RRT*-GBO for Perception-aware Trajectory Optimization using NASA's Astrobee Robot

Scientific Thesis for the procurance of the degree M.Sc.
from the Department of Electrical and Computer Engineering at the Technical University of Munich.

Supervised by

Univ.-Prof. Dr.-Ing./Univ. Tokio habil. Martin Buss
PD Dr.-Ing. habil. Marion Leibold
Chair of Automatic Control Engineering

Dr.-Ing. Roberto Lampariello
Caroline Specht, M.Sc.
German Aerospace Center (DLR)

Submitted by

cand. M.Sc. Victor Kowalski Martins
Zieblandstr. 35
80798 Munich
+49 1575 284 1574

Submitted on Munich, 03.08.2024

TECHNISCHE UNIVERSITÄT MÜNCHEN
LEHRSTUHL FÜR STEUERUNGS- UND REGELUNGSTECHNIK
UNIV.-PROF. DR.-ING./UNIV. TOKIO HABIL. MARTIN BUSS
DR.-ING. MARION LEIBOLD

February 5, 2024

M A S T E R ' S T H E S I S
for
Victor Kowalski Martins
Student ID 10770482, Degree Mechatronics and Robotics

RRT*-GBO for Perception-aware Trajectory Optimization using NASA's Astrobee Robot

Problem description:

On-orbit servicing (OOS) denotes the maintenance, repair, or enhancement of satellites, satellite parts, or other structures while they are in space. This thesis aims to contribute to the goal of enabling a robotic system to autonomously reach a target satellite for performing an OOS task.
The focus of the thesis lies in the motion planning task. The RRT*-GBO method [12] is employed for finding an optimal reference trajectory for a given scenario. This method extends the RRT* search method [7] by solving a boundary value problem (BVP) for each edge construction via gradient-based optimization (GBO).
The edges are parameterized as B-Splines, and the optimization formulation considers kinodynamic constraints and collision avoidance. The goal is to find the B-Spline parameters that minimize the system's mechanical energy while maximizing a perception metric. This metric is the density of visual landmarks in the environment captured by the robot's vision system during its trajectory. Such landmarks are also known as features.
During the actual execution of the OOS task, the robot will follow the reference trajectory provided by the RRT*-GBO method. However, it is necessary to update the robot's trajectory at consecutive time steps to account for possible disturbances and dynamic conditions of the environment. That involves solving shorter GBO problems in real-time, which requires the GBO solution to be found in an acceptable time frame. An issue is that the perception metric computation is time-consuming. Therefore, this thesis aims to achieve an efficient implementation of this metric.
The Astrobee robot from NASA [1] is the robotic platform to validate our solution. The DLR's OOS-SIM (on-orbit servicing simulator) [4] is used as an on-ground facility to simulate the orbital scenario.
This thesis aims to improve the results from the ROAM/TumbleDock Astrobee experiment campaign, which was a collaboration between MIT and DLR. The TRACE pipeline was developed to perform all the steps of an autonomous rendezvous procedure [2, 11, 3]. It stands for Tumbling Rendezvous via Autonomous Characterization and Execution and includes estimating the target's state, motion planning, localization, and robust control. The present work focuses on improving the motion planning step.
The mentioned pipeline employed a Monte Carlo approach similar to the one described in [8, 11] to find globally optimal trajectories in the motion planning step. This thesis seeks to find if the RRT*-GBO method presented in [12] provides better results than the Monte Carlo approach. However, the RRT*-GBO method introduces a new set of parameters to be tuned, such as the minimal distance of a new node to its neighbors and the size of the neighborhood. These should be explored to find combinations that yield good results for the current application.

In [2, 11, 3, 12], the robot's orientation is either not included as an optimization variable or uniquely defined by the constraints. For the present work, it is important to explore different possible orientations of the robot. A reason for that is that the perception metric varies according to the robot's field of view, which is a function of its orientation.
The perception metric to be implemented will be based on the visibility of landmarks of the environment [9, 5]. This metric is expected to improve the TRACE pipeline's localization step and must be computed in an acceptable time.

Tasks:

- Compare RRT*-GBO to a Monte Carlo approach [8, 11] for the search of globally optimal solutions.
- Investigate tuning the GBO solver [6], B-Spline, and RRT* parameters.
- Parameterize the robot's orientation to include it as an optimization variable.
- Collect images from OOS-SIM.
- Use NASA's tool [10] to generate feature maps.
- Implement generated OOS-Sim feature map into the RRT*-GBO framework, to be evaluated by the perception metric.
- Implement Astrobee-style localization [NAS20] on the OOS-SIM.
- Use RRT*-GBO to plan a reference trajectory on the OOS-SIM.
- Come up with an efficient C++ implementation of the perception metric.
- Test the real-time capability of the GBO solver on the OOS-SIM.

Bibliography:

[1] K. Albee, M. Ekal, and C. Oestreich. A brief guide to astrobee's flight software. 2020.
[2] K. Albee, C. Oestreich, C. Specht, A. Terán Espinoza, J. Todd, I. Hokaj, R. Lampariello, and R. Linares. A robust observation, planning, and control pipeline for autonomous rendezvous with tumbling targets. *Frontiers in Robotics and AI*, 8:641338, 2021.
[3] K. Albee, C. Specht, H. Mishra, C. Oestreich, B. Brunner, R. Lampariello, and R. Linares. Autonomous rendezvous with an uncertain, uncooperative tumbling target: The tumbledock flight experiments. In *Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*, 2022.
[4] J. Artigas, M. De Stefano, W. Rackl, R. Lampariello, B. Brunner, W. Bertleff, R. Burger, O. Porges, A. Giordano, C. Borst, et al. The oos-sim: An on-ground simulation facility for on-orbit servicing robotic operations. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.
[5] L. Bartolomei, L. Teixeira, and M. Chli. Perception-aware path planning for uavs using semantic segmentation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.
[6] S. G. Johnson. The nlopt nonlinear-optimization package. https://github.com/stevengj/nlop.
[7] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.
[8] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, and J. Peters. Trajectory planning for optimal robot catching in real-time. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.
[9] V. Murali, I. Spasojevic, W. Guerra, and S. Karaman. Perception-aware trajectory generation for aggressive quadrotor flight using differential flatness. In *American Control Conference (ACC)*, 2019.
[10] NASA. Astrobee robot software. https://github.com/nasa/astrobee.
[11] C. Specht, A. Bishnoi, and R. Lampariello. Autonomous spacecraft rendezvous using tube-based model predictive control: Design and application. *Journal of Guidance, Control, and Dynamics*, pages 1–19, 2023.
[12] S. Stoneman and R. Lampariello. Embedding nonlinear optimization in rrt* for optimal kinodynamic planning. In *IEEE Conference on Decision and Control*, 2014.

Supervisor:	M. Leibold (TUM), R. Lampariello, C. Specht (DLR)
Start:	05.02.2024
Intermediate Report:	05.05.2024
Delivery:	05.08.2024

(M. Leibold)
Privatdozent

Abstract

Robots can be used for on-orbit tasks in outer space, such as interacting with satellites. A free-flying robot such as NASA's Astrobee should be able to autonomously navigate in space, but a common issue is that the robot's vision-based localization system fails if its camera does not see enough visually descriptive landmarks. That makes the robot lose track of its position and orientation, and consequently fail to execute a desired trajectory. To avoid this scenario, this thesis presents a trajectory planner that guarantees the visibility of descriptive features while maintaining energy efficiency and satisfying motion constraints from the robot's actuators and obstacles in the environment. The visibility consideration in the planning of a trajectory is referred to as perception-aware planning. The trajectories are optimized by exploring all six translation and rotation degrees of freedom of the SE(3) domain with the RRT*-GBO algorithm, which uses an effective sampling-based approach to find initial guesses that converge to globally optimal solutions. The time efficiency of the RRT*-GBO algorithm is highlighted in different scenarios. Additional time savings are obtained with an offline precomputation of the feature visibility metric for a given environment. Our solution is experimentally validated for a rendezvous task between two Astrobee robots in the International Space Station (ISS) and for a satellite capture task on DLR's On-Orbit Servicing Simulator (OOS-SIM). These two experiments demonstrate a significant improvement in the accuracy of the robot's localization system by planning perception-aware trajectories.

Contents

Chapter 1

Introduction

The United Nations Office for Outer Space Affairs (UNOOSA) lists over 13000 Objects Launched into Outer Space orbiting Earth in 2024 [fOSA]. Satellites are examples of those.
On-orbit servicing (OOS) includes inspection, maintenance, relocation, and de-orbiting tasks, which require physical access to these objects. The adverse outer space conditions motivate employing robots to perform on-orbit tasks as autonomously as possible. Fig. 1.1 illustrates that.

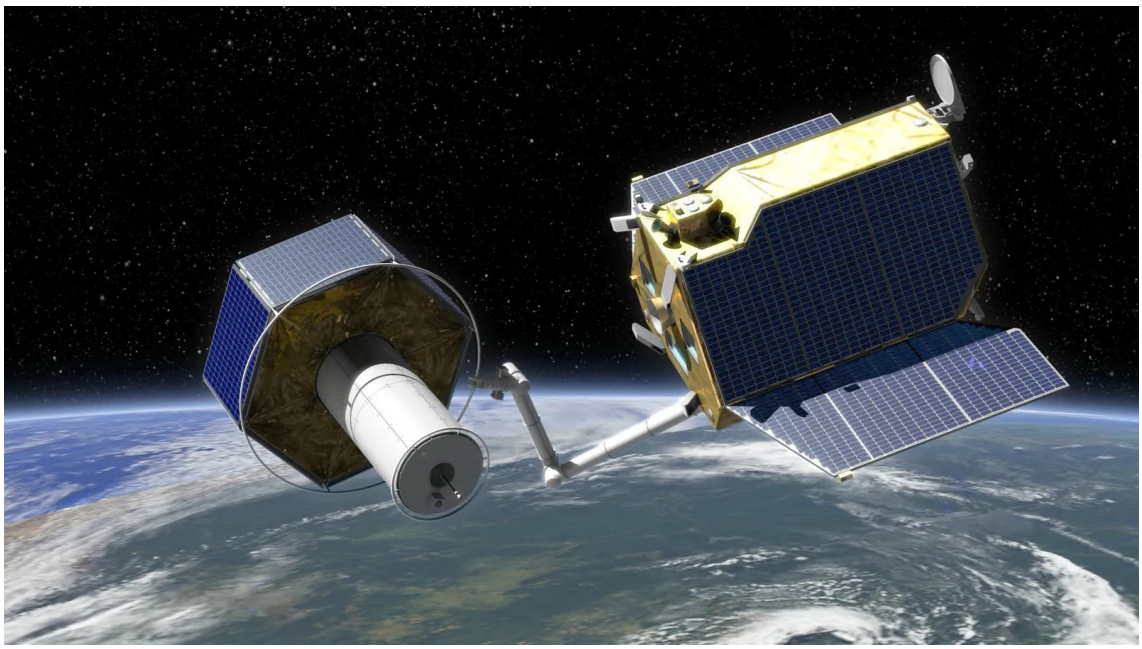


Figure 1.1: On-orbit servicing task showing a robot interacting with a satellite in space [Wol11].

Consider that a free-flying robot must interact with a nearby target satellite in space. The robot must plan a trajectory to optimally reach the satellite. Optimal-

ity often means minimal energy consumption while satisfying constraints such as the actuation capacity and avoidance of collisions. The robot's control system then commands the robot's actuators in a way that makes the robot follow the planned trajectory as accurately as possible. The actuation commands are computed as a function of the error between the real robot state and the desired robot state (from the planned trajectory) at each time step. The real robot state is, in practice, an estimative. For instance, the robot's position and orientation in the environment are commonly estimated via visual-based localization, which uses what the robot's camera sees to infer where the robot is. An estimation always involves errors whose magnitude determines the success or failure of the desired robot task. If the error in the localization estimation is too large, the robot may not reach the goal or even collapse with the target satellite and damage both. With that in view, the contribution of this thesis is a trajectory planner that optimizes the camera viewpoint for minimizing localization errors while maintaining energy efficiency.

1.1 Problem Statement

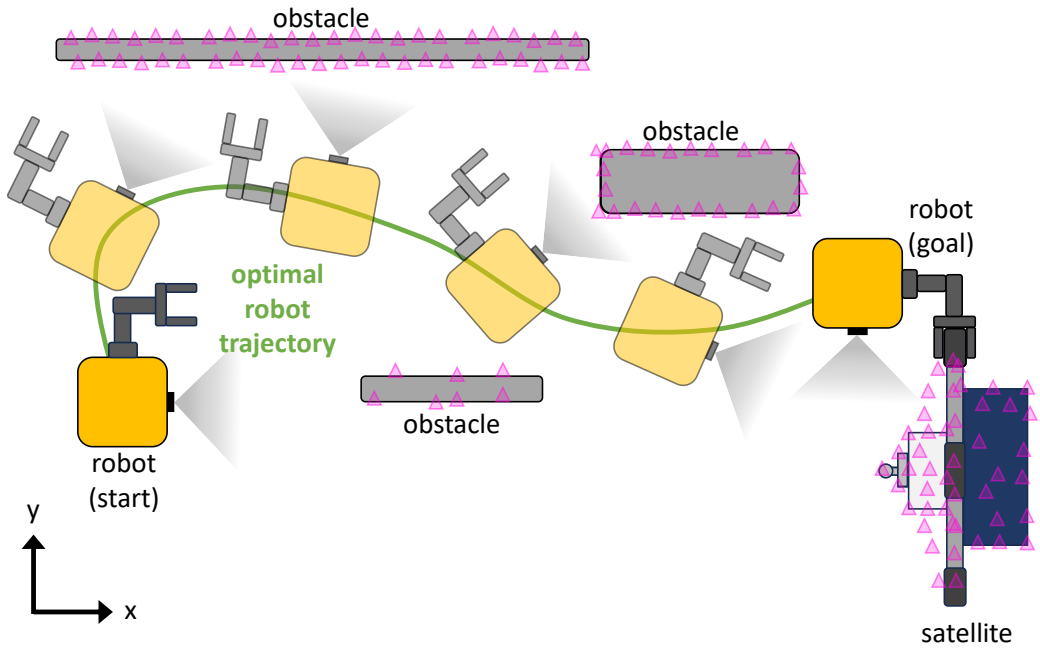


Figure 1.2: Motion planning problem. The magenta triangles are visual features that the robot uses for localization. Since the lowest (and closest) obtacle is poor in features, the planner quickly makes the robot look to regions with higher feature density.

Figure 1.2 illustrates the type of motion planning problem tackled by this work. A robot has to travel from a start state to a goal state in the best way possible. A state comprises the robot's pose (position and orientation) and its derivatives (velocity, acceleration, jerk, and so on). The trajectory to reach the goal must be feasible. Fundamentally, it means that the robot must not collide with any obstacles and that the motion constraints - including position, velocity, and actuation constraints - are respected. The quality of such a trajectory considers the number of relevant visual features the robot can keep in view while following the trajectory and the energy consumed by the actuation that the robot must exert.

1.2 Related Work

Astrobee is a free-flying robotic system developed by NASA to be a test bed for zero-gravity robotics research inside the International Space Station (ISS) [NASa, NASb]. Fig. 1.3 shows two Astrobees inside the ISS.

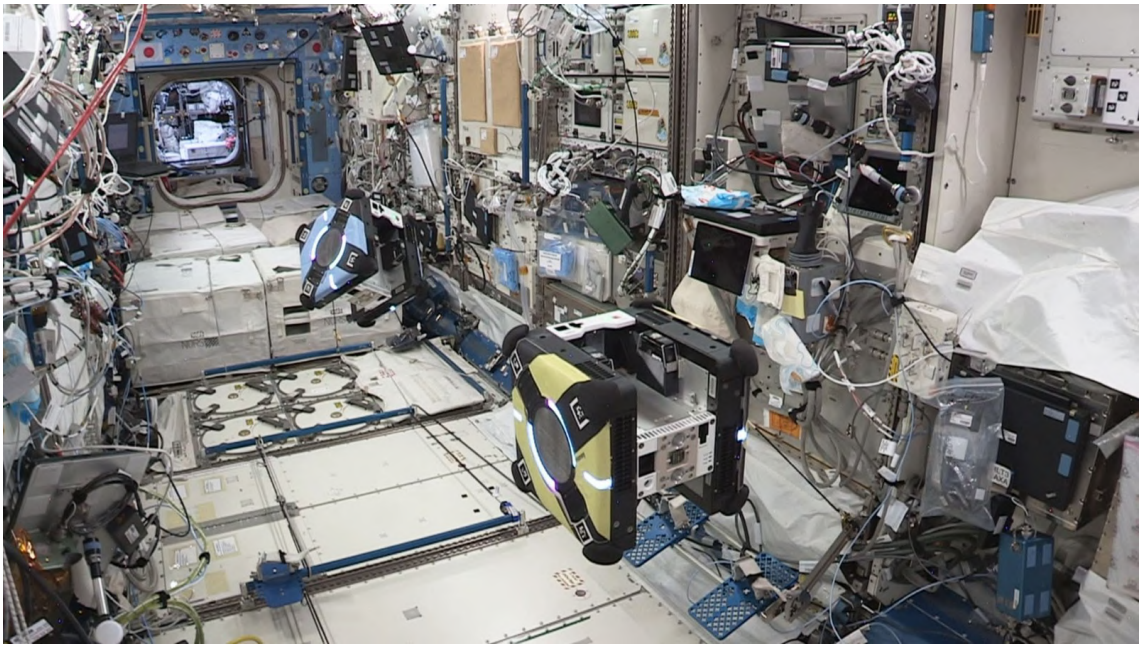


Figure 1.3: Two Astrobee robots inside the ISS [DLR]. Credit: NASA.

Albee et al. [ASM+22] uses the Astrobee robots as an experimental platform for the autonomous on-orbit servicing pipeline developed in [AOS+21, SBL23]. The pipeline comprises motion planning, control, and visual-based localization. Experimental results from [ASM+22] evidence large errors in the localization stage, impairing the task execution. In the experiments, the localization module employs the method from [CFM+16]. Soussan et al. [SKCS22] present Astroloc as an alternative method to improve Astrobee's localization accuracy.
Instead of targeting the localization method, Faraci [Far23] explores perception-aware motion planning for the Astrobee. Perception-aware planning [ZS18, SCSG19, TH22, CFD+17, FFLS18, MSGK19, BTC20] is the explicit inclusion of perception metrics in the planning stage of a robotic task. The metrics from [MSGK19, BTC20] are explored in this thesis. They model the visibility of visually descriptive regions along the planned robot trajectory. Applications of perception-aware planning in the literature are mainly focused on UAVs (Unmanned Aerial Vehicles), which makes its application to the Astrobee free-flyer a novelty. A clear difference is the domain of the planned tasks. While UAV orientations are usually constrained to a single yaw rotation on the horizontal plane, free-flyers can rotate in the complete domain of rotations SO(3).

Gradient-based nonlinear optimization [Sca85] is a common approach to perception-aware planning, whose downside is that the obtained result strongly depends on an initial guess. Sampling-based methods [KF11] evade that issue but can only guarantee optimality in infinite time (probabilistic completeness). Lampariello et al. [LNTC+11] presents a mixed approach employing a sampling-based Monte Carlo method to explore various initial guesses provided to a gradient-based optimizer. The RRT*-GBO algorithm [SL14] replaces Monte Carlo with an RRT* sampling method to locate promising initial guesses more efficiently.
Perception-aware planning usually requires a pre-computed map of a given environment, and evaluating a perception-optimized trajectory requires a localization estimator to be compared to the ground-truth trajectory. Simultaneous localization and mapping [TUI17, GKSB10] covers both requirements. The ORB-SLAM3 algorithm [CER+21] is a popular, state-of-the-art implementation of SLAM.

1.3 Structure of the thesis

To close Chapter 1, Section 1.4 gives an overview of the theoretical background that supports the solution developed in this thesis.
Chapter 2 details the design and implementation of the solution. Section2.1 formulates the trajectory optimization problem, introducing the energy cost and perception objective metrics, the motion constraints, and the gradient-based solver. Section 2.2 explains the RRT*-GBO algorithm, and Sec. 2.3 describes how SLAM is used in our solution.
Chapter3 experimentally evaluates different aspects of the designed solution. Section 3.1 investigates constrained energy-optimal trajectory planning and the different possible solutions found by the optimizer and assesses the efficiency of the RRT*-GBO algorithm. Section 2.1.2 analyzes the effectiveness of the designed perception objective metrics, the offline interpolation for time economy, and the influence of the weighting between energy- and perception-optimality in planned trajectories. Section 3.3 describes the setup and results of the Astrobee rendezvous experiment in the simulated ISS scenario, and Sec 3.3 describes the setup and results for the satellite capture experiment in DLR's OOS-SIM facility.
Chapter 4 discusses the findings from Chapter 3, and Chapter 5 concludes.

1.4 Theoretical background

This section introduces the fundamental concepts that set the basis for our methods.

1.4.1 Transformations matrices in SE(3)

A homogeneous transformation matrix T_A^B represents the relative pose (rotation and translation) of a reference frame B concerning a reference frame A. It reads as

$$\mathbf{T}_A^B = \begin{bmatrix} \mathbf{R}_A^B & \mathbf{t}_A^B \\ \mathbf{0} & 1 \end{bmatrix} . \tag{1.1}$$

The rotation matrix $\mathbf{R}_A^B$ is part of the special orthogonal group

$$SO(3) = \{\mathbf{R} \in \mathbb{R}^{3\times3} : \mathbf{R}^{-1} = \mathbf{R}^T, \det(\mathbf{R}) = 1\}, \tag{1.2}$$

and its elements are

$$\mathbf{R}_A^B = \begin{bmatrix} \mathbf{x}_A^B & | & \mathbf{y}_A^B & | & \mathbf{z}_A^B \end{bmatrix} , \tag{1.3}$$

where $\mathbf{x}_A^B, \mathbf{y}_A^B, \mathbf{z}_A^B$ are the orthogonal unit vectors that form the basis of the reference frame B represented in the coordinate system of the reference frame A.
The translation vector $\mathbf{t}_A^B \in \mathbb{R}^3$ encodes the origin of the reference frame B represented in the coordinate system of the reference frame A.
Therefore, T_A^B is part of the special euclidean group

$$SE(3) = \left\{ \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} | \, \mathbf{R} \in SO(3), \mathbf{t} \in \mathbb{R}^3 \right\} \tag{1.4}$$

that contains all rotations and translations in the tri-dimensional space.
The inverse of T_A^B represents the pose of the reference frame A with respect to a reference frame B, such that

$$(T_A^B)^{-1} = T_B^A. \tag{1.5}$$

Furthermore, homogeneous transformation matrices can be combined. Given a third reference frame C, T_A^C can be calculated with

$$T_A^C = T_A^B \cdot T_B^C. \tag{1.6}$$

By fixing a reference frame to a rigid body, homogeneous transformation matrices are a convenient way of representing the body's pose in arbitrary reference frames. For instance, the pose of a frame CoM fixed at the center of mass of a given body can be represented w.r.t the world frame W as T_W^{CoM}.

1.4.2 Angle-axis representation of rotations

The angle-axis representation of rotations $\boldsymbol{\xi} = \theta\mathbf{e} \in \mathbb{R}^3$ describes a rotation of an angle θ around an arbitrary unit vector $\mathbf{e}$. This representation is useful in the context of the gradient-based optimization method used in this thesis since incremental rotations can be represented by component-wise addition, which does not work with rotation matrices. It also has the advantage of using only three parameters against the nine parameters from a rotation matrix.

Conversions between rotation matrix and angle-axis

The angle-axis vector $\boldsymbol{\xi}$ can be converted to a rotation matrix $\mathbf{R}$ using the matrix exponential

$$\mathbf{R} = \exp(\hat{\boldsymbol{\xi}}) \tag{1.7}$$

defined by the Rodrigues' formula for rotations

$$\exp(\hat{\boldsymbol{\xi}}) \stackrel{\text{def}}{=} \mathbf{I} + \frac{\sin(\theta)}{\theta}\hat{\boldsymbol{\xi}} + \frac{1-\cos(\theta)}{\theta^2}\hat{\boldsymbol{\xi}}^2, \tag{1.8}$$

where:

$$\hat{\boldsymbol{\xi}} = \begin{bmatrix} 1 & -\xi_z & \xi_y \\ \xi_z & 1 & -\xi_x \\ -\xi_y & \xi_x & 1 \end{bmatrix}. \tag{1.9}$$

Inversely, a rotation matrix

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{1.10}$$

can be converted to an angle-axis vector $\boldsymbol{\xi}$ via the matrix logarithm

$$\hat{\boldsymbol{\xi}} = \log(\mathbf{R}). \tag{1.11}$$

As defined in [LP17], the output of the matrix logarithm $\boldsymbol{\xi}$ follows one of the three cases:

1. If $\mathbf{R} = \mathbf{I}$ then $\theta = 0$ and $\mathbf{e}$ is undefined.
2. If $\text{tr}(\mathbf{R}) = -1$ then $\theta = \pi$ and $\mathbf{e}$ is any of the following that result in a feasible solution:

$$\mathbf{e} = \frac{1}{\sqrt{2(1+r_{33})}} \begin{bmatrix} r_{13} \\ r_{23} \\ 1+r_{33} \end{bmatrix} \tag{1.12}$$

or

$$\mathbf{e} = \frac{1}{\sqrt{2(1+r_{22})}} \begin{bmatrix} r_{12} \\ 1+r_{22} \\ r_{32} \end{bmatrix} \tag{1.13}$$

or

$$\mathbf{e} = \frac{1}{\sqrt{2(1+r_{11})}} \begin{bmatrix} 1+r_{11} \\ r_{21} \\ r_{31} \end{bmatrix} \tag{1.14}$$

.

3. Otherwise $\theta = \arccos\left(\frac{1}{2}\left(\text{tr}(\mathbf{R}) - 1\right)\right) \in [0, \pi)$ and

$$\hat{\mathbf{e}} = \frac{1}{2\sin(\theta)}[\mathbf{R} - \mathbf{R}^T] \tag{1.15}$$

Conversions between angular velocity/acceleration and angle-axis derivatives

According to [NP16], the angular velocity $\boldsymbol{\omega}$ and angular acceleration $\dot{\boldsymbol{\omega}}$ are given by:

$$\boldsymbol{\omega} = \mathbf{A}(\boldsymbol{\xi})\dot{\boldsymbol{\xi}} \tag{1.16}$$

$$\dot{\boldsymbol{\omega}} = \mathbf{A}(\boldsymbol{\xi})\ddot{\boldsymbol{\xi}} + \mathbf{C}(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}), \tag{1.17}$$

where

$$\mathbf{A}(\boldsymbol{\xi}) \stackrel{\text{def}}{=} \mathbf{I} - \frac{1-\cos(\|\boldsymbol{\xi}\|)}{\|\boldsymbol{\xi}\|^2}\hat{\boldsymbol{\xi}} + \frac{\|\boldsymbol{\xi}\| - \sin(\|\boldsymbol{\xi}\|)}{\|\boldsymbol{\xi}\|^3}\hat{\boldsymbol{\xi}}^2 \tag{1.18}$$

and

$$\begin{aligned} \mathbf{C}(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}) \stackrel{\text{def}}{=} & \frac{\|\boldsymbol{\xi}\| - \sin(\|\boldsymbol{\xi}\|)}{\|\boldsymbol{\xi}\|^3}\dot{\boldsymbol{\xi}} \times (\boldsymbol{\xi} \times \dot{\boldsymbol{\xi}}) \\ & - \frac{2\cos(\|\boldsymbol{\xi}\|) + \|\boldsymbol{\xi}\|\sin(\|\boldsymbol{\xi}\|) - 2}{\|\boldsymbol{\xi}\|^4}\boldsymbol{\xi}^T\dot{\boldsymbol{\xi}}(\boldsymbol{\xi} \times \dot{\boldsymbol{\xi}}) \\ & + \frac{3\sin(\|\boldsymbol{\xi}\|) - \|\boldsymbol{\xi}\|\cos(\|\boldsymbol{\xi}\|) - 2\|\boldsymbol{\xi}\|}{\|\boldsymbol{\xi}\|^5}\boldsymbol{\xi}^T\dot{\boldsymbol{\xi}}(\boldsymbol{\xi} \times (\boldsymbol{\xi} \times \dot{\boldsymbol{\xi}})). \end{aligned} \tag{1.19}$$

Since $\mathbf{A}(\boldsymbol{\xi})$ is non-singular ([PR97]), the derivatives of the angle-axis vector can be obtained with

$$\dot{\boldsymbol{\xi}} = \mathbf{A}\left(\boldsymbol{\xi}\right)^{-1}\boldsymbol{\omega} \tag{1.20}$$

$$\ddot{\boldsymbol{\xi}} = \mathbf{A}\left(\boldsymbol{\xi}\right)^{-1}\left(\dot{\boldsymbol{\omega}} - \mathbf{C}\left(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}\right)\right). \tag{1.21}$$

1.4.3 Robot dynamics

The method from this thesis focuses on single-body free-flying robots such as the Astrobee, whose rigid body motion can be represented in a fairly straightforward way, as shown next.

The translation vector $\mathbf{t} \in \mathbb{R}^3$ defines the position of the robot's center of mass w.r.t to the world frame and $\boldsymbol{\omega} \in \mathbb{R}^3$ is the angular velocity of the robot w.r.t to a frame fixed at its center of mass.

Newton-Euler dynamics are used to describe the robot's motion, resulting in the equations

$$\ddot{\mathbf{t}} = \mathbf{F}/m \tag{1.22}$$

$$\dot{\boldsymbol{\omega}} = \mathbf{I}^{-1}(\boldsymbol{\tau} + \boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega}), \tag{1.23}$$

where $\mathbf{F} \in \mathbb{R}^3$ is the force vector, m is the mass of the robot, $\boldsymbol{\tau} \in \mathbb{R}^3$ is the torque vector, and $\mathbf{I}$ is the inertia tensor.
The actuation vector $\mathbf{u}$ is defined as

$$\mathbf{u} = \begin{bmatrix} \mathbf{F} \\ \boldsymbol{\tau} \end{bmatrix}, \tag{1.24}$$

and (1.22) and (1.23) can be reformulated as

$$\mathbf{u} = \begin{bmatrix} m\ddot{\mathbf{t}} \\ \mathbf{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega} \end{bmatrix}. \tag{1.25}$$

The instantaneous power P produced by the robot is defined as

$$P = \mathbf{u} \cdot \mathbf{v}, \tag{1.26}$$

where

$$\mathbf{v} = \begin{bmatrix} \dot{\mathbf{t}} \\ \boldsymbol{\omega} \end{bmatrix}. \tag{1.27}$$

1.4.4 Trajectory parameterization with B-Spline functions

The robot's trajectory over time s(t) is represented as the 6-dimensional vector

$$\mathbf{s}(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \\ \xi_x(t) \\ \xi_y(t) \\ \xi_z(t) \end{bmatrix}, \tag{1.28}$$

composed by stacking the three translational components from $\mathbf{t}$ and the three rotational components from $\boldsymbol{\xi}$.
Each of the components of the vector $\mathbf{s}(t)$ must be uniquely defined for every time t in a given interval $[t_0, t_f]$, as well as their derivatives up to a desired order. That can be achieved with a one-dimensional B-Spline function of degree p for each component, which is $p-1$ times continuously differentiable.
Following the definition from [BM08], it is represented as

$$s(t) = \sum_{j=0}^{m} p_j B_j^p(t), \quad t_0 \leq t \leq t_f \tag{1.29}$$

where p_j are the $m+1$ arbitrary control points, and $B_j^p(t)$ are the basis functions. The basis functions $B_j^p(t)$ are recursively defined as

$$B_j^0(t) = \begin{cases} 1, \text{ if } t_j \leq t < t_{j+1} \\ 0, \text{ otherwise} \end{cases} \tag{1.30}$$

$$B_j^p(t) = \frac{t - t_j}{t_{j+p} - t_j} B_j^{p-1}(t) + \frac{t_{j+p+1} - t}{t_{j+p+1} - t_{j+1}} B_{j+1}^{p-1}(t), \quad p > 0 \tag{1.31}$$

where t_i, $i = 0, \ldots, n_{\text{knot}}$, are the components of the *knot vector* $\mathbf{k}$, typically defined as

$$\mathbf{k} = [\underbrace{t_0, \ldots, t_0}_{p+1}, \underbrace{t_{p+1}, \ldots, t_{n_{\text{knot}}-p-1}}_{n_{\text{in}}}, \underbrace{t_f, \ldots, t_f}_{p+1}], \tag{1.32}$$

where n_{in} is an arbitrary number of "interior" points, that can be chosen as linearly spaced values between t_0 and t_f.
The k-th derivative of the B-Spline function can be computed with

$$s^{(k)}(t) = \sum_{j=0}^{m} p_j B_j^{p(k)}(t), \quad t_0 \leq t \leq t_f \tag{1.33}$$

where

$$B_j^{p(k)}(t) = \frac{p!}{(p-k)!} \sum_{i=0}^{k} a_{k,i} B_{j+i}^{p-k} \tag{1.34}$$

with

$$\begin{aligned} a_{0,0} &= 1 \\ a_{k,0} &= \frac{a_{k-1,0}}{u_{j+p-k+1} - u_j} \\ a_{k,i} &= \frac{a_{k-1,i} - a_{k-1,i-1}}{u_{j+p+i-k+1} - u_{j+i}}, \quad i = 1, \ldots, k-1. \\ a_{k,k} &= \frac{-a_{k-1,k-1}}{u_{j+p+1} - u_{j+k}} \end{aligned} \tag{1.35}$$

The number of control points $m+1$ and the length $n_{\text{knot}}+1$ of the knot vector $\mathbf{k}$ are related via

$$m + 1 = n_{\text{knot}} + 1 - (p+1). \tag{1.36}$$

Considering the definition from $\mathbf{k}$ in (1.32),

$$m + 1 = n_{\text{in}} + (p+1). \tag{1.37}$$

These $m+1$ control vertices are design parameters that define the shape of the B-Spline function and can be tuned according to the trajectory requirements.
The chosen $\mathbf{k}$ in (1.32) results in a clamped B-Spline, for which

$$s(t_0) = p_0 \tag{1.38}$$

$$s(t_f) = p_m. \tag{1.39}$$

Therefore, p_0 and p_m are the trajectory's desired start and end points. The remaining $m-1$ control points can be implicitly defined by imposing trajectory constraints. For instance, it is possible to define constraints on the derivatives of the spline at the initial and final times with

$$s^{(k)}(t_0) = b_{t_0,i}, \quad k = 1, \ldots, l \tag{1.40}$$

$$s^{(k)}(t_f) = b_{t_f,i}, \quad k = 1, \ldots, l \tag{1.41}$$

Which can be represented as a linear system of the form

$$\mathbf{A}\mathbf{p} = \mathbf{c} \tag{1.42}$$

where

$$\mathbf{A} = \begin{bmatrix} B_0^{p(1)}(t_0) & B_1^{p(1)}(t_0) & \cdots & B_{m-1}^{p(1)}(t_0) & B_m^{p(1)}(t_0) \\ \vdots & \vdots & & \vdots & \vdots \\ B_0^{p(l)}(t_0) & B_1^{p(l)}(t_0) & \cdots & B_{m-1}^{p(l)}(t_0) & B_m^{p(l)}(t_0) \\ B_0^{p(l)}(t_f) & B_1^{p(l)}(t_f) & \cdots & B_{m-1}^{p(l)}(t_f) & B_m^{p(l)}(t_f) \\ \vdots & \vdots & & \vdots & \vdots \\ B_0^{p(1)}(t_f) & B_1^{p(1)}(t_f) & \cdots & B_{m-1}^{p(1)}(t_f) & B_m^{p(1)}(t_f) \end{bmatrix}, \mathbf{p} = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{m-1} \\ p_m \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} b_{t_0,1} \\ \vdots \\ b_{t_0,l} \\ b_{t_f,l} \\ \vdots \\ b_{t_f,1} \end{bmatrix}. \tag{1.43}$$

As p_0 and p_m are known, system (1.43) can be reformulated as

$$\mathbf{A}'\mathbf{p}' = \mathbf{c}' \tag{1.44}$$

where

$$\mathbf{A}' = \begin{bmatrix} B_1^{p(1)}(t_0) & \cdots & B_{m-1}^{p(1)}(t_0) \\ \vdots & & \vdots \\ B_1^{p(l)}(t_0) & \cdots & B_{m-1}^{p(l)}(t_0) \\ B_1^{p(l)}(t_f) & \cdots & B_{m-1}^{p(l)}(t_f) \\ \vdots & & \vdots \\ B_1^{p(1)}(t_f) & \cdots & B_{m-1}^{p(1)}(t_f)) \end{bmatrix}, \quad \mathbf{p}' = \begin{bmatrix} p_1 \\ \vdots \\ p_{m-1} \end{bmatrix},$$
$$\mathbf{c}' = \left(\begin{bmatrix} b_{t_0,1} \\ \vdots \\ b_{t_0,l} \\ b_{t_f,l} \\ \vdots \\ b_{t_f,1} \end{bmatrix} - \begin{bmatrix} B_0^{p(1)}(t_0) & B_m^{p(1)}(t_0) \\ \vdots & \vdots \\ B_0^{p(l)}(t_0) & B_m^{p(l)}(t_0) \\ B_0^{p(l)}(t_f) & B_m^{p(l)}(t_f) \\ \vdots & \vdots \\ B_0^{p(1)}(t_f) & B_m^{p(1)}(t_f) \end{bmatrix} \begin{bmatrix} p_0 \\ p_m \end{bmatrix} \right). \tag{1.45}$$

System (1.45) has $2l$ equations for $m-1$ unknowns. Choosing $l = p$, there are $2p$ equations. To make the system determined, it is necessary that $m - 1 = 2p$, thus

$$m + 1 = 2(p + 1) \tag{1.46}$$

and from (1.37),

$$n_{\text{in}} = p + 1. \tag{1.47}$$

The resulting control points vector $\mathbf{p}'$ then consists of

$$\mathbf{p}' = [\underbrace{\mathbf{p}_{\text{initial}}}_{p}, \underbrace{\mathbf{p}_{\text{final}}}_{p}]^T. \tag{1.48}$$

More control points can be freely added in between the initial and final control points to change the shape of the spline without affecting the imposed constraints:

$$\mathbf{p}' = [\underbrace{\mathbf{p}_{\text{initial}}}_{p}, \underbrace{\mathbf{p}_{\text{free}}}_{n_{\text{free}}}, \underbrace{\mathbf{p}_{\text{final}}}_{p}]^T, \tag{1.49}$$

resulting in

$$m + 1 = 2(p + 1) + n_{\text{free}} \tag{1.50}$$

and

$$n_{\text{in}} = p + 1 + n_{\text{free}}. \tag{1.51}$$

1.4.5 Gradient-based nonlinear optimization

A nonlinear optimization problem can be generally formulated as

$$\mathbf{x}^* = \underset{\mathbf{x}\in\mathbb{R}^n}{\operatorname{argmin}} f(\mathbf{x}), \quad \text{s.t.} : \tag{1.52}$$

$$\mathbf{h}_{\text{eq}}(\mathbf{x}) = \mathbf{0} \tag{1.53}$$

$$\mathbf{h}_{\text{ineq}}(\mathbf{x}) \leq \mathbf{0}, \tag{1.54}$$

where $f(\mathbf{x})$ is the objective function and $\mathbf{x}$ contains the n design variables to be optimized, $\mathbf{h}_{\text{eq}}(\mathbf{x})$ is a set of equality constraints, and $\mathbf{h}_{\text{ineq}}(\mathbf{x})$ is a set of inequality constraints, which the optimal solution $\mathbf{x}^*$ must satisfy. Any of $f(\mathbf{x})$, $\mathbf{h}_{\text{eq}}(\mathbf{x})$, and $\mathbf{h}_{\text{ineq}}$ may contain nonlinear terms.
Equation (1.52) can be solved using a gradient-based solver such as the sequential least squares programming method SQLSP [Kra94], a gradient-based local optimization algorithm implemented in the nonlinear optimization library NLOpt [Joh07]. "Gradient-based" means that the gradient of the function $f(\mathbf{x})$ is used to orient the algorithm in the search for a minimum. "Local optimization" means that the result found is a local minimum, i.e., $f(\mathbf{x}^*)$ is guaranteed to be smaller than $f(\mathbf{x}^* + \Delta\mathbf{x})$ only for an infinitesimal $\Delta\mathbf{x}$. Many local minima may exist for a given nonlinear function $f(\mathbf{x})$. The global minimum is the smallest among the local minima.
The minimum $\mathbf{x}^*$ found among all the possible minima depends on the initial guess $\mathbf{x_0}$ that must be provided to the solver. It is, therefore, helpful to explore distinct initial guesses.

1.4.6 Camera model

The robot is equipped with a camera that is used for localization purposes. In essence, a camera maps 3D points of the world to a 2D image plane. The basic pinhole projection model presented in [HZ04] is used to represent the camera mapping. Using homogeneous coordinates, a 3D point/feature in the world frame

$$\mathbf{w} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T \tag{1.55}$$

can be mapped to a 2D point on the image plane

$$\mathbf{i} = \alpha \cdot \begin{bmatrix} u & v & 1 \end{bmatrix}^T \tag{1.56}$$

with

$$\mathbf{i} = \mathbf{P}\mathbf{w}\,, \tag{1.57}$$

where $\mathbf{P}$ is the projection matrix

$$\mathbf{P} = \mathbf{K}\mathbf{H}^W_{\text{cam}}. \tag{1.58}$$

The camera extrinsics matrix $\mathbf{H}^W_{\text{cam}}$ is used to transform the feature coordinates from the world frame to the camera frame. It is defined as

$$\mathbf{H}^W_{\text{cam}} = \begin{bmatrix} \mathbf{R}^W_{\text{cam}} & \mathbf{t}^W_{\text{cam}} \end{bmatrix}\,, \tag{1.59}$$

where $\mathbf{R}^W_{\text{cam}}$ is the world frame rotation matrix w.r.t to the camera frame, and $\mathbf{t}^W_{\text{cam}}$ is the translation vector from the origin of the camera frame to the origin of the

world frame, w.r.t. the world frame. The matrix $\mathbf{K}$ is the camera intrinsics matrix, defined as

$$\mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1.60}$$

where f is the camera focal length, and p_x and p_y are offsets from the principal point in the x and y directions. Expanding (1.57) leads to

$$\alpha \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f x_{\text{cam}} + p_x z_{\text{cam}} \\ f y_{\text{cam}} + p_y z_{\text{cam}} \\ z_{\text{cam}}, \end{bmatrix} \tag{1.61}$$

where x_{cam}, y_{cam}, z_{cam} are the feature coordinates w.r.t. to the camera frame, and therefore

$$u = f x_{\text{cam}} / z_{\text{cam}} + p_x \tag{1.62}$$

$$v = f y_{\text{cam}} / z_{\text{cam}} + p_y \tag{1.63}$$

$$\alpha = z_{\text{cam}} \tag{1.64}$$

The limits of the image plane are defined by the width W and the height H. Fig.1.4 illustrates the image plane relative to the camera frame.

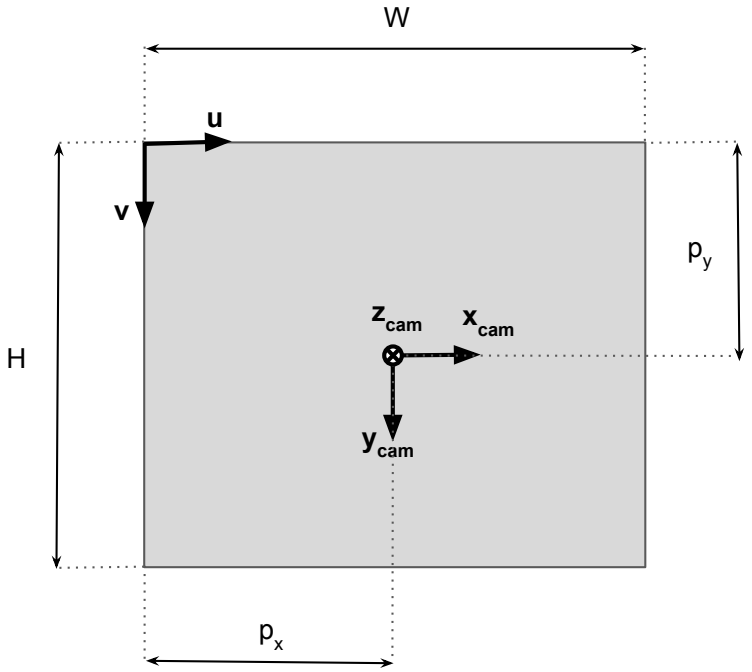


Figure 1.4: Camera frame and image plane.

1.4.7 Mapping and localization

As a robot navigates in an environment, it is fundamental to have some information about the location of existing objects (mapping) and the robot itself (localization), which is helpful for collision avoidance and correcting deviations from a planned trajectory.
That can be achieved with a purely camera-based approach called Visual Simultaneous Localization and Mapping (V-SLAM). It consists of constructing a 3D map of the environment by comparing images taken from different camera views. At the same time, estimations of the camera pose with respect to the initial camera frame are obtained.
In feature-based V-SLAM, the first step is extracting relevant characteristics of the captured images, called features. Examples of features are corners and edges.
Descriptors are mathematical operators applied to the pixels of an image to encode such features numerically. Different descriptors exist in the literature, such as ORB [RRKB11], SIFT [Low04], and BRISK [LCS11].
The extracted features must be matched across different images. As explained in [HZ04], each matched feature gets its 2D positions in pairs of images compared to triangulate the 3D position of the feature and determine how much the camera moved between the two images. The triangulation process is performed considering the epipolar geometry of the camera projections. Multiple pairwise triangulations are fed into a least-squares optimization step that estimates the actual positions of the features and the camera path.
The optimization problem finds the feature positions and camera path that minimize the reprojection error of the 3D features. The reprojection error is defined as the difference between the actual 2D coordinates of a feature on a given camera plane and the (re-)projection of the triangulated 3D position of the feature onto this camera plane. As shown before, the projection of a 3D point onto the camera plane can be obtained via (1.57), which depends on the camera's pose.
The optimization problem can be represented as:

$$\sum_i \sum_j \min_{\hat{\mathbf{w}}_j, \hat{\mathbf{H}}^W_{\text{cam},i(1)}, \hat{\mathbf{H}}^W_{\text{cam},i(2)}} |\mathbf{K}\hat{\mathbf{H}}^W_{\text{cam},i(1)}\hat{\mathbf{w}}_j - \mathbf{i}_{i(1),j}|^2 + |\mathbf{K}\hat{\mathbf{H}}^W_{\text{cam},i(2)}\hat{\mathbf{w}}_j - \mathbf{i}_{i(2),j}|^2 \,, \quad (1.65)$$

where $\hat{\mathbf{w}}_j$ is the estimated 3D position of a feature j, $\hat{\mathbf{H}}^W_{\text{cam},i(1)}$ is the estimated extrinsic matrix of the camera at the first image of the pair i, $\mathbf{i}_{i(1),j}$ is the actual 2D coordinate of the feature j at the first image of the pair i, $\hat{\mathbf{H}}^W_{\text{cam},i(2)}$ is the estimated extrinsic matrix of the camera at the second image of the pair i, $\mathbf{i}_{i(2),j}$ is the actual 2D coordinate of the feature j at the second image of the pair i.
The higher the number of matched features, the more data is available to the formulation (1.65), which leads to higher accuracy in the estimations of the poses of the camera/robot. Therefore, this work focuses on maximizing the visible features

of a precomputed map during a robot's trajectory, aiming to achieve a high number of matches and, consequently, more accurate robot localization.

Chapter 2

Technical Approach

The method developed in this thesis is an RRT*-GBO trajectory optimizer for free-flying robots in SE(3). It accounts for motion constraints and avoids collisions while trying to optimize energy consumption and visibility of features for localization purposes. Figure 2.1 shows an overview of our method. RRT*-GBO is the main module, which includes an optimizer to solve an optimization problem involving energy cost and perception objective metrics, motion constraints, and collision avoidance. The RRT*-GBO and the optimizer output an optimized trajectory and takes as inputs the desired start and goal robot states, the RRT*-GBO tuning parameters, the NLOpt's [Joh07] optimization solver tuning parameters, the B-Spline parameters, the robot dynamics properties and collision geometries, besides a feature map. The feature map is a product of the SLAM module depicted in green, which uses the camera parameters and images to output the map and robot localization estimation. Lastly, the localization is compared to the real trajectory to find the magnitude of the estimation error.

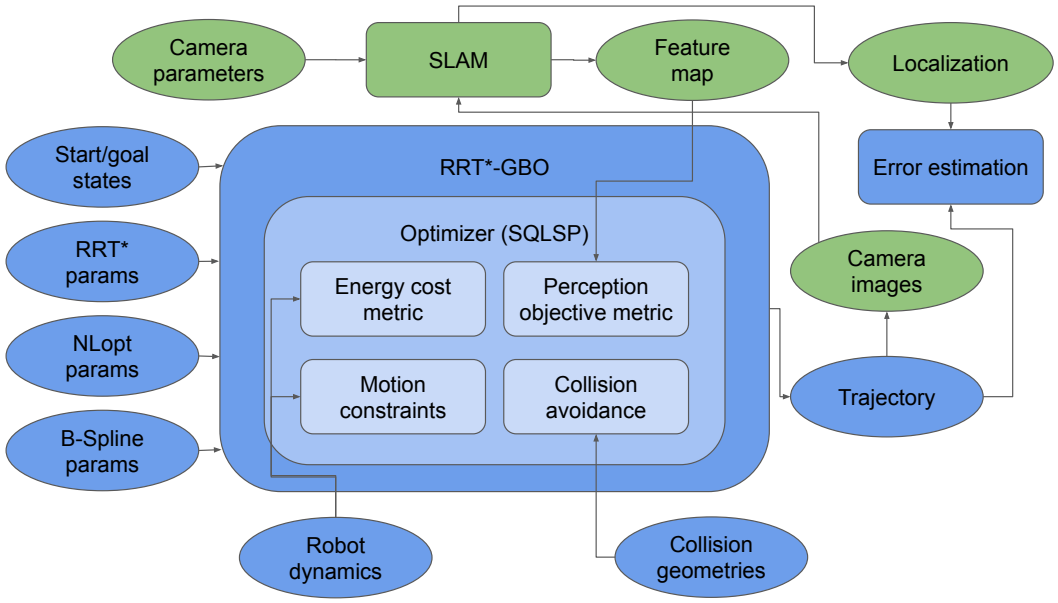


Figure 2.1: Overview of the method developed in this thesis. Square shapes represent modules, and round shapes represent inputs and outputs. The green shapes are related to SLAM.

2.1 Optimization problem

The following optimization problem must be solved:

$$\begin{aligned} \mathbf{p}^*_{\text{free}} = \underset{\mathbf{p}_{\text{free}}}{\operatorname{argmin}}\, \Gamma(\mathbf{p}_{\text{free}}), \\ \text{s.t.: } \mathbf{h}(\mathbf{p}_{\text{free}}) \leq 0 \end{aligned} \tag{2.1}$$

where $\Gamma(\cdot)$ is the cost function, $\mathbf{h}(\cdot)$ is the set of inequality constraints, and $\mathbf{p}_{\text{free}}$ are the n_{free} free control vertices of a B-Spline $\mathbf{s}(t)$ subject to boundary conditions

$$\mathbf{s}^{(k)}(t_0) = \mathbf{b}_{t_0,i}, \quad k = 0, \ldots, l \tag{2.2}$$

$$\mathbf{s}^{(k)}(t_f) = \mathbf{b}_{t_f,i}, \quad k = 0, \ldots, l \tag{2.3}$$

as introduced in Sec. 1.4.4.
Whenever the free control vertices $\mathbf{p}_{\text{free}}$ assume new values, the B-Spline trajectory and its derivatives $\mathbf{s}^{(k)}(t)$, $k = 0, \ldots, l$ are updated and sampled at time instants t_i, $i = 1, \ldots, n_{\text{samples}}$ to check constraint satisfaction and compute the components of the cost function.
The optimized robot trajectory must consume the least possible amount of energy while maximizing the number of visual features observable by the camera. A mixed cost function with weighted energy and perception terms encodes that.

2.1.1 Energy cost function

The energy cost function Γ_{energy} reflects the energy spent by the robot during the trajectory. It is designed as a sum of the instantaneous power (1.26) produced at the sampled time instants t_i:

$$\Gamma_{\text{energy}} = \sum_{i=1}^{n_{\text{samples}}} \mathbf{u}(t_i) \cdot \mathbf{v}(t_i). \tag{2.4}$$

2.1.2 Perception objective function

Given a sparse feature map of the environment, the perception objective function $\Gamma_{\text{perception}}$ quantifies the features the robot's camera sees along a trajectory. Generally, it can be defined as:

$$\Gamma_{\text{perception}} = \sum_{i=1}^{n_{\text{samples}}} \gamma(\mathbf{H}^W_{\text{cam,i}}), \tag{2.5}$$

where

$$\gamma(\mathbf{H}^W_{\text{cam,i}}) = \sum_{j=1}^{n_{\text{features}}} h(i,j). \tag{2.6}$$

is the summed feature visibility function for the robot's camera pose at a time instant t_i (represented by the extrinsics matrix $\mathbf{H}^W_{\text{cam,i}}$), and the function $h(i,j)$ is the visibility score of an individual feature j at that instant. Two alternative implementations of the perception objective function $\Gamma_{\text{perception}}$ are introduced next: the feature count function and the relaxed visibility function. After that, a method to interpolate the summed visibility function offline to save computation time is presented.

Feature count

A straightforward approach is to count the number of visible features at each time step.
The 2D projection $\mathbf{i}_{i,j}$ of a 3D feature $\mathbf{w}_j$ onto the camera plane at a time instant t_i can be obtained using the projection matrix $\mathbf{P}_i = \mathbf{K}\mathbf{H}^W_{\text{cam,i}}$ from (1.58) with

$$\mathbf{i}_{i,j} = \mathbf{P}_i \mathbf{w}_j = \alpha_{i,j} \cdot \begin{bmatrix} u_{i,j} \\ v_{i,j} \\ 1 \end{bmatrix} \tag{2.7}$$

Considering the image width W, the image height H, and the identity $\alpha_{i,j} = z_i^{\text{cam}}$ from (1.64), the visibility of a projected feature can be represented by:

$$h_{\text{FC}}(\mathbf{i}_{i,j}) = \begin{cases} 1, \text{ if } 0 \leq u_{i,j} \leq W \text{ and } 0 \leq v_{i,j} \leq H \text{ and } \alpha_{i,j} = z_i^{\text{cam}} > 0 \\ 0, \text{ otherwise} \end{cases} \tag{2.8}$$

Which composes the feature count objective function

$$\Gamma_{\text{FC}} = \sum_{i=1}^{n_{\text{samples}}} \gamma_{\text{FC}}(\mathbf{H}^W_{\text{cam,i}}), \tag{2.9}$$

where

$$\gamma_{\text{FC}}(\mathbf{H}^W_{\text{cam,i}}) = \sum_{j=1}^{n_{\text{features}}} h_{\text{FC}}(\mathbf{i}_{i,j}) \tag{2.10}$$

A clear downside of this approach is that the function h_{visible} is discontinuous, which is unsuitable for gradient computation. As an alternative, Murali et al. [MSGK19] present a (continuous) relaxed visibility function, which is further explored by Bartolomei et al. [BTC20].

Relaxed visibility

The relaxed visibility function quantifies the visibility of a feature with a real number between 0 and 1 (where 1 is complete visibility). It does that by modeling the camera view frustum as vectors from the camera's optical center to the vertices of the image plane:

$$\mathbf{v}_{tr} = \begin{bmatrix} W - p_x \\ -p_y \\ f \end{bmatrix} \tag{2.11}$$

$$\mathbf{v}_{tl} = \begin{bmatrix} -c_x \\ -c_y \\ f \end{bmatrix} \tag{2.12}$$

$$\mathbf{v}_{lr} = \begin{bmatrix} W - p_x \\ H - p_y \\ f \end{bmatrix} \tag{2.13}$$

$$\mathbf{v}_{ll} = \begin{bmatrix} -p_x \\ H - p_y \\ f \end{bmatrix} \tag{2.14}$$

This set of vectors is used to compute the visibility operator

$$\mathbf{o}(\mathbf{f}_{i,j}) = \begin{bmatrix} \frac{1}{2}\left(1 + \tanh\left(\frac{(\mathbf{v}_{tr} \times \mathbf{v}_{lr}) \cdot \mathbf{f}_{i,j}}{s}\right)\right) \\ \frac{1}{2}\left(1 + \tanh\left(\frac{(\mathbf{v}_{tl} \times \mathbf{v}_{tr}) \cdot \mathbf{f}_{i,j}}{s}\right)\right) \\ \frac{1}{2}\left(1 + \tanh\left(\frac{(\mathbf{v}_{lI} \times \mathbf{v}_{tl}) \cdot \mathbf{f}_{i,j}}{s}\right)\right) \\ \frac{1}{2}\left(1 + \tanh\left(\frac{(\mathbf{v}_{lr} \times \mathbf{v}_{ll}) \cdot \mathbf{f}_{i,j}}{s}\right)\right) \\ \frac{1}{2}\left(1 + \tanh\left(\frac{f \cdot \mathbf{e}_z \cdot \mathbf{f}_{i,j}}{s}\right)\right) \end{bmatrix}, \tag{2.16}$$

where $\mathbf{e}_z = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$, $\mathbf{f}_{i,j} = \mathbf{H}^W_{\text{cam,i}}\mathbf{w}_j$ is the position of a feature j w.r.t to the camera frame at a timestep t_i, and s is a tuning parameter. Instead of tuning s, as in [MSGK19], this thesis presents a variation of the relaxed visibility function by normalizing the frustum vectors, obtaining the vector

$$\mathbf{o}'(\mathbf{f}_{i,j}) = \begin{bmatrix} \frac{1}{2}\left(1 + \tanh\left(\frac{(\mathbf{v}_{tr} \times \mathbf{v}_{lr}) \cdot \mathbf{f}_{i,j}}{|\mathbf{v}_{tr} \times \mathbf{v}_{lr}|}\right)\right) \\ \frac{1}{2}\left(1 + \tanh\left(\frac{(\mathbf{v}_{tl} \times \mathbf{v}_{tr}) \cdot \mathbf{f}_{i,j}}{|\mathbf{v}_{tl} \times \mathbf{v}_{tr}|}\right)\right) \\ \frac{1}{2}\left(1 + \tanh\left(\frac{(\mathbf{v}_{lI} \times \mathbf{v}_{tl}) \cdot \mathbf{f}_{i,j}}{\mathbf{v}_{lI} \times \mathbf{v}_{tl}}\right)\right) \\ \frac{1}{2}\left(1 + \tanh\left(\frac{(\mathbf{v}_{lr} \times \mathbf{v}_{ll}) \cdot \mathbf{f}_{i,j}}{\mathbf{v}_{lr} \times \mathbf{v}_{ll}}\right)\right) \\ \frac{1}{2}\left(1 + \tanh\left(\mathbf{e}_z \cdot \mathbf{f}_{i,j}\right)\right) \end{bmatrix}. \tag{2.17}$$

Finally, the relaxed visibility function is

$$h_{\mathrm{RV}}(\mathbf{o}'(\mathbf{f}_{i,j})) = \prod_{k=1}^{5} (\mathbf{o}'(\mathbf{f}_{i,j}))_k, \tag{2.18}$$

the summed feature visibility for a camera pose i is

$$\gamma_{\mathrm{RV}}\left(\mathbf{H}^{W}_{\mathrm{cam},i}\right) = \sum_{j=1}^{n_{\mathrm{features}}} h_{\mathrm{relaxed}}(\mathbf{o}'(\mathbf{f}_{i,j})), \tag{2.19}$$

and the relaxed visibility objective function Γ_{RV} adds up to

$$\Gamma_{\mathrm{RV}} = \sum_{i=1}^{n_{\mathrm{samples}}} \gamma_{\mathrm{RV}}\left(\mathbf{H}^{W}_{\mathrm{cam},i}\right). \tag{2.20}$$

Summed feature visibility interpolation

The computation of (2.5) means $n_{\mathrm{samples}} \cdot n_{\mathrm{features}}$ evaluations of the visibility function h for each iteration of the optimizer. To put it in numbers, we take an example of $n_{\mathrm{samples}} = 60$, and $n_{\mathrm{features}} = 500$, which can be interpreted as a sampling rate of 1Hz for a 60s-trajectory using a map with few features. It results in 30.000 function evaluations per iteration of the optimizer, which incurs elevated computation times. The solution adopted in this thesis is to perform an offline, multi-dimensional interpolation of the summed feature visibility function γ (2.6) over all the possible robot camera poses. Considering that the camera's pose (represented as the camera extrinsics matrix $\mathbf{H}^{W}_{\mathrm{cam}}$) is a function of the robot's pose

$$\mathbf{s} = \begin{bmatrix} x & y & z & \xi_x & \xi_y & \xi_z \end{bmatrix}^T, \tag{2.21}$$

a six-dimensional grid in the robot's pose parameters is generated to compute the summed feature visibility function

$$\gamma\left(\mathbf{H}^{W}_{\mathrm{cam}}\left(\begin{bmatrix} x & y & z & \xi_x & \xi_y & \xi_z \end{bmatrix}^T_m\right)\right) \tag{2.22}$$

at each grid point m, and interpolate between the grid points to generate a continuous, approximated summed feature visibility function

$$\hat{\gamma}(x, y, z, \xi_x, \xi_y, \xi_z), \tag{2.23}$$

defined for any pose in the robot's workspace. The open-source library Btwxt [Sof24] was chosen to perform the multidimensional interpolation using cubic splines (Catmull-Rom). At the gridpoints m,

$$\hat{\gamma}\left(\begin{bmatrix} x & y & z & \xi_x & \xi_y & \xi_z \end{bmatrix}^T_m\right) = \gamma\left(\mathbf{H}^{W}_{\mathrm{cam}}\left(\begin{bmatrix} x & y & z & \xi_x & \xi_y & \xi_z \end{bmatrix}^T_m\right)\right), \tag{2.24}$$

and if the resolution of the input grid is high enough, we can assume

$$\hat{\gamma} \approx \gamma. \tag{2.25}$$

That finally brings us to the interpolated perception objective function

$$\hat{\Gamma}_{\text{perception}} = \sum_{i=1}^{n_{\text{samples}}} \hat{\gamma}(\mathbf{s}(t_i)), \tag{2.26}$$

which means $n_{\text{samples}}s$ function evaluations at each iteration of the optimizer, instead of the $n_{\text{samples}} \cdot n_{\text{features}}$ function evaluations from the original perception objective function.

2.1.3 Weighted cost function

The weighted cost function mixes the energy cost function and the perception objective function with the formulation

$$\Gamma_{\text{weighted}} = w_{\text{energy}} \left(\frac{\Gamma_{\text{energy}}}{\Gamma_{\text{energy,max}}} \right) + (1 - w_{\text{energy}}) \left(1 - \frac{\Gamma_{\text{perception}}}{\Gamma_{\text{perception,max}}} \right), \tag{2.27}$$

where

$$0 \leq w_{\text{energy}} \leq 1, \tag{2.28}$$

$$\Gamma_{\text{energy,max}} = n_{\text{samples}} \cdot \mathbf{u}_{\max} \cdot \mathbf{v}_{\max}, \tag{2.29}$$

$$\Gamma_{\text{perception,max}} = n_{\text{samples}} \cdot n_{\text{features}}, \tag{2.30}$$

where $\mathbf{u}_{\max}$ is the maximum actuation effort and $\mathbf{v}_{\max}$ is the maximum velocity achievable by the robot.

2.1.4 Constraints

The robot is subject to a few constraints that must be satisfied at every time instant.

Kinodynamic constraints

The kinodynamic constraints can be represented as a series of box constraints defined for every time instant t_i. Following the notation from Sec. 1.4.3, the translational position constraint is defined as

$$\mathbf{t}_{\min} \leq \mathbf{t}(t_i) \leq \mathbf{t}_{\max}, \tag{2.31}$$

the translational and rotational velocity constraint is defined as

$$\mathbf{v}_{\min} \leq \mathbf{v}(t_i) \leq \mathbf{v}_{\max}, \tag{2.32}$$

and the force and torque actuation constraint is defined as

$$\mathbf{u}_{\min} \leq \mathbf{u}(t_i) \leq \mathbf{u}_{\max} \tag{2.33}$$

Collision avoidance constraint

The Open Dynamics Engine (ODE) library [Smi04] is used for the collision avoidance constraint. The robot and the n_{obj} static objects in the environment are modeled as simple, convex geometries consisting of combinations of spheres and capsules.
At each time instant t_i of the trajectory, the robot's collision geometry has its pose updated and gets compared to every other collision geometry (of the static objects) to obtain the penetration depths PD_j, for $j = 1, \ldots, n_{\text{obj}}$.
Given that the metric PD_j indicates how deep two collision geometries intersect, the collision avoidance constraint is defined as

$$PD_j(t_i) \leq 0 \tag{2.34}$$

2.1.5 Solver

The gradient-based local optimization algorithm SQLSP [Kra94] from the NLOpt library [Joh07] is used to solve the presented optimization problem. First-order forward finite differences are used to obtain the required gradient information for the cost function and constraints in the form:

$$\nabla f(\mathbf{x}) \approx \frac{f(\mathbf{x} + \boldsymbol{\Delta}\mathbf{x}) - f(\mathbf{x})}{\boldsymbol{\Delta}\mathbf{x}}, \tag{2.35}$$

where $\nabla f(\mathbf{x})$ is the gradient of an arbitrary function $f \colon \mathbb{R}^n \mapsto \mathbb{R}$ and $\boldsymbol{\Delta}\mathbf{x} \in \mathbb{R}^n$ is the gradient step for a variable $\mathbf{x} \in \mathbb{R}^n$. By default, we choose $\boldsymbol{\Delta}\mathbf{x} = 1e^{-6}$. In the context of this thesis, f is either the cost Γ or a constraint h, and $\mathbf{x}$ is $\mathbf{p}_{\text{free}}$.
The solver works with termination conditions that dictate when the execution should finish. These conditions include a maximum execution time $t_{\max}$, a maximum number of iterations n_{maxiter}, the cost function variation $\Delta\Gamma$ from one iteration to the next, and the optimization parameters variation $\Delta\mathbf{p}_{\text{free}}$ from one iteration to the next. The relative variations $\Delta\Gamma_{\text{rel}} = \Delta\Gamma/\Gamma$ and $\Delta\mathbf{p}_{\text{free,rel}} = \Delta\mathbf{p}_{\text{free}}/\mathbf{p}_{\text{free}}$ can also be used.

2.2 RRT*-GBO

There is no guarantee of convexity in the optimization problem described so far; therefore, the results found can only be seen as local minima. The RRT*-GBO algorithm [SL14] is employed to explore non-convex regions and search for global minima. Due to its stochastic nature, it explores several solutions on different local minima and provides an overall better optimization result.
RRT*-GBO extends the classic RRT* algorithm by blending gradient-based optimization (GBO) into it. There are two main additions: first, RRT*-GBO solves reduced optimization problems (using simplified B-Splines) for building edges. These edges are called GBO edges. Second, when a path of edges from the root to the goal node is found, it goes through a smoothing step: the edges B-Splines are merged into one B-Spline, which is then used as an initial guess for the original optimization problem.

2.2.1 Algorithm steps

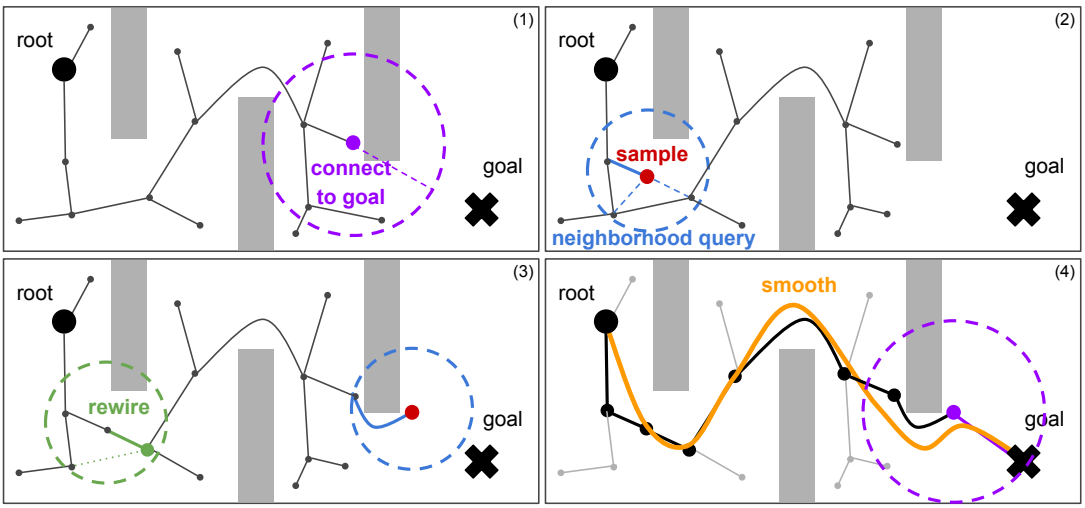


Figure 2.2: RRT*-GBO steps.

The main steps of the algorithm [SL14] are depicted in Fig. 2.2. At each iteration, the algorithm starts by trying to connect the last added node to the goal node with a GBO edge (**connect to goal**). If it is successful, the obtained path of edges goes through the smoothing step mentioned above to deliver a possible solution (**smooth**). The next step is to sample a new node in the search space (**sample**). For every existing node in the "neighborhood" of the sampled node, a GBO edge from the neighboring node to the sampled node is attempted to be built. The successfully built edges are compared, and the cheapest one is chosen to attach the sampled node to the existing tree (**neighborhood query**). The newly attached

node is used in the rewiring step to check if any nodes in the previously defined neighborhood should have their incoming edge replaced by an edge departing from the new node, in case that reduces the cost to get to the neighboring node (**rewire**). The algorithm finishes when certain stopping criteria are met, such as a maximum number of solutions found or a maximum execution time.
Pseudo-code for the algorithm is presented in Appendix A.

2.2.2 Tuning parameters

The algorithm contains some parameters that must be tuned: r_{ball}, r_{prune} and r_{connect}. The parameter r_{ball} has a dual function. First, a newly sampled node has to be at most at a r_{ball} distance away from the node that is closest to it. In case the sampled node is located further than r_{ball} away from the closest node, a "steering" step must be conducted: the vector that unifies both gets scaled down to have a norm of r_{ball}, and the sampled node location gets updated accordingly. Second, it dictates the size of the neighborhood, i.e., nodes at a maximum distance of r_{ball} are checked in the neighborhood query step.
The parameter r_{prune} is the minimum distance allowed between a newly sampled node and any other node in the existing tree. The sampled node gets pruned away if it is too close to another node, and a new node is sampled.
The parameter r_{goal} is the maximum distance that a node can be away from the goal node such that an attempt to connect both is made in the "connect to goal" step.

2.2.3 Distance metrics in SE(3)

Since the RRT*-GBO nodes are in SE(3), handling translational and rotational distances is necessary. For the translational part ($\mathbb{R}^3$), the Euclidean distance metric is used:

$$\phi_{\text{translation},\,1\rightarrow 2} = ||\mathbf{t}_2 - \mathbf{t}_1||. \tag{2.36}$$

The Geodesic on the Unit Sphere metric from [Huy09] is chosen to represent the distance between two rotations ($SO(3)$). It is defined as

$$\phi_{\text{rotation},\,1\rightarrow 2}(\mathbf{R}_1, \mathbf{R}_2) = ||\log(\mathbf{R}_1\mathbf{R}_2)|| = ||\boldsymbol{\xi}_{1\rightarrow 2}|| = \theta_{1\rightarrow 2}, \tag{2.37}$$

where $\mathbf{R}_1$ and $\mathbf{R}_2$ are the two rotations represented as rotation matrices and $\log(\cdot)$ is the logarithmic mapping introduced in (1.11). Therefore, the distance metric consists of the angle in the angle-axis representation of rotations adopted in this thesis.
As a consequence, the three parameters r_{ball}, r_{prune}, and r_{connect} turn into six parameters: $r_{\text{ball, translation}}$, $r_{\text{ball, rotation}}$, $r_{\text{prune, translation}}$, $r_{\text{prune, rotation}}$, $r_{\text{connect, translation}}$ and $r_{\text{connect, rotation}}$.
In the RRT*-GBO algorithm, it is desirable to compute the generalized distance between two nodes as a single scalar value, which requires the derivation of a unified distance metric

$$\phi_{\text{unified}}(\phi_{\text{translation}}, \phi_{\text{rotation}}) = \frac{\phi_{\text{translation}}}{r_{\text{ball, translation}}} + \frac{\phi_{\text{rotation}}}{r_{\text{ball, rotation}}}. \tag{2.38}$$

The adaptations on the RRT*-GBO algorithm for the $SE(3)$ distance metrics are presented next.
For each node N_i, $i = 1, \ldots, n_{\text{nodes}}$, the index of closest node to the sampled node is computed with:

$$i_{\text{closest}} = \underset{i}{\operatorname{argmin}}\, \phi_{\text{unified}}(\phi_{\text{translation}, i \rightarrow \text{sampled}}, \phi_{\text{rotation}, i \rightarrow \text{sampled}}). \tag{2.39}$$

The steering step becomes:

$$\mathbf{t}_{\text{sampled}} = \mathbf{t}_{\text{closest}} + r_{\text{ball,translation}} \frac{\mathbf{t}_{\text{sampled}} - \mathbf{t}_{\text{closest}}}{\phi_{\text{translation, closest} \rightarrow \text{sampled}}} \tag{2.40}$$

$$\boldsymbol{\xi}_{\text{sampled}} = \boldsymbol{\xi}_{\text{closest}} + r_{\text{ball,rotation}} \frac{\boldsymbol{\xi}_{\text{closest, sampled}}}{\phi_{\text{rotation, closest} \rightarrow \text{sampled}}}. \tag{2.41}$$

A sampled node gets discarded if

$$\phi_{\text{unified}}(\phi_{\text{translation, closest} \rightarrow \text{sampled}}, \phi_{\text{rotation, closest} \rightarrow \text{sampled}}) < r_{\text{prune, unified}} \tag{2.42}$$

where

$$r_{\text{prune, unified}} = \phi_{\text{unified}}(r_{\text{prune, translation}}, r_{\text{prune, rotation}}). \tag{2.43}$$

A node N_i is part of the neighborhood of the sampled node if

$$\phi_{\text{translation}, i \rightarrow \text{sampled}} \leq r_{\text{ball, translation}}, \tag{2.44}$$

$$\phi_{\text{rotation}, i \rightarrow \text{sampled}} \leq r_{\text{ball, rotation}}, \tag{2.45}$$

$$\phi_{\text{unified}}(\phi_{\text{translation, i} \rightarrow \text{sampled}}, \phi_{\text{rotation, i} \rightarrow \text{sampled}}) \geq r_{\text{prune, unified}}. \tag{2.46}$$

Finally, an attempt to build an edge between the sampled node N_{sampled} and the goal node N_{goal} is made if

$$\phi_{\text{translation, sampled} \rightarrow \text{goal}} \leq r_{\text{connect, translation}}, \tag{2.47}$$

$$\phi_{\text{rotation, sampled} \rightarrow \text{goal}} \leq r_{\text{connect, rotation}}. \tag{2.48}$$

2.2.4 GBO edges creation

Each GBO edge is a B-Spline $\mathbf{s}_{\text{edge}}(t)$ with fewer free control vertices and shorter duration than the full trajectory B-Spline $\mathbf{s}(t)$:

$$n_{\text{free, edge}} < n_{\text{free}}, \tag{2.49}$$

$$\Delta t_{\text{edge}} < \Delta t. \tag{2.50}$$

Its initial and final poses comprise the translation and rotation components of the parent and child nodes, respectively. The initial and final velocities, accelerations, and jerks are set to zero. The free control vertices $\mathbf{p}_{\text{free, edge}}$ are obtained by solving the optimization problem (2.1) for the B-Spline $\mathbf{s}_{\text{edge}}(t)$ using the weighted cost function Γ_{weighted}.

2.2.5 Smoothing

The smoothing step consists of transforming a path of GBO edges from the root to the goal node $\mathbf{s}^j_{\text{edge}}(t)$, $j = 1, \ldots, n_{\text{edges}}$ in an initial guess of free control vertices of a single B-Spline from the root to the goal that gets optimized by the solver.
First, the duration Δt_{edge} of each edge B-Spline must be scaled by a factor λ such that their summed duration matches the chosen duration Δt of the full trajectory B-Spline:

$$\lambda = \frac{\Delta t}{n_{\text{edges}} \cdot \Delta t_{\text{edge}}} \tag{2.51}$$

Then, the sequence of edge B-Splines is sampled at the sampling times t_i, $i = 1, \ldots, n_{\text{samples}}$ of the full trajectory B-Spline. That requires identifying which edge B-Spline $\mathbf{s}_j(t)$ spans the time t_i and sampling this B-Spline at its "local" time $t^j_i = t_i - \lambda \cdot \Delta t_{\text{edge}} \cdot (j-1)$. The samples are stored in a matrix $\mathbf{S} \in \mathbb{R}^{6 \times n_{\text{samples}}}$.
The optimization problem (2.1) is then solved using a least-squares cost function to fit the full trajectory B-Spline to the edge B-Splines samples, with

$$\Gamma = \Gamma_{\text{LS}} = \sum_{i=0}^{n_{\text{samples}}} ||\mathbf{S}_{1:6,\,i} - \mathbf{s}(t_i)||. \tag{2.52}$$

The resulting free control vertices $\mathbf{p}_{\text{free, LS}}$ are then used as an initial guess to the original optimization problem (2.1) using the weighted cost function Γ_{weighted}, which outputs the final optimized free control vertices $\mathbf{p}_{\text{free, smoothed}}$.

2.3 Mapping and localization

Our method requires a feature map of the environment as an input for the optimization problem and a way to assess the robot localization accuracy for different trajectories. The open-source, state-of-the-art algorithm ORB-SLAM3 [CEG+21] suits both requirements. All it needs is a model of the camera (intrinsics matrix) and a sequence of pictures of the environment taken by the robot.
While the robot follows a trajectory in the environment, its camera captures images that ORB-SLAM3 processes to perform a 3D reconstruction (feature map) of the regions the robot has visited and estimate the trajectory followed by the robot.
The resulting estimated map points (features) $\hat{\mathbf{f}}_{0,j}$ and estimated camera poses $\hat{\mathbf{T}}^{\text{cam},i}_{\text{cam},0}$ are relative to the (known) camera pose at the start of the image collection $\mathbf{T}^{\text{cam},0}_W$, and can be represented in the world frame with

$$\hat{\mathbf{f}}_{W,\,j} = (\mathbf{T}^{\text{cam},0}_W)^{-1} \hat{\mathbf{f}}_{0,\,j}, \tag{2.53}$$

$$\hat{\mathbf{T}}^{\text{cam},i}_W = (\mathbf{T}^{\text{cam},0}_W)^{-1} \hat{\mathbf{T}}^{\text{cam},i}_{\text{cam},0}. \tag{2.54}$$

However, $\hat{\mathbf{f}}_{W,j}$ and $\hat{\mathbf{T}}_W^{\text{cam},i}$ (for the translations) are not correctly scaled since the employed ORB-SLAM3 variant is purely visual and monocular. Our approach to retrieve the correct scaling factor β is to run a least-squares optimization

$$\beta^* = \underset{\beta}{\text{argmin}} \sum_{i=0}^{n_{\text{samples}}} ||\hat{\mathbf{t}}_{\text{cam,scaled}}(t_i) - \mathbf{t}_{\text{cam}}(t_i)||, \tag{2.55}$$

where

$$\hat{\mathbf{t}}^*_{\text{cam}}(t_i) = \mathbf{t}_{\text{cam}}(0) + \beta \cdot (\hat{\mathbf{t}}_{\text{cam}}(t_i) - \mathbf{t}_{\text{cam}}(0)) \tag{2.56}$$

are the real-scale estimated camera translations and $\mathbf{t}_{\text{cam}}(t_i)$ are the planned camera translations computed from the robot poses $\mathbf{s}(t_i)$ using $\mathbf{T}^{\text{cam}}_{CoM}$.
The real-scale estimated features are computed with

$$\hat{\mathbf{f}}^*_{W,j} = (\mathbf{T}_W^{\text{cam},0})^{-1} \cdot \begin{bmatrix} \beta & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \hat{\mathbf{f}}_{0,j}, \tag{2.57}$$

Chapter 3

Evaluation

Section 3.1 shows the foundational results of the developed method. Energy-optimal motion planning is performed in distinct scenarios with multiple local minima, and the exploration capability of the Monte Carlo and RRT*-GBO methods is evaluated. Section 3.2 introduces perception-aware motion planning. It compares the designed perception metrics, highlights the advantages of the offline multi-dimensional interpolation, and illustrates the influence of the cost terms weighting parameter w_{energy}. Section 3.3 evaluates the complete method for an Astrobee rendezvous experiment in the ISS, and Sec. 3.3 does the same for a satellite capture experiment using DLR's OOS-SIM.

3.1 Energy-optimal trajectory planning

This section presents trajectory optimization (using the energy cost function (2.4)) in increasingly complex scenarios, starting from an unconstrained trajectory in $\mathbb{R}^3$ up to a constrained trajectory in a cluttered scenario in $SO(3)$. Local minima are discussed and RRT*-GBO's performance with different hyperparameter combinations is assessed as an alternative to a Monte Carlo search method.
The Astrobee dynamics properties are used, namely the mass

$$m = 9.58 \quad [\text{kg}] \tag{3.1}$$

and the inertia tensor

$$I = \begin{bmatrix} 0.153 & 0 & 0 \\ 0 & 0.143 & 0 \\ 0 & 0 & 0.162 \end{bmatrix} \quad [\text{kg} \cdot \text{m}^2]. \tag{3.2}$$

3.1.1 Motion in $\mathbb{R}^3$ (3D translations)

We start by optimizing trajectories in $\mathbb{R}^3$, i.e., 3D translations of the robot. Each of the three dimensions of a trajectory from a root position $\mathbf{t}_{\text{root}}$ to a goal position $\mathbf{t}_{\text{goal}}$ is parameterized with a B-Spline of degree $p = 3$, with $n_{\text{free}} = 10$ free control vertices, and duration $t_f = 120$ [s]. The cost and constraint functions are evaluated at 120 sampled points of the trajectory (one per second). For a provided initial guess (either randomly generated or from RRT*-GBO), the optimization solver from Sec. 2.1.5 runs until it meets any of the stopping criteria: $\Delta\Gamma_{\text{rel}} = 1e^{-8}$, $\Delta\mathbf{p}_{\text{free,rel}} = 1e^{-8}$, $t_{\max} = 240$ [s].

Unconstrained trajectory

For $\mathbf{t}_{\text{root}} = \begin{bmatrix} 1 & 1 & 0.5 \end{bmatrix}^T$ and $\mathbf{t}_{\text{goal}} = \begin{bmatrix} 0.5 & 5 & 1 \end{bmatrix}^T$, Figures 3.1 and 3.2 shows the energy-optimal trajectory resulting from optimizing the free control vertices $\mathbf{p}_{\text{free}}$ of each dimension's B-Spline, without any motion constraints. Figure 3.1 shows that the trajectory is a straight line between the root and goal positions. Figure 3.2 shows the B-Splines for each of the 3 spatial dimensions, including the control vertices, the time parametrization, and the first and second derivatives (velocity and acceleration). It is noticeable that the velocities assume a bell-shaped profile, starting and finishing at zero and peaking at the middle of the trajectory.
The initial guess of $\mathbf{p}_{\text{free}}$ for this optimization problem is unimportant since there is a well-defined global minimum and no constraints.

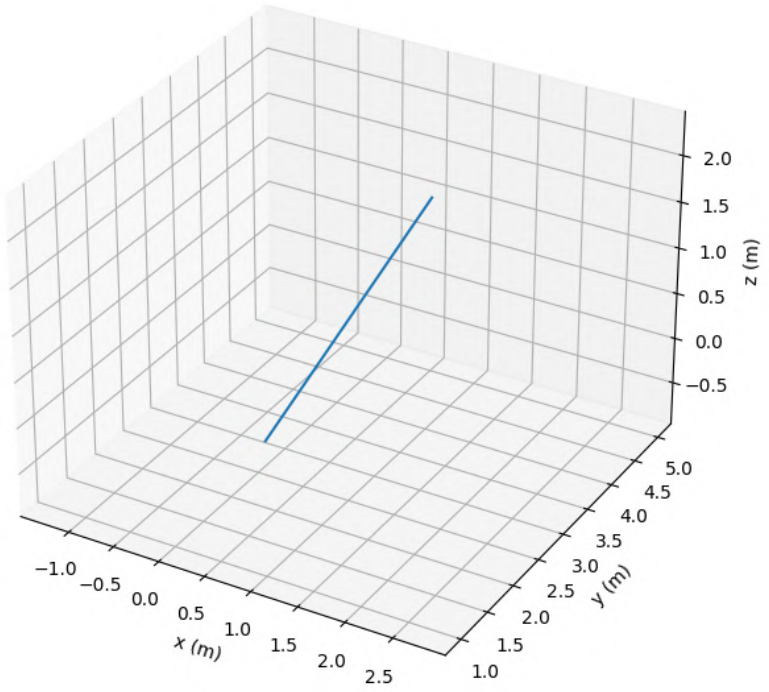


Figure 3.1: Unconstrained trajectory in $\mathbb{R}^3$ (Point-to-point trajectory)

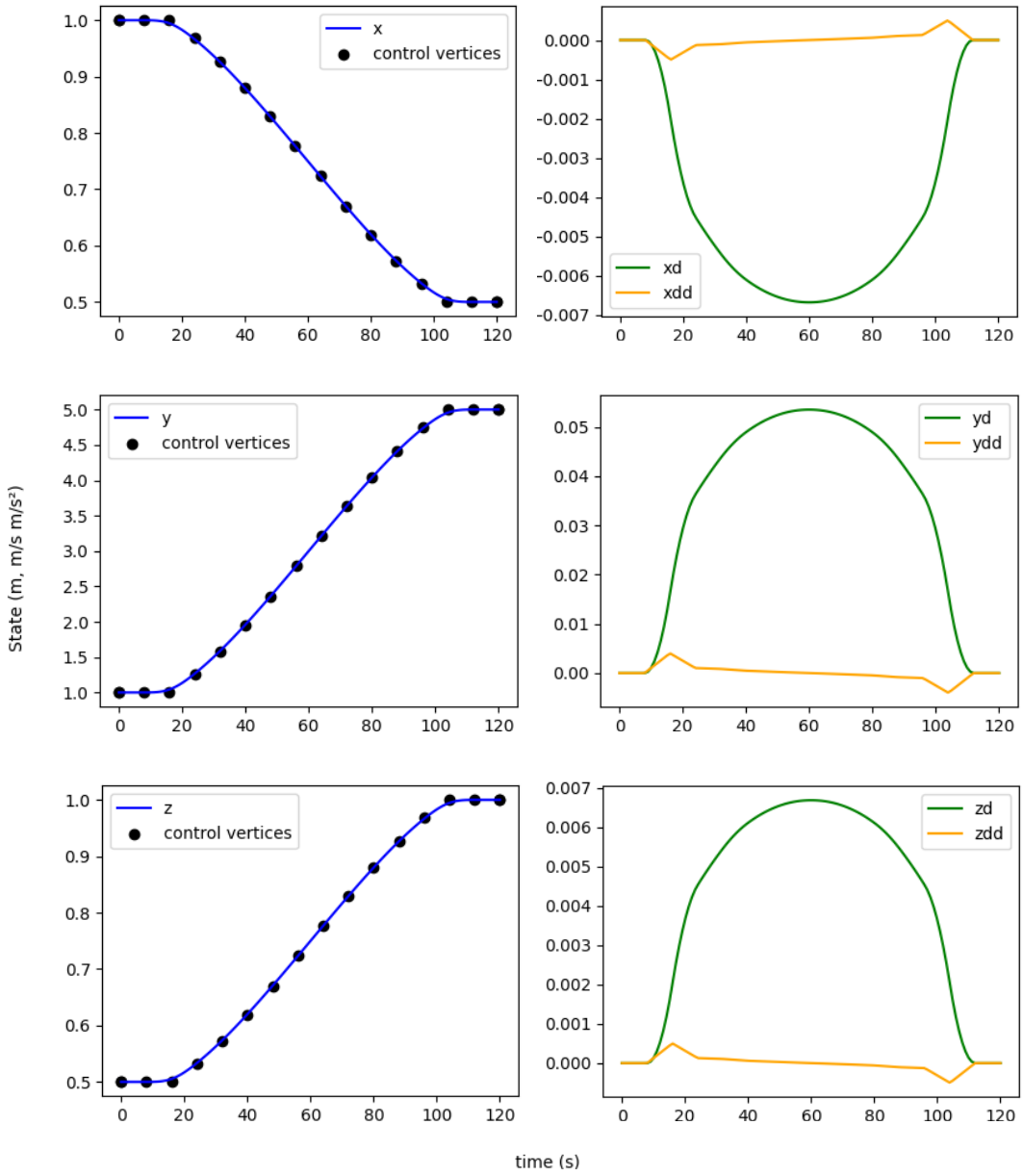


Figure 3.2: Unconstrained trajectory in $\mathbb{R}^3$ (B-Splines and derivatives)

Constraints, obstacles, and local minima

Motion constraints and obstacles are now included. The position $\mathbf{t}$, velocity $\mathbf{v}$, and force $\mathbf{F}$ are constrained to

$$\begin{bmatrix}0\\0\\0\end{bmatrix} \leq \mathbf{t} \leq \begin{bmatrix}1.5\\6.4\\1.7\end{bmatrix}, \begin{bmatrix}-0.1\\-0.1\\-0.1\end{bmatrix} \leq \mathbf{v} \leq \begin{bmatrix}0.1\\0.1\\0.1\end{bmatrix}, \begin{bmatrix}-0.849\\-0.406\\-0.486\end{bmatrix} \leq \mathbf{F} \leq \begin{bmatrix}0.849\\0.406\\0.486\end{bmatrix}. \tag{3.3}$$

Three spheres of radius 0.25 [m] are placed in the central region of the workspace, and the geometry of the robot is also represented as a sphere of 0.25 [m]. This scenario is referred to as "Three Spheres".
The choice of spheres (plus capsules, in the following sections) is due to the continuity of their surfaces. In contrast, a cube's edges and corners make the cube's surface discontinuous, which causes issues in the computation of the gradients of the collision avoidance constraints.
The energy-optimal trajectory the optimizer finds now depends on the provided initial guess of the free control vertices $\mathbf{p}_{\text{free}}$. The optimizer might even not converge to a solution, which means that the trajectory does not satisfy the constraints (i.e., it is not feasible) when any one of the optimizer's stopping criteria is met. Figure 3.3 shows the 934 trajectories that converged to a solution out of 1000 randomly generated initial guesses (Monte Carlo approach). The trajectories are grouped by cost and fall into the cost bins from the histogram in Fig. 3.4. Since the unconstrained optimal solution (straight line from root to goal) collides with the central sphere, all solutions found follow an arch pattern to deviate from the sphere. The "arching direction" depends on the initial guess. The blue solutions arching to the left (direction $-x$) in Fig. 3.3 have the lowest costs, falling into the first cost bin from Fig. 3.4. They also are the most commonly found solutions, considering the 800 hits on the first cost bin (out of the total 934 solutions found).

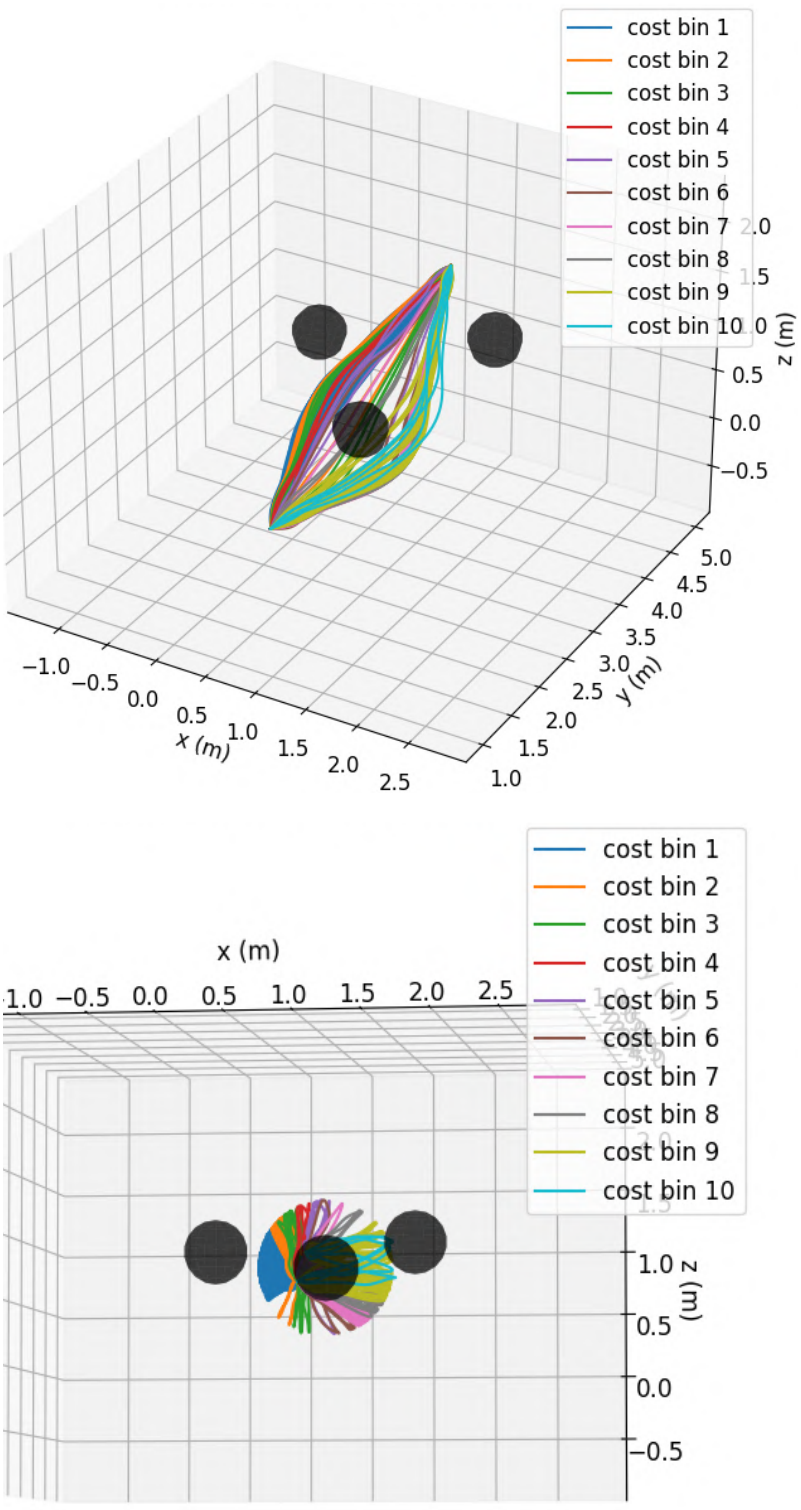


Figure 3.3: Local minima for the Three Spheres scenario (from to viewing angles).

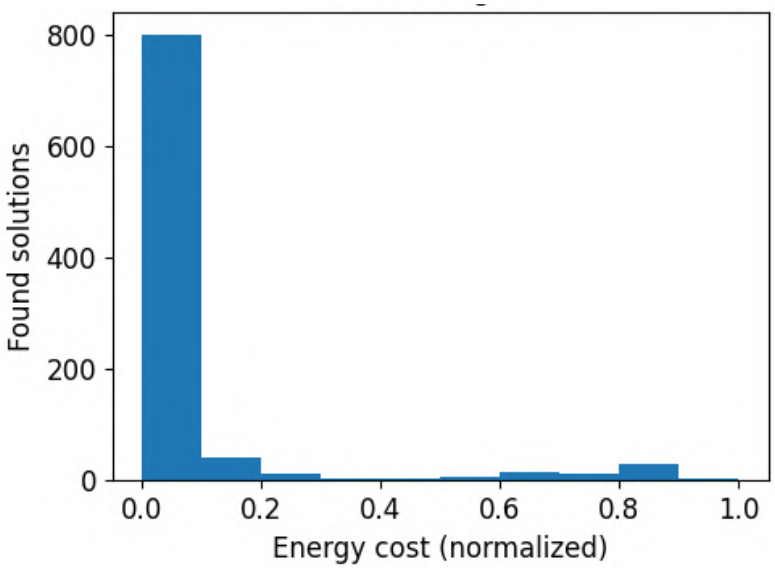


Figure 3.4: Cost histogram for the Three Spheres scenario.

More obstacles, more local minima

The previous analysis is repeated for a scenario with more obstacles, which causes a larger variety of local minima. This scenario is referred to as the "Cluster" scenario. The robot's geometry is momentarily treated as a point to highlight the active collision avoidance constraints.
Figure 3.5 shows the cost histogram of the solutions found, this time with a larger number of bins to help identify the different local minima.
Figure 3.6 shows only the most recurring solutions, i.e., the ones that fall into cost bins with over ten solutions (for better visibility). In contrast to Fig, 3.3, the pattern of the solutions is less clear here. The blue solution arching toward $+x$ and $-z$ has the lowest cost, and the orange solution arching toward $-x$ and $+z$ occurs most often. That evidences how only a few "good" initial guesses lead to the lowest-cost solution and thus motivates using RRT*-GBO to provide those, in contrast to the "brute force" Monte Carlo approach.

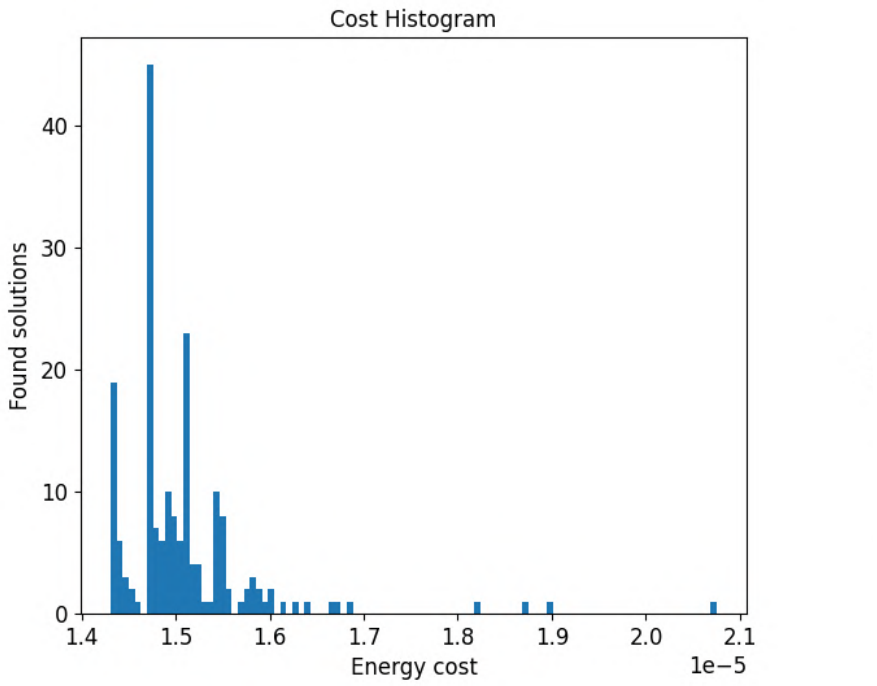


Figure 3.5: Cost histogram for the Cluster scenario.

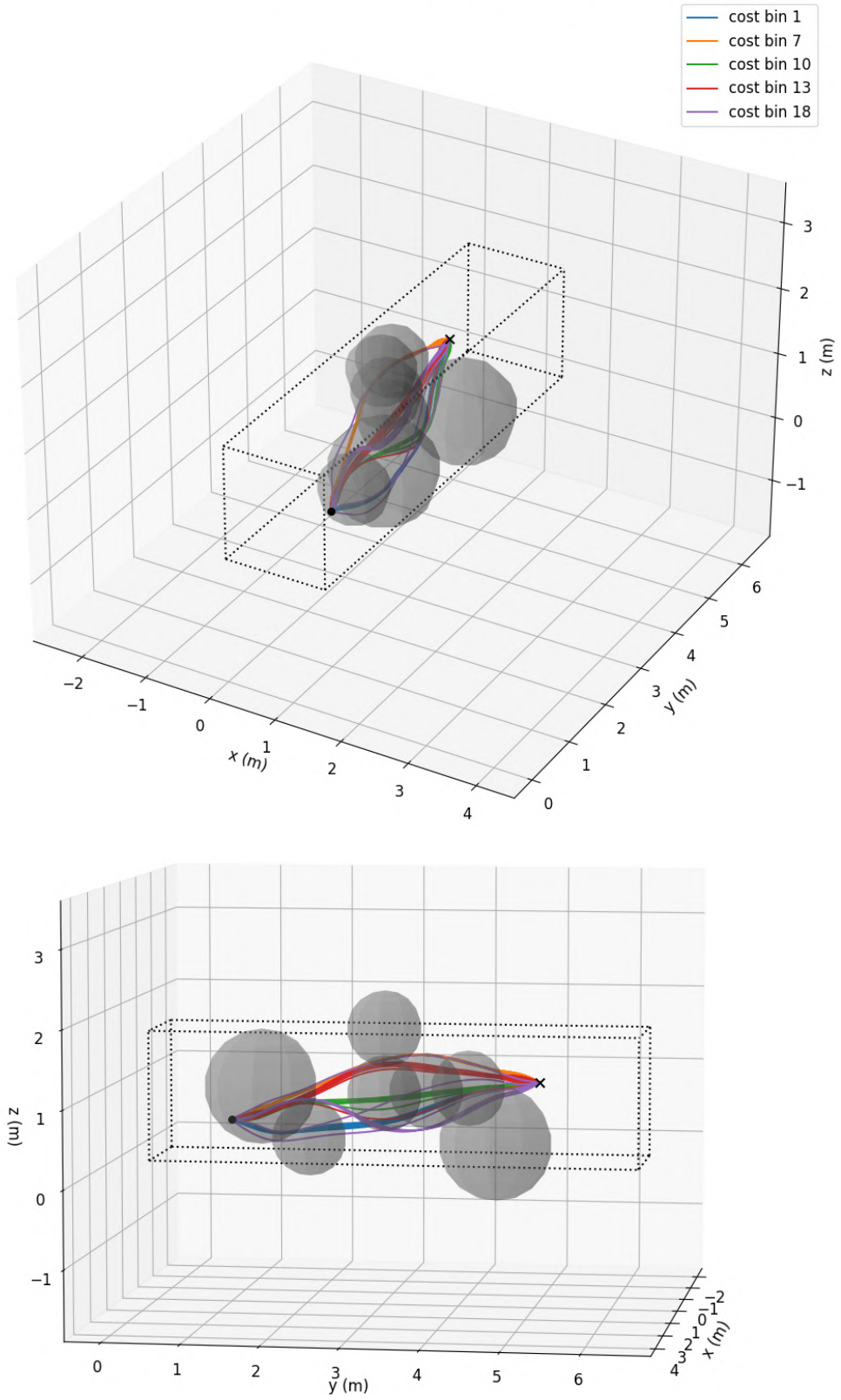


Figure 3.6: Local minima for the Cluster scenario (from two viewing angles).

RRT*-GBO to handle local minima

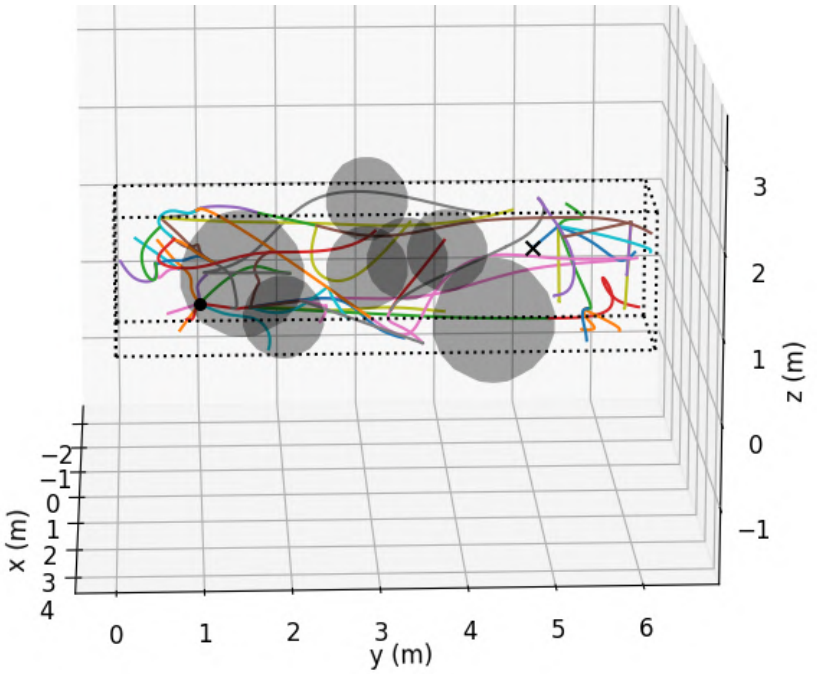
Figure 3.7 shows a result of applying the RRT*-GBO algorithm described in Sec. 3.7 to Cluster scenario. The edge B-Spline has a degree $p = 2$, with $n_{\text{free}} = 1$ free control vertex, and duration $t_f = 30$ [s]. The tuning parameters

$$r_{\text{prune}} = 0.5 \tag{3.4}$$

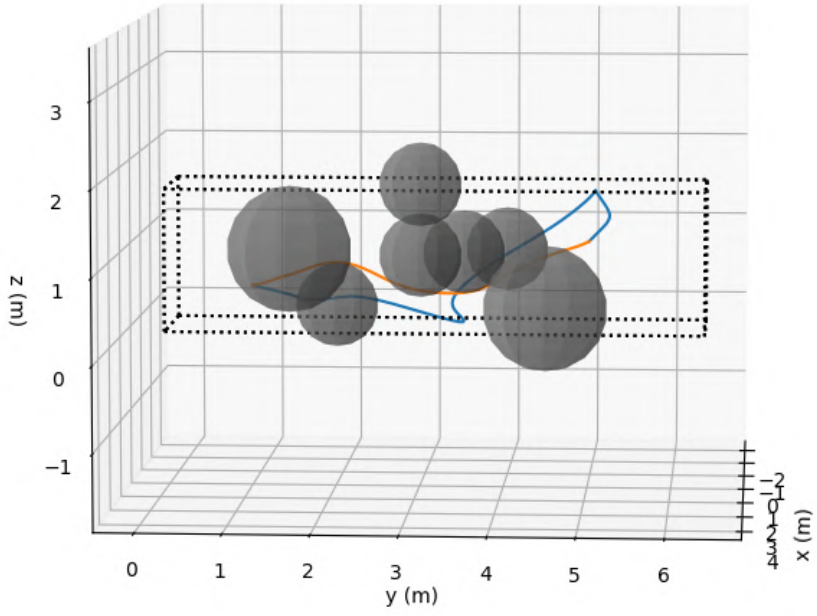
$$r_{\text{ball}} = 2 \tag{3.5}$$

$$r_{\text{connect}} = 3 \tag{3.6}$$

are used.
Figure 3.7(a) shows the (randomly colored) tree of edges constructed by the algorithm, which contains a path of edges from the root (black circle marker) to the goal (black "x" marker). This path changes throughout the iterations of the algorithm, having its edges replaced by cheaper ones. Figure 3.7(a) shows this path at a given algorithm iteration in blue, which is fed to the optimizer to find an optimized trajectory/solution in orange. The optimization from the blue trajectory to the orange trajectory is the smoothing step described in Sec. 2.2.5.
The following sections explore how RRT*-GBO performs with various tuning parameter combinations in different scenarios.



(a) Tree of edges.

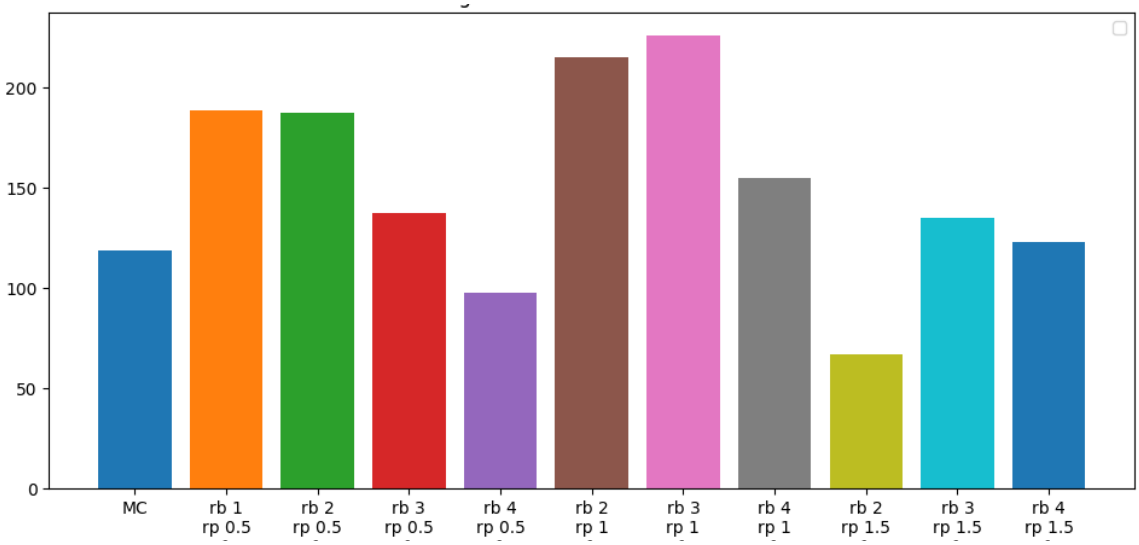


(b) Path of edges from root to goal and smoothed solution.

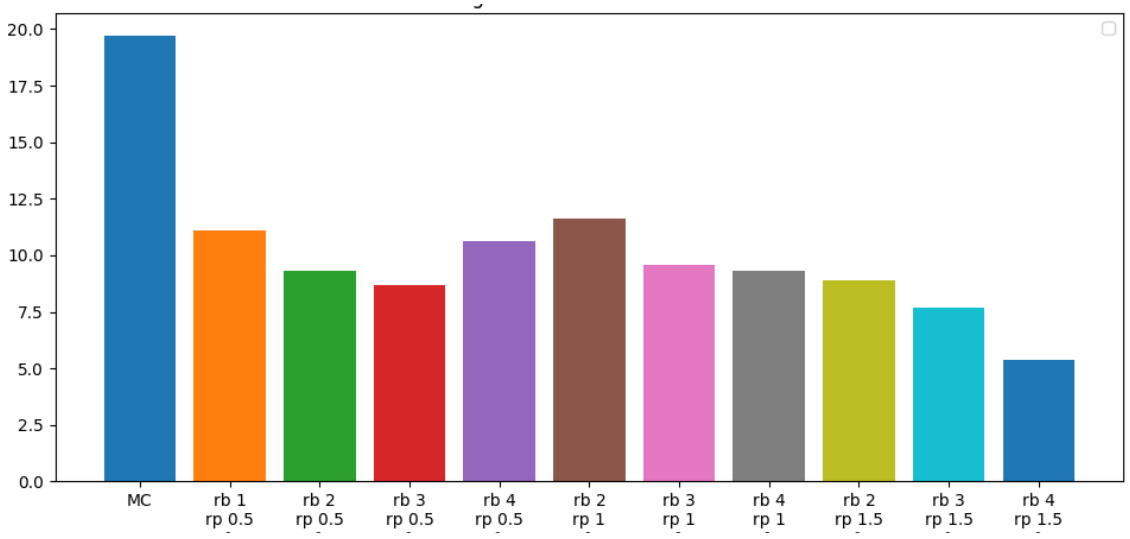
Figure 3.7: RRT*-GBO example in the Cluster scenario.

RRT*-GBO performance analysis

For the trajectory optimization in the Cluster scenario, the following experiment is conducted: 50 random initial guesses of the free control vertices p_{free} are optimized by the solver until any one of the stopping criteria is met. The converged solutions (feasible trajectories) are collected. The total computation time taken by the optimizer for all 50 guesses is used as the maximal execution time for a run of the RRT*-GBO algorithm with a given choice of the tuning parameters r_{ball} and r_{prune}. The time-limited RRT*-GBO execution is repeated for various choices of the tuning parameters. The converged solutions are collected for each of them. The whole procedure is repeated 10 times for statistical consistency, and the results are shown in Figures 3.8 and 3.9. The abbreviation "MC" stands for Monte Carlo, referring to the randomly generated initial guesses, "rb" for r_{ball} and "rp" for r_{prune}.
Figure 3.8(b) shows that the Monte Carlo approach finds more solutions, but
Fig. 3.8(a) shows that RRT*-GBO finds the best solution (i.e., the one with the lowest cost) faster when choosing a parameter combination of $r_{\text{ball}} = 4$ and $r_{\text{prune}} = 0.5$ (purple bar). The choice of $r_{\text{ball}} = 2$ and $r_{\text{prune}} = 1.5$ (olive green bar) appears to find the best solution even faster. However, Fig. 3.9 shows the best solution found by each case across all the 10 repetitions and reveals that the overall best solution found by the $r_{\text{ball}} = 2$ and $r_{\text{prune}} = 1.5$ case (again in olive green) is a local minimum belonging to the cost bin number 10 from Figures 3.5 and 3.6. All the other cases find the lowest-cost solution in the first cost bin from Figures 3.5 and 3.6.
Therefore, the case $r_{\text{ball}} = 4$ and $r_{\text{prune}} = 0.5$ (purple) provides better results than Monte Carlo and other parameter combinations.



(a) Average time to find the best solution (in seconds).



(b) Average number of solutions found

Figure 3.8: Monte Carlo and RRT*-GBO statistics in the Cluster scenario.

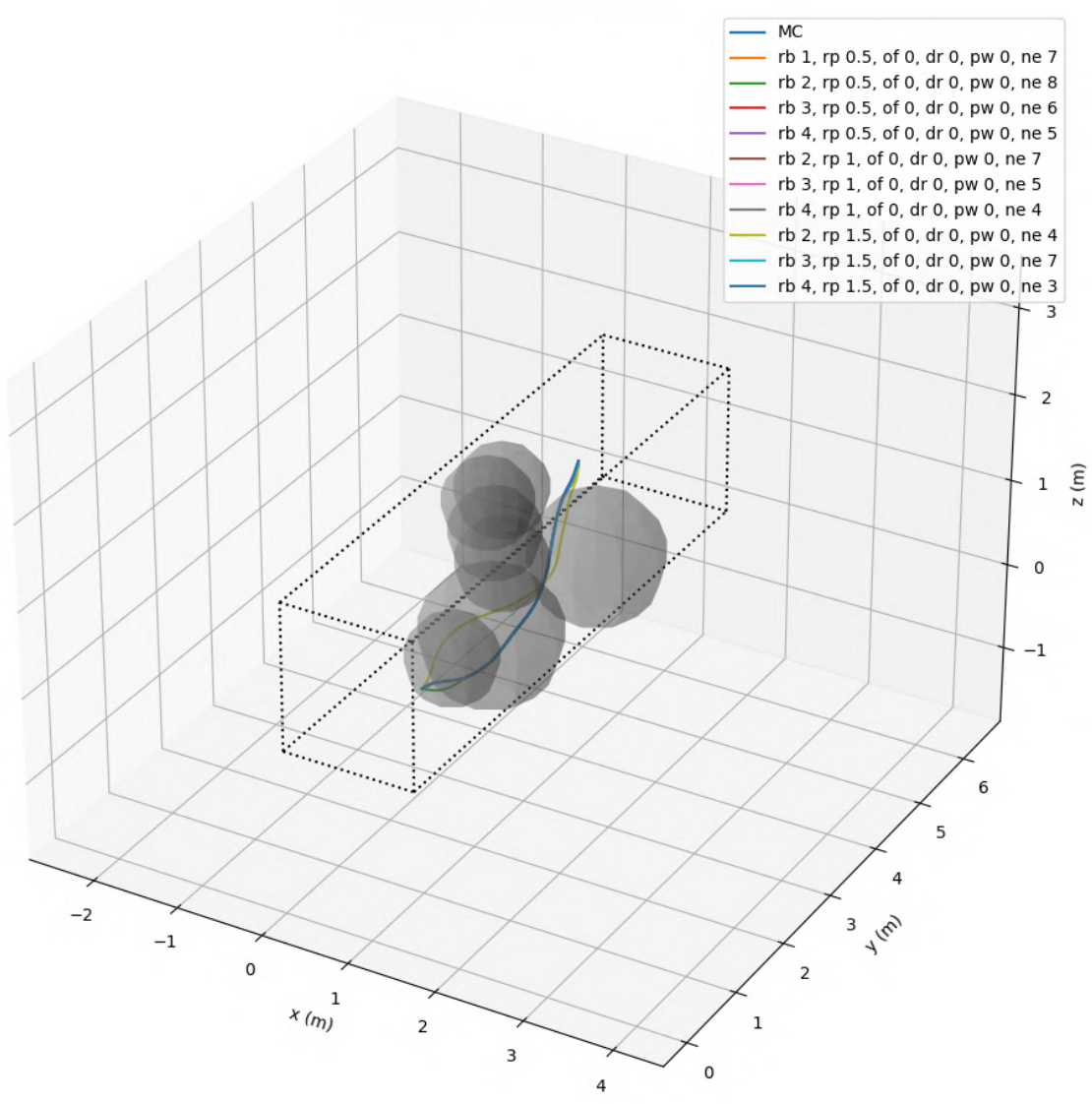


Figure 3.9: Monte Carlo an RRT*-GBO best solutions in the Cluster scenario.

The same statistical analysis done in the Cluster scenario is conducted in a new "Flowers" scenario shown in Fig. 3.10, with $\mathbf{t}_{\text{root}} = \begin{bmatrix}1 & 0.2 & 0.2\end{bmatrix}^T$ and $\mathbf{t}_{\text{goal}} = \begin{bmatrix}0.5 & 6 & 1\end{bmatrix}^T$. The results are shown in Figures 3.11 and 3.12. Again, 3.11(b) shows that Monte Carlo finds more solutions and Fig. 3.11(a) shows that RRT*-GBO finds the best solution (i.e., the one with the lowest cost) faster when choosing a parameter combination of $r_{\text{ball}} = 2$ and $r_{\text{prune}} = 1.5$ (olive green bar). Furthermore, Fig. 3.12 shows that every case's overall best solution is the same.
Thus, the case $r_{\text{ball}} = 2$ and $r_{\text{prune}} = 1.5$ (olive green) provides better results than Monte Carlo and other parameter combinations.

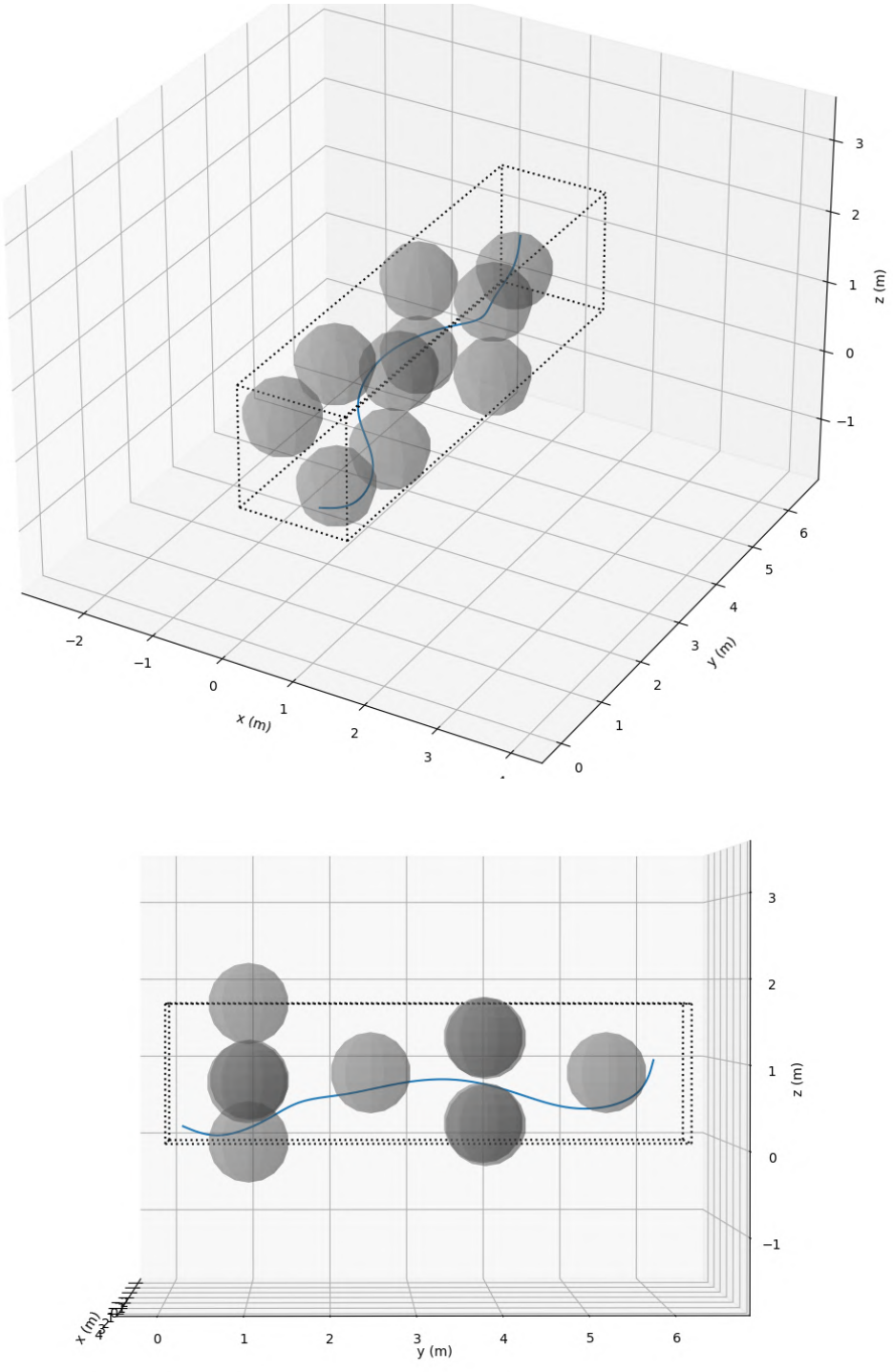
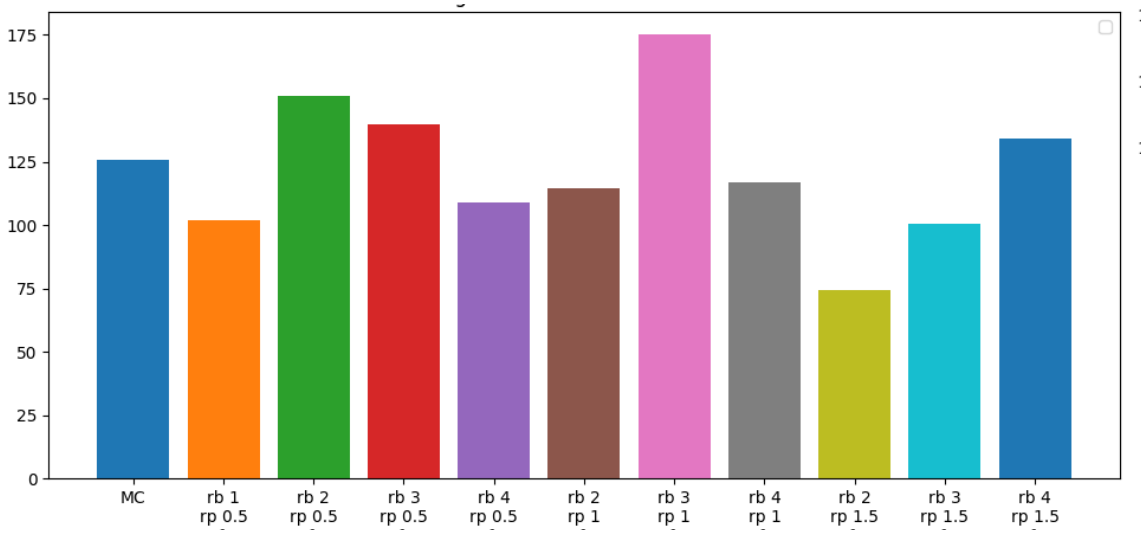
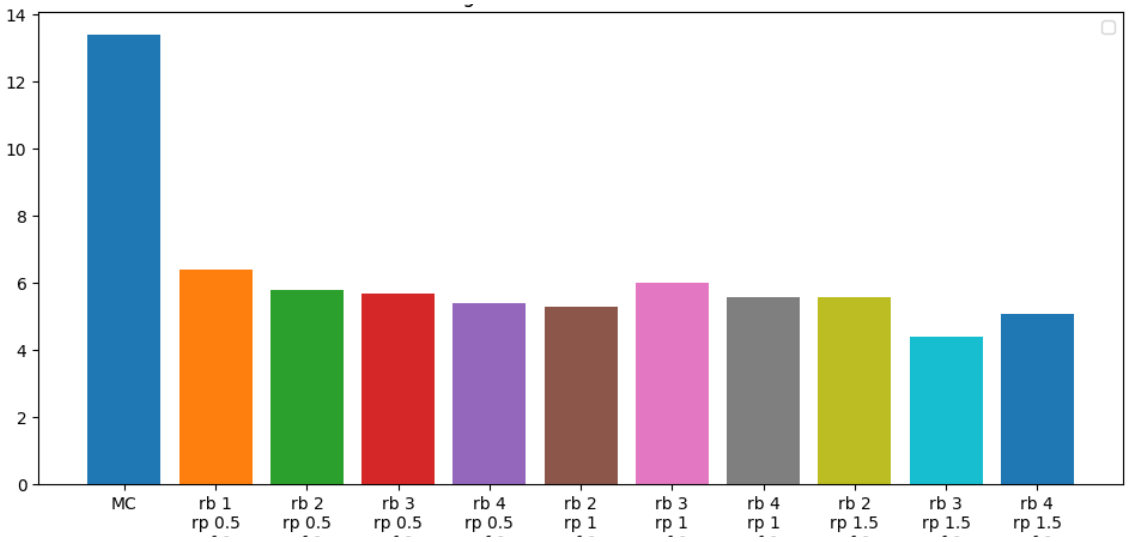


Figure 3.10: Flowers scenario (from two viewing angles).



(a) Average time to find the best solution (in seconds).



(b) Average number of solutions found

Figure 3.11: Monte Carlo an RRT*-GBO statistics in the Flowers scenario.

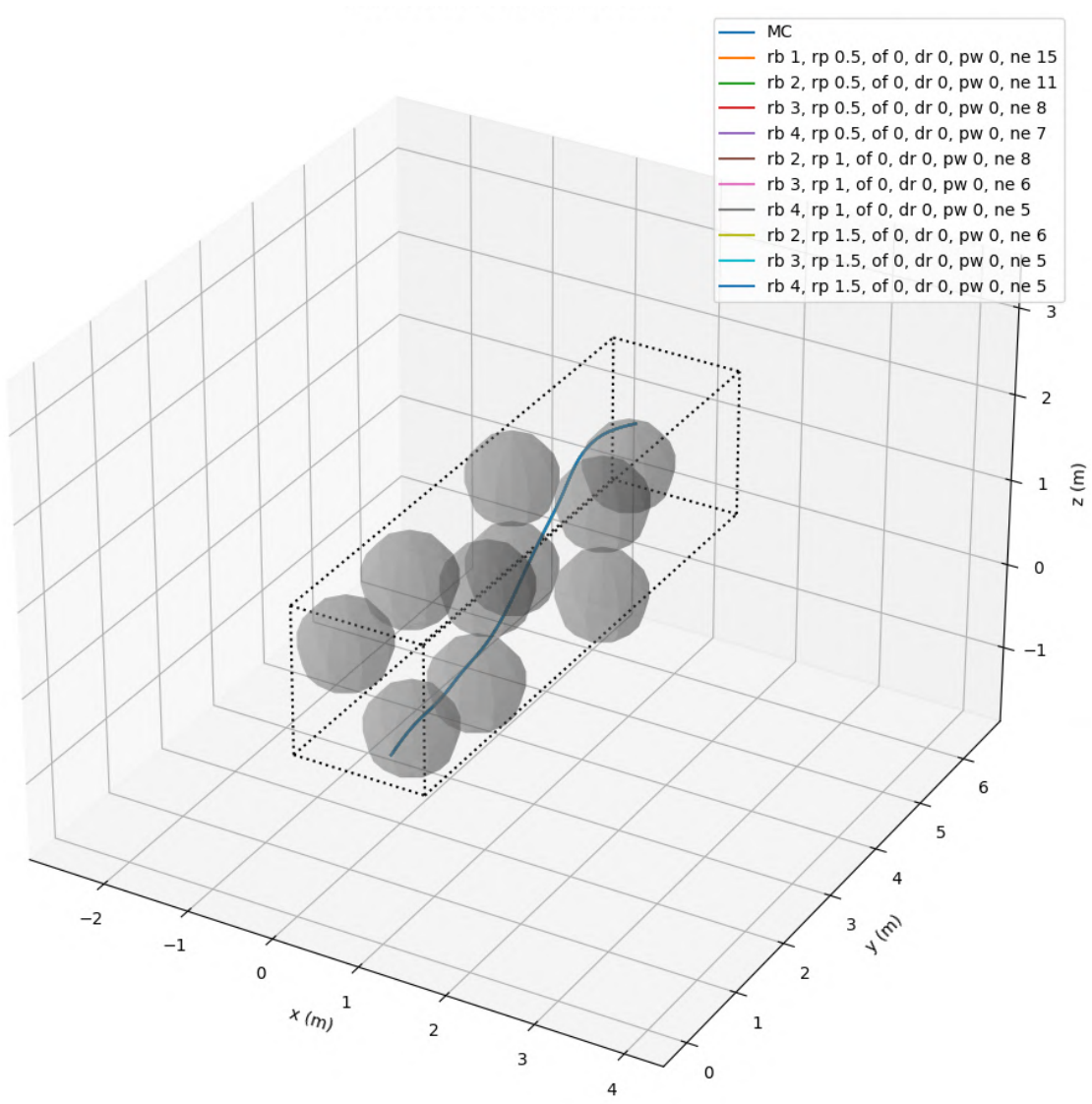


Figure 3.12: Monte Carlo and RRT*-GBO best solutions in the Flowers scenario.

3.1.2 Motion in SE(3)

Rotations in SO(3) are now introduced, being parameterized by three additional B-Splines using the angle-axis representation. The rotation B-Splines have the same properties as the translation B-Splines presented in Sec. 3.1.1, apart from the degree $p_{\text{rotation}} = 2$ in contrast to $p_{\text{translation}} = 3$.
Further motion constraints

$$\begin{bmatrix} -0.1 \\ -0.1 \\ -0.1 \end{bmatrix} \leq \boldsymbol{\omega} \leq \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix}, \begin{bmatrix} -0.0849 \\ -0.0406 \\ -0.0486 \end{bmatrix} \leq \boldsymbol{\tau} \leq \begin{bmatrix} 0.0849 \\ 0.0406 \\ 0.0486 \end{bmatrix} \tag{3.7}$$

are added.
Figures 3.13 and 3.14 show the energy-optimal trajectory from a root pose

$$\mathbf{s}_{\text{root}} = \begin{bmatrix} 1 & 0.2 & 0.2 & 0 & 0 & 0 \end{bmatrix}^T, \tag{3.8}$$

to a goal pose

$$\mathbf{s}_{\text{goal}} = \begin{bmatrix} 0.5 & 6 & 1 & \pi/2 & \pi/2 & \pi/2 \end{bmatrix}^T, \tag{3.9}$$

in the absence of obstacles. The red, green, and blue dashes are the x, y, and z basis vectors of a frame attached to the robot's center of mass, depicting the robot's orientation along the trajectory.
Figure 3.15 shows the translation B-Splines and Figure 3.16 shows the rotation B-Splines, both having a bell-shaped velocity profile.

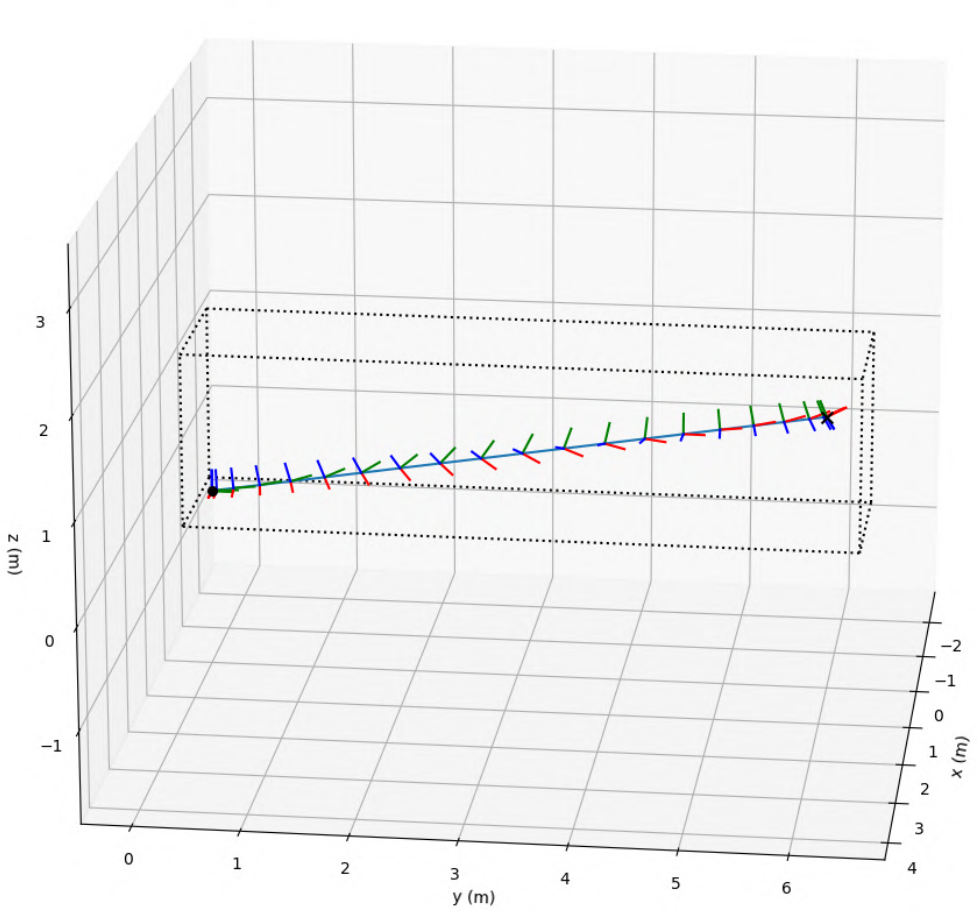


Figure 3.13: Trajectory in SE(3) (first viewing angle).

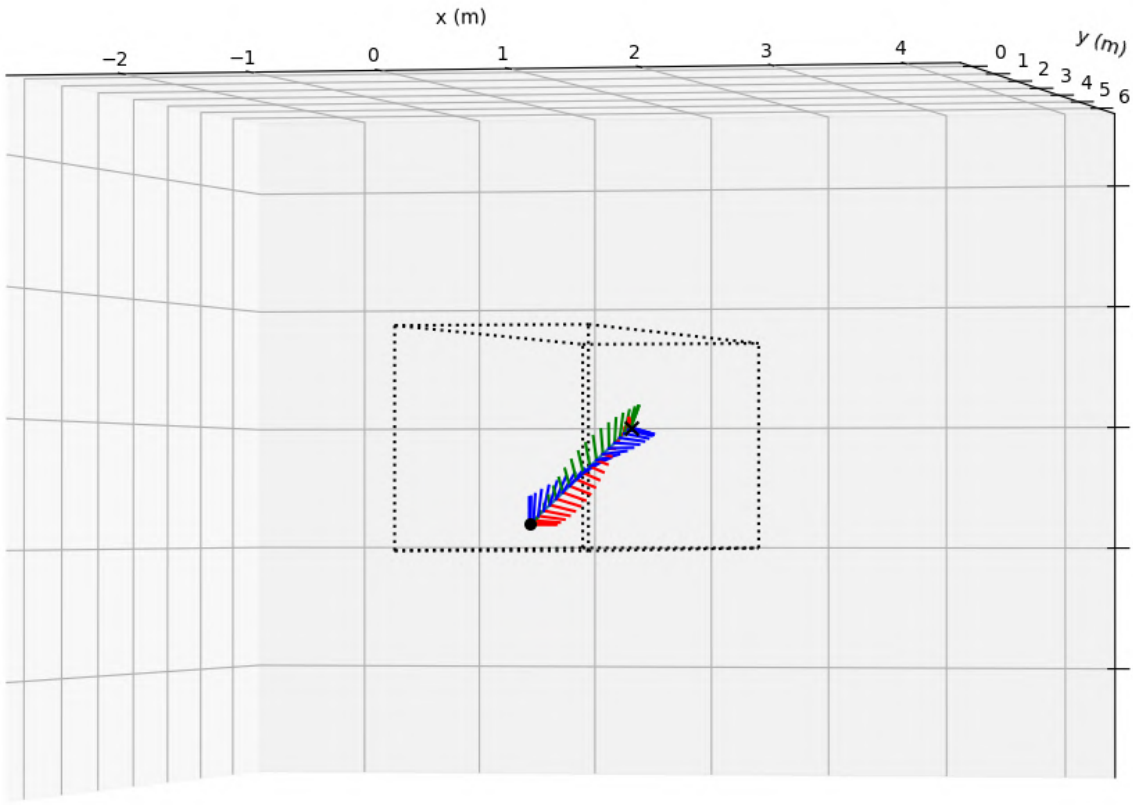


Figure 3.14: Trajectory in SE(3) (second viewing angle).

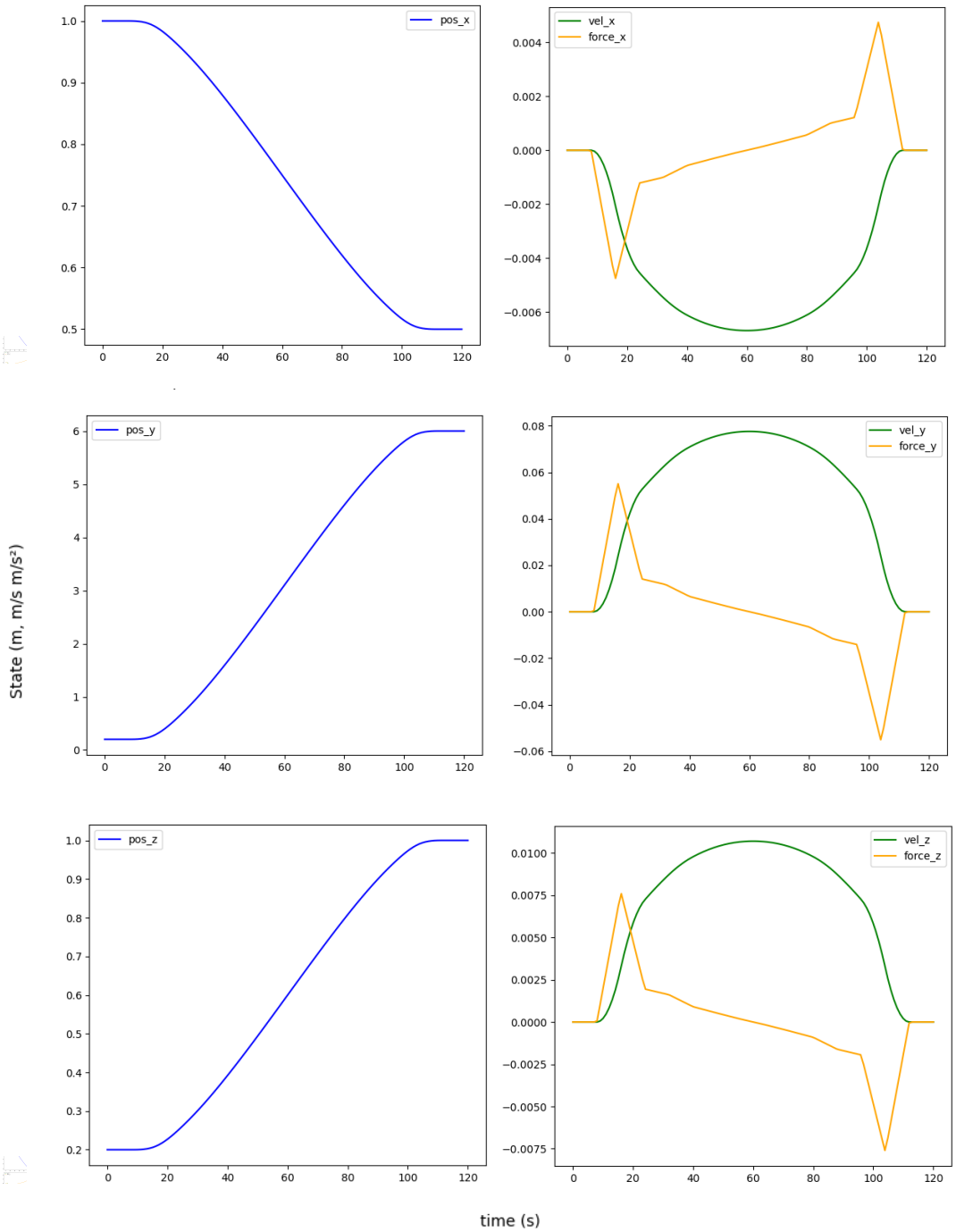


Figure 3.15: Trajectory in SE(3) (Translation B-Splines and derivatives).

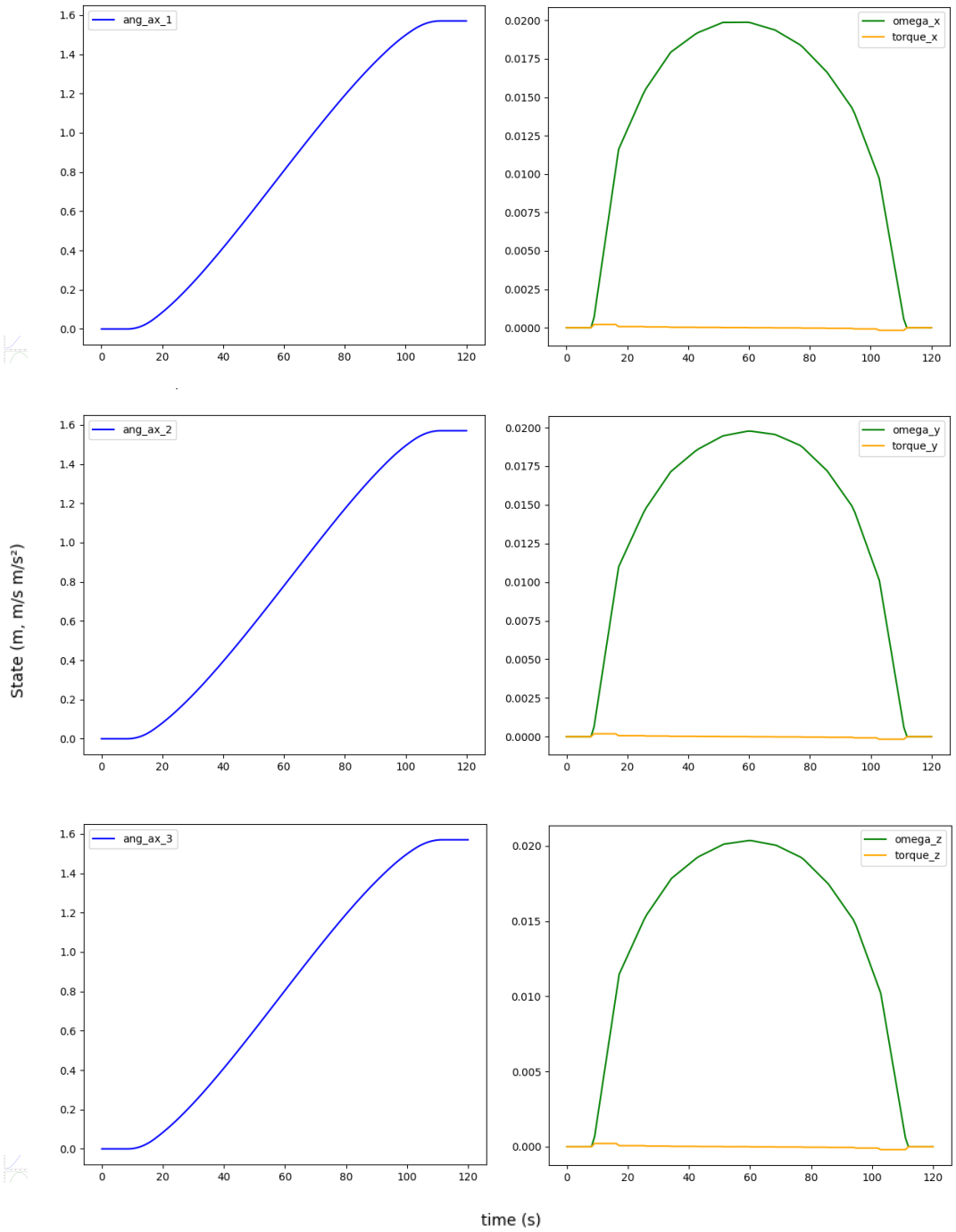


Figure 3.16: Trajectory in SE(3) (Rotation B-Splines and derivatives).

Obstacles

The robot geometry is momentarily modeled as a capsule to show that the optimizer finds solutions in which the robot's orientation is fundamental for feasibility. Capsule-shaped obstacles are included, and this scenario is called the Narrow Passage scenario.
Figure 3.17 shows the straight-line initial guess of the trajectory from a root pose

$$\mathbf{s}_{\text{root}} = \begin{bmatrix} 1 & 0.2 & 0.2 & 0 & 0 & 0 \end{bmatrix}^T, \tag{3.10}$$

to a goal pose

$$\mathbf{s}_{\text{goal}} = \begin{bmatrix} 1 & 6 & 1 & 0 & 0 & 0 \end{bmatrix}^T, \tag{3.11}$$

which causes collisions and it is not feasible. Figures 3.18 and 3.19 show the feasible, energy-optimal solution found by the optimizer.

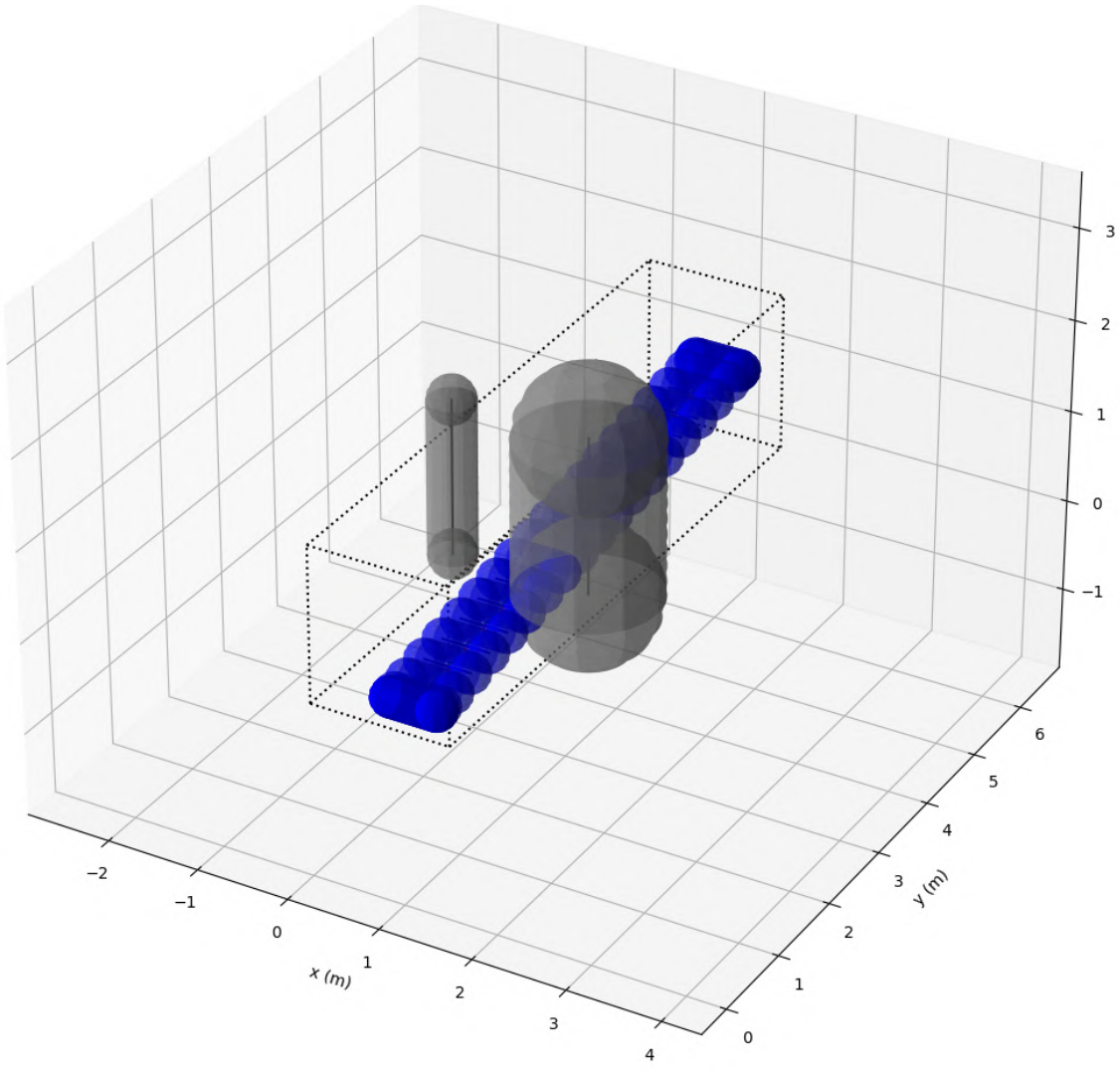


Figure 3.17: Initial guess for a trajectory in the Narrow Passage. The robot is depicted in blue for different instants of the trajectory.

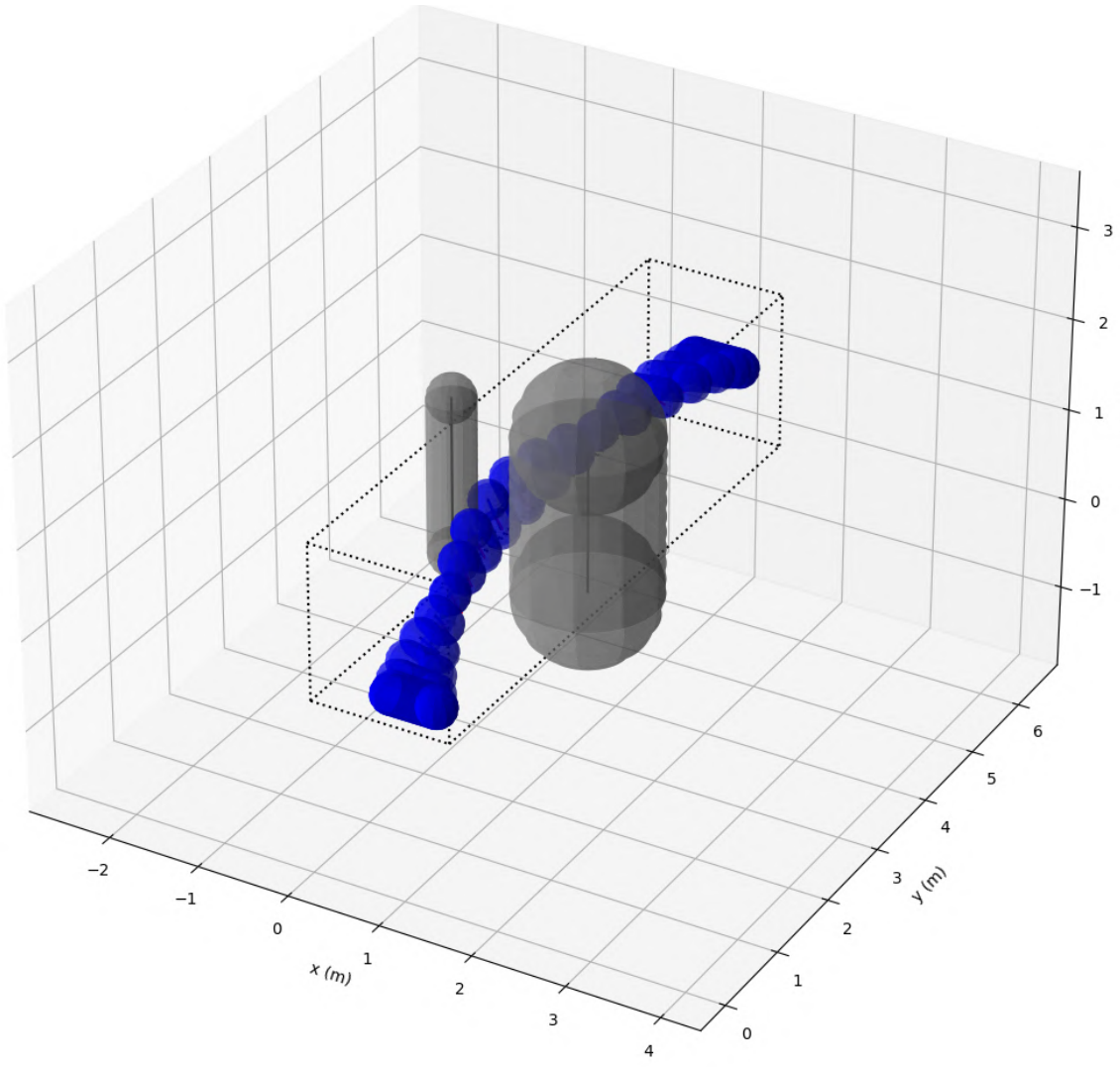


Figure 3.18: Optimized trajectory in the Narrow Passage (first view). The robot is depicted in blue for different instants of the trajectory.

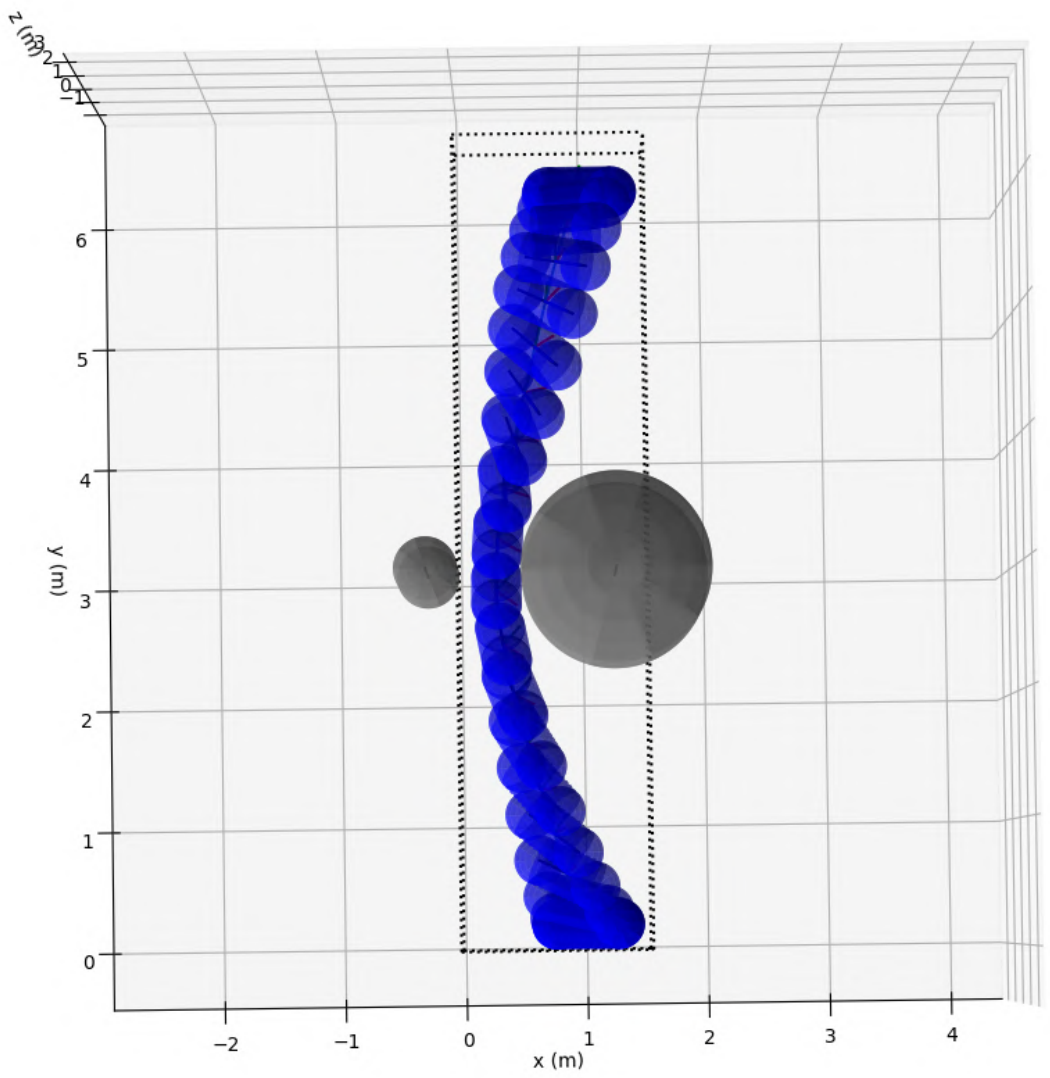


Figure 3.19: Optimized trajectory in the Narrow Passage (second view). The robot is depicted in blue for different instants of the trajectory.

RRT*-GBO performance analysis

A hard, cluttered scenario, referred to as the Maze scenario, is now used to evaluate RRT*-GBO exploration capability compared to the "brute force" Monte Carlo approach.
We define

$$r_{\text{connect}} = r_{\text{ball}}, \tag{3.12}$$

and a dynamic edge B-Spline duration

$$t_{f,\text{edge}} = 10 \cdot r_{\text{ball,translation}} + 30 \quad [\text{s}] \tag{3.13}$$

.
For

$$\mathbf{s}_{\text{root}} = \begin{bmatrix} 1.25 & 0.2 & 0.2 & 0 & \pi/2 & 0 \end{bmatrix}^T, \tag{3.14}$$

$$\mathbf{s}_{\text{goal}} = \begin{bmatrix} 0.25 & 0.2 & 1.5 & 0 & \pi/2 & 0 \end{bmatrix}^T, \tag{3.15}$$

a trajectory duration

$$t_f = 200 \quad [\text{s}], \tag{3.16}$$

and the tuning parameters

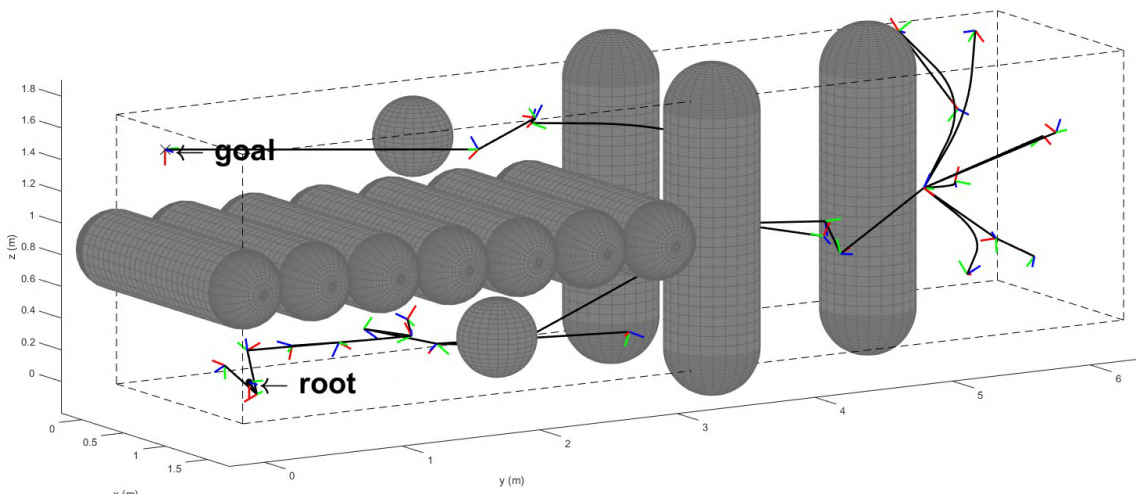
$$r_{\text{ball,translation}} = 2, \tag{3.17}$$

$$r_{\text{prune,translation}} = 0.5, \tag{3.18}$$

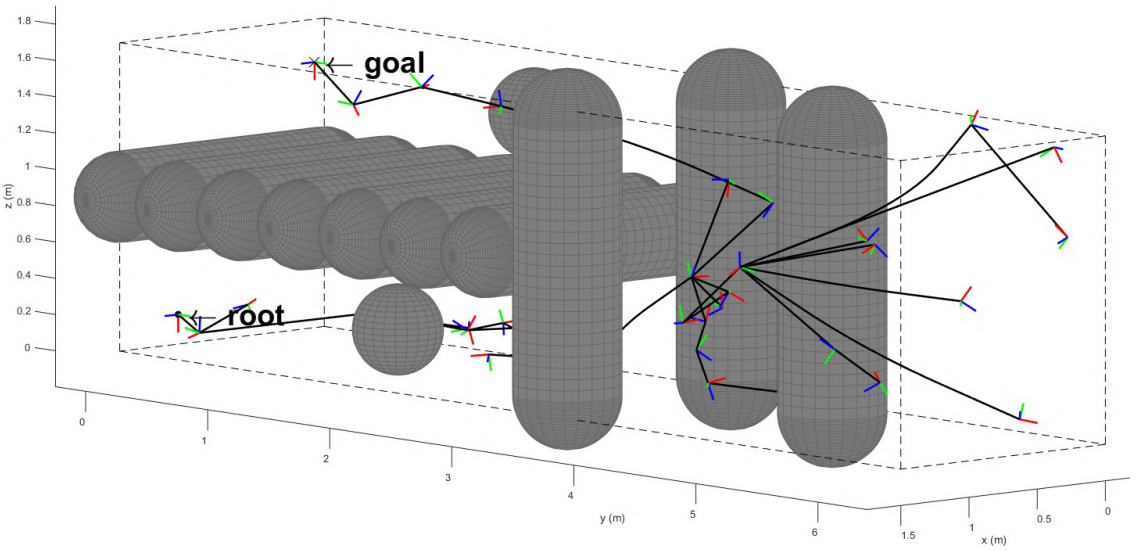
$$r_{\text{ball,rotation}} = \pi/2, \tag{3.19}$$

$$r_{\text{prune,rotation}} = \pi/12, \tag{3.20}$$

Fig. 3.20 shows an RRT*-GBO tree of edges (black stretches for translation between nodes, red-green-blue dashes for the orientation at the nodes) built in the Maze scenario. Fig. 3.21 shows the smoothing of a path of edges from the root to the goal (in black) into the final solution (in orange). Fig. 3.22 shows the robot along the final solution.

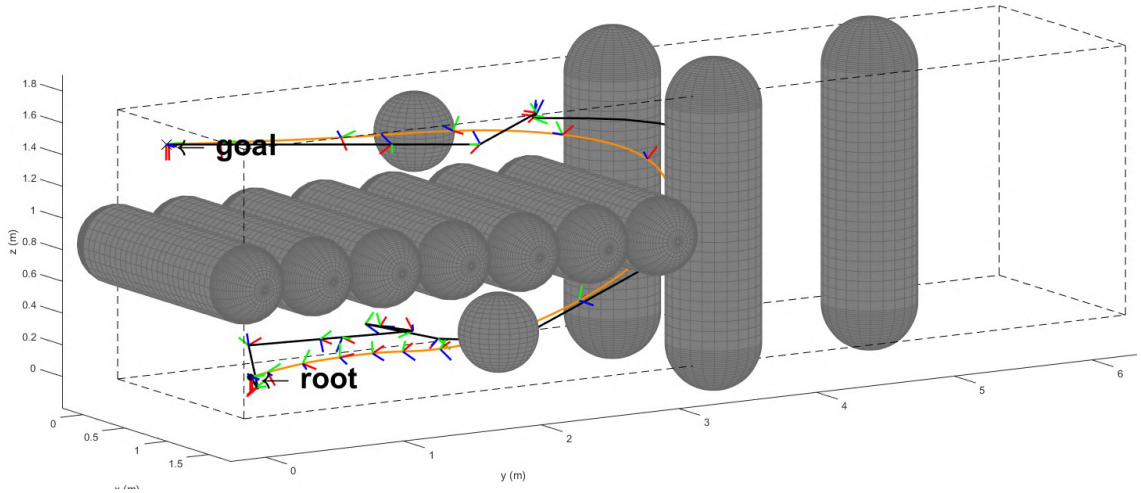


(a) View 1

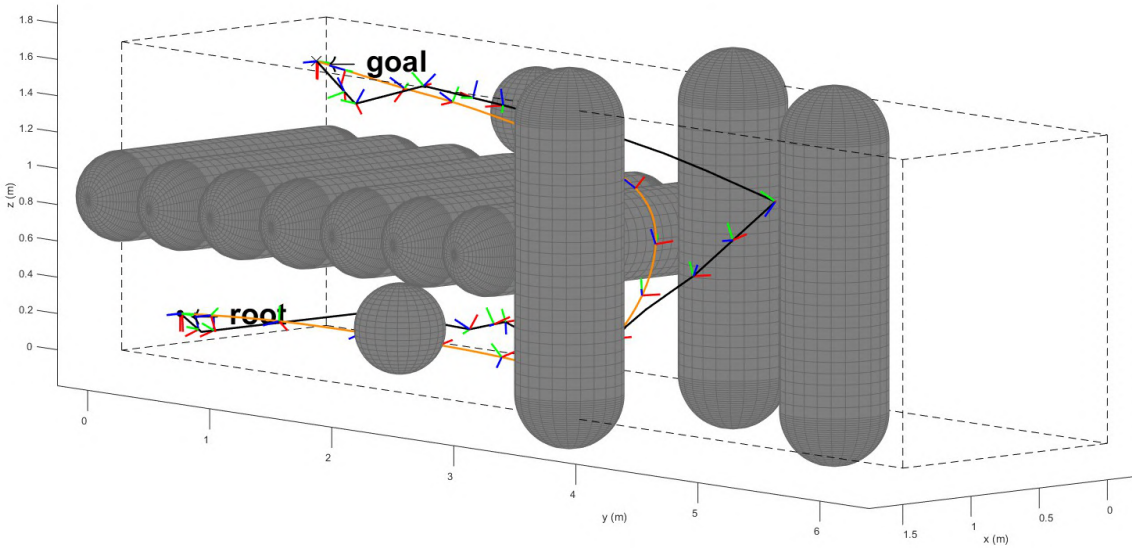


(b) View 2

Figure 3.20: Maze Scenario: RRT*-GBO tree of edges.

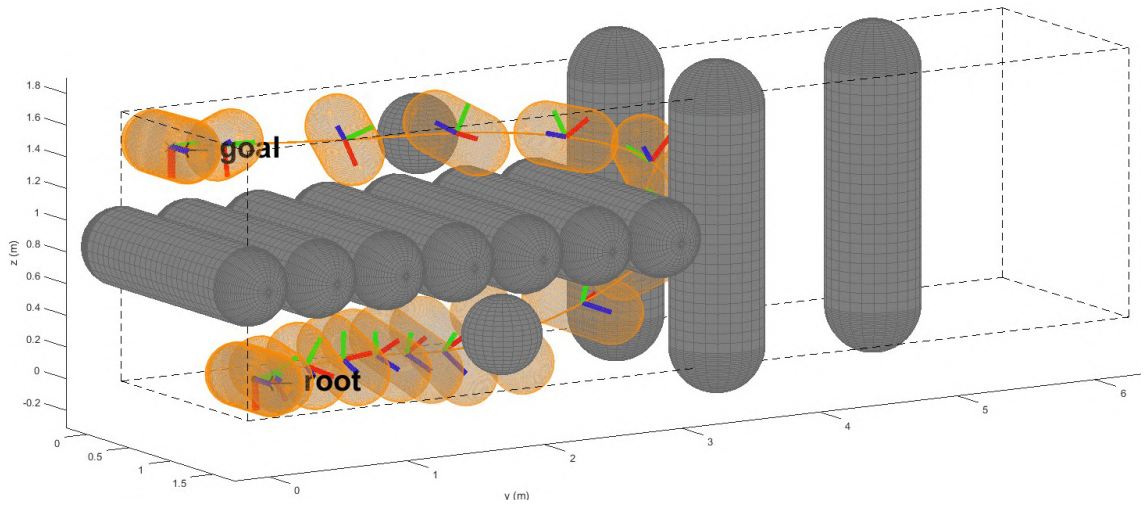


(a) View 1

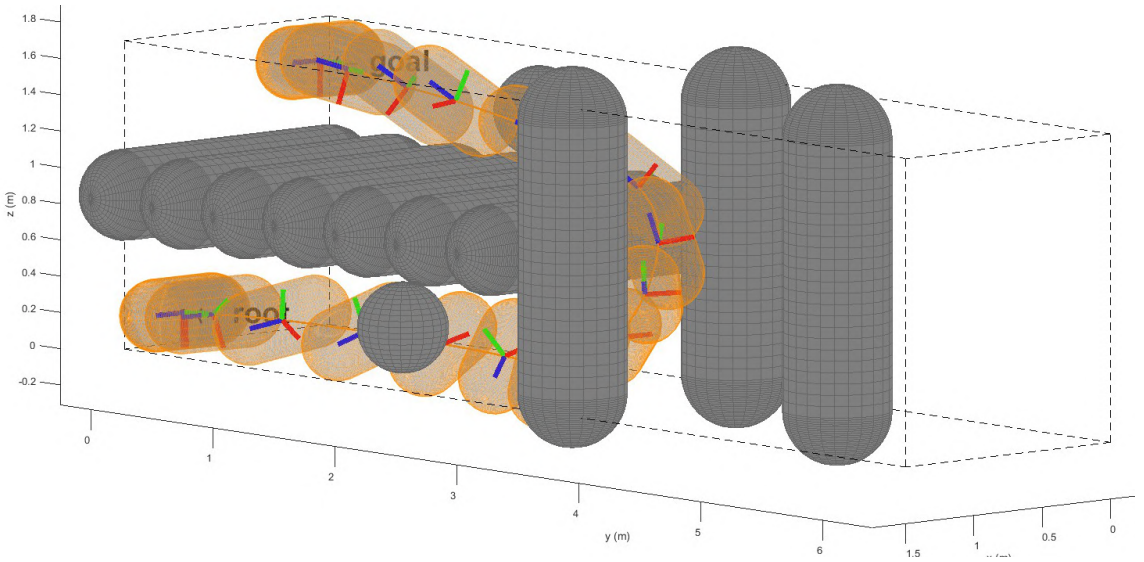


(b) View 2

Figure 3.21: Maze Scenario: RRT*-GBO smoothing. The path of edges from root to goal is in black, and the final solution is in orange.



(a) View 1



(b) View 2

Figure 3.22: Maze Scenario: RRT*-GBO solution. The robot is depicted in orange for different instants of the trajectory.

For the Maze scenario, a Monte Carlo approach with 2000 random initial guesses is tested, using the following stopping criteria: $\Delta\Gamma_{\text{rel}} = 1e^{-6}$, $\Delta\mathbf{p}_{\text{free,rel}} = 1e^{-6}$, $n_{\text{maxiter}} = 1500$. The 2000 runs took a total computation time of 39h41m16s (over one and a half days). Out of those, a single feasible trajectory was found on the 604th trial, after 12h00min45s.
As a comparison, RRT*-GBO with different tuning parameter combinations gets executed for 30 minutes, 10 times. The statistical results over the 10 "batches" are shown in Figures 3.23 to 3.27. Figure 3.28 shows the best solutions for each parameter combination, which follow a similar pattern. The abbreviation "rbt" stands for $r_{\text{ball, translation}}$, "rbr" for $r_{\text{ball, rotation}}$, "rpt" for $r_{\text{prune, translation}}$, and "rpr" for $r_{\text{prune, rotation}}$.
We focus on the combination

$$r_{\text{ball, translation}} = 3,$$

$$r_{\text{ball, rotation}} = 180^\circ,$$

$$r_{\text{prune, translation}} = 1,$$

$$r_{\text{prune, rotation}} = 30^\circ,$$

shown in dark red for a deeper analysis. Figure 3.27 shows that it finds at least a solution for every run of the algorithm. Figure 3.26 shows that it finds more than 3 solutions in a run, on average (the best average among all cases). Figure 3.25 shows that the best solution found is cheaper than the one found in every other case, on average. Figure 3.23 shows that it takes an average of $\sim$ 700 seconds ($\sim$ 11.5 minutes) to find the first solution, which is a $\sim$ 1.4 factor of the lowest time among all cases (light orange bar). Figure 3.24 shows that it takes an average of $\sim$ 1400 seconds ($\sim$ 23 minutes) to find the best solution, which is (again) a $\sim$ 1.4 factor of the lowest time among all cases (light blue bar).
The dark red combination provides above-average results compared to the other parameter combinations evaluated. Compared to Monte Carlo, it achieves a 62-times computation time reduction to find a solution (12 hours against 11.5 minutes).

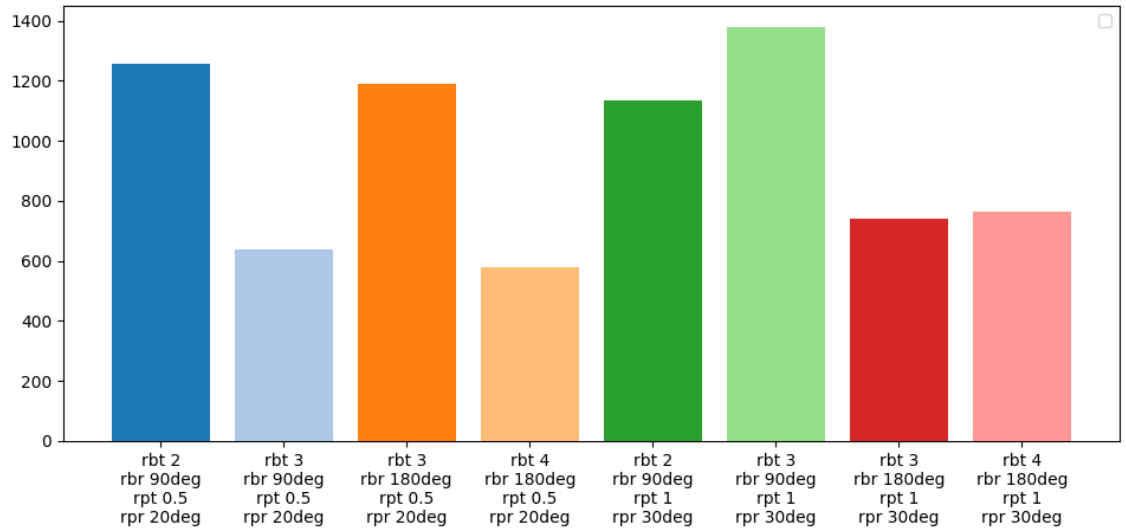


Figure 3.23: RRT*-GBO in the Maze scenario: average time taken to find the first solution (in seconds).

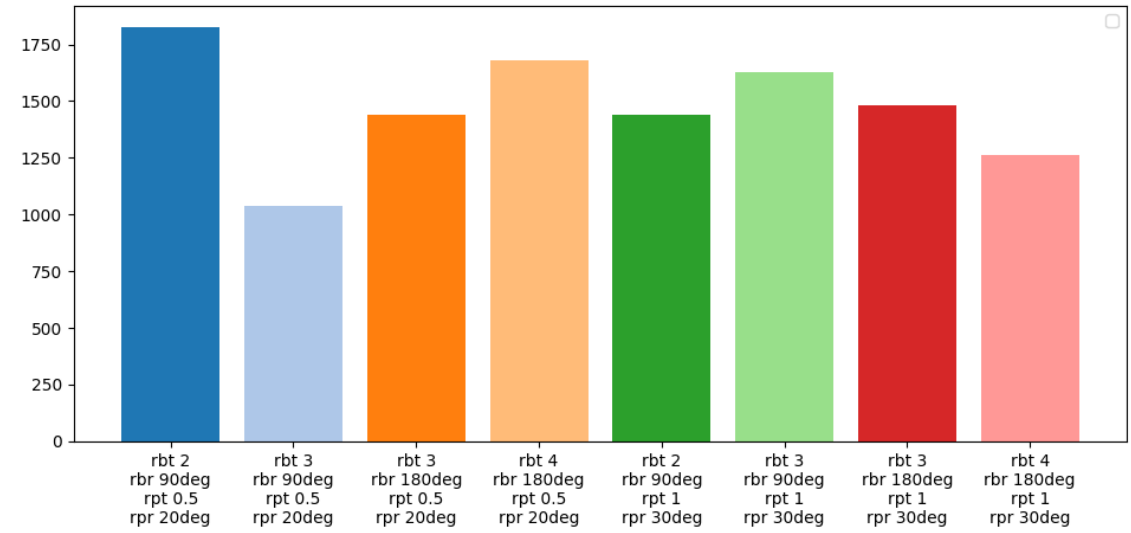


Figure 3.24: RRT*-GBO in the Maze scenario: average time taken to find the best solution (in seconds).

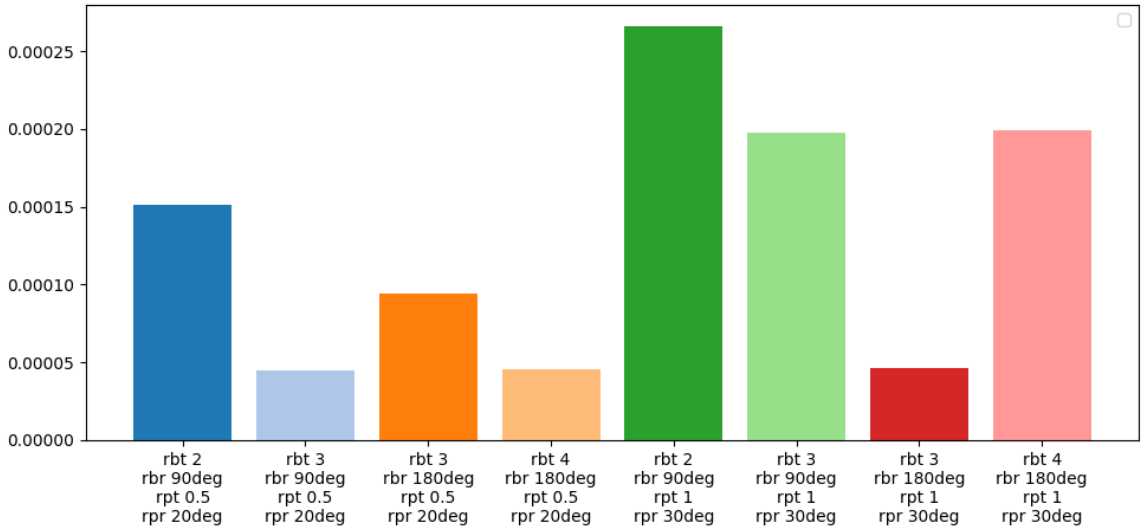


Figure 3.25: RRT*-GBO in the Maze scenario: average cost of the best solution found.

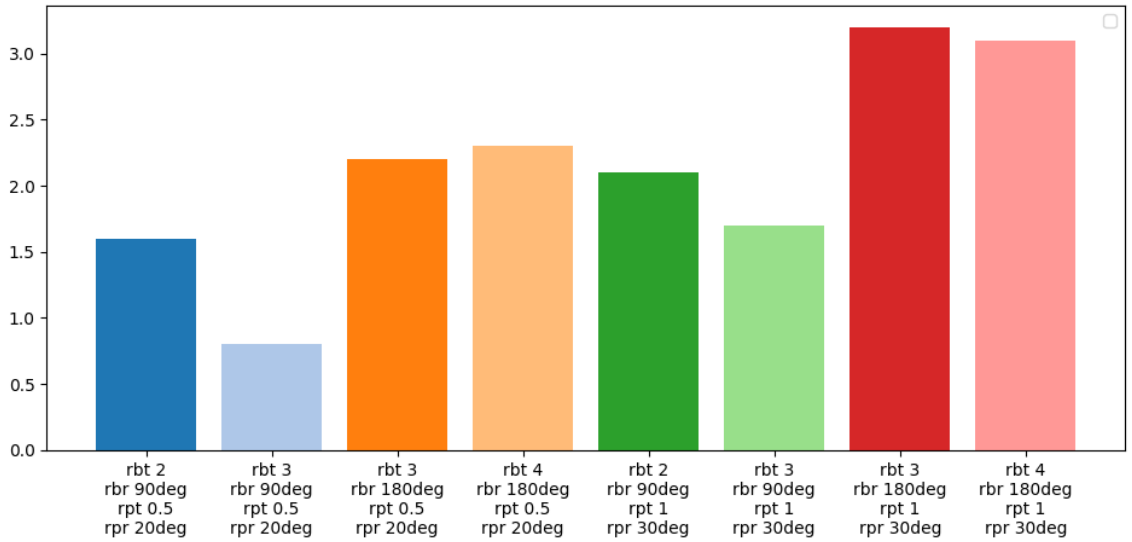


Figure 3.26: RRT*-GBO in the Maze scenario: average number of solutions found in a run.

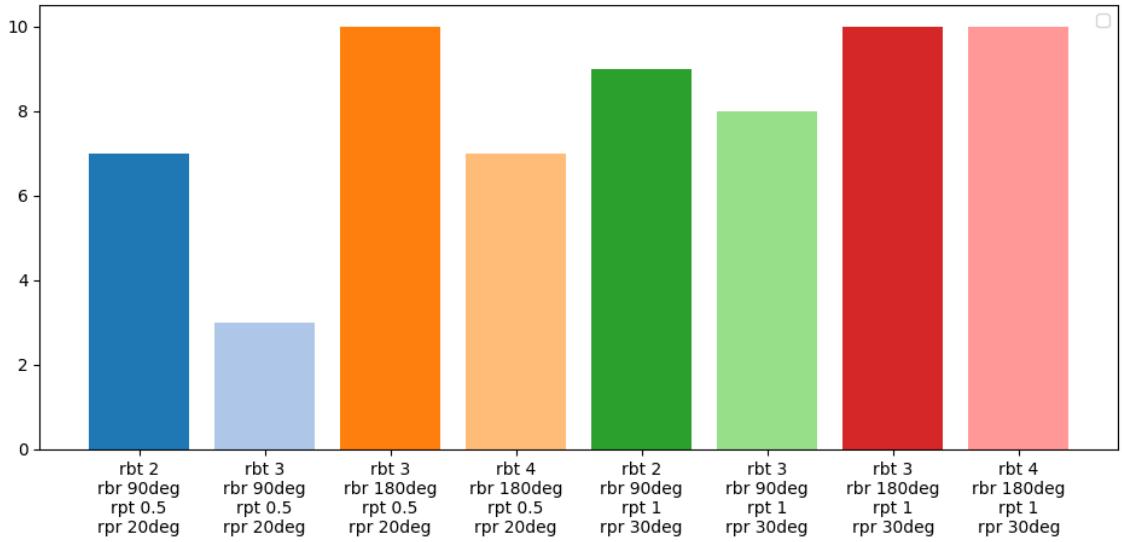
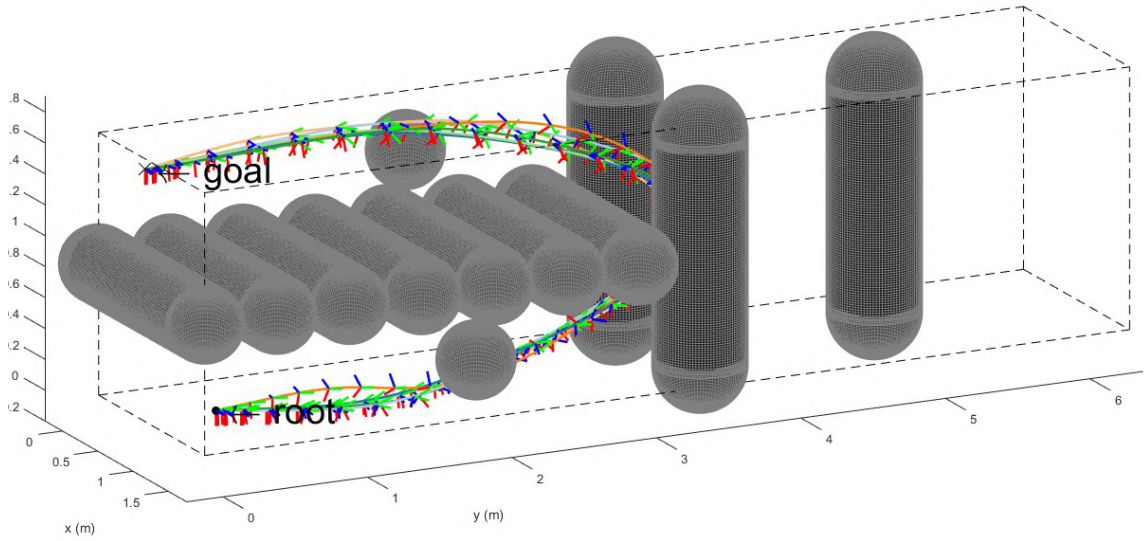
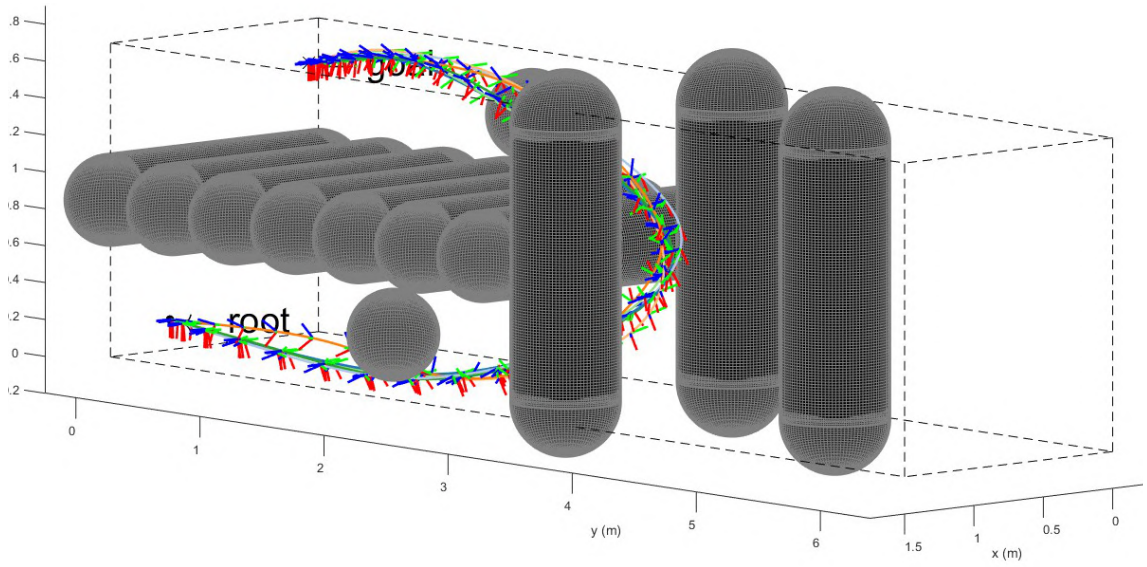


Figure 3.27: RRT*-GBO in the Maze scenario: number of runs that found at least one solution (out of 10).



(a) View 2



(b) View 2

Figure 3.28: RRT*-GBO best solutions in the Maze scenario.

3.2 Perception-awareness

This section evaluates the designed perception objective functions for perception-aware motion planning.
Figure 3.29 shows a model of the Astrobee (as an orange sphere) in a mock-up feature map (as magenta markers). The robot's body frame and camera frame are represented. The gray plane represents the camera plane, and the black dots are the projections of the visible features onto it. This visualization of the features projections shows exactly what the robot sees at a given pose.
Figure 3.30 shows a reduced map keeping only the features on the right wall ($x \approx 1.5$ [m]) and a straight point-to-point trajectory used as an initial guess for the perception-aware optimization performed in the following sections.

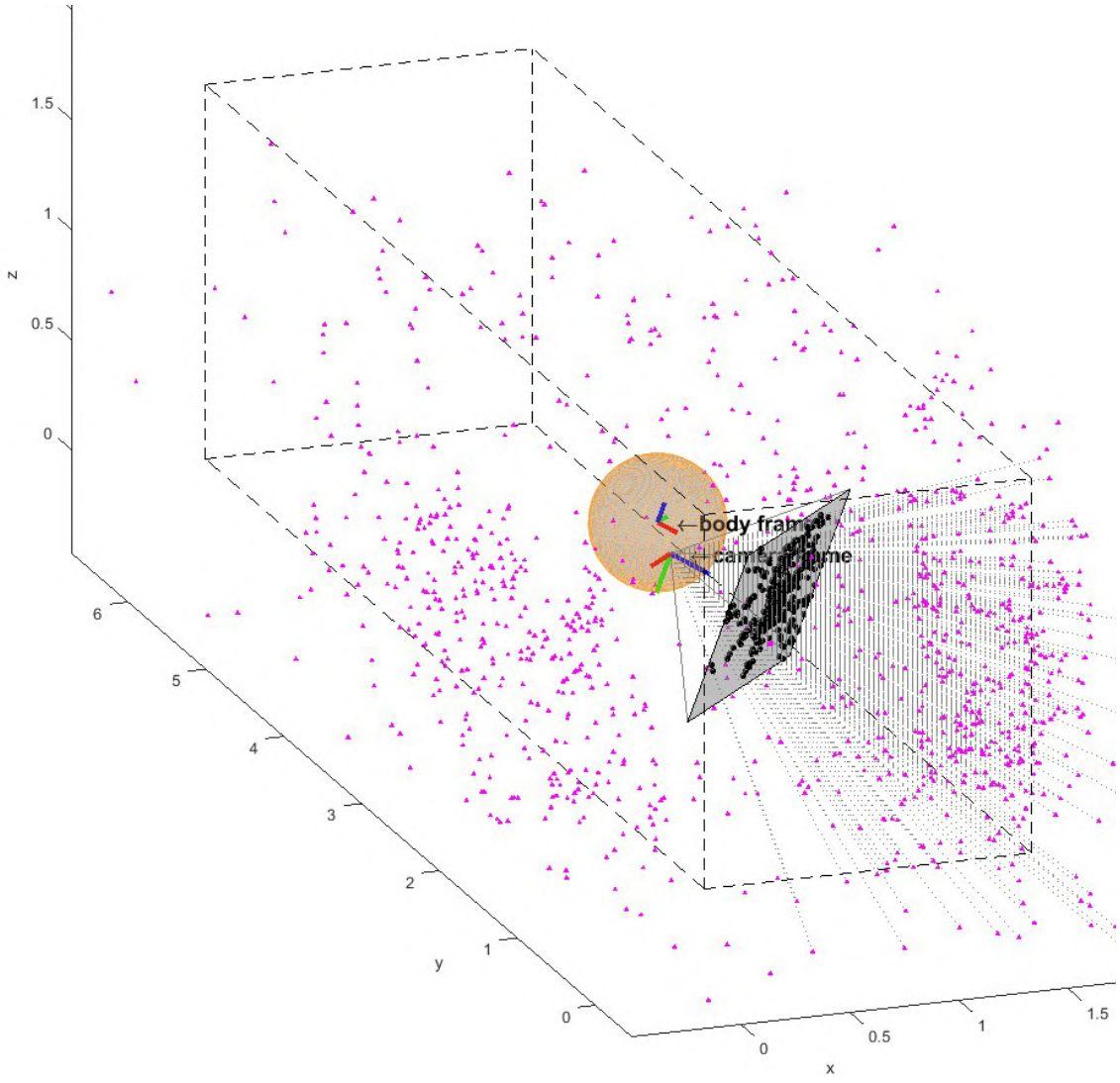


Figure 3.29: Feature projection in a mock-up feature map. Astrobee is represented as an orange sphere, features are represented as magenta markers, and the gray plane is the camera plane with feature projections as black dots.

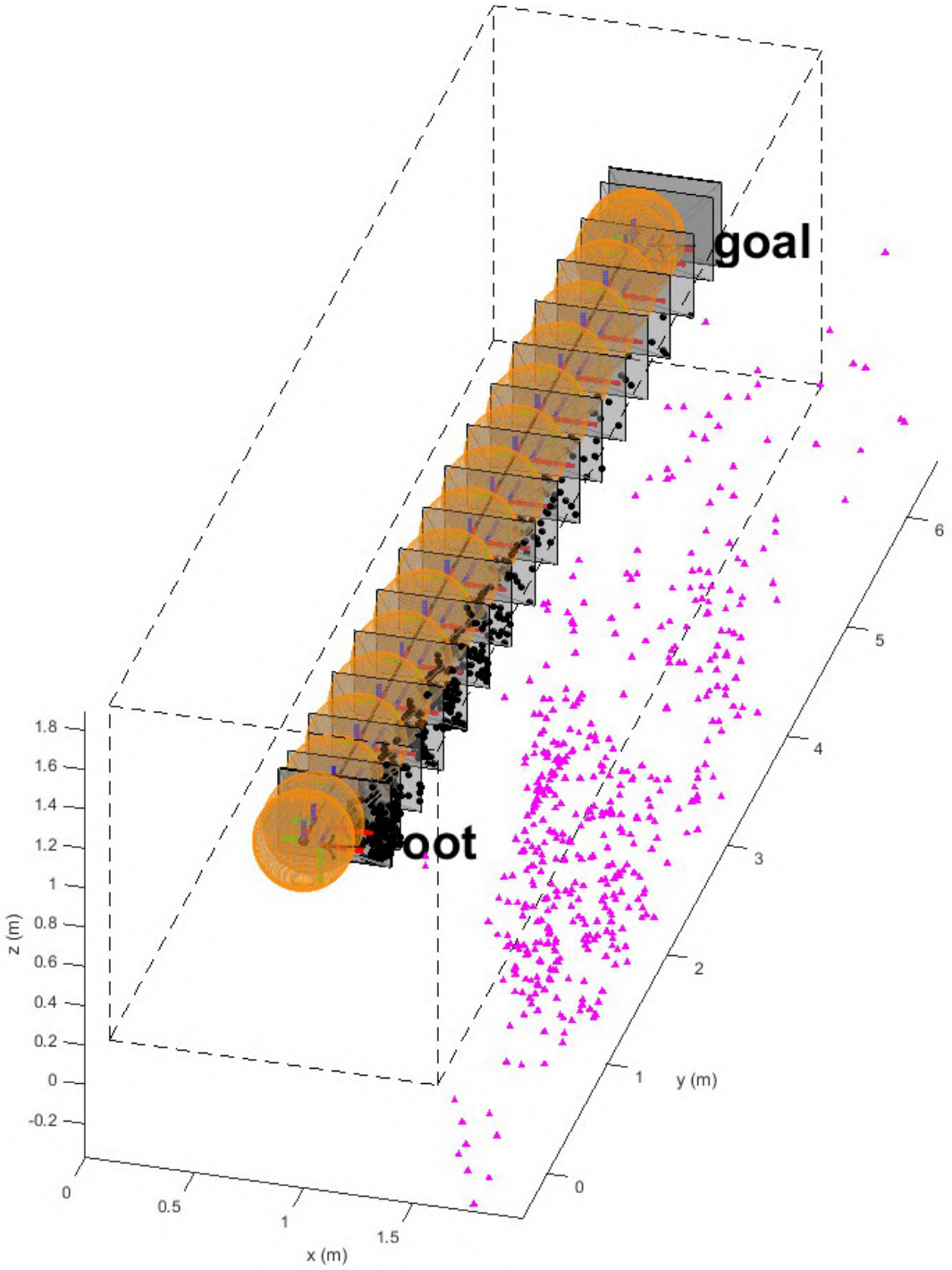


Figure 3.30: Reduced mock-up map and example trajectory.

3.2.1 Feature count vs. relaxed visibility

In this section, the trajectory from Fig. 3.30 is optimized to maximize $\Gamma_{\text{perception}}$ (without energy considerations) for $\Gamma_{\text{perception}} = \Gamma_{\text{FC}}$ and $\Gamma_{\text{perception}} = \Gamma_{\text{RV}}$.
As discussed in Sec.2.1.2, the feature count objective function Γ_{FC} (2.9) is discontinuous and thus unsuitable for the gradient computations (2.35). For instance, a small gradient step $\mathbf{\Delta s}$ applied to a robot's pose $\mathbf{s}_i$ may not cause any change in the count of features the robot sees, resulting in a gradient $\nabla\Gamma_{\text{FC}} = 0$ at every iteration. That is the case with the default gradient step $\mathbf{\Delta s} = \mathbf{1e^{-6}}$
The proposed solution is gradually increasing the gradient step $\mathbf{\Delta s}$ until gradients $\nabla\Gamma_{\text{FC}} \neq 0$ are obtained. A gradient step of $\mathbf{\Delta s} = \mathbf{1e^{-3}}$ achieves that.
Figure 3.31 shows the optimized trajectory for $\Gamma_{\text{perception}} = \Gamma_{\text{FC}}$ and $\mathbf{\Delta s} = \mathbf{1e^{-3}}$. The robot correctly starts off by taking distance from the right wall (to have more features in view) and focusing on the densest region of features at y $\approx$ 1 [m] but quickly turns to not informative regions.
In contrast, the relaxed visibility objective function Γ_{RV} (2.20) is continuous and suitable for gradient computation with the default gradient step $\mathbf{\Delta s} = \mathbf{1e^{-6}}$ (or any other gradient step).
Figure 3.32 shows the optimized trajectory for $\Gamma_{\text{perception}} = \Gamma_{\text{RV}}$. Throughout the entire trajectory, the robot stays far from the right wall (constrained by the upper left edge of the workspace box) and focuses on the densest region of features at y $\approx$ 1 [m].
Therefore, the relaxed visibility objective function shows superior results for the chosen scenario.

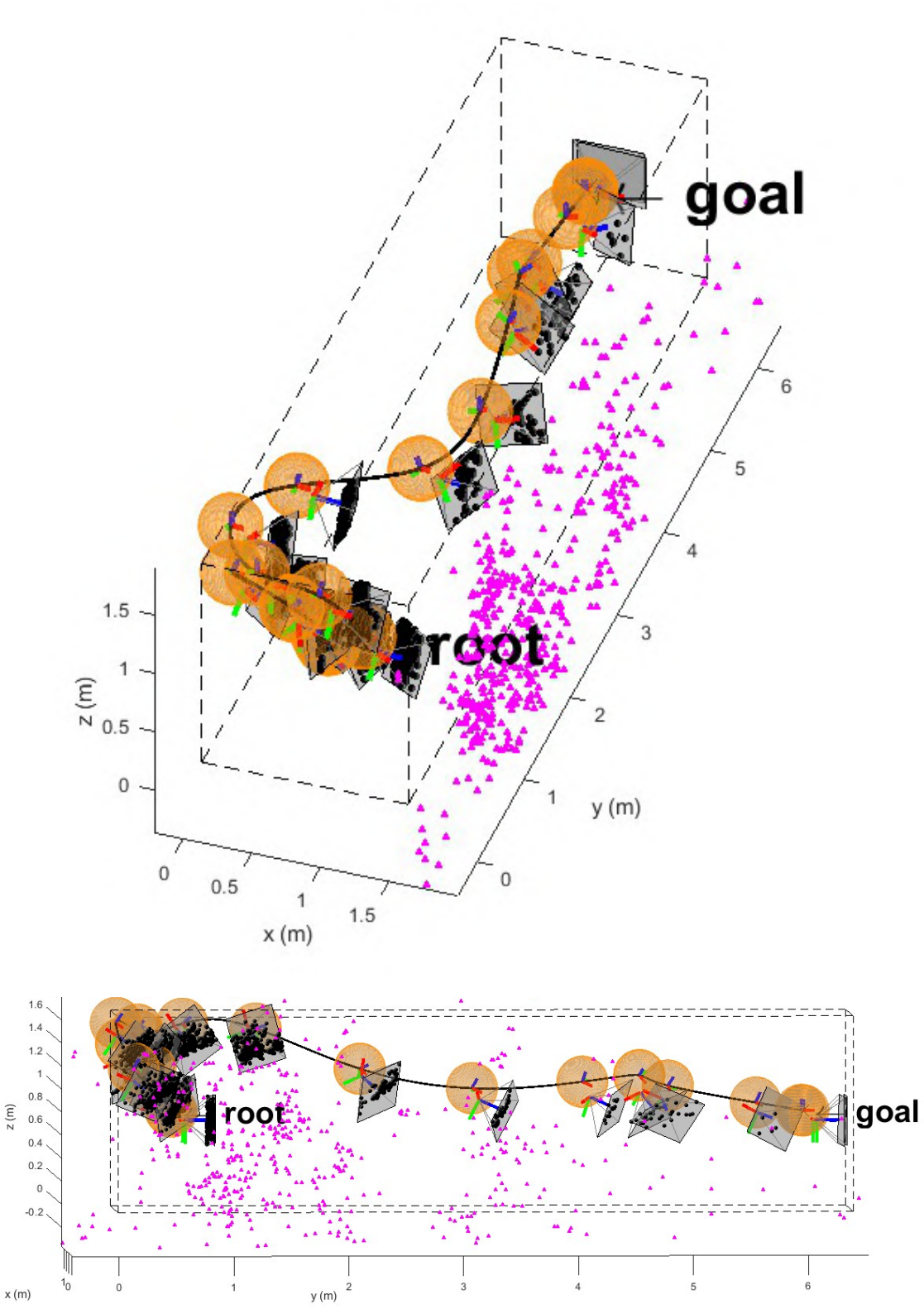


Figure 3.31: Optimized trajectory using the feature count objective function (from two viewing angles).

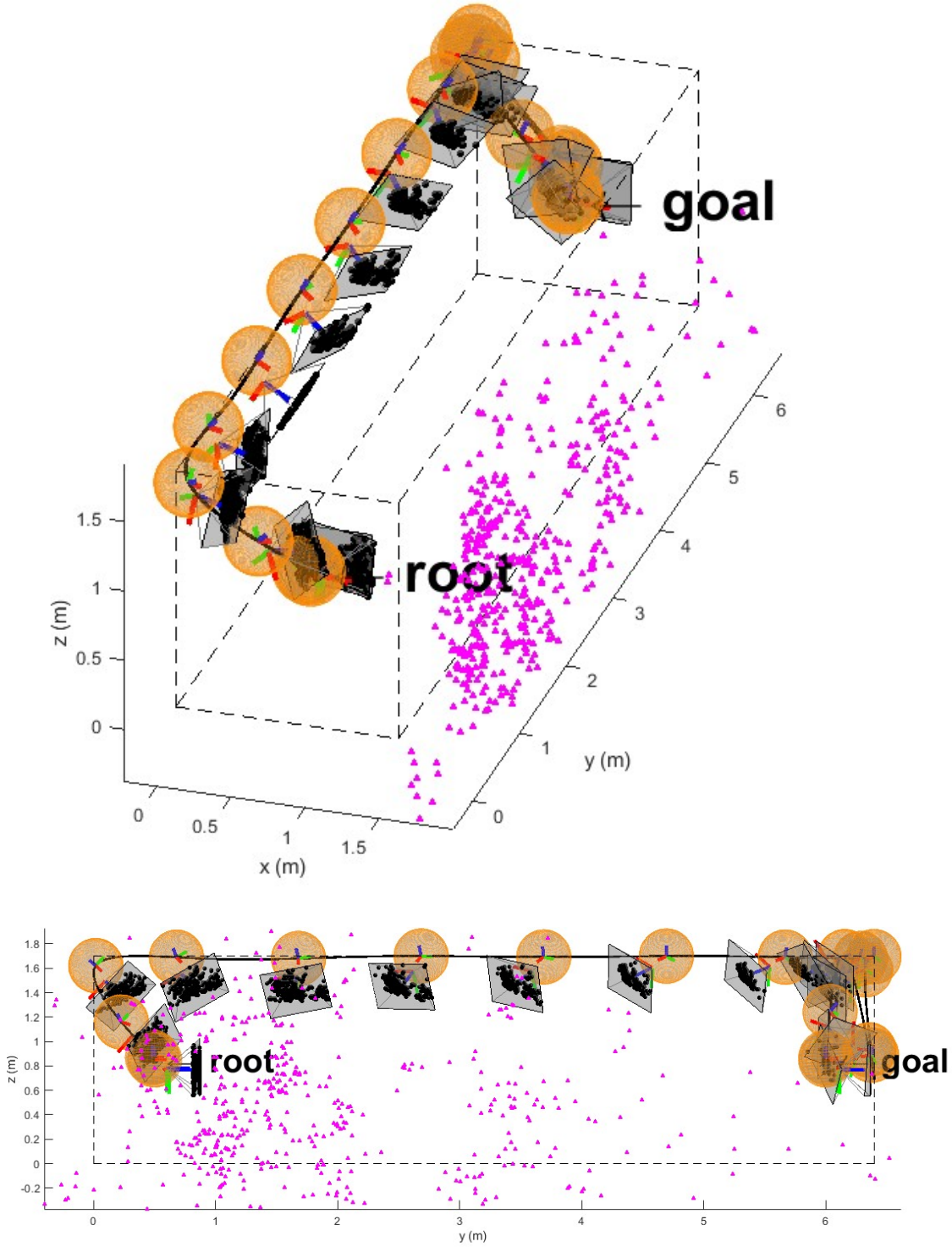


Figure 3.32: Optimized trajectory using the relaxed visibility objective function (from two viewing angles). It takes 1641 [s] to converge.

3.2.2 Interpolated perception objective function

The interpolated perception function presented in Sec. 2.1.2 is now evaluated. It can be equally applied to the feature count method to obtain $\hat{\Gamma}_{\text{FC}}$ or to the relaxed visibility objective method to obtain $\hat{\Gamma}_{\text{RV}}$ (2.20). Besides reducing computation times (as explained in Sec. 2.1.2), the interpolated feature count objective function $\hat{\Gamma}_{\text{FC}}$ also has the advantage of being continuous, and thus, suitable for gradient computation, in contrast to the original function Γ_{FC}.
The first step is to define a grid of robot poses to be interpolated. For the scenario in Fig. 3.30, the following grid is chosen:

- x: 5 linearly space points in the range $[0; 1.5]$.
- y: 10 linearly space points in the range $[0; 6.4]$.
- z: 5 linearly space points in the range $[0; 1.7]$.
- ξ_x: 19 linearly space points in the range $[0; \pi]$.
- ξ_y: 19 linearly space points in the range $[0; \pi]$.
- ξ_z: 10 linearly space points in the range $[0; \pi]$.

The summed feature visibility function γ (γ_{FC} or γ_{RV}) is evaluated at each of the $5 \times 10 \times 5 \times 19 \times 19 \times 19 = 1714750$ grid points. The function's values are interpolated across the grid points using cubic splines (Catmull-Rom) from Btwxt [Sof24] to obtain the corresponding interpolated perception objective function $\hat{\Gamma}_{\text{perception}}$ ($\hat{\Gamma}_{\text{FC}}$ or $\hat{\Gamma}_{\text{RV}}$).
Figure 3.33 shows the optimized trajectory using the interpolated feature count objective function $\hat{\Gamma}_{\text{FC}}$. Compared to the trajectory using the original feature count function in Fig. 3.31, the robot now has a good view of the features during the entire trajectory.
Figure 3.34 shows the optimized trajectory using the interpolated relaxed visibility objective function $\hat{\Gamma}_{\text{RV}}$. It is practically identical to the trajectory using the originals relaxed visibility function in Fig. 3.32. The difference is that the trajectory optimization using the interpolated function takes 169 [s] to converge to the solution, while the not interpolated function takes 1641 [s]. That consists of approximately a 10-times reduction in the optimization time.
At the grid points $\mathbf{x}_i$, the interpolated summed visibility function $\hat{\gamma}$ equals the original perception function γ. However, errors occur between the grid points. Figure 3.35 assesses the magnitude of the errors. For both the feature count and the relaxed visibility methods, it shows the average relative errors (in percent) for points in the middle of two gridpoints ($\mathbf{x}_i + 0.5\Delta\mathbf{x}$), points at 1/4 of the distance between two consecutive gridpoints ($\mathbf{x}_i + 0.25\Delta\mathbf{x}$), and points at 3/4 of the distance between two consecutive gridpoints ($\mathbf{x}_i + 0.75\Delta\mathbf{x}$). As expected, the largest errors occur in

the middle of two grid points ($\mathbf{x}_i + 0.5\Delta\mathbf{x}$) since that is the most distant position from any grid point. Furthermore, the interpolated relaxed visibility function shows higher fidelity to the original function, with errors 4 times smaller than the interpolated feature count function. That may be explained by the discontinuous aspect of the feature count function, which is hard to fit in the interpolator's continuous, smooth cubic spline function.

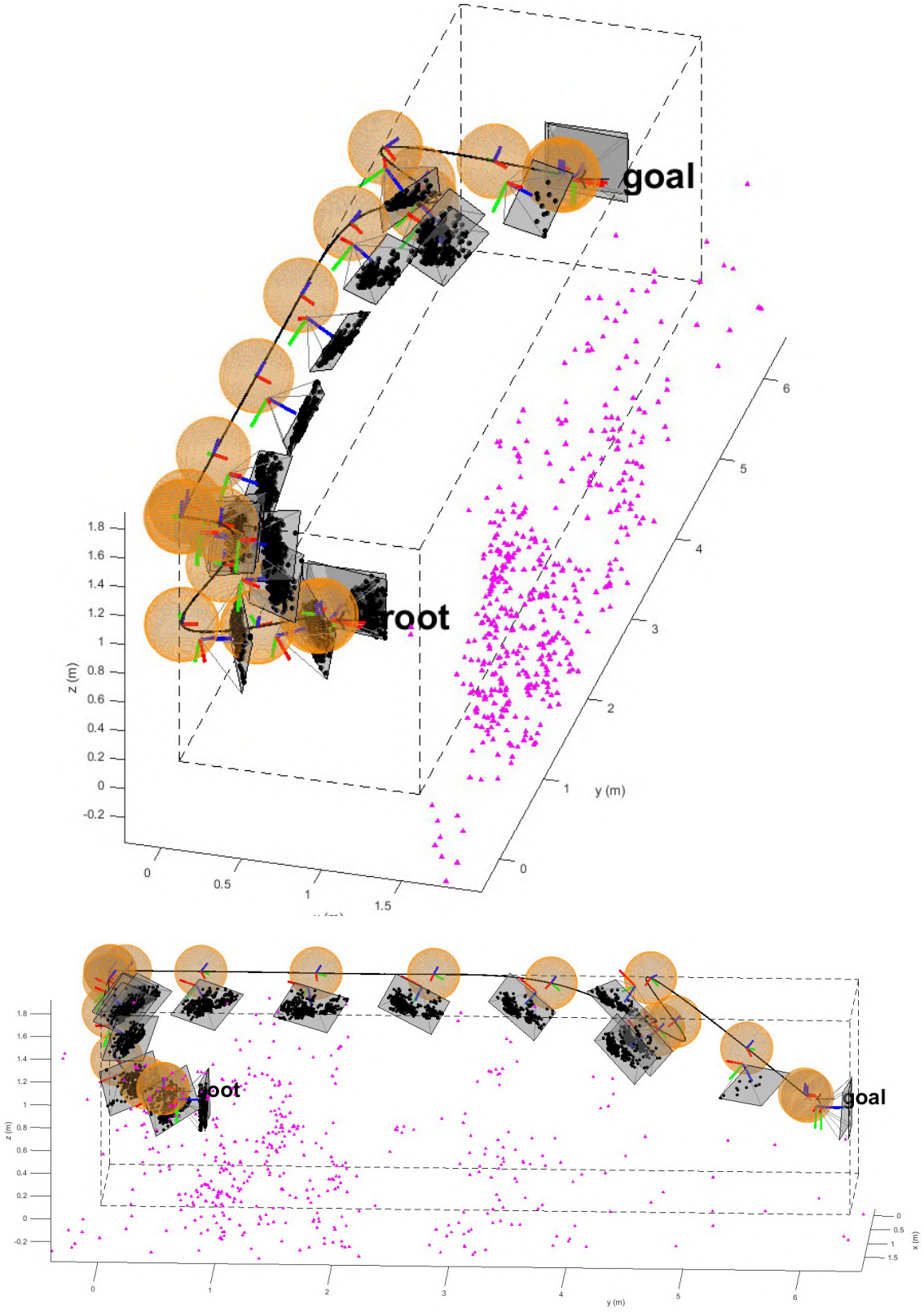


Figure 3.33: Optimized trajectory using the interpolated feature count objective function (from two viewing angles).

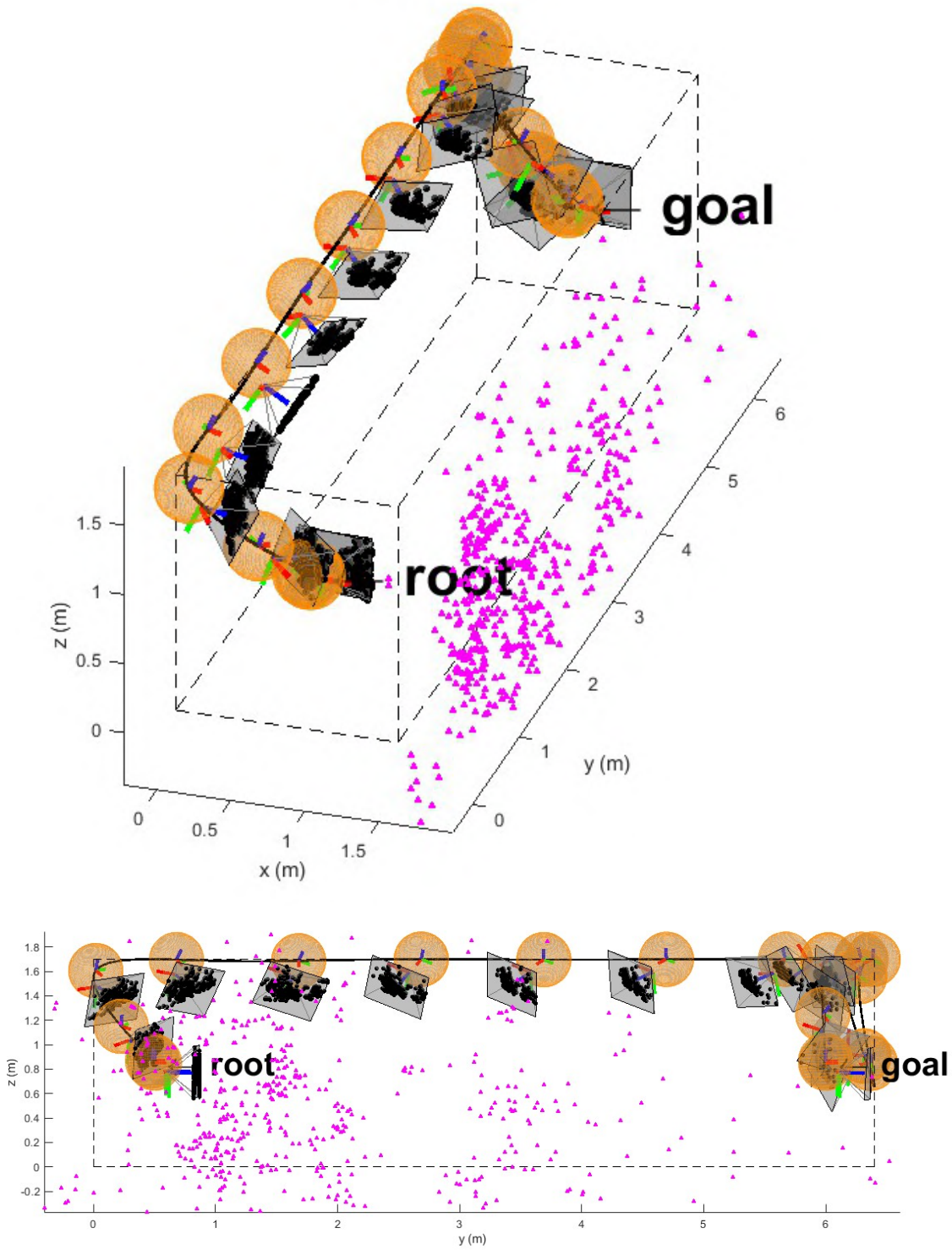
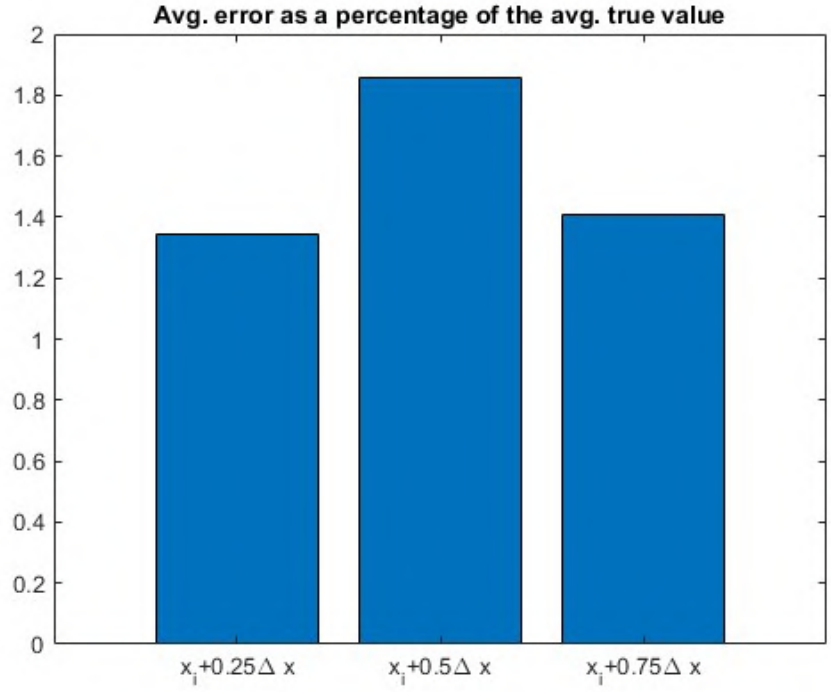
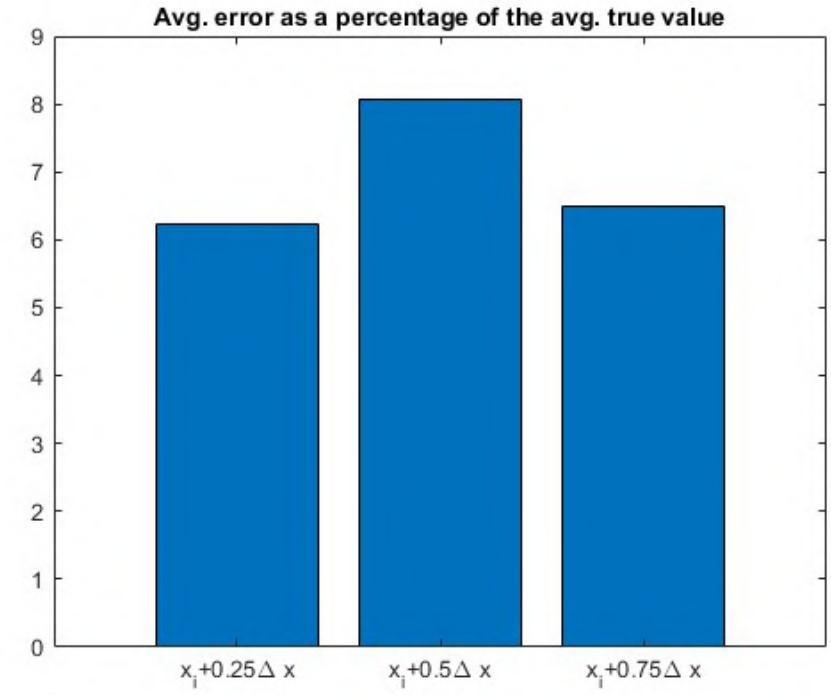


Figure 3.34: Optimized trajectory using the interpolated relaxed visibility objective function (from two viewing angles). It takes 169 [s] to converge.



(a) Relaxed visibility



(b) Feature count

Figure 3.35: Interpolation errors comparison.

3.2.3 Weighted metric analysis

Figures 3.36 to 3.39 show how the weighting term w_{energy} from the weighted cost function (2.27) affects the optimized trajectories. A mock-up feature map of the satellite model in DLR's OOS-SIM facility [ADSR$^+$15] (in magenta) is used to optimize the trajectory of the servicer robot end-effector. The LWR robot is represented in orange. Again, features are projected as black dots in the camera's image plane, represented by the gray plane. When $w_{\text{energy}} = 0$, energy consumption is disregarded, and the robot goes as far as possible from the features (constrained by the upper anterior edge of the workspace box) to ensure all of them are seen. As w_{energy} gradually increases, the motions become less aggressive. The straight-line, energy-optimal solution is found when $w_{\text{energy}} = 1$.

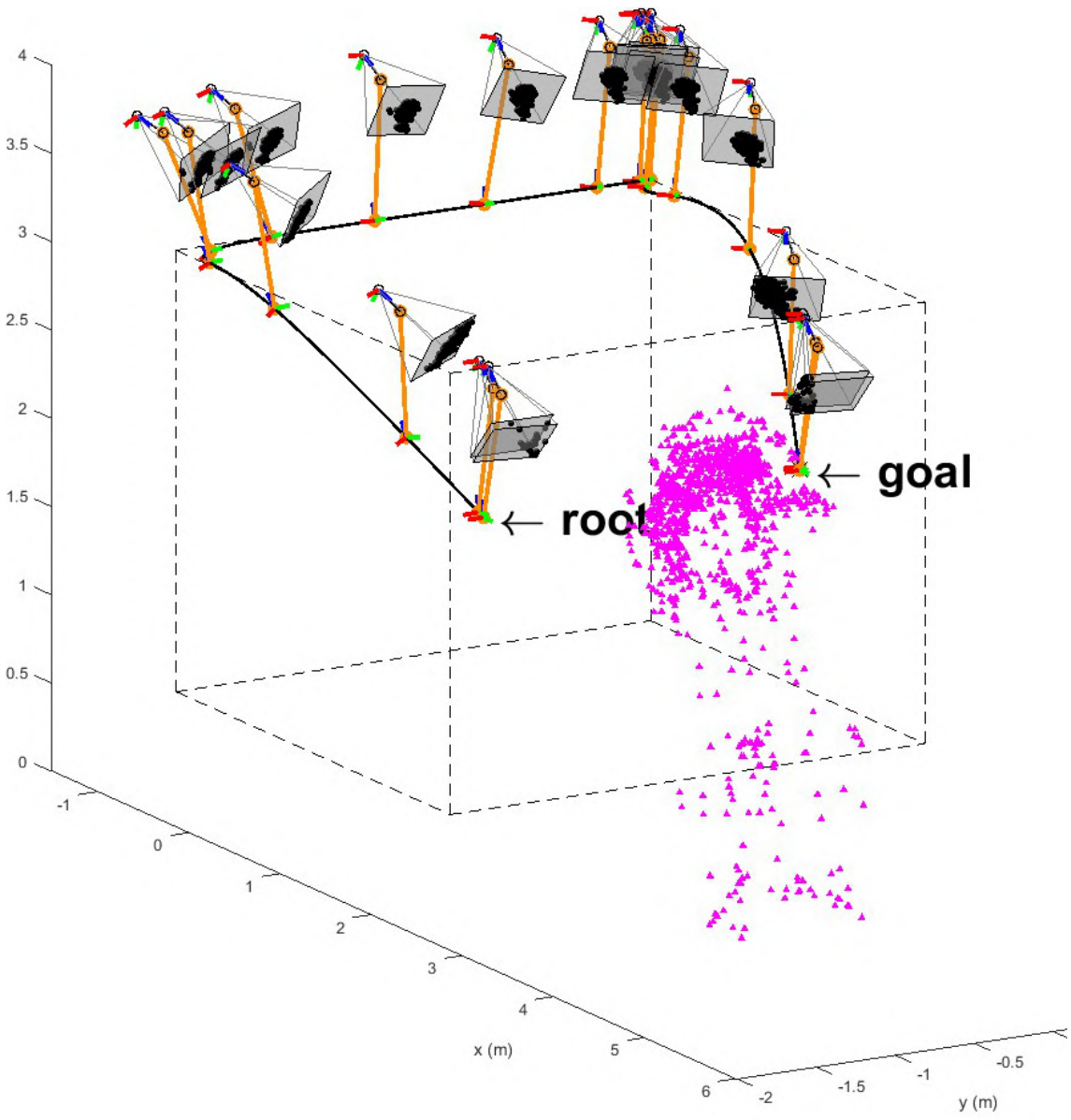


Figure 3.36: Optimized trajectory using the weighted cost function with $w_{\text{energy}} = 0$ for a mock-up OOS-SIM scenario.

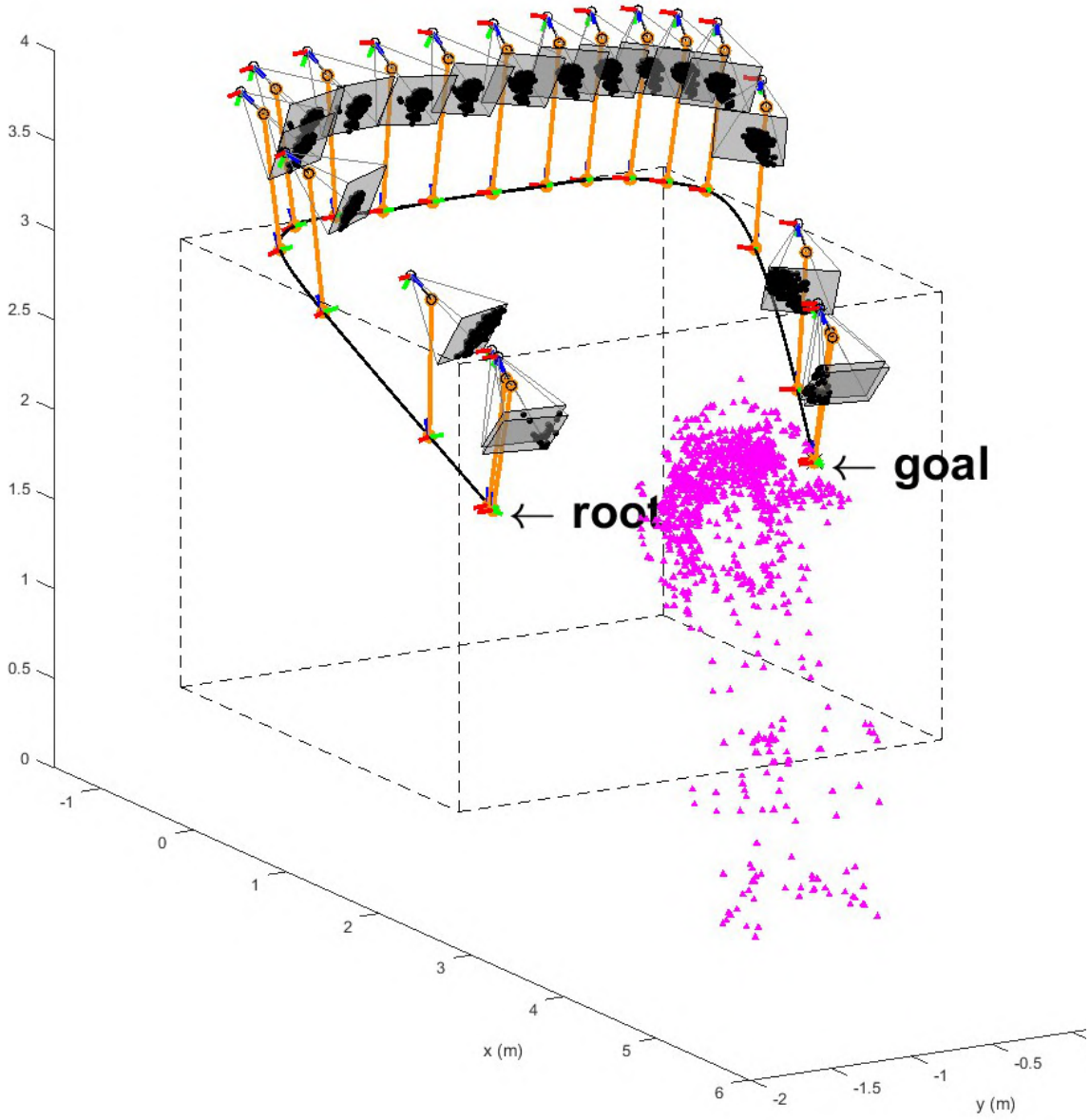


Figure 3.37: Optimized trajectory using the weighted cost function with $w_{\text{energy}} = 0.8$ for a mock-up OOS-SIM scenario.

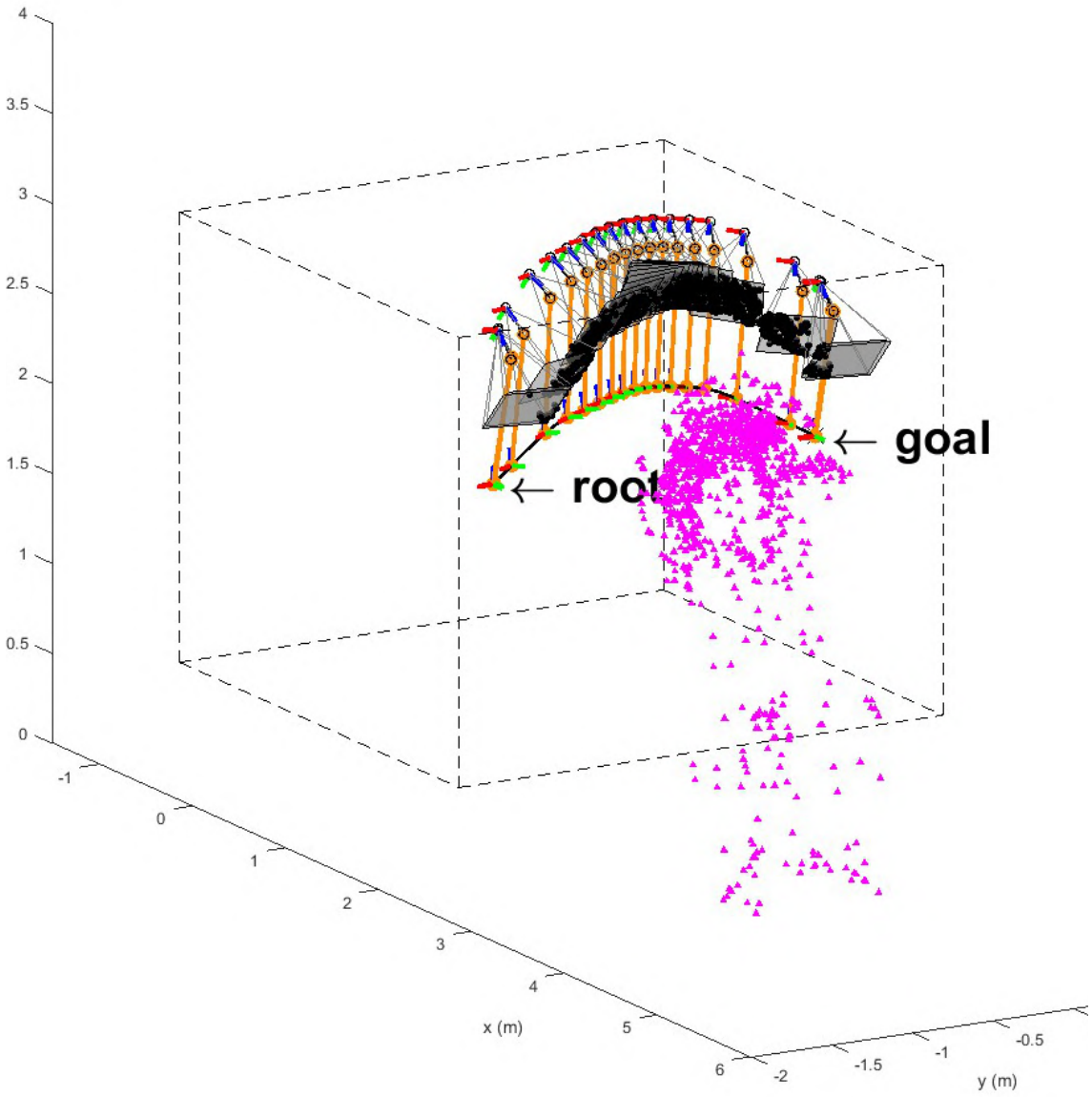


Figure 3.38: Optimized trajectory using the weighted cost function with $w_{\text{energy}} = 0.85$ for a mock-up OOS-SIM scenario.

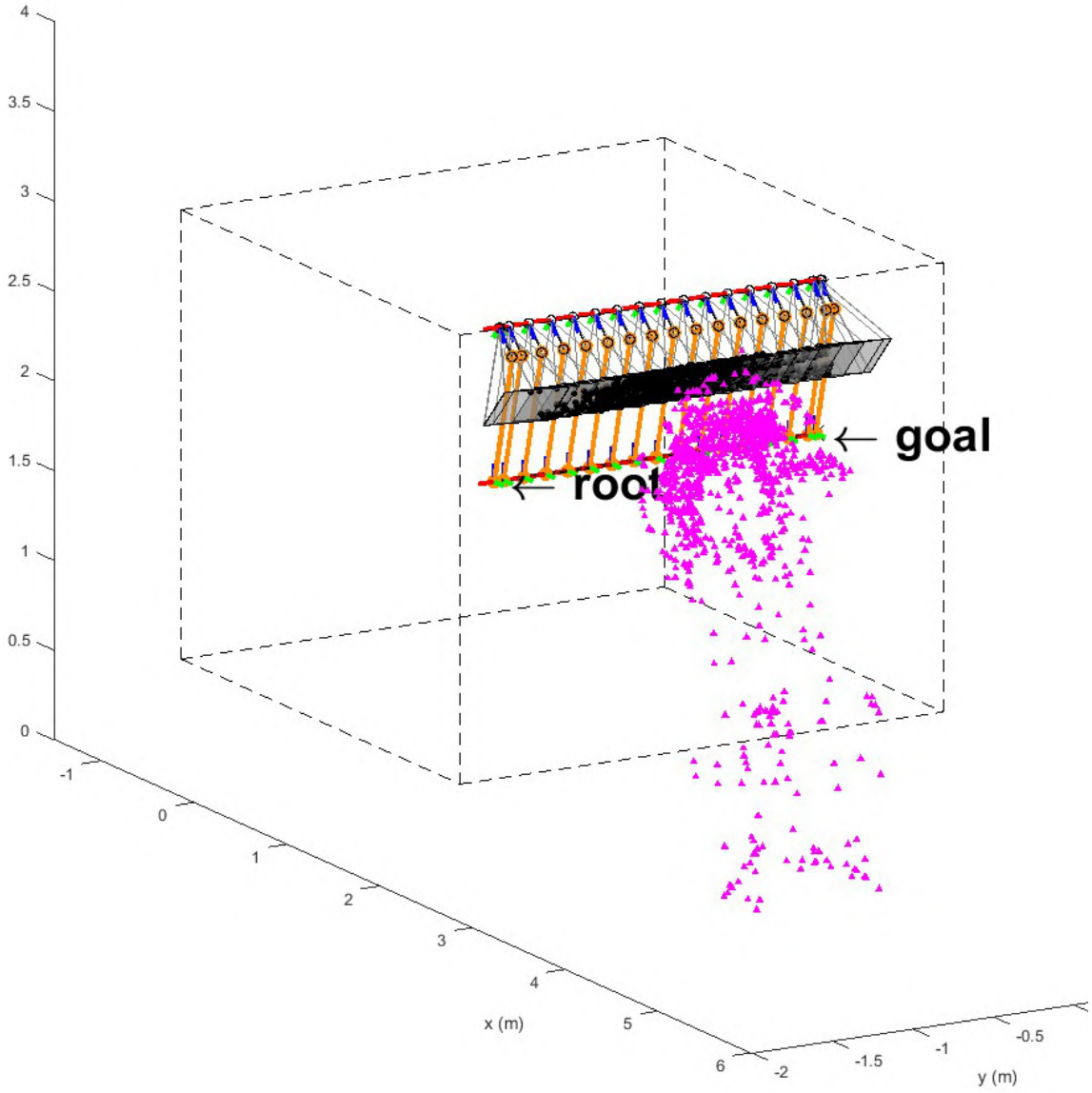


Figure 3.39: Optimized trajectory using the weighted cost function with $w_{\text{energy}} = 1$ for a mock-up OOS-SIM scenario.

3.3 Astrobees rendezvous experiment

The complete method is now evaluated for the rendezvous between two Astrobee robots in the ISS (in simulation). The experiment takes inspiration from MIT's video [Aer], which is part of the experiments from [ASM+22].
A simulation environment in Gazebo is designed for this thesis. 3D models provided by NASA are used, namely a model of the US module of the ISS and a model of the Astrobee. A trajectory commander is designed using ROS to allow the Astrobee to follow any SE(3) trajectories. The simulated Astrobee camera streams images to a ROS topic, which can be visualized/processed in real-time. Fig. 3.40 shows the simulation environment.
The simulated camera mimics the real Astrobee camera. It employs the pinhole model from Sec. 1.4.6 with the following parameters: focal length $f = 607$, principal point offset $p_x = 625$, $p_y = 515$, image width $W = 1250$, image height $H = 1030$. Therefore, the intrinsics matrix is

$$\mathbf{K} = \begin{bmatrix} 607 & 0 & 625 \\ 0 & 607 & 515 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.21}$$

The transformation matrix from Astrobee's center of mass to its camera is

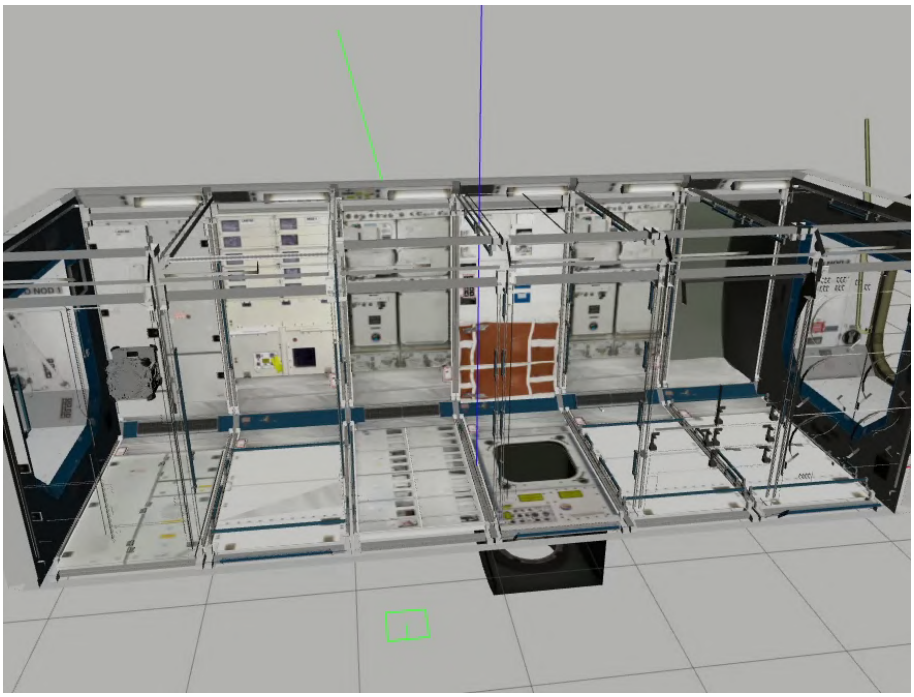
$$\mathbf{T}_{CoM}^{\text{cam}} = \begin{bmatrix} 0 & 0 & 1 & 0.1177 \\ -1 & 0 & 0 & -0.0422 \\ 0 & -1 & 0 & -0.08260 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.22}$$

and, therefore, the extrinsics matrix is

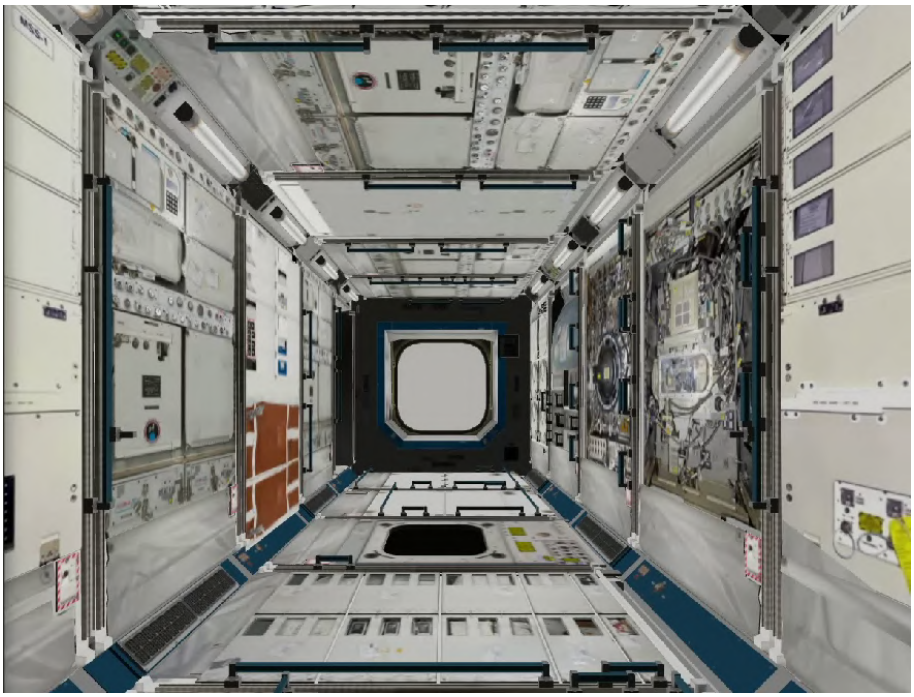
$$\begin{bmatrix} & \mathbf{H}_{\text{cam,c}}^{W} & & \\ 0 & 0 & 0 & 1 \end{bmatrix} = \left(\mathbf{T}_{CoM}^{\text{cam}}\right)^{-1} \left(\mathbf{T}_{W}^{CoM}\right)^{-1}, \tag{3.23}$$

for a commanded pose of the robot's center of mass $\mathbf{T}_{W}^{\text{CoM}}$.
Astrobee's dynamic properties from (3.1) and (3.2) are used. The robot's geometry is modeled as a sphere of radius 0.225 [m] to cover the corners of the robot's real cubic geometry of size $0.32 \times 0.32 \times 0.32$ [m]. The motion constraints from (3.3) and (3.7) are used, except from the position boundaries

$$\begin{bmatrix} -2.7 \\ -0.75 \\ 0.6 \end{bmatrix} \leq \mathbf{t} \leq \begin{bmatrix} 1.6 \\ 0.75 \\ 2.1 \end{bmatrix} \tag{3.24}$$



(a) Gazebo simulator.



(b) Camera feed from Gazebo. Transmitted via ROS, visualized in RViz.

Figure 3.40: ISS/Astrobees simulation environment.

3.3.1 Mapping and perception metric interpolation

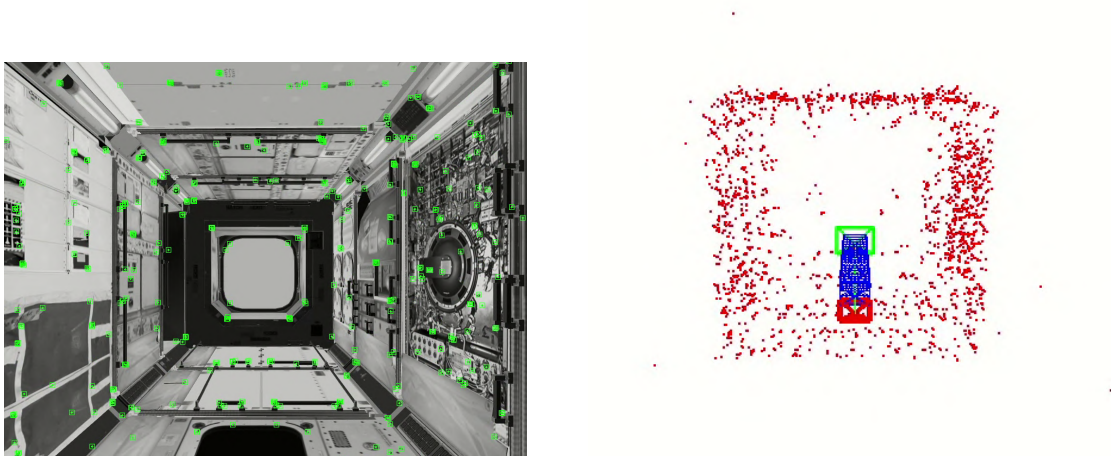


Figure 3.41: Simulated ISS's US module mapping. On the left: camera view with green markers for features. On the right: SLAM map features (red dots) and camera localization (crossed rectangles.)

A map of the simulated environment is obtained by commanding the Astrobee to move forward slowly inside the US module. The process is illustrated in Fig. 3.41. The map is downsampled by a factor of 5, resulting in 923 features.
The perception metric is interpolated for this feature map as in (2.26). The following grid is used:

- x: 20 linearly space points in the range $[-3; 3]$.
- y: 5 linearly space points in the range $[-0.75; 0.75]$.
- z: 5 linearly space points in the range $[0.6; 2.1]$.
- ξ_x: 19 linearly space points in the range $[0; \pi]$.
- ξ_y: 19 linearly space points in the range $[0; \pi]$.
- ξ_z: 10 linearly space points in the range $[0; \pi]$.

3.3.2 Trajectory planning and execution

A second Astrobee was added to the simulation. The task is to bring the first Astrobee (chaser) to the back of the second Astrobee (target) with

$$\mathbf{s}_{\text{root}} = \begin{bmatrix} -2.5 & 0 & 1.4 & 0 & 0 & 0 \end{bmatrix}^T, \tag{3.25}$$

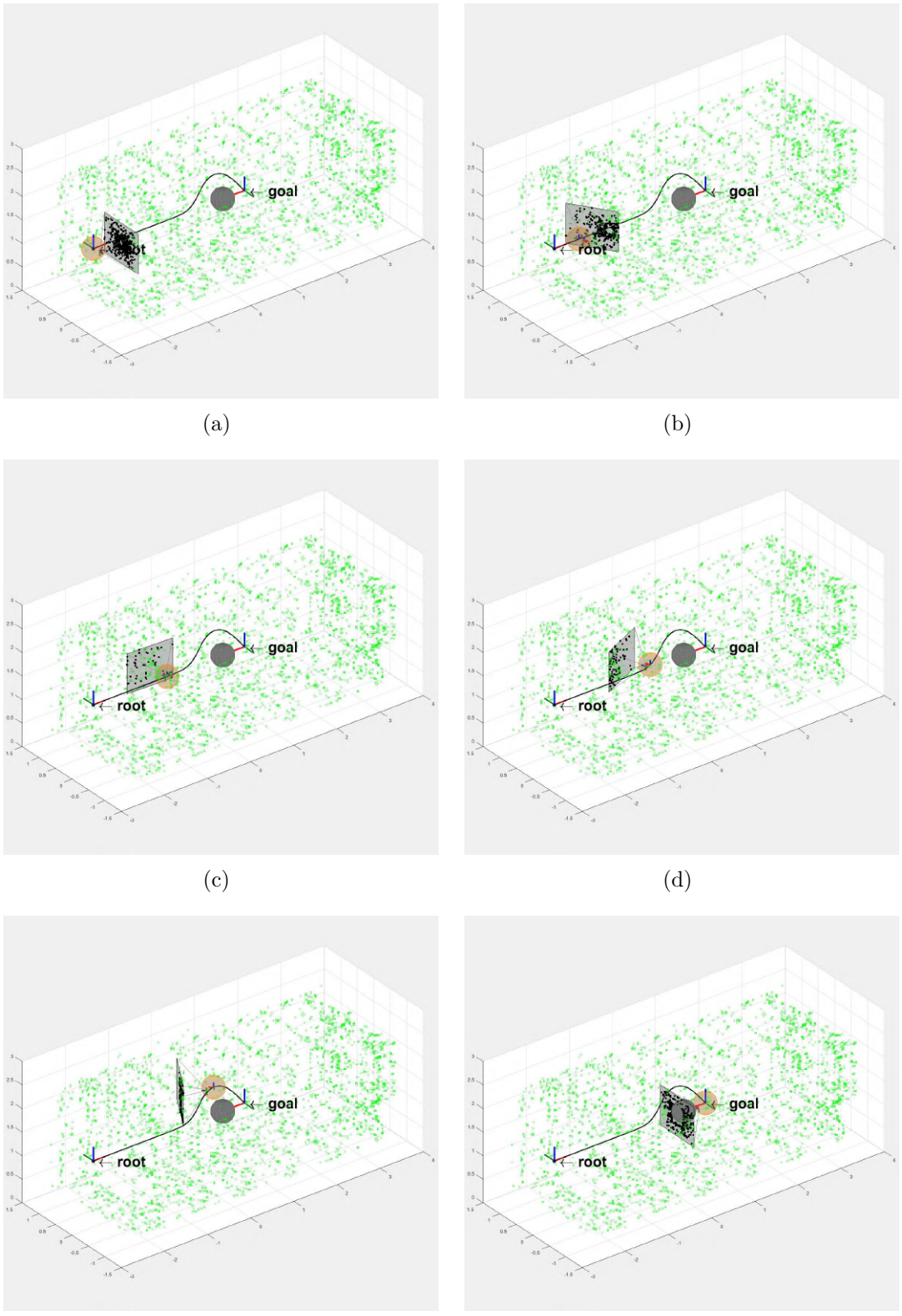
$$\mathbf{s}_{\text{goal}} = \begin{bmatrix} 1 & 0 & 1.4 & 0 & 0 & \pi \end{bmatrix}^T, \tag{3.26}$$

$$t_f = 60 \quad [\text{s}]. \tag{3.27}$$

An energy-optimal and a perception-aware trajectory are compared to evaluate the localization accuracy. In both cases, a straight-line initial guess is fed to the optimizer, skipping the RRT*-GBO edges construction part.
The optimizer's stopping criteria are set to $\Delta\Gamma_{\text{rel}} = 1e^{-6}$ and $\Delta\mathbf{p}_{\text{free,rel}} = 1e^{-6}$.

Energy-optimal

The trajectory in Fig. 3.42 is planned by optimizing only the energy spent by the robot (i.e., $w_{\text{energy}} = 1$, $w_{\text{perception}} = 0$). As the robot executes the planned trajectory, the camera view with detected features and the SLAM map and camera localization are shown in Fig. 3.43 and Fig. 3.44.
The chaser Astrobee deviates minimally from the straight-line optimal trajectory only to avoid colliding with the target Astrobee. Consequently, it has a close view of the left wall.

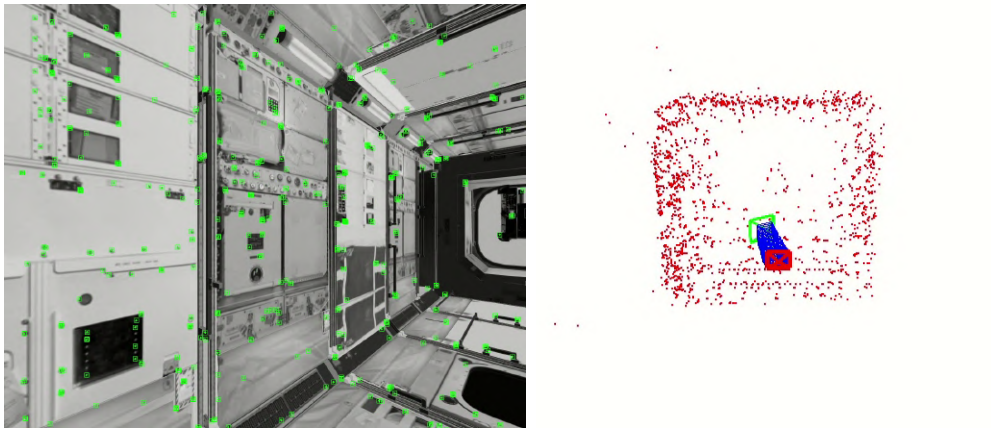


(a) (b)

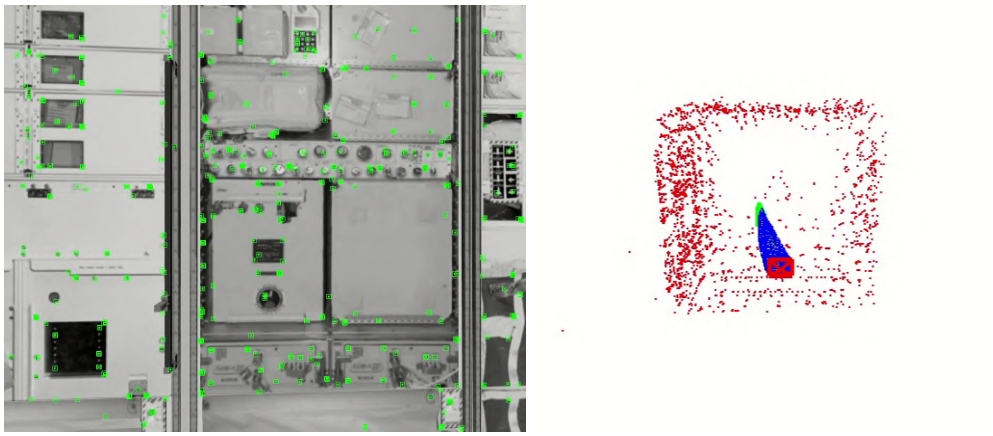
(c) (d)

(e) (f)

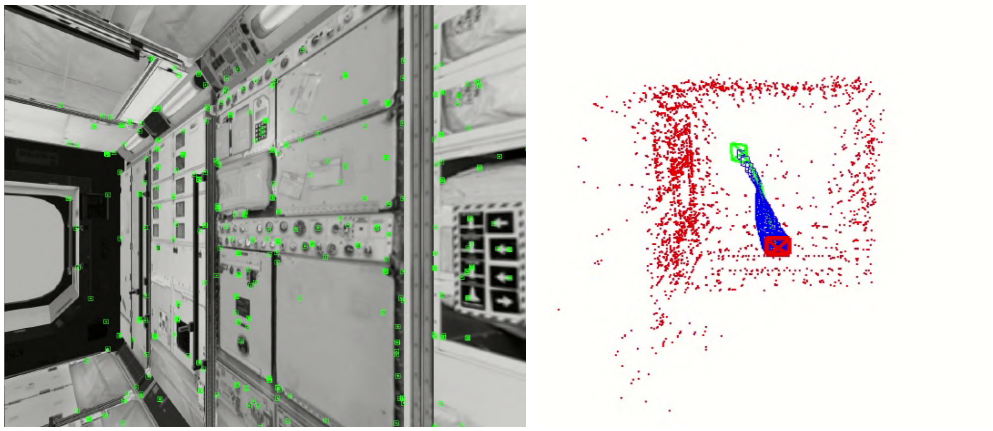
Figure 3.42: Planned energy-optimal Astrobee trajectory. The consecutive robot poses over time are shown.



(a)

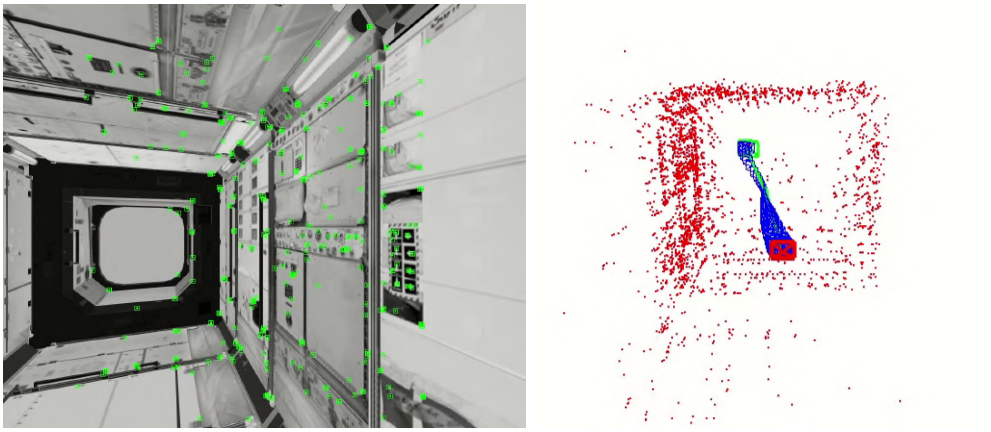


(b)

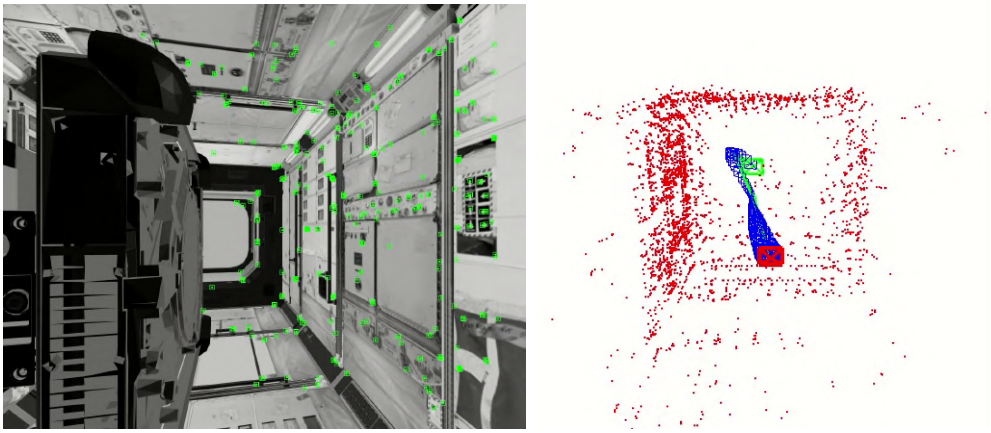


(c)

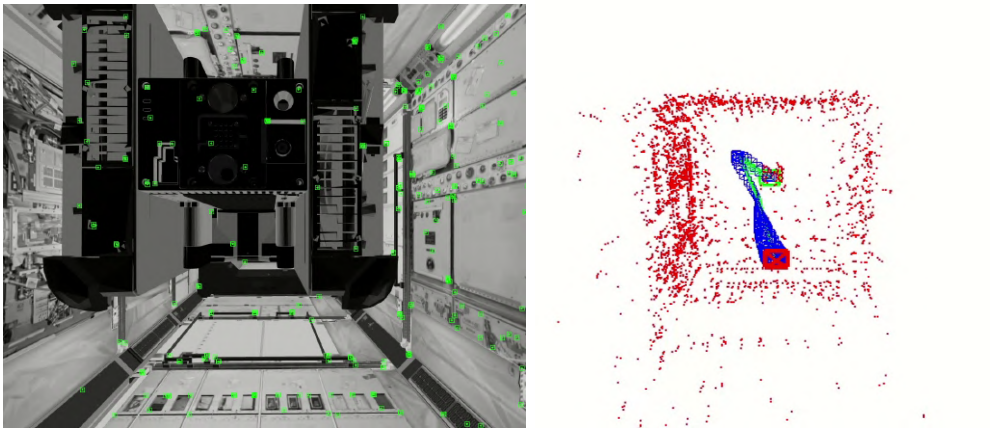
Figure 3.43: SLAM on the first half of the energy-optimal Astrobee trajectory.



(a)



(b)



(c)

Figure 3.44: SLAM on the second half of the energy-optimal Astrobee trajectory.

Perception-aware

The trajectory in Fig. 3.45 is planned using a weighted perception-aware cost metric ($w_{\text{energy}} = 0.9$, $w_{\text{perception}} = 0.1$). As the robot executes the planned trajectory, the camera view with detected features and the SLAM map and camera localization are shown in Fig. 3.46 and Fig. 3.47.
The chaser Astrobee makes a large arch away from the left wall to observe a large number of features during its trajectory.

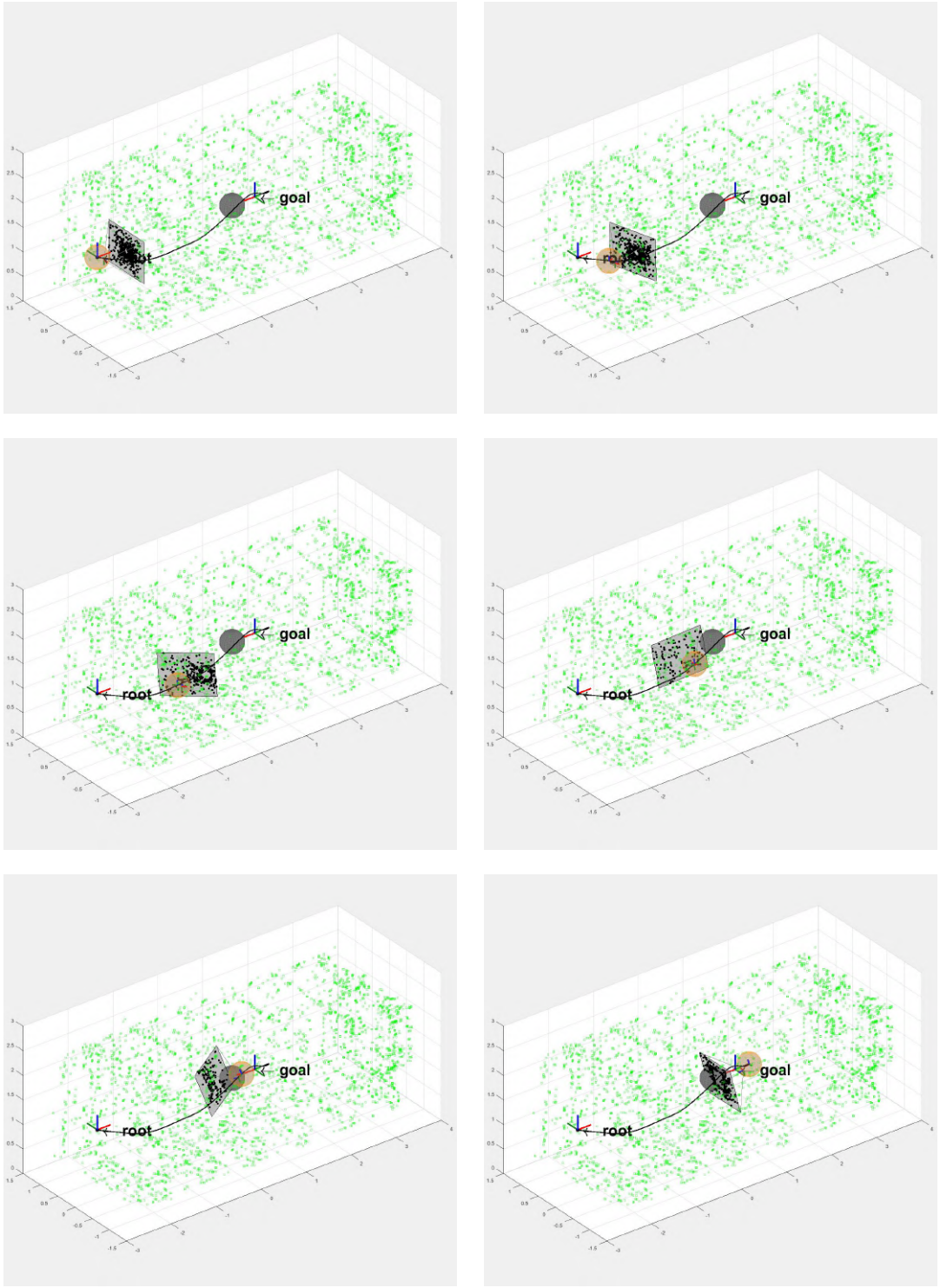
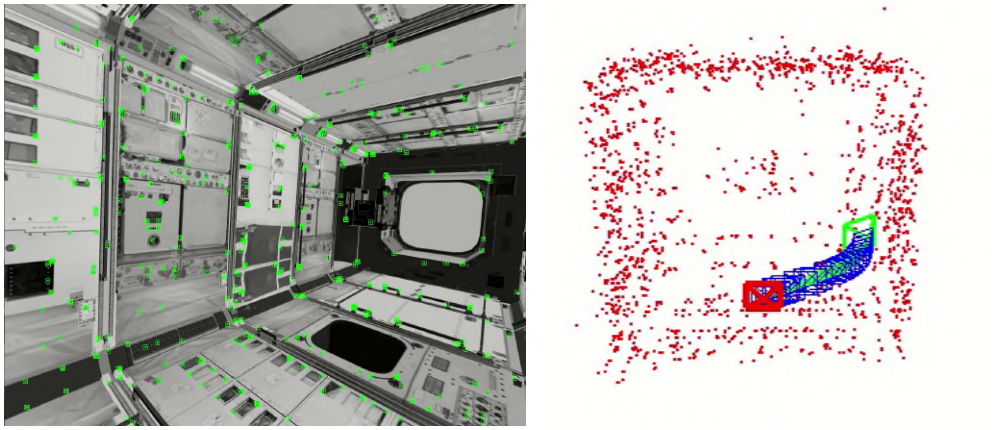
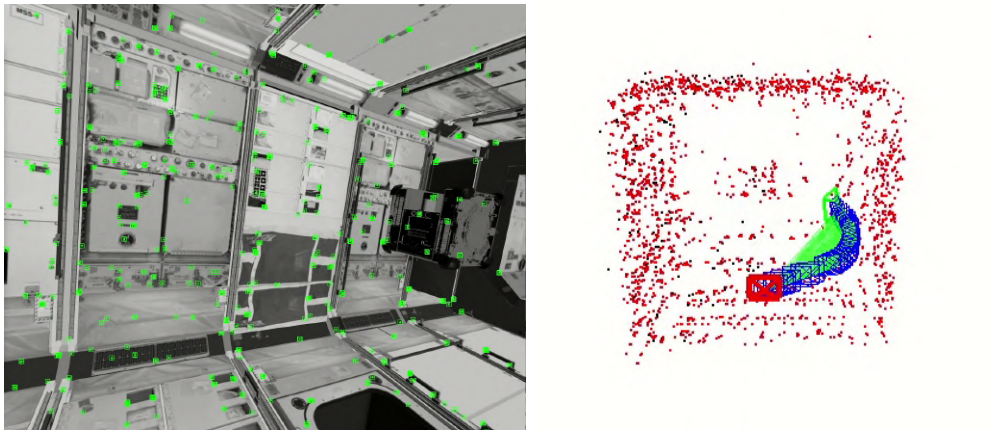


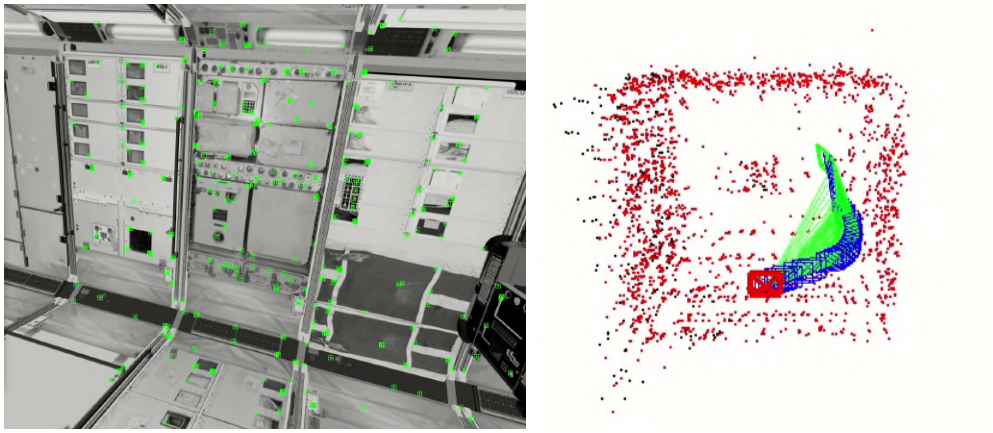
Figure 3.45: Planned perception-aware Astrobee trajectory. The consecutive robot poses over time are shown.



(a)

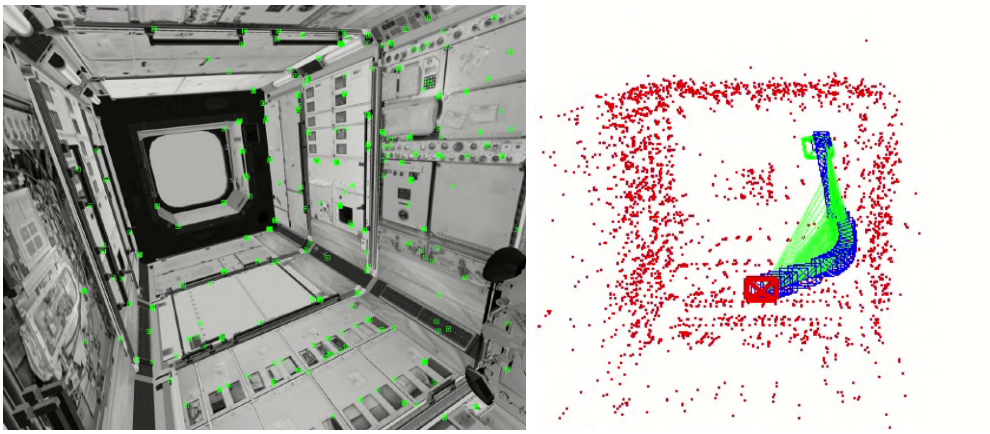


(b)



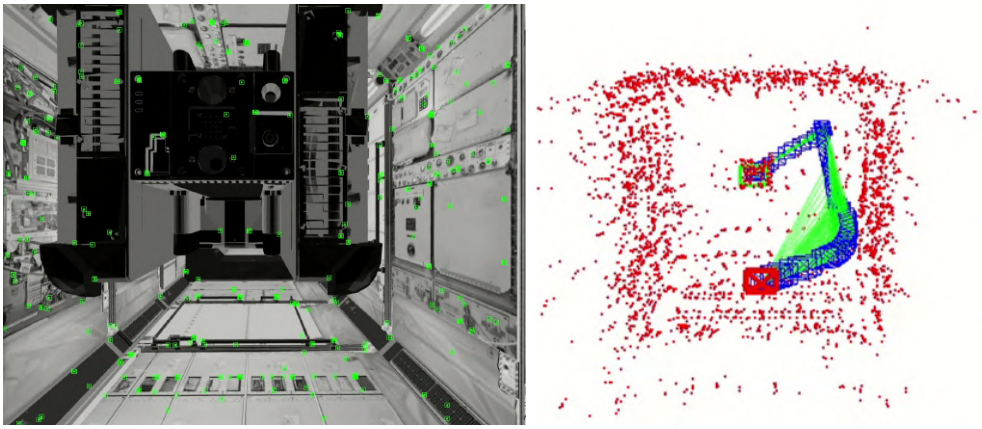
(c)

Figure 3.46: SLAM on the first half of the perception-aware Astrobee trajectory.



(a)

(b)

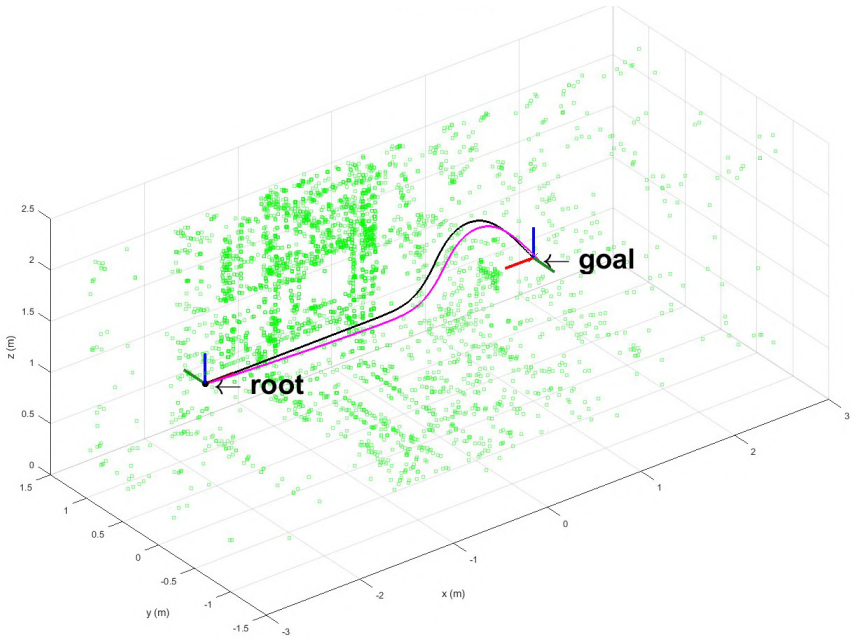


(c)

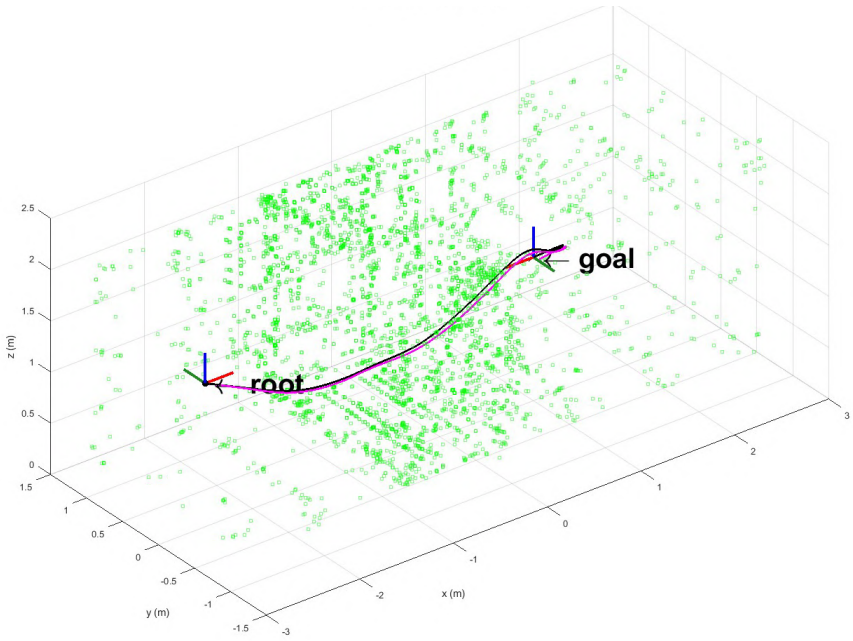
Figure 3.47: SLAM on the second half of the perception-aware Astrobee trajectory.

Localization error comparison

Figure 3.48 compares the resulting localization errors for the energy-optimal and the perception-aware trajectories for the Astrobee rendezvous task. The perception-aware localization is 37% more accurate than the energy-optimal one, with an RMS error of 5.2331 [cm] versus 8.2946 [cm].
It is also noticeable that the generated feature map in the perception-aware case is more dispersed and representative. In contrast, the one from the energy-optimal case concentrates most of the features on the left wall around x = 0 [m].



(a) Energy-optimal. RMS error: 8.2946 [cm]



(b) Perception-aware. RMS error: 5.2331 [cm]

Figure 3.48: Astrobee rendezvous localization error comparison. The real trajectory is shown in black, and the trajectory estimated by SLAM is shown in magenta.

3.3.3 RRT*-GBO solutions

The complete RRT*-GBO planner is now employed for the task at hand. The following set of tuning parameters is chosen:

$$r_{\text{ball,translation}} = 2, \tag{3.28}$$

$$r_{\text{connect,translation}} = 2, \tag{3.29}$$

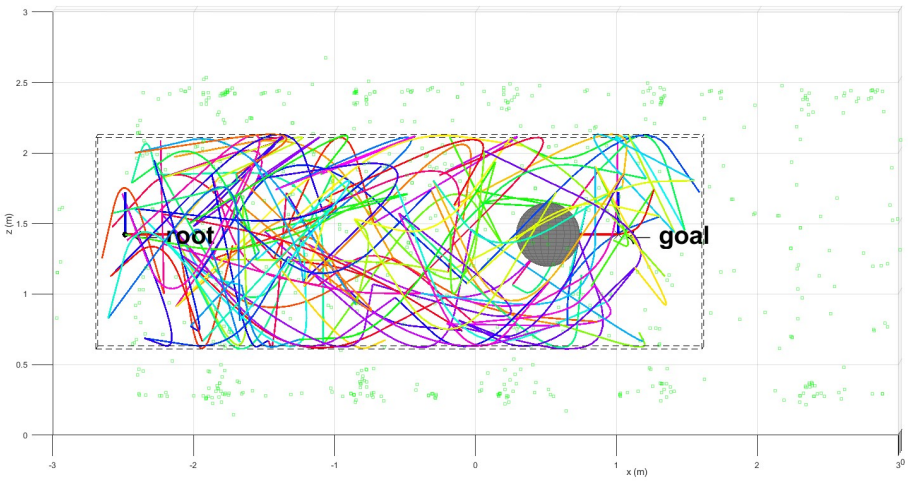
$$r_{\text{prune,translation}} = 0.5, \tag{3.30}$$

$$r_{\text{ball,rotation}} = \pi/2, \tag{3.31}$$

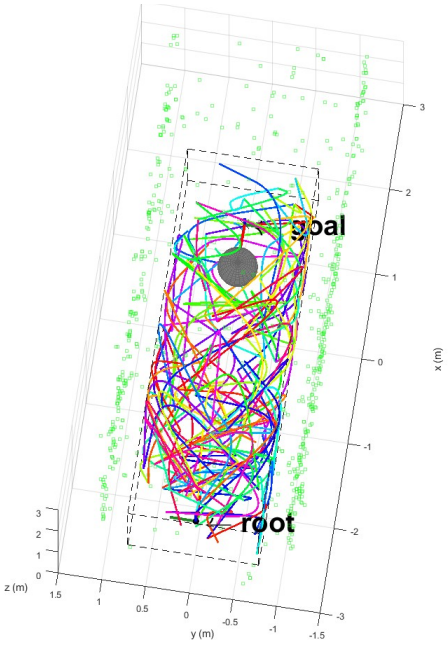
$$r_{\text{connect,rotation}} = \pi, \tag{3.32}$$

$$r_{\text{prune,rotation}} = \pi/12, \tag{3.33}$$

And he edges B-Splines have a duration $t_{f,\text{edges}} = 60$ [s].
The algorithm runs until 32 solutions are found, resulting in the dense tree of edges shown in Fig. 3.49. Figure 3.50 shows all the found solutions, which fall into the usual pattern of bringing the robot to the limits of the translational constraints to ensure its camera views the largest possible number of features.
The solutions are color-coded by the costs according to the colormap in Fig. 3.50(c). They can be put together into a few distinct groups. One trajectory is sampled from the lower-left corner group (dark blue: best solutions), one from the upper-right corner group (medium shades of blue: good solutions), and one from the lower-right corner group (light blue, green and yellow: average solutions). The robot poses over time for the selected solutions/trajectories are shown in Fig. 3.50.
The ensuing localization errors from executing them with SLAM are shown in Fig. 3.52. As expected, the localization in the upper-right corner solution (Fig. 3.52(b)) is better than in the lower-right corner solution (Fig. 3.52(c)), and both are better than in the baseline energy-optimal trajectory (Fig. 3.48(a)). However, the lower-left trajectory (Fig. 3.52(a)), which has the lowest optimized cost, shows a localization error even larger than in the baseline energy-optimal trajectory (Fig. 3.48(a)). The camera view during the execution of the lower-left trajectory (Fig. 3.53) provides some insight into the unexpected result. The robot looks at the panels with wires and electronic components on the right wall, which have the highest density of features in the environment. However, it does it by following an aggressive trajectory with large rotations. Thus, it is likely that even the best view of the features cannot compensate for the difficulty of estimating such an aggressive trajectory. This hypothesis is reinforced by comparing Fig. 3.48(b) to Fig. 3.52(b). Both trajectories reach the goal by following an arch towards the upper-right corner. The RRT*-GBO trajectory has a larger (more aggressive) arch and results in a slightly less accurate localization. If the hypothesis stands true, a finer tuning of the cost terms weighting factor w_{energy} or additional motion constraints could help avoid aggressive trajectories.

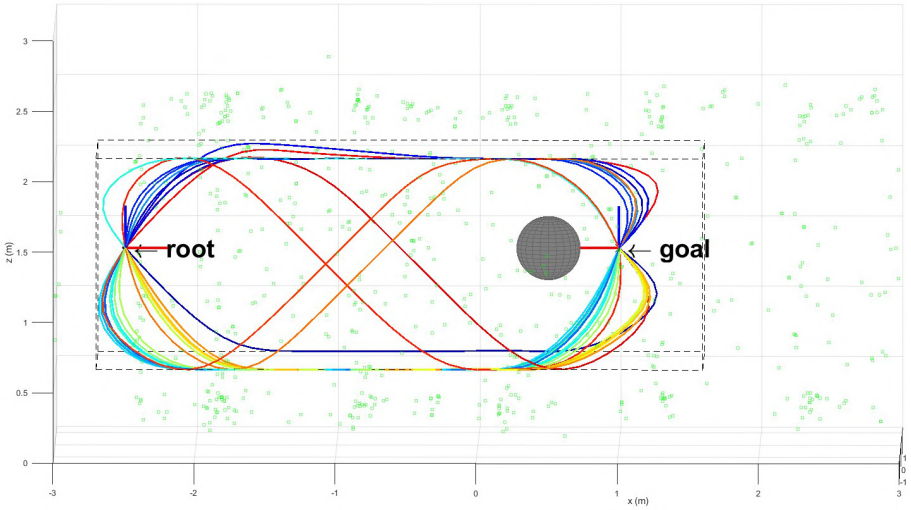


(a) Side view

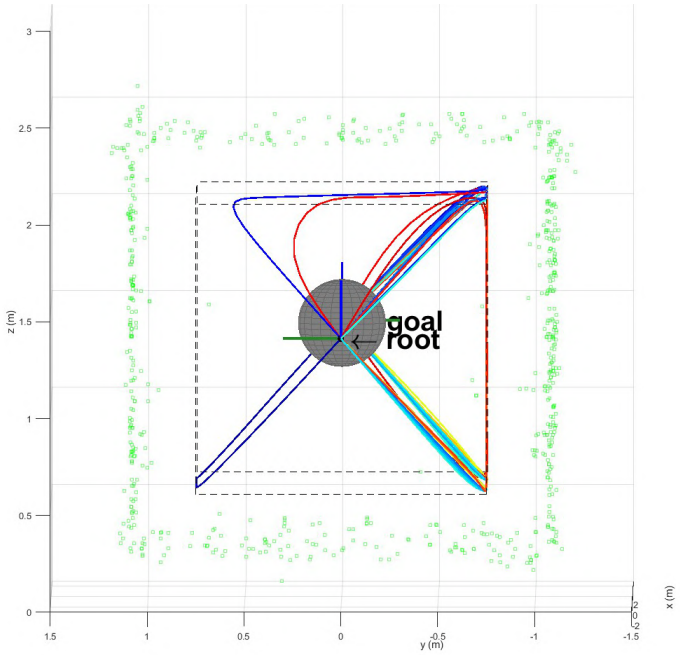


(b) Top view

Figure 3.49: Astrobee rendezvous RRT*-GBO tree. Randomly colored.



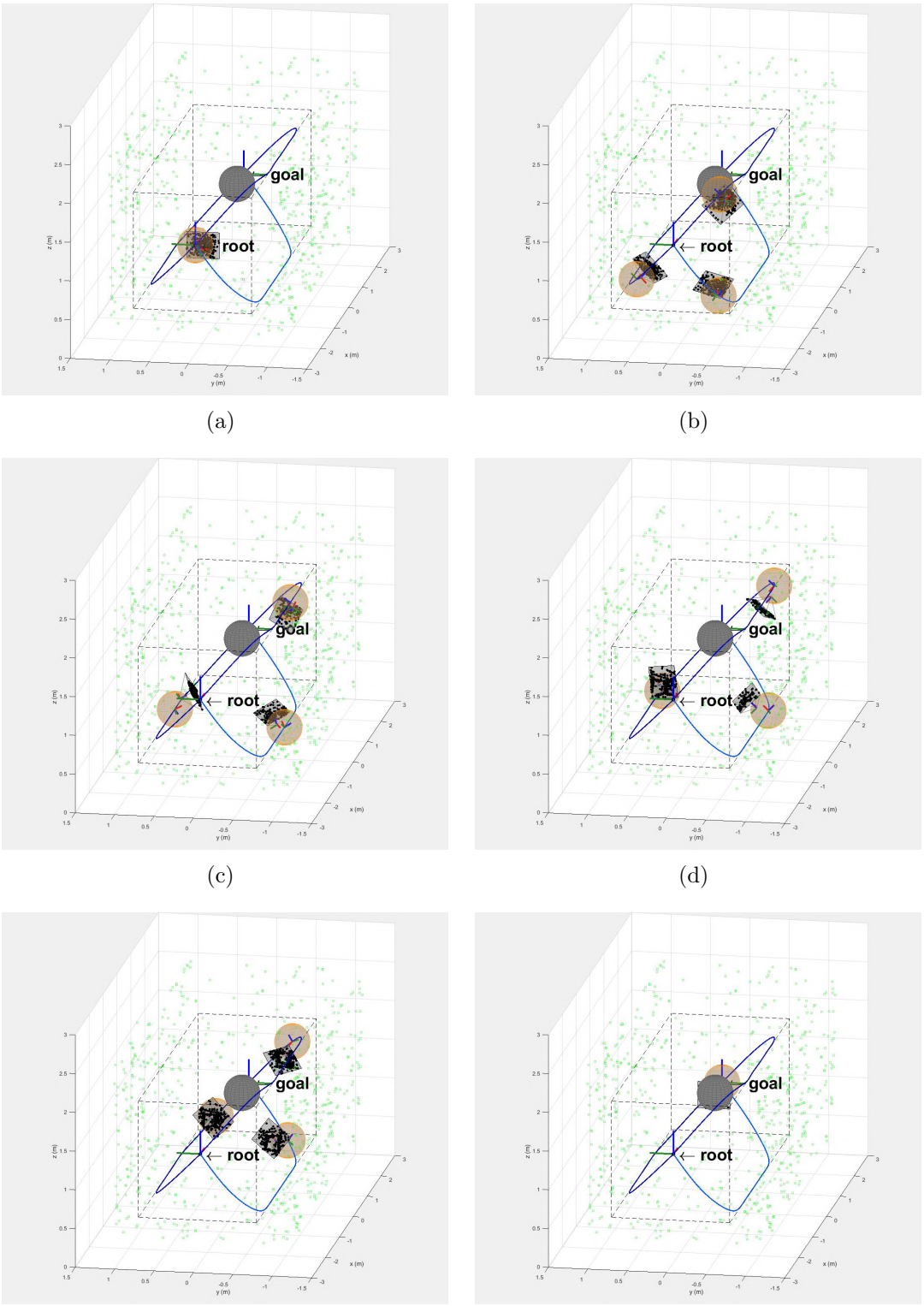
(a) Side view



(b) Front view

(c) Colormap (dark blue for best cost, dark red for worst cost).

Figure 3.50: Color-coded astrobee rendezvous RRT*-GBO solutions.

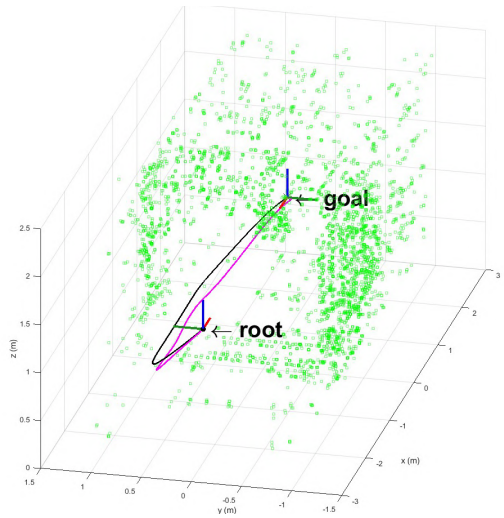


(a) (b)

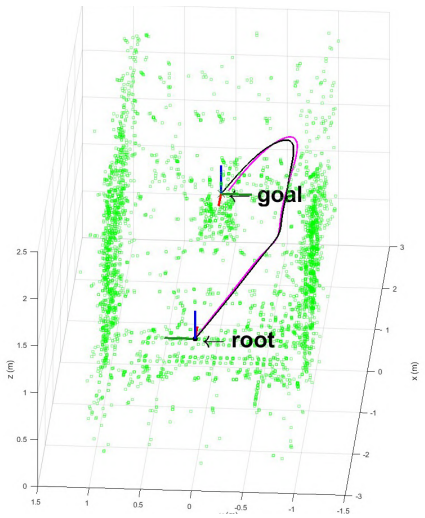
(c) (d)

(e) (f)

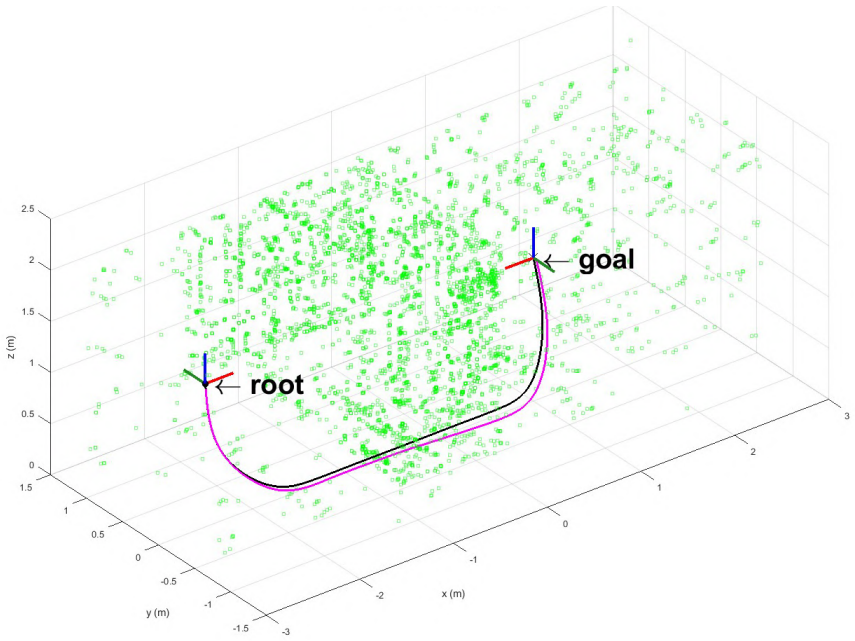
Figure 3.51: Three selected trajectories found with RRT*-GBO for the Astrobee rendezvous task. The consecutive robot poses over time for the three trajectories are shown simultaneously.



(a) Lower-left corner solution.
RMS error: 9.0482 [cm]



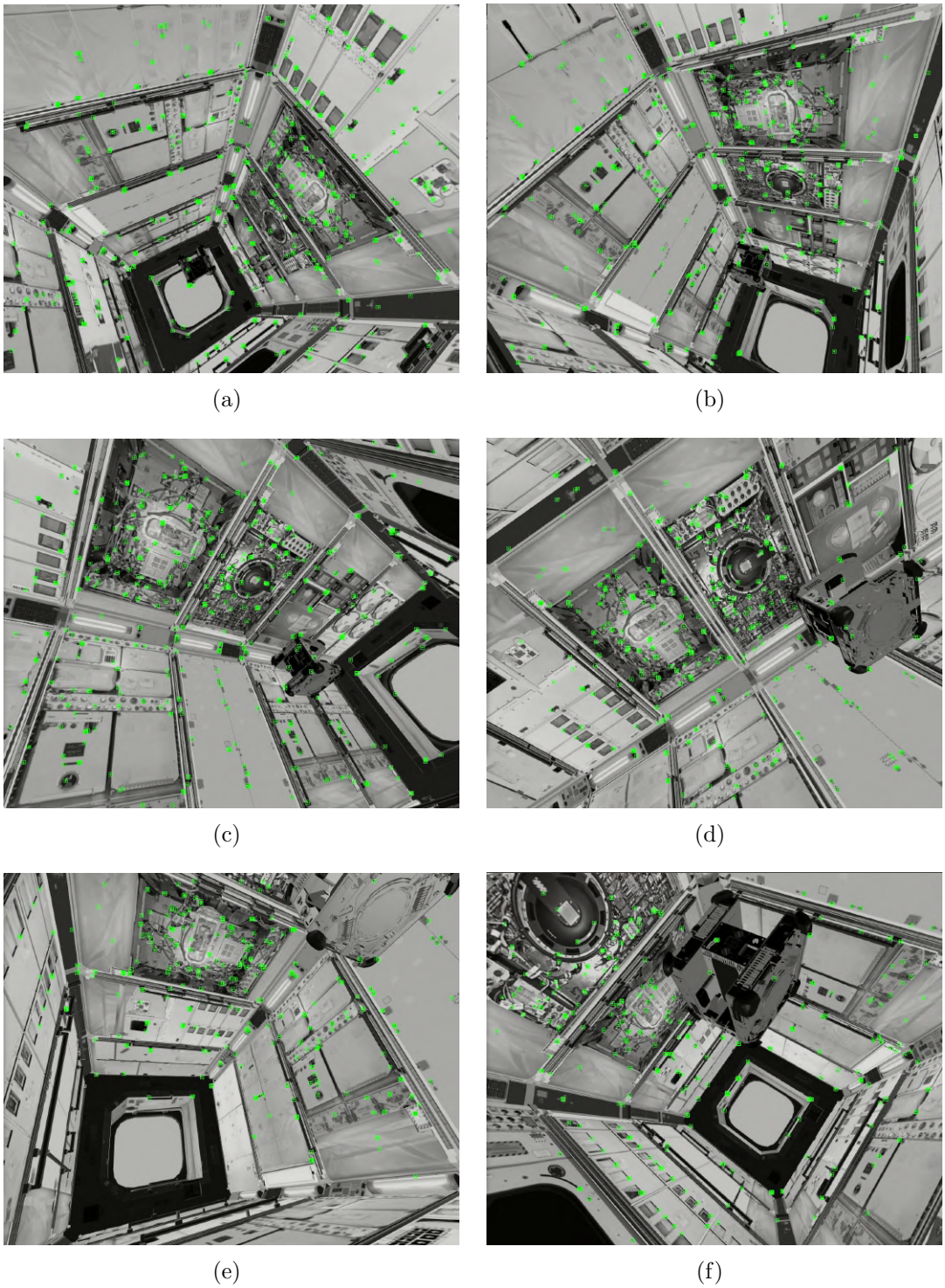
(b) Upper-right corner solution.
RMS error: 5.51 [cm]



S

(c) Lower-right corner solution. RMS error: 7.1168 [cm]

Figure 3.52: Localization error comparison for three selected trajectories found with RRT*-GBO for the Astrobee rendezvous task.



(a) (b) (c) (d) (e) (f)

Figure 3.53: Camera view with features during the execution of the lower-left corner solution found with RRT*-GBO for the Astrobee rendezvous task.

3.4 Satellite capture experiment

Lastly, the complete method is evaluated using real hardware. An on-orbit satellite capture experiment is conducted using DLR's OOS-SIM experimental facility [ADSR+15] shown in Fig. 3.54. Two large industrial manipulators (Kuka KR120) are used to simulate zero-gravity dynamics for a servicer (on the left) and a client (on the right), The servicer simulates an autonomous spacecraft and has a lightweight manipulator (Kuka LWR) with a camera attached to it. The client simulates a satellite. The task for this experiment is to have the lightweight manipulator grasp the handle on the rim of the satellite mock-up. We optimize the approaching trajectory of the servicer toward the client, which is executed by moving the KR120 while the LWR has a fixed joint configuration.

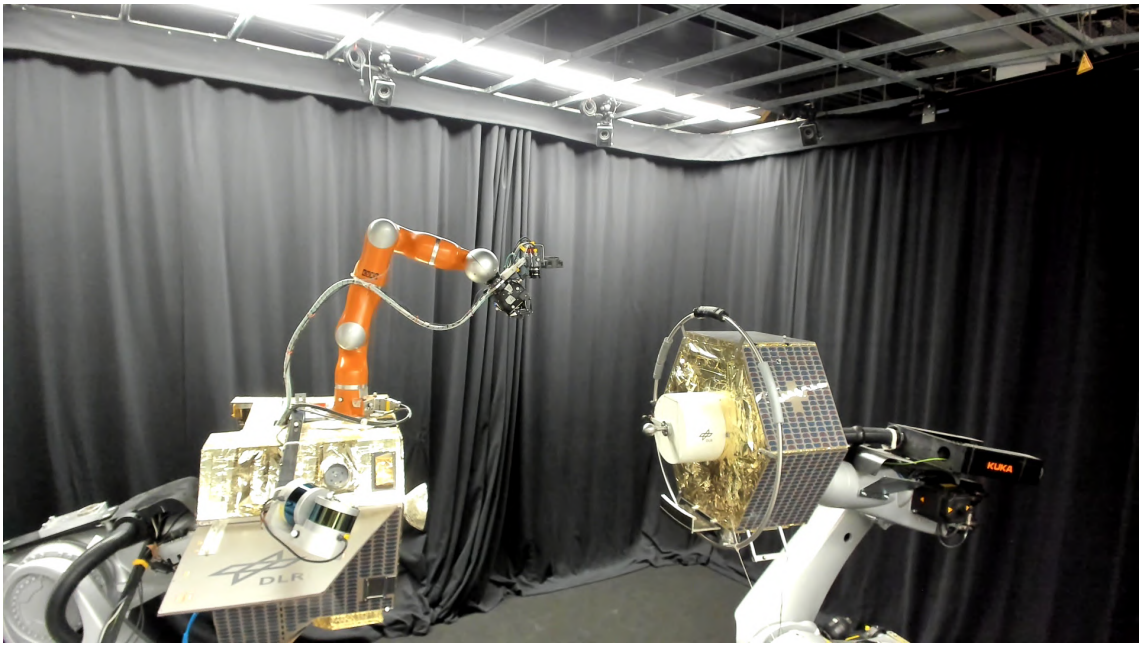


Figure 3.54: DLR's OOS-SIM experimental facility.

According to the pinhole model from Sec. 1.4.6, the OOS-SIM robot camera has the following parameters: focal length $f = 683.9$, principal point offset $p_x = 266.6$, $p_y = 229.7$, image width $W = 528$, image height $H = 406$. Therefore, the intrinsics matrix is

$$\mathbf{K} = \begin{bmatrix} 683.9 & 0 & 266.6 \\ 0 & 683.9 & 229.7 \\ 0 & 0 & 1 \end{bmatrix} . \tag{3.34}$$

The camera extrinsics matrix is

$$\begin{bmatrix} & \mathbf{H}^{W}_{\text{cam}} & & \\ 0 & 0 & 0 & 1 \end{bmatrix} = \left(\mathbf{T}^{\text{cam}}_{\text{KR120 EE}}\left(\mathbf{q}_{\text{LWR}}\right)\right)^{-1} \cdot \left(\mathbf{T}^{\text{KR120 EE}}_{W}\right)^{-1} , \tag{3.35}$$

for a commanded pose of the KR120's end-effector $\mathbf{T}_W^{\mathrm{KR120\,EE}}$. For the LWR's joint configuration $\mathbf{q}_{\mathrm{LWR}}$ shown in Fig. 3.54.

$$\mathbf{T}_{\mathrm{KR120\,EE}}^{\mathrm{cam}}\left(\mathbf{q}_{\mathrm{LWR}}\right) = \begin{bmatrix} 1 & -0.05 & 0.02 & 0.02 \\ -0.05 & -1 & 0.12 & 0.73 \\ 0.02 & -0.12 & -1 & 0.61 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.36}$$

The robot's dynamic properties and motion constraints are the same as those used for the Astrobee in the experiment from Sec. 3.3, except the position boundaries (for the KR120 end-effector).

$$\begin{bmatrix} 0 \\ -0.75 \\ 1.5 \end{bmatrix} \leq \mathbf{t} \leq \begin{bmatrix} 1.5 \\ 0.75 \\ 3 \end{bmatrix} \tag{3.37}$$

3.4.1 Mapping and perception metric interpolation

A map of the OOS-SIM is obtained by commanding the Servicer KR120 to perform small translations on a plane parallel to the front of the satellite mock-up (golden reflexive surface), as illustrated in Fig. 3.55. The obtained map is downsampled by a factor of 3, resulting in 366 features.
The perception metric is interpolated for this feature map as in (2.26). The following grid is used:

- x: 6 linearly space points in the range $[0; 1.5]$.
- y: 6 linearly space points in the range $[-0.75; 0.75]$.
- z: 6 linearly space points in the range $[1.5; 3]$.
- ξ_x: 19 linearly space points in the range $[0; \pi]$.
- ξ_y: 19 linearly space points in the range $[0; \pi]$.
- ξ_z: 10 linearly space points in the range $[0; \pi]$.

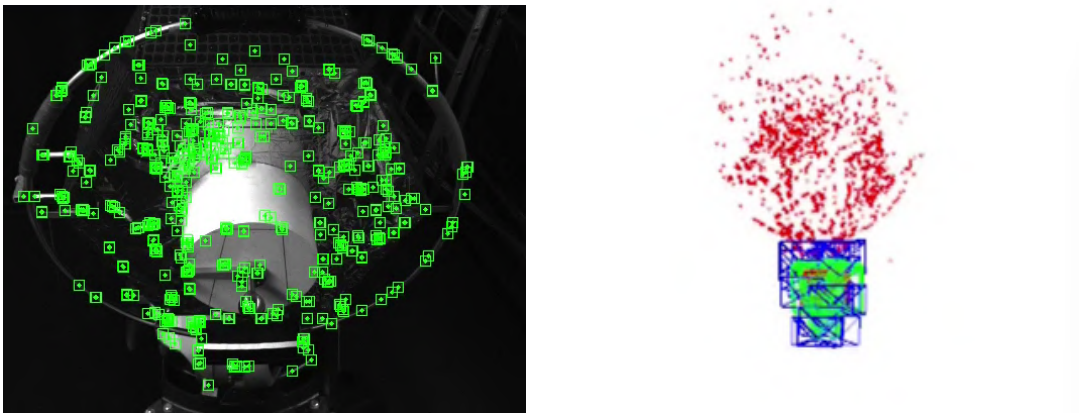


Figure 3.55: OOS-SIM mapping. On the left: camera view with green markers for features. On the right: SLAM map features (red dots) and camera localization (crossed rectangles).

3.4.2 Trajectory planning and execution

The task is to bring the servicer KR120 close enough to the servicer KR120 to allow the LWR to grasp the rim of the satellite mock-up, with

$$\mathbf{s}_{\text{root}} = \begin{bmatrix} 0.5 & 0 & 2 & 0 & 0 & -\pi \end{bmatrix}^T, \tag{3.38}$$

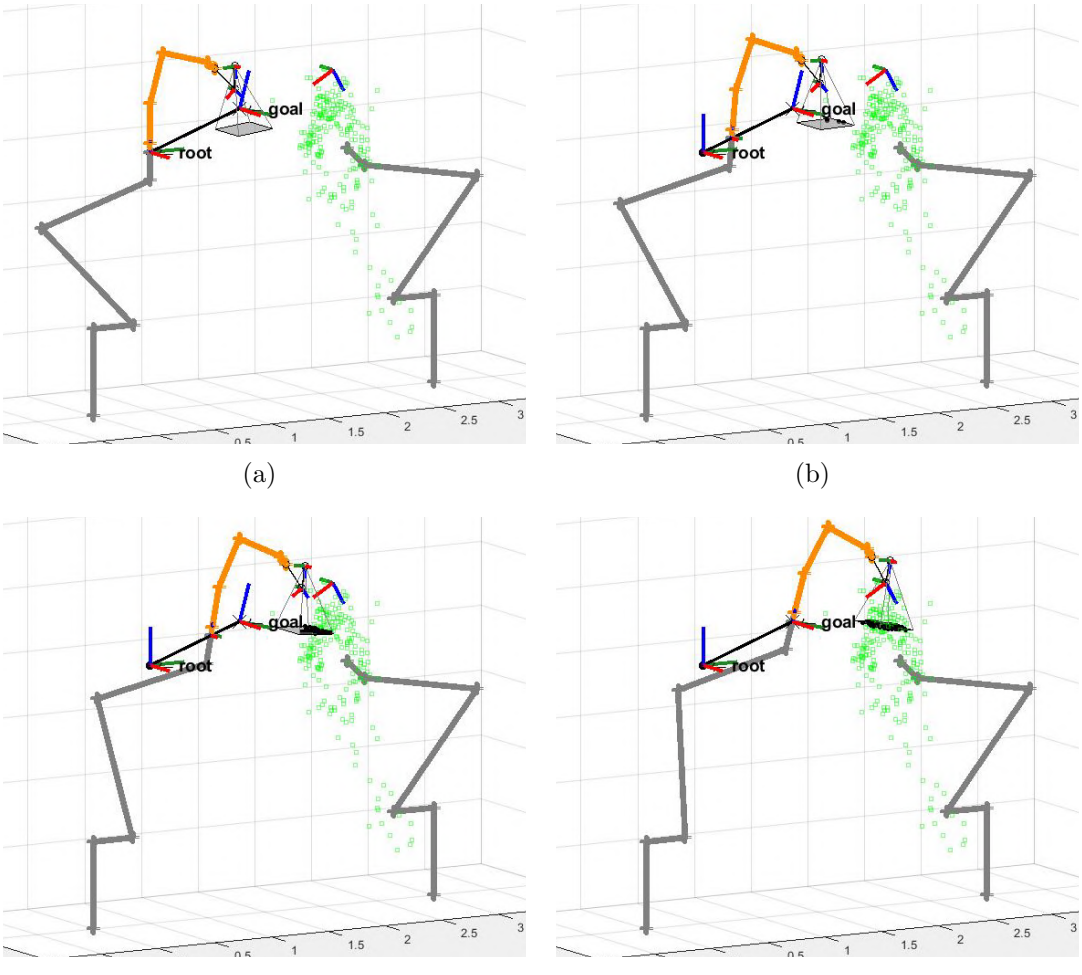
$$\mathbf{s}_{\text{goal}} = \begin{bmatrix} 1.32 & 0.05 & 2.26 & -0.18 & 0.23 & -1.51 \end{bmatrix}^T, \tag{3.39}$$

$$t_f = 60 \quad [\text{s}]. \tag{3.40}$$

As in the Astrobee rendezvous experiment (Sec. 3.3), an energy-optimal and a perception-aware trajectory are compared to evaluate the localization accuracy. In both cases, a straight-line initial guess is fed to the optimizer, skipping the RRT*-GBO edges construction part.
The optimizer's stopping criteria are set to $\Delta\Gamma_{\text{rel}} = 1e^{-4}$ and $\Delta\mathbf{p}_{\text{free,rel}} = 1e^{-4}$. This choice makes the optimizer stop earlier, which is necessary to ensure the trajectory is conservative enough to be executed by the KR120 manipulator.

Energy-optimal

The trajectory in Fig. 3.56 is planned by optimizing only the energy spent by the robot (i.e., $w_{\text{energy}} = 1$, $w_{\text{perception}} = 0$). In fact, it is exactly the straight-line initial guess since it is already optimal for this case.
The trajectory execution on the OOS-SIM facility is shown in Fig 3.57. The camera view with detected features and the SLAM map and camera localization are shown in Fig. 3.58.
Following the energy-optimal trajectory, the camera only sees the floor and, later, the white cylinder on the satellite mock-up from above. Therefore, few features are detected, and the mapping and localization initialization is delayed.



(a) (b)

(c) (d)

Figure 3.56: Planned energy-optimal OOS-SIM trajectory. The consecutive robot poses over time are shown.

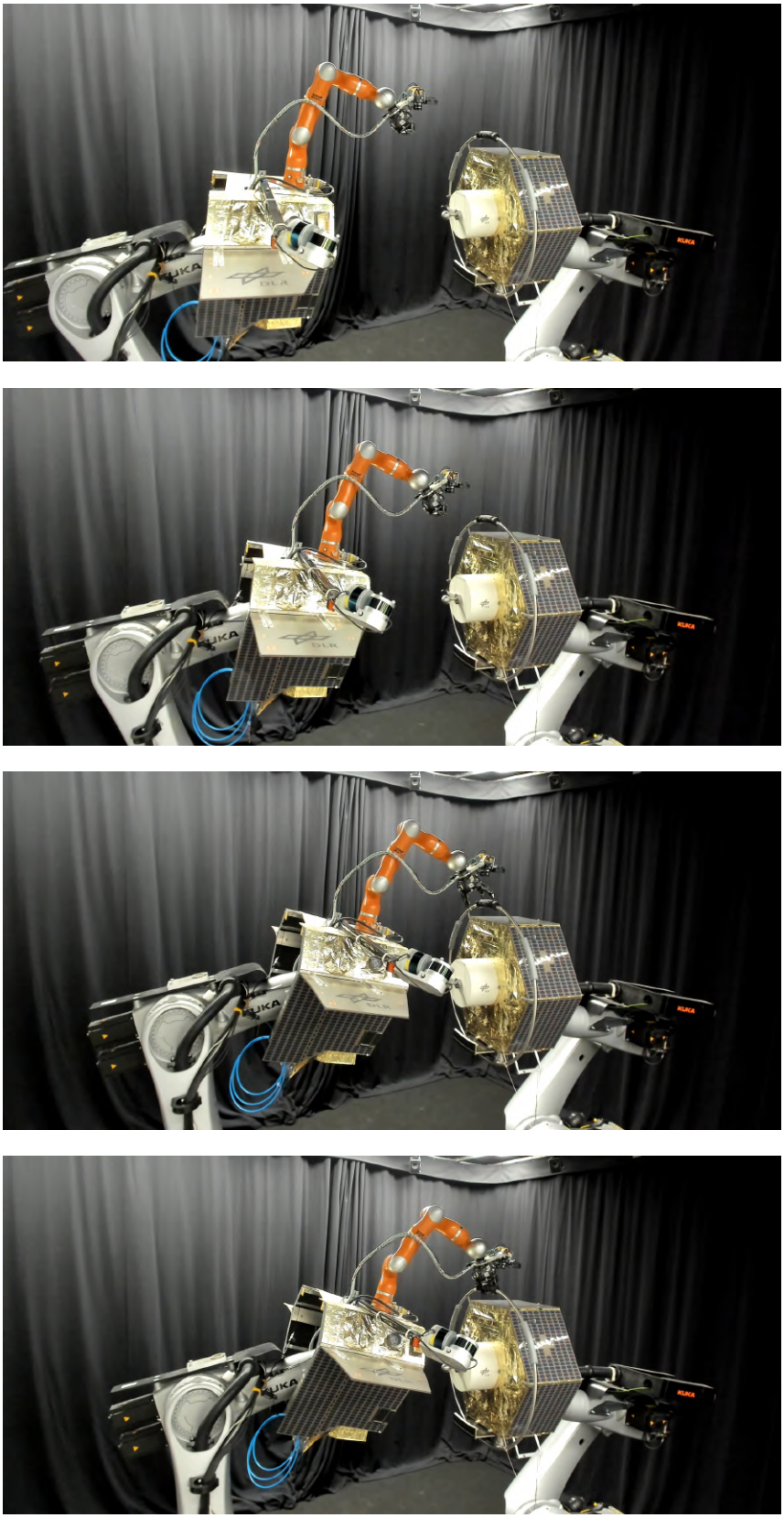
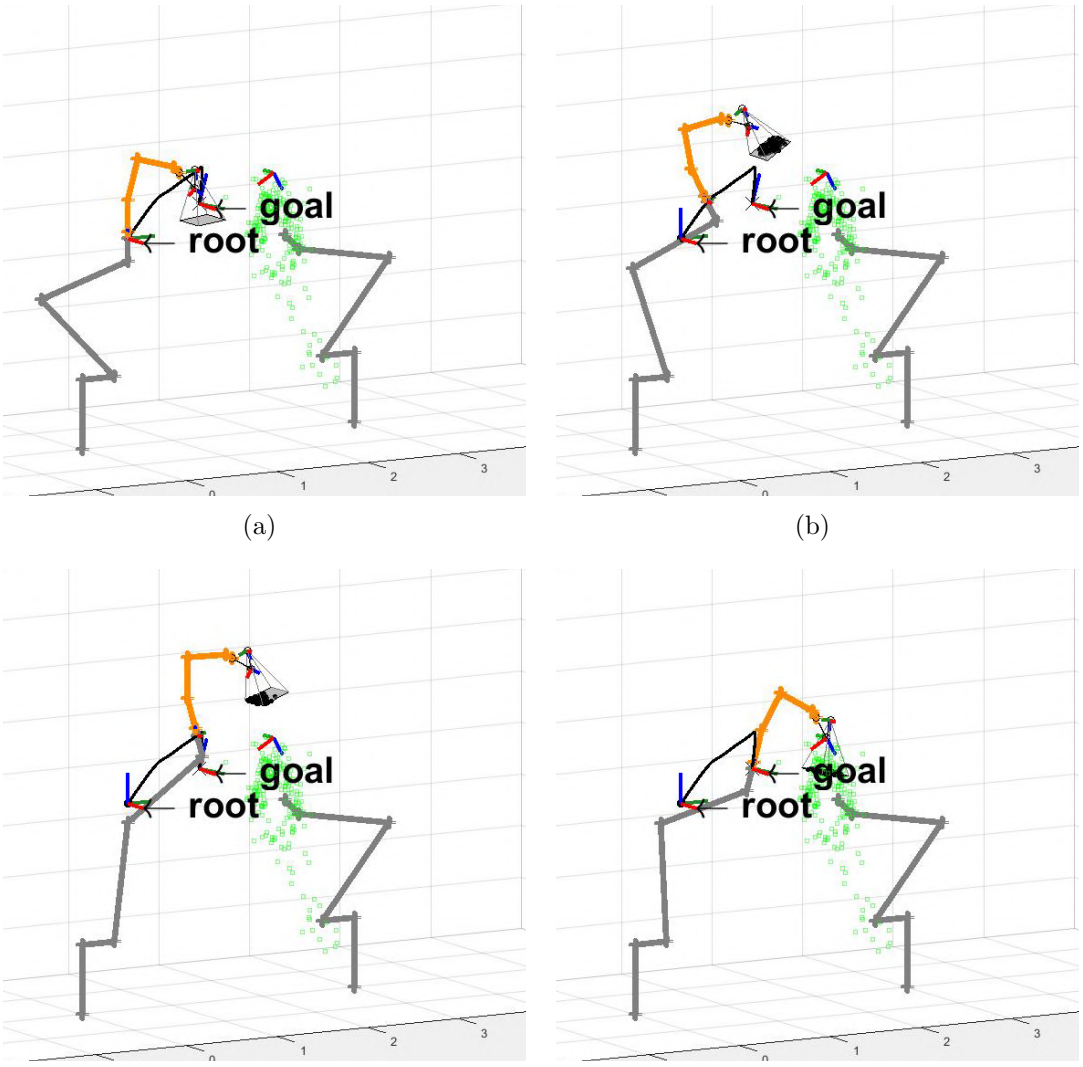


Figure 3.57: Energy-optimal trajectory execution on the OOS-SIM. The consecutive robot configurations over time are shown.

Figure 3.58: SLAM on the energy-optimal OOS-SIM trajectory.

Perception-aware

The trajectory in Fig. 3.59 is planned using a weighted perception-aware cost metric ($w_{\text{energy}} = 0.9$, $w_{\text{perception}} = 0.1$).
The trajectory execution on the OOS-SIM facility is shown in Fig 3.60. The camera view with detected features and the SLAM map and camera localization are shown in Fig. 3.61.
Following the perception-aware trajectory, the manipulator's end-effector tilts back and goes up so that the camera sees a major part of the satellite mock-up while approaching it for the grasp. Therefore, many features are detected, and the mapping and localization are soon initialized.



(a) (b) (c) (d)

Figure 3.59: Planned perception-aware OOS-SIM trajectory. The consecutive robot configurations over time are shown.

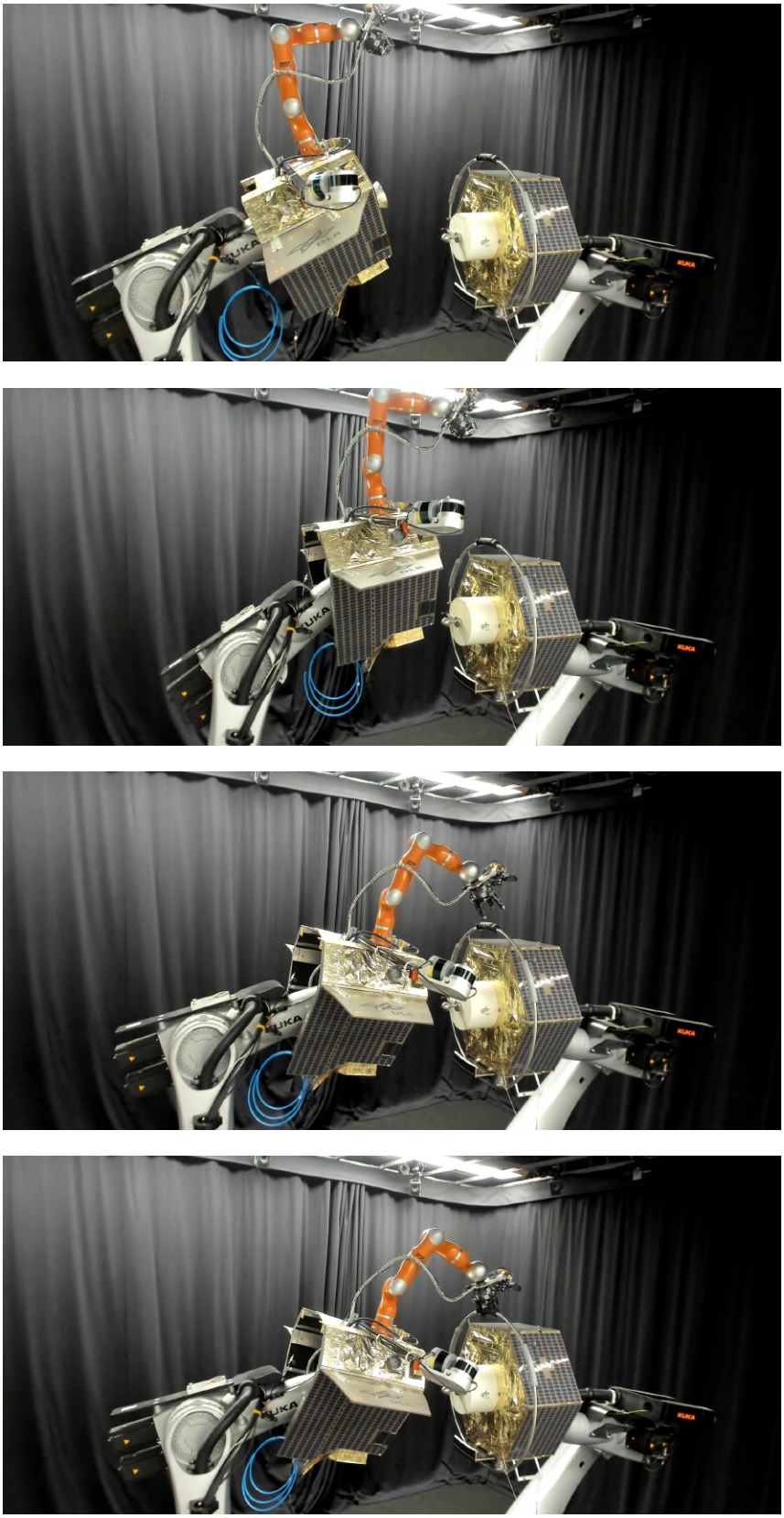


Figure 3.60: Perception-aware trajectory execution on the OOS-SIM. The consecutive robot configurations over time are shown.

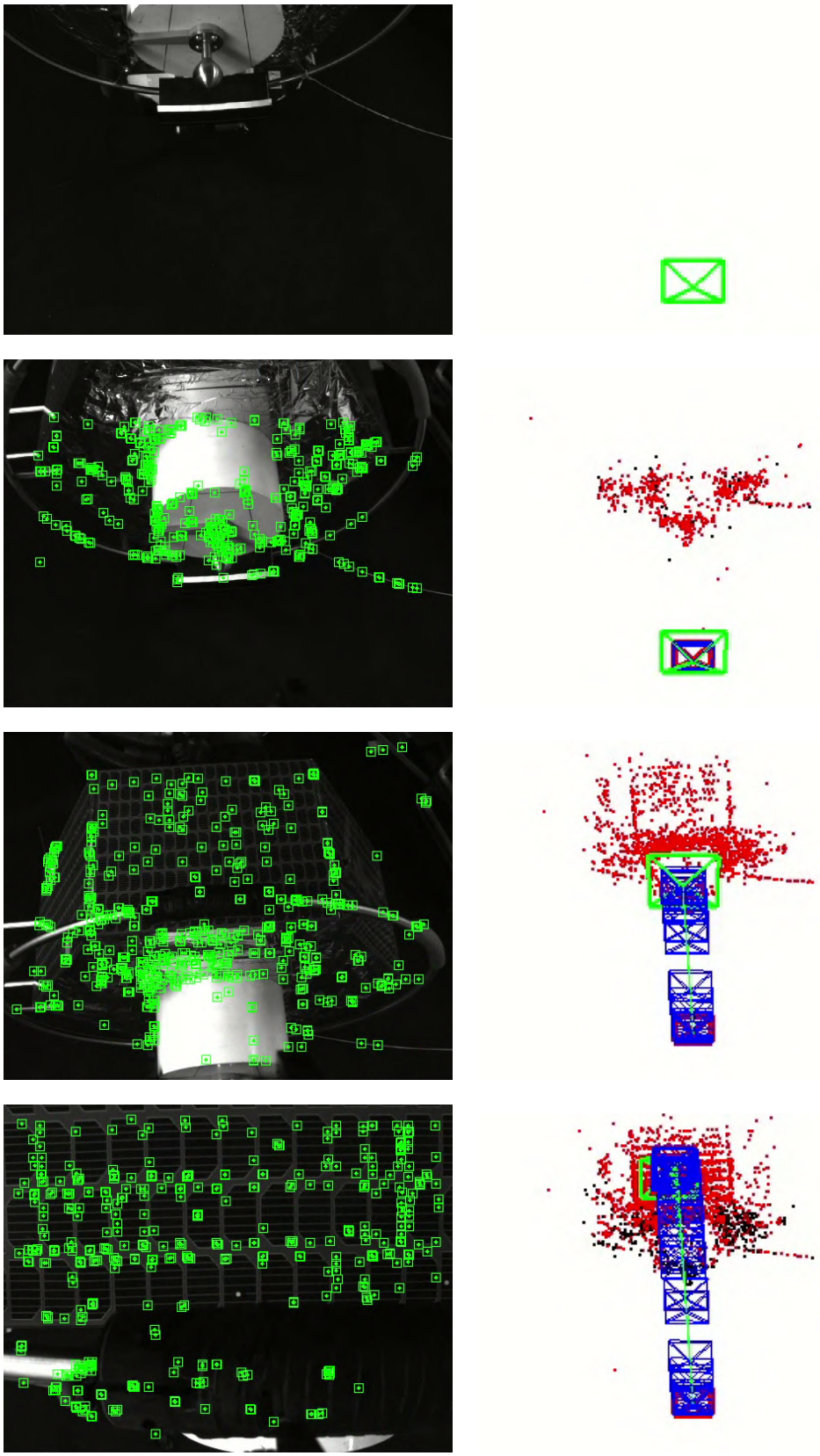
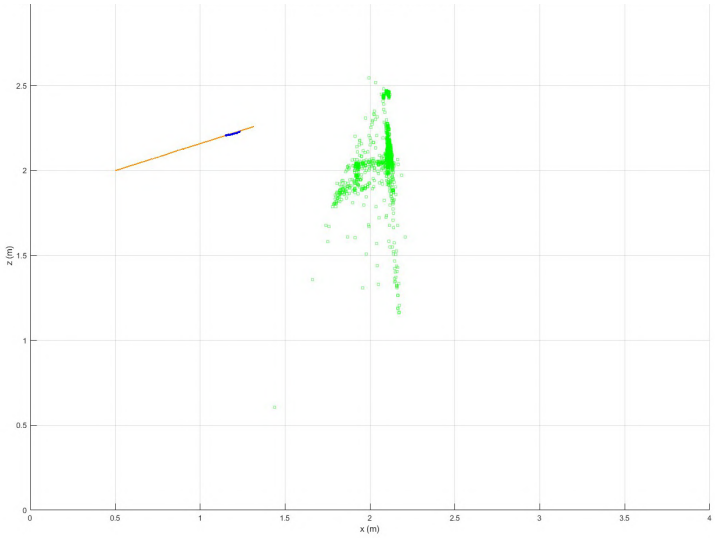


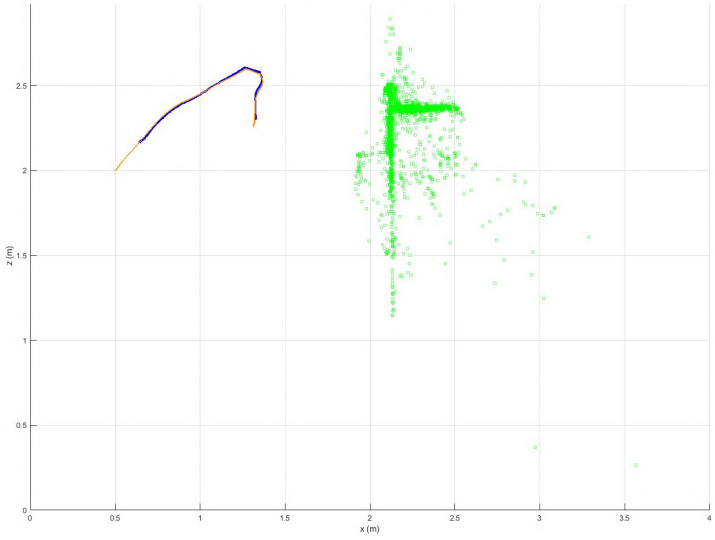
Figure 3.61: SLAM on the energy-optimal OOS-SIM trajectory.

Localization error comparison

Figure 3.62 shows that the energy-optimal trajectory leads to a late localization initialization which estimates only a minimal fraction of the robot's trajectory. In contrast, the perception-aware trajectory has the localization initialized in an acceptable time and accurately estimates the real robot trajectory. As a side-product, the feature map resulting from the perception-aware trajectory is more representative than the one from the energy-optimal case.



(a) Time to start localization: 24.6036 [s].



(b) Time to start localization: 13.9631 [s]. RMS error: 2.027 [cm]

Figure 3.62: Satellite capture localization error comparison. The real trajectory is shown in orange, and the trajectory estimated by SLAM is shown in blue.

Chapter 4

Discussion

Section 3.1 exposes local minima that prevent gradient-based optimization from finding the global minimum or even converging to a feasible solution for the imposed constraints. Different initial guesses provided to the optimizer converge to different local minima, which justifies exploring various initial guesses. A Monte Carlo approach takes random samples from the space of initial guesses. In infinite time, all possible initial guesses are attempted, and the best solution is found. In practice, time is limited, and a more clever method to search initial guesses is desirable. The method chosen is RRT*-GBO which verifiably finds informed initial guesses that lead to optimal solutions faster than Monte Carlo. The discrepancy is proportional to the complexity of the motion planning task, reaching a 62-fold computation time reduction in a very cluttered SE(3) scenario (Maze from 3.1.2). For future work, warm-starting the optimizer via look-up tables could achieve even higher time savings, as shown in [SBL23, LNTC$^+$11].

It is also clear from Section 3.1 that the performance of the RRT*-GBO in a given scenario depends on the choice of its tuning parameters. Manually adjusting the parameters can be time-consuming, which motivates an automated way to search optimal parameters such as Optuna [ASY$^+$19], a popular hyperparameter optimization framework. The energy-weighting term from (2.27) could also benefit from it, exploring the trade-off between energy- and perception-optimality evaluated in Sec. 3.2.3.

The perception-optimal solutions shown in Section 3.2 prove that the designed perception functions (2.10) and (2.19) correctly model the visibility of the environment features. The solutions assimilate a human-like, intuitive behavior of moving away from the features to have a better overview while focusing on the most descriptive region (with the highest density of features). A limitation in the perception functions (2.10) and (2.19) is that they do not take into account feature occlusion which may occur in a scenario where an obstacle blocks the camera view of certain features. Ray-tracing methods could help model the occlusion of features.

A 10-fold reduction in the optimizer's convergence time is achieved by performing an offline multidimensional cubic spline interpolation of the perception functions.

Additionally, it makes the discontinuous feature count function (2.10) suitable for gradient computation. The same could be applied to other discontinuous functions, which could then be optimized by the gradient-based solver.
The Astrobee rendezvous experiment in Sec. 3.3 establishes that energy-weighted perception-aware trajectory planning results in 1.6 times higher localization accuracy than the energy-only optimal trajectory. Furthermore, the usage of RRT*-GBO joins the findings of Sections 3.1 and 3.2 to provide informed initial guesses to energy-weighted perception-aware optimization.
Real-world validation is obtained for a satellite capture task using DLR's OOS-SIM experimental facility in Sec. 3.4. Perception-aware trajectory planning achieved accurate robot localization in contrast to practically complete localization failure for the energy-only optimal trajectory.

Chapter 5

Conclusion

This thesis presents a novel application of perception-aware trajectory optimization for on-orbit servicing tasks, targeting free-flying robots in SE(3).
All six translation and rotation degrees of freedom of the robot are explored to plan perception-optimized trajectories that contribute to accurate robot localization during the trajectory execution. The perception metric employed by the optimizer makes the robot behave in an intuitive manner during its trajectory: look at the most visually descriptive regions while maintaining enough distance from them to have a broader overview.
Major time savings are obtained by pre-computing an interpolation of the perception metric that can be efficiently queried for the planning of distinct robot trajectories. An extensive analysis is performed on the multiple possible solutions for a trajectory optimization task and the dependency on the initial guess provided. Different experiments highlight the substantial time savings in the use of the RRT*-GBO algorithm to explore promising initial guesses that lead to globally optimal solutions.
Experiments performed in a realistic simulation of the Astrobees in the ISS and also in the physical DLR's OOS-SIM facility validate this thesis' method and clearly show the improvement of robot localization accuracy.
In the future, the method could be tested for the real Astrobee robots inside the International Space Station and, finally, for the autonomous execution of a satellite servicing task in outer space.

Appendix A

RRT*-GBO Pseudo-code

Algorithm 1 shows the main loop for the RRT*-GBO iterations, composed by Algorithms 2 to 5, which, in turn, employ Algorithms 6 and 7.

Algorithm 1 main()

```
tree ← Tree(root, goal)                                   ▷ global variable
while stopping criteria are not reached do
    connect_to_goal()                                     ▷ Algorithm 2
    new_node ← sample()                                   ▷ Algorithm 3
    neighboring_nodes ← neighborhood_query(new_node)      ▷ Algorithm 4
    rewire(neighboring_nodes)                             ▷ Algorithm 5
end while
```

Algorithm 2 connect_to_goal()

```
if distance(tree.last_added_node, tree.goal_node) > r_connect then
    return
end if
new_edge ← build_edge(tree.last_added_node, tree.goal_node)          ▷ GBO
if not new_edge.feasible then
    return
end if
if tree.goal_node.incoming_edge ≠ nothing then
    new_total_cost ← tree.last_added_node.cost_to_go + new_edge.cost
    if new_total_cost ≤ tree.goal_node.cost_to_go then
        return
    end if
end if
tree.add_edge(new_edge)
smooth()                                                  ▷ Algorithm 6
```

Algorithm 3 sample()

```
sampled ← Node()
while true do                                  ▷ repeat until a feasible sample is returned
    sampled.t ← sample_position_uniformely()   ▷ random, within position limits
    sampled_quaternion ← sample_SO3()          ▷ Yer+10 [YLM10]
    sampled.ξ ← quaternion_to_angle_axis(sampled_quaternion)
    closest_node ← find_closest_in_tree(sampled)
    sampled ← steer(closest_node, sampled)     ▷ Algorithm 7
    if distance(node, sampled) ≥ r_prune and not causes_collision(sampled) then
        return sampled
    end if                                     ▷ else, re-sample
end while
```

Algorithm 4 neighborhood_query(new_node)

```
neighboring_nodes ← [ ]
best_edge ← nothing
for node in tree.nodes do
    if r_prune ≤ compute_distance(node, new_node) ≤ r_ball then
        edge ← build_edge(node, new_node)      ▷ GBO
        if not edge.feasible then
            continue
        end if
        neighboring_nodes.add(node)
        if edge.cost < best_edge.cost or best_edge == nothing then
            best_edge ← edge
        end if
    end if
end for
tree.add(best_edge)
return neighboring_nodes
```

Algorithm 5 rewire(neighboring_nodes)

```
for node in neighboring_nodes do
    edge ← build_edge(tree.last_added_node, node)                    ▷ GBO
    if not edge.feasible then
        return
    end if
    if tree.last_added_node.cost_to_go + edge.cost < node.cost_to_go then
        delete node.incoming_edge
        tree.add_edge(edge)
        path_to_goal ← get_path_to_node(tree.goal_node)
        if node in path_to_goal then
            smooth()                                                 ▷ Algorithm 6
        end if
    end if
end for
```

Algorithm 6 smooth()

```
bsplines ← [ ]
path_to_goal ← get_path_to_node(tree.goal_node)
for node in path_to_goal do
    bsplines.add(node.outgoing_edge.bspline)
end for
unified_bspline ← fit_into_one_bspline(bsplines)  ▷ least-squares at sampled times
smoothed_trajectory ← optimize(complete_bspline)    ▷ GBO (original problem)
save_solution(smoothed_trajectory)
```

Algorithm 7 steer(from_node, to_node)

```
if translation_distance(from_node.t, to_node.t) > r_ball_translation then
    relative_translation ← to_node.t - from_node.t
    capped_translation ← normalized(relative_translation)*r_ball_translation
    to_node.t ← from_node.t + capped_translation
end if
if rotation_distance(from_node.ξ, to_node.ξ) > r_ball_rotation then
    relative_rotation ← compute_relative_rotation(from_node.ξ, to_node.ξ)
    capped_rotation ← normalized(relative_rotation)*r_ball_rotation
    to_node.ξ ← apply_rotation(from_node.ξ, capped_rotiation)
end if
return to_node
```

List of Figures

Bibliography

[ADSR+15] Jordi Artigas, Marco De Stefano, Wolfgang Rackl, Roberto Lampariello, Bernhard Brunner, Wieland Bertleff, Robert Burger, Oliver Porges, Alessandro Giordano, Christoph Borst, and Alin Albu-Schaeffer. The oos-sim: An on-ground simulation facility for on-orbit servicing robotic operations. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2854–2860, 2015. doi:10.1109/ICRA.2015.7139588.

[Aer] AeroAstroMIT. How to reach a tumbling target in space. Visited on August 2, 2024. URL: https://youtu.be/IsEpmzFGFh8?si=e37BhvTOiSRgKIcb.

[AOS+21] Keenan Albee, Charles Oestreich, Caroline Specht, Antonio Espinoza, Jessica Todd, Ian Hokaj, Roberto Lampariello, and Richard Linares. A robust observation, planning, and control pipeline for autonomous rendezvous with tumbling targets. *Frontiers in Robotics and AI*, 8, 09 2021. doi:10.3389/frobt.2021.641338.

[ASM+22] Keenan Albee, Caroline Specht, Hrishik Mishra, Charles Oestreich, Bernhard Brunner, Roberto Lampariello, and Richard Linares. Autonomous rendezvous with an uncertain, uncooperative tumbling target: The tumbledock flight experiments. 06 2022.

[ASY+19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[BM08] Luigi Biagiotti and Claudio Melchiorri. *Trajectory Planning for Automatic Machines and Robots*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[BTC20] Luca Bartolomei, Lucas Teixeira, and Margarita Chli. Perception-aware path planning for uavs using semantic segmentation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5808–5815, 2020. doi:10.1109/IROS45743.2020.9341347.

[CEG+21] Carlos Campos, Richard Elvira, Juan J. Gómez, José M. M. Montiel, and Juan D. Tardós. ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.

[CER+21] Carlos Campos, Richard Elvira, Juan J. Gómez Rodríguez, José M. M. Montiel, and Juan D. Tardós. Orb-slam3: An accurate open-source library for visual, visualâinertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021. doi:10.1109/TRO.2021.3075644.

[CFD+17] Gabriele Costante, Christian Forster, Jeffrey Delmerico, Paolo Valigi, and Davide Scaramuzza. Perception-aware path planning, 2017. URL: https://arxiv.org/abs/1605.04151, arXiv:1605.04151.

[CFM+16] Brian Coltin, Jesse Fusco, Zack Moratto, Oleg Alexandrov, and Robert Nakamura. Localization from visual landmarks on a free-flying robot. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4377–4382, 2016. doi:10.1109/IROS.2016.7759644.

[DLR] DLR. Mini robots practise grasping space debris. Visited on August 1, 2024. URL: https://www.dlr.de/en/latest/news/2022/01/20220322_mini-robots-practise-grasping-space-debris.

[Far23] Tommaso Faraci. Sensor-based optimal control for an astrobee-robot on the iss. Master's thesis, Technical University of Munich, 2023. URL: https://mediatum.ub.tum.de/node?id=1747342.

[FFLS18] Davide Falanga, Philipp Foehn, Peng Lu, and Davide Scaramuzza. Pampc: Perception-aware model predictive control for quadrotors, 2018. URL: https://arxiv.org/abs/1804.04811, arXiv:1804.04811.

[fOSA] United Nations Office for Outer Space Affairs. Online index of objects launched into outer space. Visited on August 1, 2024. URL: https://www.unoosa.org/oosa/osoindex/.

[GKSB10] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010. doi:10.1109/MITS.2010.939925.

[Huy09] Du Q. Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164, 2009. doi:10.1007/s10851-009-0161-2.

[HZ04] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition, 2004.

[Joh07] Steven G. Johnson. The NLopt nonlinear-optimization package. `https://github.com/stevengj/nlopt`, 2007.

[KF11] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011. `arXiv:https://doi.org/10.1177/0278364911406761`, `doi:10.1177/0278364911406761`.

[Kra94] Dieter Kraft. Algorithm 733: TOMP–fortran modules for optimal control calculations. *ACM Transactions on Mathematical Software*, 20:262–281, 1994. `doi:10.1145/192115.192124`.

[LCS11] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *2011 International Conference on Computer Vision*, pages 2548–2555, 2011. `doi:10.1109/ICCV.2011.6126542`.

[LNTC+11] Roberto Lampariello, Duy Nguyen-Tuong, Claudio Castellini, Gerd Hirzinger, and Jan Peters. Trajectory planning for optimal robot catching in real-time. In *2011 IEEE International Conference on Robotics and Automation*, pages 3719–3726, 2011. `doi:10.1109/ICRA.2011.5980114`.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov 2004. `doi:10.1023/B:VISI.0000029664.99615.94`.

[LP17] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, USA, 1st edition, 2017.

[MSGK19] Varun Murali, Igor Spasojevic, Winter Guerra, and Sertac Karaman. Perception-aware trajectory generation for aggressive quadrotor flight using differential flatness. In *2019 American Control Conference (ACC)*, pages 3936–3943, 2019. `doi:10.23919/ACC.2019.8814697`.

[NASa] NASA. Astrobee. Visited on August 1, 2024. URL: `https://www.nasa.gov/astrobee/`.

[NASb] NASA. Astrobee guest science guide. Visited on August 1, 2024. URL: `https://www.nasa.gov/general/guest-science-resources/`.

[NP16] Huy Nguyen and Quang-Cuong Pham. Time-optimal path parameterization of rigid-body motions: Applications to spacecraft reorientation. *Journal of Guidance, Control, and Dynamics*, 39(7):1667–1671, 2016. arXiv:https://doi.org/10.2514/1.G001600, doi:10.2514/1.G001600.

[PR97] F. C. Park and Bahram Ravani. Smooth invariant interpolation of rotations. *ACM Trans. Graph.*, 16(3):277–295, jul 1997. doi:10.1145/256157.256160.

[RRKB11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011. doi:10.1109/ICCV.2011.6126544.

[SBL23] Caroline Specht, Abhiraj Bishnoi, and Roberto Lampariello. Autonomous spacecraft rendezvous using tube-based model predictive control: Design and application. *Journal of Guidance, Control, and Dynamics*, 46(7):1243–1261, 2023. arXiv:https://doi.org/10.2514/1.G007280, doi:10.2514/1.G007280.

[Sca85] L. E. Scales. *Introduction to non-linear optimization*. Springer-Verlag, Berlin, Heidelberg, 1985.

[SCSG19] Paolo Salaris, Marco Cognetti, Riccardo Spica, and Paolo Robuffo Giordano. Online optimal perception-aware trajectory generation. *IEEE Transactions on Robotics*, 35(6):1307–1322, 2019. doi:10.1109/TRO.2019.2931137.

[SKCS22] Ryan Soussan, Varsha Kumar, Brian Coltin, and Trey Smith. Astroloc: An efficient and robust localizer for a free-flying robot. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 4106–4112, 2022. doi:10.1109/ICRA46639.2022.9811919.

[SL14] Samantha Stoneman and Roberto Lampariello. Embedding nonlinear optimization in rrt* for optimal kinodynamic planning. In *53rd IEEE Conference on Decision and Control*, pages 3737–3744, 2014. doi:10.1109/CDC.2014.7039971.

[Smi04] Russ Smith. Open dynamics engine (ode), 2004. URL: https://www.ode.org/.

[Sof24] Big Ladder Software. Btwxt. https://github.com/bigladder/btwxt, 2024.

[TH22] Jesus Tordesillas and Jonathan P. How. Panther: Perception-aware trajectory planner in dynamic environments. *IEEE Access*, 10:22662–22677, 2022. doi:10.1109/ACCESS.2022.3154037.

[TUI17] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):16, Jun 2017. doi:10.1186/s41074-017-0027-2.

[Wol11] Thomas Wolf. Deutsche orbitale servicing mission. *Astra*, 2011.

[YLM10] Anna Yershova, Steven M. LaValle, and Julie C. Mitchell. *Generating Uniform Incremental Grids on SO(3) Using the Hopf Fibration*, pages 385–399. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. doi:10.1007/978-3-642-00312-7_24.

[ZS18] Zichao Zhang and Davide Scaramuzza. Perception-aware receding horizon navigation for mavs. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2534–2541, 2018. doi:10.1109/ICRA.2018.8461133.