Vaccination Center Climate Control

Model Predictive Control Programming Exercise

Andrea Da Col, Nicola Loi, Filippo Spinelli

Spring 2021

I System Modeling

1. The matrices A^c, B^c, and d^c for the continuous dynamics of the system are constructed as:

$$A^c = \begin{pmatrix} \frac{-\alpha_{F1,VC}-\alpha_{F2,VC}-\alpha_{ENV,VC}}{m_{VC}} & \frac{\alpha_{F1,VC}}{m_{VC}} & \frac{\alpha_{F2,VC}}{m_{VC}} \\ \frac{\alpha_{F1,VC}}{m_{F1}} & \frac{-\alpha_{F1,VC}-\alpha_{F2,F1}}{m_{F1}} & \frac{\alpha_{F2,F1}}{m_{F1}} \\ \frac{\alpha_{F2,VC}}{m_{F2}} & \frac{\alpha_{F2,F1}}{m_{F2}} & \frac{-\alpha_{F2,VC}-\alpha_{F2,F1}}{m_{F2}} \end{pmatrix} = 10^{-3} \times \begin{pmatrix} -0.0863 & 0.0054 & 0.0095 \\ 0.0577 & -0.2333 & 0.1756 \\ 0.3077 & 0.5269 & -0.8346 \end{pmatrix} \tag{1}$$

$$B^c = \begin{pmatrix} \frac{\beta_{11}}{m_{VC}} & \frac{\beta_{12}}{m_{VC}} & \frac{\beta_{13}}{m_{VC}} \\ \frac{\beta_{21}}{m_{F1}} & \frac{\beta_{22}}{m_{F1}} & \frac{\beta_{23}}{m_{F1}} \\ \frac{\beta_{31}}{m_{F2}} & \frac{\beta_{32}}{m_{F2}} & \frac{\beta_{33}}{m_{F2}} \end{pmatrix} = 10^{-5} \times \begin{pmatrix} 0.0190 & -0.0012 & -0.0012 \\ 0.0256 & 0.2051 & 0.0128 \\ 0.0769 & 0.1154 & 0.6923 \end{pmatrix} \tag{2}$$

$$d^c = \begin{pmatrix} d_{VC} + \alpha_{ENV,VC} * T_{ENV} \\ d_{F1} \\ d_{F2} \end{pmatrix} = \begin{pmatrix} 3900 \\ 0 \\ 0 \end{pmatrix} \tag{3}$$

2. For the exact discretization of the matrices of the continuous dynamics, the following formulas are used:

$$A = e^{T_S A^c} = \begin{pmatrix} 0.9948 & 0.0003 & 0.0006 \\ 0.0035 & 0.9863 & 0.0102 \\ 0.0180 & 0.0306 & 0.9513 \end{pmatrix} \tag{4}$$

$$B = (A_c)^{-1}(A-I)B^c = 10^{-3} \times \begin{pmatrix} 0.0114 & -0.0007 & -0.0006 \\ 0.0155 & 0.1226 & 0.0098 \\ 0.0454 & 0.0694 & 0.4053 \end{pmatrix} \tag{5}$$

$$B_d = (A^c)^{-1}(A-I)B_d^c = 10^{-3} \times \begin{pmatrix} 0.0142 & 0.0000 & 0.0001 \\ 0.0000 & 0.1528 & 0.0024 \\ 0.0001 & 0.0024 & 0.4502 \end{pmatrix} \tag{6}$$

3. The steady state temperature T_{sp} and associated steady state input p_{sp} are:

$$T_{sp} = \begin{pmatrix} 25 \\ -42 \\ -18.5 \end{pmatrix} \qquad p_{sp} = B^{-1}((I-A)T_{sp} - B_d d) = \begin{pmatrix} 8237.3 \\ -4911.1 \\ -241.5 \end{pmatrix} \tag{7}$$

To regulate the system to the computed steady-state, the delta formulation is:

$$\Delta T(k+1) = A\Delta T(k) + B\Delta u(k) = A(T(k)-T_{sp}) + B(p(k)-p_{sp}) = \\ A\begin{pmatrix} T_{VC}(k) - 25 \\ T_{F1}(k) + 42 \\ T_{F2}(k) + 18.5 \end{pmatrix} + B\begin{pmatrix} p_{VC}(k) - 8237.3 \\ p_{F1}(k) + 4911.1 \\ p_{F2}(k) + 241.5 \end{pmatrix} \tag{8}$$

4. Given the state and input constraints T_{min}, T_{max}, u_{min}, and u_{max}, the state and input constraints for the delta formulation are:

$$\begin{pmatrix} T_{VC,min} - T_{VC,sp} \\ T_{F1,min} - T_{F1,sp} \\ T_{F2,min} - T_{F2,sp} \end{pmatrix} \leq \Delta T(k) \leq \begin{pmatrix} T_{VC,max} - T_{VC,sp} \\ T_{F1,max} - T_{F1,sp} \\ T_{F2,max} - T_{F2,sp} \end{pmatrix} \implies \begin{pmatrix} -3 \\ -3 \\ -0.5 \end{pmatrix} \leq \Delta T(k) \leq \begin{pmatrix} 2 \\ 3 \\ 1.5 \end{pmatrix} \tag{9}$$

$$\begin{pmatrix} u_{VC,min} - u_{VC,sp} \\ u_{F1,min} - u_{F1,sp} \\ u_{F2,min} - u_{F2,sp} \end{pmatrix} \leq \Delta u(k) \leq \begin{pmatrix} u_{VC,max} - u_{VC,sp} \\ u_{F1,max} - u_{F1,sp} \\ u_{F2,max} - u_{F2,sp} \end{pmatrix} \implies \begin{pmatrix} -8237.3 \\ -2588.9 \\ -1258.6 \end{pmatrix} \leq \Delta u(k) \leq \begin{pmatrix} 6762.7 \\ 4911.1 \\ 241.5 \end{pmatrix} \tag{10}$$

II Unconstrained Optimal Control

5. Running the autonomous system starting at $T_{init}^{(1)}$ without any control law applied results in an unfeasible behavior in open loop, where the inputs are set to 0 by default. Figure 1 shows how the temperature of the vaccination center tends to decrease while the ones of the freezers increase, all of them approaching the external environmental temperature of 13°. It is therefore needed a control action to keep the system states within their boundaries.

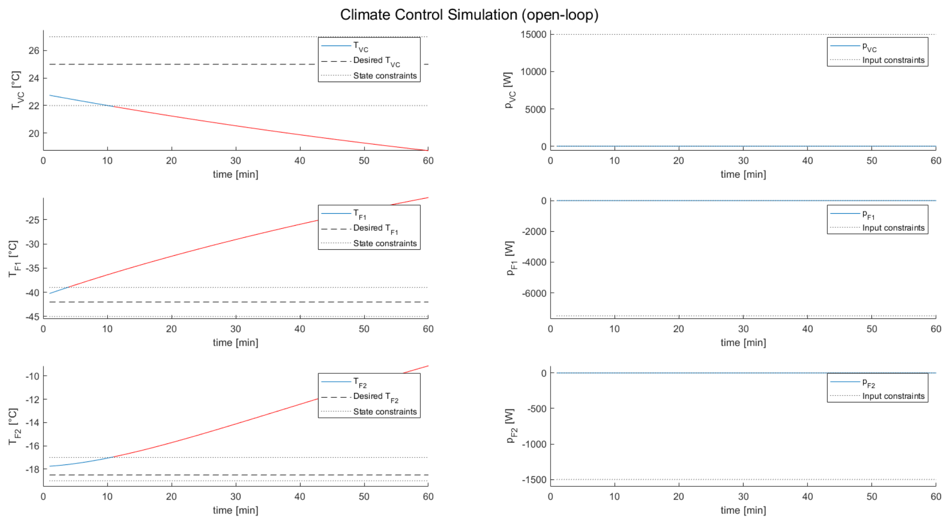

Figure 1: Uncontrolled system starting from $T(0) = T_{init}^{(1)}$

6. From the heuristic search of the best Q for $T(0) = T_{init}^{(1)}$, it can be observed that the results obtained with the different Q follow a particular trajectory in the scatter plot (Figure 2).

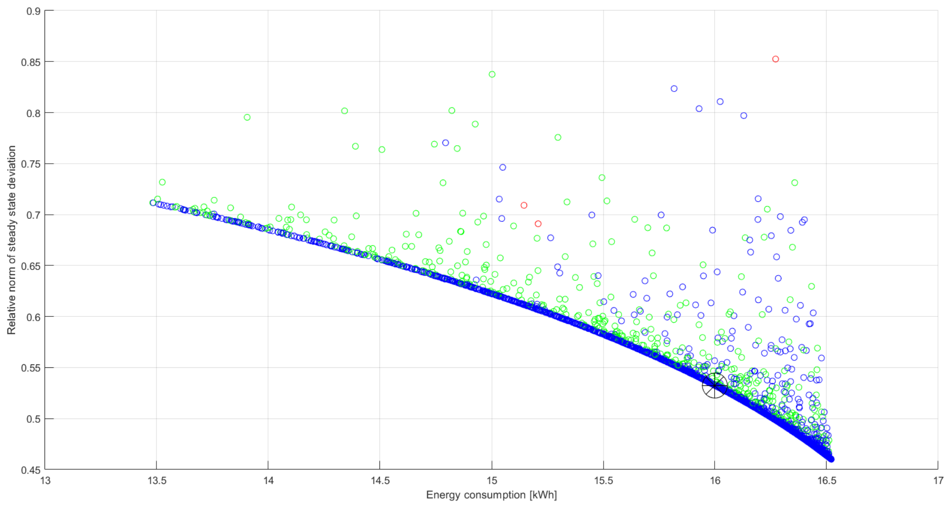

Figure 2: Q matrices performance

The relative norm of steady state deviation (y-axis) is strongly correlated to the energy consumption (x-axis). The best Q with the smallest norm that satisfies both the state constraints and the energy constraint ($\leq$16 kWh) is:

$$Q = \begin{pmatrix} 4973679 & 0 & 0 \\ 0 & 5427908 & 0 \\ 0 & 0 & 4949349 \end{pmatrix} \tag{11}$$

whose corresponding point in the previous plot is marked by a black wheel. Most of the points are blue (violation of input constraints only); the fewer green ones (no constraints violation) have generally a larger relative norm than blue ones for the same energy consumption: obviously for the system is simpler to get closer to the steady state temperature without having to respect the input constraints.

7. The trajectories of the temperatures employing the `lqr_controller` with $T(0) = T_{init}^{(1)}$ and the chosen Q are shown in Figure 3.

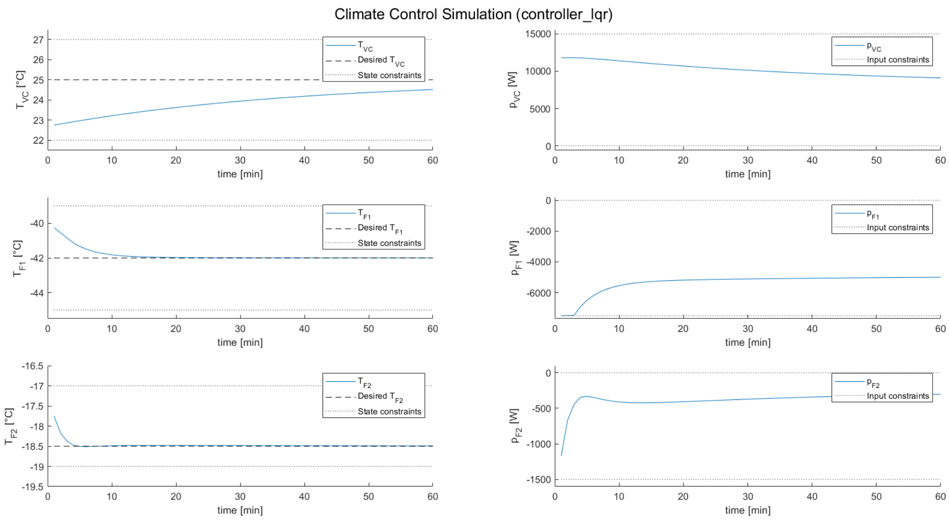

Figure 3: Closed-Loop system starting from $T(0) = T_{init}^{(1)}$

Changing the values in the diagonal of matrix Q or R will change how much the states and inputs are weighted in the optimization cost. Higher values will increase the weight of the states (Q) or the inputs (R), and vice-versa. Increasing a diagonal element in Q means that the controller should focus more on bringing that state towards the steady state, while increasing a diagonal element in R will tell the controller to use an input value closer to the steady state input.

8. Using the `lqr_controller` with the chosen Q and initial condition $T(0) = T_{init}^{(2)}$, leads to the state and input trajectories of Figure 4.

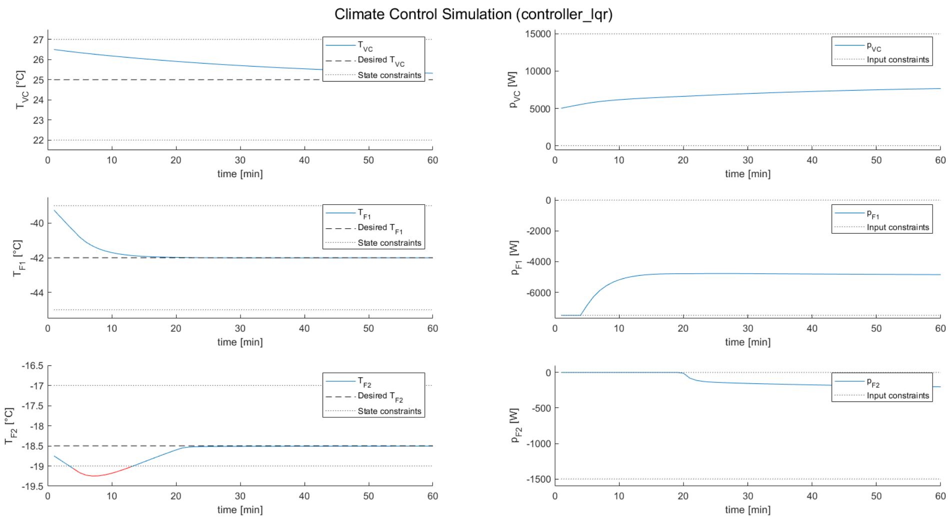

Figure 4: Closed-Loop system starting from $T(0) = T_{init}^{(2)}$

Since it is used a Q optimized to not violate state constraints for $T(0) = T_{init}^{(1)}$, it is not guaranteed that with a different initial condition they will not be exceeded. In this case with $T(0) = T_{init}^{(2)}$, the input p_{F2} saturates at the beginning and it is not able to keep temperature T_{F2} inside its constraints. In the end the controller succeeds in bringing back the temperature inside the constraints and towards the steady state, but for some minutes the state boundaries were violated.

III From LQR to model predictive controller

9. The invariant set X_{LQR} computed with the MPT toolbox, given R and the chosen Q matrices, is displayed in red in Figure 5.

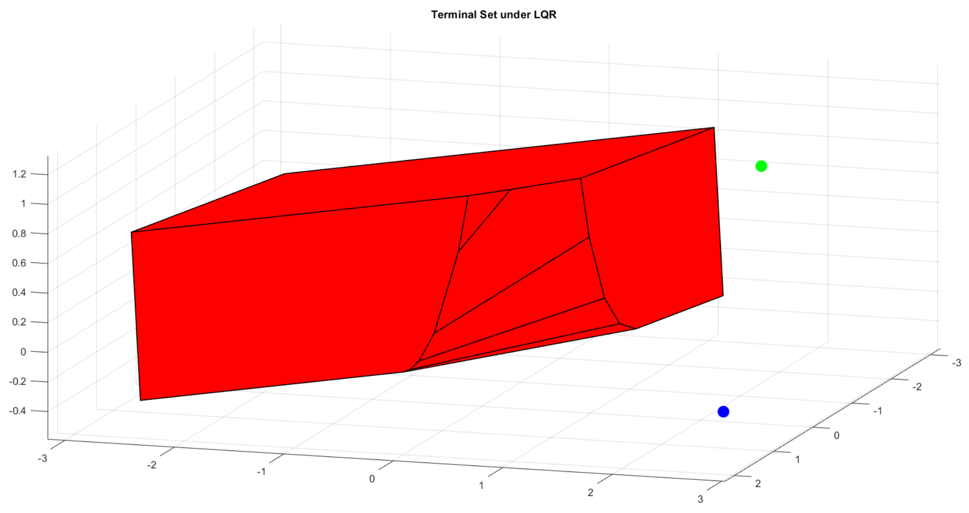

Figure 5: Invariant set X_{LQR}

The green dot represents the initial condition $T(0) = T_{init}^{(1)}$, while the blue one $T(0) = T_{init}^{(2)}$. Both are outside the invariant set, but $T_{init}^{(1)}$ is closer to it and also farthest from the limit of the state constraints, therefore controllers initialized there should work better in their task of steering the states towards their steady state values.

10. Given R and the chosen Q matrices, the solution of the Riccati equation is:

$$P^{\infty} = 10^8 \times \begin{pmatrix} 1.6156 & -0.0039 & 0.0007 \\ -0.0039 & 0.2107 & -0.0059 \\ 0.0007 & -0.0059 & 0.0804 \end{pmatrix} \tag{12}$$

from which the infinite horizon cost as a function of $x(0)$ (namely $T_{init}^{(1)} - T_{sp}$) can be calculated as:

$$J_{LQR}^{\infty}(x(0)) = x(0)^T P^{\infty} x(0) = \begin{pmatrix} -2.25 & 1.75 & 0.75 \end{pmatrix} P^{\infty} \begin{pmatrix} -2.25 \\ 1.75 \\ 0.75 \end{pmatrix} = 8.88 \times 10^8 \tag{13}$$

11. The closed-loop simulation plot using `controller_mpc_1` with $T(0) = T_{init}^{(1)}$ is presented in Figure 6.

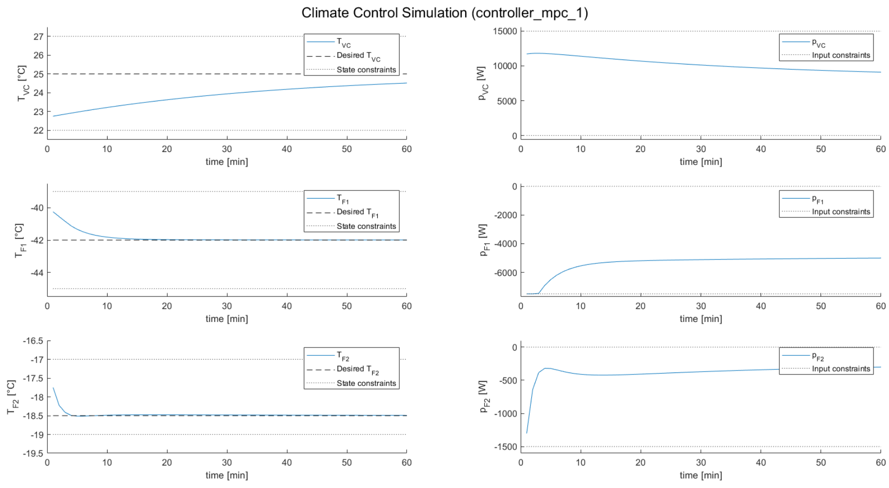

Figure 6: Closed-Loop system starting from $T(0) = T_{init}^{(1)}$

The differences with respect to the plot of `controller_lqr` of Figure 3 are very small, visible in the Figures only in the initial steps, then the numerical differences become indistinguishable in the plots. The results of the closed loops are extremely similar, but the implementation is quite different. `controller_lqr` is working with an infinite horizon (using the Matlab function `dlqr`), while `controller_mpc_1` has a finite horizon of only 30 steps, and to approximate the infinite horizon it is exploiting the LQR infinite horizon terminal cost.

But with $T(0) = T_{init}^{(2)}$, the closed loop simulation of `controller_mpc_1` (Figure 7) is different from the one of the `controller_lqr` of Figure 4.

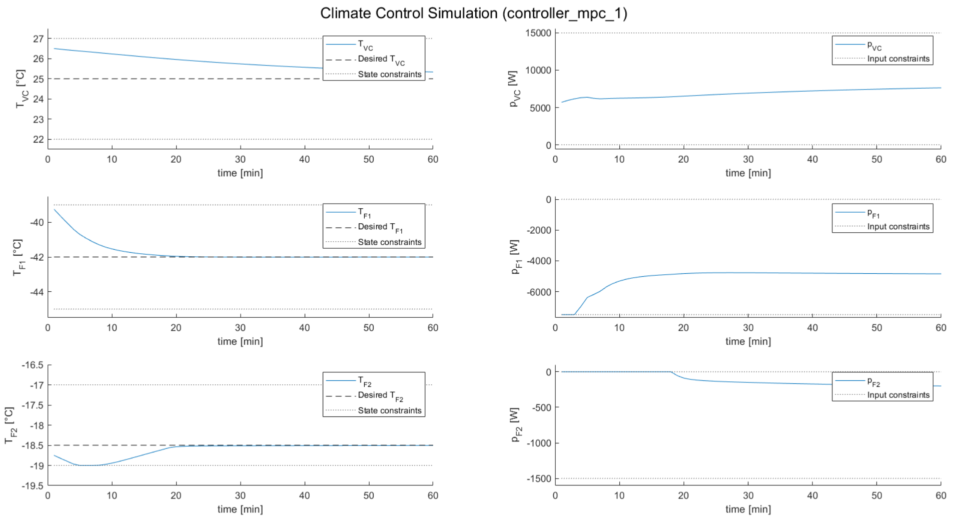

Figure 7: Closed-Loop system starting from $T(0) = T_{init}^{(2)}$

The difference is more easily visible on the trajectory of T_{F2}, which violated its state constraint with `controller_lqr`, but the same doesn't happen using `controller_mpc_1`, since it computes the input taking into account the state constraints during the optimization process. The MPC controller enhances the closed loop behaviour of the system.

IV MPC with theoretical closed-loop guarantees

12. To have the origin as an asymptotically stable point, two conditions must be met: recursive feasibility and stability of the system.

    Knowing that x(0) is feasible, the computed optimal control sequence is $\{u_0^*, u_1^*, ..., u_{N-1}^*\}$, while the corresponding state trajectory is $\{x(0), x_1^*, ..., x_N^*\}$. Applying the input u_0^* to the system, it will evolve to $x(1)$, and the new optimal control sequence is $\{u_1^*, u_2^*, ..., u_N^* = 0\}$, which is feasible (being 0 the steady state input, applying it to the terminal state 0, which is a steady state, will keep the system in the steady state $0 \in X_f$). Going on, at every time-step there will exist a feasible control sequence, hence the recursive feasibility has been proven.

    Regarding the stability, the cost chosen for the optimization, $J(x(k)) = \sum_{i=0}^{29}(x_i^T Q x_i + u_i^T R u_i)$, is a Lyapunov function, hence x(0) is Lyapunov stable.

    In conclusion, the origin is an asymptotically equilibrium since the recursive feasibility and the stability conditions are satisfied.

13. The closed-loop simulation plots employing `controller_mpc_2` are displayed in Figure 8 for the initial condition $T(0) = T_{init}^{(1)}$, and in Figure 9 for the initial condition $T(0) = T_{init}^{(2)}$:

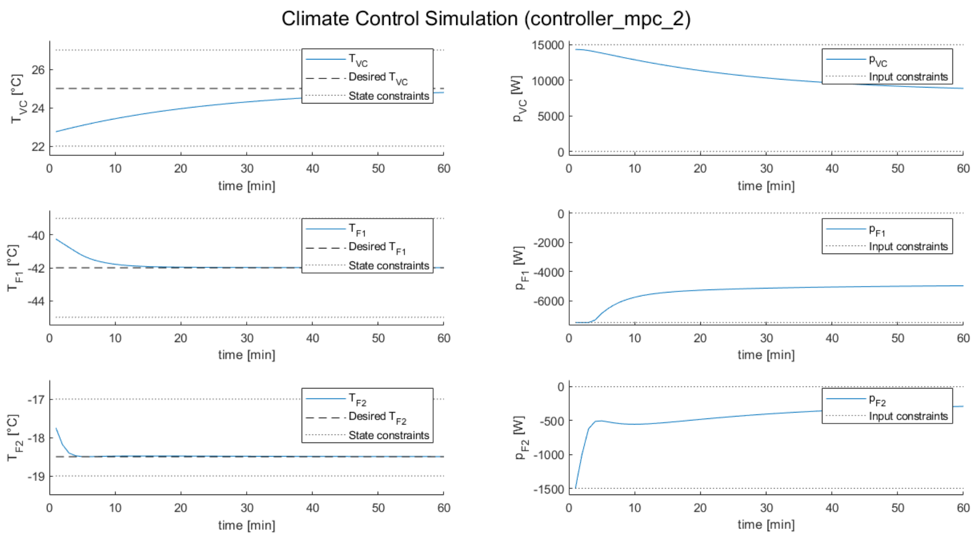

Figure 8: Closed-Loop system starting from $T(0) = T_{init}^{(1)}$

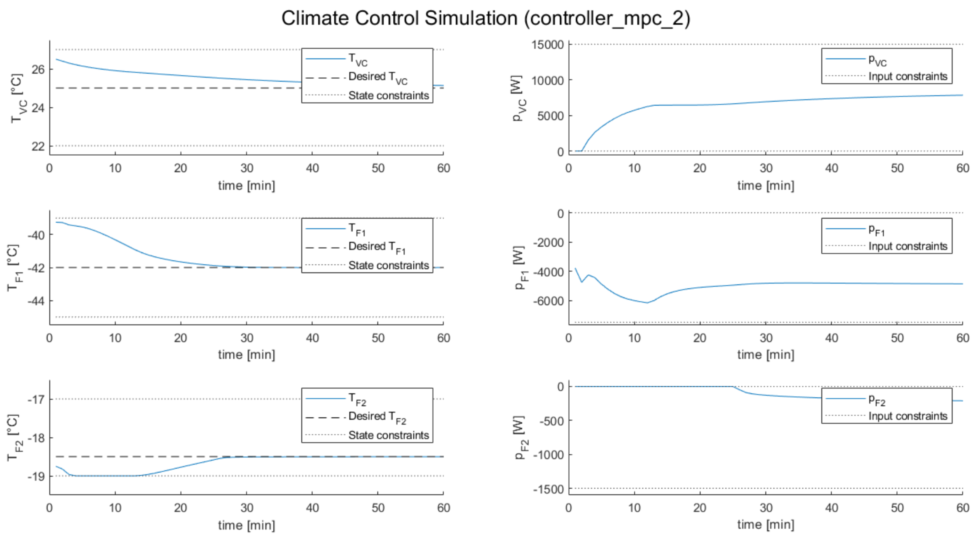

Figure 9: Closed-Loop system starting from $T(0) = T_{init}^{(2)}$

14. The closed loop simulation plot using `controller_mpc_3` with initial conditions $T(0) = T_{init}^{(1)}$ and $T(0) = T_{init}^{(2)}$ are presented in Figure 10 and Figure 11 respectively.

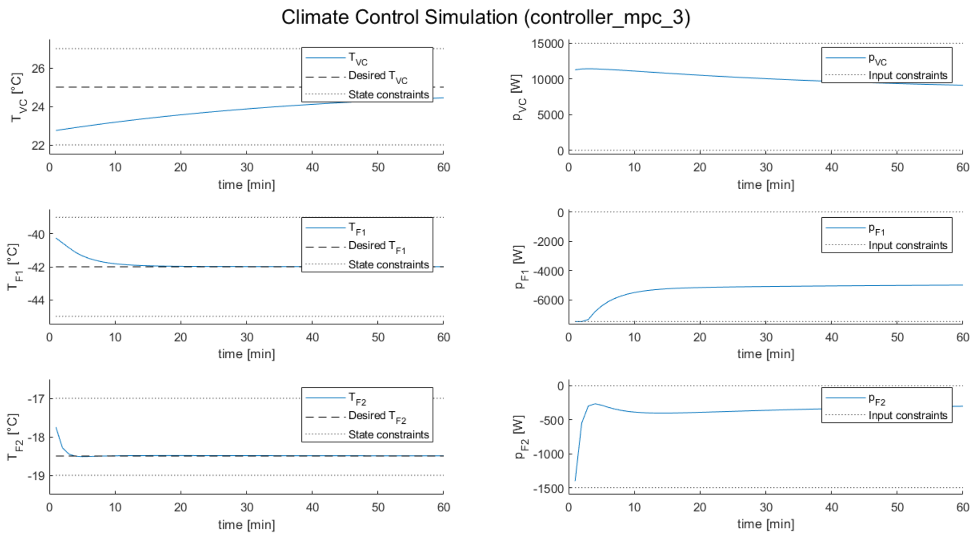

Figure 10: Closed-Loop system from $T(0) = T_{init}^{(1)}$

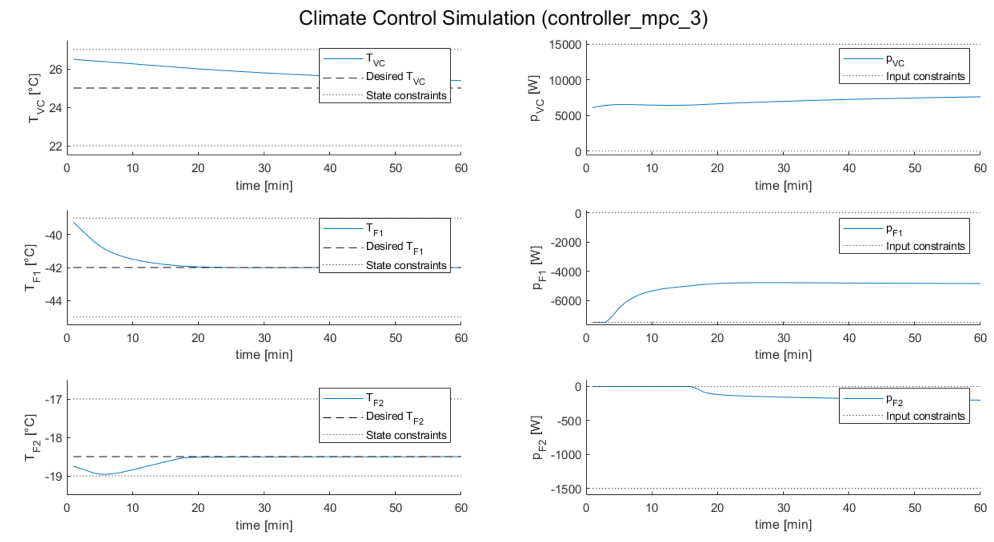

Figure 11: Closed-Loop system starting from $T(0) = T_{init}^{(2)}$

To have the origin as an asymptotically stable point, $I_f(x)$ is chosen to be $x(N)^T P^{\infty} x(N)$, to continue to have a Lyapunov function as the optimization cost (as explained in task 12). P^{∞} is the matrix of the infinite horizon LQR problem (Equation 12). The idea is to approximate the "tail" of the cost, and theoretically switch to the LQR controller once the invariant terminal set $X_f = X_{LQR}$ is reached by the sequence of the optimized state variables.

15. Figure 12 depicts state trajectories for the Closed-Loop systems (each of them employing a different MPC controller) starting at $T(0) = T_{init}^{(1)}$. Notice that the trajectory produced by `controller_mpc_1` (green) and by `controller_mpc_3` (blue) are practically the same; indeed these two controllers solve very similar optimization problems. Both exploit the same terminal cost $I_f(x)$, but `controller_mpc_3` has the terminal set $X_f = X_{LQR}$ constraint. `controller_mpc_1` has no such constraint, but nonetheless at the end is able to steer the states inside the terminal set used by `controller_mpc_3`, hence the addition of the $X_f = X_{LQR}$ constraint does not change the behaviour of the closed-loop simulation.

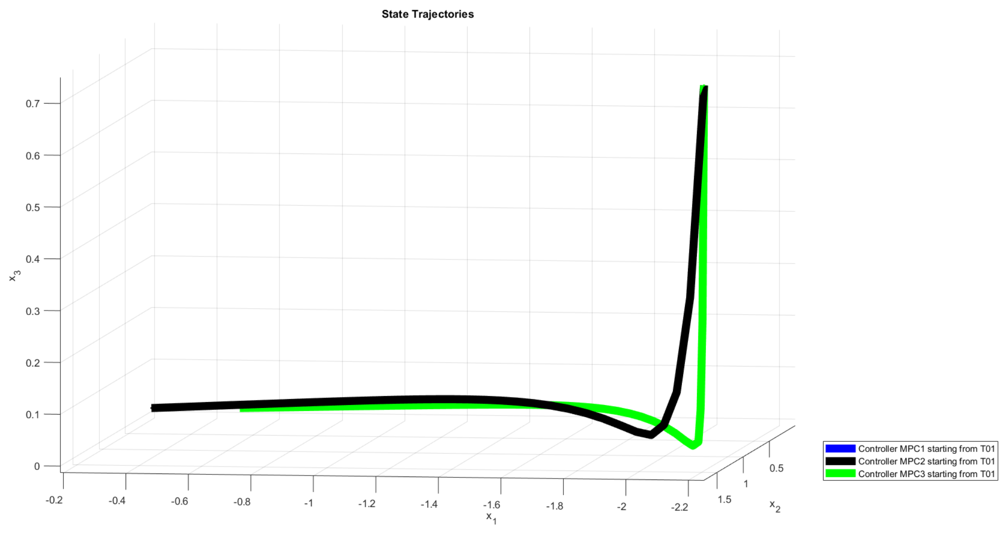

Figure 12: Trajectories for all controllers, from $T(0) = T_{init}^{(1)}$

Figure 13 shows trajectories starting at $T(0) = T_{init}^{(2)}$. The trajectories produced by `controller_mpc_1` and by `controller_mpc_3` still overlap for the same reason. In both figures we see that `controller_mpc_2` drives the state $x(k)$ closer to the origin than the other controllers thanks to the constraint on the terminal set $X_f = 0$.

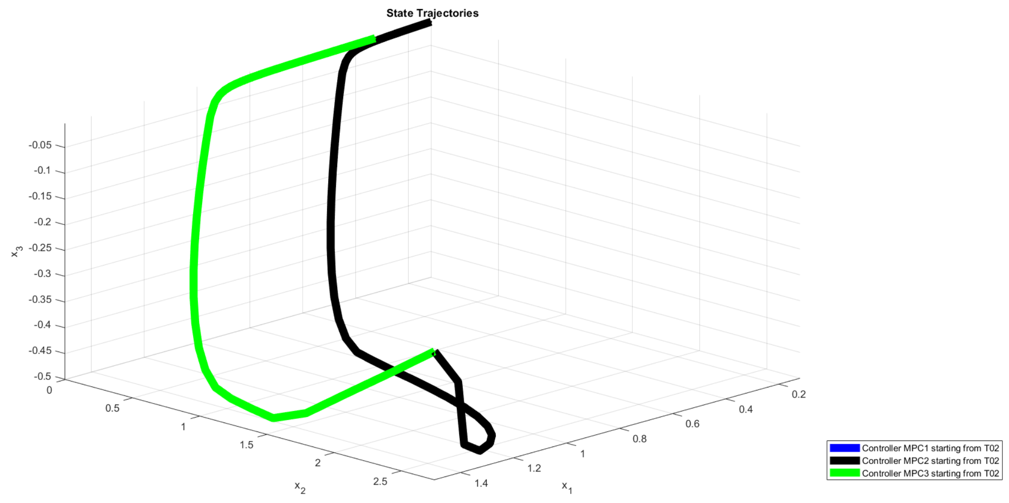

Figure 13: Trajectories for all controllers, from $T(0) = T_{init}^{(2)}$

A plot including the invariant set X_{LQR} would highlight the fact that the state evolution of the Closed-Loop using `controller_mpc_3` will enter the invariant set itself, and eventually converge to the origin.

The function `simulate_building()` provides the optimal cost as a vector with two entries, computed considering the whole simulation time, but without considering the states of the first step. The first cell is the total cost of the states (i.e. $\sum x(k)^T Qx(k)$) of the delta formulation, while the second cell gives the cost of the actual inputs (i.e. $\sum p(k)^T Rp(k)$), so without using the inputs of the delta formulation. Instead, we calculated the cost of the inputs of the delta formulation ($\sum u(k)^T Ru(k)$). The input cost computed in this way will be obviously lower, and will represent how much the power consumption is different from the steady state consumption, while the input cost computed by `simulate_building()` represents the cost of the actual energy consumption.

For a visual representation of the $T(0) = T_{init}^{(1)}$ case only, the plot in Figure 14 shows the optimal cost to go at each time step k, divided in $x(k)^T Qx(k)$ and $u(k)^T Ru(k)$, where $u(k)$ is the deviation from the steady state input at time k.

Since `controller_mpc_1` and `controller_mpc_3` produce the same results (as mentioned earlier), only the costs of `controller_mpc_3` are shown.

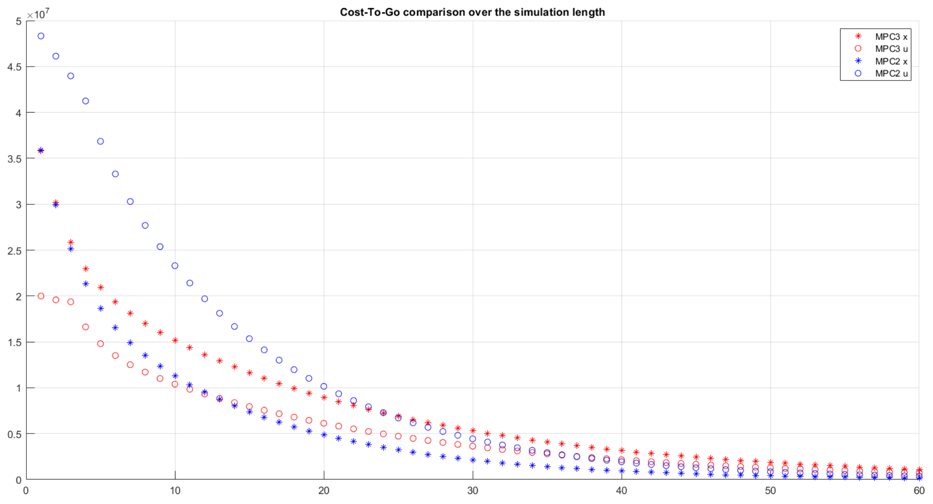

Figure 14: Optimal costs to go, from $T(0) = T_{init}^{(1)}$

Notice that the cost terms decrease gradually and, specifically, the cost associated to the input difference with respect to the steady state goes down faster in the case of `controller_mpc_2`. From the plot we can easily understand how `controller_mpc_2`, with the aim of driving the system to 0 within 30 horizon steps, deviates from the steady state input more than `controller_mpc_3`, whose objective is only to reach the terminal set. However, the cost associated to the temperature deviation from steady state is reduced faster considering 0 as terminal condition. The optimal cost to go is proposed below in the same form as it is given by the function `simulate_building()`, namely divided in two components, but as mentioned earlier, the input cost is computed differently. The state cost and input cost are calculated from the states $x(k)$ and inputs $u(k)$ of the delta formulation. For the initial state $x(0) = T_{init}^{(1)} - T_{sp}$:

$$\begin{aligned} J_{state,MPC1}(T_{init}^{(1)} - T_{sp}) &= 0.4826 \times 10^9 \\ J_{input,MPC1}(p(0) - p_{sp}) &= 0.3283 \times 10^9 \\ J_{TOT,MPC1} &= 0.8109 \times 10^9 \end{aligned} \tag{14}$$

$$\begin{aligned} J_{state,MPC2}(T_{init}^{(1)} - T_{sp}) &= 0.3271 \times 10^9 \\ J_{input,MPC2}(p(0) - p_{sp}) &= 0.6215 \times 10^9 \\ J_{TOT,MPC2} &= 0.9486 \times 10^9 \end{aligned} \tag{15}$$

$$\begin{aligned} J_{state,MPC3}(T_{init}^{(1)} - T_{sp}) &= 0.4826 \times 10^9 \\ J_{input,MPC3}(p(0) - p_{sp}) &= 0.3283 \times 10^9 \\ J_{TOT,MPC3} &= 0.8109 \times 10^9 \end{aligned} \tag{16}$$

Instead, when $x(0) = T_{init}^{(2)} - T_{sp}$:

$$\begin{aligned} J_{state,MPC1}(T_{init}^{(2)} - T_{sp}) &= 0.3204 \times 10^9 \\ J_{input,MPC1}(p(0) - p_{sp}) &= 0.1588 \times 10^9 \\ J_{TOT,MPC1} &= 0.4792 \times 10^9 \end{aligned} \tag{17}$$

$$\begin{aligned} J_{state,MPC2}(T_{init}^{(2)} - T_{sp}) &= 0.4286 \times 10^9 \\ J_{input,MPC2}(p(0) - p_{sp}) &= 0.3852 \times 10^9 \\ J_{TOT,MPC2} &= 0.8138 \times 10^9 \end{aligned} \tag{18}$$

$$\begin{aligned} J_{state,MPC3}(T_{init}^{(2)} - T_{sp}) &= 0.3204 \times 10^9 \\ J_{input,MPC3}(p(0) - p_{sp}) &= 0.1588 \times 10^9 \\ J_{TOT,MPC3} &= 0.4792 \times 10^9 \end{aligned} \tag{19}$$

Looking at the input costs given by `simulate_building()` for $T(0) = T_{init}^{(1)}$:

$$\begin{aligned} J_{input,MPC1}(p(0)) &= 8.1217 \times 10^9 \\ J_{input,MPC2}(p(0)) &= 8.9959 \times 10^9 \\ J_{input,MPC3}(p(0)) &= 8.1217 \times 10^9 \end{aligned} \tag{20}$$

and the input costs for $T(0) = T_{init}^{(2)}$:

$$\begin{aligned} J_{input,MPC1}(p(0)) &= 4.4526 \times 10^9 \\ J_{input,MPC2}(p(0)) &= 4.1967 \times 10^9 \\ J_{input,MPC3}(p(0)) &= 4.4526 \times 10^9 \end{aligned} \tag{21}$$

the real economic impact of the energy consumption of the three controllers can be visualized. `controller_mpc_2` consumes more in the first initial condition, and have also have a greater deviation from the steady state input, but it has a lower cost for the deviation of the steady state. Starting from the second initial condition, the cost for the input deviation from steady state is still greater, but the actual cost for the power consumption is lower, and in fact the cost for the deviation from the temperature steady state is greater. A trade off is made between the cost for the deviation from the steady state goal temperature and the cost for the actual energy consumption of the system.

16. The MPT Toolbox computes the maximal control invariant set X_{LQR} for the Closed-Loop system under the LQR policy. Therefore it must be $X_{LQR} \cap X_f = X_f$, since the maximal control invariant set contains all the other invariant sets. Since X_f is an invariant set under the LQR controller, then $X_{LQR} \cap X_f = X_f$ is an invariant set under the LQR controller. Moreover, since the stage cost is positive definite, and the terminal cost is a Lyapunov function (as explained in task 14), then the three assumptions for the asymptotically stability of a terminal set: i) positive definite stage cost, ii) invariant terminal set under the control law, iii) Lyapunov function terminal cost, are satisfied for the terminal set $X_{LQR} \cap X_f$.

    In conclusion, $X_{LQR} \cap X_f$ is asymptotically stable.

V Soft constraints

17. In scenario 2 there is a strong external disturbance occurring around step 40. Therefore, the normal MPC controller is not able to fulfill the constraints and runs unfeasible from that time on. From Figure 15, it is possible to easily identify such a problem, with a peak in the temperature state trajectory. To have a feasible behavior even in presence of large unexpected disturbances, it is possible to use a Soft-MPC controller, whose closed-loop simulation is shown in Figure 16.

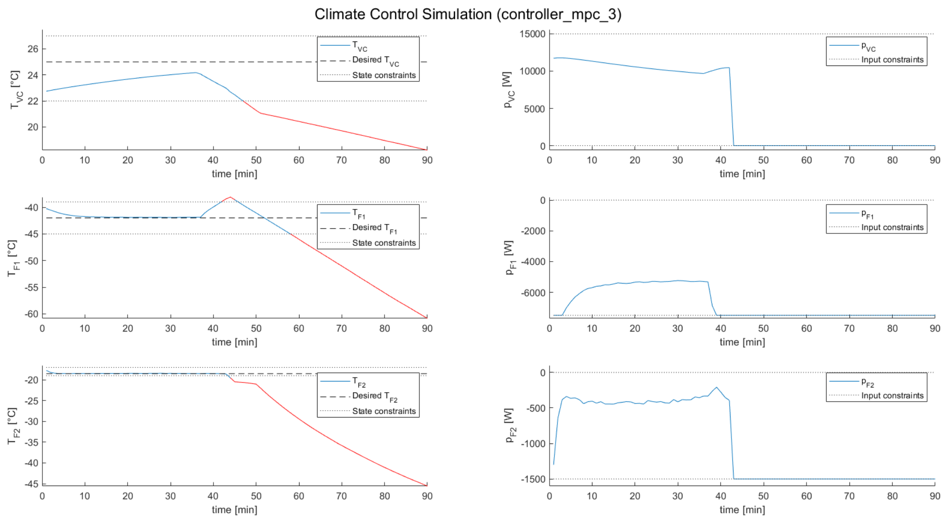

Figure 15: MPC in scenario 2, with $T(0) = T_{init}^{(1)}$

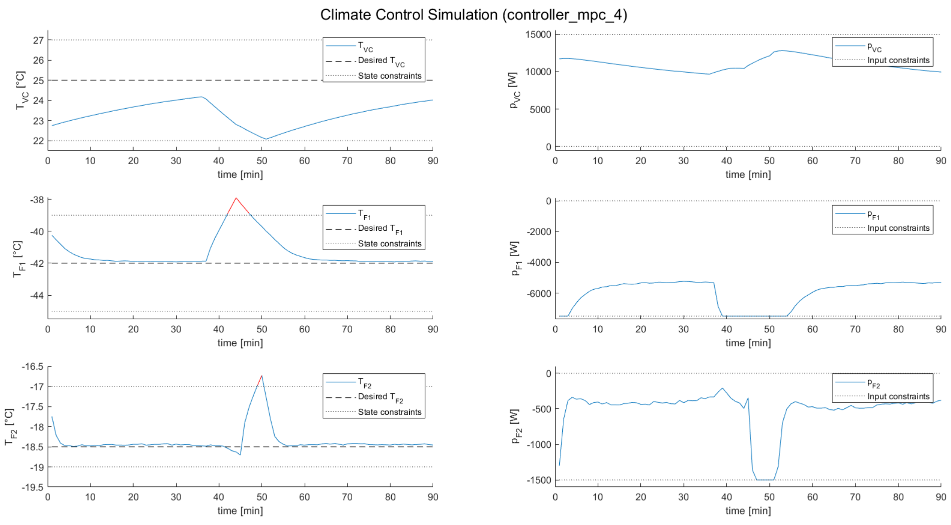

Figure 16: Soft MPC scenario 2, with $T(0) = T_{init}^{(1)}$

18. To implement a MPC controller with soft-constraints it's needed to add a slack variable in the state constraints, according to equation 22, where ε has the same dimension of the temperature state and can be any positive value.

$$X(k) \geq X_{min} - \varepsilon(k) \quad \wedge \quad X(k) \leq X_{max} + \varepsilon(k) \qquad \forall k \tag{22}$$

To avoid the optimizer to choose the slack indefinitely large (and therefore never meet any constraint), also the cost function need to be adapted penalizing any missed condition. To do so we have added a linear term and a quadratic one, as described in equation 23.

$$objective = X(k)^T QX(k) + U(k)^T RU(k) + v\|\varepsilon(k)\|_1 + \varepsilon(k)^T S\varepsilon(k) \tag{23}$$

$$S = \begin{pmatrix} 100 & 0 & 0 \\ 0 & 10000 & 0 \\ 0 & 0 & 10000 \end{pmatrix} \quad \wedge \quad v = 10000 \tag{24}$$

We have chosen v large enough to fulfill the *Exact Penalty Function* theorem, so that if `controller_mpc_3` is feasible, `controller_mpc_4` would provide the same result. S has been designed to make harder the constraints on the temperatures of the refrigerators.
The result is shown in Figure 16; we notice how the occurring disturbance makes the system to exceed its constraints, but the controller is able to recover by using the maximum allowed input for a few iterations. Changing slack parameters has no effect since the input is already saturated at that time.

19. We can verify using `scen1`, which does not have any disturbance and is feasible for all MPCs, that the soft MPC is able to achieve the same result of the normal MPC for the initial condition $T(0) = T_{init}^{(1)}$, satisfying the requirements of the problem. The simulations are shown in Figure 17 and Figure 18.

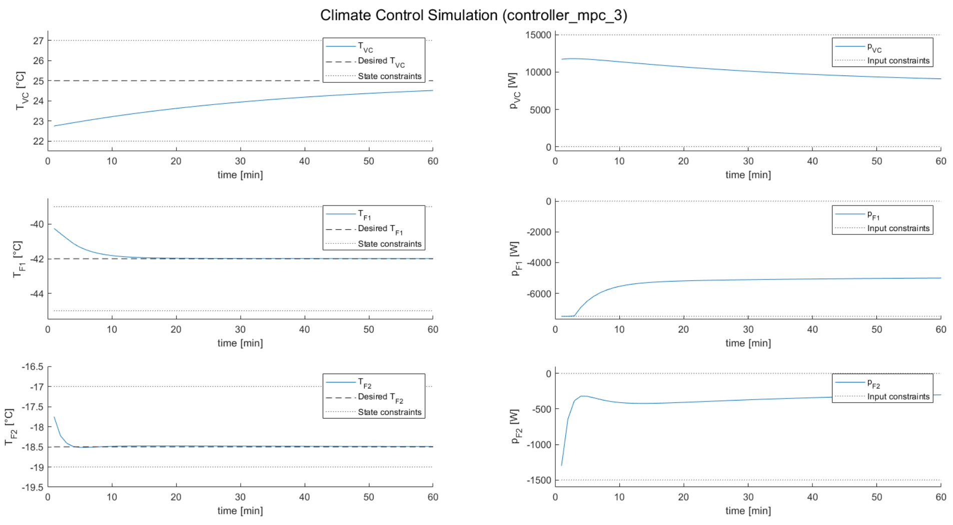

Figure 17: MPC scenario 1 starting from $T(0) = T_{init}^{(1)}$

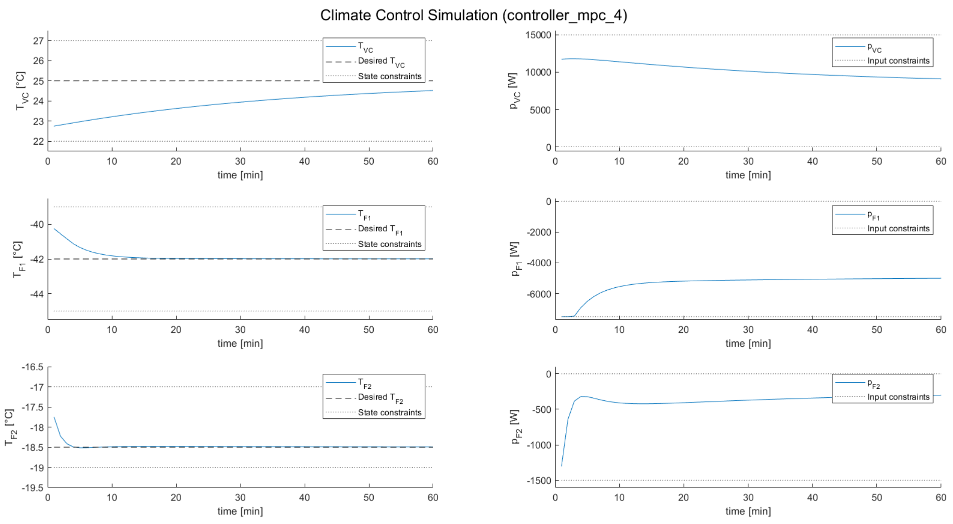

Figure 18: Soft MPC scenario 1 strating from $T(0) = T_{init}^{(1)}$

20. In this task we have to estimate from the `scen2` simulation the amount of temperature disturbance that would occur and encode it in the dynamics constraints of `controller_mpc_5`. In order to do so we have estimated each time step the temperature difference between measured and expected value, according to equation 25.

$$d_k = T_{k+1} - (AT_k + Bp_k) \tag{25}$$

Since we are not sure about the accuracy of the dynamic model (that is subjected to some approximation), and given that we want to estimate only the disturbance, we have used again equation 25 on `scen1` to find out the constant offset our estimation runs into and subtracted it. Then, given that we do not care about random noise (and trying to compensate for it is meaningless), we have kept only elements of *d(k)* whose absolute value was at least 80% of the maximum value. The resultant temperature disturbance array is described by equation 26.

$$d = \begin{pmatrix} 36 \times \mathbb{0}_{3\times 1} & \begin{matrix} -0.214 & -0.212 & -0.216 & -0.215 & -0.213 & -0.211 & -0.214 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.827 & 0.835 & 0.825 & 0.827 & 0.856 & 0.829 & 0.849 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.814 & 0.850 & 0.833 & 0.817 & 0.815 \end{matrix} & 70 \times \mathbb{0}_{3\times 1} \end{pmatrix} \tag{26}$$

In order to have a correct input for `simulate_building`, we need to add as many columns as the prediction horizon to the matrix, so that for any step a *3x30* array is provided. By including this estimated disturbance in the dynamics, we see from Figure 19 how the controller reduce the temperatures of both refrigerators just before the disturbance should occur, so that the constraints are exceeded minimally. Of course such an operation can work only if disturbance is known before both in magnitude and in time, otherwise such an approach would be detrimental.

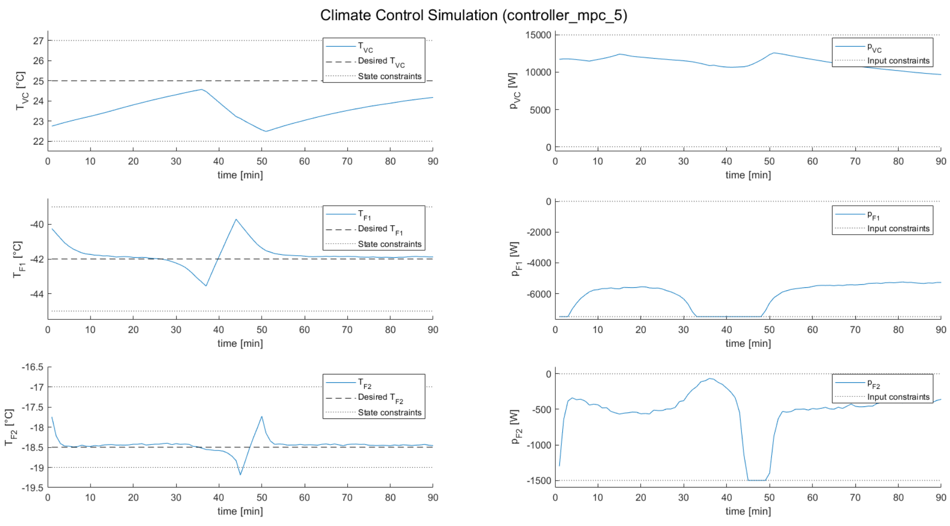

Figure 19: Soft MPC with disturbance knowledge, starting from $T(0) = T_{init}^{(1)}$

VI Offset-free MPC

21. The dynamics we want to consider in this task are expressed in equation 27, where we have defined a time-invariant disturbance.

$$x_{k+1} = Ax_k + Bu_k + B_d d_k \tag{27}$$
$$d_{k+1} = d_k \tag{28}$$
$$y_k = Cx_k + C_d d_k \tag{29}$$

Considering as states both the temperature and the disturbance, we can write the augmented state dynamics as equation 30.

$$\begin{bmatrix} x_{k+1} \\ d_{k+1} \end{bmatrix} = A_{aug} \begin{bmatrix} x_k \\ d_k \end{bmatrix} + B_{aug} u_k \tag{30}$$

$$y_k = C_{aug} \begin{bmatrix} x_k \\ d_k \end{bmatrix} + D_{aug} u_k \tag{31}$$

$$A_{aug} = \begin{pmatrix} A & B_d \\ \emptyset_{3\times3} & I_{3\times3} \end{pmatrix} \approx \begin{pmatrix} 0.9948 & 0.0003 & 0.0006 & 0.0000 & 0.0000 & 0.0000 \\ 0.0035 & 0.9863 & 0.0102 & 0.0000 & 0.0002 & 0.0000 \\ 0.0180 & 0.0306 & 0.9513 & 0.0000 & 0.0000 & 0.0005 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{32}$$

$$B_{aug} = \begin{pmatrix} B \\ \emptyset_{3\times3} \end{pmatrix} = 10^{-3} \times \begin{pmatrix} 0.0114 & -0.0007 & -0.0006 \\ 0.0155 & 0.1226 & 0.0098 \\ 0.0454 & 0.0694 & 0.4053 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{33}$$

$$C_{aug} = \begin{pmatrix} C & C_d \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{34}$$

$$D_{aug} = D = \emptyset_{3\times 3} \tag{35}$$

C is defined as an identity matrix since we measure all three temperatures, D is a null term because the input does not affect directly the output temperature but rather its time evolution, C_d is null according to equation 31. All the other terms have been already computed in previous tasks.

22. The Luenberger observer L has the purpose of estimating any missing state (in this case, since we know already all the temperatures, we look for the disturbance). The transfer function is designed so that the dynamics of the difference between real and estimated states (defined as error dynamics, equation 36) is asymptotically stable. It means that our observer would track the real disturbance, with a delay dependent on the poles. Since we want to estimate a constant disturbance in a setting where also a random time-variant noise is present, we cannot choose the poles arbitrarily fast, we just check they provide a stable system without making the optimization problem unfeasible.

$$\begin{bmatrix} x_{k+1} - \hat{x}_{k+1} \\ d_{k+1} - \hat{d}_{k+1} \end{bmatrix} = (A_{aug} - LC_{aug}) \begin{bmatrix} x_k - \hat{x}_k \\ d_k - \hat{d}_k \end{bmatrix} \tag{36}$$

To design L we have used pole-placement technique, based on the Ackermann's formula. In Matlab, $place(A^T, C^T, q)^T$, where q is an array containing the poles' positions desired. We have chosen as poles the ones in equation 37. Such a choice arises from the will of having a strongly damped system in order to reduce the estimate oscillation, even though that affects the convergence speed. By trying different combinations with poles closer to origin (and therefore leading to a faster dynamics), we saw that the resulting closed loop input would have been really discontinuous while the temperature convergence wasn't that much faster.

$$\begin{matrix} 0.5+0.2i & 0.5-0.2i & 0.4 \\ 0.6 & 0.5 & 0.55 \end{matrix} \tag{37}$$

At each iteration is needed to compute the steady state set point of the system with disturbance estimation. This happen because taking into account the disturbance in the dynamics, a different state evolution would be achieved. However, we are still interested in tracking the original set point. The matrix equation 38 provides for any estimated disturbance the new time-variant steady state. H and r have been computed as C_{ref} and b_{ref} in `compute_controller_base_parameters` as an identity matrix and the original temperature set point respectively.

$$\begin{bmatrix} A_I & B \\ HC & \emptyset_{3\times 3} \end{bmatrix} \begin{bmatrix} T_{sp} \\ p_{sp} \end{bmatrix} = \begin{pmatrix} -B_d\hat{d} \\ r - HC_d\hat{d} \end{pmatrix} \tag{38}$$

23. Implementing this MPC controller with the delta formulation, some modifications must be made in the standard offset-free formula. First of all, we need to estimate the temperatures, the disturbance and the steady state at each simulation step and provide them to the `Yalmip` optimizer as inputs. The MPC works on $\hat{T}$ and $\hat{d}$, using the time-varying T_{sp} and p_{sp} to adapt the constraint conditions (since we are using the delta formulation, we are providing only the deviation from a set-point that is different from the original one). We can verify that our problem specifics would lead to a time-varying input set point, but the temperature one would be always constant and equal to the original value.

    Then, it is needed to include the disturbance dynamics as in equation 27 into the MPC constraints. However, since the disturbance is constant and the delta formulation works already on the new steady state based on the estimated disturbance, adding in the dynamics constraint $+B_d d_k$ would result in an overbalancing, since the disturbance has been already considered in the steady state computation. This would not held for a random disturbance, but our observer does not aim to estimate it. We have however left the constraint in the implementation for sake of completeness, but they does not affect the simulation since we multiplied them by a *0* factor.

We can observe from Figure 20 and Figure 21 how the implementation is able to achieve its purpose, even though the system speed in reduced (particularly for T_{VC} and T_{F2}) and the input is not extremely smooth. This happens because the additional observer dynamics adds a delay and an estimation error.

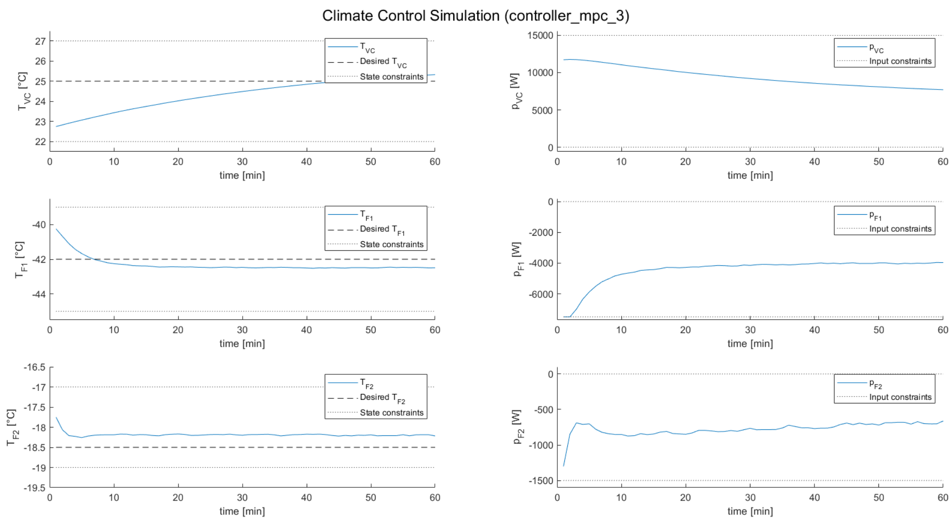

Figure 20: MPC without disturbance estimation, with $T(0) = T_{init}^{(1)}$

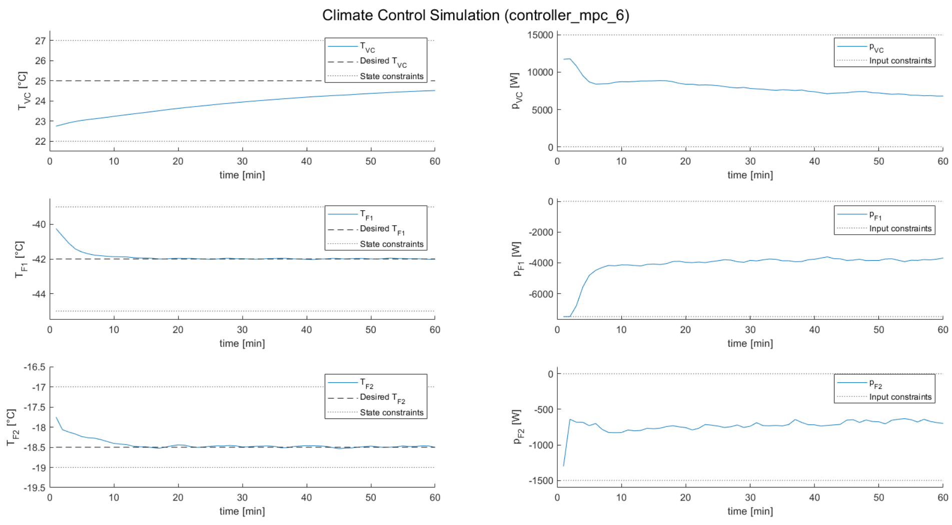

Figure 21: MPC with disturbance estimation, with $T(0) = T_{init}^{(1)}$

VII FORCES Pro

24. By properly implementing `FORCES Pro`, we achieve the same closed loop simulation (Figure 22, Figure 23). However, the average execution time for `controller_mpc_1_forces` is much lower:

$$\begin{aligned} T_{Yalmip} &= 1.577\,s \\ T_{Forces} &= 0.230\,s \\ Overspeed &= 6.778 \end{aligned} \tag{39}$$

We have obtained the final time by averaging six tests. The performance of the solvers are affected by the device used and the work condition, but anyway it can clearly be observed how `Yalmip` is outperformed by `FORCES Pro`.

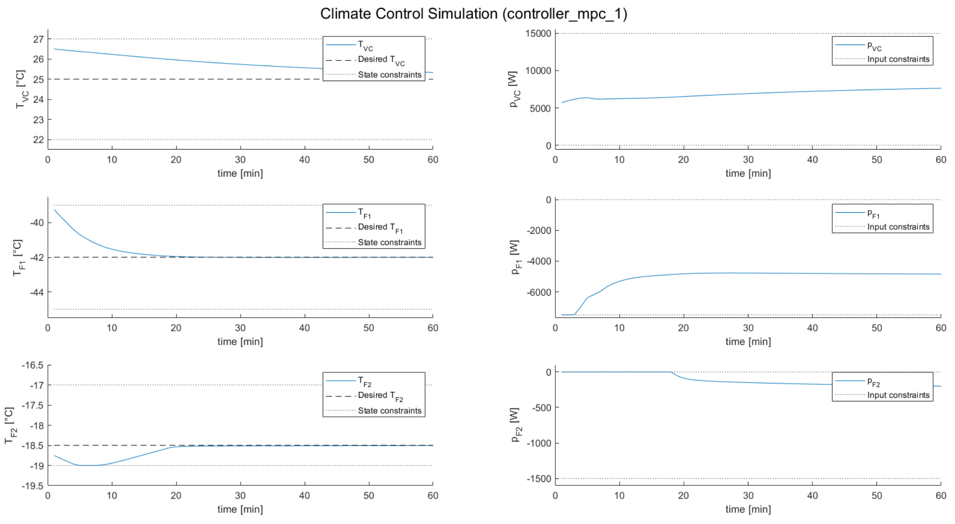

Figure 22: Yalmip MPC, with $T(0) = T_{init}^{(2)}$

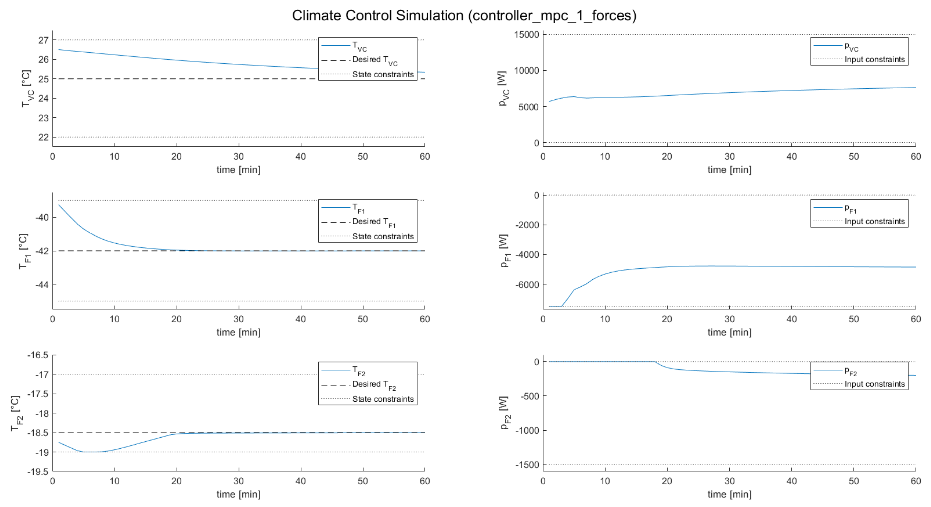

Figure 23: Forces MPC, with $T(0) = T_{init}^{(2)}$