TABLE OF CONTENTS

Introduction

This project aims to answer the question *"how many hot showers at 32* ℃ *can be taken per day over a year?"* using an approach involving tools of co-simulation. Independent models are developed for each component relevant to the system such as the source of power to heat the water, battery to store the energy, heating mechanism, water storage tank, and the shower itself. By identifying the relationships that link each of these models to one another in the form of inputs, outputs, and relevant parameters, the overall system is devised to follow a systematic and logical approach to solve this problem. It is important to also pay particular attention to the energy management strategy implemented for the battery, the water flow and temperature control for the tank, as well as the shower control. The individual models were built using the *OpenModelica* software and exported as functional mock-up units (FMUs) where the integration and co-simulation were performed using Python.

System components and problem constraints

The system that allows a shower to take place consists of several elements:

1. PV panels in an area of 6 m^2
2. Battery
3. Heat pump
4. Water tank with a volume of 2 m^3
5. Shower unit

The following constraints set the limitations for the elements of the system:

1. A shower can be taken from 6 to 8 am and from 6 to 9 pm
2. A shower lasts 10 minutes and uses 50 litres of water
3. The filling rate of the water tank is between 10% and 90%
4. Maximum temperature of water storage is 60 ℃
5. Inlet water temperature is 20 ℃
6. The shower temperature is 32 ℃
7. Two different locations are considered – Sevilla and Dunkerque

Some additional constraints to be followed while modelling are as follows:

1. At least one model contains a time-dependent equation
2. At least one model contains a differential equation
3. Solar irradiance is described by a time-dependent function
4. Sensors must be included in the system
5. Each model must be exported as an FMU

The model described above is developed and represented in the diagram shown in Figure 1. It includes all the elements of the system with inputs and outputs as well as the units and equations for some of the parts. The diagram begins with the PV panel block, where the input is *Solar irradiation* in *W/m²*, and the output is *Power* in *W* with the area of the panels and efficiency as the parameters. The next block is the *Battery*, where the input is *Power* in *W*, and the output is also *Power* in *W*. Battery efficiency, minimum and maximum state of charge, maximum power and capacity of the battery are parameters of the block. The power from the battery goes to the heat pump with the COP as its parameter, the output of which is *Heat* in *W*. The heat goes from the heat pump to the tank together with the cold-water input characterized by the *Temperature of water* in °C and *Flow rate* in *kg/s*. The parameters of the tank model are maximum and minimum volume, specific heat capacity of water, mass, and desired temperature of water. The outputs of the tank are the *Temperature of water* in °C and *Flow rate* in *kg/s*. The water is an input to the shower block. There are two controllers involved in the process – one controls the showers (shower control) and the other controls the tank and the battery (tank control). The control of the system consists of sensors and control commands. The output of the overall system is the number of showers taken.

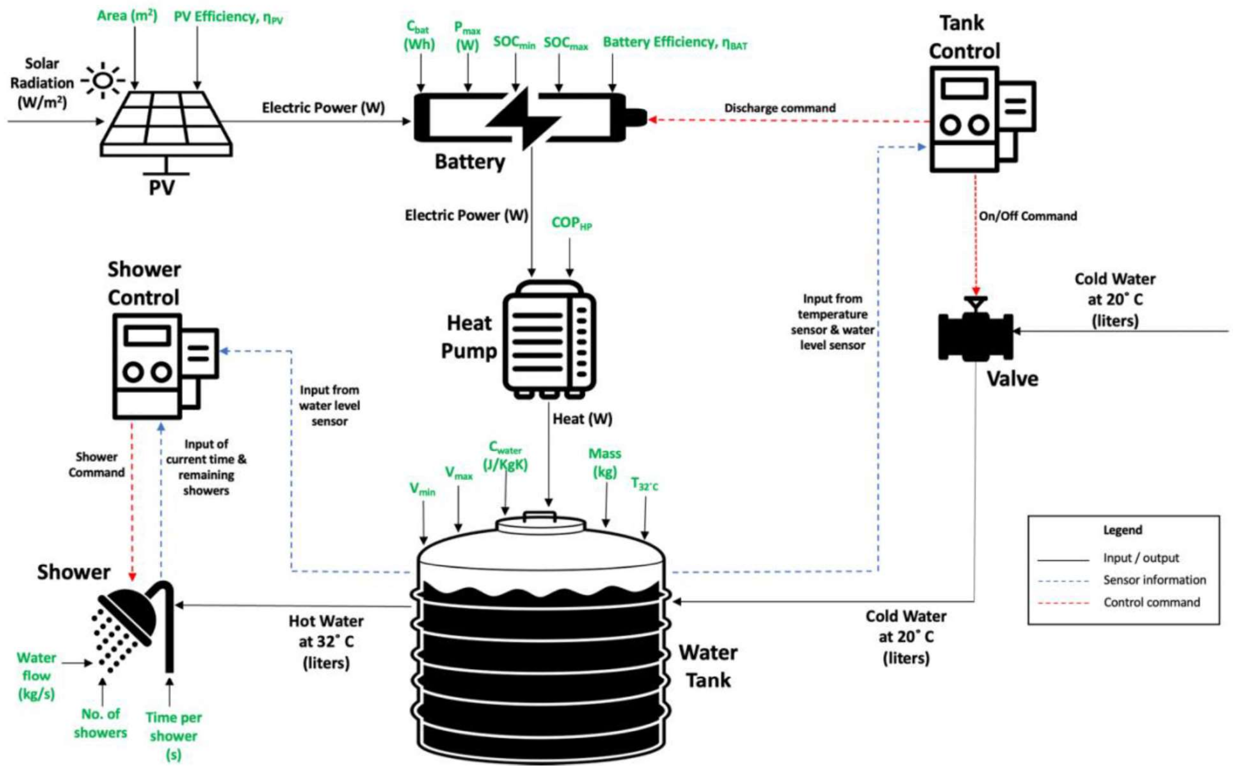


Figure 1: Process diagram of the overall system

PV model

The energy generation system consists of PV panels of 6 m^2 total area. The panels generate power from solar irradiation. For both locations – Sevilla and Dunkerque – the solar irradiation data is obtained from Global Solar Atlas[1]. The dataset consists of the average daily irradiation per hour for each month of the year. Therefore, the average daily power produced by the PV panels can be determined. An example of the data is presented in **Error! Reference source not found.**, showing the average daily solar irradiation for 12 months in Sevilla. The same data for Dunkerque can be found in Appendix A.

Table 1: Average daily solar irradiation for 12 months in Sevilla

Solar irradiance (W/m^2) in Sevilla												
Hour	**Jan**	**Feb**	**Mar**	**Apr**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Oct**	**Nov**	**Dec**
0	-	-	-	-	-	-	-	-	-	-	-	-
1	-	-	-	-	-	-	-	-	-	-	-	-
2	-	-	-	-	-	-	-	-	-	-	-	-
3	-	-	-	-	-	-	-	-	-	-	-	-
4	-	-	-	-	-	-	-	-	-	-	-	-
5	-	-	-	-	39	70	32	-	-	-	-	-
6	-	-	11	141	293	345	318	203	66	8	-	-
7	2	42	232	384	428	476	485	432	357	242	90	7
8	261	342	424	480	518	575	593	550	467	404	379	278
9	431	471	510	531	572	645	685	646	550	476	465	443
10	504	542	575	569	611	689	745	714	612	543	525	502
11	547	590	623	597	637	713	778	749	647	581	561	538
12	557	607	626	590	636	731	791	760	659	580	558	536
13	537	577	590	565	613	719	789	754	639	548	522	518
14	492	536	552	526	575	701	767	726	602	501	477	479
15	435	488	505	495	535	650	716	665	549	439	412	407
16	244	387	429	437	469	579	637	578	465	284	130	122
17	1	71	241	337	383	474	525	453	240	15	-	-
18	-	-	7	74	154	285	318	127	1	-	-	-
19	-	-	-	-	-	15	18	-	-	-	-	-
20	-	-	-	-	-	-	-	-	-	-	-	-
21	-	-	-	-	-	-	-	-	-	-	-	-
22	-	-	-	-	-	-	-	-	-	-	-	-
23	-	-	-	-	-	-	-	-	-	-	-	-
Total	**4,011**	**4,653**	**5,325**	**5,726**	**6,463**	**7,667**	**8,197**	**7,357**	**5,854**	**4,621**	**4,119**	**3,830**

[1] https://globalsolaratlas.info/map

As can be compared, the solar irradiation in Sevilla is higher than in Dunkerque, therefore the solar yield is much higher in Sevilla, and more showers could be taken based on this information. The highest yield is seen in summer as the irradiance in July is more than twice that of January.

The model of the PV panels shown on the left in Figure 2 is a block with solar irradiation as an input and power generated as an output. Area and efficiency are parameters of the model. The block on the right shows the FMU generated and exported on the Python graphical user interface (GUI) simply showing the input and output.

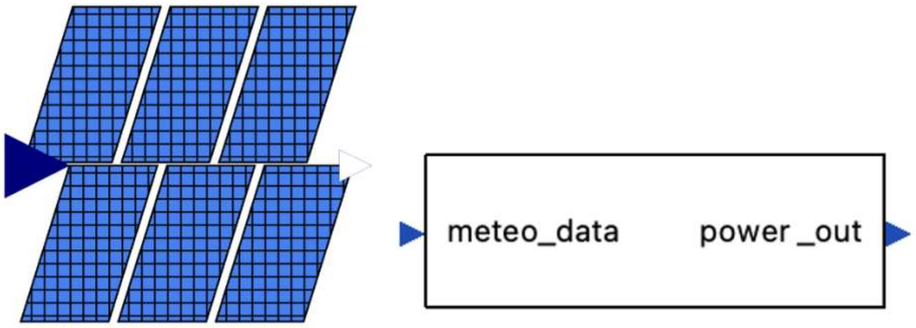


Figure 2: The PV model from OpenModelica (left) and exported FMU from Python GUI (right)

The efficiency of the PV panels is set at 20% while the total area is known to be 6 m^2. An efficiency of 20% was chosen based on the average efficiency of most commercially used PV panels [1]. To simulate the average daily power production by PV panels, the following equation is used:

$$PV\ power\ production = efficiency * solar\ irradiation * area$$

The input data is given as solar irradiance per hour, but *OpenModelica* functions at per second intervals. As a result, the data was interpolated to obtain the solar irradiance per second. The interpolation was performed in *OpenModelica* and used as an input to calculate the power generated. The following graph was produced, showing the average power generated per day for each month of the year from January to December, for a total of 12 days. For a full year, the data was extended for every day of each month and used as an input for the FMU.

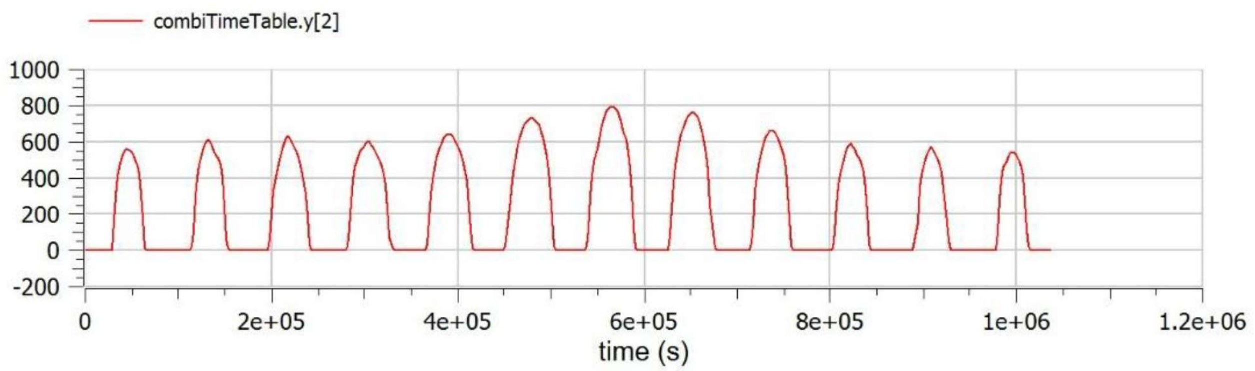
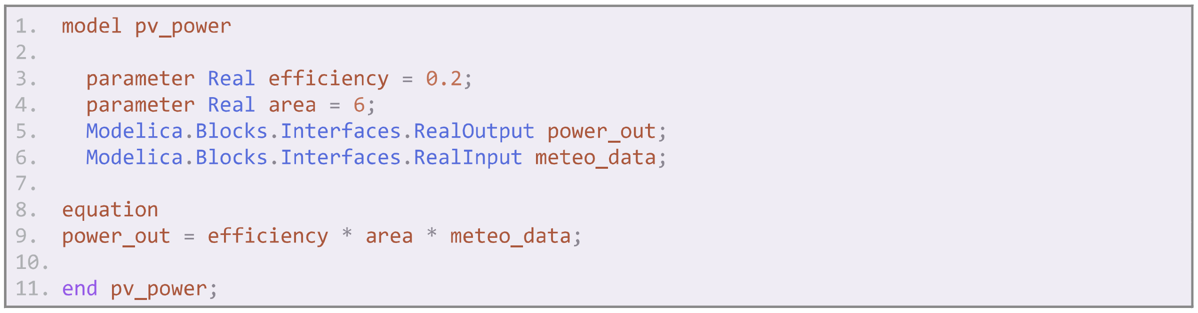


Figure 3: Interpolated PV power data for 12 days



```
model pv_power

  parameter Real efficiency = 0.2;
  parameter Real area = 6;
  Modelica.Blocks.Interfaces.RealOutput power_out;
  Modelica.Blocks.Interfaces.RealInput meteo_data;

equation
power_out = efficiency * area * meteo_data;

end pv_power;
```

Figure 4: OpenModelica code for the PV model

Battery model

The exported FMU of the battery model on the right indicates the presence of two inputs – the power inflow (which is the output from the PV model) and the discharge command. The latter is a binary input linked to the tank control which determines when the battery releases energy.

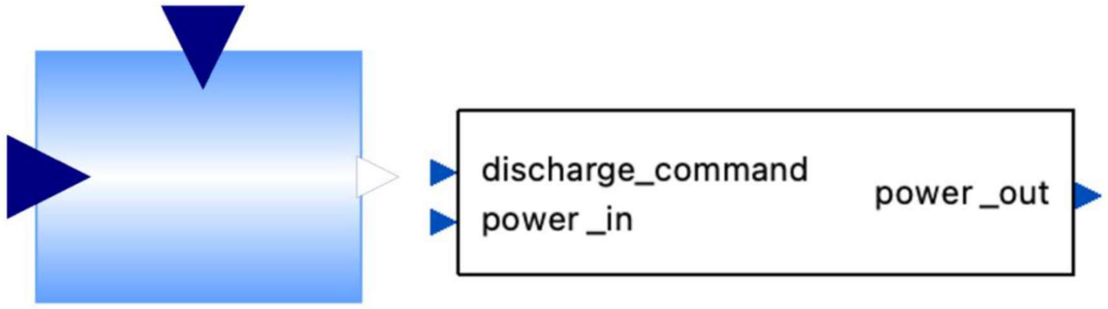


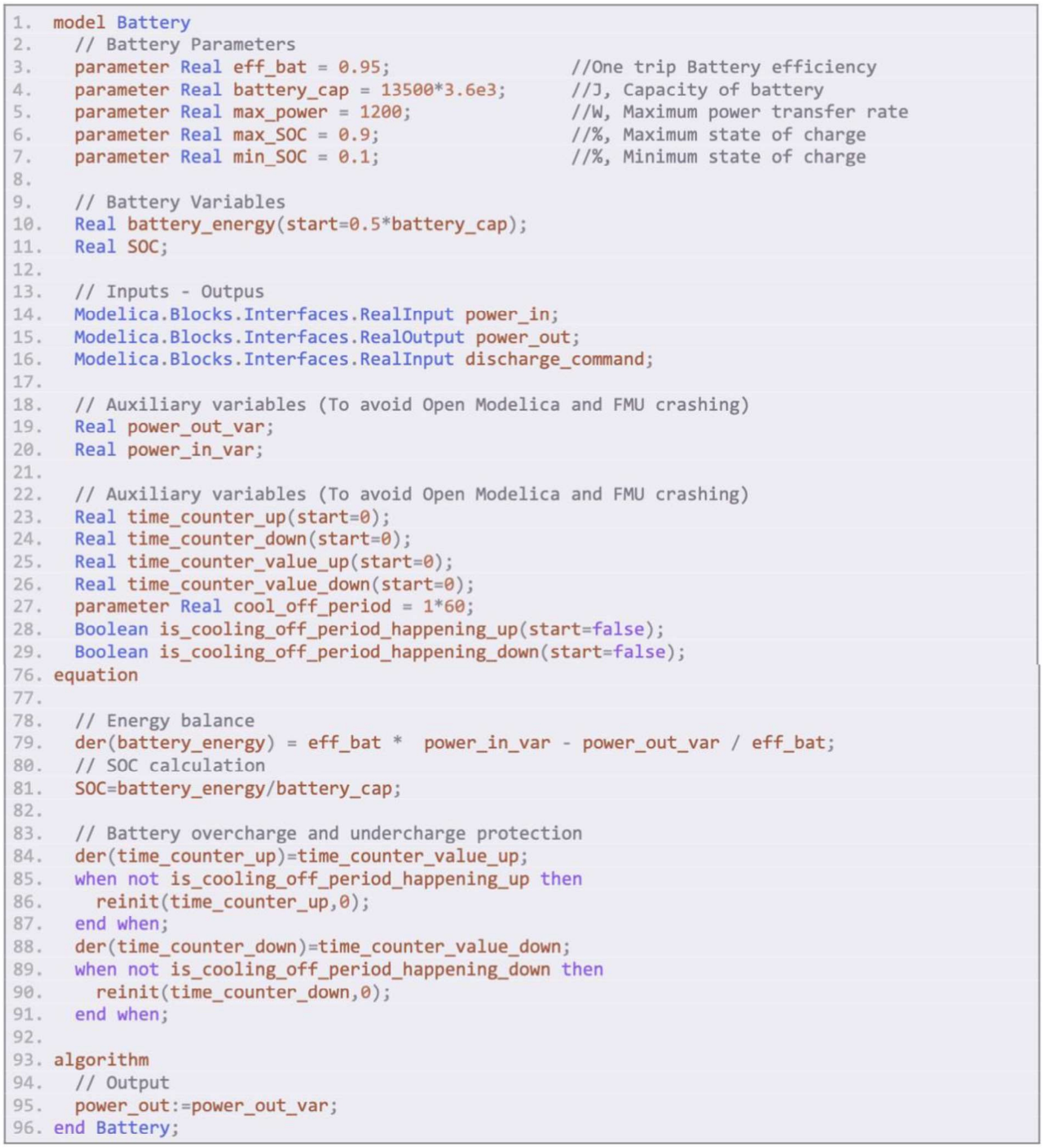
Figure 5: The battery model from OpenModelica (left) and exported FMU from Python GUI (right)

The battery model was included to store the power that was generated from the PV model. This enabled the system to be able to function during periods when solar irradiation was unavailable. In the model, several important parameters were considered: round-trip efficiency, battery capacity, maximum power, and the maximum and minimum state of charge (SOC).

The round-trip efficiency represents the amount of energy that a battery can deliver to a load compared to the amount of energy injected during its charging cycle. Thus, it refers to the ratio of the energy required to charge a battery compared to the available energy during its discharge. In our model, an efficiency of 95% was used as this is the theoretical round-trip efficiency of lithium-ion batteries [2]. The maximum and minimum SOC were set at 90% and 10% of the battery capacity respectively. In general, the input of the battery model was the PV power obtained, and the output power of the battery was used in the heat pump.

The energy balance for a battery with round trip efficiency $\eta_{in/out}$ and ignoring the calorific effects can be written as follows:

$$\frac{d}{dt}E_{tot} = \dot{Q} + \dot{W}_{elec} = \dot{W}_{elec,in} \cdot \eta_{in/out} - \frac{\dot{W}_{elec,out}}{\eta_{in/out}}$$



```
model Battery
  // Battery Parameters
  parameter Real eff_bat = 0.95;                //One trip Battery efficiency
  parameter Real battery_cap = 13500*3.6e3;     //J, Capacity of battery
  parameter Real max_power = 1200;              //W, Maximum power transfer rate
  parameter Real max_SOC = 0.9;                 //%, Maximum state of charge
  parameter Real min_SOC = 0.1;                 //%, Minimum state of charge

  // Battery Variables
  Real battery_energy(start=0.5*battery_cap);
  Real SOC;

  // Inputs - Outpus
  Modelica.Blocks.Interfaces.RealInput power_in;
  Modelica.Blocks.Interfaces.RealOutput power_out;
  Modelica.Blocks.Interfaces.RealInput discharge_command;

  // Auxiliary variables (To avoid Open Modelica and FMU crashing)
  Real power_out_var;
  Real power_in_var;

  // Auxiliary variables (To avoid Open Modelica and FMU crashing)
  Real time_counter_up(start=0);
  Real time_counter_down(start=0);
  Real time_counter_value_up(start=0);
  Real time_counter_value_down(start=0);
  parameter Real cool_off_period = 1*60;
  Boolean is_cooling_off_period_happening_up(start=false);
  Boolean is_cooling_off_period_happening_down(start=false);
equation

  // Energy balance
  der(battery_energy) = eff_bat *  power_in_var - power_out_var / eff_bat;
  // SOC calculation
  SOC=battery_energy/battery_cap;

  // Battery overcharge and undercharge protection
  der(time_counter_up)=time_counter_value_up;
  when not is_cooling_off_period_happening_up then
    reinit(time_counter_up,0);
  end when;
  der(time_counter_down)=time_counter_value_down;
  when not is_cooling_off_period_happening_down then
    reinit(time_counter_down,0);
  end when;

algorithm
  // Output
  power_out:=power_out_var;
end Battery;
```

Figure 6: OpenModelica code for the battery model

Heat pump model

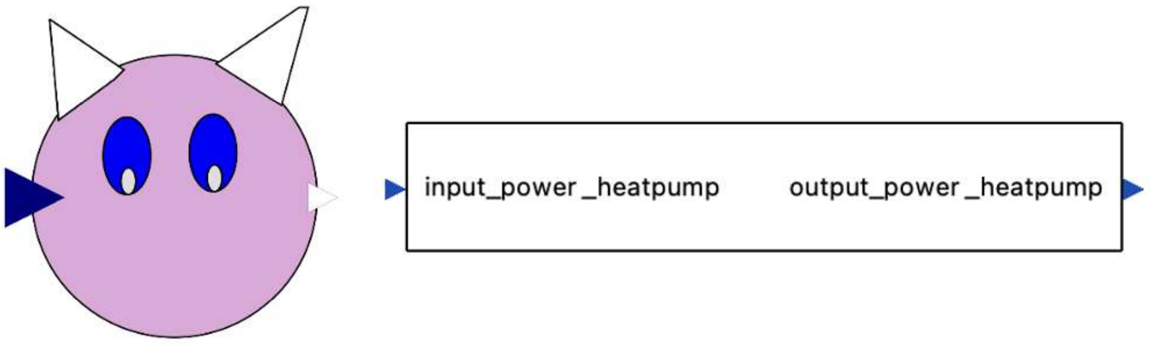


Figure 7: Heat pump model from OpenModelica (left) and exported FMU from Python GUI (right)

A heat pump is an electrically driven device that extracts heat from a low-temperature location (a source) and transports it to a higher-temperature one (a sink). To supply hot water using a heat pump, the environment is used as the source and the water tank is used as the heat sink.

The efficiency of a heat pump is described with the term coefficient of performance or COP. It is the ratio between the amount of heat delivered and the drive power required. For modern heat pumps, the value of COP can range from 3.0 to beyond 4.3 [3].

In this case, a 500 W heat pump with a COP of 4 is considered. This means that the heating provided by the heat pump will be four times the electrical energy that will be consumed.

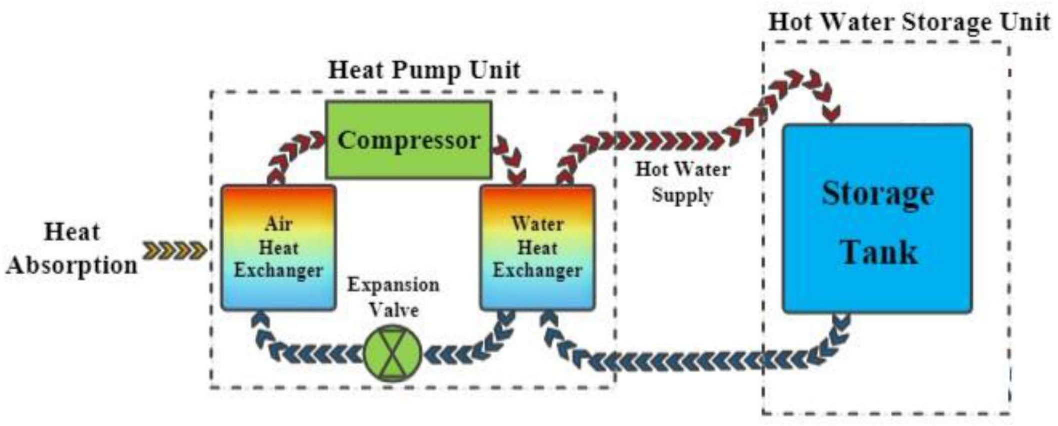
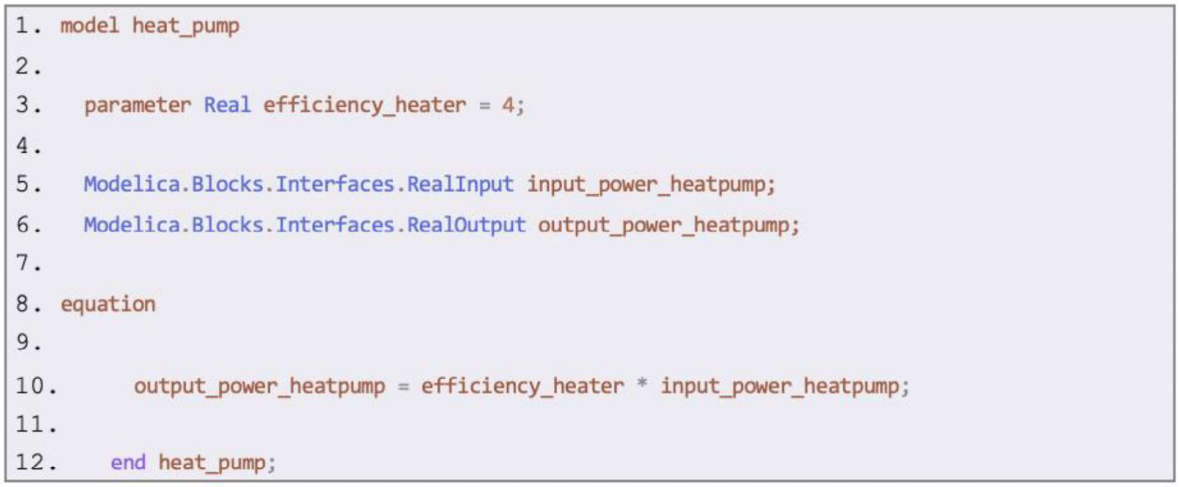


Figure 8: Schematic of a heat pump water heater [4]

The equation for the heat pump model is simply the product of the power input and COP:

$$Heat\ output = Power\ input * COP$$



```
model heat_pump

  parameter Real efficiency_heater = 4;

  Modelica.Blocks.Interfaces.RealInput input_power_heatpump;
  Modelica.Blocks.Interfaces.RealOutput output_power_heatpump;

equation

     output_power_heatpump = efficiency_heater * input_power_heatpump;

   end heat_pump;
```

Figure 9: OpenModelica code for the heat pump model

Tank model

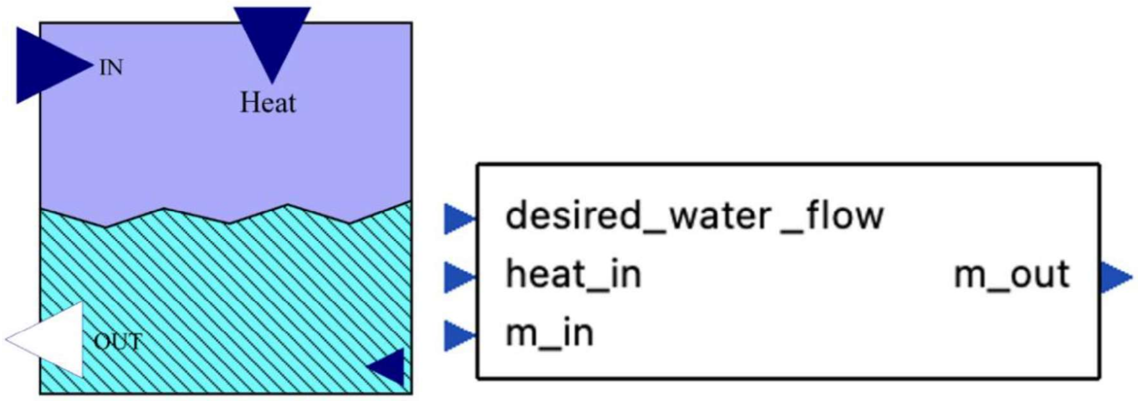


Figure 10: Tank model from OpenModelica (left) and exported FMU from Python GUI (right)

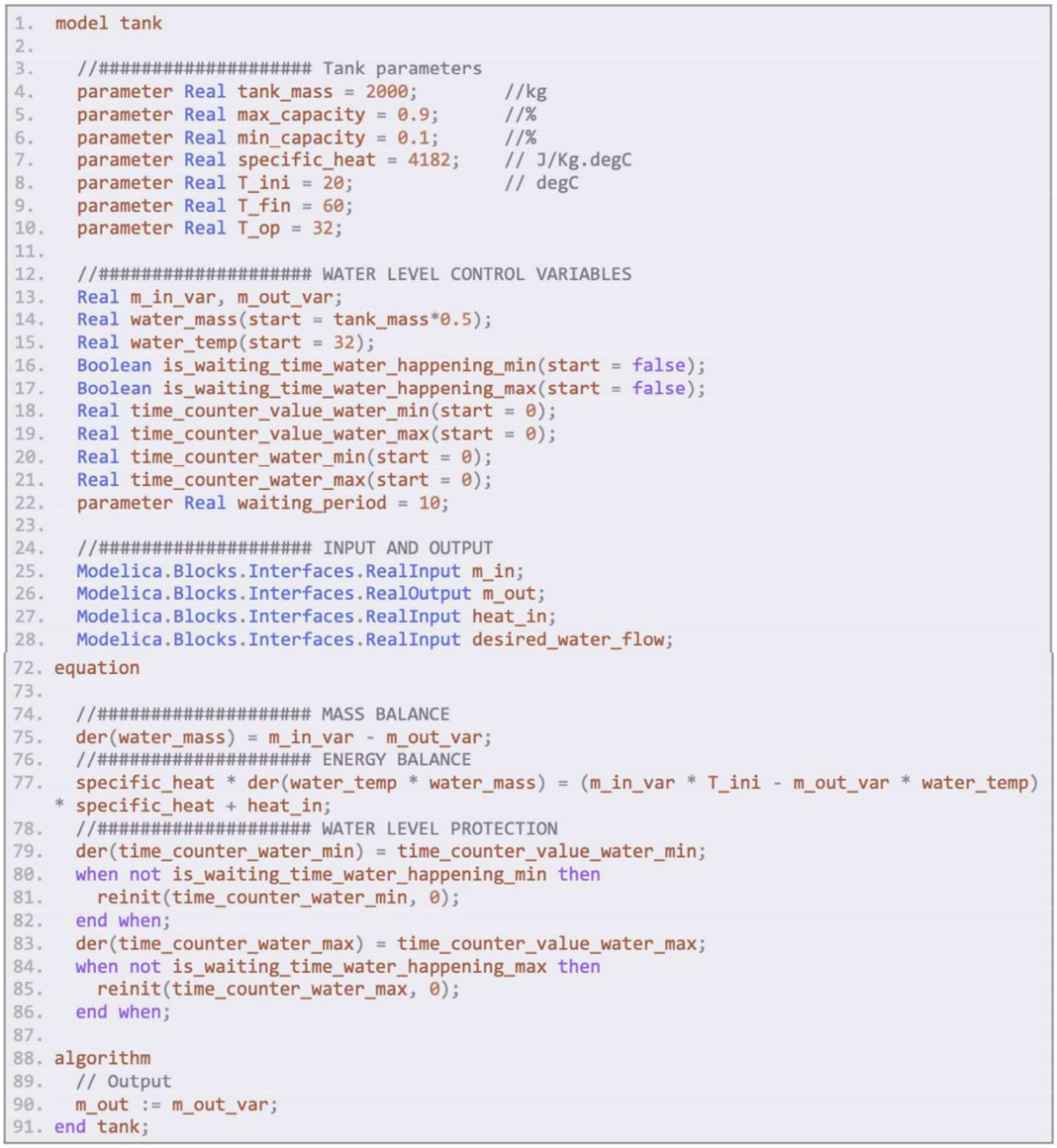
The heat generated by the heat pump heats the water in the tank. The total volume of the tank is 2 m^3, but the maximum and minimum usable capacity are set to 90% and 10% of its total volume respectively. In addition, the temperature limits are fixed between 20°C (inlet and initial temperature of the water) and 32°C (the temperature that is required for usage). The maximum temperature is set at 32°C to avoid the use of a mixer.

Two controllers were setup to ensure the constraints for the water level in the tank and the temperature are maintained. To maintain the level of water in the tank, a sensor takes the reading of the water level at each time and compares this to the reference. Based on this comparison, a valve controls whether or not water enters the tank. Similarly, the temperature controller maintains the temperature at 32°C. Based on the reading from the temperature sensor, this controller determines whether or not the battery should provide power to the heat pump to ensure that temperature does not fluctuate above or below 32°C.

The tank can be modeled with the help of a mass and an energy balance.

$$\frac{d}{dt}m = \dot{m}_{in} - \dot{m}_{out}$$

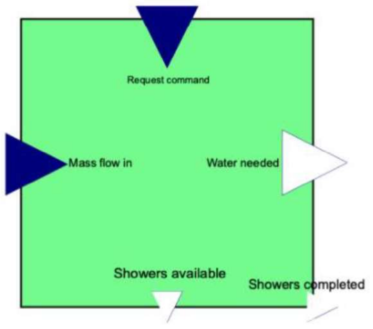
$$\frac{d}{dt}E_{tot} = c \cdot \frac{d}{dt}(T_{water} * m) = (m_{in} \cdot T_{20} - m_{out} \cdot T_{water}) \cdot c + \dot{Q}$$



```
model tank

  //#################### Tank parameters
  parameter Real tank_mass = 2000;         //kg
  parameter Real max_capacity = 0.9;       //%
  parameter Real min_capacity = 0.1;       //%
  parameter Real specific_heat = 4182;     // J/Kg.degC
  parameter Real T_ini = 20;               // degC
  parameter Real T_fin = 60;
  parameter Real T_op = 32;

  //#################### WATER LEVEL CONTROL VARIABLES
  Real m_in_var, m_out_var;
  Real water_mass(start = tank_mass*0.5);
  Real water_temp(start = 32);
  Boolean is_waiting_time_water_happening_min(start = false);
  Boolean is_waiting_time_water_happening_max(start = false);
  Real time_counter_value_water_min(start = 0);
  Real time_counter_value_water_max(start = 0);
  Real time_counter_water_min(start = 0);
  Real time_counter_water_max(start = 0);
  parameter Real waiting_period = 10;

  //#################### INPUT AND OUTPUT
  Modelica.Blocks.Interfaces.RealInput m_in;
  Modelica.Blocks.Interfaces.RealOutput m_out;
  Modelica.Blocks.Interfaces.RealInput heat_in;
  Modelica.Blocks.Interfaces.RealInput desired_water_flow;
equation

  //#################### MASS BALANCE
  der(water_mass) = m_in_var - m_out_var;
  //#################### ENERGY BALANCE
  specific_heat * der(water_temp * water_mass) = (m_in_var * T_ini - m_out_var * water_temp)
* specific_heat + heat_in;
  //#################### WATER LEVEL PROTECTION
  der(time_counter_water_min) = time_counter_value_water_min;
  when not is_waiting_time_water_happening_min then
    reinit(time_counter_water_min, 0);
  end when;
  der(time_counter_water_max) = time_counter_value_water_max;
  when not is_waiting_time_water_happening_max then
    reinit(time_counter_water_max, 0);
  end when;

algorithm
  // Output
  m_out := m_out_var;
end tank;
```

Figure 11: OpenModelica code for the tank model

Shower model



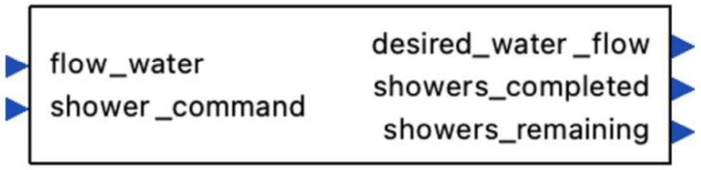
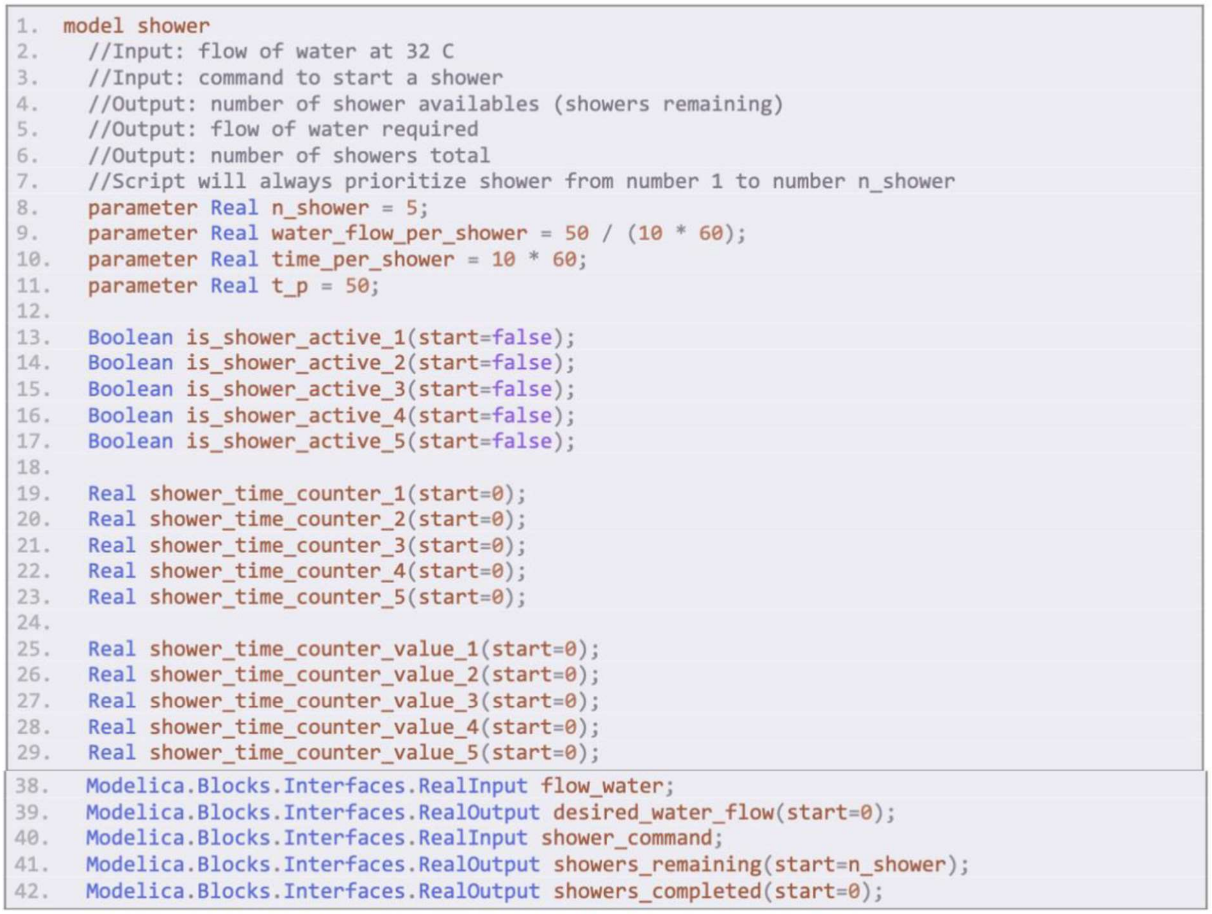


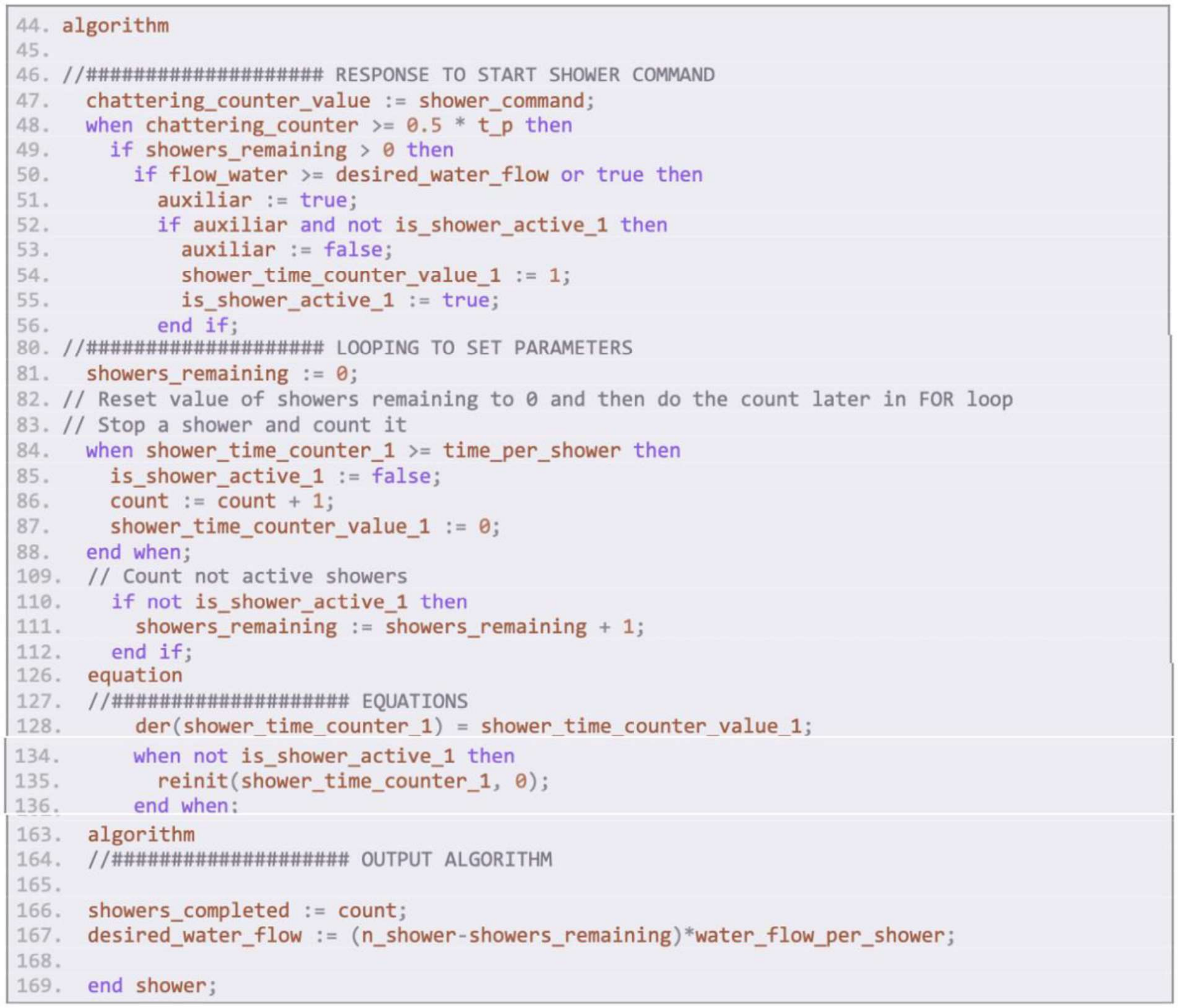
Figure 12: Shower model from OpenModelica (left) and exported FMU from Python GUI (right)

This model consists of 5 shower units that are activated during two periods: 6 to 8 am and 6 to 9 pm. The shower directly gets water from the tank at 32°C and a counter embedded tracks the number of completed showers. The shower controller checks the time of day to ensure a shower takes place within the predefined times and that the amount of water available is adequate for a shower to occur.



```
model shower
  //Input: flow of water at 32 C
  //Input: command to start a shower
  //Output: number of shower availables (showers remaining)
  //Output: flow of water required
  //Output: number of showers total
  //Script will always prioritize shower from number 1 to number n_shower
  parameter Real n_shower = 5;
  parameter Real water_flow_per_shower = 50 / (10 * 60);
  parameter Real time_per_shower = 10 * 60;
  parameter Real t_p = 50;

  Boolean is_shower_active_1(start=false);
  Boolean is_shower_active_2(start=false);
  Boolean is_shower_active_3(start=false);
  Boolean is_shower_active_4(start=false);
  Boolean is_shower_active_5(start=false);

  Real shower_time_counter_1(start=0);
  Real shower_time_counter_2(start=0);
  Real shower_time_counter_3(start=0);
  Real shower_time_counter_4(start=0);
  Real shower_time_counter_5(start=0);

  Real shower_time_counter_value_1(start=0);
  Real shower_time_counter_value_2(start=0);
  Real shower_time_counter_value_3(start=0);
  Real shower_time_counter_value_4(start=0);
  Real shower_time_counter_value_5(start=0);
  Modelica.Blocks.Interfaces.RealInput flow_water;
  Modelica.Blocks.Interfaces.RealOutput desired_water_flow(start=0);
  Modelica.Blocks.Interfaces.RealInput shower_command;
  Modelica.Blocks.Interfaces.RealOutput showers_remaining(start=n_shower);
  Modelica.Blocks.Interfaces.RealOutput showers_completed(start=0);
```

Figure 13: OpenModelica code for the shower model (1 of 2)



```
algorithm

//#################### RESPONSE TO START SHOWER COMMAND
  chattering_counter_value := shower_command;
  when chattering_counter >= 0.5 * t_p then
    if showers_remaining > 0 then
      if flow_water >= desired_water_flow or true then
        auxiliar := true;
        if auxiliar and not is_shower_active_1 then
          auxiliar := false;
          shower_time_counter_value_1 := 1;
          is_shower_active_1 := true;
        end if;
//#################### LOOPING TO SET PARAMETERS
  showers_remaining := 0;
// Reset value of showers remaining to 0 and then do the count later in FOR loop
// Stop a shower and count it
  when shower_time_counter_1 >= time_per_shower then
    is_shower_active_1 := false;
    count := count + 1;
    shower_time_counter_value_1 := 0;
  end when;
  // Count not active showers
    if not is_shower_active_1 then
      showers_remaining := showers_remaining + 1;
    end if;
  equation
  //#################### EQUATIONS
      der(shower_time_counter_1) = shower_time_counter_value_1;
      when not is_shower_active_1 then
        reinit(shower_time_counter_1, 0);
      end when;
  algorithm
  //#################### OUTPUT ALGORITHM

  showers_completed := count;
  desired_water_flow := (n_shower-showers_remaining)*water_flow_per_shower;

  end shower;
```

Figure 14: OpenModelica code for the shower model (2 of 2)

User model

Figure 15: Representation of the shower users in the model

The final user is one of the most important elements in any design. For this model, a very simple assumption has been made – there is an endless queue of people that are waiting to take a shower in the hours of 6 to 8 am and 6 to 9 pm. Given this, it is worth noticing that there is not much interest in how to distribute the showers between the morning and the evening, as the end goal is to calculate how many hot showers can be taken daily along a year given a limited input of energy.

For this reason, it is justified to assume that the number of users is not the limiting factor in the model, and in this way, the shower model never "runs out" of people, and a shower is called immediately if there is availability of water, and at least one shower is available.

Results

Once each model was successfully built on *OpenModelica*, they were exported to Python as individual FMUs. These FMUs were tested for compatibility with Python and integrated together to form the desired system for co-simulation. The Python code can be found in Appendix B. The results below show the functionality of the system, starting from the PV model and ending with the shower model for Sevilla. The simulation has been performed over a period of 3 days or 259,200 seconds in the figures shown on the left to ensure visual clarity and comparability in the results while the cumulative results over a period of 365 days are shown on the right. The simulated results for Dunkerque are shown in Appendix C.

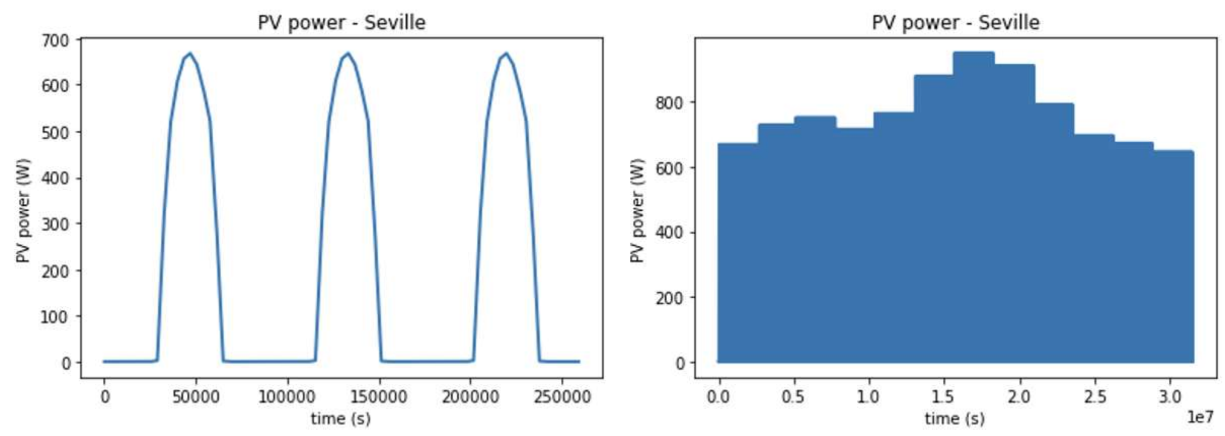


Figure 16: PV power for 3 days in January (left); PV power for 365 days (right)

As shown in the figure on the left, the maximum PV power in January is around 660 W which is seen around noon each day. The figure on the right shows that the PV power increases in the summer months as there is significantly more irradiance during this time.

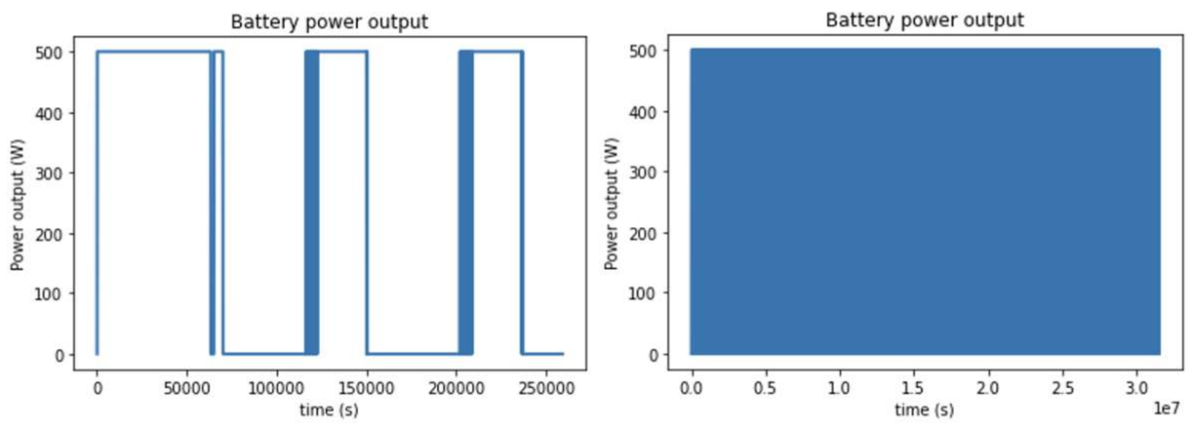


Figure 17: Battery power output for 3 days in January (left); battery power output for 365 days (right)

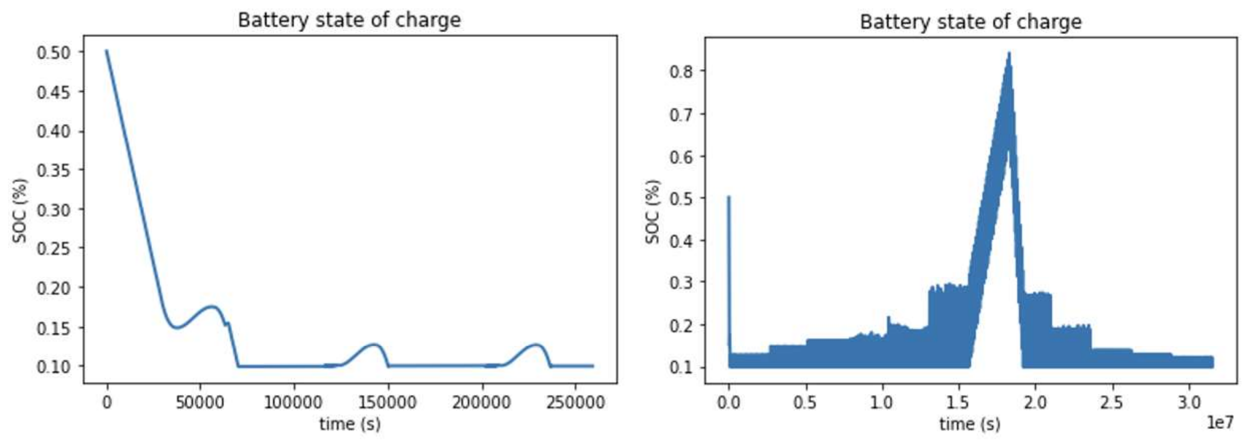


Figure 18: Battery SOC for 3 days in January (left); battery SOC for 365 days (right)

Looking at both Figure 17 and Figure 18 holistically, it can be seen from the former that the battery power output follows a step function as it is determined by the binary discharge command from the tank controller. The latter shows that the SOC of the battery rarely increases beyond 10-15% in January although it starts at 50% given the initial conditions defined in the model. The SOC is significantly higher in the summer as the PV power available is greater but is always maintained between 10 and 90% to adhere to the constraints.

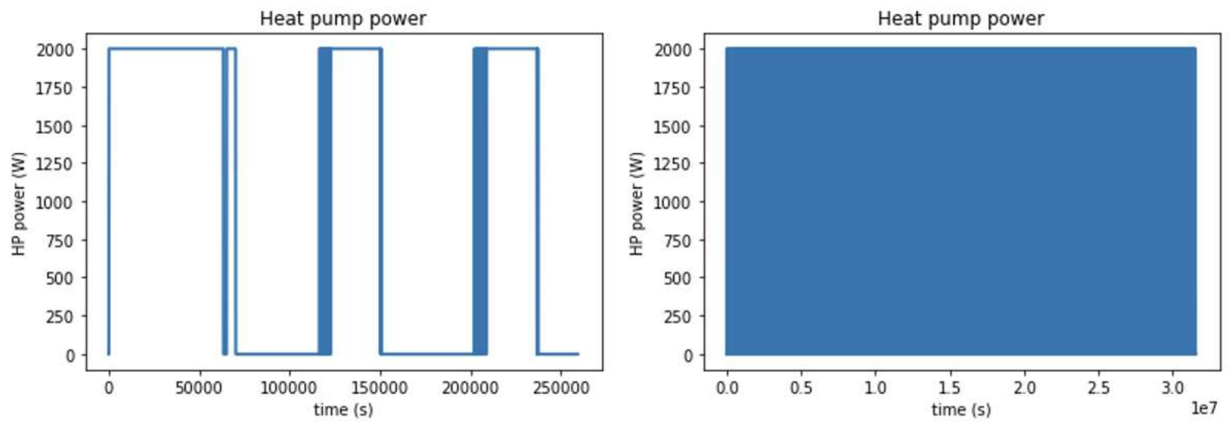


Figure 19: Heat pump power for 3 days in January (left); heat pump power for 365 days (right)

As shown in the figure above, the heat pump follows the exact same pattern as the battery power output as it is indirectly controlled by the discharge command to the battery. The only difference, however, is the value of the power which is a product of the battery power output and COP.

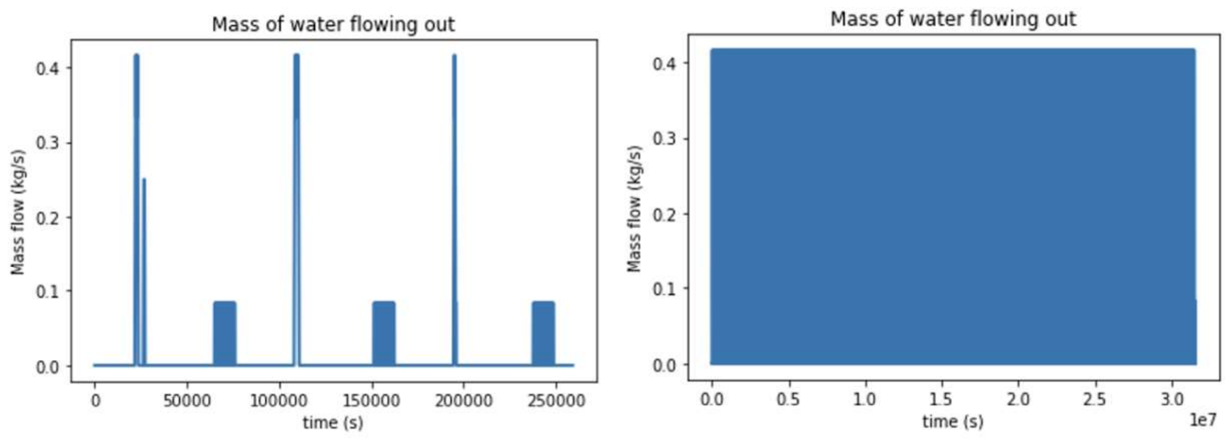


Figure 20: Water outflow from tank for 3 days in January (left); water outflow from tank for 365 days (right)

As shown in the figure above, the mass of water flowing out from the tank is determined by the showers in use. It has discrete values of 0.08, 0.25 and 0.42 kg/s, and the water outflow is consistent with the predefined shower times of 6 to 8 am and 6 to 9 pm.

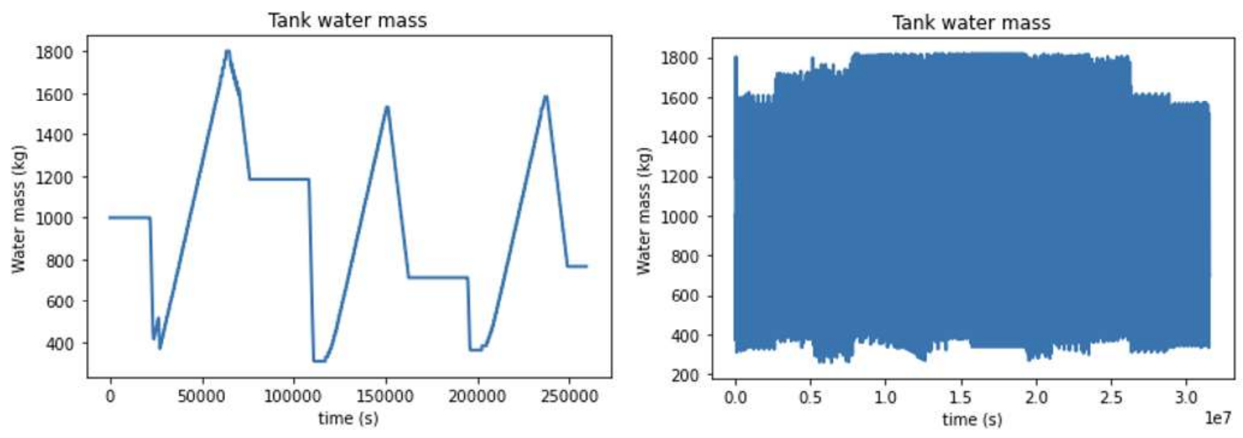


Figure 21: Tank water mass for 3 days in January (left); tank water mass for 365 days (right)

The main takeaway from Figure 21 is that the tank water mass always varies between 200 kg and 1,800 kg, demonstrating that the constraints for minimum and maximum capacity are adhered to. Moreover, the decrease in water mass always takes place during the predefined shower times of 6 to 8 am and 6 to 9 pm.

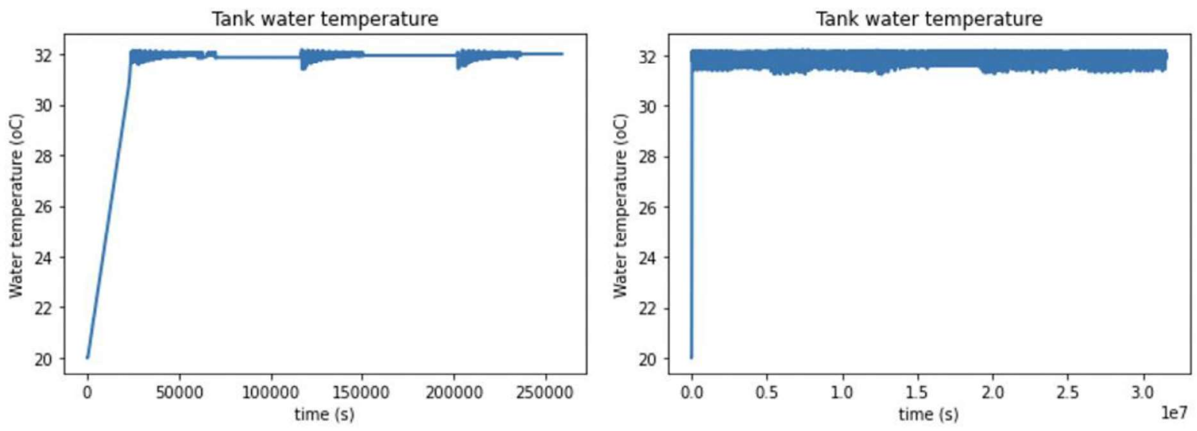


Figure 22: Tank water temperature for 3 days in January (left); tank water temperature for 365 days (right)

Figure 22 shows that the tank water temperature is always maintained at 32 ± 0.05 °C. The initial temperature is at 20 °C given the temperature of the incoming cold water.

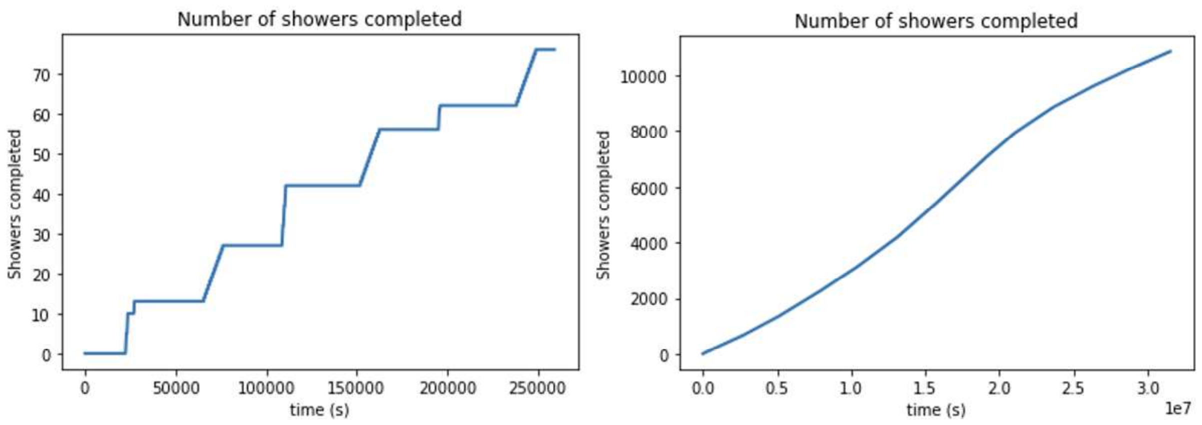


Figure 23: Showers completed for 3 days in January (left); showers completed for 365 days (right)

As shown in the figure above, the cumulative number of showers completed over a 3-day period is around 75 while the cumulative total over 365 days is around 11,000.

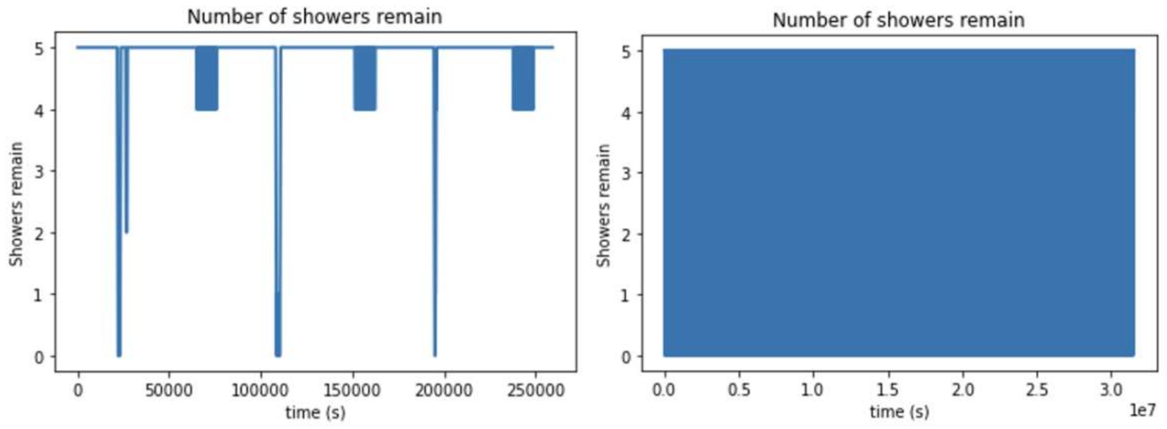


Figure 24: Showers remaining for 3 days in January (left); showers remaining for 365 days (right)

Given the presence of 5 shower units, it is necessary to identify the number of showers that remain unused at a given time. This is shown in Figure 24. It is important to note that 4 showers always remain unused in the time interval of 6 to 9 pm every day to ensure that sufficient energy has been accumulated to facilitate the showers the following morning from 6 to 8 am.

The objective – *"how many hot showers at 32 ℃ can be taken per day over a year?",* can be achieved by calculating the average number of showers per day for each of the 12 months across the entire year. These results for both Sevilla and Dunkerque are shown in Figure 25.

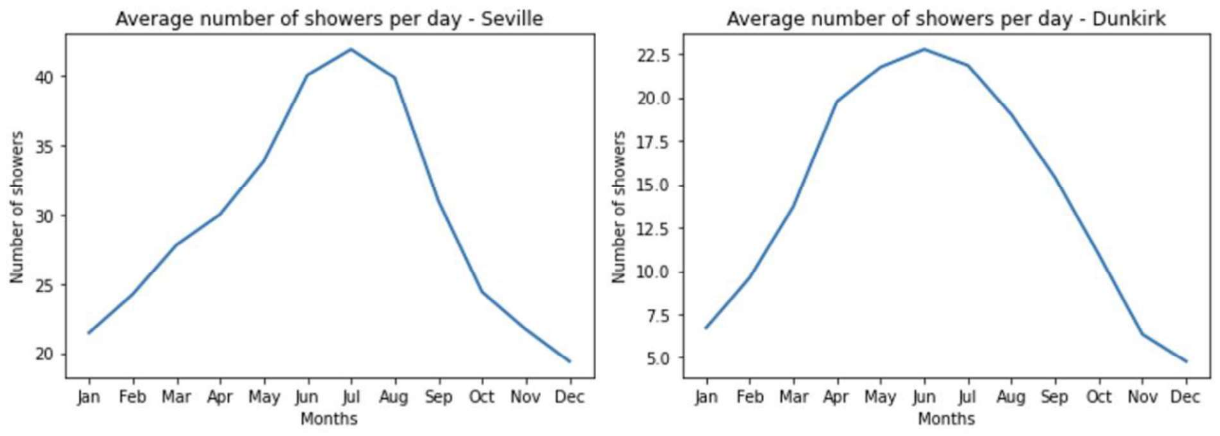


Figure 25: Average number of showers per day over a year for Sevilla and Dunkerque

The relative lack of solar irradiance in Dunkerque results in a much lower average number of showers per day compared to Sevilla. In the summer, the value for Dunkerque is almost half that of Sevilla while in the winter it is about one third.

Discussion

Looking at the project from a critical point of view, the overall model created has achieved the goal of answering the initial question of the number of showers that can be taken per day in a year. However, the model would benefit from some improvements that could be made in the future.

Firstly, the system was assumed to be perfectly insulated, therefore no heat was lost in the process particularly from the water tank. For more realistic results, the overall model should take that into account and incorporate the loss of the heat in the tank model.

The results from Dunkerque showed that the number of showers is significantly lower than at Sevilla. This is clearly because the solar irradiance in Dunkerque is about half that of Sevilla so the PV power output over the same area decreases by half reducing the overall energy available for the heating of water. It would be better suited in this case to consider an alternative source of energy, for example, wind power. There is evidence to suggest that this would be more beneficial as Dunkerque is considered one of the most promising sites for wind power in the French coast[2].

The control of the system could also be improved. First, the battery operation could be optimized considering its life cycle and optimal usage. That could be done by improving the control of the battery, ensuring that the SOC varies within certain limits. This could also increase the number of showers, as the SOC is consistently very low during the day, which reduces the energy available for evening showers. A forecasting and optimization algorithm could also be introduced to control the amount of power used. The forecasting tool would use the information of the previous days to optimize the number of showers to maintain the same maximum number every day.

[2] https://www.dunkerquepromotion.org/en/2016/10/dunkirk-will-be-the-most-profitable-wind-farm-site/

Some of the individual blocks modelled in this system can be reused directly for other applications. The PV block is the best example as it can be reused for any system involving PV panels simply with the adjustment of the parameters for a chosen location and type of panel. The heat pump model can also be used for other applications that require heat such as space heating. With several simple modifications, the heat pump model can also be changed to an air-conditioning model. It is also possible to use the battery model for any application that requires energy storage. Given the widespread use of lithium-ion batteries with renewable energy resources, a battery model such as the one used in this case can be effectively adapted to suit a wide range of other applications.

Bibliography

[1] C. f. S. Systems, "Photovoltaic Energy Factsheet," University of Michigan, 2021. [Online]. Available: https://css.umich.edu/factsheets/photovoltaic-energy-

[2] "Large Power," [Online]. Available: https://www.large.net/news/8bu43pr.html. [Accessed 24 May 2022].

[3] R. A. Aldrich, D. Owens and P. Mantha, "Residential Ground-Source Heat Pumps: In-Field System Performance and Energy Modeling Srikanth Puttagunta".

[4] A. Oshnoei, R. Khezri and S. M. Muyeen, "Model Predictive-Based Secondary Frequency Control Considering Heat Pump Water -7-Heaters," *Energies,* 2019.

[5] "Heating and Cooling with a Heat pump," *Energy Star Canada.*

APPENDIX A: Solar irradiance for Dunkerque

Solar irradiance (W/m^2) in Dunkerque												
Hour	**Jan**	**Feb**	**Mar**	**Apr**	**May**	**Jun**	**Jul**	**Aug**	**Sep**	**Oct**	**Nov**	**Dec**
0	-	-	-	-	-	-	-	-	-	-	-	-
1	-	-	-	-	-	-	-	-	-	-	-	-
2	-	-	-	-	-	-	-	-	-	-	-	-
3	-	-	-	-	-	-	-	-	-	-	-	-
4	-	-	-	-	26	57	25	-	-	-	-	-
5	-	-	-	20	141	153	136	46	-	-	-	-
6	-	-	20	181	220	218	202	183	129	1	-	-
7	-	19	153	263	277	270	257	249	225	136	-	-
8	43	167	231	317	322	316	300	292	276	212	116	37
9	150	217	261	348	347	346	326	307	296	241	167	134
10	186	241	286	374	359	355	338	328	313	260	185	166
11	208	263	317	393	372	376	364	346	324	284	196	174
12	204	272	339	387	375	382	374	348	316	260	185	166
13	184	236	295	378	372	382	378	340	298	235	164	146
14	145	208	267	359	372	381	380	332	273	213	128	108
15	68	167	240	329	349	367	357	320	247	167	37	12
16	-	46	176	273	299	309	311	281	189	45	-	-
17	-	-	40	172	222	254	247	201	56	-	-	-
18	-	-	-	24	115	162	157	51	-	-	-	-
19	-	-	-	-	-	38	26	-	-	-	-	-
20	-	-	-	-	-	-	-	-	-	-	-	-
21	-	-	-	-	-	-	-	-	-	-	-	-
22	-	-	-	-	-	-	-	-	-	-	-	-
23	-	-	-	-	-	-	-	-	-	-	-	-
Total	**1,188**	**1,836**	**2,625**	**3,818**	**4,168**	**4,366**	**4,178**	**3,624**	**2,942**	**2,054**	**1,178**	**943**

APPENDIX B: Python code for co-simulation

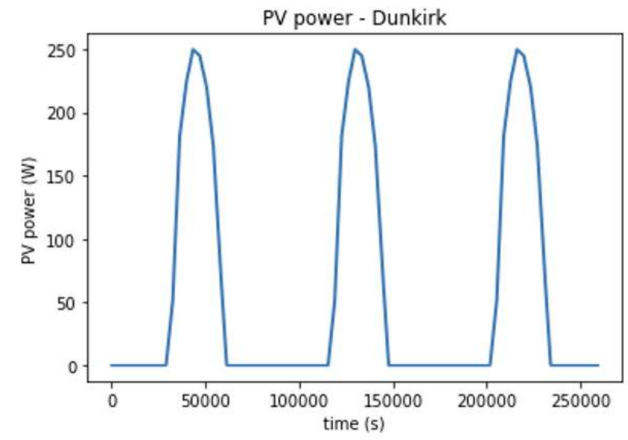
```python
#Import relevant libraries
from myFMUnew import myFMU  #local path to myFMU library may be changed to a module
import pandas as pd
import numpy as np
import math
from fmpy import *
import matplotlib.pyplot as plt

#Set FMU paths
path1 = "pv_power.fmu"
path2 = "Battery_v2.fmu"
path3 = "heat_pump.fmu"
path4 = "tank.fmu"
path5 = "shower_test.fmu"

#Get irradiance data from Excel using Pandas
seville = pd.read_excel('Solar(test).xlsx',sheet_name=0)
irradiance = seville.iloc[:,1]
time_list = seville.iloc[:,0]

#Define initial values and useful variables for simulation
dico = dict(
    meteo_data=0,
    power_in=0,
    discharge_command=0,
    SOC=0.6,
    max_power=500,
    eff_bat = 0.95,
    input_power_heatpump=0,
    heat_in=0,
    m_in=0,
    water_mass=1000,
    water_temp=20,
    desired_water_flow=0,
    flow_water=0,
    shower_command=0)

#Define time
day_size = 86400
stop_time = maxtime = day_size * 365
step_size = deltaT = 100
startTime = 0

#Define shower times
shower_start1 = startTime + 21600
shower_end1 = shower_start1 + 7200
shower_start2 = startTime + 64800
shower_end2 = shower_start2 + 10800

#Tank temperature variables
temp_min = 32
temp_margin = 0.05
water_mass_max = 1800
new_min_capacity = 500

#Define FMU instance
pv_power=myFMU(path1)
Battery=myFMU(path2)
heat_pump=myFMU(path3)
tank=myFMU(path4)
shower=myFMU(path5)
```

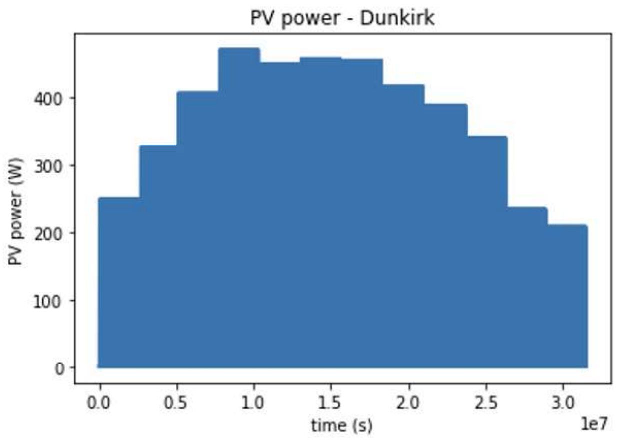
```python

#Set initial values
initial_variables = {'pv_power':['meteo_data'],
                     'Battery':['power_in','discharge_command','SOC','max_power','eff_bat'],
                     'heat_pump':['input_power_heatpump'],
                     'tank':['heat_in','m_in','water_mass','water_temp','desired_water_flow'],
                     'shower':['flow_water','shower_command']}

pv_power.init(startTime,[(var,dico[var]) for var in initial_variables['pv_power']])
Battery.init(startTime,[(var,dico[var]) for var in initial_variables['Battery']])
heat_pump.init(startTime,[(var,dico[var]) for var in initial_variables['heat_pump']])
tank.init(startTime,[(var,dico[var]) for var in initial_variables['tank']])
shower.init(startTime,[(var,dico[var]) for var in initial_variables['shower']])

#Create lists to store results for graphs
list_time=[]
list_pv=[]
list_battery=[]
list_soc=[]
list_hp=[]
list_tank=[]
list_watermass=[]
list_watertemp=[]
list_shower_completed=[]
list_shower_remain=[]
list_shower_test=[]

#Iterate through time to set and obtain FMU inputs, parameters and outputs
time = startTime
while time < stop_time:
    # Perform one step
    pv_power.doStep(time,step_size)
    Battery.doStep(time,step_size)
    heat_pump.doStep(time,step_size)
    tank.doStep(time,step_size)
    shower.doStep(time,step_size)
    pv_results=pv_power.get('power_out')
    battery_results=Battery.get('power_out')
    heatpump_results=heat_pump.get('output_power_heatpump')
    tank_results=tank.get('m_out')
    shower_results1=shower.get('showers_completed')
    shower_results2=shower.get('showers_remaining')
    shower_results3=shower.get('desired_water_flow')
    pv_power.set('meteo_data',np.interp(time, time_list, irradiance))
    Battery.set('power_in',pv_results)
    Battery.set('discharge_command',1)
    heat_pump.set('input_power_heatpump',battery_results)
    tank.set('heat_in',heatpump_results)
    shower.set('flow_water',tank_results)

    battery_results_soc=Battery.get('SOC')
    tank_results_watermass=tank.get('water_mass')
    tank_results_watertemp=tank.get('water_temp')
    tank.set('desired_water_flow',shower_results3)

    day = math.floor(time/day_size) * day_size
    if (time >= day + shower_start1 and time < day + shower_end1) or (time >= day + shower_start2 and time < day + shower_end2):
        if tank_results_watermass < new_min_capacity:
            shower.set('shower_command', 0)
        elif shower_results2 <= 4 and (time >= day + shower_start2 and time < day + shower_end2):
            shower.set('shower_command', 0)
        else:
            shower.set('shower_command', 1)
```

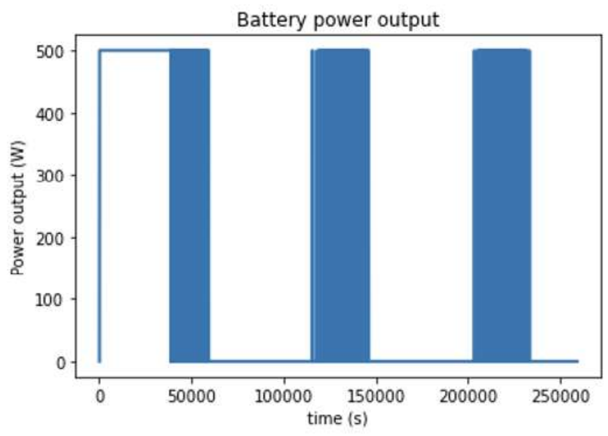
```
         else:
             shower.set('shower_command', 0)

         #Tank control to ensure temperature maintained at 32C
         if tank_results_watertemp < temp_min - temp_margin:
            tank.set('m_in',0)
         if tank_results_watertemp >= temp_min + temp_margin:
            tank.set('m_in',0.2)
         if tank_results_watermass >= water_mass_max:
            Battery.set('discharge_command',0)

         time+=step_size

         #Append FMU results to lists
         list_pv.append(pv_results)
         list_battery.append(battery_results)
         list_soc.append(battery_results_soc)
         list_hp.append(heatpump_results)
         list_tank.append(tank_results)
         list_watermass.append(tank_results_watermass)
         list_watertemp.append(tank_results_watertemp)
         list_shower_completed.append(shower_results1)
         list_shower_remain.append(shower_results2)
         list_time.append(time)

    #Function to determine average number of showers per day for a year
    def shower_counter():
        list_shower_count=[]
        list_shower_count.append((list_shower_completed[864*31]))
        list_shower_count.append((list_shower_completed[864*(31+28)]))
        list_shower_count.append((list_shower_completed[864*(31+28+31)]))
        list_shower_count.append((list_shower_completed[864*(31+28+31+30)]))
        list_shower_count.append((list_shower_completed[864*(31+28+31+30+31)]))
        list_shower_count.append((list_shower_completed[864*(31+28+31+30+31+30)]))
        list_shower_count.append((list_shower_completed[864*(31+28+31+30+31+30+31)]))
        list_shower_count.append((list_shower_completed[864*(31+28+31+30+31+30+31+31)]))
        list_shower_count.append((list_shower_completed[864*(31+28+31+30+31+30+31+31+30)]))
        list_shower_count.append((list_shower_completed[864*(31+28+31+30+31+30+31+31+30+31)]))
        list_shower_count.append((list_shower_completed[864*(31+28+31+30+31+30+31+31+30+31+30)]))
        list_shower_count.append((list_shower_completed[864*(31+28+31+30+31+30+31+31+30+31+30+30)]))
        list_shower_count[11]=(list_shower_count[11]-list_shower_count[10])/31
        list_shower_count[10]=(list_shower_count[10]-list_shower_count[9])/30
        list_shower_count[9]=(list_shower_count[9]-list_shower_count[8])/31
        list_shower_count[8]=(list_shower_count[8]-list_shower_count[7])/30
        list_shower_count[7]=(list_shower_count[7]-list_shower_count[6])/31
        list_shower_count[6]=(list_shower_count[6]-list_shower_count[5])/31
        list_shower_count[5]=(list_shower_count[5]-list_shower_count[4])/30
        list_shower_count[4]=(list_shower_count[4]-list_shower_count[3])/31
        list_shower_count[3]=(list_shower_count[3]-list_shower_count[2])/30
        list_shower_count[2]=(list_shower_count[2]-list_shower_count[1])/31
        list_shower_count[1]=(list_shower_count[1]-list_shower_count[0])/28
        list_shower_count[0]=(list_shower_count[0])/31

        return list_shower_count
    list_shower_count=shower_counter()
    list_months = ['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']

    #Create plots
    plt.plot(list_time,list_pv,linewidth=2.0)
    plt.title("PV power - Seville")
    plt.xlabel("time (s)")
    plt.ylabel("PV power (W)")
    plt.show()
```

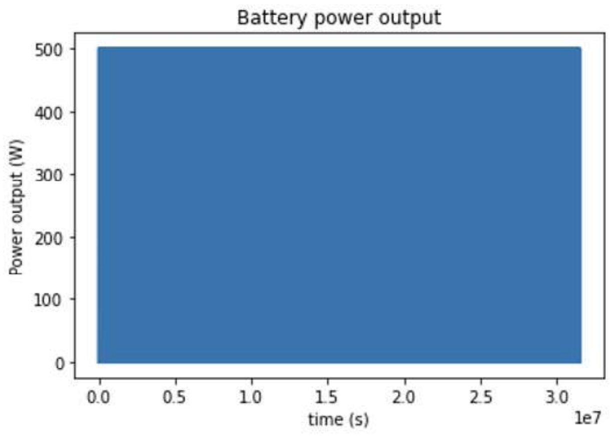
```python

    plt.plot(list_time,list_battery,linewidth=2.0)
    plt.title("Battery power output")
    plt.xlabel("time (s)")
    plt.ylabel("Power output (W)")
    plt.show()

    plt.plot(list_time,list_soc,linewidth=2.0)
    plt.title("Battery state of charge")
    plt.xlabel("time (s)")
    plt.ylabel("SOC (%)")
    plt.show()

    plt.plot(list_time,list_hp,linewidth=2.0)
    plt.title("Heat pump power")
    plt.xlabel("time (s)")
    plt.ylabel("HP power (W)")
    plt.show()

    plt.plot(list_time,list_tank,linewidth=2.0)
    plt.title("Mass of water flowing out")
    plt.xlabel("time (s)")
    plt.ylabel("Mass flow (kg/s)")
    plt.show()

    plt.plot(list_time,list_watermass,linewidth=2.0)
    plt.title("Tank water mass")
    plt.xlabel("time (s)")
    plt.ylabel("Water mass (kg)")
    plt.show()

    plt.plot(list_time,list_watertemp,linewidth=2.0)
    plt.title("Tank water temperature")
    plt.xlabel("time (s)")
    plt.ylabel("Water temperature (oC)")
    plt.show()

    plt.plot(list_time,list_shower_completed,linewidth=2.0)
    plt.title("Number of showers completed")
    plt.xlabel("time (s)")
    plt.ylabel("Showers completed")
    plt.show()

    plt.plot(list_time,list_shower_remain,linewidth=2.0)
    plt.title("Number of showers remain")
    plt.xlabel("time (s)")
    plt.ylabel("Showers remain")
    plt.show()

    plt.plot(list_months,list_shower_count,linewidth=2.0)
    plt.title("Average number of showers per day - Seville")
    plt.xlabel("Months")
    plt.ylabel("Number of showers")
    plt.show()

    pv_power.terminate()
    Battery.terminate()
```

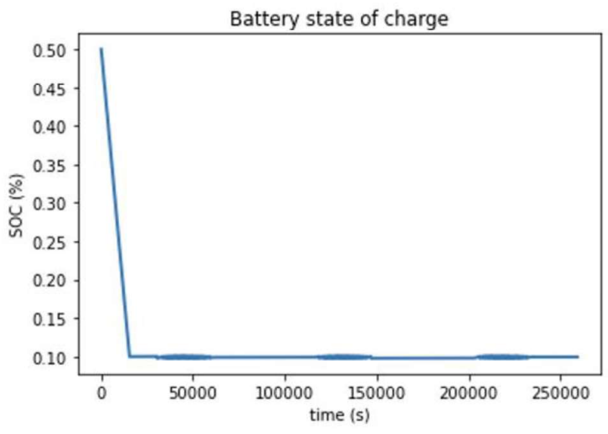
Appendix C: Results for Dunkerque

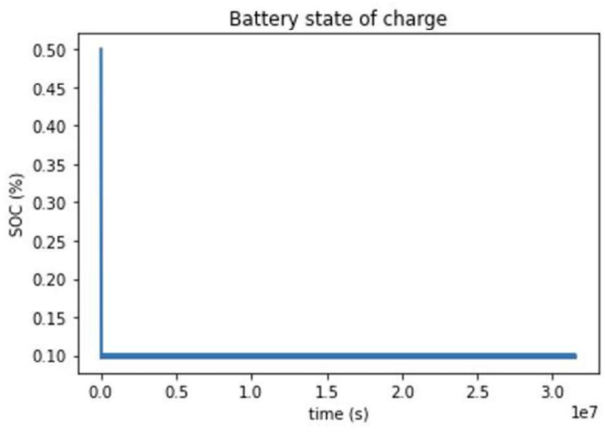


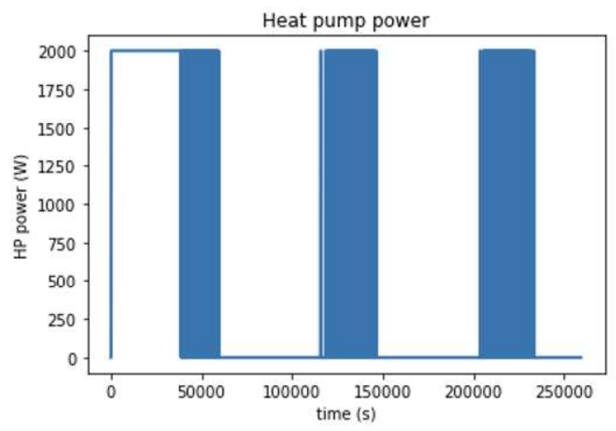


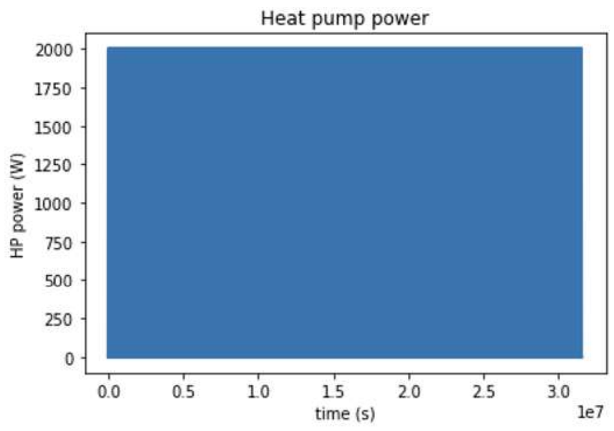


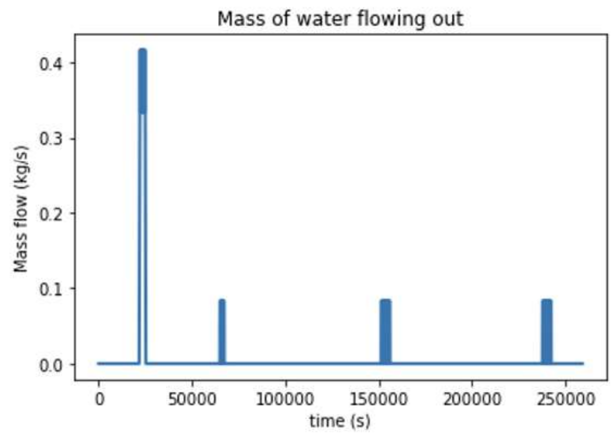


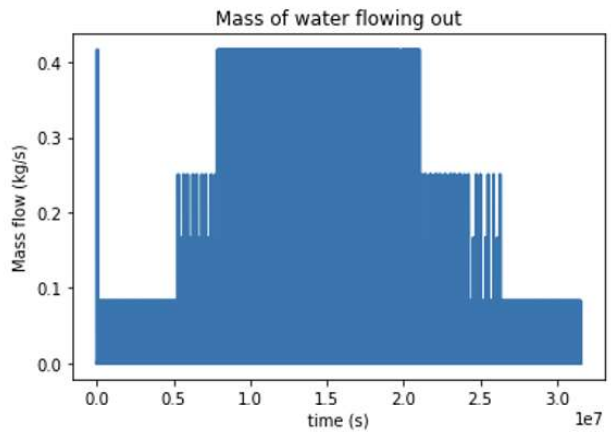


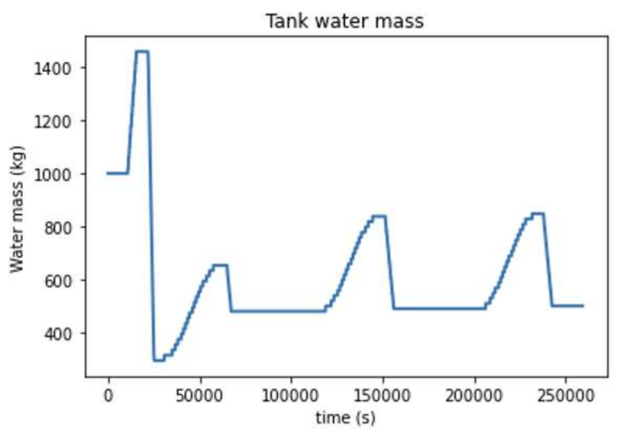


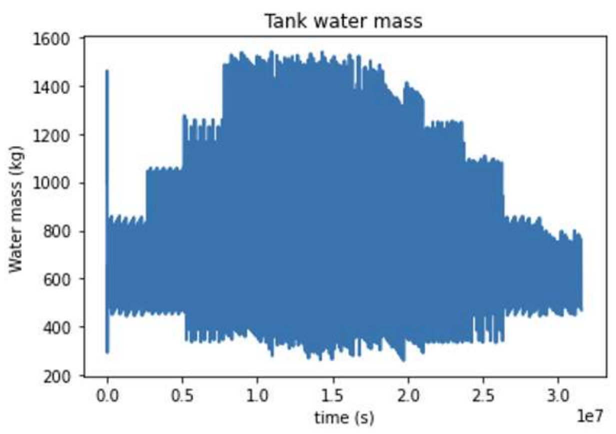

Heat pump power
HP power (W)
time (s)
0
250
500
750
1000
1250
1500
1750
2000
0
50000
100000
150000
200000
250000

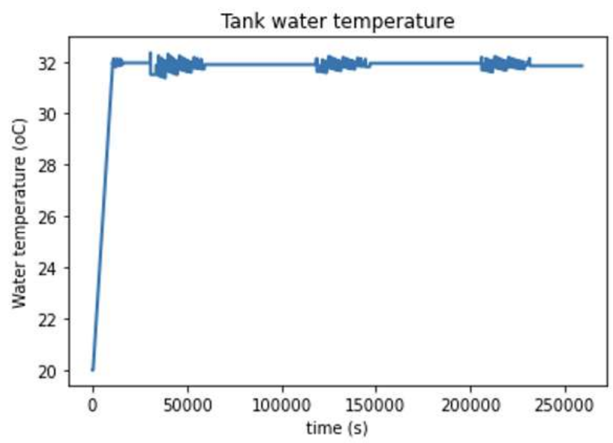

Heat pump power
HP power (W)
time (s)
1e7
0
250
500
750
1000
1250
1500
1750
2000
0.0
0.5
1.0
1.5
2.0
2.5
3.0

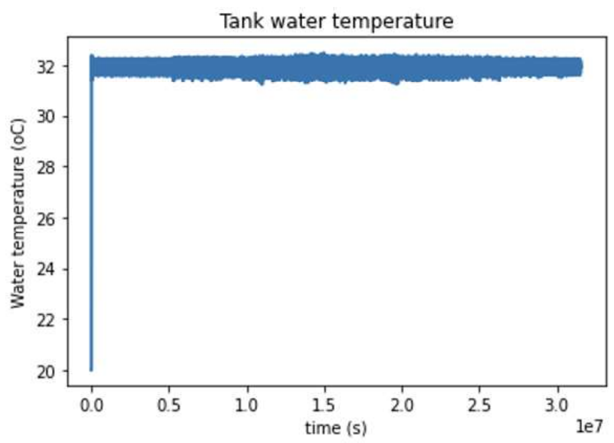

Mass of water flowing out
Mass flow (kg/s)
time (s)
0.0
0.1
0.2
0.3
0.4
0
50000
100000
150000
200000
250000

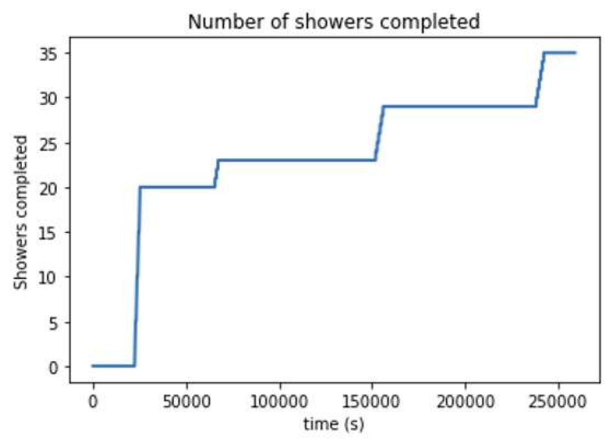

Mass of water flowing out
Mass flow (kg/s)
time (s)
1e7
0.0
0.1
0.2
0.3
0.4
0.0
0.5
1.0
1.5
2.0
2.5
3.0

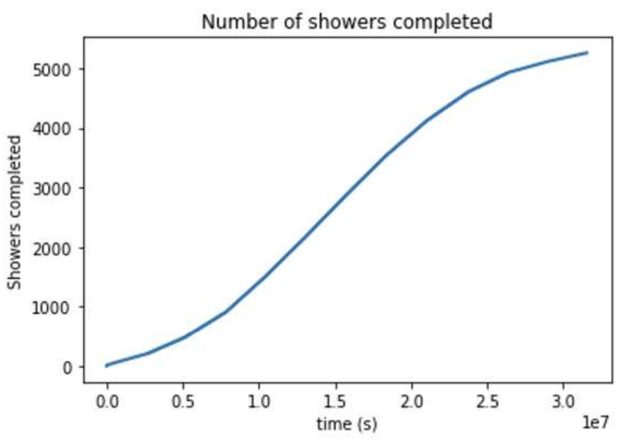

Tank water mass
Water mass (kg)
time (s)
400
600
800
1000
1200
1400
0
50000
100000
150000
200000
250000

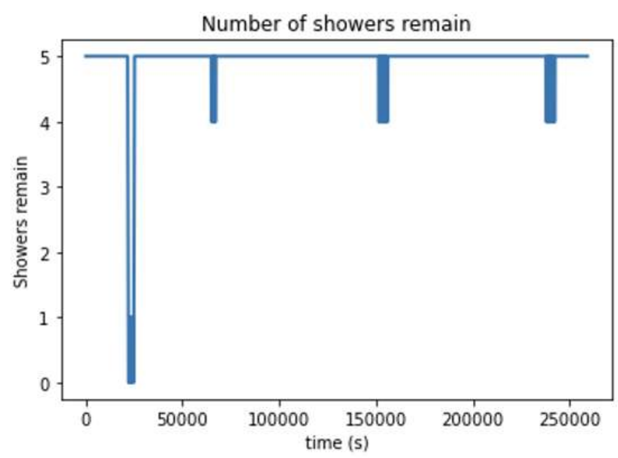

Tank water mass
Water mass (kg)
time (s)
1e7
200
400
600
800
1000
1200
1400
1600
0.0
0.5
1.0
1.5
2.0
2.5
3.0

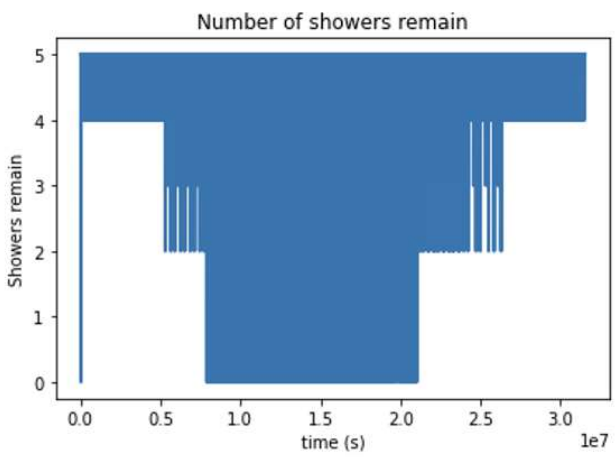

Tank water temperature
Water temperature (oC)
time (s)


Tank water temperature
Water temperature (oC)
time (s)
1e7


Number of showers completed
Showers completed
time (s)


Number of showers completed
Showers completed
time (s)
1e7


Number of showers remain
Showers remain
time (s)


Number of showers remain
Showers remain
time (s)
1e7