University of Padova

Department of Information Engineering

Master Thesis in Constrol System Engineering

VISUAL ENVIRONMENT ASSESSMENT FOR SAFE AUTONOMOUS QUADROTOR LANDING

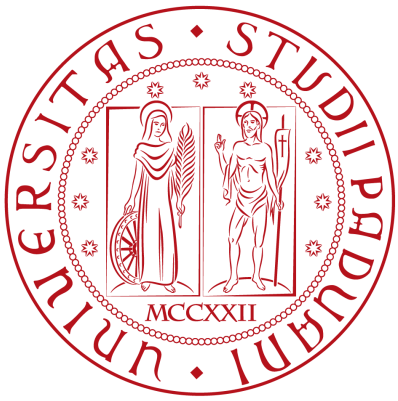
Supervisor
Prof. Carli Ruggero
University of Padova

Co-supervisor
Prof. Giuseppe Loianno
New York University

Master Candidate
Mattia Secchiero

Student ID
2053075

Academic Year
2022-2023


UNIVERSITAS STUDII PADUANI
MCCXXII


DEPARTMENT OF
INFORMATION
ENGINEERING
UNIVERSITY OF PADOVA


NYU
TANDON SCHOOL
OF ENGINEERING


ARPL
Agile Robotics
and Perception Lab

"The ultimate value of doing the work of personal growth is not to only feel better about ourselves.
We do the work so that we can contribute to how those around us feel about themselves."

— Simon Sinek

Abstract

Autonomous identification and evaluation of safe landing zones are of paramount importance for ensuring the safety and effectiveness of aerial robots in the event of system failures, low battery, or the successful completion of specific tasks. In this thesis it is presented a novel approach, for detecting and assess potential landing sites for safe quadrotor landing. The proposed solution efficiently integrates both 2D and 3D environmental information and eliminates the need for external aids such as GPS and computationally intensive elevation maps. Semantic data derived from a Neural Network (NN), is combined with geometric data obtained from a disparity map, to extract environmental features and critical geometric attributes such as slope, flatness, and roughness. In particular, this method efficiently combines both metric and semantic information, making it also more robust, compared to other solutions that solely relies on one type of information only. Based on those attributes, several cost metrics are defined to evaluate safety, stability, and suitability of regions in the environments and identify the most suitable landing area. In this way we have a comprehensive evaluation of all the relevant aspects related to the safe site detection. This approach runs in real-time on quadrotors equipped with limited computational capabilities. Experimental results conducted in diverse environments demonstrate that the proposed method can effectively assess and identify suitable landing areas, enabling the safe and autonomous landing of a quadrotor in unknown environments.

Contents

Listing of figures

Listing of tables

Listing of acronyms

NN . . . . . Neural Network

UAV . . . . . Unmanned Aerial Vehicles

GPS . . . . . Global Positioning System

DEM . . . . . Digital Elevation Model

SfM . . . . . Structure from Motion

LiDAR . . . . . Light Detection and Ranging

VIO . . . . . Visual Inertial Odometry

IMU . . . . . Inertial Measurement Unit

SOTA . . . . . State of the Art

ROS . . . . . Robot Operating System

SLAM . . . . . Simultaneous Localization and Mapping

PCL . . . . . Point Cloud Library

SGD . . . . . Stochastic Gradient Descent

LR . . . . . Learning Rate

SWaP . . . . . Size, Weight and Power

1 Introduction

Unmanned Aerial Vehicles (UAVs) have become increasingly popular platforms to assist humans in several complex and dangerous applications such as surveillance, law enforcement, mapping, search and rescue, delivery services and precision agriculture [1, 2]. The development of novel autonomous algorithms coupled with the drop in price/performance ratio of processors and sensors supported also the execution of complex tasks such as collaborative transportation [3], autonomous flight [4], collision avoidance [5], exploration [6] as well as shipping and delivery [7] or industrial inspection [8].

One of the most important aspect, that must be considered for autonomous aerial vehicles, is related to the hazards that they might pose to people and structures, in case of system failure or other dangerous situations. In order to ensure safety and overall mission success in the aforementioned applications, it is commonplace to equip aerial robots with intelligent landing capabilities [9, 10, 11]. These mitigate the risks posed by mechanical and sensory failures, ensuring secure operations in challenging scenarios. This not only minimizes potential threats posed to individuals and structures, but also enables the successful execution of tasks that require such capabilities.

Figure 1.1: AI-generated pictures of drone applications such as package delivery, industrial inspection, environment mapping and collaborative transportation.

For example, in the case of a drone delivery system that operates in urban environments, the ability to accurately identify suitable landing zones becomes crucial for the successful delivery of packages. In agriculture, drones play a pivotal role in monitoring crops, assessing plant health, and optimizing agricultural practices. Upon completing these tasks, the drone requires a reliable and safe landing procedure. In a low battery scenario, the ability to accurately identify suitable landing zones becomes crucial to avoid catastrophic outcomes. The drone must quickly assess the available options for landing, taking into account terrain factors such as slope, obstacles, roughness and safety for both the drone and the people on the ground. Once a suitable landing zone is identified, the drone can execute an emergency landing, avoiding any potential threat. In these situations, the drone's ability to autonomously identify safe landing sites can mean the difference between a successful mission completion, or not.

However, current state-of-the-art solutions tend to be fragile and computationally intensive, often require preliminary environment knowledge and still offer limited autonomy. Similarly, the majority of commercially available landing solutions often relies on manually designed, predefined navigation policies to aid humans in guiding the robots to land near the intended or required location, therefore offering minimal or no autonomy also for the landing process.

To address those limitations, this thesis presents several significant contributions. First, it is used a novel visual environment detection and assessment approach for the safe autonomous landing of aerial robots. Compared to the state-of-the-art solutions, this method efficiently combines metric and semantic information, leveraging both RGB images and disparity maps extracted from the environment, to assess the suitability of a possible landing zone. Specifically, the 3D points of the environment, generated from the disparity map, are projected onto the segmented RGB image to efficiently process only the ones associated with safe regions, obtaining an effective and efficient solution. Second, several cost metrics are defined, based on critical geometric attributes such as slope, flatness, and roughness, extracted from the visual information. The points are processed to verify if they respect those attributes, making us able to classify them as hazardous or safe for landing. Furthermore, this method does not need to build and store expensive elevation maps. Instead, it directly generates and updates a 2D binary map of safe and unsafe landing zones, without sacrificing any relevant information compared to other existing approaches. From the mapped environment it is finally retrieved the most suitable landing zone, by minimizing a cost function that takes into account the drone's distance and the obstacle's distance from a possible landing area. Finally, this pipeline operates on-board, without relying on any off-board streaming of data, GPS or pre-obtained information.
Through several tests, in different and challenging indoor environments, this thesis demonstrates that the proposed framework successfully enables safe quadrotor landings.

1.1 Agile Robotics and Perception Lab

This thesis's project was developed at the Agile Robotics and Perception Lab (ARPL), within the New York University (NYU). The ARPL [12] is dedicated to conducting both fundamental and applied research in the field of robotics autonomy. The lab's primary objective is the development of agile autonomous machines capable of independent navigation in unstructured, dynamically changing environments, relying solely on onboard sensors, and without the need for external infrastructure like GPS or motion capture systems. These machines are designed

to operate proactively, fostering collaboration with humans and among themselves, while effectively exploring and extracting valuable insights from unknown environments.

1.2 Thesis structure

Beginning with Ch. 2, the state-of-the art (SOTA) solutions are presented and discussed, in order to compare the works that have been done in the safe site detection and autonomous landing field. Following, a brief introduction on the proposed system, where the main steps to derive a safe landing zone are introduced, together with an overview of the quadrotor dynamics, the Robot Operating System (ROS) and the camera calibration procedure. In Ch. 4 and Ch. 5 are discussed in details the data processing, map creation and autonomous landing procedures. Here we can find all the steps that are implemented, in order to map the safe and unsafe landing zones of the environment, in order to lead the drone to land in the most suitable landing area. Finally, in Ch. 6 are given some results of our framework. In particular, the system setup, the Neural Network training and evaluation and the environmental assessment results are discussed. Following, in Ch. 7 the thesis is concluded by introducing some discussion and future works, useful for improving the proposed framework.

2

Related Work

Several works address the problem of identifying a safe landing zone and implementing autonomous landing procedures. The vast majority of the existing approaches are vision-based, given the Size, Weight, and Power (SWaP) constraints of small-size aerial robots.

For example in [13] the authors compute the local slope and roughness of a Digital Elevation Model (DEM) to detect and avoid hazards such as steep slopes, rocks, cliffs, and gullies. The DEM is generated employing a Structure from Motion (SfM) algorithm, since a single monocular camera is used. SfM is used to track the features' motion between consecutive images and derive the environment depth of the selected features. First, the DEM is partitioned into square regions the size of the lander footprint. In each region both the slope and roughness thresholds are evaluated, creating an associated map of safe and unsafe landing locations. The lander will have constraints on the maximum slope and maximum roughness that can be handled by the mechanical landing system. These thresholds are set by the user. Selection of the safe site will occur by generating binary images from the slope and roughness maps; parts of the maps that are above the threshold are hazardous while parts that are below are not a hazard.

Another approach, as described in [14], defines a cost function that evaluates the physical prop-

erties of the local neighborhood within an elevation map region to identify safe landing areas. Also in this case, a single monocular camera is used to estimate the depth of the environment. REMODE [15], a probabilistic approach to monocular dense reconstruction, is used to compute the depth maps. This method combines Bayesian estimation and convex optimization for image processing, to produce a compact and efficient representation of a depth map. A robot-centric, fixed size map is used to store the mapped environment. Even if this solution could be less subject to drift, however, it confines the environmental knowledge exclusively to the region beneath the drone.

The solution proposed in [10] instead infers a safe landing zone by evaluating slope and roughness from the DEM of the environment, obtained using a SfM algorithm. Here, a robot-centric, multi-elevation fixed size map is used. Therefore, also in this case environmental knowledge is confined to only the region beneath the drone. To analyze the safeness of a possible landing area, the detector first evaluates the coarsest layer for slope and roughness, and if successful then performs a hazard analysis in the finer layers. If an area is declared as unsafe in a coarse layer it is no longer evaluated in subsequent layers, saving computation time. The landing site detector result is a binary map which annotates if a map cell is a valid landing site or not. Finally, by applying a distance transform, locations with a maximum distance to any obstacles can be selected and collected in a list of landing site candidates.

Conversely, the authors in [9] apply a slope and roughness threshold to the DEM image gradient, to only retain flat terrains. Since the elevation maps are pre-determined, a Neural Network (NN) segments the RGB images of the possible landing area as a final evaluation step. The semantic information is used to asses the validity of the area and to overcome the fact that the maps could be outdated. However, relying on pre-determined maps constrains the applicability of this method to regions where such maps are available. Additionally, it necessitates the use of other technologies, such as GPS, to verify the drone's position relative to the pre-obtained map, making it less versatile for general-purpose applications.

Other works such as [11] and [16] directly implement semantic segmentation on a DEM. Segmentation is employed to classify hazardous and safe landing locations without resorting to plane fitting techniques or gradient thresholding. However, these approaches rely on pre-collected

LiDAR data to get the elevation models, which may not be suitable for small-size UAVs and might not always have access to up-to-date maps.

Yet, it is essential to develop general-purpose solutions that do not rely on GPS [13, 9] or pre-determined maps [9, 11, 16]. The former would be unsuitable for indoor or GPS-denied environments, while the latter's reliance on pre-defined environment poses a challenge in dynamic settings, potentially leading to catastrophic outcomes. [14] and [10] further illustrate this by employing a fixed-size map, limiting environmental awareness to the area directly below the drone, thus disregarding a big portion of the overflown area. In [17], the 3D information are lacking, consequently failing to comprehensively address crucial factors like slope, flatness, and roughness.

Compared to the aforementioned existing solutions, we directly construct a variable dimension 2D binary map to guide the drone toward a safe landing location. In such a way, our approach does not need to derive any elevation map, resulting in a lighter and more efficient implementation, while still evaluating all the relevant aspects related to the safe site detection. In addition, our pipeline stands out by its independence from external aids such as GPS, off-board or pre-obtained geometric information and autonomously implements inspection and landing behaviours.

3

System Overview

Figure 3.1 gives an overview of the entire system's pipeline. The drone leverages a stereo camera pair combined with data coming from an Inertial Measurement Unit (IMU) to compute Visual-Inertial Odometry (VIO). VIO is an advanced sensor fusion technology used for real-time localization. Combining the data from a stereo pair and an IMU it estimates the position and orientation (pose) of a device or vehicle. VIO operates by simultaneously tracking visual features across stereo images and leveraging the IMU's measurements of linear accelerations and angular velocities. Inertial data, integrated over time, assists in compensating for short-term motion and accelerations, enhancing robustness. VIO algorithms fuse the visual and inertial information to continuously update pose estimates, providing accurate localization even in GPS-denied environments. The real-time estimation of the drone's position and orientation, with respect to the fixed world reference frame ($\mathcal{F}_W$), is essential for autonomous flight capabilities such as exploration and autonomous landing.

The 2D occupancy grid, on the other hand, is generated considering the segmented RGB images and the disparity maps obtained from a stereo pair. This occupancy grid directly embeds both semantic and geometric information, representing the safe and unsafe landing regions of

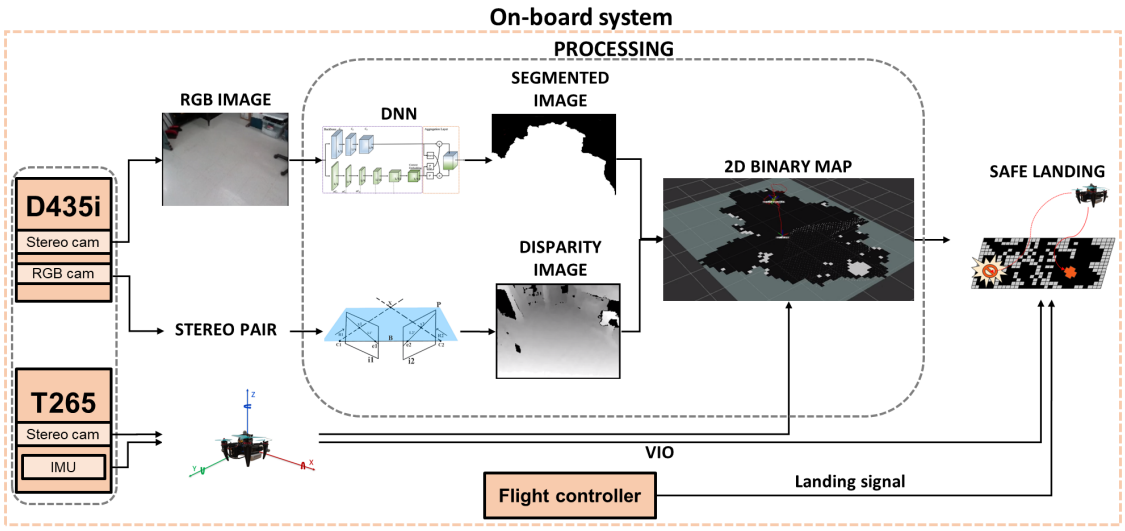


Figure 3.1: Overview of the autonomous safe site detection and landing system: it is used the ARPL's quadrotor with a NVIDIA Jetson NX for computation and a stereo camera for VIO & mapping the environment. All the algorithms run in real-time onboard.

the environment. A two-dimensional representation has been chosen, since it is an efficient on-board map representation for UAVs with SWaP constraints. At the perception level, one key distinction between this approach and other methodologies lies in the way the map of safe and unsafe landing locations is generated. In most other approaches, the workflow involves the initial construction of a DEM or an elevation map. Subsequently, a binary map of the environment is created, and safe landing areas are identified within this map. In contrast, our method simplifies this process by directly creating a 2D variable-dimension occupancy grid, without the need for elevation maps. This grid encodes a binary classification, distinguishing safe and unsafe landing locations, while still retaining all the essential 3D information. Furthermore, it also employs both metric and semantic information, unlike many other approaches that rely solely on one type of information, increasing the robustness of the overall solution. The grid is dynamically updated based solely on the safe point cloud data that meet all the safe site criterion. This representation is essential to understand the distribution of safe and unsafe landing locations over the environment.

Finally, based on this representation, in the evaluation step it is found the actual landing zone. From the 2D map and the drone's position we exploit the best landing zone by minimizing a

cost function that considers

(a) the drone's distance from a potential safe area;

(b) the distance of the closest unsafe point to a possible safe area.

This process is iterated across the entire map to find a safe zone, large enough to accommodate the drone during landing while also ensuring a safety margin. Once the inspection behaviour concludes, or the necessity of landing arises, the safe landing coordinates are retrieved and the landing behaviour is performed autonomously.

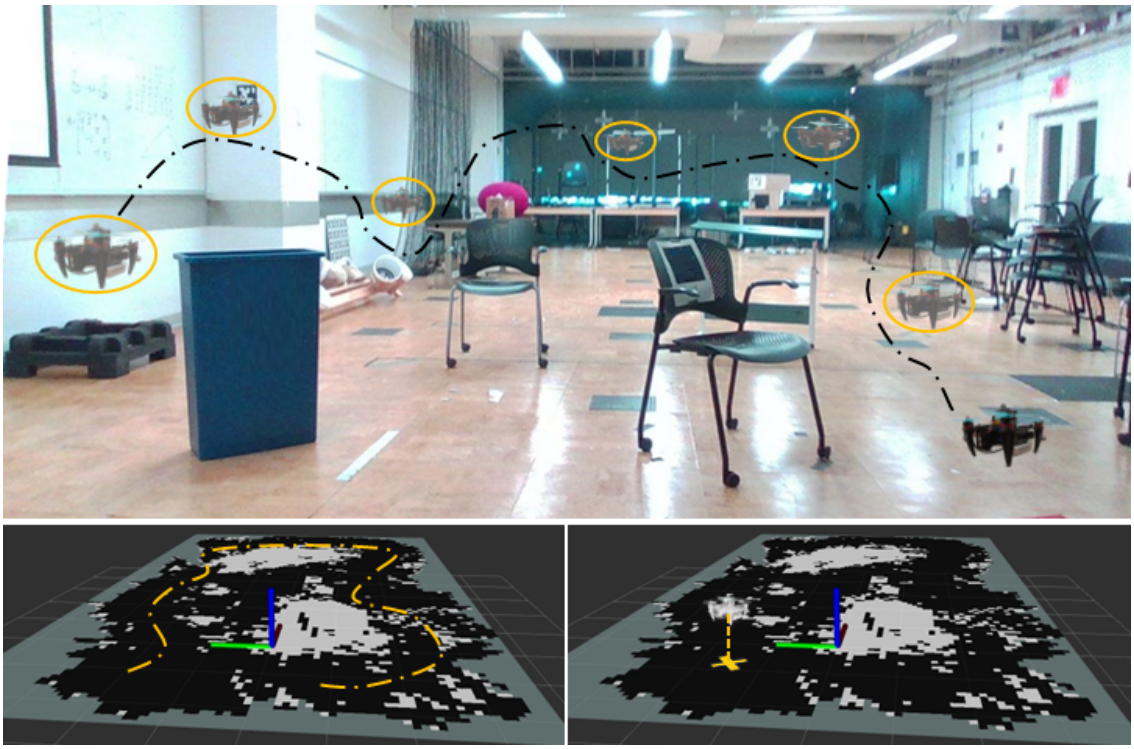


Figure 3.2: *Top*: the drone navigating and mapping the environment. *Bottom*: the associated 2D binary map of the safe and unsafe landing locations (*left*) and the chosen safe landing spot in the 2D map (*right*). The black areas are the safe landing locations, while the gray ones are unsafe.

3.1 Robot Operating System

The whole system relies on the Robot Operating System (ROS) middleware [18], a robust and widely adopted framework that plays a central role in facilitating seamless communication and integration among the various modules of the proposed architecture. ROS, specifically designed for robotic applications, offers a comprehensive suite of tools, libraries, and conventions that streamline the development and operation of complex robotic systems.
ROS operates on the Ubuntu 20.04 operating system and is installed on the Nvidia Jetson platform, enhancing its compatibility and performance in resource-constrained environments.
In the context of our real-time and resource-constrained applications, ROS proves to be an invaluable asset. It follows a modular and distributed architecture that allows developers to break down complex robotic systems into smaller, manageable components called nodes. These nodes can communicate with each other through a publish-subscribe mechanism, known as the ROS topic system. This decentralized architecture enhances fault tolerance and scalability while promoting code reuse and maintainability, enhancing the good software engineering principles [19].
Additionally, ROS provides a range of built-in libraries and tools for tasks such as perception, motion planning, and control. It also offers a thriving ecosystem of open-source packages and community support, which accelerates development and reduces the time-to-market for robotic projects.

3.2 Quadrotors Dynamics and Control

Quadrotor control is needed to implement trajectory tracking capabilities, used during the room inspection and autonomous landing. A brief introduction and explanation on the quadrotor dynamics is here given, to better understand how an attitude and position control can then be implemented, to track the desired pose given to the drone.

3.2.1 The standard N-rotor model

The standard n-rotor is made of a rigid body with n propellers spinning about their own axis. In particular, the number n of propellers and the axes mutual orientations determine if the n-rotor is an under-actuated or fully-actuated system. To describe the dynamics of the n-rotor, we introduce the body frame $\mathcal{F}_\mathcal{B}$, whose origin $\mathcal{O}_\mathcal{B}$ coincides with the Center of Mass (CoM) of the platform, and the inertial world frame $\mathcal{F}_\mathcal{W}$: the position of $\mathcal{O}_\mathcal{B}$ in $\mathcal{F}_\mathcal{W}$ and the orientation of $\mathcal{F}_\mathcal{B}$ with respect to $\mathcal{F}_\mathcal{W}$ are respectively denoted by the vector $\mathbf{p} \in \mathbb{R}^3$ and by the rotation matrix $\mathbb{R} \in \mathbb{SO}(3)$, hence the pair $\mathcal{X} = (\mathbf{p};\mathbf{R}) \in \mathbb{R}^3 \times \mathbb{SO}(3)$ describes the full-pose of the vehicle in $\mathcal{F}_\mathcal{W}$. The twist of the platform is indicated by the pair $(\mathbf{v}, \boldsymbol{\omega})$ where $\mathbf{v} = \dot{\mathbf{p}} \in \mathbb{R}^3$ denotes the linear velocity of $\mathcal{O}_\mathcal{B}$ in $\mathcal{F}_\mathcal{W}$, and $\boldsymbol{\omega} \in \mathbb{R}^3$ is the angular velocity of $\mathcal{F}_\mathcal{B}$ with respect to $\mathcal{F}_\mathcal{W}$, expressed in $\mathcal{F}_\mathcal{B}$. Thus, the kinematics is governed by the relation

$$\begin{aligned} \dot{\mathbf{p}} &= \mathbf{v} \\ \dot{\mathbf{R}} &= \mathbf{R}[\boldsymbol{\omega}]_x \end{aligned} \tag{3.1}$$

where for the last expression we recall the expression of the tangent plane to $\mathbb{SO}(3)$ at $\mathbb{R}$ (where $[\boldsymbol{\omega}]_x$ is also represented with the hat map and $\hat{\boldsymbol{\omega}} : \mathbb{R}^3 \to \mathfrak{so}(3)$). The motion equations are derived using the standard Newton-Euler approach for the dynamics and considering the forces and torques that are generated by each propeller. The i-th propeller, with $i = 1 \dots n$, rotates around its own spinning axis passing through the center $\mathcal{O}_{\mathcal{P}_i}$ with a controllable spinning rate $\boldsymbol{\omega}_\mathbf{i} \in \mathbb{R}$. There exist two kinds of propellers, spinning clockwise (CW) or counterclockwise (CCW) with respect to their axis ($\hat{\mathbf{u}}_{\mathbf{z}_i}$). If the propeller is CW then its angular velocity in $\mathcal{F}_\mathcal{B}$ is $-\boldsymbol{\omega}_\mathbf{i}\hat{\mathbf{u}}_{\mathbf{z}_i}$, otherwise is $+\boldsymbol{\omega}_\mathbf{i}\hat{\mathbf{u}}_{\mathbf{z}_i}$. We define $u_i = \omega_i\|\omega\| \in \mathbb{R}$ as the control input. The propeller applies at $\mathcal{O}_{\mathcal{P}_i}$

- 1. a thrust force $\mathbf{f}_\mathbf{i} \in \mathbb{R}^3$ that, expressed in $\mathcal{F}_\mathcal{B}$, is equal to $\mathbf{f}_\mathbf{i} = c_{fi}ui\hat{\mathbf{u}}_{\mathbf{z}_i}$ where $c_{fi} \in \mathbb{R}^+$ is a constant parameter.

- 2. a drag moment $\boldsymbol{\tau}^\mathbf{d}_\mathbf{i} \in \mathbb{R}^3$ whose direction is opposite to the angular velocity of the propeller and whose expression in $\mathcal{F}_\mathcal{B}$ is $\boldsymbol{\tau}^\mathbf{d}_\mathbf{i} = c_{\tau i}ui\hat{\mathbf{u}}_{\mathbf{z}_i}$ where $c_{\tau i} \in \mathbb{R}$ is a constant parameter (positive if the propeller is CW and negative otherwise).

Considering now the entire UAV (as shown in Fig. 3.3), $\boldsymbol{\tau}_{\mathbf{i}}^{\mathbf{t}} = \mathbf{p_i} \times \mathbf{f_i} \in \mathbb{R}^3$ represents the thrust moment associated to the i-th propeller, and the total control force $\mathbf{f_c} \in \mathbb{R}^3$ and the total control moment $\boldsymbol{\tau}_\mathbf{c} \in \mathbb{R}^3$ applied at $\mathcal{O}_\mathcal{B}$ and expressed in $\mathcal{F}_\mathcal{B}$ are

$$\begin{aligned} f_c &= \sum_{i=1}^{n} f_i = \sum_{i=1}^{n} c_{f_i} \hat{\mathbf{u}}_{\mathbf{z_i}} u_i \\ \tau_c &= \sum_{i=1}^{n} (\boldsymbol{\tau}_{\mathbf{i}}^{\mathbf{t}} + \boldsymbol{\tau}_{\mathbf{i}}^{\mathbf{d}}) = \sum_{i=1}^{n} (c_{f_i} \mathbf{p_i} \times \hat{\mathbf{u}}_{\mathbf{z_i}} + c_{\tau_i} \hat{\mathbf{u}}_{\mathbf{z_i}}) u_i \end{aligned} \tag{3.2}$$

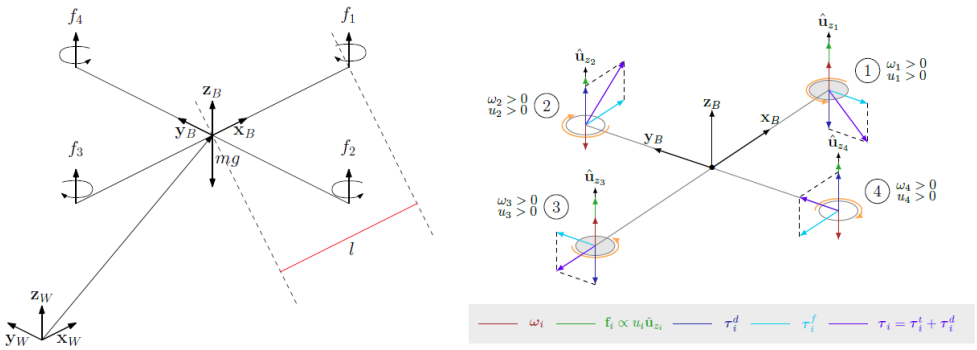


Figure 3.3: Schematic picture of a quadrotor.

A summary of forces and torques acting on a quadcopter is given as an example in the left of Fig. 3.3. Introducing the control input vector $\mathbf{u} = \begin{bmatrix} u_1 & \cdots & u_n \end{bmatrix}^T \in \mathbb{R}^n$, Eq. (3.2) can be shortened as

$$\mathbf{f_c} = \mathbf{F}\mathbf{u}; \quad \text{and} \quad \tau_c = \mathbf{M}\mathbf{u}; \tag{3.3}$$

where the control force input matrix $\mathbf{F} \in \mathbb{R}^{3\times n}$ and the control moment input matrix $\mathbf{M} \in \mathbb{R}^{3\times n}$ depend on the geometric and aerodynamic parameters introduced before. In the case of the standard coplanar quadrotor, the computation above gives:

$$\mathbf{F} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ cf & cf & cf & cf \end{bmatrix}$$

$$\mathbf{M} = \begin{bmatrix} 0 & lcf & 0 & -lcf \\ -lcf & 0 & lcf & 0 \\ -\|c_\tau\| & \|c_\tau\| & -\|c_\tau\| & \|c_\tau\| \end{bmatrix}$$

where l is the arm length.

Neglecting the second-order effects (such as the gyroscopic and inertial effects due to the rotors and the flapping), the dynamics of the n-rotor is described by the following system of Newton-Euler equations:

$$m\ddot{\mathbf{p}} = -m\mathbf{g}\mathbf{e}_3 + \mathbf{R}\mathbf{f}_c = -m\mathbf{g}\mathbf{e}_3 + \mathbf{R}\mathbf{F}\mathbf{u}; \tag{3.4}$$

$$\mathbf{J}\dot{\boldsymbol{\omega}} = -\boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} + \boldsymbol{\tau}_c = -\boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} + \mathbf{M}\mathbf{u}; \tag{3.5}$$

where $\mathbf{g} > 0$, $\mathbf{m} > 0$, and $\mathbf{J} \in \mathbb{R}^{3\times3}$ are the gravitational acceleration, the total mass of the platform, and its positive definite inertia matrix, respectively. Here, $\mathbf{e}_i$ is the i-th canonical basis vector of $\mathbb{R}^3$ with $i = 1, 2, 3$.

Finally, it can also be provided a unified overview of the kinematics and dynamics equations, with respect to an $X = (\mathbf{p}, \mathbf{R}) \in \mathbb{R}^3 \times \mathbb{SO}(3)$ representation:

$$\dot{\mathbf{p}} = \mathbf{v} \tag{3.6}$$

$$\dot{\mathbf{R}} = \mathbf{R}[\boldsymbol{\omega}]_x \tag{3.7}$$

$$m\ddot{\mathbf{p}} = -m\mathbf{g}\mathbf{e}_3 + \mathbf{R}\mathbf{F}\mathbf{u} \tag{3.8}$$

$$\mathbf{J}\dot{\boldsymbol{\omega}} = -\boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} + \mathbf{M}\mathbf{u} \tag{3.9}$$

3.2.2 Platform control

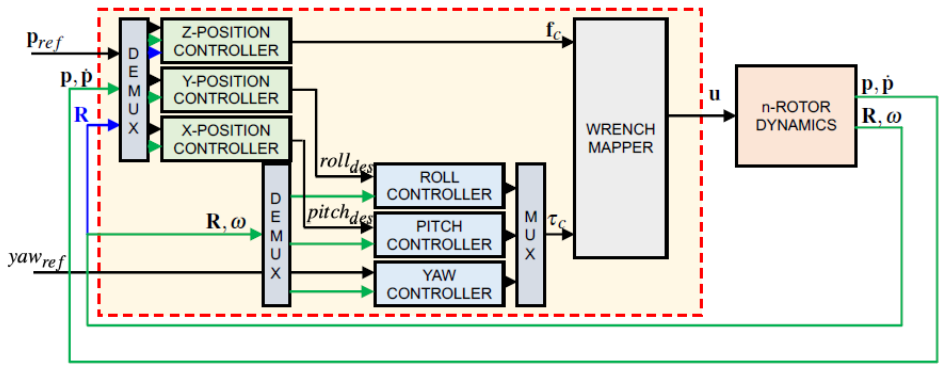


Figure 3.4: Cascade control scheme for a quadrotor platform.

An example of full control scheme is given in Fig. 3.4 where the idea of cascaded control is shown: the controller is partitioned into two main blocks, one related to position control and the other related to attitude control. The former takes a position reference p_{ref} as the input together with the feedback position (and velocity) signal; the latter considers the information related to the body frame orientation (reference rotations, feedback signals). More importantly, the position controller may produce a desired rotation to reach the position reference, which has to be aligned with the orientation reference inside the attitude controller. These two blocks produce the wrench pair (force and moment), which is translated through the wrench mapper into the actual n-rotor commands (rotational speed of the propellers). A note about an additional feedback acting on the position controller is due: the vertical force to compensate the gravity may change according to the platform orientation, as, for example, in the case of the quadrotor. For this reason, a solution could be to take into account (possibly non-zero) the roll and pitch angles, in order to get exact vertical compensation.

Considering the system dynamic equations, starting from a diagonal inertia matrix and the

control inputs given in the body frame, we can now design the control action. In this case, a non-linear $\mathbb{SE}(3)$ controller [20] has been chosen. A brief explanation of the controller is here given for reference.

Geometric Tracking Control on $\mathbb{SE}(3)$

[20] develops a controller to follow a prescribed trajectory $x_d(t)$ of the location of the center of mass and the direction of the body-fixed axis representing the yawing (or heading) angle of a quadrotor UAV.This controller is developed directly on the nonlinear configuration Lie group and thereby avoid any singularities and complexities that arise in local coordinates. As a result, it is able to achieve almost global exponential attractiveness to the zero equilibrium of tracking errors.
The overall controller structure is similar to what illustrated in Fig. 3.4.

Tracking Errors: the tracking errors are defined for x, v, R, and ω as

$$\mathbf{e}_p = \mathbf{p} - \mathbf{p}_d, \tag{3.10}$$

$$\mathbf{e}_v = \mathbf{v} - \mathbf{v}_d. \tag{3.11}$$

for the position and the velocity. The attitude and angular velocity tracking error instead are chosen to be modeled by the error function on $\mathbb{SO}(3)$

$$\Phi(\mathbf{R}, \mathbf{R}_d) = \frac{1}{2}\text{tr}[I - \mathbf{R}^T\mathbf{R}_d], \tag{3.12}$$

where $\mathbf{R}_d$ is the desired attitude. This is locally positive-definite about $\mathbf{R}^T\mathbf{R}_d = I$ within the region where the rotation angle between $\mathbf{R}$ and $\mathbf{R}_d$ is less than $180°$. This set can be represented by the sublevel set of Φ where $\Phi < 2$, namely $L_2 = \{\mathbf{R}_d, R\mathbf{R} \in \mathbb{SO}(3) | \Phi(\mathbf{R}, \mathbf{R}_d) < 2\}$, which almost covers $\mathbb{SO}(3)$.
From this representation we can then derive the attitude tracking error, considering the deriva-

tive of the error function, as

$$\mathbf{e}_R = \frac{1}{2}(\mathbf{R}^T\mathbf{R}_d - \mathbf{R}^T\mathbf{R}_d)^\vee. \tag{3.13}$$

where the vee map $\vee : \mathfrak{so}(3) \to \mathbb{R}^3$ is the inverse of the hat map. Also the tracking error for the angular velocity follows:

$$\mathbf{e}_\omega = \boldsymbol{\omega} - \mathbf{R}^T\mathbf{R}_d\boldsymbol{\omega}_d, \tag{3.14}$$

where $\boldsymbol{\omega}_d$ is the desired angular velocity. This instead is attained comparing the tangent vectors $\dot{\mathbf{R}}$ and $\dot{\mathbf{R}}_d$, however, since they lie in different tangent spaces, $\dot{\mathbf{R}}_d$ is transformed to a vector in $\mathbf{T}_\mathbf{R}\mathbb{SO}(3)$.

Tracking Controller: for given smooth tracking commands and some positive gains $\mathbf{k}_x$, $\mathbf{k}_v$, $\mathbf{k}_R$, $\mathbf{k}_\omega$, the control inputs $\mathbf{f_c}$ and $\mathbf{M}$ are chosen as follows:

$$f_3 = -\left(-\mathbf{k}_p\mathbf{e}_p - \mathbf{k}_v\mathbf{e}_v - mg\mathbf{e}_3 + m\ddot{\mathbf{p}}_d\right) \cdot \mathbf{R}\mathbf{e}_3, \tag{3.15}$$

$$\mathbf{M} = -\mathbf{k}_R\mathbf{e}_R - \mathbf{k}_\omega\mathbf{e}_\omega + \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega} - \mathbf{J}\left(\hat{\boldsymbol{\omega}}\mathbf{R}^T\mathbf{R}_d\boldsymbol{\omega}_d - \mathbf{R}^T\mathbf{R}_d\dot{\boldsymbol{\omega}}_d\right), \tag{3.16}$$

where $f_3 \in \mathbb{R}$, the desired attitude $\mathbf{R}_d$ is given by

$$\mathbf{R}_d = [\vec{\mathbf{x}_{B,d}}; \vec{\mathbf{y}_{B,d}}; \vec{\mathbf{z}_{B,d}}] \in \mathbb{SO}(3), \tag{3.17}$$

and $\mathbf{f_c}$ is given by

$$\mathbf{f}_c = [0; 0; f_3]^T \in \mathbb{R}^3, \tag{3.18}$$

These control inputs $\mathbf{f}_c$ and $\mathbf{M}$ are designed to achieve asymptotic stability of the complete dynamics. Proof of the propositions are left in the paper.

3.3 Camera Calibration

Camera calibration is a fundamental process in computer vision, serving as the foundation for accurate and reliable image-based measurements and 3D reconstructions. At its core, camera calibration involves the precise determination of a camera's intrinsic and extrinsic parameters, which define how a 3D world is projected onto a 2D image. The intrinsic parameters encompass factors like the camera's focal length, principal point, and lens distortions, while extrinsic parameters describe the camera's position and orientation in relation to the world.
The importance of camera calibration cannot be overstated. It is the cornerstone of various applications and inaccurate calibration can introduce distortions, leading to significant errors in analyses and visualizations. In this case, camera calibration is of outmost importance given the needs of accurate environment perception. Bad camera calibration would translate to inaccurate point cloud generation and reprojection, leading to badly mapped environments.

Camera calibration is typically achieved by capturing images of a known pattern or scene from multiple angles, allowing for the estimation of the camera's internal characteristics and its relationship to the world. Advanced algorithms, often involving nonlinear optimization, are used to fine-tune these parameters, minimizing the difference between observed and predicted image features. The usual pattern used for camera calibration are either the chessboard or Aprilgrids, as shown in Fig. 3.5.
As an example, after having collected N images of a chessboard pattern, the camera calibration process can be summarized in the following steps:

- For each image:

    1. list the M $3D$ corner position in the pattern reference frame;
    2. find the corner position in the image reference frame.

- Initialize the calibration parameters to default values.
- Solve the minimization problem.

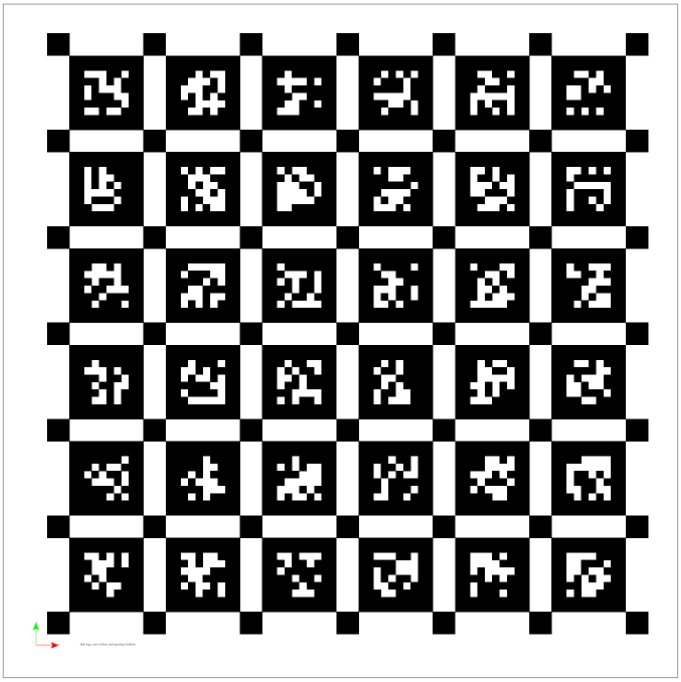


Figure 3.5: Example of Aprilgrid table for camera calibration.

Where the minimization problem is:

$$\arg\min_{k,f,u,v} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \left\| \mathbf{K}[\mathbf{I}|0]\mathbf{T}_i\tilde{\mathbf{P}}_{ij} - \tilde{\mathbf{p}}_{ij} \right\|^2, \tag{3.19}$$

and $\tilde{P}_{ij}$ is the corresponding point in the real world of the camera point $\tilde{p}_{ij}$. k, f, u and v are the intrinsic parameters of the camera matrix $\mathbf{K}$ and $\mathbf{T}$ is the rigid transformation between camera frame and world frame.

The resulting calibrated camera parameters, including focal length, distortion coefficients, and camera pose, are invaluable assets in the field of computer vision, providing the foundation for accurate and consistent image-based measurements and analyses.

3.3.1 Kalibr camera calibration toolbox

In order to calibrate the cameras used on-board the quadrotor, the Kalibr toolbox has been used. In particular, the multiple camera calibration tool [21, 22] is employed to get the intrin-

sic and extrinsic parameters of the stereo cameras.
First of all, a rosbag of the camera topics is needed. A bag is a file format in ROS for storing ROS message data. Bags (so named because of their .bag extension) have an important role and a variety of tools are available to store, process, analyze, and visualize them. Rosbag, which subscribe to one or more ROS topics, can store the serialized message data in a file as it is received. In this case, they are used for camera data logging. In particular, they are provided to the Kalibr toolbox, along with the camera models (pinhole, omnidirectional, ...), the topics name and the calibration target configuration (chessboard or Aprilgrids) in order to run the calibration process.

The output of the toolbox is similar to what presented here below:

```
cam0:
  cam_overlaps: [1, 2]
  camera_model: pinhole
  distortion_coeffs: [0.10991964802569217, -0.20591315602062998,

  -0.0010625517472595677, -0.00010589144830592429]
  distortion_model: radtan
  intrinsics: [871.2883628468551, 869.9645091763605,

  642.0717149764321, 363.99973152427737]
  resolution: [1280, 720]
  rostopic: /camera/color/image_raw/compressed

cam1:
  T_cn_cnm1:
  - [0.99968194380927, -0.025033648220278145,

  -0.0030541248338636384, -0.015170353376510008]
  - [0.025035195581193274, 0.9996864601381954,
```

```
  0.0004694662647537339, 0.000319965166510877]
  - [0.0030414147906621647, -0.0005457775606470463,

  0.999995225950066, -0.000128806986730986]
  - [0.0, 0.0, 0.0, 1.0]
  cam_overlaps: [0, 2]
  camera_model: pinhole
  distortion_coeffs: [0.010655057217719139, -0.013653134712594252,

  0.00013491679277989854, 0.0003197993006507017]
  distortion_model: radtan
  intrinsics: [407.2156773067505, 407.21439757381,

  425.8518697539763, 237.3132280207186]
  resolution: [848, 480]
  rostopic: /camera/infra1/image_rect_raw/compressed

cam2:
  T_cn_cnm1:
  - [0.999997706883476, -0.0001446954247101149,

  0.002136654162360765, -0.04988974835464921]
  - [0.00013491105315555164, 0.9999895084595456,

  0.004578730154728138, -0.00011693324290381489]
  - [-0.00213729426687157, -0.004578431396903121,

  0.9999872348880992, 6.568259336910408e-05]
```

```
- [0.0, 0.0, 0.0, 1.0]
cam_overlaps: [0, 1]
camera_model: pinhole
distortion_coeffs: [0.00937010510626876, -0.008377459723644879,

0.0006704570497315444, -0.00044930741452454086]
distortion_model: radtan
intrinsics: [407.1511945970345, 407.2256475460319,

425.1052640602839, 237.25284644837876]
resolution: [848, 480]
rostopic: /camera/infra2/image_rect_raw/compressed
```

where for each camera we can retrieve the distortion parameters, the intrinsic parameters and even the rigid transformation between camera frames, if needed. If the calibration has been done correctly then the cameras' parameters will be accurate, otherwise the calibration process needs to be repeated.

An indication on how accurate the camera parameters are, can be given by the reprojection error. The reprojection error of a camera calibration is a measure of how well the estimated intrinsic and extrinsic parameters fit the observed image points. The reprojection error is related to the accuracy of the estimated parameters. In particular, when the reprojection error is low, it means that the estimated intrinsic and extrinsic parameters, when used to project 3D world points onto the 2D image plane, closely match the actual observed image points. In other words, the calibration process has accurately modeled how the camera transforms 3D points into 2D image coordinates. Conversely, a high reprojection error indicates that there is a significant discrepancy between the projected 2D points based on the estimated parameters and the actual observed image points. This suggests that the estimated parameters are not accurate in describing the camera's behavior. So, the lower the reprojection error, the more accurate the intrinsic and extrinsic parameters are in describing the camera's behavior. In our case, reprojection errors lower than 1 gave good results. This is shown in Fig. 3.6.

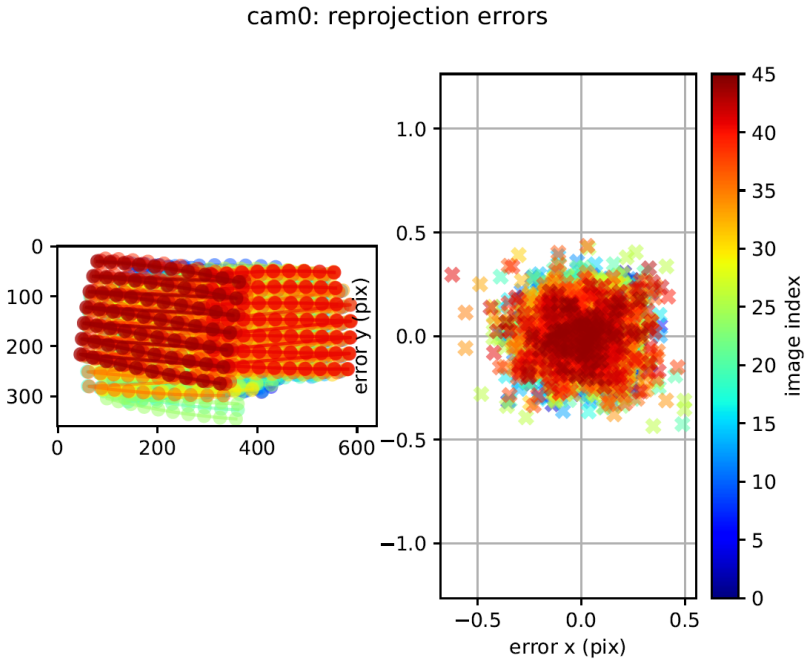


Figure 3.6: Example of camera reprojection errors.

In the following chapters it is detailed how the safe site detection and assessment process are implemented. In particular, in Ch. 4 it is shown how the metric and semantic information are obtained and processed to update the 2D binary map of the safe and unsafe landing zones. In Ch. 5 instead, it is discussed the final evaluation step and the autonomous landing procedure. Finally, in Ch. 6 are given more details about the system setup, the NN training and evaluation and some qualitative and quantitative results of the proposed framework.

4

Safe Site Detection

In the following sections it is shown how the semantic and metric information are obtained and processed, to define the suitability and safeness of a safe landing zone. Starting with section 4.1, it is presented how environmental features are extracted through a NN from RGB images. In section 4.2 instead, the terrain properties such as flatness, steepness and roughness are extrapolated from the point cloud of the environment, to evaluate critical 3D attributes for the safe landing. Combining both 2D and 3D information, a comprehensive evaluation of the environment can be done, not disregarding any relevant aspect.

4.1 Semantic information

4.1.1 Semantic image segmentation

Semantic image segmentation is a fundamental computer vision task that involves classifying each pixel in an image into a predefined category or class. It plays a crucial role in understanding and interpreting visual content, as it provides a detailed and meaningful understanding of the

objects and regions present within an image. This technology has found extensive applications across various domains, revolutionizing fields such as autonomous driving, medical image analysis, robotics, and more.

At its core, semantic image segmentation is used for scene understanding. By assigning a specific label or class to each pixel, it allows machines to differentiate between objects, people, animals, and background elements within an image. For instance, in the context of autonomous vehicles, semantic segmentation enables the car's perception system to identify pedestrians, road signs, other vehicles, and road boundaries, facilitating safe and efficient navigation.

The continual advancement of semantic image segmentation techniques has been driven by the pursuit of higher accuracy, faster processing speeds, and improved model generalization. Innovations in deep learning, particularly Convolutional Neural Networks (CNNs) [23], have been pivotal in achieving remarkable progress in this field. The development of more complex and deeper neural network architectures, such as U-Net [24], SegNet [25], and DeepLab [26], has significantly enhanced segmentation performance.

Furthermore, the availability of large annotated datasets, like COCO [27] and Pascal VOC [28], has played a pivotal role in training sophisticated segmentation models. Transfer learning, where pre-trained models on vast datasets are fine-tuned for specific tasks, has become a standard practice, further boosting segmentation accuracy.

Several techniques have been proposed to address the challenges of class imbalance, pixel-wise labeling, and real-time processing in semantic segmentation. The utilization of attention mechanisms, multi-scale feature fusion, and novel loss functions has contributed to more robust and accurate results. Additionally, the integration of semantic segmentation with other computer vision tasks, such as object detection and instance segmentation, has opened up new possibilities for holistic scene understanding.

In the proposed framework, segmented images are needed for assessing the suitability of a landing zone, a delicate procedure, crucial for the safety and success of aerial missions. In the context of landing zone assessment, the drone must process the captured imagery to identify suitable areas for landing, taking into account factors such as obstacles, terrain, and environmental conditions. To achieve this, a deep neural network is employed for image segmentation. However, the challenge lies in striking a balance between the accuracy of the segmentation results and the inference time of the network, given the limited computational resources available on the drone. In particular, for small aerial vehicles the on-board resources are limited, therefore a trade-off between computational speed and accuracy is often encountered. Optimizing this balance is critical because a more accurate segmentation can lead to safer and more precise landing zone identification, but it may also increase the time required for the drone to make decisions and take action. Conversely, prioritizing inference speed might sacrifice segmentation accuracy, potentially leading to sub-optimal landing zone assessments. Therefore, in the context of resource-constrained UAVs, it becomes imperative to develop and deploy efficient algorithms and models that can provide a good compromise between accuracy and inference time. This compromise ensures that the drone can make timely and reliable assessments of landing zones while operating within its computational limitations. This is why the BiSeNetV2 [29] model has been adopted. Compared to an encoder-decoder structure or the pyramid pooling modules often used in semantic segmentation, this network proposes a bilateral structure, namely treats the spatial details and categorical semantics separately to achieve high accuracy and high efficiency for real-time segmentation tasks. The features extracted by the two branches of the dual-pathway backbone are then merged together by an aggregation layer. Additionally, to enhance the inference time of the network, the trained BiSeNetV2 model is optimized it through the NVIDIA TensorRT library [30]. This significantly accelerates the network's performance and also reduces the model dimensions.

The overall goal of the network is to recognize image regions that are suitable for landing such as grass fields, pavements, roads, floors, etc. For each pixel in the image the NN associate a semantic meaning to it

$$f : (u, v) \mapsto C \tag{4.1}$$

where at each location (u, v) the pixel is characterized by a label C. Knowing the relation between the label $C \in \mathbb{R}^+$ and the corresponding class (e.g., $C : 0 \mapsto$ safe landing, $C : 1 \mapsto$ people, $C : 2 \mapsto$ obstacles, ...) we are able to infer if the region is suitable for landing.
An example of the segmentation results is given in fig. 4.1.

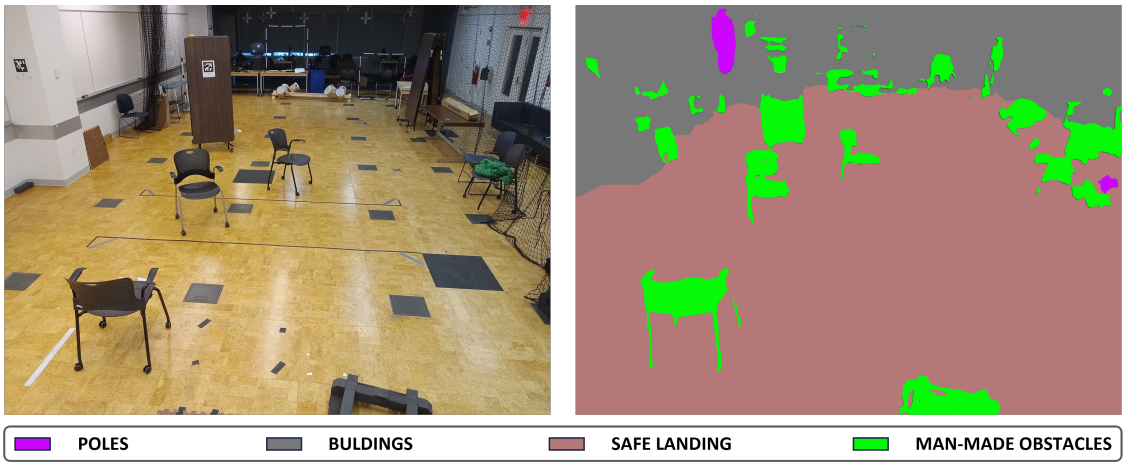


Figure 4.1: Segmentation result: on the *left* column the RBG image, on the *right* column the segmentation result.

4.2 Geometric information

Semantic information alone is not sufficient for a comprehensive understanding of the environment. With only 2D information from the segmented images, assessing critical terrain properties such as slope, roughness, and elevation variations becomes challenging. Hence, it is of utmost importance to also consider 3D information of the environment, which in this case is obtained through stereo vision.

4.2.1 Stereo vision and Epipolar geometry

Stereo vision, also known as stereo correspondence or stereo matching, is a technique used to recover depth information from a pair of stereo images taken from slightly different viewpoints. The key concept behind stereo vision is to find correspondences between points or features in the ”left” and ”right” images and use these correspondences to compute the disparity map,

which encodes the depth information.

1. The first step is to rectify the stereo images. Rectification is essential to ensure that corresponding points in the left and right images lie on the same scanlines, simplifying the correspondence search. This process involves finding the epipolar lines (lines along which corresponding points must lie) and warping the images accordingly. To better understand this processed a brief explanation on the epipolar geometry follows.

Epipolar geometry

Epipolar geometry is a fundamental concept in stereo vision that describes the relationship between two cameras and their images when establishing correspondences. It helps constrain the search for correspondences by defining epipolar lines and epipoles.
Consider a stereo camera setup with two cameras: the left camera ($C1$) and the right camera ($C2$). These cameras have their respective image planes, with the left image plane ($i1$) and the right image plane ($i2$). There's a baseline distance (B) between the two cameras, representing how far apart they are.
For a given point in the left image $x1$, there is a corresponding epipolar line in the right image. This epipolar line is the intersection of the right image plane ($i2$) with a plane called the epipolar plane (P). The epipolar line in the right image corresponding to point $x1$ in the left image can be represented as:

$$a_2x_2 + b_2y_2 + c_2 = 0 \tag{4.2}$$

Here, (x_2, y_2) are the coordinates of any point on the epipolar line in the right image, and a_2, b_2, and c_2 are coefficients determined by the camera's parameters.
The epipolar constraint states that the corresponding point $x2$ in the right image must lie on the epipolar line associated with point $x1$ in the left image.
The epipole ($e2$) in the right image is the point where all epipolar lines intersect. A similar epipole ($e1$) exists in the left image.

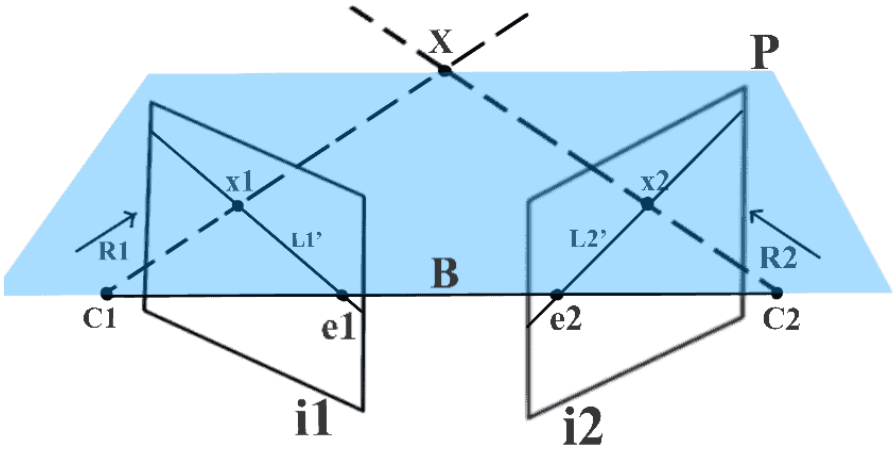


Figure 4.2: Visualization of the epipolar geometry for stereo vision.

The essential matrix (**E**) encapsulates information about the relative pose of the two cameras and is crucial for stereo vision. It is defined as:

$$\mathbf{E} = [\mathbf{t}]_x \mathbf{R} \tag{4.3}$$

Here, **t** represents the translation vector between the cameras, and **R** is the rotation matrix representing their relative orientation.
The fundamental matrix (**F**) relates image points to epipolar lines in both images and is computed from the essential matrix and camera intrinsic matrices. It is defined as:

$$\mathbf{F} = {\mathbf{K}_2}^{-T} \mathbf{E} {\mathbf{K}_1}^{-1} \tag{4.4}$$

Where $\mathbf{K}_2$ and $\mathbf{K}_1$ are the intrinsic matrices of the left and right cameras, respectively.
By understanding epipolar geometry and the principles of stereo vision, we can efficiently find corresponding points and calculate disparity maps, enabling the estimation of depth and 3D reconstruction.

2. After rectification, the goal is to find correspondences between points in the left and right images. This is often done by comparing pixel values along the epipolar lines. One common method for correspondence search is sum of squared differences (SSD) or normalized cross-correlation (NCC) matching. For each pixel in the left image, a window (often a small square) is defined around it. Let's denote this window as $W_L(x, y)$, where (x, y) represents the center pixel coordinates. A corresponding window is defined in the right image along the same epipolar line. This right image window, denoted as $W_R(x', y')$, corresponds to the same physical point as the left window due to the epipolar geometry. At this point, the similarity between these windows is computed using one of the following two formulas: *Sum of Squared Differences (SSD)*, that measures the sum of squared pixel intensity differences between the two windows and is defined as

$$\text{SSD}(x', y') = \sum_{i,j} [I_L(x + i, y + j) - I_R(x' + i, y' + j)]^2 \tag{4.5}$$

where $I_L(x+i, y+j)$ is the intensity of the pixel at coordinates $(x+i, y+j)$ in the left image, $I_R(x' + i, y' + j)$ is the intensity of the pixel at coordinates $(x' + i, y' + j)$ in the right image and the summation is performed over the window size.

Normalized Cross-Correlation (NCC), taht measures the normalized cross-correlation between the two windows as

$$\text{NCC}(x', y') = \frac{\sum_{i,j} [I_L(x + i, y + j) \cdot I_R(x' + i, y' + j)]}{\sqrt{\sum_{i,j} [I_L(x + i, y + j)]^2} \cdot \sqrt{\sum_{i,j} [I_R(x' + i, y' + j)]^2}} \tag{4.6}$$

where $I_L(x+i, y+j)$ and $I_R(x' + i, y' + j)$ are pixel intensities as before and the summation is performed over the window size.

The position (x', y') of the minimum SSD or maximum NCC is considered the correspondence, indicating the matching point in the right image for the selected point in the left image.

3. Disparity is the horizontal shift between corresponding points in the left and right images. It's inversely proportional to the depth of the scene. Disparity can be computed from the disparity equation:

$$D = x1 - x2 \tag{4.7}$$

where D is the disparity value, $x1$ is the x-coordinate of the point in the left image and $x2$ is the x-coordinate of the corresponding point in the right image.
Once we have the disparity map (a map of disparity values for each pixel), we can convert it to depth values using the stereo camera parameters. The depth Z is inversely proportional to disparity D and directly proportional to the baseline between the two cameras B and the focal length f:

$$Z = \frac{f \cdot B}{D} \tag{4.8}$$

where Z is the depth, f is the focal length of the cameras, B is the baseline between the two cameras and D is the disparity.

Incorporating 3D data allows for a more holistic assessment, enabling the drone to make informed decisions about landing zones that consider both semantic and metric characteristics. Consequently, from the stereo pair it is derived a disparity map that enables us to capture and analyze the encoded depth of the scene.

However, this representation is not suitable for the next processing steps. In order to analyze the terrain properties we first need to convert the disparity maps into point clouds. By utilizing the camera's intrinsic and extrinsic parameters, the Cartesian coordinates (x, y, and z) of each point can be calculated, generating the associated point cloud as shown in Eq. (4.9)

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} z \cdot \frac{u - c_x}{f_x} \\ z \cdot \frac{v - c_y}{f_y} \\ \frac{f \cdot B}{D} \end{bmatrix}, \tag{4.9}$$

where x, y and z are the 3D coordinates in the point cloud and u, v are the pixel coordinates in the image corresponding to the point. c_x and c_y are the principal point offset (usually half the image width and height), f is the focal length of the camera, B the baseline between the left and right cameras and D the disparity value at the corresponding pixel in the disparity map. Once the point cloud has been generated, it is re-projected onto the segmented image to only keep the points associated to a safe landing region. This is done as shown in the following equation

$$\begin{bmatrix} u' \\ v' \\ w \\ * \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_{SL}^{RGB} & \mathbf{t}_{SL}^{RGB} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \tag{4.10}$$

where f is the focal length and c is the optical center. Starting from a point (x, y, and z) we determine its corresponding pixel coordinate (u, v), using the transformation equations

$$u = \frac{u'}{w},\ v = \frac{v'}{w}. \tag{4.11}$$

It is important to note that in this case, the RGB camera frame ($\mathcal{F}_{RGB}$) and the left stereo camera frame ($\mathcal{F}_{SL}$) are not perfectly aligned. Thus, it is taken into account the rigid transformation between the two frames ($\mathbf{H}_{SL}^{RGB} \in \mathbb{SE}(3)$), as indicated in the second term of the right-hand side of Eq. (4.10). This adjustment ensures the accurate projection of 3D points, perceived by the stereo camera, onto the 2D image, taken by the RGB camera. This idea is also

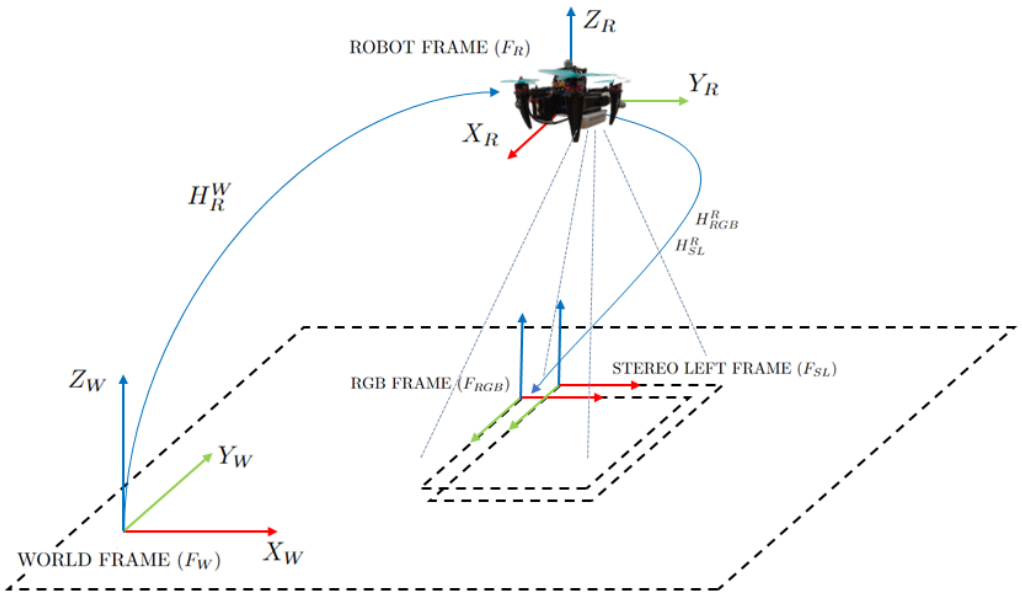


Figure 4.3: Transformation between frames.

pictured inf Fig. 4.3.

Once we have the pixel coordinates (u, v), we check if the corresponding image location falls within a class C associated with a safe landing site. In the positive case, we retain the $3D$ point; otherwise, it is flagged as unsafe and discarded. By only considering the safe points we speed-up the computation, improving the performance of the algorithm.

Subsequently, the point cloud is filtered and down-sampled to enhance its quality and suitability for safe landing site detection. This process is implemented by utilizing the Point Cloud Library (PCL) [31]. The point cloud is initially down-sampled using a voxel grid filter. The "leaf size" parameter controls the voxel size, with larger values resulting in greater down-sampling. In our case, we choose to retain one point each $0.1 \times 0.1\,\mathrm{m}^2$, striking a balance between processing speed and accuracy. Then, a statistical outlier removal filter is applied in order to remove the points that significantly deviate from the heuristic distribution of the point cloud. To further improve the point cloud quality, we employ a Moving Least Square (MLS) smoothing filter, resulting in a smoother point cloud less affected by noise. Finally, a plane fitting algorithm [32] is executed to identify planar regions within the point cloud. This allows to retain only the points associated to a flat surface, that also satisfy predefined slope and roughness thresholds.

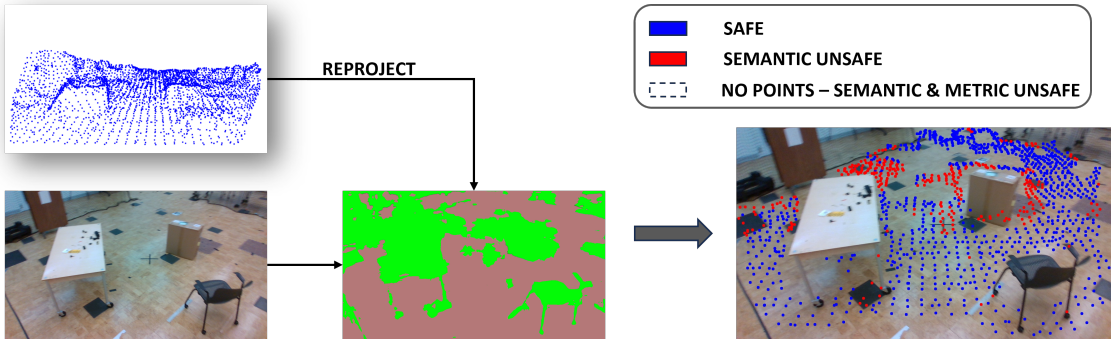


Figure 4.4: Point cloud processing: the point cloud of the environment (*top left*) is projected onto the segmented RGB image (*center*) to only retain the safe point based on the semantic information. The remaining points are further processed to evaluate also the slope and roughness thresholds.

However, to correctly assess the metric properties of the scene, we first have to ensure that the point cloud is aligned with the world reference frame ($\mathcal{F}_W$), which conveniently coincides with our map reference frame ($\mathcal{F}_{2DM}$). To this end we consider the rigid transformation between the left stereo frame and the world frame $\mathbf{H}_{SL}^{W} \in \mathbb{SE}(3)$, since the point cloud is originally aligned with $\mathcal{F}_{SL}$. By leveraging the drone's odometry, we can compute $\mathbf{H}_{SL}^{W}$ and apply the necessary compensation to get the true plane's inclination. Moreover, in this way, we also compensate for the drone's Roll, Pitch, and Yaw (RPY) rotation since it can be tilted with respect to the surface, thus influencing the plane's orientation. If the drone is not parallel to the plane and its orientation will not be compensated then, the drone's orientation will be considered as the plane's orientation, affecting the ture plane's inclination.
Subsequently, the plane's inclination can be computed considering the angle between its normal vector $\hat{\mathbf{n}} = (\mathbf{n_x}, \mathbf{n_y}, \mathbf{n_z})$ and the $\mathbf{z}$-axis of the world frame (parallel to the gravity vector $\hat{\mathbf{g}}$) as

$$\varphi_x = \text{atan2}(\mathbf{z}, \mathbf{n_y}), \ \varphi_y = \text{atan2}(\mathbf{z}, \mathbf{n_x}). \tag{4.12}$$

In our specific setup, any planes with inclinations exceeding 15° are considered unsafe. Furthermore, we can establish whether a point qualifies as a plane inlier or not by evaluating its distance to the fitted plane, thus defining the maximum roughness admitted. To retrieve the

distance of a point from the fitted plane, we compute the magnitude of the perpendicular vector connecting the point to the plane as

$$\text{dist} = \frac{|Ax + By + Cz + D|}{\sqrt{A^2 + B^2 + C^2}}, \tag{4.13}$$

where x, y and z are the point's coordinates and A, B, C and D are the plane coefficients. In our case if a point is more than 0.05 m away from the plane it is considered unsafe. The slope and roughness thresholds are reported in Tab. 4.1.

THRESHOLD	**VALUE**
SLOPE	15°
ROUGHNESS	0.05m

Table 4.1: Thresholds parameters.

This pipeline is executed separately for each scene, thereby contributing to an enhanced solution. The overall map is a composition of processed images and point clouds, all of which satisfy the safe site criteria detailed in this section.

5

Environment Assessment and Autonomous Landing

In this chapter it is discussed the final evaluation process and the autonomous landing procedure. In particular, in section 5.1 it is shown how, from the 2D map and the drone's position, the most suitable landing area is computed. In section 5.2 instead, it is presented the autonomous landing behaviour.

5.1 Environment Assessment

Up to this point, the knowledge acquired has given a general understanding of the distribution of safe and unsafe points across the map. However, the objective is to pinpoint the optimal landing location.

To achieve this, one should search for a patch on the map that spans an area approximately 1.85 times the size of the drone. This patch must be a sufficiently large region consisting exclusively of safe points and accounting for an additional safety margin. The patch is considered

safe when every cell within it has a safety probability "p" of 95% or higher. This means that we ensure all cells in the designated patch are highly likely to be safe, with safety probabilities ranging from 0% to 100%, where 100% indicates complete safety. Once this area is found, we compute the associated cost

$$J = \alpha \cdot J_d + \beta \cdot \frac{1}{J_{un}} \tag{5.1}$$

with $\alpha + \beta = 1$ and $\alpha,\ \beta \in [0, 1]$, J_d is the distance between the center of the safe zone and the drone while J_{un} is the distance between the center of the safe zone and the closest unsafe point. J_d and J_{un} are both computed as Euclidean distances.

This approach prioritize landing zones that are not only closer to the drone, but also farther away from unsafe areas. Fine-tuning the parameters α and β enables us to adjust the behavior for identifying the safest landing zone. By increasing α and reducing β, the algorithm tends to find a safe landing zone that is closer to the drone, but potentially nearer to obstacles. Instead, by increasing β and reducing α the algorithm tends to find safe landing zones farther from the obstacles but at the same time farther from the drone. The parameter's choice depends on the desired behaviour. If we want to minimize the drone's distance from a possible landing area then we will choose a scenario where $\alpha > \beta$, otherwise if we want to maximize the distance from the surrounding obstacles we will choose $\alpha < \beta$.

Additionally, our final evaluation takes into account the drone's battery consumption, which is influenced by the Euclidean distance between the drone and the 3D landing location, as shown in [33]. By iterating this last step over the whole map we aim to minimize the cost function and only keep the safest landing zone, namely the one associated to the lowest cost possible. Whenever new areas are overflown or the drone changes its position, the environment is re-perceived, the map is continuously updated, and the best safe landing zone is published accordingly.

In summary, our safe site detection pipeline employs a comprehensive evaluation of region safety, taking into account both semantic information and geometric information, including flatness, roughness, steepness, a distance transform and the drone size.

5.2 Autonomous Landing

The landing step is pretty straightforward and doesn't require any particular process to be involved. Once the landing is required, a minimum snap trajectory generation algorithm [34, 35] is used to find a path from the drone's actual position to the safe landing spot. The "*snap*" here pertains to the fourth derivative of position concerning time, signifying the rate of change in acceleration. Therefore, a "*snap trajectory*" considers not just the drone's basic motion parameters (position, velocity and acceleration) but also its jerk profile throughout the entire flight. The core principle of the "minimum snap" trajectory generation algorithm is to identify and compute trajectories that keep jerk to a minimum. This is paramount for attaining smoother and more precisely controlled drone movements, particularly in scenarios demanding rapid and agile flight maneuvers.
In fact, given a curve that we want to follow, we can calculate the necessary force and torque to move the quadrotor along it, as shown from the quadrotor's dynamic equations introduced in chapter 3.2. We can observe that the quadrotor can create a net force along the direction of its propellers. This net force has to move the quad along the curve, thus the quadrotor's orientation needs to be tangent to the curve. The quadrotor also create a torque around all 3 world axis. Thus the quadrotor's torque needs to change its orientation as it moves along the curve. Now, to calculate the torque needed to change the orientation of the quadrotor along the curve, we take the second derivative of the quadrotor's orientation. This is defined by the acceleration direction, so we take the second derivative of the acceleration, which is the fourth derivative of the position. So, torque is directly related to the fourth derivative of position (the *snap*). Overall, by minimizing the snap we will have a curve that requires the least reorientation of the quadrotor and will guarantee smoothness.

Once the trajectory between the starting and the goal destination is found, the drone follows this two-steps behaviour:

- Fly above the safe landing zone.
- Decrease the height until it reaches the ground level.

This approach is employed since no collision avoidance behaviour has been utilized, thus prioritizing safety and avoiding any possible crash. However, map knowledge could be integrated in this process to make the drone aware of the obstacles distribution in the environment. In such a way a single-step landing behaviour could be implemented, since the trajectory would be generated also considering the obstacle's locations.

5.3 Safe Site Detection and Environmental Assessment Pseudocode

The overall visual environmental assessment pipeline can be condensed in pseudocode 5.1.

Algorithm 5.1 Safe Landing Site Detection and Evaluation

```
for each RGB image and corresponding PC
    Project PC onto segmented RGB
    if Segmented Label at (u, v) is Safe (C = C_S)
        Extract Safe PC Points → PC_S : (x, y, z)
    end if
    if PC_S : (x, y, z) has Safe Slope AND Safe Roughness
        Store Safe Landing Site → PC_SL : (x, y, z)
        Update 2D map : PC_SL : (x, y, z) → 2D_MAP
    end if
end for

for each cell of 2D_MAP
    if Probability of 5x5 Patch > 95%
        Identify Safe Patch → SAFE_PATCH
    end if
    if Cost Function J(SAFE_PATCH) is Minimal
        Update Best Landing Site → BEST_LANDING
    end if
end for

return BEST_LANDING
```

For each perceived scene we first verify the semantic and metric attributes, to update the 2D

occupancy grid of the safe and unsafe location, here called "$2D\_MAP$". Simultaneously, each time the drone moves or the environment is updated, we check if there is a large enough area, to fit the drone and guarantee a safety margin for safe autonomous landing. Once this area is found, we compute its relative cost based on Eq. (5.1). If the patch found has a lower cost than the previous or initial patch, then a new best landing area has been found. Once the landing is required, the drone will fly towards the most suitable landing zone.

6 Results

To validate the proposed approach, a series of real experiments are executed in a challenging, large indoor environment measuring $26 \times 10 \times 4$ m^3, situated at the Agile Robotics and Perception Lab (ARPL, New York University). In particular, the validation process involved two key aspects:

(a) **environmental changes**, to simulate different evaluation and landing scenarios;

(b) **waypoints diversification**, to simulate different inspection strategies.

6.1 System Setup

The employed drone is a compact and versatile system, with a diameter of 0.27 m and a weight of 1.1 kg, built and developed within the NYU's ARPL laboratory. It is equipped with a PX4 Autopilot flight controller, for high level position control, and a Nvidia Jetson NX computing board. The PX4 flight controller has been calibrated through the QGroundControl control

station interface [36].

For convenience, without loss of generality of the proposed approach, two stereo cameras are employed to decouple the localization and safe landing evaluation, due to the camera characteristics, introducing as well some redundancy in the system. The first one, a RealSense T265 tracking camera, employed to obtain a robust VIO and the second one, the RealSense D435i, responsible for mapping the environment and detecting the safe landing zone. Depending on mission requirements and constraints, it is also possible to effectively operate with a single stereo camera, either pointing directly downward or tilted at a 45-degrees angle, not affecting the approach and results of this work.

In Fig. 6.1 it is shown the drone structure.

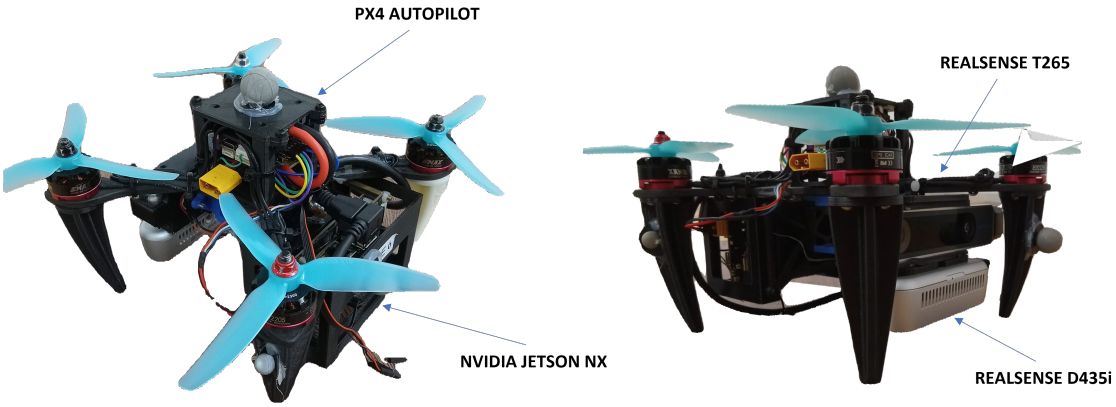


Figure 6.1: Drone structure.

To efficiently manage the different modules of the system, it is employed a nodelet-based approach, a feature inherent to ROS. Nodelets optimize computation and minimize latency by sharing memory space among individual processing nodes within a single ROS node. This design choice significantly reduces the need for inter-process communication, eliminating the associated overhead. By harnessing the power of ROS and the efficiency of nodelets, this system not only achieves real-time performance but also ensures optimal resource utilization. This synergy allows us to meet the stringent demands of these applications, where timely and resource-aware processing is of paramount importance.

6.2 Neural Network Training and Evaluation

To train the network, it is leveraged the ADE20K semantic scene parsing dataset [37], since it provides a wide set of indoor and outdoor environments, covering a wide range of classes and examples relevant for addressing the safe landing task. Also other datasets were taken under consideration, for example the Pascal context 59 Dataset [38] and the CityScapes Dataset [39]. However, those datasets does not provide a comprehensive set of environments. In particular, they only provides outdoor examples thus disregarding all the indoor environments, needed for this project. Other minor datasets were also analyzed but not considered as well, given their limited size and examples.

Rather than using the original 150 classes, we have manually clustered them into 11: *water, people/animals, sky, trees, man-made obstacles, nature obstacles, safe landing site, light, vehicles, background, buildings*. In particular, for each of the new 11 classes, the previous 150 were clustered in a meaningful way, i.e. in the "*building*" class we will find walls, buildings, edifices, windows, doors, houses, skyscrapers and so on. In the "*man-made obstacles*" class we will find beds, tables, chairs, desks, boxes, shelves, pillows, etc. In such a way we enhance the inference time and we only keep classes meaningful to our task.

The training pipeline is based on the robust and versatile PyTorch-based mmsegmentation framework developed by openMMLab [40]. OpenMMLab is renowned for its contributions to computer vision and deep learning, and mmsegmentation is a shining example of their commitment to advancing the field. It provides a user-friendly and highly customizable platform for developing state-of-the-art semantic segmentation models, making it an ideal choice for our training pipeline. One of the standout features of mmsegmentation is its extensive collection of pre-implemented segmentation algorithms, network architectures, and pre-trained models. These resources serve as a solid foundation for the training process, allowing us to efficiently explore various model architectures and fine-tune them for our specific use cases. Moreover, mmsegmentation seamlessly integrates with the PyTorch ecosystem [41], capitalizing on PyTorch's flexibility and scalability.

The training consists in an iteration-based process using a Stochastic Gradient Descent (SGD)

optimizer for 160 K iterations, and two NVIDIA GPUs with batch size equal to 8. The optimizer parameters are reported in Table 6.1.

Optimizer parameters	**Value**
LR	0.05
Momentum	0.9
Weight Decay	0.0005
LR Decay Type	Polynomial Decay
Polynomial Decay parameters	**Value**
LR_{min}	1×10^{-4}
Power	0.9

Table 6.1: NN Training parameters

6.2.1 Advanced Neural Network training

In the quest for optimizing neural network training, it has been chosen a combination of techniques that go beyond the conventional use of Stochastic Gradient Descent (SGD) alone. The reason for adopting momentum and weight decay is because they offer a comprehensive and more advantageous approach. Collectively, they serve as the pillars of enhanced convergence, regularization, and adaptable learning rates. Through this combination, we not only accelerate convergence but also improve the network's generalization capabilities, making it more robust and versatile in handling complex tasks.

Momentum [42] is a key technique used to accelerate the training process and improve convergence in neural networks. It addresses the problem of SGD being sensitive to noisy gradients, especially in regions of the loss landscape with shallow or flat minima. By introducing momentum, we can achieve the following advantages:

- **Smoothing Gradients**: Momentum accumulates a moving average of past gradients, reducing the impact of noisy gradient updates. This leads to smoother convergence and faster training.

- **Escape Local Minima**: The accumulated momentum allows the optimizer to escape shallow local minima and explore the loss landscape more effectively. This helps in finding better and more generalized solutions.

Mathematically, momentum can be incorporated into the update rule as follows:

$$\begin{aligned} v_t &= \beta \cdot v_{t-1} + \nabla L(\theta_t) \\ \theta_{t+1} &= \theta_t - \alpha \cdot v_t \end{aligned} \tag{6.1}$$

where v_t is the momentum at iteration t, β is the momentum coefficient, $\nabla L(\theta_t)$ is the gradient of the loss function with respect to the model parameters θ_t and α is the learning rate.
Weight Decay [43], also known as L2 regularization, is employed to combat overfitting by adding a penalty term to the loss function. It encourages the model to have smaller weight values, which results in a simpler and more generalizable model. The key benefits of weight decay are:

- **Regularization**: It acts as a form of regularization, helping to prevent the model from fitting the training data too closely, which can lead to overfitting.
- **Improved Generalization**: Smaller weight values often lead to a model that generalizes better to unseen data, improving its performance on validation and test sets.

Weight decay can be applied by adding a regularization term to the loss function:

$$L_{\text{total}} = L_{\text{data}} + \lambda \sum_i \|\theta_i\|_2^2 \tag{6.2}$$

where L_{total} is the total loss, L_{data} is the data loss (e.g., cross-entropy loss), λ is the weight decay coefficient and θ_i are the model weights.
Finally, **polynomial Decay** [44] is a learning rate scheduling technique often used with SGD. It gradually reduces the learning rate during training, starting with a higher value and smoothly annealing it towards a minimum value. This approach is useful because:

- **Stable Training**: Initially, a high learning rate allows for rapid convergence, while gradually reducing it helps stabilize training as it approaches a minimum.

- **Fine-Tuning**: The learning rate reduction near the end of training allows for fine-tuning the model, enabling it to settle into a better minimum.

Polynomial decay can be defined as:

$$LR(t) = LR_{\text{min}} + (LR - LR_{\text{min}}) \cdot (1 - T/t)^{\text{power}} \tag{6.3}$$

where $LR(t)$ is the learning rate at iteration t, LR is the initial learning rate, LR_{min} is the minimum learning rate, T is the total number of iterations and power is a parameter controlling the rate of decay.

To further improve the quality of our segmentation results, we also performed a fine-tunig on the NN using a custom indoor dataset. The dataset includes approximately 1.2 K images of common scenes within our lab environment. Notably, the environments created for testing differ from the ones used during the fine-tuning phase, since they incorporate new scenes and objects for evaluation. Since manual labeling is a very time-consuming activity, the Segment Anything Model (SAM) [45] is used to facilitate the mask creation process.
SAM is designed to address the challenges of creating high-quality masks for various objects in images. It is trained on 11M images with over 1B masks and can produce valid segmentation masks in real-time, when prompted with different types of inputs such as points, boxes, and text. Once the masks are retrieved, we can assign the correct labels to each one of them, thus identifying the ground truth of each image.
To further accelerate this process it was also tried to use a fine tuned version of SAM. Personalize SAM [46] present an efficient one-shot fine-tuning variant that make us able to address the attention on the mask-generation process to the ones that are needed to be extracted from the image. Personalize SAM enables a more automatized way of finding segmentation masks. By providing some examples of already segmented images it is possible to fine-tune an added layer and make the model extract only the segmentation masks that are under interest by the user. However, without success, the fine tuning didn't improve the results as desired. In the end this solution was not adopted.
Furthermore, during training we employ a data augmentation pipeline to increase the dataset size. This pipeline is based on random resizing, random cropping, random flipping and photometric distortion. The fine-tuning parameters coincide with the one specified in Table 6.1, with the exception that training continued for another 80 K iterations. This resumed from $LR = 10^{-4}$ and finished with $LR_{\text{min}} = 1 \times 10^{-5}$.

The NN runs on-board the Jetson NX at 7.1 Hz and obtains a mean Intersection over Union (mIoU) of 67.51% and a mean Accuracy (mAcc) of 85.21%. For a qualitative evaluation of the segmentation results, please refer to Fig. 6.2.
In particular, we can observe in Tab. 6.2 the Intersection over Union (IoU) and the accuracy

Class	IoU (%)	Acc (%)
buildings	68.25	80.27
landing_site	61.61	69.87
water	33.66	52.18
sky	85.21	92.36
people_animals	33.06	55.84
vehicles	36.61	60.83
background	25.74	35.68
trees_poles	53.44	71.2
light	7.08	7.59
man_made_objects	45.98	69.53
nature_objects	0.0	0.0

Table 6.2: IoU and accuracy values for the BiSeNetV2 11 classes segmentation, before the fine-tuning

Class	IoU (%)	Acc (%)
landing_site	80.57	83.71
unsafe_landing_site	54.45	86.71

Table 6.3: IoU and accuracy values for the BiSeNetV2 binary segmentation, after the fine-tuning

results of the 11 classes, before the fine tuning.

And after the fine tuning, where all the classes unsafe for landing were clustered together, the results are presented in Tab. 6.3.

We can clearly see that before the fine tuning, the mean accuracy (mAcc) and the mean IoU (mIoU) where 59.24% and 46.26% respectively. While after the fine tuning both values where increased to 85.21% and 67.51%.

6.3 Environment Assessment and Autonomous Landing

In our test scenario, we operate with a map resolution of 0.1 meters, while the designated safe landing zone measures 0.5×0.5 m^2. The acquisition of RGB images and the stereo pair occurs at a rate of 30 Hz, whereas the disparity maps runs at 3 Hz, which suffices for our operational speeds. The processed point clouds are instead published at a frequency of 1.1 Hz, which are

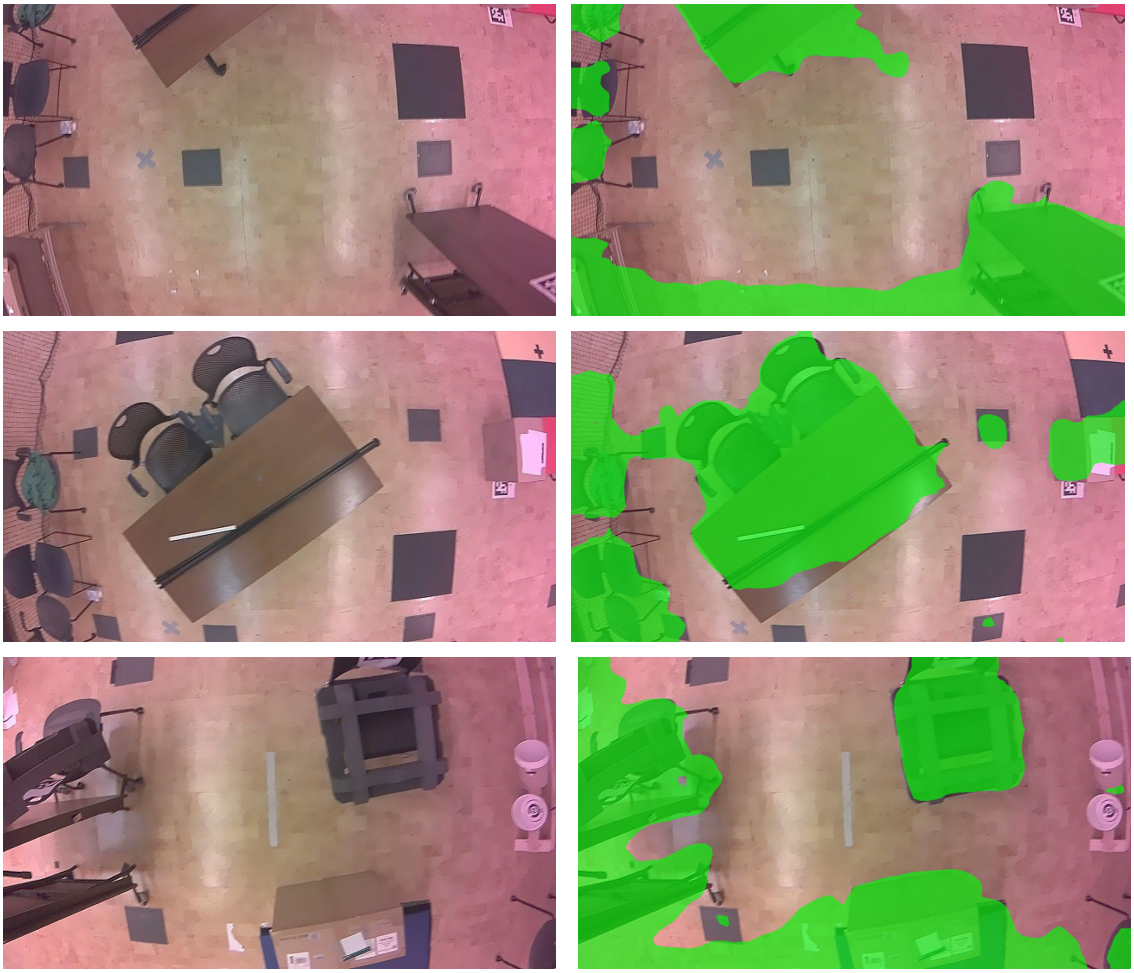


Figure 6.2: Segmentation results of the BiSeNetV2 model. On the *left* column the RGB images, on the *right* column the segmentation results.

then used to update the 2D map, while the safe landing locations have a slightly higher frequency. However, if better performances are required, we have the flexibility to increase the updating frequency of the processed points. Finally, VIO runs at 100 Hz.

For our tests, we select $\alpha = 0.65$ and $\beta = 0.35$. As detailed in Eq. (5.1), we empirically observe that these settings prioritize the term related to the drone's proximity to the safe landing area over the distance between the safe landing site and obstacles. Moreover, the slope and roughness thresholds are set respectively to 15° and 0.05 m.

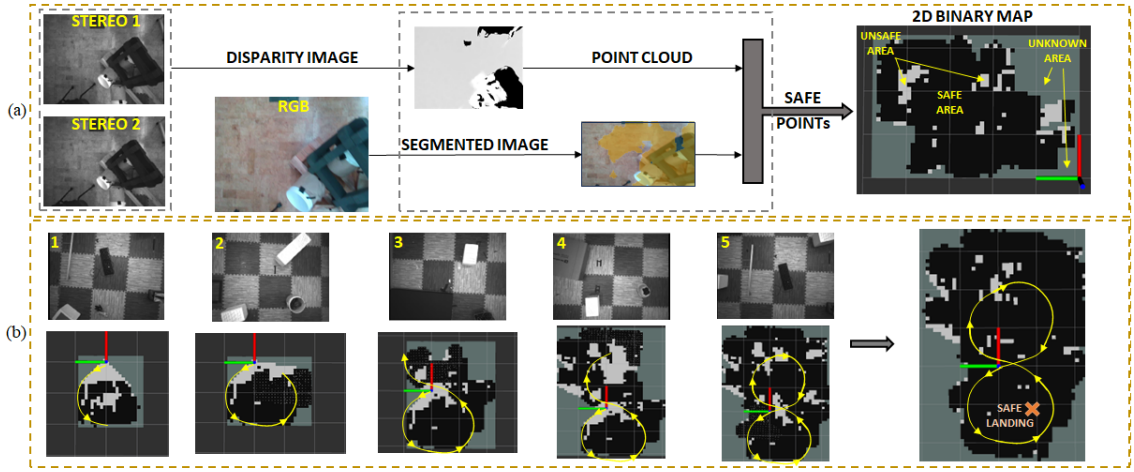
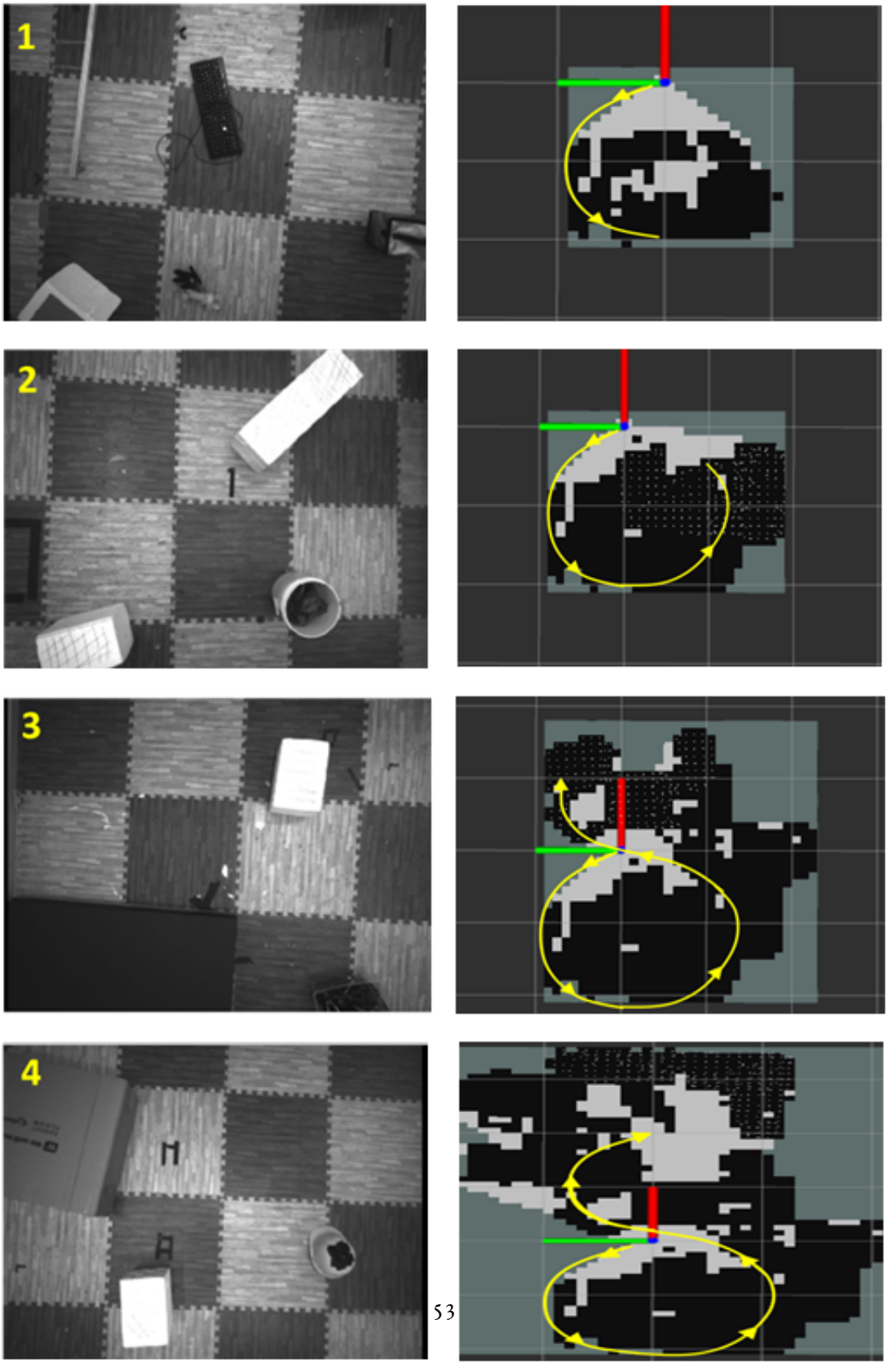


Figure 6.3: *(a)* Data acquisition & processing pipeline for the map creation and *(b)* Site evaluation and safe autonomous landing experiment in a low height, middle density environment scenario with an "8" navigation pattern.

The proposed pipeline is tested in several scenarios, including different obstacle heights and densities and different navigation patterns and speeds. In Fig. 6.3, we showcase the mapping process and a full experiment in a low height, middle density obstacles environment. After take-off, the drone follows an "8" navigation pattern and performs a couple of flight runs over the environment. As we can see from the succession of images in Fig. 6.3 (b) and 6.4, each time the drone perceive new areas, by following its path, the map is updated accordingly. Once the navigation behaviour is concluded, the 2D occupancy grid is fully updated and the drone can implement the final environmental assessment. When the safest landing area is identified, the drone finally implements the autonomous landing. The last picture in Fig.6.3 (b) and 6.4 show the whole map with the safe landing zone location and the drone's path. Out of 7 tests per-


1
2
3
4

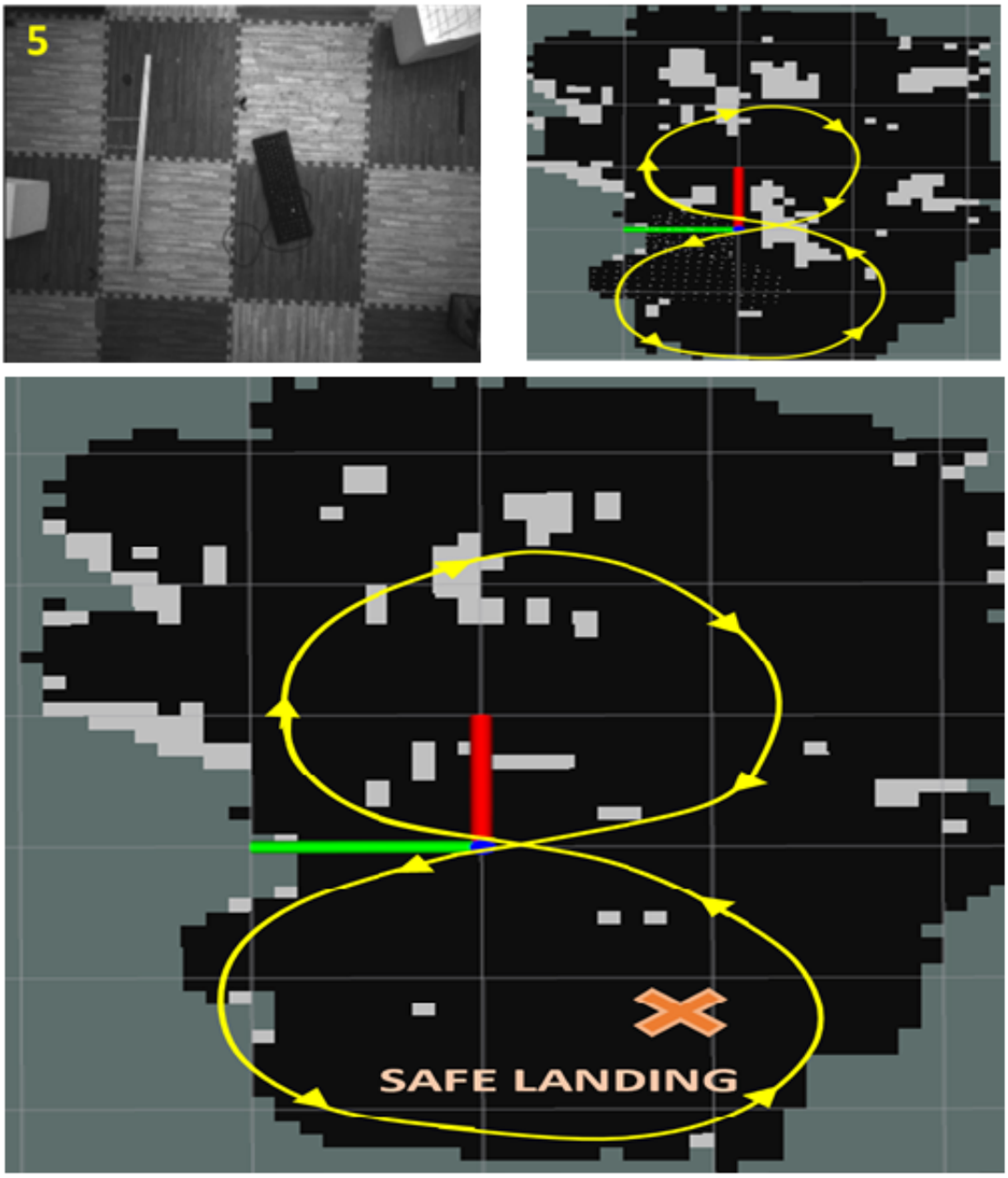


Figure 6.4: Full site evaluation and safe autonomous landing experiment in a low height, middle density environment scenario with an ”8” navigation pattern. Succession of images showing the perceived environment and the mapping process.

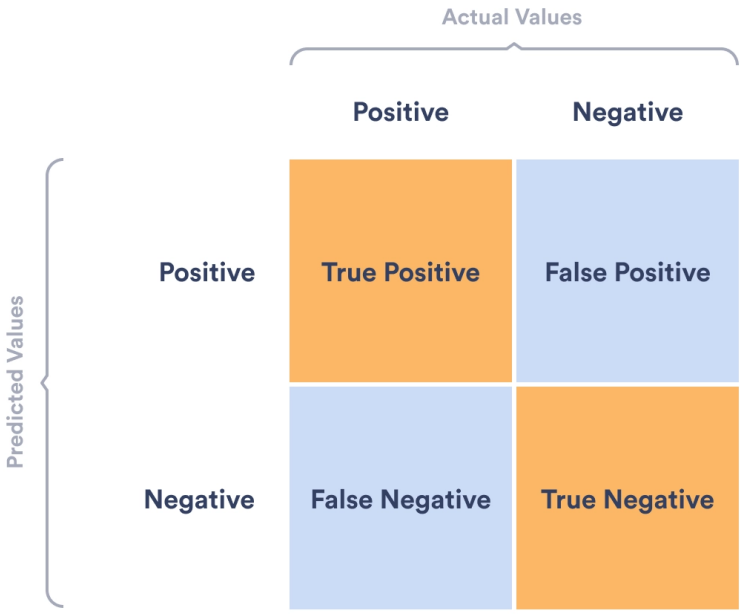


Figure 6.5: Confusion matrix

formed in different challenging environments, the drone is able to safely lands 6 times, showing its capability to detect a safe landing zone with an overall **success rate** of 85.71%. The unsuccessful landing is not attributed to any errors in segmentation or metric data but rather to the grid discretization. In this specific experiment, the grid size was excessively large in comparison to certain low-height obstacles. Through testing with a slightly smaller grid, it can effectively be addressed this issue without impacting computational efficiency. By overlaying the created 2D occupancy grid with the top view of the environment, we can also quantitatively evaluate the number of zones correctly "classified" as safe or unsafe.

To evaluate the classification accuracy we consider the following formulation:

$$\text{Acc} = \frac{TP + TN}{TP + FP + FN + TN}, \tag{6.4}$$

where TP are the true positive, TN the true negative, FP the false positive and FN the false negative. To better understand this concept we can refer to Fig. 6.5.

The pixel accuracy represent the percent of pixels that are classified correctly. The *true posi-*

tives are the number of pixel correctly classified as safe landing zones and the *false positives* are the number of pixel erroneously classified as safe landing zones. Analogously, *true negatives* and *false negatives* are the pixels that are correctly and incorrectly classified as non-safe landing zones respectively.
However, accuracy can be a misleading metric for imbalanced data sets. In the case of safe site detection, if the result of the segmentation is such that there are no hazards, the accuracy value can still be very high even in images in which there actually are different obstacles.
For this reason a tuned version of the accuracy metrics is considered: the Balance Accuracy (BAcc). The BAcc normalizes true positive and true negative predictions by the number of positive and negative samples, respectively, and divides their sum by two. In a binary classification task the formulation is given by

$$\text{Balanced Accuracy (BAcc)} = \frac{\text{TPR} + \text{TNR}}{2} \tag{6.5}$$

where TPR = TP/(TP + FN) is the true positive rate (or recall), TNR = TN/(TN + FP) is the true negative rate. TPR and TNR are also called sensitivity and specificity respectively.
Other two useful metrics for the performance evaluation are the precision, which indicate how accurate the model predict the safe (positive) points, and recall (or true positive rate) which indicates how many positive points the model correctly detected with respect to the total real positive ones.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{6.6}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \text{TPR} \tag{6.7}$$

The last metric is the F1 score, or Dice Coefficient, which combines precision and recall (it is the harmonic mean of precision and recall).

$$\text{F1} = 2 \times \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6.8}$$

Geometrically it represents two times the area of overlap divided by the total number of pixels in both the detected and real masks. F1 is very similar to the Intersect over Union quantity.

In Fig. 6.6, we show some of the results obtained during testing in four environments, considering different obstacles heights, densities, and navigation patterns. The mean accuracy (mAcc) in identifying safe landing zones is approximately 81.4%, where $5,4\%$ are false negatives and $13,2\%$ are false positives.

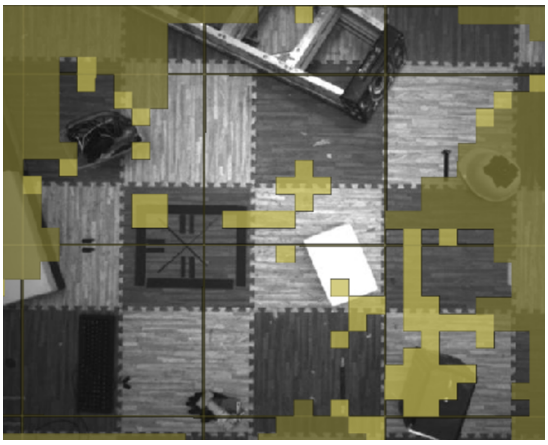
Finally, in Tab. 6.4 we present the several metrics values, as discussed before. In particular, for each image of Fig. 6.6 the Acc, BAcc, Precision, Recall and F1 are computed.

Image	Acc	BAcc	Precision	Recall	F1
Image (a)	0.765	0.614	0.784	0.937	0.854
Image (b)	0.826	0.609	0.888	0.915	0.901
Image (c)	0.914	0.668	0.923	0.985	0.953
Image (d)	0.751	0.609	0.777	0.921	0.843

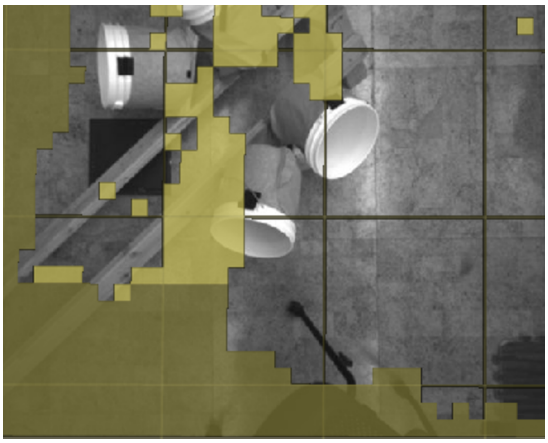
Table 6.4: Metrics results.

These metrics collectively provide insights into how well the classification is performed, considering aspects like correctness, class balance, and the balance between precision and recall.
The accuracy (Acc) measures how many of the classifications were correct for each image. On average, about 81.4% of the classifications across the proposed examples.
Balanced Accuracy (BAcc) takes class imbalance into account and computes the average accuracy across all classes. It helps handle imbalanced datasets. On average, the balanced accuracy is approximately 62.5%.
Precision quantifies how many of the positive predictions made by the model were correct. On average, about 84.3% of the positive predictions were correct.
Recall (sensitivity) measures how many of the actual positives were correctly identified by the model. On average, about 93.9% of the actual positives were correctly identified.
Finally, the F1 Score, which is the harmonic mean of precision and recall, provides a balanced measure of both precision and recall. On average, the F1 score across is 88.8%.

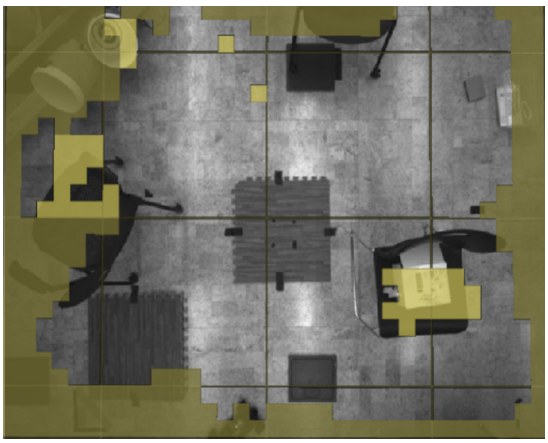
(a) High density, low height obstacles scenario. Acc = 76.5%.



(b) Long waypoints navigation pattern scenario. Acc = 82.6%.



(c) Medium density, medium height obstacle scenario 1. Acc = 91.4%.



(d) Medium density, medium height obstacle scenario 2. Acc = 75.1%.

Figure 6.6: 2D binary map of safe and unsafe landing locations, overlayed to the real environment. The light green regions are unsafe while the dark green are unknown and still to be explored. Both of them are hazardous areas for landing.

7 Conclusion

In this work, it has been presented a visual approach to autonomously detect safe landing sites on-board a quadrotor, with limited SWaP resources. The proposed approach allows accurate and efficient safe landing detection since it combines both semantic and metric information and directly computes a 2D binary map of the overflown environment, thus avoiding the creation of expensive elevation maps. Furthermore, through several tests, it is shown the ability to guarantee real-time, safe autonomous landing in real-world environments.

7.1 Discussion

In the proposed framework there are some considerations that needs to be made. In particular, a couple of limitations needs to be taken under consideration.

- **NN performances**: given the fact that the point cloud data is first processed based on the semantic information, the NN accuracy will in part limit the overall accuracy in identifying a safe landing zone.

- **Mapping**: since it has been adopted a framework that makes the drone able to map the entire overflown environment, namely it does not make use of fixed-size, robot-centric maps, we need to take into account the possibility of drift over large scale environments.
- **VIO**: the odometry is obtained through a tracking camera, thus, we are not fully independent with our software.

This is why it should be considered to enhance the NN performances by adopting more sophisticated architectures. In such a way the accuracy in identifying a safe landing zone could be improved. However, optimization will be needed to obtain similar inference times, given the higher model complexity.
For example, we can see that the DeepLabV3+ [47] network obtains better results, compared to the BiSeNetV2 model, as shown in Fig. 7.1 and Tab. 7.1.

Class	IoU (%)	Acc (%)
buildings	85.91	93.13
landing_site	84.62	91.08
water	76.34	88.27
sky	93.47	96.44
people_animals	76.77	86.71
vehicles	75.54	82.73
background	51.76	70.05
trees_poles	70.87	83.23
light	52.3	62.13
man_made_objects	74.44	84.66
nature_objects	29.68	43.06

Table 7.1: Intersection over Union (IoU) and Accuracy (Acc) values for 11 classes segmentation - DeepLabV3+

In particular, after the fine-tuning the results are given by Tab. 7.2.

Class	IoU (%)	Acc (%)
unsafe_landing_site	96.44	98.58
safe_landing_site	85.06	90.31

Table 7.2: IoU and Acc values for 2 classes segmentation - DeepLabV3+

We can clearly see that the mean accuracy (mAcc) and the mean Intersection over Union (mIoU), respectively of 94.45% and 90.75%, are higher than the ones obtained with the BiSeNetV2

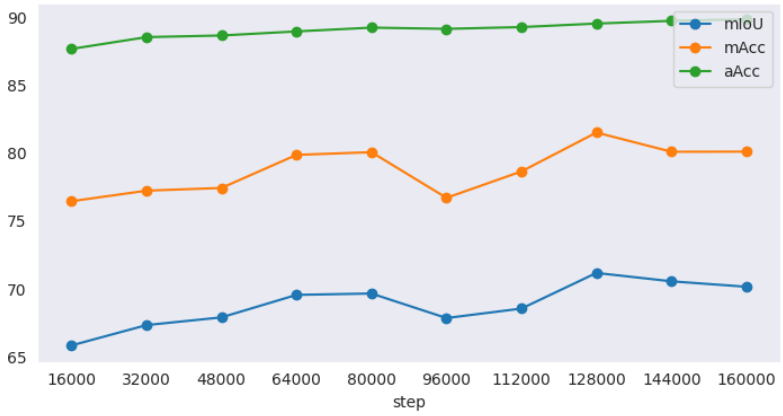


Figure 7.1: mIoU, aAcc & mAcc of 11 classes training - DeepLabV3+

model.

However, as pinpointed before, this network requires much more resources. On the Nvidia Jetson its inference time is less than 1 Hz. So, in order to obtain real-time performances as the BiSeNetV2 model, more sophisticated optimization techniques are required.

Moreover, we aim to enhance the point cloud processing and the mapping procedure. In particular, the current setup does not incorporate Simultaneous Localization and Mapping (SLAM) capabilities [48], therefore it is prone to drift for flights of extended duration. Simultaneous Localization and Mapping addresses the challenge of accurately tracking the position and orientation of a drone in real-time while simultaneously building a map of its environment. This is particularly valuable since it compensate for the drift that drones may experience during extended-duration flights, given that SLAM continuously corrects these errors, allowing the drone to maintain accurate localization throughout the entire flight.

Finally, our implementation make use of odometry information obtained from a tracking camera, thus reducing the versatility of our approach. However a solution has already been taken under consideration. In fact, a tuned version of OpenVINS [49] is being developed, to work also with the RealSense T265 tracking camera. In this way the odometry computations

would be done on-board, and not by the camera itself.

7.2 Future works

In the future, we are thinking of using a monocular camera and learning-based, efficient depth estimation techniques for perceiving the depth of the environment. Several works have already tried to address this problem, such as [50, 51, 52]. By employing monocular cameras for depth estimation we would therefore avoid the use of stereo cameras, thus reducing the cost and the weight of the drone.

Additionally, we would like to develop exploration strategies to autonomously scan the environment. This process can still leverage the multiple cost metrics employed in this work. By prioritizing areas with lower costs, we can autonomously guide the drone to directly explore areas that appears to be safer, making our drone entirely self-sufficient.

On top of that, we would like to improve our framework by solving the limitations analyzed in the "Discussion" chapter (7.1). By improving the NN performances and the mapping procedure we would further improve the method accuracy in identifying a safe landing zone.

References

[1] A. A. Laghari, A. K. Jumani, R. A. Laghari, and H. Nawaz, "Unmanned aerial vehicles: A review," *Cognitive Robotics*, vol. 3, pp. 8–22, 2023.

[2] A. Otto, N. Agatz, J. Campbell, B. Golden, and E. Pesch, "Optimization approaches for civil applications of unmanned aerial vehicles (uavs) or aerial drones: A survey," *Networks*, vol. 72, no. 4, pp. 411–458, 2018.

[3] G. Li, X. Liu, and G. Loianno, "Safety-aware human-robot collaborative transportation and manipulation with multiple mavs," 2023.

[4] M. Morita, H. Kinjo, S. Sato, T. Sulyyon, and T. Anezaki, "Autonomous flight drone for infrastructure (transmission line) inspection (3)," in *International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, 2017, pp. 198–201.

[5] N. Gageik, P. Benz, and S. Montenegro, "Obstacle detection and collision avoidance for a uav with complementary low-cost sensors," *IEEE Access*, vol. 3, pp. 599–609, 2015.

[6] A. Devo, J. Mao, G. Costante, and G. Loianno, "Autonomous single-image drone exploration with deep reinforcement learning and mixed reality," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5031–5038, 2022.

[7] C. S. Tang and L. P. Veelenturf, "The strategic role of logistics in the industry 4.0 era," *Transportation Research Part E: Logistics and Transportation Review*, vol. 129, pp. 1–11, 2019.

[8] H.-W. Choi, H.-J. Kim, S.-K. Kim, and W. S. Na, "An overview of drone applications in the construction industry," *Drones*, vol. 7, no. 8, 2023.

[9] C. Symeonidis, E. Kakaletsis, I. Mademlis, N. Nikolaidis, A. Tefas, and I. Pitas, "Vision-based uav safe landing exploiting lightweight deep neural networks," in *Proceedings of the 2021 4th International Conference on Image and Graphics Processing*, ser. ICIGP '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 13–19.

[10] P. Schoppmann, P. F. Proença, J. Delaune, M. Pantic, T. Hinzmann, L. Matthies, R. Siegwart, and R. Brockers, "Multi-resolution elevation mapping and safe landing site detection with applications to planetary rotorcraft," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 1990–1997.

[11] R. Moghe and R. Zanetti, "A deep learning approach to hazard detection for autonomous lunar landing," *The Journal of the Astronautical Sciences*, vol. 67, pp. 1–20, 10 2020.

[12] Agile robotics and perception lab (arpl). [Online]. Available: https://wp.nyu.edu/arpl/

[13] A. Johnson, J. Montgomery, and L. Matthies, "Vision guided landing of an autonomous helicopter in hazardous terrain," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2005, pp. 3966–3971.

[14] 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," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 111–118.

[15] M. Pizzoli, C. Forster, and D. Scaramuzza, "Remode: Probabilistic, monocular dense reconstruction in real time," 05 2014.

[16] K. Tomita, K. A. Skinner, and K. Ho, "Bayesian deep learning for segmentation for autonomous safe planetary landing," *Journal of Spacecraft and Rockets*, vol. 59, no. 6, pp. 1800–1808, 2022.

[17] X. Guo, S. Denman, C. Fookes, L. Mejias, and S. Sridharan, "Automatic uav forced landing site detection using machine learning," in *International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, 2014, pp. 1–7.

[18] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source robot operating system," 2009.

[19] R. R. Murphy, *Introduction to AI Robotics*, 1st ed. Cambridge, MA, USA: MIT Press, 2000.

[20] T. Lee, M. Leok, and N. Mcclamroch, "Geometric tracking control of a quadrotor uav on se(3)," pp. 5420 – 5425, 01 2011.

[21] J. Maye, P. Furgale, and R. Siegwart, "Self-supervised calibration for robotic systems," 06 2013.

[22] P. Furgale, H. Sommer, J. Maye, J. Rehder, T. Schneider, and L. Oth. Kalibr: Multiple camera calibration tool. [Online]. Available: https://github.com/ethz-asl/kalibr/wiki/multiple-camera-calibration

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[24] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," vol. 9351, 10 2015, pp. 234–241.

[25] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[26] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, 06 2016.

[27] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Doll'a r, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[28] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[29] C. Yu, C. Gao, J. Wang, G. Yu, C. Shen, and N. Sang, "Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation," *International Journal of Computer Vision (IJCV)*, vol. 129, pp. 3051–3068, 2021.

[30] Nvidia tensorrt documentation. [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt/

[31] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[32] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, p. 381–395, jun 1981.

[33] J. Park, Y. Kim, and S. Kim, "Landing site searching and selection algorithm development using vision system and its application to quadrotor," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 2, pp. 488–503, 2015.

[34] 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, April 2017.

[35] J. Mao, S. Nogar, C. M. Kroninger, and G. Loianno, "Robust active visual perching with quadrotors on inclined surfaces," *IEEE Transactions on Robotics*, vol. 39, no. 3, pp. 1836–1852, 2023.

[36] Open-source, "QGroundControl: Ground Control Station," https://http://qgroundcontrol.com/.

[37] B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso, and A. Torralba, "Semantic understanding of scenes through the ade20k dataset," *International Journal of Computer Vision (IJCV)*, vol. 127, pp. 302–321, 2019.

[38] R. Mottaghi, X. Chen, X. Liu, N.-G. Cho, S.-W. Lee, S. Fidler, R. Urtasun, and A. Yuille, "The role of context for object detection and semantic segmentation in the wild," 2014.

[39] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," 2016.

[40] M. Contributors, "MMSegmentation: Openmmlab semantic segmentation toolbox and benchmark," https://github.com/open-mmlab/mmsegmentation, 2020.

[41] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[42] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," *30th International Conference on Machine Learning, ICML 2013*, pp. 1139–1147, 01 2013.

[43] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org.

[44] S. Ruder, "An overview of gradient descent optimization algorithms," 2017.

[45] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," 2023.

[46] R. Zhang, Z. Jiang, Z. Guo, S. Yan, J. Pan, H. Dong, P. Gao, and H. Li, "Personalize segment anything model with one shot," *arXiv preprint arXiv:2305.03048*, 2023.

[47] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer International Publishing, 2018, pp. 833–851.

[48] S. Thrun and J. J. Leonard, *Simultaneous Localization and Mapping*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 871–889. [Online]. Available: https://doi.org/10.1007/978-3-540-30301-5_38

[49] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "OpenVINS: A research platform for visual-inertial estimation," in *Proc. of the IEEE International Conference on Robotics and Automation*, Paris, France, 2020.

[50] A. Masoumian, H. A. Rashwan, J. Cristiano, M. S. Asif, and D. Puig, "Monocular depth estimation using deep learning: A review," *Sensors*, vol. 22, no. 14, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/14/5353

[51] M. Poddar, A. Mishra, M. Kewlani, and H. Pei, "Self-supervised learning based depth estimation from monocular images," 2023.

[52] Rajanna, R. Agarwal, and J. B. Simha, "A monocular camera depth estimate approximation using deep learning," in *2022 International Conference on Futuristic Technologies (INCOFT)*, 2022, pp. 1–4.

Acknowledgments

I would like to express my sincere gratitude to Professor Ruggero Carli (carlirug@dei.unipd.it) and Giuseppe Loianno (loiannog@nyu.edu) for providing me with the incredible opportunity to be part of this remarkable project.

In particular, I am immensely thankful to Professor Giuseppe, who generously welcomed me into his laboratory and integrated me into the thrilling and enriching environment of the ARPL.

I extend my thanks to the dedicated members of the Agile Robotics and Perception Lab, with a special mention to Nishanth Bobbili (nb3553@nyu.edu) and Yang Zhou (yangzhou@nyu.edu), who actively contributed throughout the entire project, as well as to Luca Morando and Pratyaksh P. Rao for their valuable advice. Most importantly, my appreciation goes beyond their professional support, and extends to the warmth of their friendship which transcended the boundaries of the laboratory.

Nonetheless, I wish to express my deep appreciation to my parents, my girlfriend, my friends, and all those who supported and stood by me during my time in New York. To those who were a constant source of strength, who brought smiles to my face even on the toughest days, and who went out of their way to make me feel at home on a different continent.

To everyone who believed in me and afforded me the incredible opportunity to embark on this journey, I say thank you.

To my dearest mom, dad, girlfriend Teresa, and my friend Ale, your unwavering support means the world to me.

I am profoundly thankful, thank you, all!

GRAZIE !