Assignment 4

Lorenzo Fici, Hannes Gustafsson, Davide Nascivera

September 2024

1 Introduction

In this Assignment, we will design a linear Model Predictive Control (MPC) for the Astrobee. The discrete-time linearized dynamics of the Astrobee, subjected to state and control constraints, can be represented as:

$$x_{t+1} = Ax_t + Bu_t, \tag{1}$$

with $u_t \in U$ and $x_t \in X$ for $t \geq 0$.

The sets U and X are polyhedra described by the following linear inequalities:

$$U = \{u : H_u u \leq h_u\}, \quad X = \{x : H_x x \leq h_x\}. \tag{2}$$

In this design task, we will consider the following state and control constraints:

$$x_{\min} \leq x_t \leq x_{\max}, \quad u_{\min} \leq u_t \leq u_{\max}, \tag{3}$$

with the following parameters:

$$\begin{aligned}
p_{\max} &= -p_{\min} = [1.2, 0.1, 0.1]^T, \\
v_{\max} &= -v_{\min} = [0.5, 0.5, 0.5]^T, \\
\theta_{\max} &= -\theta_{\min} = [0.2, 0.2, 0.2]^T, \\
\omega_{\max} &= -\omega_{\min} = [0.1, 0.1, 0.1]^T, \\
f_{\max} &= -f_{\min} = [0.85, 0.41, 0.41]^T, \\
t_{\max} &= -t_{\min} = [0.085, 0.041, 0.041]^T.
\end{aligned}$$

The MPC finite-horizon optimal control (FHOC) problem is formally defined as:

$$\min_{u_t} \sum_{k=0}^{N-1} \left(x_{t+k|t}^T Q x_{t+k|t} + u_{t+k|t}^T R u_{t+k|t} + x_{t+N|t}^T P x_{t+N|t} \right) \tag{4}$$

$$\begin{aligned} x_{t+k+1|t} &= A x_{t+k|t} + B u_{t+k|t}, \\ x_{t+k|t} &\in X, \\ \text{subject to:} \quad u_{t+k|t} &\in U, \\ x_{t+N|t} &\in X_f, \\ x_t|t &= x_t. \end{aligned} \tag{5}$$

where $P \succeq 0$ and X_f are the terminal cost and terminal constraint set, respectively. If the terminal constraint set X_f is control invariant, then the MPC problem is recursively feasible.

For this design task, we will consider two control invariant sets as the terminal constraint set:

- The zero terminal constraint set $X_f = \{0\}$;
- The invariant set for the closed-loop dynamics under the infinite-horizon LQR control X_{LQR}^f.

2 Questions

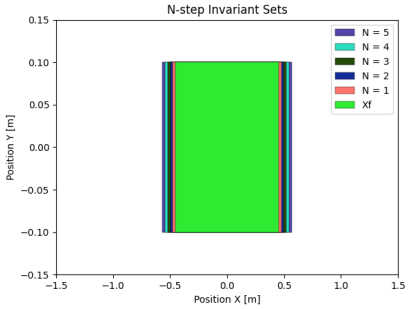
2.1 Q1

Here, our objective is to compute the control invariant set $\mathcal{K}_N(\mathcal{X}_N)$ for varying control horizons $N = 5, 10$, and 20, while maintaining the state and control constraints.

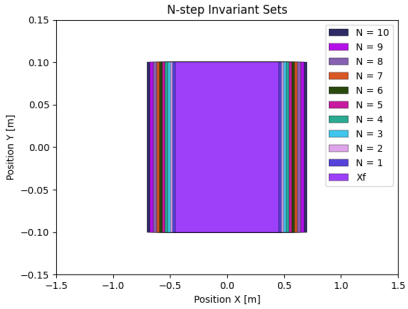
The control invariant set is a subset of the state space of a dynamical system where, if the system's state starts within this set, it can be controlled to remain within this set for all future time steps. Figure 1 shows the result of using $\mathcal{X}_f^{LQR}$ as the terminal state. As shown, this set increases with larger N, enhancing the Model Predictive Control (MPC)'s ability to steer the system toward the LQR terminal region.

For $N = 5$, the set $\mathcal{K}_5(\mathcal{X})$ is smaller, limiting control options. As N increases to 10 and 20, the sets $\mathcal{K}_{10}(\mathcal{X})$ and $\mathcal{K}_{20}(\mathcal{X})$ grow, providing greater control flexibility but also increasing computational complexity. Overall, this highlights the trade-off between control horizon length and feasibility in practical implementations. A similar behavior is shown in Figure 2, with the the zero terminal constraint.

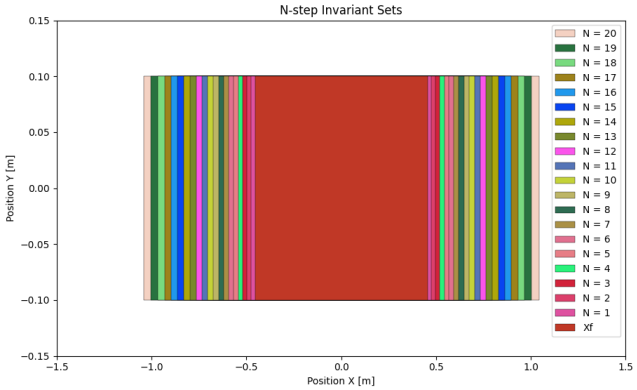
Moreover, even if every time it is printed on the terminal that the system converges with determinedness index 10, we noticed that the control set enlarges as N increases, so did doesn't really match the index that is printed. After additional research, it seems that this is because that print comes from the computation of the LQR invariant set which is not related to the maximal controllable set.



(a) Q1, N=5



(b) Q1, N=10



(c) Q1, N=20

Figure 1: Comparative results for Q1 with different values of N, for the terminal state $\mathcal{X}_f^{LQR}$

2.2 Q2

We analyze how the control invariant set $\mathcal{K}_N(\mathcal{X}_N)$ is affected by scaling the control constraints. Specifically, we compare the original constraints with a scaled version where $3 \cdot u_{\max} = -3 \cdot u_{\min}$.

As shown in Figure 3, increasing the control limits significantly expands the invariant set. This greater flexibility allows the controller to access a wider range of trajectories, enhancing its ability to steer the system toward desired terminal states.

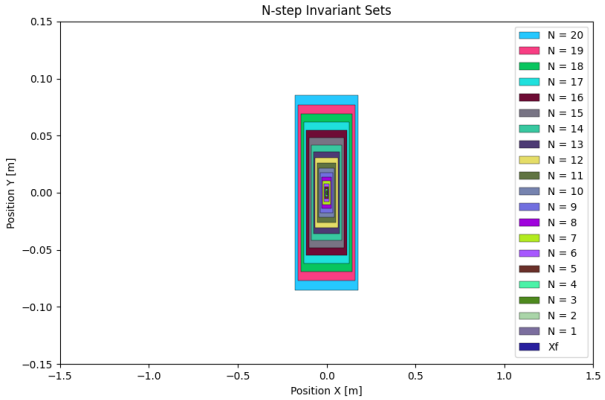


Figure 2: The control invariant set for the terminal state $\mathcal{K}_{20}(\mathcal{X}_f^{origin})$

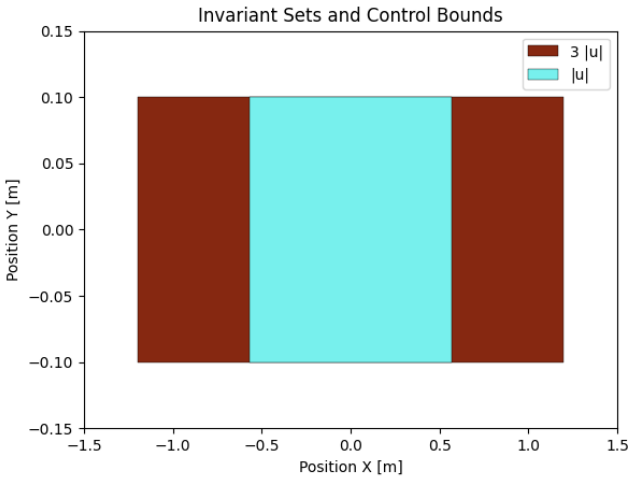


Figure 3: Q2, scaling the constraint on u

2.3 Q3

1. How the state constraints are set

The state constraints in the Model Predictive Control (MPC) class are set within the constructor method `__init__`. Specifically, the constraints are handled in the for-loop that iterates over the prediction horizon `self.Nt`. Within each iteration, constraints are added to ensure that the state and control variables remain within their defined bounds. This is done using:

```
# State constraints
if xub is not None:
    non_inf_indx = (xub != np.inf).flatten()
    if np.any(non_inf_indx):
        con_ineq += [x_t[non_inf_indx] <= xub[non_inf_indx].flatten()]
if xlb is not None:
    non_inf_indx = (xlb != -np.inf).flatten()
    if np.any(non_inf_indx):
        con_ineq += [x_t[non_inf_indx] >= xlb[non_inf_indx].flatten()]
```

Here, `xub` and `xlb` represent the upper and lower bounds on the state variables. Only the elements that have finite values are included in the inequality constraints `con_ineq`.

2. How the objective function is formulated

The objective function for the MPC is formulated as a sum of running costs and a terminal cost. The running cost is added at each time step `t` and involves a quadratic form:

$$\text{obj} += \text{self.running\_cost}((x_t - x0\_ref), Q, u_t, R) \tag{6}$$

The `running_cost` function is defined as:

```
self.running_cost = lambda x, Q, u, R: cp.quad_form(x, Q) + cp.quad_form(u, R)
```

This represents a weighted sum of the state deviation from the reference $(x_t - x0_{ref})$ and the control effort u_t. The terminal cost is added outside the loop:

```
obj += self.terminal_cost(x_vars[:, self.Nt] - x0_ref, self.P)
```

The terminal cost function is similarly defined as:

```
self.terminal_cost = lambda x, P: cp.quad_form(x, P)
```

This ensures that the final state is penalized according to matrix P.

3. How the terminal constraint is set

The terminal constraint is handled after the for-loop that constructs the constraints over the horizon. If a terminal constraint `terminal_constraint` is provided, it is applied using the polytope's constraint matrix H_N and vector H_b:

```
if terminal_constraint is not None:
    H_N = terminal_constraint.A
    H_b = terminal_constraint.b
    con_ineq += [H_N @ x_vars[:, self.Nt] <= H_b]
```

This constraint ensures that the terminal state x_N lies within the specified polytope.

4. What the variable `param_s` holds

It appears there is no variable explicitly named `param_s` in the provided code. However, the parameters `x0`, `u0`, and `x0_ref` defined in the constructor are used as parameter inputs to the MPC problem and represent the initial state, initial control, and reference state, respectively.

5. Why we only select the first element of the control prediction horizon in the mpc_controller method

In the `mpc_controller` method, only the first element of the control prediction horizon is applied to the system:

```
_, u_pred = self.solve_mpc(x0)
return u_pred[:, 0].reshape(self.Nu, 1)
```

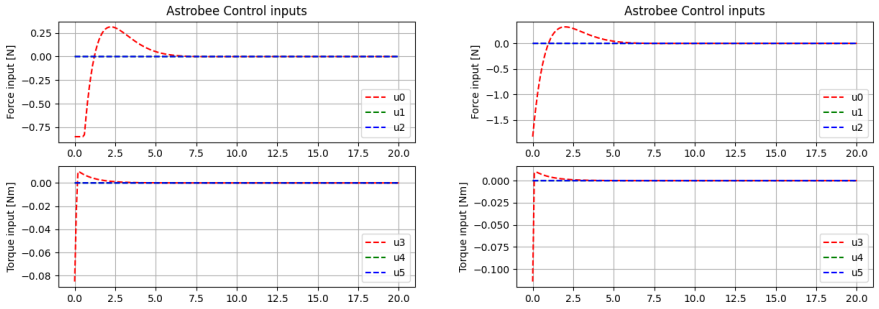
This is because MPC uses a receding horizon strategy. At each step, the optimization problem is solved for the entire horizon, but only the first control input is implemented. The system state is then updated, and the optimization is resolved at the next step, taking into account the updated state. This approach provides feedback and allows the controller to adapt to disturbances or model inaccuracies.

2.4 Q4

Here, we are tasked with simulating the system with the current setup, and compare its performance against increasing the control input bound by a factor x3.

Figure 4b shows the control inputs for both cases.

As shown in Table 1, increasing the upper and lower bounds for the input reduced the integral error in both position and attitude. However, this change also led to an increase in the system's overshoot.



(a) Default control limits u_{lim} (b) Setting $u_{max} = -u_{min} = 3 * u_{lim}$

Figure 4: Q4 - Comparison of increasing the control bounds

Although the input constraints have been expanded, the control inputs generated by the MPC remain within reasonable limits. This is due to the optimization problem balancing the objective function, where the control effort is penalized by the weight matrix R. As a result. At the same time, the magnitude of the inputs has increased, it has not escalated excessively, preventing a significant rise in control energy.

	Energy Used	Pos. Integral Err.	Att. Integral Err.
u_{lim}	156.20	3.94	0.91
$3 * u_{lim}$	160.52	3.56	0.88

Table 1: Performance comparison of increasing the control bound

2.5 Q5

Setting the terminal state to $\mathcal{X}_{origin} = \{\mathbf{0}\}$, the solver fails to find a solution. This can be explained by recalling the results from Q1, in particular Figure 2, where the control invariant set for the terminal state $\mathcal{K}_{10}(\mathcal{X}_{origin})$ does not include our initial vector, which offsets the Astrobee by 0.2 in the x-direction.

If we choose the terminal state to be $\mathcal{X}_f^{LQR}$ on the other hand, the control invariant set for $\mathcal{K}_{10}(\mathcal{X}_f^{LQR})$ do contain our initial vector, yielding the problem feasible.

2.6 Q6

To counteract the problem not being feasible in Q5, we extend the MPC horizon to $T = 50$. This results in a the MPC taking 0.0247 seconds to solve, as compared to the MPC taking 0.0069 seconds to solve for $T = 10$ with the terminal state $\mathcal{X}_f^{LQR}$ (an increase by a factor x3.5).

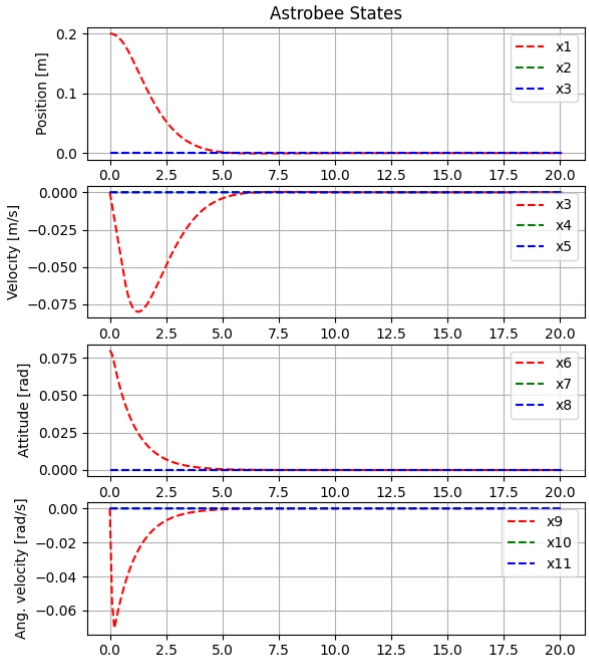


Figure 5: States evolution, $N = 50$, terminal state $\mathcal{X}_{origin} = \{\mathbf{0}\}$

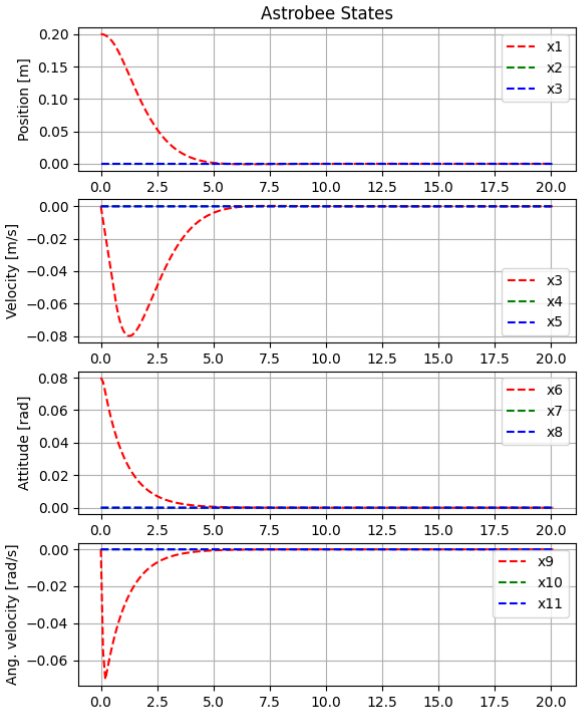


Figure 6: Q7 - Baseline states dynamics

2.7 Q7

We will now investigate what effect the tuning parameters have on the performance of the controller. We initialize $Q = np.eye(12)$ and $R = np.eye(6) * 0.01$ and study the effect of four experiments. Although the instructions said we should use $R = np.eye(6)$, the code actually re-scaled the R by 0.01, which we decided to stick with for the whole experiments. We increased the MPC-horizon from the default of 10, to 20 to make the experiments feasible.

The baseline is shown in Figure 6. Total Energy used was 156.17.

2.7.1 Multiply R by 10

In this first experiment, we multiple R by 10. This penalizes use of control inputs, which makes the system move slower towards the reference state.

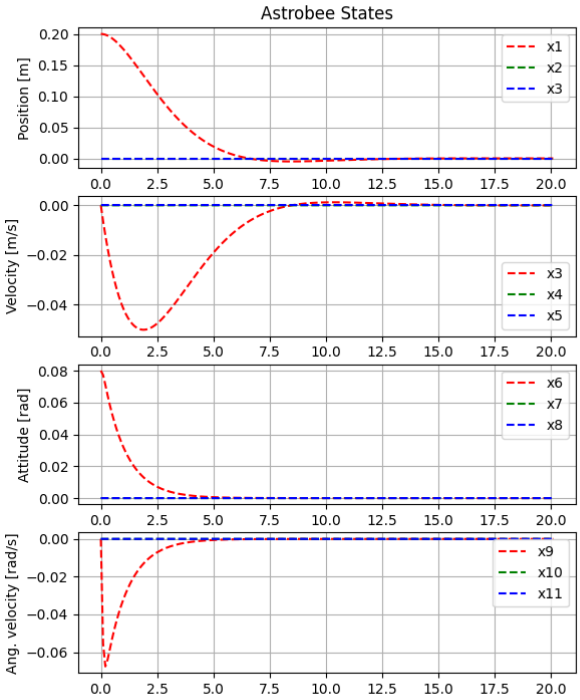


Figure 7: Q7 - Multiply R by 10

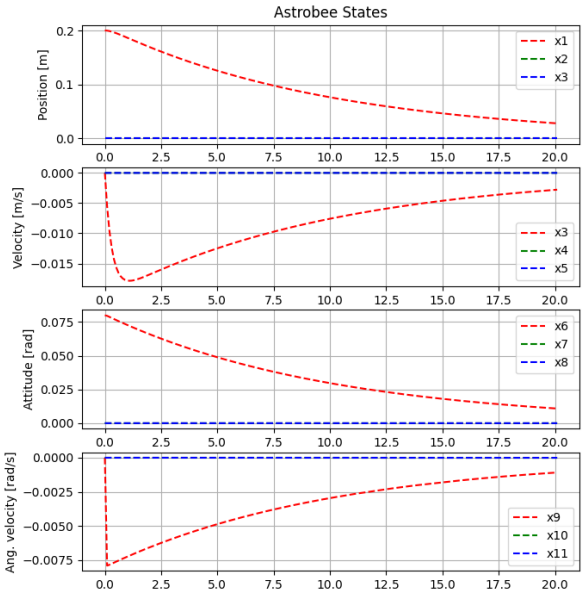


Figure 8: Q7 - Add 100 to the velocity components

2.7.2 Add 100 to the velocity components

Here, we add 100 to the velocity components of the diagonal of Q, Q[3:6] and Q[9:].

We noticed that if we keep R scaled by 0.01, reverting the multiplication performed in 2.7.1, adding 100 to the velocity components cause the MPC problem to become infeasible. Instead if we don't revert the multiplication carried out in 2.7.1, then it is possible to solve the problem. This is because adding a penalty in the cost matrix for the velocity causes the system to move slowly. It is possible to solve this problem by increasing the time horizon to 20, giving the system more time to reach the target.

Simulating the system with the new parameters we obtain a maximum module of the transitional & attitude velocity of 0.017 and 0.0075 while the original ones were 0.08 and 0.07, more than one measurement order.

2.7.3 Add 100 to the position and attitude components

Here, we revert the velocity components to their initial value of 1 and add 100 to the position and attitude components of the diagonal of Q, Q[0:3] and Q[6:9]. This penalizes deviations in position from the reference point, which makes the controller act with stronger inputs to correct the position.

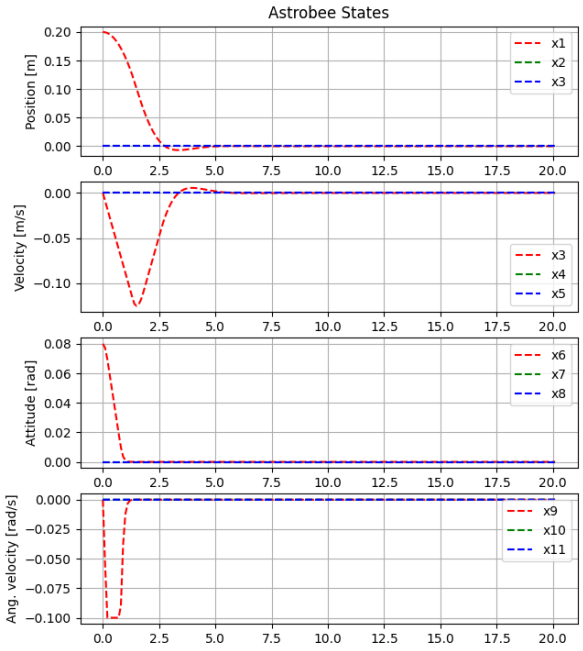


Figure 9: Q7 - Add 100 to the position and attitude components

Simulating the system with the new parameters results in a quicker convergence to the zero condition for both position and attitude. Qualitatively, the response speed can be considered approximately twice as fast.

2.7.4 Increase all elements of Q by 100

Here, we interpret this experiment as increasing all elements of Q by 100 on the diagonal. This penalizes deviations from the reference state, which forces the control inputs to be larger in magnitude. The energy used was 199.13, which agrees with the expectation.

2.7.5 Relation to the LQR controller

Throughout these experiments, the behavior we observed mirrored principles found in Linear Quadratic Regulator (LQR) controllers. The adjustments to the Q and R matrices directly influenced the cost function in the LQR framework, emphasizing the balance between minimizing state errors and control effort. Increasing penalties on control actions resulted in smoother, less aggressive

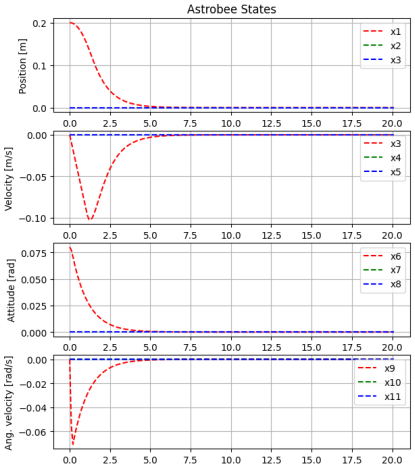


Figure 10: Q7 - Increase all elements of Q by 100

behavior, while raising penalties on state errors prompted quicker corrections.

3 Conclusion

In this report, we successfully implemented and analyzed a MPC for navigating and stabilizing an Astrobee system. We explored different control horizons, constraint variations, and tuning parameters, we understood how the effects of these factors influence the performance and energy efficiency of the controller.

We first set up our controller where the simulation results highlighted the trade-offs between computational complexity and control precision, with longer horizons providing better control but at the cost of increased computational time.

We also saw how varying the control limits and terminal constraint sets influences system behavior, further reinforcing the adaptability of MPC in constrained environments. The MPC's ability to handle system constraints and multiple-input multiple-output systems, optimizing energy efficiency is excellent compared to other control techniques seen previously in other control courses like PID.

This assignment deepened our understanding of how MPC can be calibrated to meet specific performance requirements.