Università degli Studi di Padova

Department of Information Engineering

Ph.D. Course in Information Engineering

Mobile Robotics in Unknown Environments: Towards Full Autonomy

Nicola Lissandrini

Advisor	prof. Angelo Cenedese
Director and Coordinator	prof. Andrea Neviani

February 28, 2023

Nicola Lissandrini
Mobile Robotics in Unknown Environments:
Towards Full Autonomy
Ph.D. Thesis,
Advisor: prof. Angelo Cenedese

Università degli Studi di Padova
Ph.D. School in Information Engineering
Department of Information engineering
Via Giovanni Gradenigo, 6, Padua, Italy
35100

Abstract

Mobile robotics has become increasingly popular in recent years as it provides an automated and cost-efficient solution to a variety of tasks. Traditionally, human operators would have full responsibility on the robot actions with teleoperation. Recent advances in sensors and algorithms have paved the way for robots to be able to operate autonomously or with little human intervention. Autonomous operation in known and structured environments has been vastly studied over the last decades, but such scenarios are limited to specific and laboratory applications. Real world contexts are characterized by unknown and unstructured scenarios that the robot must sense and adapt while performing the prescribed task. Recently, great effort has been given to the development of strategies to face these challenges. However, the pursuit of full autonomy is hindered by the limited hardware capacity of mobile robots, that constrain the computational capacity available to realize the desired operation. In this thesis we present several strategies to cope with uncertainties and unknown environment for both ground and aerial robot mobility, with a particular focus on the efficiency and the compliance with real time constraints. First, we consider a task of robust robot coordination for object transportation. Then, a novel approach for reactive navigation in unknown environments is presented, with theoretical proofs and experimental validation. Additionally, we present a motion estimation algorithm for unknown environments with the purpose of aerial physical interaction. Overall, particular attention is given to the efficient implementation of the proposed methodologies, which is a key factor for achieving full autonomy.

Contents

1 Introduction

The future is already here – it's just not evenly distributed

— **William Gibson**

Autonomous mobile robotics is emerging as a core technology in several application fields (Rubio et al., 2019), ranging from industrial (D'Andrea, 2012; Ball et al., 2017) to the civil (e.g. service robotics (Paden et al., 2016)), and emergency and military contexts (e.g. field robotics (Lippi and Marino, 2018; Lissandrini et al., 2019)). Recent advances in the development of rich-data acquisition sensors (Raj et al., 2020; Lluvia et al., 2021) and high-performance embedded microprocessors (Tu et al., 2019) endorse the ability of robots to efficiently perceive and interpret the surrounding environment, and thus quickly react and adapt to unknown and dynamic scenarios. Still, mobile robotics faces critical challenges: a full autonomous robot needs to balance reactiveness and accuracy under the constraint of minimal computational capabilities. The main cause of such limitation is twofold: on one hand, mobile devices applications are hindered by the limited battery capacity that forces the robot to be both mechanical and electrically efficient; on the other hand, full autonomy requires that all or most of the information upon which an agent can rely comes from on-board sensors, which can entail the processing of a large set of data. One possible approach to deal with this complexity is to resort to a multi-agent scheme (Ismail et al., 2018): in this way the problem can be simplified to set of smaller problems that can also be solved in parallel by two different computing machines (López-González et al., 2020). However, this is not always possible, and it strongly depends on the specific context. A more general approach is to employ algorithms that are as efficient as possible, relying on little computational capabilities and process the data not more than strictly needed. Efficiency and timely-contained computation, indeed, are crucial in practical applications. While it could be acceptable that the total energy consumption results in a shorter battery duration, real-time constraints represent the most critical issues (Koulamas and Lazarescu, 2018) as they are needed for the motion task itself: the planning and control computation is required in the exact moment that the system dynamics demand. What makes mobile robotics challenging for real world applications is that these basics and physics-related constraints must be entangled with application-oriented requirements:

- **Mobility and energy consumption**
  Other than to mere mechanical design and electrical efficiency, these aspects are heavily linked to the choice of sensors and actuators. More limited sensor capabilities need a more advanced algorithmic efficacy (Stefek et al., 2020), and more lightweight actuators require a more complex planning on the robotics task. At the same time, lightweight actuators can allow robots to move with greater agility and speed, thus reducing the overall energy consumption.

- **Safety in interaction**
  Real world applications are characterized by a certain level of uncertainties that may arise from internal or external entities. In the former case it could be another collaborating robot, that would require robust synchronization (Zhang et al., 2006); the latter could be represented by the environment which in many cases is totally or in part unknown. To this it is often added an amount of clutteredness and unpredictability: to cope with such environments the planning strategy must guarantee a high level of safety in order to avoid collisions and guarantee the accomplishment of the prescribed task (Xia et al., 2020).

- **Accuracy**
  Strongly related to safety, but also to the ability of performing delicate tasks, the estimation and planning tasks need to be able to provide precise results in terms of distance from the ground truth and the activity requirements, namely the distance from a target location, as well as safety and repeatability requirements (Wen et al., 2021).

- **Reliable autonomy**
  The robot should be able to complete the desired activity with less intervention of the human as possible, even in presence of unexpected events. In particular, this entails that the robotic task can only be programmed from a high, task-organization level, and, in particular, planning and navigation must be computed in full autonomy (Wahrmann et al., 2019), only relying upon sensors and on-board algorithms. Reactive-

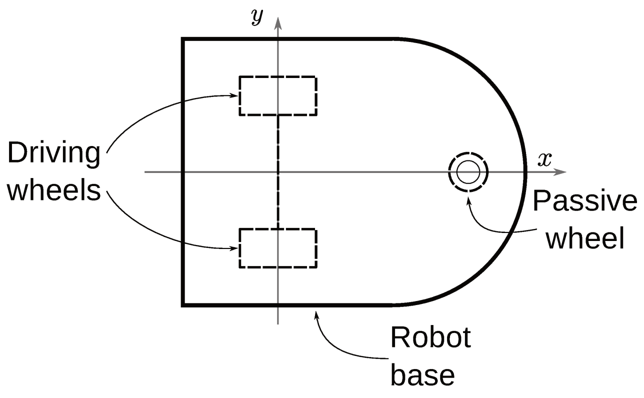


Figure 1.1.: Schematic example of differential steering robot

ness (Chatila, 1995) is also crucial to cope with dynamic environments and scenarios that are unpredictably changing, but it should not compromise the performances in terms of accuracy.

This work tries to develop the fundamental steps towards an integrated approach that accounts for all these constraints at once, rather than focusing on a single one. In particular, major importance is given to the real time capabilities of the proposed algorithm. This comes from a careful development of performant strategies to solve robotics tasks, but also from a particular attention to an efficient and modular software implementation.

These general aspects and challenges are shared by both ground and aerial vehicles. The employment of ground mobile robots is well established in literature (Eskandarian et al., 2019). Locomotion is the core characteristic, and it is generally achieved via wheeled systems. Due to its simplicity, one of the most common methods is the differential steering, represented in figure 1.1, where the two wheels are controlled independently to obtain the desired linear and angular velocity for the robot. While versatile for many applications, these systems are underactuated and this can be a limiting factor in some specific context where complex trajectory are required to be followed with maximum precision. In these cases it could be preferred to adopt omnidirectional wheels (e.g. those in figure 1.2) that allow the robot to instantly move in any direction. On the other hand, these wheels tend to be sensitive to ground imperfections, so the proper locomotion system should be carefully evaluated for each specific application. The navigation system is often abstracted from the physical robot structure and is limited to calculating a trajectory, and then a distinct, lower-level controller is used to guide the robot along such trajectory, taking into account its physical constraints. The

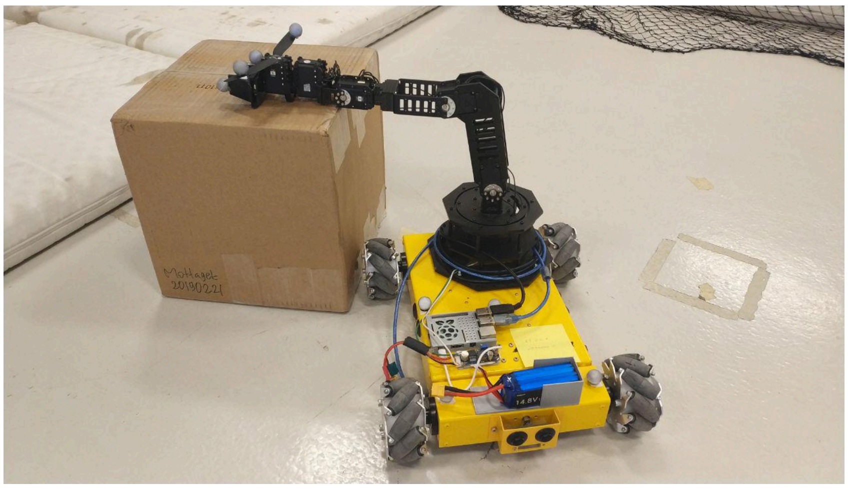


Figure 1.2.: Example of mobile robot composed of an omnidirectional-wheeled base mounting a 6 d.o.f. manipulator.

second important aspect of mobile robots is the perception of the robot relative to its surroundings. While motion capture systems can be employed to obtain a ground truth of the robot's position with submillimetric precision (Menolotto et al., 2020), it still suffers from the occlusion problem relying on such technology greatly limits the application of the developed strategies and, most importantly, limits the applications to confined laboratory arenas. The converse approach is to equip the robot with on board sensors such as cameras or LiDAR (Light Detection and Ranging) (Cheng and Wang, 2018) and design the navigation system to leverage their measurements in order to achieve the prescribed goal. To achieve complex and active tasks such as manipulation and transportation (Sandakalum and Ang Jr, 2022), it is common to equip the mobile vehicle, that serves as a base of the robot with, a manipulator with one or more degrees of freedom (e.g. in figure 1.2). Ground mobile arms can then be leveraged to exert possibly high forces to the target object in order to transport or manipulate the payload. In turn the reachable workspace is limited by the size of the manipulator and the areas that are actually accessible by the ground vehicle, which may be constrained by physical obstructions such as rough terrain, stairs and obstacles. Legged locomotion can address some of these limitations (Bruzzone et al., 2022), however their implementation is often very costly and hardware-demanding. Also, it does not approach the problem of the limited workspace as concerns reaching higher altitude, if needed.

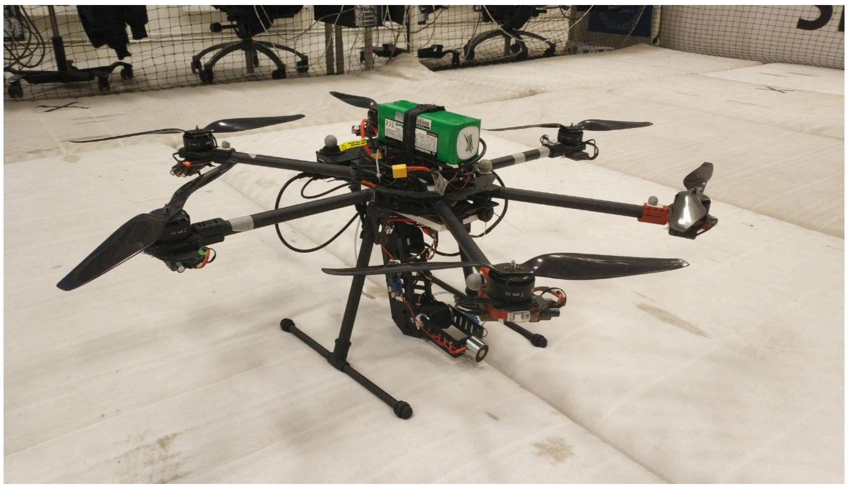


Figure 1.3.: Example of an aerial robot composed of an hexarotor with a 2 d.o.f. arm.

An alternative approach is to resort to aerial robots: Unmanned Aerial Vehicles (UAVs) and Micro Aerial Vehicles (MAVs) have received an enormous amount of interest both from a research and an industrial point of view. In the past decades, most of the attention has been given to their role as passive observers (Shakhatreh et al., 2019), for example data sensing (Yang and Yoo, 2018), surveillance, environmental monitoring (Alsamhi et al., 2019; Hernández-Vega et al., 2018), smart cities (Ahad et al., 2020), and so on.

To really unlock the potential of a flying robot, however, the latest trend is to endow them with the ability of physically interact with the environment (Ollero et al., 2021) to perform manipulation and a various set of other tasks. The possibility to actively operate in high altitude workspace and easily overcome cluttered environments, attracted a great interest from areas like inspection and maintenance (Ollero et al., 2018). To enable this capability, the aerial vehicles have been endowed with means of manipulation. Essentially, it is possible to distinguish two directions: in the first one, emulating ground mobile robots, the drones are equipped with a several degrees of freedom manipulator, as in figure 1.3 (Lissandrini et al.; Tognon et al., 2019; Suarez et al., 2020). This results in high manipulation capabilities, but it comes at the cost of a high payload overhead and consequently requiring for larger vehicle structure and limiting the battery life, which is one of the most critical issues in aerial robotics. An alternative direction is to mount a fixed end-effector (Ryll et al., 2019; Tzoumanikas et al., 2020) on a tilt-rotor aerial vehicle (Ryll et al., 2021; Kamel et al., 2018; Rashad et al., 2020) that is able to steer and hover the vehicle in any direction (Kamel et al., 2018; Shawky et al., 2021), differently from a classical quadrotor that can only hover in horizontal position.

To allow aerial interaction task, classical quadrotors are often not powerful enough for both strategies, for different reasons, and solutions with higher number of propellers are adopted (hexarotor (Ollero et al., 2018), octarotor(Brescianini and D'Andrea, 2018)). In the first case, the dexterity of the interaction relies on the degrees-of-freedom of the manipulator, which in turn results to be heavy: in this case more propeller are needed to guarantee sufficient thrust and robustness. In the second case, the vehicle can be more lightweight but, in turn, a higher number of actuators arranged according to a proper geometry are needed to fulfill the theoretical conditions to achieve the required dexterity (Michieletto et al., 2018).

In general, the ingredients for aerial robotics are:

- **Interaction control**: for interacting robots, to the usual position control requirement is coupled by the interaction force control, in order to preserve stability of the entire system. The most common approach is to establish a desired dynamical property between position and force, through the so-called impedance (Lippiello et al., 2018; Rashad et al., 2021) admittance (Ryll et al., 2019; Yüksel et al., 2019; Gabellieri et al., 2020) control, but it is also possible to implement hybrid force-position controllers (Praveen et al., 2020).

- **Wrench estimation**: in order to perform control on the interaction force/torque, both quantities have to be estimated. While this can be directly obtained from a sensor (Cataldi et al., 2016), more recently it is preferred avoid the payload and cost of this additional sensor and estimate the wrench (i.e. the vector of torques and forces) from the position, velocity and inertial measurements (Shi et al., 2019; Ryll et al., 2019; Rajappa, 2018; Wilmsen et al., 2019).

- **Localization**: localization is needed not only to provide the relative position to the target point of contact with respect to the robot, but it is needed in the wrench estimation algorithms. Although the vehicle position can be computed from external cameras, such as motion capture systems (Jepsen et al., 2021), this limits the applicability to ad-hoc laboratory setups. Without absolute localization (Santamaria-Navarro et al., 2018), *visual odometry* exploit stereo- or depth-cameras (Kerl et al., 2013) to obtain a local motion estimate, which is well suited for aerial manipulation since only relative measurements are sufficient to obtain the interaction wrench estimation. Ultra-wide band can also be used to measure the distance to beacon and then triangulate the position of the robot (Güler et al., 2020). A very common technique is SLAM (Simultaneous Localization and Mapping), based on 3D LIDAR sensors (Milijas et al., 2021), which provides for position and orientation estimate. Recently, fusion techniques have been tested by combining SLAM with the other sensor information, as in (Paneque et al., 2019).

The thesis is structured as follows: first, we will provide a baseline implementation of collaborative robotic task where the uncertainty lies in the physical interaction between the agents that is robustly handled through a decentralized strategy based on a nonlinear model predictive controller. In this case the position of the robots and obstacles are estimated by a motion capture system available in a laboratory arena. We then try to overcome this limitation by introducing a novel planar navigation technique (the NAPVIG algorithm) that is able to navigate through simple environments that, however, are fully unknown, narrow and cluttered, through a fast and reactive trajectory planner, validated both in simulation and laboratory experiments. In the following chapter are presented mathematical results proving that the algorithm is able to fulfill precise theoretical properties that are desirable for the purposes of navigation in unknown and unpredictable environments. The next chapter presents an extension of the NAPVIG algorithm to cope with local navigation in any kind of cluttered planar environment. Preliminary results regarding self localization for purposes of physical interaction of aerial vehicles with unknown environment, with a view of future works involving aerial-ground collaboration in absence of external sensors. Finally, given the technological importance of meeting real-time constraints, we will present the frameworks developed to realize the implementations of the algorithms involved in this thesis

2 Nonlinear MPC for cooperative manipulation

2.1 Introduction

Multi-robot systems is a trending and pervasive topic in academic and industrial research, due to the strong potential impact that affects many application fields (Chung et al., 2018; Wilson et al., 2020). For instance, cooperative transportation or manipulation of large or heavy objects (Alonso-Mora et al., 2017; Lee et al., 2018), inspection and servicing of infrastructures (Suarez et al., 2016), monitoring and mapping of the environment (Cortés, 2010; Alonso-Mora et al., 2017), search and rescue operations (León et al., 2016), are just some real world applications that can benefit more by these studies and the related technological developments.

Strong results have been demonstrated on the control of single and multiple robotic systems (Mahony et al., 2012; Franchi et al., 2012) and, more recently, a lot of effort has been made to allow physical interaction among these systems and with the environment (Gawel et al., 2017). Robustness in the estimation and regulation actions for non-ideal actual scenarios and in presence of environment/agent constraints has been also considered (Zhao et al., 2017; Corah and Michael, 2017).

Heterogeneous robots with different capabilities (e.g., sensing, actuation) are an important aspect of multi-robot systems, since they offer greater flexibility and versatility in complex scenarios (Wurm et al., 2013). In this chapter we consider the problem of cooperative object transportation via aerial-ground manipulator-endowed robots, which can be beneficial in cases where the different sensing and operating workspaces of the two robotic types might be needed.

Regarding the related literature, (Nguyen and Garone, 2016) considers the cooperative object transportation by aerial-ground mobile robots, limited to a

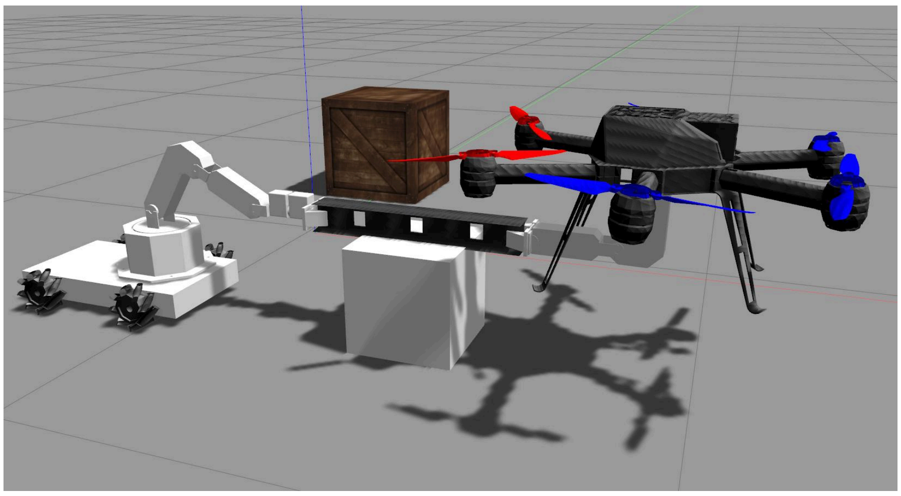


Figure 2.1.: Two heterogeneous robots transporting an object (in the Gazebo simulation environment).

coplanar case. Collaborative task control with heterogeneous robots has been also studied in (Naldi et al., 2012), where a team of ground robots is used to stabilize the aerial vehicle, and in (Kondak et al., 2014) where the interaction between a multi-rotor and an industrial manipulator is considered. Further tests for aerial ground manipulation tasks have been made in (Staub et al., 2017), where the ground vehicle is tasked to deploy the object to a position and the UAV adjusts its attitude to adapt to it.

In the context of this research, the Model Predictive Control approach in its non-linear form (NMPC) appears to be the suitable and effective framework to tackle this study, since it can be formulated as a constrained optimization problem subject to the system dynamics (Findeisen et al., 2003) and that accounts also for model uncertainties (e.g. (Nascimento et al., 2013; Nikou et al., 2017)). In (Nikou et al., 2017), in particular, the problem of cooperative manipulation is solved in a non-scalable centralized way by deriving a coupled model of the agents involved in the task, with a decentralized extension being developed in (Verginis et al., 2018); These solutions, however, do not explicitly consider heterogeneous robots and rely on the strong assumption of rigid grasping robot-object points.

This work extends the aforementioned works by proposing a multi-robot algorithm for the cooperative object transportation with collision avoidance by heterogeneous robots deriving a novel approach to address the problem, which increases robustness to non-idealities and allows the definition of a more general framework compared to the cited literature, as well as relevant experimental results. The developed scheme is decentralized, since each robot computes its own MPC control signals via a leader-follower coordination, inspired by (Verginis et al., 2018). Intuitively, a leader robot generates the desired trajectory of the grasped object, and the rest of the robots comply via an internal force minimization problem, without assuming rigid grasping points. The proposed algorithm is implemented and tested on a system of a ground and aerial robot (see Fig. 2.1 for an illustrative example), both on the robotic simulator Gazebo (Koenig and Howard, 2004) and in real laboratory experiments.

2.2 Preliminary notation and models

We consider a generic setup where N robotic agents are grasping an object. The robots are composed of a moving base and a robotic manipulator, which can have an arbitrary number of degrees of freedom. The base can be either a ground vehicle, e.g. fully actuated with holonomic wheels and 3 d.o.f., or an unmanned aerial vehicle (UAV). Figure 2.2 depicts an example of ground mobile robot composed of a base and a generic manipulator. The aim of the agents is to transport the object along a collision-free reference trajectory in a decentralized manner avoiding internal forces exerted by the manipulators.

In the remainder of this chapter, letters i, j, h, k, ℓ, are used as indexes; scalar parameters and variables are denoted by nonbold lowercase letters, while vector and matrix quantities are denoted, respectively, by bold lowercase

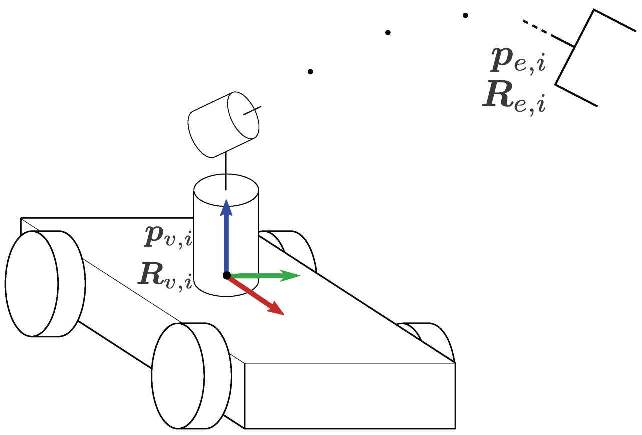


Figure 2.2.: Generic agent model

and bold uppercase symbols. Rotation matrices are defined in the Special Orthogonal group $\mathbb{SO}(3)$, while $[\cdot]_\times$ indicates the skew symmetric matrix associated to the argument vector. Given two frames $\{a\}$, $\{b\}$, as well as a world frame $\{W\}$, we denote by $\boldsymbol{T}_b^a$ and $\boldsymbol{T}_b$ the affine transformation from frame $\{a\}$ to frame $\{b\}$ and from $\{W\}$ to $\{b\}$, respectively. Similarly, $\boldsymbol{p}_a \in \mathbb{R}^3$ and $\boldsymbol{R}_a \in \mathbb{SO}(3)$ are the position and rotation matrix, respectively, of frame $\{a\}$ with respect to $\{W\}$. Frames $\{v, i\}$ and $\{e, i\}$ are the vehicle and end-effector frame of agent i.

Let $\boldsymbol{q}_i \in \mathbb{R}^{n_i}$ be the vector of joint variables describing the configuration of each manipulator, with n_i being the corresponding number of degrees of freedom. The first-order kinematics of the agents can be written as follows:

$$\text{Agent } i: \begin{cases} \dot{\boldsymbol{p}}_{e,i} &= \boldsymbol{A}_{p,i}\boldsymbol{u}_{v,i} + \boldsymbol{J}_{P,i}(\boldsymbol{q}_i)\boldsymbol{u}_{q,i} \\ \boldsymbol{\omega}_{e,i} &= \boldsymbol{A}_{\omega,i}\boldsymbol{u}_{\omega,i} + \boldsymbol{J}_{O,i}(\boldsymbol{q}_i)\boldsymbol{u}_{q,i} \\ \dot{\boldsymbol{R}}_{e,i} &= \left[\boldsymbol{\omega}_{e,i}\right]_\times \boldsymbol{R}_{e,i} \\ \dot{\boldsymbol{q}}_i &= \boldsymbol{u}_{q,i} \\ \dot{\boldsymbol{p}}_{v,i} &= \boldsymbol{A}_p\boldsymbol{u}_{v,i} \\ \dot{\boldsymbol{R}}_{v,i} &= \left[\boldsymbol{A}_\omega\boldsymbol{u}_{\omega,i}\right]_\times \boldsymbol{R}_{v,i} \end{cases} \tag{2.1}$$

where:

- $\boldsymbol{p}_{v,i},\ \boldsymbol{p}_{e,i}(\boldsymbol{p}_{v,i}, \boldsymbol{q}_i) \in \mathbb{R}^3$ are the position of the base and the end-effector of the i-th agent, respectively.

- $\boldsymbol{R}_{v,i},\ \boldsymbol{R}_{e_i}(\boldsymbol{R}_{v,i}, \boldsymbol{q}_i) \in \mathbb{SO}(3)$ refer to the corresponding rotation matrices relative to a fixed world frame $\{W\}$.

- $\boldsymbol{u}_{v,i},\ \boldsymbol{u}_{\omega,i} \in \mathbb{R}^3$ are the linear and angular input velocities applied at the base expressed in the local frame.

- $\boldsymbol{u}_{q,i}$ are the joint velocities, which are assumed to be the manipulator's control input.

- $\boldsymbol{A}_{p,i},\ \boldsymbol{A}_{\omega,i} \in \mathbb{R}^{3\times3}$ allow modeling constraints on the input velocity (e.g. reference frame transform or nonholonomic constraints)

- $\boldsymbol{\omega}_{e,i}$ is the angular velocity of the end effector in the world frame.

- $\boldsymbol{J}_{P,i}(\boldsymbol{p}_{v,i}, \boldsymbol{q}_i),\ \boldsymbol{J}_{O,i}(\boldsymbol{R}_{v,i}, \boldsymbol{q}_i) \in \mathbb{R}^{3\times n_i}$ are, respectively, the position and orientation Jacobian matrices, which depend on the structure of the manipulator.

The full state and input are defined as: $\boldsymbol{x}_i = [\boldsymbol{p}_{e,i}\ \boldsymbol{r}_{e,i}\ \boldsymbol{p}_{v,i}\ \boldsymbol{q}_i]^\top \in \mathbb{R}^{n_i}$, $\boldsymbol{u}_i = [\boldsymbol{u}_{q,i}\ \boldsymbol{u}_{v,i}\ \boldsymbol{u}_{\omega,i}]^\top \in \mathbb{R}^{p_i}$ where the lower case $\boldsymbol{r}_{e,i}$ refers to a vector representation of the rotation matrix $\boldsymbol{R}_{e,i}$, s.t. $[\boldsymbol{R}_{e,i}]_{(h,k)} = [\boldsymbol{r}_{e,i}]_{(3k+h)}$. We can then rewrite (2.1) with in compact form as $\dot{\boldsymbol{x}}_i = f_{\mathcal{A}_i}(\boldsymbol{x}_i)\boldsymbol{u}_i$, where $f_{\mathcal{A}_i}(\boldsymbol{x}_i)$ collects all the control-affine terms and can be easily inferred by (2.1).

2.3 Cooperative manipulation with MPC

In this section, we propose a decentralized algorithm for cooperative manipulation with obstacle collision avoidance. The objective is formalized as follows. Let $\{o\}$ be a frame attached to the object's. We consider the reference trajectory that the object is desired to follow, $\boldsymbol{T}_{o,ref}(t) \in \mathbb{SE}(3),\ t > 0$. The goal is to find a control law for each agent such that the object is transported along the trajectory, while ensuring collision avoidance. To avoid the object being detached from the grasps or breaking, we also aim at minimizing the internal forces and torques applied to the object by the agents.

We assume that, at time $t = 0$, the agents are still and already grasping the object, defining an initial condition for the relative transforms from object frame to each end effector's $\boldsymbol{T}^o_{e,i}(0)$, that, for $t = 0$ only, we assume that they can measure. Also, we assume that they can communicate.

The robotic agents are heterogeneous and, especially in the case of aerial vehicles, they can be characterized by a low number of degrees of freedom. This can lead to situations where a perfect compliance of the grasps is impossible, e.g., an underactuated UAV that needs to roll-pitch to generate horizontal forces and no possibility to compensate.

To allow the algorithm to be robust to such non-idealities, we first need to theoretically allow deviations from rigidity. To this aim, we consider the object gripper joints as elastic, as depicted in Fig. 2.3, with the rest condition being defined by $\boldsymbol{T}^o_{e,i}(0),\ i = 1, \ldots, N$. This can either model a case where gripper joints are actually elastic, or a case where the grasps are rigid, and

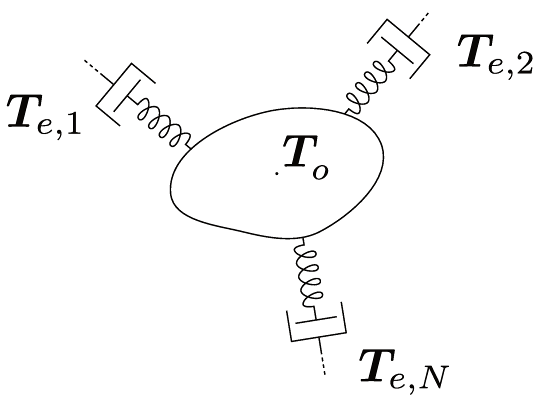


Figure 2.3.: Elastic joints object to model non-ideal rigidity

the object is elastic itself. We denote by $\boldsymbol{f}_i$, $\boldsymbol{\tau}_i$, the forces and torques applied from agent i to the respective grasping point. In view of the elastic joint model, a part of these forces/torques will result in actual object motion while the rest will be absorbed by the imaginary elastic spring. We denote the latter by $\boldsymbol{f}_{i,o}$, $\boldsymbol{\tau}_{i,o}$, which, since the initial condition at $t = 0$ is the rest condition, satisfy

$$\left\| \begin{bmatrix} \boldsymbol{f}_{i,o}(t) \\ \boldsymbol{\tau}_{i,o}(t) \end{bmatrix} \right\| = \kappa_i \operatorname{dist}\left(\boldsymbol{T}^o_{e,i}(t), \boldsymbol{T}^o_{e,i}(0)\right) \tag{2.2}$$

where $\operatorname{dist}(\boldsymbol{T}_a, \boldsymbol{T}_b) := \|\boldsymbol{p}_a - \boldsymbol{p}_b\| + \beta \|\boldsymbol{R}_a^\top \boldsymbol{R}_b - \boldsymbol{I}\|_F$, $\beta \in \mathbb{R}$, $\kappa_i > 0$ is a positive constant depending on the initial condition and physical properties of the object and the grippers, and $\|\cdot\|_F$ is the Frobenius norm. An example depicting such forces is reported in figure 2.4.

The total forces/torques $\boldsymbol{f}_o$ and $\boldsymbol{\tau}$ applied to the object are the sum of the contributions of the agents. We define as internal forces all the components that cancel out in the sum but create tension/compression stresses on the object. Given all the possible forces/torques applied at the object that produce the same acceleration for the object center of mass, we aim at outputing those with minimal internal forces that also minimize the total norm $\sum_{i=1}^N \|[\boldsymbol{f}_{i,o}(t) \;\; \boldsymbol{\tau}_{i,o}(t)]\|^2$, since internal forces do not contribute in the net force but increase the norm sum. Assumption (2.2) then allows to translate this problem into the minimization of the corresponding displacement $\boldsymbol{T}^o_{e,i}(t)$.

In particular, we address the prescribed objectives by resorting to a nonlinear MPC formulation: at each time t, given the state measure $\boldsymbol{x}_i(t)$, solve the

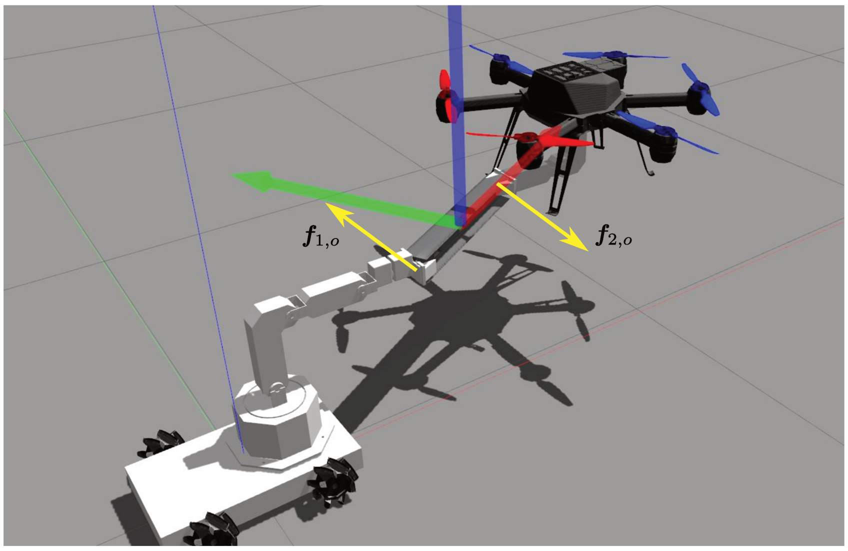


Figure 2.4.: Example of internal forces applied on the object by the agents

following Finite Horizon Optimal Control Problem (FHOCP) (Findeisen et al., 2003):

$$
\begin{aligned}
\underset{\hat{\boldsymbol{u}}_1(\cdot),\dots,\hat{\boldsymbol{u}}_N(\cdot)}{\operatorname{argmin}} \int_t^{t+T} & \Bigg\{ \operatorname{dist}(\hat{\boldsymbol{T}}_o(\tau), \hat{\boldsymbol{T}}_{o,ref}(\tau)) \\
& + \sum_{i=1}^{N} \operatorname{dist}(\hat{\boldsymbol{T}}^o_{e,i}(\tau), \hat{\boldsymbol{T}}^o_{e,i}(0)) \\
& + \sum_{i=1}^{N} \hat{\boldsymbol{u}}_i(\tau)^\top \boldsymbol{W}_u \hat{\boldsymbol{u}}_i(\tau) \Bigg\} \mathrm{d}\tau \qquad (2.3) \\
\text{subject to:} \quad & \dot{\hat{\boldsymbol{x}}}_i = f_{\mathcal{A}_i}(\hat{\boldsymbol{x}}_i)\hat{\boldsymbol{u}}_i, \qquad i = 1, \dots, N \\
& \hat{\boldsymbol{x}}_i(t) = \boldsymbol{x}_i(t) \\
& \hat{\boldsymbol{x}}_i(\tau) \in \mathcal{X}_i, \;\; \hat{\boldsymbol{u}}_i(\tau) \in \mathcal{U}_i \;\; \tau \in [t, t+T]
\end{aligned}
$$

where the hat terms $\hat{\cdot}$ denote the predicted variables, over a horizon T. The first integral term is the trajectory error, the second accounts for the minimization of internal forces, $\boldsymbol{W}_u$ is a matrix that weighs a penalty on the control effort, providing stability (Findeisen et al., 2003), T is the finite time window (MPC horizon) and $\mathcal{X}_i, \mathcal{U}_i$ are the admissible sets for state and input values for each agent, that can account for singularity-avoidance and actuation limits.

The problem can be approached in a decentralized way by resorting to a leader-follower architecture, as follows.

2.3.1 Leader and follower coordination

At the design stage, one agent is designed to be the leader. This choice has no theoretical limitations, and it is driven by experimental evaluations. The leader computes the trajectory for its end effector such that the object tracks the prescribed trajectory, accounting for the object trajectory error as if the follower agents could not alter its behavior. In other words, the forces applied to the object along the desired trajectory are produced by the leader and the action by the followers is then obtained to minimize the internal forces.

Note that: $\boldsymbol{T}_o(t) = \boldsymbol{T}_{e,\ell}(t)\boldsymbol{T}_o^{e,\ell}(t)$, with $\ell \in \{1, \dots, N\}$ being the leader index. $\boldsymbol{T}_{e,\ell}(t)$ is a quantity that can be controlled, and it is accounted by (2.1); $\boldsymbol{T}_o^{e,\ell}(t)$, on the other hand, is not controllable since it is a direct result of the forces applied at the object center of mass , due to the elasticity assumption, that are not included in the model. However, if the overall dynamics is sufficiently slow and the elasticity is sufficiently low, it is reasonable to assume that, in absence of other forces and torques, the displacement is bounded over time, i.e.,

$$\operatorname{dist}(\boldsymbol{T}^o_{e,\ell}(t), \boldsymbol{T}^o_{e,\ell}(0)) < \varepsilon_\ell \tag{2.4}$$

for a positive constant ε_ℓ. Without the ability to make predictions on $\boldsymbol{T}^o_{e,\ell}$, which is needed to estimate the object position given the end effector prediction, we will choose as estimate $\hat{\boldsymbol{T}}^o_{e,\ell}(t) = \boldsymbol{T}^o_{e,\ell}(0)$, $\forall t \in [0, T]$, satisfying (2.4). Due to this assumption, agents are not needed to measure the object frame for $t > 0$. Then, the leader aims at minimizing the cost function:

$$J_\ell(\hat{\boldsymbol{x}}_\ell(\cdot), \boldsymbol{u}_\ell(\cdot) = \int_t^{t+T} \operatorname{dist}(\hat{\boldsymbol{T}}_{e,\ell}(\tau), \hat{\boldsymbol{T}}_{e,\ell,ref}(t))\,\mathrm{d}\tau \tag{2.5}$$

through the following FHOCP:

$$\begin{aligned}
\underset{\boldsymbol{u}_\ell(\cdot)}{\operatorname{argmin}} \quad & J_\ell(\boldsymbol{x}_\ell(\cdot), \boldsymbol{u}_\ell(\cdot)) + \int_t^{t+T} \boldsymbol{u}_\ell^\top \boldsymbol{W}_u \boldsymbol{u}_\ell \,\mathrm{d}\tau \\
\text{subject to:} \quad & \dot{\hat{\boldsymbol{x}}}_\ell = f_{\mathcal{A}_\ell}(\hat{\boldsymbol{x}}_\ell)\boldsymbol{u}_\ell \\
& \hat{\boldsymbol{x}}_\ell(t) = \boldsymbol{x}_\ell(t) \\
& \hat{\boldsymbol{x}}_\ell(\tau) \in \mathcal{X}_\ell,\ \boldsymbol{u}_\ell(\tau) \in \mathcal{U}_\ell,\ \tau \in [t, t+T]
\end{aligned} \tag{2.6}$$

The solution to (2.6) $\boldsymbol{u}_\ell^*(\tau)$, for $\tau \in [t, t+T]$, defines a predicted state trajectory $\boldsymbol{x}_\ell^*(\tau)$ that is optimal with respect to the reference trajectory for the object. In particular, from the first 12 components of the state vector the predicted trajectory for the end effector pose can be extracted, which we will refer to as $\boldsymbol{T}_{e,\ell}^*(\cdot)$.

Conversely to the leader, the followers have to ensure that the trajectory planned by the leader is attained by adapting their system states and output forces/torques. The role of the followers is to minimize the internal forces, which is accomplished, due to (2.2), by minimizing the second term in (2.3). For $j \neq \ell$, define $\hat{\boldsymbol{T}}_{e,j}^o = \boldsymbol{T}_{e,j}^o(0)$, and then note that, by left-multiplication with $\boldsymbol{T}_o(t)$ and using (2.4), we obtain:

$$
\begin{aligned}
\operatorname{dist}(\boldsymbol{T}_{e,j}^o(t), \boldsymbol{T}_{e,j}^o(0)) &= \operatorname{dist}(\boldsymbol{T}_o(t)\boldsymbol{T}_{e,j}^o(t), \boldsymbol{T}_o(t)\hat{\boldsymbol{T}}_{e,j}^o) \\
&= \operatorname{dist}(\boldsymbol{T}_{e,j}(t), \boldsymbol{T}_{e,\ell}(t)(\hat{\boldsymbol{T}}_{e,\ell}^o)^{-1}\hat{\boldsymbol{T}}_{e,j}^o) + \varepsilon_j
\end{aligned}
\tag{2.7}
$$

where ε_j is an error due to the approximations of (2.4). This means that $\boldsymbol{T}_{e,j}^o$, for the followers, is controllable up to ε_j. In this way we explicitly express the displacement of the grasps from the rest condition in terms of controllable quantities. In view of (2.7) and given $\boldsymbol{T}_{e,\ell}^*(\cdot)$, which is the leader trajectory that minimizes (2.6), each follower agent j aims at minimizing the following cost function:

$$
J_j(\hat{\boldsymbol{x}}_j(\cdot), \boldsymbol{u}_j(\cdot)) = \int_t^{t+T} \operatorname{dist}(\hat{\boldsymbol{T}}_{e,j}(\tau), \hat{\boldsymbol{T}}_{e,j,ref}(\tau))\, \mathrm{d}\tau \tag{2.8}
$$

where $\boldsymbol{T}_{e,j,ref}(t) = \boldsymbol{T}_{e,\ell}^*(t)(\hat{\boldsymbol{T}}_{e,\ell}^o)^{-1}\hat{\boldsymbol{T}}_{e,j}^o$ is a transformation of the trajectory predicted from the leader. This is achieved, by iteratively solving the following FHOCP problem:

$$
\begin{aligned}
\underset{\boldsymbol{u}_j(\cdot)}{\operatorname{argmin}} \quad & J_j(\hat{\boldsymbol{x}}_j(\cdot), \hat{\boldsymbol{u}}_j(\cdot)) + \int_t^{t+T} \hat{\boldsymbol{u}}_j^\top \boldsymbol{W}_u \hat{\boldsymbol{u}}_j \, \mathrm{d}\tau \\
\text{subject to:} \quad & \dot{\hat{\boldsymbol{x}}}_j = f_{\mathcal{A}_j}(\hat{\boldsymbol{x}}_\ell)\hat{\boldsymbol{u}}_j \\
& \hat{\boldsymbol{x}}_j(t) = \boldsymbol{x}_j(t) \\
& \hat{\boldsymbol{x}}_j(\tau) \in \mathcal{X}_j, \;\; \hat{\boldsymbol{u}}_j(\tau) \in \mathcal{U}_j, \;\; \tau \in [t, t+T]
\end{aligned}
\tag{2.9}
$$

The decentralized cooperative algorithm can be then summarized as in Alg. 1.

Algorithm 1 Decentralized Cooperative Algorithm

- At time $t = 0$, every agent i measures and stores the pose of the object with respect to their end effector $\hat{\boldsymbol{T}}^o_{e,i} = \boldsymbol{T}^o_{e,i}(0)$. The leader also communicates $\hat{\boldsymbol{T}}^o_{e,\ell}$ to every follower.
- At each sample time t_k:
  1. Given a desired trajectory for object frame $\boldsymbol{T}_{o,ref}$, leader agent ℓ solves (2.6), obtaining predicted optimizing $\boldsymbol{u}^*_\ell(\cdot)$ and $\boldsymbol{T}^*_{e,\ell}(\cdot)$
  2. Leader communicates $\boldsymbol{T}^*_{e,\ell}$ to every follower
  3. Each follower $j \neq \ell$, receives $\boldsymbol{T}^*_{e,\ell}$, computes $\boldsymbol{T}_{e,j,ref}$ as in (2.8), exploiting the measured $\hat{\boldsymbol{T}}^o_{e,j}$ and the received $\hat{\boldsymbol{T}}^o_{e,j}$, and then solves (2.9), obtaining $\boldsymbol{u}^*_j(\cdot)$ and $\boldsymbol{x}^*_j(\cdot)$.
  4. Agents synchronously apply the control to the lower level controllers, that, in general, may be a function of the predicted trajectories and input: $\bar{\boldsymbol{u}}_i(t) = f_{u,i}\left(\boldsymbol{u}^*_i(\cdot), \boldsymbol{x}^*_i(\cdot)\right)$.

2.3.2 Obstacle avoidance

While collision avoidance could be implemented in the MPC constraints, the optimal solution is in most cases on the boundary of the available set and then if for some error the state falls outside this set the problem becomes infeasible. Although this could be solved with slack variables, we propose a different approach that is particularly convenient in an experimental environment: this consists in the introduction of an additional term in the cost function, which avoids increasing the number of optimization variables.

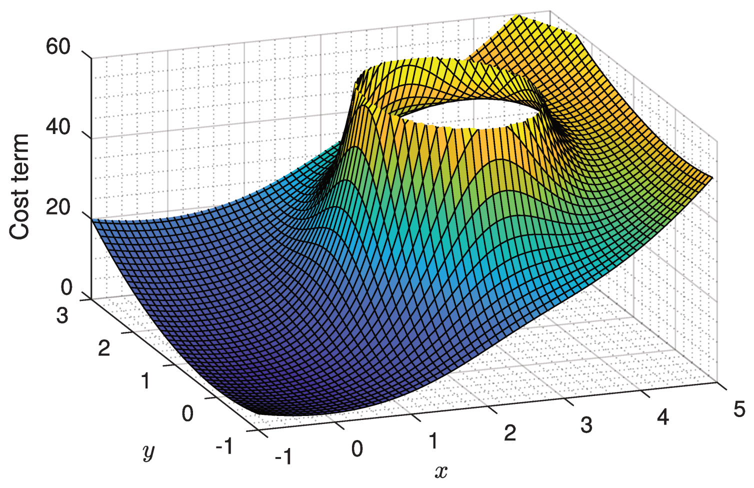


Figure 2.5.: Example of planar (x-y) components of the cost function, added to the distance penalty to the goal location.

Mobile robots are extended objects, so we consider a set of K points defined on the robot as a function of the state $\boldsymbol{p}_{i,k}(\boldsymbol{x}_i) \in \mathbb{R}^3$, e.g. $\boldsymbol{p}_v$, $\boldsymbol{p}_{e,i}$, directly extracted from the state, or any link origin $\boldsymbol{p}_{j,i}(\boldsymbol{x}_i)$, defined according the forward kinematics. A set of M obstacles is defined by their positions $\boldsymbol{o}_m \in \mathbb{R}^3$ and a radius $d_m \in \mathbb{R}$ defining the minimum distance avoiding collision. We then define:

$$J_{i,o}(\boldsymbol{x}_i) = \sum_{k=1}^{K} \sum_{m=1}^{M} C_{i,k,m} e^{-\lambda_{i,k,m}\left(\|\boldsymbol{p}_{i,k}(\boldsymbol{x}_i) - \boldsymbol{o}_m\| + d_m\right)} \tag{2.10}$$

where $C_{i,k,m}$ is the desired cost value on the boundary of the sphere defined by $(\boldsymbol{o}_m, d_m)$ and $\lambda_{i,k,m}$ determines the decay rate of the cost. Note that due to the exponential, the cost is negligible outside a radius defined by $\lambda_{i,k,m}$. Figure 2.5 shows an example the cost given by an obstacle with respect to the x and y components. Finally, in (2.6) and (2.9) we replace J_i with $J_{full} = J_i + J_{i,o}$, and the same algorithm apply.

2.4 Simulations and experiments

The proposed framework is validated through a realistic simulation (in Gazebo environment) and experimental results with 2 heterogeneous robots, where the continuous time formulation is discretized with a multiple shooting method. A common implementation for both the simulation and the experiment has been realized via a ROS network, that allows a common input-output interface, so that the same algorithm runs the on the two environment, as illustrated in Fig. 2.6.

The used heterogeneous robots consist of one ground and one aerial vehicles, as shown in Fig. 2.8. The ground robot is composed of an omnidirectional base, which is fully actuated on the floor plane, whereas the aerial robot is a planar hexacopter, both equipped with manipulators, of 4 and 2 revolute joints, respectively. The proposed framework is implemented with the ground robot being the leader and the aerial one being the follower.

To evaluate the performance of the algorithm, we consider $\boldsymbol{e}_\ell$ and $\boldsymbol{e}_o$, the position tracking error for the leader and the object, respectively derived from $\boldsymbol{T}_{e,\ell,ref}$ and $\boldsymbol{T}_{e,o,ref}$ Moreover, we consider the respective orientation metrics

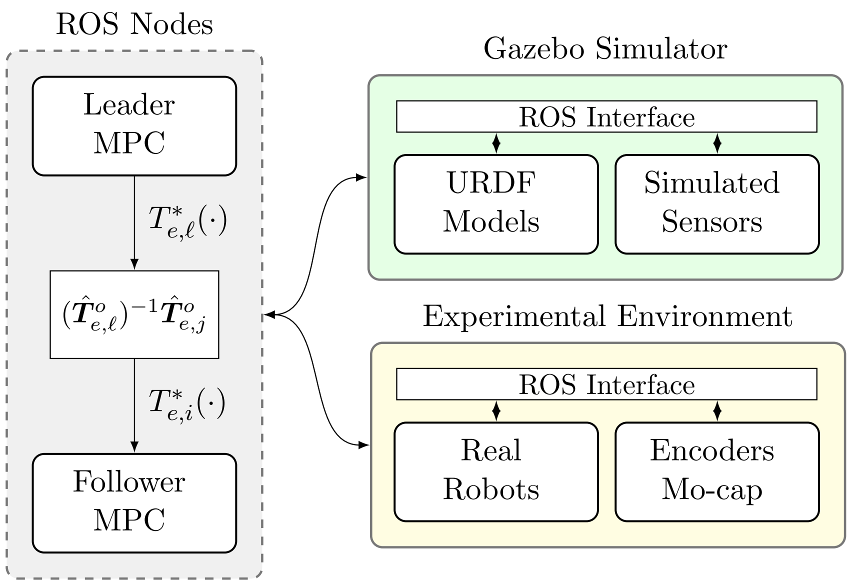


Figure 2.6.: The MPCs are implemented in ROS, allowing for a common interface to both the simulation and laboratory.

θ_ℓ, θ_o, defined as $\theta_* = \cos^{-1}(2((\tilde{\boldsymbol{q}}_*)^\top \tilde{\boldsymbol{q}}_{*,ref})^2 - 1)$, with $\tilde{\boldsymbol{q}}$ being the quaternion errors between the desired and the actual attitude. Since the follower does not follow an explicit trajectory but rather solves an optimization problem, we assess the performance for the aerial vehicle by inspecting the cost value of the MPC problem, which encodes the object displacement from the initial condition, and then it is proportional to the internal forces, according to assumption (2.2). Under ideal conditions, the MPC scheme should always be able to keep it close to zero and, in practice, this should still be bounded. We can assess the validity of the algorithm by verifying that the value does not increase over time.

2.4.1 Lower level controllers

In both the simulation and experiment, the MPCs run in different nodes on a off-board computer within the same ROS network as the robots, at $10\,\mathrm{Hz}$ and with an horizon length of $T = 1\,\mathrm{s}$, producing a velocity setpoint for the joints and the vehicle. In the case of the ground robot, these are directly supplied to the (real or simulated) motor drivers. The UAV relies on an attitude stabilization and a controller that converts the MPC command to desired roll-pitch-yaw-thrust. To increase robustness, the latter aims at tracking both the computed velocity and the first sample of pose from the trajectory predicted by the MPC.

2.4.2 Gazebo simulations

Gazebo is a multi-robot simulator based on *Open Dynamic Engine* physics-engine that allows for realistic robot simulations (Koenig and Howard, 2004). The two robots are simulated in Gazebo via custom URDF models, as represented in Fig. 2.1. The dynamic of all joints and the base of the ground robot are simulated via `ros_control`, while `RotorS` (Furrer et al., 2016) is employed for n-rotor flight simulation.

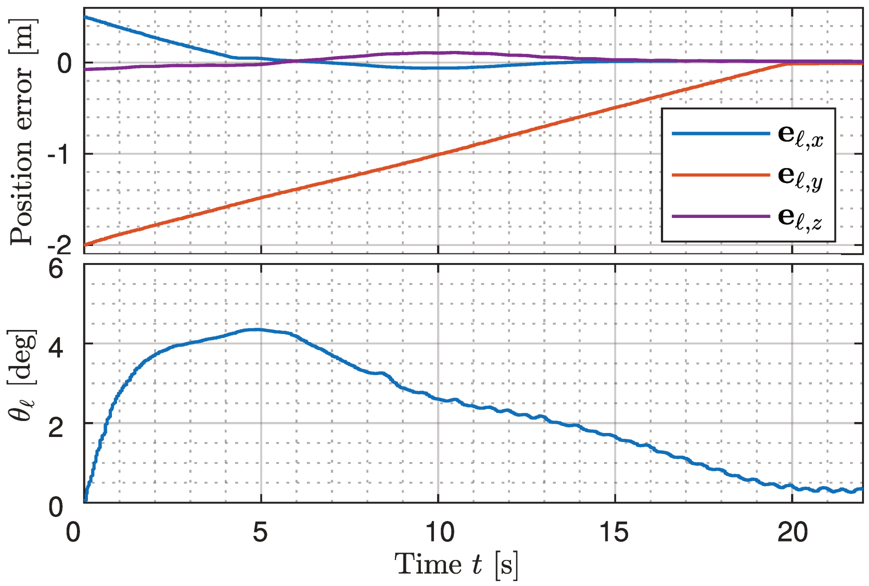
To launch the scenario, the UAV takes off and starts hovering in a predefined position near the object, in the grasp position. After this, the experiment starts ($t = 0$) when the end effectors of ground and aerial robots are at $[0.39,\ 0.02\ 0.34]^\top$ [m] and $[0.81,\ 0.00,\ 0.04]^\top$ [m]. Fig. 2.7 shows the results of a simulation where the algorithm is tested with a constant setpoint in $[0, 1.5, 0.3]^\top$ [m] and the same orientation as the initial state, with a box placed at $[0.0, 0.5, 0.0]^\top$ [m] Fig. 2.7 shows that both the ground and the aerial robots are able to drive the object error to zero by avoiding the obstacle. In particular, figure 2.7c shows that the displacement of the follower with respect to the prescribed object trajectory is bounded through the transportation and eventually converges to zero, despite an initial peak due to a delay in the reference tracking. The simulation example is clearly illustrated in the accompanying video[1]

2.4.3 Experiments

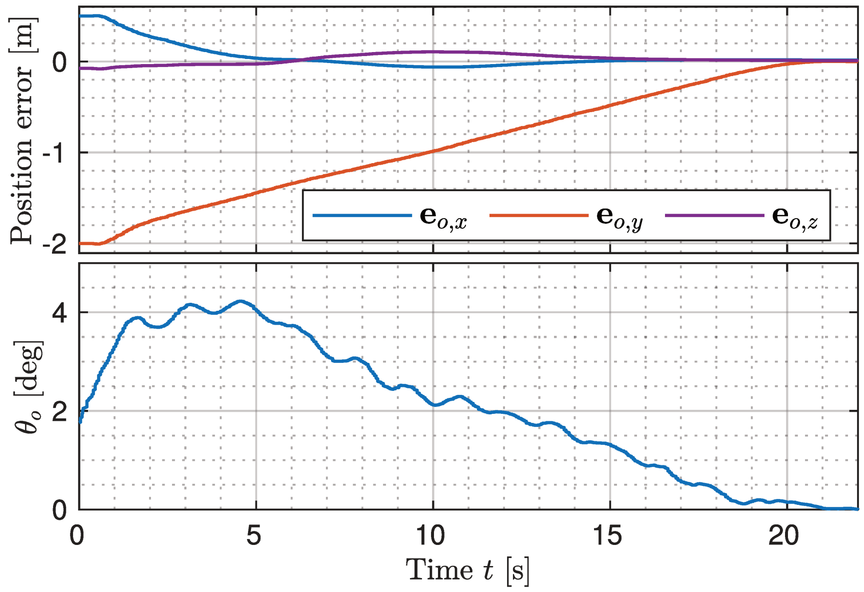
The experiment, whose setup shown in Fig. 2.8, was conducted at the Smart Mobility Lab[2], at KTH Royal Institute of Technology. A motion capture system was employed to measure the quantities that are part of the state in (2.1), i.e. the poses of ground base and end effector, and the vehicle of the UAV. The latter's end-effector, instead, was occluded by the vehicle and was estimated via the open-loop forward kinematics. In the initial configuration the robots are assumed to be already grasping the object, which is a plastic bar, 0.85 m long, that allows for some elastic deformation.

[1] https://youtu.be/e5iIwj9tm1c

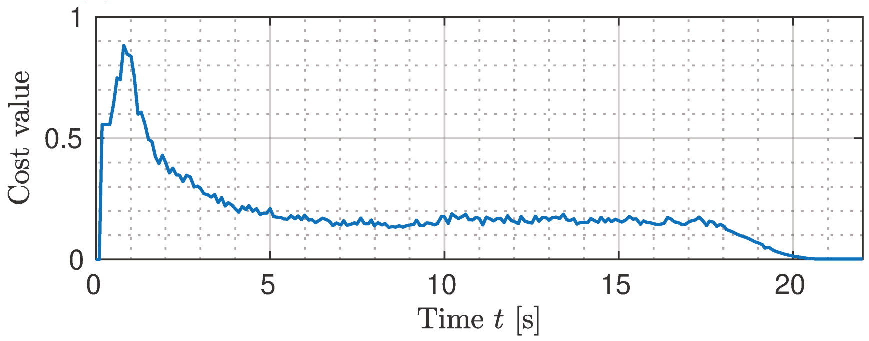
[2] https://www.kth.se/is/dcs/research/control-of-transport/smart-mobility-lab/smart-mobility-lab-1.441539



(a) Errors for ground (leader) end-effector



(b) Errors for the object estimate from aerial (follower) agent



(c) Aerial MPC objective cost value for predicted window

Figure 2.7.: Gazebo simulation results

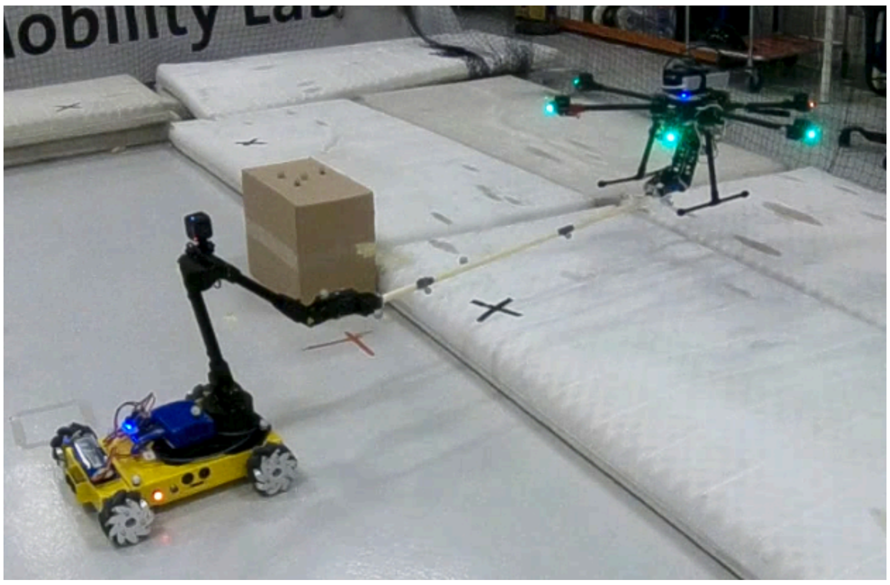
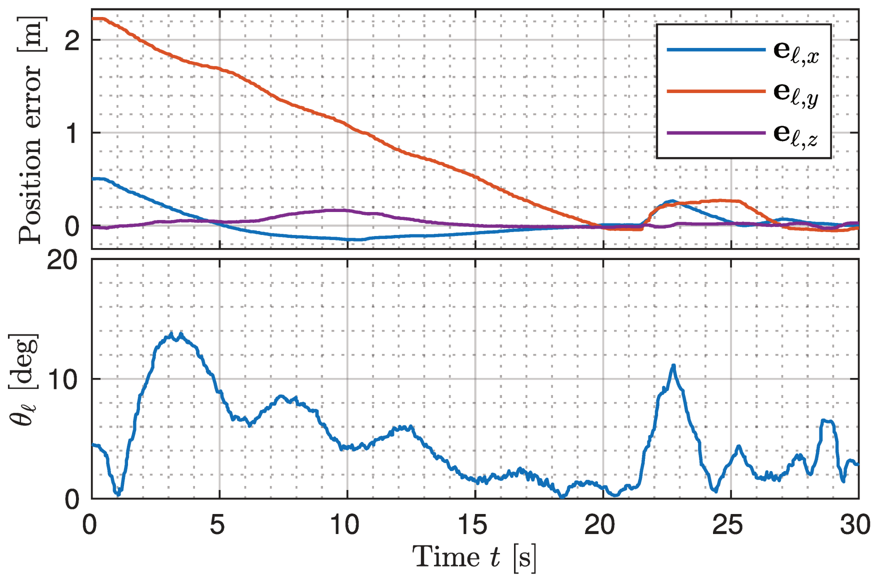


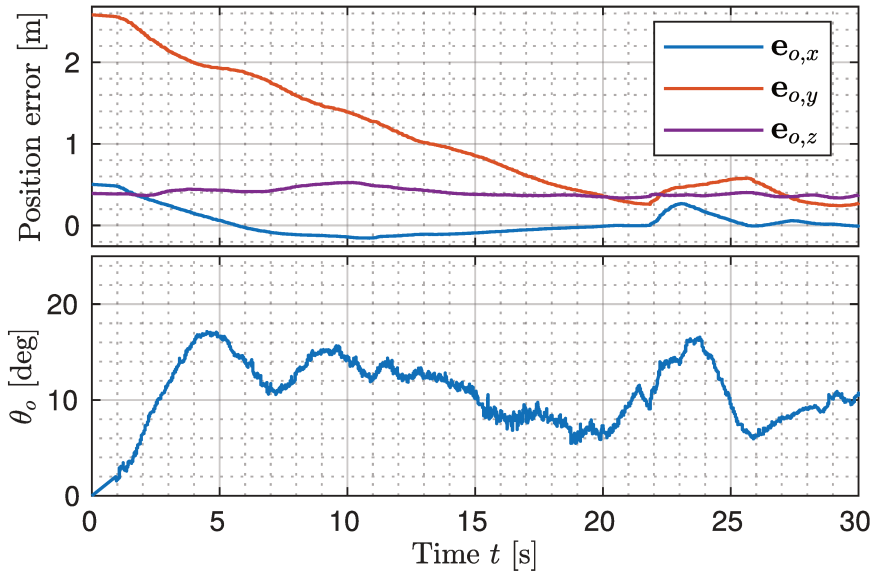
Figure 2.8.: Experimental setup

The experiment starts with the UAV hovering, grasping the object with its end-effector at $[-0.42,\ 0.60,\ 0.15]^\top$ [m], while the ground end-effector is at $[0.42,\ 0.61,\ 0.2687]^\top$ [m] and the bar at $[0.09,\ 0.60,\ 0.2067]$ [m]. Two obstacles, one traffic cone and one box, are placed at $[1.00,\ -0.61,\ 0.00]$ and $[1.00,\ -0.61,\ 0.00]^\top$ [m], forcing the vehicles to perform an avoidance maneuver. The results of a constant-setpoint tracking experiment, where the goal position is set at $[-0.0175,\ -1.5652,\ 0.3000]^\top$ [m] and the goal orientation is the same as the initial one, similar to simulation are reported in Fig. 2.9a, 2.9b and 2.9c.

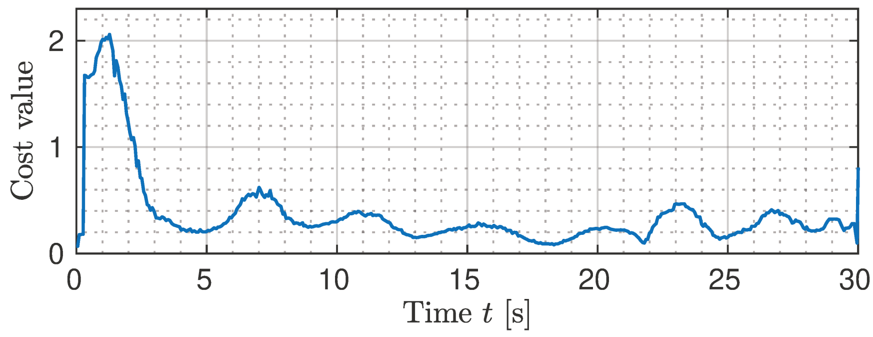
It can be noticed from Fig. 2.9 that, while the leader robot is able to converge to its setpoint, the object has some error. This means that the relative transforms between the robots and the object, $\boldsymbol{T}^o_{e,\ell}(t)$ and $\boldsymbol{T}^o_{e,j}(t)$, are not exactly equal to the initial condition $\boldsymbol{T}^o_{e,\ell}(0)$ and $\boldsymbol{T}^o_{e,j}(0)$. Fig. 2.9c shows that while the cost is bounded, and then the displacements are within the physical limits of object detachment, it is not driven to zero, as in the gazebo simulations. This cannot be attributed to the effects of the internal forces because, intuitively, they would tend to push the follower to lower the error. Instead, the degradation of performance can be caused by saturation in the low level controller of the UAV, and ground effects that arise since the latter is flying close to the ground and the obstacles. Nevertheless, both the error and the cost in Fig. 2.9c are bounded and the MPC scheme is able to keep the system stable, even when at time $t = 22\,\mathrm{s}$, when a fictitious external disturbance is simulated by applying a short impulse to the ground vehicle



(a) Tracking errors for ground (leader) end-effector



(b) Tracking errors for the object



(c) Aerial MPC objective cost value for predicted window

Figure 2.9.: Experimental results

arm joints commands. In that case, the plots show that the algorithm is able to handle the disturbance and keep the error bounded. The experiment is clearly illustrated in the accompanying video, which shows that the algorithm is able to complete the transportation task without having the object detached or damaged.

2.5 Conclusions

In this work we proposed a decentralized algorithm to coordinate a team of heterogeneous robotic agents that are designed to transport an object to a prescribed target pose. The procedure is designed to be robust to uncertainties and unmodelled dynamics such as underactuation and non-ideal tracking of the computed control inputs. The technique was tested both in a realistic simulation framework and with a laboratory experiment. In the former, the task was completed with converging errors, whereas in the latter an error is present at steady-state, mostly due to imperfect low-level control tracking. Nevertheless, the system is still able to keep the error bounded and react to unexpected external disturbances.

The main limitation of this approach is that the finite horizon optimization problem needs to be (locally) convex: in particular, in presence of multiple solutions, the algorithm will not necessarily compute the shortest path but the one following the local convexity. While in many cases this suboptimal solution can be acceptable, in more complex scenarios this can prevent a feasible solution to be found. In this sense, the proposed strategy should be intended as solving the problem of obstacle avoidance and not as a path planner. Indeed, non-convex scenarios need proper approaches and will be addressed in the following chapters.

3 NAPVIG: narrow passage navigation

3.1 Introduction

In the previous chapter we developed a technique for multi robot coordination and obstacle avoidance strongly relying on the knowledge of the position and orientation of every robot parts and the obstacles in a fixed reference frame. We now move to a new approach in order to address broader contexts where an exact localization is not possible.

In general, the path planning task is resting on two main approaches: the *graph-search* and the *sampling-based* paradigm. Graph-search algorithms rely on the discretization of the entire operating space to find the path, corresponding to a minimum cost; while sampling-based methods rest upon a sparse sample-based representation of the operating space. The probably most well-known sampling-based planner is Rapidly-exploring Random Trees (RRT) (LaValle et al., 1998). Its derivations like RRT* (Karaman and Frazzoli, 2011) and others, e.g., (Salzman and Halperin, 2016; Chen et al., 2018; Mashayekhi et al., 2020; Fu et al., 2020) are ubiquitous, and recently, further derivations of RRT have been developed to handle dynamic environments (Chandler and Goodrich, 2017; Qi et al., 2020). On the other hand, the A* algorithm and its derivations (Lindqvist et al., 2021; Pairet et al., 2021) constitute the most popular graph-search method, commonly adopted when dealing with unknown environments.

In general, the mentioned RRT and A* algorithms can find effective and feasible path planning solution for mobile robots. Nonetheless, both graph-search and sampling-based approaches tend to be computationally highly demanding in case of complex scenarios, especially in presence of narrow passages and/or cluttered environments. Indeed, in these cases requiring high precision in the path definition, the RRT algorithms are penalized by the need of a large number of samplings and the employment of ad-hoc strategies (Bun et al., 2021), whereas the need of smaller grid cells quickly deteriorates the performance of A* algorithms. Moreover, both graph-search and sampling-based methods are not well suited for reactive navigation, since they imply dynamic remapping and replanning. Thus, real-time requirements represent a further, highly challenging aspect when dealing with an unknown and dynamic environment. In this case, the state of the art solutions involve the exploitation of artificial potential fields (APF) (Paternain et al., 2017). However, although ensuring reactive navigation, their application is mostly limited to convex map configurations (Arslan and Koditschek, 2019). Other techniques are based on a dynamic window (Molinos et al., 2019), and predictive approaches (Choi et al., 2017). An alternative approach is navigating safely, farthest away from obstacles, relying on the Generalized Voronoi Diagram (GVD) (Choset and Burdick, 1995). While this allows for safe navigation, computing the entire GVD is very costly and is usually employed for offline applications and global path planning (Bhattacharya and Gavrilova, 2008). Nonetheless, it is possible to approximate the GVD given sensor data by applying selective algorithms on the Voronoi tessellation of the measurements (Mahkovic and Slivnik, 2000), or by discretizing the known map into cells and then applying computer vision techniques (Marie et al., 2019) based on the distance transform (Datta and Soundaralakshmi, 2003). Such techniques, however, being based on a discretization of the map, still require a trade-off between precision and computational load, which is critical in cluttered and narrow environments.

In this work, we present a novel approach for reactive navigation in unknown environments, that allows for computing the desired trajectory directly from LiDAR sensor data without the need of approximating a map representation with discrete grids. This allows precise computation with low computational requirements. Our approach is therefore specifically suited for embedded systems and where a very high reactivity is needed.

Conversely, the proposed algorithm exploits discrete raw measurements to build a spatially continuous landscape function, which is evaluated greedily along a section to generate a local trajectory. Although the landscape function has some conceptual similarities with APF, it is not used to generate a trajectory along its gradient, but it is used to precisely compute one point in the Voronoi diagram: the resulting point computed by the algorithm at each step depends on the initial conditions of the algorithm In doing this, the approach proves to be fast and efficient as to work in real-time, and it reveals to be inherently reactive to any event or environment change occurrence that may affect the path to travel. More specifically, the contribution of our work is the following:

- We develop a procedure to allow navigation without an a-priori map and without performing a preliminary localization, in unknown, cluttered and dynamic environments.
- Contrary to existing methods (Ramos and Ott, 2016), the algorithm is based on a continuous map representation that does not require any processing of the sensor data.
- This leads to very low computational requirement, allowing for implementation on highly dynamic and in case of limited computation capabilities.
- We conduct an extensive campaign of simulated and laboratory experiments, i) demonstrating the above claims and ii) showing a high precision and repeatability.

3.2 Problem formulation

We consider a context where a mobile robot is tasked to navigate in an unknown, unstructured and dynamic environment, possibly characterized by narrow passages. Given the large amount of uncertainties and the unpredictability that may affect such scenario, in order to achieve safe navigation, we set the goal to compute a trajectory that is the farthest away from the environment obstacles. In the next chapter, we will theoretically prove that this method is an approximation of the GVD.

Specifically, we assume that the occupied space is a time-variant subset of the plane $\mathcal{X}_{occ}(t) \subset \mathbb{R}^2$, where $t \in \mathbb{R}$ is time. Moving in the free space the robot is able to gather local measurements of the occupied space through a LiDAR mounted on board. Namely, with sampling period $T_m \in \mathbb{R}$, at time $t_k = kT_m$, $k \in \mathbb{N}$, the sensor obtains a measures set $\mathcal{M}_k := \{\boldsymbol{m}_{k,h},\ h = 1, \dots, H\}$, H being the number of measurements per sampling time. We introduce an inertial reference frame $\mathcal{F}_W$ and a robot frame $\mathcal{F}_t$, which is the pose of the robot at the continuous time $t \in \mathbb{R}$. For each sampling time we also consider a reference frame $\mathcal{F}_k$ corresponding to the pose of the robot at discrete time t_k relative to the inertial frame $\mathcal{F}_W$, so that the measurements $\boldsymbol{m}_{k,h}$ are expressed in $\mathcal{F}_k$. We will refer to frame $\mathcal{F}_k$ as *measurements frame*. Given that the measurements are obtained at discrete sampling times, we also consider the relative frame $\mathcal{F}_{k,t}$ that is the relative frame between the discrete time measurements frame $\mathcal{F}_k$ and the continuous time robot frame $\mathcal{F}_t$.

To cope with scenarios characterized by dynamic space configurations, the goal is to iteratively compute a trajectory $\boldsymbol{\xi}(t) \in \mathbb{R}^2$ which satisfies the safety requirements with respect to the (static or dynamic) obstacles and that is supplied at each time t to the low level controllers that steer the robot towards the computed position in a pure pursuit fashion. It is important to notice that $\boldsymbol{\xi}(t)$ is expressed in the robot frame $\mathcal{F}_t$.

3.3 Landscape function definition

The focus of the proposed algorithm is to find a trajectory that is at the maximum distance from the obstacles. In this view, we define a *landscape* function that maps each point of $\mathbb{R}^2$ to a value related to the distance of the nearest measurement, in analogy with the grid-based distance transform. Unlike the latter, however, we define a function that is spatially continuous and does not need to be computed at every point of a grid, as it will be clear later.

The idea is to consider a Gaussian-like function centered on each measurement. Formally, we define a *Gaussian peak*:

$$\Gamma : \mathbb{R}^2 \times \mathbb{R}^2 \to (0, 1] : (\boldsymbol{x}, \boldsymbol{m}) \mapsto e^{-\frac{\|\boldsymbol{x}-\boldsymbol{m}\|^2}{2\sigma^2}}. \tag{3.1}$$

Then, for a measurement set $\mathcal{M}_k$, we define:

$$\check{\mathcal{L}}_k(\boldsymbol{x}) = \max_{\boldsymbol{m}\in\mathcal{M}_k} \Gamma(\boldsymbol{x}, \boldsymbol{m}) \tag{3.2}$$

It is easy to see that, given a measurement set, the value of this function is proportional to the distance to the closest obstacle, according to a Gaussian

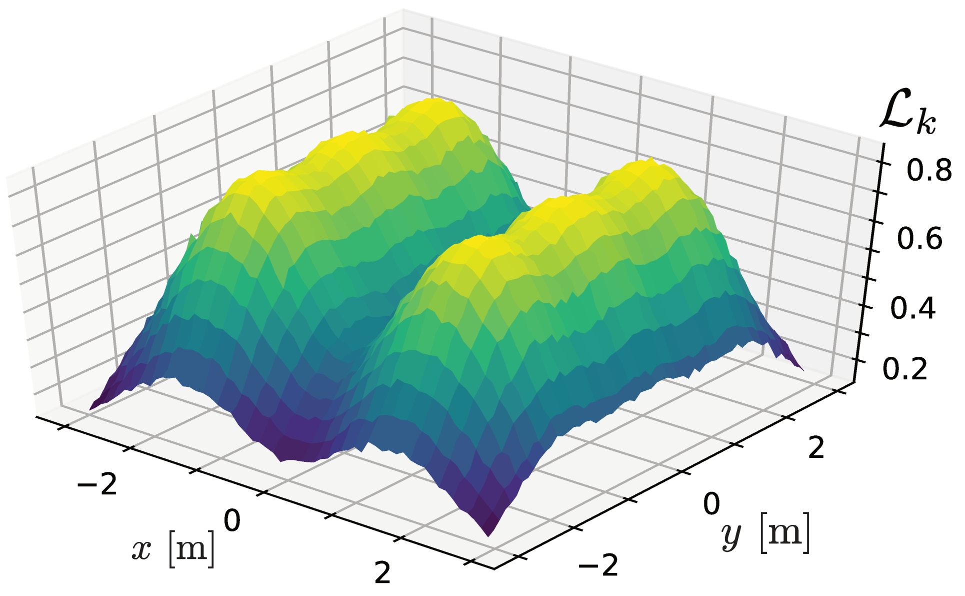


Figure 3.1.: Example of a landscape function computed with a Monte Carlo method. Intense yellow areas correspond to the measurement sources; dark blue areas are the safe regions.

profile, obtaining a representation of the space characterized by elevations in correspondence with the border of the obstacles and depressions in the free space, with analogies to the distance transform.

The choice of taking the *max* instead of the *sum*, which is what is usually done in classic artificial potential fields methods, is crucial. Indeed, taking the sum would produce different peak values depending on the measurements spatial density, which can be highly non-uniform, and then producing an unequal representation of space, with high peaks in correspondence to more dense areas. On the contrary, with the max operation we obtain a map that retains the information of distance from the obstacles. Nonetheless, while it can lead to a balanced landscape, the resulting function is not differentiable. The approach we propose performs a constrained optimization on this function, as we will introduce in the next section, hence the possibility to evaluate this function only in the points considered by the optimization algorithm. Nevertheless, the latter is based on a gradient method and function (3.2) as it is, is non-differentiable. Indeed, to solve this issue, we obtain a smooth version by performing a convolution with a Gaussian kernel. We define the result of this smoothing as the *landscape function*, namely, for the measures $\mathcal{M}_k$ taken at time t_k:

$$\mathcal{L}_k(\boldsymbol{x}) = \int_{\mathbb{R}^2} \check{\mathcal{L}}_k(\boldsymbol{w})\kappa(\boldsymbol{x}-\boldsymbol{w})\,\mathrm{d}\boldsymbol{w} \tag{3.3}$$

where the kernel is defined as:

$$\kappa(\boldsymbol{x}) = \sqrt{\frac{\sigma^2 + \delta^2}{2\pi\sigma^2\delta^2}} e^{-\frac{\|\boldsymbol{x}\|^2}{2\delta^2}} \tag{3.4}$$

with $\delta > 0$ being the smoothing radius and choosing the gain to compensate the amplification of the convolution operation.

A representative example of this function is depicted in Fig. 3.1. The landscape function is differentiable, since in this formulation the gradient is mathematically computed only on the kernel, which is trivially differentiable:

$$\frac{\partial}{\partial \boldsymbol{x}} \mathcal{L}_k(\boldsymbol{x}) = \int_{\mathbb{R}^2} \check{\mathcal{L}}_k(\boldsymbol{w}) \frac{\partial}{\partial \boldsymbol{x}} \kappa(\boldsymbol{x} - \boldsymbol{w}) \, \mathrm{d}\boldsymbol{w} \tag{3.5}$$

The integral in (3.5) cannot be computed in closed form, but it can be approximated with a Monte Carlo method, efficiently exploited thanks to the Gaussian kernel choice. Indeed, note that:

$$\mathcal{L}_k(\boldsymbol{x}) = \int_{\mathbb{R}^2} K_{na} \check{\mathcal{L}}_k(\boldsymbol{w}) e^{-\frac{\|\boldsymbol{x}-\boldsymbol{w}\|^2}{2\delta^2}} \, \mathrm{d}\boldsymbol{w}, \tag{3.6}$$

where K_{na} is the gain in (3.4), has the form of the expected value of function of a Gaussian random variable $\boldsymbol{W} \sim \mathcal{N}(\boldsymbol{x}, \delta^2 \boldsymbol{I}_2)$:

$$\mathcal{L}_k(\boldsymbol{x}) = \mathbb{E}\left[[] \ \boldsymbol{w}\right] 2K_{na}\pi\delta^2 \check{\mathcal{L}}_k(\boldsymbol{w}) \tag{3.7}$$

Then, we can estimate the expectation by drawing a number N_{mc} of samples from $\boldsymbol{W}$, $\bar{\boldsymbol{w}}_1, \ldots, \bar{\boldsymbol{w}}_{N_{mc}}$, and compute the sample mean:

$$\mathcal{L}_k(\boldsymbol{x}) = \frac{1}{N_{mc}} \sum_{i=1}^{N_{mc}} 2K_{na}\pi\delta^2 \check{\mathcal{L}}_k(\boldsymbol{w}) \tag{3.8}$$

Finally, we compute the gradient from (3.8):

$$\frac{\partial}{\partial \boldsymbol{x}} \mathcal{L}_k(\boldsymbol{x}) = \frac{1}{N_{mc}} \sum_{i=1}^{N_{mc}} 2K_{na}\pi\delta^2 \frac{\partial}{\partial \boldsymbol{x}} \check{\mathcal{L}}_k(\boldsymbol{w}) \tag{3.9}$$

where the gradient of $\check{\mathcal{L}}_k$ is

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{x}} \check{\mathcal{L}}_k(\boldsymbol{x}) &= -\frac{\boldsymbol{x} - \boldsymbol{m}^*}{2\sigma^2} e^{-\frac{\|\boldsymbol{x}-\boldsymbol{m}^*\|}{2\sigma^2}}, \\ \boldsymbol{m}^* &= \underset{\boldsymbol{m} \in \mathcal{M}_k}{\operatorname{argmin}} \|\boldsymbol{x} - \boldsymbol{m}\|. \end{aligned} \tag{3.10}$$

It can be verified that the points $\boldsymbol{x}$ where the argmin is not unique correspond to the non-differentiable points of $\check{\mathcal{L}}_k$, so in that case $\frac{\partial}{\partial \boldsymbol{x}} \check{\mathcal{L}}_k(\boldsymbol{x})$ is not defined. However, this is not an issue since those points are, by definition, in the Voronoi diagram of the measurements set $\mathcal{M}_k$, that is provably a zero measure set in $\mathbb{R}^2$ and hence the probability of randomly sampling one of those points is zero.

With this technique, intuitively, we exploit the similarity of the Gaussian smoothing kernel to a Gaussian distribution around the point of interest to obtain an efficient estimate of the landscape value and gradient. To obtain the results of this work it was sufficient a number of samples $N_{mc} = 50$. Note that this parameter, together with the termination condition of the optimization algorithm, are the only tuning variables that control the trade-off between computational cost and precision. For all our experiments the values for the peak and smoothing radius are chosen to be, respectively, $\sigma = 0.2$ and $\delta = 0.02$, and they only should scale with the average size of the robot and the obstacles.

3.4 Narrow passage navigation (NAPVIG) algorithm

Given the landscape function previously introduced, the rationale is based on the observation that, in one dimension, the minimum between the two centers of two Gaussians is the middle point. We extend this concept to the 2D case by considering the restriction of the landscape function to a line, and looking for the minimum point. Experimentally, it can be shown that the 1D analogy still holds for the 2D case.

In this section, we present the algorithm to compute a single point of the trajectory, which we refer to as *NAPVIG* algorithm. Given an initial point $\boldsymbol{x}_0 \in$

$\mathbb{R}^2$, we consider a *search direction*, that is a vector in $\mathbb{S}^1 = \{\boldsymbol{r} \in \mathbb{R}^2 \;:\; \|\boldsymbol{r}\| = 1\}$, that represents the direction in which we want to look for the next sample. The input variables of the algorithm are then $\boldsymbol{x}_0$ and $\boldsymbol{r}$. Then:

- We consider the point:

$$\boldsymbol{x}^{(0)} = \boldsymbol{x}_0 + \beta_{step}\boldsymbol{r} \tag{3.11}$$

  that is β_{step} farther from $\boldsymbol{x}_0$ in the $\boldsymbol{r}$ direction.

- We consider the restriction set as the line passing through $\boldsymbol{x}^{(0)}$ that is orthogonal to $\boldsymbol{r}$. First, let $\mathcal{R}^\perp$ be the orthogonal space to $\boldsymbol{r}$. Then, the restriction set is the line:

$$\mathcal{A} = \{\boldsymbol{x}^{(0)} + \lambda \boldsymbol{r}^\perp,\ \lambda \in \mathbb{R},\ \boldsymbol{r}^\perp \in \mathcal{R}^\perp\} \tag{3.12}$$

- We look for the minimum of the landscape function $\mathcal{L}_k$ restricted to $\mathcal{A}$:

$$\boldsymbol{x}^* = \underset{\boldsymbol{x} \in \mathcal{A}}{\operatorname{argmin}}\ \mathcal{L}_k(\boldsymbol{x}) \tag{3.13}$$

  We compute this point by applying a constrained version of gradient descent, as follows:

  - We consider the orthogonal projection operator $\mathrm{P}_{\mathcal{R}} : \mathbb{R}^2 \to \mathcal{R}^\perp$ that associates a vector $\boldsymbol{v} \in \mathbb{R}^2$ to its orthogonal projection onto $\mathcal{R}^\perp$.

  - We perform a gradient descent constrained to $\mathcal{R}^\perp$, with the following update rule

$$\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} + \eta \mathrm{P}_{\mathcal{R}} \left[\frac{\partial}{\partial \boldsymbol{x}^{(i)}} \mathcal{L}_k(\boldsymbol{x}^{(i)}) \right] \tag{3.14}$$

    where $\eta > 0$.

  - The gradient descent terminates when it holds $\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(i-i)}\| < \varepsilon_{term}$, $\varepsilon_{term} > 0$.

The output of the algorithm is the point from the last iteration of the gradient descent, namely:

$$\mathrm{napvig}(\boldsymbol{x}_0, \boldsymbol{r}, \mathcal{L}_k) := \boldsymbol{x}^{(i^*)} \tag{3.15}$$

where i^* is the index in the iteration when the termination condition is satisfied.

3.4.1 Navigation strategy

We employ the procedure just introduced to reactively compute the next trajectory sample, given the current robot observations. With the landscape function $\mathcal{L}_k$ of the measurements taken at time t_k, we consider that the relative frame between the robot at a generic time t and the last measurement frame, is composed as follows: $\mathcal{F}_{k,t} = (\boldsymbol{x}_{k,t}, \boldsymbol{R}_{k,t})$, where $\boldsymbol{x}_{k,t}$ is the origin of frame $\mathcal{F}_{k,t}$ and $\boldsymbol{R}_{k,t}$ a rotation matrix representing its orientation. We then choose the initial conditions as $\boldsymbol{x}_0 = \boldsymbol{x}_{k,t}$, and $\boldsymbol{r} = \boldsymbol{R}_{k,t}[1,\, 0]^\top$. The trajectory sample at time t is then:

$$\boldsymbol{\xi}(t) = \mathcal{F}_{k,t}^{-1}\,\mathrm{napvig}(\boldsymbol{x}_{k,t}, \boldsymbol{R}_{k,t}[1,\, 0]^\top, \mathcal{L}_k) \tag{3.16}$$

where $\mathcal{F}_{k,t}^{-1}$, with abuse of notation, converts the result of the algorithm from the measurement frame, where the landscape function is defined, to the current robot frame, where the low levels controllers get values. The frame $\mathcal{F}_{k,t}$ can be retrieved by local odometry (e.g. wheels encoder, IMU, etc.). Note that if the trajectory sample is computed synchronously to the measurement sample, i.e. for $t = t_k$, we obtain the simplified:

$$\boldsymbol{\xi}(t_k) = \mathrm{napvig}([0,0]^\top, [1,0]^\top, \mathcal{L}_k) \tag{3.17}$$

Remarkably, the purpose of the generalization is to allow the trajectory to be updated at a higher rate than the LiDAR scan, which is usually lower than the odometry one. In practice, $\boldsymbol{\xi}(t)$ can be computed at $t = \ell T_o$, $\ell \in \mathbb{N}$ and T_o being the odometry sample time, given the last measurements taken at $t_k = kT_m$.

This choice for the initial condition is to look for local minima of the landscape functions that are straight ahead in the robot's frame. Using such initial condition, however, suggests a preferential navigation direction (by design), which is surely valid in environments with no bifurcation. Nonetheless, in more general scenarios there could be multiple local minima, corresponding to the different possible directions, which can be selected depending on the initial conditions. In such cases, if needed, by adopting more advanced techniques on the choice of the initial search direction it is possible to control this selection, thus allowing NAPVIG to solve the navigation problem in any kind of environment. In this work we will limit our focus on testing the performance of the core algorithm on environments without bifurcations, while the extension to more general scenarios will be addressed in chapter 5.

3.4.2 Vehicle model and control

A differential steering robot can be modelled as a unicycle, controlled in linear and angular velocity, $v, \omega \in \mathbb{R}$ respectively:

$$\begin{bmatrix} \dot{\boldsymbol{x}} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \boldsymbol{u} \tag{3.18}$$

where $\boldsymbol{u} = [v, \omega]^\top$, $\boldsymbol{x}$ and θ are the position and the orientation of the robot in the plane. The non-holonomic constraint that characterizes this vehicle needs particular attention in narrow and dynamic environments: since lateral motions are only possible through arcs, in such contexts it is desirable to reduce their radius. To this aim, we slow down the linear velocity proportionally to the angular error. Specifically, given that $\boldsymbol{\xi}(t)$ is the current desired position calculated by the algorithm, expressed in the local frame $\mathcal{F}_t$, we consider the position error as a complex value, namely $e(t) = [1,\ i]\boldsymbol{\xi}(t) \in \mathbb{C}$, where i is the imaginary unit, in order to conveniently retrieve the angular and linear error without representation problems, and we employ the following control law:

$$\begin{aligned} \omega(t) &= k_{p,\omega} \Im \log(e(t)), \\ v(t) &= k_{p,v} \|e(t)\| e^{-\frac{\omega(t)^2}{2k_{v,\omega}^2}}, \end{aligned} \tag{3.19}$$

where $\Im$ denotes the imaginary part. Here, $\omega(t)$ is a proportional control on the steering angle needed to reach the current trajectory. The linear velocity command is proportional to the distance to the trajectory sample multiplied by a factor that reduces the curvature radius when the steering angle is high.

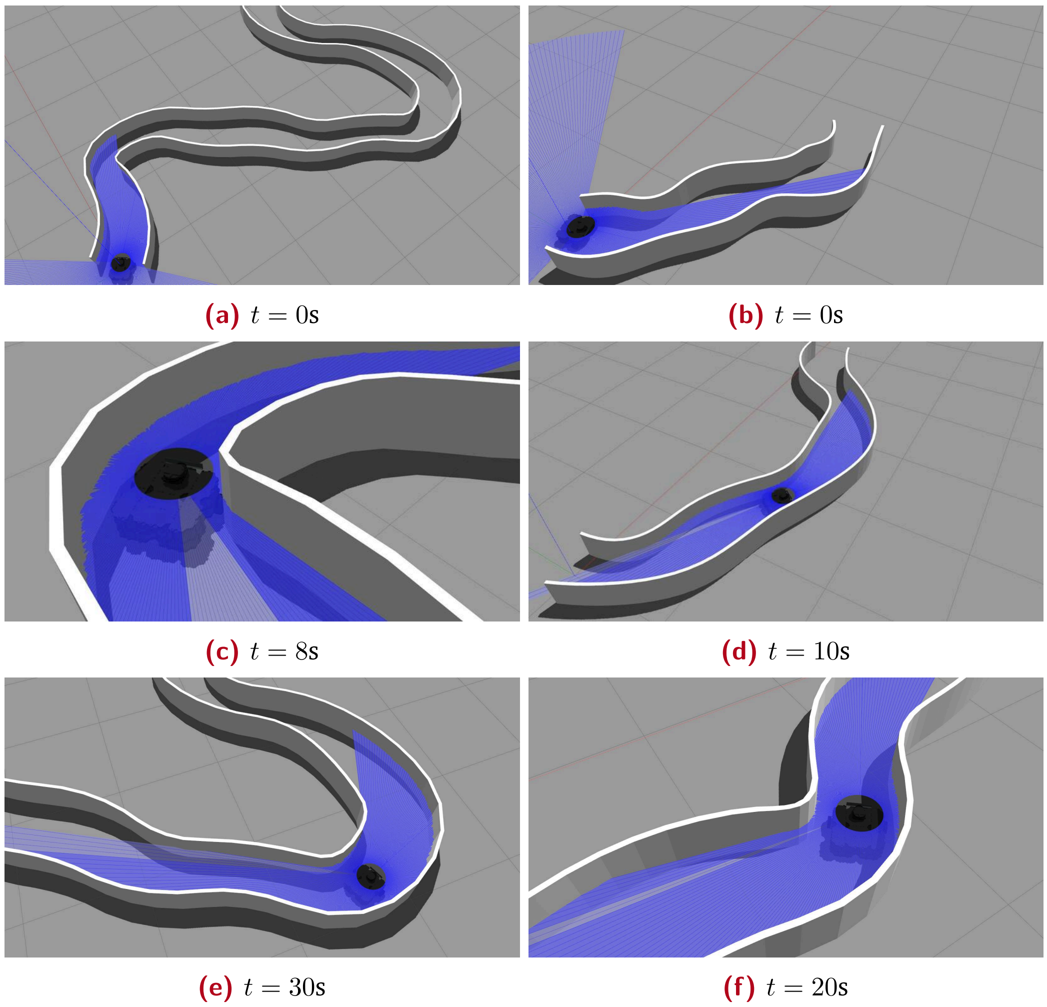

(a) $t = 0$s (b) $t = 0$s (c) $t = 8$s (d) $t = 10$s (e) $t = 30$s (f) $t = 20$s

Figure 3.2.: Gazebo experiments screenshots: static (left column) and dynamic (right column) scenarios: blue lines represent the visualization of the laser ray scan.

3.5 Simulations

In the simulated experiments, the laser scanner runs at 5 Hz while the odometry sensor at 100 Hz. The center of the corridor is defined in this case as the point with equal distance to the two walls, which we will define in this section. Fig. 3.2 shows several screenshots of the Gazebo simulation in the static/dynamic scenarios. These simulations are reported also in the accompanying video[1].

[1]`https://youtu.be/noMnzxsi4wE`

3.5.1 Environment design

We consider the context of a single, possibly time-variant, randomly generated corridor, by producing a set of random points at random angles and fixed radius:

$$\bar{\vartheta}_k \sim \mathcal{N}(0, \sigma_\vartheta^2), \; k = 1, \ldots, K_\vartheta, \sigma_\vartheta > 0 \tag{3.20}$$

To stress the approach and simulate an (extremely) time varying context, we consider a limit case where a sinusoidal disturbance is added at each time t to the generated angles:

$$\begin{aligned} \Theta(t) &= \{\vartheta_k(t) = \bar{\vartheta}_k + \sin(2\pi f_\vartheta t + \varphi_\vartheta), \; \varphi_\vartheta \sim \mathcal{N}(0, \sigma_\varphi^2)\} \\ \mathcal{K}(t) &= \{\boldsymbol{k}_{k+1}(t) = \boldsymbol{k}_k(t) + \rho \begin{bmatrix} \cos(\vartheta_k(t)) \\ \sin(\vartheta_k(t)) \end{bmatrix}, \; k = 1, \ldots, K_\vartheta\} \end{aligned} \tag{3.21}$$

with $\boldsymbol{k}(t)_0 = \mathbf{0}$, $\rho > 0$. Then, we interpolate the key points with a spline, obtaining the passage center:

$$\boldsymbol{\gamma}(\tau, t) = \text{spline}(\tau\,; \mathcal{K}(t)) \tag{3.22}$$

with a desired sampling set $\tau \in \mathcal{T} \subset \mathbb{R}$. The two walls that define part of $\partial\mathcal{X}_{coll}(t)$ are then:

$$\mathcal{W}_{1,2}(t) = \{\gamma(\tau, t) \pm \lambda_w(\tau, t)\boldsymbol{n}(\tau, t), \; \tau \in \mathcal{T}\} \subset \partial\mathcal{X}_{coll}(t) \tag{3.23}$$

where $\boldsymbol{n}(\tau, t)$ is the unit vector normal to $\boldsymbol{\gamma}(\tau, t)$ and $\lambda_w(\tau, t) > 0$ is the half width of the passage and we choose:

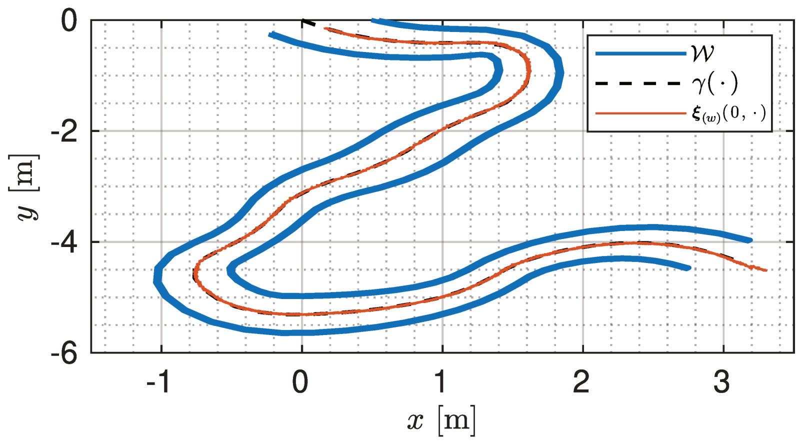
$$\lambda_w(\tau, t) = \lambda_0 + A_\lambda \cos(k_\lambda \tau - 2\pi f_\lambda t) \tag{3.24}$$

3.5.2 Static scenario

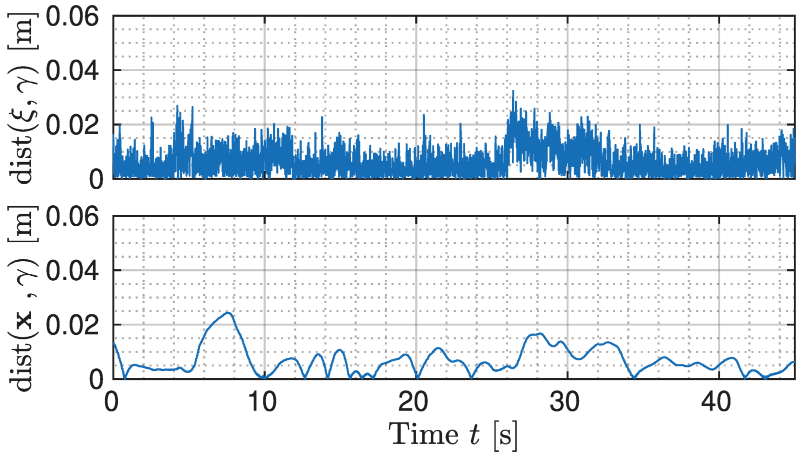
We first test the algorithm in a static scenario. We consider the environment described in the last section where the time is fixed, e.g. $\boldsymbol{\gamma}(\tau, 0)$, $\lambda_w(\tau, 0)$. Fig. 3.3a shows the corridor (in blue), its center in dashed black and the resulting trajectory computed by the algorithm. Visibly, the last two are mostly overlapping. The error with respect the center of the corridor of the trajectory $\boldsymbol{e}_{\boldsymbol{\xi}}(t) = \mathrm{dist}(\boldsymbol{\gamma}(t), \boldsymbol{\xi}_{(w)}(t))$, where $\boldsymbol{\xi}_{(w)}(t)$ is the trajectory sample expressed in the world frame, and the actual robot position $\boldsymbol{e}_{\boldsymbol{x}}(t) = \mathrm{dist}(\boldsymbol{\gamma}(t), \boldsymbol{x}(t))$ is reported in Fig. 3.3b. It appears that the algorithm is able to provide an accuracy of more than $3\,\mathrm{cm}$ from the corridor center, ensuring collision avoidance and safe navigation. Note that the main source of noise in the instantaneous trajectory computation is the gradient estimation accuracy, which can be selected to be more precise, at the cost of higher computational load and with the limitation of the accuracy of the laser measurement. Also, this simulation suggests that the designed procedure is effective and able to keep up with the real-time requirements, with an average computation time of $5\,\mathrm{ms}$. Although in this context, we are not targeting high speed performances, we verify that the resulting average speed is $0.23\,\mathrm{m/s}$, which is in line with many practical applications.

3.5.3 Dynamic scenario

In Fig. 3.4 we report the performance results of *NAPVIG* in a random dynamic environment. Fig. 3.4a shows a few corridor instances in light blue. In dark blue is the last corridor of the experiment visualized: we highlight that a-priori there does not exist any trajectory travelling the entire corridor without collision, so in this case it is mandatory resorting to the online computation. Since the corridor changes dynamically, we can only assess the distance from the corridor center by looking at the performance metrics, evaluated for each t according to the current $\boldsymbol{\gamma}(\cdot, t)$, as plotted in Fig. 3.4b. In this case, the error is slightly higher than that in the static case since, as it might be expected, the dynamic environment is more challenging to keep up with. Clearly, there are physical limitations on the rate of change of the environment, the most important of which, besides the measurement frequency, is the non-holonomic constraint that causes a delay in the path following. Nevertheless, the resulting tracked trajectory is able to avoid collisions, with an overall average speed of $0.22\,\mathrm{m/s}$.



(a) Randomly generated corridor and resulting trajectory



(b) Distance from the corridor center and the computed trajectory $e_{\boldsymbol{\xi}}(t)$ (top) and the actual robot position $e_{\boldsymbol{x}}(t)$ (bottom)

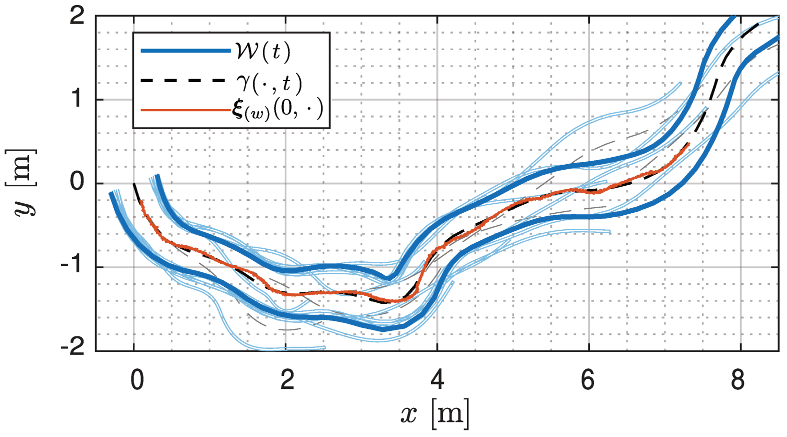
Figure 3.3.: Simulation: static environment test results

3.6 Experiments: narrow passages and cluttered scenarios

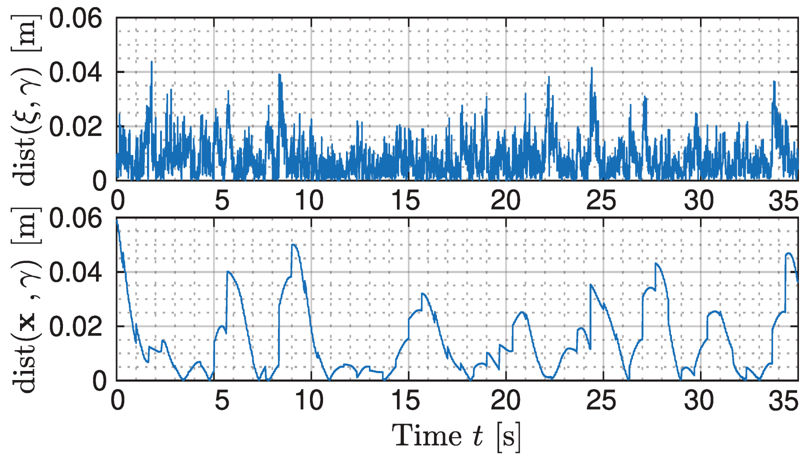
The experimental campaign has been conducted at the SPARCS Lab[2], at the University of Padova, using a custom-built differential drive robot. Videos of the experiments are provided in the accompanying video[3]

The first experiment setup is depicted in Fig. 3.5 and consists in a partially structured corridor: the length and position of the walls segments are measurable offline but they are built with imperfections, which in turn allows to assess the algorithm in a less idealized context. Fig. 3.7 shows the corridor

[2]`https://sparcs.dei.unipd.it/index.php/laboratories/sparcslab`
[3]`https://youtu.be/noMnzxsi4wE`



(a) Samples of time-varying corridors and final resulting trajectory



(b) Distance from the corridor center and the computed trajectory $e_{\boldsymbol{\xi}}(t)$ (top) and the actual robot position $e_{\boldsymbol{x}}(t)$ (bottom)

Figure 3.4.: Simulation: dynamic environment test results

walls (in black), the approximated generalized Voronoi diagram of the map computed offline (in blue), given the experimental position errors, and the robot paths using NAPVIG (in orange). The robot is initially placed with a generic pose at the beginning of the corridor (Fig. 3.7) and then the navigation algorithm drives through it towards the opposite end. The experiment is repeated several times and the plots show that after a short transient, the paths consistently converge to the center of the corridor.

A second experiment is designed in a more cluttered and irregular environment, where obstacles are placed in such a way to create a loop that the robot can travel multiple times (some snapshots of the experiment are shown in Fig. 3.6). In this scenario we assess repeatability of the navigation and control action under the same conditions while verifying the accomplishment of the task with no collision or impasse. The results reported in Fig. 3.8 present the overall path tracked in six runs along with four detailed views. After the first transient that drives the robot to the desired trajectory, starting at around $[-0.1, -1.1]$ [m], the figure and, in particular, the enlarged views,

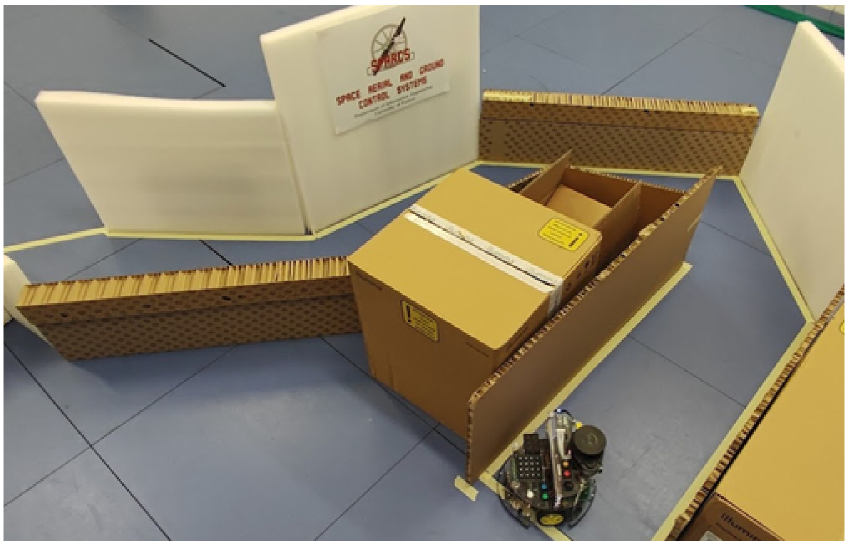


Figure 3.5.: Experiment #1: The wheeled robot passes through a narrow passage, set up with card boards.

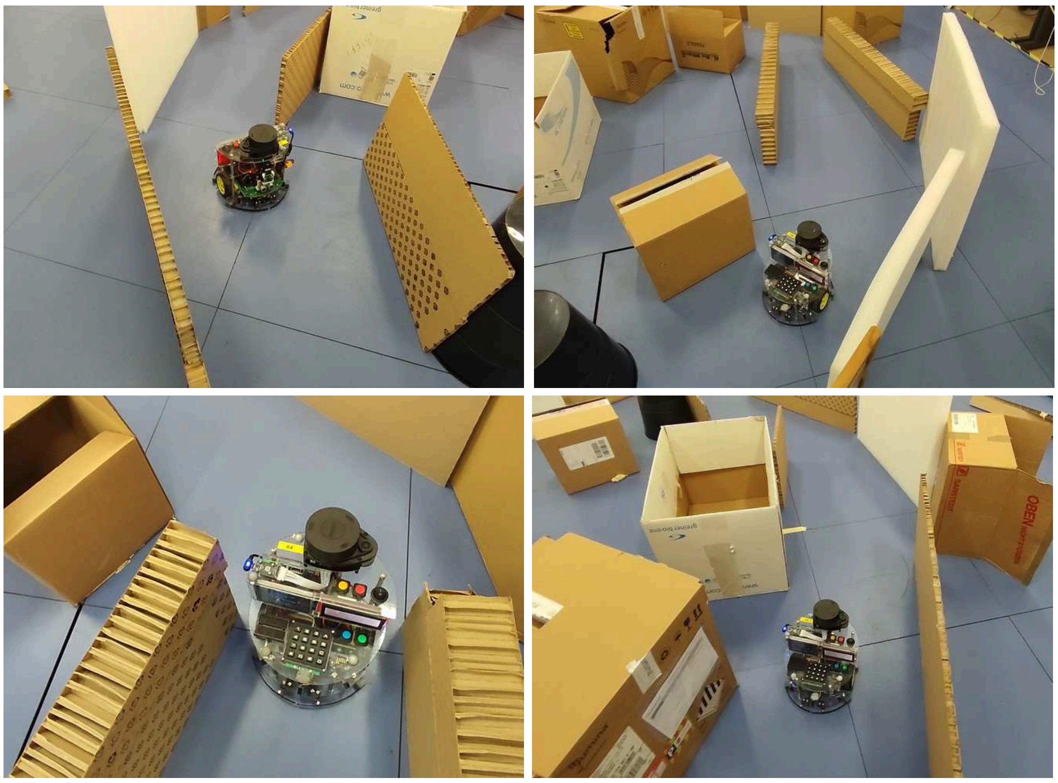


Figure 3.6.: Experiment #2: snapshots of setup.

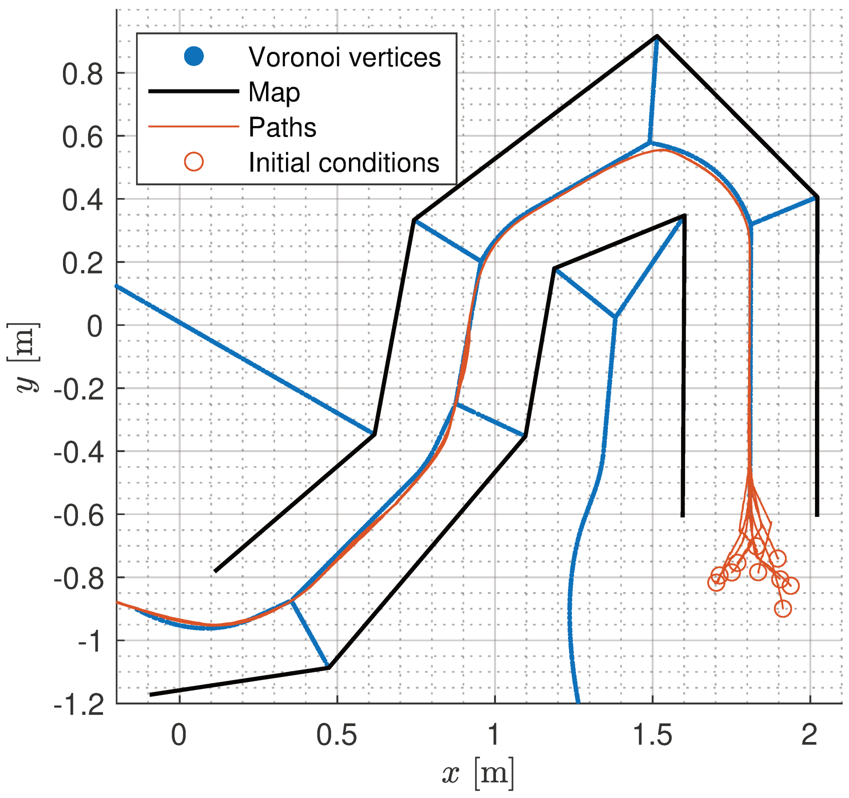


Figure 3.7.: Experiment #1: shows robustness to different initial conditions (circles) that converge to very similar paths (orange) close to GVD segments (blue).

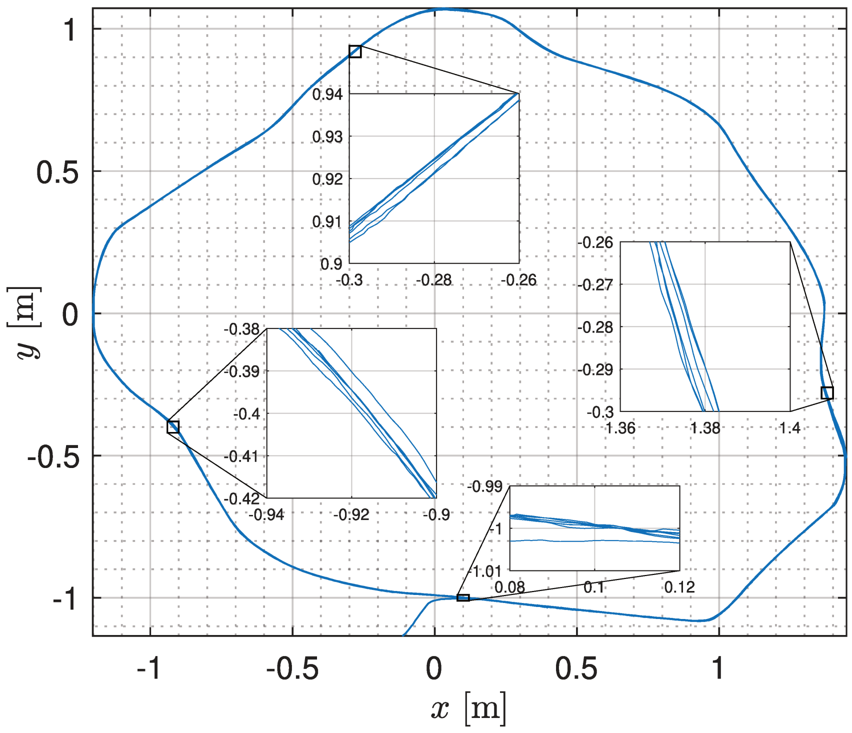
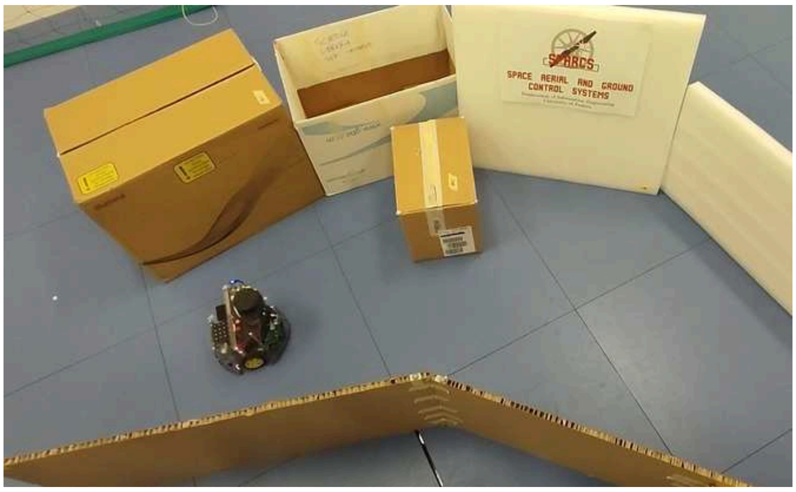
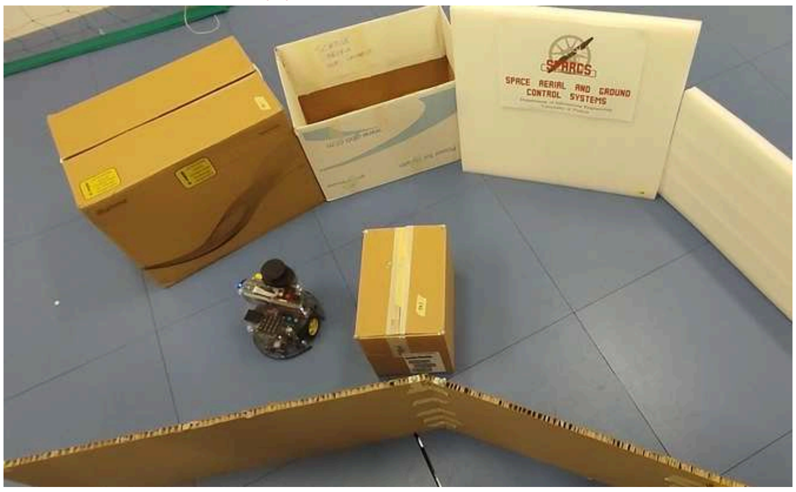


Figure 3.8.: Experiment #2: multiple loop runs show the precision and repeatability of the navigation algorithm.

show that the path repetition range is less than $1\,\mathrm{cm}$ along a total traveled distance of more than $6\,\mathrm{m}$ per loop.



(a) Initial configuration



(b) Final configuration

Figure 3.9.: Snapshots of experiment #3

Finally, given that the behavior with multiple dynamic parts has been assessed in simulation, we report a laboratory experiment with one moving obstacle. The environment is initially set up as in Fig. 3.9a and the algorithm starts computing a trajectory overcoming the obstacle on the right. The obstacle is then suddenly moved to a possible collision position and the robot reactively recomputes the trajectory to perform the navigation task. The overall path is reported in Fig. 3.10 showing the obstacle's initial and final positions in dashed gray and black, respectively: the NAPVIG algorithm drives the robot through a first straight part of the path, followed by a sudden evasion maneuver to overcome the moving obstacle.

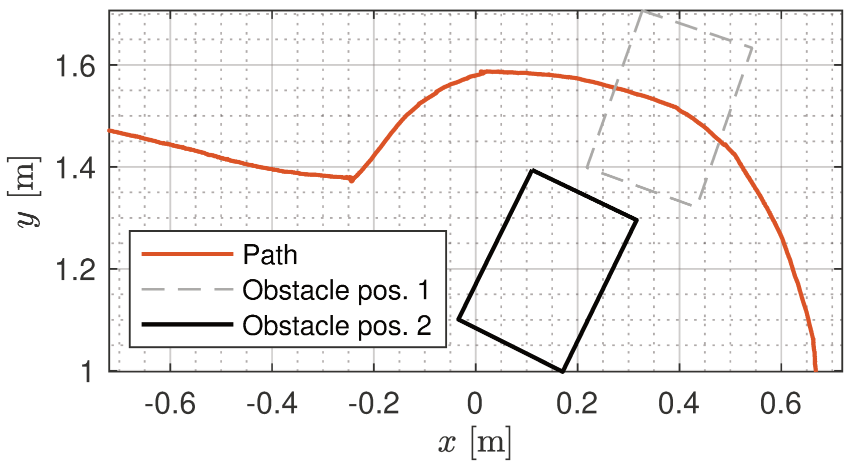


Figure 3.10.: Experiment #3: path tracked by the robot (in orange) compared to the obstacle position, initial (dashed gray) and final (solid black). The trajectory is reactively updated around point $[-0.2, 1.4]$.

3.7 Conclusions

We presented the NAPVIG algorithm for online local navigation, able to travel a dynamic environment with narrow corridors, relying only on on-board sensors. The discussion, the Gazebo real-time simulations, and the laboratory experiments, show that the approach is effective, consistent, and able to attain safe navigation, both with static and dynamic collision space configurations, while complying with the timing requirements for an online application.

While the NAPVIG algorithm is able to always compute a point farthest away to every obstacle, the path taken at bifurcations depends on its initial conditions. In this work we limited our analysis on environments where a simple choice of the latter resulted in the desired direction, focusing on the performance of the algorithm itself. Nevertheless, more advanced techniques for the decision of the initialization can address navigation in very general configurations. These aspects will be discussed in chapter 5.

4 Theoretical foundations of NAPVIG algorithm

4.1 Introduction

In the last chapter we introduced the NAPVIG algorithm that, exploiting the *landscape* representation of the configuration space based on the LiDAR measurements, and a constrained gradient descent algorithm to compute samples of the target trajectory. The method was motivated with an intuitive explanation and validated experimentally. We now proceed by defining a more theoretical formulation of both the algorithm and the Landscape function, of which we will provide their mathematical properties, and we will prove that the NAPVIG algorithm computes an approximation of the *Generalized Voronoi Diagram* of the map.

Voronoi diagrams (Fortune, 1995) have been extensively studied across a broad range of domains, from machine learning (Kolahdouzan and Shahabi, 2004) to computer graphics (Valette et al., 2008). Intuitively, given a (finite) set of points, the plane is subdivided into regions that are closer to a single point than all others. This concept can be adapted to more generic configuration space, made of continuous sets representing the obstacles, leading to the *Generalized Voronoi Diagram* (GVD), that is the set of points of the plane that are equidistant from two or more obstacles (Lee and Drysdale, 1981). Following the GVD has been one of the first solutions to the motion planning problem (Takahashi and Schilling, 1989), and is often used in combination with roadmaps (Lulu and Elnagar, 2005; Bhattacharya and Gavrilova, 2008), fast-marching methods (Garrido Bullón et al., 2011; Garrido et al., 2009) and artificial potential fields (Masehian and Amin-Naseri, 2004). One of the biggest limitations of these methods to be used in real time is the computational capacity needed to obtain accurate trajectories and the knowledge of a map representation. Indeed, fast computation need approximations (Edwards et al., 2015) and advanced algorithms (Kiseleva et al., 2019) and the computation is often uncoupled with mapping, that is often assumed to be known. For this reason, the application of GVD is limited to global navigation (Gomez et al., 2013). Only recently real time and reactive strategies employing the GVD for navigation have been effectively tested experimentally. For example, in (Marie et al., 2019) the GVD is computed with computer vision techniques starting from an omnidirectional camera, while in (Chi et al., 2021) it is combined with RRT-based methods. All these techniques are based on a grid approximation of the target space, thus requiring a trade-off between precision and computational load, which is critical in cluttered and narrow environments.

4.2 Preliminary definitions and properties

In this section we provide basic definition and results that will be useful to show the main propositions regarding the NAPVIG algorithm and the landscape function.

Definition 4.1. *Given a compact set $\mathcal{C}$, we define the* projection operator *to the set $\mathcal{C}$ as:*

$$\boldsymbol{P}_{\mathcal{C}} \; : \; \mathcal{D}_{\mathcal{C}} \; \rightarrow \; \mathcal{C} \; : \; \boldsymbol{x} \mapsto \boldsymbol{P}_{\mathcal{C}}(\boldsymbol{x}) := \operatorname*{argmin}_{\boldsymbol{y} \in \mathcal{C}} \|\boldsymbol{y} - \boldsymbol{x}\| \tag{4.1}$$

for the points $\boldsymbol{x} \in \mathcal{D}_{\mathcal{C}}$ for which the argmin *is unique:*

$$\mathcal{D}_{\mathcal{C}} = \{\boldsymbol{x} \in \mathbb{R}^2 \; : \; \exists! \; \bar{\boldsymbol{y}} \in \mathcal{C} \text{ s.t. } \|\bar{\boldsymbol{y}} - \boldsymbol{x}\| \leq \|\boldsymbol{y} - \boldsymbol{x}\|, \quad \forall \boldsymbol{y} \in \mathcal{C}\} \tag{4.2}$$

Proposition 4.1. *Given a point $\boldsymbol{y} \in \mathcal{C}$ and a compact set $\mathcal{C}$, if $\mathcal{C}$ is also convex, then the set defined previously $\mathcal{D}_{\mathcal{C}} \equiv \mathbb{R}^2$, meaning that the* $\operatorname*{argmin}_{\boldsymbol{y} \in \mathcal{C}} \|\boldsymbol{y} - \boldsymbol{x}\|$ *is always unique.*

Proof. Suppose that there exist $\boldsymbol{y}_1 \neq \boldsymbol{y}_2 \in \mathcal{C}$ s.t. $\forall \boldsymbol{y} \neq \boldsymbol{y}_1, \boldsymbol{y}_2, \;\; \|\boldsymbol{y} - \boldsymbol{x}\| > \|\boldsymbol{y}_1 - \boldsymbol{x}\| = \|\boldsymbol{y}_2 - \boldsymbol{x}\|$. Then consider the midpoint between $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$: $\boldsymbol{z} = \frac{\boldsymbol{y}_1 + \boldsymbol{y}_2}{2}$. It holds:

$$\begin{aligned} \|\boldsymbol{y}_1 - \boldsymbol{x}\|^2 &= \|\boldsymbol{y}_1 - \boldsymbol{z} + \boldsymbol{z} - \boldsymbol{x}\|^2 \\ &= \|\boldsymbol{y}_1 - \boldsymbol{z}\|^2 + \|\boldsymbol{z} - \boldsymbol{x}\|^2 + 2(\boldsymbol{y}_1 - \boldsymbol{z})^\top (\boldsymbol{z} - \boldsymbol{x}) \end{aligned} \tag{4.3}$$

Since $\|\boldsymbol{y}_1 - \boldsymbol{x}\| = \|\boldsymbol{y}_2 - \boldsymbol{x}\|$, $\boldsymbol{z}$ results to be the median of an isosceles triangle of vertices $\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{x}$ and it immediately follows that $\boldsymbol{y}_1 - \boldsymbol{z}$ is orthogonal to $\boldsymbol{z} - \boldsymbol{x}$, which is the altitude of the triangle, so the last inner product is zero. Since clearly $\boldsymbol{z} \neq \boldsymbol{y}_1$, then it follows $\|\boldsymbol{z} - \boldsymbol{x}\| < \|\boldsymbol{y}_1 - \boldsymbol{x}\|$. By the definition of argmin, $\boldsymbol{z}$ cannot be in $\mathcal{C}$, but it cannot be outside $\mathcal{C}$ either since $\boldsymbol{z}$ is a convex combination of elements of $\mathcal{C}$, and we assumed $\mathcal{C}$ convex. We conclude that the supposition $\exists \boldsymbol{y}_1 \neq \boldsymbol{y}_2$ is false. □

Proposition 4.2. *If $\mathcal{C}$ is convex, then the projection operator to $\mathcal{C}$, $\boldsymbol{P}_\mathcal{C} : \mathbb{R}^2 \to \mathcal{C}$ is uniformly continuous.*

Proof. We prove the contrapositive: if $\mathcal{P}_\mathcal{C}$ is not uniformly continuous then $\mathcal{C}$ is not convex. If $\mathcal{P}_\mathcal{C}$ is not uniformly continuous, it means that:

$$\begin{gathered} \exists \varepsilon > 0 \text{ s.t. } \forall \delta > 0\ \exists \boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^2 \text{ s.t.:} \\ \|\boldsymbol{x}_1 - \boldsymbol{x}_2\| < \delta \text{ and } \|\mathcal{P}_\mathcal{C}(\boldsymbol{x}_1) - \mathcal{P}_\mathcal{C}(\boldsymbol{x}_2)\| > \varepsilon \end{gathered} \tag{4.4}$$

Note that it must be $\boldsymbol{x}_i \notin \mathcal{C}$, $i = 1, 2$, otherwise $\mathcal{P}_\mathcal{C}(\boldsymbol{x}_i) = \boldsymbol{x}_i$ and trivially $\nexists \varepsilon > 0$. From now on, for convenience, we will use the shorthand notation $\boldsymbol{y}_i = \mathcal{P}_\mathcal{C}(\boldsymbol{x}_i)$, $i = 1, 2$. Consider the point given by the convex combination: $\boldsymbol{z} = \frac{\boldsymbol{y}_1 + \boldsymbol{y}_2}{2}$. First, it is immediate to notice the following relations:

$$\begin{gathered} \|\boldsymbol{y}_1 - \boldsymbol{z}\| = \|\boldsymbol{y}_2 - \boldsymbol{z}\| \\ \|\boldsymbol{y}_1 - \boldsymbol{z}\| + \|\boldsymbol{y}_2 - \boldsymbol{z}\| = \|\boldsymbol{y}_1 - \boldsymbol{y}_2\| \end{gathered} \tag{4.5}$$

and $\|\boldsymbol{y}_i - \boldsymbol{z}\| > \frac{1}{2}\varepsilon > 0$, for $i = 1, 2$. Call:

$$c_i := \|\boldsymbol{z} - \boldsymbol{x}_i\|, \quad i = 1, 2, \quad \bar{c} = \min\{c_1, c_2\} \tag{4.6}$$

Suppose that

$$\|\boldsymbol{y}_i - \boldsymbol{x}_i\| \leq c, \tag{4.7}$$

for $i = 1, 2$ simultaneously. By looking at the triangles $\boldsymbol{y}_i, \boldsymbol{x}_i, \boldsymbol{z}$, for $i = 1, 2$, it follows:

$$\begin{aligned} c_1 &\leq \|\boldsymbol{y}_1 - \boldsymbol{x}_1\| + \|\boldsymbol{y}_1 - \boldsymbol{z}\| \\ c_2 &\leq \|\boldsymbol{y}_2 - \boldsymbol{x}_2\| + \|\boldsymbol{y}_2 - \boldsymbol{z}\| \end{aligned} \tag{4.8}$$

Then, by (4.7) and the definition of $\bar{c}$:

$$\begin{aligned} \|\boldsymbol{y}_2 - \boldsymbol{x}_2\| \leq \bar{c} \leq c_1 \leq \|\boldsymbol{y}_1 - \boldsymbol{x}_1\| + \|\boldsymbol{y}_1 - \boldsymbol{z}\| \\ \|\boldsymbol{y}_1 - \boldsymbol{x}_1\| \leq \bar{c} \leq c_2 \leq \|\boldsymbol{y}_2 - \boldsymbol{x}_2\| + \|\boldsymbol{y}_2 - \boldsymbol{z}\| \end{aligned} \tag{4.9}$$

Since $\|\boldsymbol{y}_i - \boldsymbol{z}\| > 0$, $i = 1, 2$:

$$\begin{aligned} \|\boldsymbol{y}_2 - \boldsymbol{x}_2\| < \|\boldsymbol{y}_1 - \boldsymbol{x}_1\| \\ \|\boldsymbol{y}_1 - \boldsymbol{x}_1\| < \|\boldsymbol{y}_2 - \boldsymbol{x}_2\| \end{aligned} \tag{4.10}$$

which is not possible, and this implies that (4.7) is false. This means that $\|\boldsymbol{y}_1 - \boldsymbol{x}_1\| > \bar{c} \ \vee \ \|\boldsymbol{y}_2 - \boldsymbol{x}_2\| > \bar{c}$, i.e. at least for one i and $\exists j$ it holds:

$$\|\boldsymbol{z} - \boldsymbol{x}_i\| < \|\boldsymbol{x}_j - \boldsymbol{y}_j\| \tag{4.11}$$

Then, since by definition $\forall \boldsymbol{y} \in \mathcal{C} \|\boldsymbol{y} - \boldsymbol{x}_i\| \geq \|\boldsymbol{y}_i - \boldsymbol{x}_i\|$ it follows $\boldsymbol{z} \notin \mathcal{C}$, and being $\boldsymbol{z}$ a convex combination of points in $\mathcal{C}$, we conclude that $\mathcal{C}$ is not convex.

$\square$

Proposition 4.3. *Given a closed set $\mathcal{C} \subset \mathbb{R}^2$ and a point $\boldsymbol{x} \in \mathbb{R}^2$, $\boldsymbol{x} \notin \mathcal{C}$, it holds:*

$$\operatorname*{argmin}_{\boldsymbol{y} \in \mathcal{C}} \|\boldsymbol{y} - \boldsymbol{x}\| \subset \partial\mathcal{C} \tag{4.12}$$

Proof. Suppose $\boldsymbol{y}$ belongs to the interior of $\mathcal{C}$, i.e.:

$$\exists \mathcal{B}_\varepsilon(\boldsymbol{y}) \text{ s.t. } \mathcal{B}_\varepsilon(\boldsymbol{y}) \cap \mathcal{C} = \mathcal{B}_\varepsilon(\boldsymbol{y}), \tag{4.13}$$

where $\mathcal{B}_\varepsilon(\boldsymbol{y}) := \{\boldsymbol{x} \in \mathbb{R}^2 \ : \ \|\boldsymbol{x} - \boldsymbol{y}\| < \varepsilon\}$.

Consider the segment:

$$\mathcal{R} = \{\lambda \boldsymbol{y} + (1 - \lambda)\boldsymbol{x}, \lambda \in [0, 1]\} \tag{4.14}$$

and its intersection with the open ball $\mathcal{B}_\varepsilon(\boldsymbol{y})$:

$$\mathcal{R} \cap \mathcal{B}_\varepsilon = \{\lambda \boldsymbol{y} + (1-\lambda)\boldsymbol{x}, \quad \lambda \in [0, \varepsilon)\} \tag{4.15}$$

Those are points in the interior of $\mathcal{C}$, whose distance with $\boldsymbol{x}$ is such that:

$$\|\lambda \boldsymbol{y} + (1-\lambda)\boldsymbol{x} - \boldsymbol{x}\| = \lambda\|\boldsymbol{y} - \boldsymbol{x}\| < \|\boldsymbol{y} - \boldsymbol{x}\| \tag{4.16}$$

for all $0 < \lambda < \min \varepsilon, 1$, in conflict with the definition of $\boldsymbol{y}$. □

Definition 4.2. *We define a* configuration space *as the partition of the plane $\mathbb{R}^2$ into two complementary subsets $\mathcal{C}_{coll}$ and $\mathcal{C}_{free}$ such that $\mathcal{C}_{coll}$ is not necessarily convex, but composed of a finite union of convex sets, with the following properties:*

- $\mathcal{C}_{coll} = \bigcup_i \mathcal{C}_{coll}^{(i)}$
- $\forall \boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathcal{C}_{coll}^{(i)}, \quad (1-\lambda)\boldsymbol{x}_1 + \lambda \boldsymbol{x}_2 \in \mathcal{C}_{coll}^{(i)}, \quad \forall \lambda \in [0, 1]$
- $\mathcal{C}_{coll}^{(i)} \cap \mathcal{C}_{coll}^{(j)} = \partial\mathcal{C}_{coll}^{(i)} \cap \partial\mathcal{C}_{coll}^{(j)}, \quad \forall i \neq j$

and we call $\mathcal{C}_{coll}^{(i)}$ the i-th convex component *of $\mathcal{C}_{coll}$.*

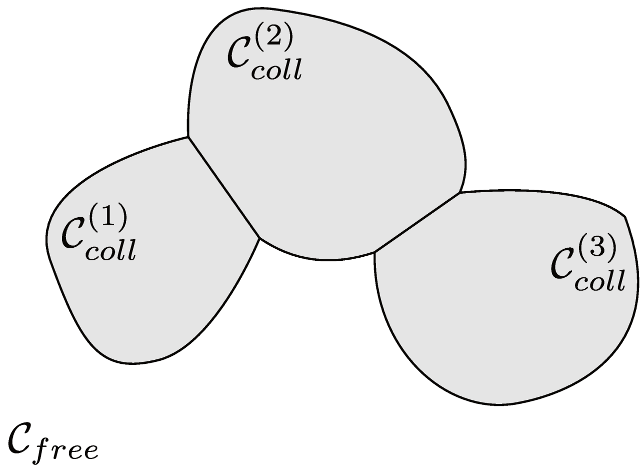

$\mathcal{C}_{coll}^{(2)}$
$\mathcal{C}_{coll}^{(1)}$
$\mathcal{C}_{coll}^{(3)}$
$\mathcal{C}_{free}$

Figure 4.1.: Example of configuration space.

Note that:

- A finite union of convex sets can approximate arbitrarily well any configuration of practical interest.

- The last property means that the convex components are either connected components or subsets of connected components that only share a part of the boundary set.

- Given any set $\mathcal{C}_{coll}$, if such decomposition exists, there is an infinite number of ways such components can be defined. Even if we could consider the decomposition that minimizes the number of components, in this discussion their number and exact definition is not relevant as long as the mentioned properties are fulfilled. For this reason we will provide all definition with a generic i, neglecting the maximum number that i can assume.

Definition 4.3. *Given a generic set $\mathcal{C}$, decomposable into convex components, s.t. $\mathcal{C} = \bigcup_i \mathcal{C}^{(i)}$, the relative* Generalized Voronoi Tessellation *is identified by the open sets:*

$$\mathcal{X}_{\mathcal{C}}^{(i)} = \{\boldsymbol{x} \in \mathbb{R}^2 \ : \ \mathrm{dist}(\boldsymbol{x}, \mathcal{C}^{(i)}) < \mathrm{dist}(\boldsymbol{x}, \mathcal{C}^{(j)}), \ \forall j\} \tag{4.17}$$

with the classical definition of distance to a set $\mathrm{dist}\, \boldsymbol{x}, \mathcal{X} = \underset{\boldsymbol{y} \in \mathcal{C}}{\mathrm{argmin}} \, \|\boldsymbol{y} - \boldsymbol{x}\|$. *The* Generalized Voronoi Diagram *is instead the union of their boundaries:*

$$\mathcal{X}_{\mathcal{C}} = \bigcup_i \partial \mathcal{X}_{\mathcal{C}}^{(i)} \tag{4.18}$$

Note that $\mathcal{X}_{\mathcal{C}}$ has no interior points, being a finite union of boundary sets. Following the classical notation we consider the Voronoi nodes as the points of the GVD that are equidistant to three or more convex components, and then we introduce the notation for the Voronoi branches as the paths connecting the nodes. Formally:

Definition 4.4. *We define a* generalized Voronoi node *as the points:*

$$\boldsymbol{x} \in \mathcal{X}_{\mathcal{C}} \ : \ \exists i, j, k, \ \textit{s.t.} \ \mathrm{dist}(\boldsymbol{x}, \mathcal{C}^{(i)}) = \mathrm{dist}(\boldsymbol{x}, \mathcal{C}^{(j)}) = \mathrm{dist}(\boldsymbol{x}, \mathcal{C}^{(k)}) \tag{4.19}$$

The Generalized Voronoi branches *are then the largest contiguous opens sets of the GVD that do not contain nodes.*

4.2.1 Landscape function generalization

We now proceed by providing a more general definition the Landscape function, that applies to both the ideal ground truth and the measurement set, by considering a generic set $\mathcal{A}$ of collision points, which can be the entire collision space $\mathcal{C}_{coll}$ (ideal case) or the finite subsample $\mathcal{M}$ of the real measurement. In this chapter, for the purposes of the discussion, we will neglect the dependency on time and samples, as we are considering each time sample individually. For $\mathcal{A}$, in both cases, we consider its decomposition in convex components $\mathcal{A}^{(i)}$.

Definition 4.5. *We consider a* peak function *as:*

$$\Gamma : \mathbb{R}^2 \times \mathbb{R}^2 \to (0,1] : (\boldsymbol{x}, \boldsymbol{y}) \mapsto e^{-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2\sigma^2}}, \tag{4.20}$$

with $\sigma > 0$ being the tuning parameter of the peak radius.

Definition 4.6. *The* raw landscape function *of $\mathcal{A}$ is defined as the superposition of Gaussian peaks:*

$$\check{\mathcal{L}}_{\mathcal{A}}(\boldsymbol{x}) = \max_{\boldsymbol{y} \in \mathcal{A}} \Gamma(\boldsymbol{x}, \boldsymbol{y}). \tag{4.21}$$

Definition 4.7. *The* smooth landscape function *of $\mathcal{A}$ is:*

$$\mathcal{L}_{\mathcal{A}} = \int_{\mathbb{R}^2} \check{\mathcal{L}}_{\mathcal{A}}(\boldsymbol{w}) \kappa(\boldsymbol{x} - \boldsymbol{w}) \, \mathrm{d}\boldsymbol{w}, \tag{4.22}$$

that is the convolution of the raw landscape function with a Gaussian kernel $\kappa(\boldsymbol{x}) = \bar{\kappa} e^{-\frac{\|\boldsymbol{x}\|^2}{2\delta^2}}$, of radius $\delta > 0$ and gain $\bar{\kappa}$.

The gain $\bar{\kappa}$ is chosen in order to compensate the amplification of the convolution, and it is designed such that if we smooth a single gaussian peak, the resulting function has the same maximum value in the peak, i.e.:

$$\begin{aligned}\Gamma(\boldsymbol{x},\boldsymbol{y})\Big|_{\boldsymbol{x}=\boldsymbol{y}} &= \int_{\mathbb{R}^2} \Gamma(\boldsymbol{w},\boldsymbol{y})\kappa(\boldsymbol{x}-\boldsymbol{w})\,\mathrm{d}\boldsymbol{w}\Big|_{\boldsymbol{x}=\boldsymbol{y}} \\ 1 &= \int_{\mathbb{R}^2} e^{-\frac{\|\boldsymbol{w}-\boldsymbol{y}\|^2}{2\sigma^2}} \bar{\kappa} e^{-\frac{\|\boldsymbol{y}-\boldsymbol{w}\|^2}{2\delta^2}}\,\mathrm{d}\boldsymbol{w} \\ &= \bar{\kappa}\int_{\mathbb{R}^2} e^{-\frac{1}{2}\frac{(\delta^2+\sigma^2)}{\sigma^2\delta^2}\|\boldsymbol{w}\|^2}\end{aligned} \tag{4.23}$$

By using the result on the Gaussian integral and solving for $\bar{\kappa}$ we obtain:

$$\bar{\kappa} = \frac{\sigma^2+\delta^2}{2\pi\sigma^2\delta^2}. \tag{4.24}$$

The projection operator as in Def. 4.1, allows us to see a straightforward but interesting property of the raw landscape function

Proposition 4.4. *Given a point $\boldsymbol{x}$ and its projections onto each convex components of $\mathcal{A}$, $\boldsymbol{P}_{\mathcal{A}^{(i)}}$, the raw landscape function can be expressed as:*

$$\check{\mathcal{L}}_{\mathcal{A}}(\boldsymbol{x}) = \max_i \Gamma(\boldsymbol{x}, \boldsymbol{P}_{\mathcal{A}^{(i)}}(\boldsymbol{x})) \tag{4.25}$$

Consider the GVD of $\mathcal{A}$, $\mathcal{X}_{\mathcal{A}}^{(i)}$. Then it also holds, for $\boldsymbol{x} \in \mathcal{X}_{\mathcal{A}}^{(i)}$:

$$\check{\mathcal{L}}_{\mathcal{A}}(\boldsymbol{x}) = \Gamma(\boldsymbol{x}, \boldsymbol{P}_{\mathcal{A}^{(i)}}(\boldsymbol{x})) \tag{4.26}$$

The second relation tells that, within each Voronoi region, the Landscape value is associated with the distance to the closest convex component.

Proof. The key element to show these relations is that, by definition, if $\|\boldsymbol{x}-\boldsymbol{y}_1\| > \|\boldsymbol{x}-\boldsymbol{y}_2\|$ then $\Gamma(\boldsymbol{x},\boldsymbol{y}_1) < \Gamma(\boldsymbol{x},\boldsymbol{y}_2)$. From this, it is clear that the only points that can maximize Γ are the point of $\mathcal{A}$ closest to $\boldsymbol{x}$: by definition, for each i, it must be the projection of $\boldsymbol{x}$. The second statement then follows trivially by the definition of $\mathcal{X}_{\mathcal{A}}^{(i)}$. $\square$

Before introducing the main result, need to understand the conditions for continuity and differentiability of the landscape functions.

Proposition 4.5. *The raw Landscape function $\check{\mathcal{L}}_{\mathcal{A}}(\boldsymbol{x})$ is continuous in all $\mathbb{R}^2$.*

Proof. From Prop. 4.4, for $\boldsymbol{x} \in \mathcal{X}_{\mathcal{A}}^{(i)}$, the landscape function is given by the peak function, that is continuous since it is uniformly continuous. We have to show that it is continuous also for $\boldsymbol{x} \in \partial\mathcal{X}^{(i)}$. We consider all the possible converging sequences $\bar{\boldsymbol{x}}_n \to \boldsymbol{x}$, that will have subsequences $\bar{\boldsymbol{x}}_n^{(i)} \in \mathcal{X}_{\mathcal{A}}^{(i)}$, $\bar{\boldsymbol{x}}_n^{(j)} \in \mathcal{X}_{\mathcal{A}}^{(j)}$ and $\bar{\boldsymbol{x}}_n^{\mathcal{X}} \in \mathcal{X}_{\mathcal{A}}$. For $\bar{\boldsymbol{x}}_n^{(i)}$ and $\bar{\boldsymbol{x}}_n^{(j)}$ the uniform continuity of $\mathcal{P}_{\mathcal{C}}$ ensures that $\check{\mathcal{L}}_{\mathcal{A}}(\bar{\boldsymbol{x}}_n) \to \check{\mathcal{L}}_{\mathcal{A}}(\boldsymbol{x})$. From the definition, it immediately holds $\check{\mathcal{L}}_{\mathcal{A}}(\bar{\boldsymbol{x}}_n^{\mathcal{X}}) = \check{\mathcal{L}}_{\mathcal{A}}(\boldsymbol{x})$, $\forall n$, so in this case the convergence is trivial. For all the other subsequences that jump between the three sets, it is possible to extract subsequences that are all entirely contained in one of those sets, each of which converge to $\check{\mathcal{L}}_{\mathcal{A}}(\boldsymbol{x})$. □

Proposition 4.6. *The raw landscape function $\check{\mathcal{L}}_{\mathcal{A}}(\boldsymbol{x})$ is piecewise differentiable within each $\mathcal{X}_{\mathcal{A}}^{(i)}$.*

Proof. From Prop. 4.4, for $\boldsymbol{x} \in \mathcal{X}_{\mathcal{A}}^{(i)}$, the landscape function is given by the peak function of the projection of $\boldsymbol{x}$, and it immediately holds:

$$\frac{\partial}{\partial \boldsymbol{x}} \check{\mathcal{L}}_{\mathcal{A}} = \frac{\partial}{\partial \boldsymbol{x}} \Gamma(\boldsymbol{x}, \boldsymbol{P}_{\mathcal{A}^{(i)}}(\boldsymbol{x})) \tag{4.27}$$

The peak function $\Gamma(\boldsymbol{x}, \boldsymbol{y})$ is trivially differentiable with respect to $\boldsymbol{x}$ for all $\boldsymbol{x} \in \mathbb{R}^2$, and it holds;

$$\frac{\partial}{\partial \boldsymbol{x}} \Gamma(\boldsymbol{x}, \boldsymbol{y}) = -\frac{\boldsymbol{x} - \boldsymbol{y}}{2\sigma^2} e^{-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2\sigma^2}} \tag{4.28}$$

The gradient of $\check{\mathcal{L}}_{\mathcal{A}}$ can then be seen as the composition of $\frac{\partial}{\partial \boldsymbol{x}} \Gamma(\boldsymbol{x}, \boldsymbol{y})$ and the function $\boldsymbol{g}(\boldsymbol{x}) := [\boldsymbol{x},\ \boldsymbol{P}_{\mathcal{A}^{(i)}}(\boldsymbol{x})]^\top$ and it is continuous as composition of continuous functions since Γ is differentiable and $\boldsymbol{P}_{\mathcal{A}^{(i)}}(\boldsymbol{x})$ is uniformly continuous in $\mathcal{X}_{\mathcal{A}^{(i)}}$, from Prop. 4.2. □

Notably, $\frac{\partial}{\partial \boldsymbol{x}} \check{\mathcal{L}}_{\mathcal{A}}(\boldsymbol{x})$ is not defined for $\boldsymbol{x} \in \partial\mathcal{X}^{(i)}$, $\forall i$, since, by definition of $\mathcal{X}^{(i)}$, there are two different convex components $\mathcal{A}^{(i)}$ that have the same distance from $\boldsymbol{x}$, so the argmax is not unique.

Proposition 4.7. *The smooth Landscape function $\mathcal{L}_{\mathcal{A}}(\boldsymbol{x})$ is differentiable in $\mathbb{R}^2$.*

Proof. The proof is straightforward by noticing that the variable $\boldsymbol{x}$ in the definition is only in the kernel:

$$\frac{\partial}{\partial \boldsymbol{x}}\mathcal{L}_{\mathcal{A}}(\boldsymbol{x}) = \int_{\mathbb{R}^2} \check{\mathcal{L}}_{\mathcal{A}}(\boldsymbol{w})\frac{\partial}{\partial \boldsymbol{x}}\kappa(\boldsymbol{x}-\boldsymbol{w})\,\mathrm{d}\boldsymbol{w} \tag{4.29}$$

and $\kappa(\boldsymbol{x})$ is differentiable everywhere. □

4.3 The NAPVIG principle theorem

We now introduce the main results that prove that every sample computed by the NAPVIG algorithm is actually an approximation of the GVD. The proof is provided in the ideal collision space $\mathcal{C}_{coll}$ and the raw landscape function. The local minimum considered in the algorithm is exactly a point of the GVD, provided that some theoretical assumptions are met. In the following, for notation convenience, we will refer to $\check{\mathcal{L}}_{\mathcal{C}_{coll}}$ as $\check{\mathcal{L}}$.

Theorem 4.1 (NAPVIG Principle)**.** *Given a configuration space characterized by collision space $\mathcal{C}_{coll}$, consider the raw landscape function $\check{\mathcal{L}}(\boldsymbol{x})$. Given a pair of initial point and a direction $(\boldsymbol{x},\boldsymbol{d}) \in \mathbb{R}^2 \times \mathbb{S}^1$ identifying the line $\mathcal{D} = \{\boldsymbol{x}+\lambda\boldsymbol{d},\ \lambda \in \mathbb{R}\}$. Consider the set of local minima of the restricted function $\check{\mathcal{L}}\big|_{\mathcal{D}}$:*

$$\mathcal{X}^* = \left\{\underset{\bar{\boldsymbol{x}}\in\mathcal{D}}{\operatorname{argmin}}\ \check{\mathcal{L}}(\bar{\boldsymbol{x}})\right\} \tag{4.30}$$

Under the hypothesis that the following relation holds:

$$\boldsymbol{d} \not\perp \boldsymbol{x} - \boldsymbol{P}_{\mathcal{C}_{coll}(\boldsymbol{x})}, \tag{4.31}$$

then the elements of $\mathcal{X}^*$ are all points of the Generalized Voronoi Diagram of $\mathcal{C}_{coll}$.

Note that the elements of the initialization pair $(\boldsymbol{x},\boldsymbol{d})$ correspond in the NAPVIG algorithm the first step $\boldsymbol{x}^{(0)}$ and the direction of the line $\mathcal{R}^\top$, so $\boldsymbol{d}$ is orthogonal to $\boldsymbol{r}_0$, i.e. $\boldsymbol{x} = \boldsymbol{x}_0 + \beta_{step}\boldsymbol{r}_0$ and $\boldsymbol{d} \perp \boldsymbol{r}_0$. The notation used in the theorem is for clarity of presentation only.

Proof. Consider a local minimum $\boldsymbol{x}^*$ of the restriction $\check{\mathcal{L}}\big|_{\mathcal{D}}$. Suppose by contradiction that $\boldsymbol{x}^*$ is not a point of the GVD. This entails that:

$$\operatorname{dist}(\boldsymbol{x}^*, \mathcal{X}^{(i)}) < \operatorname{dist}(\boldsymbol{x}^*, \mathcal{X}^{(j)}), \quad \exists i, \forall j \neq i \tag{4.32}$$

which in turn implies that $\boldsymbol{x}^* \in \mathcal{X}^{(i)}$. As a consequence, it holds:

$$\begin{aligned} \check{\mathcal{L}}(\boldsymbol{x}^*) &= \exp\left(-\frac{\boldsymbol{x}^* - \boldsymbol{P}_{\mathcal{C}^{(i)}_{coll}}(\boldsymbol{x}^*)}{2\sigma^2}\right) \\ \frac{\partial}{\partial \boldsymbol{x}}\check{\mathcal{L}}(\boldsymbol{x}^*) &= -\frac{\boldsymbol{x}^* - \boldsymbol{P}_{\mathcal{C}^{(i)}_{coll}}(\boldsymbol{x}^*)}{2\sigma^2}\exp\left(-\frac{\boldsymbol{x}^* - \boldsymbol{P}_{\mathcal{C}^{(i)}_{coll}}(\boldsymbol{x}^*)}{2\sigma^2}\right) \end{aligned} \tag{4.33}$$

Consider now the following parametrization of the restriction of $\check{\mathcal{L}}$ to $\mathcal{D}$:

$$\check{\mathcal{L}}(\boldsymbol{x})\Big|_{\mathcal{D}} = \check{\mathcal{L}}(\boldsymbol{x}^* + \lambda \boldsymbol{d}), \quad \lambda \in \mathbb{R} \tag{4.34}$$

Since $\boldsymbol{x}^*$ is a minimum and $\check{\mathcal{L}}$, from Prop. 4.6, is differentiable in $\boldsymbol{x}^*$, since $\boldsymbol{x}^* \in \mathcal{X}^{(i)}$, the derivative of the restriction is zero in $\boldsymbol{x}^*$ and it coincides with the directional derivative of $\check{\mathcal{L}}$ along $\boldsymbol{d}$, namely:

$$0 = \frac{\partial}{\partial \lambda}\check{\mathcal{L}}(\boldsymbol{x}^* + \lambda \boldsymbol{d}) = \frac{\partial}{\partial \boldsymbol{x}}\check{\mathcal{L}}(\boldsymbol{x}^*)^\top \boldsymbol{d} \tag{4.35}$$

This implies:

$$-\frac{\left(\boldsymbol{x}^* - \boldsymbol{P}_{\mathcal{C}^{(i)}_{coll}}(\boldsymbol{x}^*)\right)^\top \boldsymbol{d}}{2\sigma^2}\exp\left(-\frac{\boldsymbol{x}^* - \boldsymbol{P}_{\mathcal{C}^{(i)}_{coll}}(\boldsymbol{x}^*)}{2\sigma^2}\right) = 0 \tag{4.36}$$

The scalar part is always positive and, since by hypothesis $\boldsymbol{d}$ is not orthogonal to $\boldsymbol{x}^* - \boldsymbol{P}(\boldsymbol{x}^*)$), the numerator $\left(\boldsymbol{x}^* - \boldsymbol{P}_{\mathcal{C}^{(i)}_{coll}}(\boldsymbol{x}^*)\right)^\top \boldsymbol{d} \neq 0$, so this equality cannot be true an this proves the theorem. $\square$

This theorem is shown considering the ground truth $\mathcal{C}_{coll}$ and the raw landscape function. However, two following remarks show that the real case is a reasonable approximation of the ideal scenario. First, a real sensor can measure a noisy sample only on the border of $\mathcal{C}_{coll}$, but the useful points in the algorithm are indeed points of the border. Second, we note that the smooth landscape function is close to the raw version, and for $\delta \to 0$ (limit case) they coincide. Formally, the following proposition holds.

Proposition 4.8. *For any $\mathcal{A}$ and any $\boldsymbol{x} \in \mathbb{R}^2$:*

$$\lim_{\delta \to 0} \mathcal{L}_{\mathcal{A}}(\boldsymbol{x}, \delta) = \check{\mathcal{L}}_{\mathcal{A}}(\boldsymbol{x}) \tag{4.37}$$

Proof. The kernel $\kappa(\boldsymbol{x})$ can be seen as a Gaussian probability density function (PDF) with variance δ^2, up to a scaling factor. It is known that for $\delta \to 0$, the PDF converges to a Dirac delta, so we obtain:

$$\lim_{\delta \to 0} \mathcal{L}_{\mathcal{A}}(\boldsymbol{x}, \delta) = \int_{\mathbb{R}^2} \check{\mathcal{L}}_{\mathcal{A}}(\boldsymbol{w}) \mathfrak{d}(\boldsymbol{w} - \boldsymbol{x}) \, \mathrm{d}\boldsymbol{w} \tag{4.38}$$

where $\mathfrak{d}(\boldsymbol{x})$ is the Dirac delta: its sampling property then proves the proposition. □

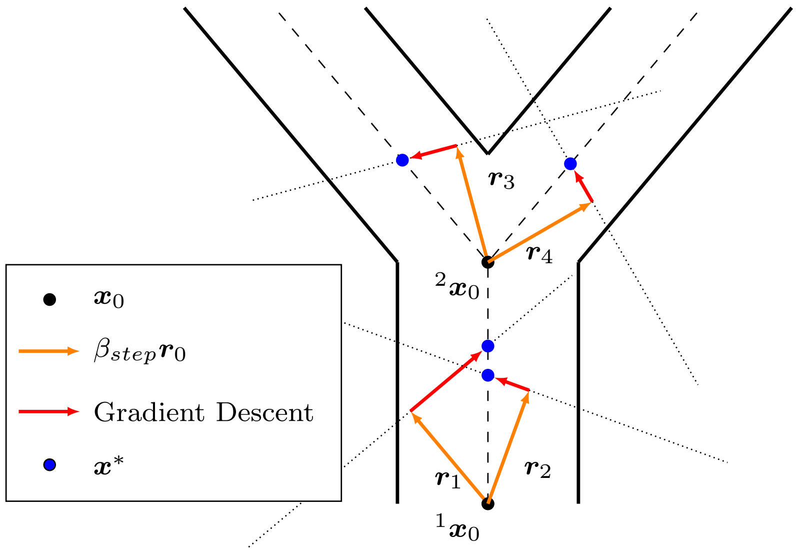


Figure 4.2.: Example of the algorithm behavior in a single corridor and in presence of a bifurcation. The walls are depicted in solid black, with the center of the corridors (corresponding to the GVD) in dashed. The orange arrows are the first step taken in the search direction, while the red one represents the trend of the gradient descent algorithm from the initial point to the output of the algorithm.

4.3.1 Remarks on the theorem

The results provided in this chapter allows a better understanding of the behavior of the NAPVIG algorithm. However, several points needs to be addressed

Hypothesis 4.31, which we refer to as *NAPVIG condition*, is an important aspect as the validity of the theorem depends on it. In practice, it means that, from $\boldsymbol{x}$, all search directions $\boldsymbol{d}$ are available except the one that is orthogonal to the direction of the closest collision point. This assumption not only is reasonable since it is only one direction out of all the possibilities, but it highlights a method to understand that a direction is wrong, since it would drive directly to the closest obstacle. Also, numerically, in this condition the projection of the gradient onto the search direction performed by the algorithm would result in a null gradient, making the optimization algorithm ineffective.

For a pair $(\boldsymbol{x}, \boldsymbol{d})$, corresponding to the NAPVIG algorithm initial conditions $(\boldsymbol{x}_0, \boldsymbol{r}_0)$, there could be multiple local minima, as explicitly considered by the theorem. In the practical implementation the algorithm, based on a gradient method, will converge to a specific one according to the initialization, so its choice is critical for the purpose of the navigation strategy. In Figure 4.2 we present an example of a corridor with a bifurcation and the outputs of the algorithm given two different pairs of initial position and search direction. Intuitively, $\boldsymbol{r}$ indicates the direction towards which it is desired to advance and compute the next point of the GVD to follow. For example, ${}^1\boldsymbol{x}_0$ is inside a single corridor, so the two different search direction $\boldsymbol{r}_{1,2}$ yield points that are slightly different but on the same GVD branch. Conversely, for ${}^2\boldsymbol{x}_0$, in proximity of the bifurcation, the two initial directions $\boldsymbol{r}_{3,4}$ yield points on different corridors. Given that the map is unknown to the agent, it is not straightforward to understand which initialization will drive towards the desired goal and different strategies in the choice of $\boldsymbol{x}_0$ and $\boldsymbol{r}$ can be employed according to the specific case requirements. A simple strategy, with reference to Figure 4.2 could be to evaluate the results with both $\boldsymbol{r}_3$ and $\boldsymbol{r}_4$ and then chose to follow the point closer to a goal location.

4.3.2 Smoother peaks merging

The smoothing of the Landscape function not only allows the optimization problem to be solved with gradient methods, but depending on the smoothing radius two or more peaks can be merged into a single peak, removing the local minimum in between them. As a result, conceptually, in the smooth Landscape function all the measurements that are close enough are implicitly and automatically consider as a single entity. We now provide a simple analysis of the 1D analogous, in order to understand qualitatively the conditions under which the two peaks are merged.

Consider two peaks centered in $\bar{x}_i \in \mathbb{R}$, $i = 1, 2$,

$$\Gamma_i(x) = \Gamma(x, x_i) = e^{-\frac{(x-\bar{x}_i)^2}{2\sigma^2}} \tag{4.39}$$

reported in black in Fig. 4.3, along with their superposition, in blue, the raw landscape function:

$$\check{\mathcal{L}}(x) = \max_{i=1,2} \Gamma_i(x) \tag{4.40}$$

We then consider the smooth function, the result of the convolution with the Gaussian kernel of radius $\delta > 0$:

$$\mathcal{L}_\delta(x) = \int_{-\infty}^{+\infty} \check{\mathcal{L}}(w) e^{-\frac{(x-w)^2}{2\delta^2}}\, \mathrm{d}w \tag{4.41}$$

where we highlighted the dependency of $\mathcal{L}$ on δ. For simplicity, we consider $x_1 = r$ and $x_2 = -r$. The integral is then:

$$\mathcal{L}_\delta(x) = \int_{-\infty}^{0} e^{-\frac{(w+r)^2}{2}-\frac{(x-w)^2}{2\delta^2}}\, \mathrm{d}w + \int_{0}^{\infty} e^{-\frac{(w-r)^2}{2}-\frac{(x-w)^2}{2\delta^2}}\, \mathrm{d}w, \tag{4.42}$$

This expression can be solved in closed form, which we report for completeness:

$$\mathcal{L}_\delta(x) = \frac{\sqrt{2}\sqrt{\pi}\delta\left(\left(\operatorname{erf}\left(\frac{\sqrt{2}\left(\delta^2 r - x\right)}{2\delta\sqrt{\delta^2+1}}\right)+1\right)e^{-\frac{r^2+2rx+x^2}{2\left(\delta^2+1\right)}} + \left(\operatorname{erf}\left(\frac{\sqrt{2}\left(\delta^2 r + x\right)}{2\delta\sqrt{\delta^2+1}}\right)+1\right)e^{\frac{-r^2+2rx-x^2}{2\left(\delta^2+1\right)}}\right)}{2\sqrt{\delta^2+1}} \tag{4.43}$$

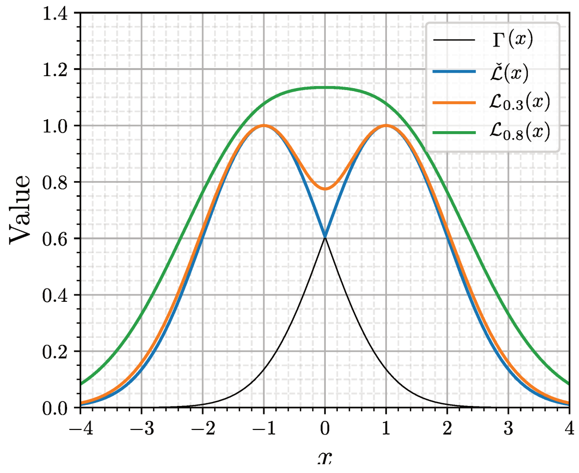


Figure 4.3.: Superimposition of gaussians centered in ± 1, smoothed with non-merging and merging kernel radius

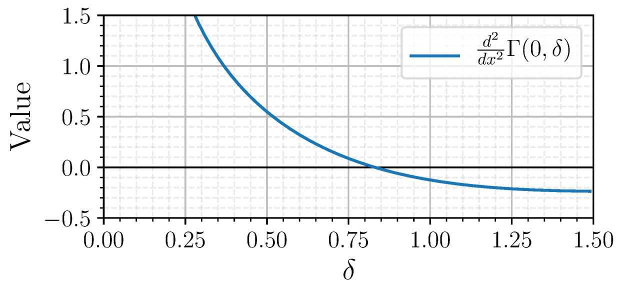


Figure 4.4.: Value of second derivative corresponding to stationary point with fixed r and varying δ

It is easy to verify that $\frac{\partial}{\partial x}\mathcal{L}(0) = 0$ for any choice of δ and r, so the origin is either a maximum point or a minimum. In the former case, it means that the two peaks have been merged by the kernel, otherwise the local minimum divides the peaks into two, as depicted in Fig. 4.3.

We are interested in finding the condition under which a kernel of radius δ merges two Gaussians of distance $2r$. Since $\frac{\partial}{\partial x}\mathcal{L}(0) = 0$, by looking at the sign of $\frac{\partial^2}{\partial x^2}\mathcal{L}(x)$ we can understand, for each choice of δ and r, if $x = 0$ is a maximum or a minimum. Consider $\frac{\partial^2}{\partial x^2}(x, r, \delta)$ as a function of its parameters. In $x = 0$ and for any r, we can numerically find δ^* such that $\frac{\partial^2}{\partial x^2}(x, r, \delta^*) = 0$, whose trend is reported in Fig. 4.4. Since $\Gamma''(0, r, \delta)$ is decreasing with δ,

$$\delta^*(r) \simeq \hat{\delta}^*(r) := a_\delta r, \tag{4.44}$$

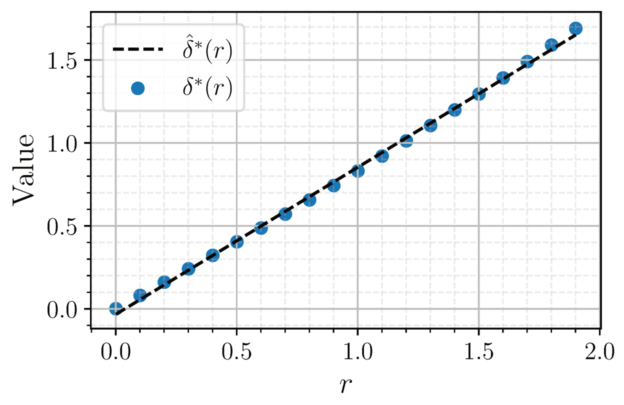


Figure 4.5.: Samples of $\delta^*(r)$ for several values of r w.r.t. linear estimation of $\hat{\delta}^*(r)$

with $a_\delta = 0.8867$ derived with least square fitting. From (4.44) we can also derive the inverse relation

$$\hat{r}^*(\delta) = a_\delta^{-1}\delta \tag{4.45}$$

Summarizing, there exists an almost linear relation between the smoothing radius needed to merge two peaks for a fixed distance.

4.4 Conclusions

In this chapter we provided a theoretical analysis on the properties of the Landscape function, which is the key element enabling the NAPVIG algorithm to precisely compute points in the GVD of the map. Such property is proved mathematically for the ideal case and the real case results to be a reasonable approximation.

5 NAPVIG-X: navigation in generic and unstructured environments

In chapter 3 we studied a scenario in which the goal was to navigate a narrow and cluttered environment with only one possible path. We restricted the scope of our analysis to this limited context in order to test the efficacy of the NAPVIG algorithm in reactively calculating safe trajectories, which in chapter 4 we proved that they correspond to an approximation of the Generalized Voronoi Diagram of the real map.

The initialization of the algorithm, as mentioned previously, affects the point of the GVD that results from the algorithm. Each choice of initial conditions corresponds to a particular branch of the GVD. In this chapter we will present a basic but effective strategies to solve the problem of local planning in generic, narrow and cluttered environments using an exploration/exploitation paradigm.

5.1 Introduction

Classically, robotic exploration is focused on constructing a map of the environment (Ocando et al., 2017), possibly including localization with SLAM methods (Al-Hourani and Ristic, 2020; Liu et al., 2020). To this aim, the most common method is to define the *frontiers*, segments delimiting the unknown and explored areas (Keidar et al., 2012; Quin et al., 2014; Sun et al., 2020), which are often detected with methods based on edge detection and region extraction techniques from computer vision, such as edge detectors and region extraction (Upadhyay et al., 2014). In recent years, reinforcement learning techniques have also been increasingly utilized (Zwecher et al., 2022; Niroui et al., 2019; Li et al., 2019), thanks also to the availability of large datasets of indoor maps (Li et al., 2020).

In this work the concept of exploration is shifted to a more local perspective: the goal is not the full coverage of the free space but the reactive discovery of paths that are leading to the target. This can be interpreted as a strategy to exit local minima in Artificial Potential Fields (APF) methods (Zhu et al., 2010; Bounini et al., 2017): indeed, although substantially different, following the GVD branches could lead to similar problems as local minima since such branches could lead to a dead end. Strategies for *local minima avoidance* (LMA) are very diverse, and they can be bio-inspired (Montiel et al., 2015a; Teimoori and Savkin, 2010) or employ evolutionary methods (Montiel et al., 2015b; Boufera et al., 2018). The latest trends integrate potential fields with reinforcement learning to solve the full stack problem (Yao et al., 2020).

Most of the cited methods are based on maps, often modelled with occupancy grids (Elfes et al., 1990). In this thesis, instead, the approach is to address local navigation without keeping track of a discretized spatial representation of the environment. To make exploration possible, then, we will keep track of tracked areas with *landmarks*. Usually, landmarks are reference points used for localization (Boucher, 2016; Prasad et al., 2020) and can be associated to semantic information. In this context landmarks will not be used for localization but will be associated to an index expressing the amount of information gathered in an area, namely, the regions that have been explored will be linked to a large value, while areas further away from the explored path will carry less or zero information.

This index will serve as a metric for an optimization algorithm that will drive a policy-switching controller. Policy-switching methods are currently utilized to effectively account for the multiple issues that may arise in distinctive situations where a particular approach is inadequate to manage all of them (Amano and Kato, 2022). The concept of policy, in the literature, is strongly linked with reinforcement learning strategies (Tidd et al., 2021; Patel et al., 2021; Guo et al., 2021). More in general, a lot of attention has been dedicated to developing neural network-based approaches for learning policies addressing the general navigation problem in various contexts. For example, (Zhang et al., 2020) combines a graph architecture with deep reinforcement learning to address dynamic environments in presence of multiple external entities; (Devo et al., 2020) propose a visual approach to search and reach a target, based on two networks that pursue these two goals separately. Reactive vision based approaches can be likened to the one proposed in this and the previous chapters, in the sense that a global map is not required to complete the task (Jin et al., 2020; Bektaş and Bozma, 2022). For instance, (Gupta et al., 2017) propose a top-down representation of the environment that stores the information that are actually useful for the trained model, while (Mirowski et al., 2016) directly exploits raw sensor data to perform navigation in complex environments.

In this work we propose a different approach that takes inspiration from the reinforcement learning methodologies but exploits the fast Voronoi approximation obtainable with the NAPVIG algorithm to define a set of algorithmically defined policies that address the problem of local navigation in cluttered and unknown environments. We refer to this algorithm as *NAPVIG-X*, as it is the extension of the NAPVIG algorithm to the broader case of generic maps.

5.2 Problem formulation

The context we consider is similar to that of chapter 3. We consider that the configuration space is characterized by a possibly time-variant collision space $\mathcal{C}_{coll}(t) \subset \mathbb{R}^2$. A ground robot is equipped with a LiDAR that, with sampling period $T_m \in \mathbb{R}$, at time $t_k = kT_m$, $k \in \mathbb{N}$, obtains a measurement set

$$\mathcal{M}_k := \left\{ \boldsymbol{m}_{k,h}, \quad h = 1, \dots, H \right\}, \quad H \in \mathbb{N}, \tag{5.1}$$

where H is the number of measurements per sample. We consider an inertial "world" reference frame $\mathcal{F}_W$, a robot frame $\mathcal{F}_t$ at continuous time $t \in \mathbb{R}$, the *measurement frame* $\mathcal{F}_k$ and the robot frame expressed in the measurement frame $\mathcal{F}_{k,t}$, analogously to what done in section 3.2, to which we remind for a more detailed description of the frames. In the reminder of this chapter, we will drop the dependency on the LiDAR scan sample when there is no ambiguity in the notation. By default, unless specified otherwise, all the quantities are referred to the measurement frame $\mathcal{F}_k$.

The robot is tasked to reach a target position $\boldsymbol{x}_f(t) \in \mathcal{C}_{free}$, possibly time varying. In the spirit of pursuing full autonomy, the robot is not aware of its own global position in the world frame and can only sense the target with on-board sensors, e.g. a camera or an external estimation algorithm. In the scope of this work we are not interested in the technical details of how the target is estimated, but we distinguish two cases:

- *Target in sight*: the robot is able to sense the position of the target, which is then expressed in the current robot frame $\mathcal{F}_t$.
- *Target not in sight*: the robot is not able to have an estimate of the target's position, so it need to explore the environment until it finds an estimate of the target.

5.3 Policy-based exploration-exploitation

The strategy that we are going to present is based on a policy-switching method that takes inspiration from the famous reinforcement learning of exploration-exploitation paradigm. Unlike the latter context, NAPVIG conveniently allows to locally access the GVD, so the policies can be defined algorithmically, with a method that is based on the optimization of a cost function.

Given that the basic assumption is that the robot does not have the knowledge of the map, and we desire not to spend computational resources for expensive mapping algorithms, the target-pursuit goal must be entangled with the need for exploring the map, since a path that might seem the shortest one leading to the target could result in dead-ends, hence the need to keep track of an exploration factor, as will be more clear in the rest of this chapter.

The full stack methodology that we will introduce is based on a set of six policies, that can be categorized as *predictive*, *reactive*, and *auxiliary*, namely:

- Predictive: *fully-exploitative* (PFT), *fully-explorative* (PFX), *partly-explorative* (PPX);
- Reactive: *legacy* (PLE);
- Auxiliary: *free-space* (PFS), *halt* (PHT).

Depending on the current status of the robot and the configuration space, each policy has the role of generating a trajectory $\boldsymbol{\xi}$ that will be fed to the low level controllers. Depending on the internal logic of each of them, they provide a result status p associated to the trajectory that can assume one of the following values: *accept* (RA), *fail* (RFA), *finalize* (RFI), *complete* (RC). The result status will be the possible transition symbols of a finite state machine that regulates the switching rules from a policy to another. We will first introduce the possible policies and then will provide a more detailed description of their purpose.

5.4 Predictive policies

The main navigation task is performed by the predictive policies: they are based on the iterative execution of the NAPVIG algorithm and are characterized by three key factors: a *search direction* decision rule (SDR), a *termination* rule (TR), and a *cost factor* that is associated to each predicted trajectory.

Given a search decision and termination rule, the prediction of a trajectory is simple and reported in Alg. 2. Each policy will then adopt a combination of SDR and TR that will define TR.terminate, the termination condition, which is function of the last sample, and SDR.decideSearch, function of the entire trajectory. Additionally, TR also specifies an exit information associated to the reason that caused the prediction to terminate, that can be one of the following: *collision* (TCO), *max window* (TMW), *target approached* (TTA), *napvig fault* (TNF). Each termination cause may arise in different context

Algorithm 2 Trajectory Prediction Algorithm

```
Require: $\boldsymbol{x}_0 \in \mathcal{C}_{free}$: initial robot position
Require: $\boldsymbol{r}_0 \in \mathbb{S}^1$: initial search direction
  function PREDICT($\boldsymbol{x}_0$, $\boldsymbol{r}_0$ SDR, TR)
      $\boldsymbol{\xi}_0 \leftarrow \boldsymbol{x}_0$
      while not TR.terminate($\boldsymbol{\xi}_k, \boldsymbol{r}_k$) do
          $\boldsymbol{r}_k \leftarrow$ SDR.decideSearch($\boldsymbol{\xi}$)
          $\boldsymbol{\xi}_{k+1} \leftarrow$ napvig($\boldsymbol{\xi}_k, \boldsymbol{r}_k$)
      end while
      $c_{\boldsymbol{\xi}} \leftarrow$ TR.cause()
      return ($\boldsymbol{\xi}, c_{\boldsymbol{\xi}}$)
  end function
```

and entails different consequences for the different policies: we will provide a specific meaning for each of them in the next paragraphs.

Then, they can predict one or more trajectory $\boldsymbol{\xi}^{(h)}$, with associated exit status $c_{\boldsymbol{\xi}}^{(h)}$, for $h = 1, \dots, H$, where $H \in \mathbb{N}$ is the total number of predicted trajectories: to each of them it is associated a cost value $J(\boldsymbol{\xi}^{(h)}, c_{\boldsymbol{\xi}}^{(h)})$. In some cases, depending on the termination condition the trajectory needs to be directly discarded: in those cases the cost of the trajectory can be associated to an infinite value. For all the policies, the best trajectory is the one that solves the following optimization problem:

$$\boldsymbol{\xi}^* = \underset{\boldsymbol{\xi}^{(h)},\ h=0,\dots,H}{\operatorname{argmin}} \; J(\boldsymbol{\xi}^{(h)}, c_{\boldsymbol{\xi}}^{(h)}). \tag{5.2}$$

5.4.1 Prediction termination rules

We now define a collection of termination rules that can be adopted in various combination in each policy. The termination rule is based on the current sample of the trajectory $\boldsymbol{\xi}_k$ and on the set of collision points $\mathcal{M}$, and can be one of the following:

- Maximum window (TMW): the number of samples reached the maximum value: $k > K_{max}$, where $K_{max} \in \mathbb{N}$ can be different for each policy;

- Collision (TCO): last sample is too close to the obstacles: $\min_{\boldsymbol{m}\in\mathcal{M}} \|\boldsymbol{\xi} - \boldsymbol{m}\| < \rho_{coll}$, where $\rho_{coll} > 0$ is the radius that circumscribes the robot's phyisical geometry. This can happen in the cases when the GVD branch on which the sample is computed is leading to a dead end, corresponding on the Voronoi boundary between two convex components of $\mathcal{C}_{coll}$ that are adjacent.

- Target approached (TTA): last sample is close to the target: $\|\boldsymbol{\xi}_k - \boldsymbol{x}_f\| < \rho_{target}$, where $\rho_{target} > 0$ is the chosen threshold.

- Napvig fault (TNF): the search direction is such to violate the *NAPVIG condition*, the theoretical assumption on which the NAPVIG algorithm is based (4.31), which is equivalent to $\boldsymbol{r}_k \parallel (\boldsymbol{\xi}_k - \boldsymbol{m}^*)$, where $\boldsymbol{m}^* =$

Label	Condition
TMW	$k > K_{max}$
TCO	$\min_{\boldsymbol{m}\in\mathcal{M}} \lVert\boldsymbol{\xi} - \boldsymbol{m}\rVert < \rho_{coll}$
TTA	$\lVert\boldsymbol{\xi}_k - \boldsymbol{x}_f\rVert < \rho_{target}$
TNF	$\boldsymbol{r}_k \parallel (\boldsymbol{\xi}_k - \boldsymbol{m}^*)$

Table 5.1.: Conditions and termination codes

$\operatorname{argmin}_{\boldsymbol{m}\in\mathcal{M}} \lVert\boldsymbol{\xi}_k - \boldsymbol{m}\rVert$, if we note that the search direction $\boldsymbol{r}_k$ is orthogonal to the direction $\boldsymbol{d}$ in the NAPVIG theorem.

The conditions are summarized in table 5.1

5.4.2 Fully-exploitative policy

The first idea is to generate a trajectory that points towards the target. More precisely, for each prediction sample we define the search decision rule:

$$\text{PFT-SDR.searchDirection}(\boldsymbol{\xi}) := \frac{\boldsymbol{\xi}_k - \boldsymbol{x}_f}{\lVert\boldsymbol{\xi}_k - \boldsymbol{x}_f\rVert} \tag{5.3}$$

Note that the points of the trajectory are necessarily on the GVD, while the target could be any point in the free space. As commonly done in Voronoi-based navigation approaches, the algorithm is meant to terminate when the last sample is sufficiently close to the target (TTA condition), and will then switch to a different control strategy. While this strategy could result in a very efficient computation since it tends to compute minimal trajectories that are greedily directed towards the target, there are no guarantees that the resulting search direction is valid or does not lead to collisions. In principles, it would make sense not to consider a maximum number of step, and let the prediction terminate either when the target is approached or when a collision occurs. However, given the limited sensing capabilities, there could be obstacles producing a dead-end that are outside the maximum range of the LiDAR sensor, so the collision cannot be detected. An easy heuristic to overcome this issue is to limit the maximum prediction window such that the trajectory does not exceed the LiDAR range. For these reasons, we consider

Termination cause	TMW	TCO	TNF	TTA $\wedge \|\boldsymbol{\xi}\| = 1$	TTA $\wedge \|\boldsymbol{\xi}\| > 1$
p_{PFT}	RFA	RFA	RFA	RFI	RA

Table 5.2.: Fully-exploitative result status for each termination cause

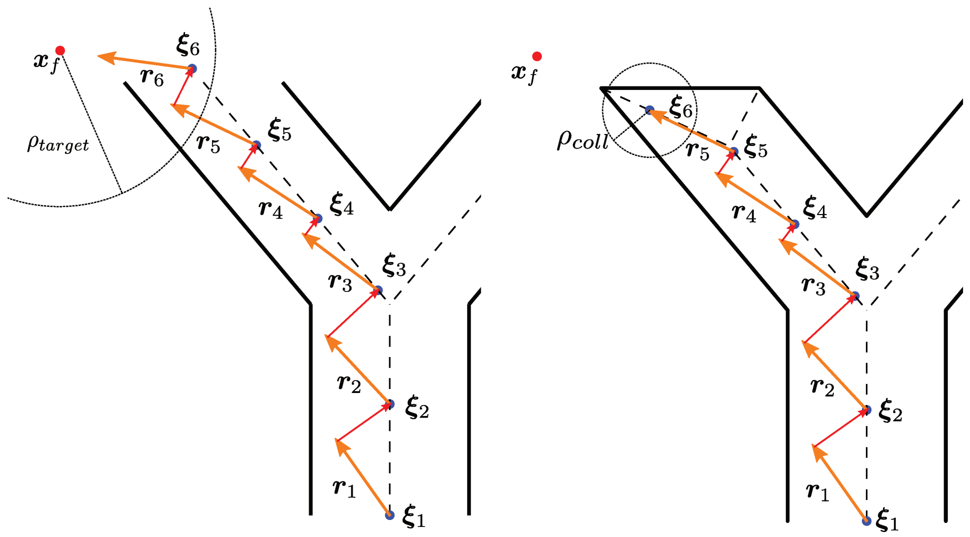
as termination rule the union of all the conditions introduced in the previous section:

$$\mathsf{PFT\text{-}TR}.\mathrm{terminate}(\boldsymbol{\xi}_k, \boldsymbol{r}_k) := (\mathsf{TMW} \vee \mathsf{TCO} \vee \mathsf{TTA} \vee \mathsf{TNF}) \tag{5.4}$$

The predicted trajectory with PFT is predict with PFT-SDR and PFT-TR, in short:

$$\boldsymbol{\xi} = \mathsf{PFT}.\mathrm{predict}(\boldsymbol{x}_0, \boldsymbol{r}_0) \tag{5.5}$$

In this case, since only one trajectory is possible, either the predicted trajectory $\boldsymbol{\xi}$: is accepted or discarded. Clearly, if the policy terminates with a collision TCO or a violation of the NAPVIG condition TNF, then the trajectory is discarded, and the result status is $p_{\mathsf{PFT}} = \mathsf{RFA}$ (i.e. *fail*). Otherwise, if the trajectory is immediately terminated by TTA, it means that $\boldsymbol{x}_0$ is already sufficiently close to the target. In this case, the (empty) trajectory is again discarded, and the result status $p_{\mathsf{PFT}} = \mathsf{RFI}$ (i.e. *finalize*) will cause the policy to switch to another method to track the final path to $\boldsymbol{x}_f$. Finally,



(a) Accepted trajectory TTA $\wedge |\boldsymbol{\xi}| > 1 \Rightarrow p_{\mathsf{PFT}} = \mathsf{RA}$.

(b) Rejected trajectory TCO $\Rightarrow p_{\mathsf{PFT}} = \mathsf{RFA}$.

Figure 5.1.: Example of predicted trajectories with PFT policy

if the policy is terminated by TTA and there is at least one sample other than the initialization, then the trajectory is accepted, and the result status is $p_{\mathsf{PFT}} = \mathsf{RA}$, (*accept*). Table 5.2 summarize the result status as function of the termination cause, where with the notation $|\boldsymbol{\xi}|$ we mean the number of samples of $\boldsymbol{\xi}$. In Fig. 5.1 are reported two illustrative examples of application of the fully-exploitative policy. Starting from $\boldsymbol{\xi}_1$, all the search directions point towards $\boldsymbol{x}_f$ (in red): this choice caused the path to follow the left corridor. In the first case (Fig. 5.1a), the latter is open, so as soon as a sample is closer to the target than ρ_{target}, the trajectory is accepted. In the second example, the left corridor is closed, and NAPVIG computes points on the branch that would lead to the vertex between the convex components of the front and left walls. At some point, that branch will be closer to the walls (collision points) than ρ_{coll}, as $\boldsymbol{\xi}_6$ results to be in the figure. The trajectory is then rejected: PFT is not able to produce a valid trajectory in this case.

Remark 5.1. It could be noticed that the dead-ends problems are similar to those affecting methods based on artificial potential fields. Nevertheless, this policy keeps the advantage of computing point in the GVD and, if a feasible trajectory exists, it is efficient in terms of number of computed samples. Otherwise, policy-switching logic will handle the case to a different policy, that in turn will result to be more expensive, as we will see in the next sections.

Remark 5.2. Differently to what happens with constrained optimization based-methods, NAPVIG does not find trajectories at the border of the collision space but farthest away from obstacles. Usually, this requires the collision radius ρ_{coll} to be quite larger than the real one, to provide a margin of safety. Instead, given the accuracy that our algorithm results to provide, the collision radius ρ_{coll} requires much less margin and can be almost the geometrical radius of the circle circumscribed to the robot, allowing the passage to narrower passages.

5.4.3 Explorative policies

Consider the example in Fig. 5.1b, where the PFT policy is rejected. Given that a direct trajectory towards the target does not exist, after having predicted a colliding trajectory, in principle the algorithm should choose the corridor on the right. However, the points on the GVD are computed individually, and since, additionally, no prior knowledge is assumed for the map, there is no semantic knowledge directly available from their computation. In view of the reactive nature of the navigation strategy that we are proposing, we now introduce a local exploration strategy based on the prediction of trajectories in multiple directions starting from the robot's position.

Given an initial robot position in measurement frame $\boldsymbol{x}_0$ and an initial direction $\boldsymbol{r}_0$, which correspond to the heading of the robot in the same reference frame, the idea is then to consider multiple directions starting from $\boldsymbol{x}_0$, spanning the entire round angle:

$$\begin{aligned} \boldsymbol{\xi}^{(h)} &:= \mathsf{PEX}.\mathrm{predict}(\boldsymbol{x}_0, \boldsymbol{r}_{\vartheta_h}), \quad h = 1, \ldots, H_{\mathsf{PEX}}, \\ \boldsymbol{r}_{\vartheta_h} &:= \boldsymbol{R}(\vartheta_h)\boldsymbol{r}_0, \quad \vartheta_h = \frac{2\pi}{H_{\mathsf{PEX}}}, \end{aligned} \tag{5.6}$$

where PEX can refer to PFX or PPX, of which the corresponding search decision and termination rules, SDR and TR, will be introduced in the next paragraph; $\boldsymbol{R}(\vartheta_h)$ is the 2D matrix that rotates by an angle ϑ_h; H_{PEX} is the desired number of initial directions. As will be clear later, this number does not need to be large, since in most situations multiple initial directions will converge to the same GVD branch, so in the experiments we consider $H_{\mathsf{PEX}} := 8$, while in the illustrative examples that are going to be reported in this section we consider $H_{\mathsf{PEX}} = 4$ for clarity of presentation.

The prediction starts with directions $\boldsymbol{r}_{\vartheta_h}$ and continues with the direction that from one sample points towards the next one, as follows:

$$\mathsf{PEX\text{-}SDR}.\mathrm{searchDirection}(\boldsymbol{\xi}) = \frac{\boldsymbol{\xi}_{k-1} - \boldsymbol{\xi}_k}{\|\boldsymbol{\xi}_{k-1} - \boldsymbol{\xi}_k\|} \tag{5.7}$$

Remark 5.3. This choice allows the exploration of the GVD branch starting from the angle ϑ_h, and it only has a local meaning, in the sense that it is not well suited for large prediction windows. This characteristic has to be interpreted in the scope of local navigation that represent the real aim of these policies: the basic assumption is that the obstacles information, gathered by a LiDAR sensor, are also local. The integration of such strategy with the actual motion of the robot will enable the full navigation of the actual map.

The prediction algorithm of both the *fully* and *partly* explorative policy is then terminated by a maximum number of samples, other than the collision-detection conditions:

$$\mathsf{PEX}.\mathrm{terminate}(\boldsymbol{\xi}_k, \boldsymbol{r}_k) := (\mathsf{TMW} \vee \mathsf{TCO} \vee \mathsf{TNF}) \tag{5.8}$$

Landmarks A key factor for this strategy is the ability of keep track of explored areas: we will achieve this by storing *landmarks* as the robot moves in the environment. Formally, we define a landmark as a triplet $\boldsymbol{\ell} = ({}^{\ell}\mathcal{F}, {}^{\ell}t, {}^{\ell}\boldsymbol{x}) \in \mathbb{SE}(2) \times \mathbb{R} \times \mathbb{R}^2$, where:

- ${}^{\ell}\mathcal{F} \in \mathbb{SE}(2)$ is the last measurement frame when the landmark is created;
- ${}^{\ell}t \in \mathbb{R}$ is the timestamp of the moment the landmark is created;
- ${}^{\ell}\boldsymbol{x} \in \mathbb{R}^2$ is the position of the robot in the moment of the landmark creation, expressed in the frame ${}^{\ell}\mathcal{F}$.

The landmarks are stored in a batch $\mathscr{L} = \{\boldsymbol{\ell}_i,\ i = 0, \ldots, N_L\}$ where $N_L \in \mathbb{N}$ is the current number of landmarks. The batch is initialized with a single landmark corresponding to the initial position: $\mathscr{L} = \{\boldsymbol{\ell}_0 = ({}^{\ell}\mathcal{F}_0, {}^{\ell}t_0, {}^{\ell}\boldsymbol{x}_0)\}$, where ${}^{\ell}\mathcal{F}_0 = \mathcal{I}$, where $\mathcal{I}$ is the identity in $\mathbb{SE}(3)$, ${}^{\ell}t_0 = 0$, and ${}^{\ell}\boldsymbol{x}_0 = \mathbf{0}$. The new landmarks are then added to $\mathscr{L}$ until $N_L \geq N_{L,max}$, that is the maximum capacity of the landmark: after that, the new landmarks replace the oldest ones, in a FILO queue.

Remark 5.4. This definition of the landmarks need a common inertial frame to which the frames ${}^{\ell}\mathcal{F}$ are referred to. Given the full-autonomy requirement, a global position is not available to the algorithm. It is possible, instead, to integrate the odometry starting from the initial position, obtaining an estimate of the frames that are knowingly affected by an error that is increasing overtime. However, as will be clear later, the real used information will be the relative frame between each ${}^{\ell}\mathcal{F}$ and the current measurement frame, that results in a bounded error since the landmarks are a finite number and the old ones are replaced with new ones. This also motivates the choice ${}^{\ell}\mathcal{F}_0 = \mathcal{I}$.

The landmark position is referred to the creation frame ${}^{\ell}\mathcal{F}_i$. However, we are going to need its position in the current frame, namely $\mathcal{F}$. We then define the function $\mathcal{F}_{i,t}$ as:

$$\mathcal{F}_{i,t} \ : \ \mathbb{R}^2 \to \mathbb{R}^2 \ : \ \boldsymbol{x} \mapsto \mathcal{F}_{i,t}\boldsymbol{x} := \mathcal{F}^{-1}\mathcal{F}_i\boldsymbol{x} \tag{5.9}$$

We consider two standard conditions upon which a new landmark is created:

- $\|\boldsymbol{x}_t - \mathcal{F}_{i,t}{}^{\ell}\boldsymbol{x}_{N_L}\| > d_{landmark}$, for a constant $d_{landmark} > 0$: when it is farther from the last landmark. This is the standard way the landmarks are created when the robot moves in the scenario.

- $t - {}^{\ell}t_{N_L} > \tau_{elapsed}$: when too much time has passed since the last landmark creation. This can help to resolve ambiguous situations where the cost for multiple trajectories is similar. This will be covered with more details in the experiment part.

According to these rules landmarks are generated corresponding to points physically visited by the robot. However, if a predicted trajectory terminates with a collision, landmarks can be created in correspondence to the samples of the trajectory $\boldsymbol{\xi}_k$.

Fully-Explorative policy We first consider the most general case where there is no target (e.g. the target is not in sight) and the goal of the navigation task is to explore the map as much as possible. The idea is that the cost function associated to each predicted trajectory $\boldsymbol{\xi}^{(h)}$ penalizes those being close to visited areas. Specifically, for each point in the map we consider a penalty term that with a Gaussian trend with the distance to the landmark:

$$J_i \ : \ \mathbb{R}^2 \to \mathbb{R} \ : \ \boldsymbol{x} \mapsto J_i(\boldsymbol{x}) := w_\ell e^{-\frac{\|\boldsymbol{x} - {}^{\ell}\boldsymbol{x}_i\|^2}{2\rho_\ell}} \tag{5.10}$$

where $w_\ell \in \mathbb{R}$ is a weight associated to the landmark cost and $\rho_\ell \in \mathbb{R}$ is a value tuning the peak radius, as shown in Fig. 5.2. The figure also suggests an heuristics on how to choose the value for ρ_ℓ, that should be such the

Gaussian is almost zero beyond the average minimum distances between the obstacles.

Depending on the characteristics of the map, it could be needed that older landmarks are weighted less than new ones, so that to allow the exploration of the same area after a desired amount of time, for instance to cope with a dynamic configuration of the map. To this aim, at the generic time t we consider an exponentially decaying weight term $e^{-\lambda(t-{}^{\ell}t_i)}$, where $\lambda \geq 0$ is the tuning parameter acting as time constant in the exponential decay. Note that in the cases this aspect is not desired, decaying can be disabled by just setting $\lambda = 0$.

The total cost associated to each point in $\mathbb{R}^2$ is then the sum of the contributions of the penalty terms of each landmark, and the total cost associated to the predicted trajectory $\boldsymbol{\xi}$, is the sum of the cost of each sample, provided that the prediction ended without faults or collisions:

$$J_{\mathsf{PFX}}(\boldsymbol{\xi}) = \begin{cases} \sum_{k=0}^{|\boldsymbol{\xi}|} \sum_{i=1}^{N_L} e^{-\lambda(t-{}^{\ell}t_i)} J_i(\boldsymbol{\xi}_k) & \text{if } c_{\boldsymbol{\xi}} = \mathsf{TMW} \\ +\infty & \text{if } c_{\boldsymbol{\xi}} = \mathsf{TCO} \vee \mathsf{TNF} \end{cases} \tag{5.11}$$

It could happen that every predicted trajectory results in a collision/fault. In this case, according to the parameter configuration, could mean that the robot cannot move without resulting in a collision within the considered prediction window. If this is actually the case, it represents a pathological condition where the exploration problem is ill-posed: the robot is placed in a confined environment. In some cases, this could be desirable: imagine the case of an

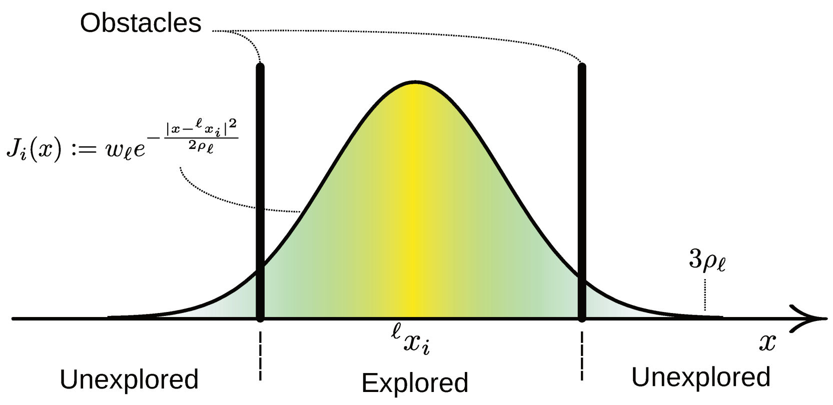


Figure 5.2.: Example of 1-D component of the cost associated to a landmark

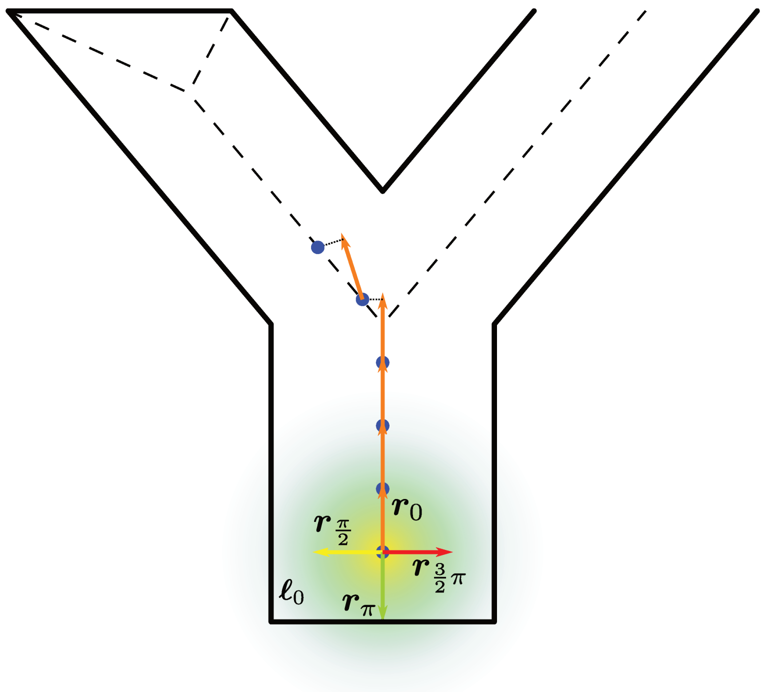
elevator, where the robot cannot move until the floor is reached, so this event should be handled properly by the policy switching rules, presented later on. In other cases, where the robot could actually move, but a trajectory is not found, it means that the parameters are not properly set for the scenario. In particular, we can point out several possible reasons:

- The considered number of initial directions H_{PEX} are not enough to identify the available branches. In typical situations (see sec. 5.7), with $H_{\mathsf{PEX}} = 8$ this is very uncommon. For scenarios configuration when this happens, a higher number of initial conditions should resolve the issue.

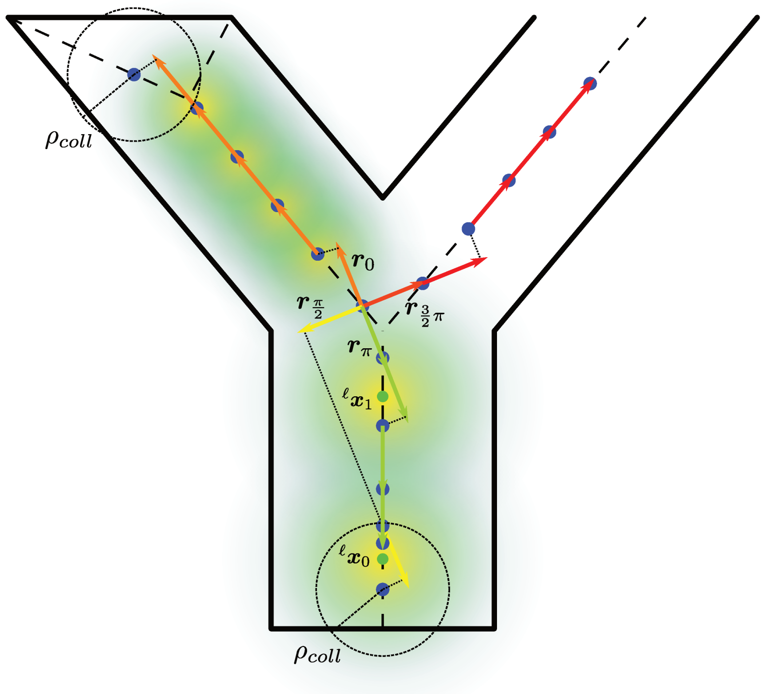
- The prediction window is too large: as mentioned, the search decision rule (PEX-SDR) is thought to be reasonable in a local neighborhood of the initial point $\boldsymbol{x}_0$. More specifically, PEX-SDR is assumed to track a single GVD branch, so in case of bifurcations in the prediction, which branch will be taken cannot be ensured. For typical applications, we consider a value of around $K_{max} = 4$.

- The NAPVIG step β_{step} is too large. Linked to the previous aspect, note everything in between the NAPVIG initialization and the initial step is not accounted, so a large value for β_{step} could even exceed an obstacle. In principle, this value should be small, but this would require a larger prediction window, so a trade-off should be considered for efficiency. For the geometries of robot and obstacles of the experiments, $\beta_{step} = 0.2$ does not report any of these issues.

Fig. 5.3 reports two phases of exploration with PFX, in which we considered the same scenario as with the fully-exploitative policy. Initially (fig. 5.3a), $\mathscr{L}$ only contains the initial landmark $\boldsymbol{\ell}_0$, placed in correspondence to the initial robot position (black circle). We then consider $H_{\mathsf{PEX}} = 4$ initial directions, for $\vartheta = 0, \frac{\pi}{2}, \pi, \frac{3}{2}\pi$, respectively. It is immediate to notice that all direction except $\vartheta = 0$ are orthogonal to the walls, resulting in a NAPVIG fault (TNF) termination. To those trajectories will then be directly associated an infinite cost and will then be discarded. The only possibility is then to predict a trajectory (in orange) in the direction of $\boldsymbol{r}_0$, which results in no collision and is terminated with the maximum window step count (TMW). An important remark: the fourth search direction $\boldsymbol{r}_4$ is in principle violating the NAPVIG condition: the orthogonal line to $\boldsymbol{r}_4$ (namely, $\mathcal{A}$ in (3.12)) intersects the GVD into two points, one on the left and one on the right. The NAPVIG condition violation means that in the initial point of the optimization we are



(a) Example of unexplored map, with the initial landmark and the predicted trajectories.



(b) Intermediate exploration step, with several landmarks and multiple predicted trajectories.

Figure 5.3.: Example of prediction-based map exploration with PFX.

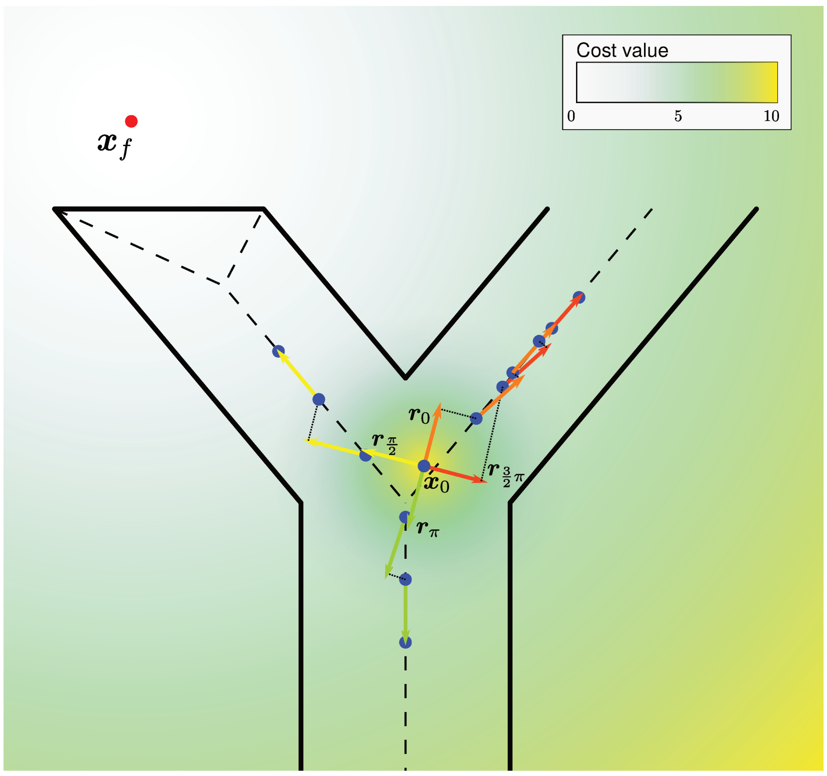
in presence of a stationary point (the gradient of the restriction is zero) then corresponding to a maximum point, so mathematically the algorithm cannot converge. However, a small deviation from the ideal condition would lead the gradient descent to actually converge, but it can in principle identify both branches. Since there is no prior knowledge on the map, there is no a-priori reason to prefer one branch to the other: if needed, both of them will be reactively explored, but this choice is taken on-line when the robot is itself in correspondence to the GVD bifurcation. At that stage, the initial search directions $\boldsymbol{r}_\vartheta$ will indisputably identify the two different branches, as will be clear in the second snapshot of exploration. At this first moment, we consider, without loss of generality and for illustrative purposes only, that the NAPVIG algorithm converges to the left branch.

Figure 5.3b shows a second stage in the exploration. We assume that the robot followed the predicted trajectory until close to the bifurcation. Note that before that sample, all the trajectories different from $\vartheta = 0$ would result in TNF. Once in that position the robot is headed as $\boldsymbol{r}_0$. Then, from the four initial search directions, colored in orange, yellow, green and red, for $\vartheta = 0, \frac{\pi}{2}, \pi, \frac{3}{2}\pi$, respectively. From the figure it is clear that the orange trajectory ends with a collision, so we consider landmarks centered in each sample of the trajectory: $\boldsymbol{\xi}_k^{(0)}$, $k = 1, \dots, K_{max}$. Note that the trajectory is very similar to the one predicted by PFT (Fig. 5.1), even if the search directions are different. The initial directions $\boldsymbol{r}_{\frac{\pi}{2}}$ and $\boldsymbol{r}_\pi$ converge to the same GVD branch, which ends with a collision, so both trajectories will add their samples to the landmark set, even if they are not represented in the picture for clarity purposes. The last trajectory, predicted from $\boldsymbol{r}_{\frac{3}{4}\pi}$, results in tracking the GVD branch of the right corridor. In this case, that is the only trajectory available since all the others result with a collision. However, with a smaller prediction horizon, e.g. $K_{max} = 3$, there would be no landmark in the left corridors as the orange trajectory would be terminated with TMW. The yellow one would be discarded as it would terminate with a collision, while the green would also be accepted. Nevertheless, it would be the one with the highest cost, considering that its samples are closer to the two landmarks. The orange and red, instead, have low cost since they are both towards unexplored areas. The algorithm, however, would choose the orange, although the costs are very similar but slightly lower: in the context where the goal is the exploration only, there is no other preferential way on choosing the trajectories, so that would be the final decision even if later the trajectory would lead to a dead-end. In that case, the robot would move backwards, since that would be the only available trajectory, until approaching the bifurcation again: at that point, the landmarks would be similar to figure 5.3b, although they are placed after visiting and not predicted, the red trajectory would be generated and in that case the orange one, that is not colliding anymore, would be discarded as the associated cost would be higher for the presence of landmarks in the left corridor. This behavior has central importance for the goals of the next policy, and will be discussed in more detail in the following section.

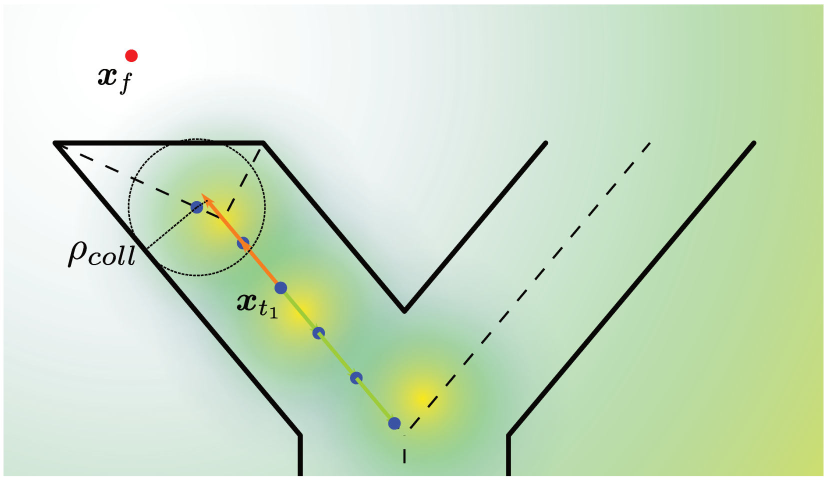
Partly-explorative policy We now consider the *target in sight* case, where the goal is reaching the target, but there is no available trajectory obtained with the *fully-exploitative* policy (PFT). We consider the same method to the GVD branches as with the *fully-explorative* policy (PFX). We then consider the same rules for landmarks generation and the same prediction rules, i.e. we consider PEX-SDR and PEX-TR as search direction and termination rule, respectively. We also consider the cost $J_{\mathsf{PFX}}(\boldsymbol{\xi})$, to which we add a term penalizing the distance to the target, defining the new cost for PPX :

$$J_{\mathsf{PPX}}(\boldsymbol{\xi}) = J_{\mathsf{PFX}}(\boldsymbol{\xi}) + \sum_{k=0}^{K_{max}} w_{target} \|\boldsymbol{\xi}_k - \boldsymbol{x}_f\|^2, \tag{5.12}$$

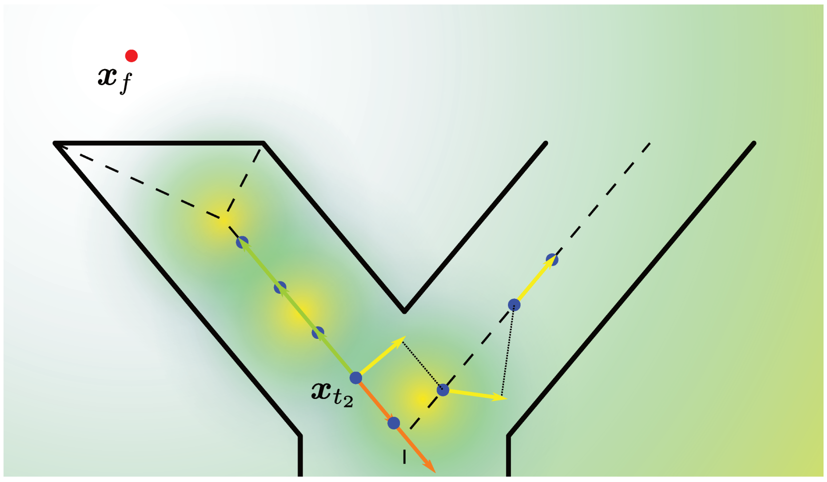
where $w_{target} \in \mathbb{R}$ is a weight balancing the trade of between exploration and exploitation: in principle it would be desirable to choose the trajectories that are closer to the target, but those trajectories may lead to a dead-end. We show the importance of this trade off with an illustrative example, of which three samples are represented in Fig. 5.4. The initial condition is reported in 5.4a: the robot is positioned in $\boldsymbol{x}_0$, headed as $\boldsymbol{r}_0$ and from the four initial directions, $\vartheta = 0, \frac{1}{2}\pi, \pi, \frac{3}{2}\pi$ are generated the predicted trajectories in orange, yellow, green and red, respectively. The value of the cost function for the individual sample is reported with a shading gradient from white to yellow. To emphasize the phenomenon that we wish to point out while maintaining the same map as the previous examples, we only consider a prediction horizon $K_{max} = 3$. More details on this in the remark 5.5. Note that red and yellow trajectories are almost overlapping, as they are tracking the same GVD branch, while the other corridors are identified by the other two. The effect of the first landmark on the total cost is nearly the same for all of them, while the target penalty clearly favors the yellow trajectory, which then results to



(a) Stage #1



(b) Stage #2

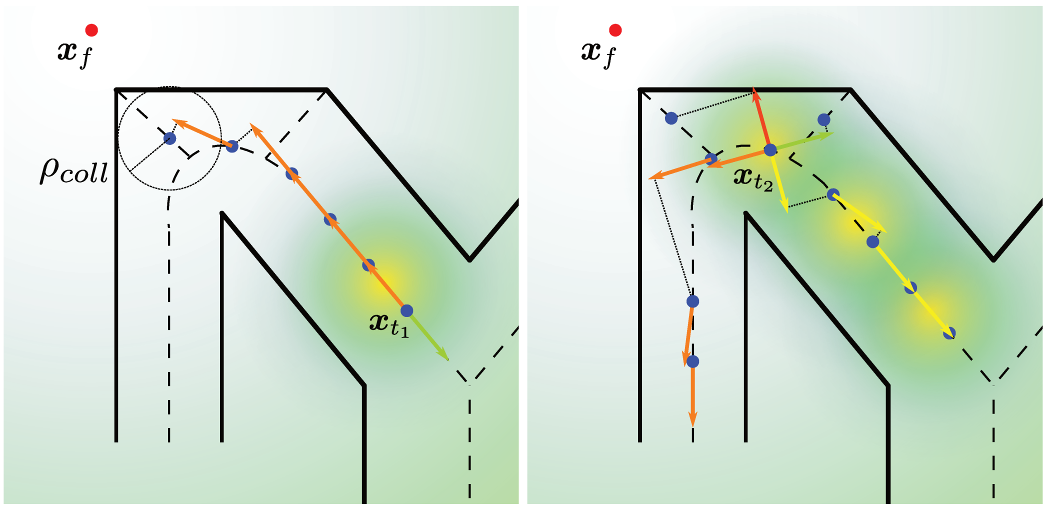


(c) Stage #3

Figure 5.4.: Example of partly-explorative policy

be the selected one. We now consider a time instant $t_1 \in \mathbb{R}$ after the robot has been following the yellow trajectory and is now placed in $\boldsymbol{x}_{t_1}$ (Fig. 5.4b) and headed as the orange arrow. From now on, in this example, we will avoid drawing the directions that results in TNF, for clarity. At this moment, the forward trajectory (orange) result in a collision (TCO): the only valid trajectory is the one that leads back towards the bifurcation, away from the target. Due to the TCO termination, all the left corridor is then marked as "explored" by the placement of landmarks. The third time instant that we consider, $t_2 \in \mathbb{R}$, shows the importance of the exploration part in the cost function: after the robot performs a 180° rotation and starts following the corridor towards the bifurcation, as soon as the backwards trajectory (green) does not result in a collision, due to the limited prediction window, if there were no exploration term in the cost function, that trajectory would have a smaller penelty, and the robot would start again following it, in an infinite loop. Instead, by considering the placed landmarks, the algorithm will favor the exploration of other branches. Indeed, in $\boldsymbol{x}_{t_2}$ the yellow trajectory ends up identifying the right corridor GVD branch, while the orange will identify the central corridor: at that point, the orange trajectory clearly would have a higher penalty from the target distance and the robot will follow the yellow one. Essentially, the exploration term allows overcoming local minima-like situations by locally increasing the cost function. Note that the ratio between the weights of the landmark and the target distance is critical: if the latter is too small, in t_2 (5.4b) the algorithm could choose the orange trajectory that would lead to the backward corridor, since the first sample of the yellow one is very close to a landmark. This aspect becomes a limitation when the map is too large: the same weight ratio suitable for a small environment might not be proper for a larger map, due to the different scale that the target distance term produces. While it could be possible to design an adaptive weighting strategy, we highlight that the proposed navigation strategy is meant to solve local navigation, and it should not be considered as a global planner.

Remark 5.5. The search decision rule SDR-PEX, that from the initial orientation $\boldsymbol{r}_{\vartheta_h}$ considers the directions to one sample from the previous one, as mentioned, is reasonable for a limited prediction length. Fig. 5.5 reports an example showing this aspect. We consider $K_{max} = 6$: the robot in $\boldsymbol{x}_{t_1}$ predict the trajectory in orange, but the configuration of the map has a sharp bent and the search decision rule ends up tracking the GVD branch leading to the



(a) Predicted trajectory from x_{t_1}, following PEX-SDR with $K_{max} = 6$

(b) Correct individuation of the corridor from x_{t_2}

Figure 5.5.: Example of prediction window size too large

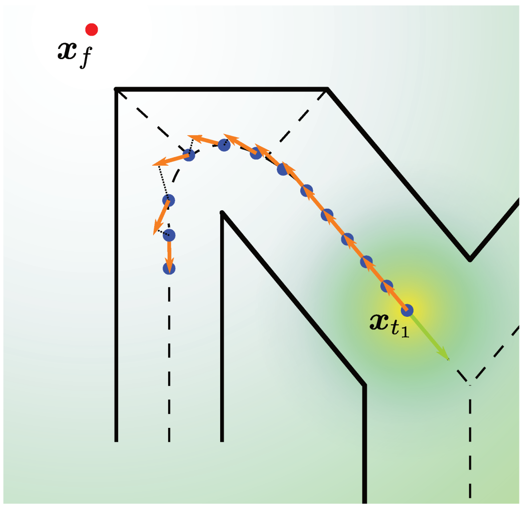


Figure 5.6.: Example of trajectory predicted with SDR-PEX and small β_{step}.

corner, as can be seen from the figure. The algorithm would then be misled into marking all the area as explored, and the robot would start following the green trajectory in the opposite direction. Instead, if we took a shorter prediction horizon, when the robot is positioned in x_{t_2} (Fig. 5.5b), it is able to track sharp bent properly. In the case of a configuration like this it is more suitable to consider shorter values for β_{step}. Figure 5.6 show that with smaller steps the search direction rule is able to capture the correct branch. Nevertheless, it might still not be a good choice to have a long prediction horizon, since the only possibility to look for other directions, and thence other branches, is in correspondence to the first sample.

5.5 Reactive and auxiliary policies

In the last sections we introduced the main policies that are aimed at calculating the path to follow in presence of the ambiguity presented in correspondence to a bifurcation, as in the accompanying examples. To those policies we combine others with the purpose of extending the navigation task or improve its computational efficiency.

Legacy policy While the predictive policies allow the identification of different GVD branches, during the navigation on a single branch, between two bifurcations, there would actually not the need to look for other directions, as it would just be enough to keep tracking the same branch. Clearly, given that there is no direct way to understand whether we are close to a bifurcation or not and, moreover, in real scenarios the GVD is characterized by a lot of spurious branches that are unlinked to a semantic meaning: imagine having an irregular knowledge of the map in the last figures, conceptually, the GVD will be characterized by a lot of branches leading to the intersection between the small segments. The smoothing properties of the landscape function mitigate this problem, but this implies that there is no straightforward way to discriminate the robot being close to a bifurcation or not. Nevertheless, we can adopt the easy heuristics of maintaining the direction computed by predictive policies for a fixed number of samples, and feed it directly to the NAPVIG algorithm. The rules for the policy switching will be introduced in the next section, now we consider that a predictive policy calculated the trajectory $\boldsymbol{\xi}^*$ at time $\bar{t} \in \mathbb{R}$. The measurement frame a that time is $\mathcal{F}_{\bar{t}}$. Then, at the generic time $t > \bar{t}$, the trajectory that the *legacy* policy computes has only one sample, and it is:

$$\begin{aligned} \boldsymbol{\xi}^{\mathsf{PLE}} &= \operatorname{napvig}(\boldsymbol{x}_t, \boldsymbol{r}_{0,\mathsf{PLE}}) \\ \boldsymbol{r}_{0,\mathsf{PLE}} &= \mathcal{F}_t^{-1}\mathcal{F}_{\bar{t}}(\boldsymbol{\xi}_1^* - \boldsymbol{\xi}_0^*) \end{aligned} \tag{5.13}$$

The initial direction $\boldsymbol{r}_{0,\mathsf{PLE}}$ is then the direction from the initial condition that the last computation towards the first predicted sample, converted into the current measurement frame, which we recall is the standard reference frame with respect to which all the quantities are referred to in this discussion. The sample computed by the policy can in general be a collision point or $\boldsymbol{r}_{0,\mathsf{PLE}}$ can trigger a NAPVIG fault. In one of those cases the policy fails, and it is associated with a result status $p_{\mathsf{PLE}} = \mathsf{RFA}$, otherwise the trajectory is accepted and $p_{\mathsf{PLE}} = \mathsf{RA}$.

Free-space policy All the proposed policies are based on the NAPVIG algorithm, that can only output points on the GVD. However, the target position $\boldsymbol{x}_f$ cannot be constrained to be on it, and it can be everywhere in the free space $\mathcal{C}_{free}$. We then adopt a technique that is typical for Voronoi-based navigation: when the robot is sufficiently close to the target, so that the space in between could be considered safe, the robot can navigate in the free space without considering obstacles at all, for the short space that separate the robot to the target. The trajectory in this case is then directly the final point itself, expressed in the last measurement frame:

$$\boldsymbol{\xi}_{\mathsf{PFS}} = \mathcal{F}_t^{-1} \boldsymbol{x}_f \tag{5.14}$$

Approaching a specific point for a unicycle is a delicate task and many existing works tackle the problem in many ways. Precise parking is outside the scope of this work, so we just consider a threshold after which the navigation task is considered completed:

$$p_{\mathsf{PFS}} = \mathsf{RC} \quad \text{if} \quad \|\boldsymbol{x}_t - \boldsymbol{x}_f\| < \rho_{term} \tag{5.15}$$

where $\rho_{term} > 0$ is the desired, small, threshold, otherwise the sample is accepted: $p_{\mathsf{PFS}} = \mathsf{RA}$

Halt policy This is a pure auxiliary policy. There are some specific conditions where the robot needs to stop. A first case is when there is no available policy, that is all the policies are rejected. In this case, the navigation task must be paused and wait for an external event to occur, e.g. a door to open. Another case, that we consider for completeness, corresponds to the target location reached. The trajectory is simply a void command:

$$\boldsymbol{\xi}_{PHT} = \mathbf{0} \tag{5.16}$$

and clearly the result status is always $p_{\mathsf{PHT}} = \mathsf{RA}$.

5.6 Policy switching rules

In the previous section we introduced a set of policies that solve the local navigation problem in specific contexts. The ability to cope with such contexts depend on specific conditions, e.g. if the target is *in sight* or not, or the success of other policies. We now introduce the rules to switch from one to another, that are based on a finite state machine. The initial state is IDLE, representing the state before the algorithm starts. Every other state is associated to a policy, the three predictive PFT, PFX, PPX (green), the reactive PLE (violet), and the auxiliary PFS, PHT (orange), as defined in the previous sections. Once the FSM is transitioned to a state, the corresponding policy is computed. The next transition then occurs according to the result status associated to the computed trajectory RA, RFA, RFI, RC, which are combined with external events, that might be related to map configuration properties or user triggered. Specifically, we consider the events:

- X-START: user-triggered algorithm start.
- X-TIS: *target in sight*, is a property that is true if there exists a direct estimate of the target by the robot.
- X-MLE: *maximum legacy count*: the maximum number of steps computed with policy *legacy* is achieved, keeping the same search direction.
- X-RESET: manual reset when *halt* policy was triggered after no collision-free trajectory can be computed.

In particular, as introduced in the corresponding section, the event X-MLE realizes the idea of the policy *legacy* implementing a more lightweight strategy when tracking a single GVD branch where we assume that there are no bifurcations. This assumption is reasonable as long as the total amount of time is short compared to the map scale and the velocities of the dynamic parts.

The complete scheme of switching rules between policies is reported in figure 5.7. The initial state, IDLE, corresponds to the moment before the algorithm is started. After that, the *fully exploitative* policy, PFT is attempted. Then we consider two cases:

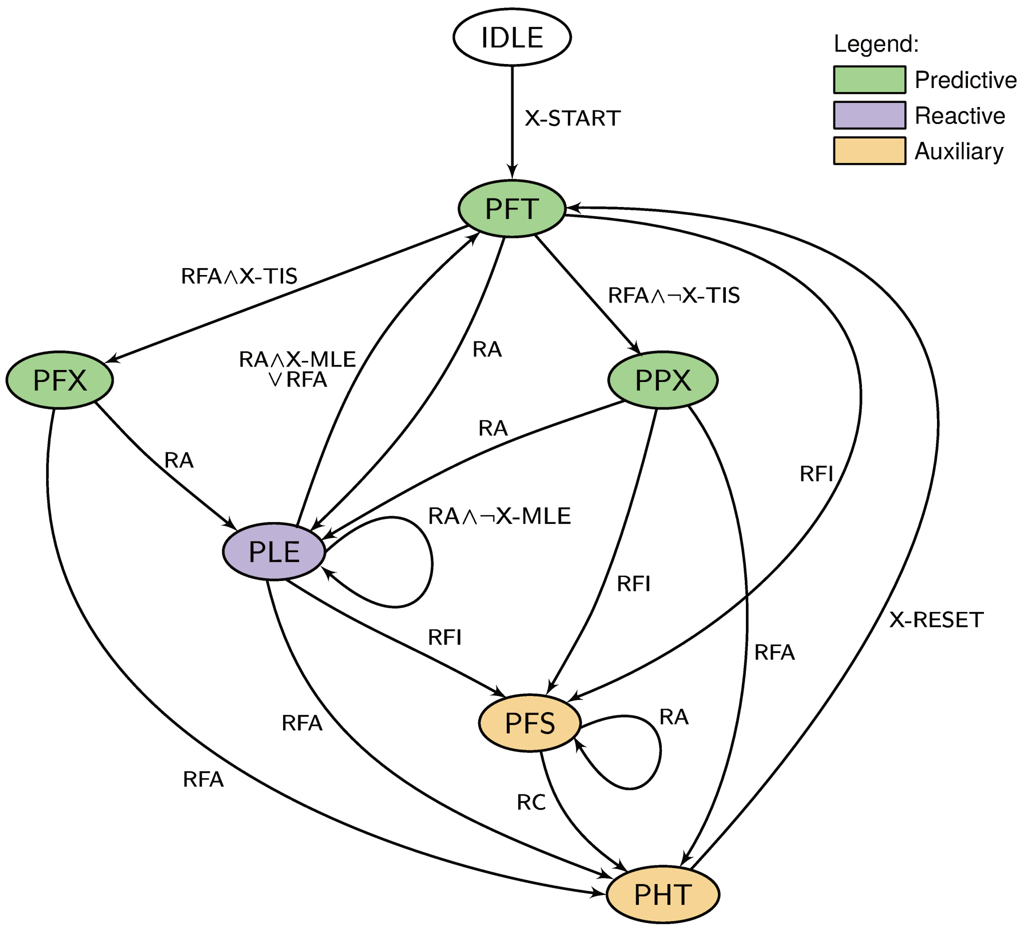


Figure 5.7.: Finite state machine describing the policy switching rules

- *target in sight*: the system computes a trajectory according to PFT. If accepted, it sends it to the low level controllers, otherwise switches to *partly explorative* policy (PPX).
- *target not in sight*: PFT immediately fails and the system directly switches to the *fully explorative* policy, PFX.

When any of the predictive policies (PFT, PFX, PPX) is accepted, the state is switched to PLE, to use the constant search direction from the previous policy. On the transition to PLE, a counter is reset: PLE$_{count} \leftarrow 0$. Then, for each accepted trajectory, the counter increments until the maximum desired value is reached, namely PLE$_{max}$. The event X-MLE is triggered when PLE$_{count}$ = PLE$_{max}$, upon which the system status is transitioned back to PFT, and the predictive policies are attempted again with the same rules.

According to the relative position of robot and target, if the latter is available, the policies can terminate with an RFI result status, when the robot is closer than a threshold to the target, and it can safely perform a parking task, switching to the *free space* (PFS) policy, where the state stays with a self loop until the required accuracy in target tracking is reached, when it switches to the *halt* policy PHT to hold its position until the algorithm is manually reset with X-RESET. The system is also transitioned to this state in correspondence of situations where no possible trajectory can be computed with any of the other policies. This occurrence requires manual intervention and should be exceptional.

5.7 Simulative validation

In the previous sections we provided examples of the algorithm behavior in an ideal scenario where the map was known and so was the GVD, with the purpose of explaining the main concepts. We now assess the validity of the overall methodology in various scenarios of practical interest, in a real-time simulation environment provided by Gazebo robot simulator. Similarly to what done in chapter 3, the implementation is realized with C++/ROS to meet real time constraints. The architecture of such implementation is a critical aspect to consider since it strongly affects the possibility of actually meet the timing requirements. For this reason, parallelism, efficiency and maintainability are of core importance, and they will be presented in more details in the appendix of this thesis. The simulated experimental setup, composed of a differential steering robot equipped with a LiDAR, controlled with a reference position as described in section 3.4.2. We now present several scenarios where various characteristics of the algorithm can be assessed.

5.7.1 Scenario #1: corridors

For the first simulation we consider the standard arena from Turtlebot3[1], to which several walls were added between the columns, shown in dark gray and white in figure 5.8, respectively. The target is assumed to be known in the robot frame at each sample time (*target in sight*), and it is placed at $\boldsymbol{x}_f = [1.5, 0]$ in the world frame, right behind the "head" of the turtlebot arena. The structure of the walls and obstacles is better reported in figure

[1] https://www.turtlebot.com/turtlebot3/

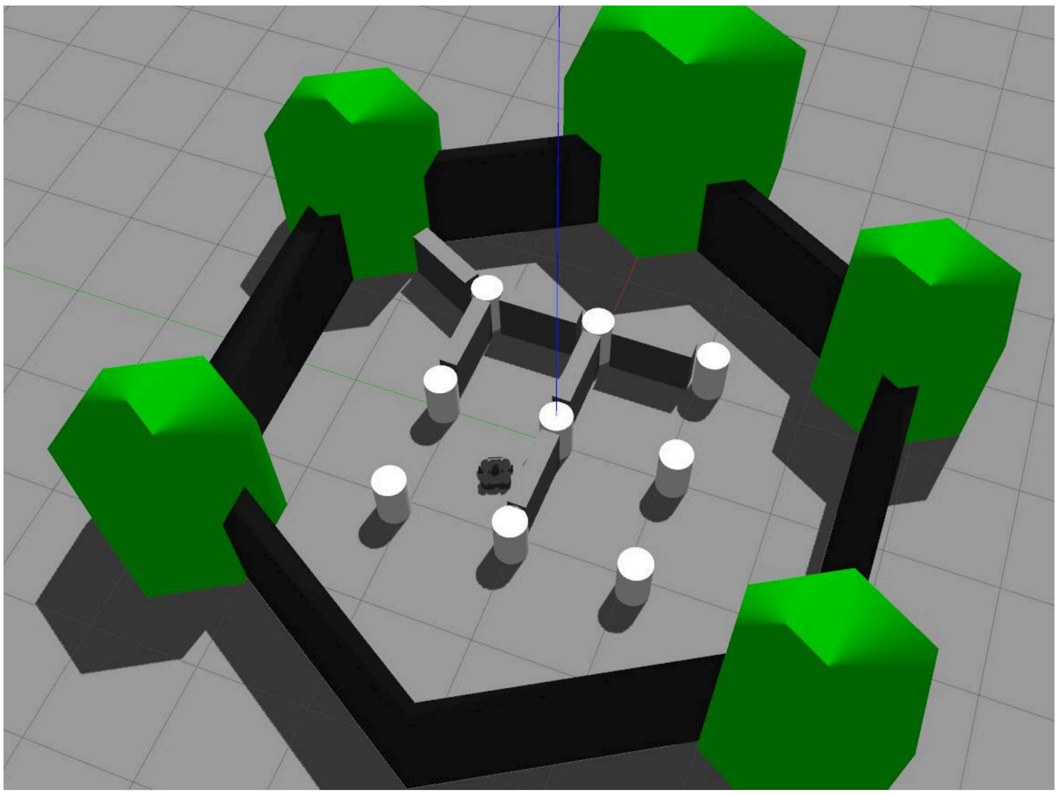


Figure 5.8.: Gazebo rendering of scenario #1 initial condition

5.9. Initially, the robot is placed in the position corresponding to the blue-white circle. Upon start, the policy PFT is attempted but fails, since the most direct passage towards the target is closed. The system then switches to PPX, and the robot starts to explore forward, since it does not detect the collision immediately given the limited prediction window required for the explorative policies (see sec. 5.4.3). As soon as the last sample of the trajectory is sufficiently close to the obstacle, corresponding to the keypoint #1, marked in green, the forward trajectory is invalidated (by TCO) and the robot is forced to move backwards, until it reaches #2, where it could choose between the branch going to south or to west, among which it chooses the one the latter, being closer to the target and since both corridors are unexplored. It then starts tracking the corridor towards #3, where all the trajectories towards north are colliding, again forcing the robot to move back, towards #4. At that point, the algorithm could choose between taking a trajectory to the south, with high distance-to-target penalty, or towards east, that has a lower value for such cost, but has an exploration cost due to the landmarks placed the first visit. Overall, the chosen tuning parameters favors the southward trajectory, even though for the philosophy of the algorithm it could in principle make the same sense to track the other way around close to

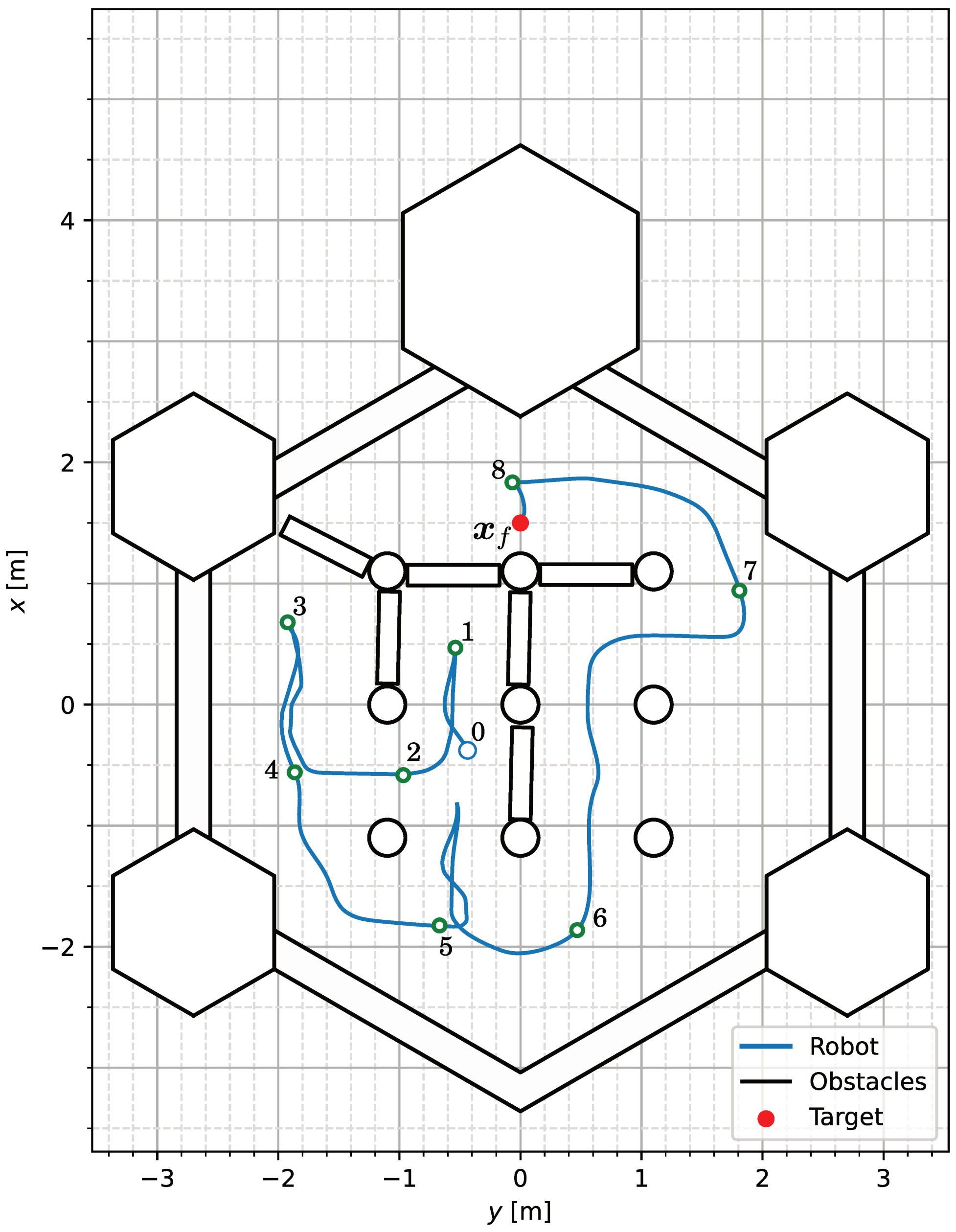


Figure 5.9.: Results of scenario #1: corridors

#2. From #4 then the path overtake the column on the south until it reaches #5, where the limited-horizon predicted trajectory finds no landmarks, since that branch is not explored, and a cheaper target cost. The cost function corresponding to the keypoint 5 is represented in figure 5.10, where it is easy to distinguish the cost related to the landmarks to that from the target-distance penalty. The robot then starts tracking the trajectory leading closer to the initial point, and it will require that the collision-free predicted paths are not only overlapping to landmarks, but also their total value exceeds the penalty of going back away from the target. This happens after a short distance, since the area around #0,#1 and #2 is heavily explored, and right after the robot is driven back to #5 and from there the algorithm takes the only unexplored path leading to #6. From #6 the algorithm proceeds with similar mechanics until #7, upon which a valid PFT trajectory is computed, which is followed until #8, that is the point on the GVD closer than the threshold required to switch to PFS and approach the target without the constraint of staying in the GVD.

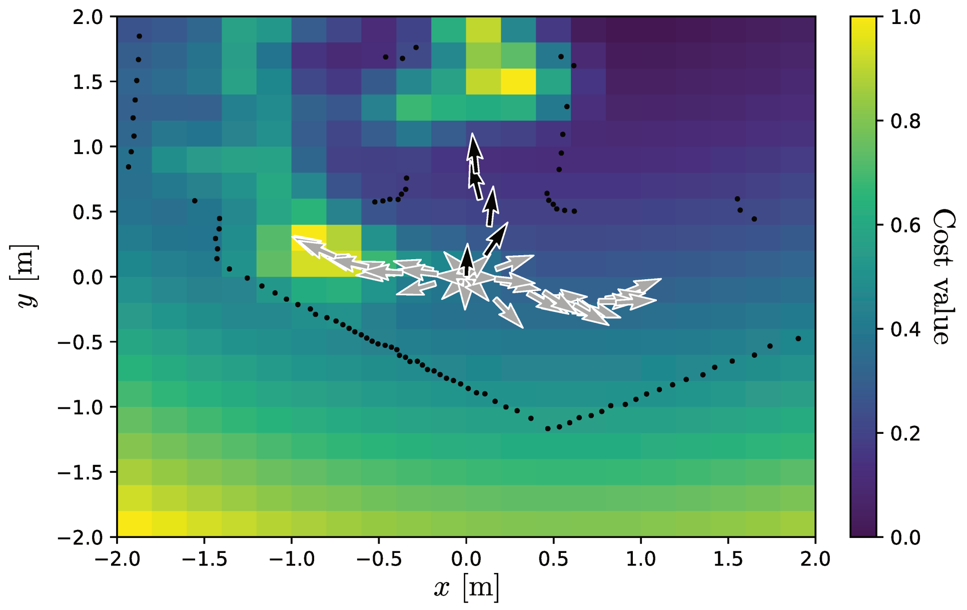


Figure 5.10.: Cost function value and predicted trajectories corresponding to keypoint 5. The selected trajectory is reported in black while the others are in gray.

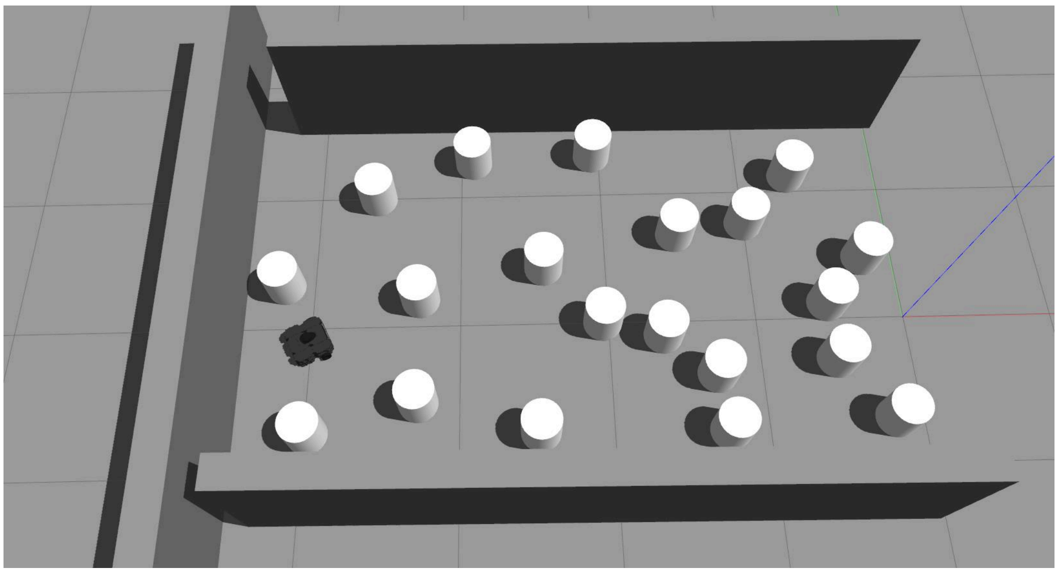


Figure 5.11.: Gazebo rendering of scenario #2 initial condition, sideways

5.7.2 Scenario #2: cluttered spaces

The first scenario is characterized by corridors that are easy to identify, large and somewhat regular. Figure 5.11 shows a different map cluttered with columns: the robot is able to pass through only some of the spaces between them, other passages are too narrow. Moreover, the configuration of the map is such to form dead-ends, and the algorithm will have to reactively identify these situations and cope with them. While this scenario is more challenging, the working mechanics is the same. A path predicted between two columns that are too close will be directly excluded by the algorithm, while the landmarks-based strategy will be able to cope with dead-ends.

We present the results of the simulation in figure 5.12, where the position tracked by the robot is reported in blue. Since clearly a PFT trajectory, if existed, would exceed the maximum number of steps, the algorithm directly starts with PFT. From the initial point, it is easy for to choose the trajectories leading directly to the target: some of the trajectories predicted from the initial angles ϑ_k will track the path passing laterally through the columns, but clearly the one leading southwards has less target penalty. Figure 5.13 report the Landscape function values for the robot placed in the keypoint #1 along with the predicted trajectories, that better shows what stated above. This snapshot also highlights that, according to the local knowledge of the robot, the branch identified by the selected trajectory (black arrows)

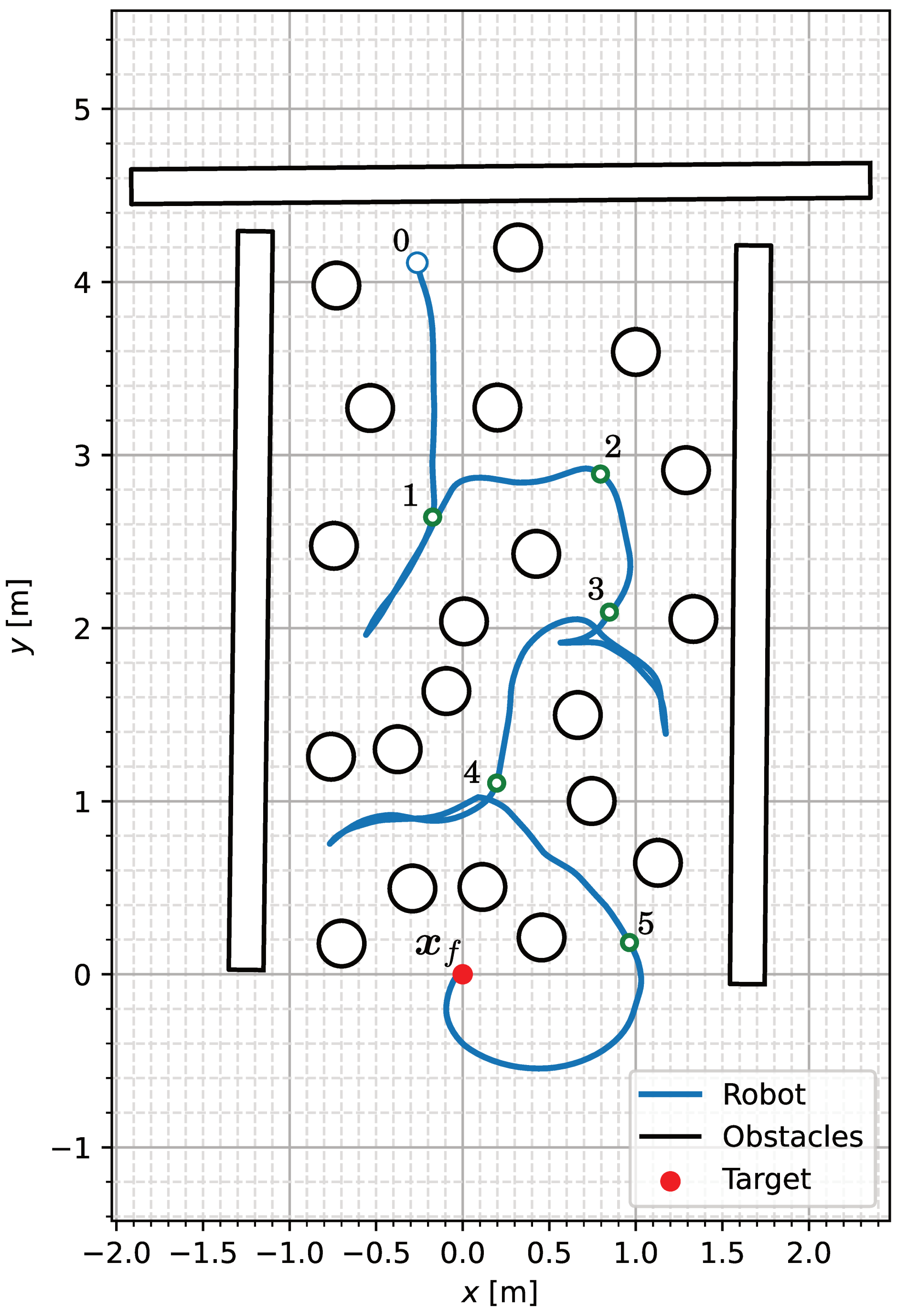


Figure 5.12.: Results of scenario #2: cluttered columns

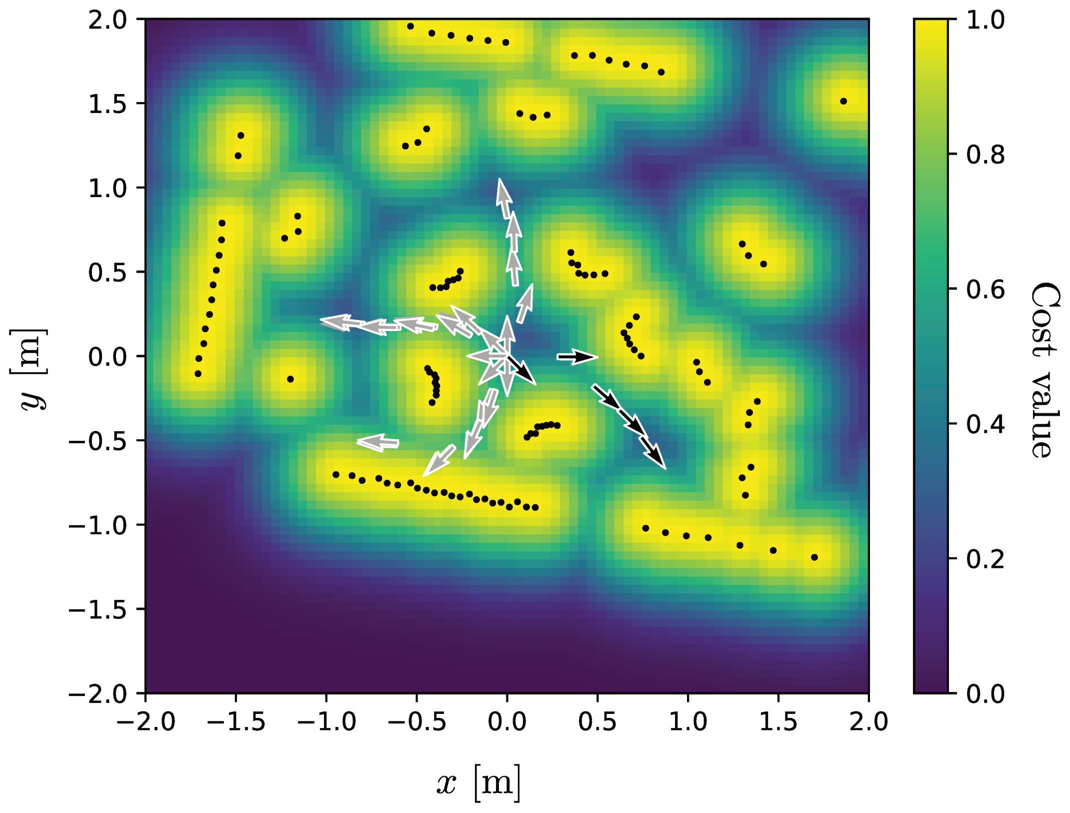


Figure 5.13.: Landscape values for keypoint #1.

may as well have an exit. Nevertheless, this is only true because a column occludes the view of the collision points of the wall behind, and this further motivates considering short-ranged predictions. From #1 then the left path is attempted, though it is quickly identified as a dead-end, so the robot is driven back to #1 and in this case the eastward trajectory is selected, given that the southward one is already explored, allowing reaching key-point #2. From #2 to #3 the path is predicted straightforwardly. In #3 then, the same situation as in #1 occurs, and the robot needs to explore the dead-end on the east side before getting back to #3 and subsequently reach #4. In #4 this occurrence repeats in the same way, even though in this case the difference in the cost between the east and west branches from #4 is mild: indeed, by repeating the same experiment, sometimes the branch towards #5 is directly taken. This randomicity derives from the impossibility of making long term predictions due to the local nature of the measurements available. Once in #5, The robot is then able to predict a PFT trajectory towards the target, which is represented in figure 5.14, along with the corresponding landscape value. Note that since no other obstacle is present on the right side of the figure, the value of the landscape function is zero and then the trajectory

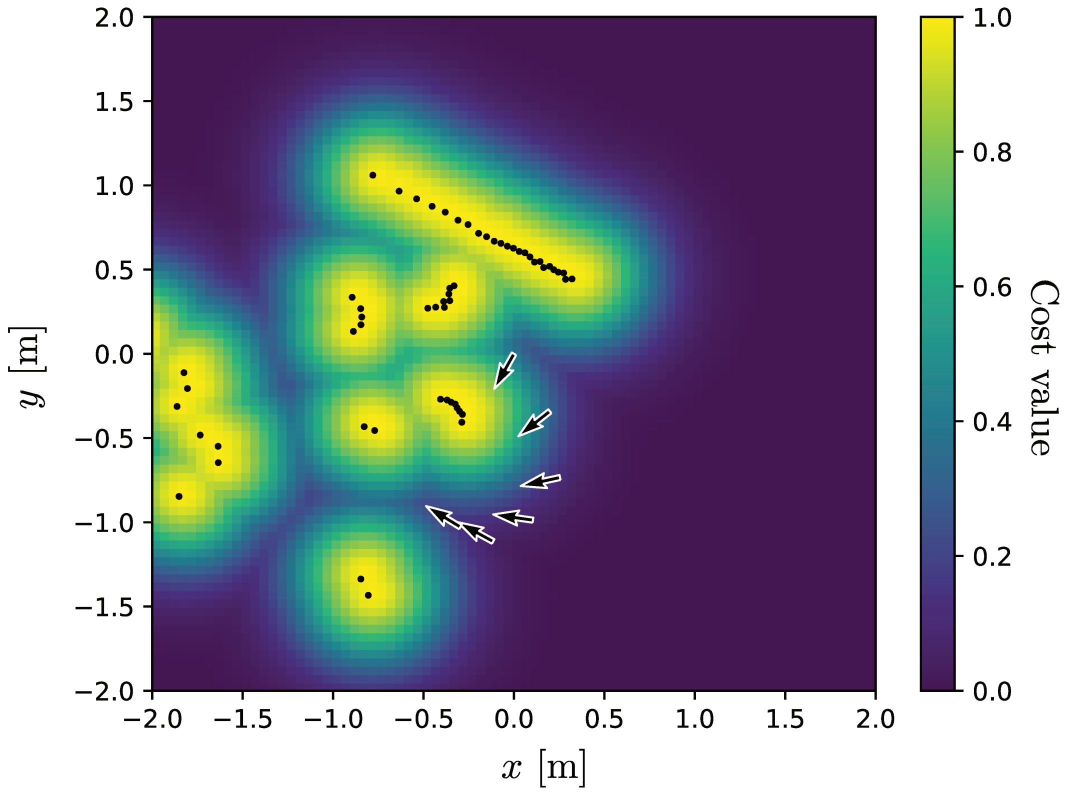


Figure 5.14.: Landscape value in #5 and valid PFT trajectory.

taken is precisely the radius of the landscape peak outside the collision space.

5.7.3 Scenario #3: target not in sight

We now present a scenario where the robot does not have an estimate of the target position and is tasked to only explore the environment. To this aim, we set the algorithm without a target point x_f, which corresponds to the event X-TIS being false. The map is the standard Turtlebot3 arena in the Gazebo simulator, the same used for scenario #1 without walls between columns. This allows the algorithm to have multiple choices to achieve the same goal of exploring each of the areas delimited by the columns and the arena's walls. The map and the results are reported in figure 5.15. The robot is initially placed in correspondence to the blue circle, the key-point #0. At $t = 0$ there only is the initial landmark ℓ_0 so every trajectory will have nearly the same cost value: given the local information available there is no prior reason to prefer one way or another. This may cause an ambiguity at first, causing the robot to travel back and forth a few times before settling for a

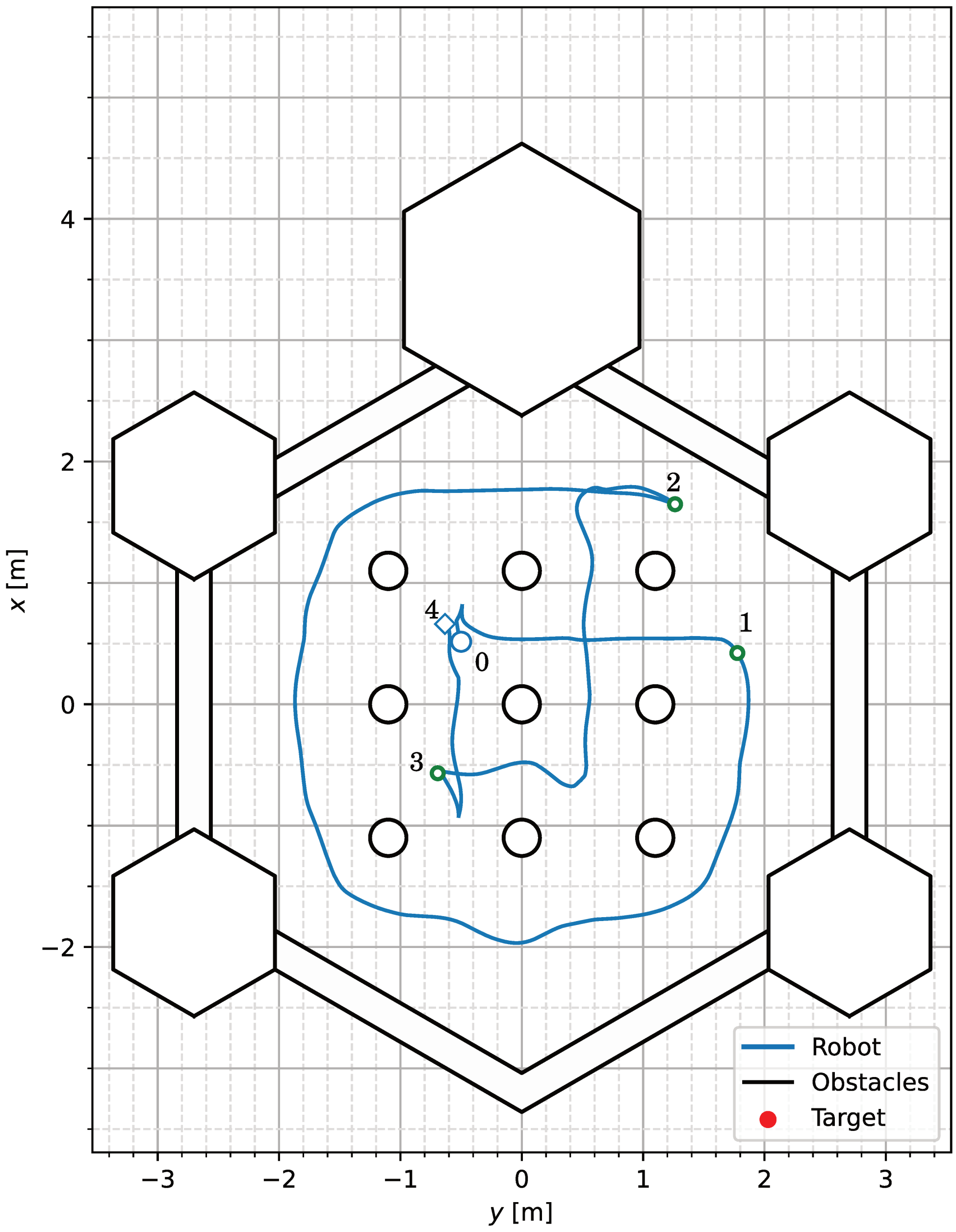


Figure 5.15.: Map and results for scenario #3

final choice. While this could be fixed with heuristics, this initial transient does not affect the final goal, so we neglect this problem in this context. In this case, the algorithm takes steadily the corridor on the right towards #1. From there, as far as exploration is concerned, there is no prior reason to move to the north or the south. However, the choice also depends on the ability for the robot to actually track the computed trajectory, given the underactuation of the unicycle model of the robot. In this case, the path leading southwards results faster to track as there is less curvature for the robot to keep up with. After that, as could be expected, the cheapest way to explore the map is by circumnavigating it through the outer corridors, as is reported in the map with the path from #1 to #2, since any other trajectory passing through the inner corridors would be closer to the landmarks set in the path from #0 to #1, so the cost would be higher, even if slightly. At #2, however, the algorithm cannot select the predicted trajectory advancing in the outer corridor, since it has a high cost since the area near #1 already has landmarks set. The robot then needs to move backwards and towards the inner area delimited by only columns. In crossing the path #0-#1 the trajectories has some penalty due to the explored branch, but less than the others, so the already-explored area is crossed in the shortest way. After that, it reaches #3, from where it initially attempts to explore the corridors on the west and on the south, shortly before receiving a high cost value due to the landmarks corresponding to the path #1-#2. The robot then has no other possibility than tracking the only unexplored area towards #0, reaching its final position (#4), marked with a blue diamond shape.

Remark 5.6. The scenario we just tested the PFX policy cannot be properly defined as local navigation: indeed, the entire map is explored. However, in correspondence of bifurcations the choice is completely determined by the cost value of each trajectory. This local behavior may not result in the highest efficiency in achieving the global goal. For example, when far from the explored area there is little penalty in overcoming a column on one side or another. This, instead, can result in completely different paths and the robot could travel across the same path multpile time before exploring new areas. As a matter of fact, the algorithm is not conceived as a global planner and there exist many tools in literature that perform exploration of a map in a more suitable way. Nevertheless, the example we provided in this section has to be interpreted as a limit case, while its proper application is overcoming a small set of obstacle, leaving the global planning for other and possibly more sophisticated strategies.

5.8 Conclusions

In this chapter we presented a policy-switching method to reactively navigate in unknown and unpredictable environments based on sensors only. The policies represent strategies to predict and select consecutive points of the GVD of the map through the iterative application of the NAPVIG algorithm introduced in section 3. The algorithm is tested in several scenarios, where it proves to efficiently solves local navigation with the limited amount of information provided by the on-board sensors.

6 Visual odometry for aerial phyisical interaction

6.1 Introduction

Visual odometry (VO) refers to the techniques utilizing camera images to determine the position and orientation of a vehicle. Specifically, the on-board camera, may it be a stereo RGB camera or a monocular depth camera (RGB-D) are exploited to obtain the relative motion between subsequent frames, as the agent moves in the environment. Possibly, visual odometry information can be integrated with inertial data from the IMU (Oskiper et al., 2007), (*Visual-inertial Odometry*, VIO), where sensors information are commonly fused exploiting Extended Kalman Filters (EKF) (Sirtkaya et al., 2013; Bloesch et al., 2015). However, purely relying upon on-board sensors inevitably produce drifts in the global position. SLAM-based methods (Grisetti et al., 2007), known as *Visual-SLAM*, can overcome this issue by keeping a global map of the environment (Kaess et al., 2008). However, in this work we restrict our attention to applications of visual odometry for aerial manipulation tasks, as introduced in chapter 1: in such contexts, a global estimate of the robot's position is not central, as it is only needed to feed the interaction wrench estimator, which captures only the differences in position rather its absolute one. In this light, we will avoid keeping a map of the environment, with the goal to save as much computational resource as possible.

Traditionally, visual odometry includes two steps: first, an algorithm extracts *features* from the point clouds (Gumhold et al., 2001; Alshawabkeh, 2020; Daniels et al., 2007), of which a matching is then determined. The most popular methods are SIFT (Lowe, 2004), SURF (Bay et al., 2006) and, more recently, ORB (Rublee et al., 2011), from which originated a variety of derivative works, the most recent ones are also based on machine learning techniques (Behl et al., 2019; Shen et al., 2019; Gojcic et al., 2019; Yang et al., 2020). After having determined the corresponding feature points, it is relatively easy to obtain an estimate of the rototranslation between those points, with classic nonlinear optimization methods.

Following the paradigm that inspired also the other works in this thesis, we aim at skipping the preprocessing of feature extraction and matching, and work directly on the raw data. Feature-less point cloud matching is a classical problem in computer graphics and it is known as *point cloud registration* (Huang et al., 2021a). The classic algorithm to address this problem is the *Iterative Closest Point* (ICP) (Besl and McKay, 1992; Rusinkiewicz and Levoy, 2001), which iteratively finds the best correspondence between every pair of points of the two point clouds and then computes the transformation needed to align such points. Due to its iterative nature and the need to align every pair of points, it is highly computational demanding and is hardly suitable to meet real-time constraints. More recently, deep learning methods have been employed to obtain this goal, where the input of the learning model are the two point clouds and the output is the transformation between the two (Wang et al., 2019; Elbaz et al., 2017). Alternatively, neural networks and optimization can be combined to obtain hybrid methods (Huang et al., 2020; Choy et al., 2020; Yuan et al., 2020). The main limitations of those methods is that computationally expensive strategies are required to cope with unpredicted variations in the noise distribution, outliers, etc.(Huang et al., 2021b).

The method we propose is an optimization strategy that leverages a uniform 3D space representation that takes inspiration from the concept of the Landscape function from chapter 3 to define a matching index between two point clouds that works directly on the raw measurements. To ensure maximum efficiency, the method is entirely formulated to exploit the properties of the Lie groups, of which we will provide a brief overview.

6.2 Mathematical preliminaries: Lie groups

Throughout this dissertation the set of rotations and transformations, $\mathbb{SO}(n)$ and $\mathbb{SE}(n)$, $n = 2, 3$, have been taken into account indirectly, without examining its properties, as they were used for their straightforward application of simply representing and combining poses. In this chapter, instead, they gain central importance, as they will be the objective of optimization. For this reason and with the aim of uniforming the notation, we introduce the basic facts of Lie theory that will be referred to later on.

The Lie group Informally, a Lie group is a smooth manifold whose elements also satisfy the group axioms. Specifically, given the set $\mathcal{G}$ together with a composition operation $\circ$, it is a *Lie group* if $\mathcal{G}$ is a smooth manifold and all its elements $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathcal{G}$ fulfill the following:

- Closure: $\boldsymbol{x} \circ \boldsymbol{y} \in \mathcal{G}$
- Existence of identity: $\exists \mathbf{1} \in \mathcal{G}$ such that $\mathbf{1} \circ \boldsymbol{x} = \boldsymbol{x}$, $\boldsymbol{x} \in \mathcal{G}$.
- Existence of inverse: $\exists \boldsymbol{x}^{-1} \in \mathcal{G}$ such that $\boldsymbol{x}^{-1} \circ \boldsymbol{x} = \mathbf{1}$
- Associativity: $(\boldsymbol{x} \circ \boldsymbol{y}) \circ \boldsymbol{z} = \boldsymbol{x} \circ (\boldsymbol{y} \circ \boldsymbol{z})$.

The group properties of a Lie group essentially ensure that the composition of two elements of the manifold remains in the manifold. The key importance of this property is that it highlights the fact that the only operation allowed is the composition $\circ$, that is conceptually separated from the *group action*. Lie groups elements can be associated to transformation acted upon different sets, e.g. rotate and scale points, etc. Formally, given a Lie group $\mathcal{G}$ and a generic set $\mathcal{V}$, the *action* of $\boldsymbol{x} \in \mathcal{G}$ on $\boldsymbol{v} \in \mathcal{V}$ is the function:

$$\cdot \; : \; \mathcal{G} \times \mathcal{V} \; \rightarrow \; \mathcal{V} \; : \; (\boldsymbol{x}, \boldsymbol{v}) \mapsto \boldsymbol{x} \cdot \boldsymbol{v} \tag{6.1}$$

that need to satisfy the following two properties:

- Identity: $\mathbf{1} \cdot \boldsymbol{v} = \boldsymbol{v}$
- Compatibility: $(\boldsymbol{x} \circ \boldsymbol{y}) \cdot \boldsymbol{v} = \boldsymbol{x} \cdot (\boldsymbol{y} \cdot \boldsymbol{v})$

In table 6.1 we report examples of commonly used Lie groups with the corresponding group action. Note, in particular, that the vector space $\mathbb{R}^n$ can be seen as a Lie group, where the vector sum realizes both composition and group action.

Tangent spaces The smoothness of the manifold allows us to consider *tangent spaces*, that is the space of all possible velocities that a point $\boldsymbol{x}(t)$ moving in the manifold $\mathcal{G}$. We denote the tangent space of $\mathcal{G}$ at $\boldsymbol{x}$ as $T_{\boldsymbol{x}}\mathcal{G}$. Particularly significant is the tangent space of $\mathcal{G}$ at the identity $\mathbf{1}$, which is

Lie Group	Target set	Action
$(\mathbb{R}^n, +)$	$\mathbb{R}^n$	$\boldsymbol{x} \cdot \boldsymbol{v} = \boldsymbol{x} + \boldsymbol{v}$
$\mathbb{SO}(n)$	$\mathbb{R}^n$	$\boldsymbol{R} \cdot \boldsymbol{v} = \boldsymbol{R}\boldsymbol{v}$
$\mathbb{SE}(n)$	$\mathbb{R}^n$	$\boldsymbol{T} \cdot \boldsymbol{v} = \boldsymbol{R}\boldsymbol{v} + \boldsymbol{t}$
$(\mathbb{C}, \circ)$	$\mathbb{R}^2$	$c \cdot \boldsymbol{v} = c \circ \boldsymbol{v}$
$\mathbb{H}$	$\mathbb{R}^3$	$\boldsymbol{q} \cdot \boldsymbol{v} = \boldsymbol{q} \circ \boldsymbol{v} \circ \boldsymbol{q}^*$

Table 6.1.: Common group actions

referred to as the *Lie algebra* of $\mathcal{G}$ and denoted with $\mathfrak{g} = T_1\mathcal{G}$. Every Lie group has an associated Lie algebra. We will avoid introducing the formal mathematical properties characterizing a Lie algebra, which would require introducing the Lie brackets product, since those concepts are not significant for the purposes of this work. The interesting property about the Lie algebra is that it is a linear vector space: we consider the basis vectors $\boldsymbol{E}_i$ such that

$$\mathfrak{g} = \text{span}\{\boldsymbol{E}_1, \dots, \boldsymbol{E}_M\}, \quad M \in \mathbb{N}, \tag{6.2}$$

where M is the dimension of $\mathfrak{g}$ and corresponds, intuitively, to the number of degrees of freedom of the manifold $\mathcal{G}$. As a result of (6.2), we can express the elements of $\mathfrak{g}$ as a linear combination of its basis elements. This creates an isomorphism between $\mathfrak{g}$ and $\mathbb{R}^m$, which we refer as *hat* function, along which we consider its inverse *vee*, as follows:

$$\begin{aligned} \text{Hat} &: \mathbb{R}^m \to \mathfrak{g} \quad \boldsymbol{\tau} \mapsto \boldsymbol{\tau}^\wedge = \sum_{i=1}^{M} \tau_i \boldsymbol{E}_i \\ \text{Vee} &: \mathfrak{g} \to \mathbb{R}^m \quad \boldsymbol{\tau}^\wedge \mapsto (\boldsymbol{\tau}^\wedge)^\vee = \boldsymbol{\tau} = \sum_{i=1}^{M} \tau_i \boldsymbol{e}_i \end{aligned} \tag{6.3}$$

where $\boldsymbol{e}_i$ are the vectors of the canonical base of $\mathbb{R}^m$. Then, instead of $\mathfrak{g}$, we can then directly work on $\mathbb{R}^m$, which is more convenient.

Elements of the Lie algebra are associated to the elements of the Lie group according to the *exponential map* $\exp()$. The elements of the Lie algebra can be considered as the velocities that keep an element $\boldsymbol{x} \in \mathcal{G}$ inside the manifold. Intuitively, given an element of the Lie algebra $\boldsymbol{\tau}^\wedge$, the $\exp$ map is the element of $\mathcal{G}$ that is obtained applying the constant velocity $\boldsymbol{\tau}^\wedge$ for the time unit. This can be seen as a mathematical way to move an element of the Lie group along the directions of the Lie algebra. Moreover, the exponential map is the only map that preserves the Lie algebra structure and composition law of the Lie group elements, making it the most convenient way to move around the manifold. In addition, it is important to note that the exponential map is invertible, so the inverse of the exponential map, called the logarithmic map, can be used to reverse the movement. This can be used to go back to the initial point of the movement or to a prior point in the Lie group. The exponential map and its inverse can be formally written as:

$$\begin{aligned} \exp: &\quad \mathfrak{g} \to \mathcal{G} \quad \boldsymbol{\tau}^{\wedge} \mapsto \boldsymbol{x} = \exp(\boldsymbol{\tau}^{\wedge}) \\ \log: &\quad \mathcal{G} \to \mathfrak{g} \quad \boldsymbol{x} \mapsto \boldsymbol{\tau}^{\wedge} = \log(\boldsymbol{x}) \end{aligned} \tag{6.4}$$

For convenience, we can also consider a direct mapping between $\mathbb{R}^m$ and $\mathcal{G}$ and vice versa by considering the composition between the exponential/logarithm and the hat/vee isomorphism, as follows:

$$\begin{aligned} \mathrm{Exp}: &\quad \mathbb{R}^m \to \mathcal{G} \quad \boldsymbol{\tau} \mapsto \boldsymbol{x} = \mathrm{Exp}(\boldsymbol{\tau}) = \exp(\boldsymbol{\tau}^{\wedge}) \\ \mathrm{Log}: &\quad \mathcal{G} \to \mathbb{R}^m \quad \boldsymbol{x} \mapsto \boldsymbol{\tau} = \mathrm{Log}(\boldsymbol{x}) = \log(\boldsymbol{x})^{\vee} \end{aligned} \tag{6.5}$$

Derivatives on Lie groups The aim of this introduction is to present the basic tools for optimization on Lie groups, for which we need to extend the concept of derivative to functions of elements of Lie groups. In this sense, we introduce the *plus* and *minus* operators:

$$\begin{aligned} \oplus &: \boldsymbol{y} = \boldsymbol{x} \oplus \boldsymbol{\tau} := \boldsymbol{x} \circ \mathrm{Exp}(\boldsymbol{\tau}) \\ \ominus &: \boldsymbol{\tau} = \boldsymbol{x} \ominus \boldsymbol{y} := \mathrm{Log}(\boldsymbol{x}^{-1} \circ \boldsymbol{y}) \end{aligned} \tag{6.6}$$

These operators are to be interpreted in the following sense: the $\oplus$ increments element $\boldsymbol{x}$ by applying a constant velocity specified by $\boldsymbol{\tau}$ for the unit of time, while the $\ominus$, conversely, find the velocity that needs to be applied for a time unit to pass from $\boldsymbol{x}$ to $\boldsymbol{y}$.

This definition allows us to seamlessly translate the definition of derivative on vector spaces to Lie groups. We recall that the Jacobian for a vector function $\boldsymbol{f}: \mathbb{R}^m \to \mathbb{R}^n$ is:

$$\boldsymbol{J_f} = \frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix} \in \mathbb{R}^{m \times n} \tag{6.7}$$

and its columns can be interpreted as the directional derivative of $\boldsymbol{f}(\boldsymbol{x})$ in the direction of the i-th element of the canonical basis $\boldsymbol{e}_i$:

$$\frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial x_i} = \lim_{h \to 0} \frac{\boldsymbol{f}(\boldsymbol{x} + h\boldsymbol{e}_i) - \boldsymbol{f}(\boldsymbol{x})}{h} \in \mathbb{R}^m \tag{6.8}$$

The concept of perturbation for the directional derivative can be extended to Lie groups by considering it a perturbation in the Lie algebra. By resorting to the $\oplus$, $\ominus$ notation, the generalization is straightforward. Given two Lie groups $\mathcal{G}$ and $\mathcal{H}$ and a function $\boldsymbol{f} : \mathcal{G} \to \mathcal{H}$, the Jacobian of $\boldsymbol{f}$ with respect to the Lie group element $\boldsymbol{x}$ is:

$$\begin{aligned} \frac{\mathrm{D}\boldsymbol{f}(\boldsymbol{x})}{\mathrm{D}\boldsymbol{x}} &= \lim_{\boldsymbol{\tau} \to 0} \frac{\boldsymbol{f}(\boldsymbol{x} \oplus \boldsymbol{\tau}) \ominus \boldsymbol{f}(\boldsymbol{x})}{\boldsymbol{\tau}} \in \mathbb{R}^{n \times m} && (6.9) \\ &= \lim_{\boldsymbol{\tau} \to 0} \frac{\mathrm{Log}(\boldsymbol{f}(\boldsymbol{x})^{-1} \circ \boldsymbol{f}(\boldsymbol{x} \circ \mathrm{Exp}(\boldsymbol{\tau})))}{\boldsymbol{\tau}} && (6.10) \end{aligned}$$

Note that the fraction by vector $\boldsymbol{\tau}$ has to be interpreted individually for each component of $\boldsymbol{\tau}$. Moreover, the limit in (6.10) corresponds to the standard derivative of the function $\boldsymbol{g} : \boldsymbol{R}^m \to \boldsymbol{R}^n$

$$\boldsymbol{g}(\boldsymbol{\tau}) := \mathrm{Log}(\boldsymbol{f}(\boldsymbol{x})^{-1} \circ \boldsymbol{f}(\boldsymbol{x} \circ \mathrm{Exp}(\boldsymbol{\tau}))) \tag{6.11}$$

evaluated for $\boldsymbol{\tau} = 0$:

$$\frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \left.\frac{\partial \boldsymbol{g}(\boldsymbol{\tau})}{\partial \boldsymbol{\tau}}\right|_{\boldsymbol{\tau}=\mathbf{0}} \tag{6.12}$$

Finally, given this relation, it is easy to verify that the chain rule also applies directly in the case of function between Lie groups.

6.2.1 The group of rototranslations

We now provide specific results regarding the group of 3D rigid motion $\mathbb{SE}(3)$, whose elements are identified by the matrix:

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \mathbf{0} & 1 \end{bmatrix} \in \mathbb{SE}(3) \subset \mathbb{R}^{4 \times 4} \tag{6.13}$$

where $\boldsymbol{R} \in \mathbb{SO}(3)$ and $\boldsymbol{t} \in \mathbb{R}^3$ are the corresponding rotation matrix and translation. The Lie algebra corresponding to $\mathbb{SE}(3)$ is denoted with $\mathfrak{se}(3)$ and is the group of matrices of the following form:

$$\boldsymbol{\tau}^\wedge = \begin{bmatrix} [\boldsymbol{\theta}]_\times & \boldsymbol{\rho} \\ \mathbf{0} & 0 \end{bmatrix} \in \mathfrak{se}(3), \quad \boldsymbol{\tau} = \begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\theta} \end{bmatrix} \in \mathbb{R}^6, \tag{6.14}$$

where $\boldsymbol{\rho} \in \mathbb{R}^3$ and $\boldsymbol{\theta} \in \mathbb{R}^3$ can be interpreted as the linear and angular velocities, respectively. In this context we avoid introducing the homogeneous representation of a point $\mathbb{R}^3$ in order to define the roto-translation as a single matrix multiplication, and we define the *group action* of $\mathbb{SE}(3)$ onto $\boldsymbol{p} \in \mathbb{R}^3$ as an external operation, also emphasizing the different nature of the group action than the composition:

$$\cdot \; : \; \mathbb{SE}(3) \times \mathbb{R}^3 \to \mathbb{R}^3 \; : \; \boldsymbol{T} \cdot \boldsymbol{p} = \boldsymbol{R}\boldsymbol{p} + \boldsymbol{t} \tag{6.15}$$

The group action $\boldsymbol{x} \cdot \boldsymbol{v}$, for fixed $\boldsymbol{v}$, can be seen as a function of $\boldsymbol{x}$. We now provide a result that will be used in the next sections: the Jacobian of the group action of the transformations $\mathbb{SE}(3)$, that can be proven to be:

$$\frac{\partial \boldsymbol{T} \cdot \boldsymbol{v}}{\partial \boldsymbol{T}} = \begin{bmatrix} \boldsymbol{R} & -\boldsymbol{R}\,[\boldsymbol{v}]_\times \end{bmatrix} \in \mathbb{R}^{3\times 6} \tag{6.16}$$

where $\boldsymbol{R} \in \mathbb{SO}(3)$ is the rotation matrix of the transformation.

6.3 Problem formulation

We consider an aerial robot endowed with a lightweight end-effector rigidly attached to a hexarotor with tilted propellers. In view of the spirit of full autonomy that characterizes this thesis, we aim at approaching the problem of wrench estimation without the knowledge of a precise estimate of the robot pose. The system is instead equipped with a depth camera that, with sample time $T_c \in \mathbb{R}$, it measures the point cloud of the environment within its field of view, namely, at time $t_k = kT_c$, $k \in \mathbb{N}$, it obtains the set:

$$\mathcal{P}_k = \{\bar{\boldsymbol{p}}_{k,h}, \; h = 1, \ldots, M\} \subset \mathbb{R}^3 \tag{6.17}$$

The points $\bar{\boldsymbol{p}}_{k,h}$ compose a 3D representation of the visible objects. Analogously to the previous chapters, we consider a *robot frame* at generic, continuous time $t \in \mathbb{R}$, namely $\mathcal{F}_t \in \mathbb{SE}(3)$, and the *measurement frame* $\mathcal{F}_k$, corresponding to the robot frame corresponding to the measurement sample t_k. In general, we denote with $\mathcal{F}_a \in \mathbb{SE}(3)$ a frame named a, and with $\boldsymbol{T}_b^a \in \mathbb{SE}(3)$ we denote the transformation between frames a and b. We distinguish the notation of these two concepts even if they both refer to elements of the same Lie group to distinguish where such element should be interpreted as a coordinate frame or a transformation. Given two sample times $t_1 \in \mathbb{R}$ and $t_2 \in \mathbb{R}$, we aim at finding the transformation $\boldsymbol{T}_{t_2}^{t_1}$ that transforms points in $\mathcal{F}_{t_1}$ to points in $\mathcal{F}_{t_2}$. For convenience of notation, we now will refer to quantities relative to t_1, $\mathcal{P}_{t_1}$, $\boldsymbol{p}_{k_1,h}$, $\mathcal{F}_{t_1}$ and so on with $\mathcal{P}_1$, $\boldsymbol{p}_{1,h}$, $\mathcal{F}_1$, and similarly for t_2.

6.4 Feature-less motion estimation

The classic approach to solve this problem is the *motion from features* strategy and is based on the two following steps:

- *Features extraction*: from the point cloud $\mathcal{P}_h$ is extracted a subset $\tilde{\mathcal{P}}_h$ whose elements $\tilde{\boldsymbol{p}}_{k,h}$ are associated to a *feature*, namely a unique and persistent identifier.

- *Motion estimation*: the motion $\boldsymbol{T}_{t_2}^{t_1}$ is estimated from the feature matches between the two sets of features extracted from two different measurements $\tilde{\mathcal{P}}_1$ and $\tilde{\mathcal{P}}_2$.

The method we propose aims at skipping the feature extraction and matching parts and work directly on the raw measurements. Ideally, what we seek to obtain is the transformation $\hat{\boldsymbol{T}}$ such that:

$$\bar{\boldsymbol{p}}_{2,h} \simeq \hat{\boldsymbol{T}} \bar{\boldsymbol{p}}_{1,h}, \quad \forall h = 1, \dots, M \tag{6.18}$$

Clearly, this requires that all the points in the two point clouds $\mathcal{P}_1$ and $\mathcal{P}_2$ are ordered such that there is perfect matching between points, which in general is not the case. The classical approach is to extract a subsampling of points of which can be established the correspondence through feature matching. In this work, instead, we introduce a *matching index* that regards the entire point cloud. Specifically, given two point clouds $\mathcal{P}_a$ and $\mathcal{P}_b$, we aim at defining an index $J(\mathcal{P}_a, \mathcal{P}_b)$ that expresses how much the two point clouds are similar, and we will resort to the maximization of this index to estimate the transformation $\hat{\boldsymbol{T}}$ between two point clouds.

6.4.1 3D Landscape function

The basic idea is to consider a generalized version of the Landscape function introduced in chapter 3, exploiting its properties of homogeneity in the representation of space. First, we provide the definition of *n-dimensional Landscape function* that is a straightforward generalization. Note that the definition of the raw Landscape function provided in chapter 4 does not change if the set $\mathcal{A}$ is a subset of $\mathbb{R}^n$. We only change the notation to specify that the vectors are to be interpreted as elements of $\mathbb{R}^n$

$$ {}^n\check{\mathcal{L}}_{\mathcal{A}}(\boldsymbol{x}) := \max_{\boldsymbol{y}\in\mathcal{A}} e^{-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2\sigma^2}} \tag{6.19} $$

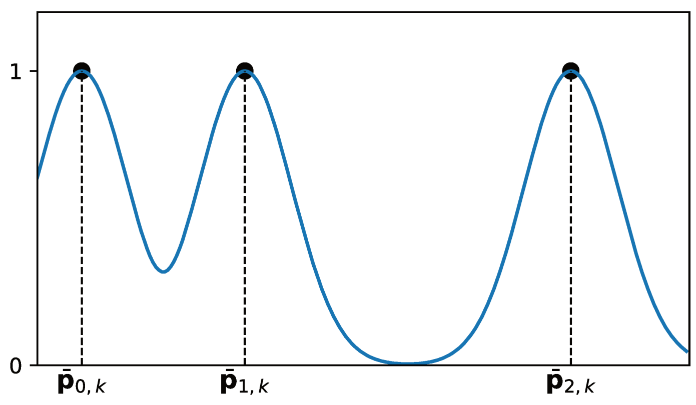
The only slight difference is in the definition of the smooth function:

$$ {}^n\mathcal{L}_{\mathcal{A}}(\boldsymbol{x}) = \int_{\mathbb{R}^n} {}^n\check{\mathcal{L}}_{\mathcal{A}}(\boldsymbol{w})\kappa(\boldsymbol{x}-\boldsymbol{w})\,\mathrm{d}\boldsymbol{w} \tag{6.20} $$

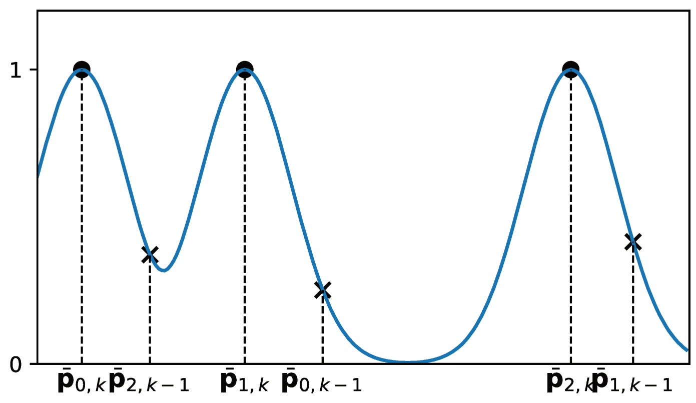
In the reminder of this section we will consider ${}^3\mathcal{L}_{\mathcal{A}}$, and we will drop the dependency on the dimensionality for clarity of presentation.

The idea is the following: we consider the landscape function on the second point cloud $\mathcal{P}_2$, and we evaluate it in the points of $\mathcal{P}_1$. To explain this concept and why it is meaningful, we refer to figure 6.1 reporting a monodimensional equivalent. Fig. 6.1a represents the three point of $\mathcal{P}_2$ in this example and the relative landscape function in blue. In 6.1b we show the points in $\mathcal{P}_1$ and the corresponding value in $\mathcal{L}_{\mathcal{P}_2}$, marked with a cross. It is clear that, the higher the value, the higher the correspondence between the point of $\mathcal{P}_1$ to one point of $\mathcal{P}_2$. If we transform the points in $\mathcal{P}_2$ by the same transformation $\boldsymbol{T} \in \mathbb{SE}(3)$, we can adjust the overall value, as exemplified in 6.1c. We formalize this concept by first defining the matching index between two point clouds:

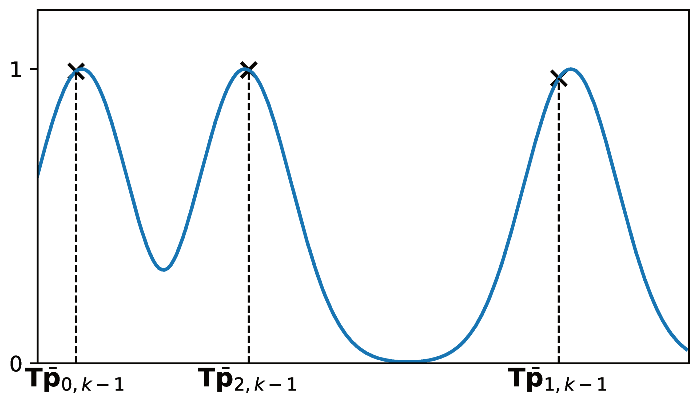
$$ J_{\mathcal{P}}(\mathcal{P}_a, \mathcal{P}_a) = \frac{1}{M} \sum_{\bar{\boldsymbol{p}}\in\mathcal{P}_b} \mathcal{L}_{\mathcal{P}_a}(\bar{\boldsymbol{p}}) \tag{6.21} $$



(a) Landscape for point cloud $\mathcal{P}_2$



(b) Landscape $\mathcal{L}_{\mathcal{P}_2}$ evaluated in the points of $\mathcal{P}_1$



(c) Landscape $\mathcal{L}_{\mathcal{P}_2}$ evaluated in the points of $\mathcal{P}_1$, transformed by the correct $\hat{\boldsymbol{T}}$

Figure 6.1.: Landscape function in 1D equivalent example

where $M = |\mathcal{P}_a|$ is the number of elements of the point clouds. Given the characteristics of the Landscape function, ideally, if two point clouds coincide, namely $\mathcal{P}_a \equiv \mathcal{P}_b$, it would hold:

$$J_{\mathcal{P}}(\mathcal{P}_a, \mathcal{P}_b) \simeq 1 \tag{6.22}$$

where the $\simeq$ sign is due to the fact that the maximum value of each peak can be slightly different from 1 due to the smoothing convolution. Figure 6.1 also shows one of the advantages of considering this representation: the point cloud measured by the sensor may often have regions where the points are denser than others. If, for example, the sum operation was used instead of the max, the areas with more concentration of samples would be given more sensitivity. This could lead to skewed results, as the regions with the highest concentration of samples could be over-represented in comparison to those with less dense samples. The landscape function, instead, guarantees homogeneity and consistency in the optimality index. It ensures that, no matter the density or distribution of the samples, all areas are treated equally, providing an accurate and reliable representation of the data.

6.4.2 Optimization on Lie groups

We now exploit the matching index $J_{\mathcal{P}}$ to set up an optimization problem to estimate the best transformation $\boldsymbol{T} \in \mathbb{SE}(3)$ that transforms one point cloud to the other, corresponding to the roto-translation that the vehicle has tracked between t_1 and t_2. First, we define the *predicted* point cloud as:

$$\boldsymbol{T} \cdot \mathcal{P} = \{\boldsymbol{T} \cdot \bar{\boldsymbol{p}}, \quad \boldsymbol{p} \in \mathcal{P}\} \tag{6.23}$$

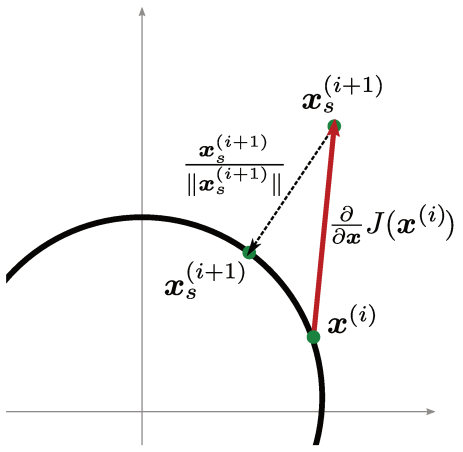
As classically done, the point clouds we are going to compare are $\mathcal{P}_2$ with the transformed version of $\mathcal{P}_1$, namely, we consider:

$$J_{1,2}(\boldsymbol{T}) := J_{\mathcal{P}}(\mathcal{P}_2, \boldsymbol{T} \cdot \mathcal{P}_1) := \frac{1}{M} \sum_{\bar{\boldsymbol{p}} \in \mathcal{P}_1} \mathcal{L}_{\mathcal{P}_2}(\boldsymbol{T} \cdot \bar{\boldsymbol{p}}) \tag{6.24}$$

We then aim at finding the best transformation $\boldsymbol{T}^*$ that produces the highest matching between $\boldsymbol{T}^* \cdot \mathcal{P}_1$ and $\mathcal{P}_2$. Formally:

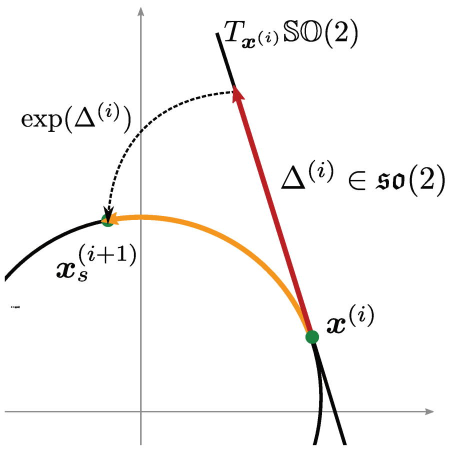
$$\boldsymbol{T}^* = \underset{\boldsymbol{T}\in\mathbb{SE}(3)}{\operatorname{argmax}}\; J_{1,2}(\boldsymbol{T}) = \underset{\boldsymbol{T}\in\mathbb{SE}(3)}{\operatorname{argmax}}\; \frac{1}{M}\sum_{\bar{\boldsymbol{p}}\in\mathcal{P}_1} \mathcal{L}_{\mathcal{P}_2}(\boldsymbol{T}\cdot\bar{\boldsymbol{p}}) \tag{6.25}$$

This is a non-linear optimization problem constrained to $\mathbb{SE}(3)$. The immediate approach to solve this problem is to resort a standard gradient descent considering a parametrization of $\boldsymbol{T}$ with a position $\boldsymbol{t} \in \mathbb{R}^3$ and a quaternion $\boldsymbol{q} \in \mathbb{S}^3 \subset \mathbb{R}^4$. While the quaternion can be seen as an euclidean vector in $\mathbb{R}^4$, to represent rotations it needs to have unit norm. In this regard, note that $\mathbb{R}^3 \times \mathbb{S}^3$ is isomorphic to $\mathbb{SE}(3)$, but $\mathbb{R}^3 \times \mathbb{R}^4$ is not. In general, if working on $\mathbb{R}^3 \times \mathbb{R}^4$, while updating the current estimate to a new one in the direction of the gradient of the cost function, there are no guarantees that the new quaternion is also unit norm and thus meaningfully represent a transformation. Figure 6.2a depicts this aspect in the simplified case of 2D rotations: $\boldsymbol{x}^{(i)}$ is an element of the unit circle $\mathbb{S}^1$, represented in black. The red arrow represents the direction of the gradient of the cost function J in $\boldsymbol{x}^{(i)}$ and the optimization step (we assume a unitary gain for simplicity), from which the algorithm would identify $\boldsymbol{x}_s^{(i+1)}$ as next candidate estimate, which is in general not in the unit circle. The easiest approach is to allow exiting temporarily from the manifold and then re-projecting onto it by normalizing the obtained value, so that $\boldsymbol{x}^{(i+1)} = \frac{\boldsymbol{x}_s^{(i+1)}}{\|\boldsymbol{x}_s^{(i+1)}\|}$. The method we will adopt, instead, will leverage the Lie group theory that we briefly overviewed in the last section. To this aim, the gradient must then be interpreted as a derivative in



$\boldsymbol{x}_s^{(i+1)}$
$\frac{\boldsymbol{x}_s^{(i+1)}}{\|\boldsymbol{x}_s^{(i+1)}\|}$
$\frac{\partial}{\partial \boldsymbol{x}} J(\boldsymbol{x}^{(i)})$
$\boldsymbol{x}_s^{(i+1)}$
$\boldsymbol{x}^{(i)}$

(a) $\mathbb{R}^2$



$T_{\boldsymbol{x}^{(i)}}\mathbb{SO}(2)$
$\exp(\Delta^{(i)})$
$\Delta^{(i)} \in \mathfrak{so}(2)$
$\boldsymbol{x}_s^{(i+1)}$
$\boldsymbol{x}^{(i)}$

(b) $\mathbb{SO}(2)$

Figure 6.2.: Optimization step in the euclidean space and on the manifold, example of the 2D rotations.

the Lie group sense. Before looking at the mathematical details, we complete the 2D example with the comparison to the case where the optimization step is directly performed in the manifold. The main difference lies in the fact that the gradient, intended in the Lie group sense, is an element of the Lie algebra and thence belonging to the tangent space of $\mathbb{SO}(2)$ (in this case depicted with the isomorphic $\mathbb{S}^1$) in $\boldsymbol{x}^{(i)}$, $T_{\boldsymbol{x}^{(i)}}\mathbb{SO}(2)$, represented with a black line in figure 6.2b. The increment $\Delta^{(i)} = \frac{\mathrm{D}J(\boldsymbol{x}^{(i)})}{\mathrm{D}\boldsymbol{x}}$ is no longer a two-dimensional vector in $\mathbb{R}^2$ but is instead an angular velocity that should be applied to $\boldsymbol{x}^{(i)}$ over a unit time, by tracking the unit circle through the yellow arc. In particular, note that $\Delta^{(i)}$ does not even have the same dimensionality as the euclidean vector, since in this case of 2D rotation it only has one dimension representing the coordinate on the tangent space $T_{\boldsymbol{x}^{(i)}}$ starting from $\boldsymbol{x}^{(i)}$. This reveals to be much more natural, since the element $\boldsymbol{x}^{(i)}$, in this example, has only one possible degree of freedom and the increment is directly expressed in this sense. Clearly, the optimization in $\mathbb{SO}(2)$ and $\mathbb{SE}(3)$ are very, different, since in the former even if the step length is not precisely the correct one, in the unconstrained update rule, the direction, corresponding to the sign of the gradient, is very clear in both cases, while in latter the possible directions are six, and the effects on translation and rotation are intermingled in a very non-linear way, making in this case the re-projection method much more sensitive. Indeed, the first method can be considered an approximation of the second if the step size is sufficiently small. Here lies the main advantage of the Lie group optimization: it allows larger step sizes and then faster convergence. We now provide the optimization steps for both methods, so to compare the results later.

Optimization on the vector space First, for the classic gradient descent step with renormalization, where the target transformation $\boldsymbol{T}(\boldsymbol{p}, \boldsymbol{q})$ is parametrized by $\boldsymbol{p} \in \mathbb{R}^3$ and $\boldsymbol{q} \in \mathbb{R}^4$, we can separate, for notation purposes, the gradient w.r.t. the position and that to the quaternion, since renormalization is only needed for the latter. For compactness, we consider the rotation of the vectors $\bar{\boldsymbol{p}}$ through the multiplication by the rotation matrix $\boldsymbol{R}(\boldsymbol{q}) \in \mathbb{R}^{3\times3}$ associated to $\boldsymbol{q}$, keeping in mind that mathematically the operation is equivalent to the quaternion action, and correspondingly $\boldsymbol{T} \cdot \bar{\boldsymbol{p}} = \boldsymbol{R}(\boldsymbol{q})\bar{\boldsymbol{p}} + \boldsymbol{p}$. By the chain rule, the euclidean gradient is:

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{p}} J(\boldsymbol{T}(\boldsymbol{p}, \boldsymbol{q})) &= \frac{1}{M} \sum_{\bar{\boldsymbol{p}} \in \mathcal{P}_1} \frac{\partial}{\partial \boldsymbol{p}} \mathcal{L}_{\mathcal{P}_2}(\boldsymbol{R}(\boldsymbol{q})\bar{\boldsymbol{p}} + \boldsymbol{p}) \\ \frac{\partial}{\partial \boldsymbol{q}} J(\boldsymbol{T}(\boldsymbol{p}, \boldsymbol{q})) &= \frac{1}{M} \sum_{\bar{\boldsymbol{p}} \in \mathcal{P}_1} \frac{\partial}{\partial \boldsymbol{p}} \mathcal{L}_{\mathcal{P}_2}(\boldsymbol{R}(\boldsymbol{q})\bar{\boldsymbol{p}} + \boldsymbol{p}) \frac{\partial}{\partial \boldsymbol{q}} \boldsymbol{R}(\boldsymbol{q})\bar{\boldsymbol{p}} \end{aligned} \tag{6.26}$$

We then consider the two estimates $\hat{\boldsymbol{p}}$ and $\hat{\boldsymbol{q}}$, computed with the update rule:

$$\begin{aligned} \hat{\boldsymbol{p}}^{(i+1)} &= \hat{\boldsymbol{p}}^{(i)} + \eta_{\boldsymbol{p}} \frac{\partial}{\partial \boldsymbol{p}} J(\boldsymbol{T}(\hat{\boldsymbol{p}}^{(i)}, \hat{\boldsymbol{q}}^{(i)})) \\ \hat{\boldsymbol{q}}_s^{(i+1)} &= \hat{\boldsymbol{q}}^{(i)} + \eta_{\boldsymbol{q}} \frac{\partial}{\partial \boldsymbol{q}} J(\boldsymbol{T}(\hat{\boldsymbol{p}}^{(i)}, \hat{\boldsymbol{q}}^{(i)})) \\ \hat{\boldsymbol{q}}^{(i+1)} &= \frac{\hat{\boldsymbol{q}}_s^{(i+1)}}{\|\hat{\boldsymbol{q}}_s^{(i+1)}\|} \end{aligned} \tag{6.27}$$

where $\eta_{\boldsymbol{p}} > 0$ and $\eta_{\boldsymbol{q}} > 0$ are the step gains of the two estimates. Note that even if the two equations are written separately, they represent a single update in the optimization algorithm, meaning that both estimates are computed in parallel rather than one after the other.

Optimization on the Lie group We formally define the update step by considering the values to be elements of a Lie group. Using the chain rule to evaluate the closed form of the gradient requires more care as each composition must be interpreted as a function between Lie groups. It is worth noticing that any vector space $\mathbb{R}^n$, equipped with the standard sum as a composition operation, can be seen as a Lie group, namely $(\mathbb{R}^n, +)$, with the Lie algebra associated to it being trivially $\mathbb{R}^n$ itself. The matching index $J_{1,2}$ is a composition that can be decomposed as:

$$\begin{array}{ccccccc} J_{1,2} & : & \mathbb{SE}(3) & \to & \mathbb{R}^3 & \to & \mathbb{R} \\ & & \boldsymbol{T} & \mapsto & \boldsymbol{T} \cdot \bar{\boldsymbol{p}} & \mapsto & \frac{1}{M} \sum_{\bar{\boldsymbol{p}} \in \mathcal{P}_1} \mathcal{L}_{\mathcal{P}_2}(\boldsymbol{T} \cdot \bar{\boldsymbol{p}}) \end{array} \tag{6.28}$$

The Jacobian of a function of a Lie group, maps elements of the tangent space of the domain group into elements of the tangent space of the co-domain. In this case:

$$\frac{\mathrm{D}}{\mathrm{D}\boldsymbol{T}} J_{1,2} \; : \; \mathfrak{se}(3) \; \to \; \mathbb{R}^3 \; \to \; \mathbb{R} \tag{6.29}$$

In practice, in place of $\mathfrak{se}(3)$ we consider its isomorphic $\mathbb{R}^6$, through the *vee* map as introduced in the preliminary section. The gradient of the matching index, after applying the chain rule, is then:

$$\begin{aligned}\frac{\mathrm{D}J_{1,2}(\boldsymbol{T})}{\mathrm{D}\boldsymbol{T}} &= \frac{1}{M}\sum_{\bar{\boldsymbol{p}}\in\mathcal{P}_1}\frac{\partial}{\partial\boldsymbol{p}}\mathcal{L}_{\mathcal{P}_2}(\boldsymbol{T}\cdot\bar{\boldsymbol{p}})\frac{\mathrm{D}\boldsymbol{T}\cdot\bar{\boldsymbol{p}}}{\mathrm{D}\boldsymbol{T}} \\ &= \frac{1}{M}\sum_{\bar{\boldsymbol{p}}\in\mathcal{P}_1}\underbrace{\frac{\partial}{\partial\boldsymbol{p}}\mathcal{L}_{\mathcal{P}_2}(\boldsymbol{T}\cdot\bar{\boldsymbol{p}})^\top}_{1\times 3}\underbrace{\begin{bmatrix}\boldsymbol{R} & -\boldsymbol{R}\,[\boldsymbol{v}]_\times\end{bmatrix}}_{3\times 6}\in\mathbb{R}^6\cong\mathfrak{se}(3).\end{aligned} \tag{6.30}$$

The value of $\frac{\mathrm{D}J_{1,2}(\boldsymbol{T})}{\mathrm{D}\boldsymbol{T}}$, then, represents the (stacked) linear and angular velocities corresponding to the direction to be applied to $\boldsymbol{T}$ of maximum increase for $J_{1,2}(\boldsymbol{T})$. The euclidean update rule can be then converted in terms of Lie groups element composition as follows:

$$\begin{aligned}\hat{\boldsymbol{T}}^{(i+1)} &= \hat{\boldsymbol{T}}^{(i)}\oplus\left(\boldsymbol{W}\frac{\mathrm{D}J_{1,2}(\boldsymbol{T})}{\mathrm{D}\boldsymbol{T}}\right) \\ &= \hat{\boldsymbol{T}}^{(i)}\circ\mathrm{Exp}\left(\boldsymbol{W}\frac{\mathrm{D}J_{1,2}(\boldsymbol{T})}{\mathrm{D}\boldsymbol{T}}\right)\end{aligned} \tag{6.31}$$

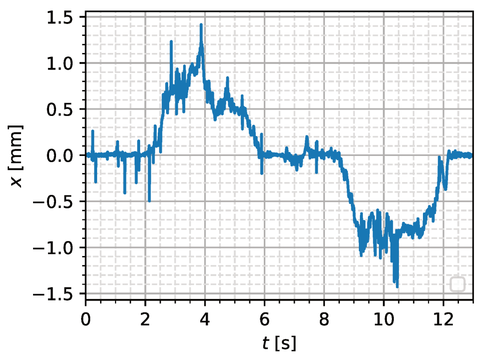
where $\boldsymbol{W} = \mathrm{diag}[\eta_{\boldsymbol{\rho}}\mathbb{1}_3,\ \eta_{\boldsymbol{\omega}}\mathbb{1}_3]\succ 0$ is a matrix with the scaling for the linear, $\eta_{\boldsymbol{\rho}}$, and angular, $\eta_{\boldsymbol{\omega}}$, velocities.

It is noteworthy that, in comparison with Euclidean optimization, this method is optimizing across a vector consisting of six variables, i.e., the three components for the linear velocity $\boldsymbol{\rho}$ and three for the angular velocity $\boldsymbol{\omega}$, as opposed to seven variables, three for the translation and four for the quaternion; the latter's derivatives, moreover, cannot be interpreted as angular velocity and thus lack physical meaning. The decreased problem size, coupled with the augmented naturalness of the solution, demonstrates greater efficacy in terms of algorithm convergence. This allows for the selection of higher gains and then resulting in fewer iteration steps, as will be demonstrated subsequently.

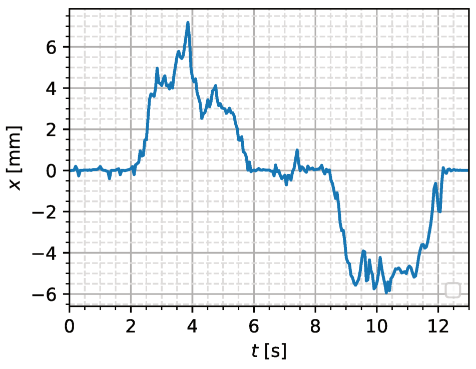
6.4.3 Real time optimization

The proposed optimization problem needs to be solved in real time as the robot moves in the environment, independent of the employed implementation strategy. Depending on the robot configuration, the target working rate could be from $30\,\mathrm{Hz}$ to $100\,\mathrm{Hz}$. This motivates the pursuit of the highest performances in the optimization algorithm, in terems of convergence speed. However, meeting these strict requirements is also undermined by the large amount of data that the depth camera records per sample, as well as the inherent amount of noise that such data is affected. We now present several strategies to mitigate such issues.

Samples windowing Achieving a rate of $30\,\mathrm{Hz}$ implies that the order of magnitude of the distances tracked are of millimeters for typical working speed of real robots. The average noise that affects the depth cameras can be significant compared to these distances. For reference, consider an experiment in which a robot longitudinally moves forth and back. Figure 6.3a shows the relative movement between each sample recorded by a motion capture system, and it is evident that the amount of noise and outliers is high compared to the average displacement, which is between 0 and $1\,\mathrm{mm}$. To address this issue, a sliding window can be used. Specifically, if $T_c \in \mathbb{R}$ is the sample period and $t_k = kT_c$ are the sample times, and $\mathcal{F}_k$ is the frame of the robot at time t_k, instead of considering the transformation between $\mathcal{F}_k$ and $\mathcal{F}_{k-1}$, namely $\boldsymbol{T}_k^{k-1} = \mathcal{F}_{k-1}^{-1}\mathcal{F}_k$, we consider, for each k, the transformation $\boldsymbol{T}_k^{k-N_w}$, comparing the current frame $\mathcal{F}_k$ with a number $N_w \in \mathbb{N}$ of samples back, $\mathcal{F}_{k-N_w}$. The effect on noise reduction of this technique is visible in figure 6.3b showing, for the same experiment, the x component of the displacement between frames of distance $N_w = 6$. This second signal is also scaled by a factor 6 with respect to the first one. This is a consequence of



(a) Relative ground truth with full sampling time



(b) Relative ground truth over $N_w = 6$ samples.

Figure 6.3.: Ground truth relative positions between frames of an illustrative experiment.

the fact that the quantities at play are not velocities but transformations. In the euclidean space, these two concepts are directly linked: it is sufficient to divide the difference in the position by the elapsed time to obtain the estimate average velocity. However, the quantities at play must be considered in the Lie group $\mathbb{SE}(3)$, where there is no notion of multiplication by a scalar, and the velocity is indeed a concept belonging to its tangent space, identified by the Lie algebra $\mathfrak{se}(3)$, which in turn is a vector space, where it is actually possible to divide. Specifically, the concept of velocity needs to pass through the logarithmic map, and it holds:

$$\boldsymbol{\nu}_k^{k-N_w} := \frac{\log\left(\boldsymbol{T}_k^{k-N_w}\right)}{N_w T_c} \in \mathfrak{se}(3) \tag{6.32}$$

where $\boldsymbol{\nu}_k^{k-N_w}$ is the generalized velocity embedding the linear and angular velocities of the body between t_{k-N_w} and t_k.

Data pruning and stochastic gradient descent Another issue that is limiting the capabilities of the algorithm is the amount of data that the 3D sensor yields. Typically, the resolution of a depth image is 640x480, which entails point cloud size of 300k. Not all of them are actually useful in the estimate. In particular, we identify two macro categories of non-informative points:

1. Points of the background are usually located close to the sensor's range limit and are, thus, more affected by noise, rendering them little informative.

2. Points that are too close to each other carry the same information, thus there should be a minimum distance between two points to be relevant.

The first category is easy to exclude, as it is sufficient to remove points farther than a threshold distance from the camera. By keeping in mind that the points are referred to the frame of the camera's optical center. We then consider:

$$\mathcal{P}' = \{\bar{\boldsymbol{p}} \in \mathcal{P} \text{ s.t. } \|\bar{\boldsymbol{p}}\| < d_{bg,max}\} \tag{6.33}$$

where $d_{bg,max} \in \mathbb{R}$ is the chosen threshold.

To address the second category we employ a heuristics that exploits the fact that in the vector of the point cloud the points are ordered row-wise, so it is relatively reasonable to assume that to close indexes correspond close points. We can then decimate the point cloud $\mathcal{P}'$ by considering one point every N_d:

$$\mathcal{P}'' = \{\bar{\boldsymbol{p}}_h \in \mathcal{P}', \quad h = 1, N_d, 2N_d, \ldots, M\} \tag{6.34}$$

Although the size of the data is heavily reduced, the size of the point clouds is still relevant if we note that for the matching index every point of the point cloud $\mathcal{P}_2$ is compared to all the points of the point cloud $\mathcal{P}_1$. Drawing inspiration from the *stochastic gradient descent* method, we can modify the update rule of the optimization algorithm as follows:

1. For each optimization step i, draw $N_{SGD,1}, N_{SGD,2} \in \mathbb{N}$ random samples from $\mathcal{P}_1''$ and $\mathcal{P}_2''$, that are the original point clouds to which are applied (6.33) and (6.34):

$$\tilde{\mathcal{P}}_i^{(i)} \subset \mathcal{P}_i'', \quad |\tilde{\mathcal{P}}_i^{(i)}| = N_{SGD,i}, \quad i = 1, 2 \tag{6.35}$$

2. Redefine for each update step the matching index as

$$J_{1,2}^{(i)}(\boldsymbol{T}) = \frac{1}{M} \sum_{\bar{\boldsymbol{p}} \in \tilde{\mathcal{P}}_1^{(i)}} \mathcal{L}_{\tilde{\mathcal{P}}_2^{(i)}}(\boldsymbol{T} \cdot \bar{\boldsymbol{p}}) \tag{6.36}$$

3. Apply the update rule using $J_{1,2}^{(i)}(\boldsymbol{T})$.

The number of the batch of point considered for each step should be enough to guarantee computational feasibility, but also it should be such that the majority of the (pruned) point cloud is statistically sampled, so all the information is considered throughout the optimization.

6.5 Simulations results

We now present the preliminary result of a simulated but real time scenario. The simulation is configured as follows:

- A synthetic point cloud of size $M = 10^3$, whose points are draw from a uniform distribution, is generated, namely

$$\mathcal{P}_0 = \{\bar{\boldsymbol{p}}_h \sim \mathcal{U}[0,1], \quad h = 1, \dots, M\} \tag{6.37}$$

- We consider a ground truth trajectory that the agent is tracking, namely $\boldsymbol{T}_{GT} : \mathbb{R} \to \mathbb{SE}(3)$, such that:

$$\boldsymbol{T}_{GT}(t) = (\boldsymbol{p}_{GT}(t), \boldsymbol{q}_{GT}(t)), \tag{6.38}$$

  where we highlighted its translation, $\boldsymbol{p}_{GT}(t) \in \mathbb{R}^3$, and rotation $\boldsymbol{q}_{GT} \in \mathbb{SO}(2)$, parametrized by a quaternion. Between two samples $k-1$ and k, the ground truth to which we will compare our estimate is then:

$$\boldsymbol{T}_{k,GT} = \boldsymbol{T}_{GT}(t_{k-1})^{-1}\boldsymbol{T}_{GT}(t_k) \tag{6.39}$$

- Each time step, the new point cloud simulating the current measurement is generated by transforming the points of the last point cloud, namely:

$$\mathcal{P}_k = \{\boldsymbol{T}_{k,GT} \cdot \bar{\boldsymbol{p}}_{k-1}, \quad \bar{\boldsymbol{p}}_{k-1} \in \mathcal{P}_{k-1}\} \tag{6.40}$$

- The estimate is then the transformation that yields maximum matching between the current $\mathcal{P}_k$, and the predicted old $\boldsymbol{T} \cdot \mathcal{P}_{k-1}$:

$$\hat{\boldsymbol{T}}_k = \underset{\boldsymbol{T} \in \mathbb{SE}(3)}{\operatorname{argmax}} \; J_{k-1,k}(\boldsymbol{T}) \tag{6.41}$$

Constant velocity trajectory We first consider the case where the agent moves with a trajectory consisting in a constant linear velocity combined with a constant rotation around axis z, specifically:

$$\boldsymbol{p}_{GT}(t) = [v_x t, 0, 0], \quad \boldsymbol{q}_{GT}(t) = \boldsymbol{q}_z(\omega_z t), \tag{6.42}$$

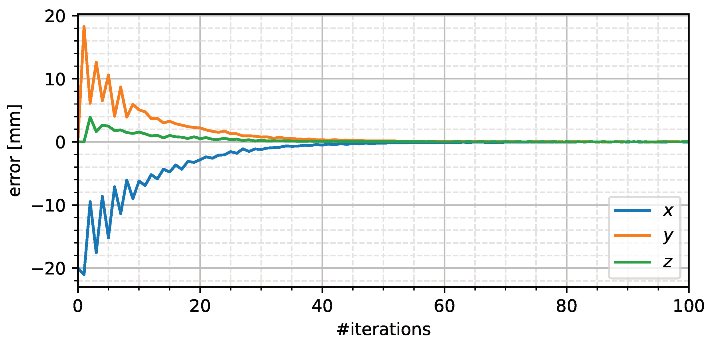
where $v_x = 3\,\mathrm{m\,s^{-1}}$, $\boldsymbol{q}_z(\theta)$ is the quaternion representing rotation about axis z by an angle θ and $\omega_z = \frac{\pi}{6}\mathrm{rad\,s^{-1}}$. The simulated velocity is chosen to be quite high, compared to the typical working conditions of aerial robots, in order to stress the convergence capabilities of the algorithm. With a sample rate of $30\,\mathrm{Hz}$, this trajectory corresponds to a transformation between samples of $\boldsymbol{p}_{k,GT} = [x, 0, 0]$, $x = 0.01\,\mathrm{m}$, and $\boldsymbol{q}_{k,GT} = \boldsymbol{q}_z(\theta)$, $\theta = \frac{\pi}{180}\mathrm{rad}$. We then initialize the algorithm by assigning the $\mathbb{SE}(3)$ identity element, $\mathbf{1}_{\mathbb{SE}(3)}$, to $\boldsymbol{T}^{(0)}$.

To understand the behavior of the algorithm, we will show the error convergence with respect to the number of iteration. We consider the following metrics to quantify the error between the estimate and the ground truth:

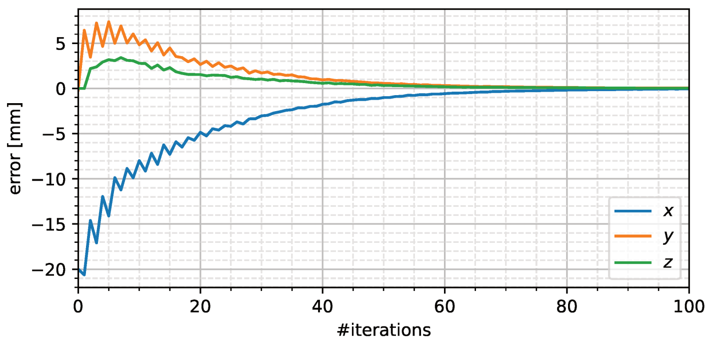
$$\boldsymbol{e}_{\mathbb{SE}(3)}(i) = \boldsymbol{T}^{(i)} \ominus \boldsymbol{T}_{k,GT} = \mathrm{Log}(\boldsymbol{T}_{k,GT}^{-1} \circ \boldsymbol{T}^{(i)}) \in \mathbb{R}^6 \cong \mathfrak{se}(3). \tag{6.43}$$

The error between $\boldsymbol{T}^{(i)}$ and $\boldsymbol{T}_{k,GT}$ can be interpreted as the linear and angular increments in the tangent space to get from the first element to the second, and they consist of six components, divided into three linear and three angular components, so that $\boldsymbol{e}_{\mathbb{SE}(3)} = [\boldsymbol{e}_{\boldsymbol{p}}, \boldsymbol{e}_{\boldsymbol{q}}]$, $\boldsymbol{e}_{\boldsymbol{p}}$, $\boldsymbol{e}_{\boldsymbol{q}} \in \mathbb{R}^3$. It is important to note that these angular components should not be considered as Euler angles, as Euler angles represent rotations around the x, y, and z axes with a prescribed order, while the angular error in $\mathfrak{se}(3)$ represents angular velocities that has to be applied simultaneously in the unit of time.

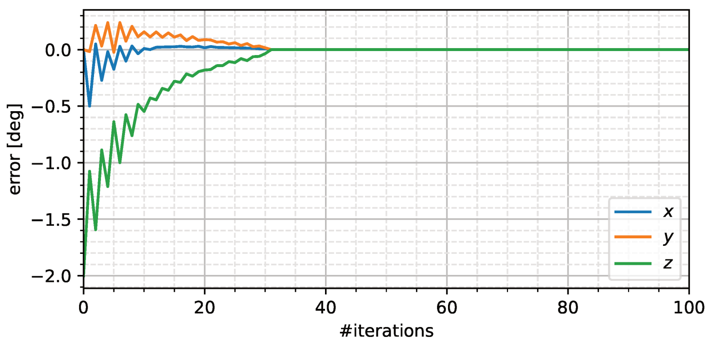
Figure 6.4 shows the convergence of the error $\boldsymbol{e}_{\mathbb{SE}(3)}$, comparing the case where the optimization is performed according the Lie groups rules (fig 6.4a and 6.4c) with the optimization with renormalization step (fig 6.4b and 6.4d). Since the motion is constant, we report the results of only the first time steps, being the subsequent ones analogous. It is evident from the plot that the former method requires 40 iterations to reach convergence of both linear and angular components, whereas the latter necessitates almost 80 iterations, thus demonstrating improved performance of a factor of two.



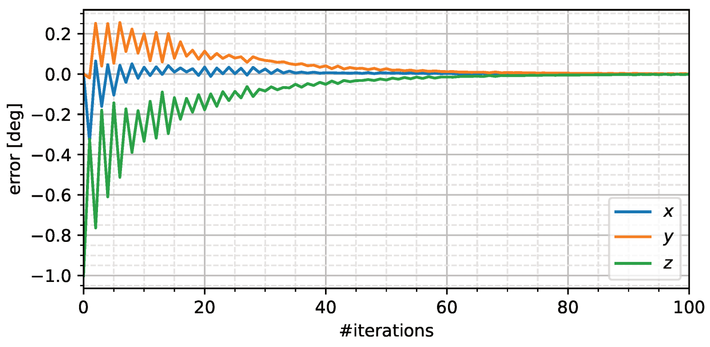
(a) Optimization in $\mathbb{SE}(3)$: linear error



(b) Optimization in $\mathbb{R}^7$: linear error



(c) Optimization in $\mathbb{SE}(3)$: angular error



(d) Optimization in $\mathbb{R}^7$: angular error

Figure 6.4.: Algorithm convergence comparison with update step in the Lie group (6.4a and 6.4c), compared to the result with the renormalization method (6.4b and 6.4d)

Sinusoidal motion We now consider a more dynamic scenario where the ground truth roto-translation follows a sinusoidal pattern over time. Specifically:

$$\boldsymbol{p}_{GT}(t) = [\bar{v}_x \sin(2\pi \bar{f} t, 0, 0] \tag{6.44}$$

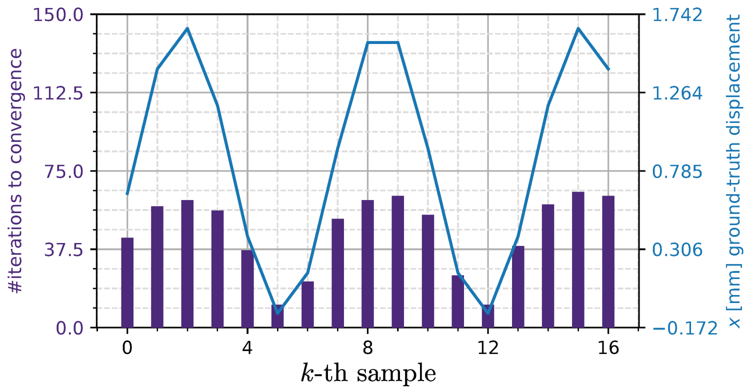
where $\bar{v} = 0.016\,\mathrm{m\,s^{-1}}$, $\bar{f} = 0.15\,\mathrm{Hz}$ are the parameter of choice. The algorithm is applied with both methods, optimization in $\mathbb{SE}(3)$ and $\mathbb{R}^7$ and two different initialization algorithms are tested. The results are reported in figure 6.5. First, as in the example for the previous simulation, the algorithm is initialized with the identity (6.5a, 6.5a). In the second part of the experiment (6.5c, 6.5d) at each sample time the algorithm is instead initialized with the previous estimate, namely:

$$\boldsymbol{T}_k^{(0)} = \hat{\boldsymbol{T}}_{k-1} \tag{6.45}$$

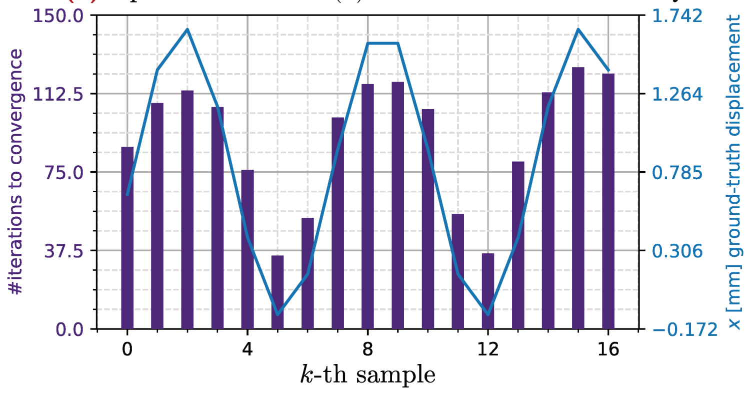
In the plots, the purple vertical bars, whose scale is indicated on the left, are the total number of iterations needed in each algorithm configuration to reach convergence, for each sample k, namely

$$N_{iter,k} = \min_i\{\|\boldsymbol{T}^{(i)} \ominus \boldsymbol{T}_{k,GT}\| < 1.5 \times 10^{-4}\}, \tag{6.46}$$

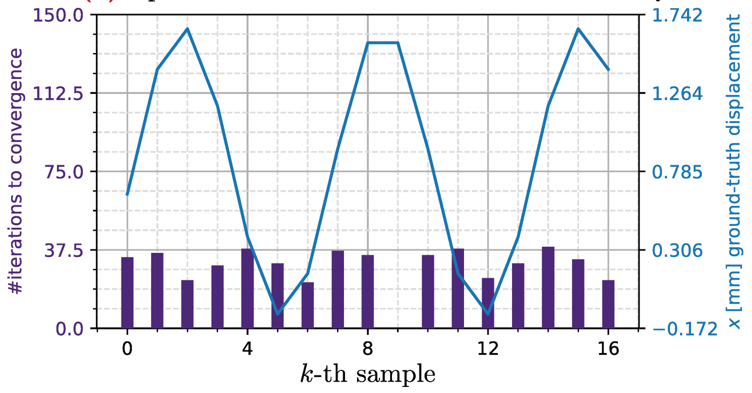
while the blue plot is the x component of the ground truth at the k-th sample and its scale is reported on the right side of the graph. The reported results confirm that the optimization on the Lie group is able to yield better results: the number of iterations is almost halved when using either type of initialization. On the other hand, figures 6.5a and 6.5a reveal a clear correlation between the number of iterations and the magnitude of the transformation. This correlation is eliminated when the second type of initialization is used, as illustrated in figures 6.5c and 6.5d. Notably, at sample $k = 9$, where the ground truth velocity is steady (as shown by the blue plot between samples 8 and 9) no iterations are needed ($N_{9,iter} = 0$), because the initialization is already below the desired threshold. This indicates that if the velocity of the agent is approximately constant, this method requires very few iterations, and the results suggest that it generally yields improved performance.



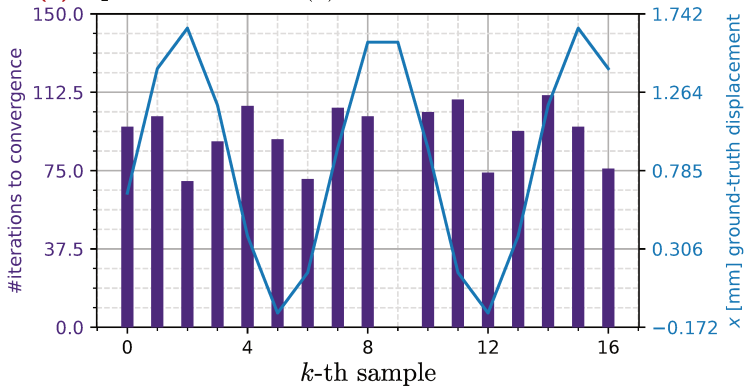
(a) Optimization in $\mathbb{SE}(3)$: initialization with identity



(b) Optimization in $\mathbb{R}^7$: initialization with identity



(c) Optimization in $\mathbb{SE}(3)$: initialization with last estimate



(d) Optimization in $\mathbb{R}^7$: initialization with last estimate

Figure 6.5.: Number of iteration steps to achieve convergence for the simulation compared to the value of the x component of the ground truth transformation, with the two different optimization manifolds and two different initialization

6.6 Conclusions and future directions

In this chapter we presented a method for estimating motion from 3D point cloud measurements, based on a feature-less matching index and with optimization performed on the Lie group of rototranslations. This method is targeted towards aerial physical interaction, wherein determining the absolute pose of the robot is not the primary goal. Rather, the relative motion is fundamental, and it is used to feed an admittance controller that can manage the interaction between the root and the environment, in order to perform robust manipulation in presence of uncertainties. The results show that the algorithm is able to converge faster when the optimization is performed within the Lie group, rather than the standard gradient descent in the Euclidean vector space. Future directions will involve its implementation in an experimental environment with a tilt-rotor hexacopter. This includes the real time processing of point clouds captured by a depth camera and the definition of a wrench-estimation filter. Preliminary and ongoing experiments using the Intel RealSense d435i depth camera show promising results, although the full-stack implementation is challenged by numerous technological difficulties.

A Frameworks for real-time performances

A.1 Introduction

The importance of real time constraints in the development of full-stack architectures for autonomous robots forces a particular attention on how the algorithms are realized in practice. The efficiency of implementation was a major factor in the development of the methodologies discussed in this thesis: an algorithm can be effectively considered efficient only if it allows an efficient implementation, resulting in a real time execution. This motivates the strong necessity that the simulations must be performed online, and this can be achieved through the ROS framework together with the real time robot simulator Gazebo (Koenig and Howard, 2004). Real time and physics engine-based simulators offer the possibility to apply the same algorithm to both the simulation and the real experiments, as long as a common ROS interface is defined, as presented in chapter 2, with minimal discrepancies when the simulation is sufficiently precise. To keep up with both the simulation and the experimental timing requirements, when the algorithmic complexity is relevant it is fundamental that the implementation is as optimized as possible. To this goal, a C++ implementation is highly preferred for several reasons. First, being a compiled language, the compiler itself optimizes the generated machine code incredibly efficiently (Godbolt, 2020), as a result of decades of development. Second, its thin runtime layer allows for the most efficient memory and resources management, of crucial importance especially when processing data-rich sensors measurements. Finally, its heavily structured architecture allow for robust large framework development, that begin to be essential when the full-stack system is composed of different modules (e.g. the NAPVIG-X algorithm in chapter 5). However, while C++ is suitable for high performance applications, it does not have the same easy access to a variety of debug and display tools that are instead one of the advantages of Python, which, other than being useful in a first prototyping phase, can be integrated online to C++ nodes thanks again to the ROS interface, which is transparent to the used language, and implement online visualization tools. In this appendix we will present a brief overview on some implementation details that are of key importance in allowing the proposed algorithms to fulfill real time requirements.

A.2 Tensors operations

Algorithms involving the processing of data-rich sensors share the necessity to perform operations on large-scale arrays, often characterized by multidimensionality. The simplest example of a multidimensional array is a colored image: the pixels are arranged in a matrix, and each pixel itself is a vector of three elements representing the red, green and blue values, thus representing a data structure with higher dimensionality than a 2D matrix. The mathematical objects that are able to model this kind of multi-array are the tensors, that can be seen as a generalization and unification of the concepts of scalar, vector, matrices and higher order of representation: they are characterized by a variable number of dimensions, so a tensor of zero dimensions corresponds to a scalar, one dimension to a vector, and so on, generalizable to n dimension.

Tensors are vastly employed in machine learning and its recent rapid growth enforced a huge effort in the development of libraries that offer highly optimized implementations of mathematical operations on tensors, such as TensorFlow (Abadi et al., 2016) or PyTorch (Paszke et al., 2019). The latter, in particular, despite being born for deep learning applications, provide an excellent math library whose optimization is advantageous even for different type of data-processing algorithms. This is the library that has been vastly used to realize the algorithm in this thesis, both in the versions for Python and C++ (`libtorch`).

The real advantage of tensors is that they are not limited to handle multidimensionality, but they allow for *vectorization* of the mathematical operations between data tensors. Its easiest example is the implementation of the sum of the elements of an array. The traditional way is by iterating over each element and increment an accumulator. Assuming `v` is an iterable vector (e.g. `std::vector<float>`), in C++ code:

```
float sum = 0;
for (float v_i : v)
    sum += v_i;
```

The vectorized version of the sum operation would be to directly call a method `sum` on the vector, now as a 1-dim tensor, `v` (in this case with type `torch::Tensor`):

```
torch::Tensor sum = v.sum (0);
```

Note that the result of a tensor operation is always another tensor, in this case of size 0, representing a scalar. The `0` as argument of the `sum` indicates that the sum should be taken with respect of the first (and only) dimension: indeed, `v` could be a $N \times M$ matrix and the sum could be taken row or column wise, indicating `0` or `1` as argument, and in this case the result would be a tensor with 1 dimension of size M, or N, respectively. Also, the variable dimension nature of tensor allows to generalize this concept to any number of dimensionality.

Single batch operations on tensors, instead of performing "for loops", results to have better performances in terms of overall execution time, even in C++ code, that, contrarily to Python, has almost no overhead in the iterations. The overall reason is the ability to operate on all dimension simultaneously, allowing for a variety of optimizations. First, it enables the multi-thread parallelization of the computation, which means that several operations can be calculated at the same time. Also, it allows for optimized memory management: for example, the data can be stored contiguously, resulting in an increased performance due to less cache misses. Furthermore, in certain operations it is possible to avoid repeated calculations by exploiting some properties of the structure of the tensor itself, for example the *broadcasting* operator, of which we provide a simple example. Consider the case where we want to find all the distances between two sets of vectors, `v1` and `v2`, 2D tensors of size $M \times D$ and $N \times D$. A tensor containing all the possible differences will be a 3D tensor of size $M \times N \times 2$: after the input tensors need to be reshaped to sizes $M \times 1 \times D$ and $1 \times N \times D$, all the entries of `p1` will be "broadcasted" to the entries of `p2`, guided by the dimensions with size 1, known as *singleton* dimensions. To then get the distances it is sufficient to employ the library function `norm`, specifying the dimension in which the norm should be taken, in this case the third, containing the two components of each point difference. Summarizing, the tensor of distances of size $N \times M$ can be obtained as follows:

```
    distances = (p1.reshape(M, 1, D) - p2.reshape(1, N, D)).norm(2);
```

Note that this line has to be intended as pseudocode, as the real syntax would have several additional elements which we neglected for presentation purposes. The traditional implementation would require three nested `for` loops, as follows:

```
    for (int i = 0; i < M; i++)
        for (int j = 0; j < N; j++) {
            float dist = 0;
            for (int d = 0; d < D; d++)
                dist += pow(p1[i, d] - p2[j, d], 2);
            distances[i, j] = sqrt (dist);
        }
```

In a test with random arrays with $M = 100$, $N = 200$, $D = 100$, performed on an Intel i7 architecture, and repeated 10000 times, on average, the C++ implementation of the loop takes $6.29\,\mathrm{ms}$ to execute while the tensor operation takes $0.23\,\mathrm{ms}$, almost 30 times faster. The same loop execution implemented with Python, for which loops have a large overhead, takes $79.85\,\mathrm{ms}$, with one order of magnitude of difference.

A.3 `lietorch`: a library for tensor based Lie group operations

In robotics, and, in particular, in estimation, it is common to apply the same transformation to a batch of points: for example, in chapter 6 this happens to the elements of a point cloud to be matched to another point cloud. These are large arrays, and we want to exploit the advantages of batch tensor optimization to the operations of the family of roto-translation. This lead to the development of a C++ library that implements the common Lie group operations such as composition, rotation, translation, etc. The library is based on PyTorch and is publicly available on GitHub[1].

[1] https://github.com/nicola-lissandrini/lietorch

The core characteristics of the library is modularity: given that all the Lie group share the same operations (composition, group action, inverse, exponential, etc.), they all derive from a common abstract class by exploiting CRTP (Curiously recurring template pattern) (Coplien, 1996), that allows static polymorphism. This is required so that it is possible to use the operator overloading to write more maintainable code: for example, given two objects representing a specific Lie group, their composition is simply `a * b`, with the standard multiplication operator. On one hand this results in self-explaining clean code, on the other hand it reveals to be more versatile as it is no longer tied to a particular implementation, but instead has a semantic meaning that is easier to interpret.

The abstract class containing all the shared methods has the following structure[2]:

```
template<class Derived>
class LieGroup {
public:
    Derived inverse ();
    Tangent log ();
    Derived compose (Derived other);
    Tensor dist (Derived other, DataType weights);
    Tensor act (Tensor v);
    Tangent differentiate (Tensor outerGradient, Tensor v);

    [...]
private:
    Tensor coeffs;
};
```

where:

- `Derived` is the actual derived class representing the specific Lie group, that are declared, following CRTP, for example, as:

  ```
  class Quaternion : public LieGroup<Quaternion>
  ```

  Thanks to the common abstract base class, derived classes only need to implement the above functions, while the operator overloading and all the common technicalities are handled by the base class `LieGroup` The

[2]Once more, this has to be interpreted as pseudo-C++, as inessential syntax details have been omitted.

available implemented Lie groups are `Rn<N>`, translations with `N` known at compile time; `Quaternion`; `UnitComplex`, 2D rotations; `Pose2`, union of `Rn<2>` and `UnitComplex`; `Pose3`, union of `Rn<3>` and `Quaternion`.

- `Tangent` is the dual class associated to each Lie group implementation representing its Lie algebra. They also derive a base class, that has the following structure:

  ```
  template<class Derived>
  class Tangent {
  public:
      LieGroup exp ();
      Derived scale (Tensor values);
      Tensor norm ();

      [...]
  private:
      Tensor coeffs;
  };
  ```

  where the methods represent the standard exponential, scaling and norm operations on the Lie algebra. For each implemented Lie group, the corresponding available implemented derivations of `Tangent`, in the same order, are: `VelocityRn<N>`; `AngularVelocity`, `ComplexVelocity`; `Twist2`; `Twist3`.

- `inverse`, `log` and `compose` are the methods for the standard operations for Lie group elements. `act` refers to the group action, and is one of the key features of the library as its implementation allows applying the group action (e.g. rotation) to multiple vectors simultaneously, stored into the tensor `v` as rows.

- `differentiate` is another key feature of the library, and allows the batch tensor product of the jacobian of the group action by a batch of vectors stored as rows of a 2D tensor. This is useful when applying the chain rule to a function of the action group, as in (6.30). The argument `outerGradient` is the gradient of the chained function, and `v` is the tensor of points where the gradient should be computed, which may be needed for the computation of the jacobian in some cases (such as with rotations).

In this brief overview we skip the details on the definition of the operator overloading, whose technicalities are beyond the scope of this thesis and for which we remind to the repository. However, we provide an example of use of the library showing the implementation of equation (6.31), that is very intuitive thanks to the overloading of operators +, implementing the $\oplus$ operation, and $*$, implementing the $\circ$ operation:

```
predicted = estimate * oldPointcloud;
outerGradient = landscape.gradient (predicted)
nextEstimate = estimate + stepSizes *
               estimate.differentiate (outerGradient,
                                       predicted).sum ();
```

Here, `estimate` is the current estimate, namely $\hat{\boldsymbol{T}}^{(i)}$ in (6.31), `oldPointcloud` is the old point cloud $\mathcal{P}_{k-1}$ from (6.40), organized with the points as rows of a 2D tensor, `landscape.gradient (predicted)` computes the gradient of the Landscape function in the predicted point cloud, which is the outer gradient of the chain rule. The last line is the update rule of the estimation algorithm. The use of operator overloading then separates the technical realization details of the Lie group operations from those of the optimization algorithm, allowing robust development of the methodologies, which becomes fundamental in the implementation of complex and full-stack systems,

A.4 `ModFlow`: a modular, run-time flow control library for robotics

A full-stack robotic system must handle complex interaction between different and asynchronous input streams (i.e. the sensors data), as well as different logic units, realizing different aspects of the algorithm. As an example, implementing the algorithm presented in chapter 5, requires different modules, such as the one for sensor data acquisition, policy switching, trajectory computation and evaluation, etc. Handling this interactions may result in a convoluted code structure that might become difficult to manage as the size of the project increases. We now briefly present `ModFlow`, another library that offers a lightweight solution to this issue and which is publicly available on GitHub, as part of a general purpose robotics and ROS library[3]. The main

[3]https://github.com/nicola-lissandrini/NLib

concept is inspired by the structure of a ROS network: the computation is subdivided into *modules*, that are connected at run time through *channels*. Each module handles the computation of a specific part of the algorithm, and modules can *emit* and receive signals through the channels. The mechanics is similar to ROS, but it has several advantages:

- It is extremely lightweight: contrary to ROS, there is almost no overhead in the communication between modules, as they are realized in the same executable, directly through function calls.

- Channels can be of any type of data, which can be passed from a module to another by reference, without the need for copying and converting the data.

- New modules can be created and connected with few lines of code, with no need for external launch files, allowing for fast prototyping.

This library, however, should not be considered as an alternative to ROS but as a complementary and a means to realize ROS nodes. Indeed, the idea is that the incoming data from a ROS topic subscriber is converted and fed to a special channel, named *source channel*, that connects the outside world to the ModFlow network, and the modules can emit signals on special channels named *sink channels*, that output processed data through a ROS publisher.

The main element needed to build a ModFlow network, is to define a class that derives `nlib::NlModFlow`, with a member function `loadModules` that defines the modules to be loaded, as follows:

```
class MyModFlow : public nlib::NlModFlow {
public:
    [...]

    void loadModules () override {
        loadModule<Module1> ();
        [...]
        loadModule<ModuleN>  ();
    }
};
```

This will load the specified modules in order. Modules themselves are extension of the class `nlib::NlModule`, as follows:

```
class Module1 : public nlib::NlModule {
public:
    [...]
    void initParams (NlParams params) override;
    void setupNetwork () override;

    void slot1 (int value, string str);
    int slot2 ();
    [...]
};
```

Here `initParams` allows a standardized method to organize parameters, for whose details we remind to the documentation of the repository. The function `setupNetwork` defines all the input and output connections of the module. Outputs are defined by creating a channel, with the function defined in the parent class `NlModule`:

```
template<typename ...T>
Channel createChannel (string name);
```

The template arguments specify the types (0, 1 or more) of the channel, for example `createChannel<int, string> ("channel1")` creates a channel with types `int` and `string`, while `createChannel<>` creates an empty channel. The module can then emit a signal on a channel, through the function

```
template<typename ...T>
void emit (Channel channel, T ...values);
```

This will result in a function call to every modules connected to the same channel. Such connection from another module to a member function (referred to, in this context, as *slot*) can be defined with the method:

```
template<typename ...T, typename M, typename R>
void requestConnection (string name, R (M::*slot)(T...));
```

The specified channel name is searched in the existing channels, and a connection is created so that every time a signal is emitted on that channel, the function specified in the `slot` argument is called, passing the arguments specified in the `emit`. The types `...T`, `M` and `R` are the slot argument (the data passed from the caller module to the receiver), the name of the module itself (for technical requirement), and the return type of the slot. They are automatically deduced by the compiler, and they are specified by the signature of the slot method. Special modules are loaded by default to handle sources and sinks. Those modules can be accessed from outside the ModFlow network via the methods `sources()` and `sinks()` in the derived `NlModFlow` object (e.g. `MyModFlow` in the example abouve). Their member functions `declareSource` and `declareSink` create inbound *source channels* and outbound *sink channels*, respectively. Aside from technicalities, they work as regular channels and slots, with the only exception that the sink slot should be a member variable of a class external to the ModFlow network, and the signal on the source channels are emitted via the method `callSource`. The presented library contains also a direct integration of ModFlow to a ROS node, but we refer to the repository for more details.

A.5 Conclusions

In this appendix we presented tools and solutions for the implementation of real time strategies to address robotics task. Tensor operations proved to be a valuable resource for this purpose as various optimization can be leveraged to handle the operations of large multidimensional arrays. We also presented two open source libraries that provide fast and modular tools that reveal to be useful in the typical realization of a robotic system.

Bibliography

Abadi M., Agarwal A., Barham P., Brevdo E., Chen Z., Citro C., Corrado G. S., Davis A., Dean J., Devin M., and others . Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Ahad M. A., Paiva S., Tripathi G., and Feroz N. Enabling technologies and sustainable smart cities. *Sustainable cities and society*, 61:102301, 2020.

Al-Hourani A. and Ristic B. Mapperbot/iscan: open-source integrated robotic platform and algorithm for 2d mapping. *International Journal of Intelligent Robotics and Applications*, 4(1):44–56, 2020.

Alonso-Mora J., Baker S., and Rus D. Multi-robot formation control and object transport in dynamic environments via constrained optimization. *The International Journal of Robotics Research*, 36(9):1000–1021, 2017.

Alsamhi S. H., Ma O., Ansari M. S., and Almalki F. A. Survey on collaborative smart drones and internet of things for improving smartness of smart cities. *Ieee Access*, 7:128125–128152, 2019.

Alshawabkeh Y. Linear feature extraction from point cloud using color information. *Heritage Science*, 8(1):1–13, 2020.

Amano K. and Kato Y. Autonomous mobile robot navigation for complicated environments by switching multiple control policies. In *IECON 2022–48th Annual Conference of the IEEE Industrial Electronics Society*, pages 1–6. IEEE, 2022.

Arslan O. and Koditschek D. E. Sensor-based reactive navigation in unknown convex sphere worlds. *The International Journal of Robotics Research*, 38(2-3):196–223, 2019.

Ball D., Ross P., English A., Milani P., Richards D., Bate A., Upcroft B., Wyeth G., and Corke P. Farm workers of the future: Vision-based robotics for broad-acre agriculture. *IEEE Robotics & Automation Magazine*, 24(3): 97–107, 2017.

Bay H., Tuytelaars T., and Gool L. V. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

Behl A., Paschalidou D., Donné S., and Geiger A. Pointflownet: Learning representations for rigid motion estimation from point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7962–7971, 2019.

Bektaş K. and Bozma H. I. Apf-rl: Safe mapless navigation in unknown environments. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 7299–7305. IEEE, 2022.

Besl P. J. and McKay N. D. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. Spie, 1992.

Bhattacharya P. and Gavrilova M. L. Roadmap-based path planning-using the voronoi diagram for a clearance-based shortest path. *IEEE Robotics & Automation Magazine*, 15(2):58–66, 2008.

Bloesch M., Omari S., Hutter M., and Siegwart R. Robust visual inertial odometry using a direct ekf-based approach. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 298–304. IEEE, 2015.

Boucher P. Waypoints guidance of differential-drive mobile robots with kinematic and precision constraints. *Robotica*, 34(4):876–899, 2016.

Boufera F., Debbat F., Monmarché N., Slimane M., and Khelfi M. F. Fuzzy inference system optimization by evolutionary approach for mobile robot navigation. *International Journal of Intelligent Systems and Applications*, 10 (2):85, 2018.

Bounini F., Gingras D., Pollart H., and Gruyer D. Modified artificial potential field method for online path planning applications. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 180–185. IEEE, 2017.

Brescianini D. and D'Andrea R. An omni-directional multirotor vehicle. *Mechatronics*, 55:76–93, 2018.

Bruzzone L., Nodehi S. E., and Fanghella P. Tracked locomotion systems for ground mobile robots: A review. *Machines*, 10(8):648, 2022.

Bun M., Kingkan C., Kongprawechnon W., and Nakahara H. Cfp rrt for an automatic robot in the narrow space and the trapped goal environment. In *International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pages 737–741, 2021.

Cataldi E., Muscio G., Trujillo M. A., Rodríguez Y., Pierri F., Antonelli G., Caccavale F., Viguria A., Chiaverini S., and Ollero A. Impedance control of an aerial-manipulator: Preliminary results. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3848–3853. IEEE, 2016.

Chandler B. and Goodrich M. A. Online rrt* and online fmt*: Rapid replanning with dynamic cost. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6313–6318, 2017.

Chatila R. Deliberation and reactivity in autonomous mobile robots. *Robotics and Autonomous Systems*, 16(2-4):197–211, 1995.

Chen L., Shan Y., Tian W., Li B., and Cao D. A fast and efficient double-tree rrt*-like sampling-based planner applying on mobile robotic systems. *IEEE/ASME Trans. on Mechatronics*, 23(6):2568–2578, 2018.

Cheng Y. and Wang G. Y. Mobile robot navigation based on lidar. In *2018 Chinese Control And Decision Conference (CCDC)*, pages 1243–1246. IEEE, 2018.

Chi W., Ding Z., Wang J., Chen G., and Sun L. A generalized voronoi diagram-based efficient heuristic path planning method for rrts in mobile robots. *IEEE Transactions on Industrial Electronics*, 69(5):4926–4937, 2021.

Choi S., Kim E., Lee K., and Oh S. Real-time nonparametric reactive navigation of mobile robots in dynamic environments. *Robotics and Autonomous Systems*, 91:11–24, 2017.

Choset H. and Burdick J. Sensor based planning. i. the generalized voronoi graph. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1649–1655, 1995.

Choy C., Dong W., and Koltun V. Deep global registration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition,* pages 2514–2523, 2020.

Chung S., Paranjape A. A., Dames P., Shen S., and Kumar V. A survey on aerial swarm robotics. *IEEE Trans. on Robotics,* 34(4):837–855, 2018.

Coplien J. O. Curiously recurring template patterns. In *C++ gems,* pages 135–144. 1996.

Corah M. and Michael N. Active estimation of mass properties for safe cooperative lifting. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 4582–4587, 2017.

Cortés J. Coverage optimization and spatial load balancing by robotic sensor networks. *IEEE Trans. on Automatic Control*, 55(3):749–754, 2010.

D'Andrea R. Guest editorial: A revolution in the warehouse: A retrospective on kiva systems and the grand challenges ahead. *IEEE Trans. on Automation Science and Engineering*, 9(4):638–639, 2012.

Daniels J. I., Ha L. K., Ochotta T., and Silva C. T. Robust smooth feature extraction from point clouds. In *IEEE International Conference on Shape Modeling and Applications 2007 (SMI'07)*, pages 123–136. IEEE, 2007.

Datta A. and Soundaralakshmi S. Fast parallel algorithm for distance transform. *IEEE Trans. On Systems, Man, And Cybernetics-Part A: Systems And Humans*, 33(4):429–434, 2003.

Devo A., Mezzetti G., Costante G., Fravolini M. L., and Valigi P. Towards generalization in target-driven visual navigation by using deep reinforcement learning. *IEEE Transactions on Robotics*, 36(5):1546–1561, 2020.

Edwards J., Daniel E., Pascucci V., and Bajaj C. Approximating the generalized voronoi diagram of closely spaced objects. In *Computer Graphics Forum*, volume 34, pages 299–309. Wiley Online Library, 2015.

Elbaz G., Avraham T., and Fischer A. 3d point cloud registration for localization using a deep neural network auto-encoder. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4631–4640, 2017.

Elfes A. and others . Occupancy grids: A stochastic spatial representation for active robot perception. In *Proceedings of the Sixth Conference on Uncertainty in AI*, volume 2929, page 6. Morgan Kaufmann San Mateo, CA, 1990.

Eskandarian A., Wu C., and Sun C. Research advances and challenges of autonomous and connected ground vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 22(2):683–711, 2019.

Findeisen R., Imsland L., Allgower F., and Foss B. A. State and output feedback nonlinear model predictive control: An overview. *European Journal of Control*, 9(2-3):190–206, 2003.

Fortune S. Voronoi diagrams and delaunay triangulations. *Computing in Euclidean geometry*, pages 225–265, 1995.

Franchi A., Secchi C., Ryll M., Bulthoff H. H., and Giordano P. R. Shared control : Balancing autonomy and human assistance with a group of quadrotor uavs. *IEEE Robotics Automation Magazine*, 19(3):57–68, 2012.

Fu Y.-T., Hsu C.-M., Chen Z.-Y., and Chou J.-H. Path planning for continuous-curvature avoidance using hierarchical four parameter logistic curves. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3358–3363, 2020.

Furrer F., Burri M., Achtelik M., and Siegwart R. *Robot Operating System (ROS): The Complete Reference (Volume 1)*, chapter RotorS—A Modular Gazebo MAV Simulator Framework, pages 595–625. Springer International Publishing, Cham, 2016.

Gabellieri C., Tognon M., Sanalitro D., Pallottino L., and Franchi A. A study on force-based collaboration in swarms. *Swarm Intelligence*, 14(1): 57–82, 2020.

Garrido S., Moreno L., and Blanco D. Exploration of 2d and 3d environments using voronoi transform and fast marching method. *Journal of Intelligent and Robotic Systems*, 55(1):55–80, 2009.

Garrido Bullón L. S., Moreno Lorente L. E., Blanco Rojas M. D., and Jurewicz Slupska P. P. Path planning for mobile robot navigation using voronoi diagram and fast marching. 2011.

Gawel A., Kamel M., Novkovic T., Widauer J., Schindler D., von Altishofen B. P., Siegwart R., and Nieto J. Aerial picking and delivery of magnetic objects with mavs. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 5746–5752, 2017.

Godbolt M. Optimizations in c++ compilers. *Communications of the ACM*, 63(2):41–49, 2020.

Gojcic Z., Zhou C., Wegner J. D., and Wieser A. The perfect match: 3d point cloud matching with smoothed densities. In *Proceedings of the*

IEEE/CVF conference on computer vision and pattern recognition, pages 5545–5554, 2019.

Gomez E. J., Santa F. M. M., and Sarmiento F. H. M. A comparative study of geometric path planning methods for a mobile robot: potential field and voronoi diagrams. In *2013 II international congress of engineering mechatronics and automation (CIIMA)*, pages 1–6. IEEE, 2013.

Grisetti G., Grzonka S., Stachniss C., Pfaff P., and Burgard W. Efficient estimation of accurate maximum likelihood maps in 3d. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3472–3478. IEEE, 2007.

Güler S., Abdelkader M., and Shamma J. S. Peer-to-peer relative localization of aerial robots with ultrawideband sensors. *IEEE Transactions on Control Systems Technology*, 2020.

Gumhold S., Wang X., MacLeod R. S., and others . Feature extraction from point clouds. In *IMR*, pages 293–305, 2001.

Guo Y., Zhang Q., Wang J., and Liu S. Hierarchical reinforcement learning-based policy switching towards multi-scenarios autonomous driving. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021.

Gupta S., Davidson J., Levine S., Sukthankar R., and Malik J. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2616–2625, 2017.

Hernández-Vega J.-I., Varela E. R., Romero N. H., Hernández-Santos C., Cuevas J. L. S., and Gorham D. G. P. Internet of things (iot) for monitoring air pollutants with an unmanned aerial vehicle (uav) in a smart city. In *Smart technology*, pages 108–120. Springer, 2018.

Huang X., Mei G., and Zhang J. Feature-metric registration: A fast semi-supervised approach for robust point cloud registration without correspondences. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11366–11374, 2020.

Huang X., Mei G., Zhang J., and Abbas R. A comprehensive survey on point cloud registration. *arXiv preprint arXiv:2103.02690*, 2021a.

Huang X., Mei G., Zhang J., and Abbas R. A comprehensive survey on point cloud registration. *ArXiv*, abs/2103.02690, 2021b.

Ismail Z. H., Sariff N., and Hurtado E. A survey and analysis of cooperative multi-agent robot systems: challenges and directions. In *Applications of Mobile Robots*, pages 8–14. IntechOpen, 2018.

Jepsen J. H., Terkildsen K. H., Hasan A., Jensen K., and Schultz U. P. Uavat framework: Uav auto test framework for experimental validation of multirotor suas using a motion capture system. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 619–629. IEEE, 2021.

Jin J., Nguyen N. M., Sakib N., Graves D., Yao H., and Jagersand M. Mapless navigation among dynamics with social-safety-awareness: a reinforcement learning approach from 2d laser scans. In *2020 IEEE international conference on robotics and automation (ICRA)*, pages 6979–6985. IEEE, 2020.

Kaess M., Ranganathan A., and Dellaert F. isam: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, 24(6):1365–1378, 2008.

Kamel M., Verling S., Elkhatib O., Sprecher C., Wulkop P., Taylor Z., Siegwart R., and Gilitschenski I. The voliro omniorientational hexacopter: An agile and maneuverable tiltable-rotor aerial vehicle. *IEEE Robotics & Automation Magazine*, 25(4):34–44, 2018.

Karaman S. and Frazzoli E. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

Keidar M., Sadeh-Or E., and Kaminka G. A. Fast frontier detection for robot exploration. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 281–294, Berlin, Heidelberg, 2012. Springer.

Kerl C., Sturm J., and Cremers D. Dense visual slam for rgb-d cameras. In *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)*, 2013.

Kiseleva E., Hart L., Prytomanova O., and Kuzenkov O. An algorithm to construct generalized voronoi diagrams with fuzzy parameters based on the theory of optimal partitioning and neuro-fuzzy technologies. In *MoMLeT*, pages 148–162, 2019.

Koenig N. and Howard A. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154, 2004.

Kolahdouzan M. and Shahabi C. Voronoi-based k nearest neighbor search for spatial network databases. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 840–851, 2004.

Kondak K., Huber F., Schwarzbach M., Laiacker M., Sommer D., Bejar M., and Ollero A. Aerial manipulation robot composed of an autonomous helicopter and a 7 degrees of freedom industrial manipulator. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 2107–2112, 2014.

Koulamas C. and Lazarescu M. T. Real-time embedded systems: Present and future, 2018.

LaValle S. M. and others . Rapidly-exploring random trees: A new tool for path planning. 1998.

Lee D.-T. and Drysdale R. L., III. Generalization of voronoi diagrams in the plane. *SIAM Journal on Computing*, 10(1):73–87, 1981.

Lee H., Kim H., and Kim H. J. Planning and control for collision-free cooperative aerial transportation. *IEEE Trans. on Automation Science and*

Engineering, 15(1):189–201, 2018.

León J., Cardona G. A., Botello A., and Calderón J. M. Robot swarms theory applicable to seek and rescue operation. In *Int. Conf. on Intelligent Systems Design and Applications*, pages 1061–1070. Springer, 2016.

Li H., Zhang Q., and Zhao D. Deep reinforcement learning-based automatic exploration for navigation in unknown environment. *IEEE transactions on neural networks and learning systems*, 31(6):2064–2076, 2019.

Li T., Ho D., Li C., Zhu D., Wang C., and Meng M. Q.-H. Houseexpo: A large-scale 2d indoor layout dataset for learning-based algorithms on mobile robots. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5839–5846. IEEE, 2020.

Lindqvist B., Agha-Mohammadi A.-A., and Nikolakopoulos G. Exploration-rrt: A multi-objective path planning and exploration framework for unknown and unstructured environments. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 3429–3435, 2021.

Lippi M. and Marino A. Cooperative object transportation by multiple ground and aerial vehicles: Modeling and planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1084–1090. IEEE, 2018.

Lippiello V., Fontanelli G. A., and Ruggiero F. Image-based visual-impedance control of a dual-arm aerial manipulator. *IEEE Robotics and Automation Letters*, 3(3):1856–1863, 2018.

Lissandrini N., Michieletto G., Antonello R., Galvan M., Franco A., and Cenedese A. Cooperative optimization of uavs formation visual tracking. *Robotics*, 8(3):1–22 (Article Number 52), 2019. ISSN 2218-6581. DOI: https://doi.org/10.3390/robotics8030052.

Lissandrini N., Verginis C. K., Roque P., Cenedese A., and Dimarogonas D. V. Decentralized nonlinear mpc for robust cooperative manipulation by heterogeneous aerial-ground robots. In *2020 IEEE/RSJ International*

Conference on Intelligent Robots and Systems (IROS), pages 1531–1536. IEEE.

Liu S., Li S., Pang L., Hu J., Chen H., and Zhang X. Autonomous exploration and map construction of a mobile robot based on the tghm algorithm. *Sensors*, 20(2):490, 2020.

Lluvia I., Lazkano E., and Ansuategi A. Active mapping and robot exploration: A survey. *Sensors*, 21(7):2445, 2021.

López-González A., Campaña J. M., Martínez E. H., and Contro P. P. Multi robot distance based formation using parallel genetic algorithm. *Applied Soft Computing*, 86:105929, 2020.

Lowe D. G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

Lulu L. and Elnagar A. A comparative study between visibility-based roadmap path planning algorithms. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3263–3268. IEEE, 2005.

Mahkovic R. and Slivnik T. Constructing the generalized local voronoi diagram from laser range scanner data. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 30(6):710–719, 2000.

Mahony R., Kumar V., and Corke P. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics Automation Magazine*, 19(3):20–32, 2012.

Marie R., Said H. B., Stéphant J., and Labbani-Igbida O. Visual servoing on the generalized voronoi diagram using an omnidirectional camera. *Journal of Intelligent & Robotic Systems*, 94(3):793–804, 2019.

Masehian E. and Amin-Naseri M. A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning. *Journal of Robotic Systems*, 21(6):275–300, 2004.

Mashayekhi R., Idris M. Y. I., Anisi M. H., Ahmedy I., and Ali I. Informed rrt*-connect: An asymptotically optimal single-query path planning method. *IEEE Access*, 8:19842–19852, 2020.

Menolotto M., Komaris D.-S., Tedesco S., O'Flynn B., and Walsh M. Motion capture technology in industrial applications: A systematic review. *Sensors*, 20(19):5687, 2020.

Michieletto G., Ryll M., and Franchi A. Fundamental actuation properties of multirotors: Force–moment decoupling and fail–safe robustness. *IEEE Transactions on Robotics*, 34(3):702–715, 2018.

Milijas R., Markovic L., Ivanovic A., Petric F., and Bogdan S. A comparison of lidar-based slam systems for control of unmanned aerial vehicles. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1148–1154. IEEE, 2021.

Mirowski P., Pascanu R., Viola F., Soyer H., Ballard A. J., Banino A., Denil M., Goroshin R., Sifre L., Kavukcuoglu K., and others . Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

Molinos E. J., Llamazares A., and Ocaña M. Dynamic window based approaches for avoiding obstacles in moving. *Robotics and Autonomous Systems*, 118:112–130, 2019.

Montiel O., Orozco-Rosas U., and Sepúlveda R. Path planning for mobile robots using bacterial potential field for avoiding static and dynamic obstacles. *Expert Systems with Applications*, 42(12):5177–5191, 2015a.

Montiel O., Sepúlveda R., and Orozco-Rosas U. Optimal path planning generation for mobile robots using parallel evolutionary artificial potential field. *Journal of Intelligent & Robotic Systems*, 79(2):237–257, 2015b.

Naldi R., Gasparri A., and Garone E. Cooperative pose stabilization of an aerial vehicle through physical interaction with a team of ground robots. In *IEEE Int. Conf. on Control Applications*, pages 415–420, 2012.

Nascimento T. P., Moreira A. P., and Conceição A. G. S. Multi-robot nonlinear model predictive formation control: Moving target and target absence. *Robotics and Autonomous Systems*, 61(12):1502–1515, 2013.

Nguyen T. and Garone E. Control of a uav and a ugv cooperating to manipulate an object. In *American Control Conference (ACC)*, pages 1347–1352, 2016.

Nikou A., Verginis C., Heshmati-Alamdari S., and Dimarogonas D. V. A nonlinear model predictive control scheme for cooperative manipulation with singularity and collision avoidance. In *25th Mediterranean Conf. on Control and Automation (MED)*, pages 707–712, 2017.

Niroui F., Zhang K., Kashino Z., and Nejat G. Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments. *IEEE Robotics and Automation Letters*, 4(2):610–617, 2019.

Ocando M. G. and others . Autonomous 2d slam and 3d mapping of an environment using a single 2d lidar and ros. In *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*. IEEE, 2017.

Ollero A., Heredia G., Franchi A., Antonelli G., Kondak K., Sanfeliu A., Viguria A., Martinez-de Dios J. R., Pierri F., Cortés J., and others . The aeroarms project: Aerial robots with advanced manipulation capabilities for inspection and maintenance. *IEEE Robotics & Automation Magazine*, 25 (4):12–23, 2018.

Ollero A., Tognon M., Suarez A., Lee D., and Franchi A. Past, present, and future of aerial robotic manipulators. *IEEE Transactions on Robotics*, 2021.

Oskiper T., Zhu Z., Samarasekera S., and Kumar R. Visual odometry system using multiple stereo cameras and inertial measurement unit. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

Paden B., Čáp M., Yong S. Z., Yershov D., and Frazzoli E. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Trans. on Intelligent Vehicles*, 1(1):33–55, 2016.

Pairet È., Hernández J. D., Carreras M., Petillot Y., and Lahijanian M. Online mapping and motion planning under uncertainty for safe navigation in unknown environments. *IEEE Trans. on Automation Science and Engineering*, 2021.

Paneque J. L., Martínez-de Dios J., and Ollero A. Multi-sensor 6-dof localization for aerial robots in complex gnss-denied environments. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1978–1984. IEEE, 2019.

Paszke A., Gross S., Massa F., Lerer A., Bradbury J., Chanan G., Killeen T., Lin Z., Gimelshein N., Antiga L., and others . Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Patel U., Kumar N. K. S., Sathyamoorthy A. J., and Manocha D. Dwa-rl: Dynamically feasible deep reinforcement learning policy for robot navigation among mobile obstacles. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6057–6063. IEEE, 2021.

Paternain S., Koditschek D. E., and Ribeiro A. Navigation functions for convex potentials in a space with convex obstacles. *IEEE Trans. on Automatic Control*, 63(9):2944–2959, 2017.

Prasad A., Sharma B., Vanualailai J., and Kumar S. Stabilizing controllers for landmark navigation of planar robots in an obstacle-ridden workspace. *Journal of Advanced Transportation*, 2020, 2020.

Praveen A., Ma X., Manoj H., Venkatesh V. L., Rastgaar M., and Voyles R. M. Inspection-on-the-fly using hybrid physical interaction control for aerial manipulators. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1583–1588, 2020.

Qi J., Yang H., and Sun H. Mod-rrt*: A sampling-based algorithm for robot path planning in dynamic environment. *IEEE Trans. on Industrial Electronics*, 68(8):7244–7251, 2020.

Quin P., Alempijevic A., Paul G., and Liu D. Expanding wavefront frontier detection: An approach for efficiently detecting frontier cells. In *Australasian Conference on Robotics and Automation, ACRA*, 2014.

Raj T., Hashim F. H., Huddin A. B., Ibrahim M. F., and Hussain A. A survey on lidar scanning mechanisms. *Electronics*, 9(5):741, 2020.

Rajappa S. *Towards Human-UAV Physical Interaction and Fully Actuated Aerial Vehicles*. Logos Verlag, 2018.

Ramos F. and Ott L. Hilbert maps: Scalable continuous occupancy mapping with stochastic gradient descent. *The International Journal of Robotics Research*, 35(14):1717–1730, 2016.

Rashad R., Bicego D., Zult J., Sanchez-Escalonilla S., Jiao R., Franchi A., and Stramigioli S. Energy aware impedance control of a flying end-effector in the port-hamiltonian framework. *IEEE transactions on robotics*, 2021.

Rashad R., Goerres J., Aarts R., Engelen J. B., and Stramigioli S. Fully actuated multirotor uavs: A literature review. *IEEE Robotics & Automation Magazine*, 27(3):97–107, 2020.

Rubio F., Valero F., and Llopis-Albert C. A review of mobile robots: Concepts, methods, theoretical framework, and applications. *International Journal of Advanced Robotic Systems*, 16(2):1729881419839596, 2019.

Rublee E., Rabaud V., Konolige K., and Bradski G. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

Rusinkiewicz S. and Levoy M. Efficient variants of the icp algorithm. In *Proceedings third international conference on 3-D digital imaging and modeling*, pages 145–152. IEEE, 2001.

Ryll M., Bicego D., Giurato M., Lovera M., and Franchi A. Fast-hex–a morphing hexarotor: Design, mechanical implementation, control and experimental validation. *IEEE/ASME Transactions on Mechatronics*, 2021.

Ryll M., Muscio G., Pierri F., Cataldi E., Antonelli G., Caccavale F., Bicego D., and Franchi A. 6d interaction control with aerial robots: The flying end-effector paradigm. *The International Journal of Robotics Research*, 38 (9):1045–1062, 2019.

Salzman O. and Halperin D. Asymptotically near-optimal rrt for fast, high-quality motion planning. *IEEE Trans. on Robotics*, 32(3):473–483, 2016.

Sandakalum T. and Ang Jr M. H. Motion planning for mobile manipulators—a systematic review. *Machines*, 10(2):97, 2022.

Santamaria-Navarro A., Loianno G., Sola J., Kumar V., and Andrade-Cetto J. Autonomous navigation of micro aerial vehicles using high-rate and low-cost sensors. *Autonomous robots*, 42(6):1263–1280, 2018.

Shakhatreh H., Sawalmeh A. H., Al-Fuqaha A., Dou Z., Almaita E., Khalil I., Othman N. S., Khreishah A., and Guizani M. Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges. *Ieee Access*, 7:48572–48634, 2019.

Shawky D., Yao C., and Janschek K. Nonlinear model predictive control for trajectory tracking of a hexarotor with actively tiltable propellers. In *2021 7th International Conference on Automation, Robotics and Applications (ICARA)*, pages 128–134. IEEE, 2021.

Shen T., Luo Z., Zhou L., Deng H., Zhang R., Fang T., and Quan L. Beyond photometric loss for self-supervised ego-motion estimation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6359–6365. IEEE, 2019.

Shi F., Zhao M., Anzai T., Chen X., Okada K., and Inaba M. External wrench estimation for multilink aerial robot by center of mass estimator based on distributed imu system. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1891–1897. IEEE, 2019.

Sirtkaya S., Seymen B., and Alatan A. A. Loosely coupled kalman filtering for fusion of visual odometry and inertial navigation. In *Proceedings of the 16th International Conference on Information Fusion*, pages 219–226. IEEE, 2013.

Staub N., Mohammadi M., Bicego D., Prattichizzo D., and Franchi A. Towards robotic magmas: Multiple aerial-ground manipulator systems. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1307–1312, 2017.

Stefek A., Van Pham T., Krivanek V., and Pham K. L. Energy comparison of controllers used for a differential drive wheeled mobile robot. *IEEE Access*, 8:170915–170927, 2020.

Suarez A., Heredia G., and Ollero A. Lightweight compliant arm with compliant finger for aerial manipulation and inspection. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 4449–4454, 2016.

Suarez A., Real F., Vega V. M., Heredia G., Rodriguez-Castano A., and Ollero A. Compliant bimanual aerial manipulation: Standard and long reach configurations. *IEEE Access*, 8:88844–88865, 2020.

Sun Z., Wu B., Xu C.-Z., Sarma S. E., Yang J., and Kong H. Frontier detection and reachability analysis for efficient 2d graph-slam based active exploration. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2051–2058, 2020.

Takahashi O. and Schilling R. J. Motion planning in a plane using generalized voronoi diagrams. *IEEE Transactions on robotics and automation*, 5(2): 143–150, 1989.

Teimoori H. and Savkin A. V. A biologically inspired method for robot navigation in a cluttered environment. *Robotica*, 28(5):637–648, 2010.

Tidd B., Cosgun A., Leitner J., and Hudson N. Learning when to switch: composing controllers to traverse a sequence of terrain artifacts. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5144–5150. IEEE, 2021.

Tognon M., Chávez H. A. T., Gasparin E., Sablé Q., Bicego D., Mallet A., Lany M., Santi G., Revaz B., Cortés J., and others . A truly-redundant aerial manipulator system with application to push-and-slide inspection in industrial plants. *IEEE Robotics and Automation Letters*, 4(2):1846–1851, 2019.

Tu X., Xu C., Liu S., Xie G., and Li R. Real-time depth estimation with an optimized encoder-decoder architecture on embedded devices. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 2141–2149. IEEE, 2019.

Tzoumanikas D., Graule F., Yan Q., Shah D., Popovic M., and Leutenegger S. Aerial manipulation using hybrid force and position nmpc applied to aerial writing. *arXiv preprint arXiv:2006.02116*, 2020.

Upadhyay S., Kumar S., and Krishna K. M. Crf based frontier detection using monocular camera. In *Proceedings of the 2014 Indian Conference on Computer Vision Graphics and Image Processing*, pages 1–7, 2014.

Valette S., Chassery J. M., and Prost R. Generic remeshing of 3d triangular meshes with metric-dependent discrete voronoi diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 14(2):369–381, 2008.

Verginis C. K., Nikou A., and Dimarogonas D. V. Communication-based decentralized cooperative object transportation using nonlinear model predictive control. *European Control Conf. (ECC)*, pages 733–738, 2018.

Wahrmann D., Hildebrandt A.-C., Schuetz C., Wittmann R., and Rixen D. An autonomous and flexible robotic framework for logistics applications. *Journal of Intelligent & Robotic Systems*, 93(3):419–431, 2019.

Wang L., Chen J., Li X., and Fang Y. Non-rigid point set registration networks. *arXiv preprint arXiv:1904.01428*, 2019.

Wen J., Zhang X., Bi Q., Pan Z., Feng Y., Yuan J., and Fang Y. Mrpb 1.0: A unified benchmark for the evaluation of mobile robot local planning approaches. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8238–8244. IEEE, 2021.

Wilmsen M., Yao C., Schuster M., Li S., and Janschek K. Nonlinear wrench observer design for an aerial manipulator. *IFAC-PapersOnLine*, 52(22):1–6, 2019.

Wilson S., Glotfelter P., Wang L., Mayya S., Notomista G., Mote M., and Egerstedt M. The robotarium: Globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of multi-robot systems. *IEEE Control Systems Magazine*, 40(1):26–44, 2020.

Wurm K. M., Dornhege C., Nebel B., Burgard W., and Stachniss C. Coordinating heterogeneous teams of robots using temporal symbolic planning. *Autonomous Robots*, 34(4):277–294, 2013.

Xia F., Shen W. B., Li C., Kasimbeg P., Tchapmi M. E., Toshev A., Martín-Martín R., and Savarese S. Interactive gibson benchmark: A benchmark for interactive navigation in cluttered environments. *IEEE Robotics and Automation Letters*, 5(2):713–720, 2020.

Yang J., Zhao C., Xian K., Zhu A., and Cao Z. Learning to fuse local geometric features for 3d rigid data matching. *Information Fusion*, 61: 24–35, 2020.

Yang Q. and Yoo S.-J. Optimal uav path planning: Sensing data acquisition over iot sensor networks using multi-objective bio-inspired algorithms. *IEEE Access*, 6:13671–13684, 2018.

Yao Q., Zheng Z., Qi L., Yuan H., Guo X., Zhao M., Liu Z., and Yang T. Path planning method with improved artificial potential field—a reinforcement learning perspective. *IEEE Access*, 8:135513–135523, 2020.

Yuan W., Eckart B., Kim K., Jampani V., Fox D., and Kautz J. Deepgmr: Learning latent gaussian mixture models for registration. In *European conference on computer vision*, pages 733–750. Springer, 2020.

Yüksel B., Secchi C., Bülthoff H. H., and Franchi A. Aerial physical interaction via ida-pbc. *The International Journal of Robotics Research*, 38(4): 403–421, 2019.

Zhang F., Chen W., and Xi Y. Motion synchronization in mobile robot networks: Robustness. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5570–5575. IEEE, 2006.

Zhang T., Qiu T., Pu Z., Liu Z., and Yi J. Robot navigation among external autonomous agents through deep reinforcement learning using graph attention network. *IFAC-PapersOnLine*, 53(2):9465–9470, 2020.

Zhao S., Dimarogonas D. V., Sun Z., and Bauso D. A general approach to coordination control of mobile agents with motion constraints. *IEEE Trans. on Automatic Control*, 63(5):1509–1516, 2017.

Zhu Y., Zhang T., and Song J.-Y. Study on the local minima problem of path planning using potential field method in unknown environments. *Acta automatica sinica*, 36(8):1122–1130, 2010.

Zwecher E., Iceland E., Levy S. R., Hayoun S. Y., Gal O., and Barel A. Integrating deep reinforcement and supervised learning to expedite indoor mapping. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 10542–10548. IEEE, 2022.