Model Predictive Control: Mini-Project

Ziyi Xu 406684 Zimeng Wang 416727 Xinran Wang 416485
ziyi.xu@epfl.ch zimeng.wang@epfl.ch xinran.wang@epfl.ch

January 11, 2026

System Dynamics

Linearization

2.1 Deliverable 2.1 – Why the linearized rocket model decomposes into four independent subsystems

Intuitive physical justification for subsystem separation. Around the trim point, the thrust produced by the propeller pair acts essentially along the body z_b axis, and the control inputs affect *either* the thrust *direction*, the thrust *magnitude*, or the *differential moment*, in physically distinct ways. This is why the linearized dynamics can be separated into independent subsystems.

- **Lateral motion is governed by the tilt of the thrust vector, where each body-frame tilt axis corresponds to one inertial lateral axis.** The unit thrust direction in the body frame is given by ${}_be_F(\delta_1, \delta_2)$, resulting in the force ${}_bF = F \cdot {}_be_F$. Under the small-angle assumption ($\delta_1, \delta_2 \ll 1$), the deflection δ_2 primarily induces a thrust component along x_b, whereas δ_1 induces a component along y_b. Near the hover equilibrium, the rotation matrix $T_{wb}(\phi)$ is approximately an identity map. Consequently, in the inertial dynamics $\dot{v} = T_{wb} \cdot {}_bF/m - g$, these orthogonal tilt directions produce independent accelerations in the inertial x and y axes:

$$\text{sys}_x : \delta_2 \rightarrow \{\omega_y, \beta, v_x, x\}, \qquad \text{sys}_y : \delta_1 \rightarrow \{\omega_x, \alpha, v_y, y\}.$$

- **Vertical motion is dominated by the thrust magnitude rather than its orientation.** The input P_{avg} modulates the total thrust magnitude $F(P_{\text{avg}})$. Since the thrust vector remains approximately aligned with the body z-axis (z_b) for small gimbal deflections, changes in P_{avg} predominantly affect the net force along the vertical axis against gravity. This yields the decoupled 2-state subsystem:

$$\text{sys}_z : P_{\text{avg}} \rightarrow \{v_z, z\}.$$

  Intuitive physical interpretation: Modulating the average throttle directly varies the force opposing gravity, thereby isolating the dynamics of the vertical states $\{v_z, z\}$ from the lateral orientation.

- **Roll is mainly created by the differential moment, not by the average thrust or gimbal tilts.** The input P_{diff} produces a differential moment $M_\Delta(P_{diff})$ and contributes to the body moment ${}_bM$ which induces changes in ω according to $\dot{\omega} = J^{-1}(-\omega \times J\omega + {}_bM)$. Near the trim point the cross-coupling term $-\omega \times J\omega$ is small, hence changing P_{diff} primarily induces a rotation about the body z_b axis, affecting ω_z and γ. This yields

$$\text{sys}_{roll} : \; P_{diff} \rightarrow \{\omega_z, \; \gamma\}.$$

Design MPC Velocity Controllers for Each Sub-System

3.1 Deliverable 3.1

3.1.1 Design procedure ensuring recursive constraint satisfaction

Following the project specification, we design each subsystem controller as a stabilizing MPC that enforces hard state and input constraints recursively.

Discrete-time delta dynamics. Starting from the trim pair (x_s, u_s), we formulate the controller in delta coordinates

$$\Delta x := x - x_s, \qquad \Delta u := u - u_s,$$

and use the discretized linear model (with sampling time $T_s = 1/20\,\mathrm{s}$)

$$\Delta x_{k+1} = A_d \Delta x_k + B_d \Delta u_k. \tag{1}$$

The commanded physical input is recovered via $u_k = \Delta u_k + u_s$.

Hard constraints in delta form. We enforce the constraints required by the linearization validity and actuation/safety limits:

$$|\alpha| \le 10^\circ, \qquad |\beta| \le 10^\circ, \qquad |\delta_1|, |\delta_2| \le 15^\circ, \qquad 40 \le P_{\text{avg}} \le 80,$$

(and the corresponding bound on P_{diff}). These limits are encoded as polyhedral sets

$$X := \{x \mid Fx \le f\}, \qquad U := \{u \mid Mu \le m\},$$

and implemented in delta form by shifting with the trim value $(x_\mathrm{s}, u_\mathrm{s})$:

$$F\Delta x_k \le f - Fx_\mathrm{s}, \qquad M\Delta u_k \le m - Mu_\mathrm{s}.$$

Terminal controller To guarantee recursive feasibility, we compute a stabilizing linear feedback $\Delta u = K\Delta x$ from the discrete-time LQR design and define the closed-loop matrix

$$A_{\text{cl}} := A_d + B_d K.$$

The terminal cost is chosen as the associated Riccati matrix Q_f.

Terminal invariant set. Under the terminal control law $\Delta u = K\Delta x$, the closed-loop dynamics are autonomous:

$$\Delta x^+ = (A_d + B_d K)\Delta x \ =: \ A_{\text{cl}}\Delta x.$$

To enforce state and input constraints under the terminal law, we restrict the state to

$$\Delta x \in X \cap K^{-1}U, \qquad K^{-1}U := \{\Delta x \mid K\Delta x \in U\}.$$

The terminal set is chosen as the maximal positively invariant set O_∞ with respect to $X \cap K^{-1}U$, computed by the pre-set iteration

$$\Omega_0 := X \cap K^{-1}U, \qquad \Omega_{i+1} := \text{pre}(\Omega_i) \cap \Omega_i, \qquad \text{pre}(S) := \{\Delta x \mid A_{\text{cl}}\Delta x \in S\}.$$

and stop when $\Omega_{i+1} = \Omega_i$, in which case

$$O_\infty = \Omega_i.$$

Finally, we choose the MPC terminal set as

$$X_f := O_\infty.$$

In the MPC problem we enforce the terminal constraint $\Delta x_N \in X_f$.

Why this implies recursive constraint satisfaction. Assume the MPC optimization is feasible at time k, producing a feasible sequence $\{\Delta u_0^\star, \ldots, \Delta u_{N-1}^\star\}$ and predicted states $\{\Delta x_0^\star, \ldots, \Delta x_N^\star\}$ that satisfy $\Delta x_i^\star \in X$, $\Delta u_i^\star \in U$ for all i and $\Delta x_N^\star \in X_f$. After applying $\Delta u_k = \Delta u_0^\star$, the successor state is $\Delta x_{k+1} = \Delta x_1^\star$. At time $k+1$, we build a candidate feasible input sequence by shifting the previous optimum and appending the terminal feedback:

$$\Delta u_i^{\text{cand}} := \begin{cases} \Delta u_{i+1}^\star, & i = 0, \ldots, N-2, \\ K\Delta x_N^\star, & i = N-1. \end{cases}$$

Since $\Delta x_N^\star \in X_f$ and X_f is positively invariant for $\Delta x^+ = A_{\text{cl}}\Delta x$ with $K\Delta x \in U$, the appended input satisfies the constraints and the resulting terminal state remains in $X_f \subseteq X$. Hence the MPC problem is feasible at time $k+1$. Therefore, feasibility and constraint satisfaction are preserved recursively.

Terminal set is primarily introduced to ensure recursive constraint satisfaction. In combination with the LQR terminal feedback and terminal cost, it also yields a stabilizing MPC (Lyapunov decrease in the terminal region).

3.1.2 Choice of tuning parameters (Q, R, H and terminal components)

Each subsystem MPC is tuned to (i) satisfy all hard state/input constraints, and (ii) meet the required settling time ($\leq 7\,\text{s}$) from the prescribed initial conditions ($5\,\text{m/s}$ for $x/y/z$ velocity controllers and $30°$ initial roll angle for the roll controller). The quadratic stage cost

$$\ell(\Delta x, \Delta u) = \Delta x^\top Q\,\Delta x + \Delta u^\top R\,\Delta u$$

is used to balance tracking performance against actuation effort, while the horizon H (with $N = H/T_s$ and $T_s = 1/20\,\text{s}$) is selected long enough to drive the state into the terminal set without making the QP unnecessarily large.

Roll subsystem. For the roll controller we use

$$Q_{\text{roll}} = \text{diag}(1,\, 20), \qquad R_{\text{roll}} = 1.$$

The larger weight on the second state (the roll angle γ) prioritizes fast reduction of the attitude error, which directly addresses the requirement of returning from a large initial roll angle to zero within the specified settling time. The input weight $R_{\text{roll}} = 1$ penalizes overly aggressive differential throttle commands and reduces chattering/saturation.

x- and y-velocity subsystems. For the x- and y-velocity controllers we use the same structure

$$Q_x = Q_y = \text{diag}(10,\, 1,\, 1), \qquad R_x = R_y = 1.$$

The higher weight on the first state penalizes the most constraint-critical internal motion (e.g., the attitude-rate/tilt-related state driving lateral acceleration), preventing the optimizer from producing overly aggressive maneuvers that would violate the linearization-validity bounds on α and β or saturate the actuator deflections. The remaining states are weighted to ensure the velocity component is driven to zero while keeping the transient smooth. The choice $R = 1$ discourages large steering inputs and helps respect the actuator limits.

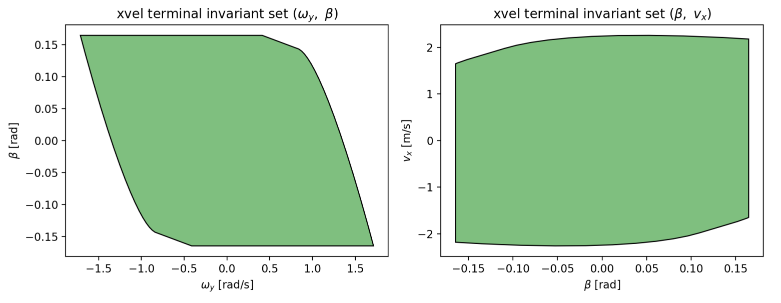
z-velocity subsystem. For the vertical velocity controller we use

$$Q_z = 1, \qquad R_z = 1,$$

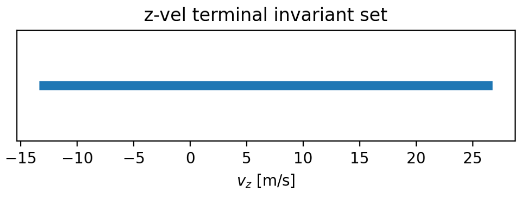
which provides a balanced trade-off between reducing the vertical velocity error and avoiding abrupt changes in the average throttle command. This is particularly important to satisfy the hard safety bound $40 \leq P_{\text{avg}} \leq 80$ while achieving a well-damped transient.

Terminal components. To guarantee recursive feasibility and improve stability, each MPC includes standard stabilizing terminal ingredients: a terminal feedback $\Delta u = K\Delta x$ computed by discrete-time LQR, a quadratic terminal cost $V_f(\Delta x) = \Delta x^\top Q_f \Delta x$ given by the associated Riccati solution, and a terminal set constraint $\Delta x_N \in X_f$, where X_f is chosen as the maximal positively invariant set $O_\infty(X \cap K^{-1}U)$ for the closed-loop dynamics $\Delta x^+ = (A_d + B_d K)\Delta x$ under constraints.

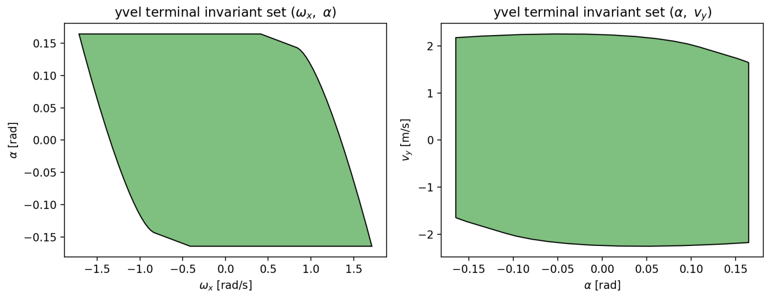
3.1.3 Plot of terminal invariant set for each of the dimensions



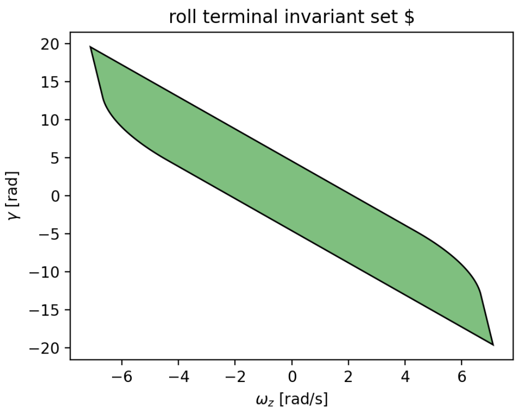
(a) x-subsystem



(b) z-subsystem



(c) y-subsystem



(d) roll subsystem (γ)

Figure 1: Terminal invariant sets X_f for each decoupled subsystem.

Terminal invariant set design (subsystem-dependent). We implement MPC in delta coordinates around the steady-state (x_s, u_s): $\Delta x := x - x_s$, $\Delta u := u - u_s$, with the real input $u = \Delta u + u_s$. Under the terminal feedback $\Delta u = K\Delta x$, the delta closed-loop is $\Delta x^+ = (A + BK)\Delta x$; hence any terminal set invariant for $(A+BK)$ remains invariant in delta coordinates. The practical issue is constraint satisfaction after shifting by (x_s, u_s).

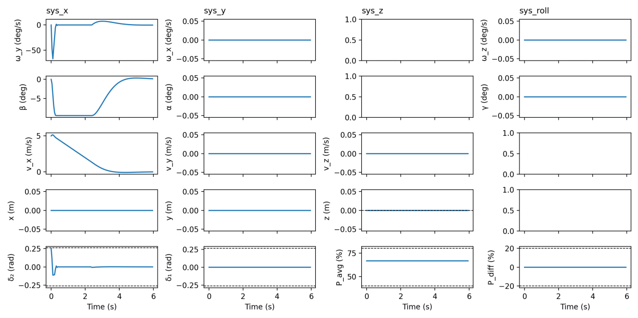
We consider two terminal-set choices: (i) *Reuse* the terminal set computed under the original constraints (Option 1)and (ii) *Recompute* the terminal set under the shifted delta constraints $F\Delta x \leq f - Fx_s$, $G\Delta u \leq g - Gu_s$ (Option 2), which typically yields a larger/maximal admissible invariant set. Design choice per subsystem: For the x-velocity, y-velocity, and roll subsystems we use **Option 1**. In these channels, the corresponding trim inputs are close to the center of their admissible ranges (i.e., small or symmetric offsets), so shifting by u_s does not significantly reduce the admissible margin; reusing the original terminal set is therefore simple and sufficient.

For the z-velocity subsystem we use **Option 2** . The vertical steady-state input (average thrust) is typically nonzero to balance gravity, which shifts and often tightens the admissible range of Δu (and may introduce strong asymmetry). Recomputing the terminal invariant set under $G\Delta u \leq g - Gu_s$ ensures terminal feasibility of the *real* input $u = \Delta u + u_s$ without rescaling.

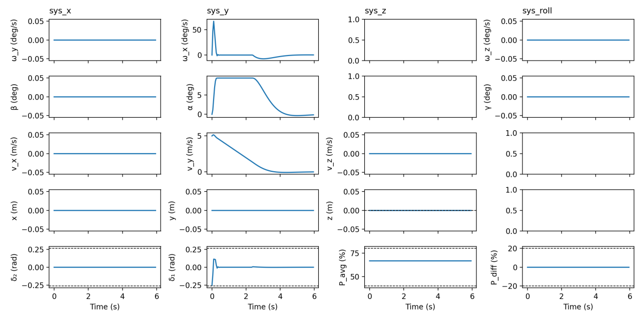
Use of tuning parameters. The shape and size of X_f are primarily determined by the terminal feedback gain K and the hard constraint bounds defining X and U. The LQR tuning matrices $(Q_{\mathrm{LQR}}, R_{\mathrm{LQR}})$ used to compute K directly affect (i) the closed-loop dynamics through A_{cl}, and (ii) the constraint mapping $K^{-1}U$ (i.e., which states would lead to input saturation under $\Delta u = K\Delta x$). More aggressive feedback (typically smaller R_{LQR} or larger penalties in Q_{LQR}) may speed up convergence but can shrink $K^{-1}U$ and thus reduce the feasible terminal region X_f, whereas a more conservative feedback can enlarge X_f at the expense of slower transients. Finally, the MPC prediction horizon H does not change the computed invariant set itself (since X_f depends on A_{cl}, X and U), but it affects whether the finite-horizon optimizer can drive the state into X_f while respecting constraints.

3.1.4 Open-loop and closed-loop plots

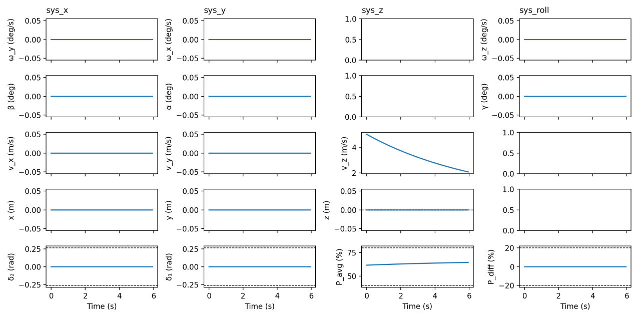
Plots of open-loop control are shown in Fig. 2, and plots of close-loop control are shown in Fig. 3



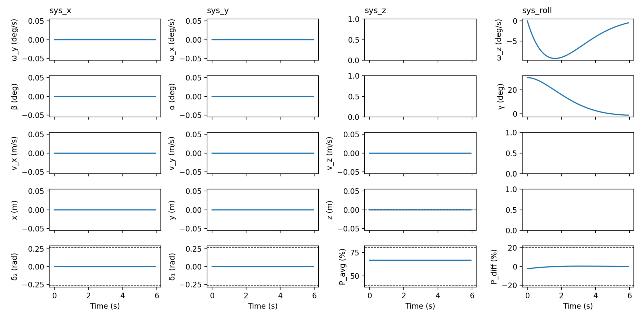
(a) $v_x(0) = 5$ m/s



(b) $v_y(0) = 5$ m/s



(c) $v_z(0) = 5$ m/s



(d) $\gamma(0) = 30°$ (stationary)

Figure 2: Open-loop plots

3.2 Deliverable 3.2

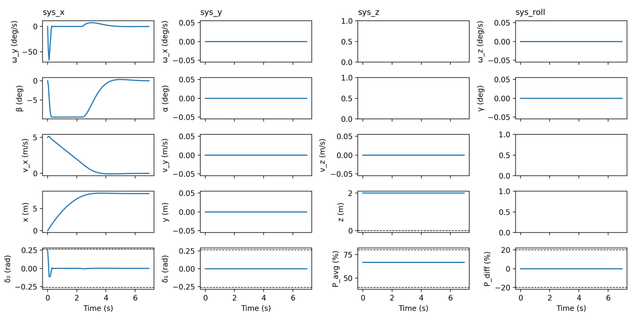
3.2.1 design procedure and choice of tuning parameters and choice of tuning parameters

We extend our controllers so that they can track constant velocity references. We define the deviation variables from the steady-state target (x_{tgt}, u_{tgt}) with

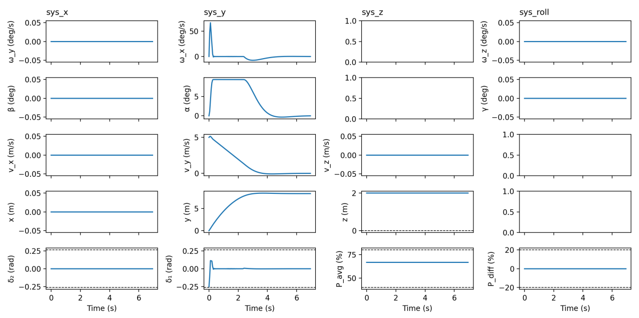
$$\Delta x := x - x_{tgt}, \qquad \Delta u := u - u_{tgt},$$

and the system model of

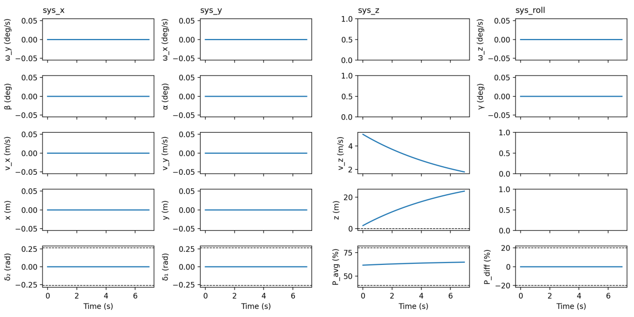
$$\Delta x_{k+1} = A_d \Delta x_k + B_d \Delta u_k. \tag{2}$$



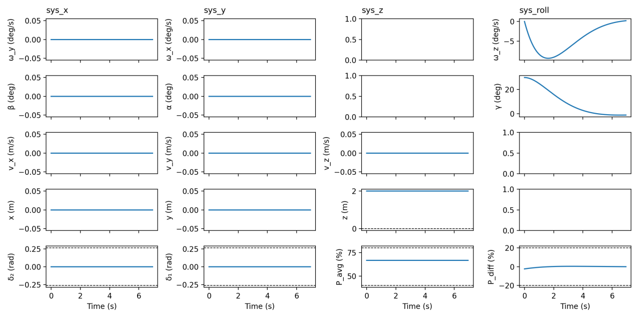
(a) $v_x(0) = 5$ m/s



(b) $v_y(0) = 5$ m/s



(c) $v_z(0) = 5$ m/s



(d) $\gamma(0) = 30°$ (stationary)

Figure 3: MPC closed-loop plots

with

$$u_k = \Delta u_k + u_{tgt}$$

and the constraint polyhedral should be shifted as :

$$F\Delta x_k \leq f - Fx_{\text{tgt}}, \qquad M\Delta u_k \leq m - Mu_{\text{tgt}}.$$

For the parameter tuning update, we initially reused the MPC tuning from Deliverable 3.1 for all subsystems to keep the controller design consistent. During the tracking experiments, we observed that the z-velocity subsystem exhibited a significantly slower response compared to the lateral subsystems, leading to an unsatisfactory tracking performance in v_z. To improve the transient speed, we increased the state penalty and reduced the input penalty for the z controller, setting $Q_z = 10$ and $R_z = 0.005$. This change makes deviations in v_z more costly while allowing stronger throttle action when needed, resulting in a much faster convergence of the vertical velocity while still respecting the imposed input constraints.

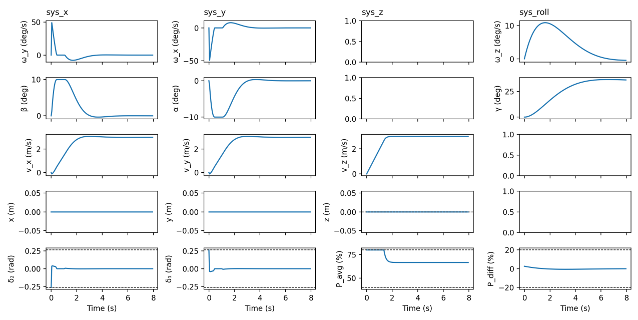
3.2.2 plots of velocity and radians tracking

The velocity and radians tracking results are shown in Fig. 4.

3.3 Deliverable 3.3 - PID Position Tracking Controller

Control objective. The goal is to bring the rocket from an initial condition $\text{pos}(0) = [50, 50, 100]^\top$ m to the position reference $\text{pos}_{\text{ref}} = [0, 0, 10]^\top$ m while tracking a roll reference $\gamma_{\text{ref}} = 0°$. Closed-loop results are shown in Fig. 5.

Controller architecture (PI position loop + MPC velocity loop). We implement a cascaded structure:


(a) open-loop tracking

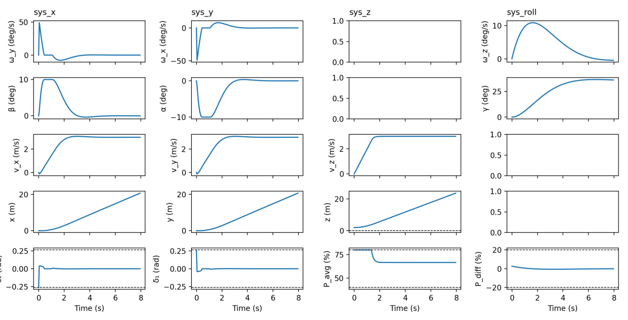

(b) close-loop tracking

Figure 4: Open-loop and closed-loop plots starting at the origin and tracking a velocity reference of 3 m/s (for x, y and z) and to 35° for roll.

- **Outer-loop position controller**
- **Inner-loop MPC** a linear MPC tracks the velocity setpoints and enforces constraints. Following Deliverable 3.1, the 12D linearized model is separated into four approximately decoupled subsystems:

$$\text{x-vel: } [\omega_y, \beta, v_x] \to \delta_2, \quad \text{y-vel: } [\omega_x, \alpha, v_y] \to \delta_1, \quad \text{z-vel: } [v_z] \to P_{\text{avg}}, \quad \text{roll: } [\omega_z, \gamma] \to P_{\text{diff}}.$$

Tuning procedure and choice of Q, R. We initially reused the (Q, R) weights from Deliverable 3.1. In closed-loop simulations, this resulted in an insufficient vertical performance (the altitude response did not properly settle around $z = 10\,\text{m}$ within the simulated horizon), and overly sharp variations in the deflection angles δ_1 and δ_2. Therefore, we retuned the MPC weights:

- **z-subsystem:** $Q_z = \text{diag}([10])$, $R_z = \text{diag}([0.5])$, to prioritize vertical speed tracking while regularizing throttle variations.
- **x/y subsystems:** $Q_x = Q_y = \text{diag}([100, 1, 1])$, $R_x = R_y = \text{diag}([0.5])$. The increased weight on angular-rate states reduces aggressive attitude transients and mitigates abrupt control actions in δ_1, δ_2.

Results. With the tuned weights, the rocket converges to $[x, y, z] \approx [0, 0, 10]\,\text{m}$ and the velocity states track the PI-generated references.The deflection angles exhibit reduced sharp peaks compared to the initial tuning.

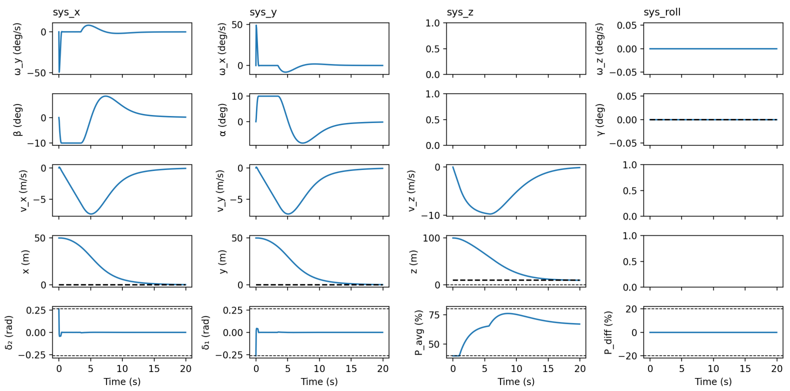


Figure 5: Closed-loop position tracking. Solid lines: states/inputs. Dashed lines: references

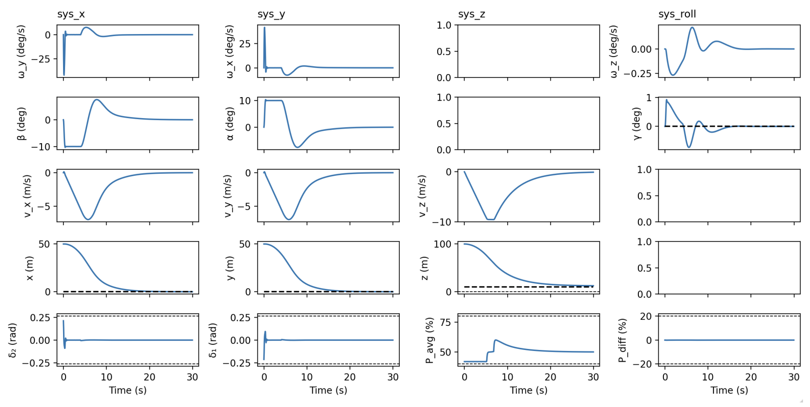


Figure 6: Completing the maneuver.

<u>Simulation with Nonlinear Rocket</u>

4.1 Deliverable 4.1

Fig. 6 show the completion of the maneuver from initial position $[50, 50, 100]$ to target position $[0, 0, 10]$.

We tuned the parameters based on Tab. 1, in order to avoid aggressive maneuvers, we made weights on angular velocities higher. Also as we see a steady state offset of z when converging, we give higher weights on the v_z and lower weights on the input P_{avg} to make the vel-z subsystem focus more on velocity tracking. The horizon is set to 7.0.

These tuning of parameters may result in infeasible or constraint violations. Hence, we (i) remove the terminal set constraint, (ii) add a slack variable for soft state constraints (listing 1) and (iii) add more safety margins in input constraints, which tightens the margin of input constraints that we should never violate (listing 2).

```
# Penalty for slack variables (L2 + L1 penalty)
cost += rho * (cp.sum_squares(self.s_var[:, i]) + cp.sum(self.s_var[:, i]))
# Soft state constraints (with slack variables)
constraints.append(self.X.A @ self.x_var[:, k] <= self.X.b + self.s_var[:, k])
```

Listing 1: (ii) slack variables

```
self.params['delta'] = (-np.deg2rad(15)+0.05, np.deg2rad(15)-0.05)
self.params['P_avg'] = (40+2, 80-2)
```

Listing 2: (iii) safety margins

Table 1: Deliverable 4.1: Parameters Tuning

subsystem	**states**	Q	R
Roll	$[w_z,\ \gamma]$	diag(1.0, 20.0)	diag(1.0)
x	$[w_y,\ \beta,\ v_x]$	diag(100.0, 1.0, 1.0)	diag(1.0)
y	$[w_x,\ \alpha,\ v_y]$	diag(100.0, 1.0, 1.0)	diag(1.0)
z	$[v_z]$	diag(10)	diag(0.005)

Offset-Free Tracking

5.1 Deliverable 5.1

5.1.1 Explanation of your design procedure and choice of tuning parameters.

The MPC uses the estimated disturbance $\hat{d}(k)$ to compensate in the prediction model:

$$x(k+1) = Ax(k) + Bu(k) + B\hat{d}(k)$$

where $d(k) \in \mathbb{R}^{n_d}$ is the disturbance vector. For constant disturbances:

$$d(k+1) = d(k)$$

For Augmented State-space Model: Define the augmented state:

$$x_a(k) = \begin{bmatrix} x(k) \\ d(k) \end{bmatrix} \in \mathbb{R}^{n_x+n_d}$$

The augmented system dynamics are:

$$\begin{bmatrix} x(k+1) \\ d(k+1) \end{bmatrix} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \begin{bmatrix} x(k) \\ d(k) \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u(k) + \begin{bmatrix} L_x \\ L_d \end{bmatrix} (C\hat{x}(k) - y_k)$$

where:

$$\hat{y}_k = C\hat{x}(k)$$

```python
Bd = self.B  # Disturbance matrix (same as input matrix)
# A_hat = [A  Bd;  0  I]
self.A_hat = np.vstack((
    np.hstack((self.A, Bd)),
    np.hstack((np.zeros((self.nu, self.nx)), np.eye(self.nu)))
))
# B_hat = [B; 0]
self.B_hat = np.vstack((
    self.B,
    np.zeros((self.nu, self.nu))
))
# C_hat = [C  Cd], where Cd = 0 (disturbance doesn't appear directly in output)
Cd = np.zeros((self.ny, self.nu))
self.C_hat = np.hstack((self.C, Cd))
# Pole placement for observer (moderate eigenvalues as in reference)
n_poles = self.nx + self.nu
poles = np.array([0.5, 0.6])
from scipy.signal import place_poles
res = place_poles(self.A_hat.T, self.C_hat.T, poles)
self.L = -res.gain_matrix.T
# Initialize estimates (as column vectors)
self.u_prev = np.zeros((self.nu, 1))
self.x_hat = np.zeros((self.nx, 1))
self.d_estimate = np.zeros((self.nu, 1))
```

Listing 3: Setting up estimator

```python
z_hat_current = np.vstack((self.x_hat, self.d_estimate))
# Observer update: z_hat_next = A_hat @ z_hat + B_hat @ u + L @ (C_hat @ z_hat
   - y)
```

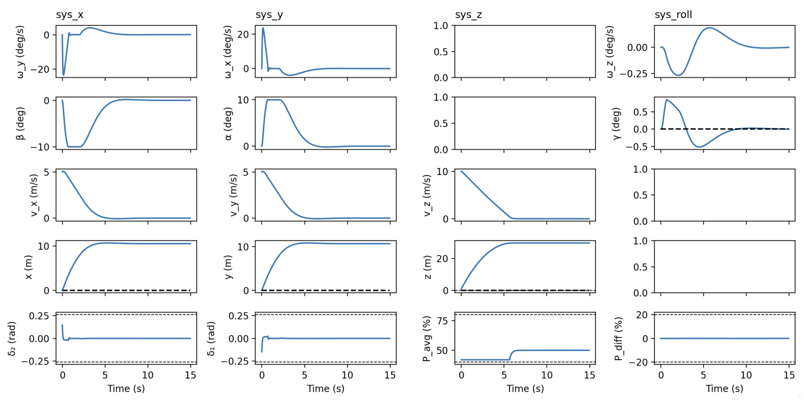


Figure 7: Simulating 15 seconds from $\text{pos}_0 = [0, 0, 1]$ and $v_0 = [5, 5, 10]$ with $v_{\text{ref}} = [0, 0, 0]$.

```
z_hat_next = self.A_hat @ z_hat_current + self.B_hat @ self.u_prev + self.L @
    (self.C_hat @ z_hat_current - y)
# Update estimates
self.x_hat = z_hat_next[:self.nx]
self.d_estimate = z_hat_next[self.nx:]
```

Listing 4: Updating estimator

```
# cost of slack variable
cost += 1e8 * cp.sum_squares(slack)
# xs = A*xs + B*us + B*d
constraints.append(self.xs_var == self.A @ self.xs_var + self.B @ self.us_var
    + self.B @ self.d_estimate_param + slack)
```

Listing 5: Steady state equilibrium and slack variable to steady-state optimizer

The steady-state target optimization accounts for the estimated disturbance:

$$\begin{aligned} \min_{x_s, u_s} \quad & u_s^T u_s \\ \text{s.t.} \quad & x_s = Ax_s + Bu_s + B\hat{d} \\ & Cx_s = r \\ & x_{\min} \leq x_s \leq x_{\max} \\ & u_{\min} \leq u_s \leq u_{\max} \end{aligned}$$

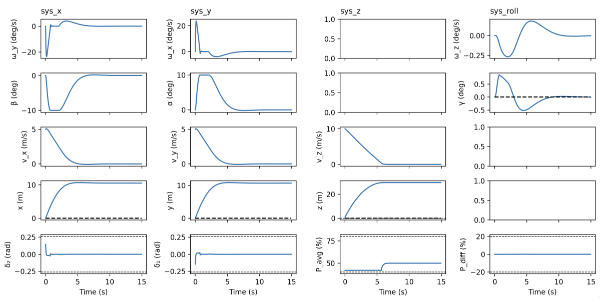
where r is given by as $y_{target} = Cx_{target}$. The steady state optimizer could easily fail due to strict equivalents, therefore, we as well add a slack variable to the equilibrium (Listing 5).

The Luenberger pole-placing is relatively easy, we choose two poles of: $[0.5, 0.6]$. The other parameters remain the same with Part 4.

In the controller, we should also include the disturbance term for offset-free tracking (Listing 6)

```
# Dynamics with disturbance: x^+ = Ax + Bu + Bd
constraints.append(
    self.x_var[:, k + 1] == self.A @ self.x_var[:, k] + self.B @ self.u_var[:,
        k] + self.B @ self.d_estimate_param)
```

Listing 6: Controller with disturbance


(a) Part 5 code. v_z final offset: $1e-9$.

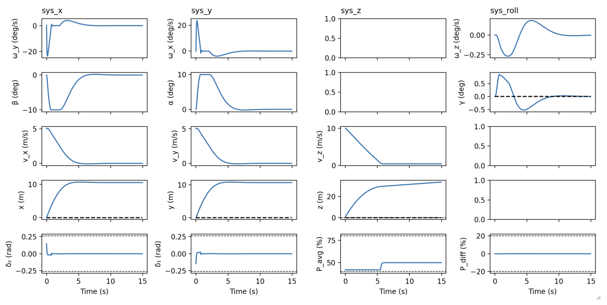
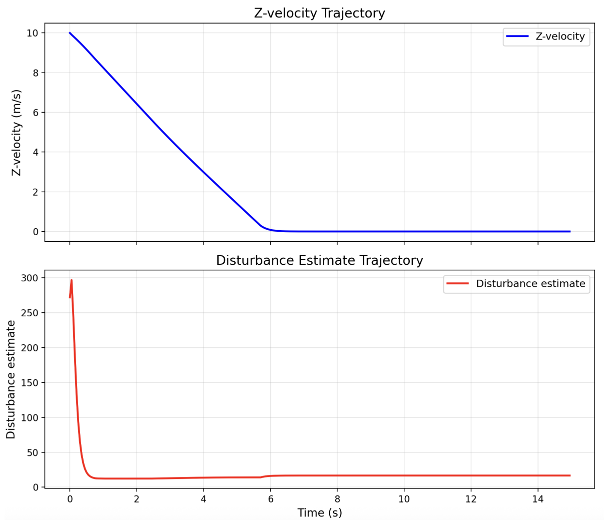
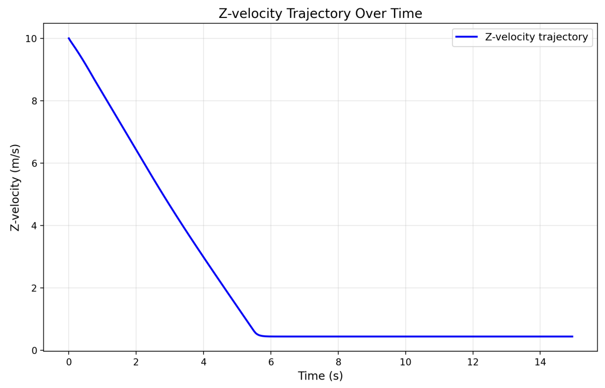

(b) Part 4 code. v_z final offset: 0.43897129

Figure 8: Comparison of Part 4 and Part 5. Simulating with `rocket.mass=1.5` and `rocket.fuel_rate=0` for 15 seconds. Comparison is shown especially in the velocity control of Subsystem Z.



(a) Part 5. Estimation disturbance converges at 16.6667, successfully pushing the estimation error of v_z to converge to 0.



(b) Part 4. Without the offset-tracking controller, there is steady-state offset error of v_z.

Figure 9: Estimation of disturbance and estimation error over time.

5.1.2 Offset-free Tracking Comparing with Part 4

Figure 8 showcase the result of with and without offset. According to the Subplot velocity of Subsystem Z, we can clearly see that with offset-free control, the closed-loop could manage to decrease the deviation and bring the initial velocity to target. In the end of the simulation, Part 5 is able to converge in v_z. We plot the estimation disturbance over time in Figure 9.

5.1.3 Estimation of disturbance with time and mass

In our approach, the disturbance is almost constant for a constant rocket mass since the simulation system is almost the same and we constrain identical initial states, while in physical world, with the initial state being different, the disturbance is not constant since there are plenty of mismatches caused by other dynamic behaviors.

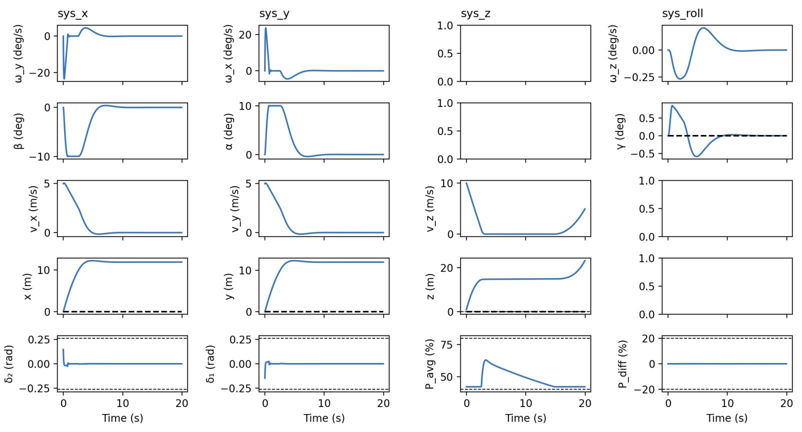


Figure 10: Simulating with `rocket.mass=2.0` and `rocket.fuel_rate=0.1` for 20 seconds.

5.2 Deliverable 5.2

5.2.1 Simulation with Different Fuel Rate

Figure 10 showcase the simulation plots.

5.2.2 Tracking Offset in Height (in vertical velocity)

When the rocket mass changes due to fuel consumption, the disturbance becomes **time-varying**:

$$d(k+1) \neq d(k)$$

The constant disturbance model $d(k+1) = d(k)$ cannot track this variation, causing the estimator to lag behind the true disturbance. This results in: (i) Persistent tracking errors that increase over time; (ii) The estimator ”chasing” a moving target; (iii) Delayed response to mass changes.

Possible modification: Model the disturbance as a random walk with process noise:

$$d(k+1) = d(k) + w_d(k)$$

where $w_d(k) \sim \mathcal{N}(0, Q_d)$ is zero-mean Gaussian process noise with covariance Q_d. The augmented system becomes:

$$\begin{bmatrix} x(k+1) \\ d(k+1) \end{bmatrix} = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} \begin{bmatrix} x(k) \\ d(k) \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} u(k) + \begin{bmatrix} 0 \\ I \end{bmatrix} w_d(k)$$

5.2.3 Distinct Properties of Trajectories

At simulation time 21.05s, the system appears to fail due to fuel exhaustion. With low fuel in the end of the system, the P_{avg} rapidly drops as v_z is already in steady state. With mass-variant tracking, there appears to be a tracking error at the end of the trajectory, being a bowl shape.

Robust Tube MPC for landing

6.1 Deliverable 6.1

6.1.1 Design Procedure and Tuning Rationale

The vertical (z) subsystem is described by the discrete-time linear model

$$x_{k+1} = Ax_k + Bu_k + Bw_k,$$

where $x = [v_z,\ z]^\top$, $u = P_{\text{avg}}$, and the additive disturbance satisfies

$$w_k \in \mathcal{W} = [-15,\ 5].$$

The altitude is subject to the state constraint

$$z \geq 0.$$

To ensure robust constraint satisfaction under bounded disturbances while achieving fast settling performance, a tube MPC framework is adopted. The control input is decomposed as

$$u_k = v_k + K(x_k - z_k),$$

where v_k is the nominal MPC input, z_k is the nominal state, and K is a stabilizing ancillary feedback gain.

Robust Tube Construction Defining the error $e_k = x_k - z_k$, the error dynamics are

$$e_{k+1} = (A + BK)e_k + Bw_k.$$

The gain K is chosen via discrete-time LQR, yielding a Schur-stable closed-loop matrix

$$A_{\text{cl}} = A + BK.$$

The disturbance-induced error is bounded by the minimal robust positively invariant (mRPI) set

$$\mathcal{E} = \bigoplus_{i=0}^{\infty} A_{\text{cl}}^i B\mathcal{W},$$

which is computed iteratively and truncated once $\|A_{\text{cl}}^i\|_2$ becomes sufficiently small. This set bounds the deviation between the actual and nominal trajectories for all admissible disturbances.

```python
def _compute_rpi_polyhedron(self, A_cl: np.ndarray, B: np.ndarray, w_min:
    float, w_max: float, max_iter: int = 50) -> Polyhedron:
    nx = A_cl.shape[0]

    W_scalar = Polyhedron.from_Hrep(
        A=np.array([[1], [-1]]),  # w <= w_max and -w <= -w_min
        b=np.array([w_max, -w_min])
    )

    # Apply affine map B to get W in state space: W = B @ W_scalar
    W = B @ W_scalar

    # Initialize with W (this is i=0 term)
    Omega = W
    itr = 0
    A_cl_ith_power = np.eye(nx)

    while itr < max_iter:
        A_cl_ith_power = np.linalg.matrix_power(A_cl, itr)
        Omega_next = Omega + A_cl_ith_power @ W
        Omega_next.minVrep() # NOTE: using minHrep() would fail!
        if np.linalg.norm(A_cl_ith_power, ord=2) < 0.02:
            print(f'Minimal robust invariant set computation converged after
                {itr} iterations.')
                break

        if itr == max_iter - 1:
            print(f'Minimal robust invariant set computation did NOT converge
                after {max_iter} iterations.')

        Omega = Omega_next
        itr += 1

    return Omega_next
```

Listing 7: Compute minimal robust positively invariant set

Constraint Tightening To guarantee satisfaction of the original constraints despite disturbances, the nominal constraints are tightened using the Pontryagin difference:

$$\tilde{\mathcal{X}} = \mathcal{X} \ominus \mathcal{E}, \qquad \tilde{\mathcal{U}} = \mathcal{U} \ominus K\mathcal{E}.$$

In particular, the altitude constraint $z \geq 0$ is enforced robustly by ensuring

$$z_k \in \tilde{\mathcal{X}} \;\Rightarrow\; x_k \in \mathcal{X}.$$

```
# Compute X_tilde = X - E (Pontryagin difference)
self.X_tilde = X - self.E
self.X_tilde.minHrep()

# Create original U as polyhedron (tightened input constraints)
# Original input constraints in delta coordinates
u_min_delta = np.array([40.0]) - self.us
u_max_delta = np.array([80.0]) - self.us
U_A = np.vstack([np.eye(self.nu), -np.eye(self.nu)])
U_b = np.hstack([u_max_delta, -u_min_delta])
U = Polyhedron.from_Hrep(A=U_A, b=U_b)

# Compute KE = {K*e : e in E}
KE = self.K @ self.E
self.U_tilde = U - KE
self.U_tilde.minHrep()
```

Listing 8: Tightening Constraints

Nominal MPC Optimization Problem At each time step, the following finite-horizon optimal control problem is solved:

$$\min_{\{z_k, v_k\}} \sum_{k=0}^{N-1} \left(\|z_k - z_{\text{target}}\|_Q^2 + \|v_k - v_{\text{target}}\|_R^2 \right) + \|z_N - z_{\text{target}}\|_P^2,$$

subject to

$$z_{k+1} = Az_k + Bv_k, \qquad z_k \in \tilde{\mathcal{X}}, \qquad v_k \in \tilde{\mathcal{U}},$$

with z_0 chosen such that

$$x_0 - z_0 \in \mathcal{E}.$$

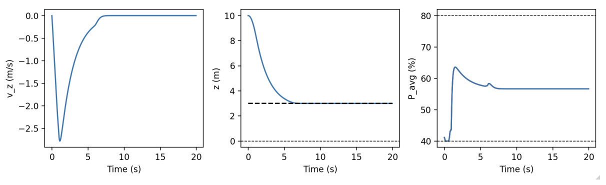
The terminal weight P is obtained from the discrete-time Riccati equation associated with the LQR design.

```python
cost = 0

# Stage cost - cost on nominal variables z and v
for k in range(self.N):
    dz = self.z_var[:, k] - self.x_target_param  # Nominal state deviation
    dv = self.v_var[:, k] - self.u_target_param  # Nominal input deviation

    # Tracking cost on nominal trajectory
    cost += cp.quad_form(dz, self.Q) + cp.quad_form(dv, self.R)

# Terminal cost on nominal state
dz_N = self.z_var[:, self.N] - self.x_target_param
cost += cp.quad_form(dz_N, self.P)

# Build constraints
constraints = []

# Initial condition: x0 in z0 + E
# This means: E.A @ (x0 - z0) <= E.b
constraints.append(self.E.A @ (self.x0_param - self.z_var[:, 0]) <= self.E.b)

# Dynamics on nominal variables z
for k in range(self.N):
    constraints.append(
        self.z_var[:, k+1] == self.A @ self.z_var[:, k] + self.B @
            self.v_var[:, k]
    )

    # X_tilde constraints: z_k in X_tilde
    constraints.append(self.X_tilde.A @ self.z_var[:, k] <= self.X_tilde.b)

    # U_tilde constraints: v_k in U_tilde
    constraints.append(self.U_tilde.A @ self.v_var[:, k] <= self.U_tilde.b)

# Terminal constraint: z_N in Xf
# constraints.append(self.Xf.A @ self.z_var[:, self.N] <= self.Xf.b)

self.ocp = cp.Problem(cp.Minimize(cost), constraints)
```
Listing 9: Tube MPC optimization problem

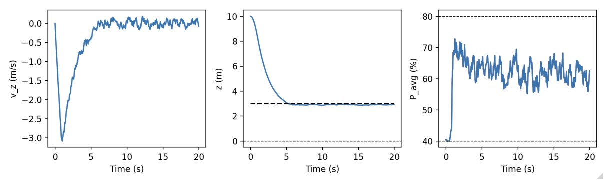
Performance and Tuning Considerations The settling-time requirement (no more than 4 seconds when descending from $z = 10$ to $z = 3$) is enforced through:

- a relatively large weight in Q on the altitude error to promote fast vertical convergence,
- a moderate input weight R to balance control effort and responsiveness,
- a prediction horizon N long enough to capture the full descent transient.

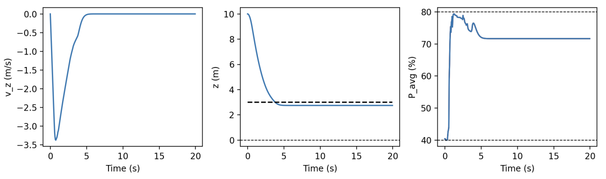
The ancillary LQR feedback provides fast local disturbance rejection, while the tube MPC structure guarantees robust constraint satisfaction. As a result, the closed-loop system achieves the required settling time while remaining robust to bounded disturbances.



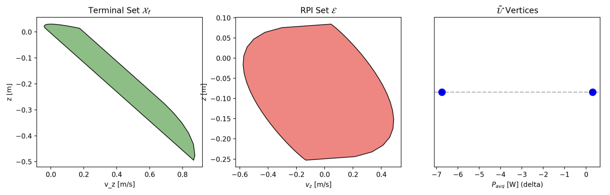
(a) Simulation with No Noise.



(b) Simulation with Random Noise.



(c) Simulation with Extreme Noise.



(d) Plot of the minimal invariant set $\mathcal{E}$, terminal set $\mathcal{X}_{\{}$, and vertices of the tightened input constraint $\tilde{\mathbb{U}}$.

Figure 11: Deliverable 6-1 results.

Results

With the approach above, we are able to converge in 5 seconds using $Q = \text{diag}(50, 400)$, $R = 0.08$ and $H = 0.25s$. The required plots are shown in Fig. 11. The $\tilde{\mathbb{U}}$ vertices are: $[-6.747401, 0.31079382]$ in our case. We acknowledge that there is an offset error in the Extreme Noise case, we think that it is due to the Full Noise input with bias, it could be effectively solve using a hard integrator at the end.

6.2 Deliverable 6.2

We created nominal systems for the other three systems without terminal constraints and state constraints. Using the parameters in Tab. 2, we could get the landing curves as shown in Fig. 12.

Table 2: Parameters Tuning for Four Subsystems

subsystem	states	Q	R
Roll	$[w_z,\ \gamma]$	diag(1.0, 10.0)	diag(0.1)
x	$[w_y,\ \beta,\ v_x,\ x]$	diag(1.0, 100.0, 1.0, 10.0)	diag(1.0)
y	$[w_x,\ \alpha,\ v_y,\ y]$	diag(1.0, 100.0, 1.0, 10.0)	diag(1.0)
z	$[v_z,\ z]$	diag(50.0, 400.0)	diag(0.08)

Nonlinear MPC

7.1 Deliverable 7.1

7.1.1 Explain your design procedure and choice of tuning parameters. Settling time is no more than four seconds when landing from x = 3, y = 2, z = 10, roll= 30 ∘ to x = 1, y = 0, z = 3, roll= 0 ∘ in the simulation using the original model.

Design Procedure

We adopt a full nonlinear MPC formulation using CasADi with the IPOPT solver, which directly handles the nonlinear rocket dynamics without requiring linearization or subsystem decomposition. The optimization problem is formulated as a direct multiple shooting approach with RK4 discretization.

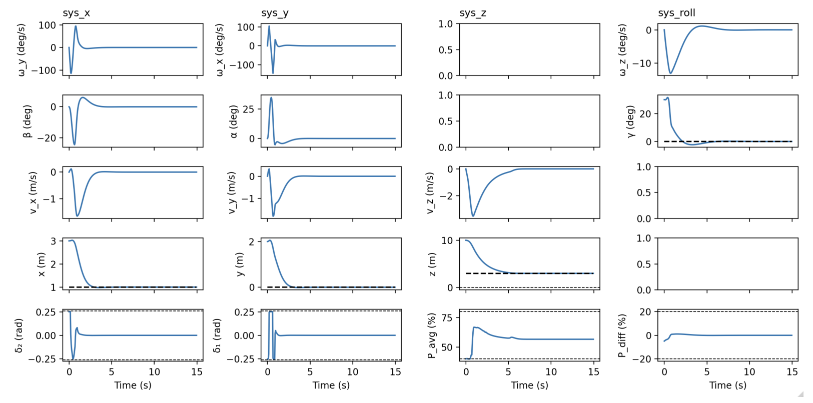


Figure 12: Landing MPC with all four subsystems from $x = 3, y = 2, z = 10, \text{roll} = 30°$ to $x = 1, y = 0, z = 3, \text{roll} = 0°$

The state vector $\mathbf{x} \in \mathbb{R}^{12}$ comprises angular velocities $(\omega_x, \omega_y, \omega_z)$, Euler angles (α, β, γ), linear velocities (v_x, v_y, v_z), and positions (x, y, z). The control input $\mathbf{u} \in \mathbb{R}^4$ includes gimbal deflections (d_1, d_2), average thrust P_{avg}, and differential thrust P_{diff}.

Tuning Parameters

The prediction horizon is selected as $H = 2.0$ s with a sampling time of $T_s = 0.1$ s, yielding $N = 20$ prediction steps. This horizon length provides sufficient lookahead for the landing maneuver while maintaining computational tractability.

The stage cost function is defined as:

$$\ell(\mathbf{x}_k, \mathbf{u}_k) = (\mathbf{x}_k - \mathbf{x}_{\text{ref}})^\top Q(\mathbf{x}_k - \mathbf{x}_{\text{ref}}) + (\mathbf{u}_k - \mathbf{u}_{\text{ref}})^\top R(\mathbf{u}_k - \mathbf{u}_{\text{ref}}) \tag{3}$$

Table 3: Parameters Tuning for Four Subsystems

subsystem	**states**	Q	R
Roll	$[w_z,\ \gamma]$	diag(1.0, 15.0)	diag(0.1)
x	$[w_y,\ \beta,\ v_x,\ x]$	diag(1.0, 10.0, 1.0, 10.0)	diag(0.1)
y	$[w_x,\ \alpha,\ v_y,\ y]$	diag(1.0, 100.0, 1.0, 10.0)	diag(0.1)
z	$[v_z,\ z]$	diag(10.0, 15.0)	diag(0.1)

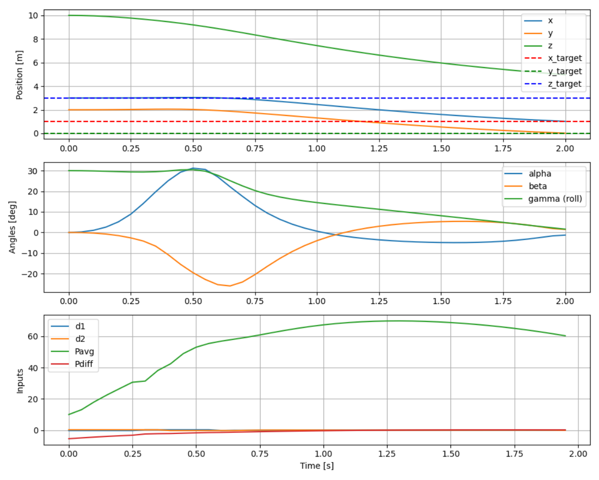
where higher weights of 10 are assigned to the Euler angles (α, β), positions (x, y) and velocity $v_z$to prioritize attitude stabilization and accurate position tracking,while angular and linear velocities receive unit weights. Since its a landing mission, z and γ are assigned weight of 15. Using parameters in Tab.3 could achieve plots in Fig.14.

The relatively small input weights permit aggressive control actions necessary for achieving the settling time constraint. The terminal cost matrix is calculated through lineared system around steady state.

Input constraints are imposed as $d_1, d_2 \in [-15°, 15°]$, $P_{\text{avg}} \in [10\%, 90\%]$, and $P_{diff} \in [-20°, 20°]$. State constraints include $z \geq 0$ to prevent ground penetration and $|\beta| \leq 80°$ to avoid gimbal lock singularities.

Justification

The selected tuning parameters achieve a settling time below four seconds by: (i) weighting position and attitude errors more heavily than rates to ensure rapid convergence to the target configuration; and


(a) Open loop behavior

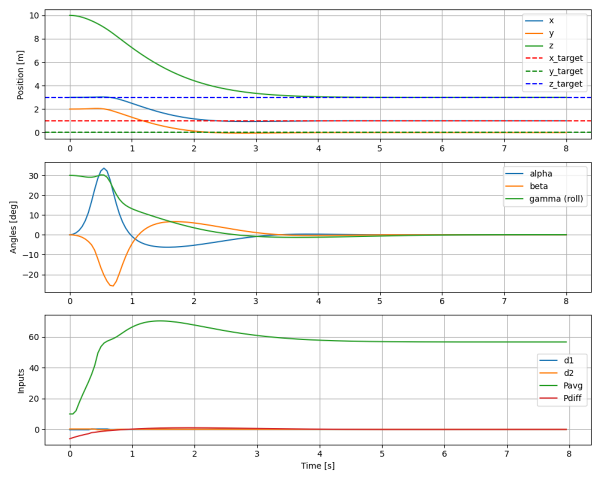

(b) Close loop behavior

Figure 13: **Open loop behavior is reasonable and Close loop bahevior is converged**

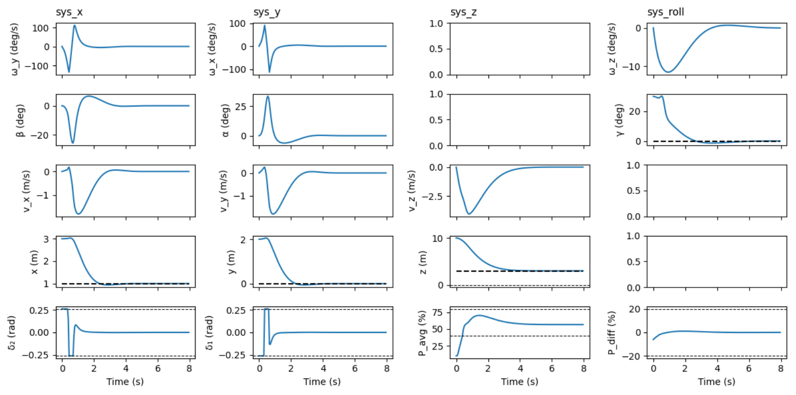


Figure 14: Landing Nonlinear MPC with all four subsystems from $x = 3, y = 2, z = 10, \mathrm{roll} = 30°$ to $x = 1, y = 0, z = 3, \mathrm{roll} = 0°$

(ii) utilizing modest input penalties to allow the controller to exploit the full actuator range during aggressive maneuvers. The warm-start strategy further enhances computational efficiency across successive optimization solves.

7.1.2 Open-loop and closed-loop plots (plot static states inputs) showing the performance of your controller in the simulation using the original model.

See Fig.13.

7.1.3 Compare your NMPC controller vs your "robust MPC and three nominal MPCs" (in Todo 6.2). Discuss the pros and cons.

Comparing the results to Figure 12, the NMPC clearly outperforms the hybridized Robust Tube MPC approach, achieving faster settling times and smoother trajectories by directly addressing system nonlinearities. The primary tradeoff, however, is the increased computational cost associated with solving the nonlinear optimization problem.