POWER SYSTEM DEMONSTRATOR IN 3D

Project Report

Submitted by

Aishwarya Krishna Reshma - 1112334

Ananthu Upendran -1112348

Saraswathi Geetharadha Umesh -1112277

Shilpa Ann Babu -1112353

Varun Bompally -1112355

in partial fulfilment of the requirements for the degree of

International Master of Science

of

DARMSTADT UNIVERSITY OFAPPLIED SCIENCES

Under the guidance of

Prof. Dr.-Ing. Athanasios Krontiris

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbeit
FACHBEREICH ELEKTROTECHNIK
UND INFORMATIONSTECHNIK

July 2022

ACKNOWLEDGEMENT

We would like to extend our gratitude to Prof. Dr.-Ing. Athanasios Krontiris for providing us a platform to exhibit our skills and knowledge to the best of our abilities to complete and to present this project that made us do a lot of research on the topic thereby expanding our knowledge which in-turn will enable us to take our skills to greater heights.

We would also like to thank Darmstadt University of Applied Sciences and Department of Electrical Engineering and Information Technology or giving us this opportunity to present this project. .

Last but not the least we would like to thank our parents, friends and colleagues who have directly or indirectly guided us, for their continuous support and encouragement during our project.

Contents

Table of Figures

1. INTRODUCTION

1.1. Motivation for the project

A new era of energy provision is sweeping the globe. Continuous increase in both urbanization and industrialization are driving energy demand and energy production. HVDC power systems seem ideal for long-distance, point-to-point power transfers.

Power System Demonstrator in [3D] helps to give a better insight to its audience by providing a real-time, 3D environment for visualizing power flow analysis in an interconnected network. The main objective to develop Power System Demonstrator in 3D is to use an open-source power system analyst as the power flow solver in the backend of the application.

In the creation of the application, Unity, is used. Unity gives the possibility to cross platform apps for a 3D game. The programming language used for programming Unity is C#. Among the available open-source power grid analysis tool, Panda power is used. Panda power Supplier provides the steady-state power flow solver available.

In our project we basically dealt with the HV case and scenarios are built accordingly. In addition to that 3D model development is also made possible to make the environment quite attractive and user friendly.

Grid Stability App is designed to provide an informative app for the enhancement in reliability and stability in power grid. It is not available for Android or iPhone. It is programmed with Unity, a 3D computer-graphics game development platform that allows for easier creating 3D-applications. Additionally, as a suggested scenario, it was specified to facilitate the reliability of the power system by increasing the rates of electricity transmitted with HVDC technique.

This network involves various loads such as residential, industrial, locomotives, etc. Along with that, we have various generating substation representations like HVDC, thermal power plants, hydropower plant, and biomass.

2. OVERVIEW

With increasing population and own energy demand continuously increasing and the lack of high-voltage equipment, power quality and efficiency are difficult to be maintained. HVDC technology was developed decades ago for long distance power transmission, but the demand for HVDC networks has been continuously increased due to the rise of microgrid, asynchronous industries, and utility reform. HVDC systems can be utilized in power distribution networks to provide reliability, redundancy as reliable source of power, and effective maintenance. Therefore, this section gives an overview about HVDC system and highlight several HVDC technologies can effectively solve vital challenges of power system.

2.1. HVDC System

Comparing HVDC technology with alternating current (AC) electricity transmission looks very similar. However, HVDC technology is a proven technology used around the world with successful projects in operation. HVDC also provides unique benefits. Some engineering applications like submarine transmission/distribution power systems are also other examples.

HVDC system is a high-voltage electric power system which uses direct current for the transmission of electrical power. HVDC system mainly consists of two converter stations at either ends of the HVDC link. At sending station, the converter transformer takes the electric power from AC network, steps the AC voltage up to the required level, rectifies it to DC and transmit it through the line. At the receiving point the inverter terminal converts DC to AC power and inject it into the receiving AC network. One other application of DC link is pint-to-point connection that is applied between two areas with two different frequencies. HVDC links also play a significant role for integration of renewable energy sources especially offshore wind farms, into the power networks.

2.1.1. Applications of HVDC System

- Wires running longer than about 25 miles (40 km) might have to use HVDC power transmission to avoid losses and resistance. AC cable characteristics like inductivity and capacitance dissipate available energy, indicating a maximum useful transmission distance.
- HVDC lines have less losses than the AC lines have in practice.
- HVDC systems can move hundreds of thousands of kilowatts of power over the same size wire as compared to AC systems that can move a maximum of only a few kilowatts
- HVDC can either prevent a faulty AC grid from damaging a connected AC grid, or alternatively, it can serve as a firewall that protects a connected AC grid from damage caused by a faulty AC grid.

- HVDC systems are capable of transmitting power in longer links, feature significantly lower total losses that conventional HV AC lines. Losses during HVDC transmission are around three percent, including cable and converter station losses.

2.1.2. Layout and working of HVDC system

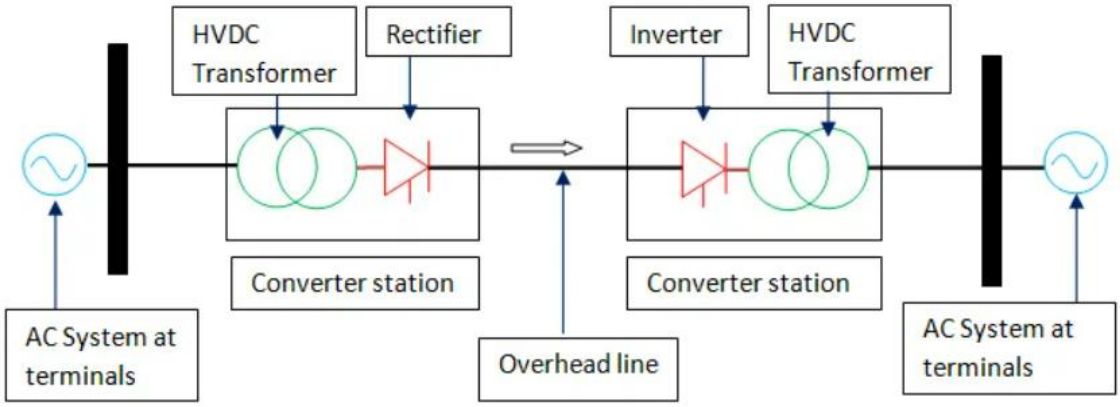

Figure 1: General Layout of HVDC system

AC power is generated in the generating station. This should first be converted into DC. The conversion is done with the help of rectifier. The DC power will flow through the overhead lines. At the user end, this DC must be converted into AC. For that purpose, an inverter is placed at the receiving end.

Thus, there will be a rectifier terminal in one end of HVDC substation and an inverter terminal in the other end. The power of the sending end and user end will be always equal (Input Power = Output Power).

HVDC has been efficiently used to connect unsynchronized systems with different frequencies while transmitting high amount of power over the distance.

The components of the HVDC Transmission system and its function are:

Converters: The AC to DC and DC to AC conversion are done by the converters. It includes transformers and valve bridges.

Smoothing Reactors: Each pole consist of smoothing reactors which are of inductors connected in series with the pole. It is used to avoid commutation failures occurring in inverters, reduces harmonics and avoids discontinuation of current for loads.

Electrodes: They are actually conductors which are used to connect the system to the earth.

Harmonic Filters: It is used to minimize the harmonics in voltage and current of the converters used.

DC Lines: It can be cables or overhead lines.

Reactive Power Supplies: The reactive power used by the converters could be more than 50% of the total transferred active power. So, the shunt capacitors provide this reactive power.

AC Circuit Breakers: The fault in the transformer is cleared by the circuit breakers. It also used to disconnect the DC link.

2.2. Basic design layout for the power system

2.2.1. Single line diagram of the grid

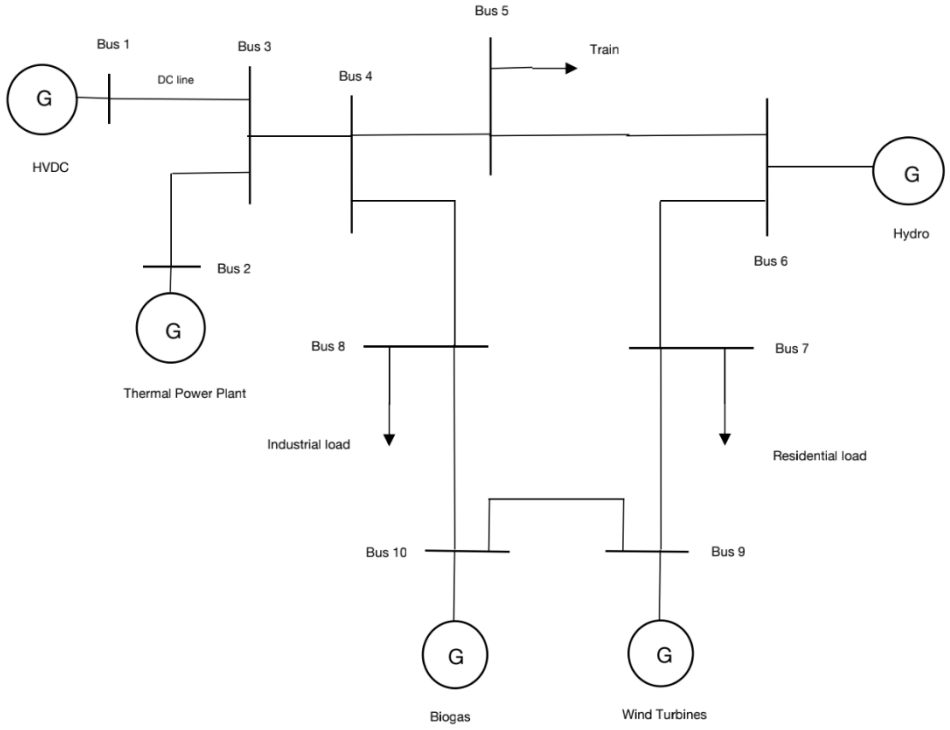

Figure 2: Single line diagram for the power system

The above figure 2 represents the single line diagram of grid which is the foundation for this project. As we can see in the figure, we have different types of generating stations, loads and an HVDC in the grid. They are listed below:

2.2.2. Generation Units:

- Thermal Power plant
- Wind Turbines
- Hydro Power plant

- Biogas Power plant
- HVDC Substation

2.2.3. Loads:

- Electric train
- Industrial load
- Residential load

In this power grid we have a total of 10 busbars named as Bus 1 to Bus 10 as well as 9 transmission lines between them.

The power output of generating stations and power rating of loads is listed below:

- Thermal Power plant - 400 MW
- Wind Turbines - 350 MW
- Hydro Power plant - 150 MW
- Biogas Power plant - 200 MW
- HVDC Substation - 100 MW
- Electric train - 550 MW
- Industrial load - 400 MW
- Residential load - 250 MW

3. POWER FLOW ANALYSIS

Power flow, or load flow, is widely used in power system operation and planning. The power flow model of a power system is built using the relevant network, load, and generation data. Outputs of the power flow model include voltages at different buses, line flows in the network, and system losses. These outputs are obtained by solving nodal power balance equations. Since these equations are nonlinear, iterative techniques such as the Newton-Raphson, the Gauss-Seidel, and the fast-decoupled methods are commonly used to solve this problem.

3.1. Need for power flow analysis

The objective of a power flow study is to calculate the voltages (magnitude and angle) for a given load, generation, and network condition. Once voltages are known for all buses, line flows and losses can be calculated. The starting point of solving power flow problems is to identify the known and unknown variables in the system.

In this network we used Newton- Raphson's method for the power low calculations of the buses and systems in the network developed.

Each bus in the system has two variables and based on these variables the buses are classified into three types mainly:

Slack Bus: The slack bus is required to provide the mismatch between scheduled generation the total system load including losses and total generation. The slack bus is commonly considered as the reference bus because both voltage magnitude and angles are specified; therefore, it is called the swing bus.

Generator Bus: The real power and voltage are specified for buses that are generators. These buses have a constant power generation, controlled through a prime mover, and a constant bus voltage. The generator buses are called regulated or PV buses because the net real power is specified and voltage magnitude is regulated.

Load Bus: Most of the buses in practical power systems are load buses. Load buses are called PQ buses because both net real and reactive power loads are specified.

For PQ buses, both voltage magnitudes and angles are unknown, whereas for PV buses, only the voltage angle is unknown.

3.1.1. Newton Raphson's Method

The power flow problem can also be solved by using Newton-Raphson method. In fact, among the numerous solution methods available for power flow analysis, the Newton-Raphson method is the most sophisticated and important. Many advantages are attributed to the Newton-Raphson (N-R) approach.

The N-R method is recent, needs less number of iterations to reach convergence, takes less computer time hence computation cost is less and the convergence is certain. The N-R method is more accurate, and is insensitive to factors like slack bus selection, regulating transformers etc. and the number of iterations required in this method is almost independent of the system size.

This method begins with initial guesses of all unknown variables (voltage magnitude and angles at Load Buses and voltage angles at Generator Buses). "Next, a Taylor Series (the Taylor series of a function is an infinite sum of terms that are expressed in terms of the function's derivatives at a single point.) is written, with the higher order terms ignored, for each of the power balance equations

included in the system of equations. The result is a linear system of equations that can be expressed as:

$$\begin{matrix}\Delta\theta \\ \Delta V\end{matrix} = \frac{1}{J} * \begin{bmatrix}\Delta P \\ \Delta Q\end{bmatrix}$$

Where ΔP and ΔQ are called the mismatch equations. ∂

And J is a matrix of partial derivatives known as a Jacobian:

$$J = \begin{matrix}\frac{\partial P}{\partial \theta} & \frac{\partial \Delta P}{\partial |V|} \\ \frac{\partial \Delta P}{\partial \Delta Q} & \frac{\partial \Delta Q}{\partial |V|}\end{matrix}$$

To determine the next guess (m + 1) of voltage magnitude and angles the linearized system of equations is solved based on:

$\theta m+1 = \theta m + \Delta\theta$

$|V|m+1 = |V|m + \Delta|V|$

The process continues until a stopping condition is met. A common stopping condition is to terminate if the norm of the mismatch equations is below a specified tolerance.

3.2. Pandapower

Panda power is an open-source Python-based tool for static analysis and optimization of three-phase AC and DC power flow problems. It uses the Pandas library to load and analyze data, and the Newton Raphson Power Flow Solver called the PYPOWER. It can be used in the power system analysis and for the optimisation in distribution and sub-transmission networks. It provides power flow, optimal power flow, state estimation, topological graph searches and short circuit calculations.

Network Models: The most used apporach in any power system analysis function is the bus-branch model (BBM). Here the network is defined as a collection of buses which will be interconnected by generic branches. The BBM can also be freely parametrized and is not bound to specific models of electric utilities. On the other hand, the user needs to calculate the impedances for each branch and summed power injections at each bus manually from the nameplate data of the grid elements. This can be cumbersome and error-prone especially for complex elements, like transformers with tap changers or more than two windings. Instead of a BBM, pandapower uses an element-based model (EBM) to model electric grids. Here, there are separate models for lines, two-winding, three-winding

transformers etc. This allows defining the network with nameplate parameters, such as length and relative impedance for lines, or short circuit voltage and rated apparent power for transformers.

Data Structure: Pandapower is based on a tabular data structure, where every element type is represented by a table that holds all parameters for a specific element and a result table which contains the element specific results of the different analysis methods. The tabular data structure is based on the Python library pandas. A pandapower network (abbreviated as net) is a python dictionary that holds all the information about the network. Most importantly, it includes an element and result table for each element. Results Table for each element type, such as line, transformer, switch etc will be present. Element table includes all parameters specified by the user. The result table is used by the power flow functions to store results. Input and output parameters have indices and should be same in both the table.

Electric Element Models: The pandapower library includes many different electric models, some of which are not available in any other open-source tool. Some of the electrical models are listed:

- Bus (bus): They represent the nodes of network. They are defined by nominal voltage $bus.vn\_kv$[1](Reference voltage in per unit system). The rated power for the per unit system is defined system wide with the parameter $net.sn\_kva$. The voltage magnitude res_bus.vm_pu and angle res_bus.va_degree
- Load (load): They are defined by the active power $load.p\_kw$ and reactive power $load.q\_kvar$. The percentage of the load which consumes a constant current is defined by the parameter $load.const\_i\_percent$, the constant impedance part is defined by the parameter $load.const\_z\_percent$. The load model includes a scaling factor $load.scaling$ that allows to scale the load.
- Static Generator (sgen): tatic generators are used to model constant power injection with active power $sgen.p\_kw$ and reactive power $sgen.q\_kvar$. The static generator model includes a scaling factor $sgen.scaling$ equivalent to the load scaling factor.
- Voltage Controlled Generator (gen): Generator elements are used to model voltage-controlled power generation units with a fixed active power injection $gen.p\_kw$ and a voltage magnitude set point gen.vm_pu. The reactive power $res\_gen.q\_kvar$ is

then calculated so that the voltage magnitude is equal to the set point. Reactive power limits $gen.q\_min\_kvar$ and $gen.q\_max\_kvar$ can be enforced in the power flow.

- External Grid (ext_grid): The external grid element model represents a voltage source with a voltage magnitude $ext\_grid.vm\_pu$ and the corresponding voltage angle $ext\_grid.va\_degree$.
- Shunt (shunt): Shunts are defined by a reactive power $shunt.q\_kvar$ and an active power $shunt.p\_kw$ value, which represents the losses. The power values equal the consumption at rated voltage $shunt.vn\_kv$. The parameter $shunt.step$ allows to model a discretely segmented shunt, such as a switchable capacitor bank.
- Line (line): The electric parameters of a line are specified relative to the length of the line $line.length\_km$. The longitudinal impedance is defined by the resistance $line.r\_ohm\_per\_km$ and reactance $line.x\_ohm\_per\_km$. The shunt admittance is defined by the capacity $line.c\_nf\_per\_km$. The line current $res\_line.i\_ka$ is calculated as the maximum current at both ends of the line. The line loading $res\_line.loading\_percent$ can be calculated as a ratio of line current $res\_line.i\_ka$ to the maximum thermal line current $line.max\_i\_ka$. A derating factor $line.df$ can be defined to consider the fact that some lines might not be utilized to their full thermal capacity. The model also provides a parameter $line.parallel$ to define the number of parallel lines.
- Impedance (impedance): An impedance element connects two buses with a per unit impedance in relation to the rated power $impedance.sn\_kva$. The forward impedance is defined by $impedance.rft\_pu$ and $impedance.xft\_pu$, the backward impedance by $impedance.rft\_pu$ and $impedance.xft\_pu$.

Power Flow: The pandapower power flow solver is predicated on the Newton-Raphson method. The implementation was originally based onPYPOWER, however has been improved with respect to robustness, runtime, and usefulness. Internal power flow parameters, such as node type for the power flow calculation (slack, PV or PQ node) or per unit conversions, are carried out automatically by pandapower. The BBM conversion is carried out before every power flow. However, if multiple subsequent power flows are performed for the same network that only differ in the nodal power injections, the conversion into a BBM becomes redundant. For this reason, pandapower offers the

possibility to reuse the BBM and the nodal point admittance matrix from previous power flow calculations. This feature can speed up applications like quasi-static time series simulations or heuristic power set point optimizations.

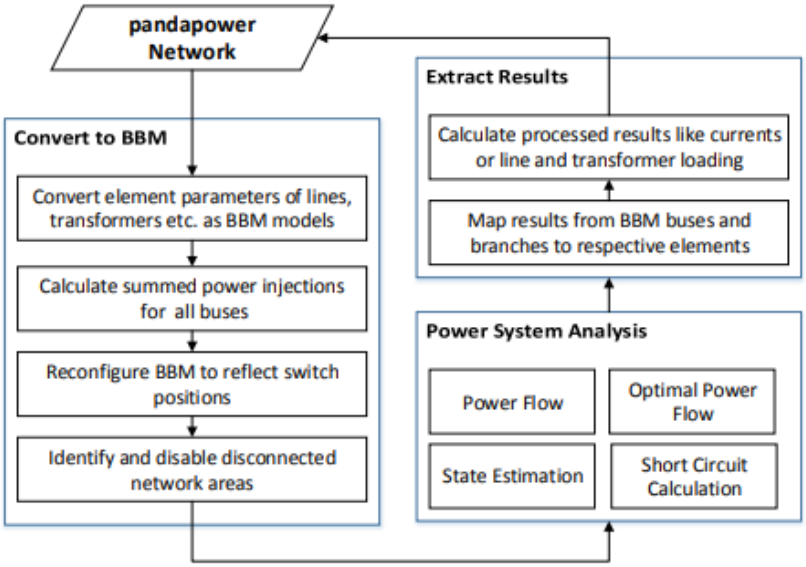

Figure 3: Electric Power Flow Analysis in Pandapower

3.3. Modelling Grid in Panda power

Panda power is an easy-to-use network calculation program aimed to automate the analysis and optimization of power systems. It uses the data analysis library pandas and is compatible with the commonly used MATPOWER / PYPOWER case format.

Firstly, we must model our grid in Panda power as we have seen in the single line diagram with the same power ratings and same type of transmission lines which we are using in the project. We used the standard lines which are available in the panda power library.

The model and line specifications are listed below:

→ Model - 490-AL1/64-ST1A 380.0
→ r_ohm_per_km - 0.059
→ x_ohm_per_km - 0.253
→ c_nf_per_km - 11.0
→ max_i_ka - 0.96
→ Type - ol

→ q_mm2 - 490

→ Alpha - 0.00403

Then we have started modelling the network in panda power by creating buses, lines, generating stations and loads as shown below:

3.3.1. Create Buses

We have created the buses as shown in the below figure 4 The voltage rating of all the buses is 380 kV.

```
#create buses
bus1 = pp.create_bus(net, vn_kv = 380, name = "Bus 1")
bus2 = pp.create_bus(net, vn_kv = 380, name = "Bus 2")
bus3 = pp.create_bus(net, vn_kv = 380, name = "Bus 3")
bus4 = pp.create_bus(net, vn_kv = 380, name = "Bus 4")
bus5 = pp.create_bus(net, vn_kv = 380, name = "Bus 5")
bus6 = pp.create_bus(net, vn_kv = 380, name = "Bus 6")
bus7 = pp.create_bus(net, vn_kv = 380, name = "Bus 7")
bus8 = pp.create_bus(net, vn_kv = 380, name = "Bus 8")
bus9 = pp.create_bus(net, vn_kv = 380, name = "Bus 9")
bus10 = pp.create_bus(net, vn_kv = 380, name = "Bus 10")
```

Figure 4: Creating Buses

3.3.2.Create Generators

```
#create generators
pp.create_gen(net, bus = bus2, p_mw = 400, max_p_mw = 700, min_p_mw = 100, vm_pu = 1.02, in_service = True, name = "The
pp.create_gen(net, bus = bus6, p_mw = 150, max_p_mw = 150, min_p_mw = -150, vm_pu = 1.02, in_service = True, name = "Hy
pp.create_gen(net, bus = bus10, p_mw = 200, max_p_mw = 300, min_p_mw = 100, vm_pu = 1.02, in_service = True, name = "Bi
pp.create_gen(net, bus = bus9, p_mw = 350, max_p_mw = 600, min_p_mw = 100, vm_pu = 1.02, in_service = True, name = "Wir
pp.create_gen(net, bus = bus1, p_mw = 100, max_p_mw = 500, min_p_mw = -500, vm_pu = 1.02, in_service = True, name = "HV
```

Figure 5: Creating Generators

Then the generators are created with their own power ratings and assigned to the buses. The power output of the generators can be changed manually by the user. So, the minimum and maximum power output are also added to the generators. The Thermal power plant and HVDC are kept as slack which means that the slack bus is used to balance the active and reactive power in the system while performing load flow studies.

The below are the minimum and maximum power ratings of generators:

→ Thermal Power plant - 100 MW & 700 MW

→ Wind Turbines - 100 MW & 600 MW

→ Hydro Power plant - -150 MW & 150 MW

→ Biogas Power plant - 100 MW & 300 MW

→ HVDC Substation - -500 MW & 500 MW

3.3.3 Create Lines

The lines are created between the buses and here we have used a standard type of line which we discussed above. Each line was given a specific length same as in the real time environment. This length of line effects in various ways like the capacitance, resistance and reactance of line increases with increase in length.

```
#create lines
line1 = pp.create_line(net, from_bus = bus2, to_bus = bus3, std_type = "490-AL1/64-ST1A 380.0", length_km = 10, name =
line2 = pp.create_line(net, from_bus = bus3, to_bus = bus4, std_type = "490-AL1/64-ST1A 380.0", length_km = 6, name = "
line3 = pp.create_line(net, from_bus = bus4, to_bus = bus5, std_type = "490-AL1/64-ST1A 380.0", length_km = 7, name = "
line4 = pp.create_line(net, from_bus = bus6, to_bus = bus5, std_type = "490-AL1/64-ST1A 380.0", length_km = 10, name =
line5 = pp.create_line(net, from_bus = bus4, to_bus = bus8, std_type = "490-AL1/64-ST1A 380.0", length_km = 6, name = "
line6 = pp.create_line(net, from_bus = bus6, to_bus = bus7, std_type = "490-AL1/64-ST1A 380.0", length_km = 9, name = "
line7 = pp.create_line(net, from_bus = bus10, to_bus = bus8, std_type = "490-AL1/64-ST1A 380.0", length_km = 6, name =
line8 = pp.create_line(net, from_bus = bus9, to_bus = bus7, std_type = "490-AL1/64-ST1A 380.0", length_km = 8, name = "
line9 = pp.create_line(net, from_bus = bus9, to_bus = bus10, std_type = "490-AL1/64-ST1A 380.0", length_km = 6, name =
```

Figure 6:Creating Lines

In the above figure 6 you can observe that there are a total of 9 lines connecting all the 10 buses and you can observe the length of each line.

3.3.4. Create DC line

In our network we have a HVDC substation and to connect that substation to rest of the network we require a DC line and you can see in the below figure 7

```
pp.create_dcline(net, from_bus = bus1, to_bus = bus3, p_mw = 100, loss_percent=1.2, loss_mw=10 ,vm_from_pu = 1.02, vm_t
```

Figure 7: Create dc line

As you can observe for DC line, we need to give the active power rating, loss percentage and active power losses.

3.3.5. Create Loads

In the end we have created loads with their power ratings and assigned to their respective buses as shown in the below figure 8

```
#create load
load1 = pp.create_load(net, bus = bus5, p_mw = 550, controllable = False, name = "Train")
load2 = pp.create_load(net, bus = bus8, p_mw = 400, controllable = False, name = "Industrial")
load3 = pp.create_load(net, bus = bus7, p_mw = 250, controllable = False, name = "Residential")
```

Figure 8:Create loads

You can observe that the loads in this case uncontrollable whereas in our project we can completely control the loads i.e., the user can change the power rating of loads and can keep the loads in service or out of service.

Once the modelling of network is done, we can run the power flow and analyse the simulation results as shown in the below figures.

```
pp.runpp(net)
net.res_bus
```

	vm_pu	va_degree	p_mw	q_mvar
0	1.020000	0.000000	-100.000000	0.000000
1	1.020000	0.000000	-413.499416	97.426733
2	1.020000	-0.420321	0.000000	0.000000
3	1.019353	-0.718035	0.000000	0.000000
4	1.018734	-0.980486	550.000000	0.000000
5	1.020000	-0.822839	-150.000000	-59.619851
6	1.019604	-0.798311	250.000000	0.000000
7	1.019209	-0.790460	400.000000	0.000000
8	1.020000	-0.582661	-350.000000	55.871383
9	1.020000	-0.630666	-200.000000	-27.407092

Figure 9: Bus result

```
net.res_line
```

	p_from_mw	q_from_mvar	p_to_mw	q_to_mvar	pl_mw	ql_mvar	i_from_ka	i_to_ka	i_ka	vm_from_pu	va_from_degree	vm_to_pu	va_to_degree
0	413.499416	-97.426733	-412.792617	95.265876	0.706799	-2.160857	0.632794	0.631038	0.632794	1.020000	0.000000	1.020000	-0.420321
1	501.592617	-54.412641	-500.993194	53.870002	0.599423	-0.542639	0.751531	0.751033	0.751531	1.020000	-0.420321	1.019353	-0.718035
2	379.408884	-37.992134	-379.009051	36.079292	0.399834	-1.912842	0.568336	0.567810	0.568336	1.019353	-0.718035	1.018734	-0.980486
3	171.110474	31.406561	-170.990949	-36.079292	0.119525	-4.672730	0.259136	0.260631	0.260631	1.020000	-0.822839	1.018734	-0.980486
4	121.584309	-15.877868	-121.548948	12.918868	0.035361	-2.959000	0.182760	0.182215	0.182760	1.019353	-0.718035	1.019209	-0.790460
5	-21.110474	28.213290	21.115348	-32.863112	0.004874	-4.649822	0.052487	0.058208	0.058208	1.020000	-0.822839	1.019604	-0.798311
6	278.634338	10.592216	-278.451052	-12.918868	0.183286	-2.326653	0.415340	0.415536	0.415536	1.020000	-0.630666	1.019209	-0.790460
7	271.350297	-36.007372	-271.115348	32.863112	0.234949	-3.144260	0.407733	0.406954	0.407733	1.020000	-0.582661	1.019604	-0.798311
8	78.649703	-19.864011	-78.634338	16.814876	0.015365	-3.049135	0.120832	0.119778	0.120832	1.020000	-0.582661	1.020000	-0.630666

Figure 10:Line result

va_to_degree	loading_percent
-0.420321	65.916080
-0.718035	78.284529
-0.980486	59.201670
-0.980486	27.149098
-0.790460	19.037483
-0.798311	6.063283
-0.790460	43.284963
-0.798311	42.472227
-0.630666	12.586617

Figure 11:Line loading percentage

```
net.res_load
```

	p_mw	q_mvar
0	550.0	0.0
1	400.0	0.0
2	250.0	0.0

Figure 12: Load result

```
net.res_gen
```

	p_mw	q_mvar	va_degree	vm_pu
0	413.499416	-97.426733	0.000000	1.02
1	150.000000	59.619851	-0.822839	1.02
2	200.000000	27.407092	-0.630666	1.02
3	350.000000	-55.871383	-0.582661	1.02
4	100.000000	0.000000	0.000000	1.02

Figure 13: Generator result

```
net.res_dcline
```

	p_from_mw	q_from_mvar	p_to_mw	q_to_mvar	pl_mw	vm_from_pu	va_from_degree	vm_to_pu	va_to_degree
0	100.0	-0.0	-88.8	-40.853235	11.2	1.02	0.0	1.02	-0.420321

Figure 14: DC line result

```
from pandapower.plotting.plotly import pf_res_plotly
import pandapower.plotting.plotly
pp.runpp(net)
pf_res_plotly(net)
```

```
No or insufficient geodata available --> Creating artificial coordinates. This may take some time
```

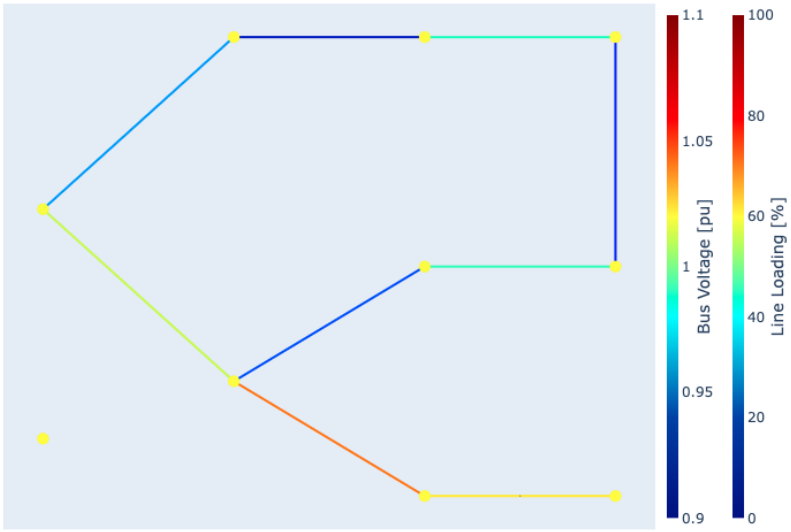

Figure 15: Plotting of network

4. UNITY MODEL: BASIC STRUCTURE

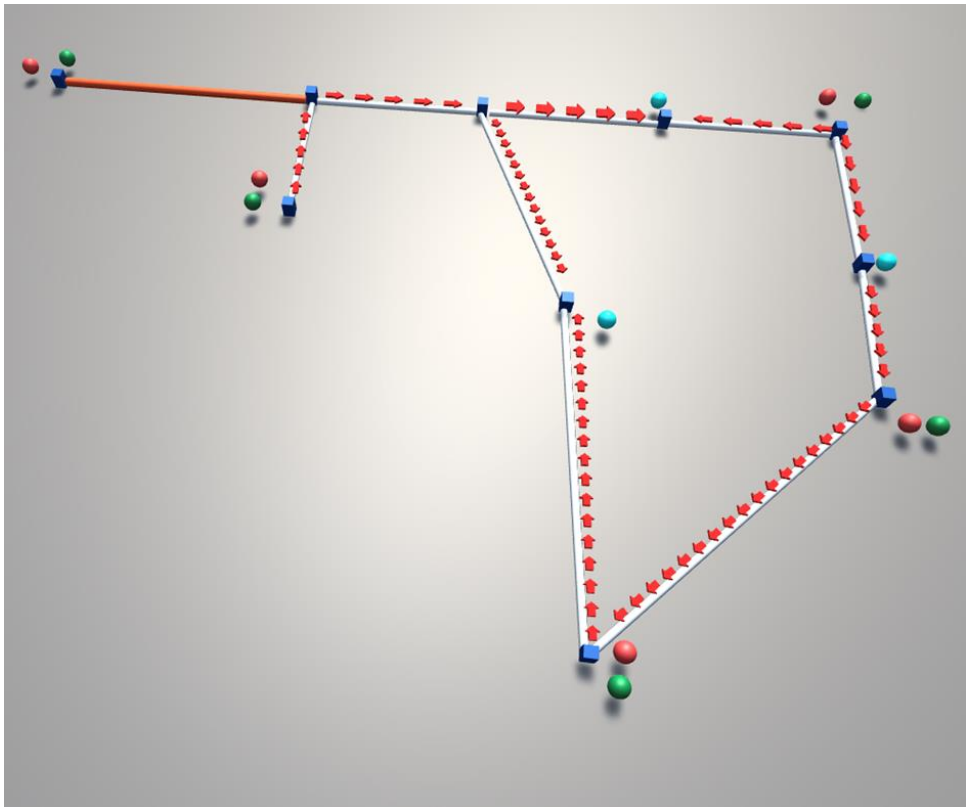

Figure 16: Basic Unity model

Firstly, we have implemented our single line diagram into unity model.

This implemented model has buses, PV and PQ generators, loads, HVDC lines, HVDC interconnectors and particle system.

Functions of various components of model are as follows:

PV and PQ generator:

These components mainly depict a generating station which can be a wind park, biomass plant and so on. From these points, power is generated and transmitted to loads via transmission lines. Here, PV generator is shown using red sphere and PQ by using green sphere.

Buses:

Buses are the components at which lines, loads and generators are connected via transmission lines. Power is fed from buses into the loads as well as is fed into the buses from generating plants. Buses are depicted by violet-coloured cubes.

Loads:

We are having three types of loads here:

a) Residential loads: These loads mainly comprise of homes, flats and so on.

b) Industrial loads: These loads are mainly industries, commercial buildings, factories and so on.

c) Electric Train: This load will be an electric locomotive which used electrical energy via electrical traction systems.

The loads are depicted by sky blue coloured spheres.

HVDC interconnectors and HVDC line:

High voltage DC line is mainly used for lossless long distance transmission purposes. HVDC interconnectors got Voltage source converters that can convert AC to DC and store the DC energy and can supply AC by converting stored DC to AC and transmitting it to loads whenever the loads require additional power supply. Also, HVDC interconnectors are used for attaining system stability whenever power outages, contingencies, faults or transient disturbances are occurred in the system.

4.1. Implementation steps:

For this implementation, we have done the following procedures:

1. Allocated sufficient base plane.

2. Inserted buses, lines, loads and generation points.

3. Connected the buses, generation plants and loads with transmission lines.

4. Used HVDC line to connect HVDC interconnector.

5. Imparted the particle system onto the lines.

6. Ensured that the (n-1) security rule has been maintained throughout the model.

4.1.1. Model details:

Following are the details of the implemented structure:

1. The model got one HVDC interconnection station and a HVDC line.

2. The model has 10 buses connected to several loads and generating stations via transmission lines.

3. It has Hydro, thermal, biogas and wind power generating stations.

4. It has 3 loads namely residential load, industrial load and an Electric traction system load or Electric train.

5. The structure also has nine AC transmission lines.

4.1.2. Panels and particle system:

The model also got particle system and Panels. Particle system mainly comprises of arrows which are flowing on transmission lines. The size of the arrows flowing on the lines depends on the loading percentage of the line or power flow through the line. For higher loading percentage of lines, thicker will be the arrows flowing and for lower loading percentage thinner will be the arrows.

The panel system consists of details of loads or generating station where user can interact by changing values and submitting thereby corresponding effect can take place in the model.

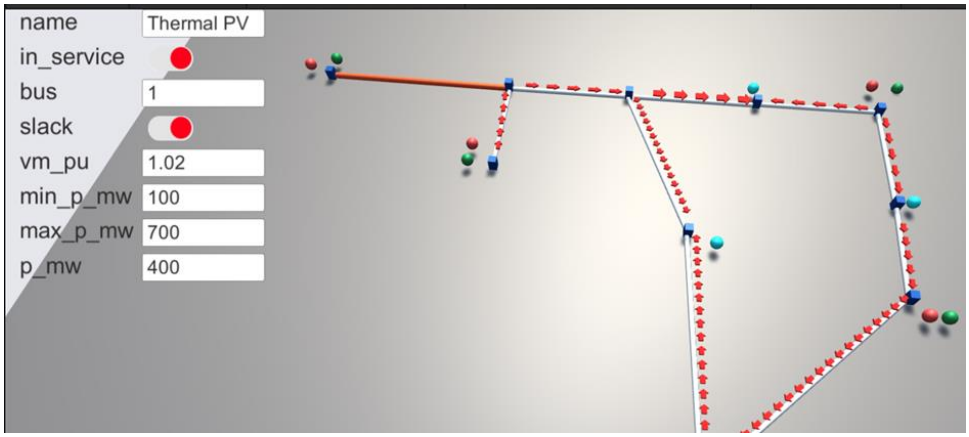

Figure 17: Generator panel

A generator panel consists of name, service status, bus detail, slack status, per unit voltage, minimum and maximum active power and active power supplying.

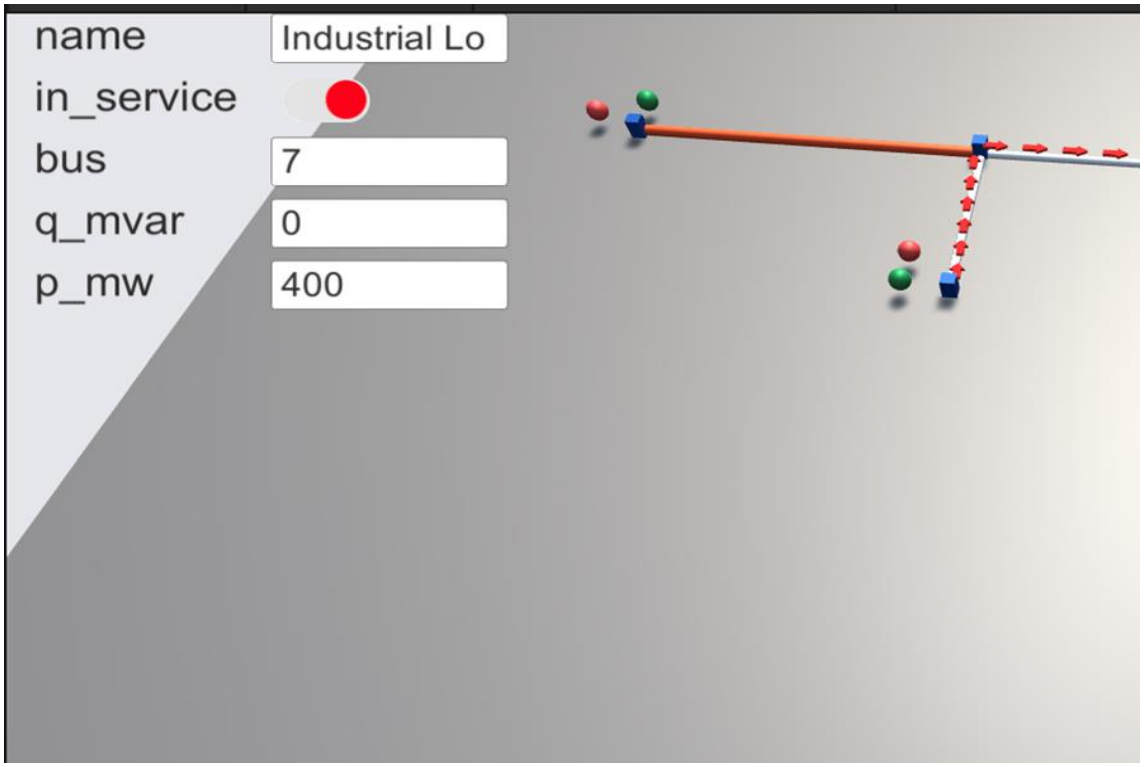

Figure 18: The load panel consists of load name, service status, and bus number, active and reactive power consumed by the load.

5. PARTICLE SYSTEM

The particle system used here are Arrows. These are implemented as prefab. As discussed above, based on loading percentage of lines, size of the arrows flowing on transmission lines varies.

5.1. Working of particle system:

Particle system works in accordance with the information given on the panels or sliders. Whenever user gives a value for active or reactive power on the panels of load or generators or chooses values for any attributes through sliding the sliders and finally by clicking the submit button, these values are transmitted to python panda power through the TCP communication channel and the power flow calculations are done by the panda power. Based on the calculated values returned to Unity from panda power, the loading percentage or power flow of the corresponding transmission lines have been changed. The particle system changes the simulation speed accordingly on these transmission lines. As a result, the user can see the power flow changes via change in the sizes of the particle system arrows flowing on the transmission lines which is in such a way that, lines having higher loading percentage got broader arrows while that with lower loading percentage has narrower arrows.

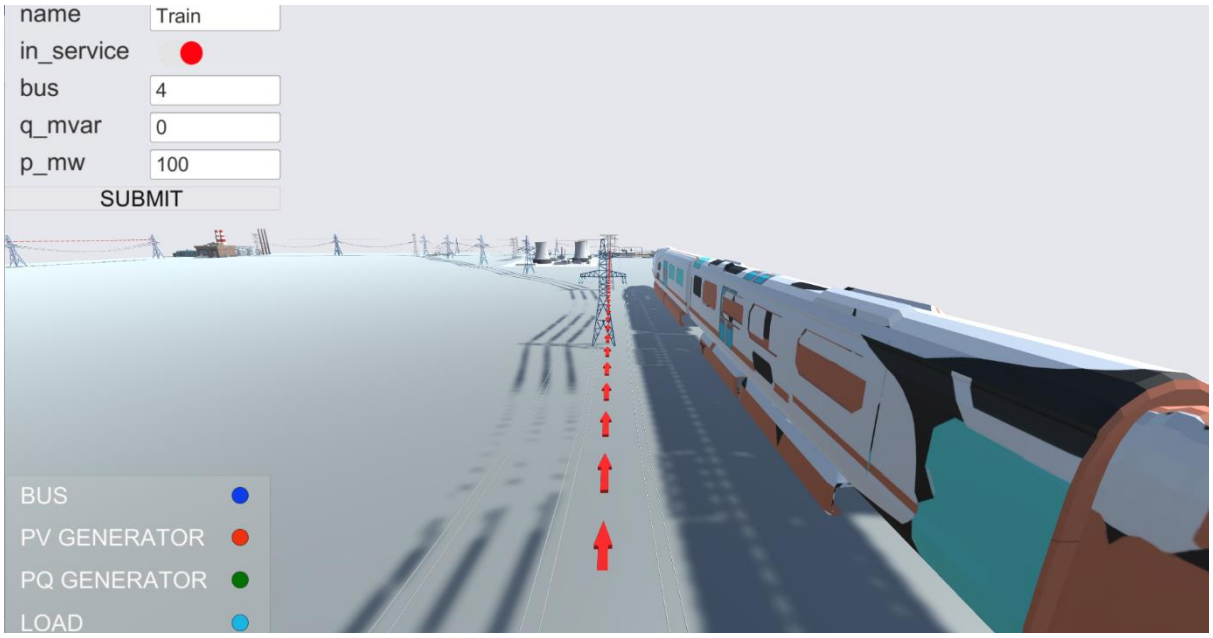

Figure 19: Particle system for train during small loading percentage

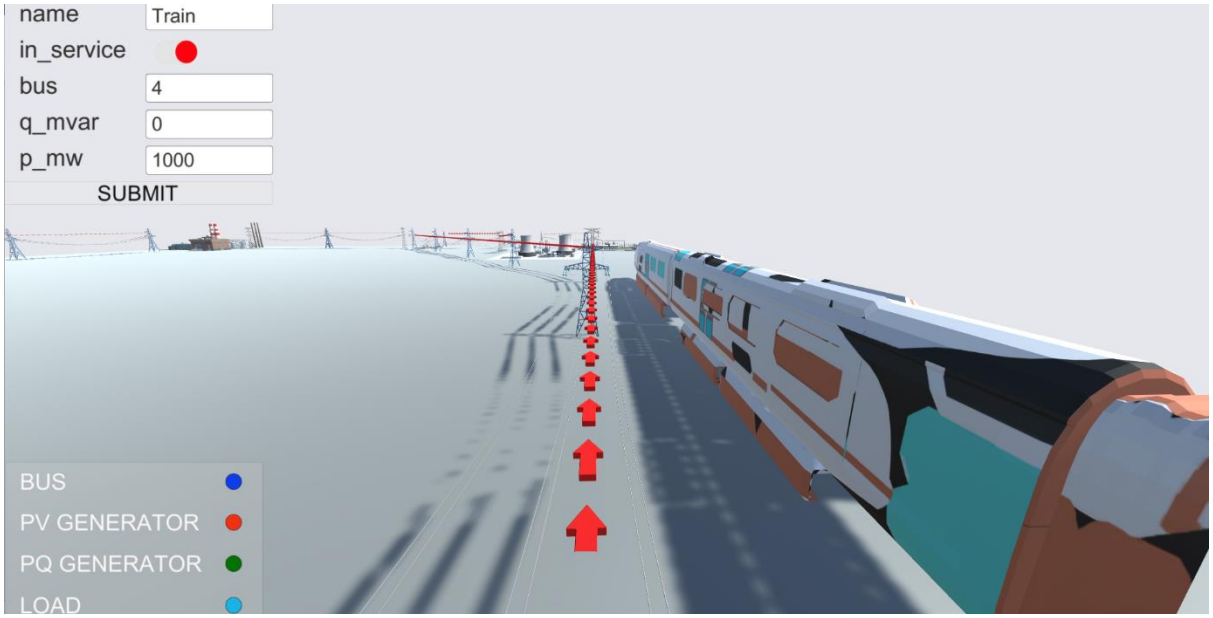

Figure 20: Particle system for train during large loading percentage

5.1.1. Difficulties faced and solutions obtained during particle system implementation:

1) When particle system is initially implemented into the model, we have not observed any changes in the simulation speed or arrows flow while changing the load or generator parameters. This is because of the reason that the simulation speed of particle system was not changing according to the user input. The problem is diagnosed to be with the server as the server is not starting for communication. Hence that problem rectified that by starting the server manually.

2) Also, in the new model the particle system was unable to attain sufficient start life time because of which the particles were not having sufficient flow length to reach from one point to another. The problem was detected with the communication system. Here the trouble is caused by the Line view script which was placed on the towers in the new model as well as in the AC lines of the old prototype model. Hence on running these two scripts are getting into a redundancy like situation between same line view scripts on AC lines and towers and limiting the arrow flow length. The problem was fixed by removing the duplicating line view script on towers and attaining the required arrow flow length.

6. INTEGRATING PANDAPOWER SOLVER AND UNITY

In Unity, C# is the default scripting language for game development. However, the solver chosen by for the application is the Pandapower solver, which is written in Python. The starting point of the project is to define a solution for communication between Unity and Pandapower that best suits the

purpose of the Power System Demonstrator application. There are two approaches introduced by Unity for communication with Python are In-Process API and Out-Of-Process API.

6.1. Embedding python into Unity

We used the Out-Of-Process API, because of the limitations in the In-Process API for communication between the python server and the unity client. In Out- Of-Process API the python scripts run in a separate process but are still linked with the Unity. In the process we can also integrate third parties. The communication occurs through a local host that is opened on a fixed port number. If we have two processes running in a system, then over IP we can use TCP communication for transporting the data packets. We can have a bi-directional connection between two end points. Whenever a new connection form they will be uniquely identified by a combination of client and server socket. In our local system we have the same IP address, send port and receiver port in a connection.

In our Power system demonstrator, we have the client and the server. The client in our case will be the Unity and the server will be Python.

1) When the game starts, in the unity environment the main network and elements will be created. Each element is packed to be sent in message.
2) Using TCP communication protocol each of these messages will be sent one by one to the python server.
3) A pandanetwork is created as soon as the data is received from unity in python. And power flow calculation is run, and the results are displayed in tables for buses and lines.
4) The values obtained from the server is sent to our client, i.e., unity as a message using the TCP communication protocol.
5) Based on the results we can visualise the system in Unity.
6) Python waits to see if there is any more message that is sent from the unity.
7) The above-mentioned process stops as soon as the server or the client stops.
8) The data from the client and server are send through the ports as a message.
9) Message can be regarded as the basic building block of data, and they cannot be sent all together at once, hence they need to be divided into similar structures of data when it is getting received at the destination.

We first write the codes for the network communication and once that is done, we use Python and C# programming language for the remaning functionalities.

6.1.1. Coding in Python

In python we have three scripts. One is for the network communication. Then we have class that is for the creation of the pandapower network. Another class is used to handle the sending and the reception of the data.

For the network communication we use a module in python that handles asynchronous programming. So, with asynchronous programming we can handle multiple threads. We also made use of event driven programming. With this type of programming, we can handle events and it can be easily recognised and handled by the program.

6.1.2. Starting the network connection

The connection mechanism between the unity and the python is started using the Network script. We need two classes for this functionality. They are:

- Server class: Checks if the main python program receives the signal and then runs the server. This class is defined with the port name and IP address of the server.
- Peer class: Peer can read from or write to the stream and is created when a connection is accepted. Characters are written to the stream by using a particular encoding.

6.1.3. Pandapower network in python

Creation of the pandapower network is done using the methods that are coded in Power_network script. In addition to the basic pandapower functions that we need for the creation of the network we need two additional methods. The first method is used to identify the changed electrical element in unity, that is basically done by returning the index of the changed element. And another method that is added to the class is to replace this changed value onto the table.

When we are running our network in play mode in unity and if we select any electrical element and change any of its parameters then a new message in JSON format is packed. This format basically has two parts. It has keys and values, and they are in a string format, together they are called the key value pair. Key is string enclosed in quotation marks and the values are either string, number, arrays, or any other object.

In pandapower each element is unique, and each element is listed with their properties in a table format.

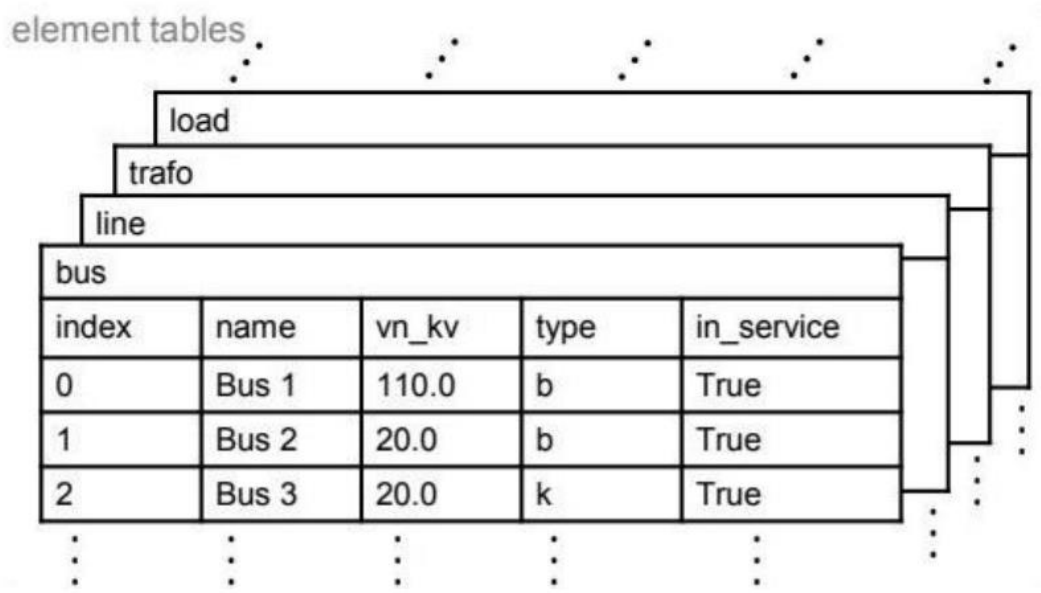

Figure 21: Elements in Pandapower

Some of the methods defined are:

- Get_element_index method: Whenever there is a change in any of the parameters of an electrical element in our power network, we can use this method to get the name of the changed element.
- Change_node method: The value after the change is updated to the correct electrical element when this method is invoked.
- Run_network method: In the play mode this method is invoked, and the power flow calculation is executed. This method is also executed each time any of the parameters of the electrical elements are changed during the play mode.

6.1.4. Sending data to Unity

Client class plays a huge role in converting the data that is understandable for both the unity and pandapower. When we have invoked the run_network method then the client class egts informed and it parses the message, i.e., it prepares a JSON format and the method json.load() is invoked, and this method will convert the JSON format to a from that is understandable for the pandapower.

6.1.5. Coding in Unity

Like the python sections in unity, we have three sections of codes. The first part is dedicated for the network connection to python. Second part of the codes basically deals with the data parsing and the conversion of the data from the pandapower to a format in C# that can be understood by unity. The final and the most important part is we need to integrate between the unity user interface and the back end pandapower.

6.1.6. Accepting Connection from Python

For accepting connections from python, we have a class named TcpServer class. This class has functions that enable unity to listen to any connections from python on the specified IP address and port. This has a built-in class, TcpListener and some additional functions. Events are invoked whenever there is a peer connection. An event informs the program that it can start transmitting the data between the peers.

7. DEFINING PANDAPOWER ELEMENTS IN UNITY

In order to create power grid in the Unity, the first aim is to develop the game object in the scene and then we assign these particular elements in the pandapower main library. After creating the game object and a material to the unity, next step is to add scripts to each of these objects so that it works in the run mode. To characterize the objects, we use different 3D elements for each factor of components in the power system. For instance in our model we used in the initial building of Unity model , a 3D square to denote the bus model and 3D sphere to denote the load and generators . Along with this , each of these objects should have all the variables that the bus, load and generators holds.

The next task is to add the script to each elements, so that it interacts with the user in the play mode. In the script element.cs , which is holding the electrical components definitions such as bus, loads, generators.

For example the definition of the class Bus is given below:

```
public class Bus
    {
        public int index;
        public string name;
        public float vn_kv;
        public bool in_service = true;

        [JsonIgnore]
        public BusResult BusResult = new BusResult();

    }
```

Figure 22: Definition of Class bus

To serialize and deserialize the data the name of the classes and the elements should be the same.

7.1. Interface between UI and Python

An interesting class Network Manager which includes all that exists in unity relating the UI in pandapower. This is the class on which network manager takes and updates predefined values of the resources. It takes all information of existing objects from the scene, send the data one by one to Python, and then when the result data coming back from Python, this class updates the old values with the new values.

NodeInspector.sc is a script that is used for controlling UI. In this script a System. Reflection is used that involves the parameters and collects information about element parameters, giving access to the data of a running program. Also, a Raycast system is applied. The Physics. Raycast method is used to detect objects with colliders in a specified ray.

The UI designed for the app has two advantages: It is automatically created and does not depend on the grid and can be used for different girds. There are a different number of buses and elements the require to appear in the scene which would fill a lot of space in the scene if it is not designed dynamically.

7.1.1. Scenario of slider in Unity

An additional scenario which we tried to implement is the slider to control the values and load parameters externally by the user.

A standard slider that can be moved between a minimum and maximum value.

The slider component is a Selectable that controls a fill, a handle, or both. The fill, when used, spans from the minimum value to the current value while the handle, when used, follow the current value.

The anchors of the fill and handle RectTransforms are driven by the Slider. The fill and handle can be direct children of the GameObject with the Slider, or intermediary **RectTransforms** can be placed in between for additional control.

When a change to the slider value occurs, a callback is sent to any registered listeners of Slider.onValueChanged.

7.1.2. Slider Implementation

Right click in the Hierarchy to create the UI element Canvas , That will spawn the EventSystem

- Create a Panel in the Canvas
- In the panel we need to add the slider and the texts required

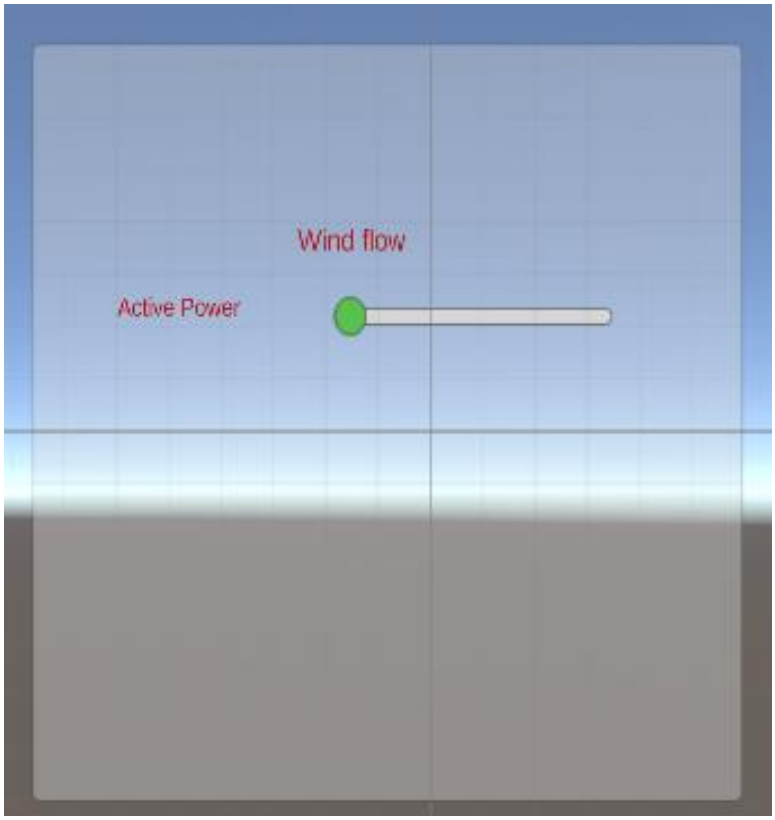

Figure 23: Slider development in Unity

- Next step is to add the scripts to the panel.
- For that , I added a script named ' open_panel ' in the scripts and it is depicted below:

```
public class open_panel : MonoBehaviour
{
    public GameObject Panel;

    public void OnMouseDown(){
        if(Panel != null)
        {
            Panel.SetActive(true);
        }
    }

}
```

Figure 24: Code for Panels

- Then next step is to add these panels to the loads that has to be controlled. Here we tried to implement the slider to the windmill generating stations.

- For that , initially we developed a script called 'Rotate_go' and is shown below:

```
void Update()
    {

        transform.Rotate(Vector3.back * speed * Time.deltaTime);
    }
    public void AdjustSpeed(float newSpeed){
        speed = newSpeed;
    }
```

Figure 25: Code for rotating the wind turbine

- Where, Vector3.back() is the Shorthand for writing Vector3Int(0, 0, -1).
- On implementing the slider value() and rotate function to the wind turbines , the user could control the rotation of wind turbines from 0 to a maximum value irrespective of the unknown electrical parameters.

8. UNITY 3D MODELS

The implementation of 3d models for representation of all generation, loads unity has been downloaded from different website.

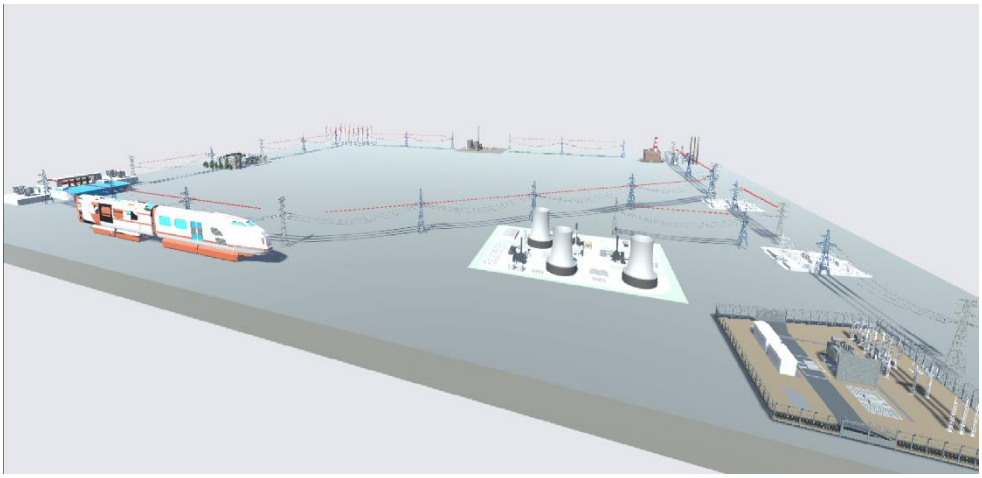

Figure 26: 3D Modelling in Unity

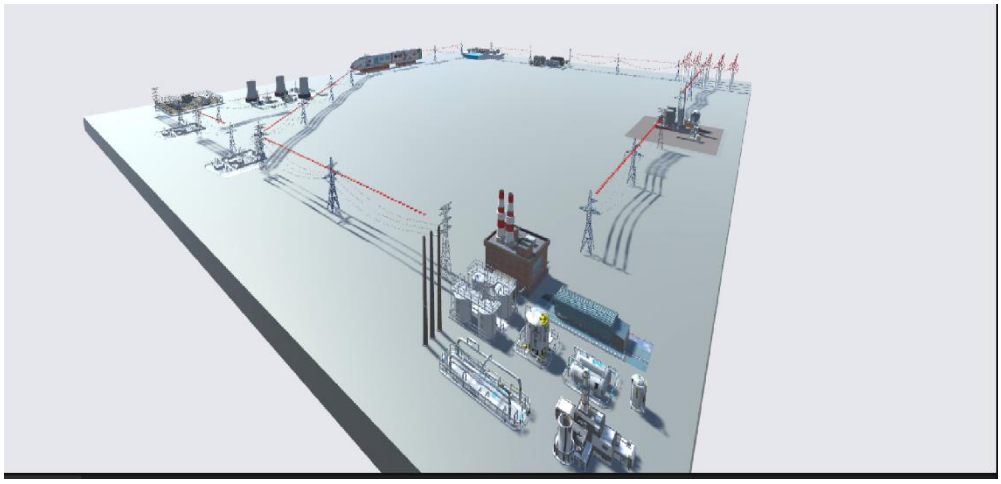

Figure 27: 3D model development in Unity

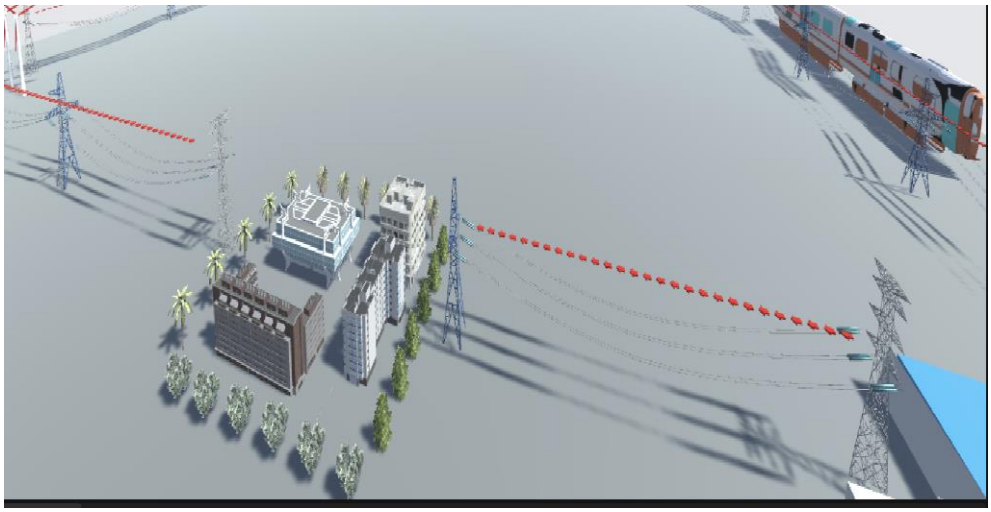

Figure 28: Power generating stations

8.1. Generation:

8.1.1. Wind power plant

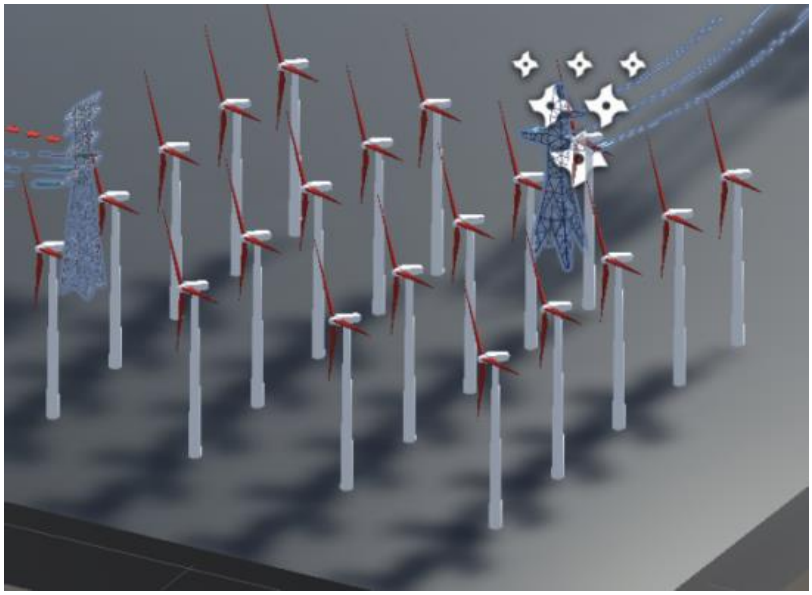

Figure 29: Windmills stations

As this module does not have animation affect for rotating blade during game mode We have given c# script for rotating blade as shown below.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class Rotation : MonoBehaviour
6   {
7       // Start is called before the first frame update
8       void Start()
9       {
10
11      }
12
13      // Update is called once per frame
14      void Update()
15      {
16          this.transform.Rotate(0, 0, 2);
17      }
18  }
19
```

Figure 30: C# script for the turbine rotation of blades

8.1.2. Hydro power plant

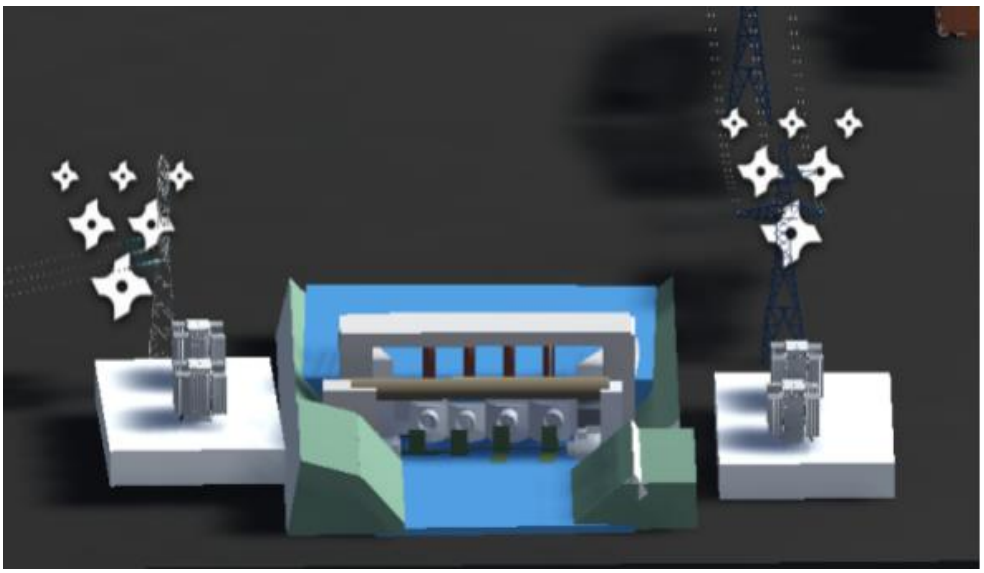

Figure 31: Hydro power plant

8.1.3. Thermal power plant

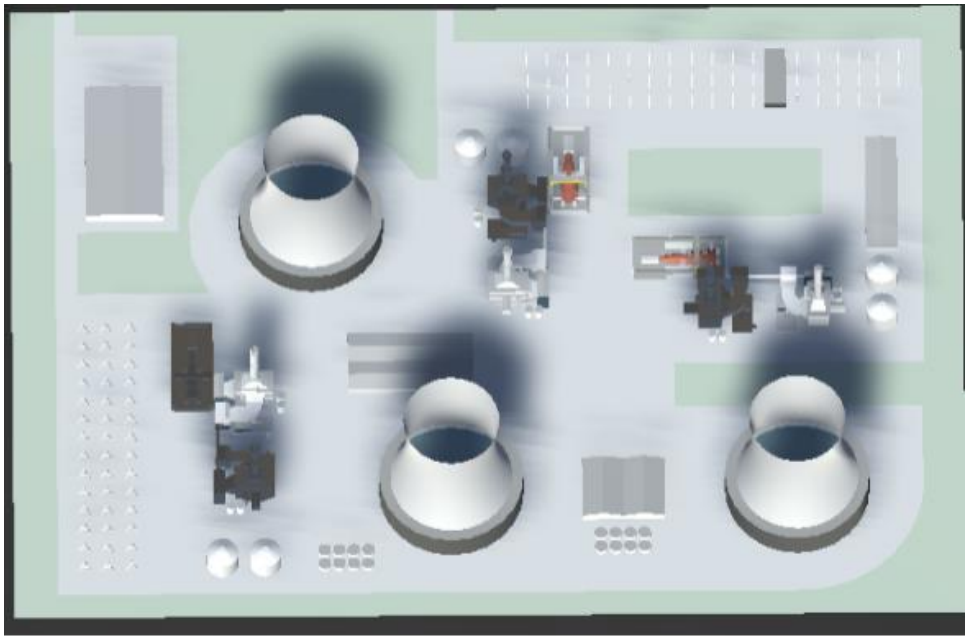

Figure 32: Thermal power plant

8.1.4. Biogas power plant

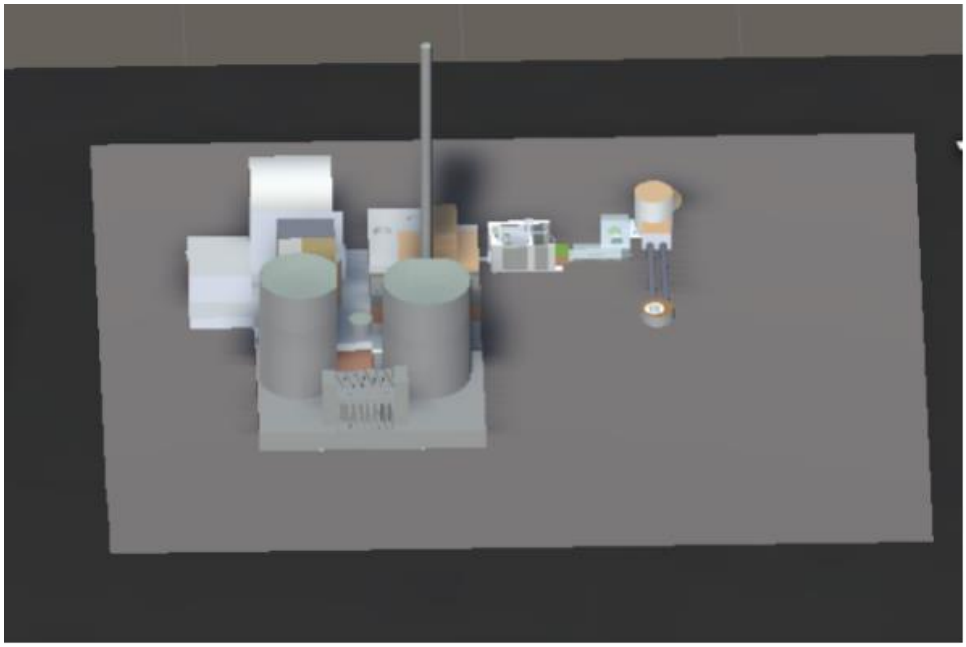

Figure 33: Biogas power plant

8.1.5. HVDC Station

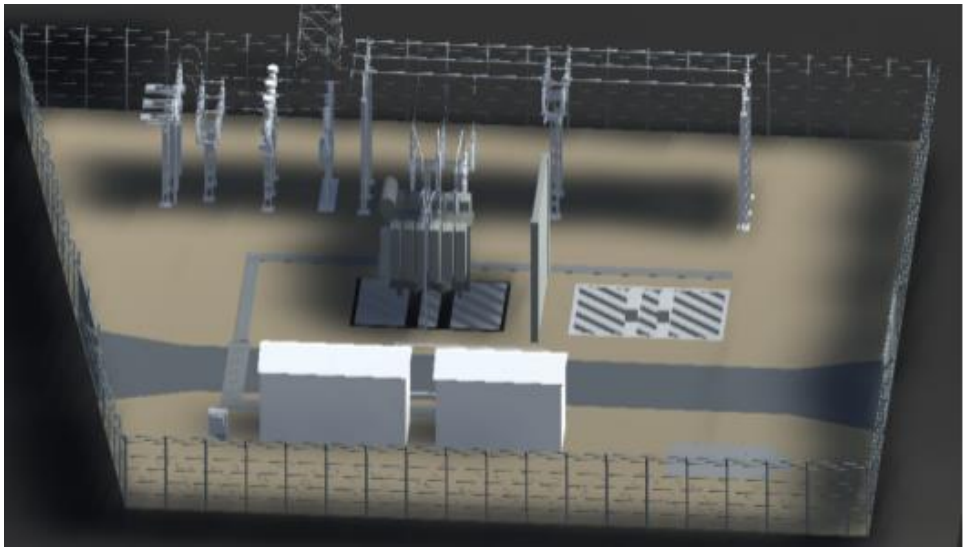

Figure 34:HVDC Station

8.2. Loads:

8.2.1. Residential loads

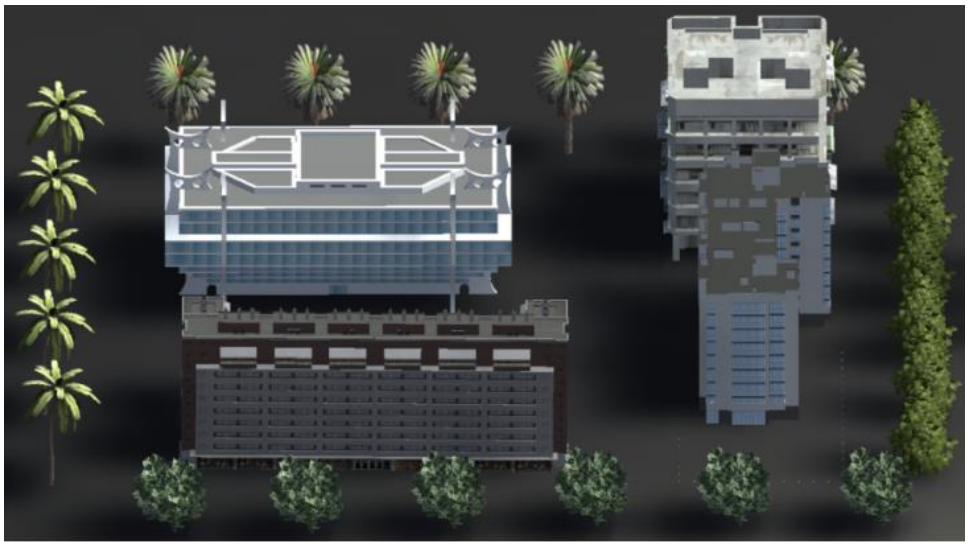

Figure 35: Residential loads

The residential load is created by using 3 to 4 residential buildings and all buildings are created as one game object in unity by using box collider.

8.2.2. Industrial loads

Figure 36: Industrial load

The industrial load also created as same as residential load by using box collider.

8.2.3. Passenger train

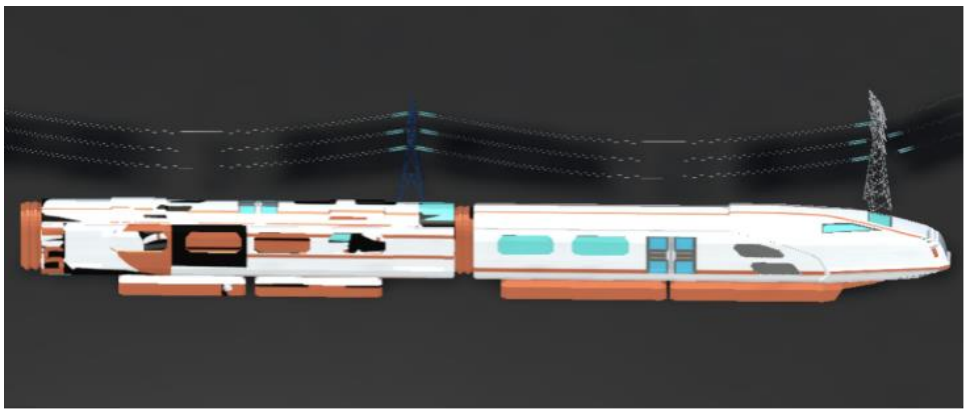

Figure 37: Passenger train load

8.2.4. Transmission line

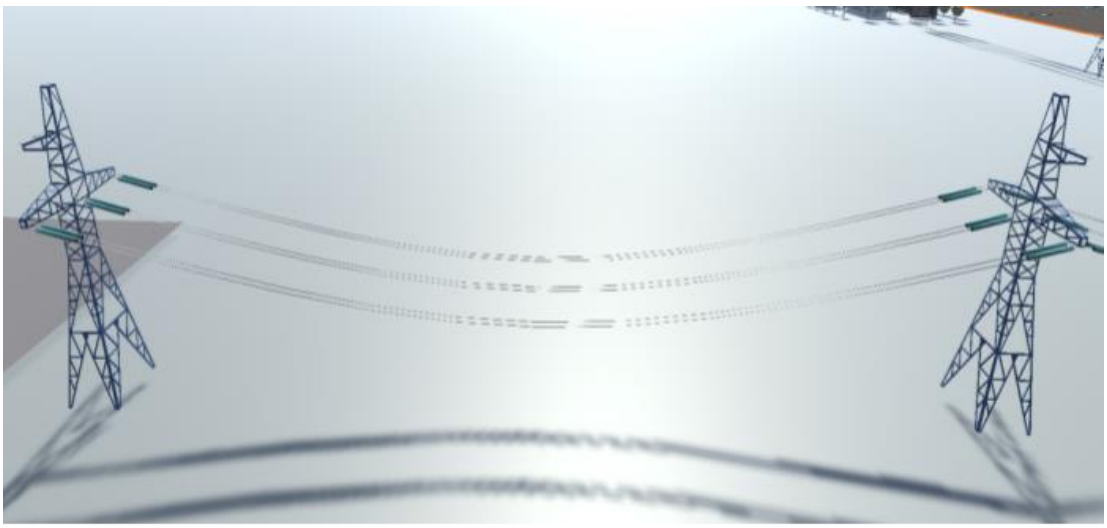

Figure 38: Transmission line

9. Importing 3d models in unity

In order to import 3d models into the unity, first we are importing a file with .obj or .fbx or .dae format in assets and then we have added a texture to each of our 3d model.

In next step, adding the respective c# script to the models i.e, buses, lines, generation and loads adding BusView, LineView, GeneratorView and LoadView scripts respectively and defining the properties that are defined for each element are accessible from inspector window as we have given in Pandapower. These properties are same as data frame in in Pandapower .As it can be seen C# script also has Bus and lines have results, which are not required to be filled out. Bus Result properties are being upload after starting the game.

And power line will consist of particle system component, purpose is to simulate the flow of energy into the line by generating and animating small arrows in the scene. The Particle System component has many properties, which can be adjusted in Inspector.

9.1. Box collider:

Box Collider are rectangular cuboids and are useful for items such as crates or chests. However, you can use a thin box as a floor, wall or ramp. To edit the box's shape, press the Edit Collider button in the Inspector

To exit the Collider, Edit Mode press the Edit Collider button again. A vertex appears in the center of each face of the Box Collider in Edit Mode. To move the vertices, drag them when the mouse is over the vertex to make the Box Collider bigger and smaller.

In our case as we see we have considered more than one residential building or industry to create them as residential and industrial loads. So, to create them as one game object we have used the box collider.

9.2. SCENARIOS 2

Our main scenario is to visualize the loading percent of lines when certain generation is turned off.

We are visualizing the change in loading percent on the size of the particle system of the line. When the loading percentage is more, the size of particle is increased and vice versa.

For example,

When the wind power plant is turned off, At first, when wind is on

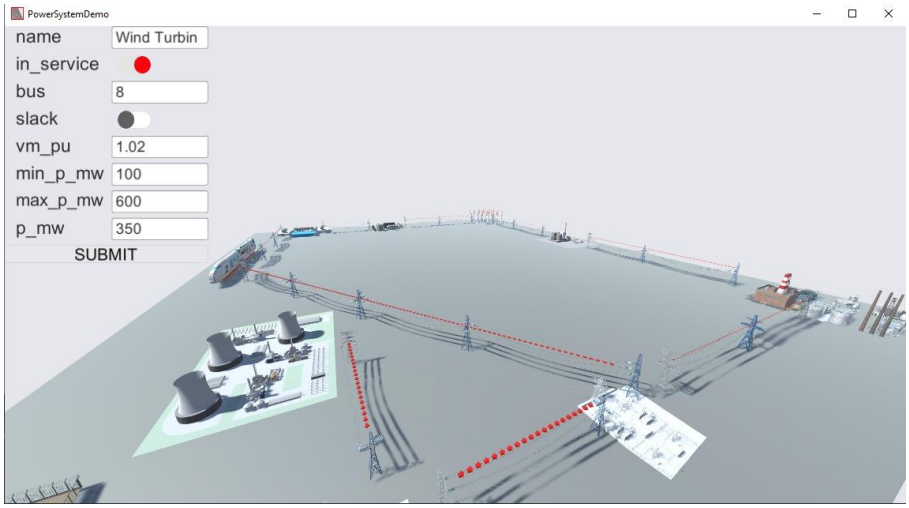

Figure 39: Particle system flow with variation in loading percent

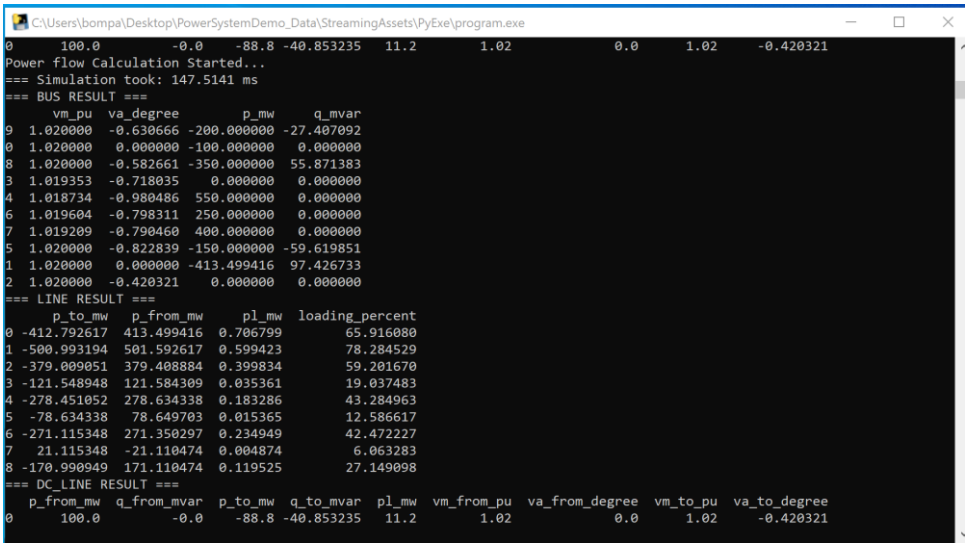

Figure 40:Server of unity (simulation values of the model)

As we can when the wind is on , the loading percentage is less than 80 % ,means loading is normal. We can also cross verify with pandapower.

]:

q_from_mvar	p_to_mw	q_to_mvar	pl_mw	ql_mvar	i_from_ka	i_to_ka	i_ka	vm_from_pu	va_from_degree	vm_to_pu	va_to_degree	loading_percent
-97.426733	-412.792617	95.265876	0.706799	-2.160857	0.632794	0.631038	0.632794	1.020000	0.000000	1.020000	-0.420321	65.916080
-54.412641	-500.993194	53.870002	0.599423	-0.542639	0.751531	0.751033	0.751531	1.020000	-0.420321	1.019353	-0.718035	78.284529
-37.992134	-379.009051	36.079292	0.399834	-1.912842	0.568336	0.567810	0.568336	1.019353	-0.718035	1.018734	-0.980486	59.201670
31.406561	-170.990949	-36.079292	0.119525	-4.672730	0.259136	0.260631	0.260631	1.020000	-0.822839	1.018734	-0.980486	27.149098
-15.877868	-121.548948	12.918868	0.035361	-2.959000	0.182760	0.182215	0.182760	1.019353	-0.718035	1.019209	-0.790460	19.037483
28.213290	21.115348	-32.863112	0.004874	-4.649822	0.052487	0.058208	0.058208	1.020000	-0.822839	1.019604	-0.798311	6.063283
10.592216	-278.451052	-12.918868	0.183286	-2.326653	0.415340	0.415536	0.415536	1.020000	-0.630666	1.019209	-0.790460	43.284963
-36.007372	-271.115348	32.863112	0.234949	-3.144260	0.407733	0.406954	0.407733	1.020000	-0.582661	1.019604	-0.798311	42.472227
-19.864011	-78.634338	16.814876	0.015365	-3.049135	0.120832	0.119778	0.120832	1.020000	-0.582661	1.020000	-0.630666	12.586617

Figure 41: Power flow analysis output in pandapower

When we turn off wind,

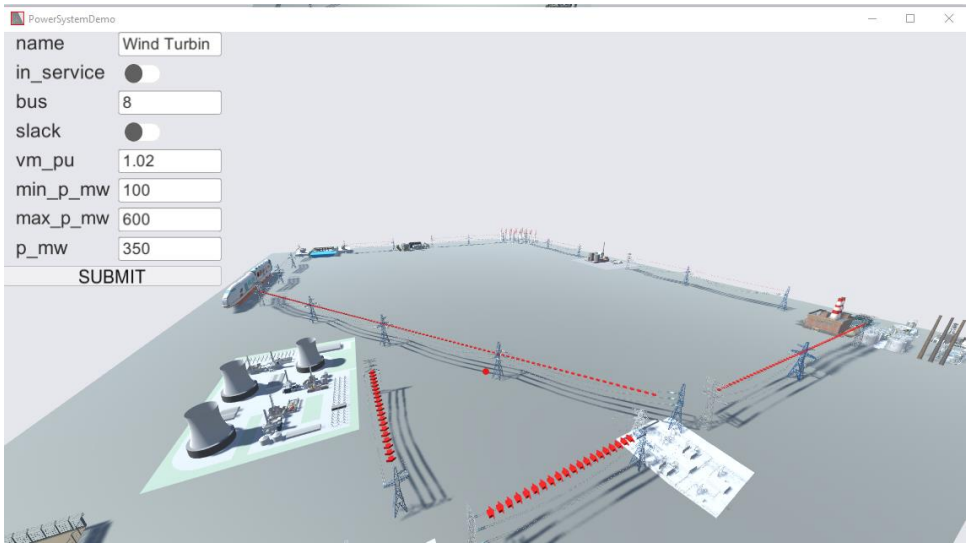

Figure 42: Line loading representation in unity when the plant is turned OFF

In unity, we can see that the size of particle system of line 1 and line 2 is increased when wind is off meaning that loading percentage

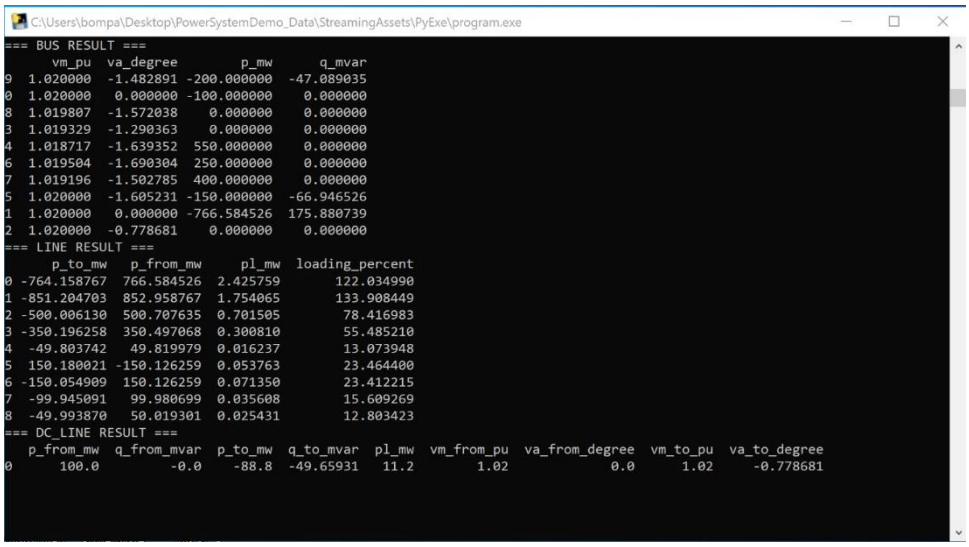

Figure 43: Server of unity (simulation values of the model)

51]:

_from_mvar	p_to_mw	q_to_mvar	pl_mw	ql_mvar	i_from_ka	i_to_ka	i_ka	vm_from_pu	va_from_degree	vm_to_pu	va_to_degree	loading_percent
-175.880739	-764.158767	181.091017	2.425759	5.210278	1.171536	1.169780	1.171536	1.020000	0.000000	1.020000	-0.778681	122.034990
-131.431707	-851.204703	135.840398	1.754065	4.408691	1.285521	1.284802	1.285521	1.020000	-0.778681	1.019329	-1.290363	133.908449
-66.136233	-500.006130	65.517145	0.701505	-0.619088	0.752803	0.752098	0.752803	1.019329	-1.290363	1.018717	-1.639352	78.416983
60.441017	-49.993870	-65.517145	0.025431	-5.076128	0.116861	0.122913	0.122913	1.020000	-1.605231	1.018717	-1.639352	12.803423
-69.704166	-350.196258	67.883557	0.300810	-1.820609	0.532658	0.531765	0.532658	1.019329	-1.290363	1.019196	-1.502785	55.485210
6.505509	-99.945091	-11.023083	0.035608	-4.517574	0.149241	0.149849	0.149849	1.020000	-1.605231	1.019504	-1.690304	15.609269
64.840613	-49.803742	-67.883557	0.016237	-3.042943	0.121801	0.125510	0.125510	1.020000	-1.482891	1.019196	-1.502785	13.073948
-14.867685	-150.054909	11.023083	0.071350	-3.844602	0.224757	0.224226	0.224757	1.019807	-1.572038	1.019504	-1.690304	23.412215
14.867685	150.180021	-17.751578	0.053763	-2.883893	0.224757	0.225258	0.225258	1.019807	-1.572038	1.020000	-1.482891	23.464400

Figure 44: Power flow analysis output in pandapower

Here from the above figures (from pandapower and unity console.), we can visualize that when wind is off, the loading percent of line 1 and line 2 is 122% and 133%.

When hydro power plant is turned off

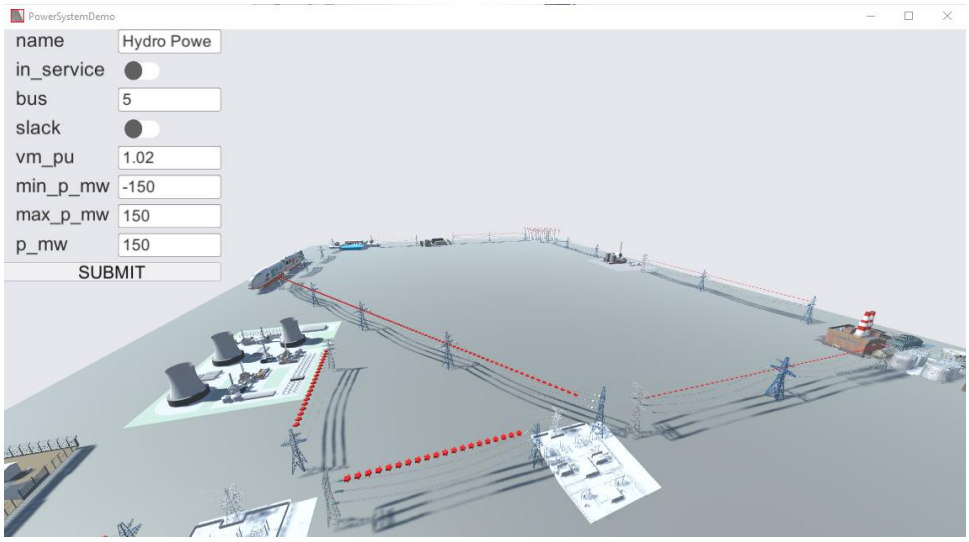

Figure 45: Line loading representation in unity

As like above example, when generation is off the loading percentage increases which we can visualise in unity with change in size of particle system.

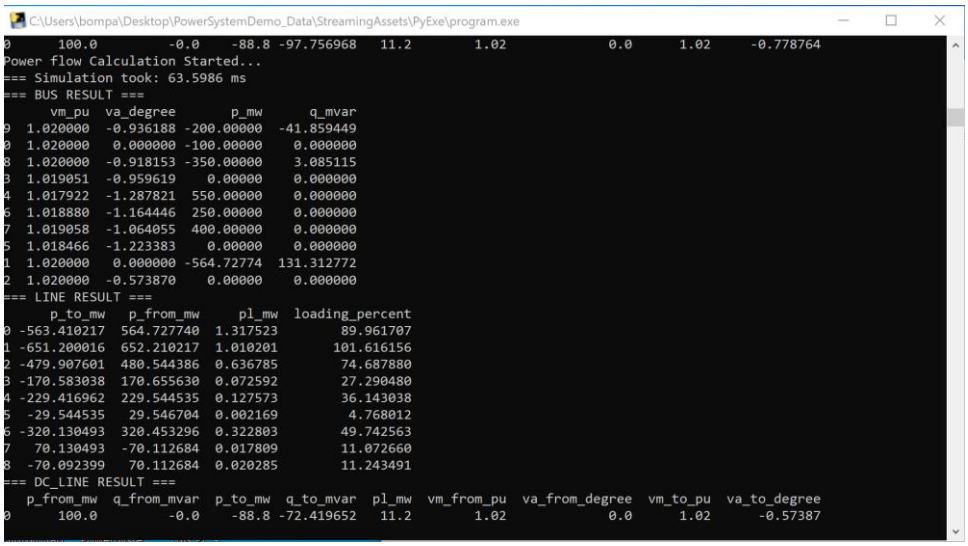

Figure 46: Server of unity (simulation values of the model)

From above figure, we can vizualize the values of loading percentage.

When biogas is turned off,

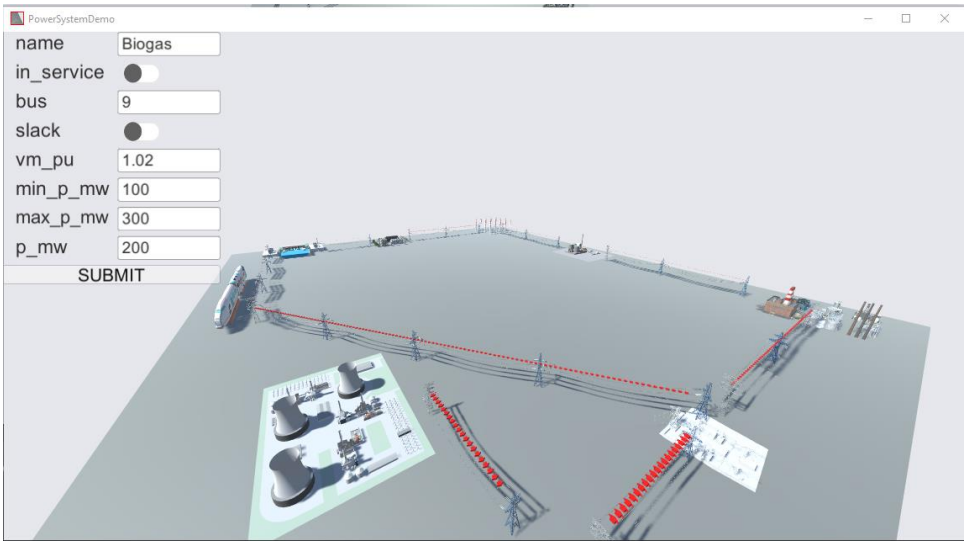

Figure 47: Overloaded condition when the generation of biogas is turned OFF

As we can see here the line 1 and 2 are overloaded which can visualize through size of particle system.

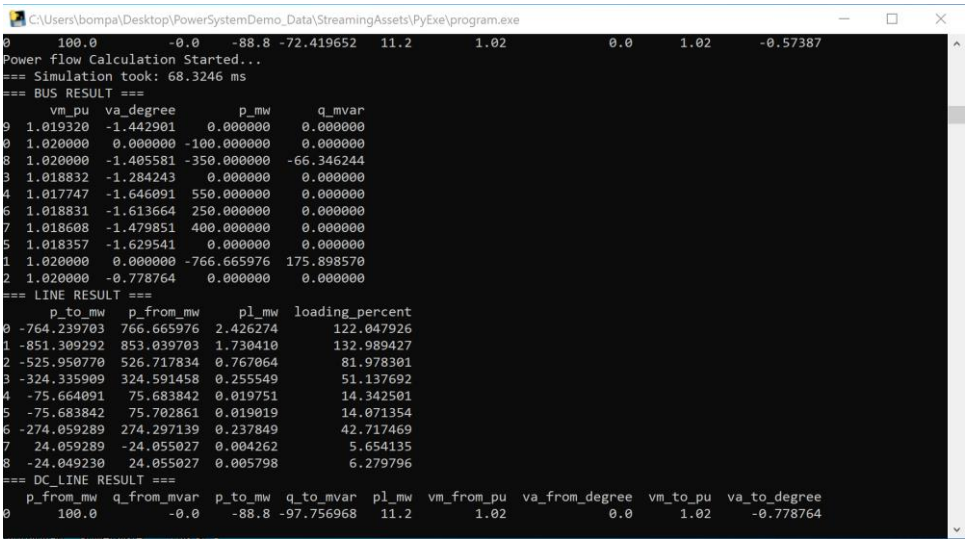

Figure 48: Server of unity (simulation values of the model)

q_from_mvar	p_to_mw	q_to_mvar	pl_mw	ql_mvar	i_from_ka	i_to_ka	i_ka	vm_from_pu	va_from_degree	vm_to_pu	va_to_degree	loading_percent
-175.898570	-764.239703	181.111055	2.426274	5.212485	1.171660	1.169904	1.171660	1.020000	0.000000	1.020000	-0.778764	122.047926
-83.354087	-851.309292	87.662862	1.730410	4.308775	1.276699	1.276236	1.276699	1.020000	-0.778764	1.018832	-1.284243	132.989427
-32.774041	-525.950770	32.441296	0.767064	-0.332745	0.786992	0.786658	0.786992	1.018832	-1.284243	1.017747	-1.646091	81.978301
27.294262	-24.049230	-32.441296	0.005798	-5.147034	0.054280	0.060286	0.060286	1.018357	-1.629541	1.017747	-1.646091	6.279796
-54.888821	-324.335909	52.877440	0.255549	-2.011380	0.490922	0.490163	0.490922	1.018832	-1.284243	1.018608	-1.479851	51.137692
-27.294262	24.059289	22.652873	0.004262	-4.641389	0.054280	0.049279	0.054280	1.018357	-1.629541	1.018831	-1.613664	5.654135
49.853438	-75.664091	-52.877440	0.019751	-3.024003	0.135085	0.137688	0.137688	1.019320	-1.442901	1.018608	-1.479851	14.342501
19.524197	-274.059289	-22.652873	0.237849	-3.128677	0.409614	0.410088	0.410088	1.020000	-1.405581	1.018831	-1.613664	42.717469
46.822048	-75.683842	-49.853438	0.019019	-3.031390	0.132589	0.135085	0.135085	1.020000	-1.405581	1.019320	-1.442901	14.071354

Figure 49: Power flow analysis output in pandapower

Here also we can observe that line 1 and line 2 are overloaded, the loading percentage is 122% and 133%.

So that user can visualize the overloading of lines by size of the particle system when any one of the generations is turned off.

10. Summary of the Project

Power system demonstrator in Unity 3D is a project aimed at developing a power system network in three-dimensional appearance for the users. It is an illustration of power flow analysis in a power network. In our project we analysed the HVDC technology which is a future o the power network. The various ways by which an HVDC power transfer can improve the power low quality has been discussed in these projects. The simulation of the power network and the the power flow are carried out with Pandapower which is an open-source Python-based power system analysis tool. For visualizing the power flow and network, a game developing platform, Unity has been included to this project. Unity is a vast platform to play on ,in the field of game and 3D model development. It included objects to represents the power elements like buses, loads and generator and scripts added to each objects enables its visualization in 3D platform. The power flow analysis which has been completed using Pandapower and is incorporated with the unity through a server so as to determine the grid parameters and the power values, loading percentage and so on.

Through these projects we expected to implement certain scenarios, which included two scenarios mainly:

- Our main scenario is to visualize the loading percent of lines when certain generation is turned off. We are visualizing the change in loading percent on the size of the particle system of the line. When the loading percentage is more, the size of particle is increased and vice versa.
- The implementation of a slider (UI) to control the generation of power in a wind mill ,where the user can externally control the speed of wind generation from a minimum to maximum value so that the user can visualize it in the run mode , which was successful to some extent.

There are future scopes for this project in the field of more UI developments and we can also develop certain scenarios which shows the cost of power generated and utilised and many more. Visualization of power network in 3D with better power flow analysis tools and model development engine can make the environment a bit more user friendly in near future.

11. APPENDIX A

1. https://sketchfab.com/feed
2. https://3dwarehouse.sketchup.com/?hl=en
3. https://sketchfab.com/3d-models/wind-turbine285032402a8543ae8bf3e3c4d8c9f98a
4. https://3dwarehouse.sketchup.com/search/q=hydro%20power%20plant&searchTab=model
5. https://3dwarehouse.sketchup.com/model/27b14d09-8268-43ac-b299-ca3b62af46fa/Natural-Gas-CCGT-Power-Plant-2400-MW?hl=en&login=true
6. https://3dwarehouse.sketchup.com/model/71a3d5c6dd015c14d85a6683f4cf7be2/power-station-integrated-algae-biodiesel-plant
7. https://sketchfab.com/3d-models/hv-substation-0a64a6e71dce416aa43c5a9b4fe8ef24
8. https://sketchfab.com/3d-models/high-voltage-transmission-line-tower-tileable-cbeec33de6d64374b6d22de2749c1313

12. BIBLIOGRAPHY

[1] Kisomi, S. P. (2022). Power System Demonstrator in 3D. *Master thesis at Darmstadt University of Applied Sciences*.

[2] Thurner, L. a.-H. (2018). Pandapower—An Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems. *IEEE Transactions on Power Systems*.