Non-linear Reference Tracking via Model Predictive Control and Extended Kalman Filter

Modelling and Control of Cyber-Physical Systems II

University of Trieste (UniTS)

Marco Tallone

January 2025

Abstract

This report presents the implementation of a Model Predictive Control (MPC) algorithm and an Extended Kalman Filter (EKF) to perform reference tracking of non-linear dynamical systems. The implemented algorithms have been tested and assessed on different non-linear systems and multiple trajectories. Results confirm the effectiveness of the proposed MPC approach in tracking the desired trajectories, even in the presence of noise and disturbances through the use of the EKF.

1 Introduction

Model Predictive Control (MPC) is a control strategy that has been widely adopted for the control of dynamical systems. The MPC approach is based on the solution of an optimization problem over a finite horizon, which allows for the consideration of constraints on the system states and inputs. When used for reference tracking, the MPC algorithm computes the optimal control input by minimizing a cost function that penalizes the deviation of the system states from the desired trajectory.
However, real-world systems are often affected by noise and disturbances, which can lead to undesired results with the adoption of an MPC controller. Moreover, whenever the complete state of a system cannot be fully measured, the use of state estimators is required to infer the unmeasured components and apply the MPC algorithm. To address this issues, the Extended Kalman Filter (EKF) can be used to estimate the states of a non-linear system by fusing the available measurements with the system dynamics and incorporating information about the process and measurement noise.
This report presents the implementation of a Model Predictive Control algorithm for non-linear dynamical systems based on successive linearization and discretization of the dynamics around operating points, as well as the development of an Extended Kalman Filter to estimate the states of the system in the presence of noise.
The objectives of this project are both to study the effectiveness of the non-linear MPC algorithm applied for tracking reference trajectories under different circumstances and also to reproduce and compare the obtained results with the ones presented by *Kunz, Huck and Summers* [2] in their work on the tracking of a helicopter model.
The following report is structured as follows. Section 2 introduces the non-linear models used for the simulations with their dynamics, while Section 3 explains how feasible trajectories can be generated for such models. Section 4 then describes the linear time varying (LTV) approximation of the models and how such approximation is used in the MPC algorithm. In Section 5 the addition of the Extended Kalman Filter is instead presented. Finally, Section 6 presents the results of the simulations followed by some final consideration in the last section.
From a technical point of view all the algorithms and the models for the dynamical systems have been developed in `MATLAB`, version `R2024b` [1]. The code has been structured in a modular way to allow for easy testing and comparison of different algorithms and models. However, with the aim of maintaining a clear and concise report of the work, the following sections will mostly present the results of this study and further concepts from a theoretical point of view, while implementation details and practical usage instructions will be available on the author's GitHub repository [4].

2 Non-linear Models

In order to test and asses the performance of the implemented MPC and EKF algorithms, two systems of different complexity have been considered. In particular, these are:

- A **Unicycle model**
- A **Helicopter model**

Both systems are mathematically described by non-linear dynamics.

2.1 Unicycle Model

The first system is a relatively simple unicycle model described by the non-linear dynamics in equations (2.1). This consists of a rigid body moving on a plane with two parallel wheels of radius r separated by a distance L.

$$\begin{cases} \dot{x} = \frac{r}{2}(\omega_1 + \omega_2)\cos(\theta) \\ \dot{y} = \frac{r}{2}(\omega_1 + \omega_2)\sin(\theta) \\ \dot{\theta} = \frac{r}{L}(\omega_1 - \omega_2) \end{cases} \tag{2.1}$$

The state of the system is hence defined by the Cartesian coordinates x and y of the center of mass of the unicycle and the heading angle θ. The inputs are instead the angular velocities of the two wheels ω_1 and ω_2.

$$\mathbf{x}(t) = \begin{bmatrix} x(t) & y(t) & \theta(t) \end{bmatrix}^T \quad \text{and} \quad \mathbf{u}(t) = \begin{bmatrix} \omega_1(t) & \omega_2(t) \end{bmatrix}^T$$

2.2 Helicopter Model

The second system represents instead a miniature helicopter model taken from the paper by *Kunz, Huck and Summers* [2]. As explained by the authors, the dynamics can be described by the Newton-Euler laws of mechanics for single rigid bodies. However, a simplification has been made in favor of an easier implementation of the MPC algorithm for this model. In particular, introducing the assumption that the pitch and roll inputs directly act on the translational accelerations, it's possible to omit the pitch and roll angular states of the helicopter body. The result is the non-linear dynamics in equations (2.2).

$$\begin{cases} \dot{x}_I = cos(\psi)\dot{x}_B - sin(\psi)\dot{y}_B \\ \dot{y}_I = sin(\psi)\dot{x}_B + cos(\psi)\dot{y}_B \\ \dot{z}_I = \dot{z}_B \\ \ddot{x}_B = b_x u_x + k_x \dot{x}_B + \dot{\psi}\dot{y}_B \\ \ddot{y}_B = b_y u_y + k_y \dot{y}_B - \dot{\psi}\dot{x}_B \\ \dot{z}_B = b_z u_z - g \\ \dot{\psi} = \dot{\psi} \\ \ddot{\psi} = b_\psi u_\psi + k_\psi \dot{\psi} \end{cases} \tag{2.2}$$

where x_I, y_I, z_I denote the position of the helicopter in the inertial frame, $\dot{x}_B$, $\dot{y}_B$, $\dot{z}_B$ the velocities of the helicopter in the body frame, ψ the yaw heading angle and $\dot{\psi}$ its rotational velocity in yaw. The control inputs for pitch, roll, thrust and yaw are instead denoted by u_x, u_y, u_z, u_ψ.
Due to implementation necessities, the only difference with respect to the original paper is the absence of the augmented integral states which were used to reduce steady-state error. The rest of the setup is consistent with the original work.
In this case we hence obtained a model having 8 state components and 4 control inputs formalized as follows.

$$\mathbf{x}(t) = \begin{bmatrix} x_I(t) & y_I(t) & z_I(t) & \dot{x}_B(t) & \dot{y}_B(t) & \dot{z}_B(t) & \psi(t) & \dot{\psi}(t) \end{bmatrix}^T$$

and

$$\mathbf{u}(t) = \begin{bmatrix} u_x(t) & u_y(t) & u_z(t) & u_\psi(t) \end{bmatrix}^T$$

3 Trajectory Generation

Prior to the implementation of the MPC algorithm for reference tracking, it's compulsory to properly define the reference trajectories that the system should follow. In particular, any desired trajectory should be feasible for the given system. Formally, given a continuous-time reference trajectory $\begin{bmatrix}\mathbf{x}_{\text{ref}}(t) & \mathbf{u}_{\text{ref}}(t)\end{bmatrix}^T$, in order for it to be feasible, it must satisfy the differential equations of the system dynamics within the given constraints:

$$\begin{aligned}&\dot{\mathbf{x}}_{\text{ref}}(t) = \mathbf{f}(\mathbf{x}_{\text{ref}}(t), \mathbf{u}_{\text{ref}}(t))\\ &\mathbf{x}_{\text{ref}} \in \mathcal{X}, \quad \mathbf{u}_{\text{ref}} \in \mathcal{U}\end{aligned} \tag{3.1}$$

where $\mathcal{X}$ and $\mathcal{U}$ are the sets of feasible states and inputs for the system while the function $\mathbf{f}(\cdot,\cdot) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ describes the non-linear state dynamics.
Th approaches considered in this study for the feasible trajectory generation task rely on the definition of *differential flatness* introduced by *Murray* [3]. Formally, a non-linear system is *differentially flat* if there exists a function $\Gamma(\cdot)$ such that

$$\mathbf{z} = \Gamma(\mathbf{x}, \mathbf{u}, \dot{\mathbf{u}}, \dots, \mathbf{u}^{(p)})$$

where $\mathbf{z}$ are the so-called *flat outputs*. For a differentially flat system, feasible trajectories can be written as functions of the flat outputs $\mathbf{z}$ and their derivatives. Hence, the general idea is to find a mapping

$$\begin{aligned}\mathbf{x} &= \mathbf{x}(\mathbf{z}, \dot{\mathbf{z}}, \dots, \mathbf{z}^{(q)})\\ \mathbf{u} &= \mathbf{u}(\mathbf{z}, \dot{\mathbf{z}}, \dots, \mathbf{z}^{(q)})\end{aligned}$$

between the original states and the flat outputs such that all states and inputs can be determined from these outputs without integration. Then, a feasible trajectory can simply be obtained by only providing the reference values for the flat outputs (and possibly their derivatives), since the remaining reference states and associated inputs can be recovered from the mapping.
It's easy to show that, for the unicycle model described by equation (2.1), the Cartesian coordinates $z_1 = x$ and $z_2 = y$ are flat outputs. The remaining heading angle state[1] and the control inputs can in fact be obtained from these two outputs and their first order derivatives as shown in equations (3.2).

$$\theta = \arctan\left(\frac{\dot{z_2}}{\dot{z_1}}\right), \quad \omega_1 = \frac{2\sqrt{\dot{z_1}^2 + \dot{z_2}^2} + L\dot{\theta}}{2r}, \quad \omega_2 = \frac{2\sqrt{\dot{z_1}^2 + \dot{z_2}^2} - L\dot{\theta}}{2r} \tag{3.2}$$

For what concerns the helicopter model instead, a set of flat outputs is given by the state components

$$z_1 = x_I, \quad z_2 = y_I, \quad z_3 = z_I, \quad z_4 = \psi$$

which allow to obtain the remaining states and inputs according to equations (3.3).

$$\begin{aligned}\dot{x}_B &= \cos(z_4)\dot{z}_1 + \sin(z_4)\dot{z}_2\\ \dot{y}_B &= -\sin(z_4)\dot{z}_1 + \cos(z_4)\dot{z}_2\\ \dot{z}_B &= \dot{z}_3\\ \dot{\psi} &= \dot{z}_4\\ u_x &= \frac{1}{b_x}\left(cos(z_4)\left[\ddot{z}_1 - k_x\dot{z}_1\right] + sin(z_4)\left[\ddot{z}_2 - k_x\dot{z}_2\right]\right)\\ u_y &= \frac{1}{b_y}\left(cos(z_4)\left[\ddot{z}_2 - k_y\dot{z}_2\right] + sin(z_4)\left[-\ddot{z}_1 + k_y\dot{z}_1\right]\right)\\ u_z &= \frac{1}{b_z}\left(\ddot{z}_3 + g\right)\\ u_\psi &= \frac{1}{b_\psi}\left(\ddot{z}_4 - k_\psi\dot{z}_4\right)\end{aligned} \tag{3.3}$$

After having identified the flat outputs and the required mappings to reconstruct state and inputs components, the generation of the feasible trajectory can then be conducted in two ways.

[1] In the implemented models, the $atan2(\cdot,\cdot)$ function has been used instead of the normal arctangent to obtain the correct heading angle in the range $[-\pi, \pi]$.

The first approach, which is the one used in this study, consists in providing an analytical expression $\mathbf{z}(t)$ for the evolution of the flat outputs over time and then sampling the resulting trajectory at discrete time steps $\{t_k\}_{k=1}^{N_{guide}}$ to obtain a finite set of N_{guide} reference points $\{(\mathbf{x}_{\text{ref},k}, \mathbf{u}_{\text{ref},k})\}_{k=1}^{N_{guide}}$. In particular, two trajectories have been generated with this technique:

- a **circular** trajectory of radius $r_0 = 0.5$ whose flat outputs are defined respectively by the parametric equations:

$$\begin{cases} z_1(t) = r_0 \cos(t) \\ z_2(t) = r_0 \sin(t) \end{cases}$$

for the unicycle model

$$\begin{cases} z_1(t) = r_0 \cos(t) \\ z_2(t) = r_0 \sin(t) \\ z_3(t) = 0 \\ z_4(t) = \text{atan2}\left(\dot{z}_2(t), \dot{z}_1(t)\right) \end{cases}$$

for the helicopter model

- a **leminscate** (or "*figure-eight*") trajectory with parameter $a = 1$ described by:

$$\begin{cases} z_1(t) = \frac{a\sqrt{2}\cos(t)}{\sin(t)^2+1} \\ z_2(t) = \frac{a\sqrt{2}\cos(t)\sin(t)}{\sin(t)^2+1} \end{cases}$$

for the unicycle model

$$\begin{cases} z_1(t) = \frac{a\sqrt{2}\cos(t)}{\sin(t)^2+1} \\ z_2(t) = \frac{a\sqrt{2}\cos(t)\sin(t)}{\sin(t)^2+1} \\ z_3(t) = 0 \\ z_4(t) = \text{atan2}\left(\dot{z}_2(t), \dot{z}_1(t)\right) \end{cases}$$

for the helicopter model

In the conducted experiments, the models performed a full revolution[2] around the respective trajectories in a determined amount of time. Specifically, due to the discretization with sampling time T_s later introduced in Section 4, the total time to complete the trajectory has been set to $T_{end} = N_{guide} \cdot T_s$ for simplicity. In this way, at each iteration, the discretized system has one time-step to reach the next reference point. Moreover, notice that the discretization assumes a constant input applied to the system at each time-step. Hence, a possibility would be to just sample the reference inputs at the beginning of each time interval $\mathbf{u}_{\text{ref},k} = \mathbf{u}_{\text{ref}}(t_k)$. However, due to the continuous-time nature of the inputs obtained with this approach, a more realistic approximation of the system's behavior can be obtained by averaging the values of the inputs over each time interval:

$$\mathbf{u}_{\text{ref},k} = \frac{1}{N_{samples}} \sum_{i=1}^{N_{samples}} \mathbf{u}_{\text{ref}}\left(t_k + \frac{i \cdot T_s}{N_{samples}}\right)$$

where $N_{samples}$ is the number of sub-samples taken in each time interval to compute the average, which depends on the specific trajectory and model considered.
When an analytical expression for the flat outputs is not available, the approach proposed by *Murray* [3] can be used to generate feasible trajectories. In this case the values of the flat outputs (and possibly their derivatives up to a determined order) for a minimal[3] set of representative points of the desired trajectory are needed. Then, between any two given points, the flat outputs can be parametrized as a linear combination of a finite set of basis functions. The coefficients of this linear combination are determined by simply solving a linear system of equations involving the known values of the flat outputs and their derivatives at the endpoints of the interval. This results in a parametrized analytical expression for the flat outputs between the given points, which can then be resampled to obtain a higher number of reference points.
Therefore, this approach presents the huge advantage of being able to generate feasible and smooth point-to-point trajectories for any arbitrary set of initial reference points.
In practice, however, to generate realistic inputs for most systems, high order derivatives of the basis functions involved in the parametrization are often required, especially when the desired trajectories present a high number of sharp turns or sudden changes in direction. This turns out to be unpractical since the known values for the flat outputs derivatives of the same order must be provided at least for the initial set of points. Hence, although implemented, this method has not been used in this study, but further information can be found in Appendix A1 or in the original work by *Murray* [3].

[2] However, the implementation easily allows to perform more than one lap around the given shapes.
[3] At least 2 points.

4 MPC Problem Formulation

The Model Predictive Control (MPC) algorithm is based on the solution of an optimization problem over a finite horizon of length N. In general, the formulation of such optimization problem requires a discrete-time representation of the model dynamics. Moreover, in order to cast the MPC problem as a Quadratic Program (QP) one and hence solve it efficiently, the model must also be linear. The models introduced in Section 2 are not only defined in continuous-time, but also highly non-linear. To overcome this issues, a Linear Time Varying (LTV) approximation of the dynamics is here presented. Then, two formulations of the MPC algorithm as a QP problem are introduced.

4.1 Linear Time Varying (LTV) Model

The LTV model is obtained by successive linearization and discretization of the non-linear dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ around a set of nominal state and input trajectories sampled at a fixed regular time intervals T_s:

$$\begin{aligned} \bar{\mathbf{x}}_k &= \bar{\mathbf{x}}(kT_s), \quad k = k_0, \dots, k_0 + N + 1 \\ \bar{\mathbf{u}}_k &= \bar{\mathbf{u}}(kT_s), \quad k = k_0, \dots, k_0 + N \end{aligned}$$

Hence, the system is first linearized around the nominal states and inputs as follows:

$$\begin{aligned} \delta\dot{\mathbf{x}}(t) &= \mathbf{A}_k \delta\mathbf{x}(t) + \mathbf{B}_k \delta\mathbf{u}(t) \\ \mathbf{A}_k = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k}, &\quad \mathbf{B}_k = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right|_{\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k} \\ \delta\mathbf{x}(t) = \mathbf{x}(t) - \bar{\mathbf{x}}_k, &\quad \delta\mathbf{u}(t) = \mathbf{u}(t) - \bar{\mathbf{u}}_k \end{aligned} \tag{4.1}$$

with $kT_s \leq t < (k+1)T_s$ for $k = k_0, \dots, k_0 + N$. Then, the obtained linear system is discretized using first order forward Euler method:

$$\begin{aligned} \delta\mathbf{x}_{k+1} &= \mathbf{A}_{d,k} \delta\mathbf{x}_k + \mathbf{B}_{d,k} \delta\mathbf{u}_k \\ \mathbf{A}_{d,k} = \mathbb{I} + T_s \mathbf{A}_k, &\quad \mathbf{B}_{d,k} = T_s \mathbf{B}_k \\ \delta\mathbf{x}_k = \mathbf{x}_k - \bar{\mathbf{x}}_k, &\quad \delta\mathbf{u}_k = \mathbf{u}_k - \bar{\mathbf{u}}_k \end{aligned} \tag{4.2}$$

This can eventually be rewritten as:

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}_{d,k} \mathbf{x}_k + \mathbf{B}_{d,k} \mathbf{u}_k + \mathbf{d}_k \\ \mathbf{d}_k &= \bar{\mathbf{x}}_{k+1} - \mathbf{A}_{d,k} \bar{\mathbf{x}}_k - \mathbf{B}_{d,k} \bar{\mathbf{u}}_k \end{aligned} \tag{4.3}$$

Using this LTV model, the MPC controller is able to solve an optimization problem for each sampling time $t \in \{t_k\}_{k=1}^{N_{guide}}$ in the prediction horizon N. The result of the optimization will be the optimal input sequence $\mathbf{U}^*_{t \to t+N|t} = [\mathbf{u}^*_{t|t}, \dots, \mathbf{u}^*_{t+N|t}]^T$ minimizing the cost function $\mathbf{J}$ (eq. 4.4), from which only the first input $\mathbf{u}^*_{t|t}$ will actually be applied. Hence, in successive applications of the algorithm, a nominal input sequence for future linearizations can be obtained as the remaining part of the optimal input sequence[4] $\bar{\mathbf{U}}_{t+1 \to t+N+1|t+1} = [\mathbf{u}^*_{t+1|t}, \dots, \mathbf{u}^*_{t+N|t}, \mathbf{u}^*_{t+N|t}]^T$ while the nominal states can be obtained by applying such inputs to the non-linear dynamics. For the first iteration of the MPC algorithm, since no pre-computed optimal sequence is yet available, the first reference input sequence can be used even if it will result in an inaccurate linearization.

4.2 MPC Formulations

The Model Predictive Control problem with the LTV model approximation of the non-linear dynamics is formulated as the optimization problem 4.4, which is solved at every time instance t.

$$\begin{aligned} \min_{\mathbf{U}_{t \to t+N|t}} \quad & \mathbf{J} = \sum_{k=t}^{t+N} \left(\Delta\mathbf{x}_k^T \mathbf{Q} \Delta\mathbf{x}_k + \Delta\mathbf{u}_k^T \mathbf{R} \Delta\mathbf{u}_k \right) \\ \text{s.t.} \quad & \mathbf{x}_{k+1|t} = \mathbf{A}_{d,k|t} \mathbf{x}_{k|t} + \mathbf{B}_{d,k|t} \mathbf{u}_{k|t} + \mathbf{d}_{k|t} \\ & \mathbf{d}_{k|t} = \bar{\mathbf{x}}_{k+1|t} - \mathbf{A}_{d,k|t} \bar{\mathbf{x}}_{k|t} - \mathbf{B}_{d,k|t} \bar{\mathbf{u}}_{k|t} \\ & \mathbf{x}_{k|t} \in \mathcal{X}, \quad k = t, \dots, t+N \\ & \mathbf{u}_{k|t} \in \mathcal{U}, \quad k = t, \dots, t+N+1 \end{aligned} \tag{4.4}$$

[4] With the last input duplicated as N nominal inputs are needed.

Where $\mathbf{U}_{t\to t+N|t} = \begin{bmatrix}\mathbf{u}_{t|t} & \dots & \mathbf{u}_{t+N|t}\end{bmatrix}^T$ is the input sequence to be optimized, while $\Delta\mathbf{x}_k = \mathbf{x}_{k|t} - \mathbf{x}_{\text{ref},k|t}$ and $\Delta\mathbf{u}_k = \mathbf{u}_{k|t} - \mathbf{u}_{\text{ref},k|t}$ are the differences between the predicted states and the reference trajectory. The matrices $\mathbf{Q} \geq 0$ and $\mathbf{R} > 0$ are the weighting matrices for the state and input errors, respectively.
This general expression of the MPC problem can be then cast into a QP problems to be solved efficiently using solvers such as `MATLAB` `quadprog` by introducing two different formulations: the *dense* (or *explicit*) formulation and the *sparse* (or *implicit*) formulation. The *dense* formulation can be obtained by introducing a vectorized notation for the state and input sequences and expressing state and inputs constraints as vectorized inequalities. The complete derivation of the *dense* formulation is delayed to Appendix A2, while here only the final form of the resulting QP problem is reported:

$$\begin{aligned}\min_{\mathbf{U}_{k|t}} \quad & \mathbf{U}_{k|t}^T \left(\mathcal{B}^T\mathcal{Q}\mathcal{B} + \mathcal{R}\right)\mathbf{U}_{k|t} + 2\left(\mathbf{x}_{k|t}^T\mathcal{A}^T\mathcal{Q}\mathcal{B} + \mathcal{D}^T\mathcal{Q}\mathcal{B} - \bar{\mathbf{X}}_{k|t}^T\mathcal{Q}\mathcal{B} - \bar{\mathbf{U}}_{k|t}^T\mathcal{R}\right)\mathbf{U}_{k|t} \\ \text{s.t.} \quad & \begin{bmatrix}\mathcal{E}_\mathbf{u} \\ \mathcal{E}_\mathbf{x}\mathcal{B}\end{bmatrix}\mathbf{U}_{k|t} \leq \begin{bmatrix}\mathcal{F}_\mathbf{u} \\ \mathcal{F}_\mathbf{x} - \mathcal{E}_\mathbf{x}\mathcal{A}\mathbf{x}_{k|t} - \mathcal{E}_\mathbf{x}\mathcal{D}\end{bmatrix}\end{aligned} \tag{4.5}$$

Alternatively, a *sparse* formulation can be obtained taking advantage of a similar vectorized notation. This second formulation presents as an easier implementation and it's more efficient for solvers that can exploit the sparsity of the associated matrices. However, it relies on the introduction of an augmented state vector $\mathbf{Z}_k = \begin{bmatrix}\mathbf{X}_k^T & \mathbf{U}_k^T\end{bmatrix}^T$ and hence the optimization problem generally involves a higher number of decision variables. Once again, while the full derivation of this formulation is reported in Appendix A3, the final resulting QP problem is the following:

$$\begin{aligned}\min_{\mathbf{Z}_{k|t}} \quad & \mathbf{Z}_{k|t}^T\mathbf{V}\mathbf{Z}_{k|t} + 2\left(-\bar{\mathbf{Z}}_{k|t}^T\mathbf{V}\right)\mathbf{Z}_{k|t} \\ \text{s.t.} \quad & \begin{bmatrix}\mathcal{E}_\mathbf{x} & \mathbf{0} \\ \mathbf{0} & \mathcal{E}_\mathbf{u}\end{bmatrix}\mathbf{Z}_{k|t} \leq \begin{bmatrix}\mathcal{F}_\mathbf{u} \\ \mathcal{F}_\mathbf{x}\end{bmatrix} \\ & \begin{bmatrix}\mathbb{I} - \mathcal{A} & -\mathcal{B}\end{bmatrix}\mathbf{Z}_{k|t} = \mathcal{D}\delta\mathbf{x}_{k|t} + \begin{bmatrix}\mathbb{I} - \mathcal{A} & -\mathcal{B}\end{bmatrix}\bar{\mathbf{Z}}_{k|t}\end{aligned} \tag{4.6}$$

Notice that this formulation includes both inequality and equality constraints on the augmented vectorized state and input sequences used in the optimization problem.
The choice between the two formulations depends on the specific application and the necessities of the problem at hand. In this project, both formulation have been implemented in the MPC algorithm and, for the tests conducted in this work, no significant differences in terms of performance and final results have been observed.
The pseudocode in Code 1 schematically summarizes the implemented MPC algorithm by only including the most relevant steps of the process.

Non-Linear MPC Algorithm with LTV Model

```
N ← set_horizon()
x ← x_0                                   // Initial conditions
Ū_1 ← U_ref,1
for k in {1, 2, ..., T_end − N} do        // Main loop
  x̄_1 ← x
  ū_1, ..., ū_N ← Ū_k
  for i in {1, ..., N − 1} do             // Prediction horizon loop
    A_i, B_i ← linearize(x̄_i, ū_i)
    A_d,i, B_d,i ← discretize(A_i, B_i)
    A, B, D ← build_formulation_matrices(A_d,i, B_d,i)
    x̄_i+1 ← simulate(x̄_i, ū_i+1, T_s)
  end
  Q, R ← set_weights()                    // Optimization phase
  E_≤, F_≤, E_=, F_= ←
  set_constraints(Q, R)
  U* ← solve(A, B, D, E_≤, F_≤, E_=, F_=)
  x ← simulate(x, U*_1, T_s)              // Apply first optimal input
  Ū_k+1 ← U*_[2,3,...,N,N]                // Set next nominal inputs
end
```

Code 1: Main steps of the implemented MPC algorithm. Notice that the algorithm runs up to $T_{end} - N$ since the prediction horizon cannot be applied to times beyond this limit.

5 Extended Kalman Filter (EKF)

The previously introduced MPC algorithm relies on the knowledge of the complete state of the system at any given time. However, in many practical applications, the complete state of a system cannot always be fully measured. Moreover, disturbances and noise can affect the measurements, leading to inaccurate predictions and control actions.
To address these issues for the studied non-linear systems, the Extended Kalman Filter (EKF) has been implemented. The EKF is the non-linear version of the Kalman Filter, which implies linearization around an estimate of the current mean and covariance.
To explain the implementation of the EKF in this study, it's assumed that the equivalent discrete-time versions of the state equation and the output transformation of the implemented non-linear systems are affected by noise as follows:

$$\begin{aligned} \dot{\mathbf{x}}_k &= \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + \mathbf{w}_k \\ \mathbf{y}_k &= \mathbf{g}(\mathbf{x}_k) + \mathbf{v}_k \end{aligned} \tag{5.1}$$

Where $\mathbf{w}_k$ and $\mathbf{v}_k$ are process and observation noises, which are both assumed to be zero-mean multivariate Gaussian noises with covariance matrices $\tilde{\mathbf{Q}}_k$ and $\tilde{\mathbf{R}}_k$, respectively.
The function $\mathbf{g}(\cdot)$ is instead the output transformation function which can be used to compute the predicted measurements $\mathbf{y}_k$ from the estimated states. In general, the measurements don't correspond to the full state of the system, which has to be estimated by introducing a gain matrix $\mathbf{K}_k$ that is computed at each time step. In the experiments conducted for this study, a natural choice for the implemented models has been to simulate a measurement of the flat outputs only, since the reconstruction of the full state of the system is then guaranteed by the previously introduced *differential flatness* property.
To derive the Kalman gain $\mathbf{K}_k$, $\mathbf{f}$ and $\mathbf{g}$ cannot be applied to the covariance directly. Instead, a matrix of partial derivatives (the *Jacobian*) is computed and evaluated at the current estimate of the state performing in this way a linearization. Such matrices are then used in the classic Kalman Filter equations to compute the desired gain.
The whole process can be divided in two main phases: the *prediction* (eq. 5.2), where a first estimate of the state is computed as well as its covariance, and the *update* (eq. 5.3), where the information from the measurements is used to correct and refine the state estimate and its covariance.

- **Prediction**:

$$\begin{aligned} \hat{\mathbf{x}}_{k|k-1} &= \mathbf{f}(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_{k-1}) && \text{Predicted state estimate} \\ \mathbf{P}_{k|k-1} &= \mathbf{A}_k \mathbf{P}_{k-1|k-1} \mathbf{A}_k^T + \tilde{\mathbf{Q}}_{k-1} && \text{Predicted covariance estimate} \end{aligned} \tag{5.2}$$

- **Update**:

$$\begin{aligned} \mathbf{K}_k &= \mathbf{P}_{k|k-1} \mathbf{C}_k^T \left(\mathbf{C}_k \mathbf{P}_{k|k-1} \mathbf{C}_k^T + \tilde{\mathbf{R}}_k \right)^{-1} && \text{Kalman gain} \\ \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \left(\mathbf{y}_k - \mathbf{g}(\hat{\mathbf{x}}_{k|k-1}) \right) && \text{Updated state estimate} \\ \mathbf{P}_{k|k} &= \left(\mathbb{I} - \mathbf{K}_k \mathbf{C}_k \right) \mathbf{P}_{k|k-1} && \text{Updated covariance estimate} \end{aligned} \tag{5.3}$$

where $\mathbf{A}_k$ and $\mathbf{C}_k$ are the Jacobians of $\mathbf{f}$ and $\mathbf{g}$ respecively computed as:

$$\mathbf{A}_k = \left. \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k} \qquad \text{and} \qquad \mathbf{C}_k = \left. \frac{\partial \mathbf{g}}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{k|k-1}}$$

The state estimate obtained from the EKF process can also be used internally by the MPC algorithm. In particular, at each iteration of the MPC algorithm, the initial state for the given prediction horizon can be estimated by the EKF from the measurements of the flat outputs only, while future states of the horizon are predicted using the non-linear dynamics and the previously explained MPC process. This strategy allows for the inclusion of both process and measurement noise only in the first step of each MPC iteration and doesn't propagate such uncertainty throughout the remaining prediction horizon. Nevertheless, in the presence of noise, this approach resulted in far better input sequences for reference tracking than the ones obtained by ignoring the disturbances in the MPC algorithm.

6 Results

7 Conclusions

References

[1] The MathWorks Inc. MATLAB version: 24.2.0 (r2024b), 2024. URL: https://www.mathworks.com.

[2] Konstantin Kunz, Stephan M. Huck, and Tyler H. Summers. Fast model predictive control of miniature helicopters. In *2013 European Control Conference (ECC)*, pages 1377–1382, 2013. doi:10.23919/ECC.2013.6669699.

[3] R.M. Murray. *Optimization Based Control*. California Institute of Technology, 2023. URL: http://www.cds.caltech.edu/~murray/books/AM08/pdf/obc-complete_12Mar2023.pdf.

[4] Marco Tallone. Non-Linear Tracking via MPC and EKF. URL: https://github.com/marcotallone/mpc-tracking.

A Appendix

A1 Murray Trajectory Generation Method

Consider the non-linear system described by the state equation $\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$ and having a defined set of flat outputs identified by $\mathbf{z}(t)$. Given a set of N_{start} reference values for the flat outputs and their derivatives up to order q

$$\{\mathbf{z}(T_1), \dot{\mathbf{z}}(T_1), \dots, \mathbf{z}^{(q)}(T_1), \dots, \mathbf{z}(T_{N_{\text{start}}}), \dot{\mathbf{z}}(T_{N_{\text{start}}}), \dots, \mathbf{z}^{(q)}(T_{N_{\text{start}}})\},$$

it's possible to generate a feasible trajectory for the system consisting in a possibly larger number $N > N_{\text{start}}$ of reference values by first building feasible parametrized trajectories between each pair of consecutive reference values and then sampling at the desired rate. To explain this method, first introduced by *Murray* [3], consider for simplicity the first two reference values (the process is similarly repeated for subsequent pairs). Without loss of generality, assume that

$$T_1 = 0 \quad \text{and} \quad T_2 = T$$

Hence, the vector $\bar{\mathbf{z}}$ of known values for the flat outputs and their derivatives at the endpoints of the considered time interval assumes the form

$$\bar{\mathbf{z}} = \begin{bmatrix} \mathbf{z}(0) & \dot{\mathbf{z}}(0) & \dots & \mathbf{z}^{(q)}(0) & \mathbf{z}(T) & \dot{\mathbf{z}}(T) & \dots & \mathbf{z}^{(q)}(T) \end{bmatrix}^T$$

Since $\mathbf{z}(t)$ must satisfy these boundary conditions, an analytical expression for the flat outputs can be found by parametrizing them as a linear combination of a set of N_{basis} smooth basis functions $\mathbf{b}_i(t)$:

$$\mathbf{z}(t) = \sum_{i=1}^{N_{\text{basis}}} \alpha_i \mathbf{b}_i(t)$$

where the coefficients α_i are determined by the linear system of equations obtained by imposing the boundary conditions on the basis functions and the corresponding derivatives up to order q:

$$\begin{bmatrix} \mathbf{b}_1(0) & \mathbf{b}_2(0) & \dots & \mathbf{b}_{N_{\text{basis}}}(0) \\ \dot{\mathbf{b}}_1(0) & \dot{\mathbf{b}}_2(0) & \dots & \dot{\mathbf{b}}_{N_{\text{basis}}}(0) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{b}_1^{(q)}(0) & \mathbf{b}_2^{(q)}(0) & \dots & \mathbf{b}_{N_{\text{basis}}}^{(q)}(0) \\ \mathbf{b}_1(T) & \mathbf{b}_2(T) & \dots & \mathbf{b}_{N_{\text{basis}}}(T) \\ \dot{\mathbf{b}}_1(T) & \dot{\mathbf{b}}_2(T) & \dots & \dot{\mathbf{b}}_{N_{\text{basis}}}(T) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{b}_1^{(q)}(T) & \mathbf{b}_2^{(q)}(T) & \dots & \mathbf{b}_{N_{\text{basis}}}^{(q)}(T) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{N_{\text{basis}}} \end{bmatrix} = \begin{bmatrix} \mathbf{z}(0) \\ \dot{\mathbf{z}}(0) \\ \vdots \\ \mathbf{z}^{(q)}(0) \\ \mathbf{z}(T) \\ \dot{\mathbf{z}}(T) \\ \vdots \\ \mathbf{z}^{(q)}(T) \end{bmatrix}$$

As explained by *Murray* [3], we obtain a linear system of the form $M\boldsymbol{\alpha} = \bar{\mathbf{z}}$. Hence, assuming that M is full column rank, we can obtain a (possibly non-unique) solution for $\boldsymbol{\alpha}$ that satisfies the trajectory generation requirements. In other words, the result is an analytical expression for the flat outputs $\mathbf{z}(t)$ (and their derivatives up to order q) between the given endpoints.
This process can then be repeated for each pair of consecutive reference values in the initial set to obtain multiple smooth point-to-point expression for the flat outputs. Finally, the obtained trajectories can be resampled at the desired time intervals and the corresponding values of the flat outputs can be used to generate feasible trajectories for the system taking advantage of the *differential flatness* property.

A2 Dense Formulation

Given an MPC prediction horizon N, the *dense* formulation can be obtained by first introducing the vectorized notation

$$\begin{aligned}\mathbf{X}_k &= \begin{bmatrix}\mathbf{x}_k^T & \mathbf{x}_{k+1}^T & \dots & \mathbf{x}_{k+N-1}^T\end{bmatrix}^T \\ \mathbf{U}_k &= \begin{bmatrix}\mathbf{u}_k^T & \mathbf{u}_{k+1}^T & \dots & \mathbf{u}_{k+N-1}^T\end{bmatrix}^T\end{aligned}$$

so that the vector of predicted states satisfies the following equality

$$\mathbf{X}_k = \mathcal{A}\mathbf{x}_k + \mathcal{B}\mathbf{U}_k + \mathcal{D} \tag{A.1}$$

where, using the matrices derived from the linear time-varying (LTV) approximation of Section 4, we have:

$$\mathcal{A} = \begin{bmatrix} \mathbb{I} \\ \mathbf{A}_{d,k} \\ \mathbf{A}_{d,k+1}\mathbf{A}_{d,k} \\ \vdots \\ \mathbf{A}_{d,k+N-2}\dots\mathbf{A}_{d,k} \end{bmatrix}, \quad \mathcal{B} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{B}_{d,k} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{A}_{d,k+1}\mathbf{B}_{d,k} & \mathbf{B}_{d,k+1} & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots & \\ \mathbf{A}_{d,k+N-2}\dots\mathbf{A}_{d,k}\mathbf{B}_{d,k} & \dots & \dots & \mathbf{B}_{d,k+N-2} & \mathbf{0} \end{bmatrix},$$

$$\text{and} \quad \mathcal{D} = \begin{bmatrix} \mathbf{0} \\ \mathbf{d}_k \\ \mathbf{A}_{d,k+1}\mathbf{d}_k + \mathbf{d}_{k+1} \\ \mathbf{A}_{d,k+2}\mathbf{A}_{d,k+1}\mathbf{d}_k + \mathbf{A}_{d,k+2}\mathbf{d}_{k+1} + \mathbf{d}_{k+2} \\ \vdots \end{bmatrix}$$

Consequently, by also introducing the large notation for the state $\mathcal{Q} = \text{diag}(\mathbf{Q}, \dots, \mathbf{Q})$ and input $\mathcal{R} = \text{diag}(\mathbf{R}, \dots, \mathbf{R})$ weighting matrices, it is possible to rewrite the cost function $\mathbf{J}$ of the optimization problem 4.4 as a large matrix equation:

$$\begin{aligned}\mathbf{J} &= \Delta\mathbf{X}_k^T\mathcal{Q}\Delta\mathbf{X}_k + \Delta\mathbf{U}_k^T\mathcal{R}\Delta\mathbf{U}_k \\ &= \left(\mathbf{X}_k - \mathbf{X}_{\text{ref},k}\right)^T\mathcal{Q}\left(\mathbf{X}_k - \mathbf{X}_{\text{ref},k}\right) + \left(\mathbf{U}_k - \mathbf{U}_{\text{ref},k}\right)^T\mathcal{R}\left(\mathbf{U}_k - \mathbf{U}_{\text{ref},k}\right) \\ &= \mathbf{U}_k^T\left(\mathcal{B}^T\mathcal{Q}\mathcal{B} + \mathcal{R}\right)\mathbf{U}_k + 2\left(\mathbf{x}_k^T\mathcal{A}^T\mathcal{Q}\mathcal{B} + \mathcal{D}^T\mathcal{Q}\mathcal{B} - \bar{\mathbf{X}}_k^T\mathcal{Q}\mathcal{B} - \bar{\mathbf{U}}_k^T\mathcal{R}\right)\mathbf{U}_k \\ &= \mathbf{U}_k^T\mathbf{H}_k\mathbf{U}_k + 2\mathbf{f}_k^T\mathbf{U}_k\end{aligned}$$

which is a quadratic function of the input sequence $\mathbf{U}_k$ and can hence be efficiently solved with solvers such as `MATLAB`'s `quadprog` function. Moreover, state and input constraints of the form $-\mathbf{x}_{\min} \leq \mathbf{x}_k \leq \mathbf{x}_{\max}$ and $-\mathbf{u}_{\min} \leq \mathbf{u}_k \leq \mathbf{u}_{\max}$ can be rewritten as

$$\boldsymbol{\varepsilon}\mathbf{x}_k \leq \mathbf{f}_{\mathbf{x}} = \begin{bmatrix}\mathbf{x}_{\max} \\ \mathbf{x}_{\min}\end{bmatrix} \quad \text{and} \quad \boldsymbol{\varepsilon}\mathbf{u}_k \leq \mathbf{f}_{\mathbf{u}} = \begin{bmatrix}\mathbf{u}_{\max} \\ \mathbf{u}_{\min}\end{bmatrix} \quad \text{with} \quad \boldsymbol{\varepsilon} = \begin{bmatrix}+1 \\ -1\end{bmatrix}$$

Hence, by stacking the constraints for all time steps as follows:

$$\mathcal{E}_{\mathbf{x}} = \text{diag}(\boldsymbol{\varepsilon}, \dots, \boldsymbol{\varepsilon}), \quad \mathcal{F}_{\mathbf{x}} = \begin{bmatrix}\mathbf{f}_{\mathbf{x}} \\ \vdots \\ \mathbf{f}_{\mathbf{x}}\end{bmatrix}, \quad \mathcal{E}_{\mathbf{u}} = \text{diag}(\boldsymbol{\varepsilon}, \dots, \boldsymbol{\varepsilon}), \quad \mathcal{F}_{\mathbf{u}} = \begin{bmatrix}\mathbf{f}_{\mathbf{u}} \\ \vdots \\ \mathbf{f}_{\mathbf{u}}\end{bmatrix}$$

and using the equality A.1, the constraints can be expressed as the compact inequality

$$\mathcal{E}\mathbf{U}_k \leq \mathcal{F}$$

where

$$\mathcal{E} = \begin{bmatrix}\mathcal{E}_{\mathbf{u}} \\ \mathcal{E}_{\mathbf{x}}\mathcal{B}\end{bmatrix} \quad \text{and} \quad \mathcal{F} = \begin{bmatrix}\mathcal{F}_{\mathbf{u}} \\ \mathcal{F}_{\mathbf{x}} - \mathcal{E}_{\mathbf{x}}(\mathcal{A}\mathbf{x}_k + \mathcal{D})\end{bmatrix}$$

The final QP problem resulting from the *dense* formulation can hence be written as:

$$\begin{aligned}\min_{\mathbf{U}_k} \quad & \mathbf{U}_k^T\mathbf{H}_k\mathbf{U}_k + 2\mathbf{f}_k^T\mathbf{U}_k \\ \text{s.t.} \quad & \mathcal{E}\mathbf{U}_k \leq \mathcal{F}\end{aligned} \tag{A.2}$$

A3 Sparse Formulation

Given an MPC prediction horizon N, the *sparse* formulation can be obtained by first introducing the vectorized notation

$$\delta\mathbf{X}_k = \mathbf{X}_k - \bar{\mathbf{X}}_k = \begin{bmatrix}\delta\mathbf{x}_k^T & \delta\mathbf{x}_{k+1}^T & \dots & \delta\mathbf{x}_{k+N-1}^T\end{bmatrix}^T$$
$$\delta\mathbf{U}_k = \mathbf{U}_k - \bar{\mathbf{U}}_k = \begin{bmatrix}\delta\mathbf{u}_k^T & \delta\mathbf{u}_{k+1}^T & \dots & \delta\mathbf{u}_{k+N-1}^T\end{bmatrix}^T$$

so that the vector of state displacements from the operating points satisfies the equality

$$\delta\mathbf{X}_k = \mathcal{A}\delta\mathbf{X}_k + \mathcal{B}\delta\mathbf{U}_k + \mathcal{D}\delta\mathbf{x}_k \tag{A.3}$$

where, using the matrices derived from the linear time-varying (LTV) approximation of Section 4, we have:

$$\mathcal{A} = \begin{bmatrix}\mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0}\\ \mathbf{A}_{d,k} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{A}_{d,k+1} & \dots & \mathbf{0} & \mathbf{0}\\ \vdots & \vdots & \ddots & \vdots & \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{A}_{d,k+N-1} & \mathbf{0}\end{bmatrix}, \mathcal{B} = \begin{bmatrix}\mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0}\\ \mathbf{B}_{d,k} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{B}_{d,k+1} & \dots & \mathbf{0} & \mathbf{0}\\ \vdots & \vdots & \ddots & \vdots & \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{B}_{d,k+N-1} & \mathbf{0}\end{bmatrix}, \mathcal{D} = \begin{bmatrix}\mathbb{I}\\ \mathbf{0}\\ \vdots\\ \mathbf{0}\end{bmatrix}$$

The large notation matrices for the state $\mathcal{Q} = \text{diag}(\mathbf{Q}, \dots, \mathbf{Q})$ and input $\mathcal{R} = \text{diag}(\mathbf{R}, \dots, \mathbf{R})$ weighting matrices are also introduced and joint in the matrix $\mathbf{V}$ as follows:

$$\mathbf{V} = \begin{bmatrix}\mathcal{Q} & \mathbf{0}\\ \mathbf{0} & \mathcal{R}\end{bmatrix}$$

Consequently, the cost function $\mathbf{J}$ of the optimization problem 4.4 can be rewritten as a quadratic function of the augmented state and input variables:

$$\mathbf{Z}_k = \begin{bmatrix}\mathbf{X}_k\\ \mathbf{U}_k\end{bmatrix} \quad \text{and} \quad \mathbf{Z}_{\text{ref},\mathbf{k}} = \begin{bmatrix}\mathbf{X}_{\text{ref},k}\\ \mathbf{U}_{\text{ref},k}\end{bmatrix}$$

as

$$\begin{aligned}\mathbf{J} &= \Delta\mathbf{X}_k^T\mathcal{Q}\Delta\mathbf{X}_k + \Delta\mathbf{U}_k^T\mathcal{R}\Delta\mathbf{U}_k\\ &= \Delta\mathbf{Z}_k^T\mathbf{V}\Delta\mathbf{Z}_k\\ &= \left(\mathbf{Z}_k - \mathbf{Z}_{\text{ref},k}\right)^T\mathbf{V}\left(\mathbf{Z}_k - \mathbf{Z}_{\text{ref},k}\right)\\ &= \mathbf{Z}_k^T\mathbf{V}\mathbf{Z}_k + 2\left(-\mathbf{Z}_{\text{ref},k}^T\mathbf{V}\right)\mathbf{Z}_k\\ &= \mathbf{Z}_k^T\mathbf{H}_k\mathbf{Z}_k + 2\mathbf{f}_k^T\mathbf{Z}_k\end{aligned}$$

Equivalentrly to the *dense* formulation, state and input constraints are also expressed in the compact matrix inequality

$$\mathcal{E}_\mathbf{Z}\mathbf{Z}_k \leq \mathcal{F}_\mathbf{Z}$$

where

$$\mathcal{E}_\mathbf{Z} = \begin{bmatrix}\mathcal{E}_\mathbf{x} & \mathbf{0}\\ \mathbf{0} & \mathcal{E}_\mathbf{u}\end{bmatrix} \quad \text{and} \quad \mathcal{F}_\mathbf{Z} = \begin{bmatrix}\mathcal{F}_\mathbf{x}\\ \mathcal{F}_\mathbf{u}\end{bmatrix}$$

and $\mathcal{E}_\mathbf{x}$, $\mathcal{E}_\mathbf{u}$, $\mathcal{F}_\mathbf{x}$ and $\mathcal{F}_\mathbf{u}$ assume the same meaning as in Appendix A2. Additionally, equation A.3 implies the following equality constraints on the augmented state and input variables:

$$(\ A.3\ ) \implies \begin{bmatrix}\mathbb{I} - \mathcal{A} & -\mathcal{B}\end{bmatrix}\delta\mathbf{Z}_k = \mathcal{D}\delta\mathbf{x}_k \implies \mathcal{E}_{\text{eq.}}\mathbf{Z}_k = \mathcal{F}_{\text{eq.}}$$

where

$$\mathcal{E}_{\text{eq.}} = \begin{bmatrix}\mathbb{I} - \mathcal{A} & -\mathcal{B}\end{bmatrix} \quad \text{and} \quad \mathcal{F}_{\text{eq.}} = \mathcal{D}\delta\mathbf{x}_k + \begin{bmatrix}\mathbb{I} - \mathcal{A} & -\mathcal{B}\end{bmatrix}\bar{\mathbf{Z}}_k$$

The final QP problem resulting from the *sparse* formulation can hence be written as:

$$\begin{aligned}\min_{\mathbf{Z}_k} \quad & \mathbf{Z}_k^T\mathbf{V}\mathbf{Z}_k + 2\mathbf{f}_k^T\mathbf{Z}_k\\ \text{s.t.} \quad & \mathcal{E}_\mathbf{Z}\mathbf{Z}_k \leq \mathcal{F}_\mathbf{Z}\\ & \mathcal{E}_{\text{eq.}}\mathbf{Z}_k = \mathcal{F}_{\text{eq.}}\end{aligned} \tag{A.4}$$