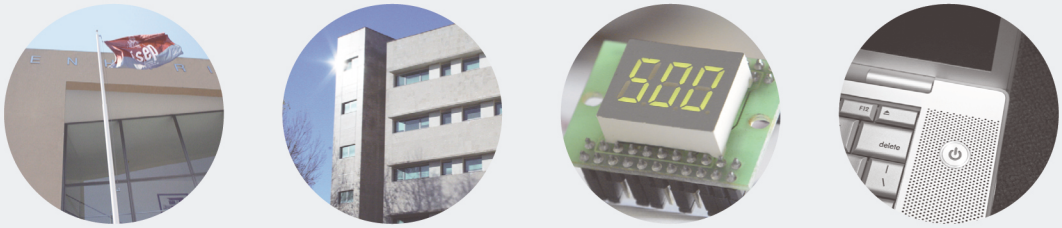
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



Emergency Landing Spot Detection for Unmanned Aerial Vehicle

GABRIEL DA SILVA MARTINS LOUREIRO
novembro de 2020

POLITÉCNICO
DO PORTO

Emergency Landing Spot Detection for Unmanned Aerial Vehicle

Master in Electrical and Computer Engineering
Branch of Autonomous Systems

Gabriel da Silva Martins Loureiro
Nº 1170081

Supervisor
André Miguel Pinheiro Dias

Academic Year: 2019-2020

Instituto Superior de Engenharia do Porto
Departamento de Engenharia Eletrotécnica
Rua Dr. António Bernardino de Almeida, 431, 4200-072 Porto

Agradecimentos

Primeiro, gostaria de agradecer ao meu orientador, Eng. André Dias, pelo desafio proposto, como também pelo auxílio durante o desenvolvimento desta dissertação.

Gostaria de agradecer também os meus colegas de mestrado que fizeram parte deste percurso.

Aos meus amigos de longa data pela amizade durante esses anos. Mesmo que indiretamente, tiveram grande contribuição para a conclusão deste mestrado.

Ao meu irmão, Bruno Loureiro, pelo carinho e preocupação demonstrado ao longo desses anos. O teu apoio foi fundamental para o fim desta etapa.

Por fim, um agradecimento especial aos meus pais, Fernando e Nazaré. Obrigado pelo vosso carinho, vossa paciência e cuidado durante toda a minha vida. Serei eternamente grato por todos esforços e desafios superados para poderem me proporcionar todas estas oportunidades. Espero poder retribuir tudo o que já fizeram por mim.

Muito obrigado.

Gabriel da Silva Martins Loureiro

Abstract

The use and research of Unmanned Aerial Vehicle (UAV) have been increasing over the years due to the applicability in several operations such as search and rescue, delivery, surveillance and others. Considering the increased presence of these vehicles in the airspace, it becomes necessary to reflect on the safety issues or failures that UAV may have and what is the appropriate action to take. Furthermore, in many missions the vehicle will not return to its original location and, in case of fail to achieve the landing spot, need to have onboard capability to estimate the best spot to safely land.

The vehicles are susceptible to external disturbance or electromechanical malfunction. In this emergencies scenarios, UAVs must safely land in a way that will minimize damage to the robot and will not cause any human injury.

The suitability of a landing site depends on two main factors: the distance of the aircraft to the landing site and the ground conditions. The ground conditions are all the factors that are relevant when the aircraft is in contact with the ground, such as slope, roughness and presence of obstacles.

This dissertation addresses the scenario of finding a safe landing spot during operation. Therefore, the algorithm must be able to classify the incoming data and store the location of suitable areas. Specifically, by processing Light Detection and Ranging (LiDAR) data to identify potential landing zones and evaluating the detected spots continuously given certain conditions.

In this dissertation, it was developed a method that analyses geometric features on point cloud data and detects potential good spots. The algorithm uses the Principal Component Analysis (PCA) to find planes in point clouds clusters. The planes that have slope less than a threshold are considered potential landing spots. These spots are then evaluated regarding ground and vehicles conditions such as the distance to the UAV, presence of obstacles, roughness of the area, slope of the spot. The output of the algorithm is the optimum spot to land and can vary during operation.

Keywords: unmanned aerial vehicle, LiDAR, landing spots detection, emergency landing, point cloud

Resumo

O uso e pesquisa de veículos aéreos não tripulados (VANT) têm aumentado ao longo dos anos devido à aplicabilidade em diversas operações, como busca e salvamento, entrega, vigilância e outras. Considerando a crescente presença desses veículos no espaço aéreo, torna-se necessário refletir sobre os problemas ou falhas de segurança que o veículo pode ter e qual é a ação apropriada a ser tomada. Além disso, em muitas missões, o veículo não retornará ao seu local original e, caso não seja possível alcançar a zona de aterragem, precisa ter a capacidade de estimar o melhor ponto para aterrar em segurança.

Os veículos são suscetíveis a perturbações externas ou mau funcionamento eletromecânico. Nesses cenários de emergência, os UAVs precisam aterrar com segurança de forma a minimizar os danos ao robô e não causar ferimentos em pessoas.

A adequação de um local de pouso depende de dois fatores principais: a distância do veículo aéreo ao local de pouso e as condições do solo. As condições do solo são todos os fatores relevantes quando a aeronave está em contacto com o solo, como declividade, rugosidade e presença de obstáculos.

Esta dissertação aborda o cenário de encontrar um local de pouso seguro durante a operação. Portanto, o algoritmo deve ser capaz de classificar os dados recebidos e armazenar a localização de áreas adequadas. Especificamente, processando dados de LiDAR para identificar possíveis zonas de aterragem e avaliando os pontos detetados continuamente, dadas determinadas condições.

Nesta dissertação, foi desenvolvido um método que analisa características geométricas em nuvem de pontos e deteta possíveis bons locais de aterragem. O algoritmo usa a Análise de Componente Principal (PCA) para encontrar planos em *clusters* de nuvens de pontos. Os planos com inclinação menor que um limite são considerados possíveis pontos de aterragem. Esses pontos são então avaliados quanto às condições do solo e dos veículos, como a distância ao UAV, presença de obstáculos, rugosidade da área, inclinação do ponto. A saída do algoritmo é o local ideal para aterrar e pode variar durante a operação.

Palavras-chave: veículos aéreos não tripulados, LiDAR, deteção de pontos

de aterragem, aterragem de emergência, nuvem de pontos

Contents

List of Figures

List of Tables

List of Algorithms and Program code

List of Acronyms

2-D Two-Dimensional.

2½-D Two-and-a-Half-Dimensional.

3-D Three-Dimensional.

CRAS Centre for Robotics and Autonomous Systems.

ECEF Earth-centered Earth-fixed.

EKF Extended Kalman Filter.

ENU East-North-Up.

FLU Forward-Left-Up.

FoV Field of View.

FRD Forward-Right-Down.

GNSS Global Navigation Satellite System.

GPS Global Positioning System.

IMU Inertial Measurement Unit.

INESC TEC Institute for Systems and Computer Engineering, Technology and Science.

INS Inertial Navigation System.

ISEP Engineering School of Porto Polytechnic.

KF Kalman Filter.

LiDAR Light Detection And Ranging.

LMS Least Median Square.

LSA Autonomous Systems Laboratory.

MEMS Microelectromechanical mirrors.

MORSE Modular Open Robots Simulation Engine.

NED North-East-Down.

NMEA National Marine Electronics Association.

P2P Peer-to-Peer.

PCA Principal Component Analysis.

PCL Point Cloud Library.

PPS Pulse Per Second.

RADAR Radio Detection And Ranging.

ROS Robot Operating System.

RTK Real Time Kinematic.

ToF Time of Flight.

UAV Unmanned Aerial Vehicle.

UTC Coordinated Universal Time.

V-REP Virtual Robot Experimentation Platform.

WGS World Geodetic System.

Chapter 1

Introduction

In recent years, the study and use of Unmanned Aerial Vehicles (UAVs), commonly known as drones, has been the focus of great interest due to the large set of research topics, such as hardware development, human-system interaction, obstacle detection, collision avoidance and considerably more [1]. These vehicles consists of a aircraft that can be remotely operated by a human operator or execute a mission autonomously [2]. In the latter case, the degree of autonomy and the mission which they are capable to achieve depend on the sensors used.

In terms of categorisation, there are several ways which UAVs can be classified: aerodynamics, landing, weight and range [3]. Generally, the classification regarding their aerodynamics is more used. For instance, there are fixed-wing, flying-wings and flapping-wing vehicles in which their wings are the main factor to generate lift. On the other hand, there are other types such as helicopters, quadcopter that uses multiple rotors to produce forceful thrust.

Given the numerous UAVs with multiple sensors, motors and design configurations, they present applicability in several kinds of operations, such as search and rescue operations[8, 9], delivery [10], surveillance [11], inspection and interaction with the environment [12, 13]. Considering the applications scenarios, there are missions where the UAV must fly in civilian airspace, i.e., they must fly over populated areas. However, they are susceptible to external disturbance or electromechanical malfunction. As a matter of fact, there are different failures scenarios that could impact the operation thus leading to an emergency landing:

- Global Positioning System (GPS) failures. In general, UAVs use a GPS message to navigate. Although there a several sensors that could aid the navigation, in the occurrence of loss of GPS signal the vehicle may need to land.

(a) Fixed-wing UAV (FALCOS) [4]

(b) Multirotor (OTUS) [5]

(c) Flapping-wing UAV (BionicOpter) [6]

(d) Helicopter UAV (Alpha 800) [7]

Figure 1.1: Examples of different UAVs.

- Loss of communication. In the event of loss of communication between the UAV and a base station, one possible action is to realize a emergency landing.
- Battery failure. If a battery failure is detected, the vehicle may not be able to continue the operation and as a result an emergency landing is necessary.
- Software and hardware errors. During operation, a mechanical fault such as broken propellers could limit the vehicle dynamic. Besides this, software crashes could occur. In this scenario, the UAV may need to land.
- Environment factors. Bad weather conditions such as strong winds and rain make the vehicle unable to carry out the mission, forcing it to land.

In this type of situations, UAVs must safely land in a way that will minimize damage to itself and won't cause any injury to humans.

Consequently, detecting a reliable landing spot is essential to safe operation. In order to determine a landing spot, a set of conditions must be considered when analysing the sensor data. These conditions are typically restraints on the surface. The reliability of a safe landing site depends on several factors, such as the distance of the aircraft to the landing site and the ground conditions. The

ground conditions are all the factors that are relevant when the aircraft is in contact with the ground. The conditions considered in this project are:

- The slope of the plane. The surface the UAV will land must have a slope smaller than a threshold;
- The roughness of the area. There are cases when the vehicle will land in a surface with vegetation or other obstacles. In this case, the height of these obstacle must not be larger than a maximum value pre-defined before the mission;
- The size of the spot. The landing spot must be large enough for the UAV;
- Presence of obstacles. The approximation to landing spot depends on the obstacles around the area. Given that, the presence of obstacles must be taken into account when evaluating the spot;
- Distance to vehicle. The UAV must be able to reach the desired spot with the remaining battery power.

Since one of the factors is the distance of the aircraft to the landing site, the suitability of a landing spot varies during the mission. Figure 1.2 illustrates the conceptual approach for emergency landing spot detection with a UAV in real time.

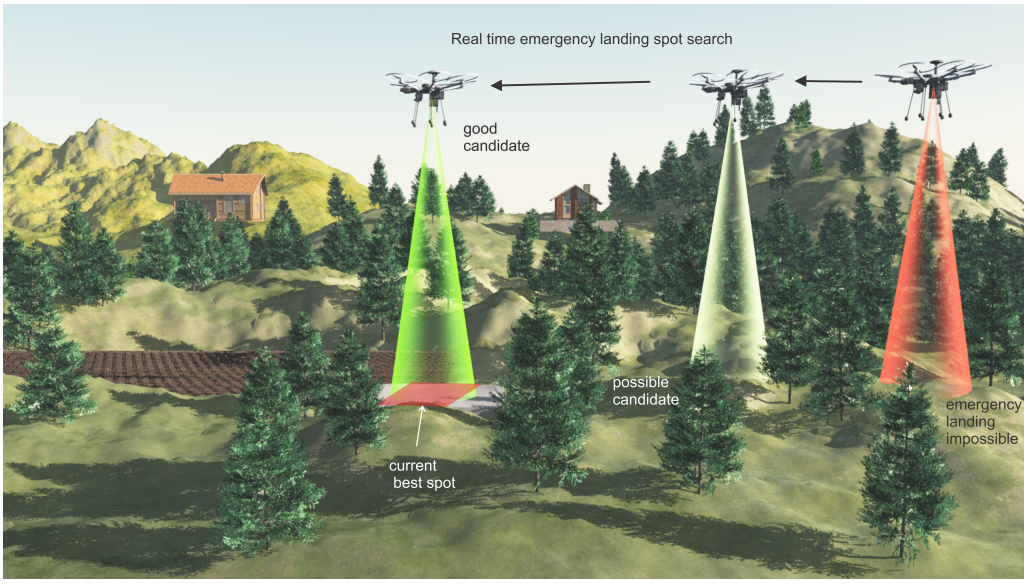


Figure 1.2: Conceptual approach for Emergency landing spot detection with a UAV.

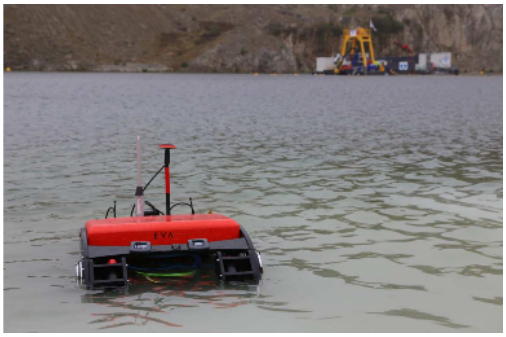
In order to increase the reliability of an UAV operation, this dissertation addresses the development of an algorithm for detect, store and select emergency

landing spots in operation time using Light Detection And Ranging (LiDAR) (LiDAR) data. The main focus is to develop an algorithm that will evaluate continuously the geometry of the terrain, based on LiDAR data, and establish a list of possible landing spots. In addition, since the probability of a spot being the best landing zone should depend on several factors, a re-evaluation must be done while the UAV is operating.

1.1 Motivation

The Centre for Robotics and Autonomous Systems (CRAS)[1], from the Institute for Systems and Computer Engineering, Technology and Science (INESC TEC)[2], and the Autonomous Systems Laboratory (LSA), from the Engineering School of Porto Polytechnic (ISEP), have been developing numerous projects in the robotics field over the past years. The result of these projects are the development of different robotic systems for several fields of research: underwater (figures 1.3a and 1.3b), water surface (figure 1.3c), land and air (1.3d).

(a) TURTLE II [14]



(b) EVA [15]

(c) ROAZ II [16]

(d) STORK I [17]

Figure 1.3: Robots developed by CRAS and LSA.

[1] https://www.inesctec.pt/en/centres/cras
[2] https://www.inesctec.pt/en

Their background of UAVs involves the fixed-wing vehicle FALCOS and, recently, multirotor robots, such as OTUS, STORK and GRIFO-X. These robots are related to multiple projects, with ROSM[3], SpilLess[4], Sunny[5] and MineHeritage[6] being examples of those multiple projects. Regarding the MineHeritage project, for instance, the UAV obtains high resolution images and maps the external region with a static Three-Dimensional (3-D) LiDAR.

Furthermore, they have won the *Grand Challenge* of two international competitions of search and rescue missions using the UAVs OTUS and STORK: the euRathlon[7] [8] and ERL Emergency Robots[8], in 2015 and 2017, respectively.

Considering these projects and vehicles, we note the different applications in which the robots can be used. However, these missions have in common the need to occasionally land in a different position than take-off origin. Besides emergency scenarios, detecting a landing location different from the original position allows optimization of missions, reducing battery consumption, for example.

1.2 Objectives

This dissertation addresses the detection of landing spots for unmanned aerial vehicles in operation time, using a LiDAR sensor. For the purpose of achieving the main objective of this dissertation, there are several intermediate objectives that need to be fulfilled:

- Study of existing works related to the detection of landing spots and their different approaches.;
- Analysis of the requirements to determine a safe landing spot. The suitability of a landing site depends on several conditions that need to be addressed in the project;
- Creation of a data processing pipeline. The pipeline is based on the acquired knowledge from the previous objectives;
- Development of a simulation environment able to provide the required tools to validate the algorithm for emergency landing spot;
- Development of a spot detection algorithm based on ground conditions given a mathematical methodology;

[3] `https://www.inesctec.pt/pt/projetos/rosm`
[4] `https://www.inesctec.pt/pt/projetos/spilless`
[5] `https://www.inesctec.pt/pt/projetos/sunny`
[6] https://mineheritage-project.eu/
[7] `https://www.eurathlon.eu/`
[8] `https://www.eu-robotics.net/robotics_league/erl-emergency/about/index.html`

- Classification of the suitable spots given different parameters;
- Performance evaluation in different environments in order to validate the developed algorithm.

1.3 Structure

This dissertation is divided in eight chapters. The next chapter presents a preliminary study of the related works to the topic of this dissertation. The chapter ends with the analyses of the different works, presenting their advantages and pointing the drawbacks. Besides this, it also shows the common assumptions made.

Chapter 3 introduces the theory in which parts of the algorithm are based. Moreover, it also presents the technologies used in the developed method. Next, chapter 4 shows the high-level software and hardware architecture, detailing their components.

Chapter 5 introduces the developed algorithm and its requirements. Beside this, the algorithms sequence of data processing blocks is displayed. Chapter 6 presents the specifications for the vehicle. In addition, it shows the libraries of algorithms used and the developed communication packages.

In chapter 7, the results are presented as well as the analyses and performance of the algorithm.

Finally, in chapter 8 we discuss the conclusions of the method for detecting safe landing spots. Here we also present suggestions or directions for further work.

Chapter 2

Related Work

In this chapter is presented the state-of-the-art regarding the topic of emergency and safe landing. Several approaches related to the detection of landing spots using an UAV is exposed. The chapter ends with a brief analysis of the presented works and comparison with this dissertation.

The survey present in this chapter resulted in the submission of the paper *Survey of Approaches for Emergency Landing Spot Detection with Unmanned Aerial Vehicles* [18] to the 23rd edition of International Conference series on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR 2020), in Moscow, Russia.

2.1 Emergency Landing Spot Detection

The main problem to be addressed is the integration in an UAV the ability to detect safe landing zones. Hence, it is necessary to process different types of data from several sensors so that it is possible to obtain information about the zone where it is intended to land. Different sensors provide distinct information regarding terrain, obstacles, presenting advantages and disadvantages in relation to each other. Thus, there are multiple approaches in the literature regarding safe landing, either for emergency situations or for other purposes.

2.1.1 LiDAR Landing Detection Systems

It is established in the literature that LiDAR sensors can provide accurate data about the environment and their scope of applicability ranges from terrestrial robots to aerial robots. The point clouds generated by the sensors are processed using geometric approaches, identifying terrain features such as slope

and roughness. This data can also be used to detect possible hazard to the aircraft.

Johnson et al. [19] developed in 2002 a system that uses LiDAR data to detect obstacles and safely land spacecrafts on Mars. Their system is based on geometric analysis of terrain characteristics. After generating a 2½-D grid map to represent the terrain, it computes a Least Median Square (LMS) estimation to fit a plane in a grid cell. Then, a Least Squares fit is applied in the remaining inlier points [20]. The incidence angle and the roughness are computed to obtain a landing cost map and select a safe landing site.

A similar approach is presented by Whalley et al. [21, 22] in 2009. The authors proposed a method that analyses a 3-D point cloud generated by a LiDAR to autonomously determine possible safe landing sites. The classification of the landing spots is realized by computing the plane of a set of points and applying geometrical constraints. These constraints are the slope of the computed plane that must be below a limit, the roughness and proximity of hazards and obstacles. The safe landing algorithm maps the 3-D point cloud data from each LiDAR scan on a grid. Next a square sliding window with the size of a landing area moves along the grid to compute the slope and roughness at each point. Then, the slope and roughness are analyzed to determine if the window is a feasible landing spot. After the entire grid is covered and the statistics are calculated, the optimal landing point is chosen. An optimum point has the minimum value of the sum of weighted roughness, slope, and distance.

An algorithm proposed by Chamberlain et al. [23], in 2011, allows a full-scale unmanned helicopter to fly through unmapped terrain and perform a safe landing without human control or input. Their approach uses a 3-D scanning LiDAR was developed that operates in two modes: a forward scan to detect obstacles and a downward scan to map the terrain (see figure 2.1). First, a map is

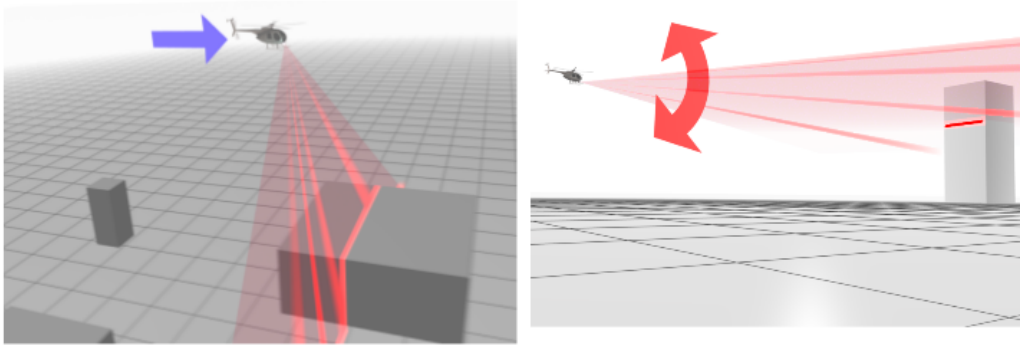


Figure 2.1: 3-D LiDAR downward and forward scan. (Adapted from [23])

built using the downward scan and then processed to remove rough and sloped areas. The next step consist of a fine evaluation in which a 3-D virtual model of the helicopter is placed on each cell of the map and a skid contact, wind

direction and presence of obstructions in adjacent area evaluations are realized. Then, their algorithm analyses the glide slope, i.e. the proper path of descent, to each possible landing spot and compares to the point cloud model. Finally, the landing spot is chosen from a optimization function based on the previous steps results.

This approach is extended in [24, 25, 26] by applying a plane fit and presenting experimental results in urban and natural environments. The algorithm applies two evaluation based on geometrical features: a coarse evaluation to identify potential landing sites and a fine evaluation to choose the optimum spot. First, a grip map is built given LiDAR points registered in global coordinates. Each cell contain points and statistical information, such as mean, minimum and maximum z-value, number of points. A first filtering step is also realized by rejecting cells in which the standard deviation in z is larger than a threshold. Then, a Least Square based on moment [27] is applied to fit a plane and compute the slope. In the fine evaluation, a 2-D Delaunay triangulation [28] of the potential landing sites data is built and used to determine the intersections with the landing skids and computing the roll and pitch of the aircraft. Finally, the volume between the triangulation and a 3-D model of the helicopter is calculated. The value of the volume is used to predict bad contact and as another measure to classify the landing site.

The problem of considering low vegetation as roughness is addressed in Maturana and Scherer [29]. The authors proposed a 3-D Convolutional Neural Networks [30] to detect landing zone for small UAVs. The algorithm incrementally creates a volumetric density map from the point cloud stream. The xy plane of the map is subdivided into non-overlapping tiles and for each tile the terrain surface height is estimated. These sub-volumes are the input of the trained neural network. Finally, the output of the algorithm is the reliability rating of the analyzed sub-volume regarding the feasibility to land. With this approach, they were able to detect small obstacles presented in vegetation.

In 2014, Whalley et al. [31] developed autonomous obstacle navigation and safe area selection on the U.S. Army Aeroflightdynamics Directorate RASCAL JUH-60A [32] helicopter. The safe landing algorithm maps the 3-D point cloud data from each LiDAR scan on a grid. The x and y values of each point falls into one cell while the height is distributed to the adjacent cells based on the relative horizontal position at the point within the cell. Next a square sliding window (figure 2.3) with the size of a landing area moves along the grid to compute the slope and roughness at each point. Then, the slope and roughness are analyzed to determine if the window is a feasible landing spot. After the entire grid is covered and the statistics are calculated, the optimal landing point is chosen. A optimum point has the minimum of the sum of weighted roughness, slope and distance.

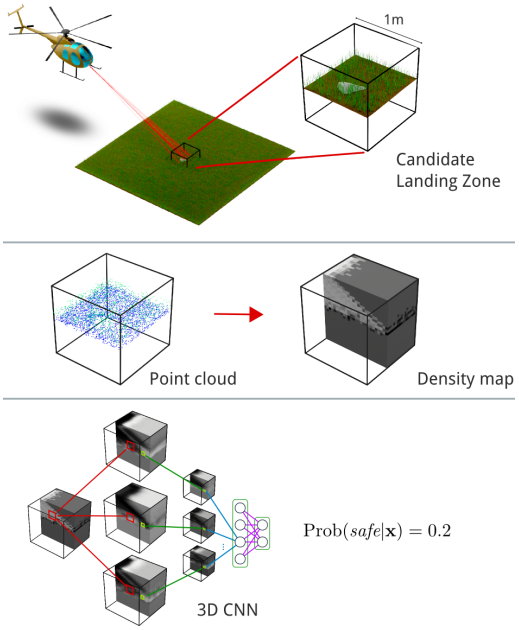


Figure 2.2: 3-D Convolutional Neural Network to detect safe landing spots [29].

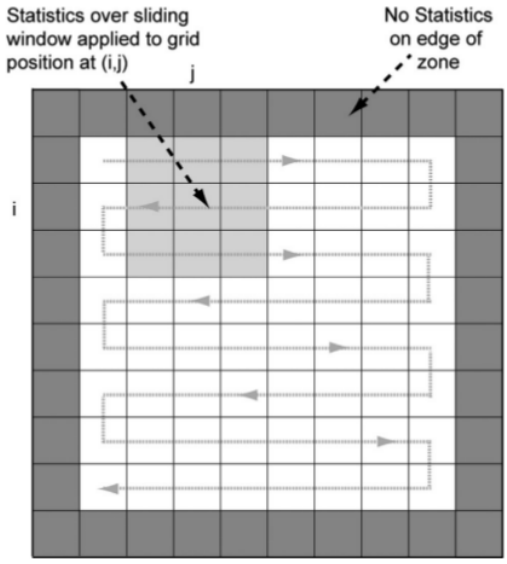


Figure 2.3: Sliding window applied over a grid [31].

The work of Lorenzo et al. [33] in 2017 presents an algorithm for landing sites detection on manycore systems using parallel processing. Their proposal is tested on several set of data with different characteristics. First, the 3-D points of the point cloud is sorted to their x coordinates and then to their y coordinates. Second, a read-only octree is built to hasten the neighbourhood search [34]. The next step of the algorithm is to fit a plane in a neighbourhood larger enough for the aircraft using Principal Component Analysis (PCA) [35]. The normals to all points are computed in parallel. Finally, a quality rate is assigned to the plane regarding the slope, roughness, skid landing requirements and presence of obstructions.

2.1.2 Vision Landing Detection Systems

In addition to the use of LiDAR, the vision system has been the most popular approach to detect and assess a landing spot. There are multiple strategies to land a UAV using vision-based algorithms. The focus of this survey is the works related to landing on an unknown or partially known site as it is the most likely to happen in an emergency scenario. However, other strategies for known environments are landing on a marker, runway or a moving platform[36]. The cameras have the advantage of being inexpensive and lighter compared to LiDARs. In Garcia-Pardo et al.[37], a safe landing site is an area that is big enough for the helicopter to land and is clear of obstacles. Their algorithm assumes that contrast in an image is higher near the obstacles. Therefore, it searches circular areas in an image in which pixels have a level of contrast below a maximum value.

The work developed by Bosch et al.[38] presents a method to detect landing areas autonomously using monocular images. The algorithm applies a homography estimation process to select points that lie in a plane. Then, a dense correlation technique is used to distinguish planar surfaces from non-planar areas. Finally, the information is saved in a stochastic 2-D grid in which each cell has the probability of being planar. However, as the grid has the probability of the region being flat, it is not capable of being used for other functions, such as obstacle avoidance.

Another technique used to detect landing spots is machine vision. In Fitzgerald et al.[39, 40] and Mejias et al. [41], a machine approach is used to locate safe landing areas for UAV forced landings given aerial imagery. The first step is a preliminary site selection in which a Canny Edge Detector [42] and line-expansion algorithm is applied to generate two binary images. These images are fused and processed to generate a preliminary map where the safe and unsafe areas are labeled (figure 2.4). The second step consists of classifying the type of surface and build a coarse slope map. Finally, an optimum landing map is built by fusing the previous maps using fuzzy linguistic as decision method.

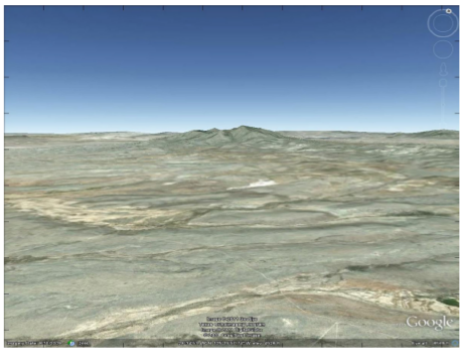
Figure 2.4: Preliminary Map example labeled regarding landing safety (Adapted from [41]).

The work is later extended in 2015 by Warren et al. [43]. The 2-D points in the preliminary maps are projected into a 3-D world model using the camera pose and intrinsic model, accounting the terrain ruggedness. Then, a 3-D reconstructions using Structure-from-Motion is realized. Finally, the algorithm applies a 3-D surface analysis regarding size, local obstacles, terrain smoothness to select the landing spot.

Eendebak et al. proposed in 2013 an algorithm for emergency landing selection in real-time operations. The method is able to detect objects in presence of camera movement. The authors used Background Estimation [45] on stabilized video and then compared the difference between the current frame of the video and the background estimate to detect moving objects and structures. A binary image is generated indicating obstacles and a distance map to all detected obstacles is built. The maximum distance in the map is computed and chosen as the landing site.

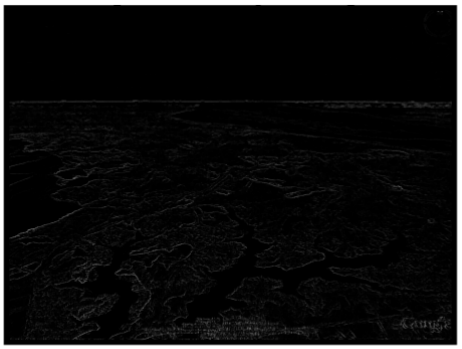
In 2013, Shen et al. [46] used aircraft-mounted cameras to acquire visual information and detect emergency landed sites (see figure 2.5). The method was validated in offline experiments. First, the ground in the image is identified by applying a hierarchical elastic horizon detection algorithm. After the horizon is detected, a Canny Edge Detector is applied to assess the roughness of the area. Furthermore, the terrain image is clustered in several clusters using the K-mean clustering method. Then, the clusters are processed and analyzed to detect the potential landing spot.

Forster et al. [47] proposed an elevation map-based technique to landing spot detection for micro aerial vehicles using a monocular camera. The images from the camera and the vehicle's pose are combined to build a depth map. Moreover, an elevation map is generated and updated given the resulting maps. In the selection step, the authors considered a landing spot as a flat surface with

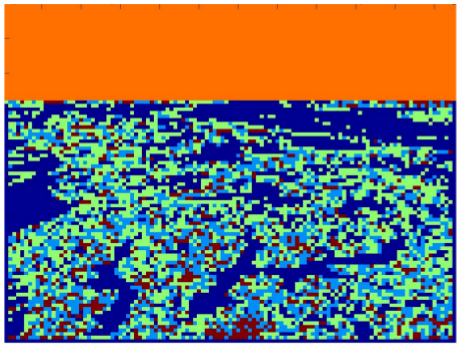


(a) Sampled image.
"

(b) Horizon detection.



(c) Edge detection.



(d) Clustering result given edge images.

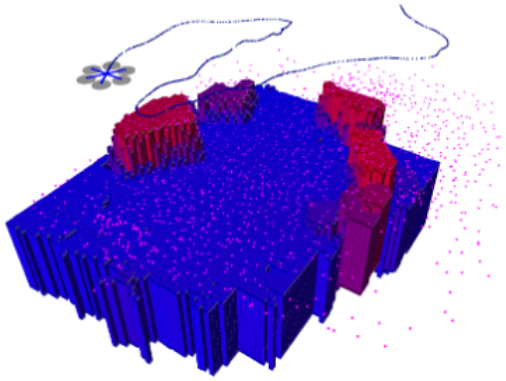
Figure 2.5: Proposed algorithm using Canny edge detector and clustering algorithm (Adapted from [46]).

a radius that depends on the size of the vehicle. Figure 2.6 shows the results of the paper. The first image corresponds to the scenario, the second is the elevation map built and, finally, the last image is a video excerpt of the landing procedure.

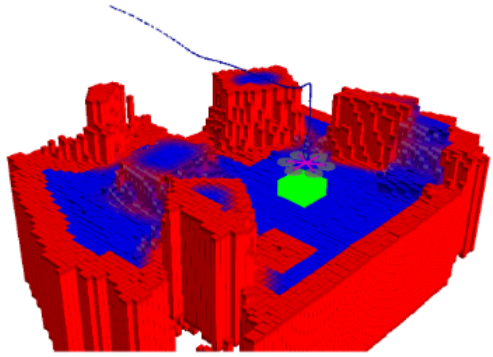
Hinzmann et al. [48] developed a vision-based algorithm to detect landing spots in unknown environments at real time. The proposed algorithm applies a segmentation technique to a camera image and classifies the resulting segmented regions as "grass" or "not grass". Then, 3-D reconstruction and consequently 2½-D elevation map algorithms are realized given the potential landing sites from the segmentation step. The optimum landing site is chosen given terrain characteristics, such as slope and roughness, Finally, a distance map is built.

Recently, Ayhan et al. [49] proposed a semi-automated emergency landing selection algorithm. The method is a offline approach to finding rectangular runway to serve as landing sites. The algorithm retrieves colour images from Google Static Maps application. The colour images are then transformed into

(a) Scenario



(b) Elevation map



(c) Landing procedure

Figure 2.6: Results of the paper [47] (Adapted from [47]).

grey-scale image and Canny edge detection to create a binary image that is finally dilated. Then, connected components are detected in order to select landing candidates. The candidates are rated considering several safety measures, such as surface type estimation, homogeneity, maximum elevation difference and other. At the end of the algorithm, the landing zones are ranked regarding their reachability.

2.1.3 Other approaches

Besides the previous works presented, other approaches have been proposed in which multiple sensors are used to solve the problem of terrain assessment and safety. Serrano [50] proposed a combination of Radio Detection And Ranging (RADAR), LiDAR and camera and a probabilistic framework to increase robustness of the selection of landing spots in landed space operation. The Least Median of Squares regression algorithm is applied to fit a plane using RADAR and LiDAR range data and determine slope and roughness. In addition, the authors used edge detection techniques to identify craters and rocks from camera imagery. Considering the terrain features, a Bayesian Networks [51] is used to assess ground safety. On their proposal, the potential landing quality is modeled using terrain, available fuel and region interest.

A similar sensors set is used by Howard and Seraji [52]. In this case, three hazards maps are built from RADAR, LiDAR and camera and used to extract measurements and terrain features. Each map is associated with a confidence variable that designate the sensor certainty. Then, the different maps are aligned and combined into a single fused map using fuzzy logic that represents terrain safety.

2.2 Discussion

After presenting the works related to the topic of detection, selection and classification of emergency and safe landing sites, a further study is realized in order to improve the development of the dissertation project. Some of the exposed approaches had already proposed solutions to overcome some limitations and drawbacks associated to the problem that this project seeks to solve. In this perspective, some premises made and proven in the current works can be either improved or applied to some extent in this dissertation.

Considering the different set of sensors, cameras are less expensive and lighter than the other sensor. However, the applicability of vision based algorithm depends on visibility conditions. Conversely, LiDAR sensors are able to overcome this hindrance and present higher accuracy. Nevertheless, LiDARs weight and processing power can be a restriction to some UAVs.

Specifically, the point cloud generated by LiDAR sensors must be spatially structured. Data structuring allows the storage and organization of information. In this context, this division leads to a more efficient data access and process. In addition, the method chosen for structuring cannot cause loss of information like the 2½-D grids and planes projected images.

Regarding the detection of landing spots, the basis of the detection algorithm is to identify a plane in the surface. Several techniques for real-time implementation of those algorithms search for geometrical features [19, 21, 23, 31, 53], such as slope and roughness, to classify suitable landing sites in a point cloud. These approaches does not present the need to train the algorithm when compared to approaches based on neural networks and machine vision [29, 39, 40, 41].

According with the exposed works, the detected spots must be classified in different levels considering the feasibility of landing. Hence, the final classification depends on terrain conditions, the presence of obstacles, distance to the UAV, remaining power and trajectory conditions. Furthermore, this rating should vary during the flight of the vehicle.

Chapter 3

Fundamentals

In this chapter it is presented a general outline of the important concepts for a better understanding of the developed algorithm. Consequently, information about mechanical functioning of sensors, mathematical models and computational concepts are described.

3.1 LiDAR

LiDAR is an active remote sensor used to compute ranges to objects by emitting laser pulses and measuring the reflection. The principle of operation is firing beams of light and computing the Time of Flight (ToF), i.e., the time it takes to the reflected pulse to be detected by the sensor, determining the distance to the target [54], as shown in figure 3.1. Currently, there are several categories

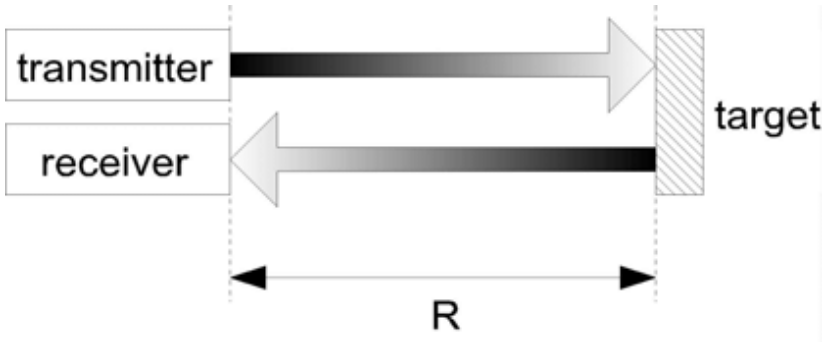


Figure 3.1: Principle of operation of a LiDAR sensor (Adapted from [55])

to the LiDAR technology, such as spinning, flash, phased-array and Microelectromechanical mirrors (MEMS) LiDARs.

3.1.1 Spinning LiDAR

In the spinning LiDARs (see Figure 3.2) a movable mirror is added to sensor in order to enable obtaining 3-D map of the environment. One main advantage of this class of LiDAR is the 360 degrees Field of View (FoV), allowing a complete measurement of the vehicle's surroundings [56]. Nevertheless, these sensors are, in general, mechanically fragile to shocks and vibrations. In addition, the service life of mechanical components is low, usually up to 2000 hours [57].

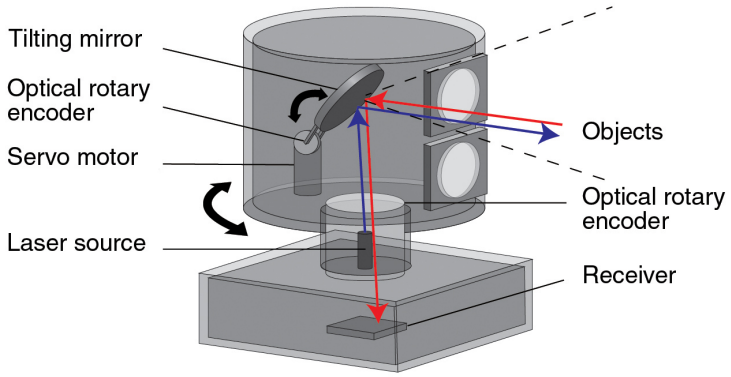


Figure 3.2: Spinning LiDAR diagram (Adapted from [58]).

Generally, the sensor returns data in Cartesian coordinates (x, y, z). These values are acquired by applying a conversion of the points in spherical to Cartesian coordinates [59]. Considering a point in the sensor's spherical coordinates system (r, ω, α), where r is the radial distance to the point, ω is the elevation angle and α is azimuth angle, the following equations transform the data to a sensor-centered Cartesian frame:

$$x = r \cdot \cos\omega \cdot \sin\alpha \tag{3.1}$$

$$y = r \cdot \cos\omega \cdot \cos\alpha \tag{3.2}$$

$$z = r \cdot \sin\omega \tag{3.3}$$

3.1.2 Solid-State LiDAR

The Solid-state LiDARs are being devised in order to find a solution to the problems of regarding the size, reliability and complexity of the mechanical based LiDARs. Their design differ from conventional LiDARs by not having movable pieces [60].

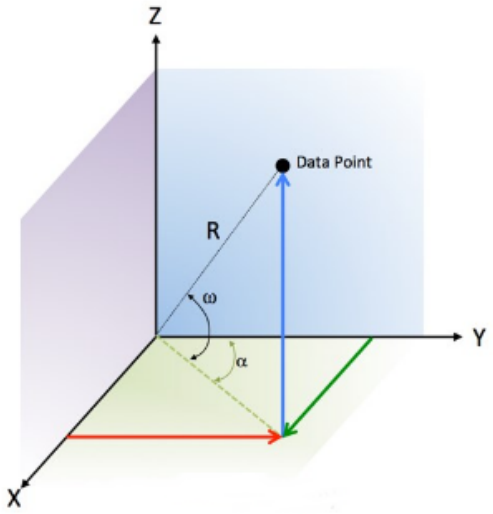


Figure 3.3: Spinning LiDAR spherical coordinates system (Adapted from [59]).

3.1.2.1 Flash-array LiDAR

In Flash LiDARS a beam of light is periodically fired towards a lens and are thereby spread it to the environment. In addition, an array of photo-sensors capture the reflected light. However, only a small portion of light is reflected, hence the need for sensitive receivers, which increases the cost of these sensors [57]. Its main advantage is the complete FoV acquirement at one moment, but despite that, the FoV coverage is typically very small.

3.1.2.2 Phase-array LiDAR

Phase-array LiDAR uses a set of optical antennas synced up in a specific way. It is possible to control the phase of the antennas, generating a radiation pattern that possesses a certain size and is pointed in a particular direction [57]. Figure 3.4 shows the operation of these sensors where the signal is "steered" to a specific direction.

Conversely, the beams produced tend to diverge more, making it hard to achieve a combination of long range, high scanning resolution, and wide FOV [56].

3.1.3 MEMS LiDAR

The MEMS LiDAR differs from spinning LiDAR by replacing the mechanical mirrors to MEMS micro-mirrors. This allows the sensor to direct the light beams during the transmission [61] (see figure 3.5).

They are generally smaller, less expensive and less sensible to vibrations when compared to spinning LiDARs due to reduction of their mechanical parts.

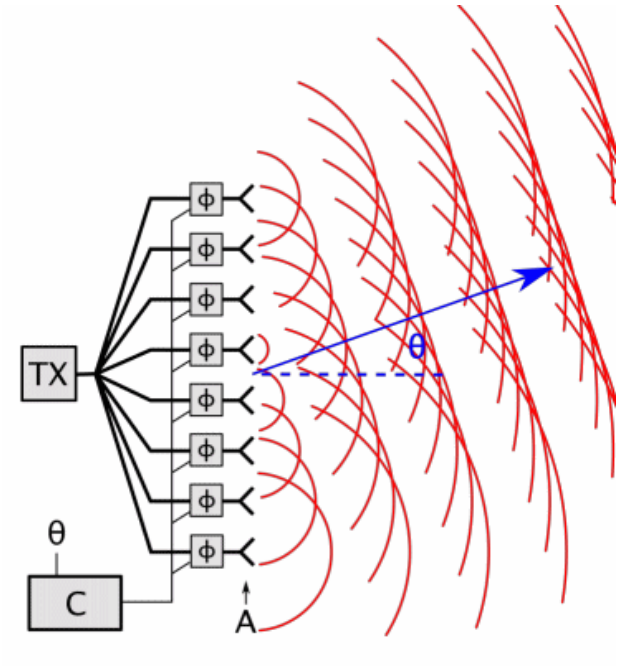


Figure 3.4: Phased-array LiDAR concept (Adapted from [57]).

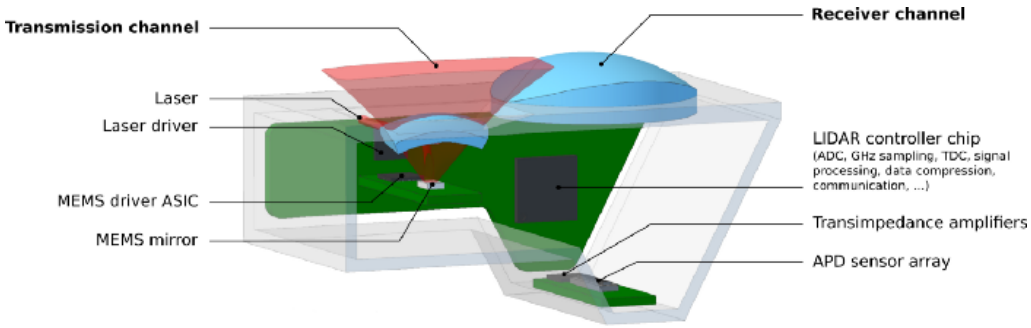


Figure 3.5: MEMS mirror LiDAR [61].

However, they are susceptible to mirrors drifting, which makes necessary the realignment and recalibration.

3.2 UAV Localization

In robotics, robot localization is the procedure of determining the robot's position and orientation, commonly called pose, in the environment. This process is usually realized by combining several measurements from different sensors. For instance, Inertial Navigation System (INS) and Global Positioning System (GPS) have been used to determine the determine the vehicle localization. In the case of UAVs, the pose is obtained in real time on-board the drone [62].

3.2.1 Reference Frames

In order to achieve the position and orientation of a robot, is necessary to understand the concept of reference frames. The reason is that each sensor produces data in a different reference frame. For instance, figure 3.6 shows some of the frames adopted in this dissertation:

- Earth-centered Earth-fixed (ECEF) frame;
- Local navigation frame;
- Body-fixed frame;
- IMU-fixed frame;

In addition, the dissertation also considers the LiDAR frame (figure 3.7).Therefore, it is required to realize a transformation between the reference frames [63].

3.2.1.1 ECEF Frame

The ECEF frame has its origin in the center of the Earth. The coordinates are denoted with a superscript e and are defined as follows: the z-axis points to the north pole; the x-axis intersects the prime meridian and the equator; the y-axis is perpendicular to other axes respecting the right-hand rule [64]. Hence, the position vector in the ECEF frame is expressed by:

$$\mathbf{p^e} = \begin{bmatrix} x^e \\ y^e \\ z^e \end{bmatrix} \tag{3.4}$$

An additional variant commonly used to define a point in ECEF frame is the geodetic coordinate system. The geodetic system describes a point regarding longitude (λ), latitude (ϕ) and height(h). The latitude express the angle between the equatorial plane and the point of interest. The longitude is obtained from the angle between the prime meridian and the measured point. Finally, the height is the vertical distance from the ellipsoid to the point of interest. Therefore, a vector in the geodetic system is given by:

$$\mathbf{p^e} = \begin{bmatrix} \lambda^e \\ \phi^e \\ h^e \end{bmatrix} \tag{3.5}$$

The two variants are related using the standard World Geodetic System (WGS), also known as WGS84. Table 3.1 shows the main parameters defined by the standard WGS84. Consequently, considering a vector in geodetic coordinates,

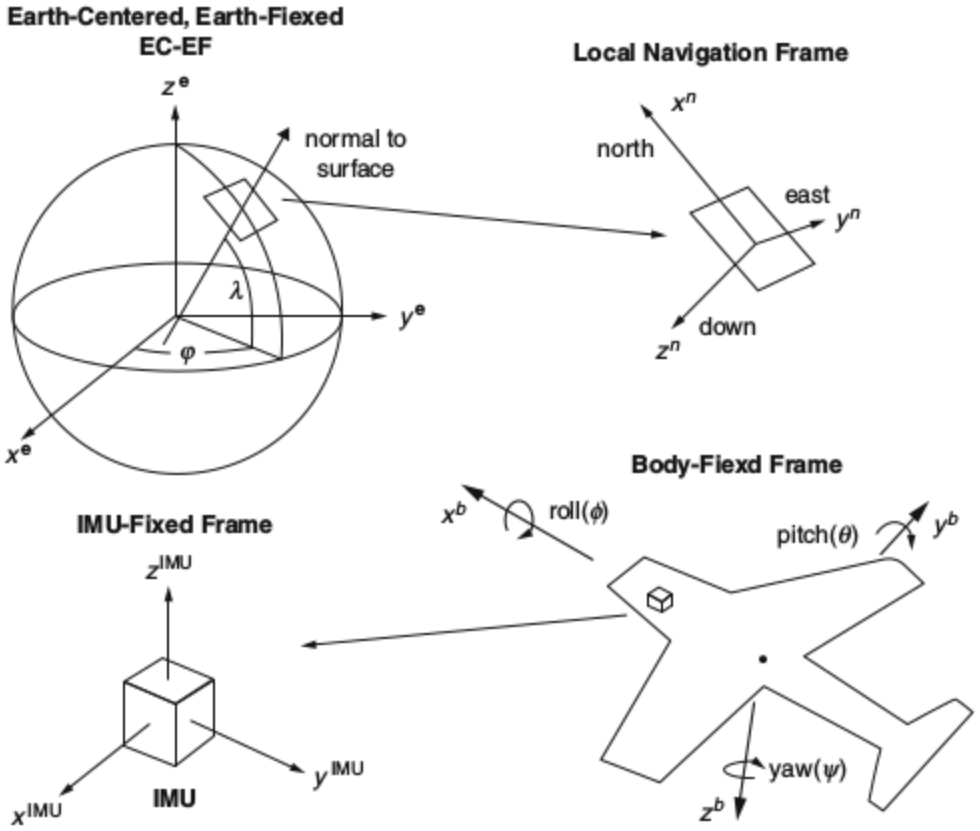


Figure 3.6: Reference frames in localization [62].

Table 3.1: WGS84 Parameters

Description	Notation	Value	Unit
Semi-major axis	a	6,378,137.0	m
Semi-minor axis	b	6,356,752.314245	m
Flattening Factor of the Earth	$1/f$	298.257223563	
Nominal Mean Angular Velocity	ω	7.292115×10^{-5}	rad/s
Geocentric Gravitational Constant	GM	$3,986,004.418 \times 10^{8}$	m^3/s^2

the transformation to Cartesian coordinates in the ECEF frame is given by:

$$\mathbf{p^e} = \begin{bmatrix} x^e \\ y^e \\ z^e \end{bmatrix} = \begin{bmatrix} (N_E + h)\cos\phi\cos\lambda \\ (N_E + h)\cos\phi\sin\lambda \\ \left[N_E(1-e^2) + h\right]\sin\phi \end{bmatrix} \tag{3.6}$$

where e^2 is the first eccentricity squared

$$e^2 = \frac{a^2 - b^2}{a^2} = 6.6943799014 \times 10^{-3} \tag{3.7}$$

and N_E is the prime vertical radius of curvature

$$N_E = \frac{a}{\sqrt{1 - e^2 \sin^2 \phi}} \tag{3.8}$$

3.2.1.2 Local Navigation Frame

The local navigation frame, also known as local tangent plane, has its origin fixed in a arbitrary point in the Earth's surface. Similarly to the ECEF frame, the local navigation frame is Cartesian and can be represented by two variants: the North-East-Down (NED) coordinates and the East-North-Up (ENU) frame.

In the local NED frame the x-axis points north, the y-axis points east and the z-axis points downwards perpendicular to the other two axes. On the other hand, in the local ENU frame the x-axis points east, the y-axis points north and the z-axis points upwards orthogonal to the previous axis. Finally, the position vector is labeled with a superscript n as follows:

$$\mathbf{p^n} = \begin{bmatrix} x^n \\ y^n \\ z^n \end{bmatrix} \tag{3.9}$$

3.2.1.3 Body-Fixed Frame

The body-fixed frame is defined arbitrarily in the body of the vehicle. There are two ways to define the coordinates: Forward-Right-Down (FRD) and Forward-Left-Up (FLU) frames. In the former case, the x, y and z-axis points forward, right and downwards, respectively, while the latter case differs by expressing the y-axis and z-axis pointing left and upwards, respectively. The coordinates vectors in the body frame is labeled with the superscript b and are defined:

$$\mathbf{p^b} = \begin{bmatrix} x^b \\ y^b \\ z^b \end{bmatrix} \tag{3.10}$$

3.2.1.4 IMU-Fixed Frame

The IMU frame has its fixed origin in the IMU embedded in the vehicle. Assuming that the accelerometers and gyroscopes of the sensor are in a perpendicular layout, then the axes of the IMU-fixed frame are aligned with each accelerometer and gyros. The vectors in this frame are assigned with the superscript IMU. For instance, the acceleration vector is defined as follows:

$$\mathbf{a^{IMU}} = \begin{bmatrix} x^{IMU} \\ y^{IMU} \\ z^{IMU} \end{bmatrix} \tag{3.11}$$

3.2.1.5 LiDAR Frame

Finally, the LiDAR frame has its origin generally fixed to device centre. Figure 3.7 show the coordinate system of a spinning LiDAR. Typically, a point in the LiDAR frame is given in polar coordinates as follows:

$$\mathbf{p}^{\mathbf{L}} = \begin{bmatrix} \alpha \\ \omega \\ r \end{bmatrix} \tag{3.12}$$

where the **L** superscript labels the LiDAR frame.

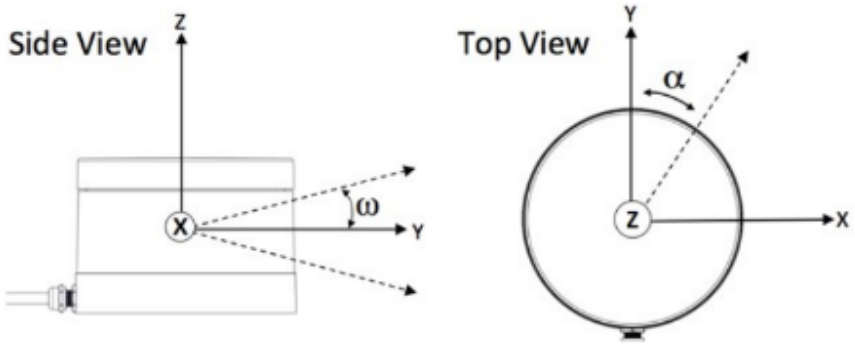


Figure 3.7: Reference frame of a spinning LiDAR (Adapted from [59]).

3.2.2 Attitude Representation

3.2.2.1 Euler Angles

One variant to represent the vehicle's orientation in space is the Euler angles parametrization ($\mathbf{\Psi} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$). In this representation, any orientation is described by combining three successive elementary rotations around the x(ϕ) , y(θ), z(ψ) axes, which are defined:

$$\mathbf{R_x}(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{3.13}$$

$$\mathbf{R_y}(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.14}$$

$$\mathbf{R_z}(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.15}$$

In order to complement the explanation, figure 3.8 illustrates the rotations in the case of an UAV body frame.

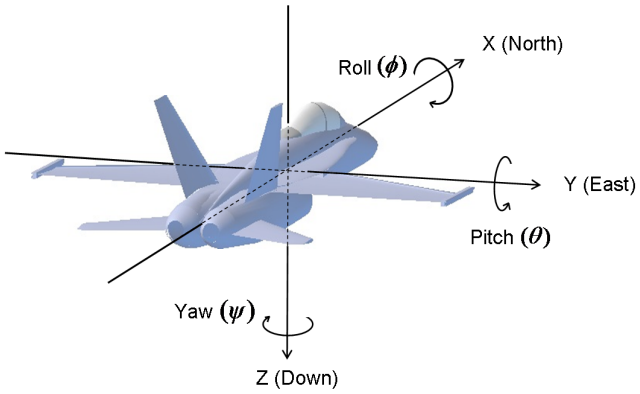


Figure 3.8: Euler Angles in a UAV [65]

These rotations are denominated roll(ϕ), pitch(θ) and yaw(ψ), respectively. The order in which these matrices are applied is important. In this dissertation is considered the Euler ZYX convention, that is, a positive yaw-pitch-roll ordering. Therefore, the resulting matrix is:

$$\begin{aligned}\mathbf{R}(\phi,\theta,\psi) &= \mathbf{R_z}(\psi)\mathbf{R_y}(\theta)\mathbf{R_x}(\phi) \\ &= \begin{bmatrix} c(\psi)c(\theta) & c(\psi)s(\theta)s(\phi) - s(\psi)c(\phi) & c(\psi)s(\theta)c(\phi) + s(\psi)s(\phi) \\ s(\psi)c(\theta) & s(\psi)s(\theta)s(\phi) + c(\psi)c(\phi) & s(\psi)s(\theta)c(\phi) - c(\psi)s(\phi) \\ -s(\theta) & c(\theta)s(\phi) & c(\theta)c(\phi) \end{bmatrix}\end{aligned} \tag{3.16}$$

where $c(\cdot) := \cos(\cdot)$ and $s(\cdot) := \sin(\cdot)$.

Following the equation 3.16, the Euler angles can be determined [66].

$$\begin{bmatrix}\phi \\ \theta \\ \psi\end{bmatrix} = \begin{bmatrix} atan2(r_{32}, r_{33}) \\ -asin(r_{31}) \\ atan2(r_{21}, r_{11}) \end{bmatrix} \tag{3.17}$$

where r_{ij} is the element in the row i and column j. The Euler angles are commonly used since is more perceptible to humans. However, this representation presents singularity when the pitch value is near $\theta = \pi/2$.

3.2.2.2 Quaternions

The aforementioned problem is avoided by using another variant to represent the orientation of the vehicle: quaternions ($\mathbf{q} = [q_0, q_1, q_2, q_3]^T$). The parameter q_0 is the scalar part while the parameters q_1, q_2 and q_3 are the vector part.

The Euler angles and quaternions are related by the equation:

$$\mathbf{q} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos(\frac{\psi}{2})\cos(\frac{\theta}{2})\cos(\frac{\phi}{2}) + \sin(\frac{\psi}{2})\sin(\frac{\theta}{2})\sin(\frac{\phi}{2}) \\ \cos(\frac{\psi}{2})\cos(\frac{\theta}{2})\sin(\frac{\phi}{2}) - \sin(\frac{\psi}{2})\sin(\frac{\theta}{2})\cos(\frac{\phi}{2}) \\ \cos(\frac{\psi}{2})\sin(\frac{\theta}{2})\cos(\frac{\phi}{2}) + \sin(\frac{\psi}{2})\cos(\frac{\theta}{2})\sin(\frac{\phi}{2}) \\ -\cos(\frac{\psi}{2})\sin(\frac{\theta}{2})\sin(\frac{\phi}{2}) + \sin(\frac{\psi}{2})\cos(\frac{\theta}{2})\cos(\frac{\phi}{2}) \end{bmatrix} \tag{3.18}$$

Considering a positive yaw-pitch-roll rotation, the resulting rotation matrix is:

$$\mathbf{R}(\mathbf{q}) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{3.19}$$

3.2.3 Frames Relations

The relationship of the coordinates of one point in two references frames can be expressed by the homogeneous transformation matrix [67]. Figure 3.9 shows the vectors $\mathbf{p}^0$ and $\mathbf{p}^1$ with respect to Frame 0 and Frame 1, respectively. Considering $\mathbf{R}_1^0$ as the rotation matrix of Frame 1 to Frame 0, the vectors are then related:

$$\mathbf{p}^0 = \mathbf{o}_1^0 + \mathbf{R}_1^0\mathbf{p}^1 \tag{3.20}$$

where $\mathbf{o}_1^0$ is the origin of Frame 1 with respect to Frame 0.

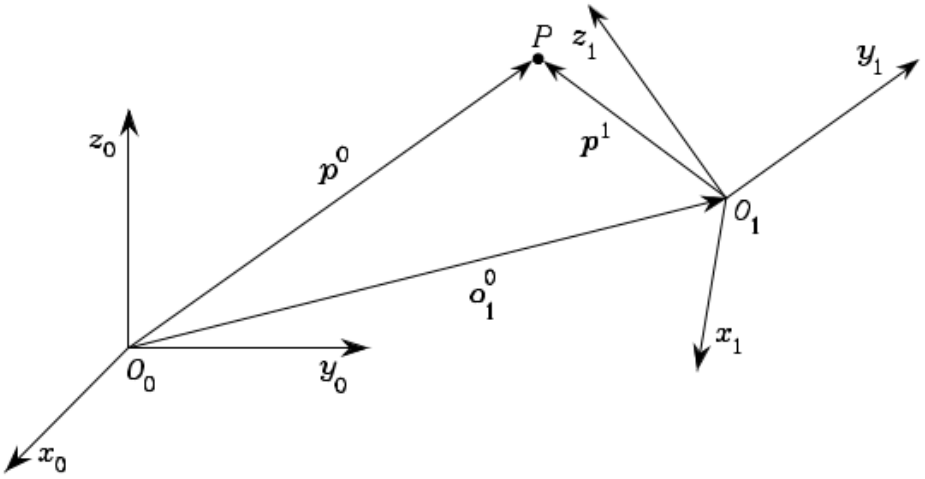


Figure 3.9: Representation of one point in two frames [67].

By adding a fourth unit component to the vectors $\mathbf{p}^0$ and $\mathbf{p}^1$ and expanding the equation 3.20, the transformation can be defined in terms of matrix:

$$\begin{bmatrix} \mathbf{p^0} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_1^0 & \mathbf{o}_1^0 \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p^1} \\ 1 \end{bmatrix} \tag{3.21}$$

Therefore, the homogeneous transformation matrix given by:

$$\mathbf{T}_1^0 = \begin{bmatrix} \mathbf{R}_1^0 & \mathbf{o}_1^0 \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{3.22}$$

express the transformation of a vector from a generic frame 1 to the generic frame 0.

3.2.4 Extended Kalman Filter

The Extended Kalman Filter (EKF) is a recursive algorithm to estimate the state of dynamic systems [68, 69]. The state is a set of variables that parameterize the mathematical model of a dynamic system. The EKF differs from the usual Kalman Filter (KF) by assuming that the system is nonlinear. Moreover, the filter has a process model and an observation model and both have additive white Gaussian noisy errors. The EKF fuses the sensors data in a two-step cycle: prediction and update.

1. Prediction Step
   First, let the nonlinear process system be modelled as

$$\mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1} \tag{3.23}$$

$$\mathbf{z}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k \tag{3.24}$$

   where $\mathbf{x}_k$ and $\mathbf{w}_{k-1}$ are the $n \times 1$ vector states and process noise vector, respectively, while $\mathbf{z}_k$ is the $m \times 1$ observation vector and $\mathbf{v}_k$ is $m \times 1$ the measurement noise vector.
   The system has initial state $\mathbf{x}_0$ with mean $\mu_0 = E[\mathbf{x}_0]$ and initial covariance $\mathbf{P}_0 = E[(\mathbf{x}_0 - \mu_0)(\mathbf{x}_0 - \mu_0)^T]$, where $E[\cdot]$ is the expectation operator. In the following derivation, the notation $\mathbf{x}_{\beta|\alpha}$ means the estimation at time β given the observations up to time $\beta > \alpha$.
   Assuming that the last optimal estimation is

$$\mathbf{x}_{k-1|k-1} = E\left[\mathbf{x}_{k-1}|\mathbf{Z}_{k-1}\right] \tag{3.25}$$

   The prediction step tries to find a prediction $\mathbf{x}_{k|k-1}$. Expanding $\mathbf{f}(\cdot)$ in Taylor Series about $\mathbf{x}_{k-1|k-1}$ and truncating at fist order gives

$$\mathbf{f}(\mathbf{x}_{k-1}) = \mathbf{f}(\mathbf{x}_{k-1|k-1}) + \mathbf{F}_k(\mathbf{x}_{k-1} - \mathbf{x}_{k-1|k-1}) \tag{3.26}$$

   where $\mathbf{F}_k$ if the Jacobian of $\mathbf{f}(\cdot)$ about $\mathbf{x}_{k-1|k-1}$ and is defined as

$$\mathbf{F} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{3.27}$$

Therefore, from equations 3.23 and 3.26 the predicted state is

$$\begin{aligned}\mathbf{x}_{k|k-1} &= E[\mathbf{x}_k|\mathbf{Z}_{k-1}] \\ &= E[\mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1}|\mathbf{Z}_{k-1}] \\ &= E[\mathbf{f}(\mathbf{x}_{k-1|k-1}) + \mathbf{F}_k(\mathbf{x}_{k-1} - \mathbf{x}_{k-1|k-1}) + \mathbf{w}_{k-1}|\mathbf{Z}_{k-1}] \\ &= \mathbf{f}(\mathbf{x}_{k-1|k-1}) \end{aligned} \tag{3.28}$$

Now, defining the prediction error as

$$\mathbf{e}_{k|k-1} \triangleq \mathbf{x}_k - \mathbf{x}_{k|k-1} \tag{3.29}$$

Substituting equations 3.23 and 3.28 in the error equation gives

$$\begin{aligned}\mathbf{e}_{k|k-1} &= \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1} - \mathbf{f}(\mathbf{x}_{k-1|k-1}) \\ &= \mathbf{F}_k\mathbf{e}_{k-1|k-1} + \mathbf{w}_{k-1} \end{aligned} \tag{3.30}$$

Consequently, the error covariance matrix is

$$\begin{aligned}\mathbf{P}_{k|k-1} &= E[\mathbf{e}_{k|k-1}\mathbf{e}^{\mathbf{T}}{}_{k|k-1}] \\ &= \mathbf{F}_k E[\mathbf{e}_{k-1|k-1}\mathbf{e}^{\mathbf{T}}{}_{k-1|k-1}]\mathbf{F}^{\mathbf{T}}{}_k + E[\mathbf{w}_{k-1}\mathbf{w}_{k-1}^T] \\ &= \mathbf{F}_k\mathbf{P}_{k-1}\mathbf{F}^{\mathbf{T}}{}_k + \mathbf{Q}_{k-1} \end{aligned} \tag{3.31}$$

where $\mathbf{Q}_{k-1}$ is process noise covariance matrix.

2. Update Step
This stage of the EKF tries to find a linear estimator for $\mathbf{x}_k$. Similar to the derivation of the predicted step, expanding $\mathbf{h}(\cdot)$ in Taylor Series about $\mathbf{x}_{k|k-1}$ and truncating at fist order results in

$$\mathbf{h}(\mathbf{x}_k) = \mathbf{h}(\mathbf{x}_{k|k-1}) + \mathbf{H}_k(\mathbf{x}_k - \mathbf{x}_{k|k-1}) \tag{3.32}$$

where $\mathbf{H}_k$ if the Jacobian of $\mathbf{h}(\cdot)$.
Defining the predicted observation

$$\mathbf{z}_{k|k-1} \triangleq E[\mathbf{z}_k|\mathbf{Z}_{k-1}] \tag{3.33}$$

Then, from equations 3.24 and 3.32

$$\mathbf{z}_{k|k-1} = \mathbf{h}(\mathbf{x}_{k|k-1}) \tag{3.34}$$

The measured observation error $\mathbf{i}_k$, also known as innovation, and the predicted observation error $\tilde{\mathbf{z}}_{k|k-1}$ are given by

$$\mathbf{i}_k = \mathbf{z}_k - \mathbf{h}(\mathbf{x}_{k|k-1}) \tag{3.35}$$

$$\begin{aligned}\tilde{\mathbf{z}}_{k|k-1} &\triangleq \mathbf{z}_k - \mathbf{z}_{k|k-1} \\ &= \mathbf{H}_k[\mathbf{x}_{k|k-1} - \mathbf{x}_k] + \mathbf{v}_{k-1} \end{aligned} \tag{3.36}$$

Squaring and taking expectations on both sides of equation 3.36 results

$$\begin{aligned}\mathbf{S}_k &= E[\tilde{\mathbf{z}}_{k|k-1}\tilde{\mathbf{z}}_{k|k-1}^{\mathbf{T}}] \\ &= \mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^{\mathbf{T}} + \mathbf{R}_k\end{aligned} \tag{3.37}$$

where $\mathbf{R}_k$ is observation noise covariance matrix.
Assuming a prediction $\mathbf{x}_{k|k-1}$ with covariance $\mathbf{P}_{k|k-1}$ and an observation $\mathbf{z}_k$, the best unbiased estimate is

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{h}(\mathbf{x}_{k|k-1})) \tag{3.38}$$

where $\mathbf{K}_k$ is an appropriate gain matrix.
The estimation error is

$$\tilde{\mathbf{x}}_{k|k} = \mathbf{x}_k - \mathbf{x}_{k|k} \tag{3.39}$$

Substituting equations 3.23 and 3.38 in 3.39

$$\begin{aligned}\tilde{\mathbf{x}}_{k|k} &= \mathbf{f}(\mathbf{x}_{k-1}) + \mathbf{w}_{k-1} - (\mathbf{x}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{h}(\mathbf{x}_{k|k-1}))) \\ &= [\mathbf{I} - \mathbf{K}_k\mathbf{H}_k]\mathbf{F}_k\mathbf{e}_{k|k-1} + [\mathbf{I} - \mathbf{K}_k\mathbf{H}_k]\mathbf{w}_{k-1} - \mathbf{K}_k\mathbf{v}_k\end{aligned} \tag{3.40}$$

The updated covariance estimate is calculated by

$$\begin{aligned}\mathbf{P}_{k|k} &\triangleq E[\tilde{\mathbf{x}}_{k|k}\tilde{\mathbf{x}}_{k|k}^{\mathbf{T}}] \\ &= \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{H}_k\mathbf{P}_{k|k-1} - \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{K}_k^T + \mathbf{K}_k\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{K}_k^T + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^T\end{aligned} \tag{3.41}$$

The value for $\mathbf{K}_k$ is chosen to minimize the trace of $\mathbf{P}_{k|k-1}$ with reference to $\mathbf{K}_k$

$$\frac{\partial tr(\mathbf{P}_k)}{\partial \mathbf{K}_k} = 0 \tag{3.42}$$

Therefore, the gain is given by

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}^{\mathbf{T}}{}_k\mathbf{S}^{-\mathbf{1}}{}_k \tag{3.43}$$

Substituting equation 3.43 in 3.41 results

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \tag{3.44}$$

In summary, the EKF equations are:

- Prediction:

$$\begin{aligned}\mathbf{x}_{k|k-1} &= \mathbf{f}(\mathbf{x}_{k-1|k-1}) \\ \mathbf{P}_{k|k-1} &= \mathbf{F}_k\mathbf{P}_{k-1}\mathbf{F}^{\mathbf{T}}{}_k + \mathbf{Q}_{k-1}\end{aligned}$$

- Update:

$$\begin{aligned}
\mathbf{S}_k &= \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H^T}_k + \mathbf{R}_k \\
\mathbf{K}_k &= \mathbf{P}_{k|k-1} \mathbf{H^T}_k \mathbf{S^{-1}}_k \\
\mathbf{x}_{k|k} &= \mathbf{x}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{h}(\mathbf{x}_{k|k-1})) \\
\mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\mathbf{P}_{k|k-1}
\end{aligned}$$

3.3 Data Processing

3.3.1 Point Cloud Downsampling

The point cloud datasets can be very large, making data processing complex and memory consuming. Furthermore, many algorithms do not need to process all points in the point cloud. Therefore, point cloud downsampling techniques are applied in order to reduce the number of points present in the cloud, enabling better performance in terms of time and memory usage.

3.3.1.1 Voxel Grid Filter

The Voxel Grid filter procedure consists on sampling the input point cloud using a 3-D voxel grid. Volumetric pixels (voxel) in a 3-D space can be understood as a group of cube units that represent a single data point on a regular grid (see figure 3.10). This technique has been used in the area of computer graphics to reduce the number of points [70].

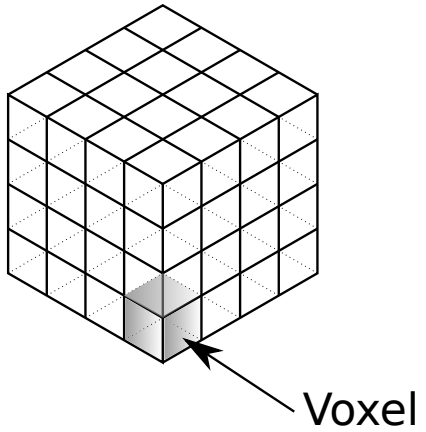


Figure 3.10: Voxel Grid with single voxel shaded in grey.

The points can be approximated using the voxel centre, but this method does not represent the surface accurately. Other approach is approximate the points that lie in each voxel with their centroid. Then, considering a voxel containing a set of n points with coordinates $[x, y, z]$, the output point $\mathbf{p_n} = \begin{bmatrix} \bar{x} & \bar{y} & \bar{z} \end{bmatrix}^T$ can

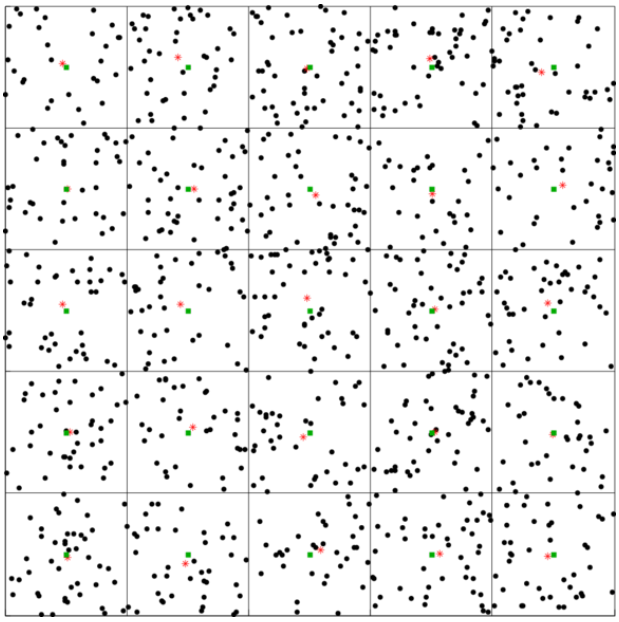


Figure 3.11: Representation of a Voxel Grid Filter in 2-D [71].

be computed:

$$\bar{x} = \frac{1}{n}\sum_{i=0}^{i=n} x_i \tag{3.45}$$

$$\bar{y} = \frac{1}{n}\sum_{i=0}^{i=n} y_i \tag{3.46}$$

$$\bar{z} = \frac{1}{n}\sum_{i=0}^{i=n} z_i \tag{3.47}$$

Figure 3.11 illustrates the filter process in 2-D for a better comprehension. The green squares are the centre of the voxel while the red asteriks are the centroid's centre. This method allows to decrease the number of points of the cloud and gives a point cloud with approximately constant density.

3.4 Data Structuring

3.4.1 Octree

Octree [72] is a hierarchical tree data structure technique for the representation of 3-D data in space. The 3-D space is subdivided into eight children (octants) by each internal nodes until the information in each node is below a threshold or the maximum tree size is achieved. Figure 3.12 shows the general representation of an octree structure.

The root node represents the entire object and the leaf nodes correspond to cubes for which no further subdivision is required. Octree differs from a regular grid by allowing it to store sparse data.

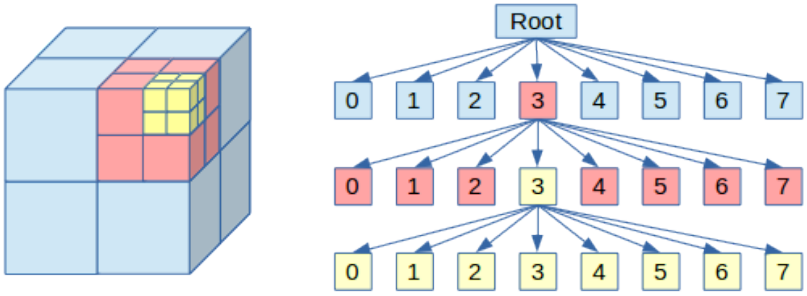


Figure 3.12: General representation of an octree[73].

Octrees decreases memory requirements by implementing lossless encoding of the data [74]. Figure 3.13 shows the effect of different resolution in an octree.

Figure 3.13: Effect of different resolutions values for an octree[74].

3.5 Plane Detection

3.5.1 General Form of the Plane Equation

A plane in the 3-D Cartesian coordinate space can be defined by a point and a vector that is perpendicular to the plane [75]. Figure 3.14 illustrates a plane in the 3-D space. Considering two points in the plane $\mathbf{p_0} = (x_0, y_0, z_0)$, $\mathbf{p} = (x, y, z)$ and defining the normal vector of a plane as:

$$\mathbf{n} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{3.48}$$

The normal vector is orthogonal to any point in the plane. Therefore, the scalar product of the normal vector and $(\mathbf{p} - \mathbf{p_0})$ is given by:

$$\begin{aligned}(\mathbf{p} - \mathbf{p_0}) \cdot \mathbf{n} &= (x - x_0, y - y_0, z - z_0) \cdot (a, b, c) \\ &= a(x - x_0) + b(y - y_0) + c(z - z_0) = 0 \end{aligned} \tag{3.49}$$

Equation 3.49 can be rewritten as:

$$ax + by + cz + d = 0. \tag{3.50}$$

where $d = -ax_0 - by_0 - cz_0$.

Finally, equation 3.50 is also known as the general form of the plane equation in which parameters a, b and c represent the normal vector and parameter d is the distance of the plane to reference origin.

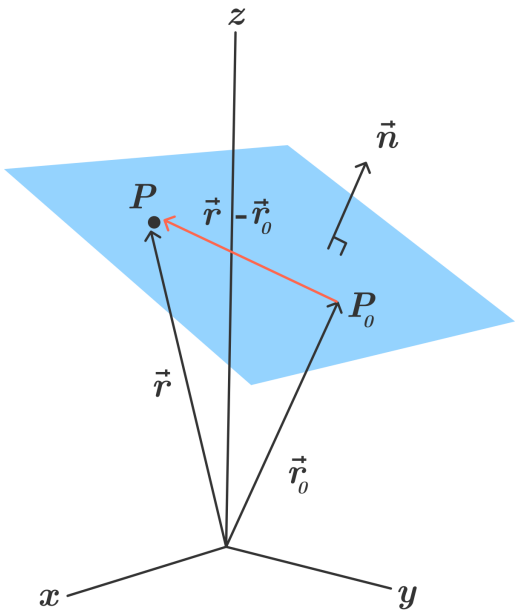


Figure 3.14: Plane representation in the Cartesian coordinate system [75].

3.5.2 Principal Component Analysis

The Principal Component Analysis (PCA) is a statistical method for reducing the dimensionality of a large dataset [76, 77]. The PCA method applies an orthogonal transformation to extract information from the dataset and map this information to a set of values called principal components.

Considering a dataset of p observation of n-dimensional vectors, the data can be written as a $n \times p$ matrix $\mathbf{X}$, whose generic element is x_{np}. First, the data X is normalized by subtracting the mean of each dimension in order to obtain a set with mean equals to zero. The mean and the subsequent mean-centered matrix are computed by:

$$u_j = \frac{\sum_{i=1}^{n} X_{ij}}{n} \tag{3.51}$$

$$\bar{\mathbf{X}} = \mathbf{X} - hu^t \tag{3.52}$$

$$\mathbf{B} = \mathbf{X} - \bar{\mathbf{X}} \tag{3.53}$$

where $j = 1, ..., p$ and h is a $n \times 1$ column vector of all 1s.

Then, the covariance matrix $\mathbf{C}$ is obtained from the matrix determined by equation 3.53. The covariance matrix is a square matrix that each element indicates the correlation between two or more random variables [78].

$$\mathbf{C} = \frac{1}{n-1} \sum_{i=1}^{n} (X - \bar{X})(X - \bar{X})^T \tag{3.54}$$

The next step of the algorithm is to apply an eigen decomposition of the covariance matrix. This process corresponds to the factorization of the matrix in a canonical form, represented in terms of eigenvalues and eigenvectors. Eigenvalues are a special set of scalars associated with a linear system of equations [79]. Each eigenvalue has a corresponding eigenvector.

Considering the covariance matrix given by equation 3.54, if there is a vector $\mathbf{v} \in \mathbb{R}^n \neq \mathbf{0}$ that satisfies:

$$\mathbf{Cv} = \lambda \cdot \mathbf{v} \tag{3.55}$$

for some scalar λ, then λ is an eigenvalue of $\mathbf{C}$ with corresponding eigenvector $\mathbf{v}$.

The equation 3.55 can be rearranged:

$$(\mathbf{C} - \lambda \cdot \mathbf{I})v = 0 \tag{3.56}$$

where $\mathbf{I}$ is the identify matrix.

The eigenvalue is determined by solving the following equation:

$$det(\mathbf{C} - \lambda \cdot \mathbf{I}) = |(\mathbf{C} - \lambda \cdot \mathbf{I})| = 0 \tag{3.57}$$

The eigenvectors are then found solving the equation 3.56 with the resulting eigenvalue. Finally, the equation 3.55 can be generalized:

$$\mathbf{V^{-1}CV} = \mathbf{D} \tag{3.58}$$

where D is the diagonal matrix of eigenvalues of C and V is the eigenvectors matrix.

After computing the eigenvalues and their respective eigenvectors, they are sorted in order of decreasing eigenvalue. The eigenvalues represent the data distribution and the principal components are the directions of the data which the

data are more spread. Therefore, the largest principal component is the direction with more variance [80].

The projection matrix **W** is built, in which the desired eigenvectors are chosen as basis vectors.

$$\mathbf{W} = \mathbf{V}_{kl} \tag{3.59}$$

where $k = 1, ..., p$, $l = 1, ..., L$ and $1 \leq L \leq p$.

Finally, the data are then projected onto the new basis by applying equation:

$$\mathbf{T} = \mathbf{BW} \tag{3.60}$$

Algorithm 3.1 summarizes the PCA method.

Algorithm 3.1 Pseudocode for the PCA algorithm.

Input: dataset **X**
Output: projected data T
1: Organize the dataset.
2: Compute the mean u_j.
3: Compute the mean-centered matrix **B**.
4: Determine the covariance matrix **C**.
5: Compute the eigenvectors and eigenvalues.
6: Sort the eigenvector in descending order regarding the eigenvalues.
7: Build the projection matrix **W**.
8: Project the data to the new basis **T**.

3.6 Robotic Middleware

Robotic middlewares implement the communication between the operating system and the software applications. They create a high-level abstraction of devices, hiding the low-level implementation of the hardware components [81]. Therefore, they manage the hardware diversity, simplifying the software development.

3.6.1 Robot Operating System - ROS

The Robot Operating System (ROS) [82] is a modular framework for the development of robotic applications software. It is a set of tools, libraries and conventions that simplifies the design of robot for multi-platform. The main characteristics of ROS are:

- Communication Peer-to-Peer (P2P);
- Open source;

- Support for several programming languages, such as C++, Python and others.

It is a communication structure that acts as a layer of the operating system on networked computers. The communication is implemented by exchanging messages through independent process called nodes. The concepts are explained below:

- **Nodes**: nodes are processes that execute some of developed code. For instance, the control system of a robot has in general several nodes. One node of ROS is written using the libraries *roscpp* (for C++ applications) or *rospy* (for Python applications);
- **Master**: provides registration and name search. Without the master, nodes are unable to exchange messages or call services;
- **Messages**: it is data structure. Support primitive data types or other structures;
- **Topics**: The messages are published through *publisher/subscriber* mechanism. A node sends a ROS message by publishing it on a topic. The topic is the name that identifies the content of the message. Another node that wants to use the data from this published message must subscribe to the appropriate topic. In this type of communication, there may be multiple publisher and subscribers to a single topic or a single node may publish or subscribe to several topics. In general, publishers and subscribes are not aware of each other. Figure 3.15 demonstrates this mode of communication.

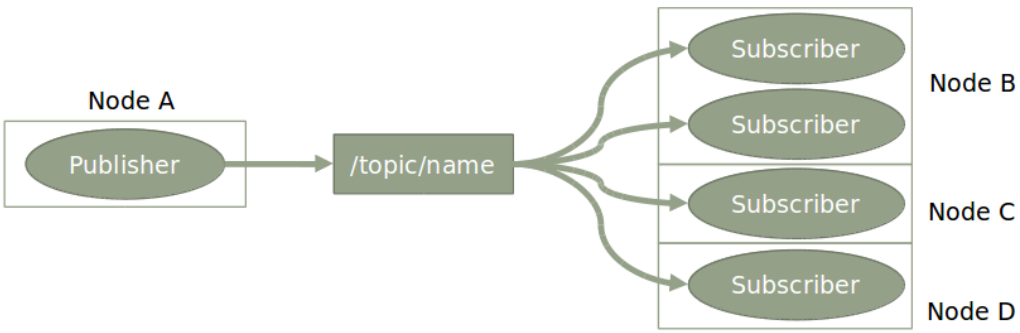


Figure 3.15: Publisher/Subscriber communication.

- **Services**: Although the publisher/subscriber mechanism is very flexible, this mode of communication is appropriate for request/response that are usually necessary in a distributed system. Thus, ROS can also communicates through a *client/server* system. Services are a pair of messages: *request*

and *response*. A node offers a service through a service name and a client can use the service by sending *request* and waiting for a *response*. This mode is summarize is figure 3.16.

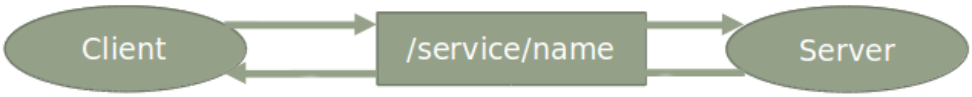


Figure 3.16: Server/client communication.

The nodes are collected into *packages* which contains information about dependencies for build, run and test. This type of organization encourages the collaborative development, as it facilitates the package migration between systems and developers.

3.7 Robotic Simulators

It is generally interesting to check that a software does what is expected and does not cause any damage to the physical system on which it is operating. Therefore, a simulator is required.

Currently, robotic simulators are essential components for the development of algorithms applied to robots. They allow the validation of concepts, verification of software feasibility without applying them to a physical system.

There are several open-source robotics simulators, such as Gazebo [83], Virtual Robot Experimentation Platform (V-REP) [84] and Modular Open Robots Simulation Engine (MORSE) [85, 86, 87].

3.7.1 Modular Open Robots Simulation Engine - MORSE

MORSE is an open-source simulator that provides several features of interest for robotic projects. It is developed in Python and makes use of the Blender Game Engine[1] to model and render the simulation (see figure 3.17). Besides this, MORSE is only operated from a command line.

Moreover, MORSE provides a set of standard robots, sensors, and actuators that can be interconnected to create any robot configuration. This flexibility allows the user to control the level of abstraction in a variety of simulation scenarios. MORSE also supports various middlewares, including ROS.

The principle of MORSE operation is based on two parts: a Blender file and a Python script. As mentioned before, the Blender file corresponds to the simulation scenario and its physical properties while the Python script holds the components and methods. The components available in MORSE are:

[1] `https://www.blender.org/`

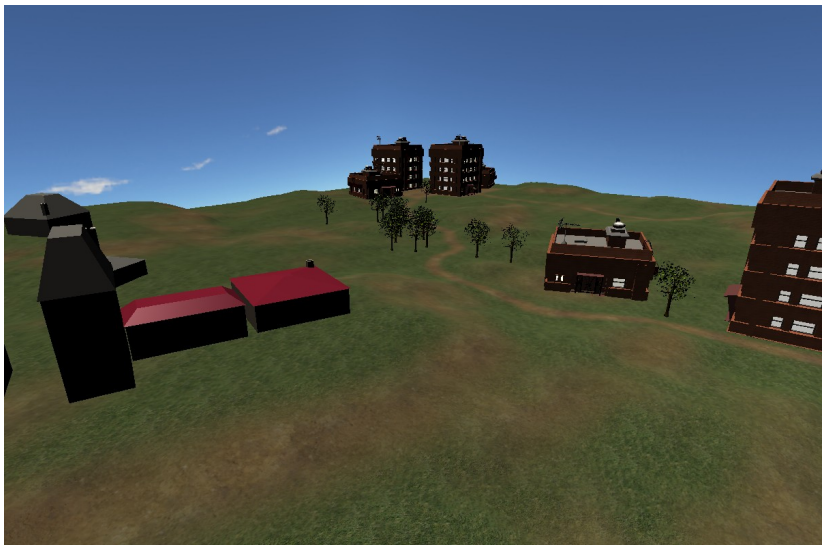


Figure 3.17: Example of a Blender environment used in MORSE [88]

- **Sensors**: represent a real sensor, providing data from simulated environment, such as robot pose, laser scan and others.
- **Actuators**: generate an action on the robot, such as moving the robot, change the position of a robotic arm and others.
- **Robots**: contain the robot model and functions. In addition to that, append the sensors to the robot.
- **Environment**: define the 3-D model scenario.
- **Middleware**: defines the communication protocol between the simulation and an user developed algorithm.

Chapter 4

System Design

After analyzing the works presented in chapter 2, some decisions were made for the design of the system in order to achieve the objectives discussed in chapter 1, specifically the algorithm for detecting landing zones for UAVs in emergencies.

The usefulness of detecting emergency landing spots for UAVs has already been described in chapter 1. Nevertheless, some hardware and software requirements are necessary in order to achieve the expected objective. First, a perception sensor such as LiDAR is needed to acquire the terrain features while a localization module is essential to allow frame transformations.

This chapter contains an analysis of the hardware and software architectures for implementing the developed algorithm. The software benefits from the Robot Operating System (ROS) [82] modular structure, using the available ROS packages and facilitating its integration in other future applications.

4.1 Hardware Architecture

The landing spot detection algorithm is designed to be executed simultaneously with others UAV tasks during operation. A spinning LiDAR provides 3-D point clouds that are processed using an onboard computer. Moreover, the computer is also responsible to configure the LiDAR.

Considering that modern LiDARs produce hundreds of thousands of points in each scan, a high transmission rate between the sensor and the computer is necessary. For instance, an Ethernet connection allows the required rate (4.1, in blue).

The point clouds are given in the LiDAR reference frame. Consequently, knowing the UAV pose is essential in order to transform the data to a fixed

frame. There are several configuration to estimate the vehicle state. In this project we are considering two setup. One configuration is that Global Navigation Satellite System (GNSS) and Inertial Measurement Unit (IMU) measurements are fused in an onboard autopilot, such as Pixhawk[1], and then the estimated pose is sent to the computer. The other layout is that the GNSS and IMU data are directly fused in the onboard computer. The GNSS allows the synchronization of the timestamps using Coordinated Universal Time (UTC) as reference. The UTC system is a primary time standard by which the world regulates clocks and time. The receiver provides a Pulse Per Second (PPS) signal and GNSS data (figure 4.1, in red) that are used to synchronize the LiDAR sensor's data timestamp, the system time of the onboard computer, and the estimated inertial data from the autopilot. Figure 4.1 represents the high-level architecture of the former configuration.

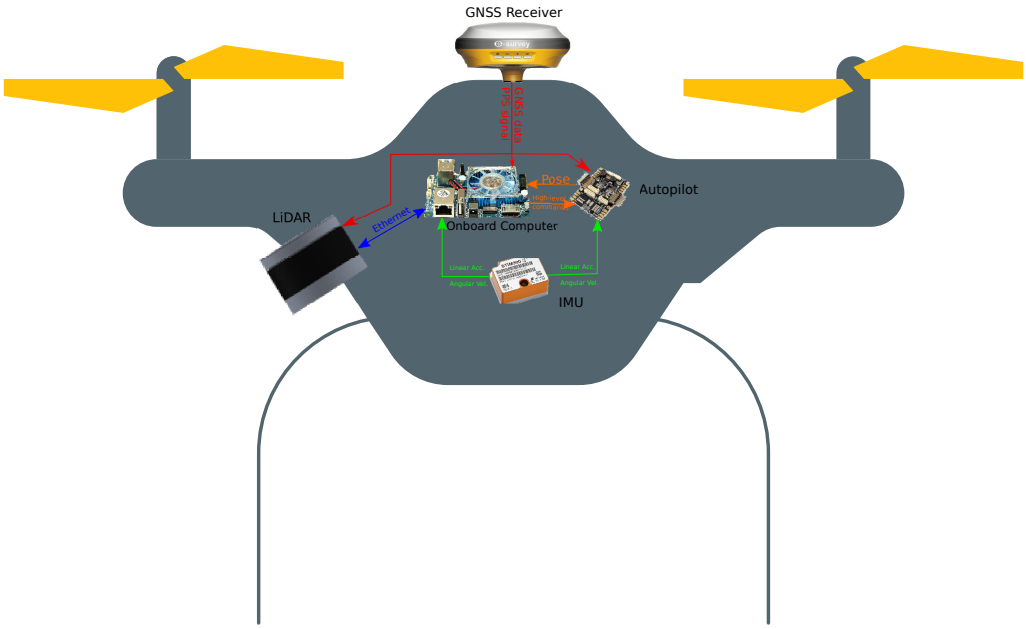


Figure 4.1: High-level hardware architecture (Adapted from [89, 90]).

4.2 Software Architecture

In figure 4.2 is presented the high-level software architecture to detected a potential landing spot given a point cloud generated by a LiDAR sensor. All the pipeline elements were developed within the ROS framework, except for the sensors input data. The software is divided into several data processing blocks:

[1] `https://pixhawk.org/`

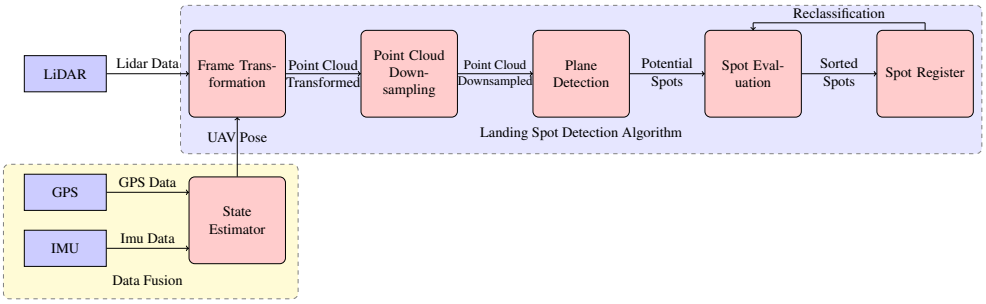


Figure 4.2: High-level software pipeline.

- **Data Fusion**: the data fusion section estimates the UAV's states by fusing the measurements from the IMU and GPS. There are several approaches to realize this procedure. For example, the estimation can be performed in the computer's ROS environment using the *robot_localization* [2] ROS package. This package applies an Extended Kalman Filter algorithm given the IMU and GPS data.

- **Frame Transformations**: considering that the LiDAR is fixed to the vehicle, then the LiDAR pose is fixed in relation to the UAV. Hence, knowing the UAV's pose and the LiDAR pose, the LiDAR point cloud can be transformed from the LiDAR frame to the local navigation frame. Using the *tf2*[3] package, the relations between the UAV, LiDAR and global coordinate frames are established. Furthermore, the package can also tracks the frame's relations over time.

- **Point Cloud Downsampling**: each LiDAR scan produces a point cloud with thousands of points. Considering that the vehicle did not travel a large distance to detect a spot, is important to accumulate the point clouds with the intention to not lose information. Conversely, accumulating the point clouds increases computational effort. Therefore, the referenced point cloud are downsampled.

- **Plane Detection**: following the downsampling of a point cloud, a plane can be detected in the new point cloud. The plane detection can be subdivide into several steps, conforming the available time, the computational power and the desired resolution. Generally, the algorithms consist in estimating the parameters from the plane equation (see section 3.50) in a limited region of the original point cloud. Moreover, the algorithms have to fit a plane in the presence of outliers, i.e., points that do not fit the plane model. Besides the landing spot detection for aerial vehicles, estimating

[2] `http://wiki.ros.org/robot_localization`
[3] `http://wiki.ros.org/tf2`

the ground conditions is also useful to detect clear paths and to make further processing less complex.

- **Spot Evaluation**: the segmented point cloud containing the detected plane are then evaluated to classify the spot's reliability. The roughness of the landing zone can be assessed by computing the standard deviation in the z axis. The higher the standard deviation, the rougher area. In addition, a high value can also indicate the presence of obstacles, such as trees or buildings. The spots are also evaluated regarding their slope as the landing zone can not be steep enough to destabilise the vehicle when landed. Furthermore, the size of area and the distance to the UAV can also be used to evaluate the spot. Each evaluation factor has different importance according to the environment. Thus, it becomes interesting to assign different weights to these factors.

- **Spot Register**: the assessed spot are then registered as a landing spot. However, The suitability of a landing point to be the optimal choice varies during operation. In this way, the points registered have to be periodically reassessed.

Chapter 5

Emergency Landing Spot Detection Algorithm

This chapter presents in detail the algorithm developed to achieve the objectives and requirements for this dissertation. First, the conception of each data processing block is presented, following with the complete description of its procedure. Finally, the algorithm architecture and its pipeline is presented. The algorithm here presented resulted in the publication of the paper *Emergency Landing Spot Detection for Unmanned Aerial Vehicle* at the ROBOT'2019: Fourth Iberian Robotics Conference [91].

5.1 Algorithm Procedure

For the purposes of fulfilling the objectives presented in chapter 1.2, the method procedure is divided into several steps: the frame transformation of the input data, the downsampling of the point cloud, the spatial structuring of the data, detection of planes, filtering of potential candidates and, finally, classification of detected spots. Besides that, there are some rules created in order to obtain the desired objective. Figure 5.1 shows the flowchart of the developed software.

5.1.1 Frames Transformation

The problem of frames transformation consists of determining the point from a reference frame to another. Figure 5.2 displays a general scenario of multiple frames of reference.

The pose of a robot is given in the local navigation coordinates system (see chapter 3.2.1.2). Consequently, the landing spot must be in the same frame.

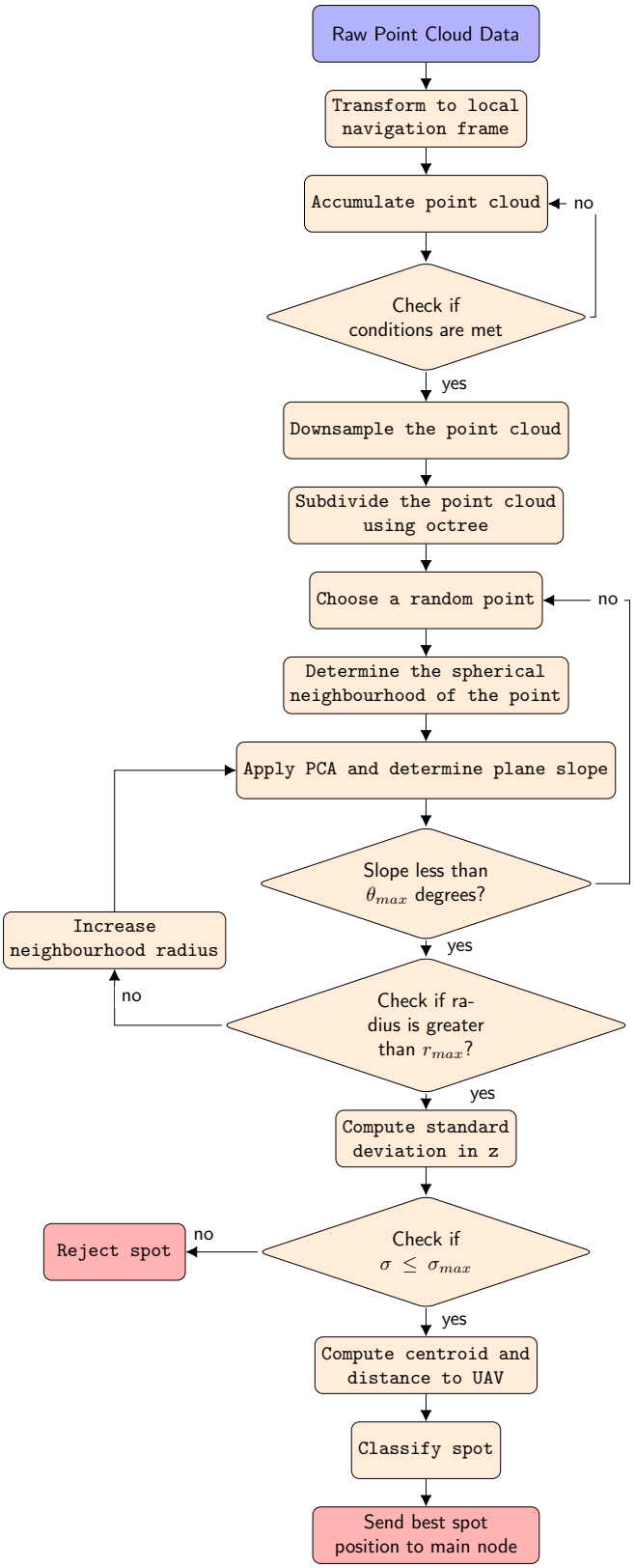


Figure 5.1: Flowchart of the developed algorithm.

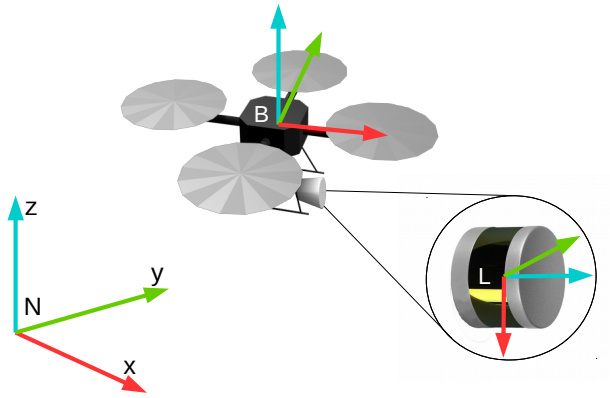
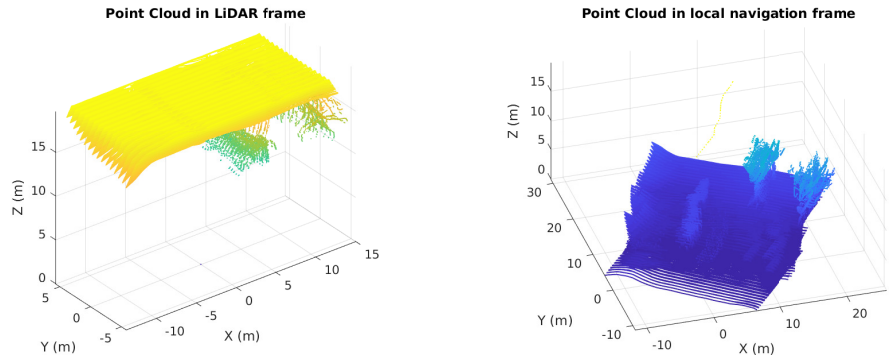


Figure 5.2: Frames of reference.

However, LiDARs report data in their own reference frame. Then, the first step of the method is to transform the point cloud from the LiDAR reference frame to the local reference frame. This procedure can be fulfilled by using transformation matrices (see chapter 3.2.3). Therefore, the relation between the frames is defined by equation 5.1.

$$\mathbf{p}^{\mathbf{n}}(t) = \mathbf{T}_{\mathbf{b}}^{\mathbf{n}}(t)\mathbf{T}_{\mathbf{L}}^{\mathbf{b}}\mathbf{p}^{\mathbf{L}}(t) \tag{5.1}$$

Considering a 3-D point $\mathbf{p}^{\mathbf{L}}(t)$ in the LiDAR reference frame, at instant t, the transformation matrix given by $\mathbf{T}_{\mathbf{L}}^{\mathbf{b}}$ transforms the point to the body reference frame. At this step, it is considered that the LiDAR is fixed to the robot, hence the $\mathbf{T}_{\mathbf{L}}^{\mathbf{b}}$ matrix not varying overtime. Finally, the $\mathbf{T}_{\mathbf{b}}^{\mathbf{n}}(t)$ matrix multiplied by $\mathbf{R}_{\mathbf{L}}^{\mathbf{b}}\mathbf{p}^{\mathbf{L}}(t)$ results in the data expressed in the local navigation reference frame.



(a) Point cloud in LiDAR frame. (b) Point cloud in the local frame.

Figure 5.3: Point cloud frame transformation.

Figure 5.3 shows a frame transformation of a point cloud. In figure 5.3a, the point cloud is provided by a LiDAR mounted in an UAV travelling 20 meters

high. Figure 5.3b shows the same point cloud transformed to the local navigation frame where the origin is set to the point $\mathbf{p_0} = (0, 0, 0)$ at the ground.

The developed algorithm does not apply the landing zone detection step for each LiDAR scan. This is due to some reasons: the vehicle may not have travelled a sufficient distance to need a new spot other than the take off point; in addition, the sensor may not return sufficient points of the environment. Therefore, the point cloud registered in the local frame is stored until it reaches the conditions to start the next step. The conditions defined are:

- Travelled distance: if the robot has traveled a long distance in relation to the last landing spot, it becomes necessary to find a new location, since, in an emergency scenario, the vehicle may not be able to reach the landing zone.
- Size of accumulate point cloud: a high number of points in the point cloud increases the execution time of search algorithms and the memory consumption of the onboard computer. For this reason, if the size of the accumulated cloud reaches a threshold, the algorithm starts the next step.

At the moment, this step is done considering only the vehicle pose. However, in future projects, it is worth consider the vehicle velocity.

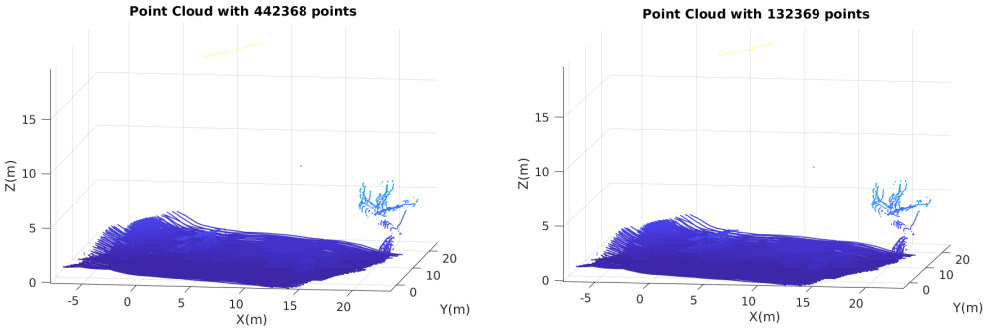
5.1.2 Point Cloud Downsampling

Accumulating the data until one of the necessary conditions are reached will increase computational effort. In order to obtain better performance in terms of execution time and memory consumption, it becomes necessary to perform the downsampling of the point cloud. Therefore, a Voxel Grid filter is applied. The filter takes a spatial average of the points in the cloud. A set of 3D volumetric pixels (voxel) grid with size v_{filter} is generated over the cloud and the points are approximated with their centroid.

Figure 5.4 shows the result of the downsampling. At first analysis, the point clouds displayed in figures 5.4a and 5.4b are almost identical. However, the downsample technique created a point cloud 3 times less in size. This method allows to decrease the number of points of the cloud and gives a point cloud with approximately constant density. Finally, the algorithm resets the original point cloud in order to free the memory.

5.1.3 Data Structuring and Neighbour Search

The next step is to determine the neighborhood of a point to perform the plane identification method. The process of detecting a plane in a point cloud is the most time-consuming process in the algorithm. Due to the large amount of data, this step needs to be optimized in order to obtain a real-time analysis. For



(a) Point cloud accumulated over time. (b) Point cloud downsampled.

Figure 5.4: Point cloud downsampling.

this reason, the point cloud is spatially structured using octree. Each internal node of the octree is subdivided into eight octants. By using a tree structure like an octree, the execution time of a search algorithm is considerably reduced [34].

There are several approaches to identify the neighbourhood of a point. In this case, the algorithm finds the spherical neighborhood of a randomly chosen point. Considering a point $p(x_p, y_p, z_p)$ in the point cloud, the neighbourhood N(p) of this point with radius r is determined by:

$$N(p) = \{\forall q : (x_q - x_p)^2 + (y_q - y_p)^2 + (z_q - z_p)^2 < r^2\} \tag{5.2}$$

where $q(x_q, y_q, z_q)$ is any point in the cloud.

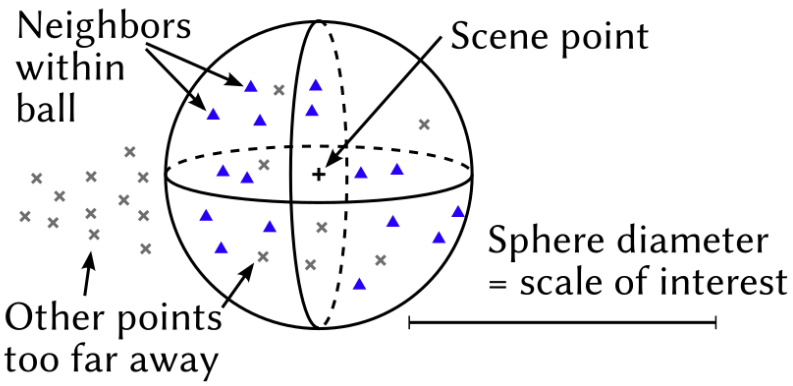


Figure 5.5: Spherical neighbourhood of a randomly chosen point (Adapted from [92]).

The minimum radius (r_{min}) of a plane is defined by the user as depends on sensor and environment characteristics. By applying the equation 5.2, all the points placed inside the sphere are considered as part of the neighbourhood.

Figure 5.5 shows a representation of this step: the blue triangles are part of the neighbourhood while the grey crosses are points that lie outside the sphere.

5.1.4 Plane Detection

After the previous step, it is possible to calculate a planar surface given the neighbourhood points. This is done satisfactorily by using the PCA algorithm. PCA applies an orthogonal transformation to map the data to a set of values called principal components. As explained in chapter 3.5.2, the principal componets are the eigenvector of the covariance matrix.

The eigenvectors determined with PCA serve as the three axes of the plane while the eigenvalues indicate the square sum of points deviations along the corresponding axis. Therefore, the eigenvector with the smallest eigenvalue represents the normal vector given by equation 3.48 and the points are bounded by the other two axes.

In this perspective, the slope of the plane is examined using the normal vector. The slope is the angle between the normal vector and the vertical vector $\hat{\mathbf{z}} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$ and is computed using equation 5.3:

$$\theta = \arccos\left(\hat{z}^T \mathbf{n}\right) \tag{5.3}$$

Consequently, a first evaluation can be done using the plane slope. If the resulting value is greater than the maximum slope (θ_{max}) permitted for the robot, the plane is rejected. Otherwise, the neighborhood radius used in the previous section is increased and the process is repeated. By doing this procedure, the method tries to find regions with different sizes that can be considered a landing spot. Figure 5.6 displays the effect of increasing the radius: the region detected in the first sphere is contained within the second; on the other hand, the points inside the third sphere are not part of a plane and are thereby rejected.

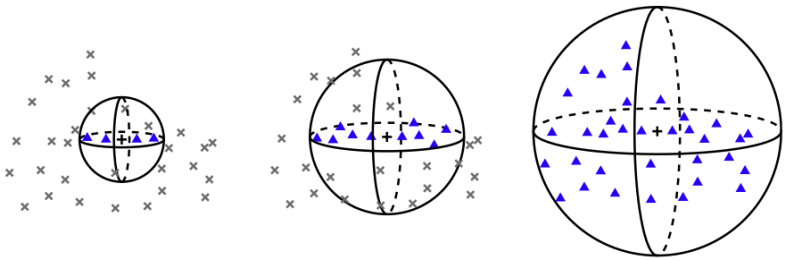


Figure 5.6: Different values for the radius of the sphere (Adapted from [92]).

The algorithm 5.1 describes the neighborhood and plane identification steps. The idea of the method is to detect planes for n_{points} random search points with increasing radius.

Algorithm 5.1 Algorithm for neighbourhood and plane identification steps.

Input: pointcloud downsampled
Output: point cloud cluster

for $j = 1$ to n_{points} **do**
  Select random point in cloud as the search point
  Declare radius and vectors of cloud indices
  float r_{min}, vector radiusInx
  While the plane is accepted, do radius search
  while planeBool = true **do**
    Start radius search;
    if ($radiusInx \geq n_{min}$) **then**
      Get points inside the neighbourhood;
      Start PCA;
      Compute plane parameters;
      Compute plane slope;
      if slope $\leq \theta_{max}$ **then**
        Increase radius;
      else
        planeBool = false
      end if
    end if
  end while
end for

In summary, only the planes that have the slope less or equal than θ_{max} are sent to the register stage. Table 5.1 describes the parameters used in the algorithm.

Table 5.1: The parameters used in the plane detection step.

Parameters	Description
n_{points}	Number of random points chosen from the cloud cluster
n_{min}	Minimum points used to fit a plane
r_{max}	Maximum radius considered for a plane
r_{min}	Minimum radius considered for a plane
θ_{max}	Maximum slope accepted for the drone

5.1.5 Registration and Classification

Given the algorithm presented in the previous section, the next step consists in evaluate the detected planes. In this perspective, many factors are considered to decide the best landing spot. Initially, the point cloud centroid is computed and is considered the spot centre. After this, each factor is computed individ-

ually. A grade (g_n) from 0 to 20 is computed for each parameter. It is worth noting that the degree of importance for each parameter depends on the operation. Consequently, each grade has a different weight previously defined by the user. Finally, each spot is classified regarding the following equation:

$$spot_{grade} = \frac{(g_1 \cdot w_1) + (g_2 \cdot w_2) + (g_3 \cdot w_3) + (g_4 \cdot w_4)}{20 \cdot (w_1 + w_2 + w_3 + w_4)} \tag{5.4}$$

Table 5.2 shows the parameters computed to classify the plane and their respective weights. In this stage, the planes that have standard deviation in the z-axis greater than the maximum standard deviation σ_{max} allowed are immediately rejected.

Table 5.2: The parameters that are analysed to classify the spot.

Parameters	**Weight**	**Description**
r_p	w_1	Spot radius
θ_p	w_2	Spot slope
σ_p	w_3	Standard deviation of the spot
d_v	w_4	Distance from the spot to the vehicle

Using these parameters, the algorithm evaluates the landing spot in terms of terrain roughness, vehicle stability when landed, obstacle clearance of a location and distance to the UAV.

Ultimately, the spots are stored and sorted from highest rate to smallest. In general, a spot quality depends on the vehicle trajectory. In this perspective, the stored spots are re-evaluated periodically. Figure 5.7 shows the functions that evaluate the four parameters. The function that evaluates the distance to the vehicle is only applied within a range of a minimum distance to a maximum distance predefined by the user. If the distance to the UAV is larger than the threshold, the lowest score is assigned. On the other hand, if the distance is less that the minimum distance, the highest score is assigned.

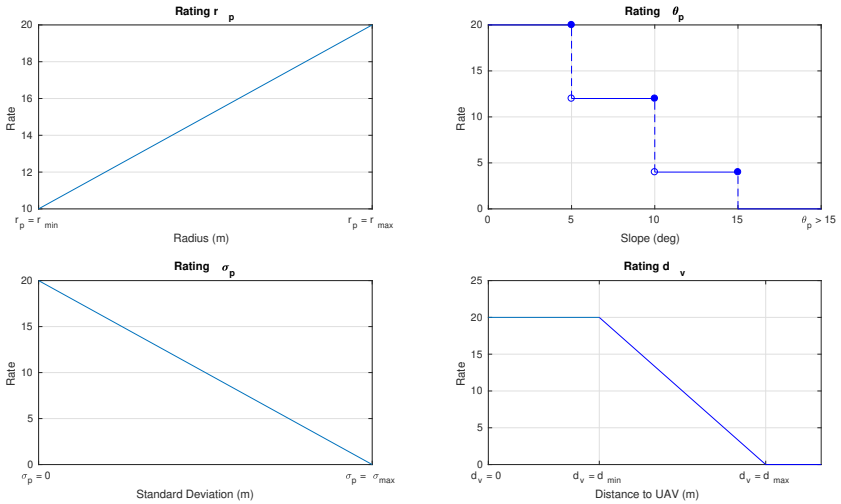


Figure 5.7: Rating function of the four parameters.

This page intentionally left blank.

Chapter 6

Implementation

In our proposal, data acquired from a spinning LiDAR mounted in an aerial vehicle is processed to extract information about the neighbouring environment, as expressed in chapter 4. This chapter exhibits the characteristics of the used UAV, specifying the key sensors needed to implement the algorithm. Moreover, useful libraries and adaptations realized during the development of the algorithm is presented.

6.1 UAV

The multirotor UAV STORK (shown in figure 6.1) is an autonomous aerial vehicle with six rotors designed to achieve real time data acquisition and processing efficiently. It has been used for several applications, including power lines inspection, mapping and SpilLess and MineHeritage projects. The robot was built allowing a modular payload assembly, i.e., the sensor can be replaced by others that supply different type of data and using the same frame.

Furthermore, the UAV can operate in both manual and autonomous modes. Currently, the low-level control of the UAV Stork is accomplished by a customized autopilot (INESC TEC Autopilot) while the high-level control is performed by an onboard computer ODROID-XU4[1] running Ubuntu 16.04 LTS[2] and ROS Kinetic Kame[3] distribution.

In terms of navigation, the UAV possess two IMUs and two GNSS receivers. In addition to the low cost sensors, STORK also has the high performance IMU STIM300[4] and the single-band GNSS receiver ComNav K501G that supports on-

[1] `https://wiki.odroid.com/odroid-xu4/odroid-xu4`

[2] `http://releases.ubuntu.com/16.04`

[3] `http://wiki.ros.org/kinetic`

[4] `https://www.sensonor.com/products/inertial-measurement-units/stim300`

board Real Time Kinematic (RTK). Generally, these setup of sensors succeeds in satisfying the requirements for many applications.

Regarding the perception assembly, that is, the sensors to extract information about the surrounding area, the UAV STORK has two visual cameras (Teledyne Dalsa G3-GC10-C2050[5] and FLIR Point- Grey CM3-U3-13S2C-CS[6]) and a 3-D spinning LiDAR sensor (Velodyne VLP-16[7]).

These sensors provide data that are used as input for processing algorithms in several modules, such as navigation, 3-D reconstruction. For instance, the Velodyne VLP-16 data is the input for the emergency landing spot detection algorithm.

Figure 6.1: STORK UAV.

6.1.1 Velodyne VLP-16

The Velodyne Puck, shown in figure 6.2, is the spinning LiDAR mounted in STORK. It provides a 360 degrees scan of the environment creating a 3-D point data. This sensor has 16 pairs of infra-red lasers and infra-red detectors that are fired rapidly to scan the surrounding area. This setup allows the sensor to acquire up to 300,000 data point per second [59]. Besides this, its principle of operation is based on the ToF methodology (see section 3.1).

There are three modes to handle the laser return. In the *Strongest* and *Last* modes, a single return is obtained, with the strongest signal being reported in the former, while the last signal is measured in the latter. On the other hand, in the *Dual* mode, both measurements are obtained.

[5] `http://www.teledynedalsa.com/en/products/imaging/cameras/genie-nano-gige`
[6] `https://www.flir.eu/products/chameleon3-usb3/`
[7] `https://velodynelidar.com/products/puck/`

Figure 6.2: Velodyne Puck LiDAR [93].

The sensor vertical Field of View (FoV) is 30 degrees where each beam of the 16 lasers is vertically spaced by 2 degrees. The horizontal FoV is 360 degrees with angular resolution depending on the sensor's rotation frequency. Table 6.1 displays the rotation frequency and their respective angular resolution. Furthermore, the measurement range is up to 100 m with $\pm$ 3 cm of accuracy.

Table 6.1: Velodyne VLP-16 horizontal angular resolution.

Frequency (Hz)	**Resoluiton** (°)
5	0.1
10	0.2
15	0.3
20	0.4

In addition to that, the firing sequence of the beams is organized according to table 6.2. This organization prevents interference between the beams.

The VLP-16 returns the 3-D points in data packets that are composed of a header, 12 data blocks, a timestamp and factory bytes. Figure 6.3 exposes the data packet of a single mode return. The data blocks are constructed with a two-byte flag, a two-byte azimuth and 32 three-byte data (two bytes for distance and on for intensity).

The data block contains the 3-D point in spherical coordinates. By using equations 3.1, 3.2 and 3.3, the Cartesian coordinates of the point can be computed, where the range and azimuth angle is obtained from the data packet and the elevation angle is given by table 6.2.

The LiDAR can also synchronize its timestamp with the Coordinated Universal Time (UTC). This trait allows the matching of the sensor data with a navigation system. In order to achieve the synchronization, it is necessary the use of an external GNSS that generates a pulse per second signal and a National Marine Electronics Association (NMEA) GPRMC message.

Table 6.2: Velodyne VLP-16 firing sequence.

Laser ID	**Vertical Angle** (°)
0	-15
1	1
2	-13
3	3
4	-11
5	5
6	-9
7	7
8	-7
9	9
10	-5
11	11
12	-3
13	13
14	-1
15	15

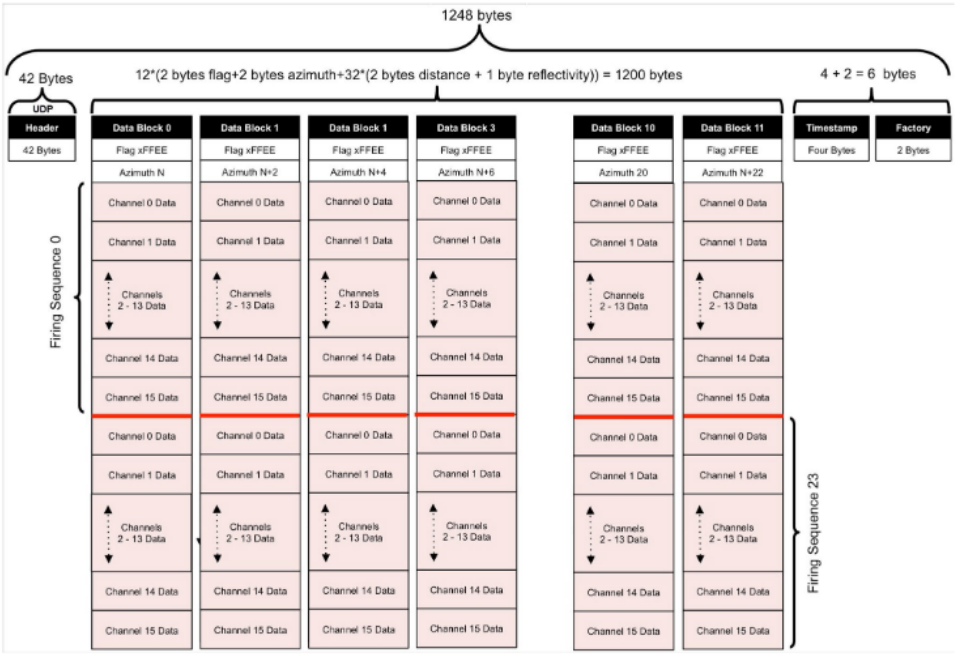


Figure 6.3: Velodyne VLP-16 data packets structure [59].

6.2 ROS Packages

As mentioned in the previous chapters, the project developed in this dissertation uses ROS as a communication middleware. One benefit of ROS is the vast number of packages already available to the user, from drivers for sensors to the implementation of algorithms for data processing. In addition, these packages are modular and can be modified according to the application.

Considering Velodyne VLP-16, there is the ***velodyne*** package that allows the connection with the device and the generation of point clouds from the received data. This package is subdivided into three: ***velodyne_msgs***, ***velodyne_driver*** and ***velodyne_pointcloud***. The ***velodyne_msgs*** package establishes the messages definitions for the Velodyne LiDARs.

The ***velodyne_driver*** corresponds to the ROS sensor driver. Hence, it is responsible for realizing the connection between the device and the computer, acquiring the data. Furthermore, it publishes the data packets in ROS messages of the type *velodyne_msgs/VelodyneScan*. Table 6.3 shows the data structure of a *VelodyneScan* message. The message is published in the topic **/velodyne_packets** and corresponds to a complete revolution of the sensor in the sensor reference frame.

Table 6.3: Velodyne Scan Message Structure.

Message	Type	Name
std_msgs/Header	uint32	seq
	time	stamp
	string	frame_id
velodyne_msgs/VelodynePacket	time	stamp
	uint8[1206]	data

The package ***velodyne_pointcloud*** subscribes to the **/velodyne_packets** topic, transform the data packets to a 3-D point cloud and subsequently publishing it as a *sensor_msgs/PointCloud2* message to the topic **/velodyne_points**. The data structure of a *PointCloud2* message is exposed in table 6.4.

This package creates a sparse point cloud, meaning that all points are valid, i.e., all points of the cloud are within an accepted range and angular interval. The *sensor_msgs/PointField* portion of the published point cloud contains the fields channels of the reported data, considering the table 6.5. The XYZ values correspond to the point in Cartesian coordinate system, intensity is the measured intensity of the beam reflection and, finally, ring encodes the vertical angle beam.

Table 6.4: PointClou2 Message Data Structure

Message	Type	Name
std_msgs/Header	uint32	seq
	time	stamp
	string	frame_id
	uint32	height
	uint32	width
sensor_msgs/PointField	string	name
	uint32	offset
	uint8	datatype
	uint32	count
	bool	is_bigendian
	uint32	point_step
	uint32	row_step
	uint8[]	data
	bool	is_dense

Table 6.5: Field channels of the PointCloud2 message published by the velodyne package.

Type	Name
float	x
float	y
float	z
float	intensity
uint16	ring

6.3 Point Cloud Library

Initially, the raw point cloud does not provide enough information about the terrain, which does not allow for qualitative assessment and detection of a landing zone. Thus, it is necessary to process the point cloud. For this reason, the Point Cloud Library (PCL) [94] is used. The PCL is an open-source library for image and point cloud processing in 2-D and 3-D. It is written in the C++ programming language and uses the Eigen[8] library to implement the mathematical operations.

Furthermore, PCL is split into several modular libraries, such as filters, segmentation, surface reconstruction and many others. These modules can be compiled separately, enabling the user to compile only the necessaries libraries, simplifying the project. During the development of this dissertation, the following modules were used:

[8] http://eigen.tuxfamily.org/index.php?title=Main_Page

- **Common**
  The commom library is the core module of PCL. It contains data structures and methods that are used by other libraries. Besides the point types defined, the PCA method is also implemented.

- **Filters**
  The filters library contains computational routines for filtering applications. The Voxel Grid Filter is implemented here.

- **Octree**
  The octree library is used to build a hierarchical data structure from the input point cloud. The implementation of an octree reduces memory allocation and computational effort.

- **Search**
  The search library contains algorithms to realize a nearest neighbours search given an octree or other data structure. In this library is implemented the radius search routine used in our algorithm.

6.4 Simulation Setup

Before applying the algorithm in experimental tests, it is interesting to evaluate the performance of the project in a simulation environment. Simulators, as the name implies, simulate real-world systems. They have the advantage of providing feedback to the user in terms of efficiency and accuracy in a controlled scenario. Therefore, it allows to understand the behaviour of the routine without experiment physically and identify unexpected events, such as segmentation faults, memory leaks and others.

6.4.1 Morse

For simulation purposes, it was chosen the Modular Open Robots Simulation Engine (MORSE) (see chapter 3). It is an open-source simulator that provides several features of interest for this dissertation, such as ROS supports and several simulation environments. In addition, MORSE has a generic 3-D LiDAR similar to the Velodyne VLP-16 that performs a 180 degree scan and publishes a point cloud message.

It was necessary to develop a Python script in order to configure the MORSE setup. The code in listing 6.1 shows an example of Python script that defines a quadrotor with two sensors: pose and LiDAR. The robot component is responsible to append the sensors to the 3-D model. Besides that, each sensor determine the communication middleware. Finally, an environment component is created using the path to a Blender model.

Listing 6.1 Python Script Example

```
from morse.builder import *

quadrotor = Quadrotor()
quadrotor.translate(0, 0, 20)
quadrotor.rotate(0, 0, 0)

pose = Pose()
quadrotor.append(pose)
pose.add_stream('ros')

velocity = RotorcraftVelocity()
quadrotor.append(velocity)

lidar = VLP16_180()
lidar.translate(0, 0, -0.25)
lidar.rotate(0, 1.57, 0)
lidar.frequency(1)
quadrotor.append(lidar)
lidar.add_service('ros')
lidar.add_stream('ros')

env = Environment('land-1/buildings_2')
env.create()
```

6.5 Developed Software

The software was organized in four ROS packages: ***tf_broadcaster***, ***landing_spot***, ***plane_detection*** and ***spot_register***. Each package contains a node that implements the algorithm explained in chapter 5.

The package ***tf_broadcaster*** contains a node that subscribes to the UAV pose topic **local_position/pose** with a *sensor_msgs/PoseStamped* message (see Table 6.6). Then,f2 the node broadcast a transformation matrix using the ROS packages ***tf2***[9]. The transformation matrix contains the data to transform a point from the LiDAR frame to the local navigation frame.

The ***landing_spot*** package has a node that subscribes to the **/velodyne_points** topic and listens to the tf message. Therefore, it is responsible to transform the point cloud to the new frame. In addition, the downsampling is applied in this package using the PCL library. The resulting point cloud is published to the **/landing_spot/points_downsampled** topic with a *PointCloud2* message. This node also subscribes to the **local_position/pose** and it is responsible to subscribe to the topic **/spot_register/spot** in which the best landing spot is published.

[9] http://wiki.ros.org/tf2

Table 6.6: Pose Message Structure.

Message	Type	Name
std_msgs/Header		header
geometry_msgs/Point	float64	x
	float64	y
	float64	z
geometry_msgs/Quaternion	float64	x
	float64	y
	float64	z
	float64	w

The ***plane_detection*** subscribes to the downsampled point cloud and realize data structuring and the PCA algorithm. Here a custom message is generated and published to the topic **/plane_detection/planes**. Table 6.7 shows the structure of the custom message. The cloud parameter represents all the points within the neighbourhood. The radius parameters is the neighbourhood radius and the coefficients represent the normal vector of the plane.

Table 6.7: Custom Plane Message Structure.

Type	Name
std_msgs/Header	header
sensor_msgs/PointCloud2	cloud
float32[]	coefficients
float64	radius

The ***spot_register*** package gets the custom message and applies the algorithm to classify the spots. The spots are stored and reclassified periodically during the operation. Finally, the best spot is published to the **/spot_register/spot** topic as a *geometry_msgs/PointStamped* message.

Figure 6.4 shows the communication architecture for the software. The blue ellipses represent the nodes while the red rectangles are the topics in which the message is published.

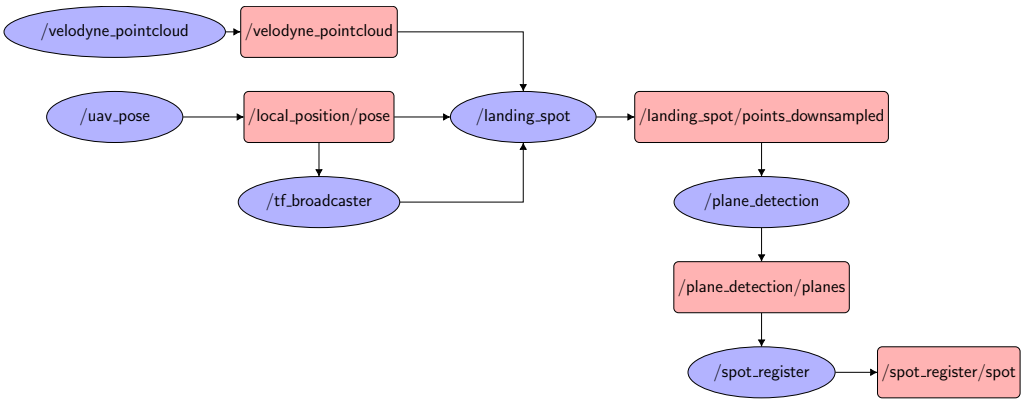


Figure 6.4: ROS communication architecture for the software.

Chapter 7

Results

This chapter presents the results obtained for the emergency landing spot detection algorithm. First, the method was tested in two simulation scenarios using MORSE. This allowed to carry out a previous analysis of the behavior of the algorithm and, in addition, to detect errors or software failures without testing on the physical robot. Furthermore, this chapter also presents results for two experimental datasets in a region with some obstructions.

Each section initially presents the parameters associated with the results obtained and concludes with an analysis of the performance of the algorithm.

7.1 Simulated Environment

In order to validate the algorithm, the simulation was performed on a computer with an Intel Core i7-6500U CPU @ 2.50GHz with 4 cores and 8 GB RAM, with a Linux 4.15.0-50-generic Ubuntu.

7.1.1 Environment I

The fist scenario is a standard MORSE environment, as shown in figure 7.1. It is important to note that this environment is relatively flat with only a few obstructions in the form of buildings. Thus, the objective of this simulation is to analyze the behavior of the algorithm in terms of time consumption, that is, the time that the method takes to perform some steps.

7.1.1.1 Parameters

During the description of the developed algorithm, in chapter 5, were introduced some parameters in whose its functionality relies on. Table 7.1 shows the values used in this simulation and their description. For each parameter

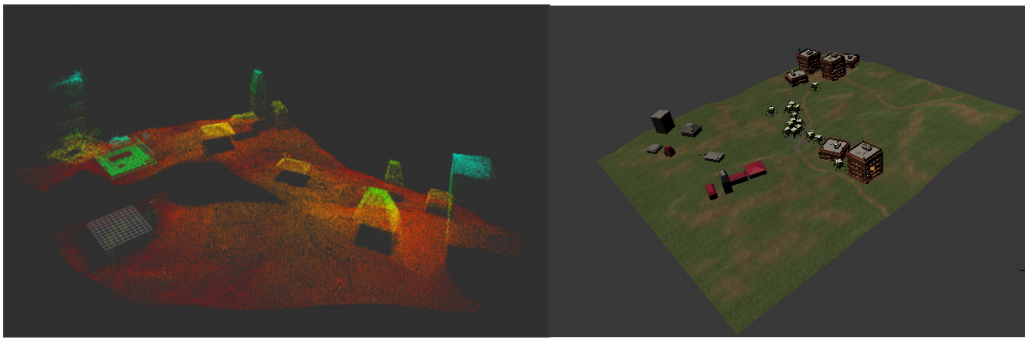


Figure 7.1: First scenario point cloud and Blender model.

listed is associated a brief description of its influence in the algorithm's behavior. In summary, the algorithm expects the vehicle to travel $tv_d = 10\ m$ from the last detected spot or that the accumulated point cloud has a maximum size of $c_{size} = 10^6$. Then, the Voxel Grid filter is applied to point cloud where each voxel has $v_{filter} = 5\ cm$ in size. The simulation was performed for different values of maximum radius and search points. It was considered that $n_{min} = 50$ points in the neighbourhood was enough to detect a spot. Finally, planes that have slope greater than $\theta_{max} = 15^\circ$ or standard deviation in z-axis greater than $\sigma_{max} = 0.20\ m$ were rejected.

Table 7.1: The parameters used in the first simulation. Each parameter is defined by the user.

Parameters	Unit	Values	Description
tv_d	m	10	Travelled distance from the latest landing spot
c_{size}	–	10^6	Maximum point cloud size to start downsampling
v_{filter}	cm	5	Size of the voxel
n_{points}	–	$[10, 20, 30, 50, 100]$	Number of random points chosen from the cloud cluster
n_{min}	–	50	Minimum points used to fit a plane
r_{max}	m	$[3, 4, 5, 10]$	Maximum radius considered for a plane
r_{min}	m	1	Minimum radius considered for a plane
θ_{max}	$degrees$	15	Maximum slope accepted for the drone
σ_{max}	m	0.20	Maximum standard deviation accepted for the plane

7.1.1.2 Results and Discussion

To obtain better control of the vehicle in the simulator, the position of the UAV along the z-axis was fixed to $z = 20\ m$. Thus, the trajectory of the UAV is illustrated in the figure 7.2. The position is given in the ENU frame coordinates and frame origin is located at point $\mathbf{p_0^n} = (0, 0, 0)$. Figure 7.3 shows the vehicle orientation during the simulation in which only UAV's yaw was controlled.

The first step of the algorithm is to perform the downsample of the point cloud. As explained in the chapter 3, voxels are generated along the cloud

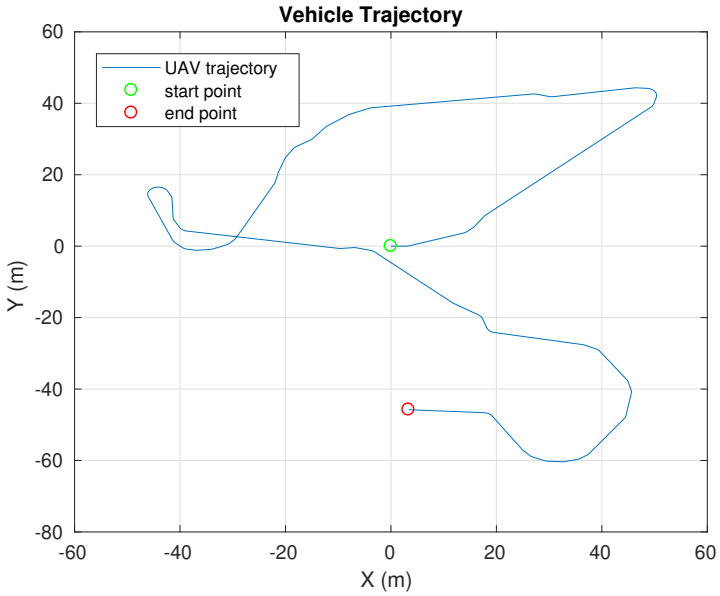


Figure 7.2: UAV trajectory for the first simulation.

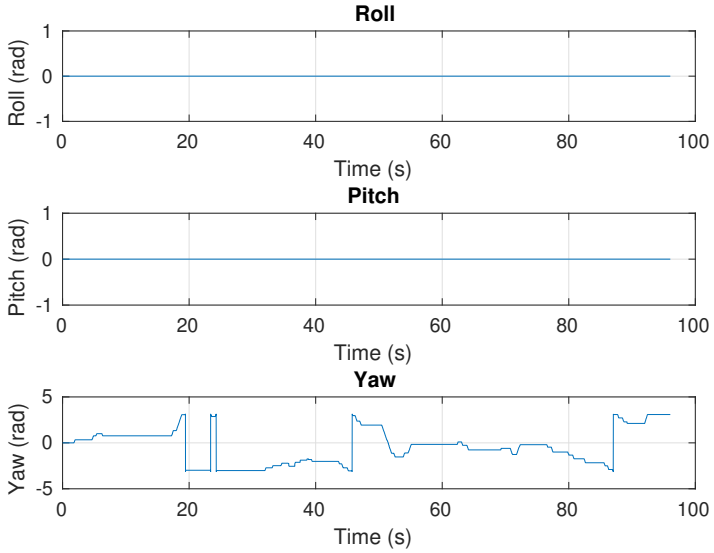


Figure 7.3: UAV orientation for the first simulation.

and the points are approximated according to the centroid of the points that lie within the voxel. The figure 7.4 shows the number of divisions along the three axis during one simulation.

Considering that the position on the z-axis had been fixed and the point clouds are in the local navigation reference frame, it is expected that the number of divisions along Z will be smaller than on the XY axes.

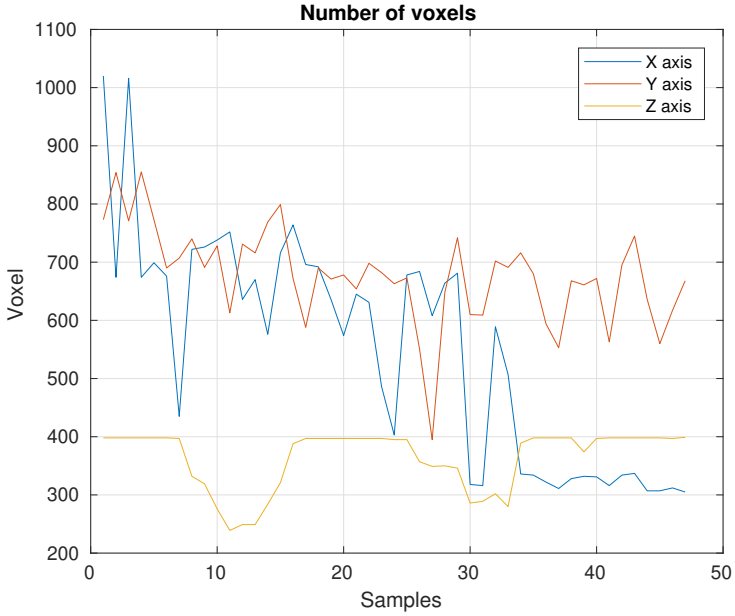
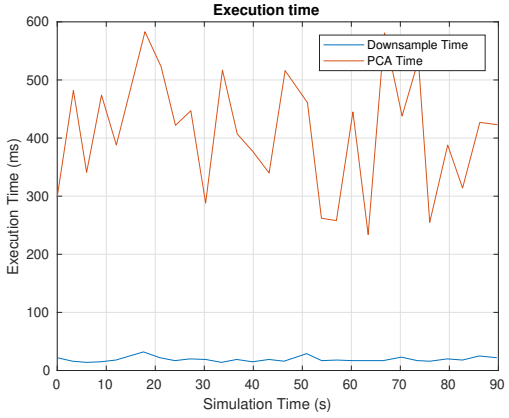


Figure 7.4: Number of divisions along the three axis.

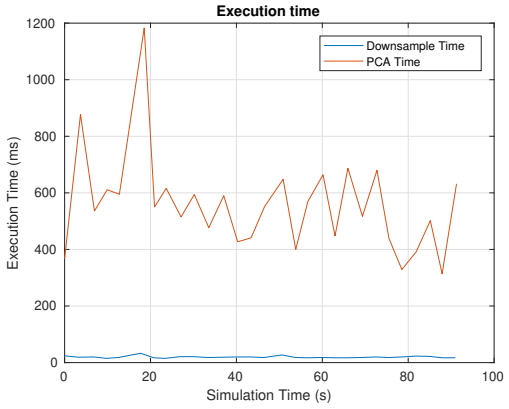
The next step of the software consists of applying the PCA algorithm to detect a plane. This step is the most time consuming of the developed project. Figure 7.5 shows a comparison of the execution time between the downsampling step and the PCA stage for several simulations with increased search points. The downsampling technique is relatively fast, taking about 20 ms. Moreover, increasing the number of search points implies more iterations for the PCA algorithm. Considering that the PCA is applied in a downsampled point cloud, downsampling the point cloud is justified because it limits the sample size.

In terms of detected spots, it is not possible to obtain a precise analysis of the performance of the algorithm in this simulation. This is due to the fact that the environment has many regions to land. However, it is possible to determine from the results in the figure 7.6 that, during the simulation, some spots were chosen (the red crosses), while other spots (black crosses) were not chosen. The reason for this is the effect of the functions that classify the spots.

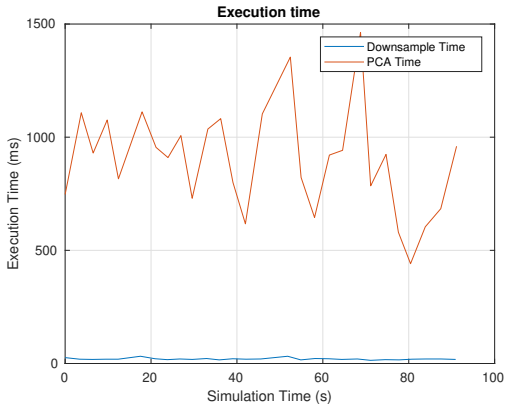
Analysing figure 7.5 and 7.6, it is possible to conclude that there is a trade-off between detected spots and the number of points sought. By increasing the



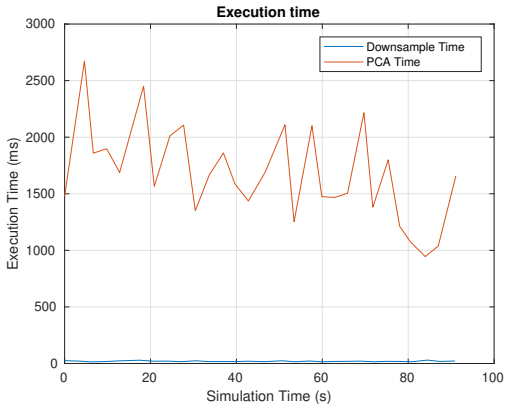
(a) Time for 20 search points and maximum 3 m for radius.



(b) Time for 30 search points and maximum 3 m for radius.



(c) Time for 50 search points and maximum 3 m for radius.



(d) Time for 100 search points and maximum 3 m for radius.

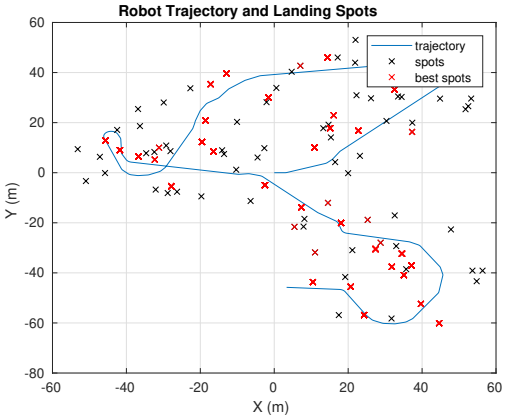
Figure 7.5: Comparison between Downsampling and PCA execution time for the first simulation.

number of points searched, more spots are detected, however, the execution time increases considerably.

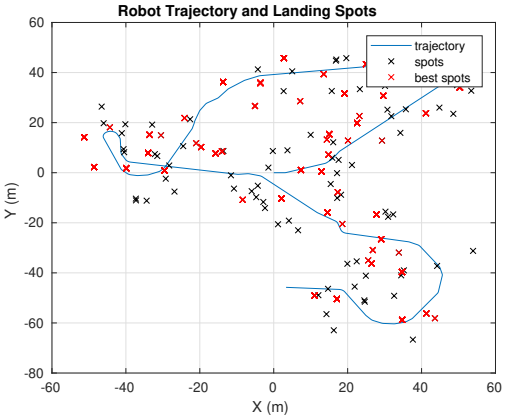
Table 7.2 summarizes the results for the simulation. Besides this, it presents the results for the simulation with the a fixed search points, but increasing maximum radius. The PCA runtime increases rapidly to a small increase in radius. This implies that the value chosen for the neighborhood radius has greater weight in this step.

7.1.2 Environment II

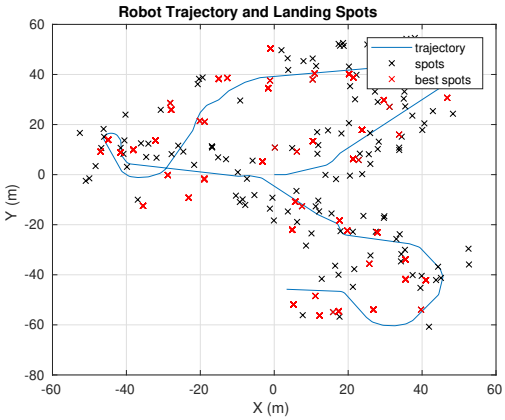
In order to test the algorithm in a bad scenario, it was developed in Blender the environment shown in figure 7.7. The set consists of 6 rectangular surfaces that represent the desired landing spots surrounded by a mountain like structures. Considering the environment and the dimension of each surface, the sim-



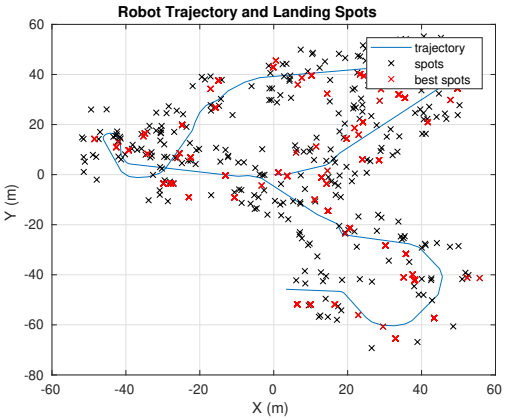
(a) Spots detected for 20 search points and maximum 3 m for radius.



(b) Spots detected for 30 search points and maximum 3 m for radius.



(c) Spots detected for 50 search points and maximum 3 m for radius.



(d) Spots for 100 search points and maximum 3 m for radius.

Figure 7.6: Spots detected for the first scenario with different parameters.

Table 7.2: Results for the simulation of the first case

Search Points (#)	Radius (m)	Downsample Mean Time (ms)	PCA Mean Time (ms)	Rejected Planes	Accepted Planes	Chosen Spots
10	3	20.54	84.73	184	146	27
20	3	19.07	408.00	495	65	41
30	3	19.55	556.93	754	116	54
50	3	20.00	898.35	1197	203	56
100	3	19.621	1674.37	2499	401	70
10	4	21.50	400.96	36	244	62
10	5	23.18	802.21	35	245	82
10	10	22.14	4583.9	24	156	7

ulations were realized for 4 different search points.

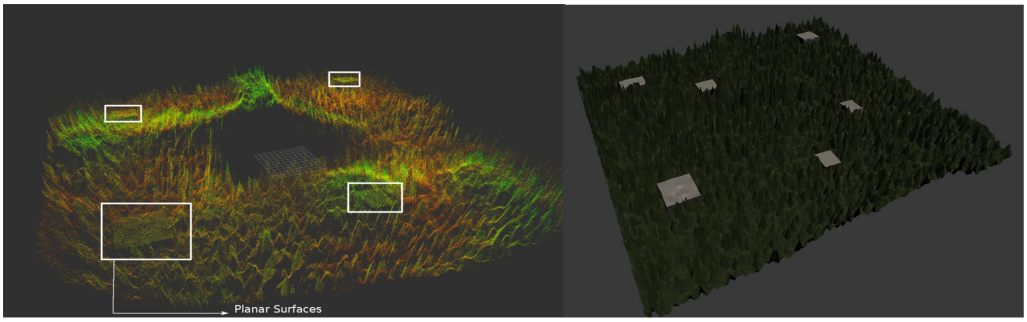


Figure 7.7: Second scenario point cloud and Blender model.

7.1.2.1 Parameters

Table 7.3 shows the values used in this simulation and their description. This parameters differ from the first simulation due to the maximum value allowed for the standard deviation in z, the number of search points tested and the maximum spot radius. The standard deviation is $\sigma_{max} = 0.15\ m$.

Table 7.3: The parameters used in the second simulation.

Parameters	Unit	Values	Description
tv_d	m	10	Travelled distance from the latest landing spot
c_{size}	–	10^6	Maximum point cloud size to start downsampling
v_{filter}	cm	5	Size of the voxel
n_{points}	–	[20, 30, 50, 100]	Number of random points chosen from the cloud cluster
n_{min}	–	50	Minimum points used to fit a plane
r_{max}	m	2	Maximum radius considered for a plane
r_{min}	m	1	Minimum radius considered for a plane
θ_{max}	$degrees$	15	Maximum slope accepted for the drone
σ_{max}	m	0.15	Maximum standard deviation accepted for the plane

7.1.2.2 Results and Discussion

The trajectory of the UAV is illustrated in the figure 7.8. The position is given in the ENU frame coordinates and frame origin is located at point $\mathbf{p_0^n} = (0, 0, 0)$. Figure 7.9 shows the vehicle orientation during the simulation in which only robot's yaw was controlled.

Regarding the number of voxels generated at the downsampling stage, figure 7.10 shows the result obtained. It is worth mention that simulator LiDAR provides zero data near the edge of the environment. Therefore, this explains the sudden drop in the number of division along the xy axes.

Figure 7.11 shows a comparison of the execution time between the downsampling step and the PCA stage for several simulations with increased search

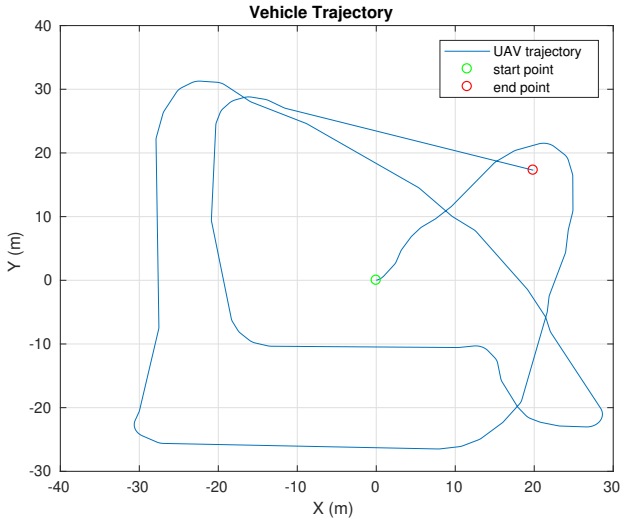


Figure 7.8: UAV trajectory for the second simulation.

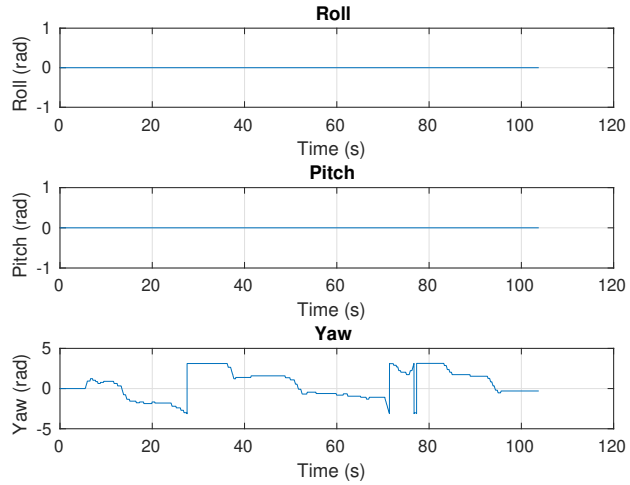


Figure 7.9: UAV orientation for the second simulation.

points. For the simulations in figures 7.11a and 7.11b, the runtime of the PCA algorithm were similar to the downsampling stage. The reason for this is that the plane are immediately rejected. Since the environment has a high variance in z, the angle between the normal vector calculated by the PCA and the vertical vector ($\hat{\mathbf{z}} = [0, 0, 1]^T$) is generally greater than the maximum allowed. As a result, the algorithm does not perform many iterations for the same search point.

Regarding the detected spots, figure 7.12 shows the results obtained and figure 7.13 displays the progression of the grades of the best spots during the simulation. The landing surfaces are the red rectangles and the black crosses are the

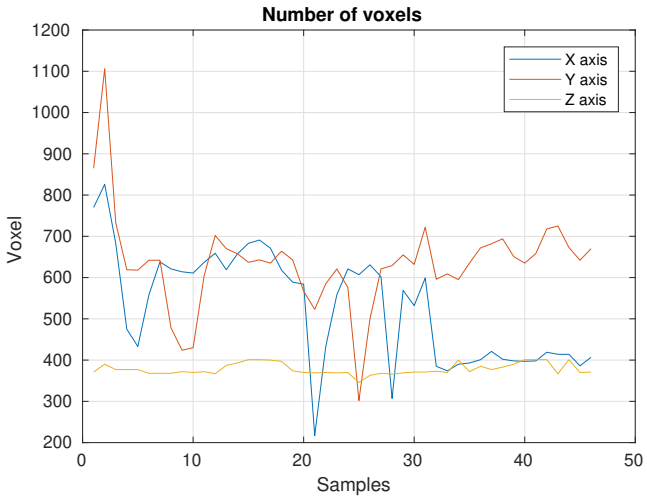


Figure 7.10: Number of divisions along the three axis for the second simulation.

detected spots. However, the algorithm was not able to find a landing spot in all planes. In this scenario, one drawback of the algorithm is that if the random search point falls near a high deviation zone like the surface edges, the algorithm may discard the plane or detect a spot outside the surface such as the result for 100 search points (figure 7.12d).

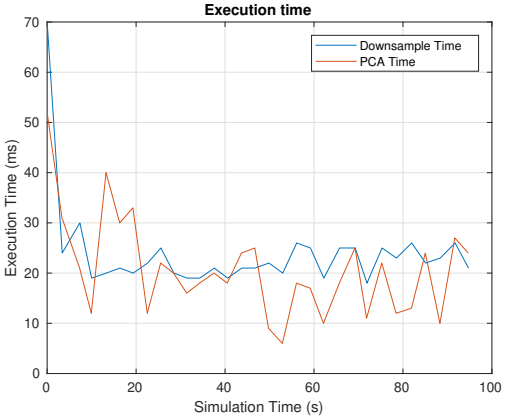
For a better understanding of how the best spot varies during the simulation, figure 7.14 displays the graphs of the best spot grade and the grades of each detected spot for the case with 50 search points. Furthermore, the instant of time that a spot is detected or is considered the best spot is marked on the graph. It was observed that the best spot varies with the distance from the vehicle.

Table 7.4 summarizes the results for the second simulation. The PCA mean runtime is smaller than the downsampling mean time in the first test. The difference between rejected and accepted plans increased considerably as expected. After this simulation, the following question was raised:

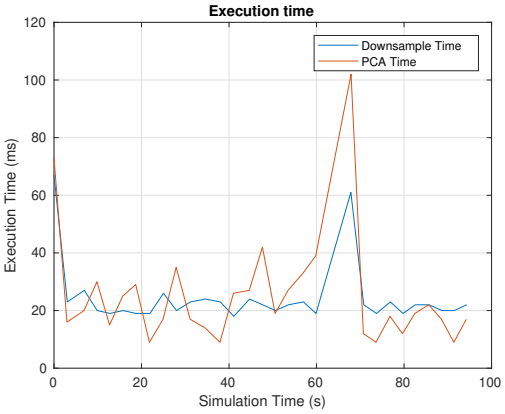
Table 7.4: Results for the simulation of the second case.

Search Points (#)	Downsample Mean Time (ms)	PCA Mean Time(ms)	Rejected Planes	Accepted Planes	Chosen Spots
20	23.77	20.64	278	22	3
30	24.27	25.30	566	54	5
50	23.57	52.43	1362	137	5
100	23.97	119.63	2712	288	8

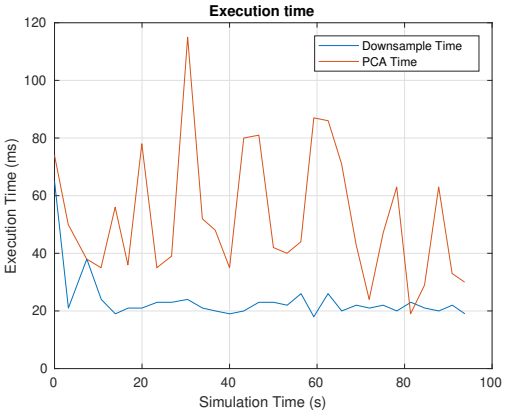
"***What is the influence of the voxel size on the other steps of the algorithm?***"



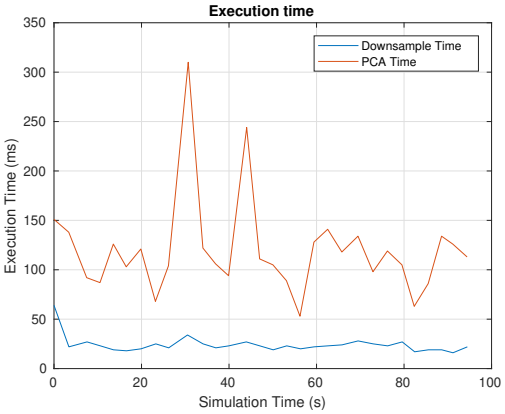
(a) Execution time for 20 search points and maximum 2 m for radius.



(b) Execution time for 30 search points and maximum 2 m for radius.



(c) Execution time for 50 search points and maximum 2 m for radius.

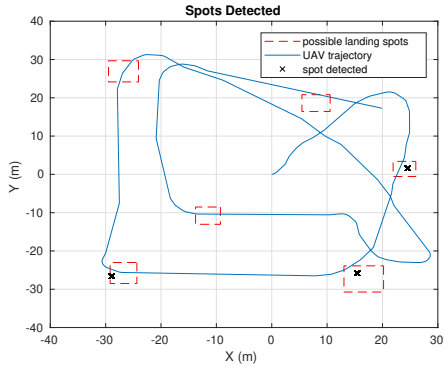


(d) Execution time for 100 search points and maximum 3 m for radius.

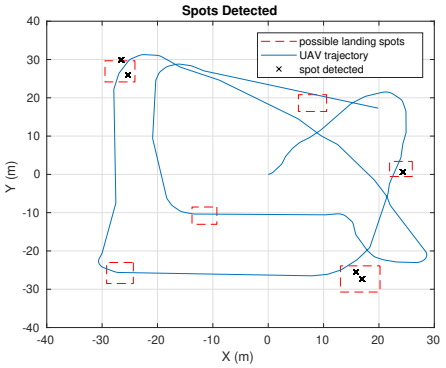
Figure 7.11: Comparison between Downsampling and PCA execution time for the second simulation.

Therefore, new simulations with different values for the size of the voxel were carried out to answer this question. Table 7.5 presents the results obtained. It was chosen the number of search points $n_{points} = 20$.

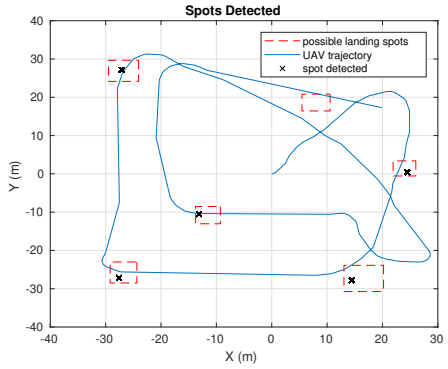
As the value of the voxel increases, the size of the new point cloud decreases considerably. This result shows that it is necessary to adjust the variable correctly, as it can lead to a great loss of information about the environment. In addition, after 1 m of voxel size, the algorithm stopped detecting landing points. In terms of the performance of the PCA, the increase in size did not reflect a major change in the execution time. However, the amount of detected planes increased, indicating that the downsampled point cloud did not present many obstructions as it should.



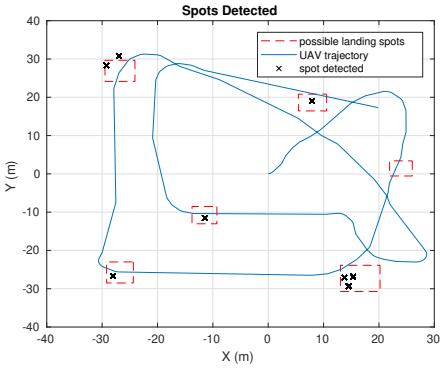
(a) Spots detected for 20 search points and maximum 2 m for radius.



(b) Spots detected for 30 search points and maximum 2 m for radius.



(c) Spots detected for 50 search points and maximum 2 m for radius.



(d) Spots detected for 100 search points and maximum 2 m for radius.

Figure 7.12: Spots detected for the second scenario with different parameters.

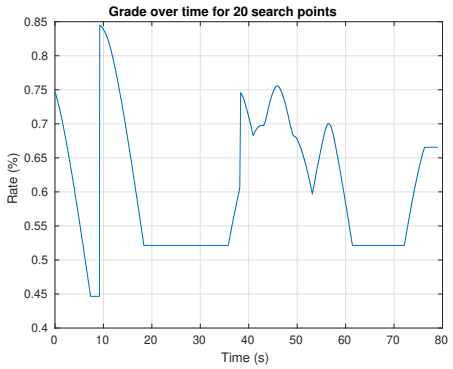
Table 7.5: Results for several values of voxel size.

Voxel size (m)	Mean Point Cloud Size	Mean Downsampled Point Cloud Size	Downsample Mean Time (ms)	PCA Mean Time (ms)	Accepted Planes	Detected Spots
0.05	183278.6	70022.4	15.507	18.907	22	3
0.10	183278.6	32146.2	15.7388	9.155	78	7
0.5	183278.6	5735.2	12.390	9.606	132	8
1.0	183278.6	1541.5	11.733	10.565	210	0

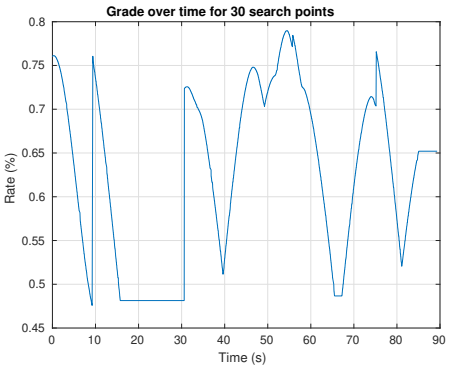
7.2 Experimental Dataset

The experimental dataset corresponds to the staircase of the Monastery of Tibães, Braga, Portugal. The stairway consists of several water fountains that are intertwined by stairs. This area has numerous trees that hinder the navigation and localization of the UAV and are considered obstacles for the detection of landing points.

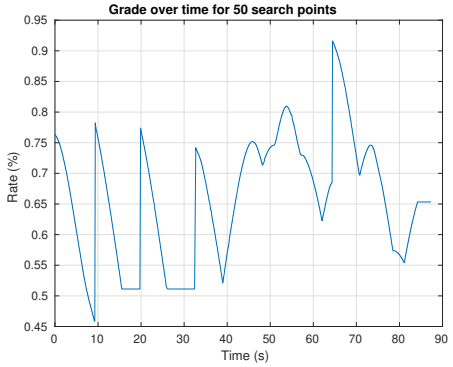
Two experiments were realized to assemble this dataset. First, the vehicle started by mapping a field of vegetation near the staircase (see figure 7.15a).



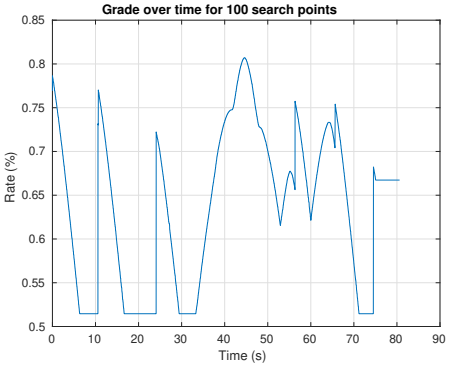
(a) Best spot grade for 20 search points and maximum 2 m for radius.



(b) Best spot grade for 30 search points and maximum 2 m for radius.

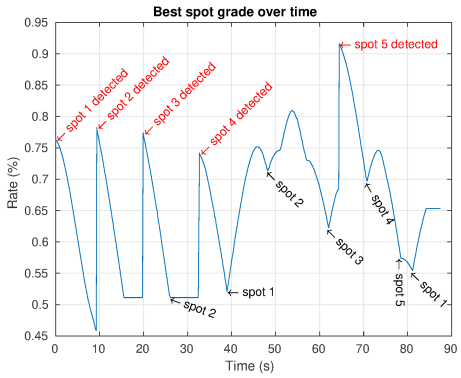


(c) Best spot grade for 50 search points and maximum 2 m for radius.



(d) Best spot grade for 100 search points and maximum 2 m for radius.

Figure 7.13: Best spots grades for the second scenario with different parameters.



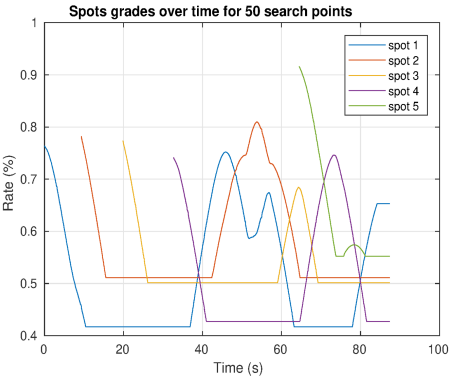


Figure 7.14: Grades of the best spot and each spot detected for 50 search points.

After that, he traveled through the different levels of the stairway. The second experiment was done at the Monastery building. The second experiment was carried out on the fence of the Monastery (figure 7.15b). This region consists mainly of buildings and vegetation. It's an open area with more landing spots.

(a) Monastery of Tibães stairway.

(b) Monastery of Tibães fence.

Figure 7.15: Monastery of Tibães [95].

7.2.1 Parameters

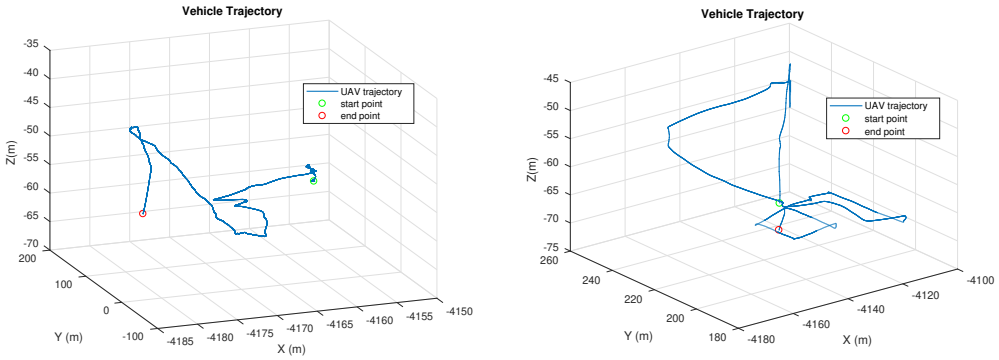
Table 7.6 shows the values used in this dataset and their description. It was considered that $n_{min} = 50$ points in the neighbourhood was enough to detect a plane and the maximum neighbourhood radius is $r_{max} = 4\ m$. Finally, planes that have slope greater than $\theta_{max} = 15°$ or standard deviation in z-axis greater than $\sigma_{max} = 0.10\ m$ were rejected.

Table 7.6: The parameters used in the dataset.

Parameters	Unit	Values	Description
c_{size}	–	30000	Maximum point cloud size to start downsampling
n_{points}	–	5	Number of random points chosen from the cloud cluster
n_{min}	–	50	Minimum points used to fit a plane
r_{max}	m	4	Maximum radius considered for a plane
r_{min}	m	2	Minimum radius considered for a plane
θ_{max}	*degrees*	15	Maximum slope accepted for the drone
σ_{max}	m	0.05	Maximum standard deviation accepted for the plane

7.2.2 Results and Discussion

Figure 7.16 shows both UAV's trajectories during the flight at the stairway and the monastery, respectively. The trajectories are represented in the local navigation frame considering the ENU system. Furthermore, the origin of the frame is located at a base station in the city of Braga.



(a) UAV trajectory for the first part of the dataset.

(b) UAV trajectory for the second part of the dataset.

Figure 7.16: UAV trajectories during the mission.

By using the ***rviz***[1] package, it was possible to view some detected spots in the raw point cloud provided by the Velodyne VLP-16. Figure 7.17,7.18,7.19 and 7.20 show some screenshots of rviz and the image taken from the camera. The red lines represent the centre of a plane detected during the PCA stage. The small red lines are detected spots that were not chosen as the best landing zone. It is worth noting that some of the detected landing spots are part of a larger spot. Besides the fact that the search points are chosen at random, another reason that justifies this behavior is the need for the maximum radius not to be too large so that the plan detection algorithm does not consume much time, as previously demonstrated by the simulations.

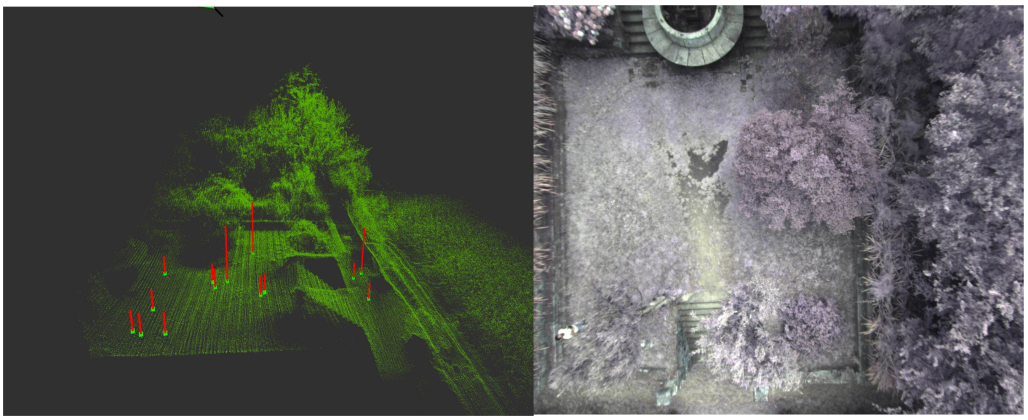


Figure 7.17: Point cloud, spots and camera image near a water fountain.

Figure 7.21 displays the progression of the rating during the dataset. The result obtained illustrates the effect of the weights in the spot classification stage.

[1] `http://wiki.ros.org/rviz`

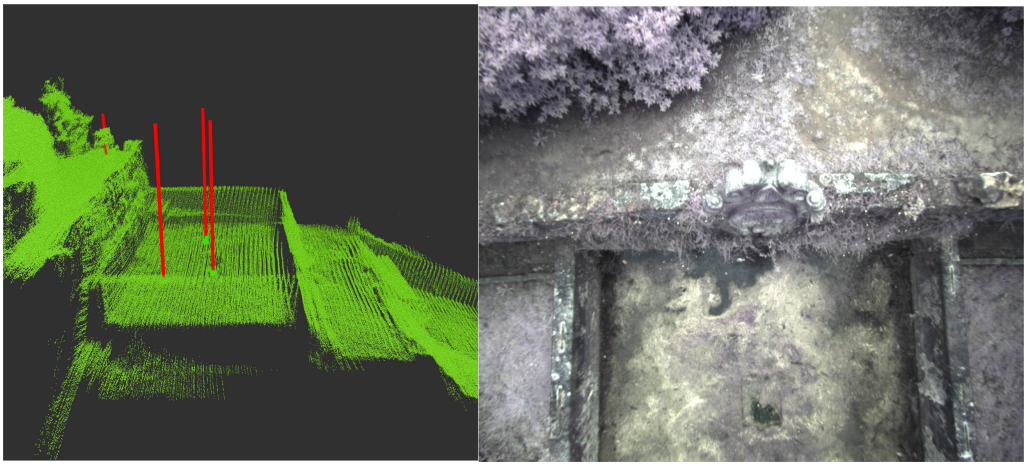


Figure 7.18: Point cloud, spots and camera image on the stairway.

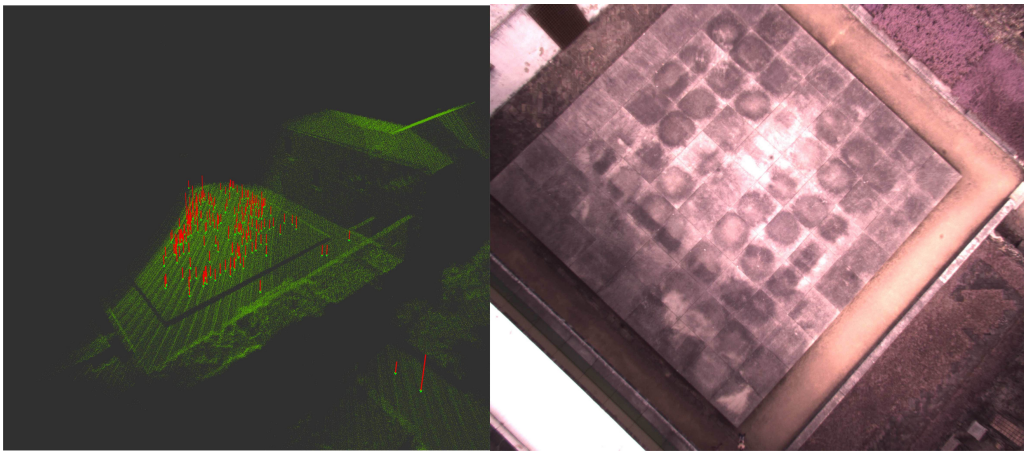


Figure 7.19: Point cloud, spots and camera image near a building.

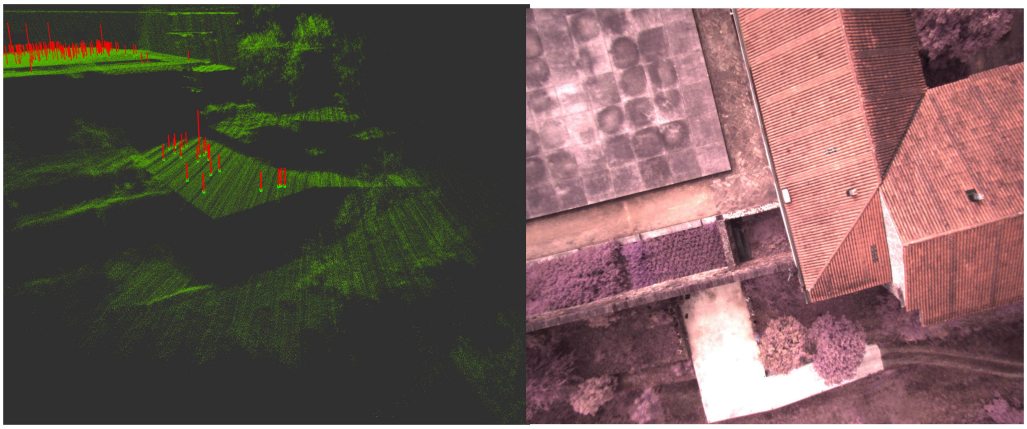


Figure 7.20: Point cloud, spots and camera image near entrance.

For instance, since the maximum allowed value for the standard deviation is very restrictive, relatively safe zones can be chosen as the best spot even with a low score.

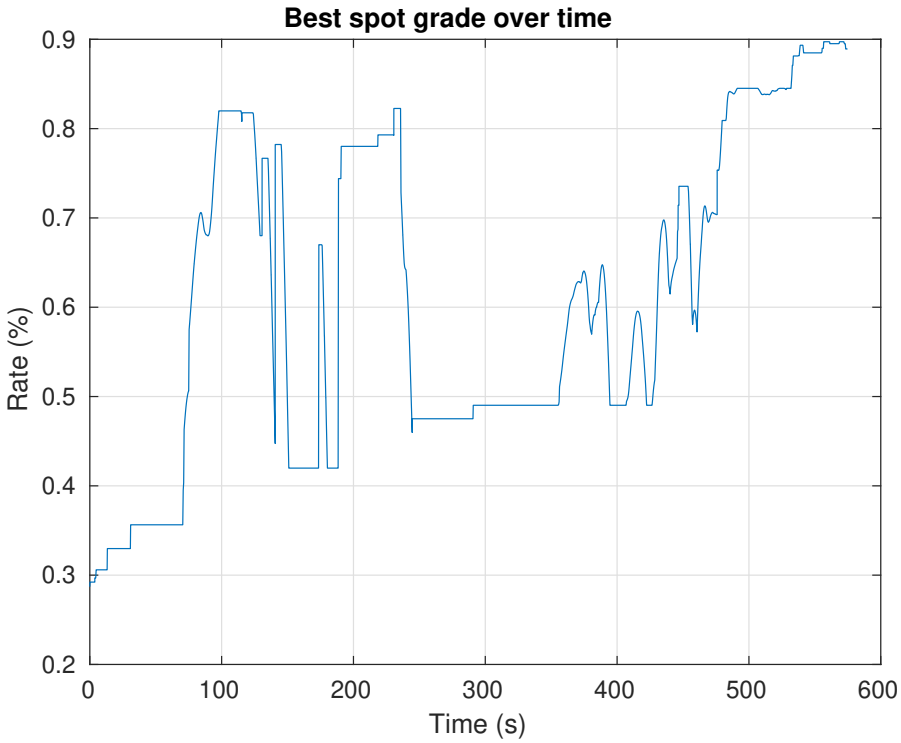


Figure 7.21: Rating of the best spot for the dataset.

Similarly to the simulation of the second environment, the performance of the algorithm was evaluated for several sizes of voxels. Thus, table 7.7 displays the results. The results obtained are similar to the simulation in relation to the loss of information with the increase of the voxel. However, a considerable reduction in the execution time of the PCA stage was observed. One explanation is that the dataset has more landing zones when compared to the second simulated environment. Therefore, many iterations are performed in the PCA step, unlike the simulation where there are only a few planar surfaces. Besides this, the number of detected spots reduces with the increasing voxel.

Table 7.7: Results for several values of voxel size in the dataset.

Voxel size (m)	Mean Point Cloud Size	Mean Downsampled Point Cloud Size	Downsample Mean Time (ms)	PCA Mean Time (ms)	Accepted Planes	Detected Spots
0.05	31155.9	23131.5	5.333	55.035	2206	372
0.10	31155.9	13945.4	4.493	39.390	1703	319
0.5	31155.9	1996.8	3.695	4.838	1440	302
1.0	31155.9	538.9	3.261	3.177	565	37

Chapter 8

Conclusions and Future Work

This dissertation has addressed the development of an algorithm capable of detecting, evaluating and selecting reliable emergency landing spots in real-time. The incoming data for processing is provided by a LiDAR sensor mounted in a multirotor UAV. Its outputs and performance were evaluated using simulation and experiment dataset. The environments evaluated contain several types of structures, vegetation. The main characteristic of our system is a geometric approach to determine the suitability of potential landing areas given LiDAR range data.

Analysing the results presented in chapter 7, the algorithm was able to fulfill the desired objectives. First, the system downsamples the point cloud quickly, then is able to detect a planar surface and segment the downsampled point cloud to a new one containing only useful data. The clustered cloud are classified regarding their size, slope, distance to the vehicle and terrain roughness. The experimental dataset shows that the detected points are reliable. Furthermore, the whole process is suitable for being adapted in real-time operation.

The algorithm was evaluated in different simulated environments. In the first simulation, we present results in an urban area. The system was able to detect many landing spots. The results for the computation time were presented for different parameters. It was observed that given certain values for the parameters, the results were adequate. Conversely, the algorithm was not able to identify all the spots for the second case. It was demonstrated that in a bad environment i.e, with few landing spots, the system is able to detect some of them, but it is necessary to reduce the rigidity of the parameters.

To the best of author's knowledge, the contribution of this dissertation is to apply a Voxel Grid Filter for the compression of the point cloud. This step allows to considerably decrease the number of points, while maintaining an adequate representation of the environment. In addition, it is not necessary to know the

region beforehand or to store the processed point cloud. Only the distance to the vehicle varies with the time of the analyzed parameters.

Although the concept of the algorithm was validated with the used dataset, it still needs to be submitted to others with different conditions. For instance, it is important to analyse the performance of the algorithm in environments with false positives, i.e., water surfaces, snow. These regions can reflect the light beams of the LiDAR and the algorithm would consider as regions with slope equal to zero, therefore, it would be chosen as a landing point.

Furthermore, the work realized during the preparation of this dissertation resulted in two papers. The paper *Emergency Landing Spot Detection for Unmanned Aerial Vehicle* was published on the ROBOT'2019: Fourth Iberian Robotics Conference, in Porto, and the paper *Survey of Approaches for Emergency Landing Spot Detection with Unmanned Aerial Vehicles* was published to the 23rd edition of International Conference series on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR 2020), in Moscow, Russia.

In terms of future work of this project, it is worth to realize more experiments in order to evaluate the effect of weights in the registration step, so that the choosing process can be optimized. Moreover, the trajectory to the landing spot could be computed and used as another weighting factor to qualify the spot. For instance, the path to a landing point may require more power to be reached. Therefore, the remaining battery power can also be considered to classify the spot.

Bibliography

[1] Kadir Alpaslan Demir, Halil Cicibas, and Naiz Arica. Unmanned aerial vehicle domain: Areas of research. *Defence Science Journal*, 65(4), 2015. [Quoted on p. 1]

[2] Chun Fui Liew, Danielle DeLatte, Naoya Takeishi, and Takehisa Yairi. Recent developments in aerial robotics: A survey and prototypes overview. *arXiv preprint arXiv:1711.10085*, 2017. [Quoted on p. 1]

[3] Gaurav Singhal, Babankumar Bansod, and Lini Mathew. Unmanned aerial vehicle classification, applications and challenges: A review. 2018. [Quoted on p. 1]

[4] Autonomous Systems Laboratory. FALCOS. `https://archive.osvaldosousa.com/lsa/falcos/index.html`. Accessed on Mar. 16, 2020. [Quoted on p. 2]

[5] INESC TEC. OTUS. `http://www.strongmar.eu/site/otus-91,`. Accessed on Feb. 28, 2020. [Quoted on p. 2]

[6] FESTO. BionicOpter. `https://www.festo.com/group/en/cms/10224.htm`. Accessed on Mar. 16, 2020. [Quoted on p. 2]

[7] Alpha Unmanned Systems. Alpha 800 UAV. `https://alphaunmannedsystems.com/alpha-800-uav/`. Accessed on Feb. 28, 2020. [Quoted on p. 2]

[8] Pedro Sousa, André Ferreira, Miguel Moreira, Tiago Santos, Alfredo Martins, André Dias, J Almeida, and E Silva. Isep/inesc tec aerial robotics team for search and rescue operations at the eurathlon challenge 2015. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 156–161. IEEE, 2016. [Quoted on p. 1, 5]

[9] Geert De Cubber, Daniela Doroftei, Daniel Serrano, Keshav Chintamani, Rui Sabino, and Stephane Ourevitch. The eu-icarus project: developing assistive robotic tools for search and rescue operations. In *2013 IEEE international symposium on safety, security, and rescue robotics (SSRR)*, pages 1–4. IEEE, 2013. [Quoted on p. 1]

[10] Amazon.com Inc. Amazon Prime Air. `https://www.amazon.com/primeair`. Accessed on Feb. 11, 2020. [Quoted on p. 1]

[11] Naser Hossein Motlagh, Miloud Bagaa, and Tarik Taleb. Uav-based iot platform: A crowd surveillance use case. *IEEE Communications Magazine*, 55(2):128–134, 2017. [Quoted on p. 1]

[12] F Azevedo, A Dias, J Almeida, A Oliveira, A Ferreira, T Santos, A Martins, and E Silva. Real-time lidar-based power lines detection for unmanned aerial vehicles. In *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 1–8. IEEE, 2019. [Quoted on p. 1]

[13] Mohammadreza Aghaei, Francesco Grimaccia, Carlo A Gonano, and Sonia Leva. Innovative automated control system for pv fields inspection and remote control. *IEEE Transactions on Industrial Electronics*, 62(11):7287–7296, 2015. [Quoted on p. 1]

[14] INESC TEC. TURTLE II. `http://www.strongmar.eu/site/turtle-ii-118`, . Accessed on Feb. 28, 2020. [Quoted on p. 4]

[15] INESC TEC. EVA. `http://www.strongmar.eu/site/eva-112`, . Accessed on Feb. 28, 2020. [Quoted on p. 4]

[16] INESC TEC. ROAZ II. `http://www.strongmar.eu/site/roaz-ii-115`, . Accessed on Feb. 28, 2020. [Quoted on p. 4]

[17] INESC TEC. STORK I. `http://www.strongmar.eu/site/stork-i-113`, . Accessed on Feb. 28, 2020. [Quoted on p. 4]

[18] Gabriel Loureiro, André Dias, and Alfredo Martins. Survey of approaches for emergency landing spot detection with unmanned aerial vehicles. In *CLAWAR2020 conference*, pages 129–136, 2020. [Quoted on p. 7]

[19] Andrew E Johnson, Allan R Klumpp, James B Collier, and Aron A Wolf. Lidar-based hazard avoidance for safe landing on mars. *Journal of guidance, control, and dynamics*, 25(6):1091–1099, 2002. [Quoted on p. 8, 16]

[20] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification and scene analysis*, volume 3. Wiley New York, 1973. [Quoted on p. 8]

[21] Matt Whalley, Marc Takahashi, P Tsenkov, G Schulein, and C Goerzen. Field-testing of a helicopter uav obstacle field navigation and landing system. In *65th Annual Forum of the American Helicopter Society, Grapevine, TX*, 2009. [Quoted on p. 8, 16]

[22] Matthew S Whalley, Marc D Takahashi, Jay W Fletcher, Ernesto Moralez III, LTC Carl R Ott, LTC Michael G Olmstead, James C Savage, Chad L Goerzen, Gregory J Schulein, Hoyt N Burns, et al. Autonomous black hawk in flight: Obstacle field navigation and landing-site selection on the rascal juh-60a. *Journal of Field Robotics*, 31(4):591–616, 2014. [Quoted on p. 8]

[23] Lyle Chamberlain, Sebastian Scherer, and Sanjiv Singh. Self-aware helicopters: Full-scale automated landing and obstacle avoidance in unmapped environments. In *Ahs Forum*, volume 67, 2011. [Quoted on p. 8, 16]

[24] Sebastian Scherer, Lyle Chamberlain, and Sanjiv Singh. Autonomous landing at unprepared sites by a full-scale helicopter. *Robotics and Autonomous Systems*, 60(12):1545–1562, 2012. [Quoted on p. 9]

[25] Sebastian Scherer, Lyle Chamberlain, and Sanjiv Singh. Online assessment of landing sites. In *AIAA Infotech@ Aerospace 2010*, page 3358. 2010. [Quoted on p. 9]

[26] Sebastian Scherer. Low-altitude operation of unmanned rotorcraft. 2011. [Quoted on p. 9]

[27] Sung Joon Ahn. *Least squares orthogonal distance fitting of curves and surfaces in space*, volume 3151. Springer Science & Business Media, 2004. [Quoted on p. 9]

[28] Franco P Preparata and Michael I Shamos. *Computational geometry: an introduction*. Springer Science & Business Media, 2012. [Quoted on p. 9]

[29] Daniel Maturana and Sebastian Scherer. 3d convolutional neural networks for landing zone detection from lidar. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 3471–3478. IEEE, 2015. [Quoted on p. 9, 10, 16]

[30] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990. [Quoted on p. 9]

[31] Matthew S Whalley, Marc D Takahashi, Jay W Fletcher, Ernesto Moralez III, LTC Carl R Ott, LTC Michael G Olmstead, James C Savage, Chad L Goerzen, Gregory J Schulein, Hoyt N Burns, et al. Autonomous black hawk

in flight: Obstacle field navigation and landing-site selection on the rascal juh-60a. *Journal of Field Robotics*, 31(4):591–616, 2014. [Quoted on p. 9, 10, 16]

[32] Nicholas A Rediess, Fernando Dones, Bruce L McManus, Lon Ulmer, and Edwin W Aiken. An advanced fly-by-wire flight control system for the rascal research rotorcraft: Concept to reality. 1995. [Quoted on p. 9]

[33] Oscar G Lorenzo, Jorge Martínez, David L Vilariño, Tomás F Pena, José C Cabaleiro, and Francisco F Rivera. Landing sites detection using lidar data on manycore systems. *The Journal of Supercomputing*, 73(1):557–575, 2017. [Quoted on p. 11]

[34] Abu Saleh Mohammad Mosa, Bianca Schön, Michela Bertolotto, and Debra F Laefer. Evaluating the benefits of octree-based indexing for lidar data. *Photogrammetric Engineering & Remote Sensing*, 78(9):927–934, 2012. [Quoted on p. 11, 47]

[35] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. [Quoted on p. 11]

[36] Weiwei Kong, Dianle Zhou, Daibing Zhang, and Jianwei Zhang. Vision-based autonomous landing system for unmanned aerial vehicle: A survey. In *2014 international conference on multisensor fusion and information integration for intelligent systems (MFI)*, pages 1–8. IEEE, 2014. [Quoted on p. 11]

[37] Pedro J Garcia-Pardo, Gaurav S Sukhatme, and James F Montgomery. Towards vision-based safe landing for an autonomous helicopter. *Robotics and Autonomous Systems*, 38(1):19–29, 2002. [Quoted on p. 11]

[38] Sébastien Bosch, Simon Lacroix, and Fernando Caballero. Autonomous detection of safe landing areas for an uav from monocular images. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5522–5527. IEEE, 2006. [Quoted on p. 11]

[39] Daniel Fitzgerald, Rodney Walker, and Duncan Campbell. A vision based forced landing site selection system for an autonomous uav. In *2005 International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pages 397–402. IEEE, 2005. [Quoted on p. 11, 16]

[40] Daniel Liam Fitzgerald. *Landing site selection for UAV forced landings using machine vision*. PhD thesis, Queensland University of Technology, 2007. [Quoted on p. 11, 16]

[41] Luis Mejias, Daniel L Fitzgerald, Pillar C Eng, and Liu Xi. Forced landing technologies for unmanned aerial vehicles: towards safer operations. *Aerial vehicles*, 1:415–442, 2009. [Quoted on p. 11, 12, 16]

[42] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986. [Quoted on p. 11]

[43] Michael Warren, Luis Mejias, Xilin Yang, Bilal Arain, Felipe Gonzalez, and Ben Upcroft. Enabling aircraft emergency landings using active visual site detection. In *Field and Service Robotics*, pages 167–181. Springer, 2015. [Quoted on p. 12]

[44] PT Eendebak, AWM van Eekeren, and RJM den Hollander. Landing spot selection for uav emergency landing. In *Unmanned Systems Technology XV*, volume 8741, page 874105. International Society for Optics and Photonics, 2013. [Quoted on p. 12]

[45] Massimo Piccardi. Background subtraction techniques: a review. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, volume 4, pages 3099–3104. IEEE, 2004. [Quoted on p. 12]

[46] Yu-Fei Shen, Zia-Ur Rahman, Dean Krusienski, and Jiang Li. A vision-based automatic safe landing-site detection system. *IEEE Transactions on Aerospace and Electronic Systems*, 49(1):294–311, 2013. [Quoted on p. 12, 13]

[47] Christian Forster, Matthias Faessler, Flavio Fontana, Manuel Werlberger, and Davide Scaramuzza. Continuous on-board monocular-vision-based elevation mapping applied to autonomous landing of micro aerial vehicles. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 111–118. IEEE, 2015. [Quoted on p. 12, 14]

[48] Timo Hinzmann, Thomas Stastny, Cesar Cadena, Roland Siegwart, and Igor Gilitschenski. Free lsd: prior-free visual landing site detection for autonomous planes. *IEEE Robotics and Automation Letters*, 3(3):2545–2552, 2018. [Quoted on p. 13]

[49] Bulent Ayhan, Chiman Kwan, Yool-Bin Um, Bence Budavari, and Jude Larkin. Semi-automated emergency landing site selection approach for uavs. *IEEE Transactions on Aerospace and Electronic Systems*, 55(4):1892–1906, 2018. [Quoted on p. 13]

[50] Navid Serrano. A bayesian framework for landing site selection during autonomous spacecraft descent. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5112–5117. IEEE, 2006. [Quoted on p. 15]

[51] Henry E Kyburg. Probabilistic reasoning in intelligent systems: networks of plausible inference by judea pearl. *The Journal of Philosophy*, 88(8):434–437, 1991. [Quoted on p. 15]

[52] Ayanna Howard and Homayoun Seraji. Multi-sensor terrain classification for safe spacecraft landing. *IEEE Transactions on Aerospace and Electronic Systems*, 40(4):1122–1131, 2004. [Quoted on p. 15]

[53] Sebastian Scherer, Lyle Chamberlain, and Sanjiv Singh. Autonomous landing at unprepared sites by a full-scale helicopter. *Robotics and Autonomous Systems*, 60(12):1545–1562, 2012. [Quoted on p. 16]

[54] Jamie Carter, Keil Schmid, Kirk Waters, Lindy Betzhold, B Hadley, R Mataosky, and J Halleran. An introduction to lidar technology, data, and applications. *NOAA Coastal Services Center*, 2, 2012. [Quoted on p. 17]

[55] Fiorani Luca. Environmental monitoring by laser radar. *Lasers and Electro-Optics Research at the Cutting Edge, Nova Publishers, New York*, pages 119–171, 2007. [Quoted on p. 17]

[56] Thimothy B. Lee. Why spinning lidar sensors might be around for another decade. `https://arstechnica.com/cars/2018/05/why-bulky-spinning-lidar-sensors-might-be-around-for-another-decade/`. Online;Accessed on Jul. 3, 2020. [Quoted on p. 18, 19]

[57] Nick Mokey. Solid-state lidar: The key to cheap self-driving cars - Digital Trends. `https://www.digitaltrends.com/cars/solid-state-lidar-for-self-driving-cars./`. Online;Accessed on Oct. 11, 2020. [Quoted on p. 18, 19, 20]

[58] Renishaw. Optical encoders and LiDAR scanning. `https://www.renishaw.it/it/optical-encoders-and-lidar-scanning--39244`. Online; Accessed on Jul. 3, 2020. [Quoted on p. 18]

[59] Velodyne LiDAR. *VLP-16 User Manual*, 2 2019. Accessed on Jul. 3, 2020. [Quoted on p. 18, 19, 24, 54, 56]

[60] LeddarTech. Leddar Solid-State LiDAR Technology Fundamentals. `https://leddartech.com/leddar-technology-overview`. Online;Accessed on Oct. 11, 2020. [Quoted on p. 18]

[61] Andy Mohd. Does Infineon's MEMS lidar portend a quantum leap for ADAS? `https://medium.com/futuremobile2025/does-infineons-mems-lidar-portend-a-quantum-leap-for-adas-c14eb939545a`. Online;Accessed on Oct. 11, 2020. [Quoted on p. 19, 20]

[62] Kimon P Valavanis and George J Vachtsevanos. *Handbook of unmanned aerial vehicles*, volume 1. Springer, 2015. [Quoted on p. 20, 22]

[63] Jay Farrell. *Aided navigation: GPS with high rate sensors*. McGraw-Hill, Inc., 2008. [Quoted on p. 21]

[64] Guowei Cai, Ben M Chen, and Tong Heng Lee. Coordinate systems and transformations. In *Unmanned rotorcraft systems*, pages 23–34. Springer, 2011. [Quoted on p. 21]

[65] ChRobotics. Understanding Quaternions. `http://www.chrobotics.com/wp-content/uploads/2012/11/F18.png`. Online;Accessed on Jul. 6, 2020. [Quoted on p. 25]

[66] Gregory G Slabaugh. Computing euler angles from a rotation matrix. *Retrieved on August*, 6(2000):39–63, 1999. [Quoted on p. 25]

[67] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010. [Quoted on p. 26]

[68] Sebastian Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3): 52–57, 2002. [Quoted on p. 27]

[69] Gabriel A Terejanu et al. Extended kalman filter tutorial. *University at Buffalo*, 2008. [Quoted on p. 27]

[70] Sergio Orts-Escolano, Vicente Morell, José García-Rodríguez, and Miguel Cazorla. Point cloud data filtering and downsampling using growing neural gas. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2013. [Quoted on p. 30]

[71] LibPointMatcher. libpointmatcher Documentation. `https://libpointmatcher.readthedocs.io/en/latest/DataPointsFilterDev/`. Online;Accessed on Oct. 10, 2020. [Quoted on p. 31]

[72] Donald JR Meagher. *Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer*. Electrical and Systems Engineering Department Rensseiaer Polytechnic . . . , 1980. [Quoted on p. 31]

[73] The Infinite Loop. Advanced Octrees 1: preliminaries, insertion strategies and maximum tree depth. `https://geidav.wordpress.com/2014/07/18/advanced-octrees-1-preliminaries-insertion-strategies-and-max-tree-depth/`. Online;Accessed on Jul. 15, 2020. [Quoted on p. 32]

[74] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34(3):189–206, 2013. [Quoted on p. 32]

[75] Hemang Agarwal, Sandeep Bhardwaj, Hobart Pao, George Keeling, Andres Gonzalez, Mahindra Jain and Jimin Khim . 3D Coordinate Geometry - Equation of a Plane. https://brilliant.org/wiki/3d-coordinate-geometry-equation-of-a-plane/. Online;Accessed on Oct. 11, 2020. [Quoted on p. 32, 33]

[76] Ian T Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016. [Quoted on p. 33]

[77] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010. [Quoted on p. 33]

[78] Weisstein, Eric W. "Covariance." From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/Covariance.html, . Online;Accessed on Jul. 20, 2020. [Quoted on p. 34]

[79] Weisstein, Eric W. "Eigenvalue." From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/Eigenvalue.html, . Online;Accessed on Oct. 11, 2020. [Quoted on p. 34]

[80] Valentina Alto. PCA: Eigenvectors and Eigenvalues. https://towardsdatascience.com/pca-eigenvectors-and-eigenvalues-1f968bc6777a. Online;Accessed on Oct. 11, 2020. [Quoted on p. 35]

[81] Ayssam Elkady and Tarek Sobh. Robotics middleware: A comprehensive literature survey and attribute-based bibliography. *Journal of Robotics*, 2012, 2012. [Quoted on p. 35]

[82] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009. [Quoted on p. 35, 39]

[83] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004. [Quoted on p. 37]

[84] Eric Rohmer, Surya PN Singh, and Marc Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326. IEEE, 2013. [Quoted on p. 37]

[85] Gilberto Echeverria, Séverin Lemaignan, Arnaud Degroote, Simon Lacroix, Michael Karg, Pierrick Koch, Charles Lesire, and Serge Stinckwich. Simulating complex robotic scenarios with morse. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 197–208. Springer, 2012. [Quoted on p. 37]

[86] Gilberto Echeverria, Nicolas Lassabe, Arnaud Degroote, and Séverin Lemaignan. Modular open robots simulation engine: Morse. In *2011 IEEE International Conference on Robotics and Automation*, pages 46–51. Citeseer, 2011. [Quoted on p. 37]

[87] "The MORSE Simulator Documentation. `https://www.openrobots.org/morse/doc/stable/morse.html`. Online;Accessed on Oct. 6, 2020. [Quoted on p. 37]

[88] MORSE Simulator. MORSE Pre-defined Environment. `https://www.openrobots.org/morse/doc/stable/user/environments/buildings_2.html`. Online;Accessed on Oct. 6, 2020. [Quoted on p. 38]

[89] Freepik. Flat Drone. `https://www.freepik.com/free-vector/basic-variety-of-flat-drones_1348538.htm`. Online;Accessed on Oct. 11, 2020. [Quoted on p. 40]

[90] DNS. ODROID-XU4. `https://cdn.antratek.nl/media/product/d9e/odroid-xu4-octa-core-computer-with-samsung-exynos-5422-g143452239825-47c.jpg`. Online;Accessed on Oct. 11, 2020. [Quoted on p. 40]

[91] Gabriel Loureiro, Luís Soares, André Dias, and Alfredo Martins. Emergency landing spot detection for unmanned aerial vehicle. In *Iberian Robotics conference*, pages 122–133. Springer, 2019. [Quoted on p. 43]

[92] Nicolas Brodu and Dimitri Lague. 3d terrestrial lidar data classification of complex natural scenes using a multi-scale dimensionality criterion: Applications in geomorphology. *ISPRS Journal of Photogrammetry and Remote Sensing*, 68:121–134, 2012. [Quoted on p. 47, 48]

[93] Velodyne Lidar. VLP-16 Puck. `https://velodynelidar.com/products/puck/`. Online;Accessed on Oct. 5, 2020. [Quoted on p. 55]

[94] Radu B Rusu and S Cousins. Point cloud library (pcl). In *2011 IEEE International Conference on Robotics and Automation*, pages 1–4, 2011. [Quoted on p. 58]

[95] We Braga. Monastery of Tibães. `https://webraga.pt/visitar/monumentos/mosteiro-de-tibaes/`. Online;Accessed on Oct. 12, 2020. [Quoted on p. 75]