Assignment 2

Lorenzo Fici, Hannes Gustafsson, Davide Nascivera

September 2024

1 Introduction to the design task

Since in Assignment 1 we did a checkout of 1 Degree of Freedom (DoF) movement, in this assignment we will focus on the movement with 3-DoF. This means that the robot will be able to translate along two directions, and rotate along one. The state variable $\mathbf{x}$ concatenates the position and velocity along the X and Y axis, as well as the rotation angle θ and corresponding angular velocity ω around the Z-axis. Accordingly, $\mathbf{x}$ is defined as

$$x = \begin{bmatrix} p_X & p_Y & v_X & v_Y & \theta & \omega \end{bmatrix}^T$$

Moreover, the control input $\mathbf{u}$ concatenates the forces f_X and f_Y, along the X and Y axis, respectively, as well as the torque τ_Z around the Z axis, according to

$$u = \begin{bmatrix} f_X & f_Y & \tau_Z \end{bmatrix}^T$$

The linearized model, valid around a rotation $\theta \approx 0$, is given by

$$\dot{x}(t) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 \\ 0 & \frac{1}{m} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{I_z} \end{bmatrix} \mathbf{u}(t)$$

where $m = 20.9$ kg is the mass of the Astrobee with the air-carriage and $I_z = 0.2517\ \text{kg} \cdot \text{m}^2$ is the inertia of the Astrobee and air-carriage around the Z-axis (around which the system is allowed to rotate).

2 Questions

2.1 Q1

We implemented the model in the function `cartesian_ground_dynamics` as following:

```
def cartesian_ground_dynamics(self):
        # Jacobian of exact discretization
        self.n = 6
        self.m = 3

        self.Ac = np.array(
            [
                [0, 0, 1, 0, 0, 0],
                [0, 0, 0, 1, 0, 0],
                [0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0, 1],
                [0, 0, 0, 0, 0, 0],
            ]
        )
        self.Bc = np.array(
            [
                [0, 0, 0],
                [0, 0, 0],
                [1 / self.mass, 0, 0],
                [0, 1 / self.mass, 0],
                [0, 0, 0],
                [0, 0, 1 / self.inertia],
            ]
        )
        return self.Ac, self.Bc
```

2.2 Q2

In the *task2.py* file we used the methods *c2d* and *set_discrete_dynamics* to create and define the discrete-time dynamics for Bumble:

```
# Get the system discrete-time dynamics
A, B = bumble.cartesian_ground_dynamics()
Ad, Bd, Cd, Dd = bumble.c2d(A, B, np.eye(6), np.zeros((6,3)))
bumble.set_discrete_dynamics(Ad, Bd)
```

2.3 Q3

We completed the constructor of the class *FiniteOptimization* specifying the rendezvous constraint after the desired time. The constraints are applied for every step $t > self.Ntr$ where $self.Ntr$ is the desired step for rendezvous.

```
if ref_type == "2d":
    x_ref = xr[t]
```

```
con_ineq.append((x_ref[0:2] - x_t[0:2]) <= self.pos_tol)
con_ineq.append((x_ref[0:2] - x_t[0:2]) >= -self.pos_tol)

con_ineq.append((x_ref[2] - x_t[4]) <= self.att_tol)
con_ineq.append((x_ref[2] - x_t[4]) >= -self.att_tol)
```

Constraints are applied on the position and on the angle θ.

2.4 Q4

We were tasked with solving the finite optimization problem using the reference state x_r from Honey and the initial state $x = [0.1, 0, 0, 0, 0.01, 0]$ for Bumble. The function `honey.set_trajectory` generated a 30-second trajectory. We then used `honey.get_trajectory(t_start=25.0)` to obtain the last 5 seconds of this trajectory. Our objective was for Bumble to track this segment and rendezvous with Honey.

Figure 1 shows the simulation results for three different metrics of interest. Figure 1a and 1b shows the Prediction vs. reference and State vs. Reference, respectively. Once fed the reference after $t = 25$, the Astrobee is able to follow the reference with good precision. Figure 1c shows the tracking error, which goes to zero quickly when the controller kicks in.

2.5 Q5

When we activate the flag for the broken thruster, his behavior is modelled summing a random value in the input of the corresponding thruster:

```
if self.broke_thruster:
    u[1] = u[1] + np.random.uniform(-0.07, 0.07, (1, 1))
```

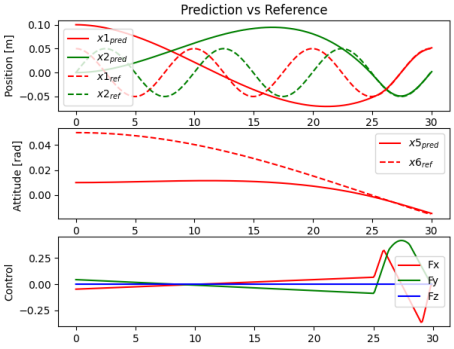
This action causes one of the input to oscillate as shown in Figure 2.

As a consequence we observe some steady state error on p_y. We expected to find the position more shaky but probably it is not like this because the inertia of the Astrobee is quite big considering it weights almost 21 kg and this attenuates the oscillations. We also tried to increase the noise, from the original -0.07 to 0.07 to the new bigger values -0.7 to 0.7. The result can be seen in Figure 3.

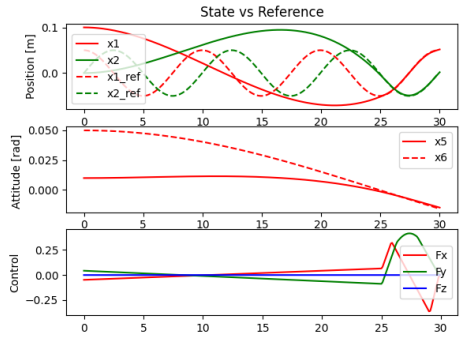
We can conclude that the steady state error grows as the uncertainty on the thruster increase.

2.6 Q6

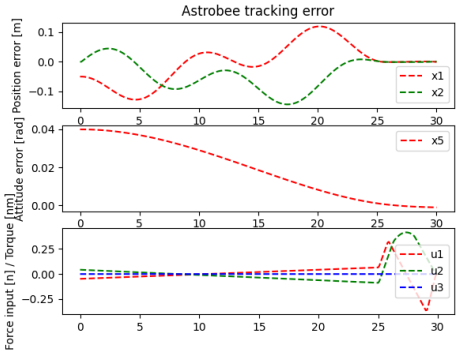
To counteract this effect, simply implementing an integral action is neither possible nor sufficient, as we are dealing with an open-loop control based on an optimization problem. We hypothesize that a potential solution could be to periodically recompute the optimization problem. The more frequently the policy is computed, the better control we will have, which in turn would help reduce the error.



(a) Q4, Prediction vs Reference



(b) Q4, State vs Reference



(c) Q4, Tracking error

Figure 1: Summary of Q4 results

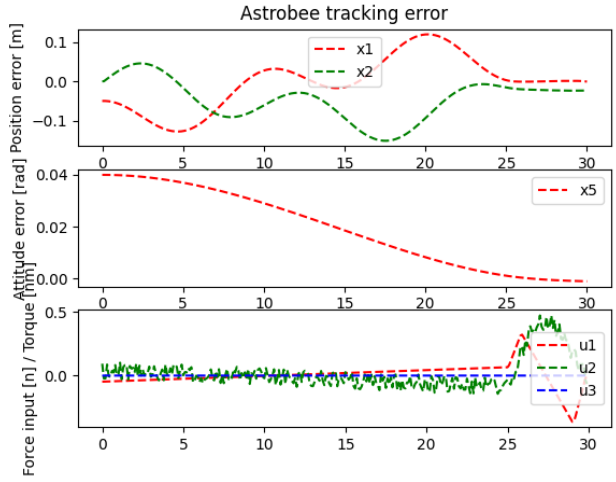


Figure 2: Q5, Broken Thruster

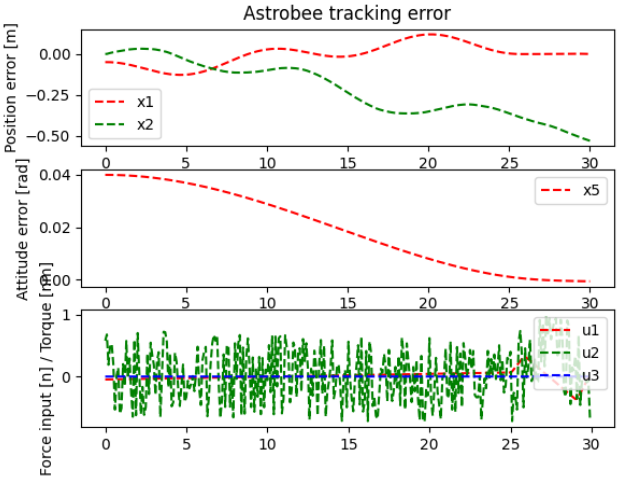


Figure 3: Q5, Broken Thruster with increased noise

2.7 Q7

Here, we were tasked with extending the model from 2D to 3D. This is done by defining the new state vector to be

$$x = \begin{bmatrix} p_X & p_Y & p_Z & v_X & v_Y & v_Z & \theta & \omega \end{bmatrix}^T$$

with the control input

$$u = \begin{bmatrix} f_X & f_Y & f_Z & \tau_Z \end{bmatrix}^T$$

.

The linearized model, valid around a rotation $\theta \approx 0$, is extended to

$$\dot{x}(t) = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & \frac{1}{m} & 0 & 0 \\ 0 & 0 & \frac{1}{m} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} \end{bmatrix} \mathbf{u}(t)$$

.

For the simulation, the initial vector was set to $x_1 = 1.0$ and the other states at zero. The results are shown in 4, which clearly shows that the Astrobee is able to control itself to the desired reference state for $t > 35.0$.

Regarding the time-to-simulation, the solver was quicker by a factor of 5x (10 s for the 2D-case, and around 2 s for the 3D-case). This was unexpected, since one would think that extending the model creates larger matrices, and thereby more multiplications to perform for the solver.

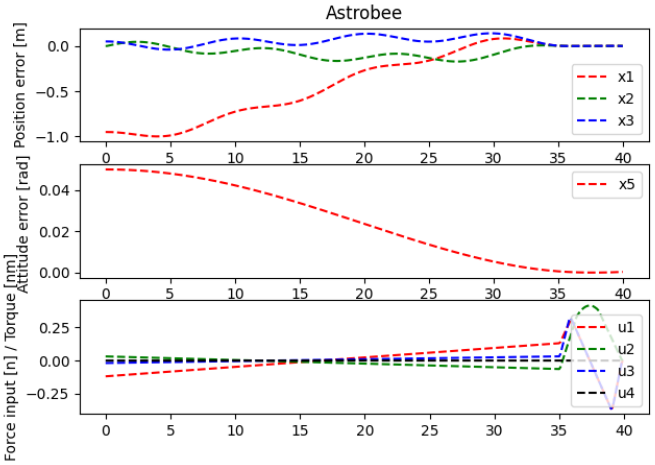


Figure 4: Q7, Tracking error for the 3D model

3 Conclusion

In this assignment, we successfully extended the 2-DoF motion control system from the extra point of Assignment 1 to a 3-DoF system, enabling the Astrobee to translate along two axes and rotate around one axis. By using the linearized model and discretizing the dynamics, we were able to simulate and control the movement of the Astrobee, both under nominal conditions and when subjected to a faulty thruster. Our implementation demonstrated robust tracking of reference trajectories, with the system quickly minimizing tracking errors.

However, comprehending the challenge of handling disturbances, such as a broken thruster, made us realise that in open-loop control, is not always easy to counter this kind of issues with an easy implementable integral action, but are needed some more complex actions.

Furthermore, we extended the system to 3D, which increased the complexity of the model but surprisingly improved the computational performance during simulations. The system maintained excellent tracking capability in 3D space, demonstrating the robustness and scalability of our control design approach.

In conclusion, this project has provided us with a deeper understanding of the dynamics of multi-DoF control systems and has prepared us for future work in dynamic optimization and advanced control strategies for robotic systems.