

12-2019

Adaptive Learning Terrain Estimation for Unmanned Aerial Vehicle Applications

Pedro L. Vergara Garcia

Follow this and additional works at: <https://commons.erau.edu/edt>

 Part of the [Aerospace Engineering Commons](#)

This Thesis - Open Access is brought to you for free and open access by Scholarly Commons. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of Scholarly Commons. For more information, please contact commons@erau.edu.

ADAPTIVE LEARNING TERRAIN ESTIMATION FOR UNMANNED AERIAL
VEHICLE APPLICATIONS

A Thesis

Submitted to the Faculty of
Embry-Riddle Aeronautical University

by

Pedro L. Vergara Garcia

In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Aerospace Engineering

December 2019

Embry-Riddle Aeronautical University

Daytona Beach, Florida


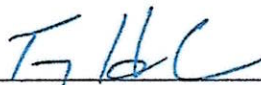
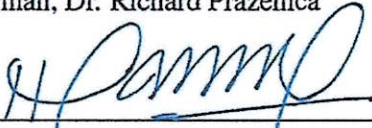



ADAPTIVE LEARNING TERRAIN ESTIMATION FOR UNMANNED AERIAL
VEHICLE APPLICATIONS

by

Pedro L. Vergara Garcia

This Thesis was prepared under the direction of the candidate's Committee Chair, Dr. Richard Prazenica, Department of Aerospace Engineering, and has been approved by the members of the Thesis Committee. It was submitted to the Office of the Senior Vice President for Academic Affairs and Provost, and was accepted in partial fulfillment of the requirements for the Degree of Master of Science in Aerospace Engineering.

THESIS COMMITTEE

 Chairman, Dr. Richard Prazenica	 Member, Dr. Troy Henderson
 Member, Dr. Hever Moncayo	
 Graduate Program Coordinator, Dr. Magdy Attia	12.5.2019 Date
 Dean of College of Engineering, Dr. Maj Mirmirani	12/5/19 Date
 Associate Provost of Academic Support, Dr. Christopher Grant	12/5/19 Date

In loving memory of my grandfather, Vinicio.

ACKNOWLEDGMENTS

Special gratitude goes to my adviser and thesis chairman, Dr. Richard Prazenica, who gave me the chance to start this journey and helped me through the end. I feel privileged to have had the opportunity to work and learn from him. He has been a great mentor, but most importantly, he has been a good friend.

I would like to thank Dr. Troy Henderson and Dr. Hever Moncayo for serving as thesis committee members. Their feedback was very valuable for the completion of this document.

To my wife, Andrea: For being by my side every day. Thanks for all the support during the hard times. Only you know the sacrifices we've both made to be where we are. For these reasons and many more, this accomplishment is not mine alone but yours as well. Thanks for pushing me to be better. This endeavor would've been impossible without you.

I'm grateful to my father, Fredy, who has been my lifelong mentor. You passed me your hopes and dreams, and for that, I'm thankful. You never stopped believing I was destined for something greater. Your faith in me put me where I am right now. Any achievement, past or future, is in significant part, thanks to you. Thanks for teaching me that the greatness of the soul is more important than anything material.

To my mother, Ibeth: For showing me the real meaning of hard work and putting up with me all these years. I'm thankful for having such a loving and caring mother. Thanks for teaching me how to be humble. But most of all, thanks for teaching me that no action has value if not done for the sake of others.

TABLE OF CONTENTS

	Page
LIST OF TABLES	vii
LIST OF FIGURES	xi
ABBREVIATIONS	xii
ABSTRACT	xiii
1. Introduction	1
1.1 Objectives	1
1.2 Cases	3
1.3 Literature Review	3
1.3.1 2D maps	4
1.3.2 3D maps	5
1.3.3 Dynamic environments	7
1.3.4 Terrain Mapping Applications for UAVs	7
2. Test Benchmark	9
2.1 Simulation Environment	9
2.1.1 ROS and Gazebo	9
2.1.2 Matlab/Simulink	12
2.1.3 Point Cloud Registration	14
2.1.4 Clustering and Plane Fitting	16
2.2 Experimental Environment	17
2.2.1 Sensors	17
2.2.2 Data Acquisition Setup	19
2.2.3 Motion Sensing	20
2.2.4 Control Volume	22
3. Mapping Techniques	25
3.1 Quadtree	25
3.2 Octomap	28
3.3 Adaptive Learning Terrain Estimation	31
3.3.1 2D-ALTE	31
3.3.2 3D-ALTE	35
4. Terrain Mapping Results	38
4.1 Simulated Terrains	38
4.1.1 Urban Navigation: Terrain 1	38
4.1.2 Urban Navigation: Terrain 2	42
4.1.3 Autonomous Landing: Terrain 1	44
4.1.4 Autonomous Landing: Terrain 2	47

	Page
4.2 Experimental Terrains	51
4.2.1 Urban Navigation: Terrain 1	51
4.2.2 Urban Navigation: Terrain 2	55
4.2.3 Autonomous Landing: Terrain 1	58
4.2.4 Autonomous Landing: Terrain 2	61
5. Analysis	65
5.1 Qualitative Analysis	65
5.1.1 Remarks	67
5.2 Quantitative Analysis	69
5.2.1 Metrics	69
5.2.2 Results	71
6. Conclusions	80
REFERENCES	83

LIST OF TABLES

Table	Page
5.1 Summary of general qualitative findings on 2D algorithms.	66
5.2 Summary of general qualitative findings on 3D algorithms.	66
5.3 Summary of case-oriented qualitative findings on 2D algorithms.	66
5.4 Summary of case-oriented qualitative findings on 3D algorithms.	67
5.5 Information on the terrain's point cloud for 2D analysis.	71
5.6 Information on the terrain's point cloud for 2D analysis.	72
5.7 Quadtree results after varying the resolution parameter.	72
5.8 Octomap results after varying the resolution parameter.	74
5.9 Results of varying tuning parameters on 2D ALTE algorithm.	75
5.10 Comparison between the best results of Quadtree and 2D ALTE.	78
5.11 Comparison between the best results of Octomap and 3D ALTE.	78
5.12 Results of varying tuning parameters on 3D ALTE algorithm.	78

LIST OF FIGURES

Figure	Page
2.1 ROS/Gazebo/Matlab network diagram.	10
2.2 Sample Gazebo simulation environment.	10
2.3 VLP16 model for ROS from Velodyne Description package	11
2.4 VLP16 mounted on the quadcopter.	11
2.5 Top-level Simulink code to communicate with Gazebo.	12
2.6 Frames for coordinate transformation.	13
2.7 Mission trajectory for sample PCR.	15
2.8 Point cloud scans at arbitrary times for sample PCR.	15
2.9 Sample point cloud after applying PCR algorithm.	16
2.10 Point cloud after ground filtering and clustering.	16
2.11 VLP-16 Information.	17
2.12 Velodyne LiDAR GUI for setting operation parameters.	18
2.13 VICON Vantage setup and network.	18
2.14 VICON single camera for motion sensing.	19
2.15 VICON-LiDAR system general structure.	19
2.16 Diagram showing VICON pearls distribution.	20
2.17 VICON pearls on LiDAR.	20
2.18 VICON virtual object.	21
2.19 VICON Vantage system tracking object.	21
2.20 Lidar-tripod-cart setup.	22
2.21 Control volume and grid.	23
2.22 Snapshot of sample terrain used to the test real environment.	23
2.23 Point cloud of sample terrain collected during test run.	24
2.24 Sensor trajectory for test run.	24
3.1 Quadtree concept.	25

Figure	Page
3.2 Quadtree map used for collision avoidance.	26
3.3 Octomap applied over time.	27
3.4 Octree Voxel Concept.	28
3.5 Free and unknown space representation.	29
3.6 Octomap representation at 3 different resolutions.	29
3.7 Octomap applied over time.	30
3.8 Adaptive partitioning of a 2D domain.	31
3.9 Simulated point cloud from virtual urban environment.	32
3.10 2D adaptive terrain map.	33
3.11 Point cloud from real flight data.	33
3.12 2D adaptive terrain map from real flight data.	34
3.13 Obstacle map from real flight data.	34
3.14 2D Adaptive Learning applied over time.	35
3.15 3D adaptive terrain map.	36
3.16 3D adaptive terrain map with fixed cell altitude.	37
3.17 3D Adaptive Learning applied over time.	37
4.1 Terrain 1 sim urban navigation snapshot.	39
4.2 Terrain 1 sim urban navigation position and attitude.	39
4.3 Terrain 1 sim urban navigation trajectory.	40
4.4 Terrain 1 sim urban navigation point cloud.	40
4.5 Terrain 1 sim urban navigation Octomap.	41
4.6 Terrain 1 sim urban navigation 3D-ALTE.	41
4.7 Terrain 2 sim urban navigation snapshot.	42
4.8 Terrain 2 sim urban navigation position and attitude.	42
4.9 Terrain 2 sim urban navigation trajectory.	43
4.10 Terrain 2 sim urban navigation point cloud.	43

Figure	Page
4.11 Terrain 2 sim urban navigation Octomap.	44
4.12 Terrain 2 sim urban navigation 3D-ALTE.	44
4.13 Terrain 1 sim landing snapshot.	45
4.14 Terrain 1 sim landing position and attitude.	45
4.15 Terrain 1 sim landing trajectory.	46
4.16 Terrain 1 sim landing point cloud.	46
4.17 Terrain 1 sim landing Quadtree.	47
4.18 Terrain 1 sim landing 2D-ALTE.	47
4.19 Terrain 2 sim landing snapshot.	48
4.20 Terrain 2 sim landing position and attitude.	48
4.21 Terrain 2 sim landing trajectory.	49
4.22 Terrain 2 sim landing point cloud.	50
4.23 Terrain 2 sim landing Quadtree.	50
4.24 Terrain 2 sim landing 2D-ALTE.	50
4.25 Terrain 1 exp urban navigation snapshot.	51
4.26 Terrain 1 exp urban navigation VICON data.	52
4.27 Terrain 1 exp urban navigation trajectory.	52
4.28 Terrain 1 exp urban navigation point cloud.	53
4.29 Terrain 1 exp urban navigation Octomap.	54
4.30 Terrain 1 exp urban navigation Octomap.	54
4.31 Terrain 2 exp urban navigation snapshot.	55
4.32 Terrain 2 exp urban navigation VICON data.	56
4.33 Terrain 2 exp urban navigation trajectory.	56
4.34 Terrain 2 exp urban navigation point cloud.	57
4.35 Terrain 2 exp urban navigation Octomap.	57
4.36 Terrain 2 exp urban navigation 3D-ALTE.	58

Figure	Page
4.37 Terrain 1 exp landing snapshot.	58
4.38 Terrain 1 exp landing VICON data.	59
4.39 Terrain 1 exp landing trajectory.	59
4.40 Terrain 1 exp landing point cloud.	60
4.41 Terrain 1 exp landing Quadtree.	60
4.42 Terrain 1 exp landing 2D-ALTE.	61
4.43 Terrain 2 exp landing snapshot.	61
4.44 Terrain 2 exp VICON data.	62
4.45 Terrain 2 exp landing trajectory.	62
4.46 Terrain 2 exp landing point cloud.	63
4.47 Terrain 2 exp landing Quadtree.	63
4.48 Terrain 2 exp landing 2D-ALTE.	64
5.1 Graphic representation of HD and MHD implementation.	71
5.2 Quadtree maps at different max-points-per-cell values.	73
5.3 Octomap at different cell-per-meter values.	74
5.4 2D Adaptive Learning maps for different tuning parameter combinations.	76
5.5 3D Adaptive Learning maps for different tuning parameter combinations.	79

ABBREVIATIONS

ALTE	Adaptive Learning Terrain Estimation
DCM	Direction Cosine Matrix
GPS	Global Positioning System
GUI	Graphic User Interface
HD	Hausdorff Distance
ICP	Iterative Closest Point
IMU	Inertia Measuremnt Unit
MHD	Modified Hausdorff Distance
MSAC	M-Estimator Sample Consensus
PCR	Point Cloud Registration
RANSAC	Random Sample Consensus
SLAM	Simultaneous Location And Mapping
RC	Remote Control
ROS	Robotic Operative System

ABSTRACT

For the past decade, terrain mapping research has focused on ground robots using occupancy grids and tree-like data structures, like Octomap and Quadtrees. Since flight vehicles have different constraints, ground-based terrain mapping research may not be directly applicable to the aerospace industry. To address this issue, Adaptive Learning Terrain Estimation algorithms have been developed with an aim towards aerospace applications. This thesis develops and tests Adaptive Learning Terrain Estimation algorithms using a custom test benchmark on representative aerospace cases: autonomous UAV landing and UAV flight through 3D urban environments. The fundamental objective of this thesis is to investigate the use of Adaptive Learning Terrain Estimation algorithms for aerospace applications and compare their performance to commonly used mapping techniques such as Quadtree and Octomap. To test the algorithms, point clouds were collected and registered in simulation and real environments. Then, the Adaptive Learning, Quadtree, and Octomap algorithms were applied to the data sets, both in real-time and offline. Finally, metrics of map size, accuracy, and running time were developed and implemented to quantify and compare the performance of the algorithms. The results show that Quadtree yields the computationally lightest maps, but it is not suitable for real-time implementation due to its lack of recursiveness. Adaptive Learning maps are computationally efficient due to the use of multiresolution grids. Octomap yields the most detailed maps, but it produces a high computational load. The results of the research show that Adaptive Learning algorithms have significant potential for real-time implementation in aerospace applications. Their low memory load and variable-sized grids make them viable candidates for future research and development.

1. Introduction

The number of unmanned air vehicles (UAVs), for commercial and military applications, has grown exponentially in the past decade. Even though these vehicles were initially meant to be remotely controlled (RC), attention has rapidly shifted towards full autonomy. There are five levels of UAV autonomy: Levels 1 and 2, known as pilot assistance, require position and attitude estimates to perform simple tasks such as altitude correction and cruise control. In Level 3, the pilot acts as a fallback system. It means the aircraft can fly autonomously given certain conditions and notifies the pilot if intervention is needed. The required sensor information is limited to the vehicle's states. Levels 4 and 5 do not need a pilot, but require information on the environment, in addition to the states.

Ongoing research focuses mainly on Levels 4 and 5, where precise information on the environment is vital for the mission's success. However, in many aerospace missions, like urban navigation and autonomous landing, the vehicle cannot rely exclusively on conventional localization methods like global positioning systems (GPS). The lack of GPS and the need for terrain information for high-level autonomy create an essential niche for real-time mapping in aerospace applications. To help address this need, Adaptive Learning Terrain Estimation (ALTE) have been developed and investigated for terrain mapping applications for path planning and obstacle avoidance. The purpose of this thesis is to further the research on ALTE by testing the 2D and 3D algorithms using custom test benchmarks. It compares the ALTE results to two conventional mapping algorithms, Octomap and Quadtree, in simulation and real environments common to UAV applications.

1.1 Objectives

The primary purpose of this investigation is to further the research on ALTE algorithms with an approach that focuses on two specific autonomous UAV missions. Each category, 2D and 3D, has a particular test scenario related to it: autonomous landing for 2D algorithms and urban navigation for 3D algorithms. The milestones to achieve this

main objective are the following:

1. Build a test benchmark that includes a simulation and real environments equipped with sensors and data acquisition (DAQ) systems for data collection. Develop point cloud registration codes and algorithms for processing point clouds such as filtering, down-sampling and clustering.
2. Develop implementation codes for the two versions of ALTE, Octomap, and Quadtree algorithms. Then, generate terrain maps from real and simulated data for analysis.
3. Define and apply metrics that quantitatively and qualitatively characterize the algorithms' performance and results.
4. Analyze the results and compare the performance of the terrain algorithms for the benchmark scenarios.
5. Provide conclusions on how to improve ALTE algorithms for the proposed real-time UAV applications.

This thesis is organized as follows: Chapter 1 provides an overview of the problem and objectives and reviews relevant literature. Chapter 2 describes the development process of the test benchmarks used for data acquisition and analysis. These benchmarks include a simulation environment, tools for data registration and processing, and a hardware setup for acquiring real data. Chapter 3 details the two versions of ALTE, Octomap, and Quadtree algorithms, and their implementation. Chapter 4 shows the results of implementing the 2D and 3D mapping algorithms in simulated and real terrains, including their corresponding point clouds and mission profiles. Chapter 5 analyzes the results using qualitative and quantitative criteria. Chapter 6 provides conclusions and recommendations.

1.2 Cases

The selected cases (missions or UAV applications), defined to provide context for analysis, represent two common scenarios where terrain mapping plays a crucial role in the mission's success. The first case, urban navigation, is important due to the high demand for aerial services and in potentially complex 3D environments. The potential lack of GPS makes terrain mapping vital for self-localization, path planning, and obstacle avoidance. The second case, terrain mapping for autonomous landings, plays a critical role in cargo vehicle and package delivery automation.

In this thesis, an urban navigation case is defined as any environment that is composed of 3D objects such as overhanging structures and buildings of different altitudes. Dynamic obstacles, like other vehicles or people, are not considered part of the environment. Mapping algorithms for this scenario should identify 3D terrain features and provide information in a useful and efficient manner such that a path planning algorithm could be implemented to generate an obstacle avoidance trajectory. A landing zone is defined as any planar site, free of obstacles, with a clear approach path, and with an area big enough for the vehicle to land. A suitable 2D mapping algorithm for this case should be able to identify obstacles and non-viable landing spots.

1.3 Literature Review

According to the literature, present research on terrain mapping is divided into 2D maps, 3D maps, and dynamic environments. Investigations have found a variety of challenges when integrating mapping into modern vehicle applications. For instance, 2D grids and elevation maps cannot incorporate uncertainty into their analyses. Section 1.3.1 summarizes applications of 2D maps, including the classical Quadtree algorithm and the most current research on this area. It also covers several propositions to mitigate the effects of noise and stochastic measurements. Collapsing the real world into a 2D representation limits the applicability of these algorithms in 3D environments.

3D mapping algorithms address the dimensional issue but produce other

complications. The high computational burdens cause difficulties in real-time implementation. To address this problem, researchers have developed several statistical approaches to reduce the sizes of these maps. Section 1.3.2 presents part of the ongoing research on 3D mapping, including the works of Hornung, Wurm, Bennewitz, Stachniss, and Burgard (2013) on Octomap. 2D and 3D maps assume the terrain is not moving nor will change suddenly (static environment). However, applications for mapping have expanded towards terrains with moving agents like people, cars, animals, or other vehicles (dynamic environments). While dynamic environments are not considered in this thesis, Section 1.3.3 reviews some sources addressing this issue (Siciliano & Khatib, 2016).

1.3.1 2D maps

Occupancy grids are the most common type of 2D maps. In fact, the Quadtree-based algorithm used in this thesis is a straightforward 2D occupancy grid. Most current tree-like data structures used in terrain mapping are based on Quadtrees. The most common application for Quadtree is image processing, as seen in (Shusterman & Feder, 1994), where it is applied to image compression, but it is not limited to this area. It also has an important role in autonomous vehicle applications, like simultaneous location and mapping (SLAM), as seen in (Vallivaara, Poikselkä, Kemppainen, & Röning, 2018), where the resulting maps are used for collision avoidance. Chapter 3 explains Quadtree in more detail.

In contrast to Quadtree, elevation maps, another type of 2D map, includes information from the third dimension. An elevation map represents the terrain by defining a grid that stores the mean altitude of the region in each cell. The advantage of this representation is the low computational demands. However, they are insufficient for representing multi-level environments in which height difference is a vital feature to consider (e.g., overhangs, tunnels, and bridges). Pfaff, Triebel, and Burgard (2007) addresses multi-level terrains using two methods. The first method allows the level map to choose a dominant height when the region contains overhanging structures. The resulting

terrain representation is no longer locally inaccurate but has certain limitations related to variance in measurements. The second method is a classification algorithm that discriminates locations into four groups: locations sensed from above, vertical structures, vertical gaps, and traversable cells. Triebel, Pfaff, and Burgard (2006) extends the capabilities of level maps by introducing multiple surfaces in different levels which managed to represent vertical objects.

Rivadeneira, Miller, Schoenberg, and Campbell (2009) use a probabilistic approach that accounts for multi-level environments and uncertainty. This technique enhances the multi-level surface mapping proposed by Triebel et al. (2006). In (Rivadeneira & Campbell, 2011), the authors extend their previous work presented in (Rivadeneira et al., 2009) with a technique named PML (probabilistic multi-level mapping) that uses formal probability theory to include modeling errors due to uncertainty. It associates measurements probabilistically instead of directly placing points in the cell. Even with all the innovative techniques described above, 2D maps are still a non-robust representation of reality. The effects of partially ignoring one dimension are significant when a high level of 3D fidelity is required.

1.3.2 3D maps

The most straightforward approach to 3D mapping consists of using a 3D grid of volumes, called voxels, conceived as binary spaces. But, contrary to the 2D version, 3D grids have a substantial memory cost. Adding the third dimension increases the computational burden exponentially since basic grids have to be predefined, needing specific a priori knowledge. In autonomous missions, the need for preliminary information is highly restrictive, so initialized grids pose an issue, and expandable grids require memory-costly operations.

Since 3D mapping literature focuses mostly on ground robots, data processing issues are secondary. Many researchers believe that they can attain computational efficiency via code optimization rather than devising memory-efficient mapping methods.

In any case, research is drifting away from memory performance into new mapping techniques like probability mapping and surfel mapping. Ryde and Hu (2009) shows a successful probabilistic approach using multi-resolution voxels, but with significant concerns regarding the occupancy probability. It assumes the occupancy is independent of neighboring cells, which is problematic when the robot is stationary due to probability accumulation. The authors acknowledge that managing the vast amounts of data coming from a 3D scanner is difficult, so they propose collapsing the data into 2D scans and then store it in occupied cell lists.

Schadler, Stückler, and Behnke (2013) propose a surfel (surface element) approach. This technique discretizes the environment into a voxel map of fixed resolution. Each cell contains a surfel element with certain characteristics; then, these surface characteristics are updated depending on the properties of the scans that fall into the respective voxel. Surfel approaches are unable to distinguish between free and unknown space and require assumptions based on the terrain. Hornung et al. (2013) propose an octree-based map that uses probabilistic occupancy estimation called Octomap, which is by far the most currently used 3D mapping technique. It implements several algorithms that keep the map compact for memory efficiency purposes. The result is a grid that outperforms most 3D mapping approaches. Saarinen, Andreasson, Stoyanov, Ala-Luhtala, and Lilienthal (2013) introduce a new approach that builds on Octomap. The Normal Distribution Transform Occupancy Maps or NDT-OM assigns a sample mean and covariance to each cell. These mean and covariance values depend on the accumulated measurements on each cell. Thus, the NDT map describes, using normal distributions, the probability of points being in a particular location. NDT-OM enables recursive updates of sequential measurements, but again, the limitations come from memory capacity. Unlike conventional Octree mapping, the memory usage of Octomap-based techniques depend solely on the size of the map, not on the amount of data received. Among other things, this independence makes Octomap one of the best performing algorithms. Chapter 3 explains

Octomap's features in more detail.

1.3.3 Dynamic environments

Dynamic environments pose a challenge to aerospace, but mapping may not be the best solution. For instance, Wolf and Sukhatme (2003) propose a technique that discriminates between static and dynamic objects, then it classifies and maintains a database of each movable agent. On the other hand, Saarinen et al. (2013) completely ignores moving objects. In both cases, the moving agents are filtered to allow the mapping of the static part of the environment.

1.3.4 Terrain Mapping Applications for UAVs

For the past decade, applications for mapping in UAVs were limited to surveillance and monitoring. Everaerts (2008) and Nex and Remondino (2014) describe aerial mapping applications for archaeology, forestry, farming, road mapping, geology, among others. These applications do not use the map for completing the mission, but rather the mission's purpose is generating the map. More recent applications drift away from surveillance towards navigation. Schmuck and Chli (2017) shows a collaborative SLAM where agents of a robot team send data to a ground station while independently running limited mapping algorithms on-board. Here, the ground station performs most of the mapping effort. The purpose of the mission is to generate a map while estimating the UAVs' positions on the map.

Mapping using LiDAR is particularly useful for GPS denied environments as in (Tang et al., 2019), where UAVs can operate in an unknown terrain using LiDAR-based maps. This application shows the full potential of mapping for UAVs. First, the sensor generates an occupancy grid map. Then, this map is fed to an online path planner and trajectory generator. Vanegas, Gaston, Roberts, and Gonzalez (2019) include another example of navigation in GPS denied environments where the vehicle uses Octomap to navigate. Stulgis, Ambroziak, and Kondratiuk (2018) shows a preliminary obstacle

avoidance algorithm that uses terrain maps to alert the pilot. This application is envisioned as an anti-collision system rather than a mapping algorithm. There is a clear trend in the research: mapping started as a surveillance tool for post-process analyses but is becoming a means for state estimation. These applications show the critical role of mapping in navigation autonomy.

2. Test Benchmark

This chapter describes the development process of the test benchmark developed for this thesis. This benchmark includes a set of tools and data acquisition methods to collect, process, and analyze point clouds. The test benchmark includes a virtual world that provides the dynamics of the vehicle and sensors with a Matlab interface for data logging. The benchmark also includes a physical control volume with a network of sensors for real-time data acquisition. Additionally, several codes for down-sampling, denoising, plane fitting, and clustering were developed as a set of tools for analyzing point clouds. This chapter provides a short description of these mathematical algorithms, as well as a detailed explanation of the hardware set up and simulation development.

2.1 Simulation Environment

The simulation has three primary components: Robotic Operating System (ROS), Gazebo, and Matlab/Simulink. ROS is a framework that allows the user to write software using a collection of tools and libraries that simplify the task of creating complex robotic systems (*About ROS*, n.d.). It handles the software infrastructure for data passing between processes (nodes). Nodes are tasks that communicate with each other using messages via logical channels called topics (*An Introduction to Robot Operating System (ROS)*, n.d.). In this particular case, there are two nodes: Matlab/Simulink and Gazebo. Gazebo is a simulation tool that provides the dynamics of objects inside the environment, while Matlab/Simulink serves as a control panel for the simulation. These nodes are located in two different machines and communicate with each other using the setup shown in Figure 2.1.

2.1.1 ROS and Gazebo

The simulation architecture is based on the virtual world template provided for the IMAV2017 challenge (*Virtual Competition*, n.d.). This world includes a quadrotor model with simple body-fixed frame dynamics, and sensors like stereo vision, laser scanner,

GPS, and inertial measurement units (IMU). Gazebo also provides a true state vector that contains the quadrotor's real attitude and position information. For convenience, Matlab uses this vector's information instead of accessing the GPS and IMU data stream. Gazebo renders the simulation to provide a graphic interface. The user can populate the environment with a variety of objects depending on the needs. A sample terrain is presented in Figure 2.2.

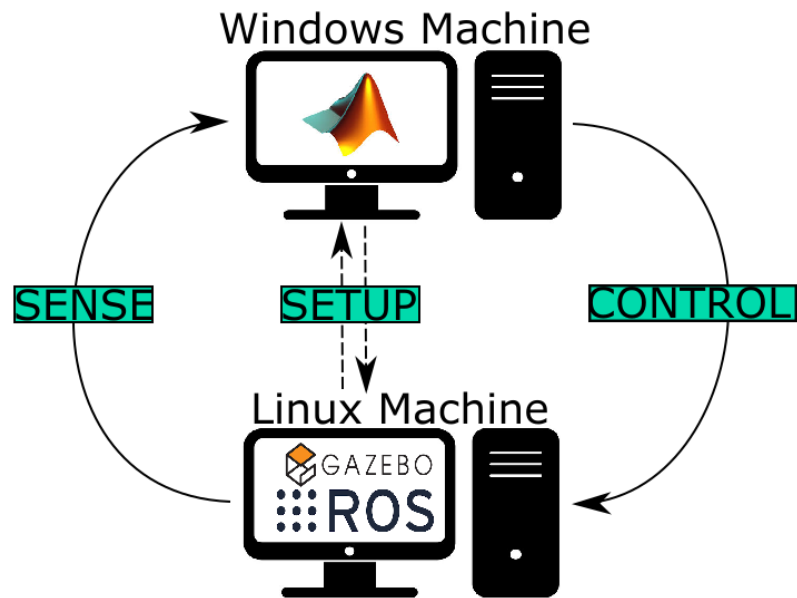


Figure 2.1 ROS/Gazebo/Matlab network diagram.

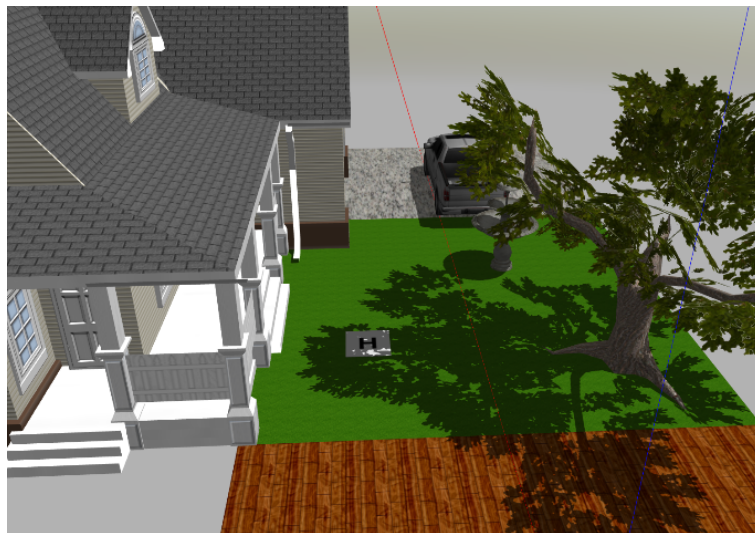


Figure 2.2 Sample Gazebo simulation environment.

The default laser scanner is a simple time-of-flight sensor (TOF), which is not useful for this thesis. Consequently, a more suitable sensor model, the VLP-16 (Velodyne Puck Lite LiDAR), was added to the original architecture. This model is based on the source package `VelodyneDescription` developed by Hallenbeck (2018). It captures all the sensor dynamics and adds Gaussian noise for increased fidelity. Figure 2.3 shows the VLP-16 rendered from its source code.

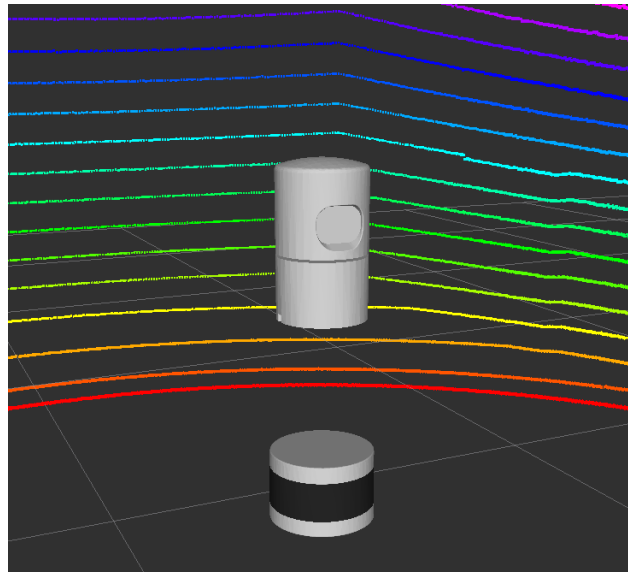


Figure 2.3 VLP16 model for ROS from Velodyne Description package (Hallenbeck, 2018).

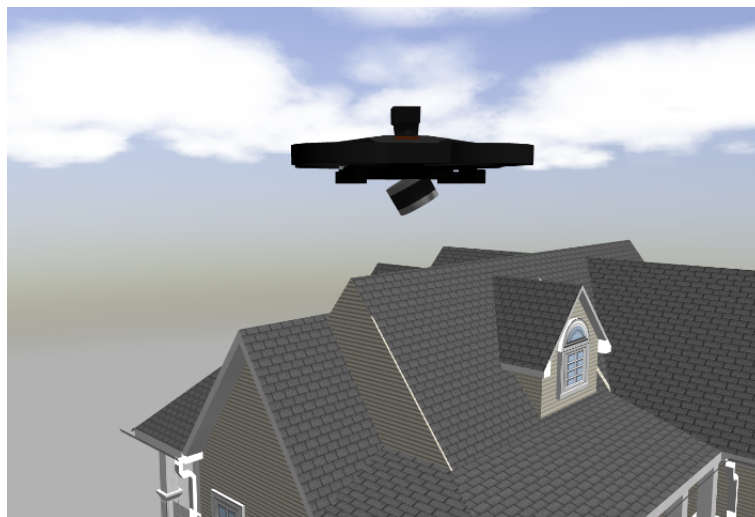


Figure 2.4 VLP16 mounted on the quadcopter.

The ‘VelodyneDescription’ package allows the user to insert a VLP-16 into Gazebo, which streams a PointCloud2 type message. This data type is a structure that contains the scans’ positions and intensities. The source code was modified to mimic the real sensor’s configuration parameters described in Section 2.2, and the mass was reduced to zero to avoid interference with the quadrotor’s flight dynamics. Figure 2.4 shows the sensor mounted on the quadrotor in Gazebo. The position has an offset to the center of mass and an inclination of 30 degrees downwards.

2.1.2 Matlab/Simulink

Matlab/Simulink sends control commands, retrieves sensor measurements, and handles all the data processing. The Simulink interface model in Figure 2.5 controls the XYZ speeds and yaw rate of the virtual quadrotor while retrieving position, attitude, and data from the laser scans.

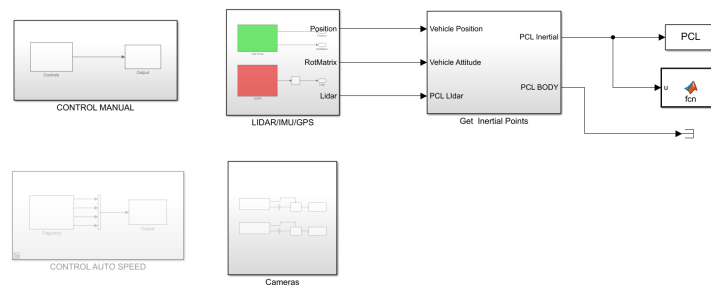


Figure 2.5 Top-level Simulink code to communicate with Gazebo.

The vehicle’s position and attitude are used to transform the LiDAR data into inertial point clouds using equation 2.1, where \vec{R}_P is the position in the inertial reference frame to the origin. In the same manner, \vec{R}_B , $\vec{R}_{S/B}$, $\vec{R}_{P/S}$ are the positions in the inertial reference frame of the vehicle relative to the origin, sensor relative to vehicle and point relative to the sensor, respectively. Figure 2.6 shows a sketch of the three mentioned frames, where I, B, and S represent the inertial frame, vehicle body-fixed frame, and sensor-fixed frame, respectively.

$$\vec{R}_P^I = \vec{R}_B^I + \vec{R}_{S/B}^I + \vec{R}_{P/S}^I \quad (2.1)$$

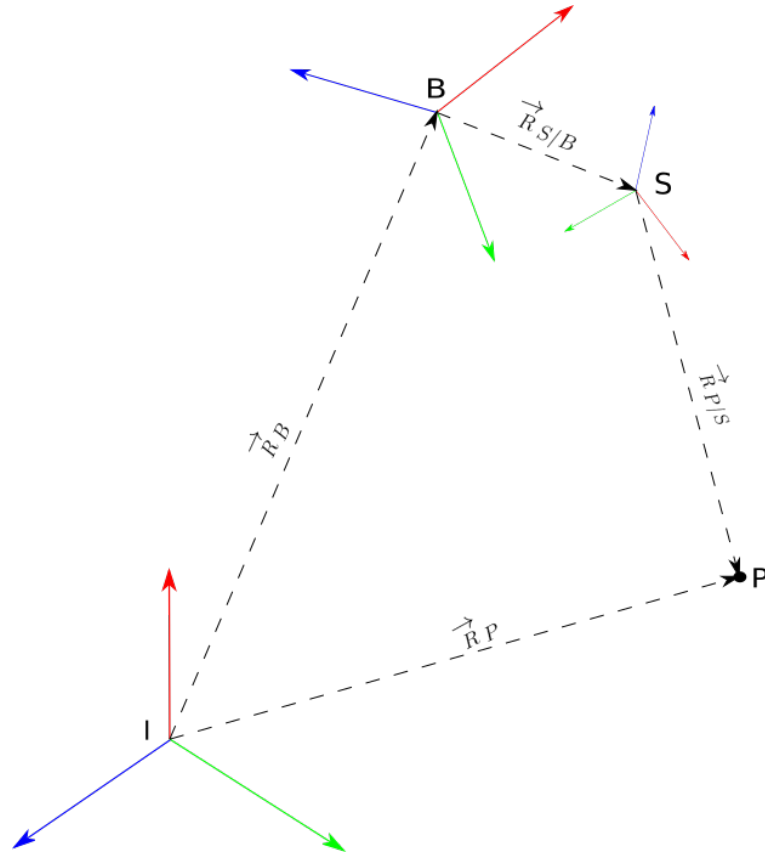


Figure 2.6 Frames for coordinate transformation.

The simulation provides the inertial position of the vehicle, quaternions for attitude and the laser scans in the sensor frame, which means the vectors used to calculate \vec{R}_P^I are not available directly. A coordinate transformation must be applied first to express every vector in the inertial frame before using Equation 2.1. The attitude quaternions and the fixed relative orientation of the sensor yield two direction cosine matrices (DCM). This is shown in Equation 2.2 which is a version of equation 2.1 that uses only available information. In Equation 2.2, the overset script on each vector represents the reference frame in which the vectors are represented. $\vec{R}_{S/B}^B$ is the position of the sensor relative to the vehicle which has to be rotated to the inertial frame. $\vec{R}_{P/S}^S$ is the position of a point

measured by the sensor in the sensor frame, so it must be rotated twice, first to the vehicle body-fixed frame and then to the inertial frame. $[DCM]_B^I$ is the standard body-to-inertial DCM and $[DCM]_S^B$ is a fixed DCM based on how the sensor is mounted on the vehicle.

$$\vec{R}_P^I = \vec{R}_B^I + [DCM]_B^I * \vec{R}_{S/B}^B + [DCM]_B^I * [DCM]_S^B * \vec{R}_{P/S}^S \quad (2.2)$$

2.1.3 Point Cloud Registration

The simulation streams information in the form of data packets or frames which contain the laser scans, attitude, and position at each time step. And, since it is necessary to have an inertial point cloud that is representative of the entire terrain rather than having several local scans, a process called Point Cloud Registration (PCR) is implemented.

The registration process has three steps:

1. **Extract:** Extract two consecutive point clouds from the data packets. This step depends on how many scans each packet includes. In this case, each packet has only one scan.
2. **Down-sample:** Usually, the quality of registration depends on the level of noise and corruption in the data. In this case, data sets are denoised using a simple grid average down-sample method. This method merges points within the same cell into one single location.
3. **Merge and filter:** Finally, both scans are saved in the same variable. The overlapped regions are filtered using a box grid filter.

Figures 2.7 to 2.9 depict a sample registration process. Figure 2.8 has four different scans at arbitrary times directly obtained from the simulation. The scans are then joined into one single point cloud as shown in Figure 2.9. For convenience, Figure 2.7 shows the vehicle's trajectory during the PCR process.

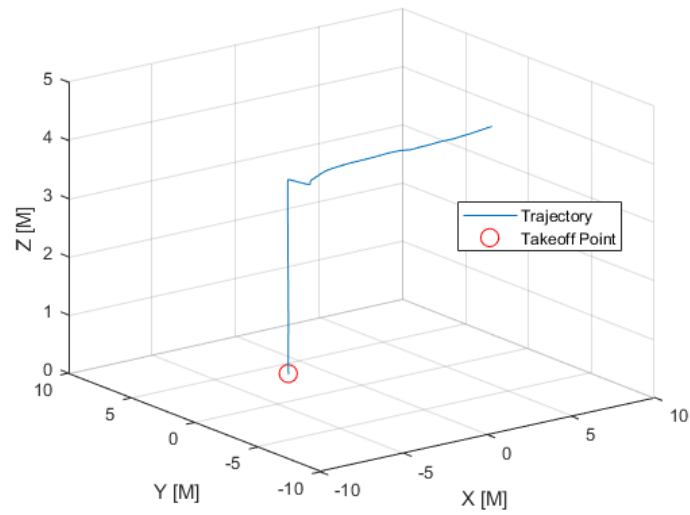


Figure 2.7 Mission trajectory for sample PCR.

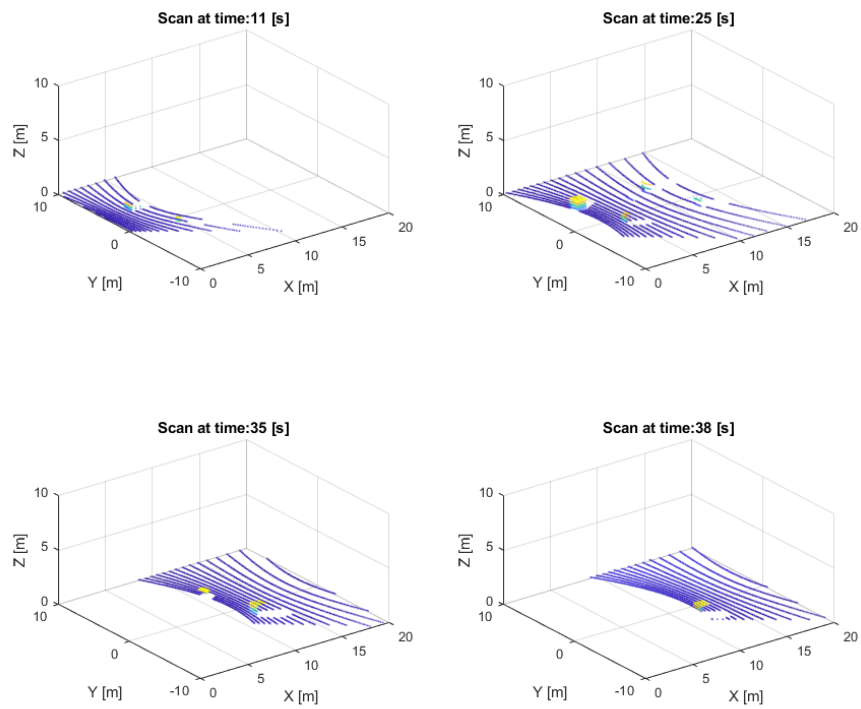


Figure 2.8 Point cloud scans at arbitrary times for sample PCR.

2.1.4 Clustering and Plane Fitting

Clustering is the process of examining points and grouping them according to some metric, the Euclidean distance, in this case. The main idea is to classify the scans in a way that points in the same cluster (group) have small distances between them while having significant distances to points in other clusters. Plane fitting the ground is a form of clustering which uses a variation of the Random Sample Consensus algorithm (RANSAC) called M-estimator sample Consensus (MSAC) (Torr & Zisserman, 2000). Figure 2.10 shows the clustering and plane fitting tools applied to the point cloud in Figure 2.9. The ground is fitted to a plane (in black) and the obstacles are grouped in four clusters.

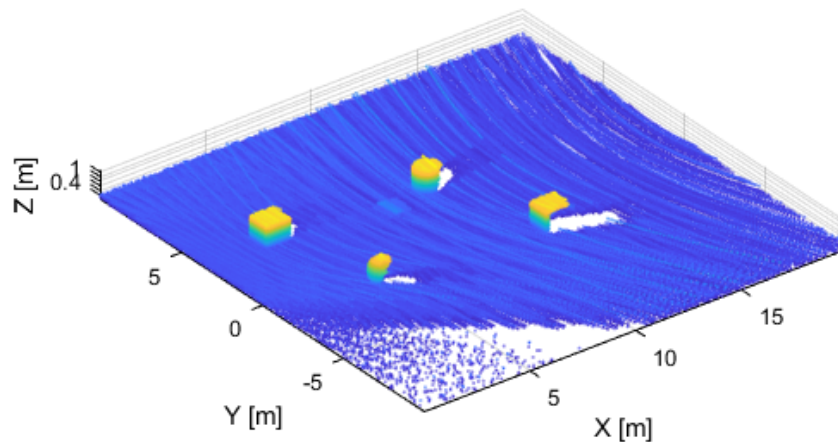


Figure 2.9 Sample point cloud after applying PCR algorithm.

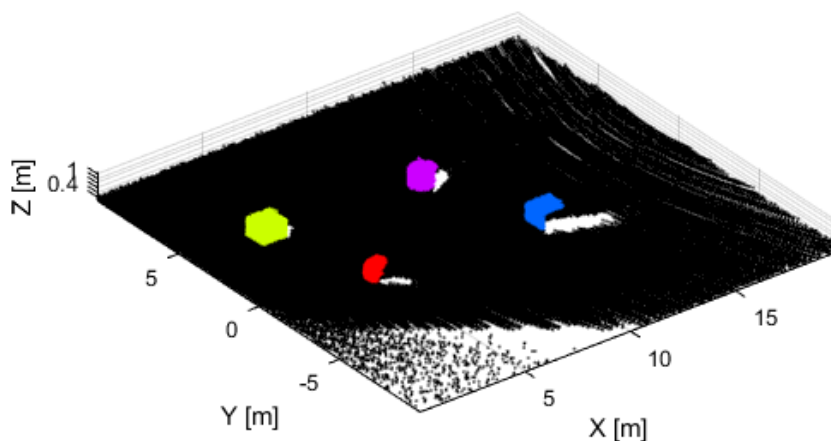


Figure 2.10 Point cloud after ground filtering and clustering.

2.2 Experimental Environment

The real environment was assembled in a testing room equipped with motion-sensing cameras and data acquisition computers. The room allows measuring position and attitude with high precision while pairing all the readings to a single time frame. The cameras replace the traditional GPS and IMU, and a laser scanner is used to gather information on the terrain.

2.2.1 Sensors

The sensors includes the LiDAR and the motion-sensing camera network. The VLP-16 LiDAR has 16 lasers rotating at 5Hz-20Hz with a range of 100 meters, 360 degrees horizontal field of view and 30 degree vertical field of view. The sensor has three return types: strongest, weakest and dual. Since each laser is shot twice every step, it can register either the strongest, weakest or both returns. Figure 2.11 shows some pertinent sensor information available in the user manual (*VLP-16 User Manual*, 2017).

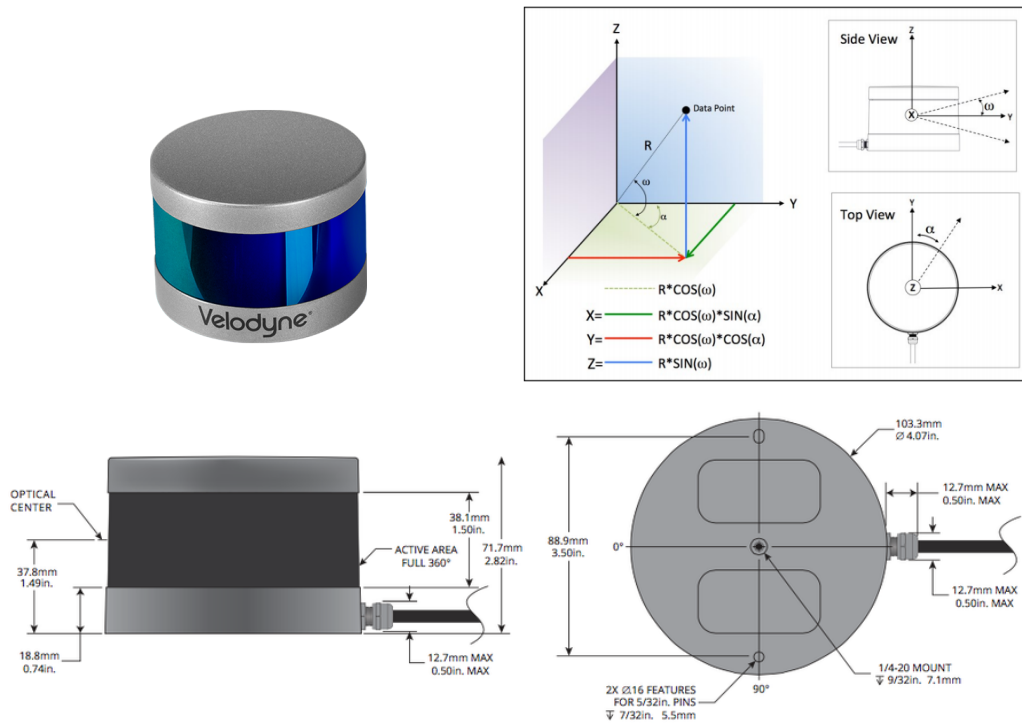


Figure 2.11 VLP-16 Information (*VLP-16 User Manual*, 2017).

Figure 2.12 shows the operating parameters used for all the tests. The field of view was set to 45 degrees left and right of the y axis to simulate the field of view of a sensor mounted on an UAV. The rate was set to the default 600 RPM and the return type to strongest. The VICON Vantage system is a network of cameras (up to 12) that track the position of objects inside their field of view. Figures 2.13 and 2.14 show the VICON Vantage setup used for the tests and an individual camera, respectively. The user manual (*VICON Vantage Reference*, 2016) provides more information on the equipment.

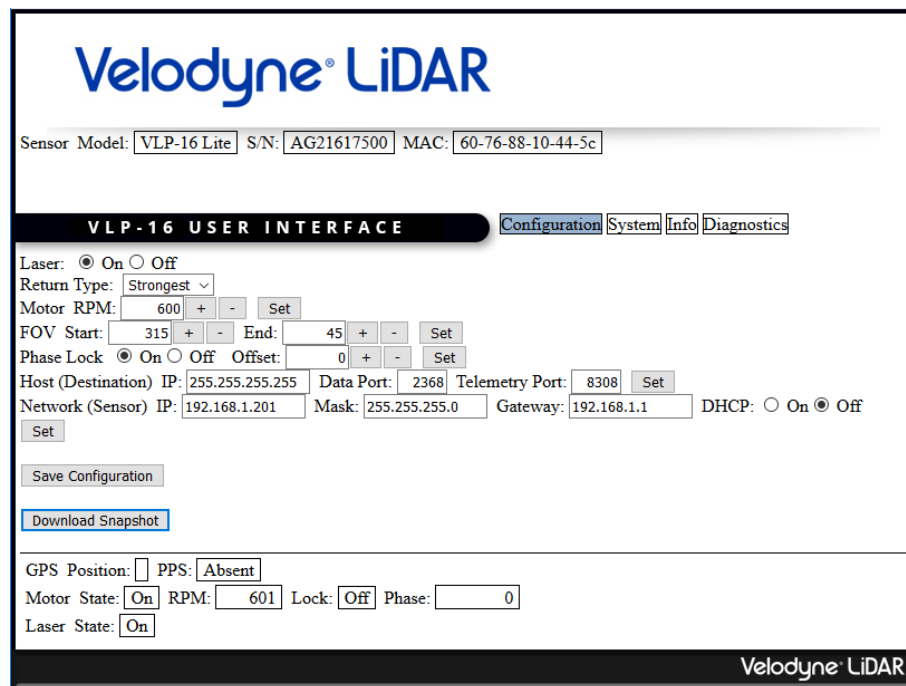


Figure 2.12 Velodyne LiDAR GUI for setting operation parameters.

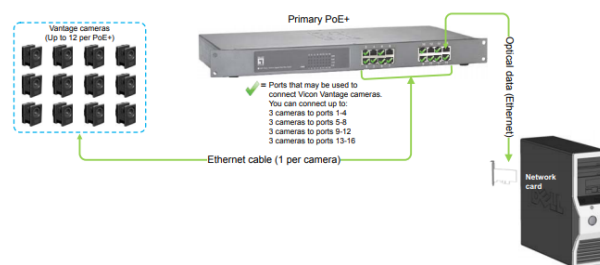


Figure 2.13 VICON Vantage setup and network (*VICON Vantage Reference*, 2016).



Figure 2.14 VICON single camera for motion sensing.

2.2.2 Data Acquisition Setup

Three computers were needed to acquire and process all the data in a near-to-real-time manner: The VICON data acquisition computer extracts the information from the VICON Vantage camera system. Then, the interface computer exports the data to Matlab and sends it to the third computer. Finally, the LiDAR data acquisition computer, which is connected directly to the VLP-16, receives the data from the scanner and the Matlab-VICON computer and uses it to obtain inertial point clouds. This data transfer is achieved using two separate networks that interface in the Matlab-VICON computer.

Figure 2.15 shows a diagram of the system and networks.

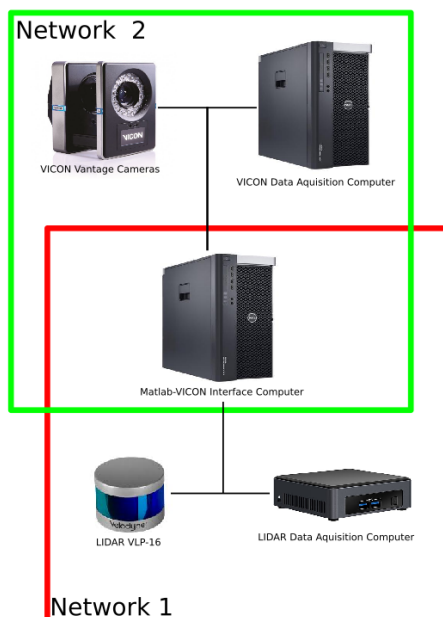


Figure 2.15 VICON-LiDAR system general structure.

2.2.3 Motion Sensing

The VICON cameras track motion using special markers (pearls) placed on the object. (*VICON Vantage Reference*, 2016) suggests the pearl arrangement should be non-symmetric to clearly identify the orientation of the object. Figures 2.16 and 2.17 show a diagram of the pearl distribution and the actual pearls on the sensor, respectively.

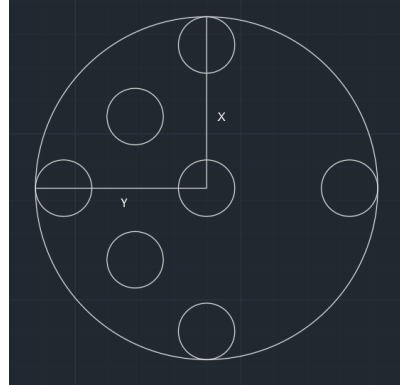


Figure 2.16 Diagram showing VICON pearls distribution.



Figure 2.17 VICON pearls on LiDAR.

In addition to placing the pearls, the VICON software must be calibrated to track the object. The calibration process includes creating a virtual entity in the software, attaching a body-fixed frame, and defining specific measurement parameters. Figure 2.18

shows the virtual entity created to track the position and attitude of the sensor. Figure 2.19 shows the same object inside the control volume while being tracked by the cameras.

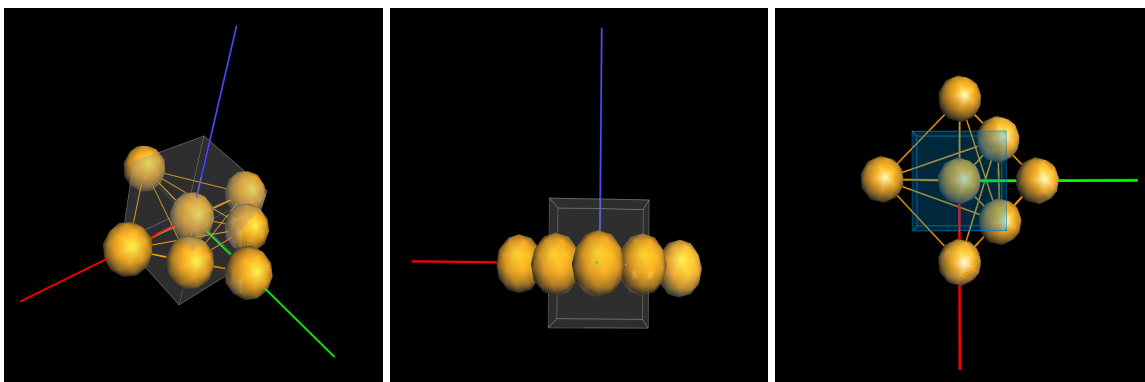


Figure 2.18 VICON virtual object.

Attaching the sensor to a flying vehicle posed a logistic challenge. For this reason, the LiDAR was mounted on a tripod to achieve a high vantage point and the desired sensor orientation. Then, it was moved around using a simple cart, as seen in Figure 2.20.

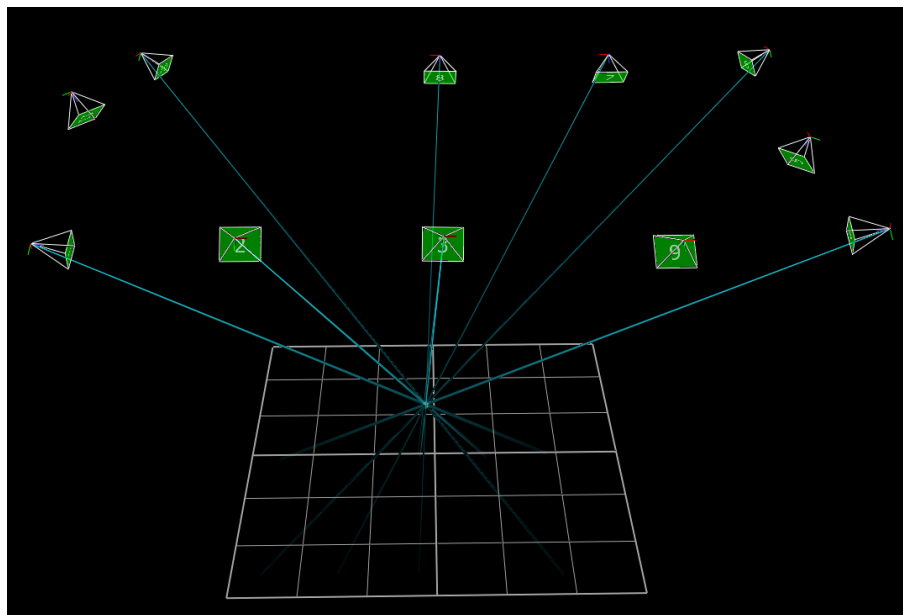


Figure 2.19 VICON Vantage system tracking object.



Figure 2.20 Lidar-tripod-cart setup.

2.2.4 Control Volume

The control volume was delimited by a grid, as seen in Figure 2.21, which eases obstacle placement. It is useful for repeating tests and rearranging obstacles with minimum error. Anything outside of the control volume is filtered out and not considered for analysis. Figure 2.22 presents a sample terrain assembled to test the real environment setup. Figure 2.23 displays the point cloud after the registration process, and Figure 2.24 presents the trajectory of the sensor inside the grid. It is worth noting that the attitude information is measured from outside the vehicle (cameras act as an external observer) contrary to the classic in-vehicle measurements (IMU reports observations from inside the aircraft). Therefore, the attitude data is not measured in the traditional roll, pitch, yaw Euler angles.



Figure 2.21 Control volume and grid.



Figure 2.22 Snapshot of sample terrain used to the test real environment.

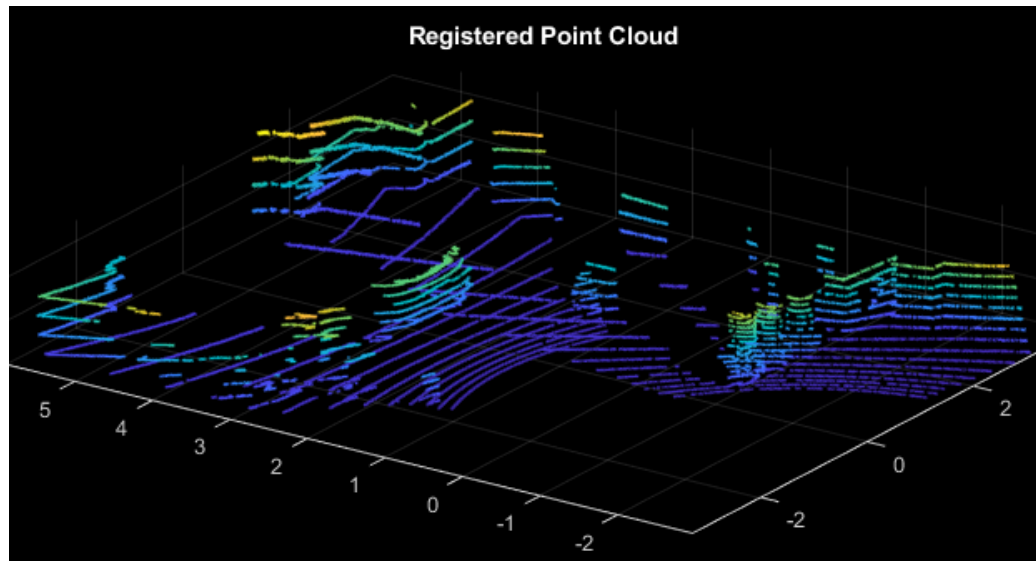


Figure 2.23 Point cloud of sample terrain collected during test run.

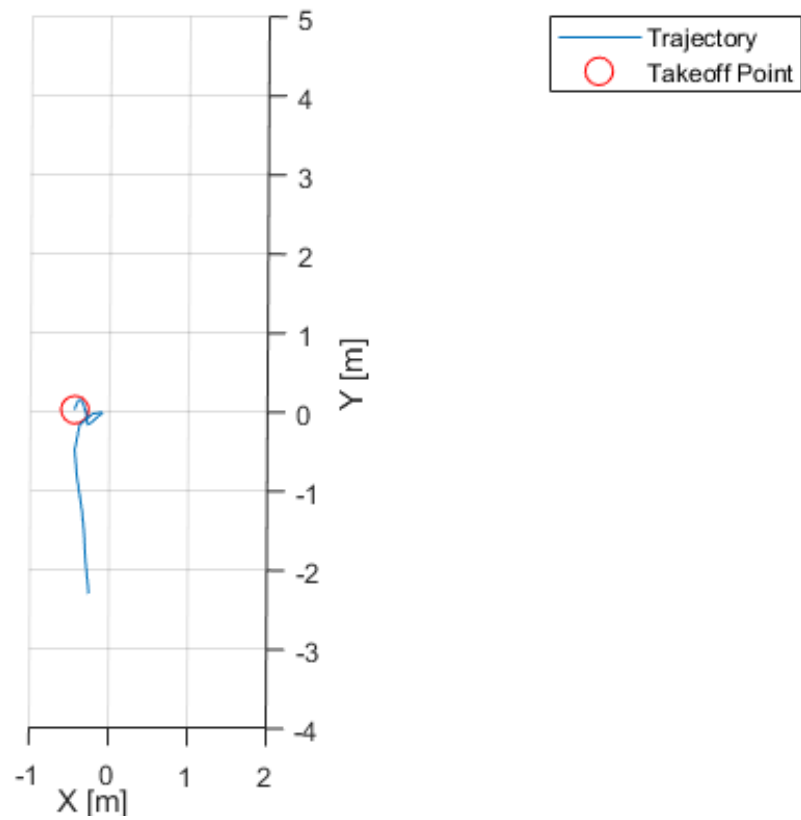


Figure 2.24 Sensor trajectory for test run.

3. Mapping Techniques

This chapter provides a brief theoretical background of the mapping techniques and a detailed description of their corresponding implementation. All the implementation algorithms run on an infinite loop and update the map when new information is received, except for Quadtree, which is implemented to a final point cloud after registration. Figures 2.7 and 2.9 depict the trajectory and sample terrain used for developing all the implementation codes. This chapter also details the development of ALTE done by Embry-Riddle, from the initial mathematical concept to the current 3D version.

3.1 Quadtree

Quadtree is a type of tree data structure in which each node has exactly four children. It is commonly used to partition a 2-D space into quadrants for ease of analysis (*Quad-Tree*, n.d.). Figure 3.1, presents the structure that Quadtree builds from data points. These points are not necessarily Cartesian coordinates, but for the purposes of this thesis it refers exclusively to physical locations in space. Ultimately, Quadtree reshapes and organizes the information so it is easier to process by other algorithms (e.g., obstacle detection and collision avoidance). These algorithms analyze only nodes that contain significant amounts of points inside them, thus, reducing the number of computations. Figure 3.2 shows an example of Quadtree applied to collision avoidance. The navigation algorithm filters out the regions with low point densities to generate a collision-free trajectory.

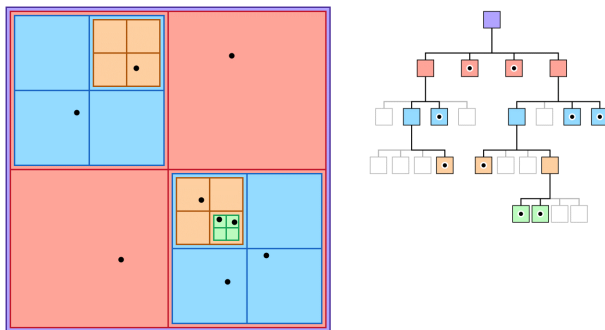


Figure 3.1 Quadtree concept (*GKQuadtree*, n.d.).

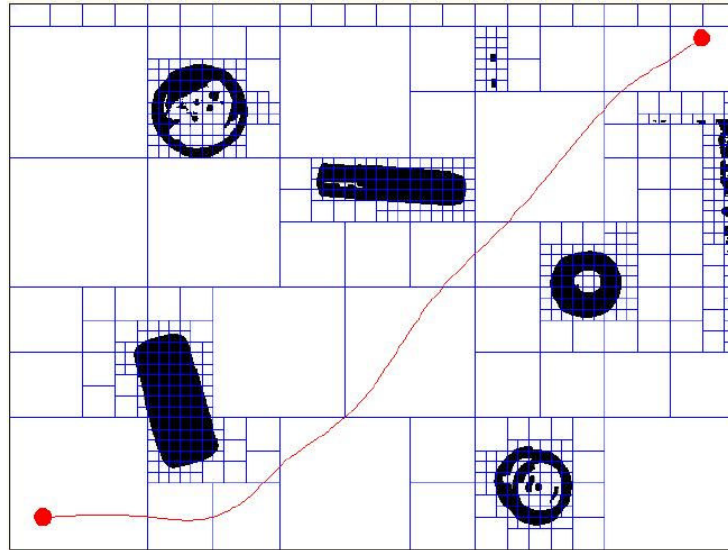


Figure 3.2 Quadtree map used for collision avoidance (Shojaeipour et al., 2010).

Quadtree is the base of all tree-like data structures. Many mapping algorithms, including Octomap and ALTE, base their data handling principles in this algorithm. The mapping algorithm is heavily based on Kirill Pankratov's function (*Quadtrees*, n.d.). It works as follows:

1. **Divide:** The algorithm divides the space into four quadrants. Then, it divides each quadrant in such a way that it contains no more than a given number of children elements (division threshold). In this case, any point is considered a children element.
2. **Check:** If a quadrant contains more points than the maximum allowed, it is divided again. If a quadrant contains fewer points than the given threshold, the algorithm will not divide any further.
3. **Repeat:** This process recurs until every quadrant meets the given criteria (number of children elements less than the given threshold)

Implementation

The implementation code adds an occupancy logic to the mapping code, which transforms the data nodes in the tree into binary functions, free or occupied. This process creates an occupancy grid with variable-sized cells, but without the ability to recursively update the map. Instead, it runs once on the final data set. Figure 3.3 shows the implementation algorithm running once on a single point cloud. The mapping algorithm is using the points up to that moment in time to generate a map. This map cannot be updated or changed since Quadtree does not have this feature. The sub-figures present frames of the map at certain points during the mapping process.

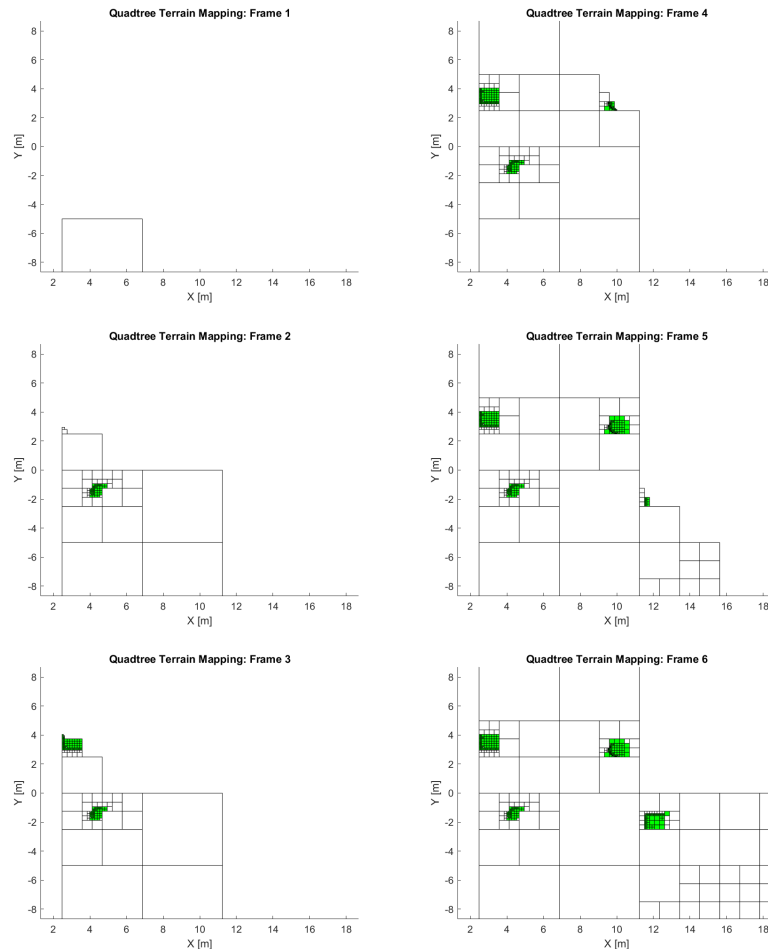


Figure 3.3 Quadtree applied to final point cloud from figure 2.9.

3.2 Octomap

Octomap uses Octree-based data structures to partition space using the same principles as Quadtree, but adding a third dimension to the analysis. The Octree data structure is a hierarchical structure used for subdivision of an environment into cubic volumes called voxels. For a given map volume, space recursively subdivides into eight voxels until achieving the desired resolution (voxel size). This subdivision generates the tree-like structure depicted in Figure 3.4 (*RoboticsOccupancyMap3D class*, n.d.).

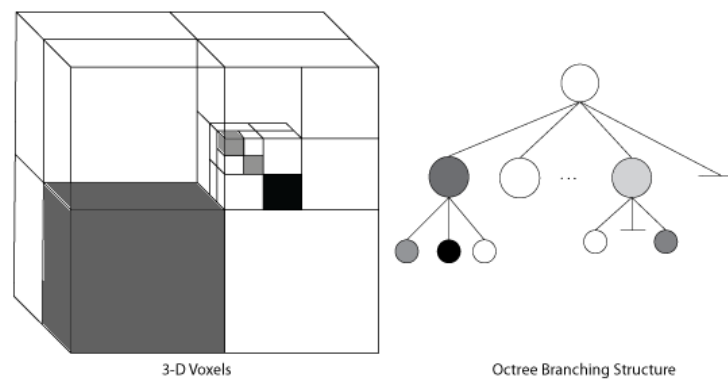


Figure 3.4 Octree voxel concept (*RoboticsOccupancyMap3D class*, n.d.).

Octomap was developed by (Hornung et al., 2013) to use the Octree principles for mapping. It sets an occupancy 3D map based on the readings of a laser scanner. The main characteristics of this mapping technique are the following:

1. **Probabilistic representation:** It performs probabilistic occupancy estimation to ensure updatability and sensor noise rejection. The occupancy estimate depends on the current measurement, prior probability, and prior estimate.
2. **Free and unknown space:** Octomap explicitly represents free space, thus, resolving the ambiguity of free and unknown space, as shown in Figure 3.5.

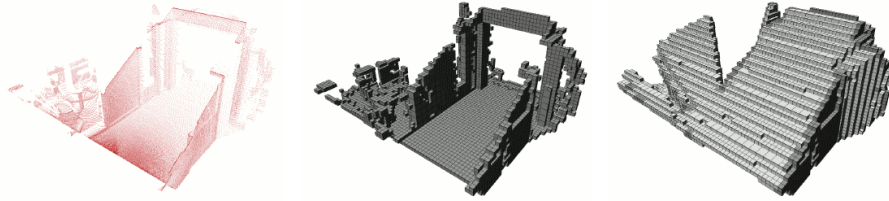


Figure 3.5 From left to right: Point cloud sample, terrain representation, free space representation (Hornung et al., 2013).

3. **Boolean occupancy:** Octomap uses boolean occupancy states that allow compact representations by merging parent and children nodes with the same state of occupancy.
4. **Adaptable occupancy estimation:** The model has a built in clamping update policy that defines upper and lower bounds of occupancy estimate. It modifies the number of updates that are needed to change the state of a voxel. This characteristic allows the model to update and keep up with fast changes in data streams.
5. **Resolution:** Upper nodes yield coarser maps, while lower nodes give access to finer representations. The resolution can be changed globally after concluding the mapping process. Therefore, the map has no local multi-resolutions. Figure 3.6 shows the same map on three different resolutions. The Matlab implementation only allows one tuning parameter: the map's cells per meter.

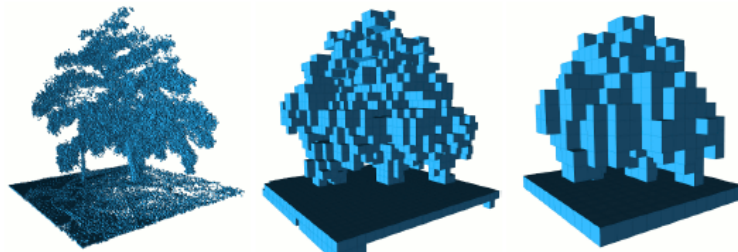


Figure 3.6 Octomap representation at 3 different resolutions (Hornung et al., 2013).

Implementation

Matlab's Navigation Toolbox provides Octomap's code architecture. The `occupancyMap3D` function creates a map object that interacts with various other functions like `insertPointCloud`, essential to the implementation. Matlab provides all the coding tools to generate and manipulate maps. (*Occupancy Map 3D*, n.d.) provides more information on the available functions. The implementation algorithm was developed with the aid of the simulation described in Chapter 2. This code reads information from the simulation and adds one point cloud to the map at each timestep. Figure 3.7 shows the time-lapse of the mapping process.

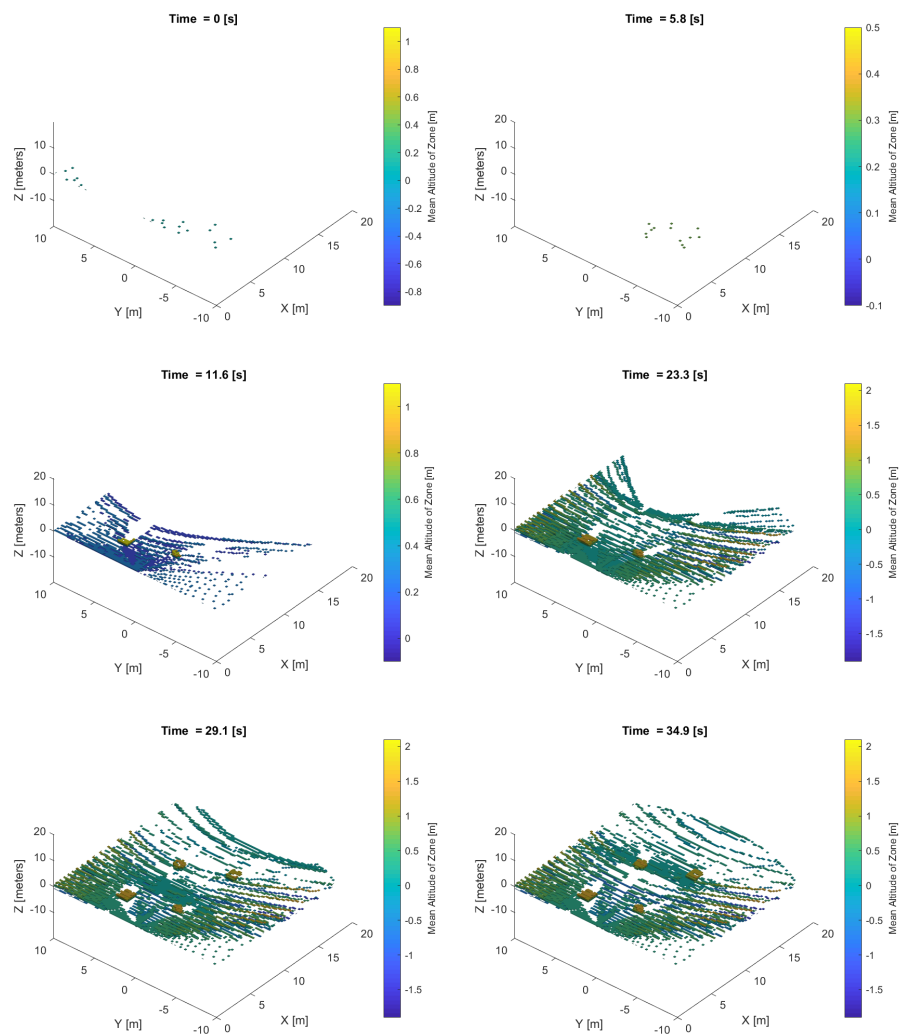


Figure 3.7 Octomap applied over time to data from Figure 2.9.

3.3 Adaptive Learning Terrain Estimation

ALTE employs multiresolution-based mathematical learning theory developed in (Binev, Cohen, Dahmen, A. DeVore, & Temlyakov, 2005) and applied to terrain estimation as in (Prazenica, Kurdila, Sharpley, & Evers, 2006). It addresses the problem of estimating a function without making any assumptions on the form of the underlying probability distribution. (Prazenica et al., 2006) originally developed and applied their algorithm over a two-dimensional domain, (Vergara, Tiwari, Prazenica, & Henderson, 2019) presented a more current application for three-dimensional environments, where a 3D version of the original algorithm was developed. These references include the development process of all the algorithms, from the mathematical learning principles to the latest 3D version, and real applications implemented in research programs. Sections 3.3.1 and 3.3.2 provide more detail on 2D-ALTE and 3D-ALTE algorithms.

3.3.1 2D-ALTE

2D ALTE generates maps in terms of piece-wise constant functions that represent a rectangular sub-domain, as seen in equation 3.1.

$$f(x,y) = \sum_I c_I * \chi_I(x,y) \quad (3.1)$$

where χ_I is the constant function over I and c_I is the average height of all point in the sub-domain. Figure 3.8 illustrates the partitioning of two-dimensional domain into sub-regions.

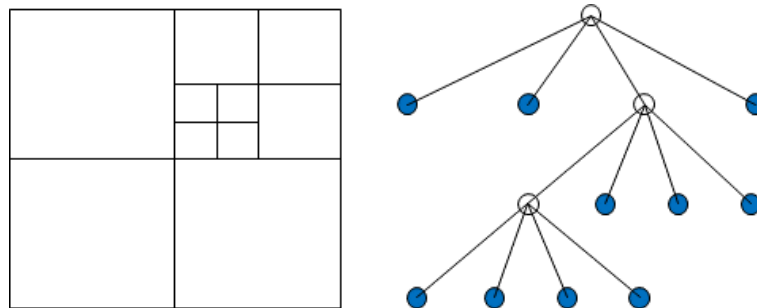


Figure 3.8 Adaptive partitioning of a 2D domain (Prazenica et al., 2006).

Equation 3.2 represents the decision to divide a sub-domain I based on the variance of the heights of the points inside the domain.

$$\sum_{z \in I} (z_i - c_I)^T (z_i - c_I) > \tau_m \quad (3.2)$$

where z_i is the altitude of the i th point inside the domain and τ_m is the variance threshold. Thus, if the variance exceeds a specified threshold, the sub-domain divides into four higher resolution sub-domains. This approach uses the basis function's height as an occupancy property.

The algorithm has three user-defined parameters. The previously mentioned variance threshold and a minimum-points threshold are used to decide on domain subdivision. The third parameter puts a ceiling on the number of divisions, which means the user can define the maximum amount of divisions allowed per cell, thus setting a maximum resolution limit. Section 5.2.2 shows the effect of varying the tuning parameters.

Figure 3.9 shows a sample point cloud and images of a virtual urban environment from past development work. This point cloud was used to generate terrain representations, as shown in Figure 3.10, where the domain is subdivided into rectangular regions, and a constant function approximates the average terrain altitude. If the data that reside in a specific region show significant variance in height, the algorithm divides the region into four sub-regions and approximates the terrain in each sub-region as a different constant function (Prazenica et al., 2006).

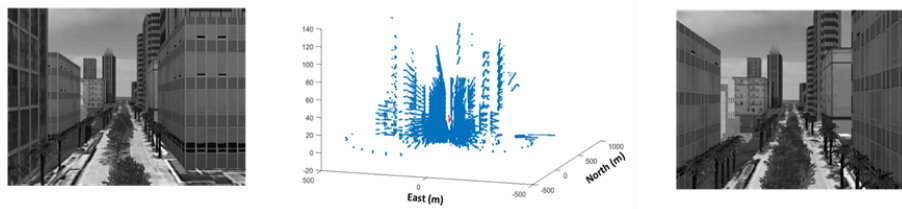


Figure 3.9 Simulated point cloud from virtual urban environment (Vergara et al., 2019).

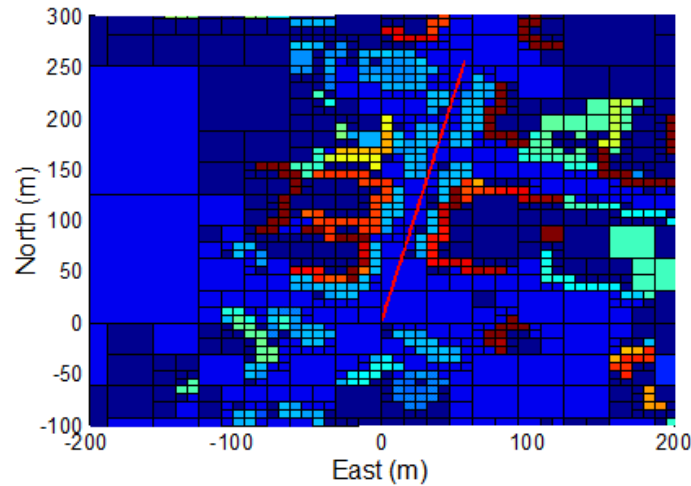


Figure 3.10 2D adaptive terrain map (Vergara et al., 2019).

2D ALTE was applied to obstacle identification by researchers to complement an autonomous flight control system. A simplified map was created by filtering out the terrain based on a height threshold and identifying the centroids and radii of distinct obstacles. Figure 3.11 shows the flight data collected during the test, and Figures 3.12 and 3.13 show the resulting adaptive map and obstacle map. This obstacle map was used in (Rivera, 2018) to test path planning algorithms.

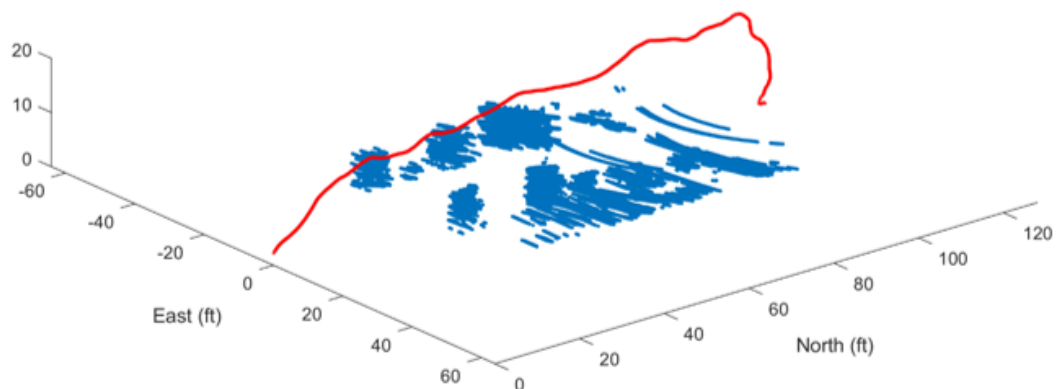


Figure 3.11 Point cloud from real flight data (Vergara et al., 2019).

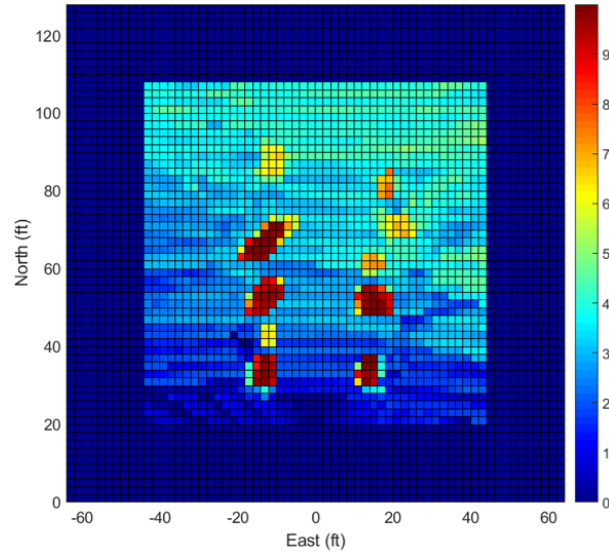


Figure 3.12 2D adaptive terrain map from real flight data (Vergara et al., 2019).

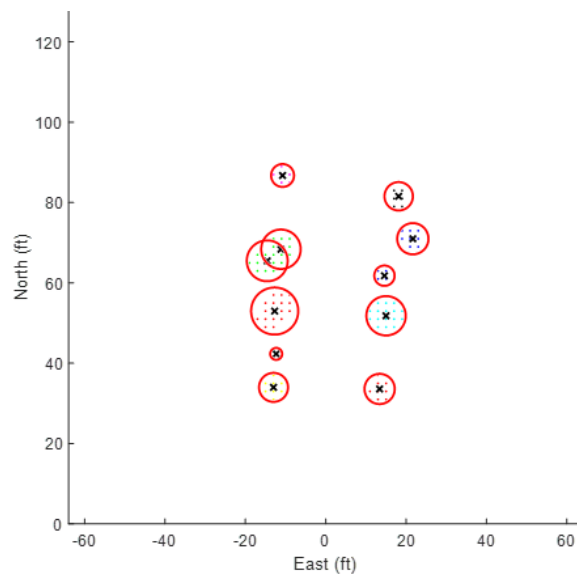


Figure 3.13 Obstacle map from real flight data (Vergara et al., 2019).

The 2D-ALTE implementation code has a similar structure to Octomap's implementation. The data frames are analyzed by the mapping algorithm one by one while registering the global point cloud in the process. Figure 3.14 shows plots of the current map at arbitrary moments during the mission.

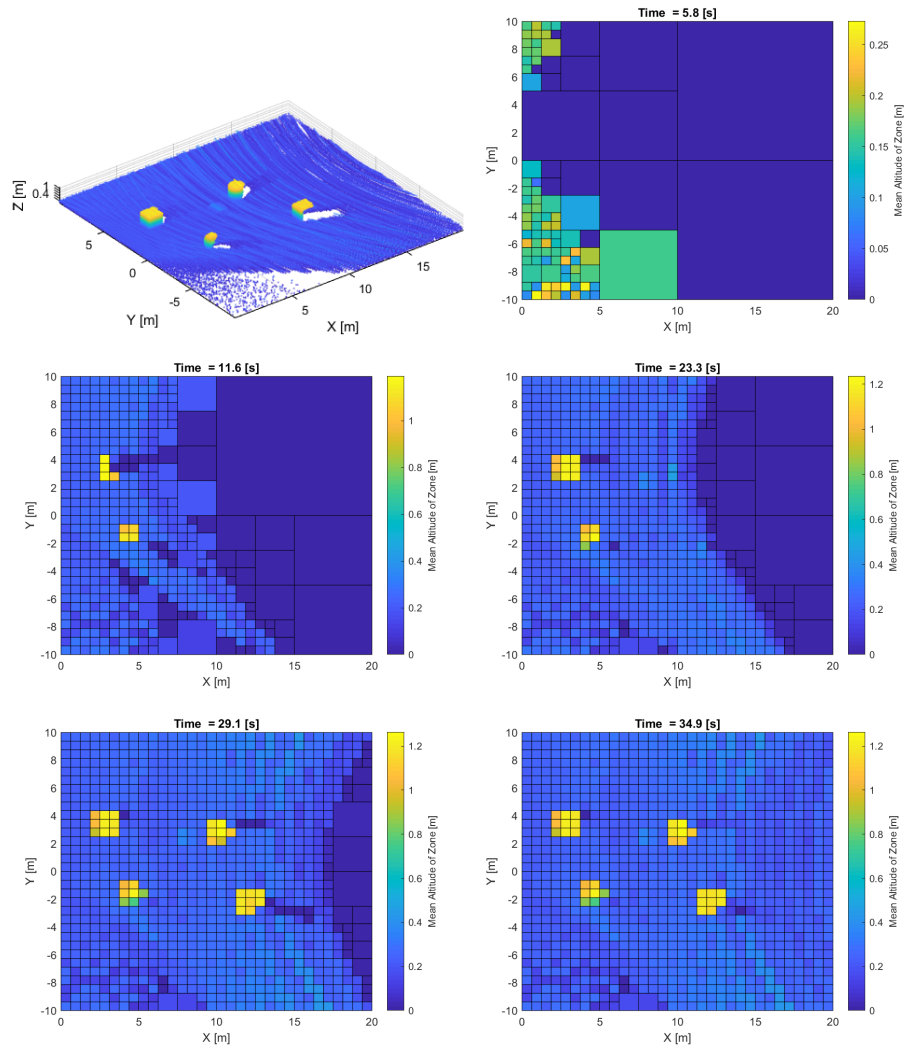


Figure 3.14 2D Adaptive Learning applied over time to data from Figure 2.9.

3.3.2 3D-ALTE

3D-ALTE represents the terrain as 3D binary functions instead of piece-wise constant functions. This means each sub-domain is either occupied or unoccupied/unknown. This algorithm resembles a multi-resolution octree representation. Therefore, an overall three-dimensional domain is defined and then adaptively partitioned into sub-domains that resemble cubes or rectangular prisms.

The decision to subdivide a given cubic region into eight sub-domains is based on the number of points inside it. Once the number of points in a sector exceeds a specified threshold, it is partitioned. A sub-domain is marked as occupied if the number of points

within it exceeds a specified number (Vergara et al., 2019). This algorithm eliminates variance from the analysis because it incorporates the third dimension completely. Recall, the 2D version calculates the variance of points' heights within a cell and uses it as a division criterion. The three user-defined parameters include thresholds for deciding cell division, cell occupancy, and the maximum number of divisions. Section 5.2.2 shows the effects of these parameters on the results.

A three-dimensional terrain representation of the urban scene in Figure 3.9, using the 3D version of the Adaptive Learning algorithm, is shown in Figure 3.15. This representation is in terms of occupied cubes at different resolution levels, with color variation to denote the maximum height of each occupied region. ERAU researchers applied the 3D ALTE concept to the flight data depicted in Figure 3.11 by fixing the cell's third dimension, as seen in 3.16. Figure 3.17 shows the resulting maps at several moments during the mission.

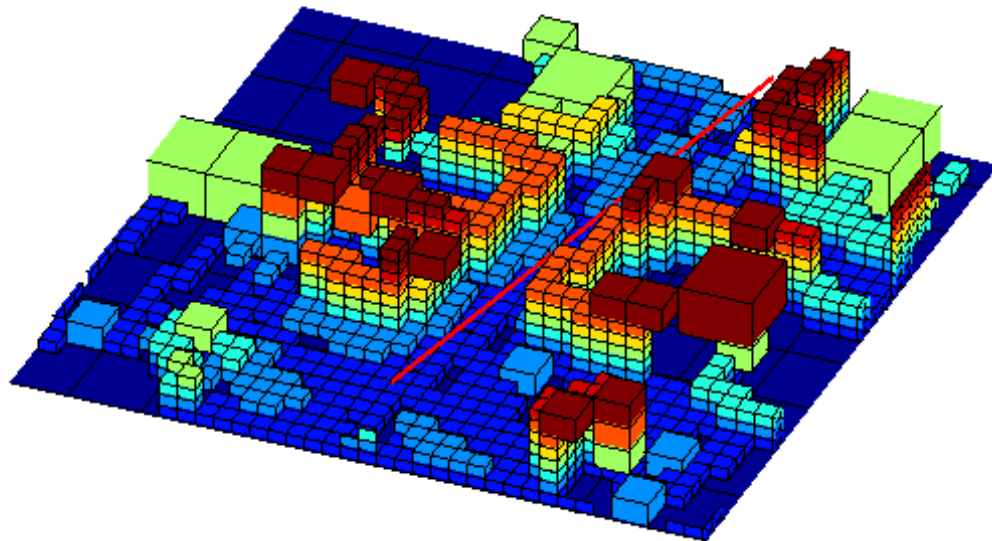


Figure 3.15 3D adaptive terrain map (Vergara et al., 2019).

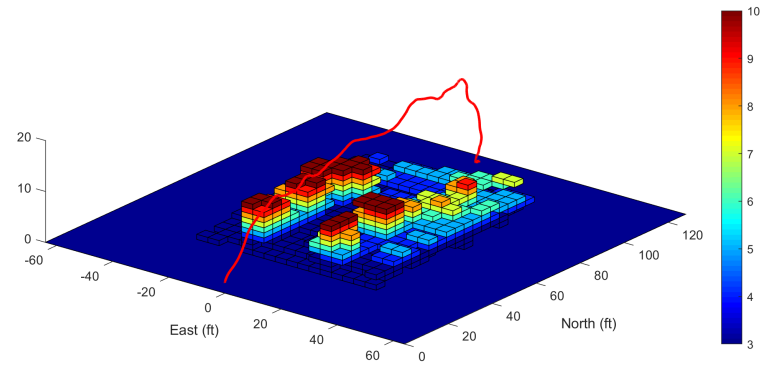


Figure 3.16 3D adaptive terrain map with fixed cell altitude (Vergara et al., 2019).

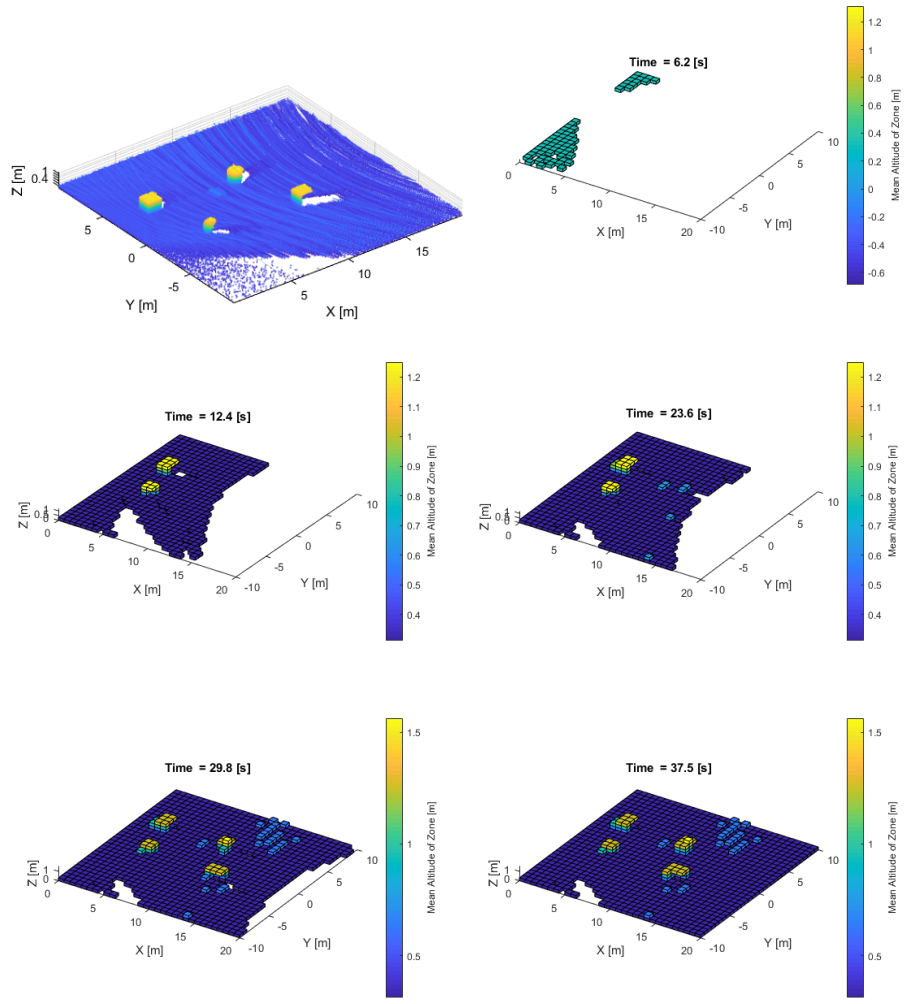


Figure 3.17 3D Adaptive Learning applied over time to data from Figure 2.9.

4. Terrain Mapping Results

This chapter details the process of building terrain estimates (maps) for the two proposed scenarios. It generates and documents all the maps from real and simulation environments. First, the terrain is built/modeled. The simulation is capable of representing a variety of obstacle setups, including real situations like the package delivery scenario used in Section 4.1.4. The real environments were limited by the objects at hand such as foam to simulate debris and marine buoys for vertical objects. The terrain was first scanned to get a global point cloud. The registration process was carried out online, meaning the LiDAR DAQ computer generated the point cloud in real-time during the mission, for both real and simulated environments.

On the other hand, the mapping process was done offline due to complications in the implementation, especially in the real terrain. Even though the map was not generated online, the mapping process used the data frames, rather than a final point cloud, processed in a loop, mimicking a real-time implementation. A plane fitting algorithm was implemented to increase mapping efficiency.

Fast dynamics due to sudden maneuvers introduce error in the point cloud registration process. This desynchronization between the LiDAR and the roll/pitch/yaw measurements happens when the dynamics are faster than the sample rate of the sensors. In this case, the sampling rate of the cameras caused some information to be lost during these maneuvers. To mitigate this effect, the maneuvers were slow and steady. However, some sudden changes could not be avoided, so they were filtered out during the mapping process.

4.1 Simulated Terrains

4.1.1 Urban Navigation: Terrain 1

The first terrain, depicted in Figure 4.1, has 3 complex obstacles and 2 walls to test the algorithm's capabilities to generate detailed maps. Figures 4.2 and 4.3 show the position/attitude over time and trajectory, respectively.

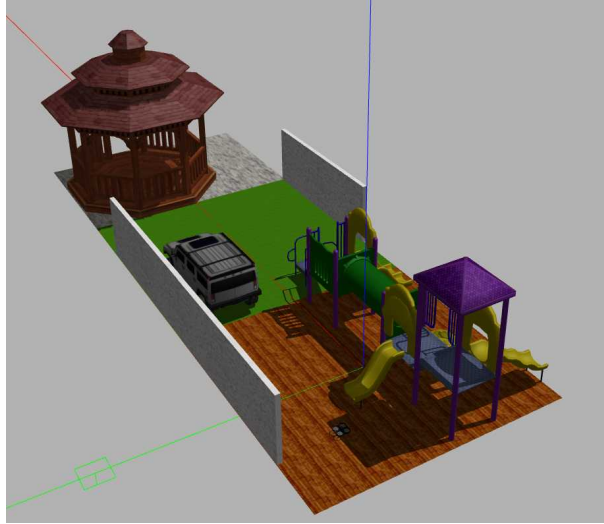


Figure 4.1 Terrain snapshot of simulated Urban Navigation, Terrain 1: Complex Objects.

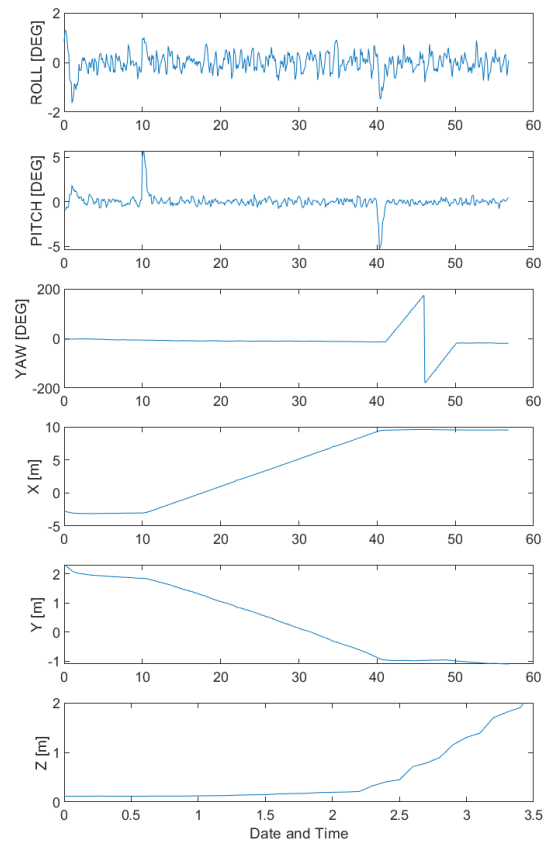


Figure 4.2 Recorded sensor position and attitude during simulated Urban Navigation: Terrain 1.

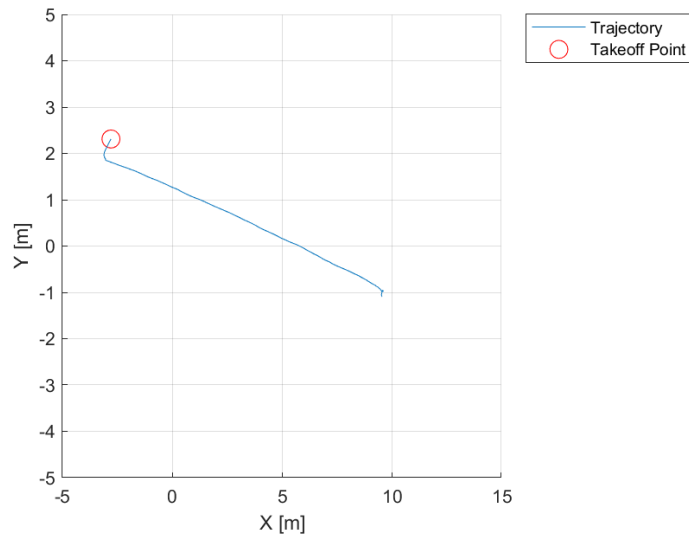


Figure 4.3 UAV trajectory for data collection of simulated Urban Navigation: Terrain 1.

Figures 4.5 and 4.6 show the mapping results from the point cloud in Figure 4.4. The resulting octomap captures all the geometries but has gaps due to its resolution of 5 cells-per-meter. In contrast, the ALTE map does not have gaps even when the maximum resolution of the map is set to a high value of 6. The occupancy threshold was set to 10 points-per-cell and the division threshold to 20 points-per-cell.

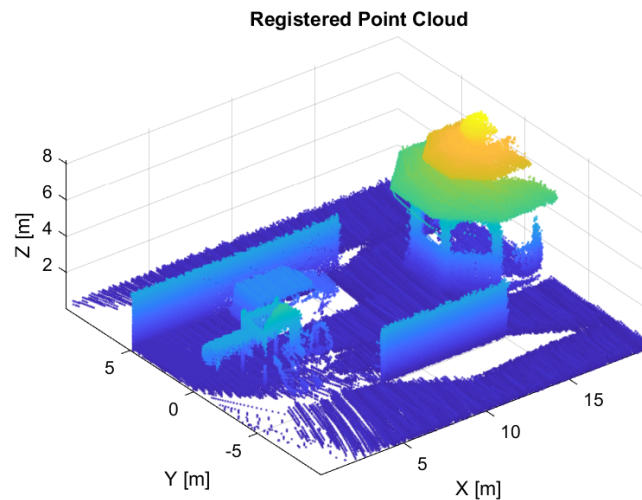


Figure 4.4 Simulated Urban Navigation: Terrain 1 point cloud after registration process.

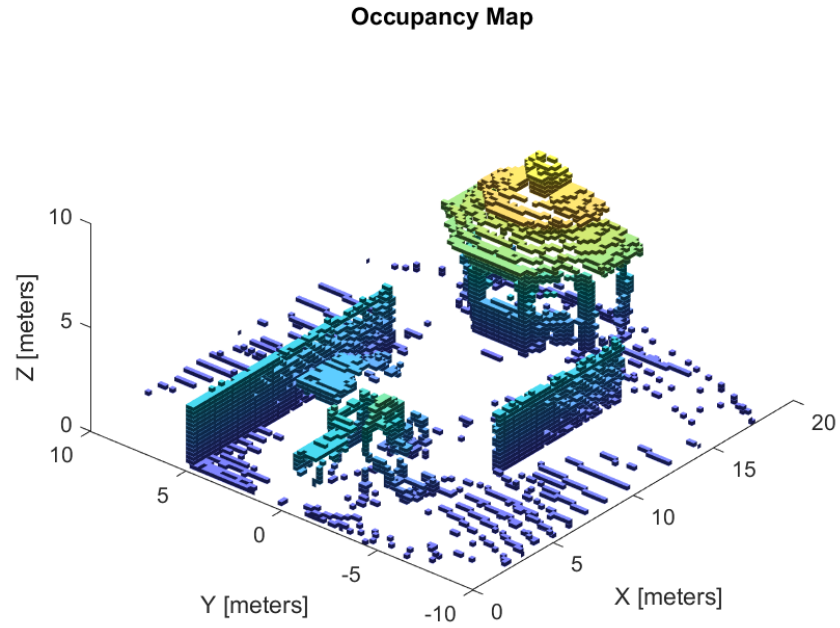


Figure 4.5 Octree applied to simulated Urban Navigation: Terrain 1.

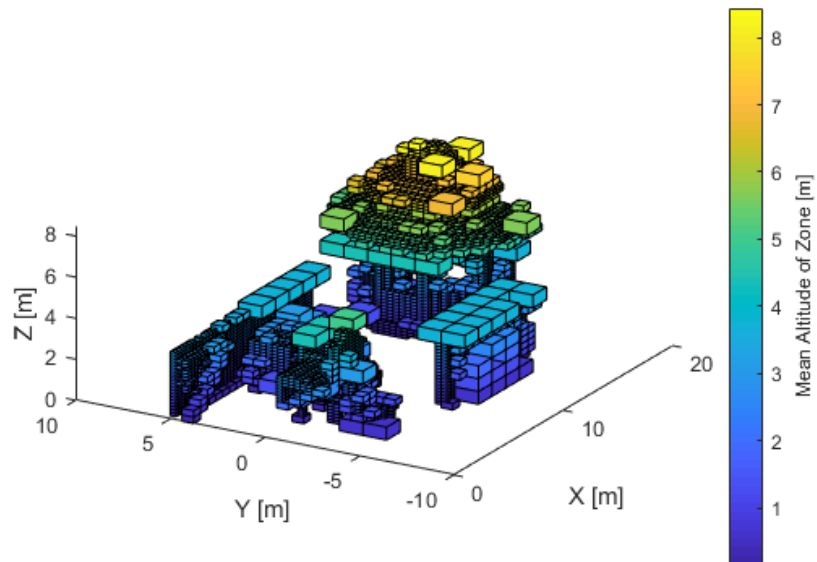


Figure 4.6 3D-ALTE applied to simulated Urban Navigation: Terrain 1.

4.1.2 Urban Navigation: Terrain 2

The terrain in Figure 4.7 is used to test the algorithms' capabilities to map overhanging structures. Figures 4.8 and 4.9 show the position/attitude and trajectory, respectively.

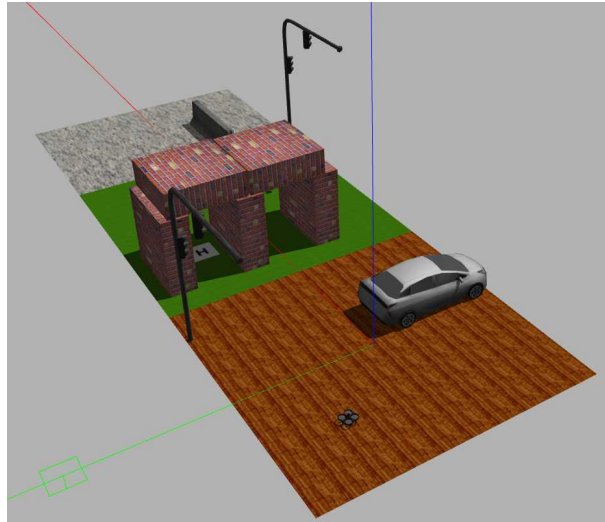


Figure 4.7 Terrain snapshot of simulated Urban Navigation, Terrain 2: Overhangs.

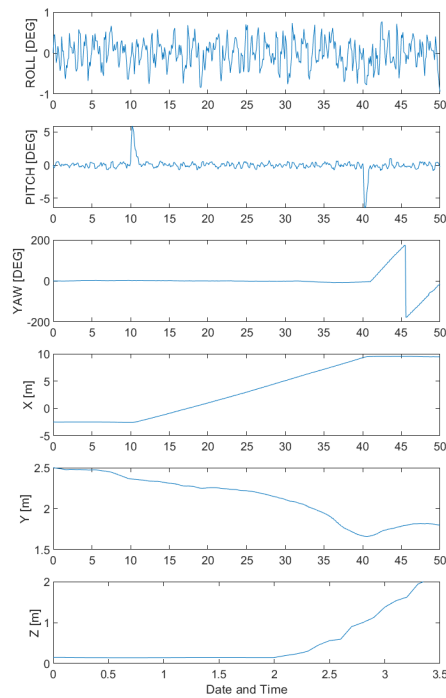


Figure 4.8 Vehicle position and attitude for Urban Navigation: Terrain 2.

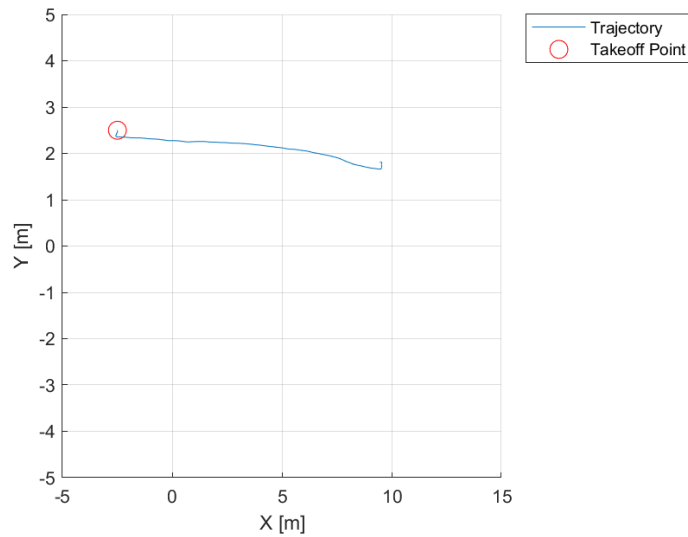


Figure 4.9 UAV trajectory for data collection of simulated Urban Navigation: Terrain 2.

The map in Figure 4.11 demonstrates Octomap's capability to generate high fidelity terrains. Traditionally, overhanging structures and slender horizontal objects are problematic for mapping algorithms. The ALTE map, shown in Figure 4.12, also captures some horizontal features, but Octomap still outperforms it in terms of map fidelity. In this case, the maps were generated with the same tuning parameters as in section 4.1.1: 5 cells-per-meter for Octomap and a combination of 6-20-10 for 3D-ALTE's maximum resolution, division criterion, and occupancy criterion, respectively. The point cloud is presented in Figure 4.10, for reference.

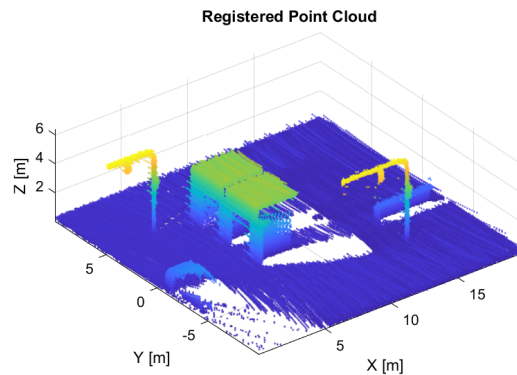


Figure 4.10 Simulated Urban Navigation: Terrain 2 point cloud after registration process.

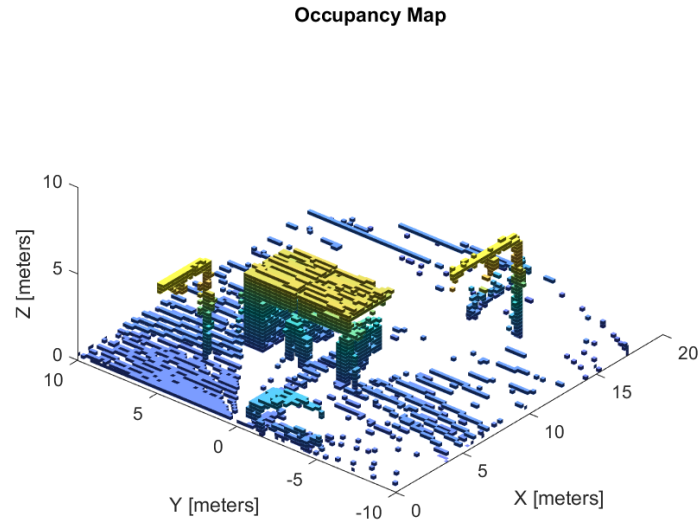


Figure 4.11 Octree applied to simulated Urban Navigation: Terrain 2.

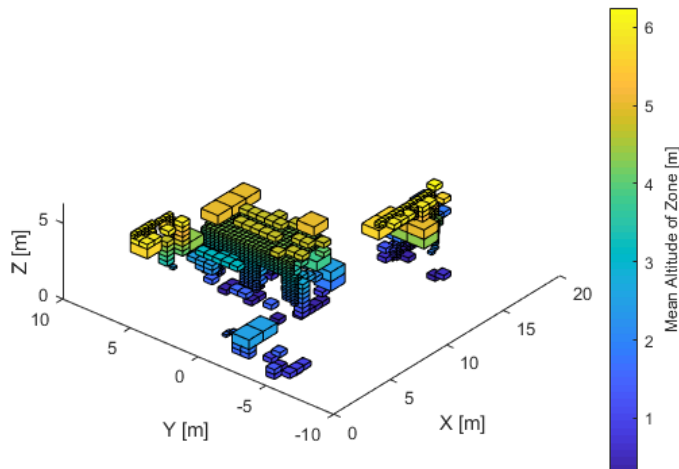


Figure 4.12 3D-ALTE applied to simulated Urban Navigation: Terrain 2.

4.1.3 Autonomous Landing: Terrain 1

This landing scenario has simple geometries. The idea is to test the algorithm's performance in a simple environment with an easy-to-identify landing zone and then model a much more complicated terrain. Figure 4.13 shows a snapshot of the environment. The mission profile is the same as in Sections 4.1.1 and 4.1.2. See Figures 4.14 and 4.15 for reference.

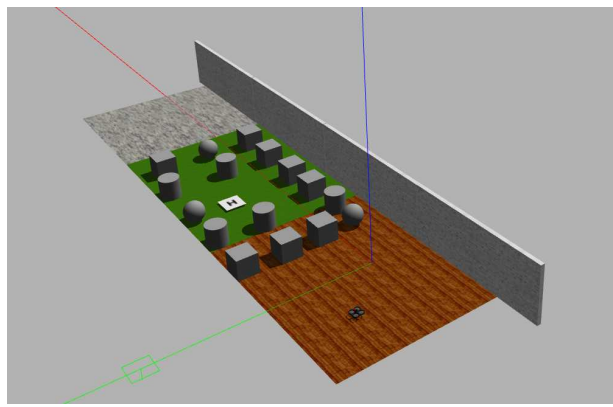


Figure 4.13 Terrain snapshot of simulated Autonomous Landing, Terrain 1: Simple Environment.

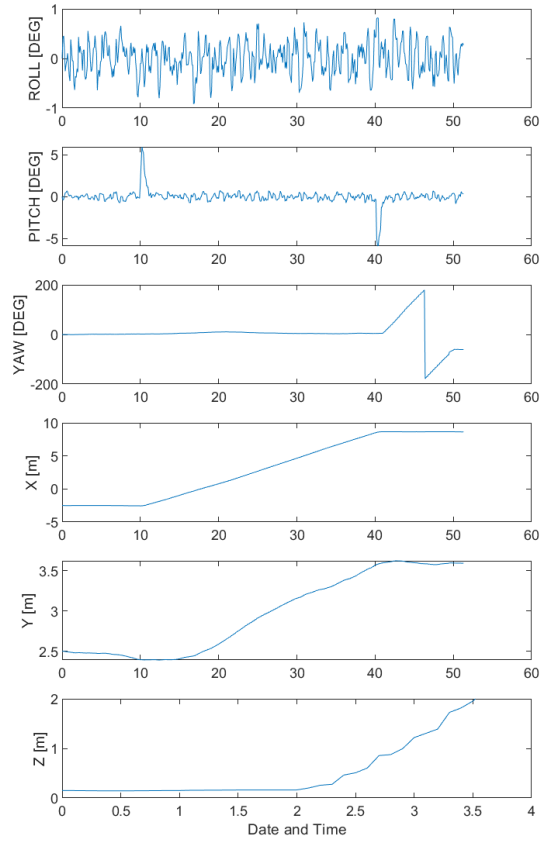


Figure 4.14 Recorded sensor position and attitude during simulated Autonomous Landing: Terrain 1.

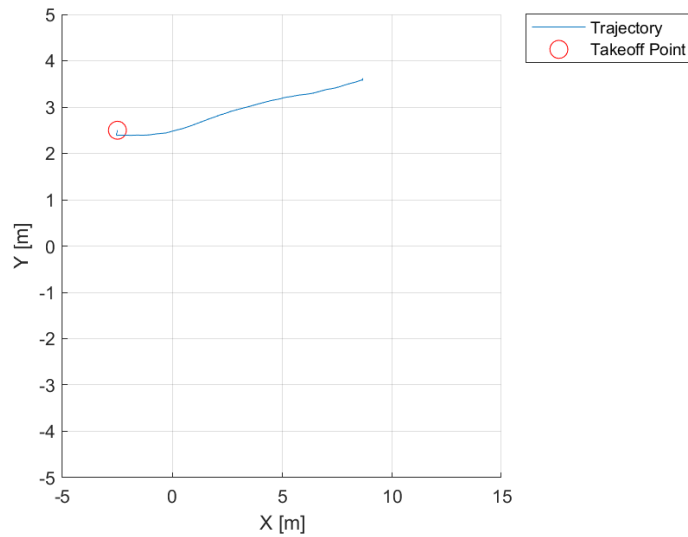


Figure 4.15 UAV trajectory for data collection of simulated Autonomous Landing: Terrain 1.

The resulting map from Quadtree is shown in Figure 4.17, where the left plot has the terrain segmentation results, and the right figure shows the resulting occupancy grid. This map leaves some free space inside the obstacles due to the way the occupancy grid is generated. Lower resolutions may achieve better results; the effects of changing parameters are analyzed in Chapter 5. The point threshold was set to 200 for this map. Figure 4.18 presents 2D Adaptive Learning results. This map is superior to its Quadtree counterpart in domain division's precision. Figure 4.16 depicts the point cloud for this scenario. The division criteria was set to 10 points and a refinement parameter kappa of 0.1, and a maximum resolution of 6.

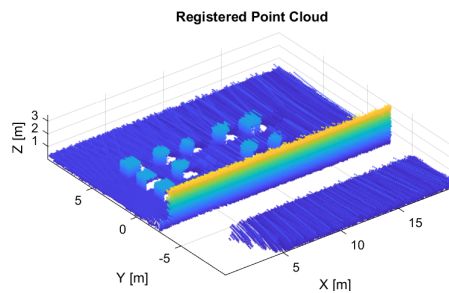


Figure 4.16 Simulated Autonomous Landing: Terrain 1 point cloud after registration process.

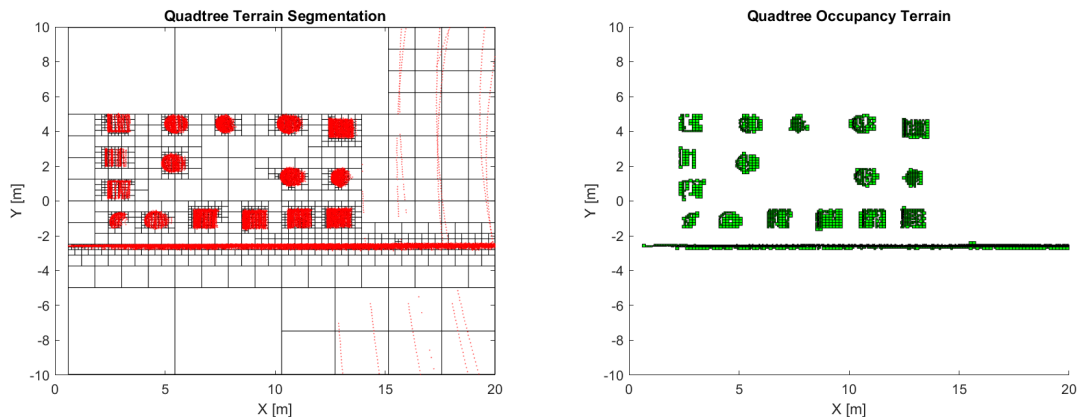


Figure 4.17 Quadtree applied to simulated Autonomous Landing: Terrain 1. Terrain segments (left) and occupancy grid (right).

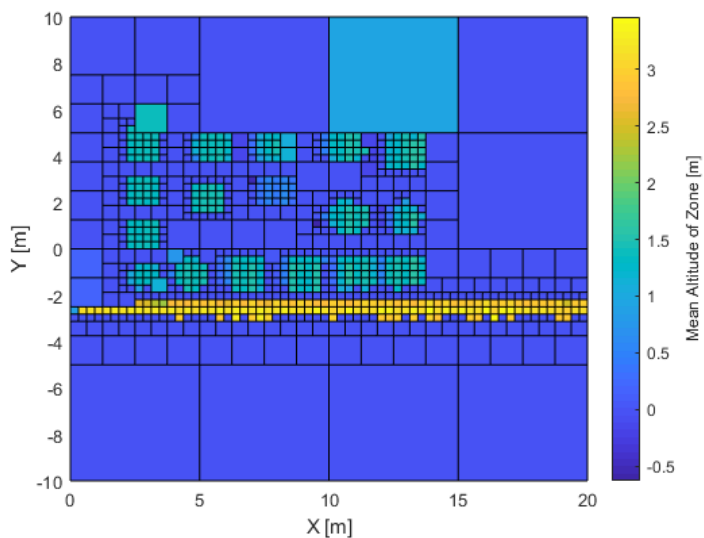


Figure 4.18 2D ALTE applied to simulated Autonomous Landing: Terrain 1.

4.1.4 Autonomous Landing: Terrain 2

Figure 4.19 shows a common package delivery scenario where a suitable landing spot is not easy to identify. The flight trajectory and maneuvers, shown in Figures 4.20 and 4.21, are the same as in previous sections.

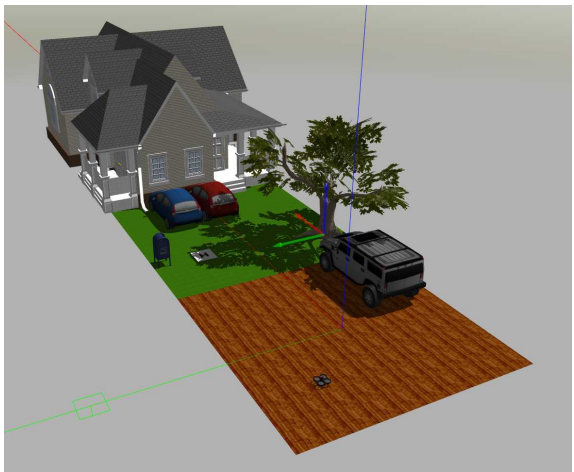


Figure 4.19 Terrain snapshot of simulated Autonomous Landing, Terrain 2: Complex Scenario.

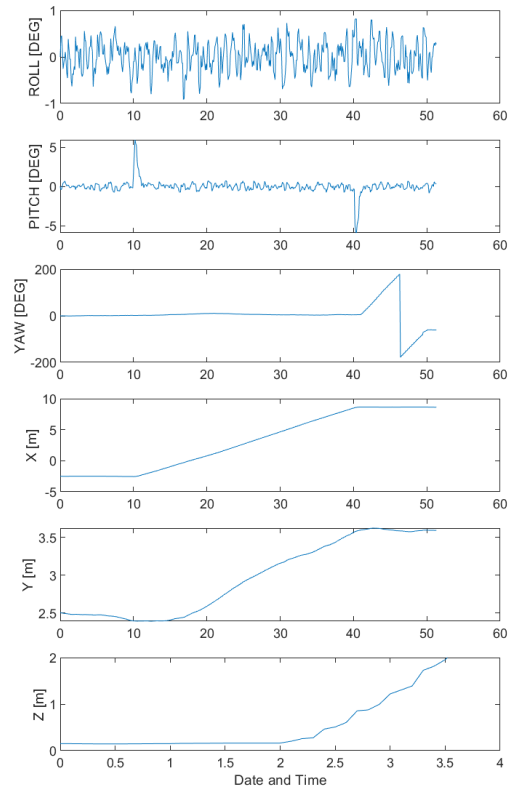


Figure 4.20 Recorded sensor position and attitude for simulated Autonomous Landing: Terrain 2.

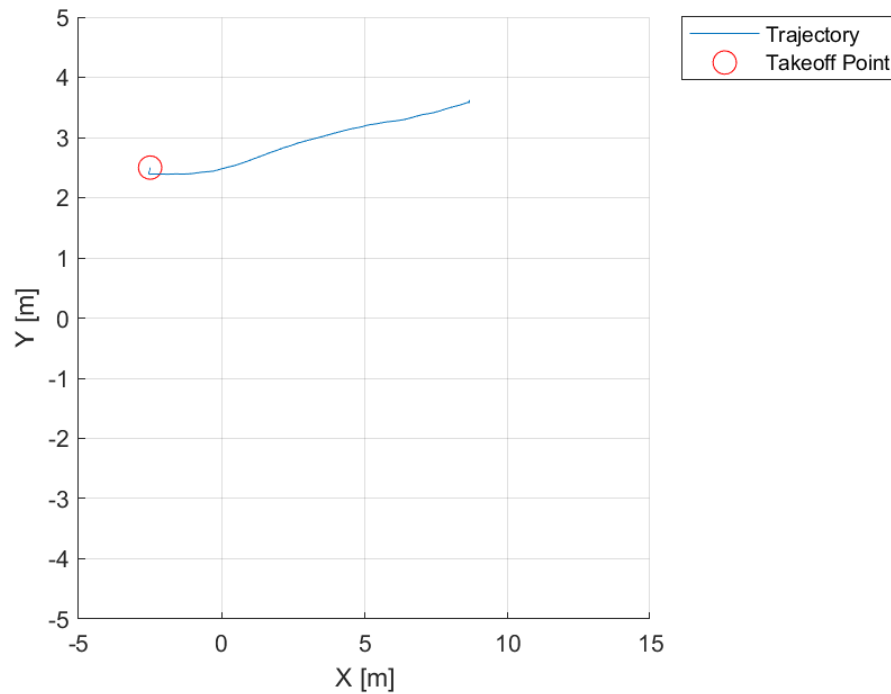


Figure 4.21 UAV trajectory for data collection of simulated Autonomous Landing: Terrain 2.

The point cloud in Figure 4.22 has no information in certain regions on the house's rooftop. These locations were higher than the UAV's altitude, so they were not scanned by the LiDAR. This provides an opportunity to analyze the algorithms' reaction to missing information. The results from Quadtree are shown in Figure 4.23. The map is of low quality due to the combination of high resolution, 25 points per cell. It shows that Quadtree is very sensitive to tuning parameters. Chapter 5 investigates the effects of changing the user-defined thresholds. Figure 4.24 shows the 2D Adaptive Learning results with the same tuning parameters as in section 4.1.3. This map is superior to the Quadtree map since it captures most of the known information. Unknown information looks the same as ground-level regions in both cases.

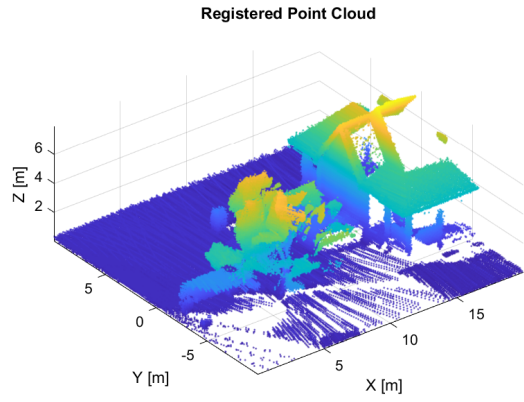


Figure 4.22 Simulated Autonomous Landing: terrain 2 point cloud after registration process.

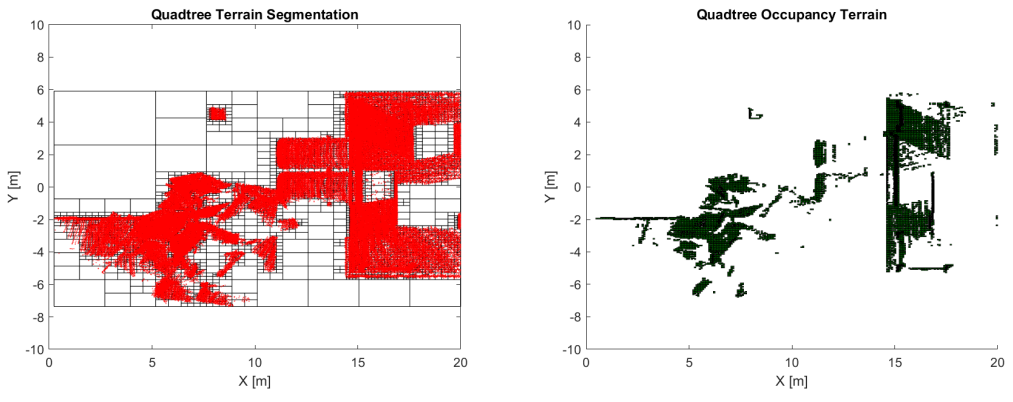


Figure 4.23 Quadtree applied to simulated Autonomous Landing: Terrain 2. Terrain segments (left) and occupancy grid (right).

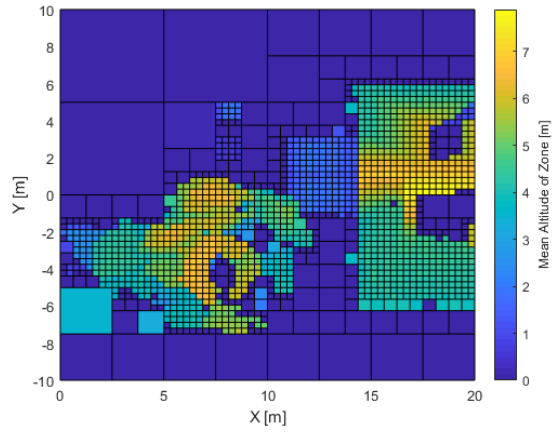


Figure 4.24 2D ALTE applied to simulated Autonomous Landing: Terrain 2.

4.2 Experimental Terrains

4.2.1 Urban Navigation: Terrain 1

The environment in Figure 4.25 mimics an urban canyon. The setup was designed for a simple back and forth trajectory with only one turn, as seen in Figure 4.27. Figure 4.26 shows the attitude and position of the sensor. As addressed in Section 2.2.4, roll, pitch and yaw are reported from an external observer point-of-view.



Figure 4.25 Terrain snapshot of experimental Urban Navigation, Terrain 1. Urban Canyon.

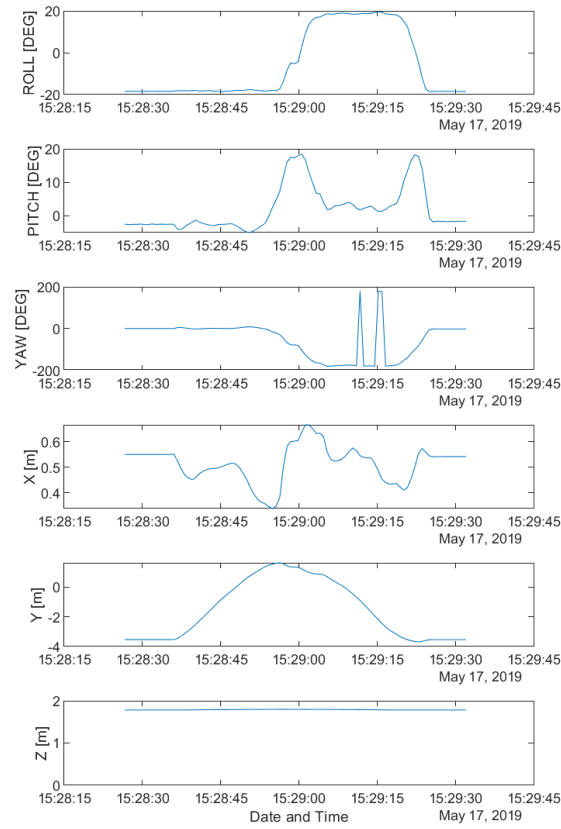


Figure 4.26 Data from VICON Vantage system for experimental Urban Navigation: Terrain 1.

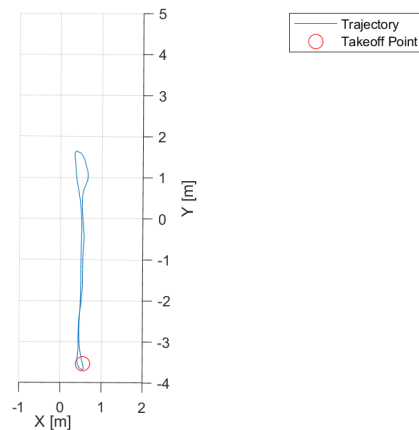


Figure 4.27 Vehicle trajectory for data collection for experimental Urban Navigation: Terrain 1.

For all experimental procedures, point cloud registration was performed online to test the setup's real-time capabilities. However, sensor data and camera observations were recorded as frames with a timestamp as well. Recording data as frames allows doing repeatable tests on the same data sets without the need to repeat the experiment. The final point cloud is presented in Figure 4.28. The global point cloud is shown in Figure 4.28. The algorithm was implemented in a loop integrating all the measurements and mapping them one step at a time to mimic a real-time mapping procedure. Figure 4.29 shows the resulting octomap at 10 cells-per-meter. The pillars are mapped, although the angle shift corrupted some of the data. A similar result can be seen in the ALTE map. See Figure 4.30 for the ALTE map with a maximum resolution of 5, division threshold of 10 points and occupancy threshold of 5 points.

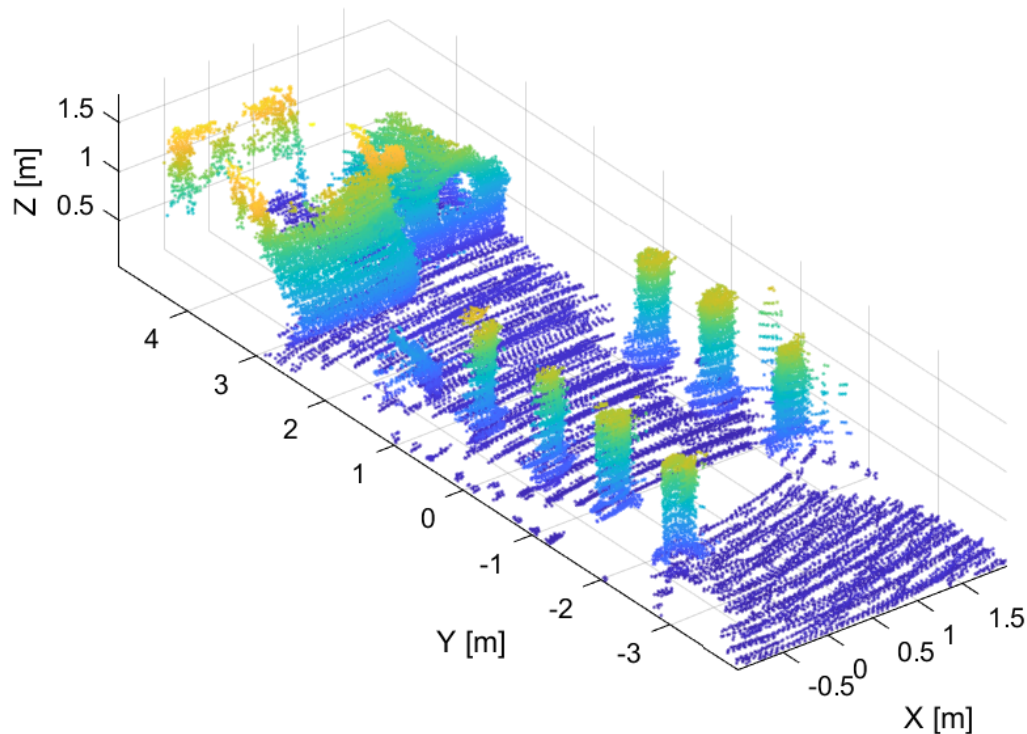


Figure 4.28 Experimental Urban Navigation: Terrain 1 point cloud after registration process.

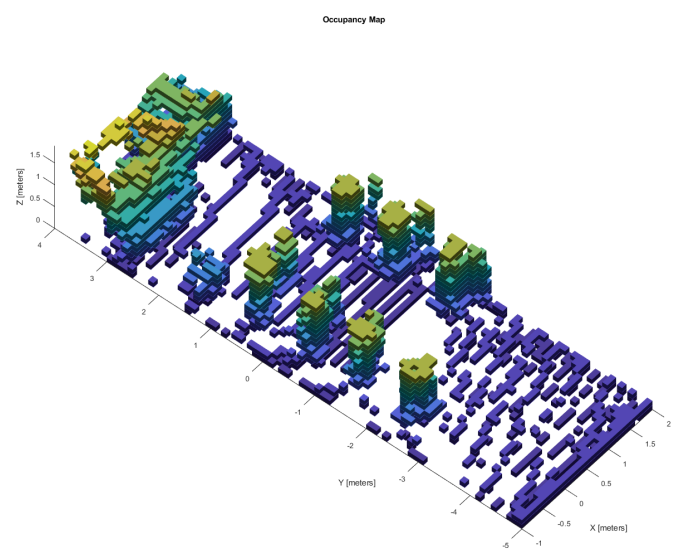


Figure 4.29 Octree applied to experimental Urban Navigation: Terrain 1.

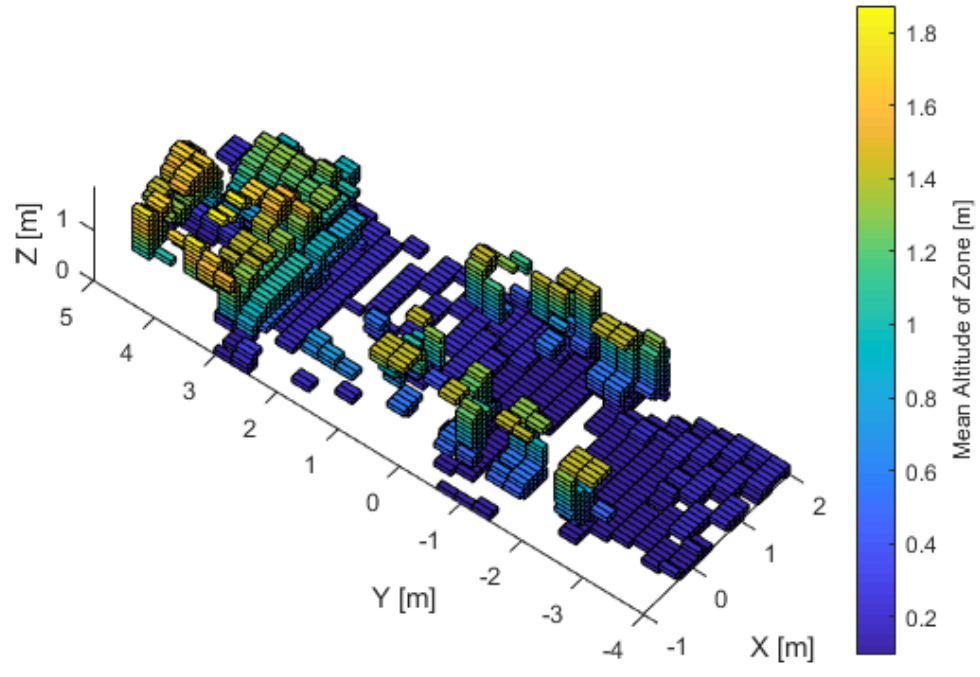


Figure 4.30 3D ALTE applied to experimental Urban Navigation: Terrain 1.

4.2.2 Urban Navigation: Terrain 2

The terrain shown in Figure 4.31 is used to test the effects of maneuvering and attitude change in the registration process camera data. The sensor's attitude, presented in Figure 4.32, shows several discontinuities, which are the effect of having an external observer measuring the attitude. A much more complex trajectory, shown in Figure 4.33, was used for this test.



Figure 4.31 Terrain snapshot of experimental Urban Navigation, Terrain 2: Complex Environment.

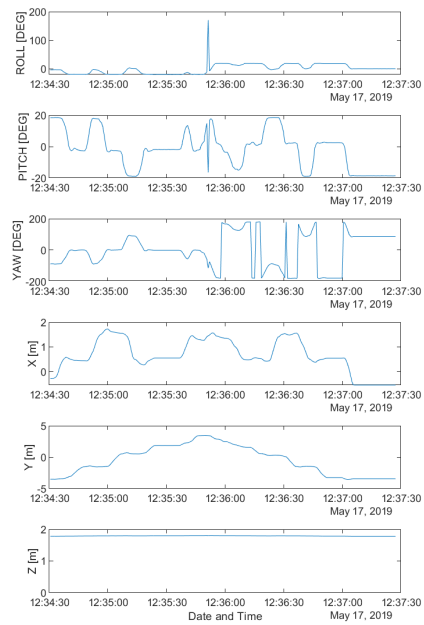


Figure 4.32 Data from VICON Vantage system for experimental Urban Navigation: Terrain 2.

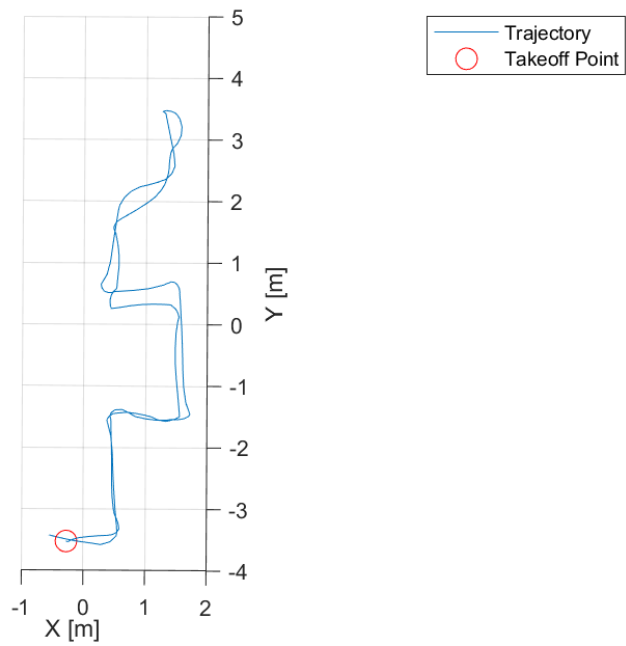


Figure 4.33 Vehicle trajectory for data collection for experimental Urban Navigation: Terrain 2.

The point cloud in Figure 4.34 shows clear signs of desynchronization due to maneuvering. Fortunately, the mapping algorithms reject these outliers to a certain degree. The corresponding Octomap and ALTE maps are shown in Figures 4.35 and 4.36, respectively. The tuning parameters are the same as in section 4.2.1.

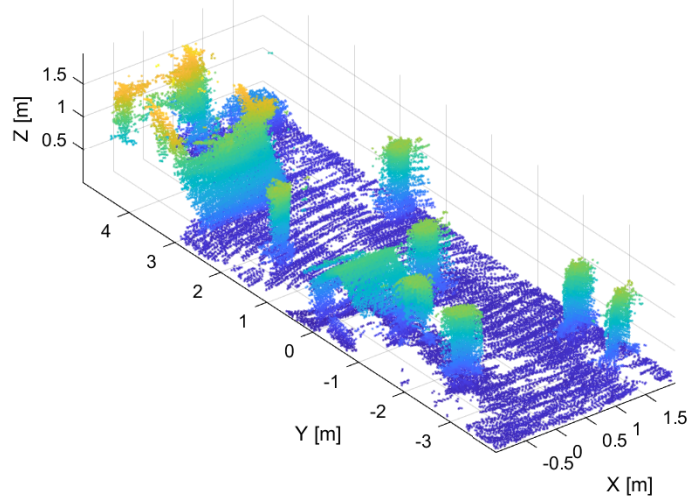


Figure 4.34 Experimental Urban Navigation: Terrain 2 point cloud after registration process.

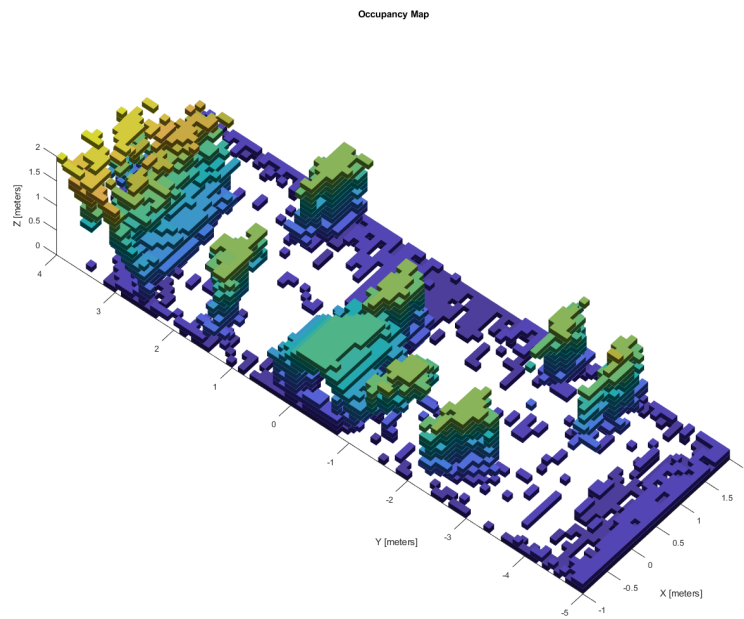


Figure 4.35 Octree applied to experimental Urban Navigation: Terrain 2.

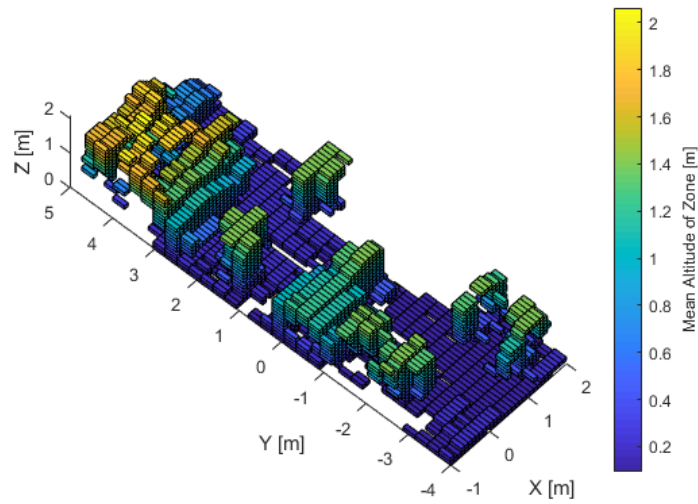


Figure 4.36 3D-ALTE applied to experimental Urban Navigation: Terrain 2.

4.2.3 Autonomous Landing: Terrain 1

The terrain shown in Figure 4.37 is a simple landing scenario that simulates debris in a flat zone. The position/attitude and trajectory of the vehicle are shown in Figures 4.38 and 4.39, respectively.



Figure 4.37 Terrain snapshot of experimental Autonomous Landing, Terrain 1: Simple Environment.

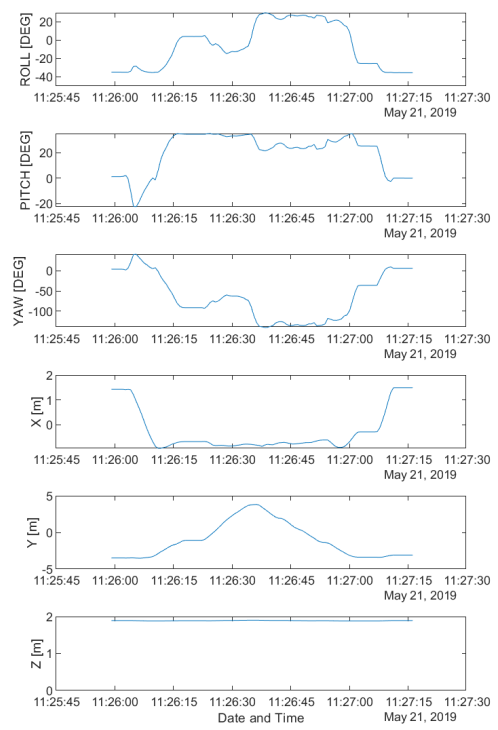


Figure 4.38 Recorded sensor position and attitude for experimental Autonomous Landing: Terrain 1.

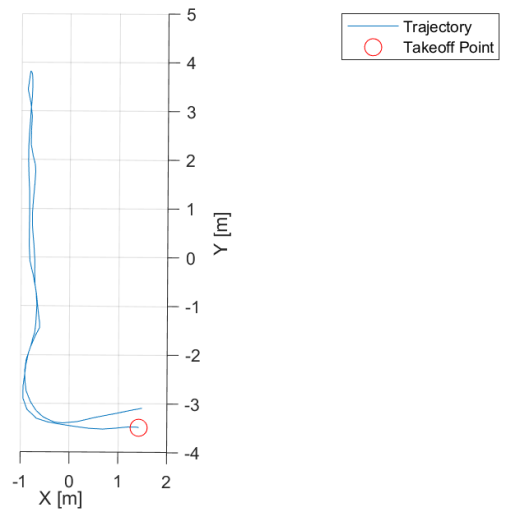


Figure 4.39 Vehicle trajectory for data collection for experimental Autonomous Landing: Terrain 1.

The point cloud in Figure 4.40 is obscured by the high contrast of altitudes between the debris and the car. The maps in Figures 4.41 and 4.42 do not capture certain obstacles because they are too close to the ground. In this case, the altitude contrast causes the quadtree map to be better than the ALTE map in terms of representing these features. The parameters are: 200 points-per-cell for Quadtree, and a variance of 0.1, maximum resolution of 6 and 10 points-per-cell for 2D-ALTE.

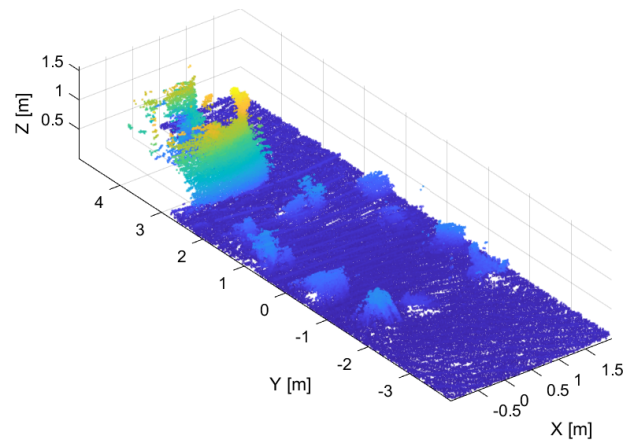


Figure 4.40 Experimental Autonomous Landing: Terrain 1 point cloud after registration process.

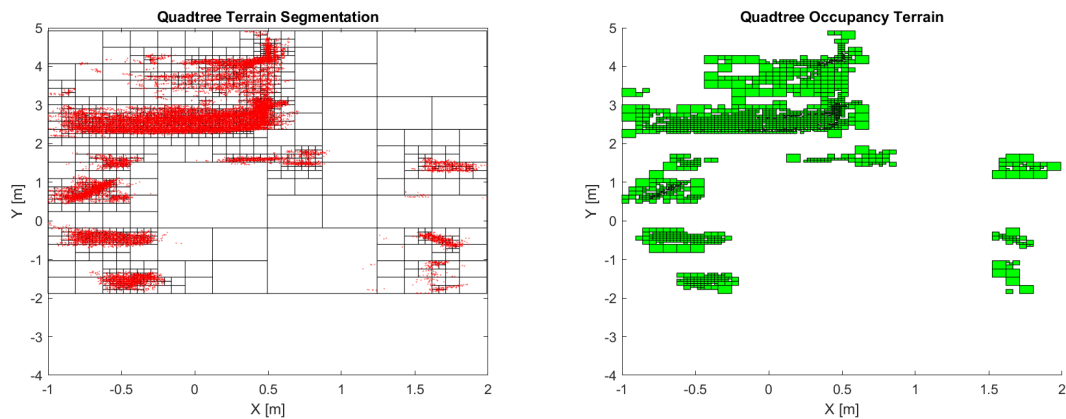


Figure 4.41 Quadtree applied to experimental Autonomous Landing: Terrain 1. Terrain segments (left) and occupancy grid (right).

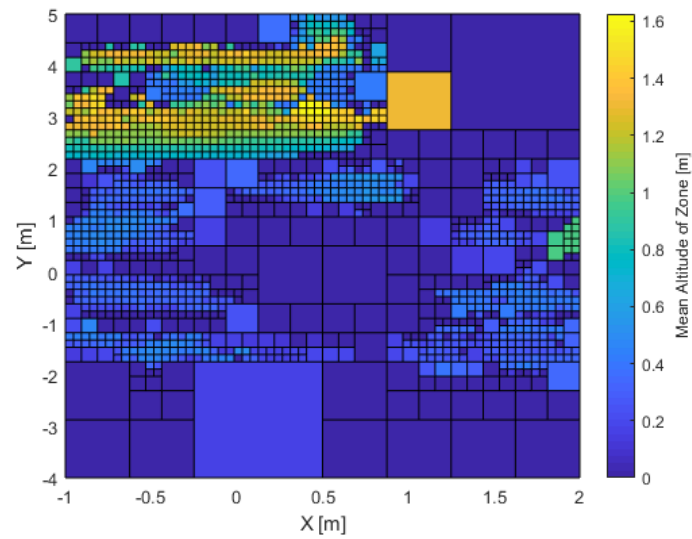


Figure 4.42 2D-ALTE applied to experimental Autonomous Landing: Terrain 1.

4.2.4 Autonomous Landing: Terrain 2

The last terrain, shown in Figure 4.43, represents a complex landing scenario.

Figures 4.44 and 4.45 show the position/attitude and trajectory of the sensor, respectively.



Figure 4.43 Terrain snapshot of experimental Autonomous Landing, Terrain 2: Complex Environment.

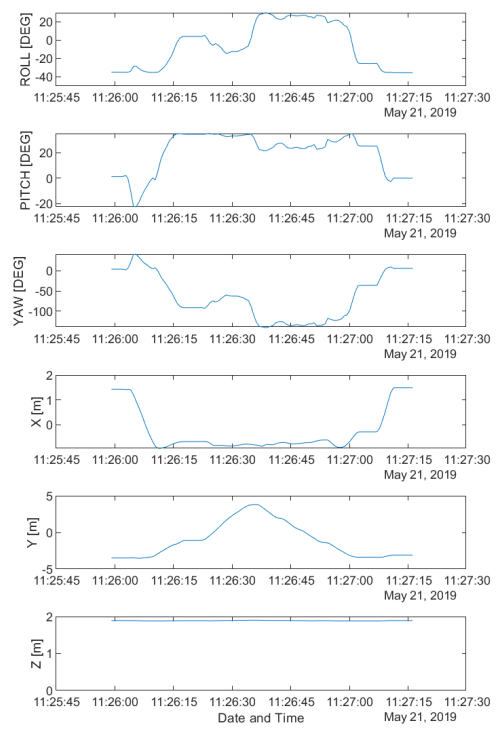


Figure 4.44 Vehicle position and attitude for experimental Autonomous Landing: Terrain 2.

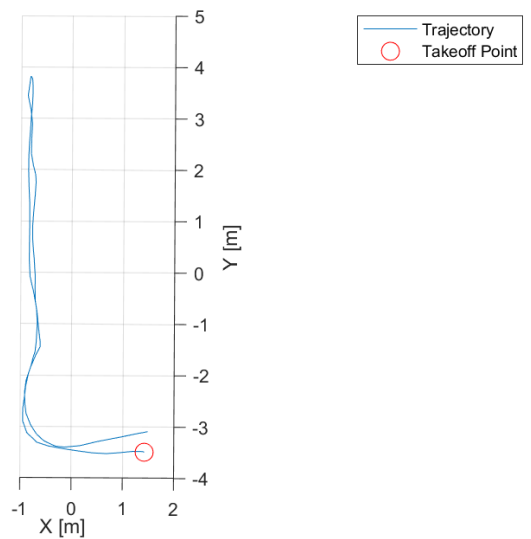


Figure 4.45 Data collection trajectory for experimental Autonomous Landing: Terrain 2.

Figure 4.46 shows the point cloud after the registration process. This time, the car was removed to prevent the height contrast from obscuring the actual obstacles. Figure 4.47 and 4.48 show the quadtree map and ALTE map, respectively. Both maps identify the obstacles and discriminate them from the ground. However, objects that are too close to the ground end up being filtered out by the plane fitting algorithm. The tuning parameters are the same as in section 4.2.3.

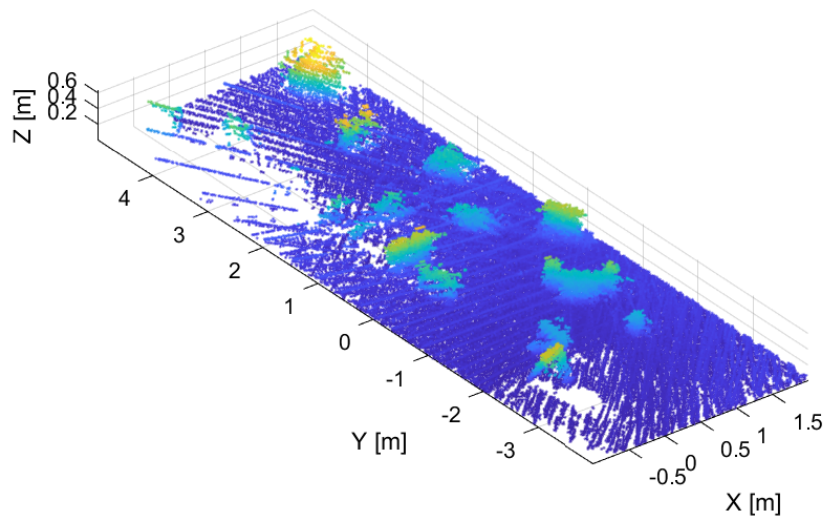


Figure 4.46 Point cloud after registration process for experimental Autonomous Landing: Terrain 2.

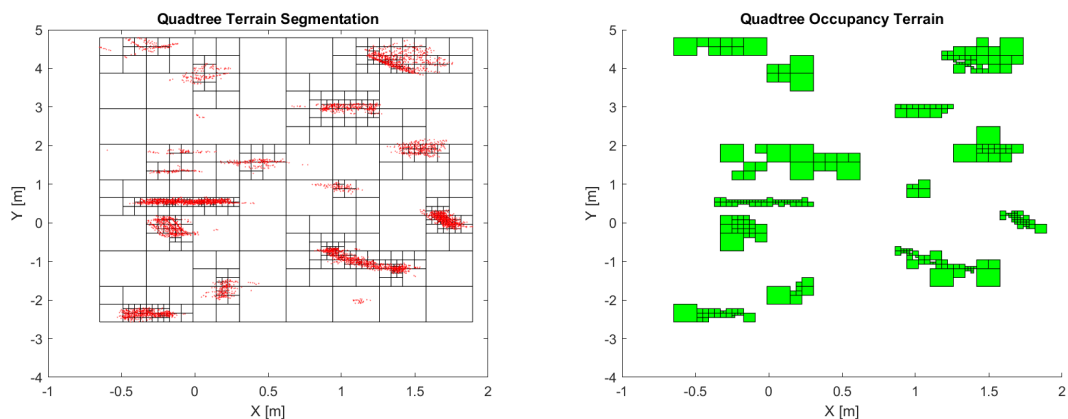


Figure 4.47 Quadtree applied to experimental Autonomous Landing: Terrain 2.

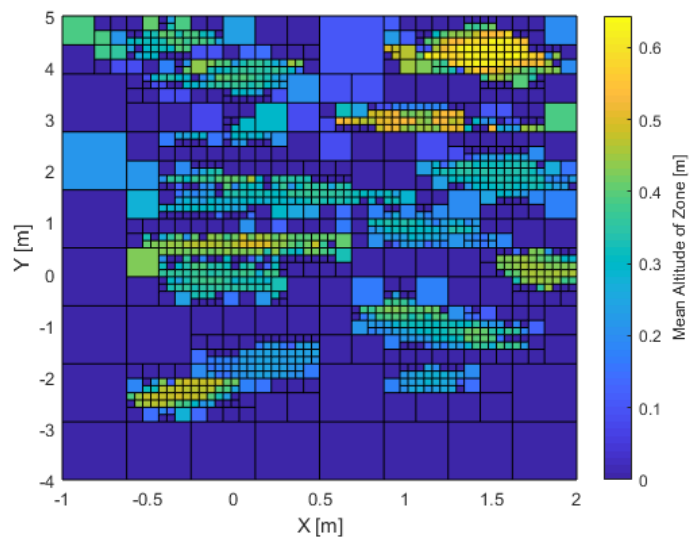


Figure 4.48 2D-ALTE applied to experimental Autonomous Landing: Terrain 2.

5. Analysis

This chapter defines the metrics, based on the specific mission profiles (cases), for comparing the different mapping approaches. This analysis was performed using previously-registered point clouds. The terrains were first scanned, registered, and then analyzed. Chapter 5 is divided into two main sections: Section 5.1 compiles several non-quantifiable observations compiled while testing and developing the algorithms, and Section 5.2 presents a review of the quantitative results of applying the metrics to maps, using different user-defined parameter combinations.

5.1 Qualitative Analysis

The qualitative analysis is based on general non-quantifiable criteria, applied to all the algorithms; as well as, on mission-specific criteria, applied to each case. The general criteria were defined by considering desirable mapping features for real-time implementation: adaptability, robustness, level of development, recursiveness, and outlier rejection.

Adaptability refers to the algorithm's ability to adjust the cells' sizes locally based on the point density. Robustness is the sensitivity to the tuning parameters and how they affect the results. Recursiveness is the ability to add new data to update maps locally. Outlier rejection is the capacity to eliminate points that are not part of the terrain. Finally, the level of development is an estimate of the algorithm's maturity for implementation in real missions.

The mission-specific criteria were based on the cases' individual needs. For instance, autonomous landing needs clear obstacle identification, outlier rejection, and accuracy for planning precise trajectories. On the other hand, urban navigation requires lightweight maps that can be processed quickly. Also, it requires information on overhanging structures and discrimination between unknown and free spaces. An appropriate balance between map size and fidelity is critical.

Three criteria were defined for each case: Obstacle identification, outlier rejection,

and map quality were selected for autonomous landing, whereas, map size, free space mapping, and overhanging structure mapping were chosen for urban navigation. Tables 5.1 and 5.2 compare the algorithms using the general non-quantifiable criteria. Each feature has a value of high, mid, or low, except for recursiveness, which can be either a yes or a no. Tables 5.3 and 5.4 present a comparison between the two algorithms using case-oriented criteria. Again, the ranges are low to high and yes or no.

Table 5.1

Summary of general qualitative findings on 2D algorithms.

2D Mapping					
Algorithms	Adaptability	Robustness	Development	Recursiveness	Outlier Rejection
Quadtree	Mid	Low	Low	No	High
2D ALTE	High	Mid	Mid	Yes	Mid

Table 5.2

Summary of general qualitative findings on 3D algorithms.

3D Mapping					
Algorithms	Adaptability	Robustness	Development	Recursiveness	Outlier Rejection
Octomap	Low	High	High	Yes	High
3D ALTE	High	Low	Low	Yes	Mid

Table 5.3

Summary of case-oriented qualitative findings on 2D algorithms.

Autonomous landing			
Algorithms	Obstacle Identification	Outlier rejection	Map Quality
Quadtree	Mid	High	Mid
2D ALTE	High	Mid	High

Table 5.4

Summary of case-oriented qualitative findings on 3D algorithms.

Urban Navigation				
Algorithms	Map Size	Free Space Mapping	Overhanging Structures	
Octomap	High	Yes	Yes	
3D ALTE	Mid	No	Yes	

5.1.1 Remarks

Quadtree

Quadtree is the basis for many tree-like structures, including Octomap and ALTE. The simplicity of this algorithm makes it versatile but with significant drawbacks, like the lack of recursiveness. Its inability to integrate scans over time into a single map makes real-time implementation highly complex. On the other hand, its adaptability and outlier rejection capabilities yield a good overall mapping performance.

For autonomous landing, Quadtree identifies obstacles with reasonable accuracy. Outlier rejection causes data to get filtered out, which could negatively impact the mapping process, especially in complex environments. In general, Quadtree is a fair candidate for autonomous landing provided a robust implementation that allows recursiveness.

Octomap

Octomap's most critical characteristic for real-time applications is its implementation architecture. It has several features that increase the algorithm's robustness and outlier rejection, which make it a superb candidate for real-time implementation. However, its high computational demands restrict its usage, especially on UAVs, where the processing power is relatively limited. Therefore, computational load is Octomap's most severe downside, and the main reason it is not implemented in aerospace missions. The lack of adaptability in the mapping process also plays a considerable role as it highly contributes to the map's size.

Regarding urban navigation, Octomap is an ideal candidate provided sufficient onboard processing power. It creates high fidelity maps that capture overhanging structures, free spaces, and most geometries with a high level of detail. This algorithm outperforms every other mapping technique in all areas except for the already mentioned computational burden and adaptability (multiresolution). Unfortunately, the computational issue makes it almost nonviable for urban navigation for most UAV platforms.

2D-ALTE

The 2D-ALTE algorithm has all the desirable features for real-time implementation. It uses variable-sized grids that adapt to point densities, it adds new information into previously existing maps, and it has fair outlier rejection mechanisms. Thus, considering its level of development, 2D-ALTE has a superior performance. However, it is still sensitive to tuning parameters, an issue that requires further investigation.

Concerning autonomous landing, 2D-ALTE excels in the most critical matters: obstacle identification and map quality. This technique estimates all the obstacles consistently with an acceptable level of fidelity, even with low resolutions. On the other hand, higher resolutions do not impose high enough computational loads. And, although outlier rejection can eliminate some useful data, it does not affect the maps considerably.

3D-ALTE

3D-ALTE's best feature is its multiresolution space division capabilities. The ability to analyze space using variable-sized grids gives this algorithm a significant advantage over Octomap. However, adaptability comes with downsides: The algorithm is sensitive to tuning parameters, which is a problem in real-time implementations. Even though 3D-ALTE's level of development is low, the resulting maps have a reasonable degree of fidelity.

In the context of urban navigation, 3D-ALTE generates compact terrain

representations that do not impose high computational loads. In this case, the map's size is highly related to free-space mapping. It appears that not discriminating vacant from unknown space has a drastic impact on the algorithm's processing requirements.

Free-space mapping, important as it is, is outweighed by the essential need for memory efficiency. Despite the low computational load, the overall terrain, including overhanging structures, is represented with a fair level of accuracy.

5.2 Quantitative Analysis

5.2.1 Metrics

The metrics are closely related to the needs of the specific cases. For instance, processing time is the primary concern for real-time urban navigation. In contrast, in autonomous landing, map accuracy is the main issue. In both cases, on-board payload limitations ultimately constraints map sizes. Three metrics were defined to categorize the algorithm's performance using these considerations.

The first metric is the code execution time. One of the main problems with real-time applications in UAVs is the limitations of computational power. This complication, added to fast-paced missions, makes code run-time a decisive factor. The code execution time takes into account only the mapping process, ignoring any prior data down-sampling and filtering. The second metric categorizes the computational burden that a map imposes. Terrain mapping represents reality using mathematical expressions called basis functions. In this context, a basis function is defined as a cell in space that has specific physical characteristics, e.g., occupancy and location. Since path planning algorithms interact with these functions, fewer functions mean less required processing power. For this reason, the second metric is the number of basis functions of each map. The third metric is a measurement of map fidelity. The Hausdorff distance (HD) quantifies the degree of mismatch between the points in two sets (Huttenlocher, Klanderman, & Rucklidge, 1993). Given two finite point sets $A = \{x_1, \dots, x_a\}$ and $B = \{y_1, \dots, y_b\}$ the Hausdorff distance is defined as

$$HD(A,B) = \max(h(A,B), h(B,A)) \quad (5.1)$$

with

$$h(A,B) = \max_{x \in A} \min_{y \in B} \|x - y\| \quad (5.2)$$

where $\|\cdot\|$ is the Euclidean norm.

$h(A,B)$ is the directed Hausdorff distance (DHD) from A to B . First, it calculates the distances from all points $x \in A$ to all points $y \in B$. From this set, it creates a subset of minimum distances by identifying the closest point in B to each point in A , and selects the maximum element of this subset. Finally, the process is repeated the other way around to calculate $h(B,A)$. The HD is the maximum element of the set $(h(A,B), h(B,A))$.

In their research, (Dubuisson & Jain, 1994) define another approach called Modified Hausdorff Distance (MHD). This metric addresses issues that the HD metric faces when analyzing noisy sets. MHD defines the directed Hausdorff distance as

$$h(A,B) = \frac{1}{N_a} \sum_{x \in A} \min_{y \in B} \|x - y\| \quad (5.3)$$

where N_a , is the number of points in set A .

The DHD rejects noise by taking the mean instead of the maximum. This approach is not a metric, mathematically speaking, since it does not satisfy the triangle inequality. However, it does satisfy non-negativity, identity, and symmetry.

Implementation

Only the HD and MHD metrics require an implementation code because analyzing run-time is trivial, and the mapping algorithms already quantify basis functions. The implementation of HD and MHD follows the works of (Danziger, 2010) and (Sasikanth, 2011), respectively. The HD is implemented using equation 5.1. Set A is the registered point cloud, which represents the true terrain, and set B is created from the map. Set B is

composed of subsets containing the coordinates of the corners of each basis function. Figure 5.1 depicts the process of building sets A and B for calculating HD and MHD.

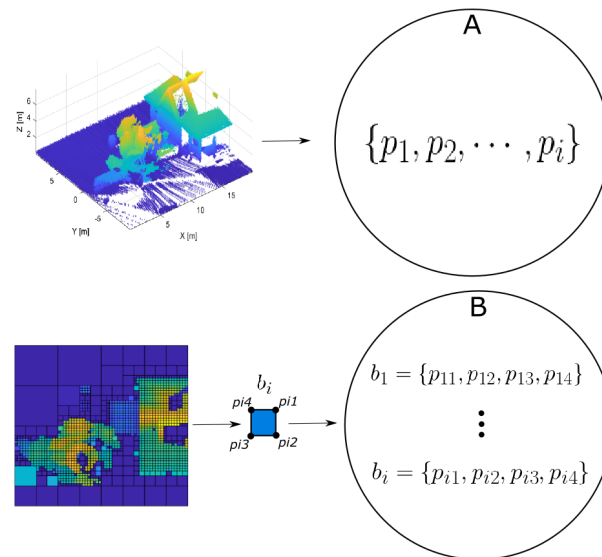


Figure 5.1 Graphic representation of HD and MHD implementation.

5.2.2 Results

The quantitative analysis was applied to two specific terrains, one for each case. The terrains for the 3D and 2D analyses, defined as SIM NAV 1 and SIM LAN 2, respectively, were selected for their complexity. See Sections 4.1.1 and 4.1.4 for more information on the terrains. For reference, Tables 5.5 and 5.6 show the number of frames used for the registration process and the total number of points in the final point clouds. The metrics are applied separately to each mapping technique using several tuning parameters. First, a comprehensive interpretation of the results is provided, and then a comparison between corresponding mapping algorithms for the two missions is provided.

Table 5.5

Information on the terrain s point cloud for 3D analysis. See Figure 4.1.

MAP	Number of Frames	Total Number of Points in Point Cloud
SIM NAV 1	569	381187

Table 5.6

Information on the terrain's point cloud for 2D analysis. See Figure 4.19.

MAP	Number of Frames	Total Number of Points in Point Cloud
SIM LAN 2	508	439032

Quadtree

The tuning parameter, Max points per cell, is a loop-breaking threshold. Quadtree runs a check on every basis function dividing it if it contains more points than the allowable value. This logic creates a loop that only stops when the number of points in all the basis functions is less or equal to Max points per cell. This iterative process produces an exponential growth in execution time and basis functions when decreasing the threshold. The accuracy of the map increases with the resolution to a certain point. At the highest resolution (1000), the terrain estimation starts losing fidelity due to the logic used for deciding occupancy.

The MHD metric has a clear relationship with the quality of the map. The MHD is inversely proportional to Max pts per cell, which means the mismatch levels increase as the resolutions become lower. The HD metric fails to deliver conclusive results due to the presence of outliers. These results show the need for the MHD metric. Figure 5.2 shows the corresponding maps for the resolutions in Table 5.7.

Table 5.7

Quadtree results after varying the resolution parameter.

Parameter	Value	Elapsed Time [s]	Basis Functions	HD [m]	MHD [m]
Max pts per cell	25	43.38	24343	5.515	1.235
	100	5.51	7048	5.310	1.029
	150	3.06	4399	5.308	1.090
	200	2.08	3565	5.310	1.206
	500	1.02	1375	5.311	1.290
	1000	0.62	757	5.106	1.168

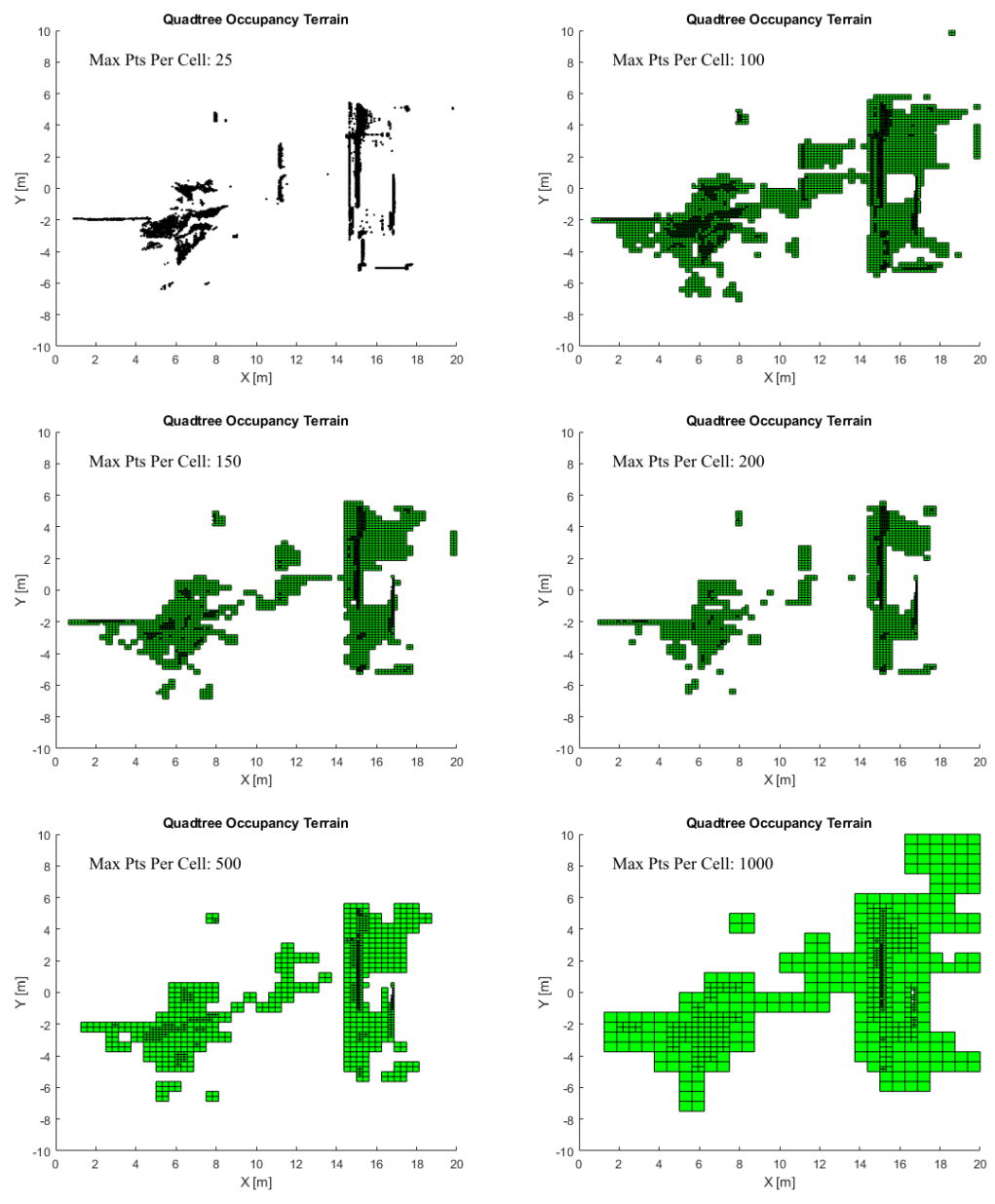


Figure 5.2 Quadtree maps at different max-points-per-cell values.

Octomap

Octomap's single tuning parameter, Resolution, is the number of cells per meter, which determines the fixed size of the cells in the 3D grid. This parameter does not set the map's memory size from the beginning, rather it defines how much computational power the algorithm requires to process data streams. Elapsed time and basis functions are inversely proportional to the size of the cell. HD and MHD also seem inversely

proportional except in the last map (Resolution: 6). It appears that the implementation method introduces an error that increases with the number of basis functions. It means that the Hausdorff approaches, implemented as proposed in this thesis, become more unreliable as the number of basis functions increases. However, most HD and MHD values are consistent with the qualitative observations of the maps. Figure 5.3 shows the maps that correspond to the results in Table 5.8.

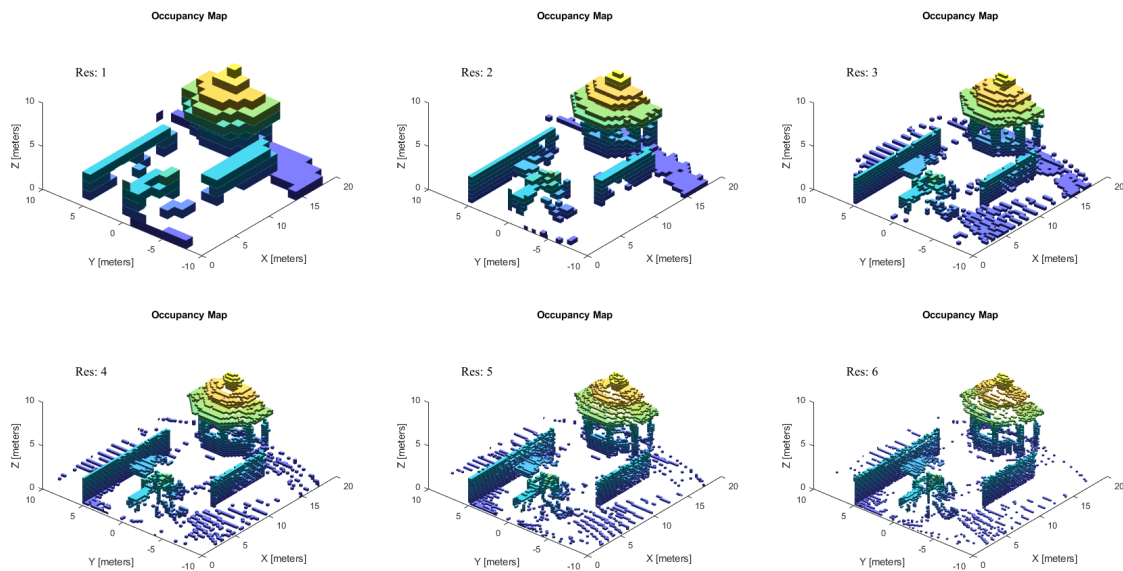


Figure 5.3 Octomap at different cell-per-meter values.

Table 5.8

Octomap results after varying the resolution parameter.

Parameter	Value	Elapsed Time [s]	Basis Functions	HD [m]	MHD [m]
Resolution	1	3.24	969	5.0070	1.2499
	2	6.96	5499	4.9430	1.0236
	3	9.92	16423	3.9773	0.4301
	4	13.82	35877	2.5772	0.3760
	5	19.34	65750	3.0499	0.3702
	6	24.61	108607	2.9875	0.4049

Table 5.9

Results of varying tuning parameters on 2D ALTE algorithm.

Parameters	Value	Elapsed Time [s]	Basis Functions	HD [m]	MHD [m]
KAPPA	0.1				
MAX RES	4	3.1455	216	5.1050	0.8496
MIN PTS	100				
KAPPA	1				
MAX RES	5	4.45	664	5.1050	0.7250
MIN PTS	10				
KAPPA	0.1				
MAX RES	6	4.9348	2136	5.1050	0.8256
MIN PTS	10				
KAPPA	0.05				
MAX RES	4	3.18	220	5.1050	0.8496
MIN PTS	100				
KAPPA	0.01				
MAX RES	5	3.51	668	5.1050	0.7250
MIN PTS	100				
KAPPA	0.01				
MAX RES	6	4.30	2176	5.1050	0.8258
MIN PTS	10				

2D-ALTE

The 2D-ALTE algorithm has three tuning parameters. Kappa, known as the refinement threshold, is used to assess the variance for dividing domains. Max Res defines the maximum number of times an area can be subdivided. Finally, Min Points represents the minimum number of points for partitioning a domain. Note that there is no occupancy threshold since this map stores the mean altitude information for each region. Kappa seems to be closely related to the algorithm's execution time but Max Res is more heavily correlated. The number of basis functions also depends on these two parameters, but it appears that Max Res is the defining factor. The HD and MHD results are consistent. The negative effects of the implementation algorithm are also present. Again, HD and MHD results are unreliable for higher resolutions. Figure 5.4 shows the maps that correspond to the parameter combinations in Table 5.9.

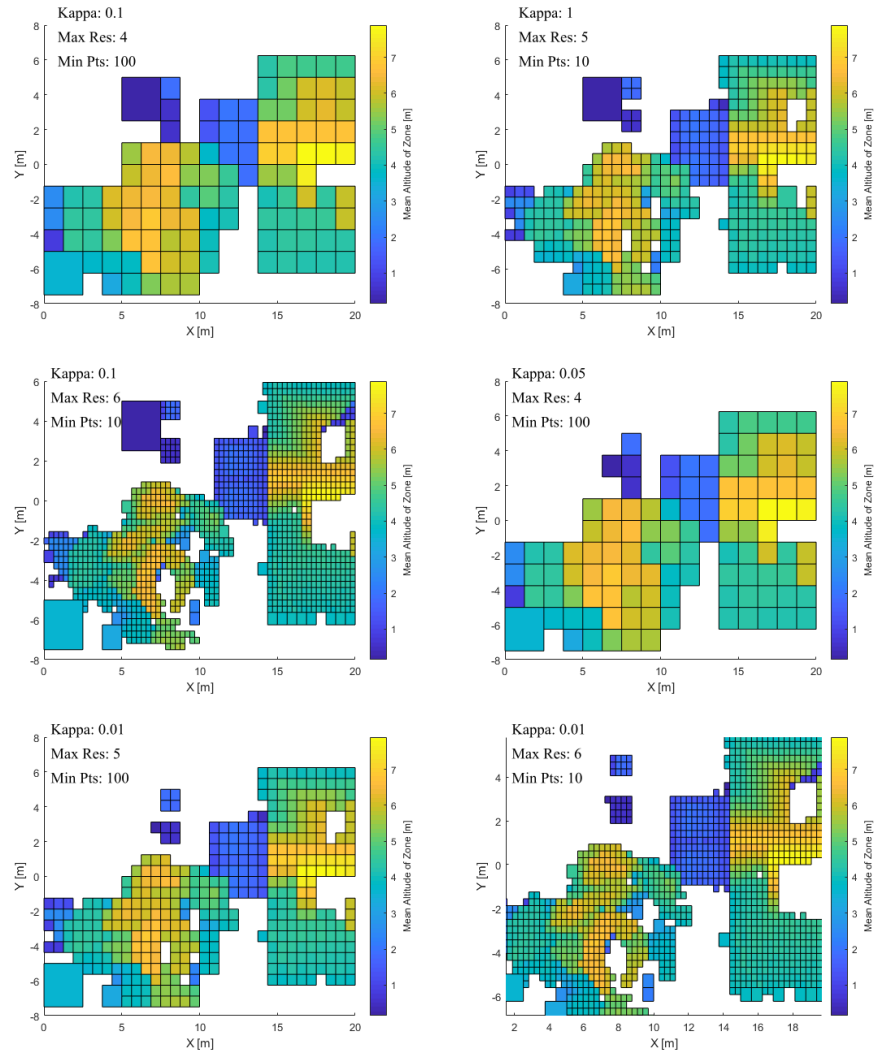


Figure 5.4 2D Adaptive Learning maps for different tuning parameter combinations.

3D-ALTE

3D-ALTE has three user-defined parameters. MIN PTS is the minimum amount of required points for a cell to be considered occupied. MIN PTS 2 is the minimum number of points a cell must have to divide into smaller cells. Finally, MAX RES is the maximum amount of times a cell can divide. MAX RES appears to be the driving factor in both execution time and number of basis functions. MIN PTS 2 also influences these metrics but to a lesser degree. MIN PTS has no considerable effect due to it being an occupancy criteria that has no effect on the division process. Resolutions of (4) produce an incomplete map due the occupancy criterion added to refine the mapping process. This

criterion checks the basis function's level in the data tree to define occupancy. The HD and MHD metrics stay consistent, giving larger values to less accurate maps, yet the values seem higher for the lowest resolution (4). This implies that the Hausdorff metrics are not sensitive enough to missing information. A new parameter is required to categorize map incompleteness, which would require the development of a novel metric implementation. The observations are quantified in Table 5.12 and the corresponding maps are shown in Figure 5.5.

Comparison

The best results for each mapping algorithm were chosen by considering all the metrics to be equally important, except for HD, due to its previously discussed inconsistency. Instead, HD can be seen as a measurement of outlier rejection when compared to MHD. The closer HD is to MHD, the better the outlier rejection is. The chosen maps show the best balance between the metrics. They have a reasonably short elapsed time, small MHD, and a not too high number of basis functions.

Table 5.10 presents the best mapping results for the 2D algorithms. Analyzing each metric separately: the elapsed time in both cases is almost the same, but ALTE has twenty times fewer basis functions and a better map in terms of MHD. The difference between HD and MHD shows that Quadtree is better for outlier rejection, which agrees with the qualitative observations. Table 5.11 compares the best 3D maps. Octomap's elapsed time is better, which means Octomap is much more efficient in analyzing data. However, the resulting number of basis functions is almost four times ALTE's number. The MHD metric shows that ALTE generates more accurate maps, but the relationship between the error in this measurement and the number of basis functions should be considered as well. ALTE is better at rejecting outliers due to the additional occupancy criterion implemented in this thesis. Overall, the ALTE algorithms have an excellent performance when compared to their counterparts. The 2D version is clearly superior in terms of efficiency and map accuracy, and the 3D version is better in critical aspects such as map size.

Table 5.10

Comparison between the best results of Quadtree and 2D ALTE.

2D Algorithms				
Mapping Technique	Elapsed Time [s]	Basis Functions	MHD [m]	MHD - HD [m]
Quadtree	3.06	4399	1.090	4.2177
2D ALTE	3.18	220	0.8496	4.2554

Table 5.11

Comparison between the best results of Octomap and 3D ALTE.

3D Algorithms				
Mapping Technique	Elapsed Time [s]	Basis Functions	MHD [m]	MHD - HD [m]
Octomap	9.92	16423	0.4301	3.5472
3D ALTE	11.59	4249	0.3688	1.5424

Table 5.12

Results of varying tuning parameters on 3D ALTE algorithm.

Parameters	Value	Elapsed Time [s]	Basis Functions	HD [m]	MHD [m]
MIN PTS	20				
MIN PTS 2	10	29.72	15273	2.3751	0.3177
MAX RES	6				
MIN PTS	20				
MIN PTS 2	10	6.89	1137	1.9756	0.5684
MAX RES	4				
MIN PTS	5				
MIN PTS 2	10	11.59	4249	1.9112	0.3688
MAX RES	5				
MIN PTS	20				
MIN PTS 2	40	66.88	12321	1.9162	0.2361
MAX RES	6				
MIN PTS	20				
MIN PTS 2	40	10.45	1065	4.3339	0.5606
MAX RES	4				
MIN PTS	40				
MIN PTS 2	10	11.5	4249	5.9167	0.3420
MAX RES	5				

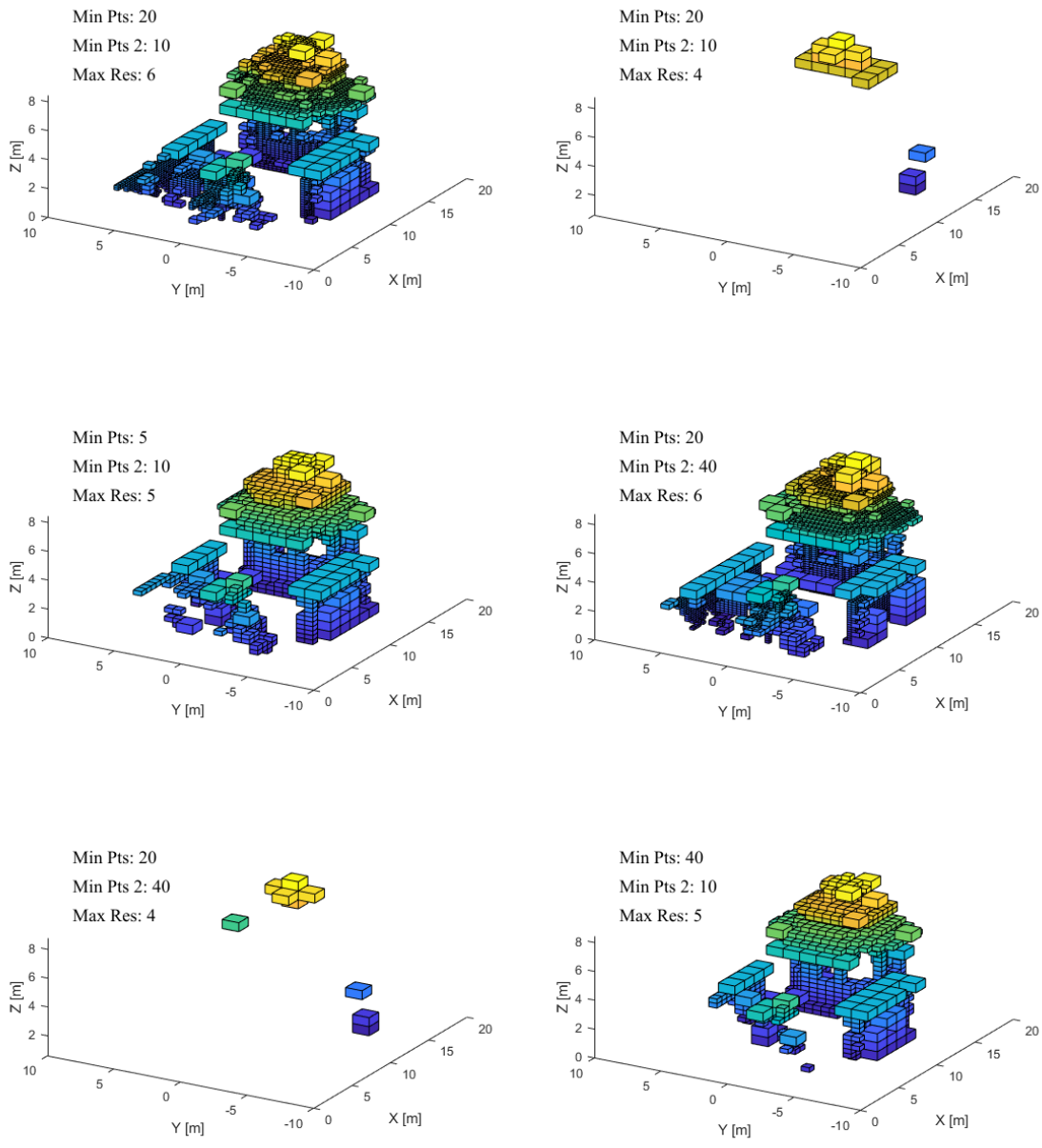


Figure 5.5 3D Adaptive Learning maps for different tuning parameter combinations.

6. Conclusions

This thesis has aimed to advance the research on ALTE for UAV applications by comparing the 2D and 3D versions against the commonly used Quadtree and Octomap algorithms. The scope of the investigation is defined by identifying two cases of interest upon which the tests focused: urban navigation for 3D algorithms and autonomous landing for 2D algorithms. The mission profiles were selected based on the needs of the aerospace industry.

These cases drove not only the tests, but also the design and building process of the real and simulated environments used for data acquisition, as well as the tools and hardware used for comparing mapping performance. These efforts resulted in a benchmark that allowed recursive testing and implementation of the algorithms under various conditions. This benchmark was used to develop implementation codes for the proposed mapping algorithms, build terrains, and generate the maps used for analysis. The analysis had two focuses: qualitative and quantitative assessment. Each pairing underwent different tests related to their specific case. The qualitative analysis ranked all the algorithms based on both real-time mapping and case-oriented desirable features.

The qualitative results show that Quadtree is a standard algorithm that lacks recursiveness. However, this simplicity makes it useful to build new mapping techniques on top of its data classification principles. On the other hand, 2D-ALTE excels in most critical areas, except in robustness, which makes it an ideal candidate for real-time implementation. Focusing on autonomous landing, Quadtree has potential applications in simple missions that do not require scan integration, whereas 2D-ALTE is a much more reliable mapping algorithm for a broader spectrum of conditions.

Octomap and 3D-ALTE have several strengths and weaknesses. For example, Octomap is far superior in terms of efficiency and robustness but fails to adapt to variations in point densities. In contrast, 3D-ALTE excels in adaptability but currently lacks robustness and overall code efficiency. For urban navigation, both algorithms deliver

reliable maps that model overhanging structures and complex geometries. Only Octomap models free space, but this causes the resulting maps to require significant memory, making them nonviable for urban navigation missions. In contrast, 3D-ALTE's multi-resolution grids create lightweight maps that are feasible for use although they lack free-space mapping.

The quantitative analysis used three metrics to compare the performance of the algorithms: code execution time, map size, and map fidelity. The code execution time took into account only the mapping process, ignoring any prior data down-sampling and filtering. The map size was expressed in terms of the number of basis functions. Map fidelity was calculated using the Hausdorff and modified Hausdorff metrics. It was found that the Hausdorff Distance is inconsistent in the presence of outliers, so a modified version was used instead, the Modified Hausdorff Distance. However, the MHD became non-reliable when analyzing maps with a high number of basis functions.

The numbers show that Quadtree produces less reliable maps that are computationally heavier and take longer to produce than 2D-ALTE's maps. Except for outlier rejection, 2D-ALTE outperforms Quadtree in all the proposed metrics. In the 3D case, Octomap is much more efficient processing data than 3D-ALTE but generates large maps that impose high computational loads, whereas 3D-ALTE is slower but produces better maps in terms of memory requirements. In any case, both mapping techniques achieve a satisfactory level of map fidelity. Still, it is not possible to determine which one is better due to the inconsistencies in MHD related to the number of basis functions.

Overall, the ALTE approaches compare favorably to powerful algorithms like Octomap and Quadtree, and should be considered for real-time applications. There are certain features that require further investigation and development before implementing the algorithms in real vehicles. For instance, 2D-ALTE can be transformed into a binary grid using the altitude as an occupancy threshold. This approach could potentially save computation time to path planners without adding too many computations to the mapping

process. In addition, many algorithms, like neighboring occupancy analysis and statistical approaches for recurring measurements, could be added as occupancy criteria to improve map estimation. 3D-ALTE is currently very sensitive to tuning parameters. This problem was solved in part by considering the level in the data tree when estimating occupancy. However, the analysis showed that this solution is still not robust enough. Based on the results, it can be concluded that both 2D-ALTE and 3D-ALTE have the potential for real-time applications in the proposed cases, and future research is recommended. The next step could be using the maps alongside a path planning algorithm and observe the performance.

REFERENCES

- About ros.* (n.d.). Retrieved from <http://www.ros.org/about-ros/> (Accessed: 02-20-2019)
- Binev, P., Cohen, A., Dahmen, W., A. DeVore, R., & Temlyakov, V. (2005, 09). Universal algorithms for learning theory part i : Piecewise constant functions. *Journal of Machine Learning Research*, 6, 1297-1321.
- Danziger, Z. (2010). *Hausdorff distance*. Retrieved from <https://www.mathworks.com/matlabcentral/fileexchange/26738-hausdorff-distance> (Accessed: 08-01-2019)
- Dubuisson, M. ., & Jain, A. K. (1994, Oct). A modified hausdorff distance for object matching. In *Proceedings of 12th international conference on pattern recognition* (Vol. 1, p. 566-568 vol.1). doi: 10.1109/ICPR.1994.576361
- Einhorn, E., & Gross, H.-M. (2015). Generic ndt mapping in dynamic environments and its application for lifelong slam. *Robotics and Autonomous Systems*, 69, 28 - 39. Retrieved from <http://www.sciencedirect.com/science/article/pii/S0921889014001663> (Selected papers from 6th European Conference on Mobile Robots) doi: <https://doi.org/10.1016/j.robot.2014.08.008>
- Everaerts, J. (2008). The use of unmanned aerial vehicles (uavs) for remote sensing..
- Gkquadtree.* (n.d.). Retrieved from <https://developer.apple.com/documentation/gameplaykit/gkquadtree> (Accessed: 02-20-2019)
- Hallenbeck, K. (2018). *Velodyne simulator*. Retrieved from <https://bitbucket.org/DataspeedInc/velodynesimulator/overview>
- Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., & Burgard, W. (2013, Apr 01). Octomap: an efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3), 189–206. Retrieved from <https://doi.org/10.1007/s10514-012-9321-0> doi: 10.1007/s10514-012-9321-0
- Huttenlocher, D. P., Klanderman, G. A., & Rucklidge, W. J. (1993, Sep.). Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9), 850-863. doi: 10.1109/34.232073
- An introduction to robot operating system (ros).* (n.d.). Retrieved from <https://www.allaboutcircuits.com/technical-articles/an-introduction-to-robot-operating-system-ros/> (Accessed: 02-20-2019)
- Nex, F., & Remondino, F. (2014, Mar 01). Uav for 3d mapping applications: a review..

- Applied Geomatics*, 6(1), 1–15. Retrieved from
<https://doi.org/10.1007/s12518-013-0120-x> doi: 10.1007/s12518-013-0120-x
- Occupancy map 3d*. (n.d.). Retrieved from
<https://www.mathworks.com/help/nav/ref/occupancymap3d.html> (Accessed:
 10-24-2019)
- Pfaff, P., Triebel, R., & Burgard, W. (2007). An efficient extension to elevation maps for outdoor terrain mapping and loop closing. *Sage*.
- Prazenica, R., Kurdila, A., Sharpley, R., & Evers, J. (2006, 07). Vision-based geometry estimation and receding horizon path planning for uavs operating in urban environments. In (p. 6 pp.). doi: 10.1109/ACC.2006.1657155
- Quad-tree*. (n.d.). Retrieved from <https://www.geeksforgeeks.org/quad-tree/> (Accessed:
 02-20-2019)
- Quadtrees*. (n.d.). Retrieved from
<http://mooring.ucsd.edu/software/matlab/doc/toolbox/datafun/private/quadtree.html>
 (Accessed: 01-26-2019)
- Rivadeneira, C., & Campbell, M. (2011). Probabilistic multi-level maps from lidar data. *The International Journal of Robotics Research*, 30(12), 1508-1526. Retrieved from
<https://doi.org/10.1177/0278364910392405> doi: 10.1177/0278364910392405
- Rivadeneira, C., Miller, I., Schoenberg, J. R., & Campbell, M. (2009, May). Probabilistic estimation of multi-level terrain maps. In *2009 IEEE International Conference on Robotics and Automation* (p. 1643-1648). doi: 10.1109/ROBOT.2009.5152767
- Rivera, K. (2018). *Design and implementation of intelligent guidance algorithms for uav mission protection* (Master's thesis). Retrieved November 2019, from Hunt Library at Embry-Riddle Aeronautical University.
- Roboticsoccupancymap3d class*. (n.d.). Retrieved from
<https://www.mathworks.com/help/robotics/ref/robotics.occupancymap3d-class.html>
 (Accessed: 02-01-2019)
- Ryde, J., & Hu, H. (2009, Sep 30). 3d mapping with multi-resolution occupied voxel lists. *Autonomous Robots*, 28(2), 169. Retrieved from
<https://doi.org/10.1007/s10514-009-9158-3> doi: 10.1007/s10514-009-9158-3
- Saarinen, J., Andreasson, H., Stoyanov, T., Ala-Luhtala, J., & Lilienthal, A. J. (2013, May). Normal distributions transform occupancy maps: Application to large-scale online 3d mapping. In *2013 IEEE International Conference on Robotics and Automation* (p. 2233-2238). doi: 10.1109/ICRA.2013.6630878

- Sasikanth. (2011). *Modified hausdorff distance*. Retrieved from <https://www.mathworks.com/matlabcentral/fileexchange> (Accessed: 08-01-2019)
- Schadler, M., Stückler, J., & Behnke, S. (2013, Oct). Multi-resolution surfel mapping and real-time pose tracking using a continuously rotating 2d laser scanner. In *2013 ieee international symposium on safety, security, and rescue robotics (ssrr)* (p. 1-6). doi: 10.1109/SSRR.2013.6719373
- Schmuck, P., & Chli, M. (2017, May). Multi-uav collaborative monocular slam. In *2017 ieee international conference on robotics and automation (icra)* (p. 3863-3870). doi: 10.1109/ICRA.2017.7989445
- Shojaeipour, S., Haris, S., Eftekhari, E., Shojaeipour, A., & Daghigh, R. (2010, 05). Vision based trajectory generation for a robotic arm using quad-tree decomposition and curve smoothing. *Advanced Materials Research*, 108, 1439-1445. doi: 10.4028/www.scientific.net/AMR.108-111.1439
- Shusterman, E., & Feder, M. (1994, March). Image compression via improved quadtree decomposition algorithms. *IEEE Transactions on Image Processing*, 3(2), 207-215. doi: 10.1109/83.277901
- Siciliano, B., & Khatib, O. (2016). *Springer handbook of robotics* (2nd ed.). Springer.
- Stulgis, A., Ambroziak, L., & Kondratiuk, M. (2018, Aug). Obstacle detection and avoidance system for unmanned multirotors. In *2018 23rd international conference on methods models in automation robotics (mmar)* (p. 455-460). doi: 10.1109/MMAR.2018.8485911
- Tang, Y., Hu, Y., Cui, J., Liao, F., Lao, M., Lin, F., & Teo, R. S. H. (2019, Jan). Vision-aided multi-uav autonomous flocking in gps-denied environment. *IEEE Transactions on Industrial Electronics*, 66(1), 616-626. doi: 10.1109/TIE.2018.2824766
- Torr, P. H. S., & Zisserman, A. (2000). MLESAC: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78, 138–156.
- Triebel, R., Pfaff, P., & Burgard, W. (2006, Oct). Multi-level surface maps for outdoor terrain mapping and loop closing. In *2006 ieee/rsj international conference on intelligent robots and systems* (p. 2276-2282). doi: 10.1109/IROS.2006.282632
- Vallivaara, I., Poikselkä, K., Kemppainen, A., & Röning, J. (2018). Quadtree-based ancestry tree maps for 2d scattered data slam. *Advanced Robotics*, 32(5), 215-230. Retrieved from <https://doi.org/10.1080/01691864.2018.1436468> doi: 10.1080/01691864.2018.1436468

- Vanegas, F., Gaston, K. J., Roberts, J., & Gonzalez, F. (2019, March). A framework for uav navigation and exploration in gps-denied environments. In *2019 ieee aerospace conference* (p. 1-6). doi: 10.1109/AERO.2019.8741612
- Vergara, P., Tiwari, M., Prazenica, R. J., & Henderson, T. (2019, Jan 06).
In (chap. Multiresolution-Based 3-D Terrain Estimation Algorithms for Complex Urban Environments). American Institute of Aeronautics and Astronautics. (0) doi: 10.2514/6.2019-1194
- Vicon vantage reference* (1st ed.). (2016, 4). (Hardware Manual)
- Virtual competition*. (n.d.). Retrieved from <http://www.imav2017.org/virtual-challenge/>
(Accessed: 07-10-2018)
- Vlp-16 user manual* (1st ed.). (2017, 12). (Hardware Manual)
- Wolf, D. F., & Sukhatme, G. S. (2003). *Towards mapping dynamic environments*.