KTH Royal Institute of Technology

EL2700 - Model Predictive Control

# Final Project: Nonlinear MPC for Space Rendezvous

**Division of Decision and Control Systems**
**School of Electrical Engineering and Computer Science**

**Authors:**
*Jacopo Dallafior, Stefano Tonini, Oscar Ceder*

**September 25, 2024**

# Contents

# Chapter 1

# Introduction

In this report, we aim to solve an on-orbit rendezvous problem using Nonlinear Model Predictive Control (NMPC). The scenario involves rendezvous with Honey, a broken Astrobee that needs to be de-orbited. Bumble, a fellow Astrobee, is tasked with approaching Honey using a controller designed through NMPC principles. The problem will be simulated in the NASA Astrobee simulator, with a performance comparison among different solutions.

# Chapter 2

# Astrobee Dynamics

## 2.1 Q1: Implement Astrobee Dynamics

The dynamics of the Astrobee are governed by the Newton-Euler equations:

$$\dot{p} = v \tag{2.1}$$

$$\dot{v} = \frac{1}{m} R(q) f \tag{2.2}$$

$$\dot{q} = \frac{1}{2} \Omega(q) \omega \tag{2.3}$$

$$\dot{\omega} = J^{-1} \left( t - \omega \times J\omega \right) \tag{2.4}$$

Where $p$ is the position, $v$ is the velocity, $q$ is the unit quaternion for attitude, and $\omega$ is the angular velocity. The functions $R(q)$ and $\Omega(q)$ represent the rotation matrix and quaternion kinematics, respectively. The inputs to the system are the force $f$ and torque $t$.

The function implementing these dynamics is defined in the file `astrobee_dynamics_quat.py`. The correctness of the implementation is verified by the function `abee.test_dynamics()`, which confirms that the dynamics adhere to the model.

## 2.2 Q2: Static Setpoint Tracking Test

In this step, we aim to design a Nonlinear MPC that tracks a static setpoint for the Astrobee. The cost function is defined as (2.5), where $e_t$ is the tracking error, and $Q$, $R$, and $P$ are the cost matrices for the state and control inputs. The state and control limits are set using `abee.get_limits()`.

$$\min_{u_{t+*|t}} \sum_{k=0}^{N-1} e_{t+k|t}^T Q e_{t+k|t} + u_{t+k|t}^T R u_{t+k|t} + e_{t+N|t}^T P e_{t+N|t} \tag{2.5}$$

The tuning parameters are adjusted in the `tuning.yaml` file to optimize the performance of the NMPC.

In the process of tuning the MPC controller for the Astrobee robot, we evaluated three sets of control parameters labeled **P1**, **P2**, and **P3**. Each set of parameters affects the system's performance in terms of convergence speed, control input aggressiveness, and the ability to reach the desired state, including both position and attitude.

Evaluation of Parameter Sets:

**Parameter Set P1**

This parameter set results in aggressive control inputs that drive the system to converge quickly to the desired state. However, the aggressiveness may lead to high actuator usage and possible overshooting, which is not ideal for smooth operation.

**Astrobee Control Inputs P1**

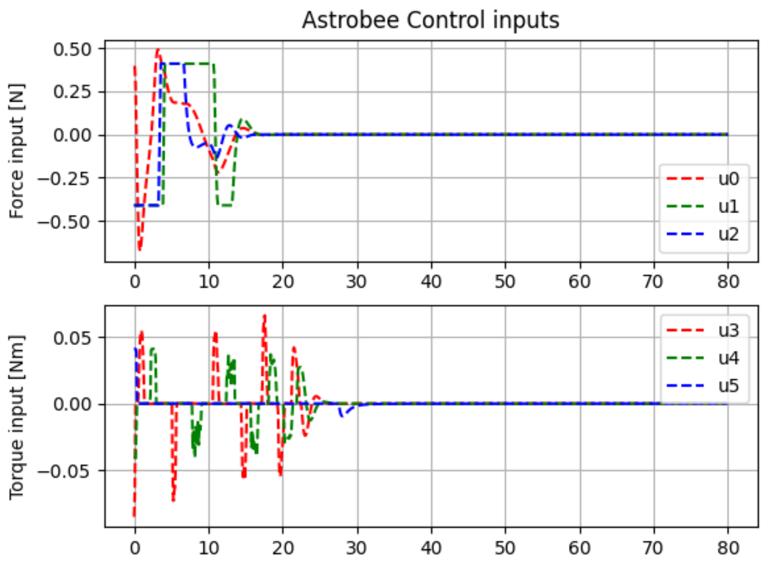

Figure 2.1: Control Inputs with Parameter Set **P1**

**Astrobee States P1**

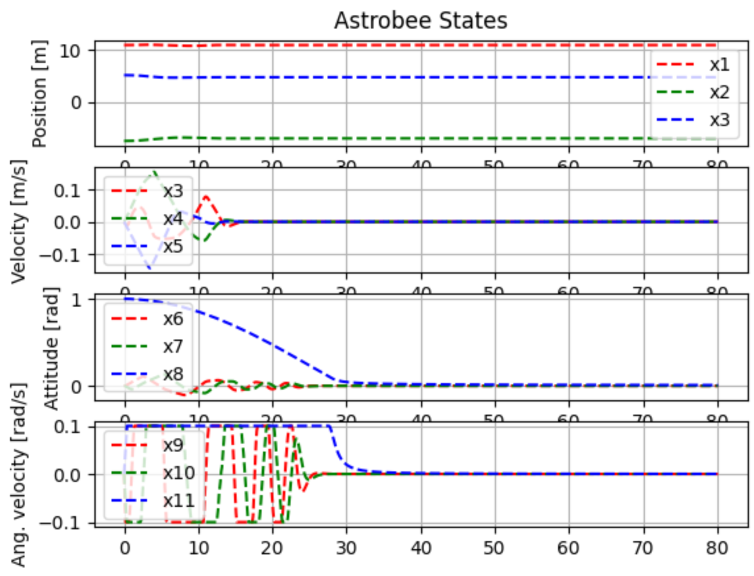

Figure 2.2: States with Parameter Set **P1**

**Parameter Set P2**
Parameter set **P2** provides the smoothest control inputs among the three. It reduces the aggressiveness of the control actions, leading to a smoother trajectory. While it guarantees convergence for the position and velocity in almost the same time as **P1**, it does not guarantee convergence to the desired attitude 'x8' within the required time frame.

**Astrobee Control Inputs P2**

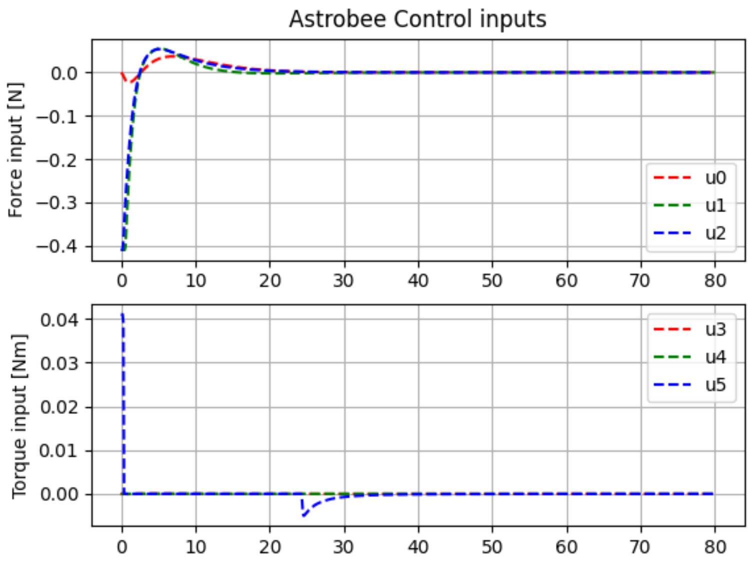

Figure 2.3: Control Inputs with Parameter Set **P2**

**Astrobee States P2**

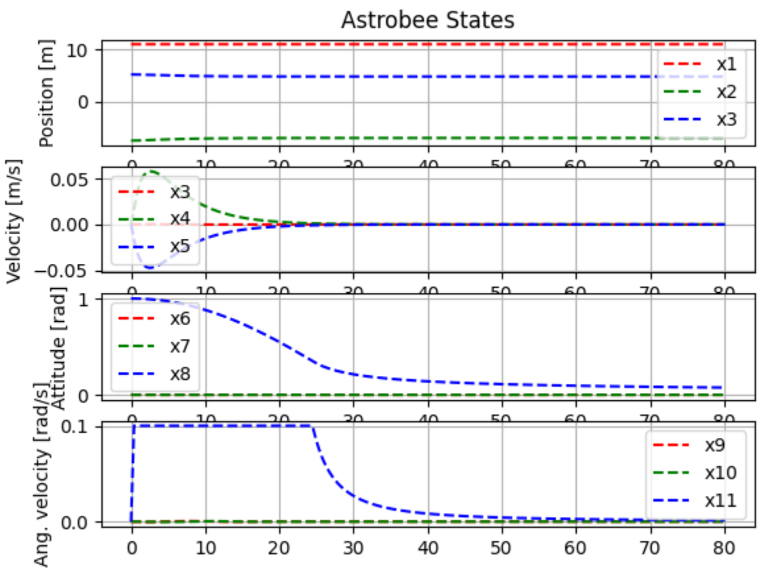

Figure 2.4: States with Parameter Set **P2**

**Parameter Set P3**

This set offers an approach with quite aggressive control inputs that drive the system to converge. It ensures convergence to the desired state within an acceptable time frame while having lower actuator usage with respect to set **P1**.

**Astrobee Control Inputs P3**

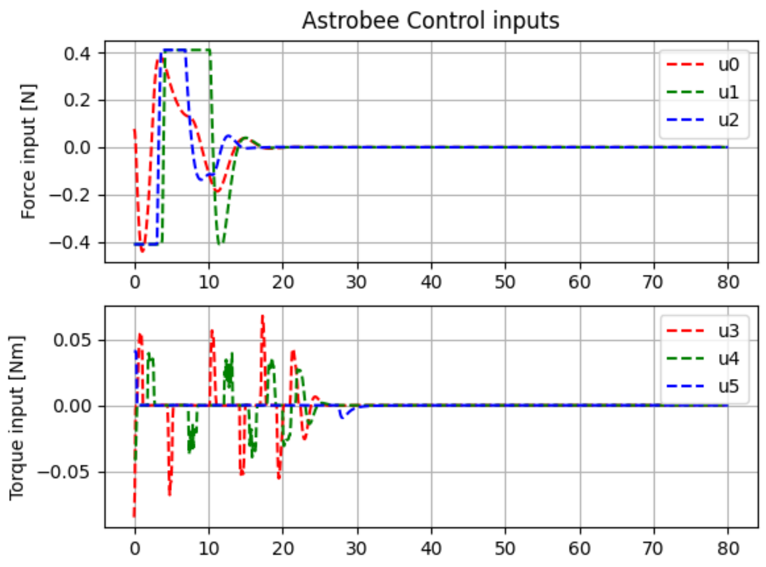

Figure 2.5: Control Inputs with Parameter Set **P3**

**Astrobee States P3**

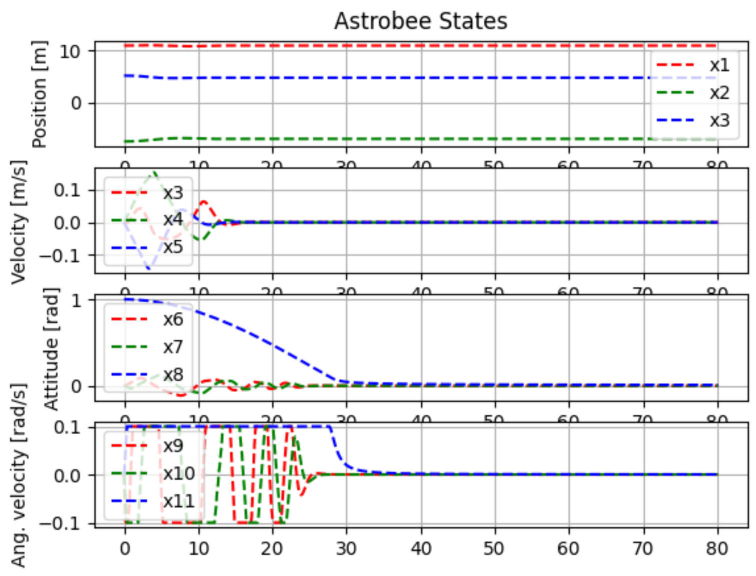

Figure 2.6: States with Parameter Set **P3**

In conclusion, parameter set **P3** is selected for implementation in our MPC controller due to its ability to guarantee convergence in both position and attitude while maintaining reasonable control input aggressiveness. This choice ensures that the Astrobee can perform its tasks effectively without compromising on performance or safety.

# Chapter 3

# NMPC for Time-Varying Reference

## Q3: Time-Varying Reference Tracking

We modify the NMPC to track a time-varying reference trajectory. This is achieved by updating the reference state $x_{\text{ref}}$ at each time step. The recorded trajectory of Honey is used as the reference, and the function `tracking_ctl` is modified to include this trajectory in the optimization.

The controller successfully tracks the trajectory, minimizing the tracking error across all state variables.

To implement the logic for the desired reference at each time step, we first define x_ref in the following way:

```
x_ref = ca.MX.sym('x_ref', self.Nx*(self.Nt+1),)
```

This variable contains $13 \times (N + 1)$ elements, since our full-state dimension is 13 and our horizon is $N$ steps.

Then, in the for loop, for every horizon step, we choose the corresponding element of the reference in the following way:

```
x_r = x_ref[t*self.Nx:(t+1)*self.Nx]
```

From the graph, it can be observed that the optimal choice between the three sets of matrices is the second one. Regarding the input, the simulation with $P2$ allows us to use a more conservative input without aggressive behavior.

In the case of $P1$, we observe a peak in all three input forces, reaching almost $0.35\,N$. With $P3$, the situation gets worse, with force peaks for $u_1$ and $u_2$ reaching $0.4\,N$. Using $P2$ minimizes the force overshoot to less than $0.1\,N$. Additionally, we manage to limit the maximal torque overshoot to around $0.01Nm$.

Having a less aggressive control results in smoother state convergence, but requires a longer time for full convergence. For position, convergence to zero with $P2$ takes almost twice the time compared to the $P1$ and $P3$ sets.

Through simulation, it is evident that the parameter set $P2$ strikes an optimal balance between stability and responsiveness. While it ensures minimal force overshoot and controlled torque, it requires a longer time for state convergence.

## Astrobee Control Inputs P1

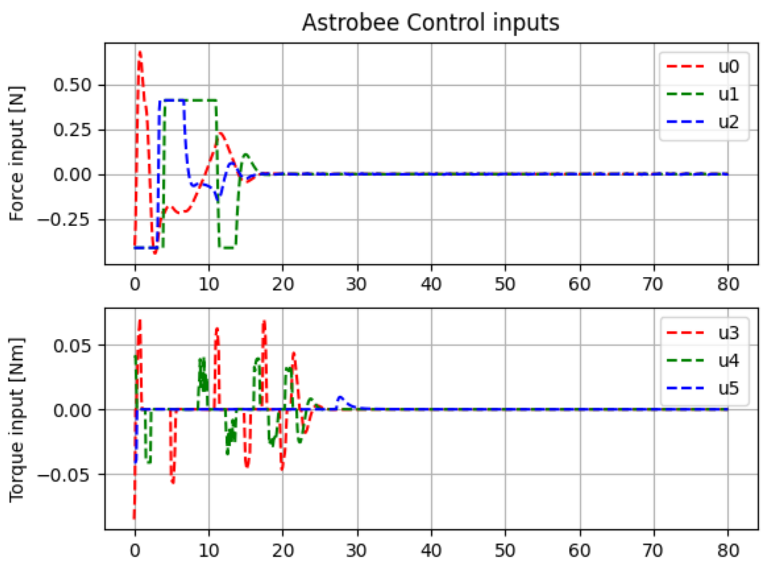

Figure 3.1: Control Inputs with Parameter Set **P1**

## Astrobee States P1

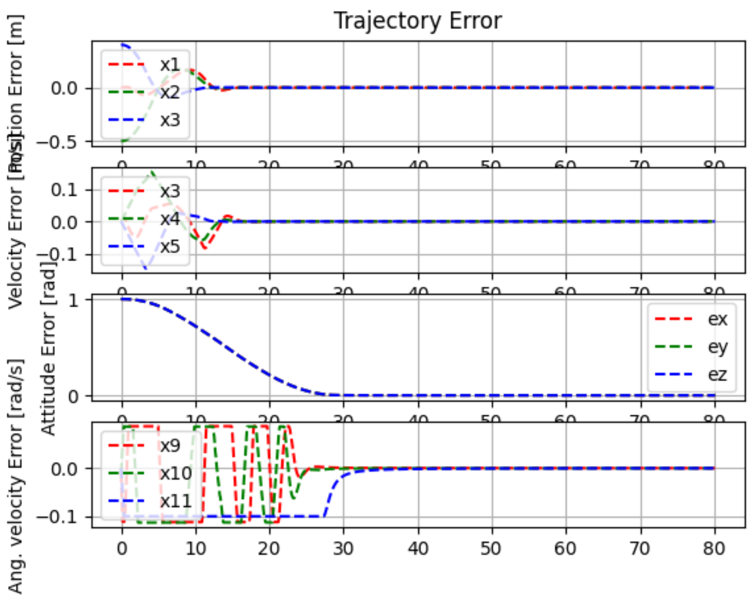

Figure 3.2: States with Parameter Set **P1**

## Astrobee Control Inputs P2

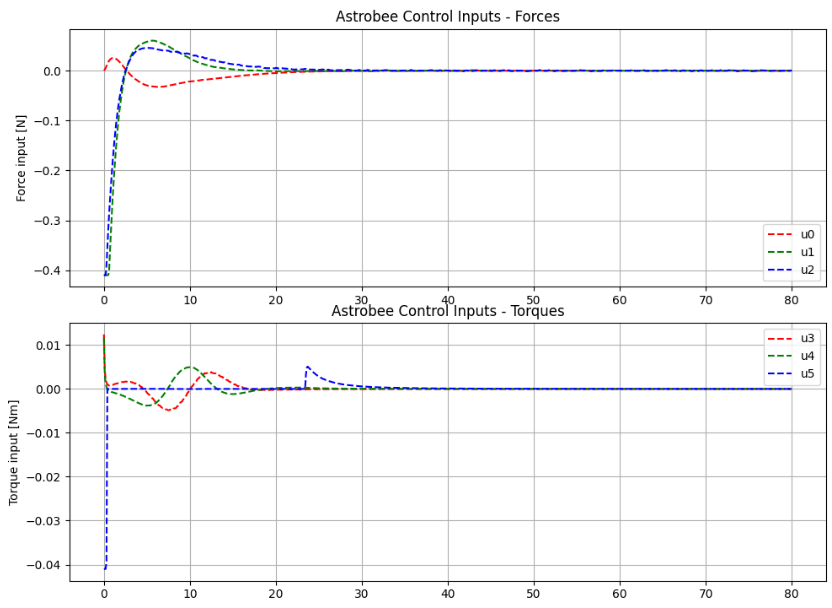

Figure 3.3: Control Inputs with Parameter Set **P2**

## Astrobee States P2

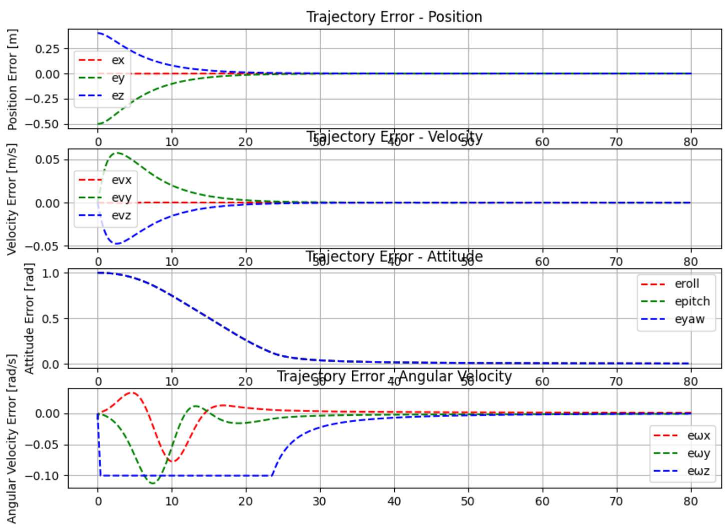

Figure 3.4: States with Parameter Set **P2**

**Astrobe Control Inputs P3**

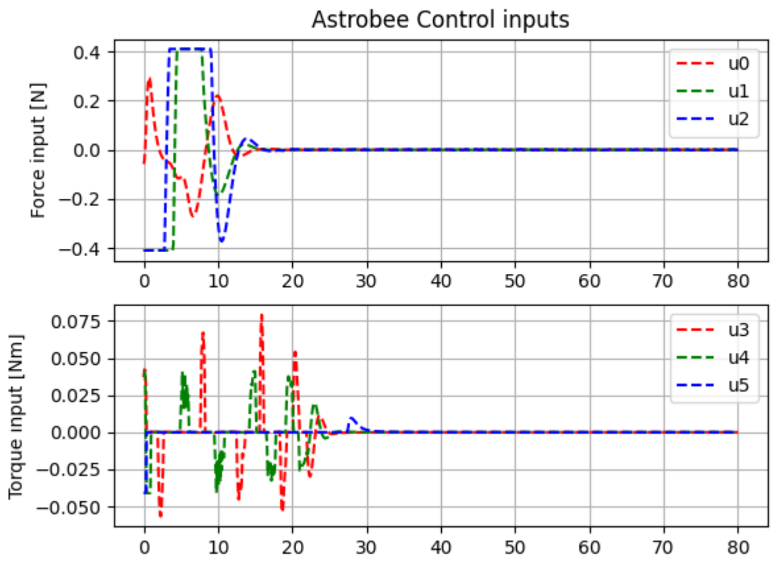

Figure 3.5: Control Inputs with Parameter Set **P3**

**Astrobee States P3**

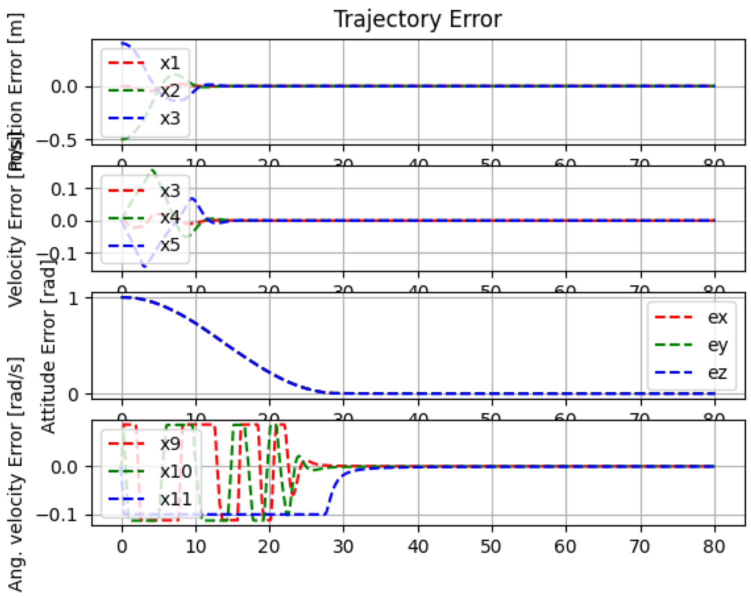

Figure 3.6: States with Parameter Set **P3**

## Q4: Forward Propagation Law

Since the future location of the target is not always given, it is necessary to model the dynamics of the target movement as well. These dynamics are modeled according to (3.1), (3.2), (3.3) and (3.4), where $\bar{v}$ and $\bar{\omega}$ are constant velocity estimates.

$$\dot{\mathbf{p}} = \bar{\mathbf{v}} \tag{3.1}$$

$$\dot{\mathbf{q}} = \frac{1}{2}\Omega(\mathbf{q})\bar{\omega} \tag{3.2}$$

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \dot{\mathbf{p}} \cdot h \tag{3.3}$$

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \dot{\mathbf{q}} \cdot h \tag{3.4}$$

After the first-order integration in (3.3) and (3.4) the quaternions are normalized according to (3.5).

$$\mathbf{q}_{k+1} = \frac{\mathbf{q}_{k+1}}{||\mathbf{q}_{k+1}||} \tag{3.5}$$

Additionally, since the Astrobee should first approach the target before it tries to track the motion of the target (3.6) is added, where $r$ is the radius from the target where motion tracking should begin.

$$\mathbf{p}_B = R_{[:,0]}(\mathbf{q}) \cdot r \tag{3.6}$$

The forward propagation function is implemented in `forward_propagate` and tested using `abee.test_forward_propagation()`.

The results of the simualtion for the predefined tuning parameters $P_1$, $P_2$ and $P_3$ are shown in Fig. 3.7, 3.8, 3.9, 3.10, 3.11 and 3.12.

**Astrobe States P1**

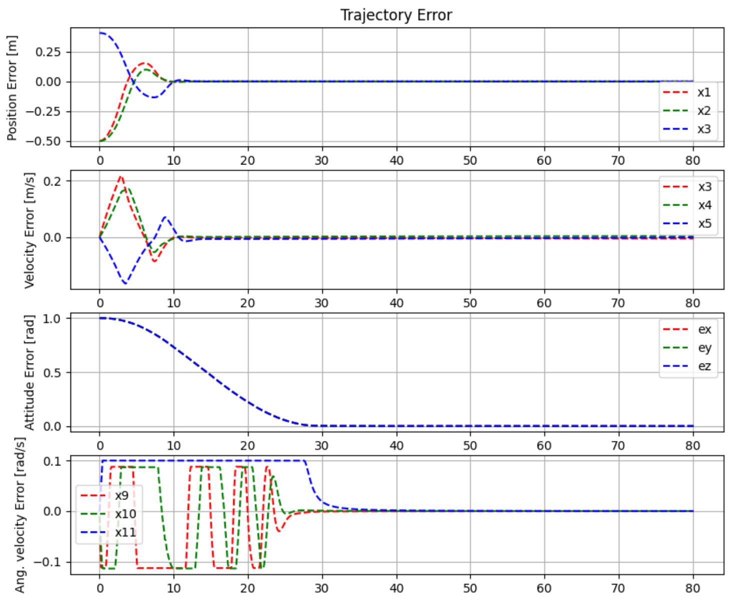

Figure 3.7: Astrobee state simulation using tuning configuration P1

## Astrobe Control Inputs P1

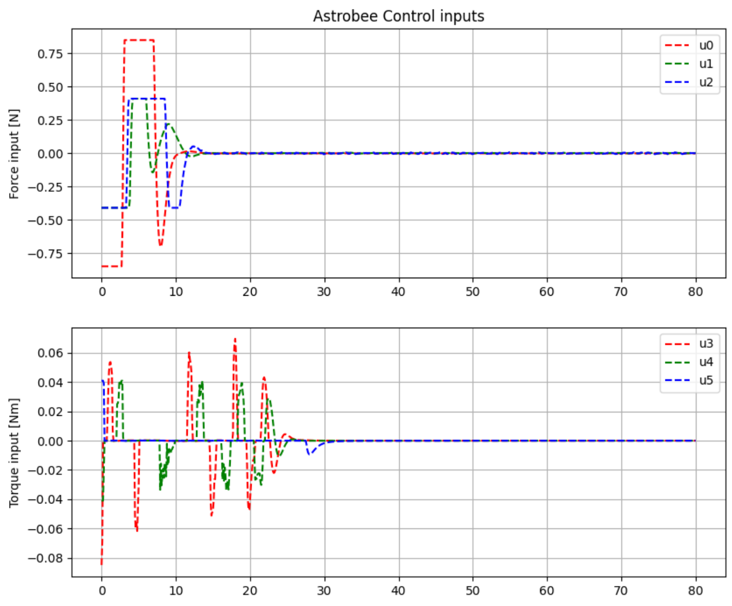

Figure 3.8: Astrobee input simulation using tuning configuration P1

## Astrobe States P2

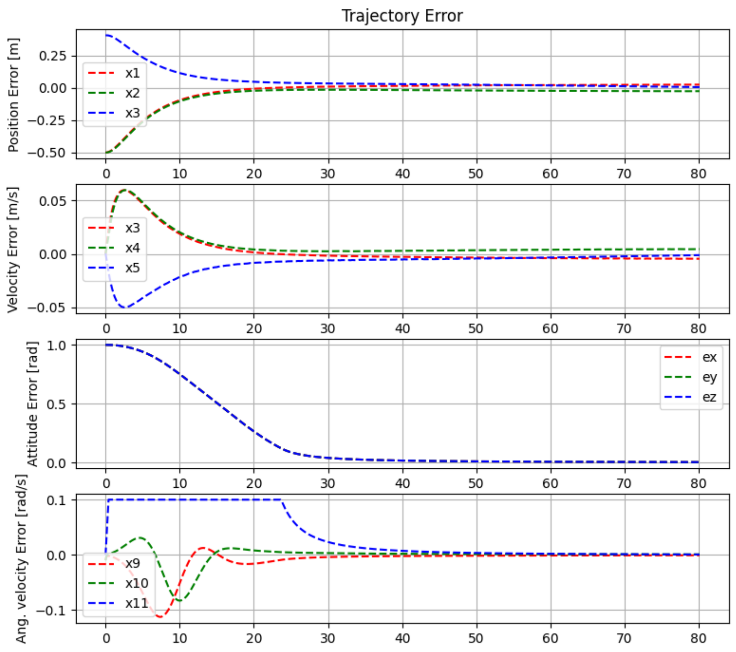

Figure 3.9: Astrobee state simulation using tuning configuration P2

## Astrobe Control Inputs P2

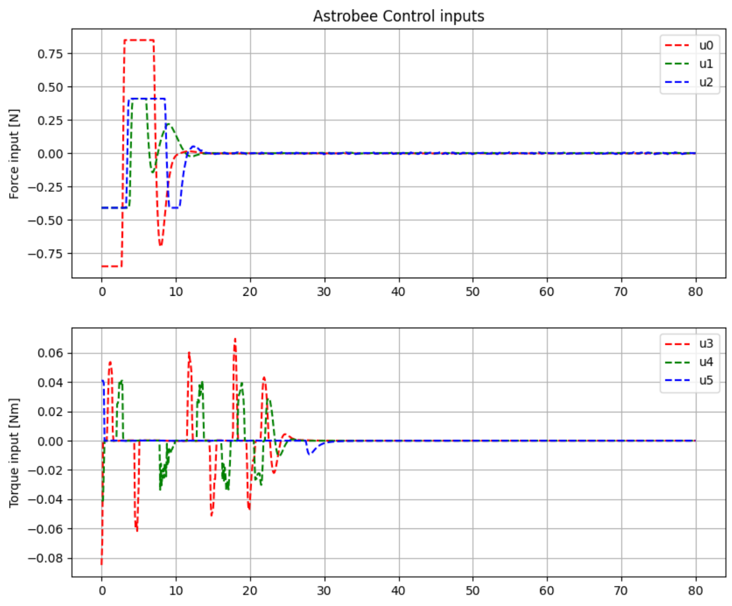

Figure 3.10: Astrobee input simulation using tuning configuration P2

## Astrobe States P3

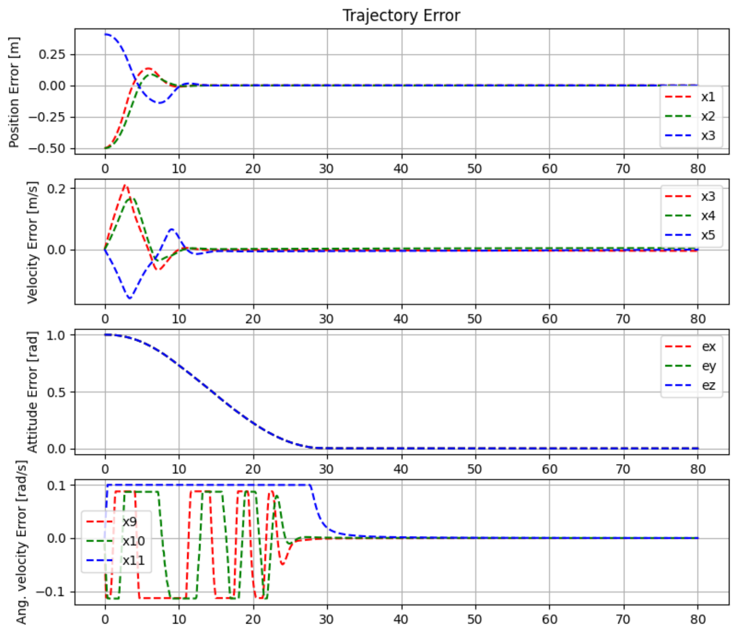

Figure 3.11: Astrobee state simulation using tuning configuration P3

## Astrobe Control Inputs P3

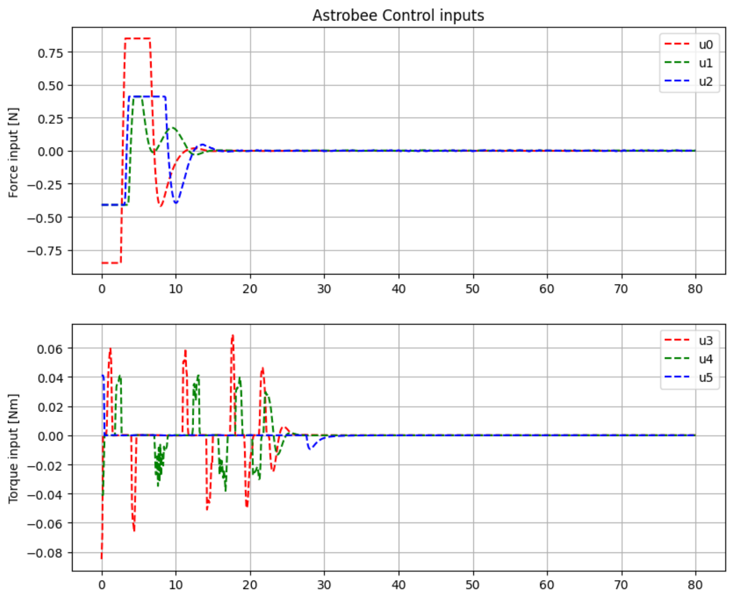

Figure 3.12: Astrobee input simulation using tuning configuration P3

# Chapter 4

# Competition

We implemented the scoring system by defining the parameters related to the MPC solver in the `mpc.py` file, while handling the parameters associated with error metrics in the `simulation.py` file. The tuning process involved adjusting our matrices to achieve the highest possible score. By experimenting with different sets of values, we focused on minimizing the convergence time (`cvg_t`), which is the time it takes for the controller to bring the system within 5 cm of the target trajectory and 10 degrees of the desired attitude. Additionally, we monitored `ss_p`, the steady-state position error, and `ss_a`, the steady-state attitude error. After optimizing these parameters, the final values obtained were:

- **Q**: [150, 150, 150, 10, 10, 10, 150, 150, 150, 10, 10, 10]
- **R**: [7, 7, 7, 35, 35, 35]
- **P_mult**: 100