.

Another good tip is to look at the hard-coded .csv files as well as the .tres file, for the tile animations. These were implemented in loadflow.hpp, powerline.cpp and tileset.cpp because the C++ files are not in the project path, but in a separate src folder which the SConstruct file can find. This makes it impossible to use relative pathing. In addition to this Godot really does not appreciate .csv files in its project path as it will try to interpret it in a lot of ways and it will spew out meaningless files during simulation which will quickly clutter your project. Due to this there are 6 hard-coded paths in the code currently, which feel unnecessary and cause tiresome work every time when integrating the code of someone else. These hard-coded paths are well documented and error messages pop-up if they are not changed, but it would improve the project to find a way to get rid of them.

The final tip relates to the fact that eventually while adding extra functionalities it might be discovered that the code slows down the simulation, and is no longer able to keep up with the simulation time specified by Godot. As of now no slowdowns have been noticed, but if they are noticed there are several areas where optimization can happen. An example of this is the way the output data from the loadflow.hpp file is given

to the rest code. As of now this is written to csv files and other code reads from this csv file. Which is done as it allows the coder to keep a constant and easily accessibly record of the state of the powerflow. However, this is of course redundant for any final product, and should be fixed in case of slowdown. To further avoid slowdowns proper memory management is key, because otherwise memory leakage will cause the simulation to become slower the longer it goes on. To this end it is advised to read up on memory management techniques in C++ like smart pointers and garbage collectors.[31]

9.2 Suggestions for progress

The next steps that are interesting to look at for future groups stem largely from the uncompleted parts of the trade-off requirements in the program of requirements, chapter 2 and the steps that would have been made if the BAP project would have continued for a longer duration of time.

The first idea is to enhance the camera functionality and add zoom and drag functionality to improve the user experience, as the map of the dutch high power grid, and the only scenario that is included in the project as of now, contains quite small icons. This functionality can easily be added by adding GDScript, C# or C++ code to the Camera2D node which is already present in the scene.

The second idea relates to adding more scenarios, since the groundwork is already there this step should not take that long, but it is very much in the interest of the user to be able to play with multiple systems. This requires finding a new data set on which to build, but as soon as that is found the system should be rebuild in Godot via the tileset and the powerline C++ files and scaled to another background image. The code already checks for a scenario number when selecting what powerflow calculations to do and what map to build, but this is currently hard-coded to scenario '0' with no other options.

Another idea is to add interactability via the computer/touchscreen by allowing the user to off nodes or lines. This is an important one, because while the foundation has been build in this project, the interactability is limited as of now. However, this can be added with relative ease as the project as of now is build in a rather modular manner. With a lot of debugging tools at the disposal of the coder.

Another functionality is to implement the simulation with a slow motion feature by interpolation the power flow results between set time points from the data. This is a very interesting feature that definitely should be possible to implement. However, this might slow the simulation down a lot, as it would require the simulation to read two time steps worth of data.

A further example of more interactability could be adding more load/generation at a node of choosing, but this will render the predictions made by the forecasting team inaccurate, so a solution will have to be found for that.

Besides these there is the idea to improve interactability by adding the ability for the player of the simulation to add and remove lines and busses. However, this would be quite complicated and will result in quite a challenge alongside with the forecasting module no longer providing relevant data for the same system. A solution for this problem with the forecasting module could be to bring the forecasting process within the C++ umbrella, but while that is possible some research would have to be done and it should not be assumed to be simple to implement. This is stated because data management is often easier in python compared to C++.

The final manner of increased functionality that will be suggested in this project is by linking the speed-up and slow-down functionalities which already exist inside of the timebase to new hardware components. This gives the user even more control over the simulation and as of now the code already works, but it has only been run with keyboard inputs instead of hardware inputs. This should be easily applicable since the entire method of communication can remain the same, only the extra input in Arduino needs to be added alongside an extra visualization module representing the speed at which time is progressing.

Appendices

Appendix A

Software installation

A.1 Software installation

To be able to use the c++ programming language in the Godot engine the installation of something called GDExtension is necessary. The main benefit of GDExtension is that it allows the Godot engine to interact with custom code and shared libraries.
This appendix is written for the sole purpose of making the installation process of GDExtension easier and faster.
Please note that this install guide is only valid for pc's running windows as operating system

A.1.1 Prerequisites

In order for the installation to work the following components are required:

- A Godot 4 executable
- A c++ compiler
- Python
- An installation of SCons
- The Godot-cpp repository

The Godot executable

To work with Godot the game engine itself is of course needed, the engine can be downloaded from the official Godot website.[1]
Please note that there are different versions of Godot, it is advised to use a Godot version that is labeled as "stable", otherwise unexpected or unsolveable errors appear.

The c++ compiler

It is advised to download and install the latest visual studio version, this makes sure that the newest c++ compiler will be installed on your device. Aside from the standard visual studio installation the visual studio build should also be installed.
Both of these can be installed through the visual studio installer, which can be downloaded from the visual studio website.[2]

When installing the build tools makes sure to check the boxes as seen in the image below.

[1] `https://godotengine.org/download/`
[2] `https://visualstudio.microsoft.com/downloads/`

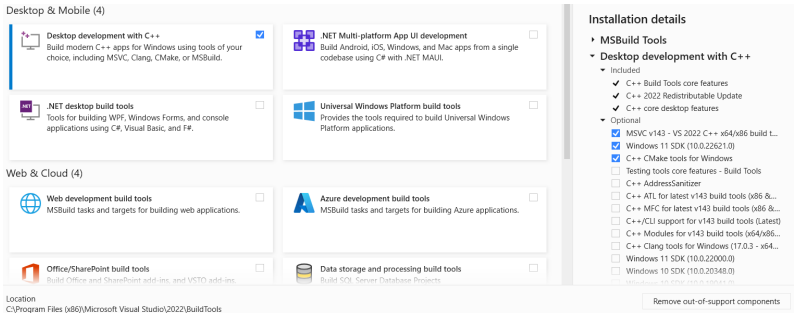


Figure A.1: MSVC Build tool installation window

Python

For the installation Python is also needed, to check if it is installed on your device run the following command in your cmd:

```
where python
```

This will find the install location of Python on your device, to check the version that is installed run this command:

```
python -version
```

This will return the version of Python that is installed on the device. If the installed version of Python is not Python3 then it this needs to be installed. This can be done from the Python website[3] or directly from the Microsoft store.
Please note that multiple versions of Python can be installed simultaneously, this might cause the wrong version to be used. Either ensure that the correct version is used, or uninstall all unnecessary and old versions of Python

SCons

To build the c++ code and make it runnable in the Godot engine it is necessary to use the build tool SCons. This build tool can be installed through the command prompt of your pc.
Before SCons can be installed something called pip needs to be installed, the command for installing pip is shown below.

```
python -m ensurepip --upgrade
```

This command might return that pip is already installed, in that case it is good to update pip to the newest possible version, this is done with the following command:

```
python -m pip install --upgrade pip
```

After pip is successfully installed and upgraded SCons can be installed using pip. To do this the following command must be executed:

```
python -m pip install scons
```

Or to install SCons only for a specific user:

```
python -m pip install --user scons
```

Godot-cpp

The Godot-cpp bindings are also needed to run c++ code in godot, these will be downloaded through GitHub, but this will be explained in the next section as the install location is important.

[3] https://www.python.org/downloads/

A.1.2 Project file structure

For the c++ code to work a specific file structure needs to be adhered to. To create the needed file structure a new folder should be created which will have three sub-folders and one file:

- A project folder which contains the Godot project
- The Godot-cpp folder
- An src folder which contains the code source files
- An SConstruct file

Godot project folder

Creating a Godot project folder is best done through Godot itself. By navigating to the main folder where the project needs to be placed and creating a new project all the necessary files are created.

Godot-cpp

The creation of the Godot-cpp folder is done by downloading it through GitHub.
To do this navigate to the main folder which now also contains your project, this is easily done by typing "cmd" into the address bar of the folder. Once in the correct folder there are two ways to install the Godot-cpp bindings:
The first way is to add it as a git sub-module, to do this the following commands should be run:

```
git init
git submodule add -b x.x https://github.com/godotengine/godot-cpp
cd godot-cpp
git submodule update --init
```

The second method is cloning the folder straight from GitHub.

```
git clone -b x.x https://github.com/godotengine/godot-cpp
git submodule update --init
```

In both these methods x.x should be replaced with the version of Godot you are planning to use.

src

The src folder is where all c++ files and their headers will be placed. This folder can be created as an empty folder for now, files will be put in it later.

SConstruct

The SConstruct file should be copied into the folder as well, the file used during this project is an example file from the Godot-cpp documentation and can be downloaded on this page.[4]

Final file structure

After following these steps the file structure should look like this:

```
Godot Example
|
+--Project/
|
```

[4] https://docs.godotengine.org/en/stable/tutorials/scripting/gdextension/gdextension_cpp_example.html

```
+--godot-cpp/
|
+--src/
|
+--SConstruct
```

A.1.3 Installation

After the file structure has been created and the prerequisites have been downloaded a few more steps are needed to complete the installation of the Godot-cpp extension.

Building the bindings

To create the c++ bindings for the used Godot version run your Godot executable with the following command:

```
godot --dump-extension-api
```

If this gives an error make sure the ”godot” part of this command matches the name of your actual Godot executable, with this the command becomes ”Godot_v4.2.2-stable_mono_win64 –dump-extension-api” for example.
To build the bindings navigate to the godot-cpp folder and run the following command from the command prompt:

```
scons platform=<platform> custom_api_file=<PATH_TO_FILE>
```

Where the platform is your Operating System type, and PATH_TO_FILE is the path to the extension_api.json file which was made in the previous step.

A.1.4 Adding c++ code

It is now possible to add custom c++ code to Godot by adding them to the src folder.
After adding a file to the src folder it is important to expand the register_types file to accommodate any new files, otherwise it will not be possible to find any custom Godot classes in the engine window.

Building the code

Every time custom c++ code is updated the custom plugin needs to be rebuilt.
The code is rebuilt using this command:

```
scons platform=<platform>
```

With platform again being the Operating System of your computer.
Please note that on some computers scons is case sensitive so SCons needs to be used.

First time building

After building the plugin for the first time one more file needs to be added. An example for this file called ”gdexample.gdextension” can be found on here.[5]
This file should be added to the bin folder of the Godot project.

[5] https://docs.godotengine.org/en/stable/tutorials/scripting/gdextension/gdextension_cpp_example.html

Please note that this manual was written based purely on personal experience.
If any issues arise during this installation it is advised to thoroughly read any error messages that might appear, as these can give helpful clues as to what is going wrong.

Appendix B

Godot workings and vocabulary

This appendix will handle the explanation of the words and concepts used when talking about Godot related topics. Almost all of these concepts are covered in length in the very helpful Godot documentation. [32] It is highly recommended to consult this documentation if the explanation in this appendix raises more questions than it answers.

The very first thing that should be made clear is that Godot works with Nodes, Nodes are the building blocks of a Godot project. A single node fulfils a single functionality, such as playing a sound or determining the light levels.

After combining multiple of these nodes into a system which shows more complex behaviour a scene will have been created. Each project has to have at least one scene, which is defined as the main scene.

Furthermore, the way to combine nodes also is dependant on the functionality that is desired. In general nodes are organized in a hierarchical structure. A node can have only one parent and multiple children, with the parent node being able to affect its children in various ways such as transforming their position or propagating signals to them.

Additionally, as mentioned earlier, nodes have functionalities, nodes with the same functionalities are called nodes with the same class. In the project in this report several different node classes are used, such as line2D, polygon2D, camera2D and tilemap. All of these classes come with different inherited properties from their class that are either kept the same or are altered in the project or added to by the c++ code, C# or GDScript, godots own programming language.

If two nodes want to talk to one another, this is done via signals. Signals are the messaging system within Godot, as they allow nodes to communicate without direct references. This means a button node can emit a signal when pressed which other nodes can listen to and respond to.

When using Godot the initialization order needs to be taken into account, because some functionality like connecting signals can cause crashes when done in the wrong place or at the wrong time. In Godot every node gets created and initialized in its constructor, after this constructor all of the node's children are constructed. If a node is constructed and it has already constructed all its children, the node is added to the scene and its ready function is automatically called. The ready function of a parent is called after all of its children have called their respective ready functions and have been added to the scene.

A node is only visible once it is added to the scene the connection of signals can only be done in the _ready function, otherwise errors will occur.

Bibliography

[1] Jan Veřmiřovský. "The Importance of Visualisation in Education". In: *Proceedings of the Conference on Education and ICT*. Department of Information and Communication Technology, University of Ostrava. Fr.Šrámka 3, Ostrava – Mariánské Hory, 709 00, Czech Republic, 2022. URL: https://www.example.com/paper-url.

[2] Max Roser. "The map we need if we want to think about how global living conditions are changing". In: *Our World in Data* (2018). https://ourworldindata.org/world-population-cartogram.

[3] *Introduction to simulation*. Dec. 2011. URL: https://ieeexplore.ieee.org/abstract/document/6147858.

[4] *Digital Twin Simulation of Connected and Automated Vehicles with the Unity Game Engine*. July 2021. URL: https://ieeexplore.ieee.org/document/9540074/citations?tabFilter=papers#citations.

[5] Christopher Hanson. *Game Time: Understanding temporality in video games*. 2018. ISBN: 9780253032829. URL: https://muse.jhu.edu/book/58726/.

[6] Karzan Hussein Sharif and Siddeeq Yousif Ameen. "Game Engines Evaluation for Serious Game Development in Education". In: *2021 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. 2021, pp. 1–6. DOI: 10.23919/SoftCOM52868.2021.9559053.

[7] Juan Soulie. *C++ Language Tutorial*. 2024. URL: https://www.cplusplus.com/doc/tutorial.

[8] Maria-Blanca Ibanez and Carlos Delgado-Kloos. "Augmented reality for STEM learning: A systematic review". In: *Computers and Education* 123 (2018), pp. 109–123. ISSN: 0360-1315. DOI: https://doi.org/10.1016/j.compedu.2018.05.002. URL: https://www.sciencedirect.com/science/article/pii/S0360131518301027.

[9] Sreemoyee Chatterjee and Suprovab Mandal. "A novel comparison of gauss-seidel and newton-raphson methods for load flow analysis". In: *2017 International Conference on Power and Embedded Drive Control (ICPEDC)*. 2017, pp. 1–7. DOI: 10.1109/ICPEDC.2017.8081050.

[10] Simon Tindemans. *Hands-on Power Grid Visualisation*. EE3L11 (BAP) Proposal. Intelligent Electrical Power Grids group, Digital Energy Studio. TU Delft, 2023. URL: mailto:s.h.tindemans@tudelft.nl.

[11] Manuel Reta-Hernández. "Transmission Line Parameters". In: *Transmission Line Parameters*. Taylor & Francis Group, LLC, 2006, pp. 13-1–13-28.

[12] W. Zomerdijk et al. "Open Data Based Model of the Dutch High-Voltage Power System". In: *Proceedings of the 2022 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*. Vol. 2022-October. IEEE, 2022, pp. 1–6. DOI: 10.1109/ISGT-Europe54678.2022.9960703. URL: https://doi.org/10.1109/ISGT-Europe54678.2022.9960703.

[13] James Kirtley. *Introduction To Per-Unit Systems*. 2021. Chap. 3.7. URL: https://eng.libretexts.org/Bookshelves/Electrical_Engineering/Book%3A_Electrical_Engineering_(Johnson)/3%3A_Introduction_to_Power_Systems/3.7%3A_Introduction_To_Per-Unit_Systems.

[14] Godot Engine. *GDExtension C++ Example*. https://docs.godotengine.org/en/stable/tutorials/scripting/gdextension/gdextension_cpp_example.html. Documentation tutorial.

[15] Godotengine. *GitHub - godotengine/godot-cpp: C++ bindings for the Godot script API*. URL: https://github.com/godotengine/godot-cpp.

[16] NangiDev. *GDSerCommPlugin: A Godot plugin to read Arduino serial input*. 2024. URL: https://github.com/NangiDev/GDSerCommPlugin.

[17] Microsoft. *System.IO.Ports Namespace*. 2024. URL: https://learn.microsoft.com/en-us/dotnet/api/system.io.ports?view=net-8.0.

[18] Ayush Gupta. *Using The Right File Format For Storing Data*. Data Science Blogathon - 11. 2022. URL: https://www.analyticsvidhya.com/blog/2021/09/using-the-right-file-format-for-storing-data/.

[19] *Pandapower—An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems*. Nov. 2018. URL: https://ieeexplore.ieee.org/abstract/document/8344496.

[20] "Towards a Survey of Visualization Methods for Power Grids." In: *arXiv: Human-Computer Interaction* (2021).

[21] Cyber Resilient Energy Delivery Consortium Illinois. *The Power Grid*. https://credc.mste.illinois.edu/applet/pg. Interactive applet.

[22] DIgSILENT. *PowerFactory Software for Power Transmission Systems*. https://www.digsilent.de/en/power-transmission.html. Software.

[23] Jacek Klucznik et al. "Nonlinear secondary arc model use for evaluation of single pole auto-reclosing effectiveness". In: *COMPEL - The international journal for computation and mathematics in electrical and electronic engineering* 34.3 (2015), pp. 647–656. DOI: 10.1108/COMPEL-10-2014-0252. URL: https://doi.org/10.1108/COMPEL-10-2014-0252.

[24] TenneT. *TenneT Power Flow Simulator*. https://netztransparenz.tennet.eu/fileadmin/user_upload/Our_Key_Tasks/Innovations/loadflow/index.html. Interactive model.

[25] Joran Kroese. *BAPPY*. Accessed: 2024-06-14. 2024. URL: https://github.com/johankrusey/BAPPY.

[26] Gopal Dahale. *Load Flow Analysis*. https://github.com/Gopal-Dahale/Load-flow-analysis. GitHub repository. 2023.

[27] James Kirtley. *5.3: Gauss–Seidel Iterative Technique*. 2024. URL: https://eng.libretexts.org/Bookshelves/Electrical_Engineering/Electro-Optics/Introduction_to_Electric_Power_Systems_(Kirtley)/05%3A_Introduction_to_Load_Flow/5.03%3A_Gauss-Seidel_Iterative_Technique.

[28] Ali M. Eltamaly and Amer Nasr A. Elghaffar. "Load Flow Analysis by Gauss-Seidel Method: A Survey". In: *International Journal of Mechatronics, Electrical and Computer Technology (IJMEC)* (2016).

[29] Avery N. Gilbert, Alan J. Fridlund, and Laurie A. Lucchina. "The color of emotion: A metric for implicit color associations". In: *Food quality and preference* 52 (Sept. 2016), pp. 203–210. DOI: 10.1016/j.foodqual.2016.04.007. URL: https://www.sciencedirect.com/science/article/pii/S0950329316300799#s0035.

[30] Kiran. Ajani et al. *Declutter and Focus: Empirically Evaluating Design Guidelines for Effective Data Communication*. Oct. 2022. URL: https://ieeexplore.ieee.org/abstract/document/9385921.

[31] PAVEL REŽNÝ. *Memory management in C++*. 2022. URL: https://is.muni.cz/th/wbbh0/Memory_management_in_C_Archive.pdf.

[32] Godot Engine Community. *Godot Engine Documentation*. 2024. URL: https://docs.godotengine.org/en/stable/.