# Energy System Integration Demonstrator

# Energy data widget

Daniel van Paassen &
Benjamin Visser

BSc Thesis
July 14, 2020

D. van Paassen, *4724968*
B.E. Visser, *4593251*

TU Delft

# Energy System Integration Demonstrator

# Energy data widget

by


## Daniel van Paassen & Benjamin Visser


to obtain the degree of Bachelor of Science
at the Delft University of Technology,


| | | |
|---|---|---|
| Student number: | 4724968 (Daniel van Paassen) | |
| | 4593251 (Benjamin Visser) | |
| Project duration: | April 20, 2020 – June 19, 2020 | |
| Thesis committee: | Dr. M. Cvetkovic, | TU Delft, supervisor |
| | Dr. ir. A. van der Meer, | TU Delft, supervisor |
| | Dr. A. Smets, | TU Delft |
| | Dr. S. Wong, | TU Delft |



An electronic version of this thesis is available at `http://repository.tudelft.nl/`.


TU Delft

# Abstract

Renewable sources are gaining more importance now than ever. As a consequence, the development and complexity of electricity grids are rising. This thesis is part of a project which aims to comprehend and package the integration of such complex grids into something more understandable such as a demonstrator tool. More precisely, how do you find a solution to model these kinds of components inside a grid to something easier and touchy? In the context of this thesis, this means that sources, loads and energy storage in today's grids need modelling to represent their operations.

Based on models in literature studies, a correct operational way of modelling is discussed and worked out. Several cases will be presented where the operation of such models is verified and presented. Experimental results show that the accuracy of these models comes very close with actual data values from manufacturers. On the basis of these results it is stated that a tool such as this thesis describes, will resemble such a transition appropriately and therefore helps for better understanding.

# Preface and Acknowledgements

Within the first week, changes were made to the outline of this project due to the drastic events. Nonetheless, a very educational and rewarding experience was gained during this graduation project. If anything, these changes have shown us that even during quarantine, a good and enjoyable venture can be had.

We would like to express our gratitude to Milos Cvetkovic and Arjen van der Meer for their continuous support and for providing us this project. Furthermore, we want to thank Siva Kaviya Trichy Siva Raman for helping us with setting up the Raspberry Pi's, and showing us the possibilities of how they can communicate. Last of all, we would like to thank our colleagues Ruben Siemensma, Victor van Doorn, Jean-Paul Rozestraten and Ali Ahmad Sohrab Ashraf.

*Daniel van Paassen & Benjamin Visser*
*Delft, June 2020*

# Contents

# 1 Introduction

## 1.1. Problem Description

The knowledge of how a city or neighbourhood could operate on sustainable sources, is often not known to the common people. Nonetheless, it is an important global transition that must be achieved. The energy demand in the world is continuously rising and, and renewable sources are needed to stop polluting the earth. Renewable implementations for energy are very popular already [1], and increasing still, which means that the market will expand. In order to positively stimulate the use of renewable sources, and tackle the complexity the integration of these sources demands, a way of explaining in simple terms is required. Energy grids are always gaining components, for example electric vehicles (EV's) have starting getting popular and appearing more common since the 20th century.

These additional components create more energy distribution options within a grid and need extra integration. Such developments require investments and primarily more apprehension. However, having the common people inform themselves about renewable sources independently, as well as an integrated solution, would be quite the challenge. A simplified, but accessible and easily visualizing tool is necessary to cross the gap of knowledge about integrated neighbourhoods powered by renewable sources. Such a tool would offer insight into the workings of these neighbourhoods without the need of understanding the engineering behind it. This tool visualizes all essential characteristics that are needed to understand and motivate people to innovate in the energy market.

The creation of such a tool is going to consist of multiple components presented visually in fig 1.1. Different models must be implemented and be able to work together to simulate the powers in different circumstances and show what influences the renewable sources, and how to operate independently on those. A case study has been chosen to simulate a new development of a neighbourhood near the city of Delft. The input parameters for this case can be changed freely in order to understand what are the best choices seen from a financial and technological aspect.


Figure 1.1: The energy integration demonstrator in clip art image

## 1.2. State of the art analysis

The transition into renewable sources is an active field of research, being one sustainable development goal from UNESCO, the development of educating about green energy has been encouraged greatly. For example, the ”Changing with the climate network” project, which contains several school activities for educating about the renewable energy use [2]. In the discussion about the conferences of the Engineering Education for Sustainable Development (EESD), it becomes clear that now there is a need for participatory methods, and interaction, with sustainable energy technologies [3]. Many developed demonstrating tools around sustainable energies are mostly about market-analysis, prototype analysis or about the policy making and how the public would adopt the renewable technologies [4]. All of these categories are not aimed at the public, but the parties already involved in renewable technologies. Only a few tools are made to offer insight about these technologies to the public with a demonstrating tool. ESDL, the Energy System Description Language project [5], has created such a model (called ESSIM) that offers insight about how a system with renewable sources works. This tool has enough customizability of the system. For example, storage options can be added. Also, it will provide a lot of information about the system as a whole, during which periods there is an excess or shortage of energy, and the CO2 emission of the system. This tool however, is completely online, and some interactive part is lost due to this. There is no interaction possible with physical table-top models and the visualization is only digital.
Another model, is the System Advisor Model (SAM) [6]. This computer model calculates performance and financial metrics of renewable energy systems. This model is very accurate and powerful, but is made for policymakers, researchers and project developers. These are audiences that have some pre-existing knowledge about the working of a renewable system. This makes the tool unsuitable as an education tool for the common people.

## 1.3. General overview

The visualisation and demonstration of power flows throughout a grid is done using a demonstrator kit. In fig.1.2 an overview of the demonstrator is shown. The demonstrator consists of Raspberry Pi’s depicted in the colour raspberry. Each client Pi is connected to a ”Hardware hub”, to which table-top models are connected, visualized by the images. The server Pi however, is connected to all client Pi’s and this is also where the visualization is located, which is done by the program called ”Dashboard” using a monitor. This demonstrator is able to show power production, consumption and the influence of changing inputs on these powers. This visualization is not only done by looking at a monitor but also has a more tangible side by the addition of a hardware hub. The hardware hub is a component which will resemble what is going on in the grid by means of actuators and sensors. Hence the demonstrator will consist of 3 submodules.

### Dashboard visualisation

In the dashboard everything will come together. The hardware hub can be controlled by the dashboard by changing the parameters. So the dashboard can work as an input but also as an output for information. The dashboard shows graphically how much a PV system has generated over a year with a resolution of 1 hour. This can also be zoomed in, for example, how much the neighbourhood consumes in 24 hours with steps of 1 hour. Furthermore, the dashboard uses all this data to make an emission analysis and a financial analysis. From these rigorous analyses the client can make decisions on for example increasing the amount of houses in the neighbourhood.

### Energy Data Widget

The Energy Data Widget is going to be the backbone of the demonstrator. It will calculate all values regarding powers and energies by means of Python. This software is implemented on Raspberry Pi’s and is able to handle 2 types of inputs, coming from a dashboard or from table-top models. When using Dashboard as main input all relevant weather and consumption patterns are retrieved through

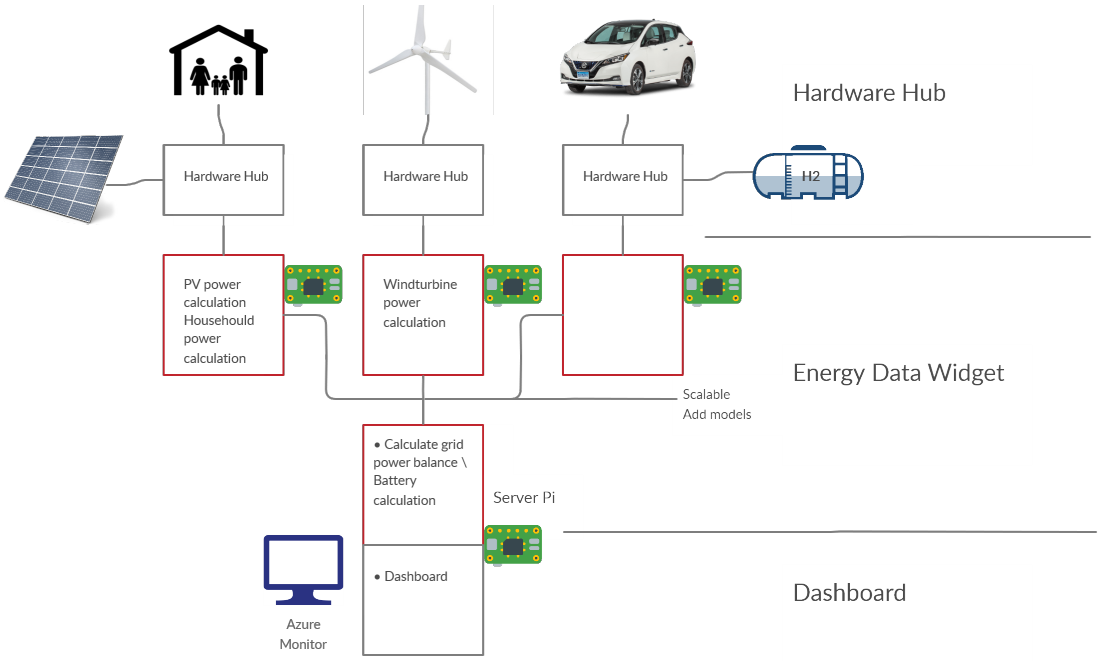



Figure 1.2: Schematic overview of the Demonstrator

an online database as well as type and amounts of components within your grid.
Besides the Dashboard the table-top models are also able to create inputs. However, they just output actual weather data created by sensors and can not change type or amount. The models should be able to operate in both modes.

### Hardware hub

The Hardware Hub is a physical setup that is able to emulate information calculated by the Raspberry Pi's, using table-top model representations of energy systems. This implies that if for instance a lot of wind energy is being generated, the table-top wind turbine model will actuate that generated power by letting a DC-motor rotate. Similarly, if the households in the neighborhood are using a lot of energy then the table-top model will actuate that by showing a small lamp shine brighter than if it weren't using consuming a lot of energy.
The Hardware Hub makes it possible to interact and play with these table-top models, it is possible to change loads by rotating potentiometers, add or remove the electric vehicles by using switches and influence generated wind power by rotating the DC-motor. The Hardware Hub is also able to sense real physical information using sensors and is able to send this information back to the Raspberry Pi's. One example the solar PV table-top model, this model is able to sense the irradiance by using a physical solar PV.

## Structure of Thesis

This thesis is going to describe the design and implementation of the "Energy Data Widget" module, which is responsible for calculations concerning powers and energies. It is also responsible for the communication between the different raspberry pi's and for gathering input data to perform all of its calculations.
First an overview of all requirements is elaborated on in Chapter 2. The architecture and structuring of the software which the data widget mainly consists of is explained in Chapter 3. In Chapter 4 it will be discussed what models are appropriate and what choices are taken concerning the requirements.

The communication and data gathering between all Pi's will be elaborated on in Chapter 5. The chosen models are then explained in Chapter 6, where the verification and simulation of the system is discussed in Chapter 7. Finally, all results along with additional future work for improvements are discussed in Chapter 8.

# 2

# Program of Requirements

This chapter will describe the requirements which the product must have and would preferably have. Below the requirements are split up for the complete system and sub module. These requirements are divided up into mandatory, necessary to function, and trade-off requirements, enriching the product.

## 2.1. Requirements for the system

These are the requirements the system should comply with in order for all sub modules to work together correctly and to have a full functioning demonstrator.

### Mandatory requirements

(a) The Demonstrator must be able to calculate the power flows from real-time simulated weather data coming from the hardware hub (table-top models).

(b) The Demonstrator must show all power flows going into and out of the system and its individual components (for example a solar panel or wind turbine).

(c) A communication needs to be established between the Pi's.

(d) Table-top models as well as the dashboard shall be able to visualize information regarding the powers of the system.

(e) The table-top models must be interactive so that the user can physically influence the individual components such as the wind turbine or solar panel.

(f) The Dashboard must be interactive as well, so that the parameters can be changed (such as the amount of EV's or solar panels), and this leads to a change in output.

### Trade-off requirements

(a) The resolution of the power flows from the table-top models should have an hourly resolution.

(b) The dashboard should be easily operable, with only a keyboard and mouse.

(c) The system should be easy to understand for laymen.

(d) The system should be built with low cost, no more than €100.

(e) The performance should not be too slow, preferably within seconds for a new simulation.

(f) The system should not be too large to make it easy to carry in for example a briefcase.

(g) The product should preferable also handle extreme cases, such as a hack on the wind turbine, disabling all power flow from that component within the system.

(h) The Demonstrator should be able to simulate a neighbourhood on in different places, otherwise the area of Delft is preferred for the case study.

## 2.2. Requirements for the Energy Data Widget

The Energy Data Widget submodule focuses on creating a communication layer between the Pi's and implementing all the requirements surrounding the generation or load of energy from the various models based on weather data and preselected parameters, such as the solar panel tilt. The requirements for the Energy Data Widget are listed as follows:

### Mandatory

1. An inter process communication protocol must be implemented to connect all the Raspberry Pi's
2. The Demonstrator should be able to use both real-time data inputs from the table-top models, or the simulated data from a year overview, using old weather data.
3. The Demonstrator must function on Raspberry Pi's.
4. The adequate software must be open source.
5. The Demonstrator must calculate power flows of models (such as wind turbine or solar panel) based on accurate weather data.
6. A load needs to be implemented.
7. The implemented models for the generation and load must have parameters to simulate different combinations of sources or loads.

### Trade-off requirements

1. The performance should not be too slow, preferably within seconds for a new simulation.
2. The renewable sources should contain wind turbines and solar panels for the chosen case study.
3. The load should be modelled as a household for the chosen case study.
4. The whole system should be easily scalable, connecting more Pi's should not be a problem.
5. The demonstrator should be able to simulate with models disconnected, no wind for example.
6. There should also be storage options such as a hydrogen tank and/or Electric Vehicles (EV's) to make the neighbourhood more national grid independent.
7. These EV's should not fully deplete, to give the owner the option to use their car.
8. The system should be able to calculate the power flows for one year coming with an hourly resolution. Preferably based on recent weather data, such as from 2019.
9. The system's operating software is written in Python 3.7 or higher.
10. The weather data should be dynamically grabbed, depending on the location of the simulation, otherwise Delft is preferred.

# 3

# Software Architecture

According to second mandatory requirement of for the Data Widget, the whole system will have two operation modes, the first is where the parameters are specified by the Dashboard (such as the amount of solar panels). After this is specified a whole list of powers in a year with hourly resolution will have to be created. The second operation mode is the realtime mode, where the powers will be displayed on the table-top models. They will additionally be able to override these powers, by interacting with them physically. For example a table-top solar panel that you can put your hand above.
The figure of the modules interconnected is depicted in Fig. 3.1. It is important to note that this is simply a possible combination of table-top modules and client Pi's, this is completely modular and all combinations are possible.
The server Pi can be seen as the centre of the system. All client Pi's share the server its memory and communicate with the server Pi to be able to start its execution. The Dashboard is also located on the server Pi as it must retrieve all outputs given by the clients.

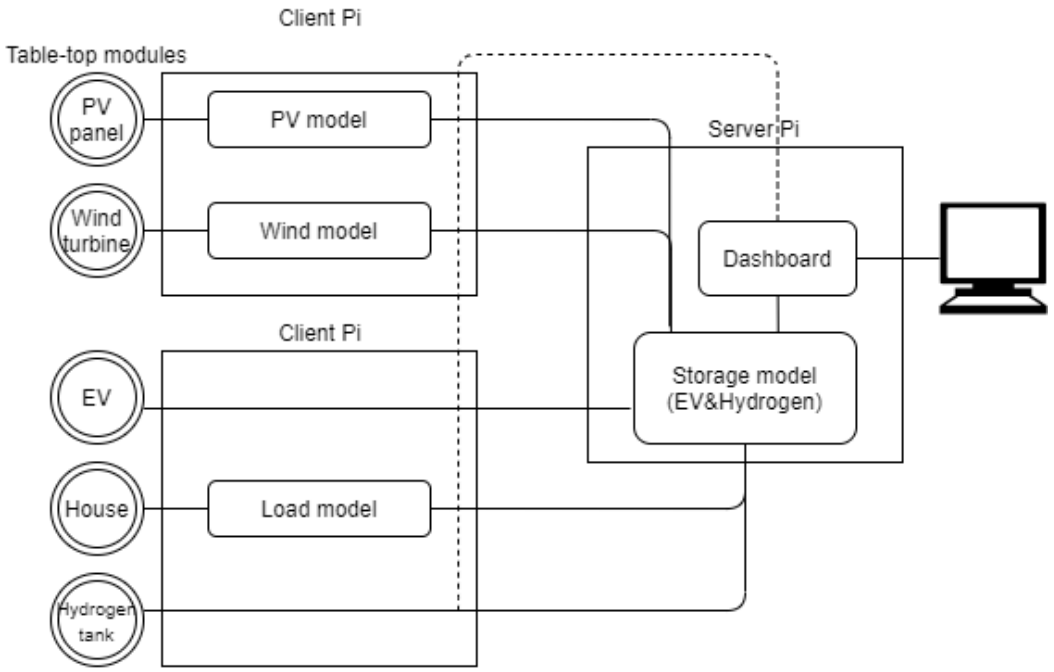



Figure 3.1: An overview of the whole Demonstrator with its components, where the continues lines represent the data signals of power flows, and the dotted line represents a control signal from the dashboard.

## 3.1. Yearly overview

For the first operation mode, the yearly overview, the table-top modules will only be used to let the Demonstrator know that they are connected or not, which results in the component appearing or not in the Dashboard. All the models except for the storage model, are able to be executed independently because they are all located on a client Pi.
Each client Pi can perform the their model calculation in parallel and so the overall performance will be faster. These models will calculate the powers for every hour in 2019, using the downloaded weather data and the parameters specified by the Dashboard, such as amount of solar panels or wind turbine type. Once refreshed by the Dashboard, these parameters are passed through the control signal, shown with a dotted line. This line goes through every client Pi to make these parameters known to all Pi's. Once all these powers have been created, they will go to the storage model in the form a of a list. The storage model will be located on the server Pi, together with the code of the Dashboard. Running the storage model on one of the clients would not work since the storage model needs all the calculated powers from the other models together as an input. Besides this, the storage model needs to know whether the EV and/or hydrogen tank are connected which is known through the clients. It will then calculate the powers of the whole year for the hydrogen tank, EV's, and the grid. This whole list is then sent to the dashboard.

### Hierarchy

All these lists of powers will be rounded to three decimals, this will reduce the size of such a list to 256 bytes and make for an almost instant transfer from one Pi to the other when it is being sent. The transfers will be done from a top module in each Pi. On the client Pi, such a module will check all the table-top models that are connected to it, and perform a calculation of that specific models when it receives a new request from the server. The server will then also have such a top-level module, which runs the dashboard and also creates a new request for all the clients whenever new inputs have been selected for a certain model within the dashboard. The storage model will can be called by the dashboard directly, since it has all the power lists, and is located on the same Raspberry Pi. All these lists are stored in Python dictionaries for convenience. A list can be accessed by extracting it from the dictionary.

## 3.2. Realtime mode

For the second operation mode, the realtime mode, all the power lists of 2019 from the yearly overview mode will be remembered. The user will then be able to select an hour to simulate from within the year, and the table-top modules will display the corresponding hours sequentially, with every second representing an hour. Additionally, the input side of the table-top models will be able to override the current energy production of the specific model (such as the power output of a PV with a hand over it). This will also go to the storage model and change its power outputs as a result. Eventually, it will all be displayed on the Dashboard.

### Hierarchy

The realtime mode will start once the dashboard has selected an hour to emulate from. This will be sent to all the clients. These clients will have a local copy of the last power list they have created, which is the same emulation as on the Dashboard. Storing this locally is not a problem for these Pi's since every list is only 256 bytes, as mentioned above. The memory of the client itself is 1GB (for the Raspberry Pi 3B+).
The client Pi's will use a time function to send the power corresponding to that hour to the table-top models, and every power that comes after that, with a pause of a second between each power. This means that eventually all the powers will be sent sequentially to the table-top models. Furthermore,

the sensors of the connected models can be physically activated. The output of these sensors will be inserted in the power list, at the position of the next emulated hour. Meaning that the overridden power value of a the emulated hour will be sent back to the table-top model as well as to the Dashboard. The Dashboard reacts to this by inserting the power value for the specific model in the power list and updating its display.

# 4

# Existing Models

The models implemented to simulate the power flows from either the generation or the load will be discussed in this chapter. These are designed conforming the Program of Requirements. Several models will be implemented to account for power solutions with renewable sources. The most important models are the ones for the generation of renewable power, which will be done with a wind turbine and/or a solar panel. For the load an accurate model should be designed that resembles a small neighbourhood in a Western European country. After this some options for the storage of energy will also be implemented, namely by a hydrogen tank and electric vehicles. This will make the neighbourhood more grid independent as it can store its own energy for use during higher demand. The electric vehicles will also be acting as a load, since the usage by the vehicle's owner will also be accounted for.
The weather data that is used for the models is retrieved from Renewables.ninja [7, 8], and will be explained in Chapter 5.

## 4.1. Possible implementations

### Wind turbine

To model the power output of a wind turbine as realistic as possible but yet maintain simplicity there are 2 main approaches which can be taken. Usually for the modelling of a wind turbine complex components such as gearboxes, rotor-speed feedback and converters should also be considered to take into account these influences on the power output [9], but seen the scope of the system these are going to be left out to ensure that the design is kept as compact and understandable as possible. These components would otherwise demand modelling of themselves which is unnecessary regarding the scope of the demonstrator.

A first approach is to use the wind power equation [10, 11], which is often used to describe the power a wind turbine produces given by Eq. 4.1 where $C_p$ stand for the power coefficient, $\rho$ stands for the air density, and $A$, the cross-sectional area of the blades which equals $\pi \cdot r^2$ and $v$ stands for the wind speed.

$$P_{windturbine} = \frac{1}{2} * C_p \cdot \rho \cdot A \cdot v^3 \tag{4.1}$$

This equation is quite accurate but there is one extra attribute and that is the power coefficient $C_p$. The power coefficient of a wind turbine is the relation between the extracted energy from the wind and the available energy in the wind stream. This coefficient relates closely with parameters such as blade angle, rotational velocity, wind speed and other turbine components. These parameters are often described as a function of the tip speed ratio, the ratio between the blades tip rotational velocity and the

wind speed [11]. This additional relation is going to demand a separate model for the power coefficient before being able to describe the power output. Which would mean creating a controller for the wind turbine to function optimally.
A second approach is to model the power curve of a wind turbine given its operating values [12]. There are various methods on how to model a power curve with their limitations and specifications. By means of a wind turbine's data sheet a mathematical function can be created that narrowly represents a wind turbine power curve. There is a variety of polynomial mathematical equations possible to use such as linear, quadratic, binomial and cubic. They are all based on the following Fig. 4.1.

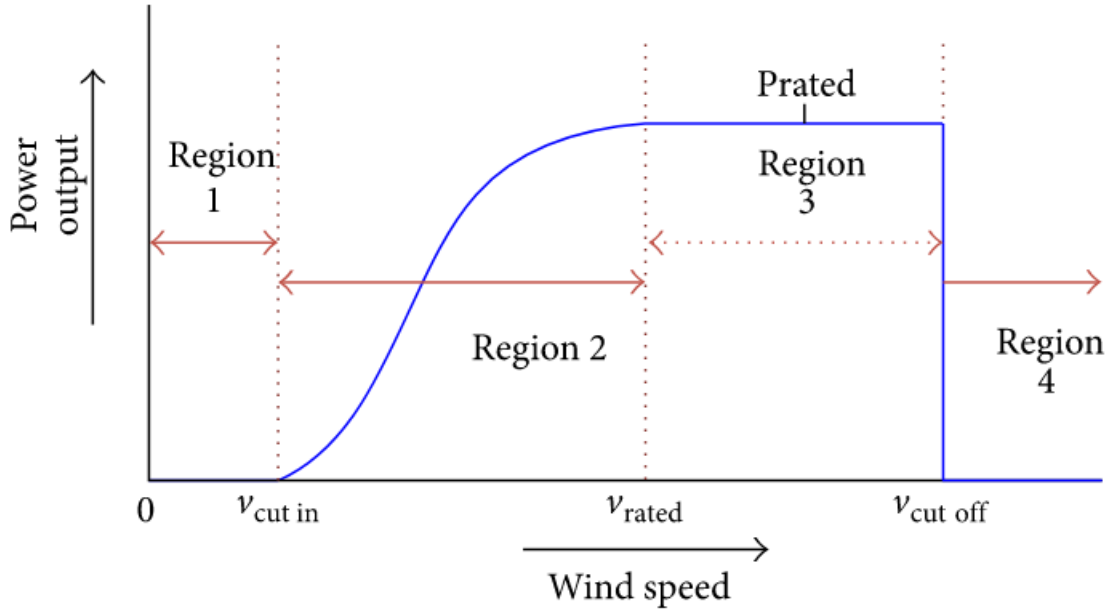



Figure 4.1: Typical wind turbine power curve symbolic [12].

When mathematical equation gets more complex the accuracy of the model is expected to increase. Therefore, a model choice has to be made that will satisfy for the whole system regarding the complexity.

## Solar panel

Various models for the solar panel can be found, one of these models [13] contains an MPPT (maximum power point tracking) algorithm on the DC-DC boost converter to get the maximum power point, which is then used for a load, through a three-phase inverter using Pulse Width Modulation. Where the solar cells have been modeled with a p-n junction and the characteristics of the PV module have been used to calculate the power output using the MPPT. Additionally, a filter has been introduced to remove the signal components higher from the inverter switching frequency.

Another model uses storage methods to balance the stability of the grid more when using the solar panels [14]. It is similar to the first model regarding the PV module itself, but takes into account that the ambient temperature is different than the cell temperature for the calculation of the photo-current.

The last way to model a solar panel that has been looked at is to gather the total irradiance that falls on the solar panel and using the parameters of the manufacturer's data sheet. The parameters used for this are the size of the panel, the efficiency of the solar panel, and the temperature coefficient of $P_{max}$. This model has the assumption that an existing MPTT algorithm is already implemented, and only the power should be calculated.

### Household

Modelling a household can be a tedious and difficult process. One way to get an accurate model would be from a modelling program that allows for enough customization to match the average load profile in a Western European country. The second approach would be to find an open-access dataset of the consumption of a (group of) home(s). This second option has limited recent resources available. Due to privacy issues, many datasets did not have the usage of individual homes in hourly resolution, but the usage in a whole year instead. Most suitable datasets were from the UK, where many government funded projects investigated the energy consumption of homes. But the most suitable set is from Germany [15]. It is the consumption of eleven households in southern Germany and has the data available in a minute, to hour resolution. The consumption could be averaged and then used with a multiplier to model the whole neighbourhood.
For the household modelling program, a load profile generator tool [16] can be used, this tool has various customization options as well as templates for households in Germany. A settlement can be made using these households templates, after which this can be averaged to the hourly load electricity consumption of a household. The simulations of this tool take more than an hour for a settlement over 100 houses so the results should be saved locally. After which it can be used with a household multiplier as a model.

### Battery

The fact that wind and solar energy are the main sources in this demonstrator with the purpose of transforming to a more renewable future, storage options must be considered. There are two type of storage options to be modelled, as can be seen in Chapter 2. The first one is the electric vehicle (Li-ion battery), which can be characterized mainly by its fast charge/discharge rates, availability and energy storage capacity [17]. The reason behind an EV instead of a fixed Li-ion battery is to make the system more future-oriented and dynamic. EV's are going to be far more available in the future and using EV's as energy storage is very an obvious solution. Based on these characteristics it follows EV's are most suitable for small and quick change in energy demands or surplus, this means that the EV's will be implemented in a Vehicle to Grid (V2G) way. To compensate for larger fluctuations and to be self sustainable (use little grid power) a hydrogen storage model is added to the system.

Hydrogen can be characterized by having a large energy storage capacity but response time to quick fluctuations in energy demand being smaller. The main differences between the two as storage options are that Li-ion batteries have a self discharge which results in capacity loss of the Li-ion battery [18], Li-ion batteries have a smaller charge rate, if no extra capacitors are added.. However, the fact that multiple EV's can be used in the Demonstrator, makes it appealing to use it for grid stability, from the high overall power that can flow to it with a high round-trip efficiency. Hydrogen is a better long-term storage method for the seasonal excess power from summer to winter.

To model the power flow going from and to the EV's and hydrogen storage an algorithm needs to be implemented which regulates the incoming and outgoing power flows from the loads and sources. This can be implemented by deciding on what storage and what kind of storage is available in the system. This is based on a design choice and will be elaborated on in the next section.

## 4.2. Model selection

### Wind turbine

There are a lot of factors that can be considered when modelling the power of a wind turbine. Regarding the demonstrator, the modelling of the wind turbine does not need to go very in depth. A model that will describe and predict the power output related to the wind speed is in this case sufficient. The usage of the wind power equation would mean create an additional controller to make the wind turbine function optimally, which would be an added step. This makes the second approach a more attractive

candidate, since it already runs optimal. Because the power curve uses actual measured manufacturer data, the output will actually resemble a wind turbine in practice. This also means errors and losses are taken into account.

The decision on which type of mathematical model would suffice, is to be made by choosing a balance between accuracy and complexity. Starting from an easy model, the linear model, they were analysed on their curve characteristics. A linear model would not suffice as the transition points in the curve would not be smooth. A polynomial function as quadratic or cubic function would represent the middle section more accurate. In Fig. 4.1 it shows the first, third and fourth region are linear for all possible turbines, as this is often the case in real turbines [19]. The power output is then limited to stay around the rated power after the rated speed to limit breakages. After $v_{cutout}$ the wind turbine is shut down. Region 2 is what requires most accurate modelling to be as representative as possible. For this reason cubic is an accurate match [12, 20]. A cubic function spline goes through the data points the smoothest which is exactly what is required.

### Solar panel

The model for the solar panel in the total picture of the Demonstrator, does not need to have the current and voltage defined as outputs. This information is not required according to the Program of Requirements (Chapter 2). Furthermore, the second solar panel model discussed in 4.1 that uses storage to stabilize the grid from the solar panels, is not needed. The neighbourhood grid will already have storage options, so this is not needed either. The final approach that uses the irradiance and converts it to the power with the cell temperature, is the most suitable for the Demonstrator. The wind speed and ambient temperature are available so can be used in this model. Furthermore, this is the best way to model different solar panels since it uses the data directly from data sheet parameters.

### Household

When comparing the two presented methods in Section 4.1, it comes down to the accuracy of the created data. Both models satisfy the requirement of an hourly resolution. The open-access data of the 11 households has a full year (in 2019), where as much as 6 residential households are continuously monitored. However, weird spikes were present in one of those households (of for example 20kW from a solar panel at night), and a couple of the households had an abnormal high average use of power (2.5+kW). This means that the sample size of reliable data would be too small to be very accurate.

The load profile generator tool however, has many advanced features, such as randomly generated profiles for every user in the home with their own traits and characteristics. For the average household, this tool resulted in 0.357 kW on average throughout the year. Compared to the 0.3413 kW statistic from the average electricity consumption in The Netherlands[1] the resulting difference is 4.5%. This makes the profile generator a good choice for modelling the load.

### Battery

As stated before, modelling of any detailed features related to these techniques are excessive, taking into account the scope of the system. The most appropriate approach for an energy storage model is the implementation of an algorithm which decides how much energy should be stored or discharged. This algorithm is going to base this on predefined parameters and limitations. The algorithm should be designed in logical way and reflect reality. The algorithm is going to include 2 types of storage as discussed before, in order to tackle the fast fluctuations in power but also the seasonal fluctuations. The design will therefore prefer EV's as storage before storing in the form of hydrogen. The EV's will then be set up in a Vehicle to Grid (V2G) way.

[1] from Nibud: "Energie en Water", 2019

# 5

# Communication

All Raspberry Pi's are connected through a router (it could also have been a switch), with Ethernet cables. One Pi acts as a host, and needs the SD card with Raspbian full to boot. The other raspberries are then booted from the host via network booting [21]. This means that these clients have a shared boot location, giving them access to the same installed packages.
Every client Pi will have the ability to check what table-top modules are connected to it. These client Pi's are then able to perform the calculations for the models that correspond to those table-top modules. This information eventually has to appear at the dashboard, and the dashboard also specifies parameters (such as amount of solar panel). However, since these models are located on client Pi's, and the dashboard on the server Pi, a communication layer has to be implemented that works in a bidirectional way.

## 5.1. MQTT protocol

The MQTT (Message Queuing Telemetry Transport) protocol has two network entities, a server and a number of clients. The server is also called an MQTT broker, to which all messages will be sent from the clients. This broker ensures that all clients, that are subscribed to a certain "topic", receive their messages. Note that in this case, the server has a different meaning than in the server-client setup in booting, mentioned in the introduction of Chapter 5. Here, the Pi that has the SD card and is connected to a monitor, will be connecting as a client to the broker as well. Clients in the MQTT protocol are able to both publish messages with a certain topic, and subscribe to incoming messages of certain topics. The data to be sent is first converted into a JSON data (a lightweight format used for data interchanging), and then extracted back into its original form at the receiver(s).

Any already existing broker on the internet could have been used for exchanging these messages (such as *mqtt.eclipse.org*). However, creating a broker on the server Pi has the advantage that no internet communication is required, as well as the lower latency. Here, the MQTT will be using the TCP [22] protocol for the communication to and from the broker. In Fig. 5.1 an example of this is depicted. In this instance the program on the server Pi has a message for the clients, but this could of course also happen in a reversed order, where one of the clients has a message published with the topic: "to_server". The server would be already subscribed to this topic and receive that message.

In case multiple messages are received at the same time, they will be queued and received in order. The advantage of this MQTT protocol, is that any amount of clients can connect to it, without any changes to the protocol. This makes a very scalable system, which is needed to satisfy the scalability requirement in the Program of Requirements.

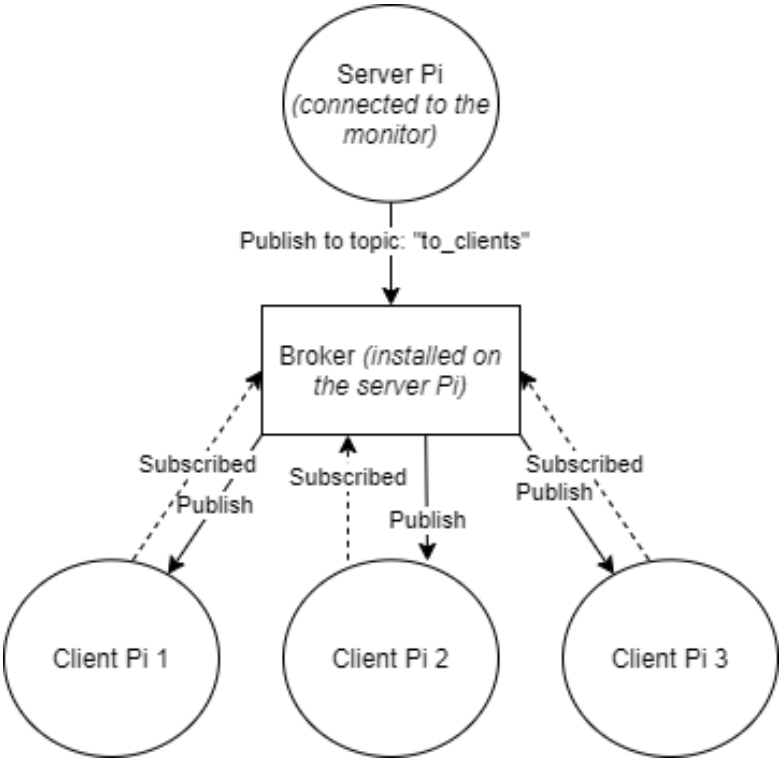



Figure 5.1: An example of the MQTT protocol, where a program on the server Pi has a message for the clients, these clients are already subscribed to the topic: ”to_clients”

### Implementation

The Mosquitto broker has been installed on the Raspberry Pi server. Additionally, the MQTT client Paho has been installed on all Pi’s, including the server. Both of these programs are from Eclipse [23]. The server would typically start the first message by specifying all parameters set by the Dashboard, such as the amount of solar panels. This message is then published with the topic ”solar”, all the clients are subscribed to this topic and will calculate the power flow for the solar energy generation for an entire year, if the solar panel table-top module is connected to the specific client. This calculation will be returned with the topic ”year_data”, to which the server is subscribed. The Python code for the MQTT that has been installed on the server, and the MQTT that has been installed on the clients, can be found in on the Github repository [24]..

## 5.2. Data gathering

### Year overview using weather data

For the Demonstrator some parameters for the renewable sources, as well as the load profile of residential houses, need to be downloaded beforehand. This data is gathered with an hourly resolution of the whole of 2019 for the location of Delft, and then stored locally. Regarding the wind turbine, the wind velocity is gathered. For the solar panel, wind velocity, together with ambient temperature, diffusion irradiance and direct irradiance. The diffusion irradiance is the irradiance that is reflected from mainly the clouds, which will be combined with the direct irradiance to form the global irradiance, i.e. all the irradiance that will be converted to energy on the solar panel. For these weather parameters, Renewables.ninja [7, 8] is used. Renewables.ninja has an API which allows the python script to open a link, set in the parameters for Delft and the angle of the solar panel. The script will then receive the weather data and store it in a JSON string locally in a text file. The angle of the solar panel is needed to get the diffusion and direct irradiance that would fall on the surface of the solar panel throughout the year.

The last data that needs to be gathered, is the residential load profile, for which a load profile generator tool is used, as described in the Household subsection in Section 4.2. LoadProfileGenerator 6.5.0 was used and a predefined settlement was simulated. The settlement was called 'AllCombinedHousehold' and contains 148 people and 62 households. This was simulated on two settings, one where the residents consumed energy as would result in the average of Germany, and one where the residents had a more energy conscious behaviour as well as more energy efficient devices, (overall the energy consumption was 32.6% less). This was then divided by 62 to get the load profile of one household. Finally it is saved locally in a text file with a JSON string format.

### Realtime inputs from table-top models

A second operation mode the Demonstrator can be put in, is using the table-top models as inputs, as is specified in Item 2. Table-top models can be freely connected to the Raspberry Pi's where the they detect what model is connected to it by means of a Tag-ID. Only when a Tag-ID is noticed will this model be active in the demonstrator.

In this mode, the current powers from the year overview are visualized on the table-top models, simulating one hour for each second that passes. Additionally, these table-top models should be able to override the powers of the year overview, by being interacted with. This means that for these "altered" powers, the value is directly pulled from such a model by means of a function.
This function starts when Dashboard gives an input in the form of an "hour" which triggers the function to start retrieving values from the table-top models. These values are then visualized back onto the Dashboard.

# 6

# Design of models

In this chapter, the actual implementation of the chosen models from Chapter 4 is explained. All of these models are implemented in Python, and they can be found in Appendix B, they can additionally be found on the Github repository [24].

## 6.1. Wind

To evaluate the power output of a wind turbine a power curve is used. This power curve resembles the power output of a wind turbine in relation to the wind speed. A power curve is based off manufacturer data values such as $v_{rated}$, $P_{rated}$, $v_{cut\_in}$ and $v_{cut\_off}$. To model the power curve a cubic function [12, 20] is used which is given by Eq. 6.1:

$$P_{windturbine}(v) = a \cdot v^3 - b \cdot P_{rated} \tag{6.1}$$

Where a and b in Eq. 6.1 are given by:

$$a = \frac{P_{rated}}{v_{rated}^3 - v_{cut\_in}^3} \tag{6.2}$$

$$b = \frac{v_{rated}^3}{v_{rated}^3 - v_{cut\_in}^3} \tag{6.3}$$

To determine the power output for a certain wind speed, Eq. 6.2 and Eq. 6.3 are first performed using the data sheet of the chosen wind turbine. These 2 constants are then filled in Eq. 6.1 along with the wind speed *v* and data sheet value $P_{rated}$ to obtain the power.
The power curve resulting from this equation is used to resemble region 2 seen in Fig. 4.1. As can be seen from Eq. 6.1, the power will always increase with increasing wind speed because of the third order polynomial. To counter this behaviour, region 3 has been implemented to match the power to $P_{rated}$ so linear, which is between the rated speed and the cut-off speed. As mentioned before, region 1 and 4 have no power output and are equal to 0. It is important to note that there is a mismatch between this cubic method approach, and wind turbines that stall after reaching the rated speed. This method assumes that at any speed higher than $v_{rated}$, the pitch angle of the blades is controlled in such a way that it prevents wind stalling and keeps the power constant.

As a power curve shows the relation between the wind speed and the power output of a wind turbine, it is therefore the case that any generator losses are included. Besides the generator losses, there are transmission losses to be accounted for due to the distance that needs to be travelled. Transmission losses are often minimal and even can be close to 0 for larger powers. For this reason transmission losses will be neglected as the wind turbines in this case are going to be relatively close to the loads. It will only have a minimal effect on the delivered power.

## 6.2. Solar

In Chapter 4 it is decided to model the power output of a solar panel according to data sheet values. By doing so, an estimation of the power is done based on actual measured values.

The schematic in Fig. 6.1 illustrates how the power output of a PV panel is modelled. To calculate the nominal power output, the solar irradiance is multiplied with the area of the PV module and with the efficiency of the panel at a temperature of 25 °C. Therefore this is the power at a module temperature of 25 °C. The irradiance is locally stored as described in Chapter 5. This irradiance is stored for three angle tilts: 30, 35 and 40 degrees. By setting these parameters when downloading the weather data, the different irradiance under different angles to the sun will be stored. The solar panels are assumed to be south-facing for Delft due to being north of the equator. PV panels however are temperature reliant, therefore an additional function to compensate for the temperature losses is mandatory.

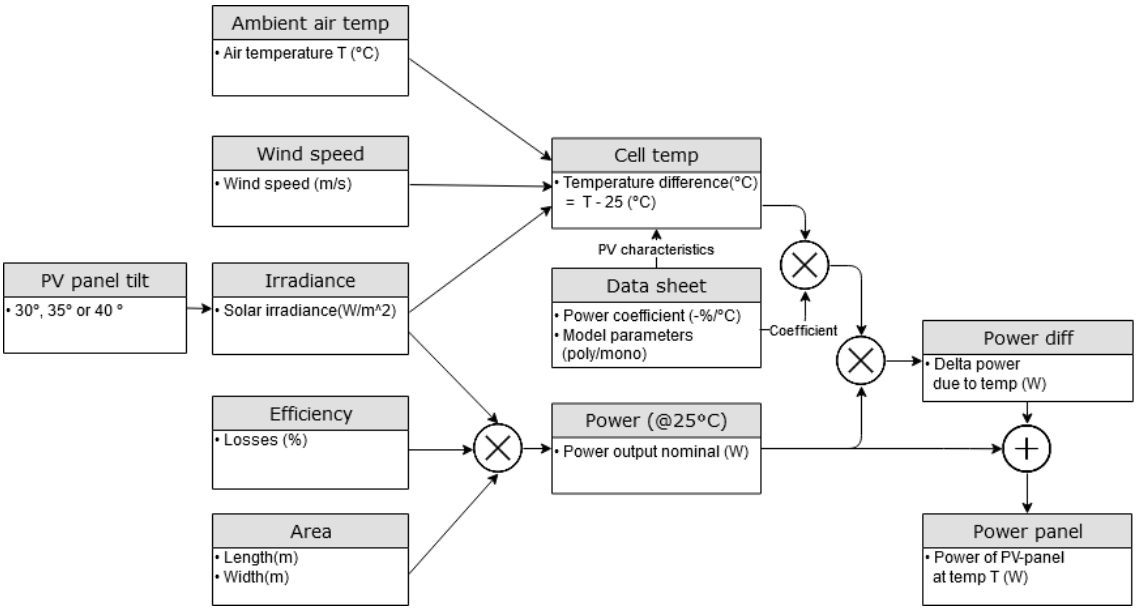



Figure 6.1: Schematic of PV panel power output.

Depending on the type of insulation, position and wind speed, the cell temperature can be different from the ambient temperature. For this, the *temperature.sapm_cell* function in the PVLib python module has been used [25]. This function calculates the cell temperature as per the Sandia Array Performance model [26]. This function requires the global irradiance, wind speed, ambient air temperature and the type of the solar panel module used.
The difference between the calculated cell temperature and the data sheet values operating temperature 25 degrees Celsius, is used to calculate the power deviation by means of a power coefficient. This power coefficient represents the degradation of power in % per increase of degree Celsius, due to PV modules performance going down for higher temperatures. This results in a percentage of power difference with the nominal power. The addition of the power difference and the nominal power results in the delivered power of the PV panel at temperature T.
Yet this is not the power delivered to the loads. PV panels actually create a DC output, but as the grid in the Netherlands functions with AC, an inverter must be installed. PV panels have their specific DC/AC inverter due to their operation characteristics. A good DC/AC inverter has a typical efficiency of above 95%. These inverter losses are included in the calculation by a multiplication factor with an efficiency factor. An inverter with the conversion efficiency of 97.2% is chosen for this case to limit these losses [27]. A final note must be made relating to the losses. It may be assumed that all PV panels are directly installed onto rooftops. For this reason, the transmission losses are minimal and can be neglected.

## 6.3. Households

The gathered load profiles, described in Section 5.2, are used to model one household. This means that two profiles are used, both modelling one household. One of the profiles is for the average electricity consumption and the other for the consumption of a household that is using less electricity. The one using less would have more efficient devices installed and disconnect appliances when they are not needed. The two parameters that are required are then the type of load (saving or average), and the amount of households that need to be modelled. This multiplier will be used with the modelling of one household to create larger size loads.

## 6.4. Battery

For the storage of any excessive energy coming from the renewable sources, solar and wind, a battery model has to be implemented. This battery model consists of multiple EV's (Li-ion) and a hydrogen storage. These storage options are modelled by means of a storage algorithm. A description of both storage options will be given first, with specifications of both storage options listed in Table 6.1.

### Electric Vehicle storage

The EV's are represented by Nissan Leaf e+ (June 2019) models. The choice for a Nissan Leaf is primarily based on an EV which is affordable for the average person. Besides, the Nissan Leaf e+ is fairly new which gives a good indication for what specifications an EV is going to have in the future. Nissan is currently looking for solutions where EV's can be used in a vehicle to grid (V2G) solution, bidirectional power flow.

The EV battery specifications are directly taken from the manufacturer database. To charge the vehicle, Magnum Cap's V2G charging station 2.0 will be used [28]. This charging station allows for bidirectional power flow and is ideal for this topology. The EV charging rate and efficiencies in Table 6.1 are pulled from the V2G station specs [28], since the conversion and control of power flow will happen in the station. The minimum SoC equals 20% which is required for Li-ion batteries to prevent capacity reduction. The total energy capacity of the Nissan Leaf e+ is taken from the database [29].

Table 6.1: Specifications of the EV [28, 29] and hydrogen storage [30, 31].

| | **Nissan Leaf e+** | **Hydrogen** |
|---|---|---|
| Energy capacity [kWh] | 56 | 396 |
| Depth of charge (DoC) [%] | 80 | 100 |
| Max state of charge (SoC) [%] | 100 | 100 |
| Power [kW] | 10 | 98 |
| Storage efficiency $\eta_s$ [%] | 90 | 70 |
| Production efficiency $\eta_p$ [%] | 93 | 60 |

### Hydrogen storage

The hydrogen storage model is characterized by an electrolyzer, a hydrogen tank and a fuel cell. This simplified model will represent the storage of hydrogen and can be seen in fig. 6.2. For the hydrogen model, the characteristics have been chosen from the average values of existing hydrogen storage in Europe [30]. The electrolyzer and fuel cell are chosen to be alkaline type, and have efficiencies $\eta_s$ and $\eta_p$ respectively. This is a design choice based on alkaline being most used in practice [30]. The storage of hydrogen is often accomplished with a compressed hydrogen tank. The tank's properties are calculated with the following specifications: if choosing a 300 L tank at 700 bar, and the density of hydrogen being 33 kg/m$^3$ (at 700 bar), then it leads to 12 kg of hydrogen gas being stored. The energy density of hydrogen gas is 33 kWh/kg. This results in one hydrogen storage tank having a capacity of 396 kWh.

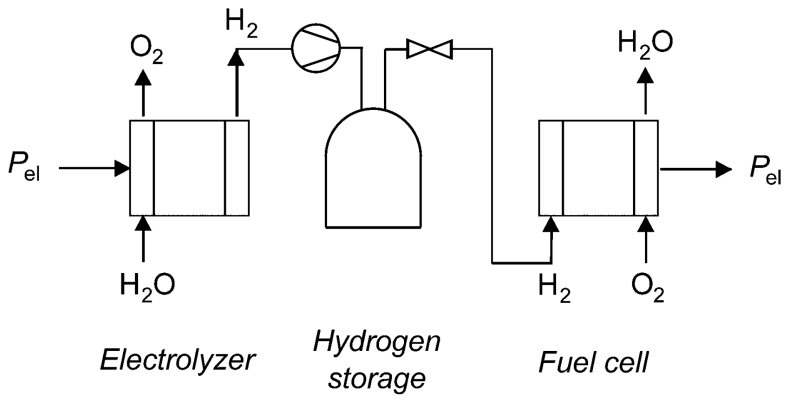



Figure 6.2: Hydrogen storage schematic[32].

## Energy storage algorithm

The algorithm for energy storage can now be derived using all characteristics listed above. This algorithm is designed based on the preference for EV storage before hydrogen storage, and as a last resort the grid. The power that has to flow to the storage, will be referenced intermittently as the amount of energy. However, since kWh is used and the resolution is hourly, this assumption is valid (storing 8 kW for an hour would mean 8 kWh of energy). An assumption was made, that the car would need 4 kWh for a round-trip to work, this equals 30 km and it is supposed to resemble the average distance to work and back, in the area of Delft, which is chosen as the preferred location for the case study (Section 2.2). Another assumption is that every car would have 3 days to be used to go to work in a workweek, and used only one day in weekend.

The algorithm for the batteries is depicted in Fig. 6.3 by means of a flow diagram. It represents one iteration of a loop, that goes through every hour of the year, meaning that every iteration the excess power flow is calculated for the next hour. For the modelling of the storage, objects were made of the cars and the hydrogen tank. So it is important to note that the first block in the flowchart, with ”Initialization”, will only happen for the first hour that the algorithm is called with. Every object has the two functions of being able to charge as well as discharge, and the EV additionally has a function of returning from work. The methods are explained in the text below:

- **Store power**
  A request of how much power has to be stored, will be sent to this method. This method then will calculate how much of this request it will be able to provide in terms of power, while having the constraints that the battery cannot be more than 100%, as well as that it cannot store at a rate faster than the max charging rate.

- **Take power**
  This method is analogues to the previous one, the difference for the EV is that instead of having the battery at lowest level of 0%, it will want to have the car at the 20% plus 4 kWh, this equals the DoC plus one round-trip to work, to ensure that the level will never be lower than the DoC.

- **Back from work** (EV only)
  Here, all cars are checked on if they need to be charged, because of the round-trip made, which will make their SoC go lower with 4 kWh and might cause the SoC to go lower than the limit. If that is the case, it will be charged to the DoC level plus 4 kwh.

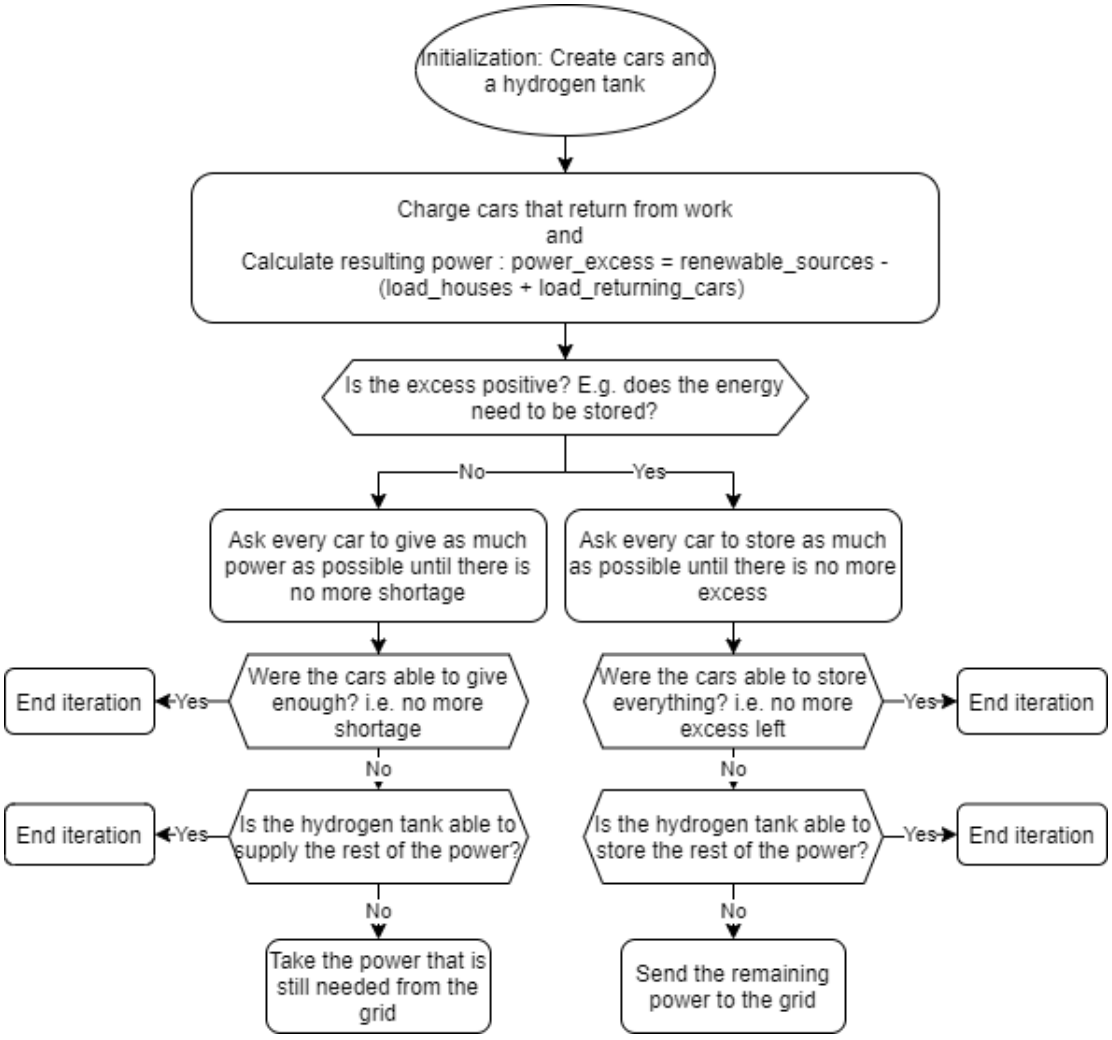



Figure 6.3: Flow diagram of battery algorithm.

# 7

# Verification

This chapter explains the integration of all models by means of simulations. First a validation of the individual components is given. After, simulations of the complete system are presented where these results will be discussed.

## 7.1. Integration of models

The Energy Data Widget contains several stages which are passed through before getting the correct output. The first is gathering data to retrieve inputs. The second, the communication between the Pi's, as the inputs are transferred to their right destination. After the inputs are received at the right model, each model except storage performs its operation. The storage model will be able to perform its calculations based on all the power lists that have been sent by the models on the client Pi's. Hence, all power lists are outputted to the correct destination, which is either the Dashboard or table-top models, or both simultaneously if required. The communication of the Pi's, sending of data between server and clients, and operation of the models is most important for a proper functioning of the data widget and will be discussed next.

## 7.2. Simulations

In this section all implemented models are tested by means of simulations. Each simulation shows certain characteristics which are elaborated. The simulations show the performance of the models with different settings. The results of the simulations all together will be discussed further on to show the performance off the entire system.

## Model simulations & results

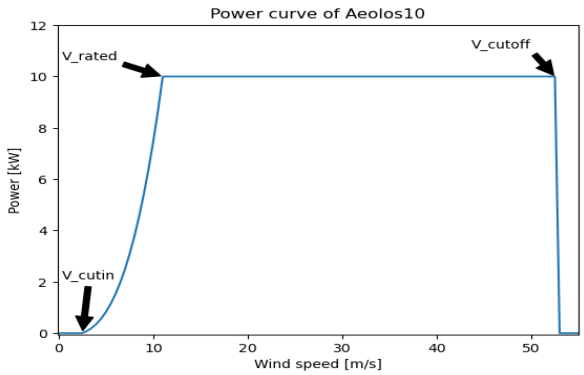
### Wind model

To verify the behaviour of the representing polynomial function of a power curve, 2 types of wind turbines are simulated. Both turbines have a set of manufacturer data values listed in Table 7.1 which are of importance. Based on these values a power curve can be modelled with a cubic function. In fig. 7.1 both turbines power curves are depicted. These turbines have very different operation values, but they show a very similar pattern. The function starts at the cut_in speed and ends at the rated speed. They also both maintain their rated powers until the cut_out speed. This means the function works properly with different turbines.

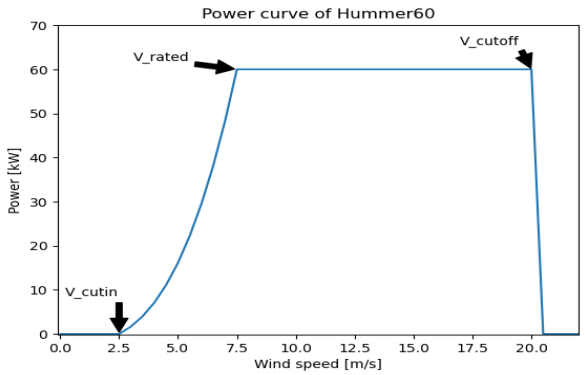
The polynomial function starts very precisely at the given cut_in and ends at the rated speed which is not always the case in practice, due to unpredictability's. Nonetheless, it can be concluded that the function visualises the shape and characteristics of a power curve pretty accurate, and that the datasheet values are interpolated well.

| Model | $V_{rated}$ | $V_{cutin}$ | $V_{cutoff}$ | $P_{rated}$ |
|---|---|---|---|---|
| Hummer H25.0-60KW | 7.5 m/s | 2.5 m/s | 20 m/s | 60 kW |
| Aeolos Aeolos-V 10kW | 11 m/s | 2.5 m/s | 52.5 m/s | 10 kW |

Table 7.1: Wind turbine specifications [33].




(a) Power curve of turbine 'Aeolos Aeolos-V 10kW'.




(b) Power curve of turbine 'Hummer H25.0-60KW'.

Figure 7.1: Power curves created by cubic function.

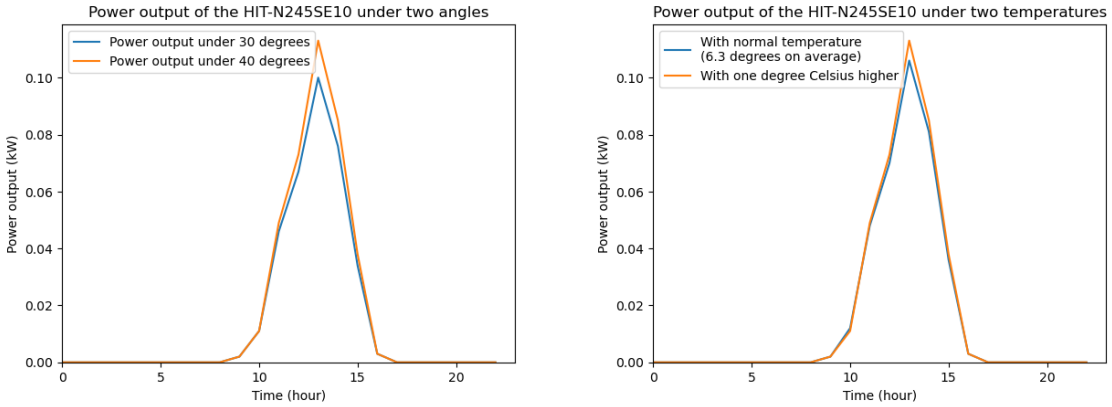
**Solar model**

The PV panel output will be simulated for the angle of 30 and 40 degrees for a time period of 24 hours in the weather data. Here it is expected that the angle of 40 will have more power output for Delft [34]. After that, a plot of the whole year under 40 degrees angle will be shown to see if the summer indeed has more power production than the winter. For all these simulations the HIT-N245SE10 will be used, with the specifications in Table 7.2.

In Fig. 7.2a the output can be seen for the solar panel under two different angles, the panel that is 40 degrees orthogonal to the earth is indeed most optimal. In Fig. 7.3, the power output of the same panel can be seen for a whole year. In the summer, the output is visibly higher. To verify the influence of the temperature in the model, as explained in 6.2, a plot of 24 hours is shown in Fig. 7.2b under a 1 degree Celsius temperature difference. Here the increase of one degree Celsius had an impact of 6.6% higher peak in the power output.

Table 7.2: Solar panel specifications [35].

| Model | Length | Width | Efficiency | T Coefficient |
|---|---|---|---|---|
| HIT-N245SE10 | 1.580 m | 0.798 m | 19.4% | -0.258 % / °C |




(a) The power output of the HIT-N245SE10 under 30, and 40 degrees orthogonal to the earth. All other parameters are the same.

(b) The power output of the HIT-N245SE10 under normal circumstances, and with one degree higher in every point.

Figure 7.2: Power output of the HIT-N245SE10 for January first.

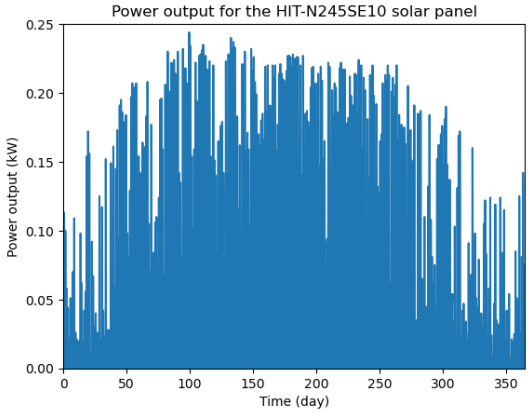



Figure 7.3: The power output of the HIT-N245SE10 for a whole year, with a panel tilt of 40 degrees.

### Household model

The load which is modelled in this system consists of households. Households often have a very particular energy consumption pattern. Fig. 7.4 depicts the energy profiles for one household of both types of household, energy-saving and average. For this particular day (first of January), there is a 36% decrease in consumption for the energy-saving household with respect to the average household. It can be seen that during prime-time, the peaks occur. And the lowest activity, is after 11PM and before 6AM.

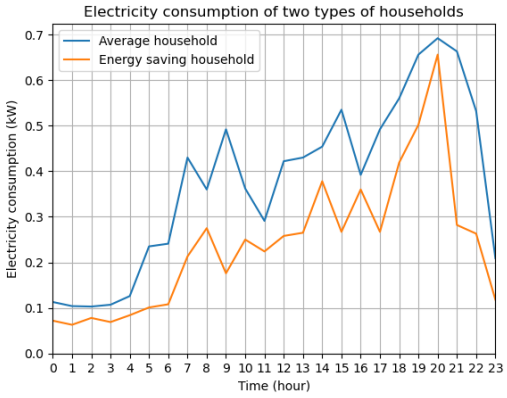
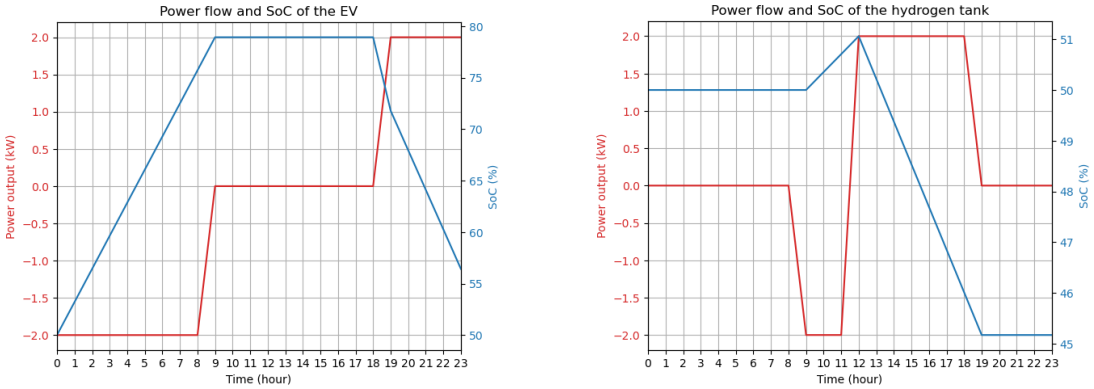



Figure 7.4: The electricity profile of two types of households, one that is saving energy and one of the average household.

### Battery model

To test the battery model by itself, both normal and extreme conditions will be used as test cases. For the normal conditions, a full day will be simulated. During this day, the sources will dominate for the first half of the day, generating a positive excess of power. In the second half of the day, the opposite happens. After instantiating the tank and the cars, the storage model is supposed to first store energy for the first half, and then provide energy for the second half of the day. For all test cases one EV and one hydrogen tank are used. In Fig. 7.5 the storage model is shown, where there is an excess of 2kW per hour for the first half of the day, and a shortage of 2kW per hour for the second half of the day, so from 12pm. The storage reacts to this by first storing that power in the EV, until the hydrogen tank is needed due to the EV being absent (at work). When the storage system has to deliver, the EV is at work which means the hydrogen tank must deliver until 6PM, which is when the EV returns from work and starts delivering the 2kW.

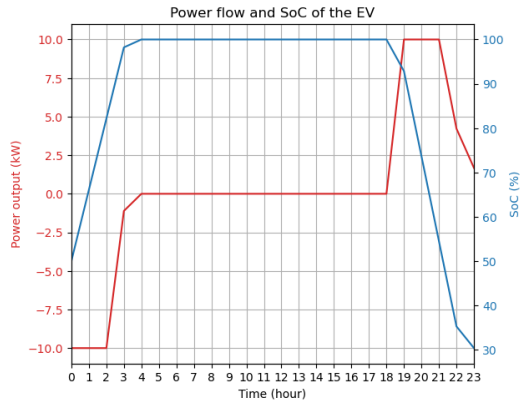



(a) SoC and power flows of an EV under normal circumstances. Negative power flow means that the EV is charging.

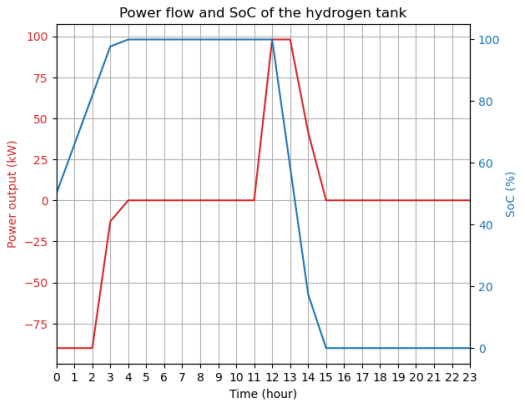
(b) SoC and power flows of a hydrogen tank under normal circumstances. Negative power flow means that the tank is charging.

Figure 7.5: Power flows and SoC of the storage model when charging and discharging. The grid is not used during this period.

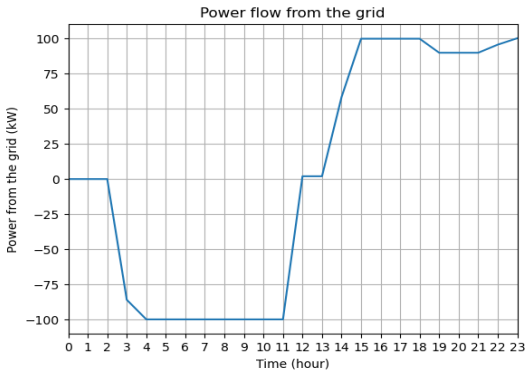
In Fig. 7.6 it is depicted how the storage model reacts under extreme circumstances. This condition holds that 100kW per hour has to be stored for the first half of the day and 100kW per hour has to be supplied by the batteries in the second half, so from 12PM. Both storage options will be used in the first hours since the EV cannot store enough on its own. From 2 AM both the EV and Hydrogen tank are near capacity, and the grid has to be utilized for the excess power in the system. In the second half of the day it can be seen that the EV starts supplying at 7PM after it returns from work, where it has lost some charge due to the journey from work, seen from the increased SoC drop from hour 6 to 7PM, with respect to the hours after that. The grid must also deliver as the EV and hydrogen tank can not sufficiently deliver enough power.




(a) SoC and power flows of an EV under extreme circumstances. Negative power flow means that the EV is charging.




(b) SoC and power flows of a hydrogen tank under extreme circumstances. Negative power flow means that the tank is charging.




(c) Power flows of the national grid under extreme circumstances. Negative power flow means that power is being supplied to the grid.

Figure 7.6: Power flows and SoC of the storage model when charging and discharging under extreme circumstances.

## 7.3. System simulation & results

The system as a whole can be tested by adding all the individual components together. All the model's power outputs except for the battery model are calculated with certain parameters, namely the type or amount of each model and the downloaded weather data of 2019. This information containing a year of powers per model will be sent by means of a working MQTT protocol from clients to the server. At the server, the battery algorithm is applied to the incoming powers. For the setup of the whole system, the parameters are chosen to be:

- For the wind power 2 wind turbines of type: ”Aeolos10” are chosen.
- For the solar power 200 ”HIT-N245SE10” solar panels are chosen.
- For the load a household demand power is created with 40 households with an average energy consumption pattern.
- For the short term storage an amount of 10 EV's is set.
- For the long term storage one hydrogen tank is set.

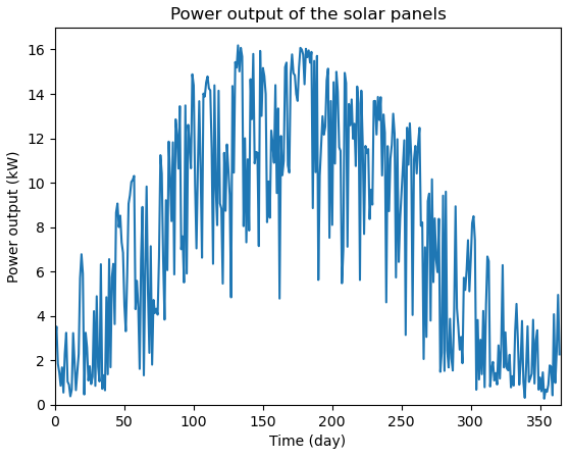
The results of these simulations can be found in Fig. 7.7. The averages of each day have been taken and presented to have a clearer overview. In Fig. 7.7a to c, the two renewable sources and the load have been depicted, and in d, the resulting powers from these are calculated. As can be seen, there is more energy to be stored in the summer. The total source power has been rectified a bit due to the turbines generating more in the winter, and the solar panels generating more in the summer. But the load in the summer is generally less, creating more power excess in the summer. In Fig. 7.7e to g, the response of the batteries on this power excess is shown. The EV indeed never fully depletes and the SoC of both the EV and hydrogen are generally higher in the summer. Even to a point where some excess power can be sold to the grid.

### Delays

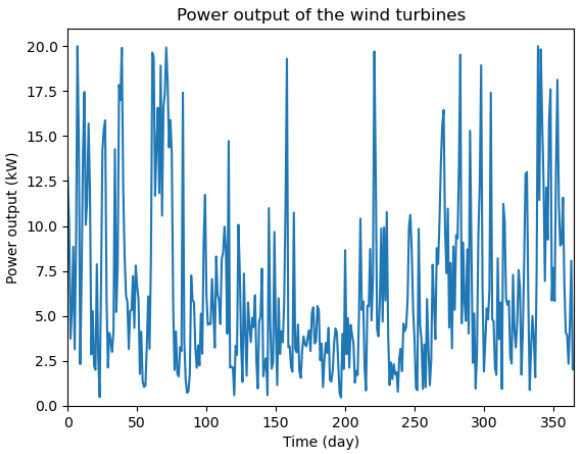
The timings of the calculations on the Raspberry Pi's has been tested for a few simple cases including a wind model, solar model, load model and storage model with varying amounts of EV's. This means that the time between sending the parameters for a model, to the time that these results arrived at the server are measured. These results were measured after the Demonstrator was initialized. Since python reads all files at the start of the simulation, the execution times might have been longer for the first request. The measured times were very consistent and their maximal value is given in Table 7.3. As can be seen from the execution times, the only real bottleneck is the storage model. With mainly the amount of EV's being the cause. This is due to the fact that an object is made for every car. Each car has its own SoC and schedule of days that it is at work. This makes for a long execution time if there are a lot of cars. This means that the speed of the battery model is not in line with the speed trade-off requirement in the PoR.
For the MQTT setup, the procedure in Chapter 5 has been followed, with one Raspberry Pi 3B+ as server, and one as client.

Table 7.3: Execution time for model requests.

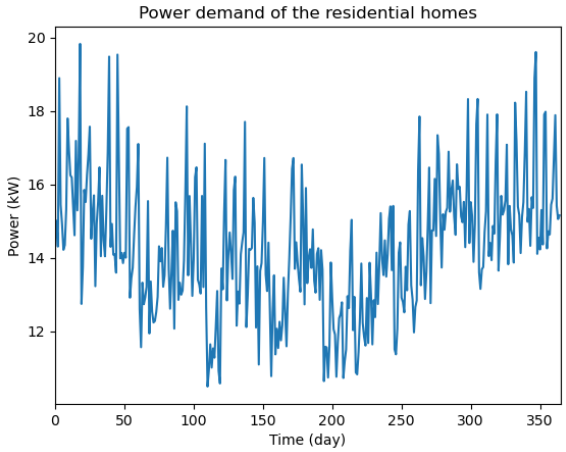
| **Model type** | Wind | Solar | Load | Battery (0 EV) | Battery (25 EV) | Battery (50 EV) |
|---|---|---|---|---|---|---|
| **Execution time (s)** | 0.185 | 0.092 | 0.095 | 0.47 | 6.3 | 12.2 |



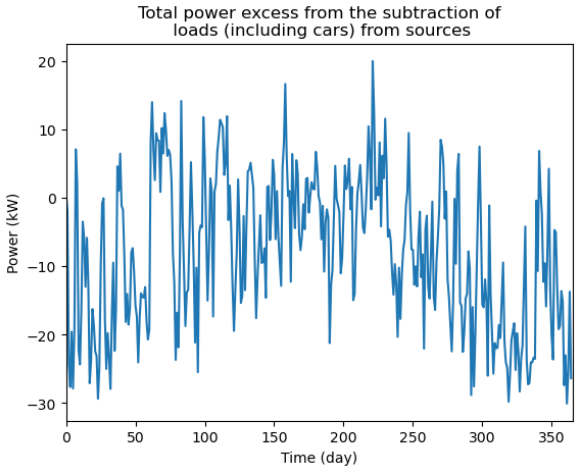

(a) Solar power



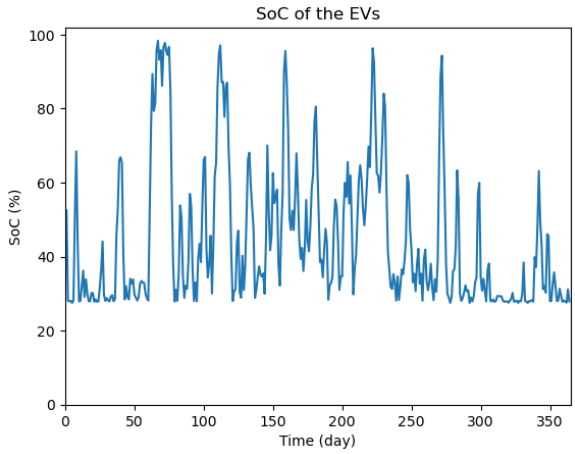

(b) Wind power



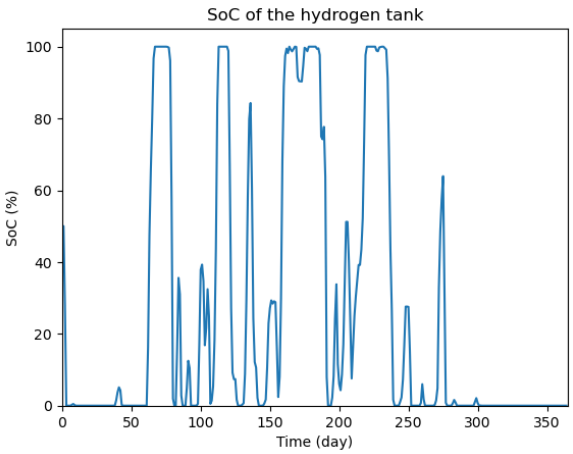

(c) Load power



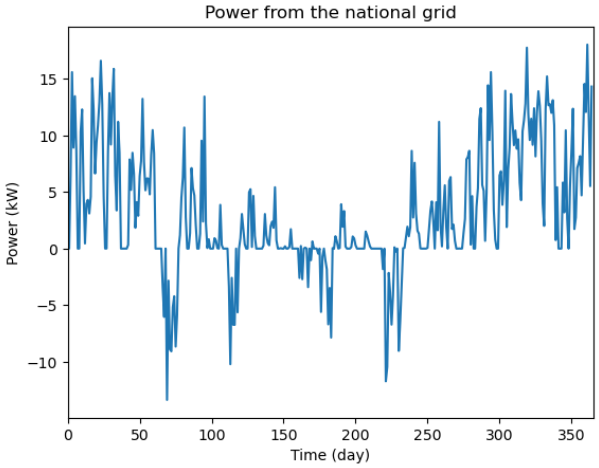

(d) Excess power, equals subtraction of the load from the sources, where the charging of the cars returning from work are added to the load.




(e) SoC of all EV's together.




(f) SoC of hydrogen storage tank.




(g) Positive power is take from grid.

Figure 7.7: A system overview including all relevant powers in the system in the year 2019.

# 8

# Discussion and conclusion

A set of source and load models have been chosen and developed. Each chosen model has been simulated using Python. Each Raspberry Pi is performing calculations based on the desired setting and can communicate through a working protocol with the other Pi's. All models should calculate and output accurate powers based on either real weather data or from table-top models. Unfortunately enough, no testing could be done regarding the real-time simulations. This is due to the fact that the Hardware Hub could not get to the production phase of the table-top models. These simulations could only be done for the yearly overview, where no real-time inputs were needed.
The simulations show that all models present very accurate power data concerning the available inputs in this system. The power curves represent the manufacturer power curves very closely, the solar panels resemble their corresponding data sheets well and are temperature dependent as expected. The battery algorithm and load show the working storage algorithm pattern and balances well the fluctuations in the grid.
All calculations were done only when needed, as described in 3, this prevented data cluttering and performing calculations unnecessarily, making the system overall more efficient. To conclude, all requirements from the Chapter 2 (PoR) have been achieved except for the interactive element which could not be tested due to unfortunate events.

## Future work

There are a few improvements that are possible in future, similar systems. The improvements consist of the complexity, the magnitude, the speed and the adaptability concerning the ”Energy Data Widget”.

- The detailing of the models is currently limited to realize the necessary powers to create a working system. This is achieved very well regarding the requirements and the functioning of the demonstrator. However increasing the scope of the models may increase accuracy or give extra information to other parts in your system. By using more detailed approaches, such as voltage current relations, and implementing them through Python scripts this can be achieved.
- The magnitude of the system can be increased by adding models. The addition of models creates a larger grid to be visualized giving an even better insight for the user. For example transmission lines or hydro-power could be added. The size of the total system increases and therefore more options can be considered for different case studies. This is possible due to the scalability of the demonstrator.
- Another often important considered improvement is the speed. Increasing the speed leads to better performance. For this case study, all models perform with enough speed, except for the

storage model. This is mainly because of the EV objects created in the python code. A different approach could be taken as considering all the EV's together as one large battery, instead of several individual ones.
Another approach to this, would be to let one of the Raspberry Pi clients keep track of which EV's are able to load and store, this type of parallel programming would greatly enhance the speed.
Moreover, Vectorization could be used. This is the use of arrays in Python, instead of *for loops*. This greatly enhances the speed but also adds a level of complexity to the code.
The last measure that could be taken, is using a stronger server instead of a Raspberry Pi. It is recommended to have a stronger server, as the clients profit from the faster performance from the server. However, this would add to the overall cost to the system, which would go against the trade-off requirement of the system being low in cost.

- To add to the adaptability, the downloaded weather data could be retrieved dynamically. This is needed in order to make the Demonstrator more accurate in places besides the Netherlands, which is where the current case study is located.

A key part of the demonstrator is the interactive component which it has. Unfortunately these components were not able to be properly tested due to certain circumstances. Therefore a future demonstrator requires the complete testing off the actual Hardware Hub with all its sensors and actuators. This implementation would enable improving the Demonstrator as a whole, based on the feedback of the table-top models.

# Bibliography


[1] D. Roberts, “New global survey reveals that everyone loves green energy - especially the chinese,” Nov 2017. [Online]. Available: https://www.vox.com/energy-and-environment/2017/11/20/16678350/global-support-clean-energy

[2] A. Alexandru, E. Tudora, O. Bica, and R. Mayer, “Educating young people for sustainable energy development,” *Bulletin of Electrical Engineering Faculty*, vol. 11, pp. 53–57, 11 2011.

[3] K. Mulder, J. Segalas, and D. Ferrer-Balas, “Educating engineers for/in sustainable development? what we knew,what we learned, and what we should learn,” *Thermal Science*, vol. 14, pp. 625–639, 01 2010.

[4] B. A. Bossink, “Demonstrating sustainable energy: A review based model of sustainable energy demonstration projects,” *Renewable and Sustainable Energy Reviews*, vol. 77, pp. 1349 – 1362, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032117302010

[5] “Esdl documentation.” [Online]. Available: https://energytransition.gitbook.io/esdl/

[6] N. Blair, A. P. Dobos, J. Freeman, T. Neises, M. Wagner, T. Ferguson, P. Gilman, and S. Janzou, “System advisor model, sam 2014.1.14: General description,” 2 2014.

[7] Renewables.ninja, “Renewables.ninja.” [Online]. Available: https://www.renewables.ninja/

[8] Pfenninger, Stefan, and Iain, “Long-term patterns of european pv output using 30 years of validated hourly reanalysis and satellite data, by pfenninger, stefan; staffell, iain,” Jan 1970. [Online]. Available: https://ideas.repec.org/a/eee/energy/v114y2016icp1251-1265.html

[9] J. Wang, Y. Ma, Z. Hu, and X. Yang, “Modeling and real-time simulation of non-grid-connected wind energy conversion system,” in *Proc. World Non-Grid-Connected Wind Power and Energy Conf*, 2009, pp. 1–5.

[10] A. Smets and M.Cvetkovic, “Ee2e21 sustainable energy supply,” 2019–2020. [Online]. Available: https://brightspace.tudelft.nl/

[11] M. Ragheb and A. M. Ragheb, “Wind turbines theory-the betz equation and optimal rotor tip speed ratio,” *Fundamental and advanced topics in wind power*, vol. 1, no. 1, pp. 19–38, 2011.

[12] Sohoni, Gupta, and R. K., “A critical review on wind turbine power curve modelling techniques and their applications in wind based energy systems,” Jul 2016. [Online]. Available: https://www.hindawi.com/journals/jen/2016/8519785/

[13] F. E. Tahiri, K. Chikh, M. Khafallah, A. Saad, and D. Breuil, “Modeling and performance analysis of a solar pv power system under irradiation and load variations,” in *2017 14th International Multi-Conference on Systems, Signals Devices (SSD)*, 2017, pp. 234–238.

[14] E. Duverger, C. Penin, P. Alexandre, F. Thiery, D. Gachon, and T. Talbert, “Pv modeling methods of an off-grid experimental microgrid,” in *2018 IEEE International Conference on Industrial Technology (ICIT)*, 2018, pp. 1061–1066.

[15] “Data platform - data package household data,” Apr 2020. [Online]. Available: https://data.open-power-system-data.org/household_data/2020-04-15/

[16] N. Pflugradt and U. Muntwyler, “Synthesizing residential load profiles using behavior simulation,” *Energy Procedia*, vol. 122, pp. 655–660, 2017. [Online]. Available: www.loadprofilegenerator.de

[17] Y. Zhang, A. Lundblad, P. E. Campana, and J. Yan, “Comparative study of battery storage and hydrogen storage to increase photovoltaic self-sufficiency in a residential building of sweden,” *Energy Procedia*, vol. 103, pp. 268 – 273, 2016, renewable Energy Integration with Mini/Microgrid – Proceedings of REM2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1876610216314941

[18] E. Gray, C. Webb, J. Andrews, B. Shabani, P. Tsai, and S. Chan, “Hydrogen storage for off-grid power supply,” Oct 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0360319910019270

[19] J. W. . S. Paul Gipe, “Wind turbine rating,” in *Wind Energy Comes of Age*, vol. 1. New York, 1995, pp. 155–161.

[20] M. Deshmukh and S. Deshmukh, “Modeling of hybrid renewable energy systems,” *Renewable and Sustainable Energy Reviews*, vol. 12, no. 1, pp. 235 – 249, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1364032106001134

[21] “Network boot your raspberry pi.” [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/bootmodes/net_tutorial.md

[22] “Specification of internet transmission control program.” [Online]. Available: https://tools.ietf.org/html/rfc675

[23] I. Eclipse Foundation, “The platform for open innovation and collaboration: The eclipse foundation.” [Online]. Available: https://www.eclipse.org/

[24] D. van Paassen, B. Visser, V. van Doorn, J.-P. Rozestraten, R. Siemensma, and S. Ashraf, “Demonstrator.” [Online]. Available: https://github.com/danielvanpaass/Demonstrator

[25] W. F. Holmgren, C. W. Hansen, and M. A. Mikofski, “pvlib python: a python package for modeling solar energy systems,” Sep 2018. [Online]. Available: https://doi.org/10.21105/joss.00884

[26] D. L. King, W. E. Boyson, and J. A. Kratochvil, “Photovoltaic array performance model,” Dec 2004. [Online]. Available: https://prod-ng.sandia.gov/techlib-noauth/access-control.cgi/2004/043535.pdf

[27] [Online]. Available: https://jimmeijer.com/sma-sunny-boy/

[28] *V2G*, Magnum Cap. [Online]. Available: https://magnumcap.com/v2g/v2g/

[29] “Nissan leaf e+.” [Online]. Available: https://ev-database.nl/auto/1144/Nissan-Leaf-eplus

[30] G. Gahleitner, “Hydrogen from renewable electricity: An international review of power-to-gas pilot plants for stationary applications,” *International Journal of Hydrogen Energy*, vol. 38, no. 5, pp. 2039 – 2061, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360319912026481

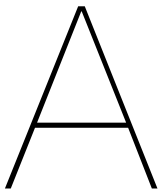
[31] U. D. of Energy. (2015) Fuel cells. [Online]. Available: https://www.energy.gov/sites/prod/files/2015/11/f27/fcto_fuel_cells_fact_sheet.pdf

[32] M. A. Pellow, C. J. Emmott, C. J. Barnhart, and S. M. Benson, “Hydrogen or batteries for grid storage? a net energy analysis,” *Energy & Environmental Science*, vol. 8, no. 7, pp. 1938–1952, 2015.

[33] L. Bauer and S. Matysik, *Wind turbine models*, 2011–2020. [Online]. Available: https://en.wind-turbine-models.com/turbines

[34] F. Biljecki, G. Heuvelink, H. Ledoux, and J. Stoter, “Propagation of positional error in 3d gis: estimation of the solar irradiation of building roofs,” *International Journal of Geographical Information Science*, vol. 29, pp. 2269–2294, 12 2015.

[35] *Photovaltoaic module*, Panasonic.



# List of Hardware

- Raspberry Pi 3B+
- PC with Python 3.7 for simulations
- UHS-I microSD 600x for the server Pi
- Linksys WRT3200ACM router

# B

# Python Code

## B.1. Models

### Solar model

```
"""An implementation of a solar energy model"""

# Datasheet imported values
PV_PARAMETERS = {
    'RSM72-6-360M': {'length': 1.956, 'width': 0.992, 'efficiency': 0.186,
     ↪  't_coefficient': -0.39},
    'HIT-N245SE10': {'length': 1.580, 'width': 0.798, 'efficiency': 0.194,
     ↪  't_coefficient': -0.258},
    'JAP60S01-290/SC': {'length': 1.689, 'width': 0.996, 'efficiency':
     ↪  0.171, 't_coefficient': -0.37},
}


# Power calculation with ambient temp as operating temp in KW
def power_calc_solar(irradiance, temperature, pv_type):
    selectedpv = PV_PARAMETERS[pv_type]
    p_nom = selectedpv['length'] * selectedpv['width'] *
     ↪  selectedpv['efficiency'] * irradiance
    p_out = (((selectedpv['t_coefficient'] * (temperature - 25)) / 100) *
     ↪  p_nom) + p_nom
    return p_out*0.972
```

### Wind model

```
import numpy as np

WIND_PARAMETERS = {
    'Hummer60': {'P_rated': 60000, 'V_rated': 7.5, 'height': 12,
     ↪  'cut_inspeed': 2.5, 'cut_outspeed': 20,
                 'diameter': 25},
    'Aeolos10': {'P_rated': 10000, 'V_rated': 11.0, 'height': 6,
     ↪  'cut_inspeed': 2.5, 'cut_outspeed': 52.5,
                 'diameter': 5.5},
```

```
}


# calculated wind power in KW
def power_calc_wind(wind_speed, turbine_type):
    """Model wind turbine by cubic function based on data sheet values"""
    selected_turbine = WIND_PARAMETERS[turbine_type]

    wind_speed = np.array(wind_speed)

    winddelta = wind_speed ** 3 - selected_turbine['cut_inspeed'] ** 3

    wind_power = winddelta * selected_turbine['P_rated'] / (
            selected_turbine['V_rated'] ** 3 -
             ↪  selected_turbine['cut_inspeed'] ** 3)

    for x in range(0, len(wind_speed)):
        if wind_speed[x] < selected_turbine['cut_inspeed'] or wind_speed[x]
         ↪  > selected_turbine['cut_outspeed']:
            wind_power[x] = 0

    for x in range(0, len(wind_speed)):
        if wind_power[x] > selected_turbine['P_rated']:
            wind_power[x] = selected_turbine['P_rated']

    # power output in kW
    p_out_wind = wind_power / 1000
    return p_out_wind
```

### Battery model

```
import numpy as np
import random

class Car():
    def __init__(self, SoC):
        self.SoC = SoC
        self.powerMax = 10
        self.energyMax = 56
        self.energyMin = 0.2 * 56 + 4  # 20% plus enough for round trip to
         ↪  work
        self.currentEnergy = SoC * self.energyMax
        self.workdays = random.sample(range(5), 3) + [
            (random.randint(5, 6))]  # every car leaves home 3x from
             ↪  workdays, + 1 in the weekend, 0 means monday
        self.efficiency_store = 0.9  # percentage
        self.efficiency_take = 0.93  # percentage

    # check if car needs to charge
    def needCharging(self):
        if self.currentEnergy <= self.energyMin:
```

```
            self.currentEnergy = self.currentEnergy + 2 *
             ↪ self.efficiency_store
            return 2
        else:
            return 0

    def getSoC(self):
        self.SoC = self.currentEnergy / self.energyMax
        return self.SoC

    # this method can be requested with a certain power that needs to be
     ↪ discharged, and will check how much of it
    # can be provided
    def takePower(self, power, day, hour):
        if day in self.workdays and 9 <= hour <= 18:
            return 0
        energy_surplus = max(self.currentEnergy - self.energyMin,
                             0) * self.efficiency_take
        power_out = min(energy_surplus, self.powerMax,
                        -power)
        self.currentEnergy = self.currentEnergy - power_out /
         ↪ self.efficiency_take
        return power_out

    def storePower(self, power, day, hour):
        if day in self.workdays and 9 <= hour <= 18:
            return 0
        energy_chargeble = max(self.energyMax - self.currentEnergy,
                               0) / self.efficiency_store
        power_out = min(energy_chargeble, self.powerMax,
                        -power)
        self.currentEnergy = self.currentEnergy + power_out *
         ↪ self.efficiency_store
        return -power_out

    # this function is to be called at the end of the day, cars that were
     ↪ at work will now have less charge
    def returnFromWork(self, day):
        if day in self.workdays:
            self.currentEnergy = self.currentEnergy - 4  # takes 4 kwh, a
             ↪ round trip to work
            return 1
        else:
            return 0


class HydrogenTank():
    def __init__(self, SoC, energyMax):
        self.SoC = SoC
        self.powerMax = 98  # kwh
        self.energyMax = energyMax
```

```
        self.currentEnergy = SoC * self.energyMax
        self.efficiency_store = 0.70
        self.efficiency_take = 0.60

    def takePower(self, power):
        energy_surplus = max(self.currentEnergy,
                             0) * self.efficiency_take

        power_out = min(energy_surplus, self.powerMax,
                        -power)
        self.currentEnergy = self.currentEnergy - power_out /
         ↪  self.efficiency_take
        return power_out

    def storePower(self, power):
        energy_chargeble = max(self.energyMax - self.currentEnergy,
                               0) / self.efficiency_store
        power_out = min(energy_chargeble, self.powerMax,
                        -power)
        self.currentEnergy = self.currentEnergy + power_out *
         ↪  self.efficiency_store
        return -power_out

    def getSoC(self):
        self.SoC = self.currentEnergy / self.energyMax
        return self.SoC

    def resetSoC(self, SoC):
        self.SoC = SoC
        self.currentEnergy = SoC * self.energyMax


def power_battery(powers, N_EV, N_hydro):
    """Go through the list of all powers and decide how the storage should
     ↪  be charged/discharged"""
    if 'power_load' in powers:
        power_load = np.array(powers['power_load'])
    else:
        power_load = np.zeros(8760)
    power_source = np.zeros((len(power_load)))
    if 'power_solar' in powers:
        power_source = power_source + np.array(powers['power_solar'])
    if 'power_wind' in powers:
        power_source = power_source + np.array(powers['power_wind'])
    PEV_out = np.zeros((len(power_load),))
    PH_out = np.zeros((len(power_load),))
    Pgrid_out = np.zeros((len(power_load),))
    EV_SoC_out = np.zeros((len(power_load),))
    H_SoC_out = np.zeros((len(power_load),))
    excess_power_out = np.zeros((len(power_load),))
    EV_load_out = np.zeros((len(power_load),))
```

```
cars = []
# create all cars
for x in range(N_EV):  # create a new cars list for the realtime
 ↪  battery as well
    car = Car(0.5)
    cars.append(car)
# create Hydrogen tank
hydro = HydrogenTank(0.5, N_hydro * 396)

# go through the year
eps = 0.00000001  # define epsilon for comparisons with floats
for x in range(0, len(power_load)):
    excess_power = power_source[x] - power_load[x]
    day = x // 24
    day_of_week = day % 7
    hour_of_day = x % 24
    EV_load = 0
    if hour_of_day == 19:  # return from work
        for i in range(0, N_EV):
            cars[i].returnFromWork(day_of_week)
    for i in range(0, N_EV):
        EV_load = EV_load + cars[i].needCharging()
    excess_power = excess_power - EV_load
    EV_load_out[x] = EV_load
    excess_power_out[x] = excess_power
    for i in range(0, N_EV):
        EV_SoC_out[x] = EV_SoC_out[x] + cars[i].getSoC() / N_EV * 100
         ↪  # store the average SoC in %
    if N_hydro:
        H_SoC_out[x] = hydro.getSoC() * 100
    if excess_power > 0:  # positive so needs to store energy
        stored_power_EV = 0
        for i in range(0, N_EV):
            stored_power = cars[i].storePower(-excess_power,
             ↪  day_of_week, hour_of_day)
            stored_power_EV = stored_power_EV + stored_power
            excess_power = excess_power + stored_power
            # the storePower function returns for example -4, meaning 4
             ↪  kwh has been stored, so update excess
            # power on this
            if abs(
                    excess_power) < eps:
                break
        PEV_out[x] = stored_power_EV
        if excess_power > eps and N_hydro:  # this means that the cars
         ↪  had not enough to store the kwh

            stored_power = hydro.storePower(-excess_power)
            PH_out[x] = stored_power
            excess_power = excess_power + stored_power
        # assign what's left to the grid
```

```
            if excess_power > eps:
                stored_power = -excess_power
                Pgrid_out[x] = stored_power
        else:  # negative so needs to take energy from batteries
            power_taken_EV = 0
            for i in range(0, N_EV):
                power_taken = cars[i].takePower(excess_power, day_of_week,
                 ↪ hour_of_day)
                power_taken_EV = power_taken_EV + power_taken
                excess_power = excess_power + power_taken
                if abs(
                        excess_power) < eps:
                    break
            PEV_out[x] = power_taken_EV
            # cars could not provide enough, look at the hydro tank
            if abs(excess_power) > eps and N_hydro:
                power_taken = hydro.takePower(excess_power)
                PH_out[x] = power_taken
                excess_power = excess_power + power_taken
            if abs(excess_power) > eps:
                power_taken = -excess_power
                Pgrid_out[x] = power_taken
    # Assign the values for export
    Pgrid = np.around(Pgrid_out.astype(np.float), 3)
    PH = np.around(PH_out.astype(np.float), 3)
    PEV = np.around(PEV_out.astype(np.float), 3)
    EV_SoC = np.around(EV_SoC_out.astype(np.float), 3)
    excess_power = np.around(excess_power_out.astype(np.float), 3)
    H_SoC = np.around(H_SoC_out.astype(np.float), 3)
    EV_load = np.around(EV_load_out.astype(np.float), 3)
    data = {'power_grid': Pgrid.tolist(), 'power_EV': PEV.tolist(),
     ↪ 'power_hydrogen': PH.tolist(),
            'EV_SoC': EV_SoC.tolist(), 'excess_power':
             ↪ excess_power.tolist(), 'H_SoC': H_SoC.tolist(),
            'EV_load': EV_load.tolist()}
    return data
```

## B.2. Power calculation script

This script calls the solar, wind and load model with the right weather data and parameters.

```
import json
import numpy as np
import pvlib
import windmod
import pvmod

# choose good model params
temp_params =
 ↪ pvlib.temperature.TEMPERATURE_MODEL_PARAMETERS['sapm']['close_mount_glass_glass']

# get weather and load paramameters from saved txt files
```

```
.
.
.

# calculate temperature per solar panel for each tilt
tcell30 = pvlib.temperature.sapm_cell(global_ir_30, temp, wind,
 ↪  **temp_params)
tcell35 = pvlib.temperature.sapm_cell(global_ir_35, temp, wind,
 ↪  **temp_params)
tcell40 = pvlib.temperature.sapm_cell(global_ir_40, temp, wind,
 ↪  **temp_params)


def power_out_wind(type_turbine, N_wind):
    windpower = windmod.power_calc_wind(wind, type_turbine)
    tot_windpower = windpower * N_wind
    tot_windpower = np.around(tot_windpower.astype(np.float), 3)
    data = {'power_wind': tot_windpower.tolist()}
    return json.dumps(data)


def power_out_solar(N_panels, tilt_panel, type_pvpanel):
    if tilt_panel == 30:
        solpower = pvmod.power_calc_solar(global_ir_30, tcell30,
         ↪  type_pvpanel)
    elif tilt_panel == 35:
        solpower = pvmod.power_calc_solar(global_ir_35, tcell35,
         ↪  type_pvpanel)
    elif tilt_panel == 40:
        solpower = pvmod.power_calc_solar(global_ir_40, tcell40,
         ↪  type_pvpanel)
    else:
        raise ValueError('tilt not defined')
    tot_solpower = solpower * N_panels
    tot_solpower = np.around(tot_solpower.astype(np.float), 3)  # rounding
     ↪  for reducing the message size
    data = {'power_solar': tot_solpower.tolist()}
    return json.dumps(data)


def power_out_load(N_load, type_load):
    if type_load == ”saving”:
        load = load_saving
    elif type_load == ”average”:
        load = load_average
    else:
        raise ValueError('load_type not defined')
    tot_load = load * N_load
    tot_load = np.around(tot_load.astype(np.float), 3)
    data = {'power_load': tot_load.tolist()}
    return json.dumps(data)
```