POLITECNICO DI MILANO
DEPARTMENT OF AEROSPACE SCIENCE AND TECHNOLOGY
DOCTORAL PROGRAMME IN AEROSPACE ENGINEERING

GUIDANCE AND NAVIGATION ALGORITHMS FOR AUTONOMOUS MULTIROTOR UAVS

Doctoral Dissertation of:
Gabriele Roggi

Supervisor:
Prof. Marco Lovera

Tutor:
Prof. Michèle Lavagna

The Chair of the Doctoral Program:
Prof. Pierangelo Masarati

2022 – XXXV

Gabriele Roggi
Dipartimento di Scienze e Tecnologie Aerospaziali
Politecnico di Milano
Via Giuseppe La Masa, 34
20156 Milano, Italy
E-mail: gabriele.roggi@polimi.it

Abstract

The interest and the possible applications of Unmanned Aerial Vehicles (UAVs) have been increasing quickly in the last few years. At the current state, the vehicles can be either teleoperated on an actuator set-point or waypoint level or execute a flight path planned on a ground control station, constantly under direct line of sight with an operator. At the same time, the platform relies on Global Navigation Satellite System (GNSS) information for its navigation. The presented way of conducting operations displays some critical issues. First, in the case of teleoperation, the stress level on the operator is quite high and this can potentially lead to errors or damages to the platform. Communication constraints between the operator and the UAV could further complicate things. Then, operations are limited by the availability of GNSS signal, thus excluding indoor or other challenging environments. Consequently, in order to deploy their full potential, these machines have to navigate autonomously without any user intervention or the need for any external infrastructure. In this framework, autonomous drones could take decisions without relying on downlinked data in a time-efficient way and could execute high-level commands and reach complex objectives without the need for a human operator who translates the goals of the mission into position waypoints.

This thesis involves the development, simulation and experimental val-

idation of guidance and navigation algorithms for this kind of machine. Concerning the great advancements of robotics in nearly every research direction for fully autonomous UAVs, this thesis offers different software and hardware solutions capable of large-scale indoor GNSS-denied navigation and collision avoidance in cluttered environments. Extensive simulation and experimental testing, during both laboratory experiments and autonomous drone competitions, are carried out.

Moreover, the dissertation tackles the problem of the systematic characterization of vision systems. In this framework, the proposed approach addresses some issues related to the integration of vision-based state estimates into autonomous navigation systems.

Finally, the problem of guidance and navigation for performing autonomous landing in different scenarios is addressed. In particular, the proposed solutions tackle some of the problems of an autonomous emergency landing in case of faults or malfunctions in an indoor environment and Air-to-Air Automatic Landing, *i.e.*, the landing of a small drone on top of a larger one during flight.

Contents

List of Figures

List of Tables

Introduction

In recent years, there has been an increasing interest in Unmanned Aerial Vehicles (UAVs) for both military and civil applications. Some examples regard their use in precision agriculture, photography, patrolling and surveillance, search and rescue, entertainment, product delivery, aerial inspection, research and many others. Different classifications of UAVs exist. For instance, they can be classified based on the size, takeoff weight, control configuration or operational purpose [1]. Depending on the specific application, requirements may vary and the appropriate configuration may differ quite significantly. Note that when referring to UAVs or drones, in this work, we will consider the category of multi-rotor Vertical Take-Off and Landing (VTOL) vehicles of small/medium size provided with a number of rotors greater than two, co-planar among them. The motion of the system is controlled by adjusting the angular speed of each propeller which, in turn, will generate thrust and torque. Another possible classification regards the autonomy level of the platform [2].

In general terms, autonomy can be defined as the ability of a system to achieve goals while operating independently of external control. This implies the need for two basic requirements, *i.e.*, self-directedness (to achieve goals) and self-sufficiency (to operate independently). This represent a huge paradigm shift with respect to automation, in which the sys-

tem requires commands, *e.g.*, a pre-planned set of instructions), to reach goals.

Referring to drones, the National Institute of Standards and Technology of United States [3] defined the levels of automation as follows:

- Remotely controlled: a human operator controls the UAV via only direct observation (without sensors feedback).
- Teleoperated: a human operator, while receiving sensors feedback, sends actuator control commands or intermediate goals to the UAV.
- Semi-autonomous: a human operator plans the mission and takes key decisions. The UAV conducts autonomous operations in between these interactions with the operator.
- Fully autonomous: the UAV accomplishes its mission without human intervention and while adapting to operational and/or environmental changes.

At the state-of-the-art, most of the civil applications belongs to the first two levels.

As a practical example consider an UAV for search-and-rescue, disaster response or surveillance missions. At the current state, the vehicles are teleoperated on an actuator set-point or waypoint level, constantly under direct line of sight with the operator and an average of three expert professionals is required for each robot to control them [4]. Wireless connection for telemetry and Global Navigation Satellite System (GNSS) could be hardly maintained, especially in challenging or indoor environments. Since such systems exhibit very limited or no autonomy, the stress level on the operator is very high, which limits the mission time drastically. Operator errors could harm the robot or, even worse, cause further damage. For these reasons, there is a large need for flying robots that can navigate autonomously, without any user intervention, or execute high-level commands like exploring and mapping a given area or finding a target, without the need to assign cumbersome actuator set-points or position waypoints [5].

The self-sufficiency of an autonomous system leads to technical difficulties for what concerns its navigation. Without the possibility of re-

lying on external navigation systems, *i.e.*, GNSS, an autonomous vehicle must estimate its ego-motion through the use of onboard sensors and computing. While in the ground robotics domain, combining wheel odometry with sensors such as laser range-finders, sonars or cameras in probabilistic Simultaneous Localization and Mapping (SLAM) framework has been proven very successful, in aerospace applications limited sensing payloads and computational power represent additional hurdles [6, 7, 8]. This weight limitation forces aerial vehicles to rely on lightweight laser scanner, cameras and lower-quality Micro Electro-Mechanical Systems (MEMS)-based Inertial Measurement Units (IMUs), all of which with limited ranges, field-of-view and noisier compared to their ground equivalents and, at the same time, the vehicle must run computationally intensive 3D SLAM algorithms on small embedded computers. Contrarily to ground vehicles, aerial vehicles are unable to directly measure odometry since they have not physical contact with the environment. Robot motion must be recovered using exteroceptive sensors, thus computing the vehicle's motion relative to reference points in the environment using sensors like cameras, *e.g.*, through techniques like Visual Odometry (VO) or Visual Inertial Odometry (VIO). The obtained odometry information is then used for initializing the estimate of the vehicle's motion between time steps. Finally, consider that an UAV cannot simply stop when its state estimates have large uncertainties, but the vehicle is likely to oscillate and degrade its sensor measurement even further.

All these reasoning also lead to additional considerations regarding the guidance of autonomous aerospace systems. Planning algorithms must not only care about finding collision-free smooth paths toward a target, but they must also consider the inherent uncertainty in the planning process itself and potential intermediate paths needed for re-localization. The exploration and mapping of an unknown environment can thus become a very complex decision making problem, in which a trade-off between exploration and localization of the agent must be found. Furthermore, for the underlying 3D path planning task, difficulties arise in dynamic environments, in which the potential motion of objects in the environment must be distinguished from the robot ego-motion. At the same time, computationally efficient re-planning capabilities assume paramount im-

portance for facing both uncertainties in the state estimate and possible moving objects.

State of the art

In the latest years, robotics advancements in nearly every research direction were integrated in fully autonomous UAVs capable of flying relying only on onboard sensors and computing.

Initial works on state estimation for aerial robots with purely onboard sensing were performed in [7] using both 2D laser range finders and stereo camera configurations. Monocular cameras were instead employed in [9, 10]. In [6], the authors used laser range finders for computing globally consistent state estimates combining scan matching odometry computation with a SLAM algorithm. The authors also employed the map produced by the SLAM algorithm for autonomously planning a path for exploring the environment. Few years later, in [8], the authors added localization, map optimization and obstacle avoidance to the framework. Both scanning laser and monocular camera were instead employed in [11]. In most of these papers, the environment was structured and a 2.5D approximation was often employed. In addition to this, only part of the software stacks was running onboard the drones.

In the following years, with the improvement of vision algorithms, flight in unstructured 3D environments became possible [12, 13, 14] using only onboard computations. At the same time, works were carried out for building accurate 3D maps and generating collision-free trajectories on them [15, 16]. Finally, in recent years, a significant amount of research has been devoted to the development of platforms equipped with all the relevant building blocks, *i.e.*, state estimation, mapping, planning and control. For instance, consider [17], in which the authors presented the first UAV running vision based localization, mapping, and planning without prior knowledge of the environment entirely onboard and in real-time. In [18], the authors designed a full navigation stack which allowed the drone to reach a goal location, while avoiding obstacles. The proposed architecture has been also employed in [19] for large-scale flight. In [20], the authors developed a vision-based autonomous UAV capable of GNSS-denied navigation, contributing also to important advancements

for 3D global planning on real maps and local planning with re-planning capabilities. In [21] an autonomous UAV is presented with focus on its state estimation and control. In addition, the authors presented the high-level tasks the drone had been able to accomplish.

Some other research groups focused on the agility of such autonomous platforms. For instance, in [22] a full stack capable of short-scale autonomous flight is presented. Similar platforms have been also discussed in [23, 24]. Finally, an open source framework (both software and hardware) for perception, planning and control of agile multicopters is available in [25]. This latter paper offers also a comparison among other architectures like [24, 22].

At the same time, in the robotics community, there has been a significant growth in the number of challenge prizes and competitions. This has been done with the aim of stimulating innovation to meet a defined challenge and to provide solutions to problems that matter to roboticists and society [26]. Robot competitions offer also a way to cope with the inherent difficulties associated with robotics benchmarking [27]. Competitions allow to test and compare methods in the controlled conditions of the specific robotic challenge.

In particular, drone racing has seen autonomous systems catching up fast with human performance. In this framework, while the first drone racing competition has been organized during the IROS conference in 2016 [28], the most important event has been the 2019 Lockheed Martin AlphaPilot challenge [29]. This challenge led to the first season of Artificial Intelligence Robotic Racing (AIRR), co-organized with the Drone Racing League (DRL) for human pilots [30]. Other than drone racing, autonomous UAVs have been object of competitions in several conferences, *e.g.*, IMAV (International Micro Air Vehicle) [31] and ICUAS (International Conference on Unmanned Aircraft Systems), and in the Mohamed Bin Zayed International Robotic Challenge (MBZIRC) [32], DARPA Fast Lightweight Autonomy Program (2015-2018) [33] and DARPA Subterranean Challenge (2018-2021) [34].

Contributions

This work involves the development, simulation and experimental validation of guidance and navigation algorithms for autonomous multirotor UAVs. The contributions of this thesis are threefold. First, different hardware and software solutions for fully autonomous UAVs are proposed and compared. In particular, large-scale indoor GNSS-denied navigation and collision avoidance in cluttered environments is addressed. These platforms have been tested in the framework of an autonomous drone competition: the Leonardo Drone Contest (LDC). With respect to the relevant literature, the proposed work contributes to show the feasibility of fully autonomous navigation and planning architectures (similarly to [20]), but with the addition of decision making and mission management layers for the accomplishment of high-level tasks (like the ones presented in [21]). The suggested software architectures have the advantage of being entirely based on the open-source firmware PX4 [35]. At the same time, the hardware architectures will be composed by commercial off-the-shelf components. These latter two aspects make the platforms easily reproducible for research purposes. Finally, each competition will also offer the opportunity for presenting educational contributions related to the technical challenges faced.

Second, a systematic approach to the characterization of vision systems is proposed. In fact, despite the effort to improve the performance of visual odometry systems, both in terms of computational speed and accuracy, published work on this topic has paid little attention to the issue of integrating visual odometry estimates into autonomous navigation systems. As a matter of fact, the availability of a description of the noise dynamics, which for other systems, such as Inertial Navigation System (INS) and GNSS, are already well known, would be of great benefit in a sensor-fusion-oriented framework.

Third and last contribution of this work regards the topic of autonomous landing. This problem is tackled considering two use-cases: Air-to-Air Automatic Landing (AAAL) and emergency landing for autonomous UAVs. AAAL becomes fundamental in the context of the Air-to-Air Automatic Refuelling (AAAR), where the mission endurance of smaller drones (followers) is extended allowing them to take-off and land on top of larger and

heavier drones (carriers) during flight. In this thesis, a non-cooperative approach for performing the task using vision for tracking and landing on the target drone has been proposed. On the other hand, emergency landing is invaluable for forced landings of autonomous systems in case of faults or malfunctions. It can be also used for planned landings, *e.g.*, saving energy during monitoring operations or for the delivery of goods [36].

Structure

The thesis is organized as follows:

- Chapter 1 offers a review of background material on guidance, navigation and control techniques for autonomous robots and, in particular, for UAVs. Since most of the topics examined come from varied backgrounds, ranging from aerospace to robotics and from control to computer science engineering, this Chapter serves also the purpose to state some terminology and notation employed throughout the work.
- Chapter 2 presents the simulation and experimental environments used for tests and experiments. This Chapter describes also the aerial platforms as well as the common software architecture they are based on.
- Chapter 3 presents a methodological approach for the quantification and modeling of visual odometry errors. The approach is then validated and, finally, a sensitivity analysis to changes in the operating conditions is presented and discussed.
- Chapter 4 describes the LDC competitions. For each competition the rules, objectives and proposed solutions are presented. The software and hardware architectures are evaluated in both simulations and experimental activities.
- Chapter 5 describes the approaches used for both AAAL and emergency landing. Simulation and experimental campaigns have been carried out for validating the proposed approaches.

Published works and M.Sc. theses

Part of the material presented in this dissertation has been published in the following works.

Conference papers:

1. G. Roggi, M. Giurato, M. Lovera, *A computer vision line-tracking algorithm for UAV GNSS-aided guidance*, XXV International Congress of the Italian Association of Aeronautics and Astronautics, Rome, Italy, 2019.

2. G. Roggi, S. Meraglia, M. Lovera. *Leonardo Drone Contest 2021: Politecnico di Milano team architecture*, 2022 International Conference on Unmanned Aircraft Systems, Dubrovnik, Croatia, 2022;

Journal papers:

1. G. Roggi, A. Niccolai, F. Grimaccia, M. Lovera, *A computer vision line-tracking algorithm for automatic UAV photovoltaic plants monitoring applications*, Energies, 2020;

2. G. Roggi, G. Gozzini, D. Invernizzi, M.Lovera, *Vision-based Air-to-Air Autonomous Landing of UAVs*, Transactions on Mechatronics, 2022 (submitted);

3. G. Roggi, S. Meraglia, M. Lovera, *Leonardo Drone Contest autonomous drone competition: overview, results, and lessons learned from Politecnico di Milano team*, Journal of Intelligent & Robotic Systems, 2022 (submitted);

4. G. Roggi, M. Lovera, *Visual odometry error modeling for multicopter UAVs* (in preparation);

Moreover, the Author has been involved in a number of *M.Sc.* theses as co-advisor:

1. N. Damino. *Autonomous landing based on computer vision fiduciary algorithms*. Automation and Control Engineering, *M.Sc.*, 2020;

2. D. Avila. *A ROS implementation of a 6-DoF EKF for indoor drone Visual SLAM*. Computer Science and Engineering, *M.Sc.*, 2020;

3. F. Chiarlo. *A statistical analysis of position and attitude estimation errors in visual odometry*. Aeronautical Engineering, *M.Sc.*, 2021;

4. G. Fontanarosa. *UAV 3D guidance algorithms*. Aeronautical Engineering, *M.Sc.*, 2021;

5. M. Padovani. *A deep reinforcement learning-based algorithm for exploration planning*. Aeronautical Engineering, *M.Sc.*, 2021;

6. P. Giannagostino. *Model identification and statistical analysis in visual odometry on an UAV*. Aeronautical Engineering, *M.Sc.*, 2022;

CHAPTER 1

Preliminaries

In this Chapter, some preliminaries about guidance, navigation and control for UAVs are presented and discussed. For sake of clarity, the meaning of these terms is the one employed in the aerospace sector, namely:

- Guidance refers to the computation of the desired path and trajectory from the vehicle's current pose (position and attitude) to the target pose. It might also include the desired velocity, angular velocity and accelerations for following the given trajectory.
- Navigation refers to the determination, at each time instant, of the vehicle's state, which might include the position, velocity, attitude etc.
- Control refers to the computation of the forces and moments, and the associated change in rotors' angular speed, needed for executing the guidance commands.

1.1 Notation

1.1.1 Reference frames

We indicate with $\mathcal{I} = (O_I, \{i_1, i_2, i_3\})$ a generic inertial frame, where O_I is the origin of the frame, and i_1, i_2 and i_3 are respectively the three orthonormal vectors pointing East, North and Up. When needed, we further distinguish the inertial frames into *odom* and *map* frame as in [37]. The *map* frame, indicated with $\mathcal{M} = (O_M, \{m_1, m_2, m_3\})$, is an East-North-Up (ENU) world-fixed frame which has its origin O_M at some arbitrarily chosen point in the world. At the same time, the *odom* frame $\mathcal{O} = (O_O, \{o_1, o_2, o_3\})$ is a world-fixed ENU frame which drifts over time with respect to $\mathcal{M}$. Its origin O_O is positioned where the robot has been initialized. The body reference frame $\mathcal{B} = (O_B, \{b_1, b_2, b_3\})$ instead is a right-handed frame attached to the body of the UAV and it is centered in its Center of Mass (CoM). The first axis lies in the plane of symmetry and it points forward, the second axis points to the left and the third axis points up. The camera-fixed frame $\mathcal{C} = (O_C, \{c_1, c_2, c_3\})$ is the right-handed frame conventionally used in the pin-hole camera model (see Appendix B). Finally, we define a target frame $\mathcal{T} = (O_T, \{t_1, t_2, t_3\})$, an ENU frame attached to a generic fiducial visual marker. The presented reference frames are graphically represented in Figure 1.1.

1.1.2 Frame transformation

The pose of a frame $\mathcal{I}$ relative to a frame $\mathcal{B}$ can be specified by its position and orientation $T_{\mathcal{IB}} = (t_{\mathcal{IB}}, \Psi_{\mathcal{IB}})$, where $t_{\mathcal{IB}}$ is the translation vector indicating the position of the origin of $\mathcal{B}$ in frame $\mathcal{I}$, and Ψ_{IB} is the orientation of frame $\mathcal{B}$ with respect to frame $\mathcal{I}$. For example, a point written in the frame $\mathcal{B}$, can be expressed in frame $\mathcal{I}$ by:

$$p_{\mathcal{I}} = R\{\Psi_{\mathcal{IB}}\}p_{\mathcal{B}} + t_{\mathcal{IB}} \tag{1.1}$$

whereas the opposite relation is:

$$p_{\mathcal{B}} = R\{\Psi_{\mathcal{IB}}\}^{\top}(p_{\mathcal{I}} - t_{\mathcal{IB}}) \tag{1.2}$$

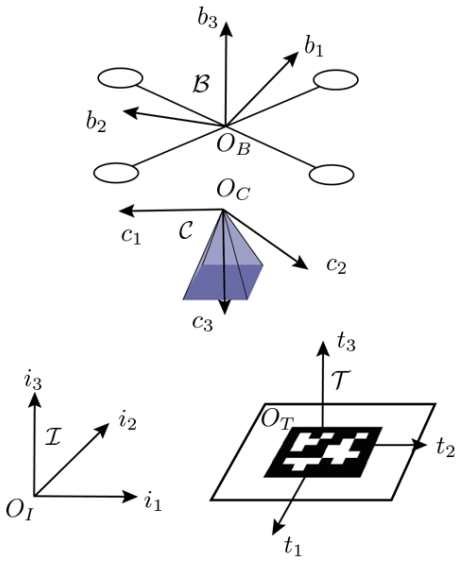


Figure 1.1: *Reference frames*

In the rest of the work, whenever no sub-index is used, the point or vector should be considered written in the global inertial frame. In the above expressions, $R\{\Psi_{\mathcal{IB}}\} \in SO(3)$ is the rotation matrix associated to the orientation $\Psi_{\mathcal{IB}}$. Its expression depends on the particular attitude parameterization we are using.

Euler angles

An attitude representation can be obtained by using a set of three angles. Consider the rotation matrix expressing the elementary rotation about one of the coordinate axes as a function of a single angle. Then, a generic rotation matrix can be obtained by composing a suitable sequence of three elementary rotations while guaranteeing that two successive rotations are not made about parallel axes. This implies that 12 distinct sets of angles are allowed out of all 27 possible combinations; each set represents a triplet of *Euler angles* [38]. In mobile robotics, usually the Z-Y-X sequence, also called Roll–Pitch– Yaw (R-P-Y) angles, is used. It includes a rotation of an angle ϕ (roll) about the first axis, then of θ (pitch) about

the new second axis and of ψ (yaw) about the newer third axis. Three elementary rotations in $SO(3)$ are considered:

$$R_Z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{1.3}$$

$$R_Y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}, \tag{1.4}$$

$$R_X(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}. \tag{1.5}$$

The matrix R is obtained by multiplying:

$$R\{\phi,\theta,\psi\} = R_Z(\phi)R_Y(\theta)R_X(\phi) = \tag{1.6}$$

$$= \begin{bmatrix} c\theta c\psi & -c\phi s\psi + s\phi s\theta c\psi & s\phi s\psi + c\phi s\theta c\psi \\ c\theta s\psi & c\phi c\psi + s\phi s\theta s\psi & -s\phi c\psi + c\phi s\theta s\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix}. \tag{1.7}$$

where s and c are abbreviations for $\sin$ and $\cos$, respectively. The defined ranges for the rotation angles are:

$$-\pi < \phi \leq \pi \tag{1.8}$$

$$-\pi/2 \leq \theta \leq \pi/2 \tag{1.9}$$

$$-\pi < \psi \leq \pi \tag{1.10}$$

Consider then the following equations to extract the Euler angles from the rotation matrix.

$$\phi = \arctan\left(\frac{r_{32}}{r_{33}}\right), \tag{1.11}$$

$$\theta = \arctan\left(-\frac{r_{31}}{\sqrt{r_{32}^2 + r_{33}^2}}\right), \tag{1.12}$$

$$\psi = \arctan\left(\frac{r_{21}}{r_{11}}\right), \tag{1.13}$$

where r_{ij} are the elements of the attitude matrix R placed at the *i-th* row and *j-th* column.

Unit quaternion

Quaternions can be seen as extended complex numbers. While regular complex numbers of unit length can encode rotations in the 2D plane, "extended complex numbers" or quaternions of unit length can encode rotations in 3D space [39]. Most commonly, the quaternion is thought as composed of a scalar and a vector part:

$$Q = q_w + q_x i + q_y j + q_z k \tag{1.14}$$

$$= q_w + q_v, \tag{1.15}$$

where q_w is referred to as the real or scalar part and $q_v = q_x i + q_y j + q_z k$ as the imaginary or vector part. Anyway, we mostly represent a quaternion Q as a 4-vector q. In this work, we will employ the so-called Hamiltonian notation, which uses the scalar part as first component, namely:

$$q_H \triangleq \begin{bmatrix} q_w \\ q_v \end{bmatrix} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} \tag{1.16}$$

The rotation matrix associated with the quaternion q can be written as:

$$R\{q\} = \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_w q_z) & 2(q_x q_z + q_w q_y) \\ 2(q_x q_y + q_w q_z) & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_w q_x) \\ 2(q_x q_z - q_w q_y) & 2(q_y q_z + q_w q_x) & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix} \tag{1.17}$$

or in a more compact form:

$$R\{q\} = (q_w^2 - q_v^\top q_v)I + 2q_v q_v^\top + 2q_w[q_v]_\times, \tag{1.18}$$

where I is the identity matrix and $[\cdot]_\times$ is the skew operator that produces the cross-product matrix:

$$[q_v]_\times \triangleq \begin{bmatrix} 0 & -q_z & q_y \\ q_z & 0 & -q_x \\ -q_y & q_x & 0 \end{bmatrix} \tag{1.19}$$

For describing rotations using quaternions we employ a notation similar to rotation matrices. Let $q_{\mathcal{IB}}$ represent a rotation from frame $\mathcal{B}$ to frame $\mathcal{I}$ analogous to $x_\mathcal{I} = R_{\mathcal{IB}} x_\mathcal{B}$. The equivalent operation is:

$$x_\mathcal{I} = q_{\mathcal{IB}} \otimes x_\mathcal{B} \otimes q_{\mathcal{IB}}^*, \tag{1.20}$$

where the quaternion product, also called Hamiltonian product, can be written as:

$$p \otimes q = \begin{bmatrix} p_w q_w - p_v^\top q_v \\ p_w q_v + q_w p_v + p_v \times q_v \end{bmatrix}, \tag{1.21}$$

and the conjugate as:

$$q^* \triangleq q_w - q_v = \begin{bmatrix} q_w \\ -q_v \end{bmatrix}. \tag{1.22}$$

Finally, for what concerns the kinematic equation in quaternion form:

$$\dot{q} = \frac{1}{2} q \otimes \begin{bmatrix} 0 \\ \omega \end{bmatrix}, \tag{1.23}$$

The same expression can be found written also in this form:

$$\dot{q} = \frac{1}{2}\Omega(\omega)q, \tag{1.24}$$

where Ω is equal to:

$$\Omega(\omega) = \begin{bmatrix} 0 & -\omega^\top \\ \omega & -[\omega]_\times \end{bmatrix} \tag{1.25}$$

It can be useful to write the equation (1.23) into discrete time for digital implementations:

$$q_{k+1} = q_k \otimes q\{\omega_k \Delta t\}, \tag{1.26}$$

where $q\{\omega_k \Delta t\}$ is the quaternion associated to the rotation $\omega_k \Delta t$, and it can be computed as an exponential of a pure quaternion v, *i.e.*, a quaternion with zero scalar part:

$$e^v = e^{u\theta} = cos\theta + usin\theta, \tag{1.27}$$

which is an extension of the Euler form for imaginary numbers. Applying (1.27) to (1.26), 23 obtain:

$$q\{\omega_k \Delta t\} = \begin{bmatrix} \cos(||\omega_k|| \Delta t/2) \\ \frac{\omega_k}{||\omega_k||} \sin(||\omega_k|| \Delta t/2) \end{bmatrix}. \tag{1.28}$$

1.2 Navigation

To date, most semi-autonomous or autonomous UAVs rely on the Global Navigation Satellite System (GNSS) to navigate outdoors. However, GNSS is not available indoors and it can be unreliable in urban settings. At the same time, most of research on autonomous drones in indoor environments exploits external motion-capture systems, *e.g.*, Vicon [40] or Optitrack [41]. Despite being useful for research purposes [42, 43], *e.g.*, for evaluating control laws or state estimation systems, they are not appropriate for other applications. Thus, aerial vehicles need to recover information about their motion through the use of exteroceptive sensors in the framework of Simultaneous Localization and Mapping (SLAM).

In the following, we first introduce some of the most common exteroceptive sensors, then we provide some background on SLAM and its main paradigms. Finally, we focus on visual odometry and visual SLAM, with some last considerations about visual inertial odometry and map representations.

1.2.1 Exteroceptive sensors

Among the exteroceptive sensors, laser range finders/scanners, RGB-D and vision sensors have been extensively employed for solving the navigation problem on UAVs [5].

Laser range finders/scanners

Laser range finders are used for computing the distance between the robot and an object belonging to the environment. We distinguish between laser range finders, *i.e.*, sensor that can measure only one target at a time, with laser scanners, *i.e.*, sensors that can scan in all directions in the environment. Laser scanners can be further classified into 2D or 3D scanners, depending on their capability of measuring distance of objects outside their plane. Note that 3D laser scanners are also named LiDARs (light detection and ranging).

In the latest year, on aerial vehicles, 2D scanners have been frequently employed, being lighter and less power consuming compared to 3D laser scanners. In [6] the authors employed a laser scanner and 2D SLAM for autonomous navigation. Similar works were carried out in [7] and [11]. Nowadays, the production of relatively cheap and light 3D scanners (Velodyne [44], Ouster [45]) allows their use on UAVs. In [18] the authors employed a 3D scanner for solving the SLAM problem, while preferring cameras for odometry computation.

Compared to other exteroceptive sensors, laser range finders and laser scanners have the advantage of providing useful information even in completely texture-less environments.

RGB-D sensors

RGB-D sensors have been frequently used for UAVs. They share many similarities with stereo cameras, being able to output features depth data. However, they reconstruct depth by illuminating a scene with a structured-light pattern. As a result, they have the advantage of estimating depth, even in areas with poor visual textures, but with the drawback of being range-limited by their projectors.

RGB-D sensors for state estimation and mapping with UAVs have been used in [46] as well as in [47], where autonomous exploration and mapping strategies are discussed.

Vision sensors

Vision sensors are lightweight, less power consuming and able to provide additional information compared to laser range finders. However, cameras result very sensitive to illumination changes and texture-less environments. In the literature, cameras have been employed in many different ways for aiding the navigation of autonomous robots. The early works focused on the use of biologically inspired algorithms, such as optical flow, allowing drones to perform very basic maneuvers, *e.g.*, takeoff, landing and reactive obstacle avoidance [48, 49, 50, 51, 52]. However, optical flow can only measure the relative velocity of image features. As a result, the position estimate of the UAV will drift greatly over time. The drift can be limited using visual odometry or visual SLAM methods. These methods were applied both using monocular cameras [14, 53, 54, 55] or stereo cameras configurations [7, 13, 56].

1.2.2 SLAM

Autonomous robots can navigate in an unknown environment through SLAM [57, 58], which is the process by which a mobile robot can build a map of an environment and, at the same time, use this map to deduce its location. In particular, both the trajectory of the platform and the location of all landmarks are estimated online without the need for any *a priori* knowledge of their location [57]. Due to the inherent noise in both sensor measurements and robot motion, SLAM problem is usually described in

probabilistic form.

As in the problem formulation of [57, 59], consider a mobile robot moving through an environment and taking measurements of a number of unknown landmarks using some kind of sensor located on the robot, as represented in Figure 1.2. We define:

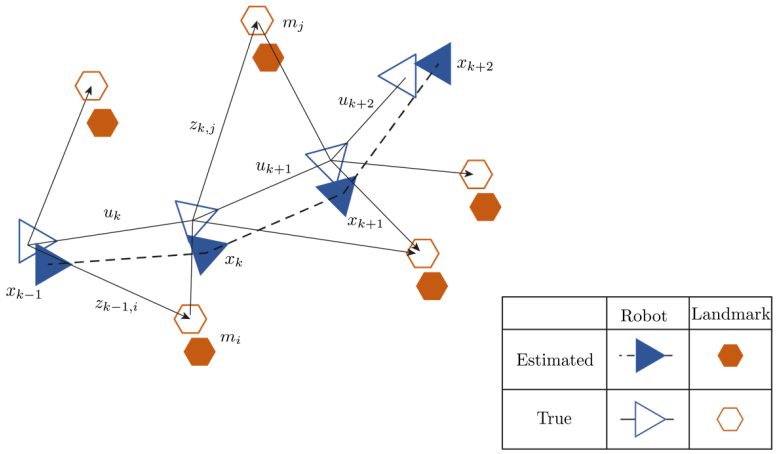


Figure 1.2: *SLAM problem representation (reworked from [57])*

- $x_{0:T} = \{x_0, x_1, x_2, \ldots, x_T\}$: the set containing the history of robot states from initial time to time T. The state vector at time instant k, x_k describes the pose (position and orientation) of the robot.

- $z_{1:T} = \{z_1, z_2, z_3, \ldots, z_T\}$: the set of all sensor measurements, taken from the robot, of the location of the landmarks. When the measurement is specified as $z_{k,i}$, it represents the observation taken from the vehicle of the location of the ith landmark at time k.

- $u_{1:T} = \{u_1, u_2, u_3, \ldots, u_T\}$: the set of all controls, *e.g.*, motion commands. The control at time k, u_k, represents the control vector, applied at time $k-1$ to drive the robot to a state x_k.

- $m = \{m_1, m_2, \ldots, m_n\}$: the set of all landmarks. m_i is a vector describing the location of the i-th landmark, whose true location,

in the simplest formulation of the problem, is assumed to be time invariant.

The quantities to estimate are: the map and the robot's pose in the environment. In probabilistic form, the problem can be formulated as the one of finding, at each time k, the probability distribution:

$$p(x_k, m|z_{1:k}, u_{1:k}, x_0). \tag{1.29}$$

This probability distribution describes the joint posterior density of the landmark position and robot's pose, at time k, given the recorded observations and control inputs up to and including time k together with the initial state x_0, which represent the initialization of the pose, used to fix the reference frame.

This formulation is also known as Online SLAM problem. If we are interested in the full robot's state evolution, the problem to be solved is the Full SLAM one, namely finding the probability distribution:

$$p(x_{0:T}, m|z_{1:T}, u_{1:T}). \tag{1.30}$$

Note that the Online SLAM means marginalizing out the previous poses:

$$p(x_k, m|z_{1:k}, u_{1:k}) = \int_{x_0} \cdots \int_{x_{k-1}} p(x_{0:k}, m|z_{1:k}, u_{1:k}) dx_{k-1} \dots dx_0 \tag{1.31}$$

Two families of algorithms exist to solve the SLAM problem:

1. Filtering-based SLAM or Bayesian SLAM: model the problem as an online estimation where the state of the system consists in the current robot position and the map. The estimate is augmented and refined by incorporating the new measurements as they become available. Due to their incremental nature, they typically solve the Online SLAM problem.

2. Keyframe-based SLAM or Graph-based SLAM: typically fall in the category of smoothing approaches. They are used to estimate the full trajectory of the robot from the full set of measurements. These

methods address the Full SLAM problem and they typically rely on least square minimization techniques.

In practical terms, control inputs u_k are usually replaced by odometry measurements to initial estimate of the robot motion between time steps. Odometry consists in computing the robot pose incrementally, composing transformations relating poses at discrete consecutive timestamps. In ground vehicles, wheel-encoders are employed for this kind of task. However, in contrast with ground vehicles, air vehicles are unable to measure odometry directly. It is true that, in principle, odometry could be obtained by double-integrating accelerations, but lightweight Micro Electro-Mechanical Systems (MEMS) Inertial Measurement Units (IMUs) are subjected to time-varying biases that would result in very high drift rates. As a consequence, exteroceptive sensors (presented in the previous Section) should be also employed for odometry computation on aerial vehicles [7, 5].

Filtering-based SLAM

In filtering-based SLAM, what we really want to achieve is a recursive solution for the problem. Thus, we start with the estimate for the distribution at time $k-1$:

$$p(x_{k-1}, m | z_{1:k-1}, u_{1:k-1}), \tag{1.32}$$

we compute the joint posterior (using Bayes theorem) following a control u_k and an observation z_k. In order to do this, we require a state transition model and an observation model for describing the effects of the inputs and observations respectively.

The observation model describes the probability of making an observation z_k when the robot and landmarks locations are known:

$$p(z_k | x_k, m). \tag{1.33}$$

At the same time, the state transition model, also called motion model, is described by the probability distribution:

$$p(x_k | x_{k-1}, u_k), \tag{1.34}$$

The SLAM algorithm can now be implemented in a standard two-step recursive prediction-correction form. The time update and measurement update laws are respectively provided in

$$p(x_k, m|z_{1:k-1}, u_{1:k}, x_0) = \int p(x_k|x_{k-1}, u_k) \times p(x_{k-1}, m|z_{1:k-1}, u_{1:k-1}, x_0)dx_{k-1}, \quad (1.35)$$

and

$$p(x_k, m|z_{1:k}, u_{1:k}, x_0) = \frac{p(z_k|x_k, m)p(x_k, m|z_{1:k-1}, u_{1:k}, x_0)}{p(z_k|z_{1:k-1}, u_{1:k})}. \quad (1.36)$$

At this point, the *map-building* and *localization* problems can be formulated as a degenerate case of the SLAM problem. In the map-building problem, we want to compute the probability distribution:

$$p(m|x_{0:k}, z_{1:k}, u_{1:k}), \quad (1.37)$$

assuming the robot's pose known at all times, subject to the knowledge of the initial state. The map m is, then, constructed fusing all the observations, obtained by the robot in different poses. On the other hand, the localization problem can be formulated as the computation of the probability distribution:

$$p(x_k|z_{1:k}, u_{1:k}, m), \quad (1.38)$$

where the landmark locations are known and the aim is the one of computing an estimate of the robot pose with respect to these landmarks.

To solve the SLAM problem, an appropriate representation for both the motion and the observation model, should be found. These models should be selected to allow a fast and accurate computation of the prior and posterior distribution in (1.35) and (1.36). One common representation is represented by state-space models with additive Gaussian noise, leading to the use of Extended Kalman Filter (EKF) to solve the SLAM problem. Alternatively, the vehicle motion could be represented as a set of samples of a more general non-Gaussian probability distribution, leading to the use of the so-called FastSLAM algorithm (Rao-Blackwellized particle filter) [60, 61].

Graph-based SLAM

A convenient way to describe the full SLAM problem (1.30) is through the so-called Dynamic Bayes networks (DBN) available in Figure 1.3.

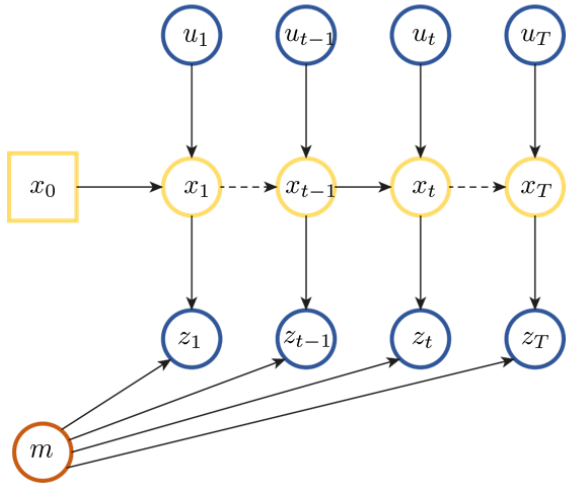


Figure 1.3: *Dynamic Bayes Network of the SLAM problem (reworked from [62])*

A DBN is a graphical model used to represent a stochastic process via a directed acyclic graph. Each node is a random variable and each edge represents a conditional dependence between variables. The motion model $p(x_k|x_{k-1}, u_k)$ is represented by the edges leading to x_k, while the measurement model $p(z_k|x_k, m)$ is represented by the arrows entering in z_k. Since DBN contains all the dependencies among variables (not represented connection imply independence), the joint probability of trajectory, map, controls and measurements can be written as the product of all conditionals [63]:

$$p(x_{0:T}, m, u_{1:T}, z_{1:T}) = p(x_0) \prod_k (x_k|x_{k-1}, u_k) \prod_i (z_i|x_{k_i}, m) \quad (1.39)$$

To obtain an optimal estimate for the set of unknowns given all available measurements and controls, the problem is converted into an equivalent

least squares formulation based on a maximum a posteriori (MAP) estimate:

$$\{x^*_{0:T}, m^*\} = \underset{x_{0:T},m}{\operatorname{argmax}}\, p(x_0) \prod_k (x_k|x_{k-1}, u_k) \prod_i (z_i|x_{k_i}, m) \qquad (1.40)$$

Usually, non-linear models with additive Gaussian noise are used for both measurement and motion models. The optimization problem is often solved with nonlinear optimization methods, *e.g.*, Gauss–Newton or Levenberg–Marquardt algorithms.

Note that, especially in a graph-based SLAM formulation, the system is usually thought as composed by two different components: the *front end* and the *back end*. The front end abstracts sensor data into models to be used for estimation, while the back end performs inference on the abstracted data produced by the front end [64]. Consequently, while the back end is, by definition, sensor agnostic, the front end is heavily dependent on the employed sensor. Note that each of the aforementioned exteroceptive sensors represent a potential candidate for the front end.

In particular, in graph-based SLAM [62]:

- The front end is represented by the construction of the graph from measurements and odometry (graph construction).
- The back end is represented by the determination of the most likely configuration of the poses given the edges of the graph (graph optimization).

Note that keyframe-based approaches have recently become the *de facto* standard formulation for the SLAM problem [64].

1.2.3 Visual odometry and visual SLAM

Regarding the odometry, when it is computed through the use of camera is called Visual odometry (VO). More formally, VO is the process of estimating the egomotion of an agent by incrementally estimating its pose through examination of the changes that the motion induces on the images of a single or multiple cameras attached to it [65, 66].

The VO pipeline can be summarized in the following steps:

1. For every new image (or image pair in the case of a stereo camera), the first step consist of detecting and extracting a set of salient features, *e.g.*, points, corners, lines.

2. The extracted 2D features are matched with those found in previous frames. The 2D features that are the reprojection of the same 3D feature in different images are called *image correspondence*.

3. The relative motion between the two different time instants is computed. The camera pose is computed by the concatenation of the retrieved transformation with the previous pose.

4. An iterative refinement, called *windowed bundle adjustment*, can be applied over the last n frames to obtain a more accurate estimate of the local trajectory.

If a stereo camera configuration, *i.e.*, two rigidly attached cameras having overlapping field of view, is used; the 3D reconstruction of the observed image features will be available. On the other hand, by using a single monocular camera, with associated benefits in terms of power and payload weight requirements, the result will be known up to an unknown scale (for mathematical details see Appendix B and Appendix C)

Beside the previously presented VO pipeline, which we can denominate *feature-based*, appearance-based methods and hybrid methods exist. In appearance-based methods, also called *direct methods* [67], the structure and motion is directly estimated from intensity values in the image. An example of hybrid method is available in [53].

As previously highlighted, the basic working mechanism of VO involves the computation of the camera path incrementally, *i.e.*, pose after pose. The errors will thus accumulate over time leading to drift of the estimated trajectory with respect to the real one. In this framework, the refinement step, performed through reprojection error minimization, can help in keeping drift as small as possible.

On the other hand, SLAM goal is to obtain a global and consistent estimate of the robot path. A map of the environment (even if not needed for other applications) is built with the aim of realizing if the robot returns to an already visited area. This process is called *loop closing*. When a

loop closure is detected, the drift in both the map and the camera can be reduced.

It is also worth pointing out that the overall objective of the two methods is different: in VO, we only care about local consistency of the trajectory and a local map (through windowed bundle adjustment) is used to obtain a more accurate estimate of it, whereas SLAM is concerned about global map consistency [65]. Note also that VO is often used as a building block for a complete SLAM algorithm to recover the incremental motion of the robot; however, to make a complete SLAM methods, we must also add some loop closing detection and global optimization step to obtain a metrically consistent map [65].

Going in details, in feature-based visual SLAM (vSLAM), in contrast with visual odometry, the bundle adjustment would involve finding the full maximum likelihood solution to the graph at each time step, and not only relative to the last n poses. Considering the number of sparse features available in each frame, the computational cost will explode quickly. To address this issue, two ways of summarizing the information gained over time have been formulated: *filtering approach* and *keyframe-based approach*. These two approaches are similar to the one presented for what concerns the general SLAM theory.

In the filtering approach (see Figure 1.4), the poses and measurements are summarized by joint probability distribution in state-space. All the poses other than the current one are marginalized out after every frame; and features, which may be observed again in the feature, are retained. The resulting graph will not grow arbitrarily with time, and will not grow during movement in the same area, adding persistent features only when new areas are explored. The downside is that the graph will become fully inter-connected since the elimination of a past pose causes new links among pairs of observed features to which it was joined [68].

In the keyframe-based approach (see Figure 1.5), the graph is solved from scratch time after time, but it is sparsified by retaining only a subset of past poses, *i.e.*, some heuristically chosen *keyframes*. The other poses, with all the measurements connected to them, are not marginalized out as in the filter, but simply discarded and they will not contribute to the estimate. The resulting graph will be larger compared to the one obtained

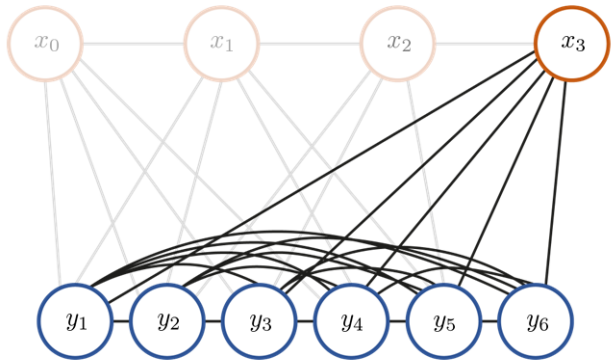


Figure 1.4: *Dynamic Bayes Network for filtering-based approaches in vSLAM (reworked from [68])*

using the filter-based approach, but the lack of marginalization will lead to a sparsely interconnected graph [68].

As an example of representative filtering-based method we can refer to MonoSLAM [69], a monocular camera vSLAM algorithm. In MonoSLAM, camera motion and structure are estimated simultaneously using an EKF. Its computational costs increases based on the size of the environment: the size of the state vector becomes larger and larger while increasing the number of feature points. On the other hand, belonging to the family of keyframe-based SLAM, PTAM (Parallel Tracking and Mapping) [70] is able to run in real time (at around $30\,\mathrm{Hz}$) by parallelizing the motion estimation and mapping tasks and by relying on efficient keyframe-based bundle adjustment (BA) [68]. However, PTAM was designed for augmented reality applications in small desktop scenes, and multiple modifications (*e.g.*, limiting the number of keyframes) were necessary to allow operation in large-scale outdoor environments [55]. A very popular keyframe-based technique, which achieved real-time SLAM in large-scale environment using local BA, fully automated relocation and loop closing, is ORB-SLAM [71]. Originally it was a monocular vSLAM, but it has been extended to stereo and RGB-D vSLAM [72].

Among the leading direct-methods (for monocular cameras), we cite LSD-SLAM [73], which is an extension of the semi-dense VO method

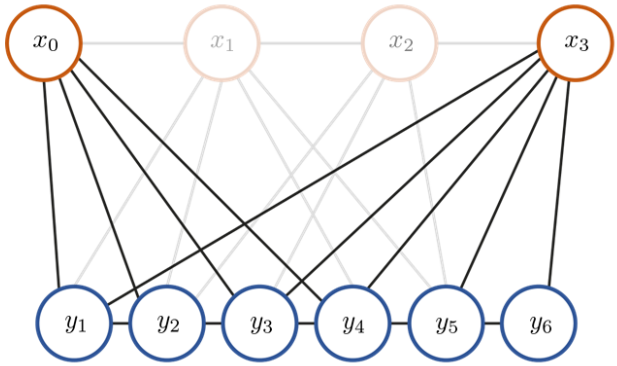


Figure 1.5: *Dynamic Bayes Network for keyframe-based approaches in vSLAM (reworked from [68])*

described in [74]. In particular, LSD-SLAM has seen adding loop-closure detection and pose-graph optimization with respect to the corresponding VO algorithm.

1.2.4 Visual inertial odometry

VO has been often combined with IMU leading to the so-called Visual Inertial Odometry (VIO). This fusion process has been extensively used for monocular VO for absolute scale structure and motion estimation [55].

Two paradigms of VIO exist: *loosely coupled* and *tightly coupled* [75]. Conceptually, loosely coupled methods process visual and inertial measurements separately by computing two independent motion estimates that are fused to get the final output. By contrast, tightly coupled methods compute the final output directly from the raw camera and IMU measurements, *e.g.*, the tracked 2D features, angular velocities, and linear accelerations. Tightly coupled approaches result more accurate that the loosely coupled ones for these two reasons [76]:

1. Using IMU integration to predict 2D feature locations in the next frame facilitate feature tracking.

2. Loosely-coupled approaches do not consider visual and inertial coupling, making them unable to correct drift in the vision-only estima-

tor.

1.2.5 Maps

The maps employed in SLAM (and produced by it) can be parameterized mainly by [62]:

1. set of spatially located landmarks;
2. dense representations, *e.g.*, occupancy grids, surface maps etc.;
3. raw sensor measurements.

The choice of the map parameterization depends on the sensor used, on the characteristics of the environment and on the estimation algorithm employed. For example, landmark maps are preferred in environments with easily identifiable features and when cameras are used. On the other hand, dense representation are often used in combination with range sensors.

Dense representations are also preferred for being exploited by guidance algorithms. In this framework, occupancy grids are commonly-used for planning collision-free paths in 2D environments [77]. Extending occupancy grids to 3D leads to huge memory requirements, especially when large spaces are considered [78]. Nowadays, Octomap [79] is the most-used map representation for online 3D map building and planning in unstructured environments [80]. Octomap is an open-source framework to generate volumetric 3D environment models using probabilistic occupancy estimation, and based on *octrees*. An *octree* is a hierarchical data structure for spatial subdivision in 3D. Each node in an octree represents the space contained in a cubic volume, usually called a *voxel*. This volume is recursively subdivided into eight sub-volumes until a given minimum voxel size is reached. The minimum voxel size determines the resolution of the octree. Since an octree is a hierarchical data structure, the tree can be cut at any level to obtain a coarser subdivision if the inner nodes are maintained accordingly. A graphical representation is shown in Figure 1.6.

In its most basic form, octrees can be used to model a Boolean property. In the context of robotic mapping, this is usually the occupancy of a volume. Using Boolean occupancy states or discrete labels allows for

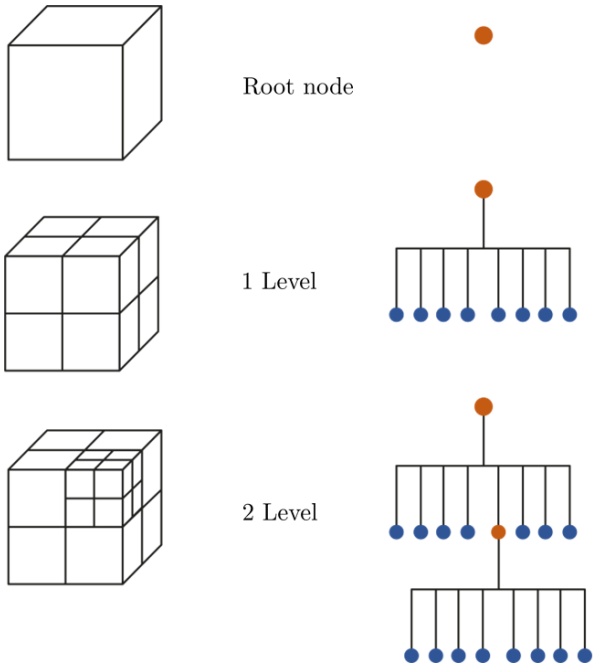


Figure 1.6: *Graphical representation of an Octree*

compact representations of the octree: if all children of a node have the same state (occupied or free) they can be pruned. This leads to a substantial reduction in the number of nodes that need to be maintained in the tree. The flexible voxel size allows representing large areas in a fast and memory efficient way. In robotic systems, one typically has to cope with sensor noise and temporarily or permanently changing environments. In such cases, a discrete occupancy label will not be sufficient. Instead, occupancy has to be determined according to a probabilistic model, meaning that multiple observations of the scene are merged together to assign the occupancy probability. This aspect results particularly important in the case of noisy sensor data, such as stereo or RGB-D cameras. Octomaps have been used for motion planning on many UAVs, *e.g.*, [17, 81].

Other 3D environment models exist, *e.g.*, point clouds, elevation maps and multi-level surface maps. Point clouds can be composed by very large amount of measurement points and are not memory efficient. Point clouds represent only areas occupied by obstacle and do not provide any means for fusing multiple measurements. Elevation maps consist of a two-dimensional grid in which each cell stores the height of the territory [82]. Whereas this approach leads to a substantial reduction of the memory requirements, it can be problematic when a UAV has to utilize these maps for navigation and guidance. Finally, multi-level surface (MLS) maps can be regarded as an extension to elevation maps. MLS maps represent 3D structure by a grid where each grid cell contains a list of surface patches. In the simplest formulation, the patch is represented by the mean and variance of the measured height at the position of the grid cell. Additionally, a depth value can be stored in the patch. This depth value reflects the fact that a surface patch can be on top of a vertical object. In these cases the depth is defined as the difference of the height of the surface patch and the height of the lowest measurement belonging to the vertical object [82]. Flat object will have depth equal to zero.

1.3 Guidance

Addressing the guidance for autonomous UAVs means solving the problem of trajectory planning of a differentially-constrained vehicle through

an environment with obstacles. This vehicle motion planning problem can be considered a special case of the general motion planning problem [83, 84].

In order to give a general problem formulation, some definitions and terminology must be introduced. The *configuration* of a robot/vehicle is a vector of n coordinates:

$$q \in C \subseteq \mathbb{R}^n, \tag{1.41}$$

where C is the C-space. The C-space can be an arbitrary manifold. In this case, we will focus on configurations expressed through a minimum set of coordinates. The C-space can be regarded as the union:

$$C = C_{free} \cup C_{obs}, \tag{1.42}$$

where C_{free} is the set of configurations in which the robot is not in collision with an obstacle and vice-versa for C_{obs}.

The state of the robot is described by a vector x that can be $x = q$, if the robot controls are the velocities, or the configurations and the velocities $(q, v) \in X$, if the control inputs are accelerations or forces. The equations of motion are:

$$\dot{x} = f(x, u), \quad u \in U, \tag{1.43}$$

which integral form can be written as:

$$x(T) = x(0) + \int_0^{\top} f(x(t), u(t))dt. \tag{1.44}$$

With these definitions, a fairly general statement of the motion planning problem is the following:

Problem. *Given an initial state $x(0) = x_{start}$ and a desired final state x_{goal}, find a time T and a set of controls $u : [0, T] \rightarrow U$ such that the motion satisfies $x(T) = x_{goal}$ and $q(t) \in C_{free}$ for all $t \in [0, T]$.*

The problem is very difficult [85], especially as the number of degrees of freedom increases. There does not exist an algorithm that provides an exact analytic solution to such a problem. Indeed, even state-of-the-art

approximation algorithms operating on a three-dimensional subspace of this problem space are difficult to compute in real time [86].

Many variations to the motion planning problem exist. First of all, we should distinguish from computing a *path*, *i.e.*, a curve traced by the vehicle in the configuration space, and a *trajectory*, *i.e.*, the state evolution of the vehicle over time. The environment can be either static (time-invariant motion planning problem) or dynamic (time-variant motion planning problem), meaning that the obstacles move over time. The problem can be *differentially-constrained* (or kinodynamic constrained), *i.e.*, a problem in which the vehicle's equation of motion acts as a constraint on the path. The opposite is true for differentially-unconstrained problems. Other problem categorization can then arise from the vehicle model employed.

A motion planning algorithm can have different properties. Planners can be *multiple query* or *simple query*. A multiple query motion planner is one that invests time in developing a good representation of the C-space in such a way that future motion planning problem in that space can be solved quickly. If the environment changes frequently a single query motion planner can be used to find quickly a solution. A planner is considered to be *complete* if and only if it finds a path when one exists. It is considered *optimal* when it returns the optimal path with respect to some criterion [1]. A weaker notion is *resolution completeness/optimality*, in which the planner finds always a solution if it exists at the level of discretization of the problem. At the same time, *probabilistic completeness/optimality* means that the probability of finding a solution, if one exists, goes to 100% as the time approaches infinity.

When considering UAVs, in the most general formulation, the configuration vector is defined by three position and three orientation coordinates. Its state can comprehend also velocities and angular velocities, for a total of 12 variables. The differential constraints arise in the form of kinematic (nonholonomic constraints) and dynamics (involving second-order or higher differential constraints). In many practical applications, a point vehicle representation is employed. This assumption greatly simplifies the problem, leading to a configuration space equal to the 3D Euclidean

[1] Note that any optimal planner is also complete.

space. Then, in order to avoid collision, the obstacles are usually inflated by the vehicle radius.

Many of the algorithms designed to solve the motion planning problem for UAV rely on a decomposition approach, first solving a path planning problem and, then, generating a trajectory that conforms to the path [86]. This latter step is often called *smoothing*. Other possible approaches are respectively trajectory optimization methods and method based on motion primitives [15].

Starting from analyzing the path planning problem, we can distinguish between search-based and sampling-based methods.

1.3.1 Search-based methods

Even if the C-space of a robot is continuous, it is usually discretized. Some free configurations are sampled from the free space and lines connecting configurations are drawn, in such a way that configurations can reach each other by a straight-line path. The resulting *graph* is now our discretized representation of the free space, where configurations are represented by nodes and paths between two configurations are represented by edges.

A popular way to discretize the space is to create a grid. A C-space grid is obtained dividing each of the n dimensions into k intervals, creating k^n grid cells, with the graph node at the center of the cell. Each cell is represented in the graph by a single node, representing the configuration at the center of the cell.

To search the graph, several algorithms can be used. One of the most popular is the A* search [87]. In addition to the graph, A* requires a function that computes an optimistic cost-to-go, namely a lower bound on the actual cost to go from a start node to a goal node. The only requirements are that this heuristic must be fast to evaluate and close to the actual cost-to-go. In the case of violation of the latter requirement, A* may terminate with a solution that is not optimal. The algorithm which preceded historically the A* is the Dijkstra's algorithm [88], which doesn't use a heuristic to guide the search, resulting in a slower algorithm. Obviously, due to the discretization, these path planners are not complete, but resolution complete. A major drawback of this approach is that it is not practical for

high-dimensional spaces.

Many variants of search-based methods can be find in the literature. A non exhaustive list comprehends Jump Point Search (JPS) [89], Weighted A* [90], Anytime A* [91], Anytime Repairing A* (ARA*) [92], D* [93], and many others. On UAVs we can find A* implemented in the path planning pipeline in [94, 95, 96, 97, 98] among others.

1.3.2 Sampling-based methods

Nowadays, sampling-based algorithms are very popular because of their simplicity and their performance. Among them, Probabilistic Roadmaps (PRMs) and Randomly Exploring Random Trees (RRTs) [99] are the most employed families of approaches.

Probabilistic Roadmap (PRM)

When a graph satisfies certain topological properties [100], and it results to be a *roadmap*, the property of completeness can be ensured. However, a roadmap is very computationally expensive. A Probabilistic Roadmap is a type of approximate roadmap, constructed from a set of configurations randomly sampled from the C-space. As the number of samples tends to infinity, the likelihood that the graph is a true roadmap goes to 100%. An advantage of a PRM graph over a grid-based graph is that the structure of the free C-space is generally captured by the PRM with many fewer nodes than with a grid graph.

Also in RRT, the idea of connecting points sampled randomly from the state space is exploited. However, RRT can be thought as a single-query PRM. The incremental nature of RRT avoids the necessity to set the number of samples *a priori*, and returns a solution as soon as the set of paths built by the algorithm is rich enough, enabling on-line implementations.

Even if both the algorithms are probabilistically complete, they are not asymptotically optimal, hence they do not tend to the optimal solution as the number of nodes goes to infinity. In [101], a modification of the basic algorithms, *i.e.*, RRT* and PRM* have been proposed in such a way to achieve asymptotic optimality. For example, RRT*, continually rewires the search tree so that the solution tends to the optimal solution as the number of nodes in the tree goes to infinity. In the UAV literature, we can

find sampling-based methods in the planning pipeline in [102, 103, 104, 105] among others.

1.3.3 Path smoothing and trajectory generation

After having generated a path, most of the current state-of-the-art methods exploit the differential flatness of the quadrotors (see Section **??**) for generating a dynamically feasible smooth trajectory [106, 22, 107]. This step is usually carried out by solving an optimization problem over a class of trajectories like minimum snap [108, 104].

Other approaches, instead of using a geometric path as prior to create a dynamic feasible trajectory, explore the space of trajectories using a set of short-duration motion primitives generated by solving an optimal control problem [16, 109]. The motion primitives compose a lattice discretization in the state space, which can be explored using search-based algorithms [110, 111] (see Section 1.3.1). Motion primitives are also used to generate many trajectory candidates, which will be evaluated based on an objective function [112, 113, 114].

However, the presented methods checks the obstacle constraints *a posteriori*. This means that these methods are either limited to short trajectories or they have to do a computationally extensive search to be able to generate a trajectory around obstacles [98].

Other approaches include the obstacles directly while solving the nonlinear optimization problem derived from the motion planning problem. An option consists in penalizing the distance to obstacles in the cost function [15, 115]. However, this leads to non-convex optimization problems and requires computationally expensive distance field representations. In other works, instead, the shape of obstacles is encoded in the constraints using successive convexification [116, 117] or a convex decomposition of the environment [118, 119, 120].

1.3.4 Reactive approaches

An additional category of methods for guidance of mobile robots is the one of reactive approaches. These methods are part of the larger class of local planners (also called collision avoidance planners). Indeed, local planners can be either map-based algorithms (as the ones presented in

the previous Sections), which compute feasible and locally-optimal paths through local maps built from sensor data or *a priori* known global maps [115], or sensor-based reactive approaches.

Map-based methods are usually applied for known environments and have limited ability to handle uncertainties. On the other hand, reactive approaches [121] do not require any early information about the environment and plan directly in the current sensor data.

Among the classical reactive approaches employed in mobile robotics we can mention: Dynamic Window Approach (DWA) [122], Vector Field Histogram (VFH) [123], Bayesian approach [124], curvature-velocity method [125] and potential field techniques [126].

On UAVs several works have been published in recent years. Many authors have employed vision for reactive obstacle avoidance in the form of optical flow [127, 128, 129, 130], images coming from monocular cameras [131] or stereo cameras depth estimation [132]. Other articles focus on the use of methods like artificial potential fields [94, 133], VFH [134] and imitation learning approaches [135].

Frequently reactive approaches are used in a layered fashion with global planners [94, 133]. In particular, the global planner is used to plan, at low frequency, an optimal collision free path/trajectory in the known environment (up to that moment), while the low-latency high-frequency reactive approach is used to avoid potential collision with unknown obstacles arising during the execution of the path/trajectory.

1.3.5 Exploration

The final category of guidance algorithms analyzed is the one related to the exploration problem. In this framework, the goal is not only to plan a collision free trajectory, but to maximize the amount of information acquired about the environment.

This problem can come in many different versions [59]:

1. a robot may seek to acquire a map of a static environment. The exploration problem is the problem of planning paths such that, when the robot has followed them, all the environment (or its unknown space) becomes known.

2. a robot might have the task to find a person in a known environment, as part of the *pursuit evasion problem*.

3. a robot seeks to determine its own pose during localization. This problem is commonly called *active localization*, and the goal is to maximize the information about the robot's own pose, *e.g.*, when a manipulator equipped with a sensor faces an unknown object.

In this work, we focus only on the first version of the problem and we split it in two separate sub-problems:

1. Select the next pose to be reached by the robot.

2. Plan a trajectory from the actual pose to the selected pose.

For the first sub-problem mainly two approaches are available in the literature, based either on a binary gain or an expected information gain. When using a binary gain, the cells of the map which have been visited are updated as "explored", while all the other cells are marked as "unexplored". This gain is at the core of a popular exploration algorithm called *frontier-based exploration* [136], which simply tries to move the robot to the nearest accessible unvisited frontiers. On the other hand, *information-based exploration algorithms*, *i.e.*, methods based on a non-binary expected information gain, evaluate different candidate positions based on the achievable potential information gain. One popular strategy, the GB-L strategy [137] evaluates a candidate observation position p using the following function:

$$f(p) = A(p) \cdot exp(-\lambda \cdot L(p)), \tag{1.45}$$

where $A(p)$ is an estimate of the unexplored area visible from p, $L(p)$ is the length of the path connecting the current robot position and p, and λ weights the new information obtainable from a position and the cost of traveling to reach that position. $A(p)$ can be computed as the difference between the area of a circle with radius r centered in p and the area of its intersection with the known segments composing the path from the current position of the robot to p. Different potential information gains have been proposed and evaluate in [138].

On UAVs examples of *frontier-based exploration* are available in [134, 47], while the most popular *information-based exploration* method is represented by the Receding Horizon "Next-Best-View" Planner (RH-NBV) [105]. Attempts to combine the two approaches have been made in the literature. In particular, the authors of [139] proposed Autonomous Exploration Planning (AEP). They used RH-NBV as local exploration strategy and a frontier exploration method as global one. Finally, the authors of [140] proposed an online exploration algorithm inspired on RRT* and on a new information gain and compared it against the RH-NBV planner and AEP.

The second sub-problem, instead, has been faced in the literature with the previously presented trajectory planning methods. The only difference is related to the fact that the planner must be able to cope with unknown environments. In the UAV literature, some authors uses optimistic planners, *i.e.*, planners which consider unknown space as free [95, 141], while others used an optimistic global planner in combination with a conservative local planner [15, 115].

1.4 Control

The control of conventional multirotors has been an extensively studied topic. Although inherently nonlinear and under-actuated, linear controllers, like PID and LQ, have been very successful for multicopters flight control [142]. To handle nonlinearity more efficiently, nonlinear controllers have been proposed using feedback linearization [143], backstepping [144], geometric control [145, 146] and many others.

In recent years, two control frameworks have become particularly popular for tracking trajectories, even at very high speed: differential-flatness-based control and nonlinear model predictive control.

Differential flatness is a system property that extends the notion of controllability from linear systems to nonlinear dynamical systems [147]. Multirotors have been proved to be differentially flat systems, and this property has been frequently used for generating dynamically feasible trajectories (see Section 1.3.3). Differential flatness based control approach have been originally proposed for UAVs in [148] and implemented as po-

sition control of a real robot in [149]. In [23] differential-flatness-based controllers have shown to be capable of tracking very aggressive trajectories.

Model predictive control (MPC) has been employed for tackling both the trajectory generation problem (see Section 1.3.3) and the trajectory tracking problem at the same time. This technique has been successively applied to quadrotors in [150, 151]. Recently, nonlinear model predictive controllers have been employed for agile trajectory tracking [152, 153].

The two classes of control methods have been compared in [154], which showed also the importance of substituting the angular controller from a PID to an incremental nonlinear dynamic inversion (INDI)-based controller [155], a control framework already widely used, especially for what concerns fault-tolerant control of multirotors [156, 157], for tackling model uncertainties and external disturbances.

For a complete review of the multirotor control methods see [158, 147]. A review of MPC control applied to quadrotors is, instead, available in [159].

For what concerns this work, we are going to present a basic mathematical model for a quadrotor and introduce the concept of differential flatness. Then, the quaternion-based cascaded P-PID controller, used throughout this work, is presented. Its choice was motivated by the fact that no stringent requirements on trajectory tracking performance was imposed.

1.4.1 Model

We consider the dynamical model of a rigid quadrotor, neglecting aerodynamic terms like drag and moments due to blade flapping. Both effects have been presented in pioneering studies [160, 161, 162]. With these assumptions, we can write the dynamics of the position r, velocity v, orientation $R_{\mathcal{IB}}$, and body rates $\omega_{\mathcal{B}} = [\omega_x, \omega_y, \omega_z]$ as:

$$\dot{r} = v \tag{1.46}$$
$$\dot{v} = -gi_3 + cb_3 \tag{1.47}$$
$$\dot{R}_{\mathcal{IB}} = R_{\mathcal{IB}}[\omega_{\mathcal{B}}]_\times \tag{1.48}$$
$$\dot{\omega_{\mathcal{B}}} = J^{-1}(\tau - \omega_{\mathcal{B}} \times J\omega_{\mathcal{B}}) \tag{1.49}$$

where c is the mass-normalized collective thrust, J is the quadrotor's inertia matrix, $g = 9.81\,\mathrm{m/s^2}$ is the gravity acceleration and τ is the three dimensional torque input. Given this model, we can show that the quadrotor is a differentially flat system, result that has been frequently used for trajectory generation (see Section 1.3.3). In other words, it is possible to prove that the state and the inputs of the system can be written as algebraic functions of four selected outputs and a finite number of their derivatives. In turn, the flat outputs are the three components of the UAV position $r = [r_x, r_y, r_z]$ and its heading ψ. The proof can be found in [163], but the authors include in the model formulation also rotor drag. The proof for the simplified model employed in this work is available in [106], but some errors in the derivation must be corrected following [163] itself (see Appendix D).

1.4.2 Cascaded controller

The controller presented in this Section is the one employed in most autopilots for multirotor UAVs and, in particular, on the autopilot firmware PX4 [35], used throughout this work.

Given the underactuated nature of conventional multirotors, position tracking is obtained by changing the UAV attitude in order to tilt the resultant of the thrust forces.

The controller architecture is composed by two loops: the position loop is the outer one, while the attitude loop is the inner one. The proportional part of the outer P-PID (see Figure 1.7) is fed with the difference between the reference position r_0 and the actual estimated one $\hat{r}$ and its output is the desired velocity in the inertial frame v_0.

By computing the error on the velocities, the inertial force computed by the position controller is given by:

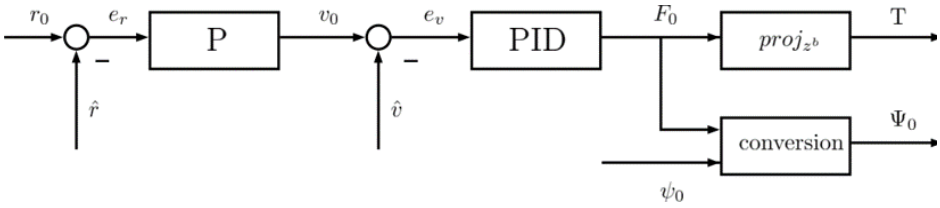


Figure 1.7: *Block diagram of outer (position control) loop (reworked from [35])*

$$F_0 = PI_v(K_{P,r}(r_0 - \hat{r}) - \hat{v}) - D_v\hat{v} + mgi_3, \tag{1.50}$$

where

$$PI_v(\cdot) \triangleq K_{P,v} + K_{I,v}\frac{1}{s} \tag{1.51}$$

$$D_v(\cdot) \triangleq K_{D,v}\frac{s}{1 + s/N_d} \tag{1.52}$$

are transfer function defining the proportional-integral and filtered derivative actions. $K_{P,(\cdot)}$, $K_{I,(\cdot)}$, $K_{D,(\cdot)}$, are the proportional, integral and derivative gains, respectively, and N_D is the first order filter constant for the derivative action. Then, given the desired yaw angle ψ_0, a geometric law is used for finding the desired total thrust T and the orientation of the body frame Ψ_0 (D.1)-(D.9). The desired orientation Ψ_0 is then parameterized in terms of quaternions and together with the actual angular velocities is used to feed the inner P-PID controller, whose output are the desired moments.

The proportional part of the inner loop P-PID (see Figure 1.8) computes the desired angular velocity ω_0 through:

$$\omega_0 = K_{P,\Psi}\text{sgn}(q_{e_w})q_{e_v}, \tag{1.53}$$

where q_{e_w} and q_{e_v} are respectively the scalar and vector part of the quaternion error computed as:

$$q_e = \hat{q}^* \otimes q_0, \tag{1.54}$$

with q_0 the desired quaternion obtained from the desired orientation Ψ_0,

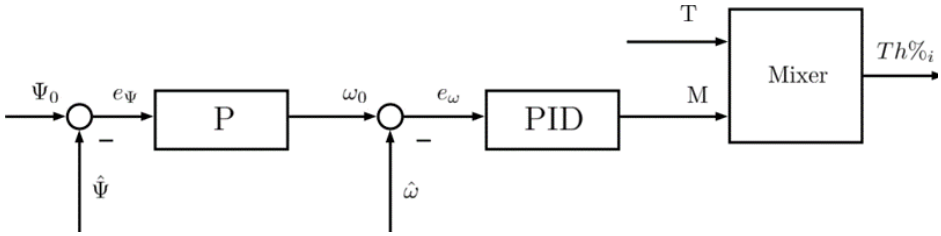


Figure 1.8: *Block diagram of inner (attitude control) loop (reworked from [35])*

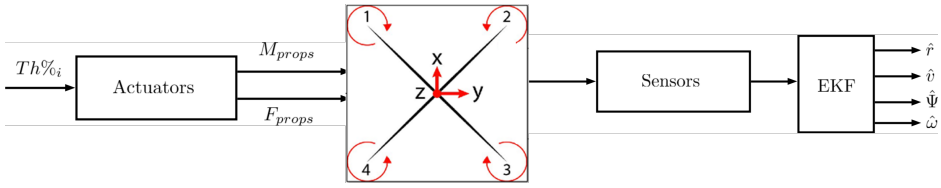


Figure 1.9: *Mixer block diagram*

and with q the estimated quaternion, obtained from the estimated orientation $\hat{\Psi}$.

The torques are then computed through:

$$\tau = PI_\omega(\omega_0 - \hat{\omega}) - D_\omega\hat{\omega} \tag{1.55}$$

with similar definitions to the ones given in equations (1.51) and (1.52). The input thrust to each motor are then computed from the desired torques and thrust using a static map, called *mixer matrix* (see Figure 1.9), which in turns depend on the geometric and propulsive properties of the UAV.

Finally, the rotor thrusts are delivered to the platform and the state, estimated using sensor measurements through the EKF2 (see Appendix A), are fed back to assure the tracking of the position reference.

CHAPTER 2

Simulation and experimental environment

In this Chapter, the simulation environment, employed for testing the software components developed throughout this work, is presented. Then, the arenas, in which the experimental tests are carried out, are shown. The last part of this Chapter is then dedicated to the presentation of all the aerial platforms used for the experiments.

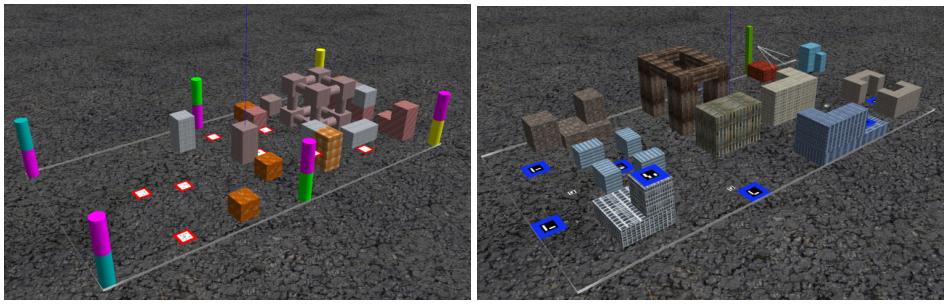
Note that, the simulation and the aerial platforms are very similar in terms of software architecture. In particular, note that while the PX4 autopilot firmware is used for low-level control and state estimation, the Robot Operating System (ROS) [164] has been employed as framework for all the high-level software running on the UAV and for the simulations.

2.1 Simulation environment

Most of the algorithms developed in this thesis have been tested and validate in simulations, before deploying them on the aerial vehicles. The PX4 flight stack, besides working on dedicated boards, offers the possibility of *Software In The Loop* (SITL) simulations on one or multiple PCs. Among the available simulators, Gazebo has been selected. Gazebo [165] is a ROS-integrated open source robotic simulation package. Among the many advantages of this simulator, we can cite a robust physics engine, high-quality graphics, open-source code and convenient customer and graphical interfaces [166]. Gazebo has also been used for simulating multicopters in various works, *e.g.*, [167, 168]. One of the drawbacks of Gazebo, in contrast with simulators with rendering engines built on Unity [169], is that it is not photo-realistic. However, our simulations have been specifically conceived for testing the interactions among different software components, *e.g.*, the interaction between the flight stack with custom developed ROS nodes, with no interest in evaluating the vision pipeline performance.

With the same goal in mind, the simulated drone, even if it does not resemble the real platforms in terms of inertia and aerodynamic properties, is equipped with all the sensors that are available on the real experimental platform. In particular, monocular cameras, stereo cameras and range finders plugins have been used. In the same way, the 3D Gazebo *world* tries to reproduce the real experimental environment. For instance, two Gazebo worlds, which try to emulate the LDC arena (see Section 2.2.2), are shown in Figure 2.1.

A schematic representation of the working mechanism of the simulation environment is shown in Figure 2.2. The PX4 SITL communicates over UDP with external APIs (ROS in our case) via MAVLink protocol (for details see [170]) as if a real drone was connected. Note that the *mavros* [171] ROS driver for MAVLink communication is employed to handle this kind of communication. The PX4 SITL to external API link is mainly used for providing position and velocity set-points to the virtual drone. At the same time, the PX4 SITL communicates via MAVLink also with QGroundControl [172], a Ground Control Station (GCS) software.



(a) *LDC2020* **(b)** *LDC2021*

Figure 2.1: *Simulation environment for reproducing experimental conditions of LDC*

This allows to use QGroundControl for controlling the simulated drone, to design and execute missions, change controller and estimator parameters, and to test the status of the simulated drone in real-time. Then, a direct link is established between the PX4 SITL and the simulator itself. This link is used to receive sensor data from the simulated world and to send the computed actuator commands to it. Finally, there is also a communication link between the Gazebo simulator and the ROS environment. This communication link is used for sending images and point clouds, in the form of ROS topics, from the simulated stereo and monocular cameras to the navigation and guidance stack.

2.2 Experimental environment

2.2.1 FlyART

The Flying Arena for Rotorcraft Technologies (FlyART) of Politecnico di Milano (shown in Figure 2.3) is a $12\,\mathrm{m}\times6\,\mathrm{m}\times4\,\mathrm{m}$ indoor facility equipped with an Optitrack [41] Motion Capture system (Mocap). The system is composed by 12 infra-red cameras which detect markers, sensitive to the infrared light, mounted on top of the UAVs. A dedicated ground control computer, running Windows 10, is used for controlling the motion capture system and providing the measured UAV poses over a Wi-fi connection. This information could be either fused on the UAV estimator, replacing

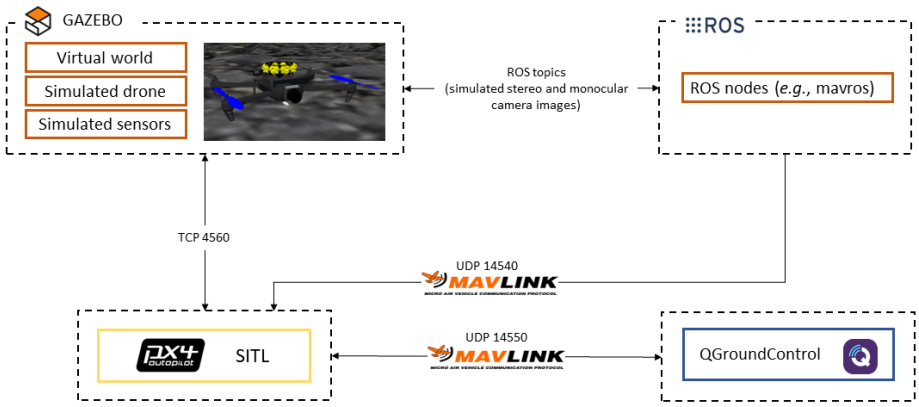


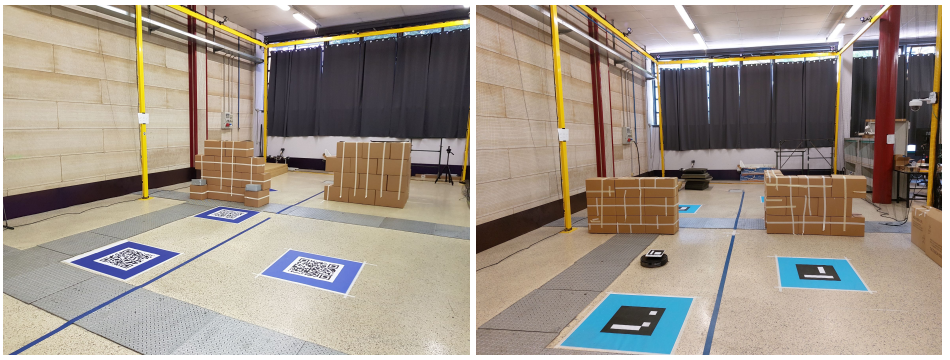
Figure 2.2: *Simulator schematic representation*

de facto GPS measurements in the indoor environment of the laboratory, or could be simply used as ground truth for accessing performance of GPS-denied navigation.

Figure 2.3: *FlyART experimental facility*

A second ground control computer, running Ubuntu 18.04 operating system and ROS Melodic, receives telemetry information from the drones via Wi-fi. In some cases, it is also used for sending commands or tasks to the aerial platforms.

The FlyART has been opportunely customized depending on the considered experimental activity. Consider, for example, Figure 2.4, in which some cardboard obstacles have been placed in the arena for accessing planning and collision avoidance capabilities and mapping accuracy.



(a) *FlyART for LDC2020* **(b)** *FlyART for LDC2021*

Figure 2.4: *Customized version of FlyART for reproducing experimental conditions of LDC*

In addition to this, 4 to 5 visual markers (QR or ArUCo) have been placed for experiments concerning localization and/or vision-based landing.

2.2.2 Leonardo Drone Contest arena

The Leonardo Drone Contest competitions are held in an indoor arena. The competition field, shown in Figure 2.5, tries to emulate an urban environment. Its dimensions are $20\,\mathrm{m} \times 10\,\mathrm{m} \times 3\,\mathrm{m}$. The boundaries of the field are delimited by a net.

The obstacles placed in the arena consist of cardboard objects with glued textures on them. The obstacles are, at most, three meters high with passages of, at least, one meter among them. Note that no motion capture system is available in the arena. Depending on the competition edition, the arena sees the presence of different kinds of visual markers (QR or ArUco), ground robots and a fixed surveillance camera (see Chapter 4).

Figure 2.5: *The Leonardo Drone Contest arena (from [173])*

2.2.3 Aerial vehicles

In the following, we show all the aerial platforms employed throughout this work. All the vehicles share the same low-level software architecture, depicted in Figure 2.6.

The software architecture shares many similarities with the simulator, except that in this case the processes are executed on three different boards. The ground control station is a laptop with the Ubuntu 18.04 distribution installed. It runs ROS Melodic and the GCS software QGroundControl. The ground control station sends commands (in the form of tasks or directly set-points) and receives high-level information from the aerial platforms, which in turn are connected to the ROS network, over Wi-fi.

The aerial platforms are equipped with two processing boards, the flight control unit and the companion computer. The former runs the low-level control and state estimation laws, while the latter, equipped with a Wi-fi module, is appointed to communicate with the ground control station and to execute all the developed navigation and guidance algorithms. The two boards communicate through MAVLink protocol using *mavros*. In particular, the companion computer sends the position and/or velocity set-points to the FCU and receives the estimated state and sensor measurements. Finally, the FCU establish a direct connection with QGroundControl, through which we can access the flight control related parameters and receive telemetry data.

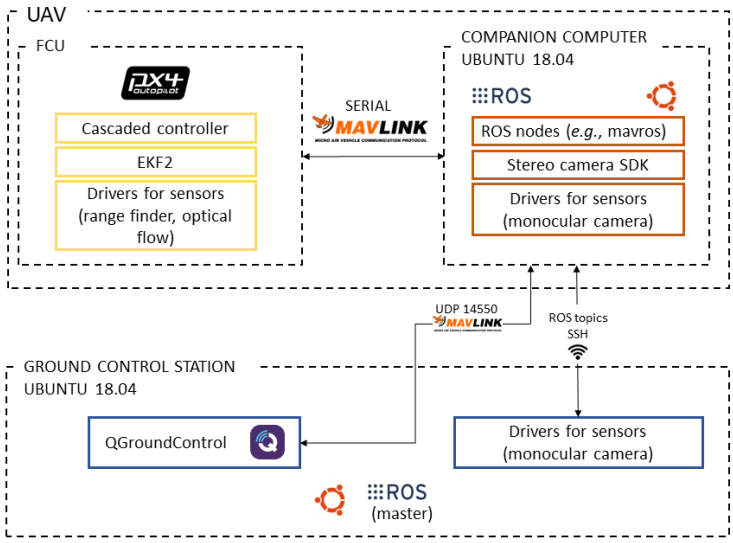


Figure 2.6: *Software architecture shared by all aerial platforms*

ROG-1

ROG-1 (see Figure 2.7) is an octocopter with coaxial propellers. It was specifically designed for the LDC first competition (see Section 4.2) and developed in collaboration with the Politecnico di Milano spin-off company ANT-X [174]. Its main features are reported in Table 2.1.

Table 2.1: *ROG-1 characteristics*

Feature	ROG-1
Take Off Weight (TOW)	3.75 kg
Size	$500 \times 500 \times 300$ mm
Motors	8 x KDE2315XF-965
Propellers	8 x carbon-nylon two-bladed 9443
Battery	4S LiPo 16 000 mAh (weight 1.35 kg)
Flight time (hovering)	17 min

The drone is equipped with a Pixhawk 4 Flight Control Unit (FCU) and an NVIDIA Jetson TX2 companion computer mounted on a Connect Tech Orbitty Carrier board. The main navigation camera is represented by a

Figure 2.7: *ROG-1*

Stereolabs ZED stereo camera (see Figure 2.8). The stereo camera, whose main features are reported in Table 2.2, is inclined slightly downward, *i.e.*, 15 degrees with respect to the horizontal plane.

Figure 2.8: *ZED stereo camera (from [175])*

Furthermore, the drone is equipped with a downward-looking monocular camera OpenMV H7 plus [176], which characteristics are shown in Table 2.3.

Finally, a TeraRanger Tower Evo (see Figure 2.9) is mounted in the

Table 2.2: *ZED and ZED2i stereo camera characteristics*

Feature	ZED	ZED2i
Weight	170 g	166 g
Size	175 × 30 × 33 mm	175 × 30 × 43 mm
FOV horizontal	90 deg	110 deg
FOV vertical	60 deg	70 deg
Power	5 V/380 mA	5 V/380 mA
Baseline	120 mm	120 mm

Table 2.3: *OpenMV H7 plus camera characteristics*

Feature	OpenMV
Weight	17 g
Size	75 × 36 × 29 mm
FOV horizontal	70.8 deg
FOV vertical	50.6 deg
Power	3.3 V/240 mA
Processor	ARM® 32-bit Cortex®-M7 CPU

upper part of the UAV. The Tower is equipped with 8 TeraRanger Evo 60 m range finders, oriented in different directions around the UAV, for perceiving potential obstacles.

Figure 2.9: *Terabee TeraRanger Tower EVO (from [177])*

ROG-2

ROG-2 (see Figure 2.10) is a platform born as evolution of ROG-1. The ROG-1 propulsion system has been upgraded, leading to higher endurance, less noise and lower weight. ROG-2 main characteristics are summarized in Table 2.4.

Figure 2.10: *ROG-2*

Table 2.4: *ROG-2 characteristics*

Feature	ROG-2
Take Off Weight (TOW)	$3.03\,\mathrm{kg}$
Size	$430 \times 300 \times 200\,\mathrm{mm}$
Motors	8 x T Motor F90 KV1300
Propellers	8 x polycarbonate two-bladed 7042
Battery	4S LiPo $16\,000\,\mathrm{mAh}$ (weight $1.35\,\mathrm{kg}$)
Flight time (hovering)	$20\,\mathrm{min}$

A minimum set of sensors are equipped for competing in the LDC2021 edition (see Section 4.3). In particular, ROG-1 and ROG-2 share the same FCU and main navigation stereo camera. On the other hand, ROG-2 does

not carry any range finder for collision avoidance, but a single laser range finder, a Garmin Lidar-Lite v3 (see Figure 2.11), points downward and it is used for altimeter purposes.

Figure 2.11: *Garmin Lidar-Lite v3*

An additional OpenMV H7 plus monocular camera is mounted pointing forward, inclined 15 degrees with respect to the vertical axis facing toward the horizontal plane. The companion computer of ROG-1 has been replaced with the Nvidia Jetson Xavier NX. This represents an upgrade in terms of computational power and memory, while, at the same time, it leads to a decrease in weight since no carrier board is needed.

ROG-3

ROG-3 (see Figure 2.12) is the last platform for testing autonomous guidance and navigation algorithms, designed in the context of the third edition of LDC (see Section 4.4). It carries the same propulsion system of ROG-2 and the same boards (FCU and companion). For completeness, ROG-3 main characteristics are shown in Table 2.5.

Table 2.5: *ROG-3 characteristics*

Feature	ROG-3
Take Off Weight (TOW)	$3.16\,\mathrm{kg}$
Size	$430 \times 300 \times 200\,\mathrm{mm}$
Motors	8 x T Motor F90 KV1300
Propellers	8 x polycarbonate two-bladed 7042
Battery	4S LiPo $16\,000\,\mathrm{mAh}$ (weight $1.35\,\mathrm{kg}$)
Flight time (hovering)	$20\,\mathrm{min}$

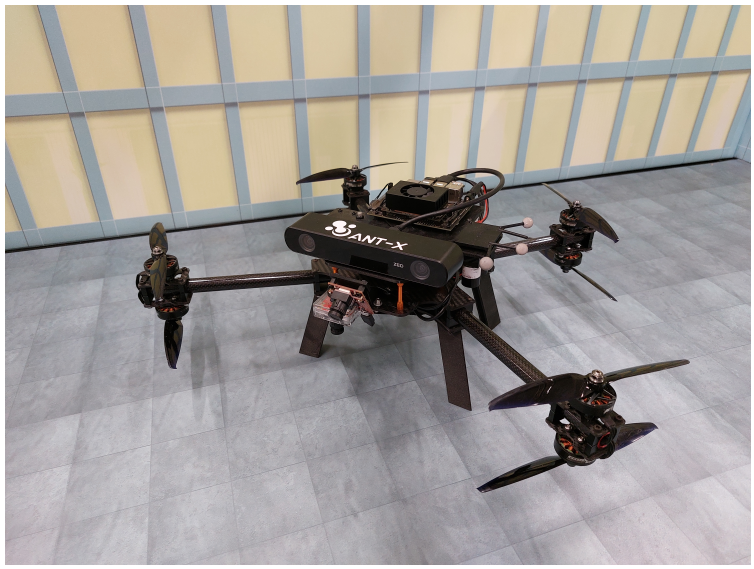


Figure 2.12: *ROG-3*

Minor changes have been made to sensor suite compared to the previous version of the platform. In particular, the ZED stereo camera has been replaced with the ZED2i. This stereo camera features are reported and compared with the ones of ZED in Table 2.2. Thanks to its wider field of view, the ZED2i is mounted with no inclination with respect to the horizontal plane. Two monocular OpenMV cameras are still equipped, even if in different locations. The same range finder is also used. One last change regards the equipment of an additional small low-resolution monocular camera for optical flow computation: the PMW3901 [178].

ANT-X

The small quadcopter, codename ANT-X, shown in Figure 2.14 has been employed in experimental activities as follower drone (see Section 5.2). It equips a Pixhwak Mini FCU and a small CPU, the NanoPi NEO Air (see Figure 2.13), which features are reported in Table 2.6, as companion computer.

The ANT-X main characteristics are summarized in Table 2.7.

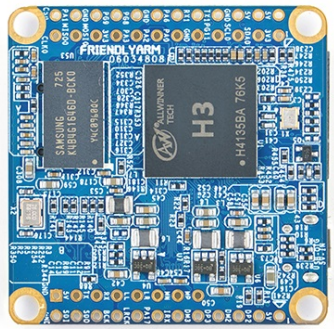


Figure 2.13: *NanoPi NEO Air (from [179])*

Table 2.6: *NanoPi NEO Air features*

Name CPU	Quad-core Cortex-A7 1.2 GHz
RAM	512 MB
Wireless	2.4 GHz 802.11 $b/g/n$
Dimensions	40 x 40 mm
Weight	7.9 g
Power	5 V - 2 A

Table 2.7: *ANT-X characteristics*

Feature	ANT-X
Take Off Weight (TOW)	0.3 kg
Size	$200 \times 200 \times 40$ mm
Motors	4
Propellers	4 x three-bladed 3-inch
Battery	3S LiPo 950 mAh (weight 1.1 kg)
Flight time (hovering)	6.5 min

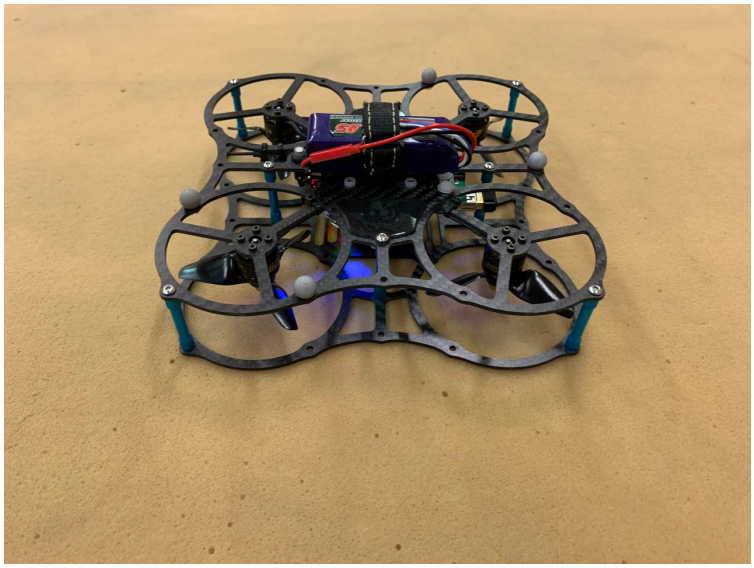


Figure 2.14: *ANT-X*

CARRIER-1

The octocopter, codename CARRIER-1, shown in Figure 2.15 has been used in experimental activities as landing target (see Section 5.2). It has a Pixhwak Mini FCU and the NanoPi NEO Air as companion computer. The CARRIER-1 main features are summarized in Table 2.8.

Table 2.8: *CARRIER-1 characteristics*

Feature	CARRIER-1
Take Off Weight (TOW)	$2.57\,\mathrm{kg}$
Size	$580 \times 580 \times 140\,\mathrm{mm}$
Motors	8 x T-motor F40 PRO II 1600KV0
Propellers	8 x two-bladed 6535
Battery	6S LiPo $8000\,\mathrm{mAh}$ (weight $1.1\,\mathrm{kg}$)
Flight time (hovering)	$16\,\mathrm{min}$

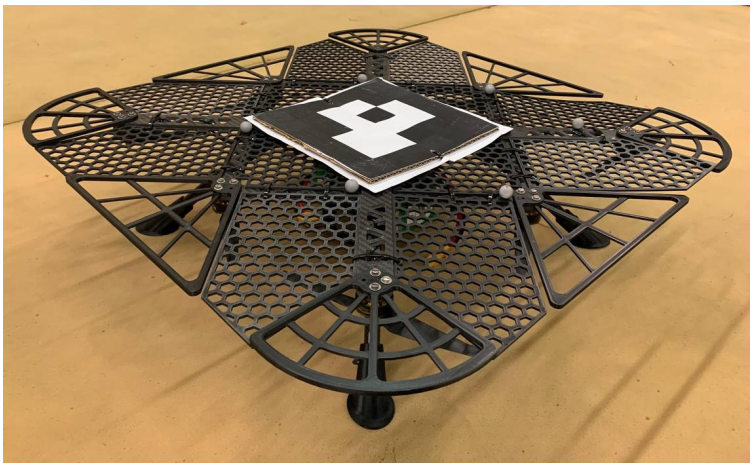


Figure 2.15: *CARRIER-1*

CHAPTER 3

Visual odometry error modeling

This Chapter aim is to define and validate a modeling approach for visual odometry errors. We propose a time-domain-analysis technique, called Allan Variance (AVAR), to model the types and magnitude of various noise terms affecting the stereo VO position measurements. The AVAR is a mathematical tool introduced by Allan in 1966 to study the error in frequency of atomic clocks [180], and it is widely used for characterizing Inertial Measurement Units (IMU) [181]. Through suitable computations, a state-space representation of the errors, which can be used in sensor-fusion framework, is obtained. The retrieved model is then employed for validation through a Kalman predictor. This analysis has been conducted on an experimental dataset collected using a multicopter UAV equipped with stereo vision. The runs of the experiment have been designed following the Design of Experiments (DoE) technique [182]. In this framework, we will also evaluate and discuss changes in the model

parameters due to environmental and flight conditions.

3.1 Motivation

Many studies have been conducted on visual odometry for aerial vehicles navigation. Consider, for instance [7], which was one of the first papers on stereo VO-based UAV navigation. In the same year, an application of EKF-based vSLAM was proposed [183]. Successively, in [14, 55], the authors employed a monocular camera and an IMU for state estimation and point-to-point steering of an UAV in an outdoor, GPS-denied environment. These works have employed monocular vSLAM algorithms based on PTAM. Similar works have been carried out also in [9, 10]. In the following years, many commercial products, *e.g.*, the Intel Realsense D435 and Intel Realsense T265, made the deployment of VO/vSLAM on UAVs easy and accessible.

However, despite the effort to improve the performance of visual odometry systems, both in terms of computational speed and accuracy, published work on this topic has paid little attention to the issue of integrating stereo visual odometry estimates into autonomous navigation systems. As a matter of fact, the availability of a description of the noise dynamics, which for other systems, such as Inertial Navigation System (INS) and GNSS, are already well known, would be of great benefit in a sensor-fusion-oriented framework.

Note that the characterization of vision systems led to additional difficulties with respect to inertial ones. In particular, while inertial sensors measure directly the state of the robot (proprioceptive), stereo vision allows to retrieve information about an agent's movement by inspecting the surrounding environment (exteroceptive). Thus, the environment and the conditions in which the tests are executed play an important role in the determination of the system's performance.

In order to assess the performance of multiple algorithms or sensors, visual odometry requires a framework for systematic and comparative study. For this purpose, this Chapter aims at offering a methodological approach for the quantification and modeling of visual odometry errors.

3.2 Related works

When considering the accuracy of stereo vision systems, we should consider a number of factors [184], namely:

- Pixel quantization, dependent on intrinsic characteristics of the cameras (focal length and resolution)
- Accuracy of the intrinsic cameras' parameters determined during calibration.
- Accuracy of the 2D segmentation process used to detect corresponding objects in the left and right camera. This algorithm provides the input for the triangulation process for disparity estimation (see also Appendix C)

To these factors, the errors of the processes involved in the odometry computation should be added. With particular reference to feature-based methods, errors can arise in:

- The feature detection and matching process over consecutive frames.
- The optimization processes typical of the refinement stage, *i.e.*, outliers rejection and possible motion and structure optimization (bundle adjustment).

In the literature, some results regarding the above factors are available. In [185] the authors evaluated the effect of pixel quantization. The inaccuracies caused by the calibration procedure have been tackled in [186]. The two error sources are simultaneously studied in [187]. In [188] the authors proposed a study on the uncertainty of stereo triangulation. In particular, the errors are analyzed when varying the length of the baseline and the extrinsic parameters uncertainty. In [189] the impact of errors of the 2D segmentation algorithm used for calibration board detection is analyzed. In [190] the depth resolution of a stereoscopic system is studied. The estimation of random and systematic errors at various depths is carried out in [184].

Passing to visual odometry accuracy, in [191] the sensitivity to the average distance of landmarks is investigated. In [192] the authors model

and analyze in statistical sense the long-range drift of stereo vision. In [193] the bias of stereo visual odometry is investigated. In particular, a bias correcting technique, even if not sufficiently fast for online use, has been proposed. Finally, in [194] the authors investigated noise models for different features detection and matching processes in stereo visual odometry.

All of these results are useful to provide insights on the nature and magnitude of errors in stereo pose reconstruction, however, they are very difficult to be used in practice.

3.3 Background

This section is meant for providing some background on the classical approach based on Allan Variance analysis and Kalman prediction employed in the modelling of IMU errors. This approach will then be generalized to deal with visual odometry problems in Section 3.4.

3.3.1 Allan Variance

Consider a time series of n values y_i, $i = 1, 2, \ldots, n$, representing the time history of the stochastic error of interest. This vector has to be split into M clusters of consecutive points of equal length τ ($\tau < n/2$), chosen as an integer multiple of the sampling interval τ_0, such that $\tau = m\tau_0$.

For each cluster, computing the average over the points contained in the cluster itself, leads to the definition of $\bar{y}_i$. The Allan Variance (AVAR) is defined as the variance of a random variable ξ, equal to the difference between the average of adjacent clusters:

$$\xi_{i+1,i} = \bar{y}_{i+1} - \bar{y}_i, \tag{3.1}$$

computed over all clusters of the same size that can be formed from the data. The AVAR expression results into:

$$\sigma^2(\tau) = \frac{1}{2(n-2m)} \sum_{i=1}^{n-2m} [\bar{y}_{i+1} - \bar{y}_i]^2. \tag{3.2}$$

The AVAR has become a standard in IMU noise analysis [195] thanks to its capability to provide an approximation of the dynamics of the noise in terms both of Power Spectral Density (PSD) and, under some restrictions, of state space representation. This noise characterization method is based on the relation between the AVAR representation of a random process and its PSD, denoted as $S(f)$. These two quantities are linked by the integral transformation

$$\sigma^2(\tau) = 4 \int_0^\infty S(f) \frac{\sin^4(\pi f \tau)}{(\pi f \tau)^2} df. \tag{3.3}$$

This transformation has no inverse formula and, thus, the AVAR does not determine a unique noise spectrum.

As a consequence, a series of fundamental noises, each one having its own PSD and associated AVAR, must be chosen. The objective is to graphically obtain an Allan Standard Deviation (ADEV), *i.e.*, the square root of the AVAR, which resembles, as close as possible, the one retrieved from the data. It is worth mentioning that this graphical comparison is carried out in a log-log plot with cluster time τ on the horizontal axis.

The PSD of each fundamental noise can be associated with the output of a stochastic model, with frequency response function $H(j\omega)$, driven by white-noise. Exploiting the relation:

$$S_o(\omega) = |H(j\omega)|^2 S_i(\omega), \tag{3.4}$$

where $S_o(\omega)$ is the output PSD and $S_i(\omega)$ the PSD of the white noise driving function, it is possible to retrieve the spectral factor $H(s)$. Finally, if $H(s)$ is a finite order polynomial function, a state space representation can be computed.

Most noise processes of interest have a PSD described by the power law $S(f) = \beta f^{-\alpha}$, where f is the frequency, α is the power law coefficient and β is a fixed value related to the noise coefficient. Despite being available in the literature [181], [196] some relevant noise terms are here introduced and discussed for completeness and consistency of notation.

White frequency noise

The white frequency noise is characterized by constant PSD $Q_N = N^2$. In the rate gyro analysis framework it often takes the name of Angular Random Walk (ARW). For short, the same nomenclature will be maintained in this work. The corresponding ADEV is given by:

$$\sigma(\tau) = \frac{N}{\sqrt{\tau}}. \tag{3.5}$$

The ADEV plot in log-log scale is a straight line with slope -1/2 and the PSD coefficient N can be obtained by looking at the value for $\tau = 1$.

The state representation of the ARW is a single white Gaussian noise ω_N forcing term in the output equation. Thus, it can be reduced to:

$$z(t) = z_N(t) = \omega_N(t), \tag{3.6}$$

where $z_N(t)$ is the contribution of the ARW to the total noise $z(t)$.

Random walk frequency noise

The random frequency noise is defined by a power law PSD with $\alpha = 2$, namely:

$$S_K(f) = \frac{K^2}{(2\pi f)^2}, \tag{3.7}$$

where K is the random-walk coefficient. This kind of noise is often called Rate Random Walk (RRW) in rate gyros characterization and the same name will be adopted also for this work. By substituting $S_K(f)$ into (3.2), one obtains:

$$\sigma(\tau) = \frac{K}{\sqrt{3}}\sqrt{\tau}. \tag{3.8}$$

Therefore the log-log ADEV plot of a pure RRW is a straight line with slope 1/2. The coefficient K can be read on the ADEV plot, even if other noises are present, by first finding a range of values of τ over which the plot resembles a straight line with slope 1/2, and then extrapolating such line until it reaches $\tau = 3$.

Finally, being the spectral factor $H(s) = 1/s$, the state representation of a pure RRW is:

$$\dot{z}_K(t) = \omega_K(t) \tag{3.9}$$
$$z(t) = z_K(t) \tag{3.10}$$

where $z_K(t)$ is the contribution of the RRW to the total noise $z(t)$ and $\omega_K(t)$ is a driving white Gaussian noise with PSD $Q_K = K^2$.

Bias instability

The origin of this noise is the electronics or other components that are susceptible to random flickering. The bias instability PSD is defined as follows (from [196]):

$$S_B(f) = \begin{cases} \frac{B^2}{2\pi}\frac{1}{f} & \text{for } f < f_0 \\ 0 & \text{for } f > f_0 \end{cases} \tag{3.11}$$

where B is the bias instability coefficient and f_0 the cut-off frequency. The ADEV results:

$$\sigma(\tau) = \frac{2B^2}{\pi} \tag{3.12}$$
$$\times \left[\ln 2 - \frac{\sin^3 x}{2x^2}(\sin x + 4x\cos x) + C_i(2x) - C_i(4x)\right]^{1/2}$$

where $x = \pi f_0 \tau$ and C_i is the cosine-integral function:

$$C_i(x) = -\int_x^{+\infty} \frac{\cos t}{t} dt. \tag{3.13}$$

The log-log ADEV plot presents a straight line with slope +1 for $\tau < \frac{1}{f_0}$. For $\tau > \frac{1}{f_0}$, the ADEV can be approximated as:

$$\sigma(\tau) \simeq B\sqrt{\frac{2\log 2}{\pi}} = 0.664B, \tag{3.14}$$

which is a straight line with slope 0. The coefficient B can be estimated by dividing the value of the ADEV in the zero-slope region by 0.664. In IMU analysis, bias instability noise is used to represent the contribution given by the part of the ADEV with slope 0. However, bias instability comes with a downside: its PSD cannot be related directly to a rational spectral factor since it is not an even function of f.

As can be seen, the PSD for bias instability does not admit a rational spectral factor, which limits its applicability in a state estimation framework. A valid approximation for bias instability can be represented by the Gauss-Markov (GM) first order process, described in the following.

Gauss-Markov process

The Gauss-Markov process can be modelled as

$$\dot{z}_B(t) = -\frac{1}{T_B} z_B(t) + \omega_B(t) \tag{3.15}$$

$$z(t) = z_B(t), \tag{3.16}$$

where $z_B(t)$ is the approximation of the bias instability noise, T_B is the correlation time and $\omega_B(t)$ is a driving white Gaussian noise with PSD Q_B. The spectral factor is

$$H(s) = \frac{1}{s + 1/T_B}, \tag{3.17}$$

hence the PSD:

$$S_B(f) = \frac{Q_B}{(1/T_B)^2 + (2\pi f)^2}. \tag{3.18}$$

Substituting the expression of the PSD into (3.2), we obtain:

$$\sigma(\tau) = T_B \left[\frac{Q_B}{\tau} \left(1 - \frac{T_B}{2\tau} \left(3 - 4e^{-\frac{\tau}{T_B}} + e^{-\frac{2\tau}{T_B}} \right) \right) \right]^{1/2}. \tag{3.19}$$

The ADEV for a GM process can be split into three regions. The first one, for $\tau << T_B$, can be approximated as $\sigma(\tau) = \sqrt{Q_B \tau / 3}$, while the

last one, for $\tau >> T_B$, can be approximated as $\sigma(\tau) = T_B\sqrt{Q_B/\tau}$. A key aspect is that this process has always an ADEV plot characterized by two asymptotes with slope +1/2 and -1/2 respectively. In the middle region, for $\tau \simeq 1.89T_B$, the ADEV equation is simplified to $\sigma(\tau \simeq T_B) \simeq 0.4365\sqrt{Q_B T_B}$.

The GM process is a very convenient model for two reasons. First, it provides a rational model for bias instability (the flat portion). Second, as its ADEV a parabola-like concave shape, it can be used to reconstruct the ADEV of signals having regions characterized by a concave trend.

Drift Rate Ramp

The drift rate ramp and the flicker walk frequency modulated (FM) have different nature but present the same PSD, and therefore the same ADEV. The flicker walk FM, or flicker FM, is an ideal noise defined by its own PSD, while the drift rate ramp is often the result of deterministic errors. Here, to avoid confusion, the noise associated with this PSD will be called just drift rate ramp, or rate ramp. It is described by the power law with $\alpha = 3$:

$$S_R(f) = \frac{R^2}{(2\pi f)^3}. \tag{3.20}$$

Being $S_R(f)$ an odd function of f, the rate ramp, as the bias instability, is not related to any state space representation. The resulting ADEV is:

$$\sigma(\tau) = \frac{R\tau}{\sqrt{2}}. \tag{3.21}$$

Its ADEV plot is a straight line with slope +1, and the coefficient R can be found by looking at the value of the ADEV for $\tau = \sqrt{2}$.

3.3.2 Kalman predictor

In order to compare the performance of the error model with experimental data, a one-step-ahead Kalman predictor is often employed. Consider the following continuous-time state-space system:

$$\dot{x} = Ax + Bu + Gw \tag{3.22}$$
$$y = Cx + Du + Vv \tag{3.23}$$

with no deterministic input, *i.e.*, $u = 0$ and stochastic process noise w and measurement noise v. The processes w and v are independent. The corresponding discrete-time model with sampling time T_s can be written as:

$$x_{k+1} = A_d x_k + \tilde{w}_k \tag{3.24}$$
$$y_k = C_d x_k + \tilde{v}_k \tag{3.25}$$

where $C_d = C$, while the state matrix A_d is equal to:

$$A_d = e^{AT_s}. \tag{3.26}$$

$\tilde{w}_k \sim \mathcal{N}(0, Q_d)$ is a white Gaussian random variable with covariance Q_d, and $\tilde{v}_k \sim \mathcal{N}(0, R_d)$ is a white Gaussian random variable with covariance R_d.

Following [197], the discrete time process noise covariance matrix can be computed as:

$$Q_d = \int_0^{T_s} e^{A(T_s - s)} G Q G^\top e^{A^\top (T_s - s)} ds. \tag{3.27}$$

Note that for small T_s we have $Q_d \approx (GQG^\top)T_s$. The discrete time measurement noise covariance matrix R_d can be found from the PSD R as:

$$R_d \approx \frac{VRV^\top}{T_s}. \tag{3.28}$$

After having computed the asymptotic covariance matrix of the estimate P, solution of the discrete-time algebraic Riccati equation

$$A_d P A_d^\top + Q_d - A_d P C_d^\top (C_d P C_d^\top + R_d)^{-1} C_d P A_d = 0, \tag{3.29}$$

the steady-state Kalman gain is computed as

$$K = A_d P C_d^\top (C_d P C_d^\top + R_d)^{-1}. \tag{3.30}$$

Finally, the predicted state $\hat{x}$ and output $\hat{y}$ can be found through the following relations:

$$\hat{x}_{k+1} = A_d \hat{x}_k + K(y_k - C_d \hat{x}_k) \tag{3.31}$$
$$\hat{y}_{k+1} = C_d \hat{x}_{k+1}. \tag{3.32}$$

3.4 Proposed approach

In this Section, the approach proposed for visual odometry error analysis is presented along with the designed experimental campaign.

3.4.1 Statistical design of experiments

Statistical design of experiment refers to the process of planning the experiment so that appropriate data that can be analyzed by statistical methods will be collected, resulting in valid and objective conclusions [182]. There are two aspects to take into consideration in the experimental problem: the design of the experiment and the statistical analysis of its results.

First of all, in general, potential design factors must be identified and separated from nuisance variables when planning the experiment. In the former category we can insert the design factors, which are the variables that were chosen to be studied in the experiment; the held-constant factors, which may have an effect but are not of interest, and the allowed-to-vary-factors, *i.e.*, the ones that either the experimenter cannot control or something that can be balanced out by simply randomizing the experimental runs.

Once having identified the factors, other fundamental aspects regard the range over which they will be varied and the levels at which the experimental runs will be conducted.

If several factors are involved, the most efficient experiment is obtained with a factorial design, *i.e.*, an experimental design in which all possible combinations of the levels of the factors are investigated.

Usually two levels are selected, leading to a 2^k factorial design, where k represents the number of factors. This is often a good choice since can be easily extended to a 3^k design if an indication about the curvature of the process we are modeling is needed.

In factorial design, we are interested in main effects and interactions between factors. The main effect of a factor can be defined as the change in response caused by the change in the level of the factor. At the same time, an interaction is found if the response between the levels of one factor varies at all the levels of the other factors.

To conclude, compared to varying one factor at a time, the factorial design results fundamental for capturing interactions (if they exist) and it is more efficient in terms of number of experimental runs needed.

3.4.2 Allan variance for visual odometry

In order to model the experimental ADEV, two models have been considered and analyzed. The first is composed by GM, RRW and ARW, while the second also comprehends the drift rate ramp term.

For what concerns the first case, the AVAR results equal to:

$$\sigma^2(\tau) = \frac{N^2}{\tau} + \frac{K^2}{3}\tau + \\ + \frac{T_B^2 Q_B}{\tau}\left(1 - \frac{T_B}{2\tau}\left(3 - 4e^{-\frac{\tau}{T_B}} + e^{-\frac{2\tau}{T_B}}\right)\right). \tag{3.33}$$

While usually the four coefficients T_B, Q_B, N and K are determined by graphically comparing the experimental ADEV with the obtained one, in this case, for repeatability purposes, an optimization problem has been set. First of all, we write equation (3.33) using the parameter vector $\Theta = [\theta_1, \theta_2, \theta_3, \theta_4] = [N, K, Q_B, T_B]$, whose components will be constrained to be positive:

$$\sigma^2(\tau;\Theta) = \frac{\theta_1^2}{\tau} + \frac{\theta_2^2}{3}\tau + \\ + \frac{\theta_4^2\theta_3}{\tau}\left(1 - \frac{\theta_4}{2\tau}\left(3 - 4e^{-\frac{\tau}{\theta_4}} + e^{-\frac{2\tau}{\theta_4}}\right)\right). \tag{3.34}$$

For each cluster length τ_i for $i = 1, \ldots, L$, we can use the experimental value of the AVAR $\bar{\sigma}^2(\tau_i)$ for defining a cost function to be optimized, namely:

$$J(\Theta) = \sum_{i=1}^{L} (\bar{\sigma}^2(\tau_i) - \sigma^2(\tau_i; \Theta))^2. \tag{3.35}$$

However, the optimization process can be improved considering that the confidence of the Allan Variance improves as the number of independent clusters is increased. In particular, define δ the relative deviation of $\hat{\sigma}(\tau, M)$ from $\sigma(\tau)$:

$$\delta = \frac{\hat{\sigma}(\tau, M) - \sigma(\tau)}{\sigma(\tau)}, \tag{3.36}$$

where $\hat{\sigma}(\tau, M)$ denotes the estimate of the ADEV obtained from M independent clusters, and $\sigma(\tau)$ is the true value of the ADEV. Note that the following relations hold:

$$\hat{\sigma}(\tau, M) \xrightarrow[M \to \infty]{} \sigma(\tau). \tag{3.37}$$

Through some computations [198] the standard deviation, indicated as σ, of the random variable δ can be obtained:

$$\sigma[\delta] \propto M^{-1/2} = (n/m)^{-1/2}. \tag{3.38}$$

This quantity indicates the relative uncertainty of $\sigma(\tau)$ due to a finite number of measurements. Equation (3.38) shows that the estimation uncertainty for the ADEV in the region of short cluster length τ is small and increases proportional to $\sqrt{\tau}$.

Thus, the standard deviation of the experimental ADEV can be written as [181]:

$$\sigma[\bar{\sigma}(\tau)] = k \sqrt{\frac{m}{n}} \bar{\sigma}(\tau), \tag{3.39}$$

where k is an empirical constant. Approximating the variance of the experimental AVAR with [198]:

$$\sigma^2[\bar{\sigma}^2(\tau)] = (2\sigma[\bar{\sigma}(\tau)])^2, \tag{3.40}$$

and defining the weight w_i:

$$w_i = \frac{1}{\sigma^2[\bar{\sigma}^2(\tau_i)]}, \tag{3.41}$$

the cost function could be defined as:

$$J_w(\Theta) = \sum_{i=1}^{L} w_i(\bar{\sigma}^2(\tau_i) - \sigma^2(\tau_i;\Theta))^2. \tag{3.42}$$

Considering the model with the addition of rate ramp, a similar optimization problem has been set for finding the five parameters $\Theta = [T_B, Q_B, N, K, R]$. However, since the PSD is an odd function of the frequency f, its spectral factor would result in a non-integer order model. Thus, a state space form will be not readily available.

3.4.3 Statistical analysis of results

The analysis of variance (ANOVA) [199] can be used to determine whether and which main factors and interactions play a significant role in the performance of the system.

Following the approach of [182], we consider the effects model for a two factors design. Defining the observed response y_{ijk} when the first factor is at the i-th level ($i = 1, 2, \ldots, a$) and the second factor is at the j-th level ($j = 1, 2, \ldots, b$) for the k-th replicate ($k = 1, 2, \ldots, n$), we can write the following expression:

$$y_{ijk} = \mu + \tau_i + \beta_j + (\tau\beta)_{ij} + \epsilon_{ijk}, \tag{3.43}$$

where μ is the overall mean effect, τ_i are the effects of the ith level of the first factor, β_j are the effects of the jth level of the second factor, $(\tau\beta)_{ij}$ represent the effects of the interaction between τ_i and β_j and ϵ_{ijk} is a random error component. It is worth noting that factors and interactions have been defined as deviations from the overall mean:

$$\sum_{i=1}^{a} \tau_i = \sum_{j=1}^{b} \beta_j = \sum_{i=1}^{a} (\tau\beta)_{ij} = \sum_{j=1}^{b} (\tau\beta)_{ij} = 0. \tag{3.44}$$

In the ANOVA we are interested in testing the hypotheses regarding main factors:

$$H_0 : \tau_1 = \tau_2 = \cdots = \tau_a = 0$$
$$H_1 : \text{at least one} \quad \tau_i \neq 0$$

$$H_0 : \beta_1 = \beta_2 = \cdots = \beta_b = 0$$
$$H_1 : \text{at least one } \beta_i \neq 0$$

and interactions:

$$H_0 : (\tau\beta)_{ij} = 0 \text{ for all } i, j$$
$$H_1 : \text{at least one } (\tau\beta)_{ij} \neq 0$$

If all the null hypotheses H_0 presented above are true, then all the estimated mean squares should be equal to a value that we indicate as σ^2. An unbiased estimator of σ^2 is represented by the mean square of the errors ϵ_{ijk}. As a result, for evaluating if data are supporting the null hypothesis H_0 we can inspect the ratio between the effect or interaction mean square and the error mean square, denoted as F. If we obtain a large value, it means that this effect of interaction cannot be neglected. At the same time, we can look at the p-value, *i.e.*, the probability of obtaining that outcome if the null hypothesis was true. Thus, small p-values represent strong evidence against the null hypothesis.

However, this analysis is valid if and only if the observations are adequately described by the model reported in equation (3.43) and if the errors are normally and independently distributed with zero mean and constant but unknown variance σ^2. These assumptions can be checked looking at the normal probability plot of the residuals of the ANOVA, which is a graphical technique in which data lie on a straight line if they are approximately normally distributed.

3.5 Experimental results

In this Section, the models obtained from the Allan Variance analysis are discussed and validated through the use of a Kalman predictor. Finally, the results obtained through the ANOVA performed on the designed experimental campaign are reported.

3.5.1 Factorial design for visual odometry analysis

All experiments have been carried out in the FlyART using the ROG-1 aerial platform (see Chapter 2). The stereo camera frame rate for VO computation has been set to 640x480 and the frame rate to 15 fps. The visual odometry algorithm employed is the one provided in the ZED SDK[1]. For considerations about the choice of the software refer to Section 4.2.2. During the flights, position and yaw measurements for drone state estimation are provided by the Mocap, which is also used for collecting ground truth data. Finally, note that the position set-points are sent over Wi-Fi directly from a ground control station.

A factorial design has been carried out specifically for this analysis. It is worth noting that a lot of potential design factors can lead to important variations in the visual odometry performance and that can be interesting to be investigated.

The camera pitch, being a factor difficult to be changed by the experimenter and not necessarily of interest, has been considered a held-constant factor. Furthermore, an example of allowed-to-vary factor is the external ambient light, which is a small percentage of the one inside the laboratory, in turn controllable and that can be used as design factor.

In typical indoor operating modes, the factors that need to be considered are: shape of trajectory, flight altitude (which results in a change in the distance of tracked features), lighting conditions, changes in drone's attitude between frames, and speed.

One caveat regards the choice of speed as design factor. Performing the same trajectory at different speeds would result in time series of different lengths, leading to possible errors in the evaluation of the response

[1] https://www.stereolabs.com/developers/release/

variable. To face this issue, a closed trajectory can be selected and repeated multiple times when the speed is increased. As a consequence, the two factors are not independent and only the speed has been considered in the analysis.

In light of the aforementioned design factors, the experimental activity for characterizing visual odometry will be formulated as a 2^4 full factorial design.

The chosen trajectory is circular with 1 meter radius. The angular speed will be either $0.2\,\mathrm{rad/s}$ or $0.6\,\mathrm{rad/s}$. In this latter case, the trajectory will be traveled three times. For what concerns the drone's (or camera) attitude in the global inertial frame, it will be either kept fixed or varied in such a way that the first body axis is, at each instant in time, oriented as the velocity vector. This latter will be named yaw following mode in this work. The flight altitude will be either $1\,\mathrm{m}$ or $2\,\mathrm{m}$. Finally, the average light in the environment will be maintained around $100\,\mathrm{lux}$ or $250\,\mathrm{lux}$. In these conditions, the time series last about $32\,\mathrm{s}$, which represents a good tradeoff between the effort spent for performing the experiment and the amount of information captured. It is worth noting that longer experiments, which are carried out in the same operating conditions, are used for validation.

The 16 flights with the corresponding conditions are reported in Table 3.1.

3.5.2 Error models

An example of visual odometry position estimate and corresponding ground truth in the East-North-Up (ENU) frame, as well as the set-points provided to the drone, is shown in Figure 3.1, where the errors are already clearly visible, despite the short duration of the experiment.

The experimental ADEV has been computed for the visual odometry position error (obtained with respect to the Mocap measurements) in the inertial ENU frame. Considering the model comprising GM, RRW and ARW, the results are shown in Figure 3.2, which refers to the analysis of data collected in run 2 (see Table 3.1).

It is worth noting that the fit between experimental and constructed curves improves greatly by adding in the model formulation the rate ramp

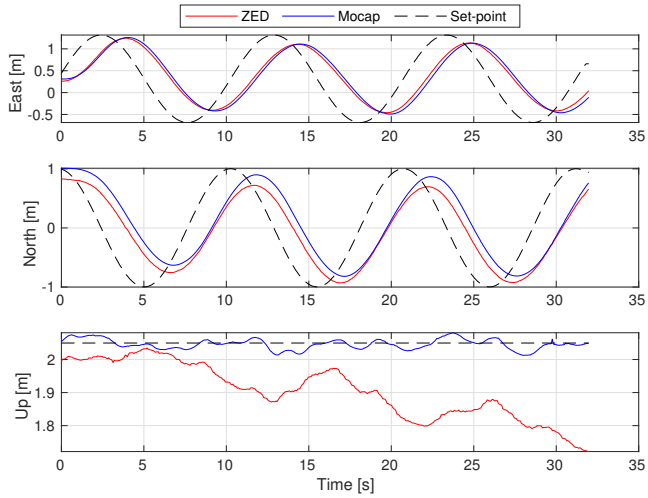


Figure 3.1: *Position results of an experimental run (run 16)*

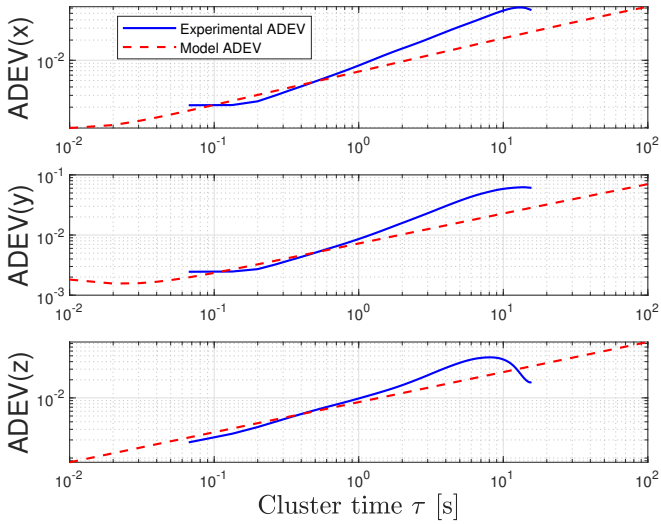


Figure 3.2: *Experimental and model ADEV for the three axis of the visual odometry position error in the inertial frame (run 2)*

Table 3.1: *Description of the experimental runs*

Run	Altitude [m]	Speed [rad/s]	Yaw Following [No/Yes]	Avg. Light [lux]
1	1	0.2	N	100
2	2	0.2	N	100
3	1	0.6	N	100
4	2	0.6	N	100
5	1	0.2	Y	100
6	2	0.2	Y	100
7	1	0.6	Y	100
8	2	0.6	Y	100
9	1	0.2	N	250
10	2	0.2	N	250
11	1	0.6	N	250
12	2	0.6	N	250
13	1	0.2	Y	250
14	2	0.2	Y	250
15	1	0.6	Y	250
16	2	0.6	Y	250

noise. This updated version is shown in Figure 3.3.

However, as already mentioned in Section 3.4, this model cannot be readily described by a state space representation. Note that methods to construct (high-order) rational approximations exist in the literature, *e.g.*, the Oustaloup approximation [200]. However, this would lead to an increase in the number of states and, consequently, in the required real-time computational load in a sensor-fusion framework. Additionally, in the case of weakly observable states, the more complex model can be less robust to unmodeled dynamics and nonlinearities.

For the model composed by ARW, RRW and GM, the state space form can be obtained combining (3.15) and (3.9) for the state equation and (3.6), (3.10), (3.16) for the output one:

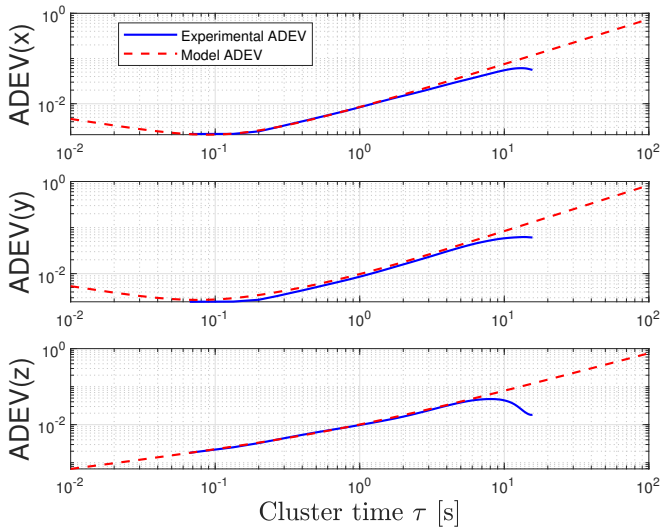


Figure 3.3: *Experimental and model (considering also rate ramp noise) ADEV for the three axis of the visual odometry position error in the inertial frame (run 2)*

$$\begin{Bmatrix} \dot{z}_B \\ \dot{z}_K \end{Bmatrix} = \begin{bmatrix} -\frac{1}{T_B} & 0 \\ 0 & 0 \end{bmatrix} \begin{Bmatrix} z_B \\ z_K \end{Bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{Bmatrix} \omega_B \\ \omega_K \end{Bmatrix} \tag{3.45}$$

$$z = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{Bmatrix} z_B \\ z_K \end{Bmatrix} + \omega_N \tag{3.46}$$

The corresponding process noise PSD matrix Q is defined as:

$$Q = \begin{bmatrix} Q_B & 0 \\ 0 & K^2 \end{bmatrix}, \tag{3.47}$$

while the only term appearing in the measurement noise PSD matrix R results to be the ARW PSD N^2.

3.5.3 Kalman predictor

In order to assess the performance of the models, a validation experiment has been executed in the same conditions of the 9th run of Table 3.1, but performing 12 times the same trajectory, for a total duration of $380\,\mathrm{s}$.

The simplified and high-fidelity models are retrieved from ADEV optimization (as in Figure 3.2 and 3.3) for the 9th run, and for brevity only the x-axis error is considered. In order to validate the obtained models, their PSD is compared with its estimate based on experimental data. The PSD estimate is computed through the Welch method [201] using no averaging to maintain a low bias. The comparison, represented in Figure 3.4, shows that the simpler model is able to reproduce the experimental data quite well. Thus, it will be used for the following analysis.

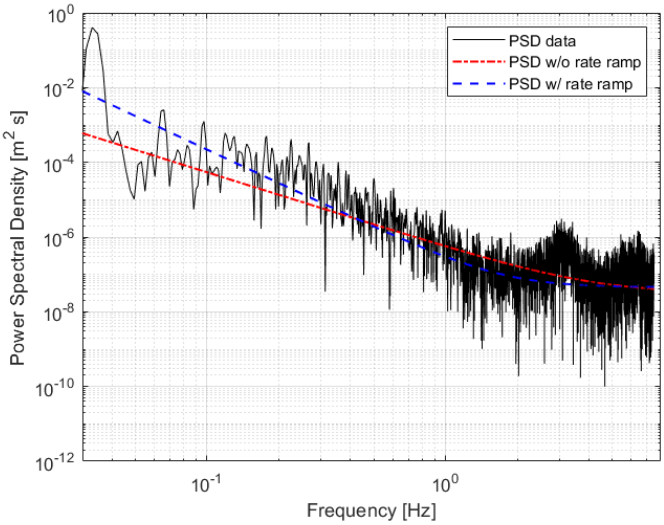


Figure 3.4: *Comparison between the models' PSDs, obtained with and without the inclusion of rate ramp noise from run 9, with the estimate of the PSD of experimental data obtained from a validation run (only x-axis considered)*

The model of the AVAR for the x-axis resulted in the following values for the noise parameters:

$$K^2 = 8.59e^{-5}\ \mathrm{m}^2/\mathrm{s}$$
$$N^2 = 4.26e^{-8}\ \mathrm{m}^2\mathrm{s}$$
$$Q_B = 1.27e^{-6}\ \mathrm{m}^2/\mathrm{s}$$

and the correlation time of $T_B = 25.06\,\mathrm{s}$. After having discretized the obtained state-space formulation for the simplified error model, a Kalman predictor, as presented in Section 3.3.2, has been implemented. The comparison between the obtained predicted output and the experimental data is shown in Figure 3.5. The difference between the two signals, as well the 3σ bounds for the prediction are shown in Figure 3.6.

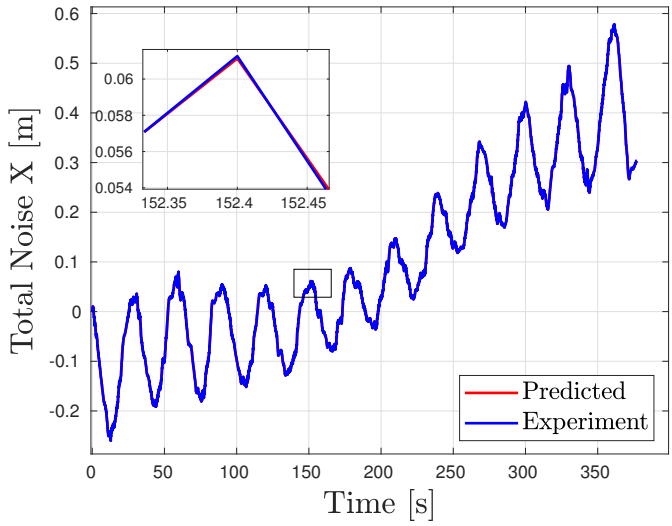


Figure 3.5: *Comparison between Kalman predictor output, with noise parameters of 9th run's model, and validation experimental data*

The results show good generalization capabilities of the obtained model, even using a longer validation experiment, which justifies the initial choice of a dataset of shorter runs' duration. The same procedure has been carried out also for the North and Up axes and similar results (omitted for brevity) were obtained.

3.5.4 Analysis of Variance

As already pointed out in Section 3.1, the characterization of vision systems is highly dependent on the environmental and flight conditions. In order to assess the changes in the produced models, the optimization for

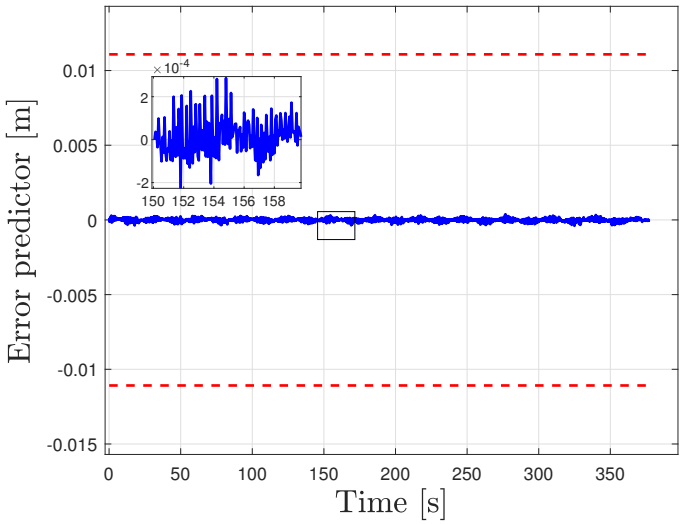


Figure 3.6: *Error of Kalman predictor output, with noise parameters of 9th run's model, with respect to validation experimental data and corresponding 3σ bounds*

the ADEV coefficients has been run on the 16 experiments of the factorial design presented in Table 3.1 and the computed spectral densities have been used as response variable in an analysis of variance. In particular, in order to obtain a single numerical value for the analysis, the three values of the PSD of the considered noise (corresponding to the three axis of the inertial reference system) are summed up and the square root is taken.

Let us first consider the main effects of the factors. The main effect of a factor can be computed as the difference between the average response at the low level and the one at the high level of the factor we are considering.

With particular reference to the RRW, the results are depicted in Figure 3.7.

It can be observed that the RRW greatly increases with the increase in speed and altitude. At the same time, the GM slightly decreases with the use of the yaw following mode, while it is basically constant with the change in ambient light.

When it comes to interactions, their effect can be estimated looking

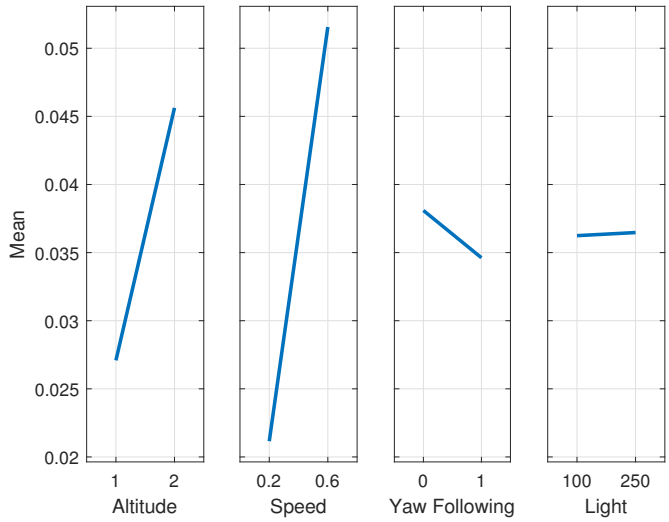


Figure 3.7: *Main effects plot of RRW spectral density*

at if the effect of a factor depends on the level chosen for another one. Interactions can be graphically interpreted from the plot in Figure 3.8.

Parallel lines in this plot mean that no interaction between the two factors exists. In our case, yaw following seems to interact with lighting conditions, speed and altitude, while speed and altitude look uncorrelated. However, this interpretation is subjective and cannot be the sole technique for the analysis of interactions. In order to reach an objective conclusion, we consider the ANOVA for the model comprising of both main effects and interactions (2-way), shown in Table 3.2.

Note that the main effects already highlighted, speed and altitude, have very large F values and very small p-values. Among the interactions, as already pointed out in the graphical analysis, yaw following seems to be related to the other factors. It is worth noting that, looking at their p-value, they seem less significant from the statistical point of view. In order to check the validity of the model with main effects and interactions consider Figure 3.9, in which the normal probability plot of the residuals of the ANOVA is shown.

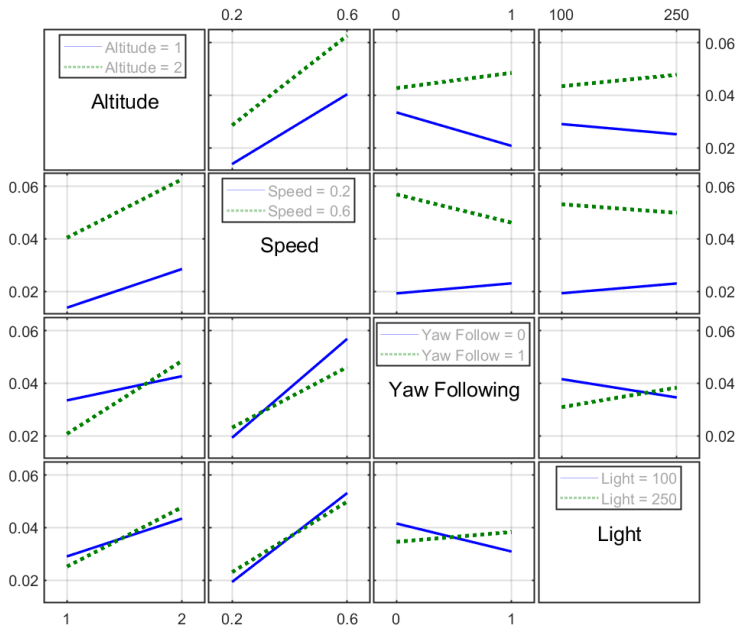


Figure 3.8: *Interaction plot of RRW spectral density*

Table 3.2: *2-way ANOVA on the RRW spectral density*

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-Value
Altitude	0.00137	1	0.00137	65.46	0.0005
Speed	0.0037	1	0.0037	176.04	0
Yaw Following	0.00005	1	0.00005	2.3	0.1902
Light	0	1	0	0.01	0.9248
Altitude*Speed	0.00006	1	0.00006	2.73	0.1593
Altitude*Yaw Following	0.00035	1	0.00035	16.47	0.0097
Altitude*Light	0.00007	1	0.00007	3.24	0.1316
Speed*Yaw Following	0.00021	1	0.00021	10.15	0.0244
Speed*Light	0.00005	1	0.00005	2.34	0.1868
Yaw Following*Light	0.00021	1	0.00021	9.95	0.0253
Error	0.00011	5	0.00002		
Total	0.00617	15			

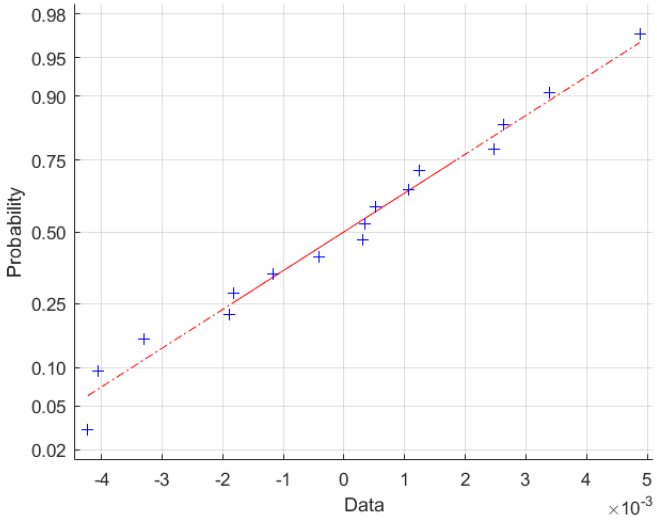


Figure 3.9: *Normal probability plot of the 2-way ANOVA's residuals for RRW*

The residuals for all the experimental runs lie approximately on a straight line, meaning that they are normally distributed and that the model with main effects and interactions adequately describes the observations.

For what concerns the ARW, comparing the normal probability plot of the residual of the two analysis of variance, *i.e.*, 1-way and 2-way ANOVA, the model which best approximates the linear behaviour results to be the one with interactions. Its results are shown in Table 3.3.

The only non-negligible factor for the ARW spectral density seems to be the altitude. In particular, considering the main effects plot (see Figure 3.10), to an increase in altitude corresponds a decrease in the ARW spectral density.

On the other hand, interactions are not statistically significant. Looking instead to the spectral density of the GM, the conclusion are not so clear. Also in this case, the model which best fits the data is the one with interactions (2-way ANOVA). The results are shown in Table 3.4.

The F values are quite small with respect to the other noise sources, thus no clear indication about main effects and interactions for the GM

Table 3.3: *2-way ANOVA on the GM spectral density*

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-Value
Altitude	$9.79{\cdot}10^{-7}$	1	$9.79{\cdot}10^{-7}$	6.29	0.054
Speed	$1.66{\cdot}10^{-7}$	1	$1.66{\cdot}10^{-7}$	1.06	0.3494
Yaw Following	$2.57{\cdot}10^{-9}$	1	$2.57{\cdot}10^{-9}$	0.02	0.9027
Light	$4.62{\cdot}10^{-8}$	1	$4.62{\cdot}10^{-8}$	0.3	0.6095
Altitude*Speed	$2.50{\cdot}10^{-7}$	1	$2.50{\cdot}10^{-7}$	1.61	0.2607
Altitude*Yaw Following	$2.28{\cdot}10^{-8}$	1	$2.28{\cdot}10^{-8}$	0.15	0.7177
Altitude*Light	$9.53{\cdot}10^{-8}$	1	$9.53{\cdot}10^{-8}$	0.61	0.4694
Speed*Yaw Following	$7.55{\cdot}10^{-10}$	1	$7.55{\cdot}10^{-10}$	0	0.9472
Speed*Light	$2.19{\cdot}10^{-8}$	1	$2.19{\cdot}10^{-8}$	0.14	0.7229
Yaw Following*Light	$3.44{\cdot}10^{-7}$	1	$3.44{\cdot}10^{-7}$	2.21	0.1974
Error	$7.79{\cdot}10^{-7}$	5	$1.56{\cdot}10^{-7}$		
Total	$2.71{\cdot}10^{-6}$	15			

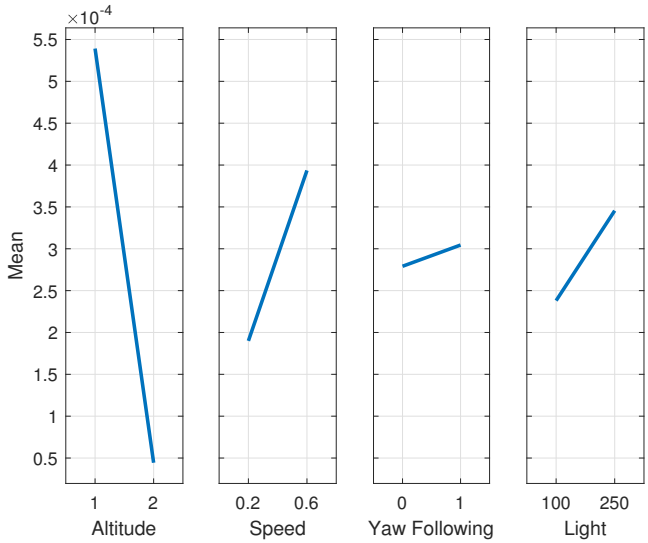


Figure 3.10: *Main effects plot for the ARW spectral density*

Table 3.4: *2-way ANOVA on the GM spectral density*

Source	Sum of Squares	Degrees of Freedom	Mean Square	F	P-Value
Altitude	$2.56{\cdot}10^{-6}$	1	$2.56{\cdot}10^{-6}$	0.11	0.7544
Speed	$7.91{\cdot}10^{-6}$	1	$7.91{\cdot}10^{-6}$	0.34	0.5867
Yaw Following	$1.48{\cdot}10^{-5}$	1	$1.48{\cdot}10^{-5}$	0.63	0.4633
Light	$3.49{\cdot}10^{-5}$	1	$3.49{\cdot}10^{-5}$	1.49	0.277
Altitude*Speed	$1.83{\cdot}10^{-5}$	1	$1.83{\cdot}10^{-5}$	0.78	0.4179
Altitude*Yaw Following	$8.34{\cdot}10^{-5}$	1	$8.34{\cdot}10^{-5}$	3.56	0.118
Altitude*Light	$3.75{\cdot}10^{-5}$	1	$3.75{\cdot}10^{-5}$	1.6	0.2618
Speed*Yaw Following	$3.88{\cdot}10^{-5}$	1	$3.88{\cdot}10^{-5}$	1.65	0.2548
Speed*Light	$4.81{\cdot}10^{-6}$	1	$4.81{\cdot}10^{-6}$	0.21	0.6696
Yaw Following*Light	$4.18{\cdot}10^{-5}$	1	$4.18{\cdot}10^{-5}$	1.78	0.2393
Error	0.00012	5	$2.35{\cdot}10^{-5}$		
Total	0.0004	15			

can be retrieved.

To summarize, the RRW, the noise term related to the drift, *i.e.*, the rate of increase in the bias of the visual odometry position measurement with respect to the true position, resulted mostly dependent on the speed and, to a lesser extent, on the altitude of the performed flight. In this respect, the speed can act by placing additional burden on the feature detection process and on the matching over consecutive frames, accelerating the accumulation of errors. On the other hand, the altitude increases the average distance between the tracked features and the camera, which in turn increase the error in the triangulation process. The ARW, *i.e.*, the white noise directly affecting the measurements, seems decreasing with the increase in altitude. However, the statistically significance of these results are lower with respect to the ones belonging to the RRW analysis. The GM spectral density, which represents the bias that varies with time in a correlated fashion, is not clearly dependent with the considered flight and environmental conditions. The same holds true also for the GM correlation time.

3.6 Concluding remarks

The problem of characterizing and modeling the errors affecting stereo odometry position measurements has been presented. The Allan Variance has been employed as a tool for the determination of the noise terms affecting the visual odometry measurements. These noise terms have been used as driving inputs of a state space model having as states the odometry errors. For models' validation, a one-step-ahead Kalman predictor has been implemented and applied on a validation dataset showing good performance and, thus, good generalization capabilities of the proposed approach. Furthermore, a four-factors factorial design has been used for collecting and analyzing experimental data. The values of the models' noise terms coefficients have been studied against changes in flight altitude, speed, change in the drone's attitude and lighting conditions. Results have showed that an increase in altitude and speed play a key role in the increase in the noise term related to non time-correlated bias. In this respect, ambient ambient light (in non extreme conditions) and changes in attitude result of smaller importance.

CHAPTER 4

Leonardo Drone Contest autonomous drone competitions

In this Chapter, the solutions proposed for the *Leonardo Drone Contest* (LDC) are presented. As already mentioned in the Introduction, the LDC is an annual autonomous drone competition among Italian universities, which has already seen the conclusion of its third edition. In this framework, we are going to discuss the three autonomous guidance and navigation solutions presented in the three past competitions. In particular, for each edition, the rules, objectives and overview of the proposed solutions will be presented. Then, the guidance and navigation architecture will be analyzed in detail along with the obtained simulation and experimental results. Finally, comments and takeaways of the proposed solution will be provided.

4.1 Related works

As already mentioned in the Introduction, several drone competitions have been organized in the previous years. Their aim was very broad ranging from drone racing [1] to the autonomous accomplishment of complex tasks. In both the cases, in the literature, several papers have been published to describe the software and hardware architectures employed by the teams.

For what concern autonomous drone racing, the first competition was held at the 2016 International Conference on Intelligent Robots and System (IROS). Since then, the event was proposed annually at each IROS conference. In these competitions, there was no pre-defined platform to be used, but it could be designed by the teams. One of the few requirements for the platform was to carry all sensing and computing onboard. A collection of the hardware platform proposed in these competitions (from 2016 to 2019) is available in [202]. At the same time, in [28] the software solutions employed by the teams in IROS 2016 and 2017 were analyzed. Finally, in [203], the approach employed for winning the IROS 2018 drone race has been presented.

Inspired by this competition, other international competitions were launched: the AlphaPilot organized by Lockheed Martin in collaboration with the Drone Racing League, and the Game of Drones, organized by Microsoft, Stanford University and the University of Zurich.

The AlphaPilot was held in 2019 and organized in two stages. In the first phase, the teams had to complete a simulated race using the AirSim simulator [204]. In the second phase, the selected teams were provided with a standard vehicle on which they had to upload their software solution for competing in a real drone race. Some of the solutions proposed in the Alphapilot have been published in research papers by different teams, namely: MAVLab [205], UZH RPG [206], XQuad [207].

Game of drones was held as part of the conference on Neural Information Processing Systems (NeurIPS) in 2019. Similar to the autonomous drone racing at IROS and AlphaPilot competitions, the aim was to traverse a race track in the shortest possible time. However, this competition

[1] With drone racing we identify all the competitions in which the drone should fly at high speed in unknown or partially unknown environments and cross a number of gates in a predefined sequence.

was entirely run in the high-fidelity simulation environment AirSim [208]. Participants had the choice of three tiers: Planning only, Perception only, or Full Autonomous Racing. The aim was to combine challenges from adversarial planning and real-time perception and to encourage fusing learning and model-based approaches [209]. Solutions for this competition have been object of different scientific publications [210, 211, 212].

Other than drone racing, autonomous UAVs have been object of competitions in conferences such as IMAV (International Micro Air Vehicle) [213] and ICUAS (International Conference on Unmanned Aircraft Systems) [214].

Beside conferences, some events were created on-purpose to push the limits of the existing technology. The first competition was the International Aerial Robotics Competition (IARC) [215] of 1991. In this competition, a small unmanned helicopter had the goal of moving a metallic disc from one side of an arena to another in an autonomous way. Starting from 2009, multicopters started to replace small helicopters in the IARC competitions [216, 217, 218].

Another example is the Mohamed Bin Zayed International Robotic Challenge (MBZIRC) [32]. Two editions were held in 2017 and 2020 respectively, each of which composed by three different challenges, with an upcoming competition that will occur in 2023. In the 2017 edition, Challenge 1 required a UAV to locate, track, and land on a moving vehicle; Challenge 2 required a ground autonomous robot to locate and navigate to a panel, and physically operate a valve stem on it with the appropriate tool; Challenge 3, coined ”Treasure hunt”, required a team of UAVs to cooperate to search, track, pick up, and drop a set of static and moving objects. Some teams published their solution for Challenge 1 and 3 [219, 220], while some others only regarding Challenge 3 [221]. In the 2020 edition, Challenge 1 required to pursue a target UAV to capture a ball attached to it, while a second UAV should find and pop balloons scattered throughout the arena; Challenge 2 required a larger UAV to pick, transport and place bricks to build a wall; Challenge 3 required a UAV to autonomously find fires and extinguish them with an onboard fire extinguisher. Solutions to the mentioned challenges can be found respectively in [222, 223], [224] and [225].

4.2 First edition

The first edition of the Leonardo Drone Contest (LDC) took place in the LDC arena (see Section 2.2.2) on 17-18 September 2020. In the arena, the obstacles shape, their dimension and position were unknown to the participants. On the other hand, six poles of different colors were placed at known positions on the field as represented in Figure 4.1.

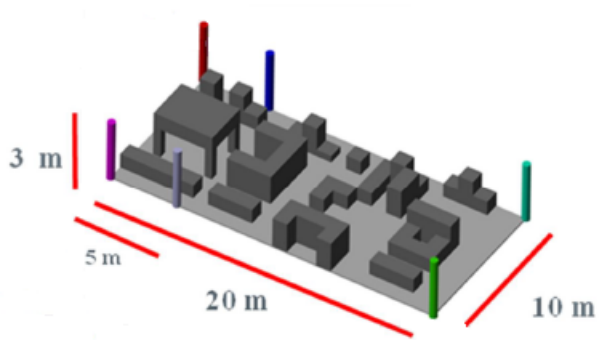


Figure 4.1: *Rendering example of the competition field indicating positions and colors of poles*

The contest was composed of two phases, held respectively in the two days of the competition. In the first phase, the drone had the aim of exploring autonomously the environment. At the same time, the drone task was to localize 10 QR codes, serving also as landing pads for the second phase. The landing pad was made of a $1\,\mathrm{m}\times1\,\mathrm{m}$ blue square containing a $50\,\mathrm{cm}\times50\,\mathrm{cm}$ QR marker (see Figure 2.4a). The QR markers contained a unique alphanumeric string each.

Just before the start of the second day, the teams were given a list of 5 QR codes by the competition judges. This list indicated the ordered sequence of pads to be reached by the drone and on which the drone was meant to land. Hence, the teams had the opportunity of planning an optimized path and upload it on the drone. The team performing the largest number of consecutive valid landings would have been identified as the winner. Note that a landing was considered valid if all the drone's contact points with the ground were inside the $1\,\mathrm{m}\times1\,\mathrm{m}$ landing pad.

4.2.1 Solution overview

The ROG-1 (see Section 2.2.3) platform was specifically designed to take part to this competition. For what concerns state estimation, stereo VO was fused with Inertial Measurement Unit (IMU) data in a *loosely coupled* fashion (see Section 1.2.4) for obtaining an estimate of the drone pose in the *odom* frame $\mathcal{O}$. Note that, while the visual odometry was computed on the companion computer, the fusion process was carried out on the FCU using the EKF2 (see Appendix A).

The open source RTAB-Map package [226] was employed for localization in the *map* frame $\mathcal{M}$, using as input the stereo camera images. RTAB-Map stands for Real-Time Appearance-Based Mapping and it is a RGB-D, Stereo and LiDAR graph-based SLAM approach, in which loop closures are found using a bag-of-word approach (see [227] for more details). Note that the map produced by the RTAB-Map algorithm in the first phase of the competition was saved in the form of an Octomap.

Two different planning algorithms were used depending on the considered phase of the competition. During the exploration phase, the RH-NBV planner was employed. Starting from stereo camera images, the aim of this algorithm was to produce a collision-free path from the current pose to the viewpoint which maximized the acquired knowledge of the environment (for details see Section 1.3.5). While the environment was explored, the downward-looking monocular camera was used for detecting the QR codes scattered around the field. At the same time, the QR codes' positions were computed and saved, given the best estimate of the drone position in the *map* reference frame obtained from the SLAM algorithm. The overall system architecture for this phase is depicted in Figure 4.2.

In the second day, once the list of landing pads to be reached was known, a path was planned using an offline A*, which took as input the retrieved Octomap. The path was then uploaded on the UAV. Onboard the UAV an online local planner was running. Its aim was to avoid possible collisions, which could arise due to uncertainties in both the map and the UAV state estimate. The local planner was an implementation of Vector Field Histogram (VFH*) [228], which utilized the range information coming from the TeraRanger Tower EVO. Due to the planar nature

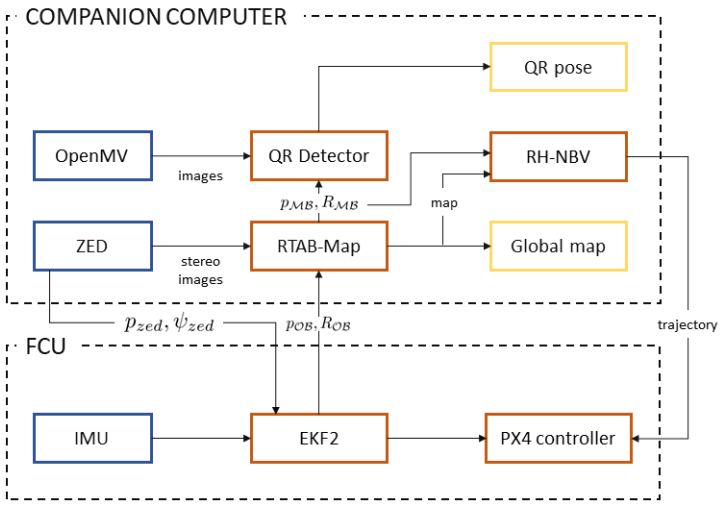


Figure 4.2: *System architecture for the exploration phase in LDC first edition*

of the sensor, a 2D implementation of the planner was used. The resulting architecture for the landing phase is shown in Figure 4.3.

4.2.2 Navigation

First of all, multiple ROS-compatible implementations for computing visual odometry, given the stereo images coming from the ZED stereo camera, have been analyzed and compared in experimental tests.

Visual odometry

The following methods have been considered:

- RTAB-Map, which was already presented for its visual SLAM implementation. The package also provides a stereo visual odometry pipeline (no loop closing enabled), which is the one we will consider in this framework.
- ORB-SLAM2 [71] is a graph-based stereo SLAM already mentioned in Section 1.2.3. The SLAM system is based on keyframes contain-

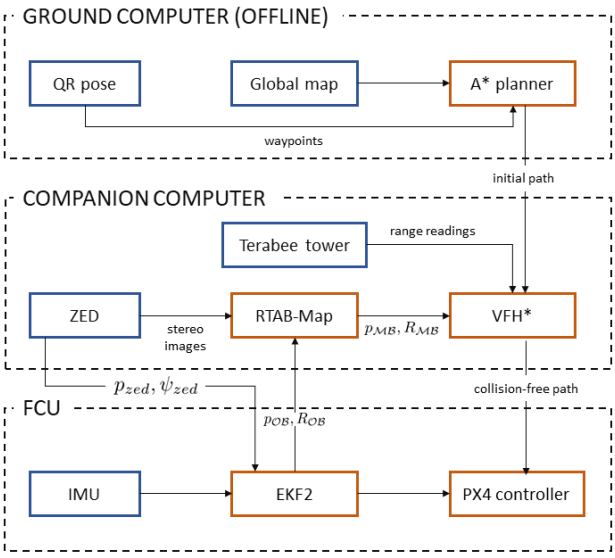


Figure 4.3: *System architecture for the landing phase in LDC first edition*

ing a set of features and the camera pose. A local and global bundle adjustment are used to correct a recent set of keyframes and to optimize the map and trajectory respectively.

- LibViso2 [229] is a feature-based VO library for monocular and stereo cameras. Features are extracted by filtering the images with a corner and blob mask and performing non-maximum and non-minimum suppression on the filtered images.

- ZED-VO is the proprietary visual odometry software provided in the ZED SDK [230].

A simple trajectory, at constant flight altitude and in good lighting conditions, has been selected for this experiment and has been traveled by the drone. On the recorded dataset, all presented algorithms have been executed and compared. In particular, a comparison has been carried out considering the Euclidean norm of the difference between the visual odometry estimate and the ground truth position as error metric. It is worth

noting that the image undistortion and stereo rectification have been performed on the Jetson TX2 Graphics Processing Unit (GPU) by the ZED SDK. The results are graphically shown in Figure 4.4.

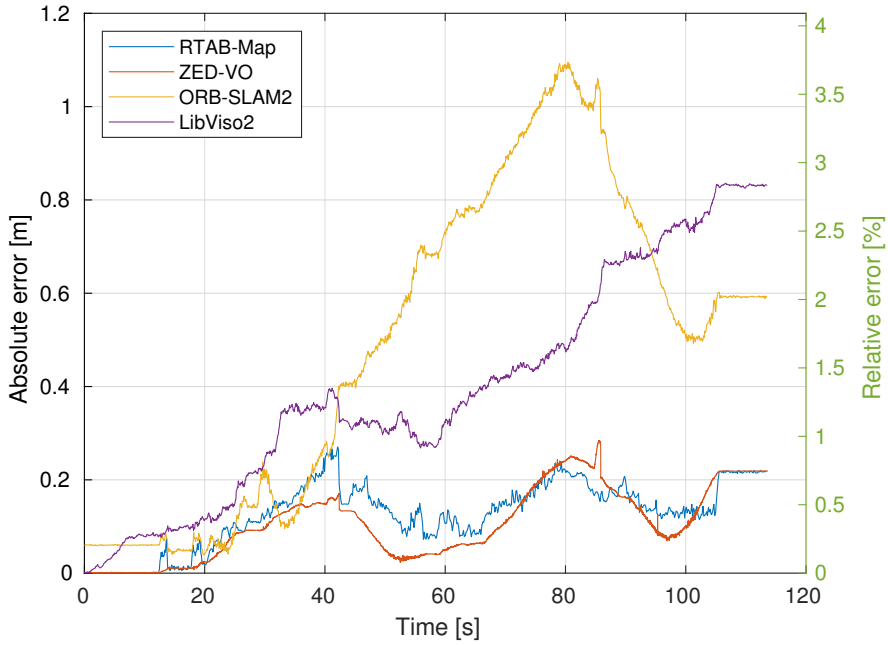


Figure 4.4: *Comparison among stereo visual odometry algorithms using ZED stereo camera*

The mean absolute and relative error (with respect to the traveled distance) are reported in Table 4.1.

All the algorithms show good results in the proposed experiment, with the best performance obtained by the ZED-VO algorithm. Considering the available hardware, *i.e.*, with GPU acceleration, this algorithm outperforms open-source algorithms with CPU implementations. Similar results, using ground vehicles, can be also found in [231]. An experimental comparison of ROS implementations of VO algorithms can be also found in [232], where different hardware (stereo, mono, RGB-D cameras) are used. A comparison among monocular VO algorithms is instead shown

Table 4.1: *Mean absolute and relative error of analyzed stereo visual odometry algorithms using ZED stereo camera*

Algorithm	Mean error [m]	Relative error [%]
RTAB-Map	0.122	0.416
ZED-VO	0.106	0.360
LibViso2	0.395	1.346
ORB-SLAM2	0.492	1.676

in [233]. For what concerns aerial vehicles, the performance of monocular VIO algorithms is analyzed and compared in [234, 76].

A similar experiment has been also carried out, but using the Intel RealSense D435i camera [235]. This time, the RTAB-Map and ORB-SLAM2 RGB-D SLAM versions have been used. The algorithms have been run on an Intel I7 processor laptop. The same trajectory of the previous experiment has been traveled. The results are graphically shown in Figure 4.5.

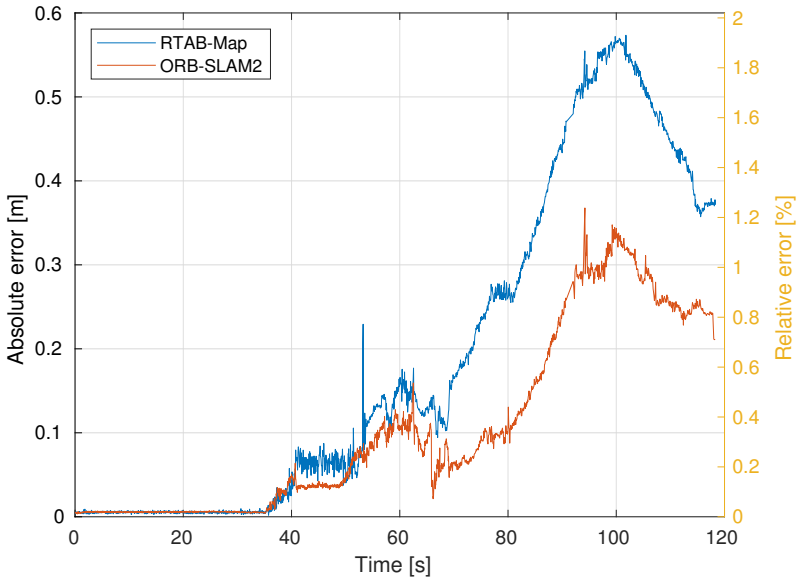


Figure 4.5: *Comparison of RTAB-Map and ORB-SLAM2 for Intel D435 depth camera*

The mean absolute and relative error are reported in Table 4.2.

Table 4.2: *Mean absolute and relative error of analyzed stereo visual odometry algorithms using Intel D435 depth camera*

Algorithm	Mean error [m]	Relative error [%]
RTAB-Map	0.401	1.404
ORB-SLAM2	0.177	0.6197

The obtained results with the Intel D435i camera are comparable with the results obtained through the stereo implementations of both RTAB-Map and ORB-SLAM2. However, the ZED stereo camera, with the available computing board, and its proprietary algorithm outperforms other implementations. Thus, this configuration will be used for computing VO for the remaining of this work. A drawback of the selected camera is that it needs a powerful GPU for running its algorithm, while other commercial solutions, *e.g.*, Intel T265 [236], can perform the task without any additional resource. Furthermore, the weight of the ZED stereo camera is slightly higher, weighing $164\,\mathrm{g}$ against the $74\,\mathrm{g}$ of the Intel D435i. An extensive comparison between the Intel RealSense and ZED for robotic vision applications is available in [237].

EKF-SLAM

As already mentioned in the solution overview (see Section 4.2.1), the position and yaw angle, output of the VO algorithm are fused in the EKF2 with IMU data. The EKF2 returns the estimates of position and attitude of the UAV in the *odom* frame $\mathcal{O}$, $p_{\mathcal{OB}} = [p_x, p_y, p_z]^\top$ and $R_{\mathcal{OB}}$ respectively, which will be used for feedback control.

For what concerns the localization in the *map* frame $\mathcal{M}$, an alternative solution to the one presented in the solution overview, was designed and tested in simulations. This solution involved the knowledge of the position of the colored poles placed at the boundary of the competition field (see Figure 4.1). This problem, *i.e.*, the one of localizing a robot in a field where known colored landmarks were placed, was already faced in the Robocups. In this framework, many different localization approaches could be found in the literature [238], [239].

We implemented an EKF-SLAM approach, which exploits a very simple kinematic model as motion model, and the colored poles and the QR codes available on the field as visual landmarks (a similar work is available in [240]).

For what concerns the colored poles, they are detected and identified through color segmentation on images captured by monocular cameras, equipped specifically for this task. In the simulations, four cameras have been placed on the drone, facing in each direction, *i.e.*, with the axis c_3 forming an angle of 0, 90, 180 and 270 degrees with respect to b_1. Consider Figure 4.6 for a simulation example. Then, the distance, elevation and azimuth of the poles with respect to the robot frame are computed. Note that the distance is retrieved from the monocular images knowing the dimensions of the pole itself.

On the other hand, the QR markers are detected and localized using the ROS ViSP visual servoing library [241]. The output of this algorithm is the pose of the QR center in the downward-looking camera frame $\mathcal{C}$ and its identifier. Note that, while the poles' positions were known beforehand, the QR markers' positions were unknown and their localization was the aim of the first phase of the competition.

The state of the EKF-SLAM is composed by the bias term $b_{\mathcal{MO}} = [b_x, b_y, b_z]^\top$ and the position of each QR marker in the map frame $\tilde{p}_{\mathcal{M}_i} = [\tilde{x}_i, \tilde{y}_i, \tilde{z}_i]^\top$ with $i = 0, 1, \ldots, 9$. The bias term, represents the translation term $t_{\mathcal{MO}}$, which appears in the frame transformation

$$p_{\mathcal{M}} = R_{\mathcal{MO}} p_{\mathcal{O}} + t_{\mathcal{MO}}, \tag{4.1}$$

used for computing the position of a point in the map frame $\mathcal{M}$, $p_{\mathcal{M}}$, starting from the knowledge of the position of a point in the *odom* frame $\mathcal{O}$, $p_{\mathcal{O}}$. Thus, the bias term can be regarded as the drift accumulated by the VIO during the flight. Assuming the evolution of this term as a random walk process, the motion model can be written in discrete time state space form as :

$$\hat{b}_{\mathcal{MO}_k} = b_{\mathcal{MO}_{k-1}} + w_{k-1}, \tag{4.2}$$

where $w_k \sim \mathcal{N}(0, Q)$ is a Gaussian random vector with covariance Q.

For what concerns the measurement model, when the jth pole is detected and identified, its known position in the *map* reference frame $\bar{p}_{\mathcal{M}}$,

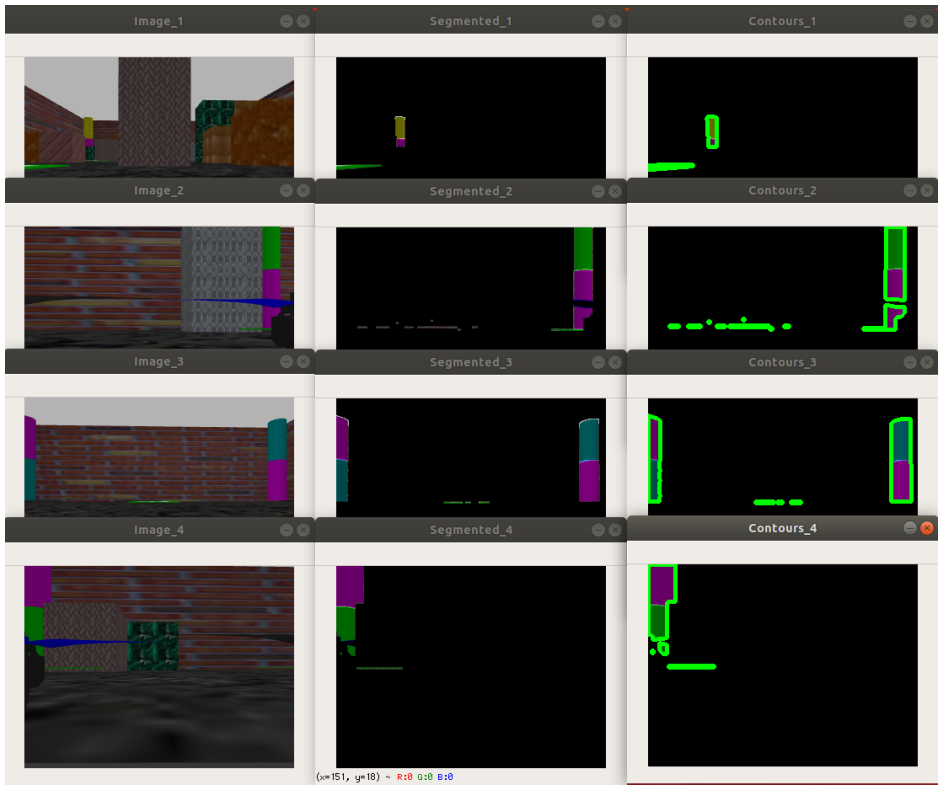


Figure 4.6: *Color segmentation for colored poles' detection and identification*

is transformed into the robot body frame, $\mathcal{B}$, namely:

$$\bar{p}_{\mathcal{B}_j} = [\bar{x}_j, \bar{y}_j, \bar{z}_j]^\top = R_{\mathcal{MB}}^\top(\bar{p}_{\mathcal{M}_j} - (b_{\mathcal{MO}} + R_{\mathcal{MO}} p_{\mathcal{OB}})). \tag{4.3}$$

$$= R_{\mathcal{OB}}^\top(\bar{p}_{\mathcal{M}_j} - (b_{\mathcal{MO}} + p_{\mathcal{OB}})), \tag{4.4}$$

where we have omitted index k and where equation (4.4) is based on the following assumption:

Assumption 1. *We assume that $R_{\mathcal{OB}} = R_{\mathcal{MB}}$, which also lead to $R_{\mathcal{MO}} = I_3$, where with I_3 we indicate the 3×3 identity matrix. In other words, we are assuming that the attitude estimated by the VIO $R_{\mathcal{OB}}$ shows a very slow drift. The validity of this assumption has been confirmed during the experimental campaign in the FlyART.*

Consider, in fact, Figure 4.7, in which the difference between the heading angle estimate produced by the VIO and the ground truth are shown. Despite the length of the flight (about 10 minutes), the drift amounts to a few degrees, confirming the mentioned assumption.

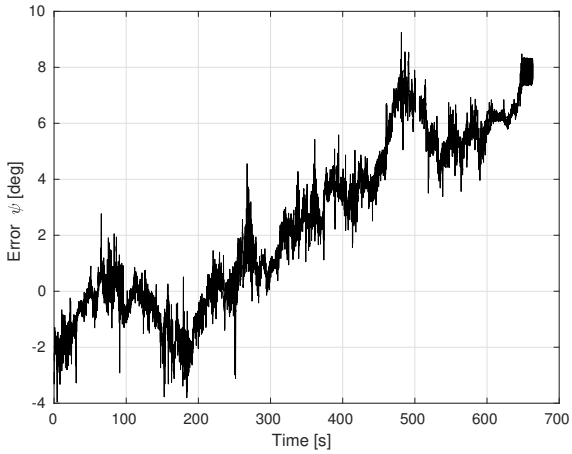


Figure 4.7: *Error between heading angle ψ estimated by the VIO and ground truth*

This assumption is also the reason why the drone attitude $R_{\mathcal{MB}}$, or equivalently $R_{\mathcal{MO}}$, has not been included in the filter state.

Then, the azimuth, elevation and distance of the jth colored pole can be computed, respectively, as:

$$\alpha = \arctan\left(\frac{\bar{y}_j}{\bar{x}_j}\right) + v_\alpha \tag{4.5}$$

$$\beta = \arctan\left(\frac{\bar{z}_j}{\sqrt{\bar{x}_j^2 + \bar{y}_j^2}}\right) + v_\beta \tag{4.6}$$

$$\rho = \sqrt{\bar{x}_j^2 + \bar{y}_j^2} + v_\rho. \tag{4.7}$$

where v_α, v_β and v_ρ are respectively zero mean Gaussian white noises with variances r_α, r_β and r_ρ. The measurement noise covariance is equal to:

$$R_p = \begin{bmatrix} r_\alpha & 0 & 0 \\ 0 & r_\beta & 0 \\ 0 & 0 & r_\rho \end{bmatrix}. \tag{4.8}$$

The three values are collected in a vector: $y_p = [\alpha, \beta, \rho]^\top$.

On the other hand, once the ith QR marker is detected by the downward-looking camera, the following measurement model (omitting index k) is applied:

$$\begin{aligned} y_c &= t_{\mathcal{CT}} = \\ &= R_{\mathcal{CB}}(R_{\mathcal{MB}}^\top(\tilde{p}_{\mathcal{M}_i} - (b_{\mathcal{MO}} + R_{\mathcal{MO}} p_{\mathcal{OB}}))) + t_{\mathcal{CB}} + v_c, \end{aligned} \tag{4.9}$$

where v_c is a zero mean Gaussian white noise with covariance R_c: $v_c \sim \mathcal{N}(0, R_c)$. This model represents simply two consecutive frame transformation (see equations (1.1) and (1.2)): from inertial *map* frame to body frame and from body frame to camera frame. Using again Assumption 1, equation (4.9) simplifies into:

$$y_c = R_{\mathcal{CB}}(R_{\mathcal{OB}}^\top(\tilde{p}_{\mathcal{M}_i} - (b_{\mathcal{MO}} + p_{\mathcal{OB}}))) + t_{\mathcal{CB}} + v_c. \tag{4.10}$$

All the terms, except the state, are now known: $T_{\mathcal{OB}} = (p_{\mathcal{OB}}, R_{\mathcal{OB}})$, *i.e.*, the drone pose in the *odom* frame, is retrieved by the EKF2 and $T_{\mathcal{CB}} = (t_{\mathcal{CB}}, R_{\mathcal{CB}})$ is the known and constant camera pose with respect to the drone body frame. For initializing the landmarks, *i.e.*, insert them in the state vector, the inverse measurement model should be employed. It can be obtained inverting equation (4.10):

$$\tilde{p}_{\mathcal{M}_i} = R_{\mathcal{OB}} R_{\mathcal{CB}}^{\top}(z_c - t_{\mathcal{CB}}) + (b_{\mathcal{MO}} + p_{\mathcal{OB}}). \tag{4.11}$$

where z_c is the position measurement output of the ViSP QR tracker, representing the term $t_{\mathcal{CT}}$. The entire pseudo-code is shown in Algorithm 1.

The code has been validated in the simulation environment of Figure 2.1a. The resulting estimated position of the robot in the *map* frame $\mathcal{M}$, namely $p_{\mathcal{MB}} = b_{\mathcal{MO}} + p_{\mathcal{OB}}$ is shown[2] in Figure 4.8, while the estimate of the position of the QR markers is reported in Table 4.3.

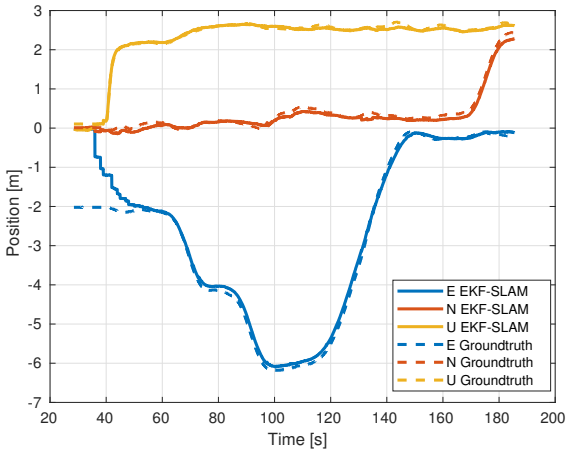


Figure 4.8: *EKF-SLAM simulation results*

Even if the approach showed good performance in the simulation environment, the solution was not implemented on the platform due to the following limitations. First, the solution would have required additional

[2]Assumption 1 has been exploited again

Algorithm 1 EKF-SLAM using poles and QR markers

procedure EKF-SLAM($\bar{p}_{\mathcal{M}_j}, p_{\mathcal{OB}}, R_{\mathcal{OB}}, z_{p_k}, z_{c_k}$)

$$G = \begin{bmatrix} 1 & 0 & 0 & \underbrace{0 \cdots 0}_{\text{3N}} \\ 0 & 1 & 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 \end{bmatrix}$$

$\hat{x}_k = x_{k-1}$

$\hat{P}_k = F_k P_{k-1} F_k^\top + G^\top Q G$

for observed landmark **do**

if landmark is the jth pole **then**

Extract $b_{\mathcal{MO}} = x_k(1:3)$

Compute $y_{p_k} = h_p(b_{\mathcal{MO}}, \bar{p}_{\mathcal{M}_j}, p_{\mathcal{OB}}, R_{\mathcal{OB}})$ (see (4.4)-(4.7)).

Compute innovation $\delta_{p_k} = z_{p_k} - y_{p_k}$

Evaluate Jacobian $H_{p_k} = H_{p_k}(b_{\mathcal{MO}}, \bar{p}_{\mathcal{M}_j}, p_{\mathcal{OB}}, R_{\mathcal{OB}})$

$S_k = H_{p_k} \hat{P}_k H_{p_k}^\top + R_p$

$K_k = \hat{P}_k H_{p_k}^\top S_k^{-1}$

$x_k = \hat{x}_k + K_k \delta_{p_k}$

$P_k = (I - K_k H_{p_k}) \hat{P}_k$

else if landmark is the ith QR marker **then**

if landmark ith has not been initialized **then**

Extract $b_{\mathcal{MO}} = x_k(1:3)$

Compute $\tilde{p}_{\mathcal{M}_i} = \tilde{h}_c(b_{\mathcal{MO}}, z_c, p_{\mathcal{OB}}, R_{\mathcal{OB}}, t_{\mathcal{CB}}, R_{\mathcal{CB}})$ (see (4.11))

Initialize state $x_k(4+3i:6+3i) = \tilde{p}_{\mathcal{M}_i}$

Evaluate Jacobian $\tilde{H}_{c_k} = \tilde{H}_{c_k}(b_{\mathcal{MO}}, z_c, p_{\mathcal{OB}}, R_{\mathcal{OB}}, t_{\mathcal{CB}}, R_{\mathcal{CB}})$

Initialize covariance $P_k = \tilde{H}_{c_k} \hat{P}_k \tilde{H}_{c_k}^\top + R_c$

else

Extract $b_{\mathcal{MO}} = x_k(1:3)$

Extract $\tilde{p}_{\mathcal{M}_i} = x_k(4+3i:6+3i)$

Compute $y_{c_k} = h_c(b_{\mathcal{MO}}, \tilde{p}_{\mathcal{M}_i}, p_{\mathcal{OB}}, R_{\mathcal{OB}}, t_{\mathcal{CB}}, R_{\mathcal{CB}})$ (see (4.10))

Compute innovation $\delta_{c_k} = z_{c_k} - y_{c_k}$

Evaluate Jacobian $H_{c_k} = H_{c_k}(b_{\mathcal{MO}}, \tilde{p}_{\mathcal{M}_i}, p_{\mathcal{OB}}, R_{\mathcal{OB}}, t_{\mathcal{CB}}, R_{\mathcal{CB}})$

$S_k = H_{c_k} \hat{P}_k H_{c_k}^\top + R_c$

$K_k = \hat{P}_k H_{c_k}^\top S_k^{-1}$

$x_k = \hat{x}_k + K_k \delta_{c_k}$

$P_k = (I - K_k H_{c_k}) \hat{P}_k$

end if

end if

end for

end procedure

Table 4.3: *Estimated and ground truth QR markers positions*

	QR code id 4	QR code id 6	QR code id 7
$\hat{x}[m]$	-1.98	-5.92	-0.12
$x_{GT}[m]$	-2.00	-6.00	0.00
$\hat{y}[m]$	-2.06	0.03	2.92
$y_{GT}[m]$	-2.00	0.00	3.00
$\hat{z}[m]$	0.08	-0.08	-0.05
$z_{GT}[m]$	0.00	0.00	0.00

cameras to properly work. Possible occlusions due to the obstacles present on the field limited greatly the amount of measurements obtained, leading to the need of increasing the available field of view. Processing more camera images would have also implied a greater computation overhead for the overall system. Furthermore, while the azimuth and elevation were very accurate, distance estimates were not. This limited the accuracy of the proposed solution. The robustness of the solution was also difficult to assess, since it was not possible to test the system on the competition field before the actual contest. A concern regarded possible changes in lighting conditions, which could have led to issues in the color segmentation task. Finally, adopting this strategy would have reduced the generality of the solution and limited its deployment in other application domains.

Consequently, as already mentioned in the solution overview, the mapping and localization task was carried out by RTAB-Map. The resulting TF tree for the ROG-1 platform is shown in Figure 4.9.

The graph containing the ROS topics involved in the navigation procedure is shown in Figure 4.10. Note how, on one side, the odometry produced by the ZED stereo camera is remapped to the *mavros* node for sending it to the Pixhawk 4 FCU, while the stereo camera images are used for mapping and localization through RTAB-Map. At the same time, the OpenMV monocular camera is used for detecting and identifying QR markers. Without the use of the EKF-SLAM algorithm, the QR markers are mapped simply by applying the inverse measurement model (4.11), given the best estimate of the UAV pose in the *map* frame. Finally, the TeraRanger Tower EVO sensor measurements are combined and they will be published as a point cloud for collision avoidance, as we will see in the

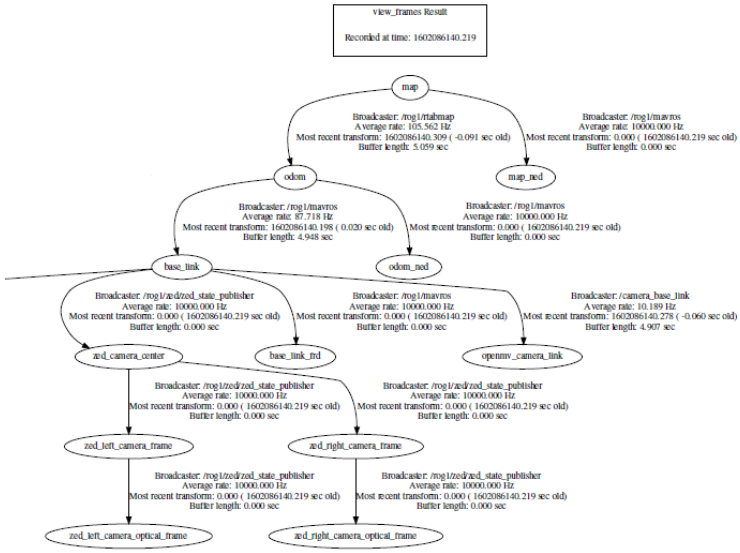


Figure 4.9: *TF tree for ROG-1 navigation (last connection associated to the TeraRanger Tower EVO, to which the 8 laser scanner TFs are connected)*

next Section.

4.2.3 Guidance

As mentioned in the overview, two different guidance strategies were employed depending on the phase of the competition. In the first phase, the RH-NBV planner was employed. This algorithm produces a trajectory for reaching the best waypoint from which exploring the environment. The exploration is interrupted after a certain amount of time or when the volume of the explored area exceeds a given threshold. The algorithm shows good performance both in simulations and in the FlyART arena. Referring to the experimental results consider Figure 4.11.

As shown in the Figure, even if the algorithm is able to complete the exploration of all the volume of the arena while avoiding obstacles, the output trajectory often makes the drone stay in the already explored area. Indeed, the algorithm is strongly dependent on the λ parameter of equation (1.45). This explains also the recent developments highlighted in

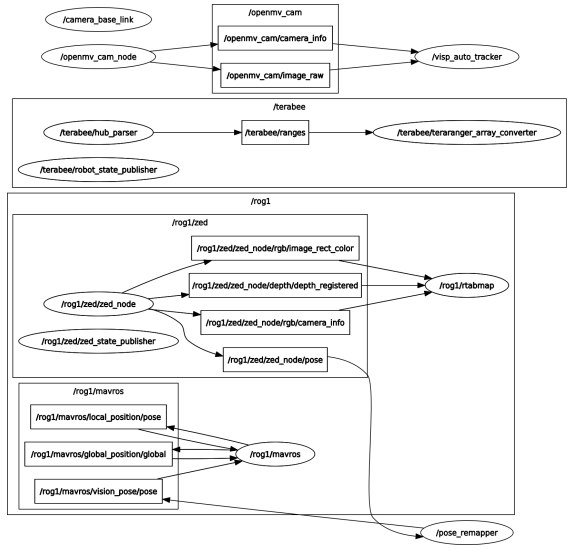


Figure 4.10: *ROS computation graph for ROG-1 navigation*

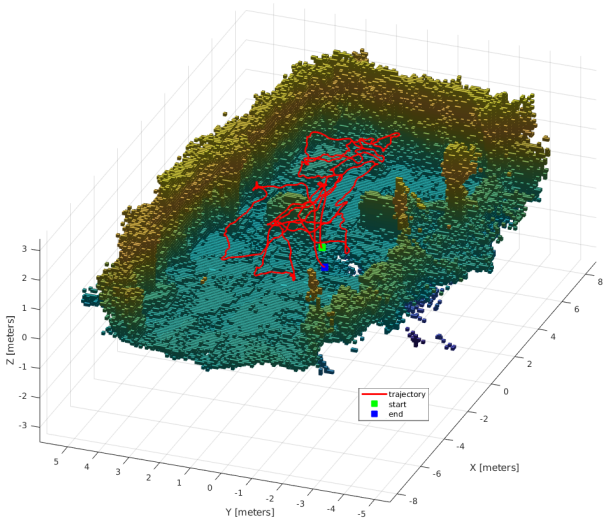


Figure 4.11: *Exploration trajectory and resulting Octomap during experimental activities in FlyART*

Section 1.3.5. Note that similar results were obtained also in the LDC competition arena.

The second phase started with the drone performing a path planned using a 3D A* algorithm on the retrieved Octomap. The path goals were a subset of the QR markers identified during the exploration phase. However, due to possible inaccuracies in the map and/or in the tracking of the planned path, a local planner was designed to avoid potential obstacles.

The implemented algorithm is a version of VFH [123, 242, 228] (see Section 1.3.4). Even if a 3D extension has been proposed in literature [243], a 2D implementation is used in this work. This is due to the nature of the sensor measurements we have available on the UAV. Indeed, it takes as input the point cloud produced by the 8 range finders belonging to the TeraRanger Tower EVO, which all belong to a plane. The range readings are collected in *histogram grid*, *i.e.*, a map grid of the environment. Each cell of this Cartesian histogram is incremented for each range reading, creating a probability distribution with a small computational overhead.

As the vehicle moves, only a portion of the histogram grid is considered. In particular, a window of fixed size, of which the robot occupies the central position, is considered. This window is called *active region* and the cells belonging to this area are called *active cells*.

The active region is then transformed into a one dimension *polar histogram*. Each sector of the polar histogram will contain a value that represents the polar obstacle density in the direction of that sector.

Based on the polar histogram values, each opening is considered in the selection process and associated to a direction. For narrow openings only the central direction is considered, while for large openings three directions are considered: toward the left part of the opening, toward the right part and in the central direction. To these directions, the current and the previously selected direction are added with appropriate weights. Finally, an optimization problem is solved for selecting the direction of drive of the robot.

Once the direction has been selected, a position set-point at a given distance from the robot, in that direction, is chosen. A look-ahead verification has been also added as in VFH* [244]. A tree with *a priori* selected depth is produced, where the leaves represent the future waypoints start-

ing from the actual robot position (root). However, differently from VFH* [228], which searches the tree using A*, a simple Breadth-First Search (BFS) is employed.

The performance of the algorithm is shown here with a very simple simulation example. A trajectory has been planned for the UAV, passing through an obstacle. The set-points computed by the local planner, and belonging to a collision-free path, are shown in Figure 4.12.

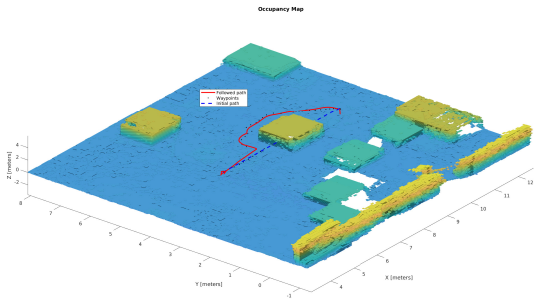


Figure 4.12: *Collision avoidance simulation using a version of VFH**

Note that similar results have been obtained also in experimental activities in the FlyART arena.

For what concerns the landing procedure, once the landing pad position $\tilde{p}_{\mathcal{M}_i}$ has been reached, a visual feedback is employed. Refer to Figure 4.13 for the block diagram of the adopted landing procedure. The initial position set-point $\tilde{p}_{\mathcal{M}_i}$ is modified with the output of a PI controller p_{cam}. This controller takes as input the error between the drone and the center of the marker, e_{cam}. This measurement is actually z_c, obtained exploiting the QR ViSP tracker, opportunely transformed in the UAV body frame. Lastly, $F(s)$, which represents the complementary sensitivity of the position control loop, will drive the drone to the desired position. This control law is applied first for aligning the drone at the QR marker, keeping its flight altitude, and then a descent trajectory is commanded, while keeping its position in the plane.

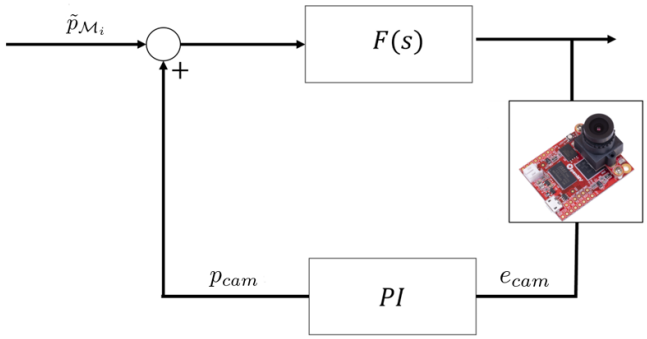


Figure 4.13: *Landing controller architecture in LDC first edition*

4.2.4 Final considerations

As far as the results of the competition were concerned, on the first day, the UAV autonomously explored about 60% of the competition field, finding 7 out of the 10 QR markers placed on the field. On the second day, 4 out of 5 valid landings were performed, of which 3 consecutive, leading to the best result obtained and the win for our team. Despite the achievements, some problems were evident:

- The major problem was related to the altitude estimation. The visual odometry showed a very strong drift and the SLAM algorithm was not able to compensate for it, leading to considerable variations in the altitude kept by the drone.

- While the local planner was successful for collision avoidance, it showed some problems in cluttered environments. The local nature of the algorithm made the vehicle, sometimes, unable to reach the goal, starting to visit the same place over and over. In addition to this, sub-optimal paths, in terms of distance traveled, were often selected.

4.3 Second edition

The second edition took place in Turin on 28-30 September 2021, in the indoor LDC arena (see Section 2.2.2). The contest was composed of four rounds over three days. In each round the drone had to autonomously take off and explore the environment searching for a specific ground robot, which in turn carried information about the task the drone had to perform. It is worth pointing out that, this time, some initial knowledge of the field was given to the teams in the form of a map, with associated East-North reference frame, as shown in Figure 4.14.

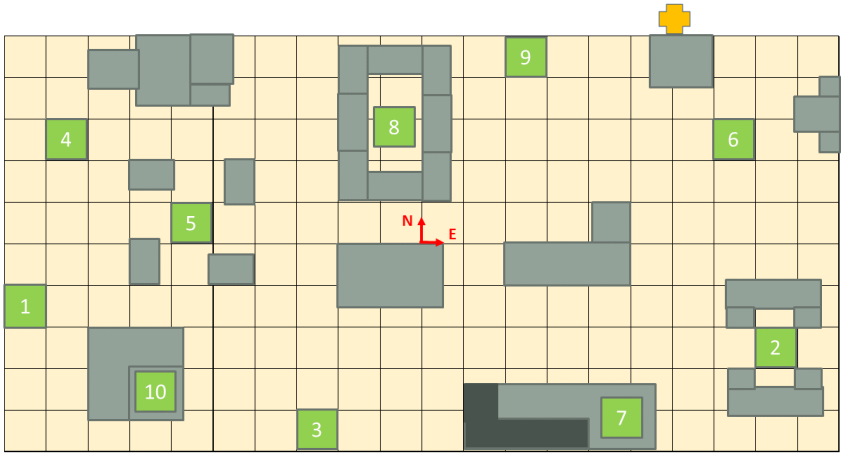


Figure 4.14: *Initial knowledge of the environment in LDC second edition*

The heights of the obstacles in the map were also known. The numbered squares on the map represented the landing pads, which were $1\,\mathrm{m}\times1\,\mathrm{m}$ cyan squares each containing a $50\,\mathrm{cm}\times50\,\mathrm{cm}$ ArUco marker (with corresponding identifier).

Three ground robots (in the form of vacuum cleaners), each identified by a unique $19\,\mathrm{cm}\times19\,\mathrm{cm}$ ArUco marker (ranging from 21 to 26), were moving randomly in unknown confined regions on the field. The information provided by the robots were given in the form of a string of 10 digits from 0 to 9. The i-th number indicated the reward associated with a valid landing on the landing pad identified by ArUco number i. The ground

robot with the available information and an example of landing pad are shown in Figure 4.15.

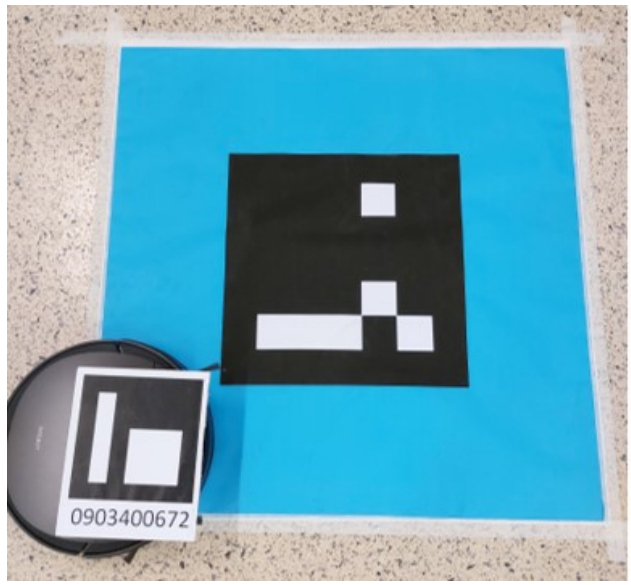


Figure 4.15: *The ground robot and landing pad*

Two ground robots, whose identification ArUco code numbers were communicated to the team before starting the round, were collaborative agents. The remaining one was the *intruder* that carried the exact rewards associated to each landing. The drone exploration was assisted by a fixed pan–tilt–zoom (PTZ) camera placed in correspondence of the cross icon in Figure 4.14, which could be controlled by the teams. After having found the *intruder* robot, the drone had to send a visible picture of the rewards to the team's ground control station. This action had to be executed within 30 minutes from the start of the round. At this point, the sequence of landing spots to maximize the reward had to be selected. No stringent requirements for the computation of the sequence were given by the rules. The sequence could be either selected by the team based on their judgment or could be computed through an optimization problem. After having determined the sequence, the drone had to reach and land on top of the pads in the selected order. The landing was considered valid if all

the UAV contact points with the ground were inside the $1\,\mathrm{m} \times 1\,\mathrm{m}$ square. The points corresponding to valid landings were summed up for obtaining the round result. The team obtaining the highest amount of points in the three (out of four) best rounds was proclaimed the winner. It is worth pointing out that, in each round, the UAV takeoff position was represented by a different landing pad, communicated to the teams before the start of the run. The reward string and the ground robots' region of motion varied across rounds.

4.3.1 Solution overview

The ROG-2 (see Section 2.2.3) platform was employed in this competition. This time, the ground control station was not only used for receiving and displaying telemetry information, but also used for the teleoperation of the fixed PTZ camera and for sending the sequence of landing pads to be visited by the drone.

For state estimation purposes, the same solution presented in the first edition was adopted. However, in order to enhance the accuracy of the altitude estimation, in this competition edition, a range finder was employed as an altimeter. A localization algorithm was custom developed for computing an estimate of the drone position in the *map* frame $\mathcal{M}$. To reach this goal, the known positions of the ArUcos in the world map (Figure 4.14) and the downward-looking monocular camera images were exploited.

The map produced using the ZED point cloud, in the form of an Octomap, was employed for planning collision-free paths and trajectories to be fed to the PX4 drone controller.

The planner module, running online and onboard the robot, was composed by a Djikstra's global planner and an A* local planner. The global planner was meant for computing a list of intermediate waypoints for reaching the goal using the *a priori* information on the environment. The local planner, as opposed to the first competition, relied only on visual information, without the need of additional range finders.

Note also that an additional forward-looking monocular camera was equipped to search for ground robots. This camera utilized different image resolutions compared to the stereo camera.

In this framework, a decision making algorithm was custom developed for identifying waypoints to be reached in order to maximize the probability of finding the ground robots. The algorithm had access to information coming from both the monocular cameras available onboard and on the fixed PTZ camera.

4.3.2 Navigation

For what concerns the localization, a Kalman Filter (KF), which fuses information coming from drone's odometry, monocular cameras images and laser altimeter, has been implemented. The state of the filter is the bias term $x = b_{\mathcal{MO}} = [b_x, b_y, b_z]^\top$. As already in the EKF-SLAM of Section 4.2, we have assumed the evolution of this bias term as a random walk process, the motion model is thus:

$$\dot{x} = \dot{b}_{\mathcal{MO}} = \eta_w, \tag{4.12}$$

where η_w is a white Gaussian noise with Power Spectral Density (PSD) $S_w(\omega) = W$. We write equation (4.12) in discrete time state space form as:

$$x_k = Fx_{k-1} + w_{k-1}, \tag{4.13}$$

where F is equal to the identity matrix I and $w_k \sim \mathcal{N}(0, Q)$ is a Gaussian random vector with covariance Q.

The images from the forward-looking and downward-looking cameras are analyzed through the ArUco marker detector, which will be presented in details in Section 5.2. Once a marker related to a landing pad is detected, its position with respect to the camera, written in the camera frame $\mathcal{C}$, is computed in a way similar to equation (4.9), through (we omit index k for brevity):

$$y_c = t_{\mathcal{CT}} = R_{\mathcal{CB}}(R_{\mathcal{OB}}^\top(t_{\mathcal{MT}} - (b_{\mathcal{MO}} + p_{\mathcal{OB}}))) + t_{\mathcal{CB}} + v_c. \tag{4.14}$$

where v_c is a zero mean Gaussian white noise with covariance R_c: $v_c \sim \mathcal{N}(0, R_c)$ and $t_{\mathcal{MT}}$ represents the ArUco marker position in the *map* frame, retrieved by the approximate world map (Figure 4.14). Note that for retrieving equation (4.14) we have made use of Assumption 1.

In order to increase the altitude estimation accuracy, the laser altimeter measurements are also used. The measurement model is:

$$y_l = b_z + p_z + v_l, \tag{4.15}$$

where $v_l \sim \mathcal{N}(0, R_l)$ is a zero mean white Gaussian noise with covariance R_l and p_z is the third component of the vector $p_{\mathcal{OB}} = [p_x, p_y, p_z]^\top$. Again the index k has been omitted. However, this model does not take into account the presence of obstacles on the field. Indeed, when flying over obstacles, the measured distance does not correspond to the UAV altitude. To overcome this issue, a measurement outliers detector has been applied to interrupt the fusion process whenever the drone flies over an obstacle. Note that a similar approach has been also employed in the attempt of reconstructing the scale of monocular vision using range finder information in [73]. Our method was also used for rejecting possible outliers in the ArUco relative pose estimates. In particular, this check is performed through a χ^2-test based on the Mahalanobis distance of the measurement innovation [245].

Writing in state space form the measurement model obtained combining (4.14) and (4.15), we have:

$$y_k = [y_c, y_l]_k^\top = Cx_k + v_k, \tag{4.16}$$

with $v_k = [v_c, v_l]_k^\top$. Furthermore, we group the measurement noise covariance matrices as:

$$R = \begin{bmatrix} R_c & 0 \\ 0 & R_l \end{bmatrix}. \tag{4.17}$$

Writing the filter in the usual prediction-correction form, the following

equations have been employed:

$$\hat{x}_k^- = F\hat{x}_{k-1}^+ \tag{4.18}$$
$$P_k^- = FP_{k-1}^+F^\top + Q \tag{4.19}$$
$$z_k = y_k - H_k\hat{x}_k^- \tag{4.20}$$
$$Z_k = H_kP_k^-H_k^\top + R \tag{4.21}$$
$$K_k = P_k^-H_k^\top Z_k^{-1} \tag{4.22}$$
$$\hat{x}_k^+ = \hat{x}_k^- + K_kz_k \tag{4.23}$$
$$P_k^+ = (I - K_kH_k)P_k^-. \tag{4.24}$$

where P^- and P^+ are the *a priori* and *a posteriori* state estimation error covariance matrices respectively, and $\hat{x}^-$ and $\hat{x}^+$ are the *a priori* and *a posteriori* estimates of the state.

Inliers are validated by checking the Normalized Estimation Error Squared (NEES) after the computation of z_k, namely:

$$z_k^\top Z_k^{-1} z_k \leq \chi_{th}^2, \tag{4.25}$$

with χ_{th}^2 equal to the 0.95 probability quantile of the χ^2 distribution. If the measurement passes the test, we proceed by computing the Kalman gain K_k and by updating the filter state and covariance, otherwise the measurement is discarded.

Compared to the first edition architecture and TF tree (Figure 4.9), the developed localization KF filter is the responsible for publishing the transform between *odom* and *map* in place of RTAB-Map.

Finally, knowing the drone pose in the *map* frame given by the localization and having the ZED point cloud, the mapping pipeline of RTAB-Map is used for producing a 3D map in the form of an Octomap. The overall navigation architecture is depicted in Figure 4.16.

4.3.3 Decision-making

The decision-making module is responsible for managing the high-level objectives of the mission. This module outputs the desired goal waypoints to be sent to the planning and control module. In particular, the waypoints are computed through different approaches based on the output of a state

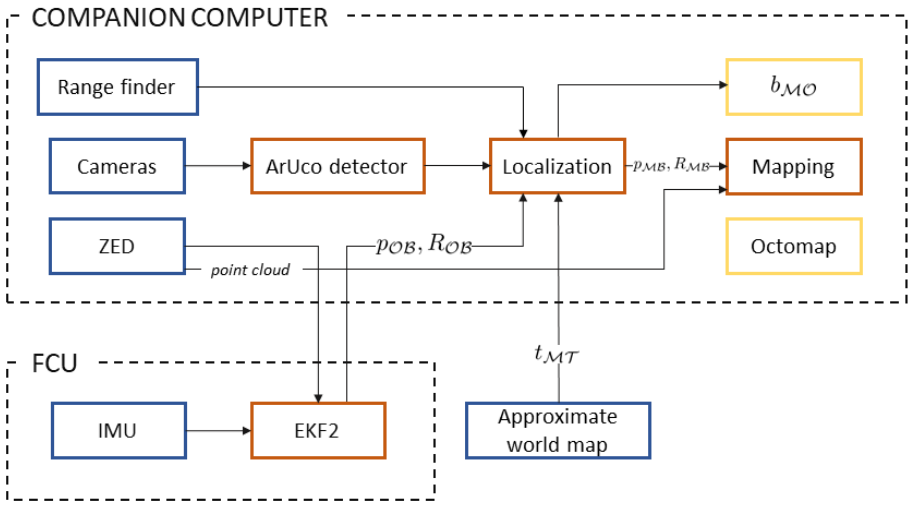


Figure 4.16: *Navigation architecture in LDC second edition*

machine, which manages the different phases of the competition. The workflow of the state machine and associated inputs are shown in Figure 4.17 and described in the following.

When the operator sends the starting signal, the decision-making module provides a takeoff set-point.

After takeoff, the UAV is supposed to find the *intruder* robot. The problem at hand, in general terms, can be formulated as the one of finding an optimal patrolling strategy [246]. Despite the existence of solutions to this problem in similar frameworks [247, 248], our problem can be simplified considering the fact that the ground robots to be monitored are constrained to move in relatively small areas. As a consequence, the drone does not need to visit multiple times the same area for verifying if an *intruder* has appeared. Thus, the problem can be treated as an art gallery problem or coverage planning problem in a known environment [249, 250] (see also Section 1.3.5). In this work, we propose a greedy approach for steering the UAV towards the area of the field where the probability of finding a ground robot is greater. In the state machine of Figure 4.17 this procedure is called *probabilistic exploration*. At each it-

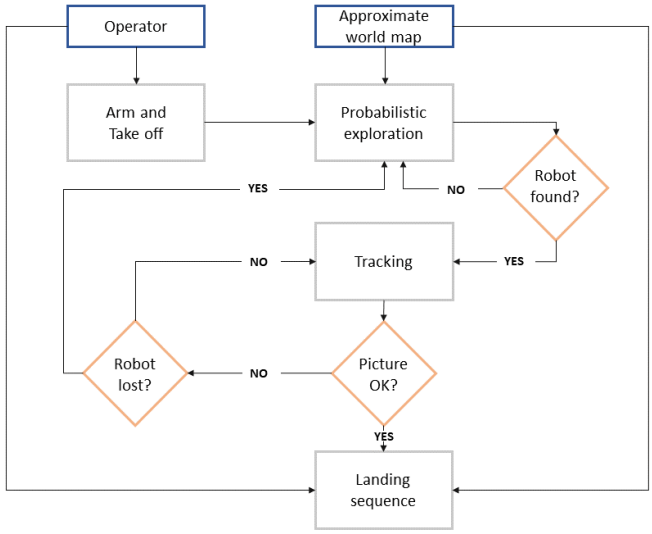


Figure 4.17: *Decision making architecture in LDC second edition*

eration, this algorithm outputs the (x, y) coordinates which maximize the probability of finding the *intruder* robot (the z-coordinate of the generated waypoints will be equal to the drone flight altitude). The probability values are stored in a grid map of $1\,\mathrm{m}$ resolution. This probability map, denoted as P_{map}, is initialized considering the approximate world map of Figure 4.14 and the following assumptions:

Assumption 2. *1) The ground robots will not be positioned on obstacles or too close to them; 2) It will be more likely to find the ground robots in large open spaces.*

Thus, the initial probability values are computed based on the number of free, *i.e.*, with no obstacles, contiguous cells. A graphical representation of the initial map is shown in Figure 4.18.

During the environment exploration, the information coming from all the available cameras, *i.e.*, downward-looking and forward-looking cameras available onboard and the fixed PTZ camera, are used for updating the probability values. In particular, under certain conditions, the probability values of some cells are decreased to a user defined value. As

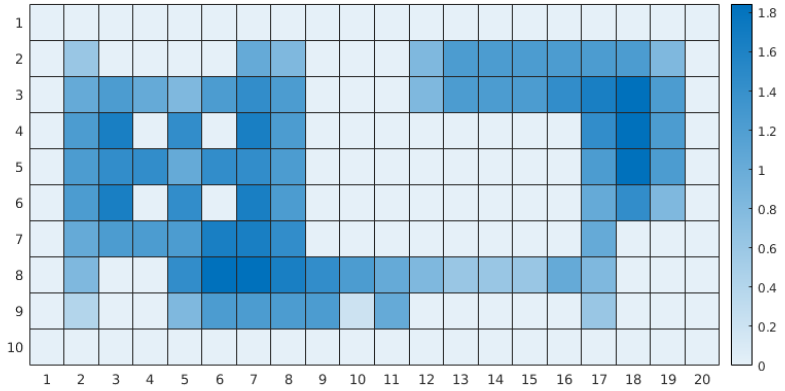


Figure 4.18: *Initial probability map*

a consequence, the probability values of the other cells in the map will increase to have an overall properly defined probability distribution.

Consider now the pseudo-code reported in Algorithm 2. The algorithm takes as input the initial probability map P_{map}.

At each iteration, the images coming from the cameras are processed searching for ArUco markers corresponding to ground robots. If one of these fiducial markers is found, the corresponding code $aruco_{id}$ and position in the camera frame $t_{\mathcal{CT}}$ are retrieved. If the $aruco_{id}$ is the *intruder* one, *i.e.*, not one of the two identification numbers communicated before the start of the round, the probability exploration algorithm outputs the waypoint corresponding to the position occupied by the ground robot in the *map* and terminates (the decision-making module will shift to the *tracking* phase). In this regard, the *Camera2Map* function is employed for finding the position of the ArUCo in the *map* frame, given its position in the camera frame. This can be done by applying the inverse measurement model of equation (4.14), namely:

$$t_{\mathcal{MT}} = (p_{\mathcal{OB}} + b_{\mathcal{MO}}) + R_{\mathcal{OB}} R_{\mathcal{CB}}^{\top} (t_{\mathcal{CT}} - t_{\mathcal{CB}}). \tag{4.26}$$

On the other hand, if the $aruco_{id}$ corresponds to a collaborative ground robot, the probability values of the area surrounding the robot are de-

Algorithm 2 Probabilistic exploration

Input P_{map}
Output waypoint

while *intruder* is not found **do**
  Process image and compute $t_{\mathcal{CT}}$ and $aruco_{id}$
  if not $aruco_{id}$ **then**
    P_{map} = DiscountCameraFovs(P_{map})
    wp = BestWaypoint(P_{map})
    return wp
  else
    if $aruco_{id}$ is *intruder* **then**
      $t_{\mathcal{MT}}$ = Camera2Map($t_{\mathcal{CT}}$)
      wp = $t_{\mathcal{MT}}$
      return wp
    else
      $t_{\mathcal{MT}}$ = Camera2Map($t_{\mathcal{CT}}$)
      P_{map} = DiscountArea(P_{map}, $t_{\mathcal{MT}}$)
      wp = BestWaypoint(P_{map})
      return wp
    end if
  end if
end while

creased (*DiscountArea* function). This update rule is based on the assumption that the robots will not be close to each other.

After the map update, the *BestWaypoint* function retrieves the waypoint corresponding to the cell of maximum probability:

$$\arg\max_{(i,j)} \; p_{map}(i,j). \tag{4.27}$$

In the case of no ArUco markers identified in the images, the *DiscountCameraFovs* function is employed. The discounted cells depend on the camera which has produced the observation. For the downward-looking camera, only the cell occupied by the drone at update time is discounted. This rule has been implemented considering that the resolution of the map is approximately equal to the area covered by the downward-looking camera field of view with the drone flying at an altitude of about $1\,\mathrm{m}$. For the forward and fixed PTZ camera, first, the direction of the camera in the *map* frame is computed and approximated to the nearest 45 degree angle. Then, in the computed direction, the cells on which a robot could be correctly identified, if present, are discounted. These cells range from a minimum value d^-, determined by the field of view of the camera and the orientation of the camera itself, and a maximum value d^+, which corresponds to the maximum distance at which an ArUco marker can be detected (see Figure 4.19).

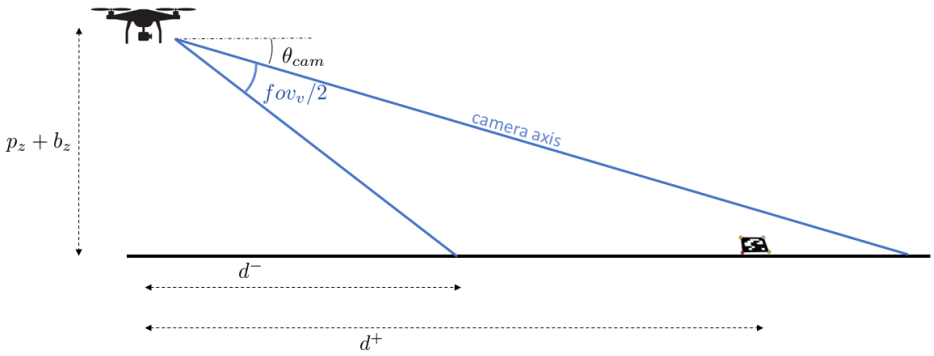


Figure 4.19: *Graphical representation of high oblique aerial field of view*

In particular, d^- is computed as:

$$d^{-} = (p_z + b_z)/\tan(\theta_{cam} + fov_v/2) \tag{4.28}$$

where θ_{cam} and fov_v are respectively the pitch angle with respect to the horizontal plane and the vertical field of view of the considered camera. On the other hand, d^{+} is determined experimentally.

Also in this case, the *BestWaypoint* function is called for computing the output waypoint after the map update.

During the *tracking* state, the decision-making module keeps computing the robot position in the *map* frame according to function *Camera2Map*. This phase ends when:

1. the robot has been lost, *i.e.*, it is outside the field of view of the cameras. In this latter case, the tracking cannot continue since no information about the position of the ground robot can be obtained. Thus, the state machine will resort to the *probabilistic exploration* phase in the neighbourhood of the current position of the drone.

2. the robot is in the field of view and a readable picture of the reward string has been taken. In this case, the operator will decide the landing sequence to follow in order to maximize the reward considering the remaining endurance of the drone and the probability of success of each landing (this could be also computed through an optimization process on the drone available CPU). The output will be the sequence of waypoints corresponding to the selected landing pads as indicated in the initial approximate world map.

Finally, if the robot is in the field of view, but the reward string is not readable, the state machine will remain in the *tracking* phase.

4.3.4 Guidance

The guidance module takes as inputs the waypoints $p^0_{\mathcal{MB}}$ generated by the high-level decision making. An overview of the architecture is shown in Figure 4.20.

In this module, a common cascaded approach composed by a global planner and a local planner, is utilized. The global planner generates a path from the current position of the drone to the given waypoint. To

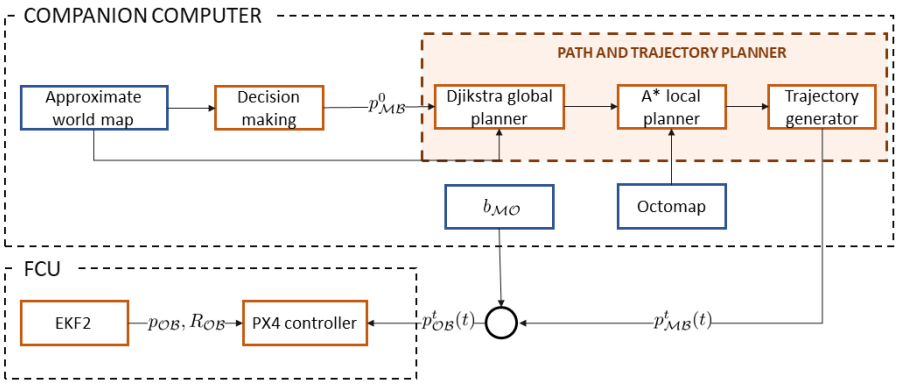


Figure 4.20: *Planning architecture in LDC second edition*

this aim, it exploits the knowledge of the approximate world map with a coarse resolution (1 m). The planner is a 2D implementation of Djikstra's shortest path algorithm. Successively, the generated path is pruned for removing the collinear points. The remaining points are, one at a time, used as goals into a local planner. The local planner, based on the A* algorithm, is responsible for computing a collision-free 3D path using the Octomap produced by the navigation module. With respect to the planner used in the first competition, the A* algorithm showed a remarkable improvement in the performance: the possibility of having access to a global Octomap and of planning online allows the planner to quickly find the shortest path, even in cluttered environments.

The planner pseudocode is presented in Algorithm 3. The A* algorithm, using the *ApplyAstar* function, computes a path from the starting point, *i.e.*, the drone current position in the *map* reference frame (retrieved through the *GetDronePos* function), to the next point of the global planner path. The most recent obstacles configuration in the form of an Octomap, retrieved through the *GetObstacles* function, is used by the planner. Note that a 3D implementation of A* is employed. It expands, for each node, the 26 contiguous cells in the 3D space. At each planner execution, the path is checked against newly emergent obstacles. It is worth mentioning that collision checking is conducted inflating the robot dimensions by a

safety radius (*IsValid* function). If the previously computed path is still valid, it is further processed and sent to the drone controllers (as presented in the following of this Section). Otherwise, a re-planning procedure is started. In addition to this, if the goal is one of the invalid points, a new target point, in the neighbourhood of the previous one, is selected (*PickRandTarget* function). Finally, in the case of difficulties in computing an admissible path, the algorithm tries to decrease the safety radius up to a saturation level through the *ShrinkRadius* function.

Algorithm 3 Local planner

Input $goal$
Output $path$

while $goal$ is not reached **do**
  $obst$ = GetObstacles()
  if IsValid($path_{prev}$, $obst$) **then**
    $path$ = $path_{prev}$
    return $path$
  else
    if $goal$ is in $obst$ **then**
      while $goal$ is not in $obst$ **do**
        $goal$ = PickRandTarget($goal$)
      end while
    end if
    $start$ = GetDronePos()
    $path$ = ApplyAstar($goal$, $start$, $obst$)
    if not $path$ **then**
      ShrinkRadius()
      continue
    else
      return $path$
    end if
  end if
  $path_{prev}$ = $path$
end while

The resulting path from the local planner should be, then, turned into a smooth trajectory, $p^t_{\mathcal{OB}}(t)$, before sending it to the drone controller. In particular, the starting and ending points of the local path are constrained

to be both in hover, *i.e.*, with zero velocity. Considering the drone as a simple point-mass system, the time-optimal trajectory can be computed in closed form, and its result is a bang-bang acceleration trajectory [84]. We additionally impose a constraint on the maximum velocity, resulting in a trapezoidal velocity profile. Note that the dynamics of the system is neglected since the system is limited in the attainable velocities and acceleration by the perception pipeline constraints. However, for further trajectory models please refer to [153].

Going in details, we have assumed that the various kinematic and dynamic constraints are reflected in the constraints on the maximum velocity and accelerations of the curvilinear abscissa $s(t)$. The constraints we refer to are:

$$-\dot{s}_M \leq \dot{s}(t) \leq \dot{s}_M \text{ with } \dot{s}_M > 0, \tag{4.29}$$

and

$$-\ddot{s}_M^- \leq \ddot{s}(t) \leq \ddot{s}_M^+ \text{ with } \ddot{s}_M^- > 0 \; \ddot{s}_M^+ > 0, \tag{4.30}$$

The two values $\ddot{s}_M^-$ and $\ddot{s}_M^+$, indicating respectively maximum acceleration and maximum deceleration, can be different. For the considerations made above, we have considered the following constraints:

$$\begin{aligned}
s(t_0) &= s_0 \\
s(t_f) &= s_f \\
\dot{s}(t_0) &= \dot{s}(t_f) = 0 \\
\ddot{s}(t_0^-) &= \ddot{s}(t_f^+) = 0 \\
\ddot{s}(t_0^+) &= \ddot{s}_M^+ \\
\ddot{s}(t_f^-) &= \ddot{s}_M^-,
\end{aligned}$$

and we have defined the three intervals:

$$\begin{aligned}
\mathcal{I}_1 &:= \{t : t_0 \leq t < t_1\} = [t_0, t_1) \\
\mathcal{I}_2 &:= \{t : t_1 \leq t < t_2\} = [t_1, t_2) \\
\mathcal{I}_3 &:= \{t : t_2 \leq t \leq t_f\} = [t_2, t_f].
\end{aligned}$$

Then, the acceleration results:

$$\ddot{s}(t) = \begin{cases} \ddot{s}_M^+ & t \in \mathcal{I}_1 \\ 0 & t \in \mathcal{I}_2 \\ -\ddot{s}_M^- & t \in \mathcal{I}_3 \end{cases}, \tag{4.31}$$

while for the velocity we have:

$$\dot{s}(t) = \begin{cases} \ddot{s}_M^+(t - t_0) + \dot{s}_0 & t \in \mathcal{I}_1 \\ \dot{s}_M & t \in \mathcal{I}_2 \\ \dot{s}_M - \ddot{s}_M^-(t - t_2) & t \in \mathcal{I}_3 \end{cases}, \tag{4.32}$$

and, finally, for the position:

$$s(t) = \begin{cases} \frac{1}{2}\ddot{s}_M^+(t - t_0)^2 + \dot{s}_0(t - t_0) + s_0 & t \in \mathcal{I}_1 \\ \dot{s}_M(t - t1) + s_1 & t \in \mathcal{I}_2 \\ -\frac{1}{2}\ddot{s}_M^-(t - t_2)^2 + \dot{s}_M(t - t_2) + s_2 & t \in \mathcal{I}_3 \end{cases}. \tag{4.33}$$

Imposing the previously defined constraints with $s_0 = 0$ (for simplicity), we can find the time intervals:

$$t_1 - t_0 = \frac{\dot{s}_M}{\ddot{s}_M^+}, \tag{4.34}$$

$$t_2 - t_1 = \frac{s_f}{\dot{s}_M} - \frac{1}{2}\left(\frac{\dot{s}_M}{\ddot{s}_M^+} + \frac{\dot{s}_M}{\ddot{s}_M^-}\right), \tag{4.35}$$

and

$$t_f - t_2 = \frac{\dot{s}_M}{\ddot{s}_M^-}. \tag{4.36}$$

Having the evolution in time of the $s(t)$, $\dot{s}(t)$ and $\ddot{s}(t)$, we only need to write it in Cartesian coordinates.

It is worth noting that the generated trajectory is written in the *map* frame, while the UAV state estimate is in the *odom* frame. Consequently, the trajectory is transformed into the correct frame by exploiting the state of the localization filter $b_{\mathcal{MO}}$.

Note also that the yaw angle is commanded to have the first body axis, at each instant in time, oriented as the velocity vector. This is done for coping with the possibility of newly emergent obstacles during drone motion.

If during the execution of the trajectory the planner finds that the related path is no longer valid, the drone stops in its current position. Consequently, a collision-free path is computed in the updated situation. Thus, the controller will start tracking the newly produced trajectory.

Finally, the generated position set-points are sent to the UAV FCU (see Sections 2.2.3 and 1.4). For the landing execution, a procedure similar to the one of the first edition is employed. The only difference resides in the fact that, this time, the tracking is also employed while descending.

4.3.5 Simulation results

In simulation, particular attention has been spent for dealing with aspects related to the localization. In fact, due to the reduced dimensions of the laboratory flight test area compared to the competition field, simulation is the only way to test factors coming into play in the long-run. In addition to this, we focus on the z-axis, which has been highlighted as one of the most critical factors in the performance of the UAV in the first competition. Consider Figure 4.21, in which the results of the localization are shown. The odometry has been deliberately simulated with high noise values in order to test the localization robustness to drift. Nevertheless, the localization is able to recover the "true" altitude, even rejecting measurements corresponding to the presence of obstacles under the drone. Note that the sum of the bias $b_{\mathcal{MO}}$, state of the filter presented in Section 4.3.2, and the position estimated by the EKF2 $p_{\mathcal{OB}}$ is represented as the output of the localization process.

As already mentioned, the localization output is used for retrieving the transformation between the *map* and *odom* reference frames. This information is then used for transforming the set-points in the *odom* reference frame. The result of this procedure is shown in Figure 4.22. The set-point is clearly changing for compensating the drift of the altitude estimate. The drone is, thus, able to keep the altitude constant, solving one of the problems which arose in the first competition.

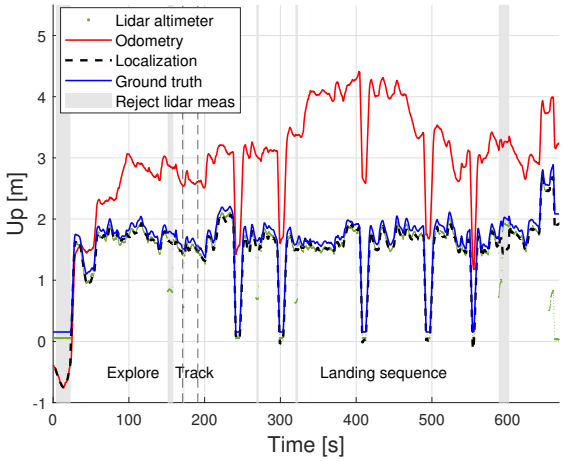


Figure 4.21: *Comparison between odometry and localization (Up component) in simulation*

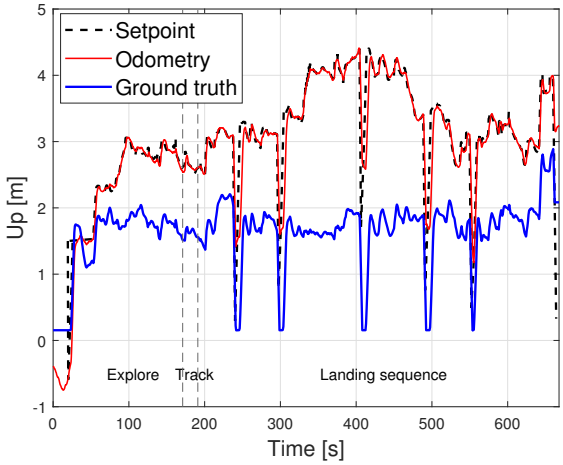


Figure 4.22: *Set-point (Up component) sent to the controller. Comparison with odometry (VIO) and ground truth*

Note that a video of the simulation is available online[3].

4.3.6 Experimental results

In this Section, the experimental results are shown and discussed. The presented experiments have been conducted in both the FlyART and in the LDC arena (see Sections 2.2.1 and 2.2.2 respectively) during the competition.

FlyART experiment

The experiment can be split up in its main phases according to the decision-making algorithm. After takeoff, the UAV conducted an exploration phase of the laboratory environment, looking for the *intruder* robot. After having found it with the forward-looking camera, the tracking phase begun. The UAV was driven towards the robot and started taking pictures of it. After that, the landing sequence was selected while the UAV held its position. Finally, the drone reached and landed on each of the selected pads according to the given sequence. The video of the entire experiment is available online[4].

For what concerns the navigation performance, the East and North localization position components are shown in Figure 4.23 and Figure 4.24 respectively. In these plots, the localization is compared with the visual odometry output, which is in turn affected by drift. In order to be in the same reference of the localization position, the odometry curve has been shifted by an offset, namely the drone initial position in the *map* frame, which corresponds to the position of one of the ArUco landing pads available in the laboratory. At the same time, taking off from one of the ArUco markers, the localization curve shows a sharp transient toward the value of the ground truth.

For the Up direction, the altitude estimation results are plotted in Figure 4.25. The localization algorithm shows good performance, even if the visual odometry already started a remarkable drift in the short time frame of the experiment.

[3]Visit `https://www.youtube.com/watch?v=N7o6_CEzCn4&t=33s`
[4]Visit `https://www.youtube.com/watch?v=lGsVxvEKR4A`

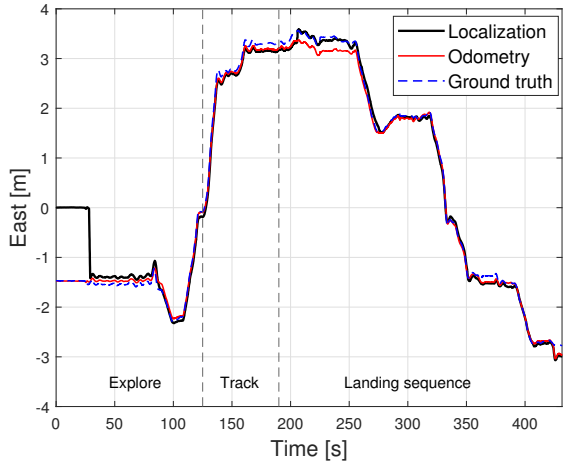


Figure 4.23: *Comparison between odometry and localization (East axis)*

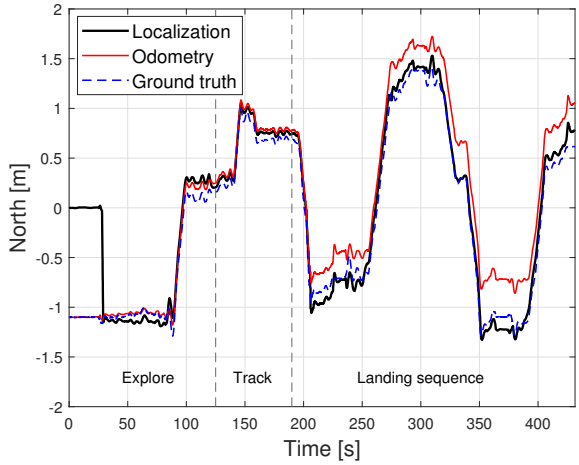


Figure 4.24: *Comparison between odometry and localization (North axis)*

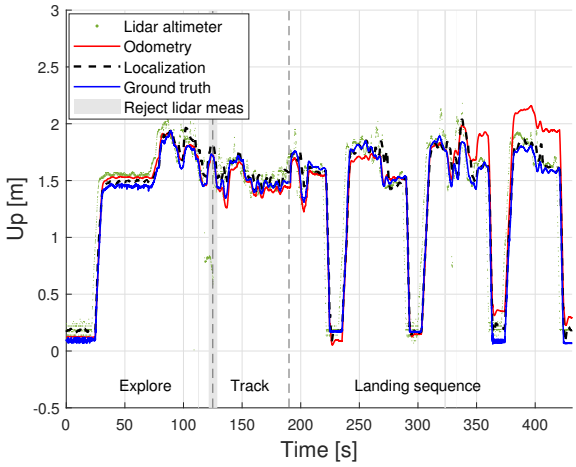


Figure 4.25: *Comparison between odometry and localization (Up component). Added patches in which the range finder measurements are rejected*

As far as the performance of the vision-based landing is concerned, all the four landings can be considered valid according to the competition rules. The ground truth path of the drone during the experiment with the associated 4 landings are shown in Figure 4.26.

Competition experiment

This experiment corresponds to the last round of the LDC second edition. The video of this round is available online[5].

The localization results are shown for what concerns the East and North position components respectively in Figure 4.27 and Figure 4.28. Also in this case, the localization is compared with the visual odometry output. The odometry curve has been shifted by the drone initial position in the *map* frame corresponding to the ArUco marker 3 in Figure 4.14, *i.e.*, $x = 2.5\,\mathrm{m}$, $y = -4.5\,\mathrm{m}$.

Furthermore, in the same plots, the drone position obtained from the measurements coming from the forward and downward-looking monocular cameras are plotted and considered as the best approximation of the

[5]Visit `https://www.youtube.com/watch?v=zfQVKgepwNE`

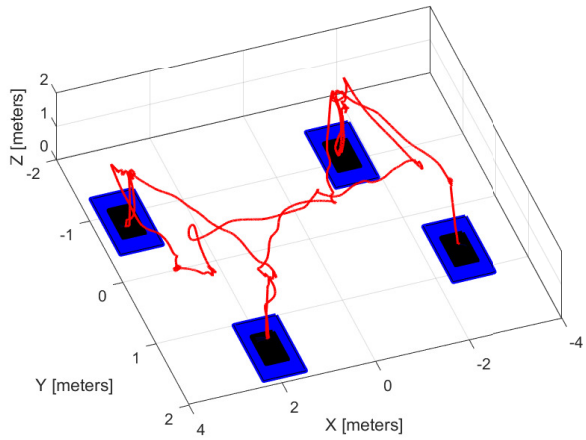


Figure 4.26: *Ground truth path and landings*

ground truth. To do this, the the ArUco marker detection outputs are used to retrieve the drone position in the *map* frame through the inverse measurement model of equation 4.26.

Again note how, immediately after taking off from the starting pad, the localization algorithm estimates the initial position correctly.

Passing to the navigation in the Up direction, the altitude estimation results are plotted in Figure 4.29. The visual odometry shows a remarkable drift, which again justifies the additional use of the laser altimeter. On the other hand, the localization algorithm shows good performance. In fact, it has correctly rejected the laser measurements corresponding to obstacles (as shown by the patches in the plot). In this plot, also the sequence of the six executed landings can be appreciated. Note that the third and the last one have been executed respectively on ArUco 7 and 10, which are placed over obstacles of height $0.7\,\mathrm{m}$ and $2\,\mathrm{m}$.

Similarly to the laboratory experiment, the mission has been managed by the decision-making module. For the first $200\,\mathrm{s}$ the UAV has conducted an exploration phase for finding the *intruder* robot. Having found it with the forward-looking camera, the tracking phase begun at about $400\,\mathrm{s}$. The UAV is driven towards the robot and takes pictures of it. After that, we

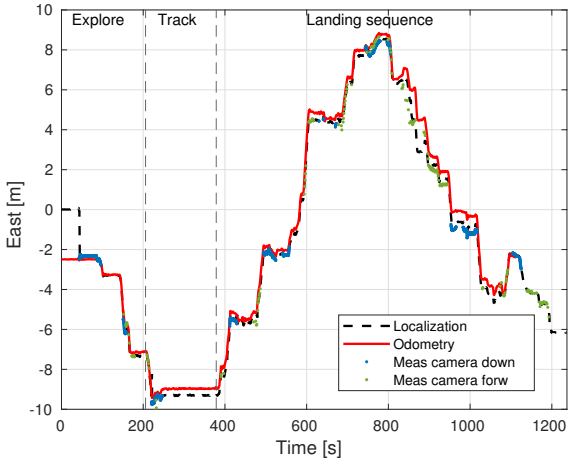


Figure 4.27: *Comparison between odometry and localization during LDC second competition (East axis)*

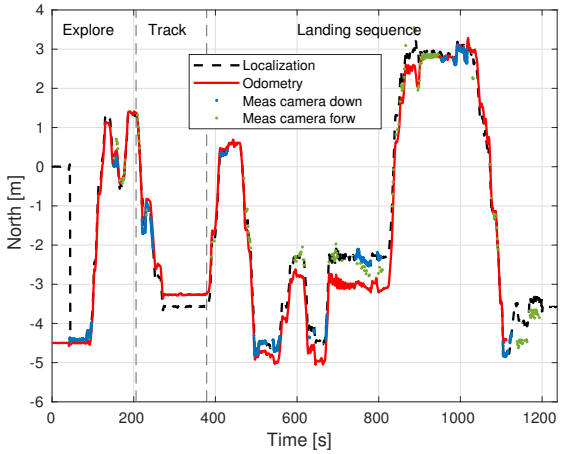


Figure 4.28: *Comparison between odometry and localization during LDC second competition (North axis)*

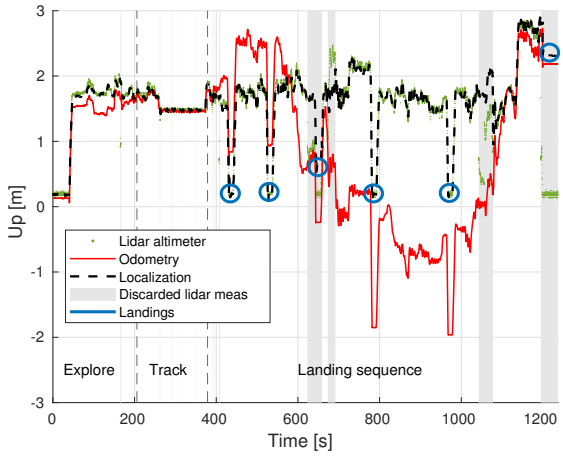


Figure 4.29: *Comparison between odometry and localization during LDC second competition (Up axis). Added patches in which the range finder measurements are rejected.*

can see the drone holding the position while the team was selecting the landing sequence to be performed in the last phase. Finally, the drone reached and landed on the selected pads in the given sequence.

The overall mission lasted 20 minutes and the drone travelled about $100\,\mathrm{m}$. In Figure 4.30 we can see the point cloud generated by the drone during the mission and the localization position, which represents the most reliable estimate of the UAV state we have.

4.3.7 Final considerations

For what concerns the competition results, the UAV autonomously completed three of the four rounds of the competition. Our team won the contest scoring 5, 7 and 23 points respectively on the second, third and fourth rounds. Note that the maximum achievable scoring results were respectively 12, 19 and 31.

Despite the achievements some criticalities were still present. It is worth noting that, during the competition, outside the field, spectators could move around to see the experimental platforms during the task exe-

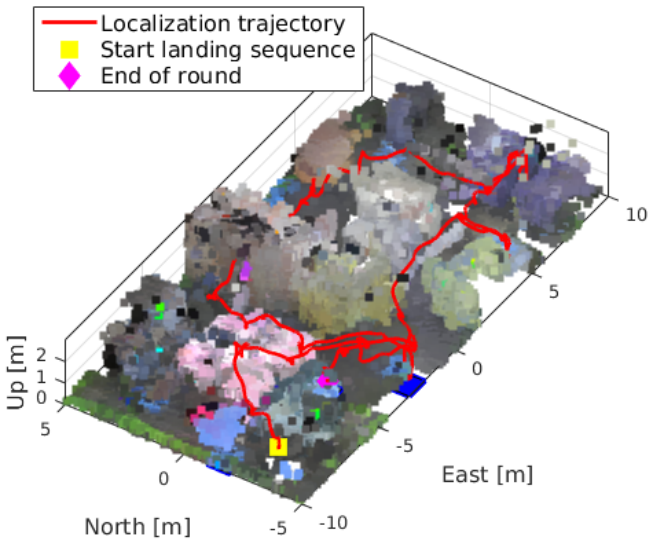


Figure 4.30: *Final point cloud obtained during LDC second competition with traveled path during the landing phase*

cution. This aspect had an impact on the accuracy of the visual odometry employed for state estimation purposes. In addition to this, on one side of the field, a uniform black texture-less wall was present, making one side of the field very difficult to navigate into. So, compared to the laboratory environment, the quality of the flight was greatly affected.

Furthermore, during the competition, some of the landings resulted to be invalid[6]. This behaviour seems still related to the visual odometry performance and was not so noteworthy in laboratory results (see Figure 4.26). The designed vision-based controller allows the drone to accurately align with the underlying ArUCo marker, but, during the descent phase, the controller seems unable to compensate for the drone motion due to state estimation inaccuracies. The performance is even worsened by the fact that the ArUCo marker pose estimate gets interrupted when the marker leaves the camera field of view during drone descent.

During the solution development, many technical challenge have been faced. For example, the choice of the range finder (utilized for altime-

[6]Note that large part of the unattained points in the various rounds were due to inaccurate landings

ter purposes) resulted crucial. On the competition floor, covered by a carpet, many commercial solutions (Terabee TeraRanger $3\,\mathrm{m}$, VL53L1X) showed some limitations: they were either unable to return a measurement or very inaccurate, especially when increasing the flight altitude. The selected Lidar-Lite v3, instead, has shown remarkable accuracy and robustness to changes in flight conditions.

From the software point of view, the range finder altimeter measurements rejection mechanism was a critical aspect in the design of the solution. First, consider that the range finder measurements were not fused directly in the EKF2 to avoid making fast upward/downward moves when flying over a building. An intermediate layer was, thus, inserted to process the measurements before fusing them with stereo camera and IMU data. The idea was to design a state machine capable of estimating and filtering out abrupt changes in the measured altitude. However, this resulted in a strong sensitivity of this process to the rate and quality of the measured altitude. To solve this problem, we started exploiting the altimeter measurement at the localization level, rejecting outliers, *i.e.*, measurements corresponding to the sensing of a building below the drone (as presented in Section 4.3.2). The proposed procedure presents only a limitation about the time spent while hovering over an obstacle. In fact, after rejecting many range measurements, with consequent increase of the estimation error covariance matrix components, a measurement could satisfy the NEES check. If this latter condition is met, the estimated z-position component would decrease and the drone altitude would increase accordingly.

4.4 Third edition

The third edition took place in Turin on 6-7 October 2022, in the indoor LDC arena (see Section 2.2.2). Also this time, some knowledge of the environment was given to the teams. In particular, the 2D map of the field (see Figure 4.31) and the height of the obstacles were known.

However, some additional obstacles were added in unknown positions. They were $3\,\mathrm{m}$ high poles of either $50 \times 50\,cm$ or $30 \times 30\,cm$. A picture of one of them is shown in Figure 4.32. Furthermore, in this competition,

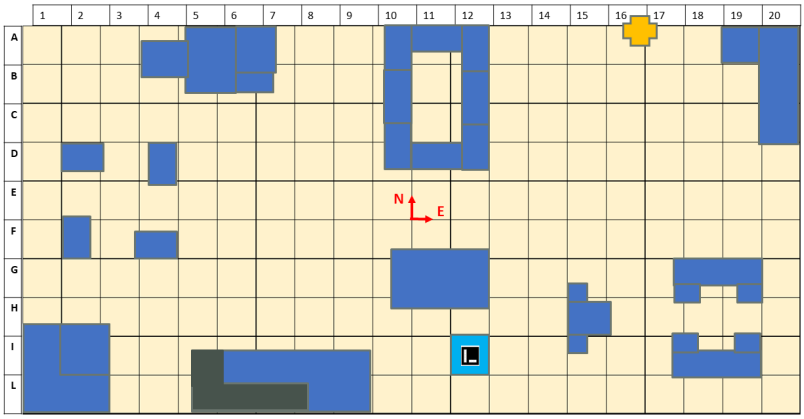


Figure 4.31: *Initial knowledge of the environment in LDC third edition*

only the landing pad identified by ArUco 1 (see Figure 4.15) was present onto the field, and it represented the takeoff position for the drone.

Three rounds were faced over the two days of the competition. In each round the drone had to autonomously take off and explore the environment searching for a ground robot, identified by the ArUco marker 21 (see Figure 4.15). Then, the drone had the aim of tracking the moving ground robot for at least 10 seconds. In the case of accomplishment of this task, the competition judges would give to the team the list of tasks to be performed in order to acquire points for the final scoreboard[7].

The tasks were of two types, *i.e.*, landing and photos, and were accompanied by a letter and a number, *e.g.*, A17. This latter represented the position on the field (see Figure 4.31) where the task should be performed. In the landing case, the drone had to land on the designated cell. Note that the square was delimited by white tape (see Figure 4.32), but without a unique identifier, as it was the case with the ArUco of the second edition (see Section 4.3). The landing was considered valid if all the UAV contact points with the ground were inside the $1\,\mathrm{m}\times1\,\mathrm{m}$ square. In the photos case, the code represented the cell in which to search for find-

[7]Differently from the second edition, in this competition no indications about the tasks to be performed was reported on the ground robot.

Figure 4.32: *Unknown obstacles present onto the field*

ing a small ArUco marker (identifier 15). The marker was placed on the side of a building in the proximity of the cell and a photo of it must be taken, sent to the ground control station, and shown to the competition judges.

Clearly, the accomplishment of each task conferred a given number of points to the team. Moreover, the ability of mapping correctly the unknown poles on the field provided some additional points.

Note that, also this year, the drone exploration was assisted by a fixed pan–tilt–zoom (PTZ) camera placed in correspondence of the cross icon in Figure 4.31, which could be controlled by the teams. It is worth pointing out that, in each round, the poles positions and tasks to be accomplished differed. The team obtaining the highest amount of points in the three rounds was proclaimed the winner.

4.4.1 Solution overview

The ROG-3 (see Section 2.2.3) platform was employed in this edition of the contest. As in the second competition, the ground control station was employed for teleoperation of the fixed PTZ camera, for sending the sequence of tasks to the drone and for receiving and displaying telemetry

information. Furthermore, the ground control station was also used for showing the video feed of the tracking phase and to display the pictures taken by the drone's cameras to the competition judges.

With respect to the second edition, the state estimation pipeline was modified. The laser altimeter information were not anymore exploited at the localization level, but they were pre-processed, along with VO position estimates, before sending them to the EKF2 (see also Appendix A). In addition to this, the additional optical flow available on the platform was used for providing velocity measurements and improving the state estimation performance. Moreover, the localization algorithm was augmented for compensating the lack of most of the ArUCo visual markers that were available on the field in the second edition. In particular, the proposed localization algorithm employed both the information obtained from the single ArUco available on the field and the Adaptive Monte Carlo Localization (AMCL) [251, 252] algorithm, which compared the known map of the field (see Figure 4.31) with the 2D projection of the 3D point cloud produced by the stereo camera.

The mapping task relied upon the RTAB-Map algorithm, which took as input directly the stereo camera information. Part of the produced map was then used for planning collision-free paths. In particular, in this edition, a simplified version of A* was employed. Indeed, it took as input the global map at relevant heights (the UAV flight altitude plus/minus the vertical occupancy of the drone itself) and planned paths at the drone flight altitude. This change led to a great reduction in the memory usage and computational time.

Finally, trajectories were generated as in the second edition and fed to the PX4 drone controller.

4.4.2 Navigation

The navigation architecture is summarized in Figure 4.33. As mentioned in the overview, before remapping the visual odometry in the EKF2, a KF is used for estimating the bias accumulated along the z-direction by the visual odometry using laser altimeter measurements. Similarly to the localization filter of the second edition of LDC, the motion and measurement models of equations (4.12) and (4.15) have been employed. For

what concerns the rejection of outlier, *e.g.*, in the case of the UAV flying over an obstacle, the same approach of equation (4.25) is applied. This helps avoiding possible jumps in the altitude estimate.

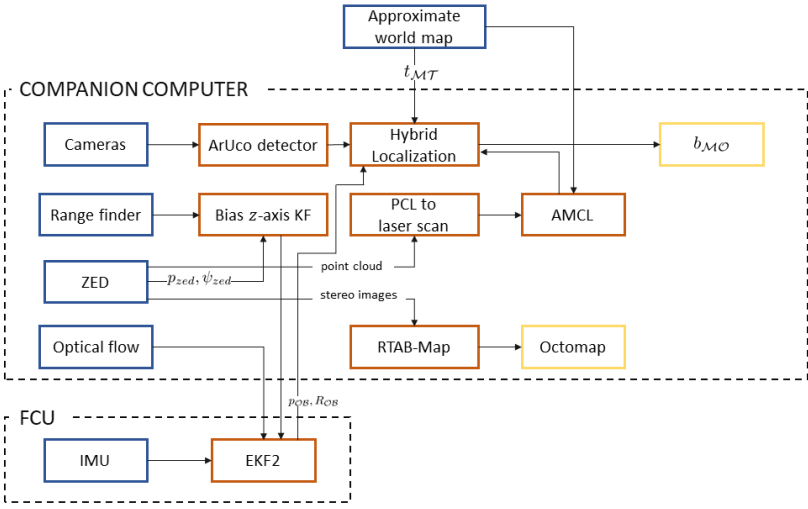


Figure 4.33: *Navigation architecture in LDC third edition*

Then, the bias compensated visual odometry is fused with IMU information using the EKF2 for estimating the pose of the UAV in the *odom* frame: $T_{\mathcal{OB}} = (p_{\mathcal{OB}}, R_{\mathcal{OB}})$.

Moreover, in this edition, optical flow measurements have been added for improving the state estimation pipeline.

Optical flow

In this framework, two optical flow sensors have been tested and compared before equipping them on the platform. Two flights have been performed in the FlyART arena on the ROG-3 platform using both the PX4FLOW [253] and the PMW3901. The measurements coming from the two sensors have been compared with the measurement model employed in the EKF2 (see Appendix A). The distance, needed in the model of equations (A.12) and (A.13), for both the experiments has been retrieved from the motion capture system. Note that no objects were placed

on the floor, meaning that the flight altitude was equal to the distance between drone and the features seen by the camera. The PX4FLOW and PMW3901 results are shown, respectively, in Figures 4.34-**??** and Figures 4.36-4.37.

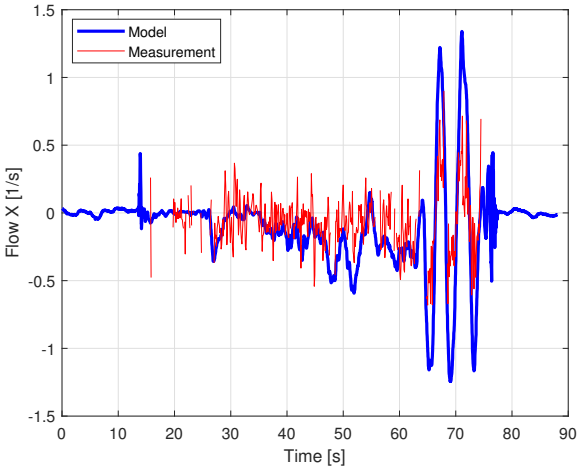


Figure 4.34: *Comparison between measurement model and actual measurements of the PX4FLOW optical flow sensor (x axis)*

As can be seen from the results, even if both measurements are very noisy, the PMW3901 optical flow outperforms the PX4FLOW one.

Localization

A localization algorithm has been designed for exploiting both the *a priori* knowledge on the environment and the presence of ArUco markers, the pseudocode of which is shown in Algorithm 4. The idea for this hybrid localization algorithm has been taken from [254] and it shares many similarities with the localization algorithm presented for the second edition. In particular, the filter estimates the same state, namely $b_{\mathcal{MO}}$, but only focusing on the $x - y$ plane. Then, the state evolves according to the same motion model (4.12). Each time an ArUco of known position is detected[8], the measurement model of equation (4.14) is applied. Each

[8]Note that in the competition environment only ArUco 1 was available

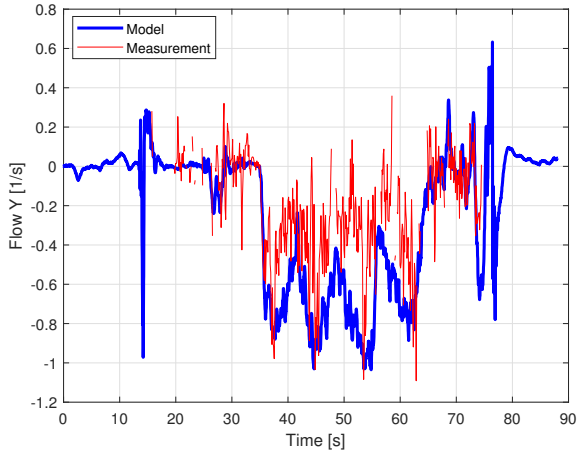


Figure 4.35: *Comparison between measurement model and actual measurements of the PX4FLOW optical flow sensor (y axis)*

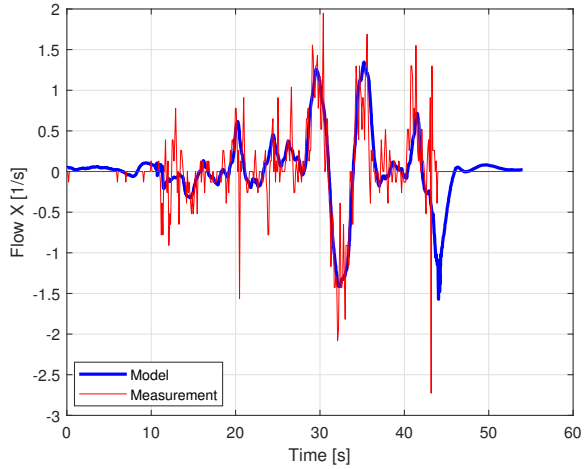


Figure 4.36: *Comparison between measurement model and actual measurements of the PMW3901 optical flow sensor (x axis)*

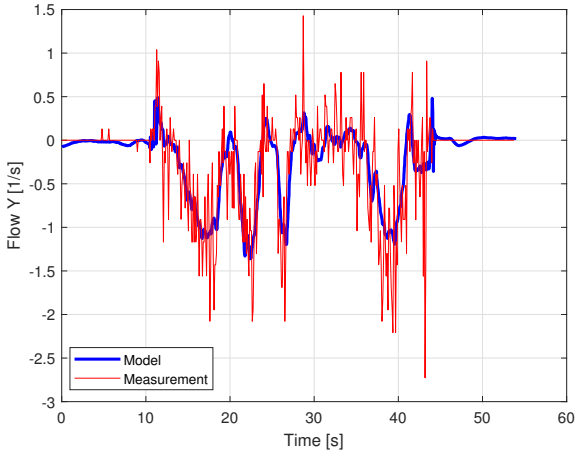


Figure 4.37: *Comparison between measurement model and actual measurements of the PMW3901 optical flow sensor (y axis)*

time such kind of correction is performed, the AMCL particle filter is reset with the last estimated state. At the same time, whenever an AMCL estimates is available, this is used as a measurement z_{amcl} for the hybrid localization filter. The outlier rejection scheme of equation (4.25) is used for dealing with AMCL states with high estimated covariance (EKF_{or}).

Algorithm 4 Hybrid Localization

$\hat{x} \leftarrow x_{init}$
while True **do**
  if z_{aruco} **then**
    $\hat{x} \leftarrow$ Correction $EKF(z_{aruco})$
    $x_{amcl} \leftarrow \hat{x}$ ▷ Reinitialize AMCL filter
  else if z_{amcl} **then**
    $\hat{x} \leftarrow$ Correction $EKF_{or}(z_{amcl})$
  end if
end while

The AMCL particle filter uses as measurements an equivalent laser scan produced by the projection of the 3D stereo camera point cloud onto a plane. The 3D point cloud is opportunely filtered out from outliers

and only points in a range of heights are considered. This is done for comparing the laser scan with the 2D map of Figure 4.38, in turn produced starting from the map represented in Figure 4.31.

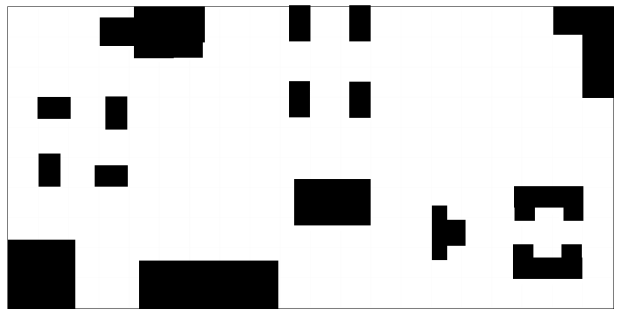


Figure 4.38: *2D map input of the AMCL particle filter*

For what concerns the TF tree, as in the second edition architecture, the hybrid localization is responsible for publishing the TF between *odom* and *map*.

4.4.3 Guidance

The guidance architecture is shown in Figure 4.39. In this case, the decision making architecture is simpler with respect to the previous version one. Indeed, some pre-planned waypoints are selected, based on where the ground robot could be placed (large areas without obstacles). At the start of the mission, after takeoff, the UAV reaches these waypoints, in a given order, until it finds the ground robot. Note that the entire right part of the map (positive East axis according to Figure 4.31) is not explored since it is covered by the fixed PTZ camera. After having found the ground robot, either by the drone or by the fixed camera, the UAV goal is the one of reaching it. Then, once the ground robot has been reached, the UAV tracks it for 10 seconds. For doing so, a similar approach of the one employed in Section 5.2, is exploited. In the case the ground robot is lost during tracking, two possible cases arises. If the ground robot has been seen by one of the cameras, the UAV reaches that position and tracks

it. Otherwise, the UAV recovers the pre-planned coverage path.

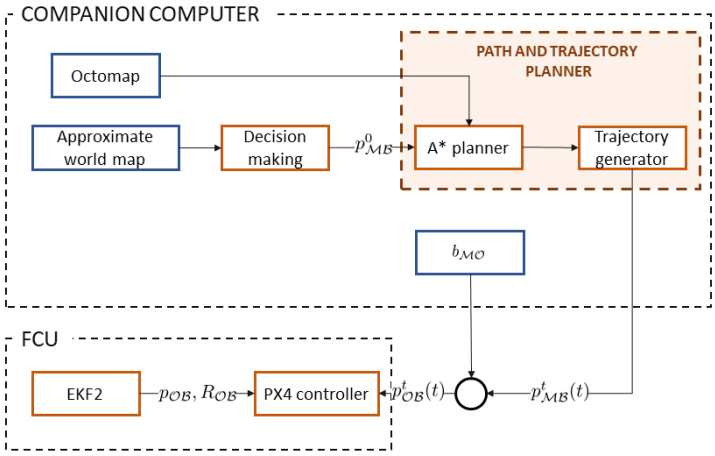


Figure 4.39: *Guidance architecture in LDC third edition*

Each of the obtained waypoints, except the tracking ones[9], are submitted to a 2D A* planner. The planner takes as input the Octomap produced by RTAB-Map, but consider only obstacles placed at an height relevant for the UAV current flight altitude. This procedure has been applied to spare memory occupation and computational time. In addition to this, the A* actually has been proposed in its *Weighted* variant. The cost function is modified according to the following equation:

$$f = w_1 \cdot g + w_2 \cdot h = g + \epsilon h, \tag{4.37}$$

where $\epsilon = w_2/w_1$ is a tunable weight. This approach has been chosen to expand less nodes and, thus, to speed up the planning process. Clearly, by selecting $w_1 = 1$ and $w_2 = 1$, the method degenerates into the regular A* case.

The vision-based landing architecture is similar to the one proposed in Figure 4.13. However, no ArUco markers are available as reference for the alignment before landing. Consequently, a custom computer vision

[9]No verification on the presence of obstacles is performed during tracking. This is done to achieve greater tracking performance and it is possible since the ground robot to track is constrained in a large area without obstacles.

algorithm has been designed for detecting the squares on the floor delimited by the white tape (see for example Figure 4.40). Note that the square must be entirely in the camera field of view and, if more multiple squares are detected, the nearest one to the UAV current position is selected. Thus, the error component e_{cam} will be represented by the difference between the center of the camera and the target center of the rectangle, but transformed in the body frame $\mathcal{B}$.

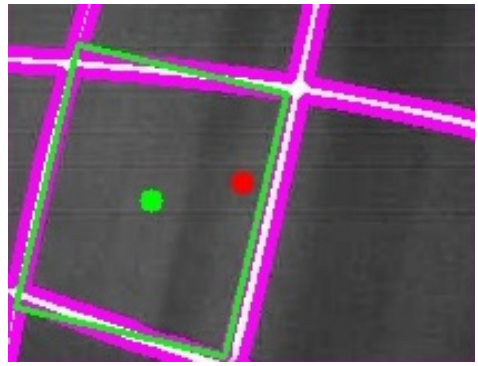


Figure 4.40: *Output of algorithm for detecting landing squares. The red dot represents the optical center of the camera, while the green dot is the center of the target square.*

This information will be exploited only for alignment with the center of the square. Then, the descent is executed providing the FCU with only a velocity command.

4.4.4 Experimental results

Being the competition mostly similar to the second one, in this Section we will focus only on the navigation performance. In particular, we first consider the results obtained for what concerns the compensation of the bias on the z-axis of the visual odometry, which is then fused in the EKF2. Consider Figure 4.41, obtained during a complete flight on the LDC competition field. The visual odometry displays a great drift over the flight. At the same time, the altitude estimate of the EKF2 remains accurate exploiting the laser altimeter measurements. Note also that the laser measurements, corresponding to obstacles below the drone, are correctly re-

jected, not causing a change in the drone flight altitude.

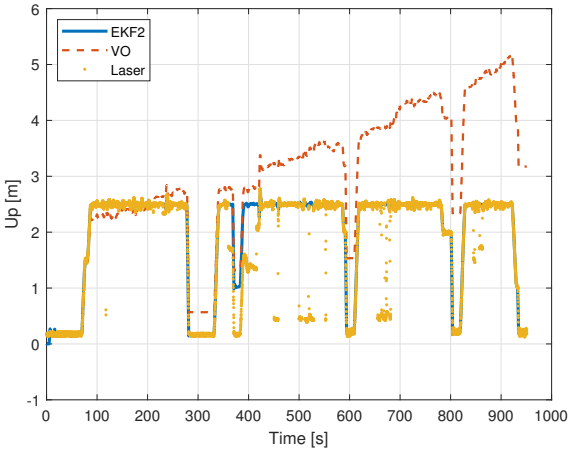


Figure 4.41: *Performance of bias compensated VO (Up component)*

For what concerns the localization architecure, we consider the benefits that can be produced by using the AMCL particle filter over a possible VIO-only solution. In order to not over complicate things, only the AMCL output is shown, not considering its further integration in the localization pipeline, which leads to even greater benefits in terms of accuracy. Due to the lack of any motion capture system on the field, to be used as ground truth, we rely upon retrieving the position of the UAV from the known position of the ArUco markers combined with camera measurements. Hence, we are able to retrieve an accurate indication of the position of the UAV in the map frame $\mathcal{M}$ every time an ArUCo marker is detected by the downward and/or forward cameras. Consider now Figures 4.42 and 4.43, where the East and North components of the position are shown respectively[10]. The map of Figure 4.38 has been used as input to the AMCL algorithm.

It is clearly visible that, in correspondence of the availability of ArUco markers, the AMCL improves over the EKF2 estimate. More formally,

[10]Note that these data have been obtained during another long flight in the LDC arena (different from the one of Figure 4.41). Before this flight, some more ArUco markers were positioned on the field to have some more ground truth points.

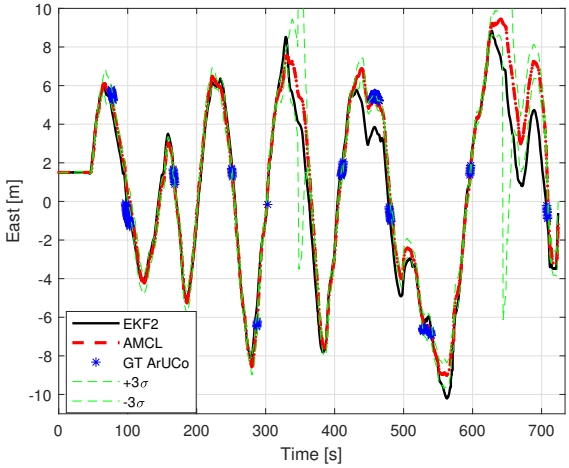


Figure 4.42: *Comparison AMCL with VIO (East component)*

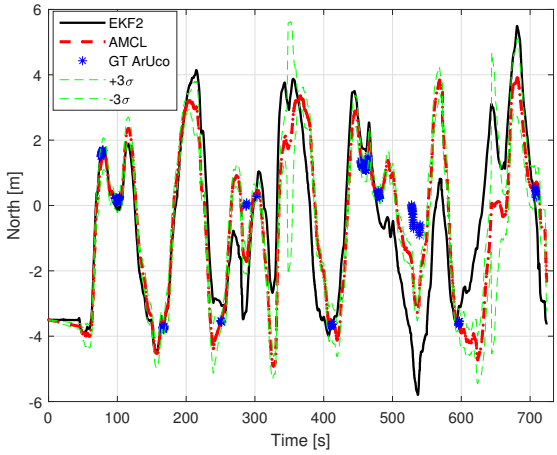


Figure 4.43: *Comparison AMCL with VIO (North component)*

we can compare the RMSE obtained between the AMCL and the ground truth and the EKF2 and the ground truth (clearly only when an ArUco is available). The results are summarized in Table 4.4.

Table 4.4: *RMSE between AMCL and ground truth and EKF2 and ground truth*

RMSE	AMCL [m]	EKF2 [m]
East	0.78	2.01
North	0.68	3.35
Combined EN	1.03	3.90

In view of the integration in the hybrid localization of Algorithm 4, we can also note the 3σ bounds of the AMCL estimate, which will greatly influence the acceptance or rejection of the measurement itself. At least where we have indications about the ground truth position, we can see that the covariance of the estimate has low values, making it suitable for fusion in the hybrid localization filter.

4.4.5 Final considerations

In this third and last edition, the quality of flight resulted greatly improved thanks to the integration of the optical flow and of the laser altimeter directly in the state estimation filter. The landing maneuver, performed using velocity commands, resulted very accurate compared to the second edition. Indeed, leaving the position control loop open makes the VIO irrelevant for the maneuver and beneficial for its execution. Coming to the competition results, the drone performed 11 (over a total of 15) valid tasks (6 valid landings and 5 valid photos) over the three rounds, leading to a win for our team. In addition to this, all the unknown poles were correctly mapped.

4.5 Concluding remarks

In this Chapter, the solutions proposed in the three editions of the *Leonardo Drone Contest* have been presented. First, the aim of the competitions, their rules and objectives have been described. Successively, we presented an overview of the software and hardware architectures. Then, the

guidance and navigation modules were analyzed in detail, providing also simulation and experimental results obtained both in the laboratory and on the competition field. Finally, achievements and technical challenges were discussed.

CHAPTER 5

Autonomous landing

In this Chapter two different problems related to UAV autonomous landing are tackled: autonomous emergency landing and Autonomous Air-to-Air Landing (AAAL). In the following, the two problems are presented and a short introduction about previous works is provided. Moreover, guidance, navigation and control solutions for accomplishing the two tasks are discussed.

5.1 Autonomous emergency landing

The capability of performing an autonomous landing in the case of faults is an essential ability for an autonomous drone. In particular, the UAV needs to identify, approach and land at a safe site without any intervention. In this framework, we focus on a solution for the problem in an indoor and GNSS-denied environment similar to the one of the LDC (see Chapter 4).

Furthermore, we split the problem in two main sub-problems, *i.e.*, landing zone detection and path planning, and solve them separately.

5.1.1 Related work

In the literature, the landing zone detection problem is also called Safe Landing Area Determination (SLAD). Even if research works can be classified based on the sensor employed [255], in this work we prefer to classify them based on the data structure from which information is retrieved. Among them we can cite Digital Elevation Models (DEM) [256], Shuttle Radar Topography Mission Maps (SRTM) [257] or, more commonly, point clouds and images.

For what concerns point clouds, usually some kind of grid or voxel map is built starting from the 3D point cloud registration. The grid contains statistical information such as mean, minimum and maximum height, number of points. Then, some kind of plane fitting method is applied to compute the slope and the roughness of the various areas in the attempt to detect potential landing sites. Similar methods have been applied for full scale helicopters [258, 259, 260] and for UAVs [261].

For images a variety of methods to find flat areas have been proposed. Homography estimation for detecting features belonging to a plane have been used in [262, 263]. At the same time, feature extraction methods, *e.g.*, edge and corners detectors have been applied in [264, 265, 266]. A local contrast descriptor to assess roughness of the ground has been employed in [267]. In [36], the authors computed directly a vision-based elevation map onboard using an efficient method. Stereo camera depth maps are used for evaluating costmaps and detecting landing sites in [268]. In [269] a visual multiple target tracking mechanism has been utilized to help in the selection of a crash/landing site in populated areas. Finally, semantic information extracted from monocular images has been employed in a reinforcement landing framework in [270].

Turning to the problem of path planning in case of faults (in general for path planning refer to Chapter 1), different approaches have been proposed. For instance, in [271] the authors exploit an anytime RRT*, with checks on feasibility of the designed route, for gliding of a fixed-wing aircraft with Loss of Thrust (LoT). For a similar problem an hybrid A*

search on an extended tangent graph is carried out in [272]. Landing runway selection and trajectory generation of an aircraft after complete LoT is also tackled in [273]. A similar work, but for helicopter autoration after LoT, is carried out in [274]. They utilized a so called RRT* − AR (Alternate Routes) for finding a new landing trajectory after the failure.

For what concerns multicopters, we can mention [275], in which the authors utilized camera images for detecting potential landing spots. Then, they compute and/or verify the landing trajectories using the method presented in [112].

5.1.2 Safe Landing Area Determination

The starting point for the implementation is the definition of what is considered a safe place to land. In many works, *e.g.*, [267], a safe place is an area which fulfills the following conditions:

1. The area is big enough for the UAV to land within.

2. The area is clear, *i.e.*, free of either natural or artificial obstacles.

Note that additional constraints can be placed if a preference on landing on ground or on rooftops (as in [276]) exists.

The proposed approach starts by transforming the input point cloud, which can be either generated by a stereo camera (as presented in the LDC competition; see Chapter 4) or by a laser scanner, into an elevation grid map. Each element of the elevation map will contain the height h of the highest obstacle which belong to that cell. Then, similarly to [36], a cost matrix is computed to identify a local neighborhood of radius r in which the surface is flat, which should be somewhat related with the size of the UAV. The proposed cost matrix is represented by the standard deviation:

$$\sigma(i,j) = \frac{1}{N-1} \sum_{(u,v)\in\mathcal{R}(i,j,r)} (h(u,v) - \mu(i,j))^2 \tag{5.1}$$

where

$$\mu(i,j) = \frac{1}{N} \sum_{(u,v)\in\mathcal{R}(i,j,r)} h(u,v) \tag{5.2}$$

and $\mathcal{R}(i,j,r)$ is the set of cells around coordinate (i,j) that are located within a radius r.

All the areas with values of σ over a threshold will be discarded, leading to a binary grid. We consider as potential safe landing areas the ones which maximize the Euclidean Distance Transform (EDT) applied to the mentioned binary grid. In other words, we will pick the cells which satisfy the threshold on flatness, furthest from all cells that do not satisfy the mentioned criterion. Note that we do not select only a single point which maximizes the EDT, but the group of cells having the top percentile values. This is done considering that the safest areas are the ones with large connected components satisfying the said criterion. Finally, in the connected areas, the value with maximum EDT is selected.

Note that a value higher than the σ threshold is *a priori* assigned at the boundaries of the point cloud. This is meant for constraining the selection of the landing site to belong to the explored area.

5.1.3 Path planning

In case of emergency, *e.g.*, in the case of LoT, the path planner should not only plan a collision-free path, but a path as safe as possible, *i.e.*, the furthest from obstacles. In this view, a safety-aware A* [277] has been implemented. Differently from its standard version, the path is optimized for the risk associated with it, instead of its length.

Similarly to the formulation of [277], the function g represents the integral of the risk between the initial state to the node we are expanding, namely:

$$g(x_n) = \int_{x_{start}}^{x_{n-1}} r_c(x)dx + \int_{x_{n-1}}^{x_n} r_c(x)dx \tag{5.3}$$

$$= g(x_{n-1}) + c(x_{n-1}, x_n), \tag{5.4}$$

with

$$c(x_{n-1}, x_n) = \frac{r_c(x_{n-1}) + r_c(x_n)}{2} \text{dist}(x_{n-1}, x_n), \tag{5.5}$$

where $\text{dist}(x_{n-1}, x_n)$ is the Euclidean distance between two nodes. The admissible heuristic is formulated as:

$$h(x_n) = \frac{r_c(x_n) + r_c(x_{goal})}{2} \text{dist}_{min} + \tag{5.6}$$

$$= +(\text{dist}(x_n, x_{goal}) - \text{dist}_{min}) r_{c_{min}}. \tag{5.7}$$

where in our grid-based configuration dist_{min}, *i.e.*, the minimum distance between two adjacent nodes, is equal to the adopted resolution. $r_{c_{min}}$ is instead the minimum value of the risk-cost function.

Differently from [277], we employed the inverse (since we are solving a minimization problem) of the EDT, already computed for what concerns the safe landing detection algorithm, as risk indicator r_c.

In this case, similarly to the third edition of LDC, we implemented weighted version for the safety-aware A*. This also allows to generalize the algorithm into an Anytime variant (see also Chapter 1), by opportunely scaling the heuristic weight for having a quick solution in case of emergency, but being able to compute an optimal minimum risk solution if possible.

In the proposed formulation, re-planning is performed if the difference between the EDT of the cells belonging to the path increase over a certain threshold during the execution of the path itself.

Finally, note that when multiple potential safe landing areas are found by the SLAD algorithm, the one that will be used is selected either based on the length of the path, if multiple paths can be computed in the available time, or based on the first valid solution found.

5.1.4 Trajectory planning

Starting from the available minimum risk path, a minimum snap trajectory is generated following the approach of [108, 104]. Based on the assumption of having an indication about the type of fault and its consequences on the performance of the platform, the optimization process

can be also stopped for having reached the maximum available thrust and torques output of the UAV. This is easily implementable considering that the quadrotor is a differentially flat system (see Appendix D) and, thus, an algebraic relationship between the trajectory (flat output) and the generalized inputs (thrust and torques) exist.

5.1.5 Simulation results

To test the different steps of the proposed algorithm, some simulations have been performed using Matlab. The simulated environments are of the same size of the LDC arena, namely $20 \times 10\,\mathrm{m}$, and they are generated randomly. An example of the resulting point cloud, with additive noise, is shown in Figure 5.1.

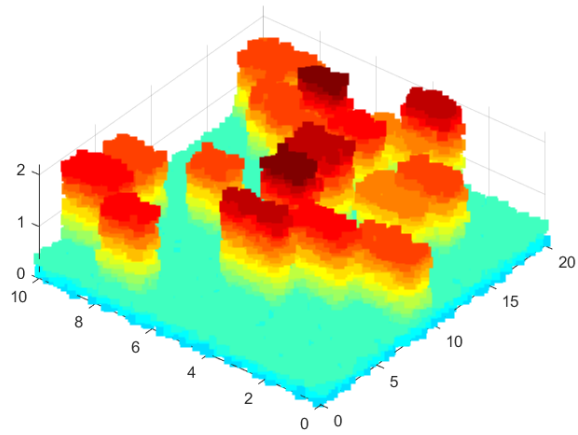


Figure 5.1: *Randomly generated point cloud*

According to the presented procedure, the EDT is applied to the binary map of cells which satisfy the flatness criterion. Then, the cells with the highest value of distance have been selected. The results of these procedures are shown in Figure 5.2.

From the plot, it is clear that two large connected components are present. Thus, we have found two potential landing sites, which are shown in Figure 5.3.

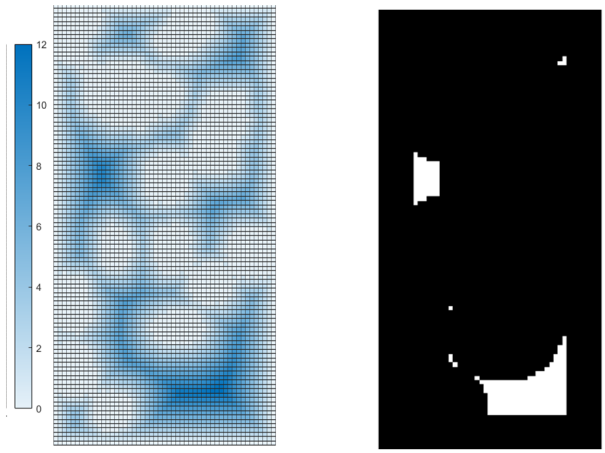


Figure 5.2: *Euclidean Distance Field and associated potential landing areas*

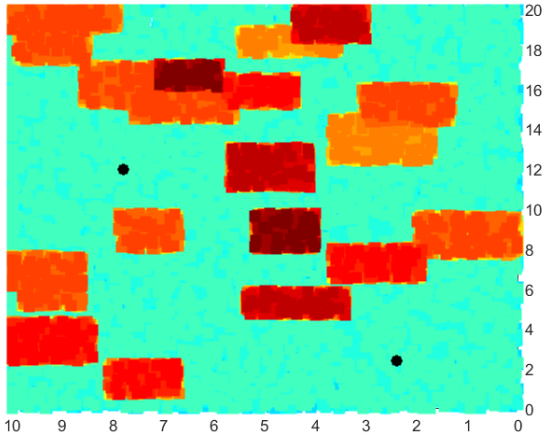


Figure 5.3: *Potential landing sites*

Assuming a starting position $(x = 7,\ y = 5)$, the safety-aware A* is applied to the two potential solutions. No solution exist for reaching the first solution $(x = 12,\ y = 8)$, while a safe solution is found for the landing site $(x = 2,\ y = 2)$. The resulting path and minimum-snap trajectory are shown respectively in Figure 5.4 and Figure 5.5.

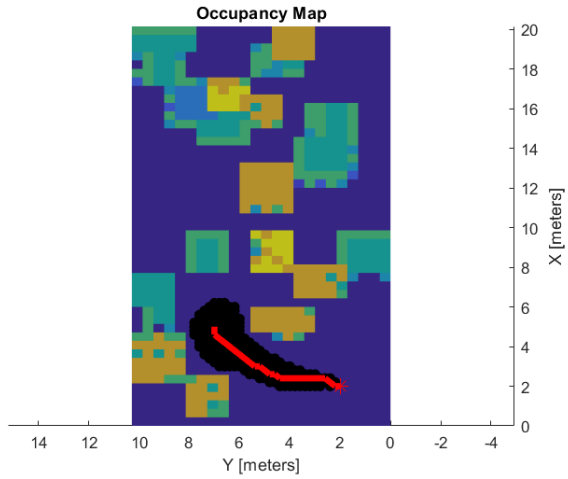


Figure 5.4: *Planned path for the example situation*

Consider also the difference between the conventional A* and the safety-aware proposed version. The paths, planned by the two different approaches and with the same start and target nodes, are shown in Figures 5.6 and 5.7. We can clearly see how a longer path has been computed by the safety-aware A* to stay away from the central obstacle.

To test the overall procedure, simulations have been also performed on the Gazebo simulator (see Section 2.1). The obtained result is shown in Figure 5.8. In the plot, we can see the point cloud accumulated until the trigger of the fault. This latter is simply a user input and, in this case, was triggered while the UAV was exploring the environment, traveling toward positive East, in correspondence of the diamond marker. While flying, the simulated UAV was continuously computing a possible landing area based on the accumulated point cloud. In this particular case, the point $(x = -3.5, y = -0.9)$ was selected (blue dot). Then, the safety-

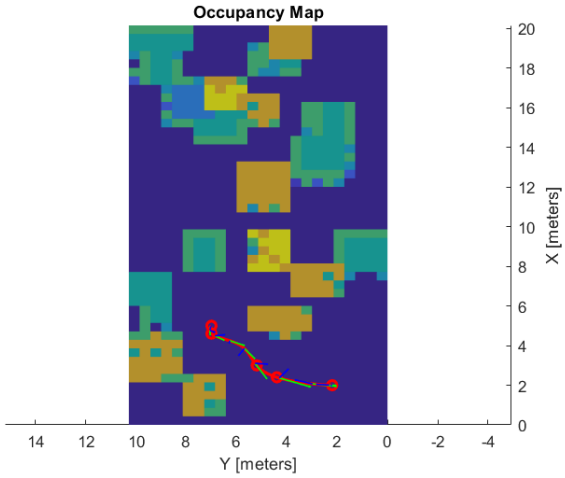


Figure 5.5: *Generated trajectory for the example situation*

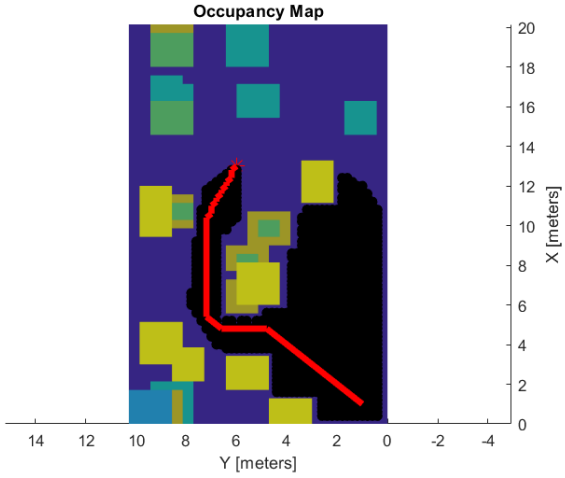


Figure 5.6: *Generated path using A^**

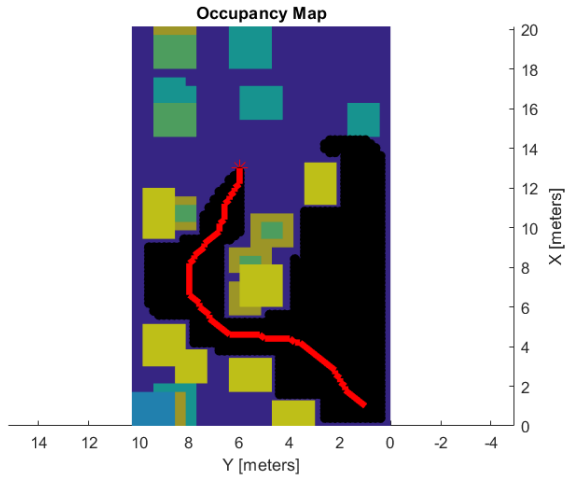


Figure 5.7: *Generated path using safety-aware A**

aware A* was applied for computing the path (red line) leading to the safe landing area. Finally, a minimum snap trajectory (green line) was calculated, starting from the retrieved path for reaching that area. Note that, once the drone reaches the landing site, the landing maneuver is actually commanded via a descending velocity set-point.

Experimental activities were carried out using the ROG-3 platform (see Section 2.2.3). The point cloud was produced using the mapping pipeline shown in Section 4.4. Also in this case, after a small exploration phase, the emergency signal was sent to the platform, which computed a safe path and trajectory from its current position to the pre-computed safe landing area. The results are graphically shown in Figure 5.9 using the same markers of the simulation plot.

5.1.6 Final considerations

In this Section, we discussed a possible solution for the problem of autonomous emergency landing for multirotor UAVs in indoor environments. The point cloud of the surrounding environment has been exploited for generating an elevation map, which in turn is used for detecting potential landing sites. The path from the current UAV position to the selected land-

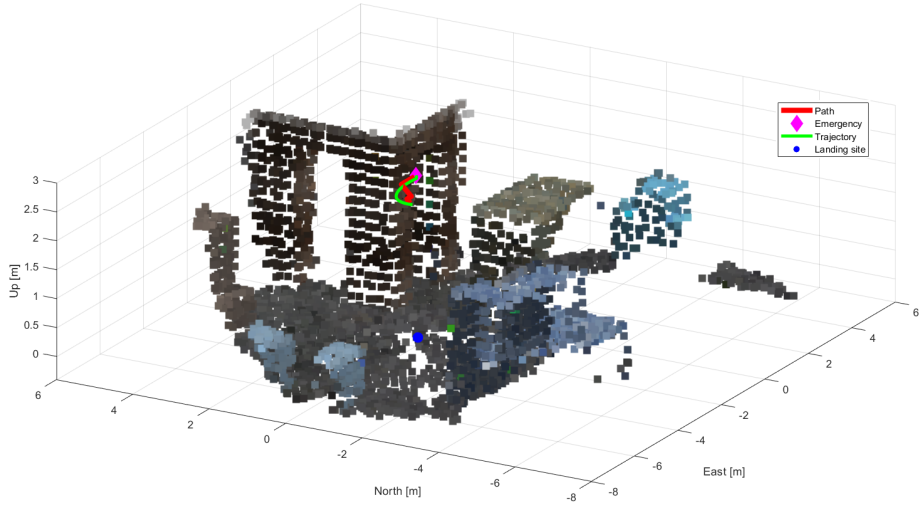


Figure 5.8: *Emergency landing simulation results*

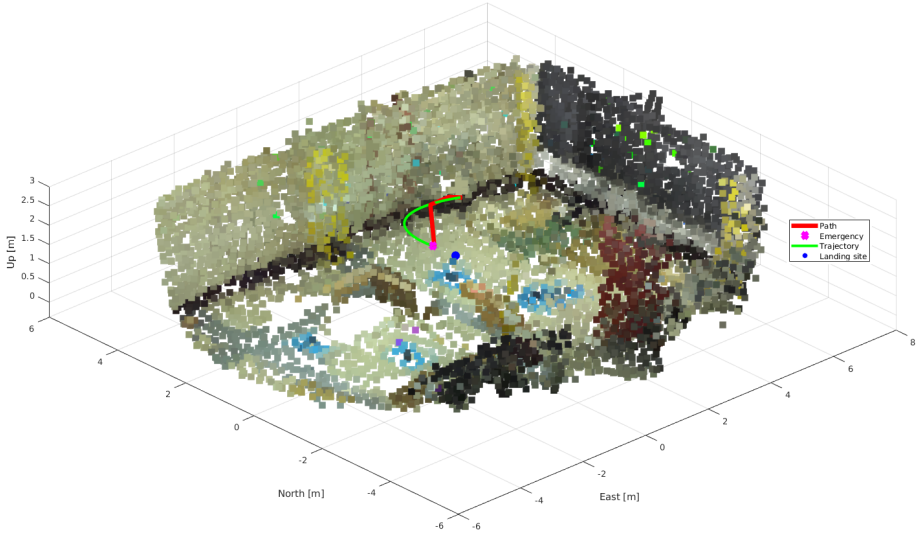


Figure 5.9: *Emergency landing experimental results*

ing area is computed using a safety-aware A*. Then, a feasible trajectory is computed considering the available control inputs. In the development phase, simulation tests have been performed using Matlab on randomly generated point clouds. In this framework, the selection of landing areas, path planning and trajectory generation libraries were tested separately. Finally, the overall solution has been implemented and validated both in SITL simulations and in experiments on a real platform.

5.2 Autonomous Air-to-Air Landing (AAAL)

This Section explores the design of a procedure to enable AAAL, *i.e.*, the landing of a small (follower) over a larger UAV (target). This problem is not only technologically complex but it is also risky and dangerous. In the case of a multirotor, the wake of the propellers generates an unsteady flow field around it, such that, when flying close, the two UAVs perturb each other. In this work, the authors propose a vision-based approach for tracking and landing on the target. To this aim, a Kalman Filter (KF) is designed to estimate the target state using information coming from a camera mounted on the follower drone. At the same time, a Quasi Time-Optimal (QTO) control law and an hybrid logic (see also [278]) are used to perform the landing. An experimental campaign has been carried out to verify the proposed landing approach.

5.2.1 Related works

In the literature many examples of landing of VTOL UAVs on moving platforms exist, especially onto ground vehicles. One of the first examples in shown in [279]. In this work, an autonomous landing has been accomplished through, Image-Based Visual Servoing (IBVS), *i.e.*, using a controller acting directly in the image space. Position-Based Visual Servoing (PBVS) has been instead employed with the same purpose in [280]. However, visual servoing represents a valid option to some extent because it requires the landing platform to be visible throughout the entire duration of the task [281].

In order to deal with missing visual information, model-based approaches have been proposed to predict the motion of the landing target

[282, 283]. In this framework, we distinguish between cooperative and non-cooperative approaches. To avoid any doubt related to the taxonomy, we consider cooperative an approach which requires the knowledge of the trajectory, state and/or measurements coming from the moving platform, non-cooperative otherwise.

Among the non-cooperative approaches, the authors of [281], to deal with missing visual detections, as well as, to estimate the full state of the platform (namely the position and velocity and possibly the acceleration), used an Extended Kalman Filter (EKF). Similar works have been carried out in the MBZIRC 2017 for landing on a moving vehicle. In particular, the solution proposed by the University of Bonn [220] employed an EKF for estimating the state of the ground vehicle, while the ETH Zurich [219] implemented a particle filter, and the Czech Technical University in Prague [284] used an Unscented Kalman Filter (UKF). In the mentioned work, the predicted trajectory of the moving vehicle is then tracked using Model Predictive Control (MPC) as outer loop. Note that in [281] only VIO was employed for the aerial vehicle state estimation, while in the works competiting in the MBZIRC, visual navigation was combined with valid measurements of GPS/RTK positions.

On the other hand, in [285], a cooperative scenario is presented. A multirotor UAV has the aim of landing on an high velocity ground vehicle, on top of which a fiducial marker is placed. The position, velocity and acceleration of both the multirotor UAV and the ground vehicle are jointly estimated fusing inertial measurements and GPS of both the ground and the aerial vehicles, along with the marker pose measurements reconstructed using a camera mounted on the drone. A similar problem has been tackled by [286] and by [287]. However, only GPS position of the ground vehicle is communicated to the UAV, in the case of missing visual detection. While in [285] PID is employed for the landing phase, in [286] MPC and INDI (see Section 1.4) flight control are combined. Finally, in [287] a velocity controller is designed combining a control barrier function (CBF) and a control Lyapunov function (CLF).

To the best of the author's knowledge, the work presented in this Chapter represents one of the few examples of a multirotor landing on a moving flying platform. Exceptions regard [288] and [289], where a small UAV

lands on a larger flying platform, although with this latter in near hover condition.

Note that in previous works from our laboratory [290, 278], the air-to-air landing problem was addressed in a cooperative scenario. A motion capture system was used to determine the position and velocity of the target drone and communicated to the follower one for performing the landing. However, while in [290] the target was in hovering conditions, in [278] the more challenging problem of a moving target was tackled.

5.2.2 Autonomous landing strategy

In the considered scenario, the follower has to perform the landing maneuver autonomously in a non-cooperative way, *i.e.*, using only the state of the target, estimated as explained in Section 5.2.5, whose motion cannot be controlled. The landing problem is formulated regardless of the specific UAV actuation mechanism because it is assumed that there exist control laws for the control force and torque such that any bounded velocity trajectory is asymptotically tracked, *e.g.*, [291, 292]. In our experiments to fit our design we use a customized version of the controller presented in Section 1.4.2. From this consideration the kinematic model:

$$\dot{x}_f = u, \tag{5.8}$$

is used for control design, where $u \in \mathbb{R}^3$ is a virtual input, corresponding to the follower velocity in the inertial frame $\mathcal{I}$ and $x_f \in \mathbb{R}^3$ is the position of the center of mass of the follower with respect to the inertial frame.

The target UAV is considered as a flat moving surface, described by a disk with center $x_t \in \mathbb{R}^3$ and radius $r_t \in \mathbb{R}_{>0}$. The motion of the target is described by the following kinematic model:

$$\dot{x}_t = v_t \tag{5.9}$$

$$\Omega_t = \left\{ y \in \mathbb{R}^3 : e_3^\top (y - x_t) = 0,\ \|y - x_t\| \leq r_t \right\} \tag{5.10}$$

where $v_t \in \mathbb{R}^3$ is velocity of the point x_t, resolved in $\mathcal{I}$. The above definition implies the attitude of the target be constantly aligned with the gravity direction, which is the most desirable condition to land. However, the landing strategy proposed in this work is robust to small attitude motion of the target.

Before stating the autonomous landing problem addressed in this work, we make the following assumptions.

Assumption 3. *1) Images coming from the camera mounted on the follower are assumed to provide information about the position of the visual marker frame $\mathcal{T}$ attached to target in the camera frame $\mathcal{C}$, resolved in $\mathcal{C}$; 2) at the initial time, the follower is above the target and during all the landing operations the visual marker is in the field of view of the camera sensor; 3) the position of the follower x_f and its attitude R_f are assumed to be known at all times.*

Based on the second bullet of Assumption 3, we are considering the final phase of the landing, when the follower is sufficiently close to target. The considered air-to-air landing problem can be formalized as follows.

Problem. *Consider the UAV kinematic model in equation* (5.8)*, under Assumption 3, find a control law for u such that x_f converges* safely *to a point in the set Ω_t defined in* (5.10) *in finite time.*

The adverb "*safely*" in the problem above encodes the requirement that the follower has to land from above the target and in a sufficiently slow manner, which allows minimizing perturbation effects between the two drones.

The problem has been tackled, on one way, by estimating the state of the target using a KF that combines the estimates of the state of the follower, coming from the onboard state estimator, and the measurement of the relative position of the target with respect to the follower, coming from a dedicated vision-based algorithm. At the same time, we have designed a control law for u, considering the UAV kinematic model in (5.8), under assumption 3, such that x_f converges safely, *i.e.*, from above and in a slow manner, in finite time to the flat surface of radius r_t that represents a safe position on the target.

In the next Sections we are going to present, first, our landing strategy, which combines a quasi time-optimal control law for tracking and a hybrid logic to perform the landing in a safe manner, and then the KF employed for target state estimation.

5.2.3 Quasi time-optimal tracking

Thanks to the assumption that the velocity of the follower is directly controllable, to track a desired sufficiently smooth trajectory $x_d(t)$, we can use input u in feedback strategy. We split the tracking error defined as $p := x_f - x_d$, in a planar $p_\perp := [p_1, p_2]^\top$ and a vertical p_3 component. Therefore, the error dynamics is $\dot{p} = u - \dot{x}_d = u - v_d$, where $v_d := \dot{x}_d$ is the desired velocity. The following control law is used

$$u_\perp(p_\perp, v_{d_\perp}) := -\text{sat}^\perp_{v_M}(p_\perp) + v_{d_\perp} \tag{5.11}$$

$$u_3(p_3, v_{d_3}) := -\text{sat}^{v_M}_{v_m}(k_3 p_3) + v_{d_3} \tag{5.12}$$

where $\text{sat}^{v_M}_{v_m}(p_3) := \min(\max(p_3, -v_m), v_M)$ is a scalar saturation function with saturation levels $v_m,\ v_M \in \mathbb{R}_{>0}$,

$$\text{sat}^\perp_{v_M}(p_\perp) := \min\left(k_\perp, \frac{v_M}{\|p_\perp\|}\right) p_\perp \tag{5.13}$$

and $k_\perp,\ k_3$ are scalar positive gains. The control law has been split in a planar (5.11) and vertical (5.12) component for reasons related to the hybrid logic.

The tracking properties of the proposed law are formalized by the following theorem, whose proof can be found in [278].

Theorem 1. *Consider the system* (5.8) *controlled by* (5.11)-(5.12). *Given any desired trajectory $x_d(t)$, $v_d(t)$ such that $\dot{x}_d(t) = v_d(t)$, for any positive $k_\perp$, k_3, v_m and v_M, the equilibrium point $p = 0$ is Globally Asymptotically Stable (GAS).*

In parallel with the definition given in [293] for the double integrator case, we refer to the control law (5.11)-(5.12) as quasi time-optimal.

5.2.4 Hybrid logic

In this Section the hybrid logic implemented to solve the landing problem is presented. It is based on three operating modes:

- Synchronization (Mode 0) during which the follower is far from the target and has to get close to a position at a certain height h_s above the target in a sufficiently fast way.

- Approach (Mode 1) during which the follower starts a sufficiently slow and controlled descent towards the target.

- Land (Mode 2) when the follower has reached a sufficiently close point above the target landing surface and the landing command is activated.

The proposed strategy can be described as a hybrid automaton, modelled using the framework of [294]. The different working modes are selected through the logical state $q \in Q := \{0, 1, 2\}$. Then, the state p evolves according to differential equations in the following domains defined for each mode:

$$C_0 := \left\{ p \in \mathbb{R}^3 : \begin{cases} \|p_\perp\| \geq r_m & \text{if} \quad p_3 \geq h_a \\ \|p_\perp\| \geq r_t & \text{if} \quad 0 \leq p_3 < h_a \end{cases} \right\} \tag{5.14}$$

$$C_1 := \left\{ p \in \mathbb{R}^3 : \|p_\perp\| \leq r_t, \quad p_3 \geq h_a \right\} \tag{5.15}$$

$$C_2 := \left\{ p \in \mathbb{R}^3 : \|p_\perp\| \leq r_t \quad p_3 = 0 \right\} \tag{5.16}$$

where $h_a \in \mathbb{R}_{>0}$ defines the altitude at which the approach phase should end while $\mathbb{R}_{>0} \ni r_m < r_t$ is the radius at which the synchronization phase should end. The choice $r_m < r_t$ will guarantee that switches between modes occur with hysteresis to avoid chattering phenomena.

We refer to [278] for the definition of the main elements of the hybrid logic: the set of edges, that identifies possible transitions between modes, the guard conditions that give for each edge the set to which the state has to belong for transitions between the two modes of the edge, and the reset map, which describes for each edge and state the jump of the state during a transition between the two modes of the edge. In Figure 5.10 the hybrid automaton is represented.

To solve the problem the following hybrid control laws are implemented:

- During the synchronization mode ($q = 0$) the objective of the control law

$$u = \begin{bmatrix} -\text{sat}_{v_s}^{\perp}(p_\perp) + v_{t_\perp} \\ -\text{sat}_{v_s}^{v_s}(k_3(p_3 - h_s)) + v_{t_3} \end{bmatrix} \tag{5.17}$$

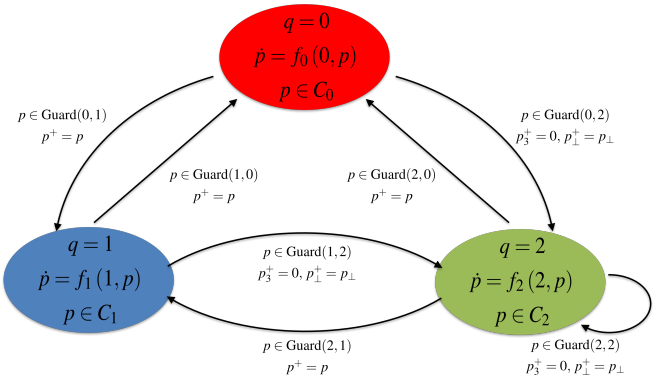


Figure 5.10: *Hybrid Automaton for landing strategy.*

is to track a point at a distance $\mathbb{R}_{>0} \ni h_s \gg h_a$ above the target point x_t, namely, $x_s := x_t + [0,\ 0,\ h_s]^\top$ which represents the safety approaching point.

- During the approaching mode ($q = 1$) the objective of the control law
$$u = \begin{bmatrix} -\text{sat}_{v_s}^{\perp}(p_\perp) + v_{t_\perp} \\ -\text{sat}_{v_a}^{v_s}(k_3 p_3) + v_{t_3} \end{bmatrix} \tag{5.18}$$
is to track the target point x_t, with approaching vertical speed bounded by v_a.

- Finally during land mode ($q = 2$) after the follower is brought to the target surface, in our experiment this corresponds to disarm, *i.e.*, turn off the rotors so that it falls by gravity on the target, the follower stays on the target surface due to friction and $u = v_t$.

The main result, whose proof can be found in [278], is here summarised.

Theorem 2. *Considering the closed-loop error dynamics obtained from* (5.8) *with the hybrid control law presented, under the two assumptions, for any choice $h_s > h_a$, and positive $k_\perp$, k_3, v_s and v_a all the closed loop solutions starting in the set $\Omega_0 := \{(q,p) \in Q \times \mathbb{R}^3 : p_3 \geq 0\}$ converge to the set $\Omega_\ell := \{(q,p) \in Q \times \mathbb{R}^3 : q = 2, \|p_\perp\| \leq r_t,\ p_3 = 0\}$ in finite time.*

To reduce the landing time when the follower exits the landing domain C_1 the height h_s is used as an additional state of the logic which is updated by setting it equal to the relative vertical distance achieved just before exiting the C_1 domain.

5.2.5 Vision-based target state estimation

Vision is used to estimate the position and velocity of the moving platform in an inertial frame. In order to simplify the detection task, an ArUco marker has been attached to the target drone and employed as visual marker (see Figure 2.15). It is worth pointing out that the approach can be generalized to different tags and detection algorithms.

The ArUco marker detection is carried out on the images coming from the camera equipped on the follower, exploiting the OpenCV library [295]. The marker detection process is comprised of two main steps:

1. Detection of marker candidates. In this step, the image is analyzed to find square shapes, candidates to be markers. An adaptive thresholding is applied for markers segmentation and, then, contours are extracted from the thresholded image. Contours that do not approximate to square shape are finally discarded.

2. Marker codification. In this step, marker bits are extracted from each detected marker. For doing so, a perspective transformation is first applied to obtain the marker in its canonical form. Then, the Otsu thresholding [296] is applied to the canonical image to separate white and black bits. Finally, the bits are analyzed to determine if the marker belongs to the specified dictionary.

Then, once the marker has been detected and identified, if the camera calibration parameters are known, the pose of the marker with respect to the camera (or the pose of the camera) can be reconstructed solving the Perspective-n-Point (PnP) problem. The resulting output is compliant with Assumption 3.

The position x_t and the velocity v_t of the target in the inertial frame are estimated through a discrete-time KF. Let the state vector be defined

as $x = \begin{bmatrix} x_t & v_t^\top \end{bmatrix}$. A kinematic model is used for the target motion, corresponding to the following discrete-time model:

$$x_k = Fx_{k-1} + w_{k-1} \tag{5.19}$$
$$y_k = Hx_k + v_k \tag{5.20}$$

with

$$F = \begin{bmatrix} I_3 & I_3\Delta t \\ 0_3 & I_3 \end{bmatrix} \tag{5.21}$$

where Δt is the sampling period of the filter. The process noise w_k is assumed to be a zero-mean Gaussian white noise with covariance matrix Q. Similarly, the measurement noise v_k is a Gaussian random vector with zero mean and covariance R, uncorrelated with the process noise w_k. It is worth noting that this model can be easily substituted by a more detailed one, if some more knowledge about the motion of the target is available. The filter prediction step is written as in equations (4.18)-(4.19). The correction phase is performed each time a measurement, *i.e.*, the 3D position of the moving target, is provided by the ArUco detection and pose reconstruction algorithms.

Similarly to the localization filter presented in Section 4.3.2, the measurement model, linear in the state, at a certain instant k, can be written as:

$$y = R_{\mathcal{CB}}(R_{\mathcal{IB}}^\top(x_t - x_f)) + t_{\mathcal{CB}} + v, \tag{5.22}$$

where with frame $\mathcal{B}$ we consider the body frame of the follower. As can be seen from equation (5.22), the position and attitude of the follower in the inertial frame are needed. They are directly retrieved from the follower estimator (the EKF2 presented in Appendix A). The filter correction step can be written as in equations (4.20)-(4.24). Finally, inliers are validated by checking the Normalized Estimation Error Squared (NEES) as in equation (4.25).

5.2.6 Simulation results

The approach has been first validated in simulation, using the simulation environment of Section 2.1. The only difference consists in the use of two simulated UAVs, the follower equipped with a virtual downward looking

camera, and the target equipped with a flat surface on which an ArUco marker is attached.

The vision-based target detection and automatic landing strategy are implemented as ROS nodes and the follower desired velocity is communicated to the PX4 autopilot firmware. Note that the drones initial positions and the target trajectory are commanded using the Matlab ANT-X [174] proprietary tool. A video of the simulation is available online[1]. A simulation of AAAL is shown with the target moving along a circular trajectory of radius $R = 1\,\mathrm{m}$ and angular frequency of $\omega = 0.2\,\mathrm{rad/s}$. In Figure 5.11 the 3D position trajectories of the two drones are shown together with the planned trajectory for the target. In the figure, circle, cross and star markers are used to identify the switches of the logic variable q: in correspondence of circle markers synchronization mode ($q = 0$) is activated, in correspondence of cross markers the approach phase ($q = 1$) begins, while star markers indicates when the disarm command is sent to the follower ($q = 2$).

In Figure 5.12 and Figure 5.13 the true and estimated relative in-plane and vertical distance are shown together with the evolution of the logic variable.

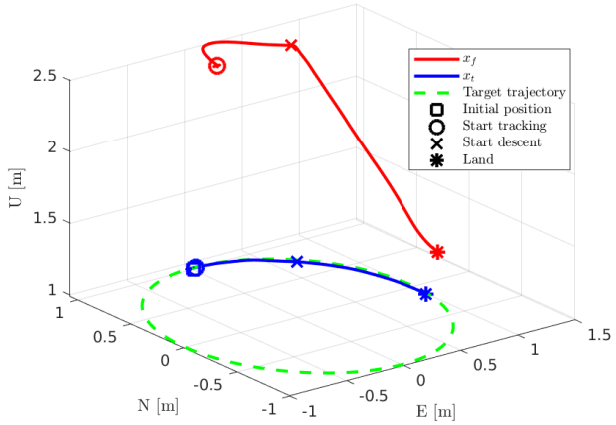


Figure 5.11: *3D position of follower and target during AAAL simulation*

[1] Visit `https://www.youtube.com/watch?v=LsINkVjS6R0`

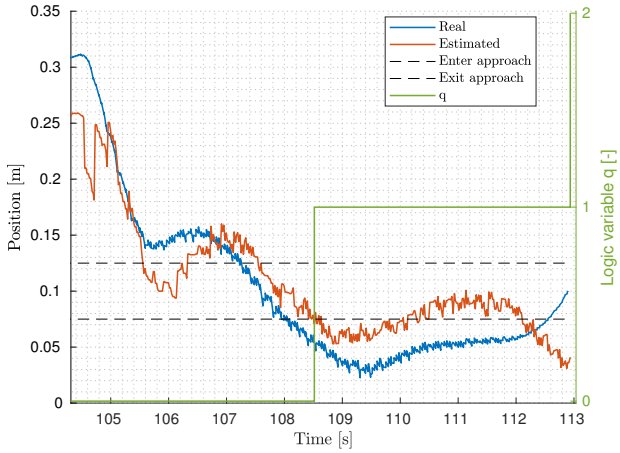


Figure 5.12: *Horizontal position of follower and target during AAAL simulation*

5.2.7 Preliminary test results

Flight tests are carried out inside the FlyART, using as aerial vehicles the ANT-X and CARRIER-1 drones (see Section 2.2). An ArUco marker of $19\,\mathrm{cm}$ width has been attached to the target, while a small landing gear has been built for the follower (Figure 5.14).

Before performing the vision-based landing, preliminary tests have been conducted focusing on the quality of the vision-based target state estimation when varying some camera parameters. In particular, with the target still on the ground and the follower moving slowly above it[2], we have tried to quantify the mean μ and the standard deviation σ of the error between the true and the estimated position of the target at different values of camera resolution (QQVGA, QVGA, VGA)[3]. The obtained results are shown in Table 5.1. Given the trade-off between accuracy and CPU load, the QVGA resolution has been chosen for all the subsequent tests.

Before conducting vision-based landing experiments, we have analyzed the performance of the tracking system when degrading operating conditions, starting from the results obtained in [278], where the motion

[2]With the marker always in the camera field of view.

[3]Higher resolutions have not been considered because of the CPU load.

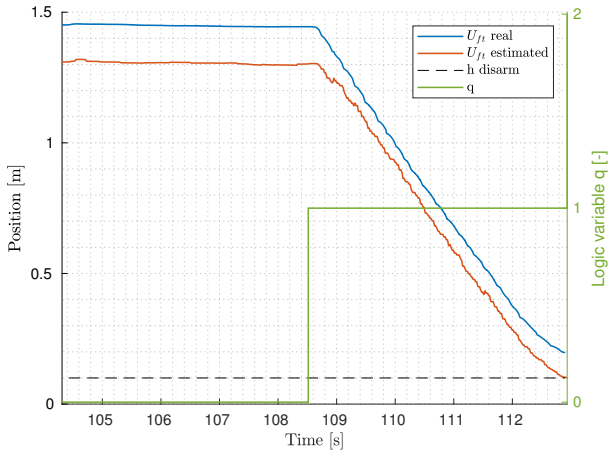


Figure 5.13: *Vertical position of follower and target during AAAL simulation*

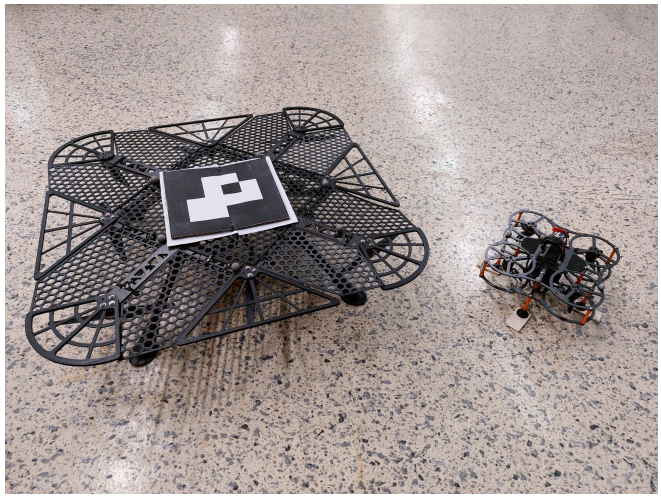


Figure 5.14: *ANT-X and CARRIER-1 drones.*

Table 5.1: *Mean and standard deviation of the error between the true end the estimated position of the target.*

	QQVGA	QVGA	VGA
Image rate [Hz]	50	30	15
ArUco pose rate [Hz]	6	4	4
μ_x [m]	0.071	0.042	0.053
σ_x [m]	0.035	0.042	0.039
μ_y [m]	0.043	0.014	0.019
σ_y [m]	0.039	0.035	0.042
μ_z [m]	-0.042	0.045	0.097
σ_z [m]	0.041	0.077	0.063
CPU load	78%	85%	88%

capture system was employed for retrieving the target state. In particular, the following measurements have been utilized, both in simulations and experiments, for estimating the target state based on the proposed filter (Section 5.2.5):

- Motion capture measurements at $50\,\mathrm{Hz}$ (similarly to [278]);
- Motion capture measurements under-sampled at $4\,\mathrm{Hz}$ (the rate at which we get ArUco pose estimates according to Table 5.1);
- Motion capture measurements under-sampled at $4\,\mathrm{Hz}$ with addition of noise (using noise information from Table 5.1);
- Vision-based measurements.

To compare these cases we perform a test with the follower in synchronization mode at altitude $h_f = 2.5\,\mathrm{m}$ and estimating the target moving along a circular trajectory of radius $R = 1\,\mathrm{m}$ and angular frequency $\omega = 0.2\,\mathrm{rad/s}$ at constant altitude $h_t = 1\,\mathrm{m}$. The estimation of the position in all the cases is comparable. The main difference resides in the estimate of the velocity. In Figure 5.15 the estimates of the velocity of the target obtained from the filter using the under-sampled and noisy Mo-Cap measurements and the vision measurements are compared with the true ones. With this test we demonstrate the equivalence between the estimation obtained with the vision and the estimation obtained under-sampling

the Mo-Cap measurements at $4\,\mathrm{Hz}$ and adding a noise with standard deviation of the order of magnitude of the ones reported in Table 5.1, *i.e.*, $\sigma_x = \sigma_y = 3.5 \times 10^{-2}\,\mathrm{m}$ and $\sigma_z = 7 \times 10^{-2}\,\mathrm{m}$.

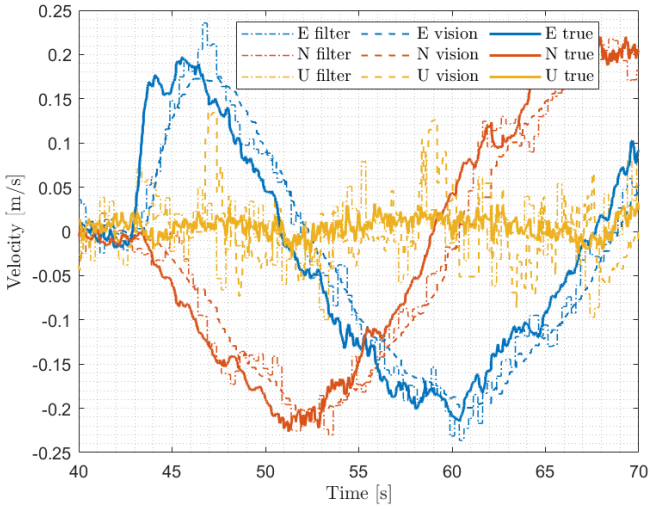


Figure 5.15: *True and estimated velocity of the target. The estimates are obtained from the KF described in Section 5.2.5 using the under-sampled and noisy Mo-Cap measurements and the vision measurements.*

5.2.8 Vision-based landing results

The landing has been performed in two conditions: with the target moving along a circular trajectory of radius $R = 1\,\mathrm{m}$ and angular frequency $\omega = 0.2\,\mathrm{rad/s}$ and with the target moving along a linear trajectory with velocity $v_t = 0.4\,\mathrm{m/s}$.

In the case of circular trajectory the follower starts in $[0,\ 0.75,\ 2.5]^\top \mathrm{m}$ while the circular trajectory is centered in $[0,\ 0,\ 1]^\top \mathrm{m}$ with respect to the inertial frame with ENU convention of the flying arena. In Figure 5.16 the in-plane position trajectories of the two drones are shown together with the target desired trajectory. The switches of the logic variable q are represented as in the simulation results.

As can be observed, at the beginning the synchronization mode is active and the follower tracks the target keeping the vertical distance con-

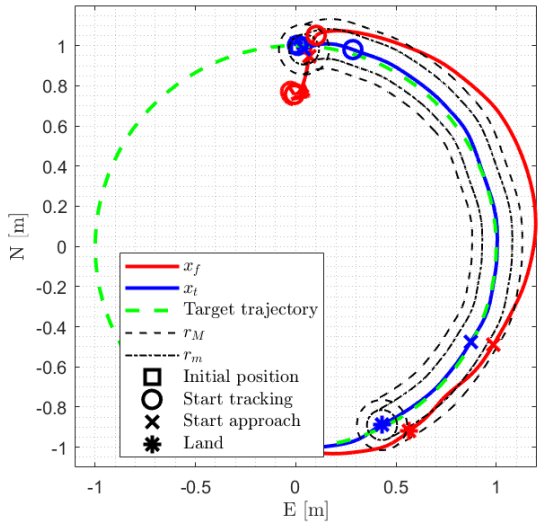


Figure 5.16: *In-plane position of follower and target in AAAL experiments on a circular trajectory.*

stant. When the in-plane error $p_\perp$ is less then $0.0075 = r_m < r_t = 0.0125\text{m}$, the logic variable switches to 1 and the follower starts the approach phase. Right after entering, the follower exits the C_1 region because of the delay in the estimate of the velocity of the target and, therefore, the logic state is switched back to the synchronization mode again. When the in-plane distance distance is again lower than r_m, q jumps to 1. Finally it keeps in the C_1 domain until the landing mode is activated ($q = 2$). In Figure 5.17 and Figure 5.18 the true and estimated relative in-plane and vertical distance are shown together with the evolution of the logic variable. In particular, some considerations can be made:

- The relative distance obtained from vision measurements underestimates the true one. This could lead to dangerous situations, *e.g.*, landing outside the target surface. For this reason we chose conservative bounds r_m and r_t with respect to the ones in [278] considering the values for the mean of the error reported in Table 5.1. In this way, even if at the end of the approach phase the true in-plane relative distance is slightly outside the bound of the domain, the landing can be

considered safe.

- During the first approach, some spikes in the estimated in-plane relative distance could be seen. This was due to the loss of some packages of data in the telemetry.
- During the final part of the descent, at an height above the target, which depends on the dimension of the marker and on the camera FOV, the tag cannot be seen by the monocular camera. In our case this happened at $\sim 40\,\mathrm{cm}$, and from that moment until the disarm command, the follower continues the descent and uses the position estimated only with the prediction step of the KF. Note that, in the literature, the problem of loss of visual information near the marker during landing is solved either through range finders, *e.g.*, [281, 284] ,or through multiscale visual fiducial markers on the landing pad [285].

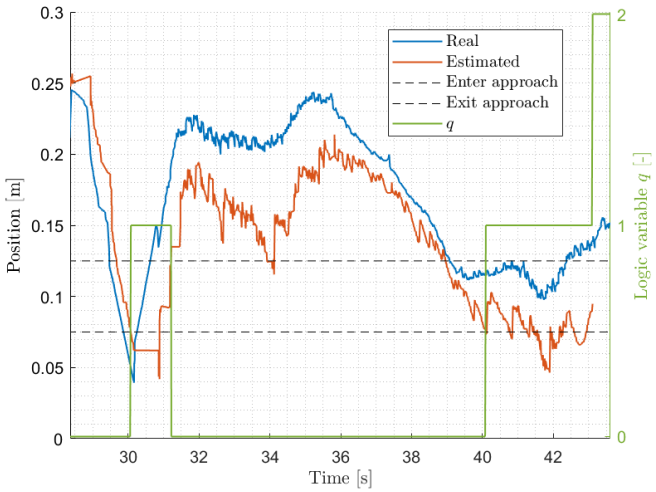


Figure 5.17: *True and estimated in-plane relative position time history in AAAL experiments on a circular trajectory*

The proposed approach is limited by the delay of the estimated target velocity (Figure 5.15), which in turn is due to the fact that the model used for the prediction step is kept as general as possible. As can be seen

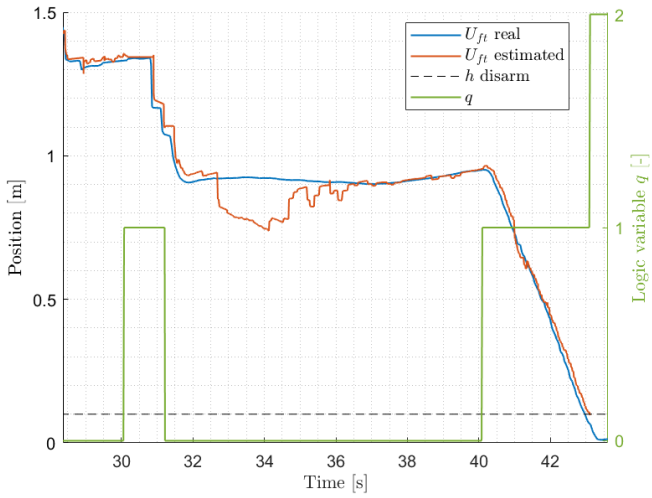


Figure 5.18: *True and estimated relative vertical position time history in AAAL experiments on a circular trajectory*

in Figure 5.19, when a linear trajectory with velocity $v_t = 0.4\,\mathrm{m/s}$ is considered for the target, the results improve in terms of estimation performance because of the matching between the real motion of the target and the model used in the estimator. Note that similar results, in terms of velocities achieved both in circular and in linear trajectories, have been obtained in [280].

A video of the experiment is available online[4].

5.2.9 Final considerations

In this Section, we tackled the problem of vision-based AAAL of UAVs. A KF state estimation approach has been used for determining the position and velocity of the target drone in a non-cooperative manner. The availability of this information has been exploited for performing a safe landing combining an hybrid logic with a QTO tracking controller. Preliminary tests have shown how visual information degrade performance of such designed filter compared to more accurate positioning systems.

[4]Visit `https://www.youtube.com/shorts/g7ojYgm35M8`

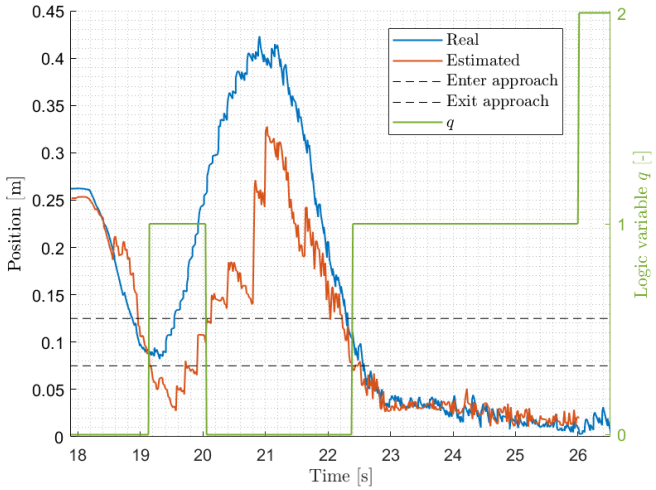


Figure 5.19: *True and estimated in-plane relative position time history in AAAL experiments on a linear trajectory.*

Finally, the proposed strategy has been validated through simulations and an experimental campaign involving two multirotor UAVs.

5.3 Concluding remarks

In this Chapter, the guidance, navigation and control solutions proposed for autonomous landing have been addressed concerning two different use-cases: emergency landing and vision-based Autonomous Air-to-Air landing. The software architectures for both the problems have been presented in detail. Finally, simulation and experimental results have been showed.

Conclusions

This dissertation focuses on the development, simulation and experimental validation of guidance and navigation algorithms for autonomous multirotor UAVs. Three software and hardware architectures for autonomous UAVs have been proposed in the framework of the Leonardo Drone Contest, an autonomous drone competition. The platforms have been extensively tested on long duration autonomous flights in GNSS-denied indoor environments. In particular, navigation in partially known environments, planning, collision avoidance, interactions with ground robots and human agents are some of the aspects tackled during the three competitions. Simulation and experimental results, obtained both in laboratory and on the competition field, have been presented and discussed.

A second contribution of this work is related to the definition of a systematic approach to the characterization of visual systems. Visual odometry performance, critical in the developed autonomous drones, have been analyzed in a variety of operating conditions. In particular, the Allan Variance analysis technique has been used to model the errors affecting the stereo VO position measurements. The models have been then validated and compared when varying the flight conditions.

Finally, autonomous landing has been tackled in two use-cases. The first is the vision-based Air-to-Air Landing, in which a small drone lands

on top of a larger drone during flight. A non-cooperative approach using vision for tracking has been proposed. The landing maneuver has been validated in simulations and experiments without any assumption on the motion of the target platform. The second use-case is related to the autonomous emergency landing, a fundamental capability that an autonomous system should possess. In particular, we have focused on a solution for the problem in an indoor and GNSS-denied environment. The UAV, first, identifies potential landing spots based on the map acquired up to that moment and, then, plans a feasible plan and trajectory from the current drone position to the selected landing area.

Bibliography

[1] M. Hassanalian and A. Abdelkefi, "Classifications, applications, and design challenges of drones: A review," *Progress in Aerospace Sciences*, vol. 91, pp. 99–131, 2017.

[2] T. Elmokadem and A. V. Savkin, "Towards fully autonomous UAVs: A survey," *Sensors*, vol. 21, no. 18, p. 6223, 2021.

[3] H.-M. Huang, "Autonomy levels for unmanned systems (ALFUS) framework volume I: Terminology version 2.0," 2004-09-30 2004.

[4] R. R. Murphy, *Disaster robotics*. MIT press, 2014.

[5] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, "Autonomous, vision-based flight and live dense 3D mapping with a quadrotor micro aerial vehicle," *Journal of Field Robotics*, vol. 33, no. 4, pp. 431–450, 2016.

[6] A. Bachrach, R. He, and N. Roy, "Autonomous flight in unknown indoor environments," *International Journal of Micro Air Vehicles*, vol. 1, no. 4, pp. 217–228, 2009.

[7] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy, "Stereo vision and laser odometry for autonomous helicopters in GPS-

denied indoor environments," *Unmanned Systems Technology XI*, vol. 7332, p. 733219, 2009.

[8] S. Grzonka, G. Grisetti, and W. Burgard, "A fully autonomous indoor quadrotor," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 90–100, 2012.

[9] M. Blösch, S. Weiss, D. Scaramuzza, and R. Siegwart, "Vision based MAV navigation in unknown and unstructured environments," *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 21–28, 2010.

[10] J. Engel, J. Sturm, and D. Cremers, "Camera-based navigation of a low-cost quadrocopter," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2815–2821, 2012.

[11] S. Shen, N. Michael, and V. Kumar, "Autonomous indoor 3D exploration with a micro-aerial vehicle," *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9–15, 2012.

[12] J. Engel, J. Sturm, and D. Cremers, "Scale-aware navigation of a low-cost quadrocopter with a monocular camera," *Robotics and Autonomous Systems*, vol. 62, no. 11, pp. 1646–1656, 2014.

[13] K. Schmid, P. Lutz, T. Tomić, E. Mair, and H. Hirschmüller, "Autonomous vision-based micro air vehicle for indoor and outdoor navigation," *Journal of Field Robotics*, vol. 31, no. 4, pp. 537–570, 2014.

[14] D. Scaramuzza, M. Achtelik, L. Doitsidis, F. Friedrich, E. Kosmatopoulos, A. Martinelli, M. Achtelik, M. Chli, S. Chatzichristofis, L. Kneip, *et al.*, "Vision-controlled micro flying robots: from system design to autonomous navigation and mapping in GPS-denied environments," *IEEE Robotics & Automation Magazine*, vol. 21, no. 3, pp. 26–40, 2014.

[15] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online UAV replanning," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5332–5339, 2016.

[16] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3529–3536, 2019.

[17] M. Burri, H. Oleynikova, M. W. Achtelik, and R. Siegwart, "Real-time visual-inertial mapping, re-localization and planning onboard MAVs in unknown environments," *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1872–1878, 2015.

[18] K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, *et al.*, "Fast, autonomous flight in GPS-denied and cluttered environments," *Journal of Field Robotics*, vol. 35, no. 1, pp. 101–120, 2018.

[19] X. Liu, G. V. Nardari, F. C. Ojeda, Y. Tao, A. Zhou, T. Donnelly, C. Qu, S. W. Chen, R. A. F. Romero, C. J. Taylor, and V. Kumar, "Large-scale autonomous flight with real-time semantic SLAM under dense forest canopy," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5512–5519, 2022.

[20] H. Oleynikova, C. Lanegger, Z. Taylor, M. Pantic, A. Millane, R. Siegwart, and J. Nieto, "An open-source system for vision-based micro-aerial vehicle mapping, planning, and flight in cluttered environments," *Journal of Field Robotics*, vol. 37, no. 4, pp. 642–666, 2020.

[21] T. Baca, M. Petrlik, M. Vrba, V. Spurny, R. Penicka, D. Hert, and M. Saska, "The MRS UAV system: Pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 1, pp. 1–28, 2021.

[22] G. Loianno, C. Brunner, G. McGrath, and V. Kumar, "Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and IMU," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 404–411, 2016.

[23] E. Tal and S. Karaman, "Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 3, pp. 1203–1218, 2020.

[24] A. Antonini, W. Guerra, V. Murali, T. Sayre-McCord, and S. Karaman, "The blackbird UAV dataset," *The International Journal of Robotics Research*, vol. 39, no. 10-11, pp. 1346–1364, 2020.

[25] P. Foehn, E. Kaufmann, A. Romero, R. Penicka, S. Sun, L. Bauersfeld, T. Laengle, G. Cioffi, Y. Song, A. Loquercio, *et al.*, "Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight," *Science Robotics*, 2022.

[26] J. Dias, K. Althoefer, and P. U. Lima, "Robot competitions: What did we learn?," *IEEE Robotics & Automation Magazine*, vol. 23, no. 1, pp. 16–18, 2016.

[27] J. Stuckler, D. Holz, and S. Behnke, "RoboCup@Home: Demonstrating everyday manipulation skills in RoboCup@Home," *IEEE Robotics and Automation Magazine*, vol. 19, no. 2, pp. 34–42, 2012.

[28] H. Moon, J. Martínez-Carranza, T. Cieslewski, M. Faessler, D. Falanga, A. Simovic, D. Scaramuzza, S. Li, M. M. O. I. Ozo, C. de Wagter, G. C. de Croon, S. Hwang, S. Jung, H. Shim, H. Kim, M. Park, T.-C. Au, and S. J. Kim, "Challenges and implemented technologies used in autonomous drone racing," *Intelligent Service Robotics*, vol. 12, pp. 137–148, 2019.

[29] "Alphapilot AI Drone Innovation Challenge." `https://www.lockheedmartin.com/en-us/news/events/ai-innovation-challenge.html`, Access date: 17/01/2022.

[30] C. De Wagter, F. Paredes-Vallés, N. Sheth, and G. de Croon, "Learning fast in autonomous drone racing," *Nature Machine Intelligence*, vol. 3, no. 10, pp. 923–923, 2021.

[31] "International Micro Air Vehicles Conferences and Competitions." http://www.imavs.org/, Access date: 17/01/2022.

[32] "Mohamed Bin Zayed International Robotics Challenge (MBZIRC)." https://www.mbzirc.com/, Access date: 17/01/2022.

[33] "DARPA Fast Lightweight Autonomy (FLA)." https://www.darpa.mil/program/fast-lightweight-autonomy, Access date: 17/01/2022.

[34] "DARPA Subterrean Challenge." https://www.subtchallenge.com/, Access date: 17/01/2022.

[35] "PX4 documentation." https://docs.px4.io/v1.10/en/, Access date: 06/05/2022.

[36] C. Forster, M. Faessler, F. Fontana, M. Werlberger, and D. Scaramuzza, "Continuous on-board monocular-vision-based elevation mapping applied to autonomous landing of micro aerial vehicles," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 111–118, 2015.

[37] "ROS REP105." https://www.ros.org/reps/rep-0105.html, Access date: 14/09/2022.

[38] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 2010.

[39] J. Solà, "Quaternion kinematics for the error-state Kalman filter," *CoRR*, vol. abs/1711.02508, 2017.

[40] "Vicon - motion capture system." https://www.vicon.com/, Access date: 31/08/2022.

[41] "Optitrack - motion capture system." https://optitrack.com/, Access date: 31/08/2022.

[42] S. Lupashin, M. Hehn, M. W. Mueller, A. P. Schoellig, M. Sherback, and R. D'Andrea, "A platform for aerial robotics research and demonstration: The flying machine arena," *Mechatronics*, vol. 24, no. 1, pp. 41–54, 2014.

[43] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The GRASP multiple micro-UAV testbed," *IEEE Robotics & Automation Magazine*, vol. 17, no. 3, pp. 56–65, 2010.

[44] "Velodyne Lidar." `https://velodynelidar.com/`, Access date: 24/09/2022.

[45] "Ouster Lidar." `https://ouster.com/`, Access date: 24/09/2022.

[46] A. Bachrach, S. Prentice, R. He, P. Henry, A. S. Huang, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Estimation, planning, and mapping for autonomous flight using an RGB-D camera in GPS-denied environments," *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1320–1343, 2012.

[47] S. Shen, N. Michael, and V. Kumar, "Stochastic differential equation-based exploration algorithm for autonomous indoor 3D exploration with a micro-aerial vehicle," *The International Journal of Robotics Research*, vol. 31, no. 12, pp. 1431–1444, 2012.

[48] S. Hrabar, G. S. Sukhatme, P. Corke, K. Usher, and J. Roberts, "Combined optic-flow and stereo-based navigation of urban canyons for a UAV," *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3309–3316, 2005.

[49] F. Ruffier and N. Franceschini, "Visually guided micro-aerial vehicle: automatic take off, terrain following, landing and wind reaction," *2004 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3, pp. 2339–2346, 2004.

[50] V. Lippiello, G. Loianno, and B. Siciliano, "MAV indoor navigation based on a closed-form solution for absolute scale velocity

estimation using optical flow and inertial data," *2011 IEEE Conference on Decision and Control (CDC) and European Control Conference (ECC)*, pp. 3566–3571, 2011.

[51] S. Zingg, D. Scaramuzza, S. Weiss, and R. Siegwart, "MAV navigation through indoor corridors using optical flow," *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3361–3368, 2010.

[52] J.-C. Zufferey and D. Floreano, "Fly-inspired visual steering of an ultralight indoor aircraft," *IEEE Transactions on Robotics*, vol. 22, no. 1, pp. 137–146, 2006.

[53] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 15–22, 2014.

[54] S. Weiss, D. Scaramuzza, and R. Siegwart, "Monocular-SLAM-based navigation for autonomous micro helicopters in GPS-denied environments," *Journal of Field Robotics*, vol. 28, pp. 854–874, 11 2011.

[55] S. Weiss, M. W. Achtelik, S. Lynen, M. C. Achtelik, L. Kneip, M. Chli, and R. Siegwart, "Monocular vision for long-term micro aerial vehicle state estimation: A compendium," *Journal of Field Robotics*, vol. 30, no. 5, pp. 803–831, 2013.

[56] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, "Vision-based state estimation and trajectory control towards high-speed flight with a quadrotor," *Robotics: Science and Systems*, vol. 1, p. 32, 2013.

[57] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[58] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): part II," *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[59] S. Thrun, *Probabilistic robotics*. ACM New York, NY, USA, 2002.

[60] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, *et al.*, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," *AAAI/IAAI Conference on Artificial Intelligence*, vol. 593598, 2002.

[61] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, *et al.*, "FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," *International Joint Conference of Artificial Intelligence (IJCAI)*, vol. 3, pp. 1151–1156, 2003.

[62] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[63] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Incremental smoothing and mapping," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1365–1378, 2008.

[64] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[65] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robotics & Automation Magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[66] F. Fraundorfer and D. Scaramuzza, "Visual odometry : Part II: Matching, robustness, optimization, and applications," *IEEE Robotics and Automation Magazine*, vol. 19, no. 2, pp. 78–90, 2012.

[67] M. Irani and P. Anandan, "All about direct methods," *International Workshop on Vision Algorithms*, pp. 267–277, 1999.

[68] H. Strasdat, J. Montiel, and A. J. Davison, "Real-time monocular SLAM: Why filter?," *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2657–2664, 2010.

[69] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[70] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," *2007 IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 1–10, 2007.

[71] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[72] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[73] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," *European Conference on Computer Vision*, pp. 834–849, 2014.

[74] J. Engel, J. Sturm, and D. Cremers, "Semi-dense visual odometry for a monocular camera," *IEEE International Conference on Computer Vision (ICCV)*, pp. 1449–1456, 2013.

[75] P. Corke, J. Lobo, and J. Dias, "An introduction to inertial and visual sensing," *The International Journal of Robotics Research*, vol. 26, no. 6, pp. 519–535, 2007.

[76] D. Scaramuzza and Z. Zhang, "Visual-inertial odometry of aerial robots," *CoRR*, vol. abs/1906.03289, 2019.

[77] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[78] H. Moravec, "Robot spatial perception by stereoscopic vision and 3D evidence grids," *Perception*, 1996.

[79] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013.

[80] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart, "Signed distance fields: A natural representation for both mapping and planning," *RSS 2016 workshop: geometry and beyond-representations, physics, and scene understanding for robotics*, 2016.

[81] L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Autonomous visual mapping and exploration with a micro aerial vehicle," *Journal of Field Robotics*, vol. 31, no. 4, pp. 654–675, 2014.

[82] R. Triebel, P. Pfaff, and W. Burgard, "Multi-level surface maps for outdoor terrain mapping and loop closing," *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2276–2282, 2006.

[83] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard, *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[84] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[85] J. H. Reif, "Complexity of the mover's problem and generalizations," *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pp. 421–427, 1979.

[86] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous UAV guidance," *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1, pp. 65–100, 2010.

[87] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transac-*

tions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100–107, 1968.

[88] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[89] D. Harabor and A. Grastien, "Online graph pruning for pathfinding on grid maps," *AAAI/IAAI Conference on Artificial Intelligence*, vol. 25, no. 1, pp. 1114–1119, 2011.

[90] I. Pohl, "Heuristic search viewed as path finding in a graph," *Artificial intelligence*, vol. 1, no. 3-4, pp. 193–204, 1970.

[91] E. A. Hansen and R. Zhou, "Anytime heuristic search," *Journal of Artificial Intelligence Research*, vol. 28, pp. 267–297, 2007.

[92] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," *Advances in neural information processing systems*, vol. 16, 2003.

[93] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354–363, 2005.

[94] M. Nieuwenhuisen, D. Droeschel, M. Beul, and S. Behnke, "Obstacle detection and navigation planning for autonomous micro aerial vehicles," *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1040–1047, 2014.

[95] J. Chen, T. Liu, and S. Shen, "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments," *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1476–1483, 2016.

[96] S. Liu, M. Watterson, S. Tang, and V. Kumar, "High speed navigation for quadrotors with limited onboard sensing," *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1484–1491, 2016.

[97] L. Heng, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Autonomous obstacle avoidance and maneuvering on a vision-guided MAV using on-board processing," *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2472–2477, 2011.

[98] J. Tordesillas, B. T. Lopez, M. Everett, and J. P. How, "FASTER: Fast and safe trajectory planner for navigation in unknown environments," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 922–938, 2021.

[99] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[100] J.-C. Latombe, *Robot motion planning*, vol. 124. Springer Science & Business Media, 2012.

[101] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[102] H. Yu and R. Beard, "A vision-based collision avoidance technique for micro air vehicles using local-level frame mapping and path planning," *Autonomous Robots*, vol. 34, no. 1, pp. 93–109, 2013.

[103] L. Matthies, R. Brockers, Y. Kuwata, and S. Weiss, "Stereo vision-based obstacle avoidance for micro air vehicles using disparity space," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3242–3249, 2014.

[104] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," *Robotics Research*, pp. 649–666, 2016.

[105] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding Horizon" Next-Best-View" planner for 3D exploration," *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1462–1468, 2016.

[106] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2520–2525, 2011.

[107] M. J. Van Nieuwstadt and R. M. Murray, "Real-time trajectory generation for differentially flat systems," *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, vol. 8, no. 11, pp. 995–1020, 1998.

[108] A. Bry, C. Richter, A. Bachrach, and N. Roy, "Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments," *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 969–1002, 2015.

[109] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Search-based motion planning for aggressive flight in SE (3)," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2439–2446, 2018.

[110] S. Liu, N. Atanasov, K. Mohta, and V. Kumar, "Search-based motion planning for quadrotors using linear quadratic minimum time control," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2872–2879, 2017.

[111] B. MacAllister, J. Butzke, A. Kushleyev, H. Pandey, and M. Likhachev, "Path planning for non-circular micro aerial vehicles in constrained environments," *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3933–3940, 2013.

[112] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.

[113] M. Ryll, J. Ware, J. Carter, and N. Roy, "Efficient trajectory planning for high speed flight in unknown environments," *2019 International Conference on Robotics and Automation (ICRA)*, pp. 732–738, 2019.

[114] P. Florence, J. Carter, and R. Tedrake, "Integrated perception and control at high speed: Evaluating collision avoidance maneuvers

without maps," *Algorithmic Foundations of Robotics XII*, pp. 304–319, 2016.

[115] H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto, "Safe local exploration for replanning in cluttered unknown environments for microaerial vehicles," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1474–1481, 2018.

[116] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[117] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1917–1922, 2012.

[118] B. Landry, R. Deits, P. R. Florence, and R. Tedrake, "Aggressive quadrotor flight through cluttered environments using mixed integer programming," *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1469–1475, 2016.

[119] R. Deits and R. Tedrake, "Efficient mixed-integer planning for UAVs in cluttered environments," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 42–49, 2015.

[120] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-D complex environments," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1688–1695, 2017.

[121] C. Stachniss and W. Burgard, "An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments," *2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 1, pp. 508–513, 2002.

[122] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *Robotics & Automation Magazine, IEEE*, vol. 4, pp. 23 – 33, 04 1997.

[123] J. Borenstein and Y. Koren, "The vector field histogram - fast obstacle avoidance for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 7, pp. 278 – 288, 07 1991.

[124] H. Hu and M. Brady, "A bayesian approach to real-time obstacle avoidance for a mobile robot," *Autonomous Robots*, vol. 1, pp. 69–92, 03 1994.

[125] R. Simmons, "The curvature-velocity method for local obstacle avoidance," *1996 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 4, pp. 3375–3382 vol.4, 1996.

[126] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *1985 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, pp. 500–505, March 1985.

[127] A. Beyeler, J.-C. Zufferey, and D. Floreano, "Vision-based control of near-obstacle flight," *Autonomous Robots*, vol. 27, no. 3, pp. 201–219, 2009.

[128] S. Hrabar and G. Sukhatme, "Vision-based navigation through urban canyons," *Journal of Field Robotics*, vol. 26, no. 5, pp. 431–452, 2009.

[129] J. Conroy, G. Gremillion, B. Ranganathan, and J. S. Humbert, "Implementation of wide-field integration of optic flow for autonomous quadrotor navigation," *Autonomous Robots*, vol. 27, no. 3, pp. 189–198, 2009.

[130] A. M. Hyslop and J. S. Humbert, "Autonomous navigation in three-dimensional urban environments using wide-field integration of optic flow," *Journal of guidance, control, and dynamics*, vol. 33, no. 1, pp. 147–159, 2010.

[131] I. Lenz, M. Gemici, and A. Saxena, "Low-power parallel algorithms for single image based obstacle avoidance in aerial robots,"

2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 772–779, 2012.

[132] H. Oleynikova, D. Honegger, and M. Pollefeys, "Reactive avoidance using embedded stereo vision for MAV flight," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 50–56, 2015.

[133] S. Scherer, S. Singh, L. Chamberlain, and M. Elgersma, "Flying fast and low among obstacles: Methodology and experiments," *The International Journal of Robotics Research*, vol. 27, no. 5, pp. 549–574, 2008.

[134] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys, "Vision-based autonomous mapping and exploration using a quadrotor MAV," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4557–4564, 2012.

[135] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, "Learning monocular reactive UAV control in cluttered natural environments," *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1765–1772, 2013.

[136] B. Yamauchi, "A frontier-based approach for autonomous exploration," *1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pp. 146–151, 1997.

[137] H. H. González-Banos and J.-C. Latombe, "Navigation strategies for exploring indoor environments," *The International Journal of Robotics Research*, vol. 21, no. 10-11, pp. 829–848, 2002.

[138] F. Amigoni, "Experimental evaluation of some exploration strategies for mobile robots," *2008 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2818–2823, 2008.

[139] M. Selin, M. Tiger, D. Duberg, F. Heintz, and P. Jensfelt, "Efficient autonomous exploration planning of large-scale 3-D envi-

ronments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1699–1706, 2019.

[140] L. Schmid, M. Pantic, R. Khanna, L. Ott, R. Siegwart, and J. Nieto, "An efficient sampling-based method for online informative path planning in unknown environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1500–1507, 2020.

[141] M. Pivtoraiko, D. Mellinger, and V. Kumar, "Incremental micro-UAV motion replanning for exploring unknown environments," *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2452–2458, 2013.

[142] S. Bouabdallah, A. Noth, and R. Siegwart, "PID vs LQ control techniques applied to an indoor micro quadrotor," *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 2451–2456 vol.3, 2004.

[143] D. Lee, H. Jin Kim, and S. Sastry, "Feedback linearization vs. adaptive sliding mode control for a quadrotor helicopter," *International Journal of Control, Automation and Systems*, vol. 7, no. 3, pp. 419–428, 2009.

[144] S. Bouabdallah and R. Siegwart, "Backstepping and sliding-mode techniques applied to an indoor micro quadrotor," *2005 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2247–2252, 2005.

[145] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor UAV on SE (3)," *2010 IEEE Conference on Decision and Control (CDC)*, pp. 5420–5425, 2010.

[146] D. Invernizzi and M. Lovera, "Trajectory tracking control of thrust-vectoring UAVs," *Automatica*, vol. 95, pp. 180–186, 2018.

[147] T. P. Nascimento and M. Saska, "Position and attitude control of multi-rotor aerial vehicles: A survey," *Annual Reviews in Control*, vol. 48, pp. 129–146, 2019.

[148] J. Ferrin, R. Leishman, R. Beard, and T. McLain, "Differential flatness based control of a rotorcraft for aggressive maneuvers," *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2688–2693, 2011.

[149] K. Sreenath, N. Michael, and V. Kumar, "Trajectory generation and control of a quadrotor with a cable-suspended load - a differentially-flat hybrid system," *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4888–4895, 2013.

[150] M. Bangura and R. Mahony, "Real-time model predictive control for quadrotors," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 11773–11780, 2014.

[151] M. Kamel, K. Alexis, M. Achtelik, and R. Siegwart, "Fast nonlinear model predictive control for multicopter attitude tracking on SO(3)," *2015 IEEE Conference on Control Applications (CCA)*, pp. 1160–1166, 2015.

[152] D. Bicego, J. Mazzetto, R. Carli, M. Farina, and A. Franchi, "Nonlinear model predictive control with enhanced actuator model for multi-rotor aerial vehicles with generic designs," *Journal of Intelligent & Robotic Systems*, vol. 100, no. 3, pp. 1213–1247, 2020.

[153] P. Foehn, A. Romero, and D. Scaramuzza, "Time-optimal planning for quadrotor waypoint flight," *Science Robotics*, vol. 6, no. 56, p. eabh1221, 2021.

[154] S. Sun, G. Cioffi, C. De Visser, and D. Scaramuzza, "Autonomous quadrotor flight despite rotor failure with onboard vision sensors: Frames vs. events," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 580–587, 2021.

[155] E. J. Smeur, Q. Chu, and G. C. De Croon, "Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 3, pp. 450–461, 2016.

[156] S. Sun, L. Sijbers, X. Wang, and C. de Visser, "High-speed flight of quadrotor despite loss of single rotor," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3201–3207, 2018.

[157] S. Sun, X. Wang, Q. Chu, and C. d. Visser, "Incremental nonlinear fault-tolerant control of a quadrotor with complete loss of two opposing rotors," *IEEE Transactions on Robotics*, vol. 37, no. 1, pp. 116–130, 2021.

[158] H. Lee and H. J. Kim, "Trajectory tracking control of multirotors from modelling to experiments: A survey," *International Journal of Control, Automation and Systems*, vol. 15, no. 1, pp. 281–292, 2017.

[159] H. Nguyen, M. Kamel, K. Alexis, and R. Siegwart, "Model predictive control for micro aerial vehicles: A survey," *2021 European Control Conference (ECC)*, pp. 1556–1563, 2021.

[160] G. Hoffmann, H. Huang, S. Waslander, and C. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," *AIAA Guidance, Navigation and Control Conference*, p. 6461, 2007.

[161] P. Martin and E. Salaün, "The true role of accelerometer feedback in quadrotor control," *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1623–1629, 2010.

[162] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *IEEE Robotics and Automation Magazine*, vol. 19, no. 3, pp. 20–32, 2012.

[163] M. Faessler, A. Franchi, and D. Scaramuzza, "Differential flatness of quadrotor dynamics subject to rotor drag for accurate tracking of high-speed trajectories," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 620–626, 2017.

[164] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "ROS: an open-source Robot Operating System," *ICRA workshop on open source software*, vol. 3, no. 3.2, p. 5, 2009.

[165] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 2149–2154, 2004.

[166] I. Afanasyev, A. Sagitov, and E. Magid, "ROS-based SLAM for a Gazebo-simulated mobile robot in image-based 3D model of indoor environment," *International Conference on Advanced Concepts for Intelligent Vision Systems*, pp. 273–283, 2015.

[167] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. v. Stryk, "Comprehensive simulation of quadrotor UAVs using ROS and Gazebo," *International conference on simulation, modeling, and programming for autonomous robots*, pp. 400–411, 2012.

[168] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "Rotors—a modular Gazebo MAV simulator framework," *Robot Operating System (ROS)*, pp. 595–625, 2016.

[169] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," *Conference on Robot Learning*, 2020.

[170] "MAVLink." `https://mavlink.io/en/`, Access date: 21/09/2022.

[171] "mavros." `http://wiki.ros.org/mavros`, Access date: 21/09/2022.

[172] "QGroundControl." `http://qgroundcontrol.com/`, Access date: 21/09/2022.

[173] "Leonardo Drone Contest autonomous drone competition." `https://www.leonardo.com/it/innovation-technology/open-innovation/drone-contest`, Access date: 21/09/2022.

[174] ANT-X website. `https://antx.it/`, Access date: 28/06/2022.

[175] "Stereolabs." https://www.stereolabs.com/, Access date: 16/10/2022.

[176] "OpenMV H7 Plus." https://openmv.io/products/openmv-cam-h7, Access date: 16/10/2022.

[177] "TeraRanger Tower EVO." https://www.terabee.com/shop/lidar-tof-range-finders/teraranger-tower-evo/, Access date: 16/10/2022.

[178] "PMW3901." https://docs.px4.io/main/en/sensor/pmw3901.html, Access date: 16/10/2022.

[179] "NanoPi NEO Air." http://wiki.friendlyarm.com/wiki/index.php/NanoPi_NEO_Air. Accessed: 05-11-2022.

[180] D. Allan, "Statistics of atomic frequency standards," *Proceedings of the IEEE*, vol. 54, no. 2, pp. 221–230, 1966.

[181] J. A. Farrell, F. O. Silva, F. Rahman, and J. Wendel, "IMU error state modeling for state estimation and sensor calibration: A tutorial," *Preprint of IEEE Control Systems Magazine*, 2020.

[182] D. C. Montgomery, *Design and analysis of experiments*. John Wiley & Sons, 2017.

[183] S. Ahrens, D. Levine, G. Andrews, and J. P. How, "Vision-based guidance and control of a hovering vehicle in unknown, GPS-denied environments," *2009 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2643–2648, 2009.

[184] W. Sankowski, M. Włodarczyk, D. Kacperski, and K. Grabowski, "Estimation of measurement uncertainty in stereo vision system," *Image and Vision Computing*, vol. 61, pp. 70–81, 2017.

[185] F. Fooladgar, S. Samavi, S. M. R. Soroushmehr, and S. Shirani, "Geometrical analysis of localization error in stereo vision systems," *IEEE Sensors Journal*, vol. 13, no. 11, pp. 4236–4246, 2013.

[186] Y. Xu, Y. Zhao, F. Wu, and K. Yang, "Error analysis of calibration parameters estimation for binocular stereo vision system," *2013 IEEE International Conference on Imaging Systems and Techniques (IST)*, pp. 317–320, 2013.

[187] K. Schreve, "How accurate can a stereovision measurement be?," *15th International Workshop on Research and Education in Mechatronics (REM)*, pp. 1–7, 2014.

[188] G. Di Leo, C. Liguori, and A. Paolillo, "Propagation of uncertainty through stereo triangulation," *2010 IEEE International Instrumentation Measurement Technology Conference*, pp. 12–17, 2010.

[189] G. Di Leo and A. Paolillo, "Uncertainty evaluation of camera model parameters," *2011 IEEE International Instrumentation and Measurement Technology Conference*, pp. 1–6, 2011.

[190] M. Kytö, M. Nuutinen, and P. Oittinen, "Method for measuring stereo camera depth accuracy based on stereoscopic vision," *Three-Dimensional Imaging, Interaction, and Measurement*, vol. 7864, pp. 168 – 176, 2011.

[191] S. Chiodini, M. Pertile, and S. Debei, "Visual odometry system performance for different landmark average distances," *2016 IEEE Metrology for Aerospace (MetroAeroSpace)*, pp. 382–387, 2016.

[192] R. Jiang, R. Klette, and S. Wang, "Statistical modeling of long-range drift in visual odometry," *Asian Conference on Computer Vision*, pp. 214–224, 2010.

[193] S. Farboud-Sheshdeh, T. D. Barfoot, and R. H. Kwong, "Towards estimating bias in stereo visual odometry," *2014 Canadian Conference on Computer and Robot Vision*, pp. 8–15, 2014.

[194] P. F. Alcantarilla and O. J. Woodford, "Noise models in feature-based stereo visual odometry," *Computing Research Repository (CoRR)*, vol. abs/1607.00273, 2016.

[195] IEEE, "IEEE standard specification format guide and test procedure for single-axis interferometric fiber optic gyros," *IEEE Std 952-1997*, pp. 1–84, 1998.

[196] N. El-Sheimy, H. Hou, and X. Niu, "Analysis and modeling of inertial sensors using Allan variance," *Instrumentation and Measurement, IEEE Transactions on*, vol. 57, pp. 140 – 149, 2008.

[197] N. Kasdin, "Discrete simulation of colored noise and stochastic processes and 1/f alpha power law noise generation," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 802–827, 1995.

[198] P. Lesage and C. Audoin, "Characterization of frequency stability: Uncertainty due to the finite number of measurements," *IEEE Transactions on Instrumentation and Measurement*, vol. 22, no. 2, pp. 157–161, 1973.

[199] A. Gelman, "Analysis of variance—why it is more important than ever," *The Annals of Statistics*, vol. 33, no. 1, pp. 1–53, 2005.

[200] A. Oustaloup, F. Levron, B. Mathieu, and F. Nanot, "Frequency-band complex noninteger differentiator: characterization and synthesis," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 1, pp. 25–39, 2000.

[201] P. Welch, "The use of fast Fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms," *IEEE Transactions on Audio and Electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967.

[202] L. O. Rojas-Perez and J. Martínez-Carranza, "On-board processing for autonomous drone racing: an overview," *Integration*, vol. 80, pp. 46–59, 2021.

[203] E. Kaufmann, M. Gehrig, P. Foehn, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Beauty and the beast: Optimal methods meet learning for drone racing," *2019 International Conference on Robotics and Automation (ICRA)*, pp. 690–696, 2019.

[204] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," *Field and Service Robotics*, pp. 621–635, 2018.

[205] C. De Wagter, F. Paredes-Vallés, N. Sheth, and G. Croon, "The sensing, state-estimation, and control behind the winning entry to the 2019 artificial intelligence robotic racing competition," *Field Robotics*, vol. 2, pp. 1263–1290, 03 2022.

[206] P. Foehn, D. Brescianini, E. Kaufmann, T. Cieslewski, M. Gehrig, M. Muglikar, and D. Scaramuzza, "Alphapilot: Autonomous drone racing," *Autonomous Robots*, pp. 1–14, 2021.

[207] A. Rezende, V. R. Miranda, P. A. Rezeck, H. Azpúrua, E. R. Santos, V. M. Gonçalves, D. G. Macharet, and G. M. Freitas, "An integrated solution for an autonomous drone racing in indoor environments," *Intelligent Service Robotics*, vol. 14, no. 5, pp. 641–661, 2021.

[208] R. Madaan, N. Gyde, S. Vemprala, M. Brown, K. Nagami, T. Taubner, E. Cristofalo, D. Scaramuzza, M. Schwager, and A. Kapoor, "Airsim drone racing lab," *NeurIPS 2019 Competition and Demonstration Track*, pp. 177–191, 2020.

[209] H. J. Escalante and R. Hadsell, "NeurIPS 2019 competition and demonstration track: revised selected papers," *NeurIPS 2019 Competition and Demonstration Track*, pp. 1–12, 2020.

[210] D. Kim, H. Ryu, J. Yonchorhor, and D. H. Shim, "A deep-learning-aided automatic vision-based control approach for autonomous drone racing in game of drones competition," *NeurIPS 2019 Competition and Demonstration Track*, pp. 37–46, 2020.

[211] F. Ölsner and S. Milz, "Catch me, if you can! a mediated perception approach fully autonomous drone racing," *NeurIPS 2019 Competition and Demonstration Track*, pp. 90–99, 2020.

[212] S. Shin, Y. Kang, and Y.-G. Kim, "Evolution algorithm and online learning for racing drone," *NeurIPS 2019 Competition and Demonstration Track*, pp. 100–109, 2020.

[213] C. Sampedro, H. Bavle, A. Rodríguez-Ramos, A. Carrio, R. A. S. Fernández, J. L. Sanchez-Lopez, and P. Campoy, "A fully-autonomous aerial robotic solution for the 2016 international micro air vehicle competition," *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 989–998, 2017.

[214] A. Farooq, A. Anastasiou, N. Souli, C. Laoudias, P. S. Kolios, and T. Theocharides, "UAV autonomous indoor exploration and mapping for SAR missions: Reflections from the ICUAS 2022 competition," *2022 19th International Conference on Ubiquitous Robots (UR)*, pp. 621–626, 2022.

[215] "International Aerial Robotics Competition (IARC)." `http://www.aerialroboticscompetition.org/pastmissions.php`, Access date: 26/09/2022.

[216] D. Ellis, T. Brady, I. Olson, and Y. Li, "Autonomous quadrotor for the 2012 international aerial robotics competition," *2012 Third Symposium on Indoor Flight Issues, International Aerial Robotics Competition*, 2012.

[217] J. L. Sanchez-Lopez, J. Pestana, J.-F. Collumeau, R. Suarez-Fernandez, P. Campoy, and M. Molina, "A vision based aerial robot solution for the mission 7 of the international aerial robotics competition," *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1391–1400, 2015.

[218] M. Deegan, A. Dziedzic, C. Jiang, R. Moon, D. Pisarski, and R. Wunderly, "Autonomous quadrotors for the 2019 international aerial robotics competition," *Proceedings of the International Aerial Robotics Competition*, pp. 1–11, 2019.

[219] R. Bähnemann, M. Pantic, M. Popović, D. Schindler, M. Tranzatto, M. Kamel, M. Grimm, J. Widauer, R. Siegwart, and J. Nieto, "The ETH-MAV team in the MBZ international robotics challenge," *Journal of Field Robotics*, vol. 36, no. 1, pp. 78–103, 2019.

[220] M. Beul, M. Nieuwenhuisen, J. Quenzel, R. A. Rosu, J. Horn, D. Pavlichenko, S. Houben, and S. Behnke, "Team NimbRo at

MBZIRC 2017: Fast landing on a moving target and treasure hunting with a team of micro aerial vehicles," *Journal of Field Robotics*, vol. 36, no. 1, pp. 204–229, 2019.

[221] Á. R. Castaño, F. Real, P. Ramón-Soria, J. Capitán, V. Vega, B. C. Arrue, A. Torres-González, and A. Ollero, "Al-Robotics team: A cooperative multi-unmanned aerial vehicle approach for the Mohamed Bin Zayed International Robotic Challenge," *Journal of Field Robotics*, vol. 36, no. 1, pp. 104–124, 2019.

[222] M. Vrba, Y. Stasinchuk, T. Báča, V. Spurnỳ, M. Petrlík, D. Heřt, D. Žaitlík, and M. Saska, "Autonomous capture of agile flying objects using UAVs: The MBZIRC 2020 challenge," *Robotics and Autonomous Systems*, vol. 149, p. 103970, 2022.

[223] M. Beul, S. Bultmann, A. Rochow, R. A. Rosu, D. Schleich, M. Splietker, and S. Behnke, "Visually guided balloon popping with an autonomous MAV at MBZIRC 2020," *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 34–41, 2020.

[224] C. Lenz, M. Schwarz, A. Rochow, J. Razlaw, A. S. Periyasamy, M. Schreiber, and S. Behnke, "Autonomous wall building with a UGV-UAV team at MBZIRC 2020," *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 189–196, 2020.

[225] V. Walter, V. Spurný, M. Petrlík, T. Báča, D. Žaitlík, and M. Saska, "Extinguishing of ground fires by fully autonomous UAVs motivated by the MBZIRC 2020 competition," *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 787–793, 2021.

[226] M. Labbé and F. Michaud, "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, no. 2, pp. 416–446, 2019.

[227] M. Labbé and F. Michaud, "Appearance-based loop closure detection for online large-scale and long-term operation," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 734–745, 2013.

[228] I. Ulrich and J. Borenstein, "VFH*: Local obstacle avoidance with look-ahead verification," *2000 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3, pp. 2505–2511, 2000.

[229] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3D reconstruction in real-time," *IEEE Intelligent Vehicles Symposium (IV)*, pp. 963–968, 2011.

[230] "Zed sdk." `https://www.stereolabs.com/developers/release/`, Access date: 24/09/2022.

[231] R. Giubilato, S. Chiodini, M. Pertile, and S. Debei, "An evaluation of ROS-compatible stereo visual SLAM methods on a nVidia Jetson TX2," *Measurement*, 04 2019.

[232] I. Z. Ibragimov and I. M. Afanasyev, "Comparison of ROS-based visual SLAM methods in homogeneous indoor environment," *2017 14th Workshop on Positioning, Navigation and Communications (WPNC)*, pp. 1–6, 2017.

[233] A. Quattrini Li, A. Coskun, S. M. Doherty, S. Ghasemlou, A. S. Jagtap, M. Modasshir, S. Rahman, A. Singh, M. Xanthidis, J. M. O'Kane, *et al.*, "Experimental comparison of open source vision-based state estimation algorithms," *International Symposium on Experimental Robotics*, pp. 775–786, 2016.

[234] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2502–2509, 2018.

[235] "Intel D435i." `https://www.intelrealsense.com/depth-camera-d435i/`, Access date: 14/09/2022.

[236] "Intel T265." `https://www.intelrealsense.com/tracking-camera-t265/`, Access date: 14/09/2022.

[237] V. Tadic, A. Toth, Z. Vizvari, M. Klincsik, Z. Sari, P. Sarcevic, J. Sarosi, and I. Biro, "Perspectives of realsense and zed depth sensors for robotic vision applications," *Machines*, vol. 10, no. 3, 2022.

[238] D. Stronger and P. Stone, "A comparison of two approaches for vision and self-localization on a mobile robot," *2007 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3915–3920, 2007.

[239] W. Hong, C. Zhou, and Y. Tian, "Robust Monte Carlo Localization for humanoid soccer robot," *2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pp. 934–939, 2009.

[240] A. Sousa, P. Costa, A. Moreira, and A. Carvalho, "Self localization of an autonomous robot: using an EKF to merge odometry and vision based landmarks," *2005 IEEE Conference on Emerging Technologies and Factory Automation*, vol. 1, pp. 7 pp.–233, 2005.

[241] "ROS ViSP visual servoing library." `http://wiki.ros.org/visp_auto_tracker`, Access date: 14/09/2022.

[242] I. Ulrich and J. Borenstein, "VFH+: Reliable obstacle avoidance for fast mobile robots," *1998 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, pp. 1572–1577, 1998.

[243] S. Vanneste, B. Bellekens, and M. Weyn, "3DVFH+: Real-time three-dimensional obstacle avoidance using an Octomap," *MORSE 2014 Model-Driven Robot Software Engineering*, no. 1319, pp. 91–102, 2014.

[244] A. Babinec, F. Duchoň, M. Dekan, Z. Mikulová, and L. Jurišica, "Vector Field Histogram* with look-ahead tree extension dependent on time variable environment," *Transactions of the Institute of Measurement and Control*, vol. 40, no. 4, pp. 1250–1264, 2018.

[245] A. Santamaria-Navarro, G. Loianno, J. Sola, V. Kumar, and J. Andrade-Cetto, "Autonomous navigation of micro aerial vehicles using high-rate and low-cost sensors," *Autonomous Robots*, vol. 42, no. 6, pp. 1263–1280, 2018.

[246] N. Basilico, "Recent trends in robotic patrolling," *Current Robotics Reports*, pp. 1–12, 2022.

[247] S. Hoshino, S. Ugajin, and T. Ishiwata, "Patrolling robot based on bayesian learning for multiple intruders," *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 603–609, 2015.

[248] S. Hoshino, T. Ishiwata, and R. Ueda, "Optimal patrolling methodology of mobile robot for unknown visitors," *Advanced Robotics*, vol. 30, no. 16, pp. 1072–1085, 2016.

[249] H. González-Banos, "A randomized art-gallery algorithm for sensor placement," *Proceedings of the seventeenth annual symposium on Computational geometry*, pp. 232–240, 2001.

[250] E. Semsch, M. Jakob, D. Pavlicek, and M. Pechoucek, "Autonomous UAV surveillance in complex urban environments," *IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, vol. 2, pp. 82–85, 2009.

[251] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," *AAAI/IAAI Conference on Artificial Intelligence*, vol. 1999, no. 343-349, pp. 2–2, 1999.

[252] D. Fox, "KLD-sampling: Adaptive particle filters," *Advances in neural information processing systems*, vol. 14, 2001.

[253] "PX4FLOW." `https://docs.px4.io/main/en/sensor/px4flow.html`, Access date: 16/10/2022.

[254] J. Borer and M. Pryor, "Continuous hybrid localization in environments with physical and temporal sensor occlusions," *2021 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 118–124, 2021.

[255] M. S. Alam and J. Oluoch, "A survey of safe landing zone detection techniques for autonomous unmanned aerial vehicles (UAVs)," *Expert Systems with Applications*, vol. 179, p. 115091, 2021.

[256] M. Garg, A. Kumar, and P. Sujit, "Terrain-based landing site selection and path planning for fixed-wing UAVs," *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 246–251, 2015.

[257] M. Aydin and E. Kugu, "Finding smoothness area on the topographic maps for the unmanned aerial vehicle's landing site estimation," *2016 Sixth International Conference on Digital Information and Communication Technology and its Applications (DICTAP)*, pp. 183–186, 2016.

[258] S. Scherer, L. Chamberlain, and S. Singh, "Autonomous landing at unprepared sites by a full-scale helicopter," *Robotics and Autonomous Systems*, vol. 60, no. 12, pp. 1545–1562, 2012.

[259] L. Chamberlain, S. Scherer, and S. Singh, "Self-aware helicopters: Full-scale automated landing and obstacle avoidance in unmapped environments," *67th Annual Forum of the American Helicopter Society, Virginia Beach, VA*, vol. 2, 2011.

[260] O. G. Lorenzo, J. Martínez, D. L. Vilariño, T. F. Pena, J. C. Cabaleiro, and F. F. Rivera, "Landing sites detection using LiDAR data on manycore systems," *The Journal of Supercomputing*, vol. 73, no. 1, pp. 557–575, 2017.

[261] G. Loureiro, A. Dias, A. Martins, and J. Almeida, "Emergency landing spot detection algorithm for unmanned aerial vehicles," *Remote Sensing*, vol. 13, no. 10, p. 1930, 2021.

[262] S. Bosch, S. Lacroix, and F. Caballero, "Autonomous detection of safe landing areas for an UAV from monocular images," *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5522–5527, 2006.

[263] R. Brockers, P. Bouffard, J. Ma, L. Matthies, and C. Tomlin, "Autonomous landing and ingress of micro-air-vehicles in urban environments based on monocular vision," *Micro-and Nanotechnology Sensors, Systems, and Applications III*, vol. 8031, p. 803111, 2011.

[264] L. Mejias and D. Fitzgerald, "A multi-layered approach for site detection in uas emergency landing scenarios using geometry-based image segmentation," *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 366–372, 2013.

[265] Y.-F. Shen, Z.-U. Rahman, D. Krusienski, and J. Li, "A vision-based automatic safe landing-site detection system," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 1, pp. 294–311, 2013.

[266] T. Hinzmann, T. Stastny, C. Cadena, R. Siegwart, and I. Gilitschenski, "Free lsd: Prior-free visual landing site detection for autonomous planes," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2545–2552, 2018.

[267] P. J. Garcia-Pardo, G. S. Sukhatme, and J. F. Montgomery, "Towards vision-based safe landing for an autonomous helicopter," *Robotics and Autonomous Systems*, vol. 38, no. 1, pp. 19–29, 2002.

[268] M. Mittal, R. Mohan, W. Burgard, and A. Valada, "Vision-based autonomous UAV navigation and landing for urban search and rescue," *The International Symposium of Robotics Research*, pp. 575–592, 2019.

[269] P. C. Lusk, P. C. Glaab, L. J. Glaab, and R. W. Beard, "Safe2ditch: Emergency landing for small unmanned aircraft systems," *Journal of Aerospace Information Systems*, vol. 16, no. 8, pp. 327–339, 2019.

[270] L. Bartolomei, Y. Kompis, L. Pinto Teixeira, and M. Chli, "Autonomous emergency landing for multicopters using deep reinforcement learning," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[271] P. Váňa, J. Sláma, J. Faigl, and P. Pačes, "Any-time trajectory planning for safe emergency landing," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5691–5696, 2018.

[272] N. F. Meuleau, C. J. Plaunt, D. E. Smith, and T. B. Smith, "An emergency landing planner for damaged aircraft," *AAAI/IAAI Conference on Artificial Intelligence*, 2009.

[273] E. M. Atkins, I. A. Portillo, and M. J. Strube, "Emergency flight planning applied to total loss of thrust," *Journal of Aircraft*, vol. 43, no. 4, pp. 1205–1216, 2006.

[274] S. Choudhury, S. Scherer, and S. Singh, "RRT*-AR: Sampling-based alternate routes planning with applications to autonomous emergency landing of a helicopter," *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3947–3952, 2013.

[275] X. Wu and M. Mueller, "Towards a consequences-aware emergency landing system for unmanned aerial systems," *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1025–1030, 2018.

[276] V. R. Desaraju, N. Michael, M. Humenberger, R. Brockers, S. Weiss, J. Nash, and L. Matthies, "Vision-based landing site evaluation and informed optimal trajectory generation toward autonomous rooftop landing," *Autonomous Robots*, vol. 39, no. 3, pp. 445–463, 2015.

[277] S. Primatesta, G. Guglieri, and A. Rizzo, "A risk-aware path planning strategy for UAVs in urban environments," *Journal of Intelligent & Robotic Systems*, vol. 95, no. 2, pp. 629–643, 2019.

[278] G. Gozzini, D. Invernizzi, S. Panza, M. Giurato, and M. Lovera, "Air-to-Air Automatic Landing of unmanned aerial vehicles: A quasi time-optimal hybrid strategy," *IEEE Control Systems Letters*, vol. 4, no. 3, pp. 692–697, 2020.

[279] D. Lee, T. Ryan, and H. J. Kim, "Autonomous landing of a VTOL UAV on a moving platform using image-based visual servoing," *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 971–976, 2012.

[280] J. Lin, Y. Wang, Z. Miao, H. Zhong, and R. Fierro, "Low-complexity control for vision-based landing of quadrotor UAV on

unknown moving platform," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2021.

[281] D. Falanga, A. Zanchettin, A. Simovic, J. Delmerico, and D. Scaramuzza, "Vision-based autonomous quadrotor landing on a moving platform," *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pp. 200–207, 2017.

[282] P. Vlantis, P. Marantos, C. Bechlioulis, and K. Kyriakopoulos, "Quadrotor landing on an inclined platform of a moving ground vehicle," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2202–2207, 2015.

[283] J. Kim, Y. Jung, D. Lee, and D. H. Shim, "Landing control on a mobile platform for multi-copters using an omnidirectional image sensor," *Journal of Intelligent & Robotic Systems*, vol. 84, no. 1, pp. 529–541, 2016.

[284] T. Báča, P. Štěpán, V. Spurný, D. Hert, R. Pěnička, M. Saska, J. Thomas, G. Loianno, and V. Kumar, "Autonomous landing on a moving vehicle with an unmanned aerial vehicle," *Journal of Field Robotics*, vol. 36, no. 5, pp. 874–891, 2019.

[285] A. Borowczyk, N. Tien, A. Nguyen, D. Nguyen, D. Saussie, and J. Le Ny, "Autonomous landing of a multirotor micro air vehicle on a high velocity ground vehicle," *IFAC Papers Online*, vol. 50, no. 1, pp. 10488–10494, 2017.

[286] K. Guo, P. Tang, H. Wang, D. Lin, and X. Cui, "Autonomous landing of a quadrotor on a moving platform via model predictive control," *MDPI Aerospace*, vol. 9, no. 1, 2022.

[287] G. Niu, Q. Yang, Y. Gao, and M.-O. Pun, "Vision-based autonomous landing for unmanned aerial and ground vehicles cooperative systems," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6234–6241, 2021.

[288] N. J. Sanket, C. D. Singh, C. M. Parameshwara, C. Fermüller, G. C. H. E. de Croon, and Y. Aloimonos, "EVPropNet: Detecting

drones by finding propellers for mid-air landing and following," *2021 Robotics. Science and Systems (RSS)*, 2021.

[289] K. P. Jain and M. W. Mueller, "Flying batteries: In-flight battery switching to increase multirotor flight time," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3510–3516, 2020.

[290] P. Giuri, A. Marini Cossetti, M. Giurato, D. Invernizzi, and M. Lovera, "Air-to-air automatic landing for multirotor UAVs," *2019 CEAS Conference on Guidance, Navigation and Control (EuroGNC)*, 2019.

[291] M.-D. Hua, T. Hamel, P. Morin, and C. Samson, "Introduction to feedback control of underactuated VTOL vehicles: A review of basic control design ideas and principles," *IEEE Control Systems Magazine*, vol. 33, no. 1, pp. 61–75, 2013.

[292] D. Invernizzi, M. Lovera, and L. Zaccarian, "Dynamic attitude planning for trajectory tracking in thrust-vectoring UAVs," *IEEE Transactions on Automatic Control*, vol. 65, no. 1, pp. 453–460, 2020.

[293] F. Forni, S. Galeani, and L. Zaccarian, "A family of global stabilizers for quasi-optimal control of planar linear saturated systems," *IEEE Transactions on Automatic Control*, vol. 55, no. 5, pp. 1175–1180, 2010.

[294] R. Goebel, R. G. Sanfelice, and A. R. Teel, *Hybrid dynamical systems*. Princeton University Press, 2012.

[295] "OpenCV." `http://https://opencv.org/`, Access date: 14/09/2022.

[296] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[297] "PX4 Github page." `https://github.com/PX4/PX4-ECL/tree/master/EKF/documentation`, Access date: 06/05/2022.

[298] A. Khosravian, J. Trumpf, R. Mahony, and T. Hamel, "Recursive attitude estimation in the presence of multi-rate and multi-delay vector measurements," *2015 American Control Conference (ACC)*, pp. 3199–3205, 2015.

[299] J. Sola, "Course on SLAM," *Technical Report IRI-TR-16-04, Institut de Robòtica i Informàtica Industrial*, 2017.

[300] E. Gagliardi, "Pose estimation of multi-stereo camera sensors with non overlapping fields of view," *M.Sc. thesis, Politecnico di Milano*, 2019.

[301] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

APPENDIX $\mathcal{A}$

EKF2

In this Appendix, the Extended Kalman Filter (EKF) algorithm used by PX4 [35] is presented. The filter is able to process a variety of sensor measurements and output a state vector composed by the following 24 elements:

- Quaternion q defining the rotation from NED local reference to body frame
- Velocity in NED v local reference frame (m/s)
- Position in NED p local reference frame (m)
- IMU delta angle bias $\Delta\Phi_b$ in body frame (rad)
- IMU delta velocity bias Δv_b in body frame (m/s)
- Earth magnetic field m in NED (gauss)

- Vehicle body frame magnetic field bias m_b (gauss)
- Wind velocity in NE v_w (m/s)

The overall architecture of the algorithm is presented in Figure A.1.

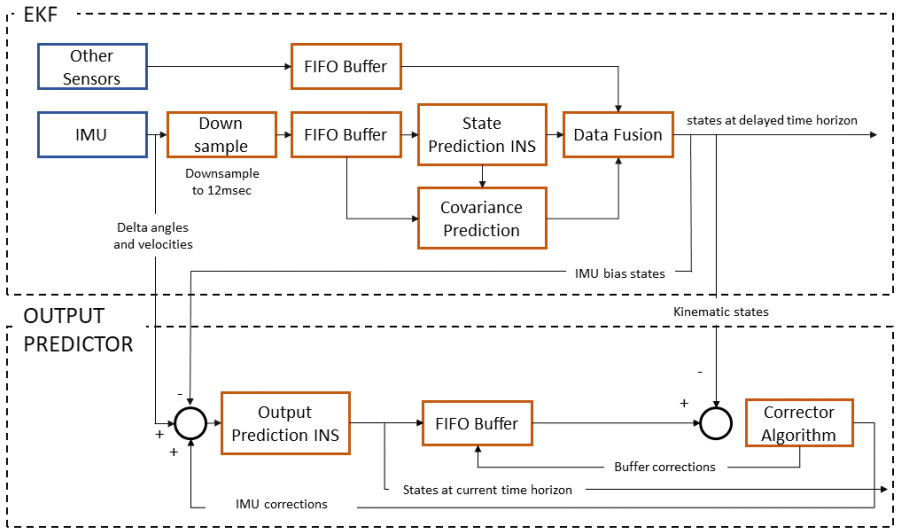


Figure A.1: *EKF2 scheme (reworked from [297])*

It consists of two separate parts: the actual EKF and an *output predictor*. In turn, the output predictor is composed by an actual prediction step and a corrector algorithm, which is actually an observer. This structure has been inspired by the work of [298]. This paper proposed a cascade combination of a predictor with an attitude observer (or filter), in which the predictor compensated for the effects of delays, while the observer processed the predicted outputs, estimating the actual attitude. In the paper, it was also showed that the predictor could be coupled with any asymptotically stable observer or filter. In particular, the EKF2 implementation employs a complementary filter.

As depicted in Figure A.1, the EKF takes a downsampled version of the IMU and buffers it using a FIFO (first in-first out) buffer. The same is done for all the other sensor data, *i.e.*, magnetometer, barometer, optical

flow and vision data. In order to allow for different time delay relative to the IMU data, the EKF works in a delayed time horizon. The time horizon delay and, consequently, the length of the buffers are determined by the largest expected time delay of the sensors (tunable input parameters). The EKF retrieves sensors data to be used at the correct time (time of arrival of the measurements minus the corresponding expected time delay) and fuse them to correct the predicted state and covariances.

A.1 Prediction

For what concerns the prediction step of the EKF, an Inertial Navigation System (INS) prediction is used. First, the measured acceleration and angular velocities are integrated numerically for obtaining what are called in Figure A.1 "delta angles" and "delta velocities".

$$\Delta v_k = \int_{t_k}^{t_{k+1}} a(t)dt \tag{A.1}$$

$$\Delta \Phi_k = \int_{t_k}^{t_{k+1}} \omega(t)dt. \tag{A.2}$$

Then the IMU bias (part of the state) are removed, leading to a "corrected version" of these quantities.

$$\Delta v_{c_k} = \Delta v_k - \Delta v_{b_k} \tag{A.3}$$

$$\Delta \Phi_{c_k} = \Delta \Phi_k - \Delta \Phi_{b_k}. \tag{A.4}$$

After having updated $\Delta \Phi_{c_k}$ by removing the Earth angular velocity, the predicted quaternion is computed through:

$$q_{k+1} = q_k \otimes q\{\Delta \Phi_{c_k}\}. \tag{A.5}$$

The quaternion is then converted into the rotation matrix $R_{\mathcal{IB}}$. This latter is employed for rotating the corrected velocity:

$$\Delta v_I = R_{\mathcal{IB}} \Delta v_{c_k}. \tag{A.6}$$

Then, the corrected velocity, written in the inertial frame $\mathcal{I}$, is compensated for the acceleration of gravity g:

$$\Delta v_I = \Delta v_I - g\Delta t, \tag{A.7}$$

where Δt is the filter update time: $\Delta t = t_{k+1} - t_k$. Finally the velocity and position predictions are computed as:

$$v_{k+1} = v_k + \Delta v_I \tag{A.8}$$

$$p_{k+1} = p_k + \frac{v_k + v_{k+1}}{2}\Delta t. \tag{A.9}$$

A.2 Correction

Each sensor is fused using an appropriate measurement model. Note that IMU data is not used as an observation in the filter.

For the GPS position and velocity, barometer and vision-based sensors position the trivial observation model is used. Basically, in those cases, a direct observation of the states is performed.

A.2.1 Magnetometer

The magnetometer is assumed to be aligned with the body frame. It experiments a magnetic field vector which is written as the sum of a field, expressed in the inertial frame and rotated through the appropriate transformation, and the bias, directly written in the body frame.

$$M_m = R_{\mathcal{IB}}^{\top} m + m_b. \tag{A.10}$$

The earth frame magnetic field declination is also used as an observation (used to keep correct heading when operating without absolute position or velocity measurements)

$$\psi_{decl} = \arctan\left(\frac{m_E}{m_N}\right). \tag{A.11}$$

A.2.2 Optical flow

Optical flow observation equation assumes a sensor aligned with the z-body frame at a distance D from a stationary scene in the navigation frame. For what concerns the signs of the flow, consider a Forward-Right-Down reference frame associated with the flow sensor. For the flow in the x direction:

- A sensor right-hand rotation about the x axis induces a positive flow.
- A sensor linear motion along the positive y axis induces a negative flow.

For the flow in the y direction:

- A sensor right-hand rotation about the y axis induces a positive flow.
- A sensor linear motion along the positive x axis induces a positive flow.

Thus, if we consider the x-axis of the sensor aligned with the x-axis of the drone body frame b_1, we get:

$$flow_x = -\frac{v_{b_y}}{D} \tag{A.12}$$

$$flow_y = \frac{v_{b_x}}{D}, \tag{A.13}$$

with

$$v_b = R_{\mathcal{IB}}^{\top} v. \tag{A.14}$$

A.2.3 Distance sensor

The distance sensor can be used in different modalities. If it is the primary source of height estimation, the measurement model simply corresponds to the observation of the state. If the range sensor is employed after having lost another source of measurement, *e.g.*, Global Positioning System (GPS), an height sensor offset is computed in such a way that the current

measurement matches the current height estimate. The height offset h_{off} is then employed in the measurement model:

$$z_{range} = p_z + h_{off}. \tag{A.15}$$

For the documentation relative to the different available modes of range finders see [35].

A.2.4 Vision-based sensors velocity

Visual odometry observation equation assumes measurement of velocity states rotated into the body frame.

$$v_b = R_{\mathcal{IB}}^{\top} v. \tag{A.16}$$

A.2.5 Wind velocity

The airspeed observation equation assumes a sensor that measures the magnitude of velocity relative to the wind field:

$$\begin{bmatrix} v_{rel_N} \\ v_{rel_E} \\ v_{rel_D} \end{bmatrix} = \begin{bmatrix} v_N \\ v_E \\ v_D \end{bmatrix} - \begin{bmatrix} v_{w_N} \\ v_{w_E} \\ 0 \end{bmatrix}, \tag{A.17}$$

and:

$$TAS = \sqrt{v_{rel_N}^2 + v_{rel_E}^2 + v_{rel_D}^2}. \tag{A.18}$$

A.3 Output prediction

On the other hand, the output predictor takes as input the IMU measurements at the current time (and at full rate) and applies an Inertial Navigation System (INS) prediction similar to the one presented for the EKF. The IMU biases to be used in the prediction model are taken directly from the EKF. The only difference resides in the fact that a contribute $\Delta\Phi_{corr}$, output of the complementary filter, is summed to $\Delta\Phi_c$ of equation (A.4), namely:

$$\Delta\Phi_c = \Delta\Phi_c + \Delta\Phi_{corr}. \tag{A.19}$$

The states, output of the predictor, are then stored in a FIFO buffer. This is done to allow to compute the differences between the EKF states and the predicted ones at the delayed fusion time horizon. Clearly, the predictor will retrieve the oldest predicted states belonging to the FIFO buffer for performing this operation.

It is worth noting that the predicted state is also the one employed for computing the feedback of the control laws. This is done for achieving minimum latency between the IMU measurements and the computation of the state (less than $100\,\mu$s).

The differences obtained are then used for computing the corrections Δp_{corr}, Δv_{corr} and $\Delta\Phi_{corr}$ in the complementary filter. As already seen for the attitude in equation (A.19), these corrections are then applied at the current time horizon so that the predicted states track the EKF states at the delayed fusion time horizon. For what concerns position and velocity corrections, they are applied to all the elements of the FIFO buffer. This is not done for the attitude states due to the quaternion operations that would be needed and their associated computational costs.

In conclusion, the output predictor is used to propagate the states forward, from the fusion time horizon to current time, using the buffered IMU data. Doing so, the predictor filters out the sudden changes in the states that occur when measurements are fused. Note that the time constants of the complementary filter are set as inputs of the system.

APPENDIX $\mathcal{B}$

Monocular camera model

A monocular camera is a projective sensor that associates points in the 3D space with points on a 2D image plane. Each projected point (pixel) will contain photometric information related to the photometric properties of the external 3D point [299]. The behaviour of the monocular camera is model by the so-called *pinhole camera* model. A pinhole camera (shown in Figure B.1) consists of an *optical center* C, an *optical axis* and a plane, called *image plane*. The image plane is perpendicular to the optical axis and it is situated at a distance f from the optical center, called *focal length*. The point where the optical axis intersects the image plane is called *principal point*. Conventionally, the camera reference frame has its origin in the optical center, with the c_3-axis aligned with the optical axis.

Following the approach of [299], we now derive an algebraic representation of the pinhole camera, as a mapping from a point in 3D space $\pi_{\mathcal{C}} = [x_{\mathcal{C}}, y_{\mathcal{C}}, z_{\mathcal{C}}]^\top$, called *object point* to a 2D point $P = [X, Y]^\top$, be-

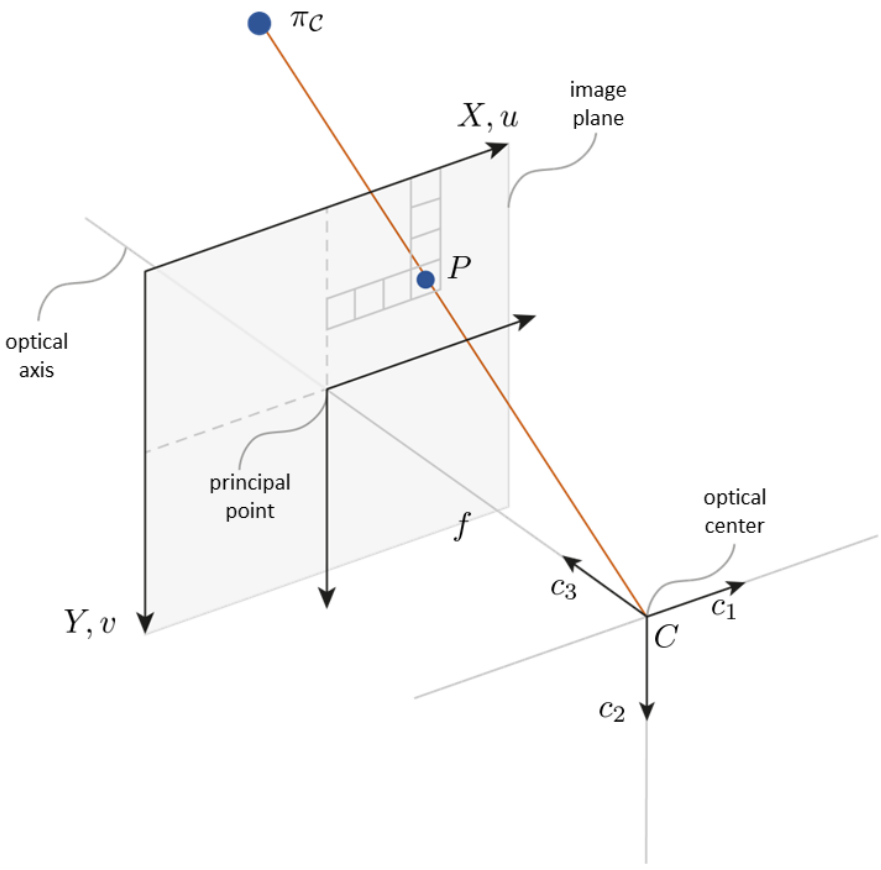


Figure B.1: *Pin-hole camera model (reworked from [300])*

longing to the image plane, called *image point*. This operation can be done simply applying triangle similarities, *i.e.*,

$$\frac{X}{f} = \frac{x_{\mathcal{C}}}{z_{\mathcal{C}}} \tag{B.1}$$

$$\frac{Y}{f} = \frac{y_{\mathcal{C}}}{z_{\mathcal{C}}}, \tag{B.2}$$

leading to the expression:

$$P = \begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} x_{\mathcal{C}} \\ y_{\mathcal{C}} \end{bmatrix} \frac{f}{z_{\mathcal{C}}}. \tag{B.3}$$

This equation is usually written in homogeneous coordinates:

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \bar{P} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\mathcal{C}} \\ y_{\mathcal{C}} \\ z_{\mathcal{C}} \end{bmatrix} = K_f \pi_{\mathcal{C}}, \tag{B.4}$$

where $\bar{P}$ denotes the use of homogeneous coordinates. The next step is to represent image point P in pixel instead of metric units, namely $P^{pix} = [u, v]^\top$. Suppose that the principal point, in pixels, is indicated as $P_0^{pix} = [u_0, v_0]^\top$ and that s_u and s_v are the number of pixels per distance unit in the horizontal and vertical direction, then the following relations hold:

$$u = u_o + s_u X \tag{B.5}$$

$$v = v_0 + s_v Y, \tag{B.6}$$

These relations can be also put in homogeneous coordinates as:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \bar{P}^{pix} = \begin{bmatrix} s_u & 0 & u_0 \\ 0 & s_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = K_s \bar{P}. \tag{B.7}$$

Concatenating the two results, we obtain:

$$\bar{P}^{pix} = K_s K_f \pi_{\mathcal{C}} = K \pi_{\mathcal{C}}, \tag{B.8}$$

where

$$K = K_s K_f = \begin{bmatrix} f_u & 0 & p_u \\ 0 & f_v & p_v \\ 0 & 0 & 1 \end{bmatrix}, \tag{B.9}$$

is called *calibration matrix* or *intrinsic matrix*, and its parameters are called *intrinsic parameters*, because they only depends on the camera internal configuration. In the expression of the calibration matrix $f_u = f s_u$ and $f_v = f s_v$ are interpreted as the focal length measured respectively in terms of the horizontal and vertical pixels.

On the other hand, the *extrinsic parameters* are represented by the camera pose $T_{\mathcal{IC}} = (t_{\mathcal{IC}}, R_{\mathcal{IC}})$. From their knowledge, the coordinates of the object point in the inertial frame $\pi_{\mathcal{I}}$ can be retrieved. Thus, the complete pin-hole camera model result:

$$\bar{P}^{pix} = K R_{\mathcal{IC}}^{\top} (\pi_{\mathcal{I}} - t_{\mathcal{IC}}). \tag{B.10}$$

Finally, we retrieve the Cartesian coordinates of the projected pixels from its homogeneous coordinates as:

$$P^{pix} = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \bar{P}_1^{pix} / \bar{P}_3^{pix} \\ \bar{P}_2^{pix} / \bar{P}_3^{pix} \end{bmatrix}, \tag{B.11}$$

where the sub-index i indicates the i-th component of the considered vector. Equation (B.11) can be unpacked into the single components as:

$$u = \frac{f_u}{z_{\mathcal{C}}} x_{\mathcal{C}} + u_0 \tag{B.12}$$

$$v = \frac{f_v}{z_{\mathcal{C}}} y_{\mathcal{C}} + v_0. \tag{B.13}$$

Note that a monocular camera is not able to measure the distance to the perceived objects $z_{\mathcal{C}}$. This leads to the impossibility of inverting equations (B.11). In other words, given an image point, we are not able to recover the object point that generated it. However, we can find the locus of object points projecting to the pixel of interest, as a semi-infinite straight line parameterized by the unmeasured distance r:

$$\pi_{\mathcal{I}} = t_{\mathcal{IC}} + rR_{\mathcal{IC}}v_{\mathcal{C}}, \tag{B.14}$$

where

$$v_{\mathcal{C}} = K^{-1}\bar{P}^{pix}/||K^{-1}\bar{P}^{pix}||. \tag{B.15}$$

The pinhole model can be thought as a linear approximation of real cameras. Indeed, imperfections in lens manufacturing cause the presence of nonlinear effects in the mapping from object points to image points, called *distortion*. The most common kinds of distortion effect are *radial* and *tangential*.

Before describing and applying the distortion models, we represent points in normalized image coordinates, namely we define $x'_{\mathcal{C}}$ and $y'_{\mathcal{C}}$ as:

$$x'_{\mathcal{C}} = \frac{x_{\mathcal{C}}}{z_{\mathcal{C}}} = \frac{u - u_0}{f_u} \tag{B.16}$$

$$y'_{\mathcal{C}} = \frac{y_{\mathcal{C}}}{z_{\mathcal{C}}} = \frac{v - v_0}{f_v}. \tag{B.17}$$

Radial distortion is mainly due to flawed radial lens curvature, and causes a radial displacement of points with respect to the principal point. The distorted points coordinates x'' and y'' are:

$$x'' = x'(1 + k_1r^2 + k_2r^4 + k_3r^6) \tag{B.18}$$

$$y'' = y'(1 + k_1r^2 + k_2r^4 + k_3r^6), \tag{B.19}$$

where and $r = x'^2 + y'^2$ is the squared distance from the principal point. The values k_1, k_2, and k_3 are called *radial distortion parameters*.

Tangential distortion depends instead on errors in lens placement. In particular, it arises when the lens is not parallel to the image plane. The mathematical model is represented by the following equations:

$$x'' = x' + (2p_1x'y' + p_2(r^2 + 2x'^2)) \tag{B.20}$$

$$y'' = y' + (p_1(r^2 + 2y'^2) + 2p_2x'y'), \tag{B.21}$$

where p_1 and p_2 are the *tangential distortion parameters*. The two models are usually combined together in a single expression:

$$x'' = x'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + (2p_1 x' y' + p_2(r^2 + 2x'^2)) \quad \text{(B.22)}$$

$$y'' = y'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + (p_1(r^2 + 2y'^2) + 2p_2 x' y'). \quad \text{(B.23)}$$

Finally, the distorted point coordinates are retrieved as:

$$u_d = \frac{f_u}{x}'' + u_0 \quad \text{(B.24)}$$

$$v_d = \frac{f_v}{y}'' + v_0. \quad \text{(B.25)}$$

If all the distortion parameters $D = [k_1, k_2, p_1, p_2, k_3]$ are known, the image deformation can be corrected by computing for each pixel its ideal position on the image plane. This software correction is called *undistortion*, and it makes possible to apply the pinhole model to real cameras with negligible error. Distortion parameters can be considered intrinsic because they describe the internal configuration of the camera. Intrinsic parameters can be found through the process called *camera calibration* [301].

APPENDIX C

Stereo camera model

We consider a stereo camera as two rigidly attached cameras whose field of views have some overlapping region. This setup allows the evaluation of depths by exploting a second viewpoint, in a procedure called *triangulation*. In the simplest formulation (*standard model*), the two cameras are modeled as two identical pin-hole cameras with parallel optical axes and co-planar image planes [299] (see Figure C.1).

Conventionally, the two optical centers C_L and C_R are aligned along the local c_1 axes, which are also made co-linear. Furthermore, the local axes $\{c_1, c_2, c_3\}$ of the left pin-hole camera are used as local axes of the stereo camera. The two optical centers are found at a distance b, called *stereo baseline*. The plane joining them with the object point $\pi_{\mathcal{C}}$ is called *epipolar plane*, which in turn intersects the image planes on two co-linear straight lines, called *epipolar lines*.

Again, following the formulation of [299], we derive the stereo cam-

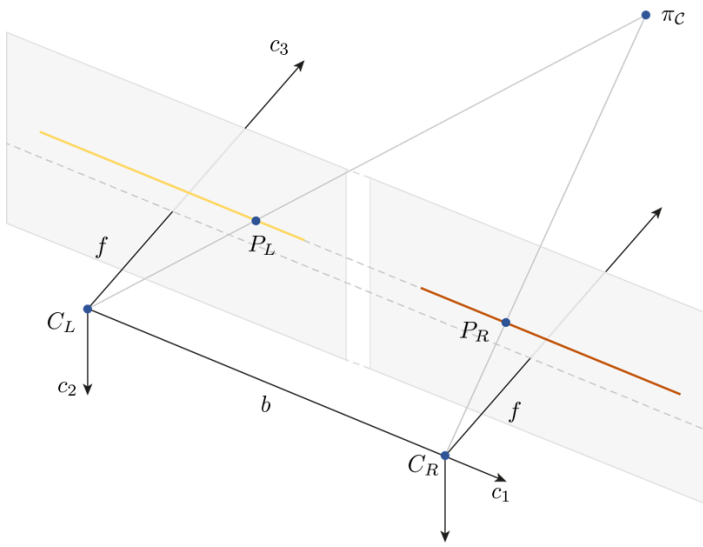


Figure C.1: *Stereo camera standard model (reworked from [300])*

era model starting from the left and right cameras' pinhole models (see Appendix B). Given the object point π_C in the stereo local coordinates, the image pixels in the left (subscript L) and right (subscript R) images are obtained respectively as:

$$\begin{bmatrix} u_L \\ v_L \\ 1 \end{bmatrix} = \bar{P}_L^{pix} = K \begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix}, \tag{C.1}$$

and

$$\begin{bmatrix} u_R \\ v_R \\ 1 \end{bmatrix} = \bar{P}_R^{pix} = K \begin{bmatrix} x_C - b \\ y_C \\ z_C \end{bmatrix}. \tag{C.2}$$

While the vertical pixel coordinates in the standard model coincide:

$$v_L = v_R, \tag{C.3}$$

the horizontal measurements allow us to observe the depth z_C through the following relation:

$$u_L - u_R = \frac{f_u}{z_{\mathcal{C}}}(x_{\mathcal{C}} - (x_{\mathcal{C}} - b)) = \frac{f_u}{z_{\mathcal{C}}} b. \tag{C.4}$$

The stereo observation model is then obtained considering $u = u_L$, $v = v_L$ and the *disparity* $d = u_L - u_R$:

$$s = \begin{bmatrix} u \\ v \\ d \end{bmatrix} = \begin{bmatrix} u_0 + f_u x_{\mathcal{C}}/z_{\mathcal{C}} \\ v_0 + f_v y_{\mathcal{C}}/z_{\mathcal{C}} \\ f_u b/z_{\mathcal{C}} \end{bmatrix}. \tag{C.5}$$

In the case of disparities $d > 0$, the model can be inverted to obtain the object point from stereo measurements s:

$$\pi_{\mathcal{C}} = \begin{bmatrix} x_{\mathcal{C}} \\ y_{\mathcal{C}} \\ z_{\mathcal{C}} \end{bmatrix} = f_u \frac{b}{d} \begin{bmatrix} (u - u_0)/f_u \\ (v - v_0)/f_v \\ 1 \end{bmatrix}, \tag{C.6}$$

which, knowing the pose of the camera, can be expressed in the inertial frame as:

$$\pi_{\mathcal{I}} = t_{\mathcal{IC}} + f_u \frac{b}{d} R_{\mathcal{IC}} \begin{bmatrix} (u - u_0)/f_u \\ (v - v_0)/f_v \\ 1 \end{bmatrix}, \tag{C.7}$$

Note that the degenerate case with $d \to 0$ corresponds to points that are very far away, in which case the two images coincide, leading the stereo camera to behave like a unique monocular camera. This happens when the baseline is too small compared to the distance to observe [299].

The only remaining aspect is related to problem of finding the image pixels in the L and R images that corresponds to the same object point. This problem is also known as *stereo correspondence*. Referring to Figure C.1, the point, associated to the pixel u_L, lies somewhere on the line defined by C_L and u_L. This line belongs to the epipolar plane and thus it projects on the epipolar line. As a consequence, the correspondent pixel u_R must be found along the epipolar line of the R image. As already mentioned, in the standard model of the stereo camera, the L and R epipolar lines coincide and, thus, the pixel u_R is defined in the right image by the ordinate of equation (C.3).

Now it is interesting to inspect what is the effect of noise on the perceptions of P_L and P_R. Suppose to have radial uncertainty over the image points. The back-projection of the uncertainty circle around a point creates a cone, with vertex in the camera center and the circle as directrix. The uncertainty of the reconstructed point lies in the intersection of the two back-projection cones [300]. Note also, that in operating conditions the point to camera distance is much greater than the baseline. Consequently, the depth estimation uncertainty is greater with respect to the other two coordinates. The uncertainty will also increase as the point depth increases. The phenomenon is graphically represented in Figure C.2.

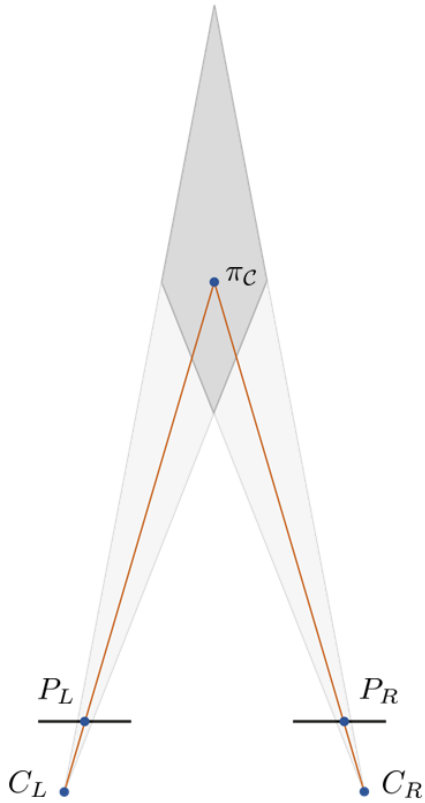


Figure C.2: *Uncertainty propagation in triangulation (reworked from [300])*

APPENDIX $\mathcal{D}$

Differential flatness proof

Consider the quadrotor model of equations (1.46)-(1.49).

In the following, we will show that the state $[r, v, R_{\mathcal{IB}}, \omega_{\mathcal{B}}]$ and the inputs $[c, \tau]$ can be written as algebraic functions of four selected outputs, which in turn are the three components of the UAV position $r = [r_x, r_y, r_z]$ and its heading ψ, and a finite number of their derivatives.

First of all, the position, velocity and acceleration of the center of mass are simply r, $\dot{r}$ and $\ddot{r}$, respectively. To see that $R_{\mathcal{IB}}$ is a function of the flat outputs and their derivatives consider equation (1.47). We have:

$$b_3 = \frac{t}{||t||}, \tag{D.1}$$

where

$$t = \begin{bmatrix} \ddot{r}_x, & \ddot{r}_y, & \ddot{r}_z + g \end{bmatrix}^\top, \tag{D.2}$$

defining the z axis of the quadrotor body frame. Given the yaw angle, we

can write the unit vectors of an intermediate reference frame $\mathcal{H}$:

$$h_1 = [\cos\psi, \sin\psi, 0]^\top \tag{D.3}$$
$$h_2 = [-\sin\psi, \cos\psi, 0]^\top \tag{D.4}$$
$$h_3 = [0, 0, 1]^\top, \tag{D.5}$$

which form the rotation matrix:

$$R_{\mathcal{IH}} = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix}. \tag{D.6}$$

Provided that $h_1 \times b_3 \neq 0$, we can retrieve b_1 and b_2 as:

$$b_2 = \frac{b_3 \times h_1}{||b_3 \times h_1||} \tag{D.7}$$
$$b_1 = b_2 \times b_3. \tag{D.8}$$

With these expressions, we can uniquely determine:

$$R_{\mathcal{IB}} = \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}. \tag{D.9}$$

In order to show that the body rates $\omega_{\mathcal{B}}$ are functions of the flat outputs and their derivatives, we take the derivative of (1.47):

$$\dot{a} = \dot{c}b_3 + c\dot{b}_3, \tag{D.10}$$

where a is the acceleration: $\ddot{r} = a$. Its derivative $\dot{a}$ is also called *jerk*. Making explicit the reference frames in which the unit vectors are written, and considering that we are taking the derivative of a vector represented in world coordinates $b_3 = b_3^{\mathcal{I}}$, we get:

$$\dot{b}_3 = \dot{b}_3^{\mathcal{I}} = \frac{d}{dt}(R_{\mathcal{IB}} b_3^{\mathcal{B}}), \tag{D.11}$$
$$= \dot{R}_{\mathcal{IB}} b_3^{\mathcal{B}} \tag{D.12}$$
$$= R_{\mathcal{IB}} [\omega_{\mathcal{B}}]_\times b_3^{\mathcal{B}}., \tag{D.13}$$

where we have substituted the expression for the attitude kinematics (1.48). Putting together the expressions (D.10) and (D.13), we get:

$$\dot{a} = \dot{c} b_3 + c R_{\mathcal{IB}} [\omega_{\mathcal{B}}]_\times b_3^{\mathcal{B}}. \tag{D.14}$$

Left multiply the previous expression by $b_1^\top$:

$$b_1^\top \dot{a} = b_1^\top \dot{c} b_3 + c b_1^\top R_{\mathcal{IB}} [\omega_{\mathcal{B}}]_\times b_3^{\mathcal{B}}. \tag{D.15}$$

and considering the expression of the rotation matrix (D.9) and the relations among unit vectors: $b_1^\top b_1 = 1$, $b_1^\top b_2 = 0$, $b_1^\top b_3 = 0$; we get:

$$b_1^\top \dot{a} = c \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} [\omega_{\mathcal{B}}]_\times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \tag{D.16}$$

which leads to:

$$b_1^\top \dot{a} = c \omega_y. \tag{D.17}$$

Similarly, we left multiply (1.47) by $b_2^\top$ and through similar computations, we get:

$$b_2^\top \dot{a} = -c \omega_x. \tag{D.18}$$

Finally, we should get a relation for retrieving ω_z. Considering equation (1.48) and left multiplying it by $b_2^\top$, we get:

$$b_2^\top \dot{R}_{\mathcal{IB}} = b_2^\top R_{\mathcal{IB}} [\omega_{\mathcal{B}}]_\times. \tag{D.19}$$

For what concerns the right-hand term, we get:

$$b_2^\top R_{\mathcal{IB}} [\omega_{\mathcal{B}}]_\times = \begin{bmatrix} \omega_z & 0 & -\omega_x \end{bmatrix}, \tag{D.20}$$

Instead, we can compare the left-hand term with the derivative of b_1, which can be computed similarly to (D.13) as:

$$\dot{b}_1 = R_{\mathcal{IB}} [\omega_{\mathcal{B}}]_\times b_1^{\mathcal{B}}. \tag{D.21}$$

Thus right multiplying (D.19) by $b_1^{\mathcal{B}}$, we obtain:

$$b_2^\top R_{\mathcal{IB}} [\omega_{\mathcal{B}}]_\times b_1^{\mathcal{B}} = \begin{bmatrix} \omega_z & 0 & -\omega_x \end{bmatrix} b_1^{\mathcal{B}}, \tag{D.22}$$

and using (D.21), it simplifies into

$$b_2^\top \dot{b}_1 = \omega_z. \tag{D.23}$$

Similarly to (D.7), we can compute b_1 as:

$$b_1 = \frac{h_2 \times b_3}{||h_2 \times b_3||}, \tag{D.24}$$

and calling $\hat{b}_1 = h_2 \times b_3$, this relation becomes:

$$b_1 = \frac{\hat{b}_1}{||\hat{b}_1||}. \tag{D.25}$$

Taking its derivative as the one of a unit vector:

$$\dot{b}_1 = \frac{\dot{\hat{b}}_1}{||\hat{b}_1||} - \hat{b}_1 \frac{\hat{b}_1^\top \dot{\hat{b}}_1}{||\hat{b}_1||^3} \tag{D.26}$$

Since b_1 and $\hat{b}_1$ are both perpendicular to b_2, we can write (D.23) as:

$$\omega_z = b_2^\top \frac{\dot{\hat{b}}_1}{||\hat{b}_1||}. \tag{D.27}$$

We now compute the derivative $\dot{\hat{b}}_1$:

$$\dot{\hat{b}}_1 = \dot{h}_2 \times b_3 + h_2 \times \dot{b}_3. \tag{D.28}$$

The two needed derivatives $\dot{h}_2$ and $\dot{b}_3$ are computed as:

$$\dot{b}_3 = R_{\mathcal{IB}}[\omega_{\mathcal{B}}]_\times b_3^{\mathcal{B}} = R_{\mathcal{IB}} \begin{bmatrix} \omega_y \\ -\omega_x \\ 0 \end{bmatrix} = \omega_y b_1 - \omega_x b_2, \tag{D.29}$$

where we have carried out the computations of equation (D.13), and:

$$\dot{h}_2 = R_{\mathcal{IH}}[\omega_{\mathcal{H}}]_\times h_2^{\mathcal{H}} = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} \dot{\psi} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = -\dot{\psi} h_1. \tag{D.30}$$

Grouping the results together:

$$\dot{\hat{b}}_1 = -\dot{\psi} h_1 \times b_3 + \omega_y h_2 \times b_1 - \omega_x h_2 \times b_2, \tag{D.31}$$

and combining it with (D.27):

$$\omega_z = b_2^\top \frac{1}{||\hat{b}_1||}(-\dot{\psi} h_1 \times b_3 + \omega_y h_2 \times b_1 - \omega_x h_2 \times b_2) \tag{D.32}$$

$$= \frac{1}{||\hat{b}_1||}(\dot{\psi} h_1^\top (b_2 \times b_3) - \omega_y h_2^\top (b_2 \times b_1) + \omega_x h_2^\top (b_2 \times b_2)) \tag{D.33}$$

$$= \frac{1}{||\hat{b}_1||}(\dot{\psi} h_1^\top b_1 + \omega_y h_2^\top b_3), \tag{D.34}$$

we have the ω_z relationship we were looking for. Note that in the previous expression the vector triple product of three arbitrary vectors a, b, and c, has been employed.

$$a^\top (b \times c) = -b^\top (a \times c). \tag{D.35}$$

Clearly, we need to show that also the input c, which appears in the equations for body rates (D.17)-(D.18), can be written as an algebraic relationship of the flat outputs and their derivatives. This proof is trivial since the net thrust is simply $c = ||t||$. Turning to the three-dimensional torque input τ, we notice that we need to show that also the angular acceleration can be written as functions of flat outputs and derivatives. We proceed by taking the derivative of (1.49):

$$\ddot{a} = \ddot{c} b_3 + \dot{c}\dot{b}_3 + \dot{c} R_{\mathcal{IB}} [\omega]_\times b_3^{\mathcal{B}} + c \dot{R}_{\mathcal{IB}} [\omega]_\times b_3^{\mathcal{B}} + c R_{\mathcal{IB}} [\dot{\omega}]_\times b_3^{\mathcal{B}} \tag{D.36}$$

$$= \ddot{c} b_3 + 2\dot{c} R_{\mathcal{IB}} [\omega]_\times b_3^{\mathcal{B}} + c R_{\mathcal{IB}} [\omega]_\times^2 b_3^{\mathcal{B}} + c R_{\mathcal{IB}} [\dot{\omega}]_\times b_3^{\mathcal{B}} \tag{D.37}$$

where we have used the expression for $\dot{b}_3$ (D.13). Note also that the second derivative of the acceleration $\ddot{a}$ is called *snap*.

With procedure similar to the one used for the angular rates, we left-multiply the expression by $b_1^\top$ and with similar arguments we obtain:

$$b_1^\top \ddot{a} = 2\omega_y \dot{c} + \omega_x \omega_z c + \dot{\omega}_y c. \tag{D.38}$$

On the other hand, if we left multiply by $b_2^\top$ we get:

$$b_2^\top \ddot{a} = -2\omega_x \dot{c} + \omega_y \omega_z c - \dot{\omega}_x c. \tag{D.39}$$

For retrieving the last equation (the one for $\dot{\omega}_z$), we compute the derivative of the following equation:

$$\omega_z ||\hat{b}_1|| = (\dot{\psi} h_1^\top b_1 + \omega_y h_2^\top b_3), \tag{D.40}$$

and, by substituting the expression for (D.31), we obtain:

$$\begin{aligned} &- \dot{\omega}_y h_2^\top b_3 - \omega_y \dot{h}_2^\top b_3 - \omega_y h_2^\top \dot{b}_3 + \dot{\omega}_z ||h_2 \times b_3|| + \omega_z \frac{d}{dt}(||\hat{b}_1||) = \\ &= \ddot{\psi} h_1^\top b_1 + \dot{\psi} \dot{h}_1^\top b_1 + \dot{\psi} h_1^\top \dot{b}_1, \end{aligned} \tag{D.41}$$

We now need to derive the expressions for the terms appearing in it. Using equation (D.30), we obtain:

$$- \omega_y \dot{h}_2^\top b_3 = \omega_y \dot{\psi} h_1^\top b_3, \tag{D.42}$$

The second term, using expression (D.13), is equal to:

$$- \omega_y h_2^\top \dot{b}_3 = -\omega_y^2 h_2^\top b_1 + \omega_x \omega_y h_2^\top b_2, \tag{D.43}$$

which, considering that $h_2^\top b_1 = 0$, simplifies into:

$$- \omega_y h_2^\top \dot{b}_3 = \omega_x \omega_y h_2^\top b_2. \tag{D.44}$$

For the third term, we start by exploiting the expression for the derivative of a vector:

$$\frac{d}{dt}(||\hat{b}_1||) = \frac{\hat{b}_1^\top \dot{\hat{b}}_1}{||\hat{b}_1||}. \tag{D.45}$$

Now, we can simplify the term using expressions (D.31) and (D.25).

$$\omega_z \frac{\hat{b}_1^\top \dot{\hat{b}}_1}{||\hat{b}_1||} = \omega_z b_1^\top \dot{\hat{b}}_1 = \omega_z b_1^\top (-\dot{\psi} h_1 \times b_3 + \omega_y h_2 \times b_1 - \omega_x h_2 \times b_2). \quad \text{(D.46)}$$

Using the property (D.35), we find:

$$\omega_z b_1^\top (-\dot{\psi} h_1 \times b_3 + \omega_y h_2 \times b_1 - \omega_x h_2 \times b_2) = \omega_z (-\dot{\psi} h_1^\top b_2 + \omega_x h_2^\top b_3). \quad \text{(D.47)}$$

For the fourth term, in a way similar to (D.30), we get $\dot{h}_1$ as:

$$\dot{h}_1 = R_{\mathcal{IH}} [\omega_{\mathcal{H}}]_\times h_2^{\mathcal{H}} = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} \dot{\psi} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \dot{\psi} h_2. \quad \text{(D.48)}$$

and, finally, we can carry on the computations of the relation (D.21):

$$\dot{b}_1 = R_{\mathcal{IB}} \begin{bmatrix} 0 \\ \omega_z \\ -\omega_y \end{bmatrix} = \omega_z b_2 - \omega_y b_3. \quad \text{(D.49)}$$

Putting together (D.41) with (D.42), (D.44), (D.47), (D.48) and (D.49), we get the final constraint for finding the angular accelerations:

$$\begin{aligned} &- \dot{\omega}_y h_2^\top b_3 + \dot{\omega}_z ||h_2 \times b_3|| = \ddot{\psi} h_1^\top b_1 - \omega_y \dot{\psi} h_1^\top b_3 - \omega_x \omega_y h_2^\top b_2 + \\ &+ \omega_z \dot{\psi} h_1^\top b_2 - \omega_x \omega_z h_2^\top b_3 + \dot{\psi}^2 h_2^\top b_1 + \omega_z \dot{\psi} h_1^\top b_2 - \omega_y \dot{\psi} h_1^\top b_3, \end{aligned} \quad \text{(D.50)}$$

which in turn, considering that $h_2^\top b_1 = 0$, and regrouping the terms, leads to:

$$\begin{aligned} &- \dot{\omega}_y h_2^\top b_3 + \dot{\omega}_z ||h_2 \times b_3|| = \ddot{\psi} h_1^\top b_1 - 2\omega_y \dot{\psi} h_1^\top b_3 + \\ &- \omega_x \omega_y h_2^\top b_2 + 2\omega_z \dot{\psi} h_1^\top b_2 - \omega_x \omega_z h_2^\top b_3, \end{aligned} \quad \text{(D.51)}$$

Since the term $\dot{c}$ appears in the other two equations for finding angular accelerations (D.38)-(D.39), we have to show that it can be written as a

function of the flat outputs and derivatives. This can be done by projecting equation (D.10) on b_3:

$$b_3^\top \dot{a} = \dot{c} b_3^\top b_3 + c b_3^\top R_{\mathcal{IB}} [\omega_{\mathcal{B}}]_\times b_3^{\mathcal{B}} \tag{D.52}$$

which simplifies into:

$$\dot{c} = b_3^\top \dot{a}. \tag{D.53}$$

Once the angular accelerations are known, the torques input τ can be found from equation (1.49).