OSEM: Future Recommendations

# Current Status:

In this iteration, the following control strategy depicted in figure 1 was designed and implemented on the hardware.

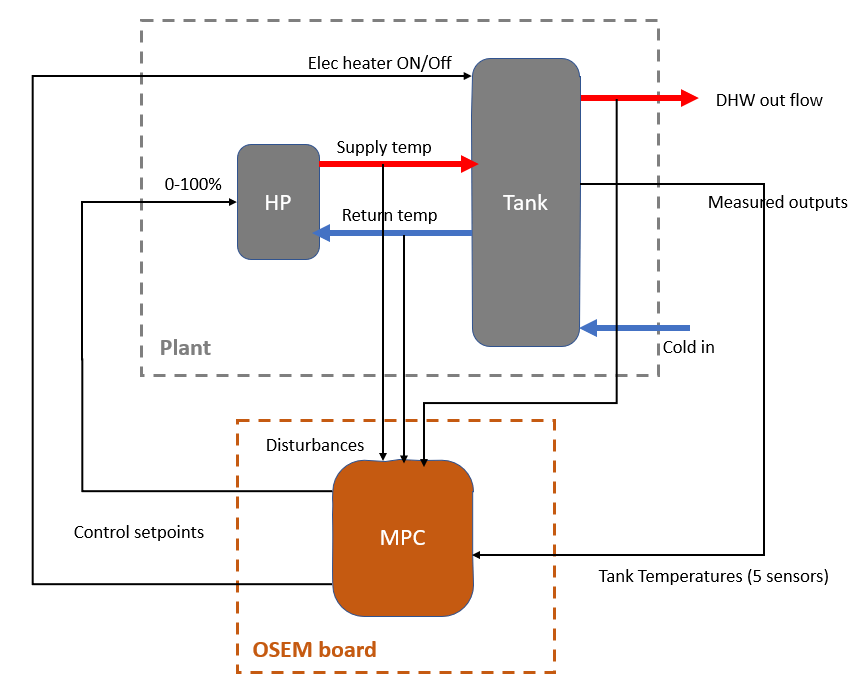


Figure : 1st Iteration of the OSEM

# Future recommendations

## System Identification of the plant model

In this iteration, the dynamic model of the plant was developed from first principles. The model parameters conform to the experimental setup at TNO lab. If the controller is applied to a different system. The model parameters need to be changed to conform to that particular system. However, since the system at TNO lab is an experimental one, all the parameters are known with good degree of accuracy. This may not be the case in other systems where commercial heat pump and storage tank are used and not all the parameters are published by the manufacture.

To address this issue. The plant model can be identified from data suing system identification techniques. A script can be written that captures input/output data of the plant and fits the data into an “empty” model to find out the parameters of the system. Thereby obtaining an accurate model for the specific application. Another advantage of using system identification is that the more data captured during the operation of the system, the more accurate the model can continuously be made over time.

## Prediction of disturbances

In the current implementation, the prediction of the MPC is only based on the dynamic model of the heat pump and the tank. The disturbances are measured, as shown in figure 1. However, when the prediction is carried out, it is assumed that the some of the measured disturbances (DHW flow rate and ambient temperature) remain constant for the duration of the prediction horizon. This is a valid assumption when the prediction horizon is short. However, when the prediction horizon is long, this will cause inaccuracy in the predictions.

An improvement to the current controller can be made by introducing predictors of ambient temperature and DHW flow rate. For ambient temperature, there exist multiple free APIs that can provide accurate predictions of the weather conditions, those APIs can be integrated in the MPC code. Examples of weather APIs are the [Open Weather API](https://openweathermap.org/), and the Dutch [KNMI API](https://www.knmi.nl/kennis-en-datacentrum/achtergrond/data-ophalen-vanuit-een-script), which provides a tutorial on how to write a script to extract weather predictions.

The prediction of DHW flow rate can be done locally, because each house may have a slightly different DHW usage pattern depending on the habits of the occupants. The current implemented system already includes a flow meter to measure the DHW flow rate. This already provides the data source. A data-driven model can be developed to predict the future DHW tapping based on the past data. Time-series prediction has been the subject of many publications in literature for various applications. This [publication](https://www.sciencedirect.com/science/article/pii/S0360544221009270) describes developing a model for prediction of DHW.

## PV Panels and Energy Prices

In the current iteration, no assumption was made about the source electric energy nor the price of electric energy. In the current iteration, the cost function is defined as follows:

The full explanation of the cost function is elaborated in WP4 document. The relevant variables in this discussion are the matrices Q and R, which can be chosen by the designer.

The matrix R is the weight assigned to the control action. The higher the weight, the more the control action will contribute to the cost function. Therefore, the higher the weight, the more the controller will try to restrict the actuator. In this iteration, the matrix R was chosen as:

This simply tells the controller to restrict the electric heater (The first actuator), and freely manipulate the heat pump. This is logical since in most cases, the COP of the pump is higher than 1 and therefore it is more efficient to use than the electric heater.

However, when the installation includes PV panels, this can change the discussion. When there is abundance of PV. It can be argued that it’s better to use maximize the self-use of PV power instead of exporting it to the grid, given the changes to the SDE subsidy which will restrict the compensation of PV export.

Therefore, the MPC algorithm can be improved to adapt to that situation as follows: Instead of having a constant weight as in the current iteration. A dynamic weight (i.e. time varying weight) can be assigned to the electric heater. This dynamic weight can be a function of the solar irradiation. The higher the solar irradiation, the higher the PV production, and consequently, the more the electric heater can be used “cheaply”. The solar irradiation can be read from the same APIs described in the previous section.

The weight of the heat pump actuator can also be made dynamic. The COP of the heat pump is dependent on the ambient temperature. Assigning the weight of the heat pump relative to the ambient temperature will incentivize the heat pump to operate when COP is favorable.

Based on the discussion above, the following changes can be introduced to the R matrix:

Where (t) is the dynamic weight of the heat electric heater. This weight can be defined as:

Where is the solar irradiation [W/m2] at time t. is a function that normalizes the weight value between 0 and 1 to keep the weights numerically comparable and the calculations stable.

With similar reasoning, is the dynamic weight of the heat pump, which can be defined as:

Where is the ambient temperature at time t. And f1 is a function that normalizes the value of the weight between 0 and 1.

In most households, the energy pricing is based on day/night tariff. However, if dynamic pricing is present. The prices can be included in the weight calculations using similar approach.

The matrix Q is the weight assigned to the deviation of the temperatures from the setpoint. The higher the weight, the more it contributes to the cost function, therefore, the higher the weight, the more the controller will try to keep the temperature close to the set point. In this iteration, the matrix Q was implemented as:

This ensures that the upper layers are kept at the desired setpoint at all times, while the lower tanks are allowed to drop. It was shown in the main document that this approach contributes to keeping the tank stratified and therefore maximizing the useful energy content of the tank. However, it can be argued that always keeping the upper layers at higher temperatures also costs money. Saving can be made by ensuring the tank has higher temperature at the moments when DHW draw-off happens, and then allowed to slightly drop when there is no demand.

This can be achieved by changing the constant weights into dynamic weights. Assigning higher weights when there is DHW tapping, and lower weights when there is no tapping. The recommended dynamic weight would look as follows:

Where is the flow rate of DHW at time t. and g is a function that normalizes the weight between 0 and 10 to keep the weights numerically comparable and the calculations stable.

## Additional Actuator:

In the current implementation, 2 actuators are present in the system. Namely, the heat pump set point and the electric heater setpoint. Analysis of the model indicates that a third actuator may improve the control results. By controlling the flow rate between the heat pump and the spiral heat exchanger inside the tank, more degree of freedom is present in the system and mathematically more control can be excreted. The flow rate of the water inside the spiral exchanger has a great influence on the heat transferred from the heat pump to the tank. It also has a big influence on the amount of heat transferred from the upper layers to the lower layers, and consequently on the stratification of the tank.

A controllable valve can be placed between the heat pump and the tank. The control setpoint of the valve can then be actuated by the MPC. OSEM hardware board already contains output ports for controlling 2 valves. So, no upgrade is required in the hardware or firmware of OSEM board.

The behavior of the valve and its effect on the heat transfer are highly nonlinear. So, it needs to be investigated whether a linearized model can still provide sufficiently accurate results.

## Nonlinear MPC

In the current implementation, the nonlinear model was linearized into a linear state space model, based on which a linear MPC was designed. This was justified by the fact the system will always operate close to the linearization point (40 [C]). However, the dynamics of the heat pump and the stratified tank are highly nonlinear. Therefore, it is reasonable to say that using the nonlinear model for prediction, and for the design of a nonlinear MPC can provide more accurate results.

Moreover, in the nonlinear MPC, the cost function can be freely defined by the designer. To give a context: In the linear MPC, the form of the cost function is restricted to the equation shown in the previous section. The designer can only choose the matrices Q and R to force the controller perform in the required manner. However, in nonlinear MPC, the designer can write the objective function from scratch and also choose nonlinear equality and inequality constraints. For example, the cost function can be written directly in Euros with dynamic energy prices, or KWh. Nonlinear MPCs can also be designed and implemented in MATLAB/Simulink.

## Python Implementation

In this iteration, the MPC code running on the Raspberry Pi is a C code autogenerated using Simulink. MATLAB/Simulink has a huge advantage because the mathematical modelling, simulations and data analysis can be carried out quickly (More time efficient). Moreover, the code can be autogenerated and deployed on the hardware in straightforward step. This makes MATLAB/Simulink an attractive choice for speedy prototyping and developing a first proof of concept. The only disadvantage of MATLAB/Simulink is that its expensive to acquire the license for commercial purposes and is not affordable for most SMEs.

During the first iteration, Python was investigated as possible development platform for the MPC. Indeed, Python offers a number of libraries for simulating and deploying MPC controllers. In the literature folder of OSEM GitHub repository, the results of this investigation are reported. As well as a few publications describing MPC implementation in Python.

[Model predictive control python toolbox (DO-MPC)](https://www.do-mpc.com/en/latest/) is a comprehensive open-source toolbox for model predictive control (MPC) and moving horizon estimation (MHE). do-mpc enables the efficient formulation and solution of control and estimation problems for linear and nonlinear systems, including tools to deal with uncertainty and time discretization. The modular structure of do-mpc contains simulation, estimation and control components that can be extended and combined to fit many different applications.

[paho-mqtt](https://pypi.org/project/paho-mqtt/) package n python enables applications to connect to an MQTT broker to publish messages, and to subscribe to topics and receive published messages. It also provides some helper functions to make publishing one off messages to an MQTT server very straightforward.

With the 2 libraries mentioned above, a Python script can be developed to implement the MPC algorithm, and to exchange data on the with OSEM hardware using MQTT. As well as scripts to acquire weather data from APIs.