Lehrstuhl für Computergestützte Modellierung und Simulation
TUM School of Engineering and Design
Technical University of Munich

Model-based UAV mission planning for photogrammetric capture of existing buildings

Scientific work to obtain the degree

Bachelor of Science (B.Sc.)

at the Institute of Informatics of Ludwig-Maximilians-University Munich.

Supervised by Prof. Dr.-Ing. Frank Petzold
Prof. Dr.-Ing. André Borrmann
Ann-Kristin Dugstad, M.Sc., Florian Noichl, M.Sc.
Lehrstuhl für Computergestützte Modellierung und Simulation

Submitted by Betina Koleva (10911199)
Arcisstraße 21
D-80333 München
e-Mail: Betina.Koleva@campus.lmu.de

Submitted on 17. January 2022

Acknowledgments

First and foremost, I would like to sincerely thank my thesis advisors Ms. Ann-Kristin Dugstad and Mr. Florian Noichl of the Chair of Computational Modeling and Simulation at the Technical University of Munich, for giving me this opportunity. Without their support and guidance, this project would not have been possible.

I appreciate the effort of Ms. Dugstad to manage the communication between the Technical University of Munich and Ludwig Maximilian University.

I wish to express my profound gratitude to Prof. Dr.-Ing. Frank Petzold and Prof. Dr.-Ing. André Borrmann for supervising this thesis. Without their agreement, I would not have been able to work on this project as an LMU student.

Finally, I would like to thank my family for their support and encouragement.

It was a pleasure developing this project and writing the thesis.

Abstract

At the scene of an emergency, such as an accident or a natural disaster, first responders require an actual and rapid analysis of the environment with all its possible dangers. Gaining insight into the operating environment will help first responders effectively prevent radical damages and save potential victims.

With outdoor Unmanned Aerial Vehicles (UAVs), first responders receive scanned information of disaster zones fast and accurately. That obtains a precise representation of the situation for emergency response work. An essential requirement for the usability of the drone scans is the preliminary calculation of an optimal path. A drone path aims to generate good scans for a better overview of the scene considering parameters such as a reasonable distance and overlap. A distance between the drone and the examined build environment ensures scans accuracy and collisions avoidance. The overlap helps the 3D reconstruction of the object to be scanned.

This bachelor thesis focuses on developing a module that generates a route allowing the scanning of buildings. Software for automated path planning is developed by collecting parameters describing what conditions an optimal path must fulfill and inspecting existing path planning strategies for outdoor UAVs. The generated tool produces a path, represented as several waypoints, that allows an optimal point cloud acquisition depending on the given camera and the criteria defined with the objectives of the thesis. The thesis is part of the EU-funded INTREPID project for helping the first responders.

Zusammenfassung

Während eines Notstandes, beispielsweise eines Unfalls oder einer Naturkatastrophe, müssen Ersthelfer eine aktuelle und schnelle Umgebungsanalyse mit all ihren möglichen Gefahren erhalten. Ein Überblick in die Betriebsumgebung führt zur einer schnelleren Reaktion der Ersthelfer bei der Rettung von potenziellen Opfern und zur Vermeidung radikaler Schäden.

Mit einem „Outdoor Unmanned Aerial Vehicle“ (UAV) erhalten die Ersthelfer eine schnelle, präzise und gescannte Information über die Katastrophengebiete. Dadurch erhält man eine genaue Darstellung der Situation für die Notfalleinsätze. Eine wesentliche Voraussetzung für die Verwendung der UAV Fotogrammetrie ist die vorläufige Berechnung eines optimalen Pfades. Ein Drohnenpfad zielt darauf hin, verwertbare Scans, für einen besseren Überblick über die Szene zu verschaffen, wobei die Parameter, wie beispielsweise ein vernünftiger Abstand oder eine Überlappung etc. berücksichtigt werden. Ein Abstand zwischen der Drohne und der untersuchten Umgebung gewährleistet die Genauigkeit der Scans und die Vermeidung von Kollisionen. Die Überlappung hilft der 3D-Rekonstruktion des zu scannenden Objekts.

Das Ziel der vorliegenden Bachelorarbeit ist die Entwicklung eines Moduls, das einen Pfad generiert, die das Scannen von Gebäuden ermöglicht. Im Rahmen dieser Abhandlung wird daher eine Software für die automatisierte Pfadplanung entwickelt. In dieser Hinsicht werden Parameter gesammelt, welche beschreiben, inwiefern diverse Bedingungen ein Pfad optimal erfüllt werden müssen. Außerdem werden bestehende Pfadplanungsstrategien für Outdoor-UAVs untersucht. Das generierte Modul erzeugt einen Pfad, der eine optimale Punktwolkenaufnahme in Abhängigkeit von der gegebenen Kamera und den definierten Parametern ermöglicht. Die Bachelorarbeit ist ein Teil des EU-finanzierten INTREPID-Projekts für die Unterstützung der Ersthelfer.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

2D	two-dimensional
3D	three-dimensional
BIM	Building Information Modeling
FOV	Field of View
GSD	Ground Sampling Distance
IFC	Industry Foundation Classes
MOGA	Multi-Objective Genetic Algorithm
NBIMS	National Building Information Modeling Standard
SA	Simulated Annealing
SRMSRE	Standardised Root Mean Squared Re-projection Error
TSP	Travelling Salesman Problem
UAV	Unmanned Aerial Vehicle

Chapter 1

Introduction

1.1 Motivation

At the scene of an emergency, such as an accident or a natural disaster, first responders operate in a dangerous and chaotic environment to prevent radical damages and rescue potential victims. The lack of insight into the unsafe environment presents an obstacle to quick and effective intervention. The goal of the INTREPID (Intelligent Toolkit for Reconnaissance and assessment in Perilous Incidents) research project is to create a unique platform to provide three-dimensional (3D) exploration and analysis of the scene of an emergency. The project contributes an accurate analysis of the dangerous areas by integrating intelligence amplification and extended reality concepts with Smart Cybernetic Assistants and innovative deep indoor networking and positioning capabilities (INTREPID, 2020). Thus, first responders are rapidly notified of the hazardous conditions in the operating environment, which results in an immediate and targeted response.

The utilization of outdoor Unmanned Aerial Vehicle (UAV) for scanning a site presents one part of the project. The utilization of drones has been widely adopted in a variety of fields requiring remote sensing, and drones are a fundamental tool for surveying and mapping an area with hard-to-reach locations. Some advantages of using a drone are data acquisition at high spatial resolution for a target area and reduced operational costs (Domingo et al., 2019; Tang & Shao, 2015). The drone-captured images serve for reconstructing a point cloud, from which a high-quality 3D model of the mapped area can be generated. Accordingly, scanning disaster scenes with UAVs will obtain a precise representation of the situation for emergency response work. Outdoor UAV scans of the disaster scene are a beneficial information source for first responders. Thus, neutralizing threats operations can start immediately as the area is preliminarily examined.

Retrieving a successful overview of the built environment from drone remote sensing in an efficient way is a significant research problem. Several works have overcome this problem by using path planning preparation. Planning a path before execution is advantageous because it helps achieve a given goal while preventing unnecessarily long flight duration and high costs. The calculation of a drone path, allowing optimal scan acquisition of the building site, is in the framework of this bachelor thesis. This study concentrates on inspecting optimally set parameters and existing path planning strategies for outdoor UAVs with the objective of developing a module that generates paths that allow for an optimal point cloud acquisition.

1.2 Thesis objectives

This bachelor thesis focuses on implementing a path planning module for a given outdoor drone and a Building Information Modeling (BIM) input. The project goal is the drone's ability to scan the outside of the building from above and side. The scans serve the create a quality 3D reconstruction. The generated path, represented by a number of waypoints and their position defined by x, y, and z coordinates, is required to fulfill some objectives defined based on comprehensive literature analysis. The selection of parameters is one of the major topics to be investigated in this thesis. That is due to ensuring a good scanning with a high-quality 3D reconstruction. One requirement for safe flight execution is distance parametrization. The background is that selecting a reasonable distance helps collisions avoidance or drone damages during the flight while considering scans efficiency. A certain percentage of overlap, for instance, supports a detailed and accurate 3D reconstruction of the object to be scanned. Further requirements are generated based on state-of-the-art, conducted as the first step of this thesis. The method is implemented in python and tested with the BIM model on different build environments.

1.3 Layout of the thesis

The bachelor thesis has the following structure:

- Chapter 2 introduces the theoretical background for the optimal selection of flight and sensor parameters. The chapter also presents a comprehensive literature analysis of the existing path planning strategies for drone image-capturing missions.
- Chapter 3 describes the concept of the developed method in detail. The chapter provides the workflow of the different stages of the method to be implemented.
- Chapter 4 discusses the implementation of the method.
- Chapter 5 focuses on the results and validations of the implemented method.
- Chapter 6 summarizes the results and proposes future investigations.

Chapter 2

State of the Art

The following chapter presents the theoretical basis of this thesis regarding outdoor UAV path planning. The first section provides an overview of the benefits and the use of UAV remote sensing, followed by literature research about the possible flight and sensor parameters, which aims to define the objectives that the developed method for the thesis should fulfill. The last section is dedicated to drone path planning by presenting the algorithms used in the literature for defining the UAV route. Gaining insight into the different possibilities is the basis for deciding on a suitable algorithm for the goal of the thesis: developing a path planning tool, which optimizes the point cloud acquisition.

2.1 Overview

With the recent technical advances in UAV technologies, drones are becoming a powerful remote sensing tool. The usage of UAVs for remote sensing provides a low-cost alternative to the manned aircraft for data acquisition at high spatial resolution for both indoor and outdoor target areas. With drones, hard-to-reach and dangerous locations became accessible for analysis. Drones have the advantage of being more flexible, cost-effective, and enabling controlled flight repeatability due to improving autopilots (TANG & SHAO, 2015; WATTS et al., 2012). Flying forward and backward at a constant velocity is beneficial for more complex missions such as target location or obstacle avoidance (HERNANDEZ-LOPEZ et al., 2013). The possibility of low-altitude flight and the provided lower Ground Sampling Distance (GSD) allows observations from closer positions, thus finer spatial resolution data collection (ANDERSON & GASTON, 2013). The flight time depends only on the fuel or battery life, and drones are useful for missions beyond human limitations (GUPTE et al., 2012).

Formerly used mainly for military purposes, nowadays drone remote sensing finds applications in many civilian fields. UAVs are widely utilized in meteorology, agriculture, land surveying, traffic monitoring, scientific research, and natural disaster management (TANG & SHAO, 2015). The capability of a high spatial resolution benefits urban applications and planning. UAV remote sensing supports infrastructure inspections and rescue operations in complex buildings. The construction sector applies UAVs remote sensing to the automated creation of images and point cloud data of particular construction objects for enabling construction management. The visual data acquisition became uncomplicated through flight missions in a known structure of the construction site. (FREIMUTH & KÖNIG, 2019). In this line of thought, the utilization of BIM provides an opportunity to predefine a path by extracting positions of objects to be inspected.

The utilization of BIM, either generated during the planning phase of the building construction or created based on collected point cloud data, provides an extensive overview of the state of the construction process. According to the National Building Information Modeling Standard (NBIMS), BIM model is a digital representation of the physical and functional characteristics of a facility. It serves as a shared information source about a facility establishing a reliable decision-making basis for different involved parties such as stakeholders and contractors (SMITH, 2009). Large projects require construction management to maintain schedules of all construction processes. The construction process is also associated with constant changes and modifications, leading to delays and inconsistencies. Visualizing the project with BIM improves the coordination and estimate of costs and risks (FREIMUTH & KÖNIG, 2019). Commonly used for BIM is the Industry Foundation Classes (IFC) format. IFC is the standardized, digital description of building and construction data, which allows the exchange of the BIM model without loss of data (BUILDINGSMART, 2021). The format saves the whole geometric representation and allows editing. Figure 2.1 represent a visualization of Technical University main campus using IFC. The visualization is realized with Python Ifcopenshell - an open-source software library that helps users working with IFC file format.

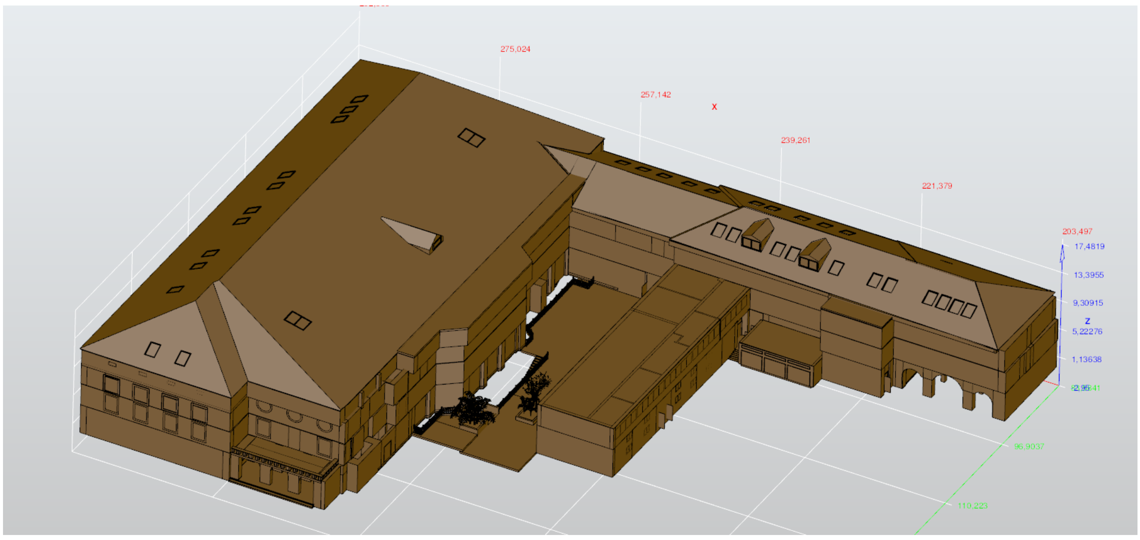


Figure 2.1: Technical University main campus, Munich.

In drone missions utilizing the BIM model can be used as a simulation environment for drone path planning. The model geometry of the examined building is used to create a collision-free flight plan. In addition, the use of BIM for path planning helps to evaluate the visual coverage and safety of execution before flight deployment. Determining drone paths is essential for creating efficient scans while preventing missing the collection of informative visual data and longer flight duration (IBRAHIM & GOLPARVAR-FARD, 2019). The drone-captured images are used to reconstruct a point cloud, from which a high-quality 3D model of the mapped area can be generated. One primary problem is the optimal selection of flight and sensor parameters of the drone. Parameters such as altitude, image overlap, and sensor resolution should be considered. Finding an optimal path in a 3D model can be challenging as this problem has been widely addressed in the literature. Many authors found solving the mission planning in several ways. The solution can be defined by the number of waypoints, their coordinates, and the camera orientation (Ibrahim & Golparvar-Fard, 2019).

2.2 Parametrization

Optimally set parameters before drone flight planning allow high-quality scans of the target object or area while considering the execution efficiency. However, prioritizing scan quality can result in longer flight duration and longer data processing time to achieve reconstruction precision. A critical open question is how to define the parametrization best in order to achieve the optimization goal. The optimal parameter selection can be challenging, and multiple adjustments may be needed to obtain an accurate reconstruction efficiently. Some of the parameters to be considered in path planning for drone remote sensing are the selection of flight altitude, flying speed, image overlap, and digital camera parameters, particularly spatial resolution, and focal length (Eisenbeiss, 2009). As identified by Manconi et al. (2019), all those factors influence the GSD and the accuracy of the result. Therefore, the parametrization needs to be adjusted precisely.

Much recent literature has investigated the parameter selection for drone-based remote sensing in forest and agriculture research. For example, the paper of Dandois et al. (2015) explores the connection between the photographic overlap and the optimal data collection. The study shows that maximizing forward overlap increases the accuracy. On the other hand, it results in more photos and increased processing time, highlighting significant trade-offs between data quality and efficiency. Nevertheless, the optimization of the acquisition in this research is limited, because the author does not take the effect of parameters adjustment into consideration. A systematic study of optimal parametrization was performed by Seifert et al. (2019). The author discusses the selection of flight and sensor parameters and their impact on the generation of good-quality point clouds efficiently. The study concentrates on the influence of drone variables on flight time, reconstruction details and precision, and data processing time. The flight and sensor parameters are optimized to ensure image reconstruction accuracy in a suitable flight and image processing time. The main trade-off occurs between quality and efficiency. Therefore, the path planner can directly select the parameters such as altitude, image overlap, and sensor resolution to reach an optimal compromise. The study shows that flight time is linearly related to flight height. The lower the altitude, the higher the spatial resolution, but the longer the flight time. In their study about the influence of flight parameters of UAV orthomosaics, Mesas-Carrascosa et al. (2016) evaluate altitude above ground level as one of the most significant parameters for accuracy, flight duration, and area coverage. Confirming the conclusion of Seifert et al. (2019), the author points out that a very low selected altitude guarantees a high spatial resolution but covers a limited area and therefore increases flight duration. According to Seifert et al. (2019), lower flight altitude also leads to more images processed per unit area, which increases data processing time. The drone camera and altitude define GSD that determines the distance between two pixels on the ground. Flying too close to the surface results in many images needing processing. Conversely, flying too far worsens the reconstruction details (ZOU et al., 2019). A higher altitude affects the precision of the image reconstruction in a negative linear pattern. Therefore, a low altitude scan results in higher spatial resolution, which yields a more detailed and precise reconstruction in exchange for a longer processing time. Image processing time also depends on the forward overlap. Increasing the forward overlap affects the processing time exponentially. Another disadvantage of high forward overlap is the negative impact on flight duration. In order to achieve accuracy, more images in each lap are captured, and thus the number of total laps increases (MESAS-CARRASCOSA et al., 2016). SEIFERT et al. (2019) summarize that low flight height, combined with high forward overlap, leads to the best reconstruction precision and detail. Figure 2.2 illustrates the results of his study about the impact of different parameters on reconstruction precision. The author proves that the Standardised Root Mean Squared Re-projection Error (SRMSRE) slightly increases with the altitude. Oppositely, the high overlap and higher sensor resolutions lead to significantly smaller errors. Studies by DANDOIS et al. (2015) and TORRES-SÁNCHEZ et al. (2015) also suggest that the higher forward overlap increases the accuracy. Similarly, DOMINGO et al. (2019) does not recommend a reduction of the forward overlap due to the fact that the lower number of images acquired during drone remote sensing produces a larger accuracy error rate. Commonly, the side overlap is suggested to be reduced in a range between 50 – 70 % while keeping the forward overlap high in order to achieve the best-quality scans (DANDOIS et al., 2015; DOMINGO et al., 2019; SEIFERT et al., 2019). Regarding the sensor resolution, SEIFERT et al. (2019) conclude that higher sensor resolution drastically increases the precision. Sensor resolution influences the processing time in a linear pattern, especially in combination with a low altitude.

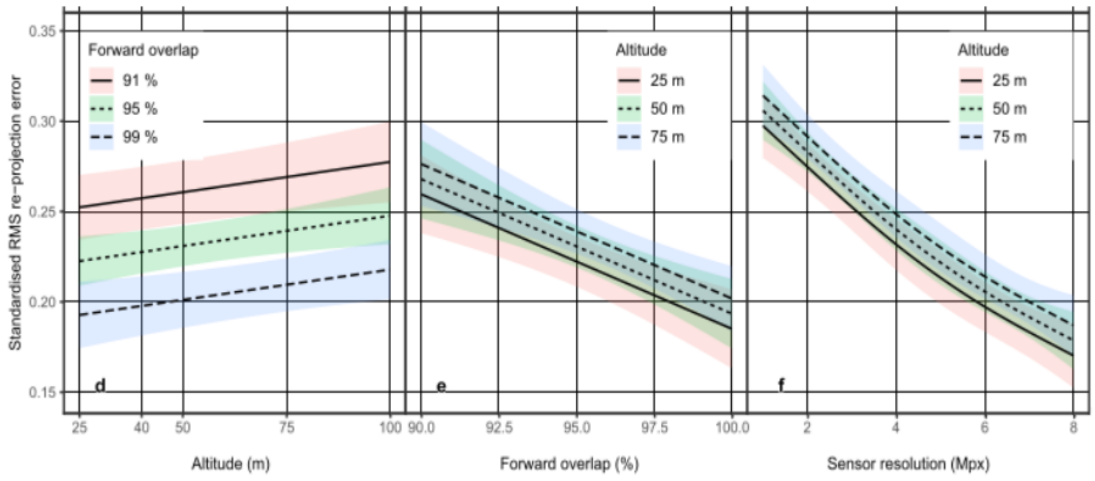


Figure 2.2: Observations for Standardised Root Mean Squared Re-projection Error versus flight/sensor parameters (Seifert, 2019).

Although there are many recent studies, the research in parametrization for urban studies remains limited. The land surveying research gives a good overview of setting the optimal parameters problem, but most studies only aim to gain data acquisition above the target area. Surveying forest height with remote sensing, for instance, demands a nadir viewing to capture it from above. Compared with that, investigating a building requires scanning performed for both the façades and the roof of the building. Some of the requirements overlap the ones expected in forestry research, such as altitude, image overlap, and flight speed. A reasonable horizontal distance between the drone and the examined building is also a factor in the collision-free capture of a façade. In addition to the drone safety, information about the vertical walls is provided, and not only about the horizontal planes (Roca et al., 2014). Zou et al. (2019) summarizes the parameters needed to develop automated path planning algorithms: setting the distance and altitude between the drone and the target object and camera angle of view. Equally significant is also the definition of the UAV speed and an adequate overlap of images.

2.2.1 Altitude and Distance

The literature review shows that altitude is an important requirement in order to generate accurate and detailed scans in high spatial resolution while considering the flight time and data processing time. In addition, setting a distance between the drone and building ensures a better overview of the scene for creating precise 3D reconstruction. The drone should fly at a maximum distance from a building to avoid collisions or drone damages during the flight. At the same time, a minimum distance needs to be determined, at which the scans still provide high visual coverage and accuracy for detailed 3D reconstruction.

Several researchers performed their studies using different altitudes to evaluate which one yields the best precision. For instance, Dandois et al. (2015) suggests four levels of flight altitude above the canopy in a range between 20-80 m with predefined automated drone waypoint paths based on the designated flight altitude. The study also shows that coarsening the GSD by increasing the drone acquisition height reduces the point cloud position accuracy. According to Seifert et al. (2019), a relatively low flight height in a range between 15-30 m above the desired object to be scanned in combination with the highest overlap possible improves the reconstruction precision and detail. Domingo et al. (2019) suggest that instead of choosing the flight altitude, it should be determined according to the camera spectral sensitivity to generate constant 10 and 15 m GSD pixel resolution images. As a result, the drone flight is executed with a 325 m flight height above the ground. GSD can be calculated using the following equation:

$$GSD = \frac{p}{f} * H$$

where GSD is the ground sample distance (cm), *p* is pixel size (mm) on the sensor, *f* is the focal length (mm), and *H* is the distance from the camera projection center to the ground in cm (Seifert et al., 2019). To simplify the calculation of altitude and distance between the drone and building, Roca et al. (2014) propose a 10 m distance from both the ground and the façade. The same contributions have been made by Themistocleous et al. (2016) capturing the Foinikaria Church in Cyprus. Perazzo et al. (2016) point out that a predefined altitude might differentiate from the actual one during execution because factors such as weather conditions are not considered in the stage of path planning. Due to this, a source of altitude precision control is needed. In this paper about path planning algorithms, a bounded precision error is suggested as a solution for the problem of measuring the difference between the planned and actual altitude. In summary, the results of the literature review set the first objective of the thesis. The tool should generate paths ensuring a 10 m distance between drone and building. This fixed requirement keeps the solutions robust and safe.

2.2.2 Image Overlap

Several studies have been focused on the overlap ratio effect on accuracy. According to HÖHLE (2013), the standard forward and side overlap is about 60%. In recent years, the majority of authors disproved this statement with their studies: a fixed forward overlap above 90% leads to the best reconstruction detail and precision. Unlike the forward overlap, the side overlap is maintained in the range between 50 – 70 %. DOMINGO et al. (2019) state that reducing the side overlap from 80% to 70% while keeping the forward overlap high might be a solution to reduced flight duration. In addition, the paper of SEIFERT et al. (2019) shows that the middle levels of side overlap yield an optimum reconstruction accuracy. Therefore, the second objective for the thesis based on the literature is keeping forward overlap above 90%.

2.2.3 Camera angle of view

Regarding the camera angle, few researchers have addressed a precise parametrization. SEIFERT et al. (2019) state that the effect of a short focal length while flying close to the canopy increases the number of viewing angles. The angle of view describes the capture of multiple objects from the camera sensor and the determination of their distance (Hell, 2019). To receive a precise reconstruction, each point of the area to be captured needs to be scanned from multiple angles while also considering the Field of View (FOV). Defining the maximum area imaged by the camera, FOV is directly related to the distance to the surface. Drone performed at a higher altitude from the ground capture a more extended field of view in exchange for image quality and longer flight time (SEIFERT et al., 2019; TAURO et al., 2015). The forward overlap also depends on FOV along with altitude. As identified by DANDOIS et al. (2015), a camera with a higher resolution but narrower FOV increases in GSD and at the same time decreases in overlap for a constant altitude. HAMMOND et al. (2020) suggest using an oblique angle because a higher amount of information about the face is collected within the view. SEIFERT et al. (2019) add that although the oblique angle is not recommended for forest canopy scanning due to the need for nadir view, it provides complexity to identifying matching points. DANDOIS et al. (2015) highlight that point cloud position stability decreases rapidly after view-angle exceeds 20 degrees, which results in an increased error in canopy height estimation. This factor contributes to determining an angular range for the thesis method between 15 – 20 degrees.

2.2.4 Flight speed

Flying speed is a requirement highly influenced by the model of the drone itself. In the case of the thesis, the drone speed is up to 20 m/s. Flight speed is related to altitude and flight efficiency. Lower flight height, which yields the reconstruction accuracy and precision, requires a reduced flight speed to avoid motion blur. At the same time, low speed increases the flight duration, and the covered area possible with one battery load is limited (SEIFERT et al., 2019). For capturing wheat fields located in Spain, MESAS-CARRASCOSA et al. (2016) focus on adjusting the speed in order to keep stable forward and side overlap while considering the altitude. With a flight height above the ground of 60 m, the flight speed was set to 2 m/s, for higher altitude – 3 m/s, while the highest speed limit of the drone reaches 12 m/s. The selected by MESAS-CARRASCOSA et al. (2016) flight speed is equal to one-sixth of the maximum speed, or one-quarter for higher altitude. All these flight speed assumptions contribute in the case of capturing the scans in motion. Therefore, this adjustment aims to provide good quality scans without motion blur. For this thesis, the scan capturing occurs in the resting state, so the motion blur is not a factor for good scans. However, considering the literature review, in addition to the suggestion of SEIFERT et al. (2019), a speed range between 3 – 6 m/s can be provided.

2.2.5 Conclusion for the objectives

In summary, the literature review about flight and sensor parameters sets the starting point for the implementation. The following objectives are contributed:

- A 10-meter distance between the drone and the building for safety and efficiency.
- A forward overlap above 90% for achieving good reconstruction accuracy.
- An angular in a range between 15 – 20 degrees.
- Flight speed in a range between 3 – 6 m/s.

2.3 Path Planning

An essential requirement for the usability of UAV remote sensing is the preliminary calculation of a path. Path planning is the process of finding a path between points in a certain environment. Drone missions without predefined paths can increase operational costs and flight time. The path is a sequence of waypoints with an initial take-off position and a final landing position. The drone path may be closed, which means that the start and landing points can be equal (PERAZZO et al., 2016). The goal of drone path planning is to find the shortest route from an initial point to a final one through an area with or without obstacles which maximizes the data acquisition in a suitable time.

A key problem is how to define a convenient path for the drone. Finding drone path planning algorithms is part of the NP-hard optimization problems, which is due to the fact that deterministic algorithms cannot find a solution in a reasonable time (TUBA et al., 2017). Some literature sources interpret the drone path planning problem as a Travelling Salesman Problem (TSP), which also belongs to the NP-hard problems. The main task is the creation of a sequence for visiting a given list of places. Each place except the primary one must be visited once, and the entire travel distance of the traveling salesman is the shortest possible (OBERLIN et al., 2010). For UAV path planning, the objective of TSP is to plan a path that allows the UAV to visit all waypoints needed for scanning the object successfully.

Several different methods to plan the UAV path are used to fulfill goals as minimizing the route or flight duration while collecting accurate scans for 3D reconstruction. The literature review offers a palette of examples for solving UAV path planning. From artificial potential field explored by ZHU et al. (2006), which combines the attraction to the target and repulsion from obstacles to the commonly used graph search methods such as Dijkstra and A* algorithm. According to HUSSEIN et al. (2012), the classical graph-based algorithms are efficient for path planning problems by providing feasible solutions. The drawback of those approaches is that they consume much time to find a solution, especially in complex environments. Those algorithms are also prone to be getting trapped in local optimum and never reach the global one when dealing with a large environment with several possible solutions. Therefore, probabilistic methods are introduced as a less expensive alternative. Some algorithms are Simulated Annealing (SA), genetic algorithms, and ant colony optimization. The following subsections present some of the most utilized ones in the literature.

2.3.1 A* Algorithm

The A* algorithm is an extension of the Dijkstra algorithm, which finds the shortest path in a graph starting from an initial node. The distance to each neighboring node is calculated and compared, and the one with the lowest distance is selected. The disadvantage of Dijkstra is that the algorithm consumes a large amount of time to find the global optimum solution. A* algorithm aims to find an optimal path in a shorter computational time (XIONG et al., 2015). The algorithm uses best-first search to find the lowest cost from the marked node to the target one while introducing a heuristic function to approximate the costs (RODENBERG et al., 2016; ZAREMBO & KODORS, 2013). With the evaluation of the distance between the current node and the neighbors, A* also considers the distance between the neighbors and the target node. At each iteration, the algorithm determines the following node according to the F-value, which is the sum of two functions:

$$F = H + G$$

The G-function represents the exact movement cost to reach the target, and the H is the non-overestimated cost from the current node. ZAREMBO and KODORS (2013) point out that the estimated heuristic cost must be admissible. That means that function H should not overestimate the cost to achieve the goal. A* always finds the optimal way if the cost estimation value of the H-function is underestimated.

Several studies have been published on path planning using the A* algorithm for optimization. In much relevant literature addressing path planning, the algorithm operates on two-dimensional (2D) space. XIONG et al. (2015) provide a literature example of pathfinding optimization based on the A* algorithm implemented in 3D space. In the study, the author presents a path planning approach in an indoor environment with the help of a voxel model. A voxel is a unit that represents a point in 3D space. The method introduced in the paper transforms the geometric input data into voxelized data to inherit the semantic information and uses the A* algorithm to find the most suitable path. RODENBERG et al. (2016) demonstrate an indoor pathfinding optimization based on the A* algorithm using an octree representation to segment and structure a point cloud. An octree is a data structure used to divide the 3D space by recursively subdividing it into eight equal cubes, called octants. The subdivision proceeds until blocks of uniform color are obtained, or a predetermined level of decomposition is reached (SAMET, 1982). The author indicates with the term uniform color if a block is empty or occupied. The octree structures the space efficiently and largely empty space can be represented by an empty node higher in the octree, which is advantageous for path planning, as it reduces the number of nodes in the octree. In addition, the octree contains neighbor connectivity which aims to speed up the pathfinding (RODENBERG et al., 2016). The method of RODENBERG et al., 2016 produces an octree to segment the point cloud, which is geometrically pre-processed to fit in the octree. The occupied and empty nodes are generated, and the empty ones are filtered. Until all empty nodes are computed, calculating the distance to the closest occupied node proceeds, so the A* algorithm can find the optimal path.

2.3.2 Genetic Algorithm

The Genetic Algorithm belongs to the class of evolutionary algorithms. Inspired by Charles Darwin's theory of natural selection, it presents a simulation of the biological process of human genetic improvement. The basic idea is to generate a set of optimal solutions for a predefined goal while iterating over a population, also called generation. It starts with randomly selected feasible solutions and filters the best individuals from the generation for each iteration. Basic operations as crossover and mutation are performed with a certain probability to evolve in the next generation. When the goal is reached, the process terminates (Strubel et al., 2017).

The Genetic Algorithm is mainly applied as a single-objective optimization solution. However, much UAV pathfinding research papers are investigating multi-objective problems. For instance, Ibrahim and Golparvar-Fard (2019) presents a method for automated flight planning and visual data collection, that generates a 3D fight plan template around the scanned structure to evaluate flight execution safety and camera visibility. The generated flight path is optimized using a Multi-Objective Genetic Algorithm (MOGA) to maximize visual quality and minimize UAV flight duration. MOGA selects individuals for crossover and mutation operation based on a constant weight of each multiple objective function to find an optimum solution in reduced computational time (Murata, Ishibuchi, et al., 1995). For the case in the study of Ibrahim and Golparvar-Fard (2019), new solutions are simulated for each iteration. The quality metrics of the solutions are calculated along with the flight duration in order to construct a 3D surface with non-dominated solutions, also referred to in the literature as Pareto optimal solutions. These solutions provide a suitable compromise between all the objectives. Eliminating all dominated solutions and keeping only the ones on the Pareto surface yields the best optimization results.

2.3.3 Simulated Annealing

Another probabilistic approach used in the literature for path planning optimization is Simulated Annealing (SA). The algorithm achieves a global optimization among many local optima for large-scaled problems. The idea of the algorithm comes from the physical annealing process for metal cooling. Annealing in metallurgy is a technique involving the temperature-controlled cooling of the materials to alter their physical properties. The SA algorithm imitates the material heating and cooling to reach a globally optimum solution (Hammond et al., 2020). It is an iterative model considering a predefine cooling schedule, which consists of an initial and final temperature, annealing schedule, and a number of iterations (Hussein et al., 2015). Hussein et al. (2015) explains that the initial temperature represents the starting temperature of the algorithm, while the final one is the temperature at which the algorithm becomes greedy and accepts only optimal solutions. The annealing schedule decides the temperature decrease related to the initial one iteratively.

Hussein et al. (2012) evaluate the ease of the implementation and the ability to solve complex problems with avoidance of the local optima trap as the main advantages of using the SA algorithm. Additionally, several comparative studies conclude that SA consumes less time for path planning optimization as opposed to other algorithms (Daryanavard & Harifi, 2019; Hussein et al., 2012). The SA algorithm is also established as a possible solution to TSP problem with multiple UAVs as described by Behnck et al. (2015). In the case of the study, SA is employed to find the optimized solution for two drones. First, the two initial paths are initialized by putting the points of interest into the corresponding UAV path. During the optimization, SA is applied to each path with the aim to improve the route while reducing the distance.

2.4 Conclusion

A large number of studies in the literature have examined the optimal drone path planning and the accompanying parameters helping to achieve the initial goal. To summarize, the literature review shows that parametrization is a significant part of mission planning. The precise set of parameters improves the remote sensing efficiency and accuracy of 3D reconstruction while considering the safety of the drone and time duration. The flight and sensor parameters established from the literature review serve as a background for implementing the thesis path planning method, particularly for candidate scanning points creation. The dedicated path planning algorithm part of the literature review provides different approaches to achieve high-quality scans efficiently. The majority of prior research has employed either a graph-based algorithm or probabilistic approach. The literature review raises the question of which algorithm to select for achieving a given goal. For the case of this thesis, the developed method must output a path that generates high-quality scans. A possible solution to path planning is utilizing the Traveling Salesman Problem because a given list of waypoints needs to be visited for scanning the building. The A* algorithm helps find the path between two given points, from which the TSP decides on the lowest cost. The next chapter about the methodology provides a more detailed structure of the method to be developed.

Chapter 3

Methodology

3.1 Overview

The following chapter presents the concept of the method implemented for the thesis. The method generates a path planning for outdoor drones that scan built environments. Figure 3.1 illustrates the workflow of the presented tool.

The first step of the method considers the building model preprocessing into a format that makes path planning more convenient. The input for the implemented method is a BIM model in IFC format. The 3D building model is converted into a binary voxel-grid using the external software Binvox, and the voxels are structured in an octree. A one-meter voxel grid size is defined considering the size of the building to create a realistic voxelized model.

The next step is the simplification of the voxelized model. Since the thesis presents an outside mission planning, the voxels representing the building inside are not visited and can be ignored. Filling the inside empty voxels using a flood fill algorithm aims to define the space occupied by the building model. Thus, inaccessible voxels are not considered in the path planning. A space extension on a flood-filled grid creates a boundary box for the route preparation around the building. A building dilatation gives a protected area with the aim that the drone is never too close to the building at any moment.

The second major part of the thesis method is the definition of waypoints that describes the locations from which the drone should be able to scan the built environment effectively. The calculation of the waypoints takes into account the flight and sensor parameters set in Section 2.2.5. A waypoint is created at a given distance away from the building to avoid collisions and ensure high visual coverage and away from all neighborly points to ensure a certain percentage of overlap. The coordinates of the nodes from which the building is scanned are defined by computing a rotation to optimally align a node with its neighborly occupied voxel, indicating a building surface and the fixed distance. The point candidates capturing the edges of the building are determined by rotation over a corresponding axis.

The next step is performing a path planning algorithm. For the method of thesis, a Traveling Salesman Problem is implemented using A* path planning algorithm in 3D space to find the next nearest neighbor. There are three approaches in consideration with TSP. The first approach uses TSP with the nearest neighbor dividing the waypoints into clusters. The second one takes an initial point and sorts the rest of the waypoints by distance away from it in ascending order. The closest n-points to the initial point are selected, and the TSP is performed on them to find the nearest neighbor. The last TSP uses the cheapest insertion implementation. Each point is inserted on the path position, where it produces the lowest cost. In the last step, the gained coverage with a particular set of points in their predefined sequence is analyzed.

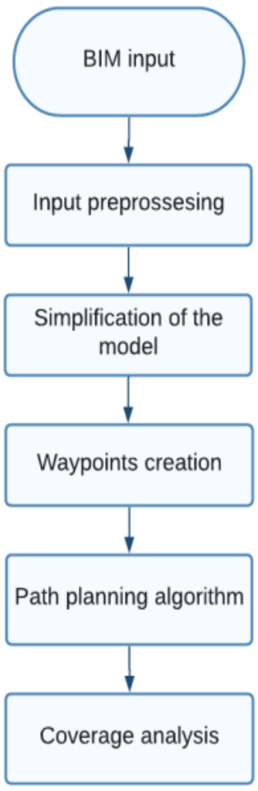


Figure 3.1: Workflow overview.

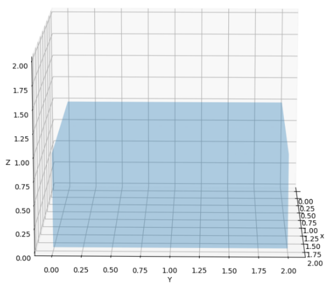
3.2 Preprocessing

The first part of the method is dedicated to the transformation of the given BIM model in IFC format to a voxel-based representation of the building. The transformation aims to create a more convenient structure for path planning. Voxelization is a process of converting data with geometric information into a 3D voxel grid (FANG & CHEN, 2000).

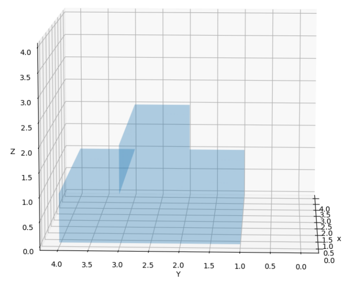
The method takes BIM model in IFC format as input and represents it as voxel-based data. In order to ensure a precise voxelized representation of a building, the building measurements in meters are taken into consideration. Calculating the bounding box directly from the IFC file allows obtaining an overview of the width, height, and length of the building model to be voxelized. Thus, voxels of size 1x1 can be produced, securing realistic voxelization of one-meter-sized cubes. Finding the nearest power of two to the longest dimension of the bounding box defines the voxel grid size. However, obtaining the correct building representation requires a bounding box adjustment which is realized by extending the longest building dimension using the formula:

$$\frac{2^n}{l} \times (b - a) + a$$

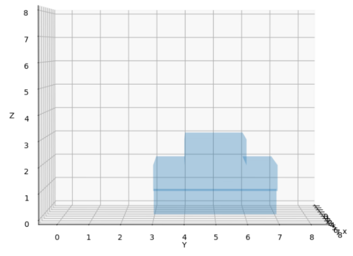
where n is the nearest next power of 2 to the longest building dimension l, b is the maximum value in the bounding box for this dimension, while the a is the minimum. The following example provides a clearer understanding of the formula. The Technical University main campus model, presented in Figure 2.1, has 89m length, 67m width, and 20m height as measurements. The resulting bounding box from the IFC file has the coordinates [203.5, 83.6, -2.6, 292.9, 150.9, 17.5]. The longest dimension is x, meaning the next power of 2 will be 128. Using the formula $\frac{2^7}{128} \times (292.9 - 203.5) + 203.5$, the bounding box is adjusted to the new proportions of the longest dimension. This returns the new coordinates of the bounding box [203.5, 83.6, -2.6, 332, 150.9, 17.5] considered for the voxelization.



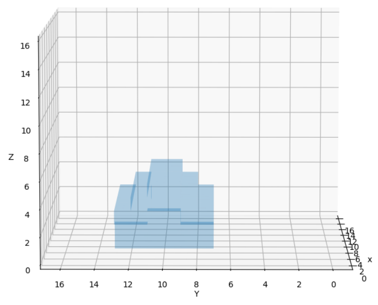
(a) Level 1



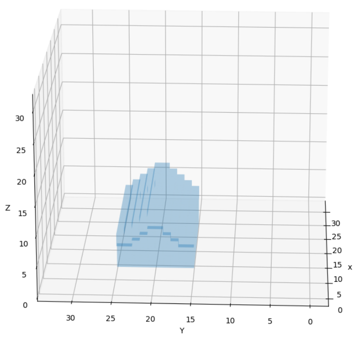
(b) Level 2



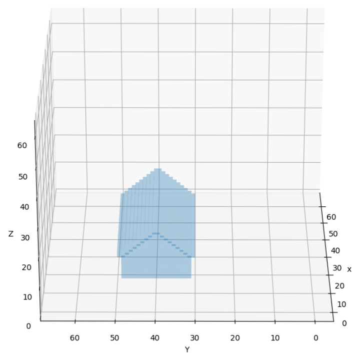
(c) Level 3



(d) Level 4



(e) Level 5



(f) Level 6

Figure 3.2: The voxel precision for the different depths in the octree.

The measurement and the voxel grid size, in addition to the BIM model in IFC format, are used as an input to the voxelization process. First, the IFC is converted to an object file. The voxel transformation occurs with the help of the external software binvox mesh voxelizer, created by MIN (2004 - 2019a). Octree segments and structures the created voxel-based data. The octree structure recursively subdivides the voxel grid space into eight equal cubes until reaching the minimum voxel size. An advantage of the octree is effective structuring the space and largely empty space can be represented by an empty node higher in the octree. The octree architecture in this thesis is mainly used to generate a grid from a level of an octree. Each level represents provides different precision. A deeper level yields a more precise vision of the building, which is needed to detect details in more complex models. The following example is a simple house with four walls and a tiled roof provided by IFCOPENSHELL.ORG (2021b). Figure 3.2 illustrates the possible different levels and their impact on the result. With the highest level of the octree, meaning just the root of the octree is taken, the shape of the house has the form of a simple cube. With each lever deeper, the voxels are more precise, which results in a more detailed illustration. The lowest level chosen in the thesis presents the voxelization of one-meter-sized cubes. The limit of this size is due to the initial choice of voxels size in the previous step.

3.3 Simplification of the model

The resulting array from the octree representation gives a starting point for producing waypoints. The waypoints creation is executed considering guaranteeing a certain overlap for the 3D reconstruction of the object to be scanned. The occupied voxels are defined in the input model and marked with one in the octree array representation. Model simplification is applied to prevent creating waypoints in unreachable positions. Waypoints should not be created inside or close to the building or outside the voxel grid. Furthermore, eliminating inapplicable voxel positions yields a faster computational time since fewer nodes need to be visited. The following workflow presents the simplification steps.

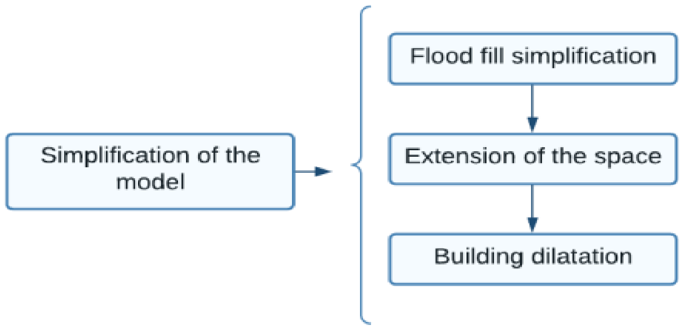


Figure 3.3: Simplification of the model - workflow.

The empty voxels inside the building, which are not relevant for scanning the surface of the building, must not be considered as possible coordinates candidates for a waypoint creation. Therefore, a flood fill algorithm is proposed to mark those voxels as unavailable. Flood fill algorithm, also known as seed fill algorithm, takes a given node in a multidimensional array and determines the connected area around the node. In computer graphics, the algorithm captures pixels of one color in a digital image and replaces them with a new color (VANDEVENNE, 2018). The algorithm displays the connected area to the starting node with the same color. In the case of the thesis, the flood fill algorithm marks the accessible voxels on which path planning can succeed. Since the flood filling occurs for the outside empty voxels, all inside voxels remain with the same value. Reversing the results will switch the values inside and outside, meaning all inside voxels become flood-filled, which considers situations where the building model contains more than the building, shown in Figure 3.4.

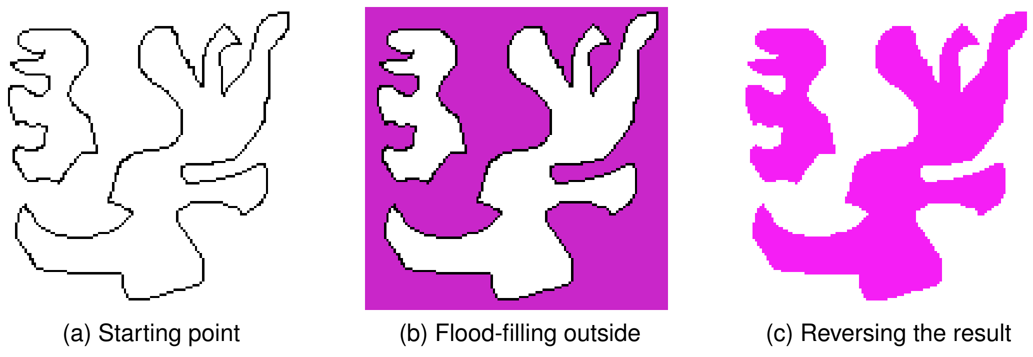


Figure 3.4: Flood fill algorithm, implemented for the thesis applied in computer graphics.

An extension of each axis of the flood-filled grid with a fixed amount prevents waypoints from being created outside the grid space. The thesis suggests each axis to be extended with the chosen distance away from the building. The last simplification step is dilating the resulting array from the flood fill algorithm. The main idea of the dilatation is to ensure a minimum distance around the building in all directions. This area will remain free of waypoints.

3.4 Waypoints creation

Waypoints represent the coordinates visited from the drone for scanning the building and the direction it faces the building. Therefore, this selection is essential to secure the efficiency of the scanning process. Creating of points is performed on the resulting array from the flood fill algorithm in view of the fact that the inside occupied voxels are not reachable.

The new coordinates result from a rotation to align a vector between a voxel and its occupied neighbor. The waypoints denoting the edges of the building in the voxel-based model are created by calculating a rotation matrix and the Euler angle for the corresponding axis. A point is located at a fixed distance from the building. The selected distance must provide good scans and avoid collisions. Furthermore, a distance between two neighborly waypoints is calculated in a way to ensure a certain amount of overlap between the two images.

The last step of the waypoint creation is filtering irrelevant points. The function filters points located in coordinates occupied or outside the voxel grid. The building dilatation created in the simplification step prevents waypoints from being close to the surface to catch possible outliers at the edge of the building.

3.4.1 Structure of the algorithm

A point is created at a given distance from the building to avoid collisions while ensuring scanning quality. Furthermore, two neighborly waypoints must be located at a certain distance from each other to secure a percentage of overlap between two images. This is due to the fact that the lowest level of octree precision set in the thesis yields the most accurate representation of the building model, as explained in Section 3.2. Each voxel in created preprocessing part represents one cubic meter from the building. Each occupied outside voxel receives a waypoint, from which the drone will be able to scan this piece of the building. To control the percentage of overlap between two images, a particular distance between two points guarantees the overlap.

First, the distance results from a calculation between the overlap, the desired distance from waypoint to object, and the camera field of view. The camera field of view and the distance are set by default. Finding the visibility length with given camera parameters means breaking the problem into two right triangles. Knowing the angle α from FOV and the adjacent side, the opposite side corresponding to length and shown in Figure 3.5, can be calculated by using the formula:

$$(\tan(\frac{\alpha}{2}) * distance) * 2$$

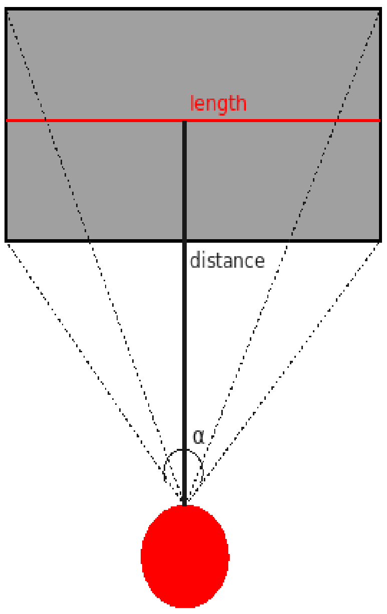


Figure 3.5: Camera Field of View with given distance and searched length in red.

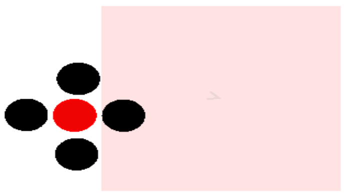
Distance is received by multiplying the resulting length to the complementary of chosen overlap:

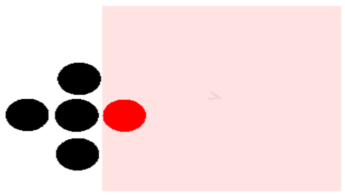
$$distance = length * (1 - \frac{overlap}{100})$$

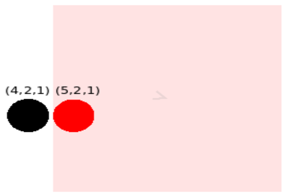
The waypoint algorithm starts at a given position by collecting the coordinates of the neighborly voxels surrounding the particular node (3.6a). It then iterates over the list of neighboring points until finding any neighbor on the building surface, defined by whether the voxel at the neighbor position is occupied (3.6b). In this case, the function calculates the optimal rotation matrix to align a vector between the current point and the neighbor. In the example from Figure 3.6c, the current node has the coordinates (4,2,1), and the position of its neighborly voxel is (5,2,1), meaning a movement along the x-axis occurs. Multiplying the optimal rotation matrix calculated with the Kabsch algorithm by the selected distance from the building represented as a vector and summing the result with the coordinates of the neighbor evaluates the coordinates of the newly created point. The resulting rotation matrix for the example in Figure 3.6d is multiplied with the distance vector and added to the coordinates of the neighbor voxel, resulting in the coordinates of the waypoint:

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -5 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 5 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix}$$

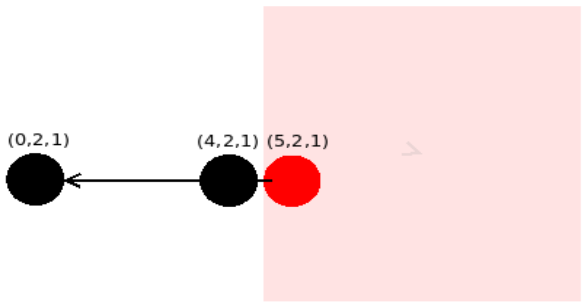
In addition to the position, calculating the normalized vector determines the direction facing the building. Before appending the parameters of the new point to the list with waypoints, the distance calculated with the overlap determines the point creation. If there is a waypoint already created in the surrounding, the particular one remains uncreated, to keep the selected overlap.


(a) Neighbouring points of the red voxel


(b) An occupied neighbourly voxel found



(c) The coordinates of the voxels



(d) Creating a new point considering the coordinates and distance

Figure 3.6: Creating a waypoint.

The collection of neighborly positions is limited to six neighbors at each axis as shown in Figure 3.7, meaning the neighbors at the diagonals remain unvisited. This implementation leads to the problem of finding the waypoints denoting the edges of the building model.

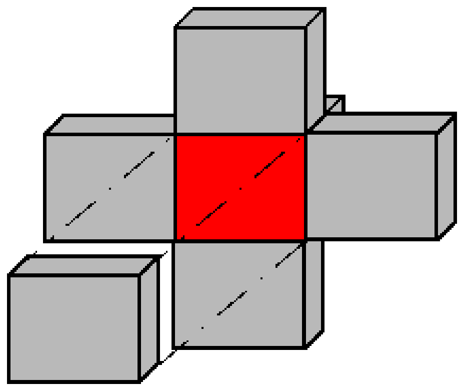


Figure 3.7: Visited neighborly position of the red node.

One way to overcome this problem is presented in the following steps. Each neighbor of the current point positioned on the surface is inspected to indicate placement on a building edge. If at least one other neighbor of this neighbor is an empty voxel, besides the current one, the Euler angle for the corresponding axis is calculated. The dot product of the Euler angle matrix and scalar product of the estimated rotation and distance as a vector is added to the current point to create a new waypoint. The rotation angle increases by 15 degrees after each iteration, which results in a circle around the neighbor point, illustrated in Figure 3.8. Among the position of each point in the rotation, the normalized vector is calculated. The normalized vector determines the direction of the node to the building. The resulting tuple of both parameters is appended to the list with waypoints if adding this point does not violate the fixed percentage of overlap.

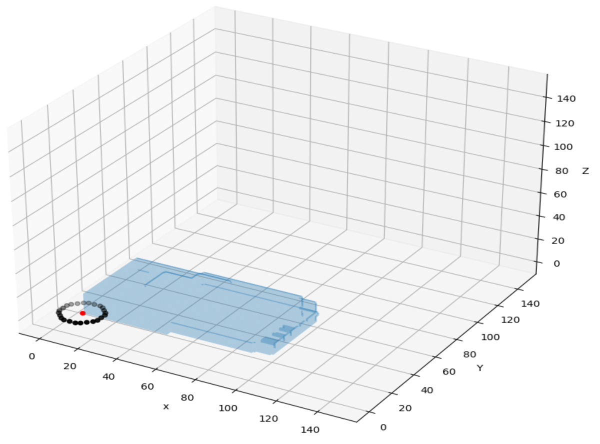


Figure 3.8: Rotation circle over z axis.

After iterating over the whole grid, the list with tuples needs some adjustment in order to filter the irrelevant points. All created points on the surface or inside the building are filtered out, as well as the point outside the bounding box. The waypoints left after the refinement are the position visited with the TSP.

3.4.2 Mathematical background

Waypoints creation uses three main mathematical paradigms - obtaining an optimal rotation to relate two sets of vectors, Euler angle, and calculating the normalized vector. First, to find the optimal rotation for aligning two sets of vectors, the following equation is used:

$$E = \frac{1}{2} * \sum_{i=1}^{n} w_n (Ux_n - y_n)^2$$

where x_n and y_n $(n = 1, \ldots, N)$ are two given vectors sets and w_n is the weights corresponding to each vector. The equation finds an orthogonal matrix *U* which minimizes the function *E*. The usage of the Kabsch algorithm aims to estimate the optimal rotation matrix for aligning two sets of vectors (KABSCH, 1978). A Rotation matrix is a transformation matrix that rotates the coordinate system through a counterclockwise angle θ around respective axes. In 3D space, the basic rotation matrix is rotation over one of the axis and has the following form:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Kabsch algorithm takes two sets of vectors. Both vector sets are translated in order for their centroid to correspond with the origin of the coordinate system by subtracting from the point coordinates of the respective centroid. The second step is to compute the covariance matrix, which leads to the optimal rotation matrix (ISBUDEEN et al., 2011).

The Euler angles can describe any 3D rotation as a sequence of three rotations with respect to a fixed axis. Written in terms of rotation matrices D, C, and A, a general rotation A has the form $A = BCD$. The three angles giving the rotations are the Euler angles. The most common definition for a rotation given by Euler angles (ϕ, θ, and ψ) is shown in Figure 3.9. It starts with rotation by an angle ϕ over the z-axis using D. It follows with

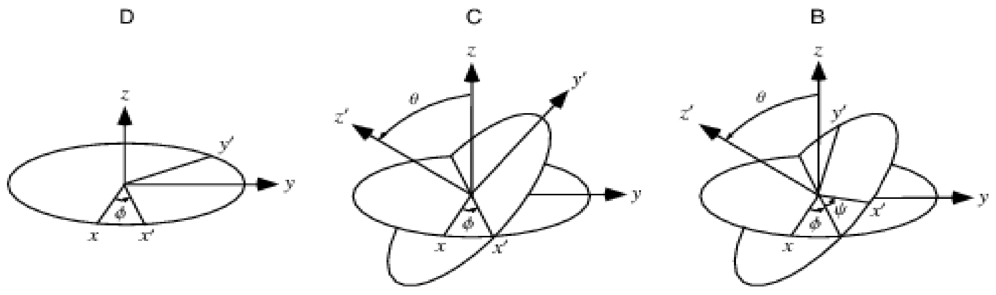


Figure 3.9: Euler angles (Wolfram MathWorld, 2021).

rotation by an angle θ over the former x-axis (now x') using C, and the third rotation by an angle ψ is over the former z-axis (now z') using B (WOLFRAMMATHWORLD, 2021a). For the thesis, multiple elementary rotations along a given axis are calculated with a sequence of angles with the help of the SciPy library (SCIPY.ORG, 2021).

In the process of waypoints creation, a point is given a position in coordinates and direction facing the building, represented as the normalized vector between two points (WOLFRAMMATHWORLD, 2021b). In mathematics, a vector is a geometrical object with direction and magnitude, also known as the norm of the vector. A vector from initial point A to point B, also denoted by $\overrightarrow{AB}$, where B is called the "tail" of the vector, and A is the "head". The norm of any vector $\overrightarrow{x}$ describes the length of the object and is calculated by the formula:

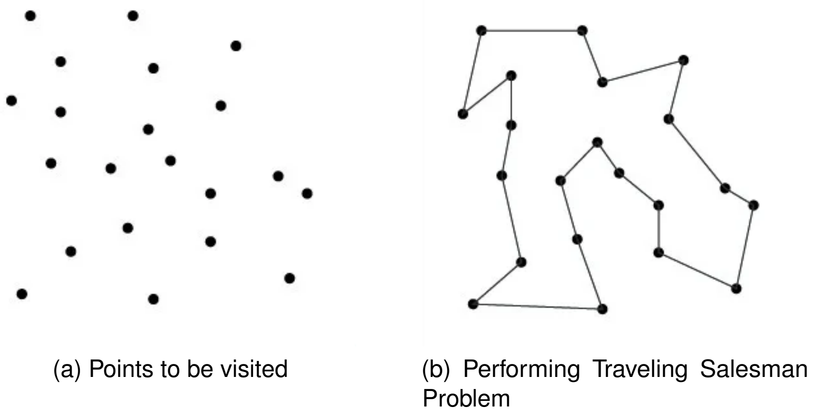
$$\|\overrightarrow{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

Normalization of a vector is the process of determining the unit vector $\hat{X}$. A unit vector, which is a vector with norm equal to 1 and the same direction with the given vector x, is defined by: $\hat{X} = \frac{\overrightarrow{x}}{\|\overrightarrow{x}\|}$. In this thesis, the normalized vector calculation provides the direction towards the building from a given point.

3.5 Path planning

This thesis proposes path planning using the Traveling Salesman Problem for calculating the path based on the waypoints determined in the last section.

The Traveling Salesman Problem is a problem in combinational optimization. It describes the search for the most efficient route that connects all cities from given a set for a salesman considering the cost of travel or the distance in-between. The salesman travels to all locations once and returns to the starting location. The idea is to minimize the distance traveled by selecting the following city for the route demanding the least distance or cost. This problem is an NP-hard problem, meaning there is no polynomial-time algorithm that is known to efficiently solve every TSP (MA, 2020). TSP can be represented as a graph, where cities are the vertices and the connectivity between the vertices, can be called edges, as shown in Figure 3.10. A path consists of a list of vertices connected via edges.


(a) Points to be visited

(b) Performing Traveling Salesman Problem

Figure 3.10: Traveling Salesman Problem (Ma, 2020).

In the case of this thesis, three main approaches regarding TSP are compared. The A* algorithm is used to find an existing path between every two nodes for all three strategies.

The first two approaches use the nearest neighbor method, where a waypoint is connected to the next closest unvisited point. A TSP with n points to be visited has the complexity $O(n^2)$ in the worse case. The function takes the positions of waypoints and a defined start point as input. The paths between the starting point and each other waypoint are calculated using the A* algorithm. The A* algorithm in 3D implemented for the thesis takes a starting node and calculates the shortest path to goal one if that path exists. The path is defined by the exact movement cost to reach the target G and the non-overestimated cost required to extend the path to the goal. The selected path minimizes the formula $F = G + H$. The TSP with nearest neighbor method compares the length of all routes calculated with the A* and returns the shortest path. The goal node of the shortest path becomes the next point. The difference between the two approaches is that the first one divides the waypoints into clusters and performs the TSP between all elements in two neighborly clusters. The second one takes a starting point and sorts the rest of the waypoints by distance from the closest to farthest. Then it sets the closest n-points to the initial point and performs the TSP on them. The last strategy uses TSP with the cheapest insertion. Instead of connecting to the next nearest node, the algorithm places the next one in the position resulting in the cheapest possible costs. The TSP approach using the cheapest insertion algorithm has the complexity to $O(n^2 log2(n))$ in the worse case.

3.5.1 Traveling Salesman Problem with clusters

The waypoints created are separated into n groups of equal variance using the K-means clustering method. The K-means algorithm used to produce clusters divides a set of N samples X into K disjoint clusters C, each described by the mean μ_j of the samples in the cluster (SCIKIT-LEARN.ORG, 2020). The algorithm requires a defined amount of clusters as input besides the data to be divided. Figure 3.11 illustrates division into 10 clusters using K-means method.

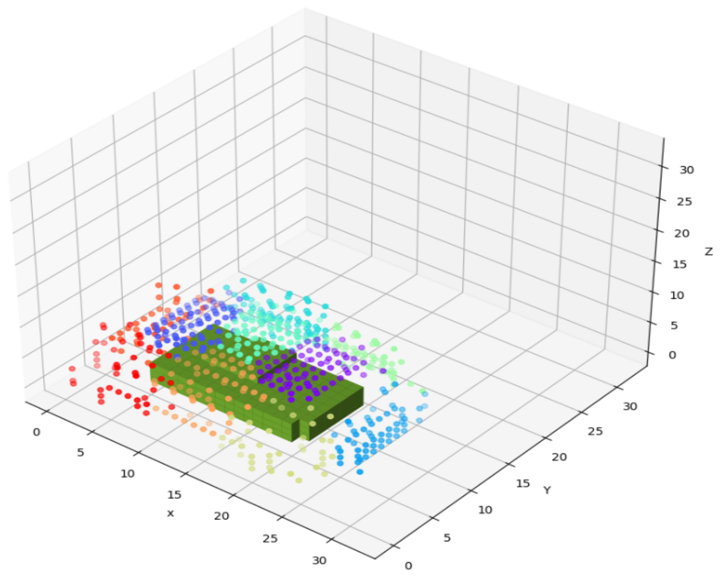
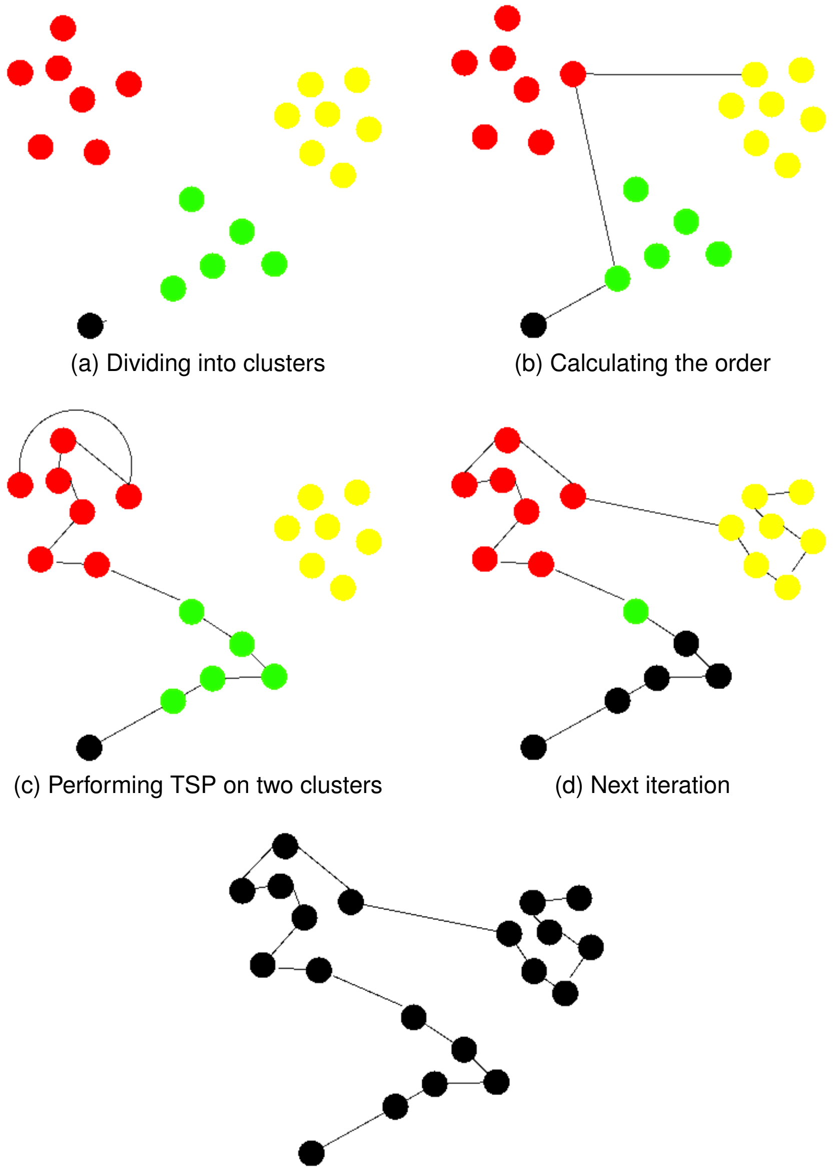


Figure 3.11: Division into 10 clusters.

Once divided into different clusters (Figure 3.12a), extracting an element from each cluster and performing the TSP with a given starting point will return the rough cluster order concerning the starting point (Figure 3.12b). After defining the order, the path between all points within every two clusters is calculated (Figure 3.12c). The algorithm calculates the next nearest neighbor beginning with the start point and the costs needed to reach it with the A* algorithm. Each element has a cluster-id, the path to the initial point, and the path length. Deciding on the next node depends on the shortest length. The TSP with the nearest neighbor is calculated for each element in the two clusters related to the current node. After each iteration, only the elements of the first cluster are to be appended to the final path and the last first cluster element becomes the starting position for the next iteration as shown in Figure 3.12d. The algorithm terminates the iterations when TSP is performed on the last two clusters, whose elements are added to the final path (Figure 3.12e).


(a) Dividing into clusters

(b) Calculating the order

(c) Performing TSP on two clusters

(d) Next iteration

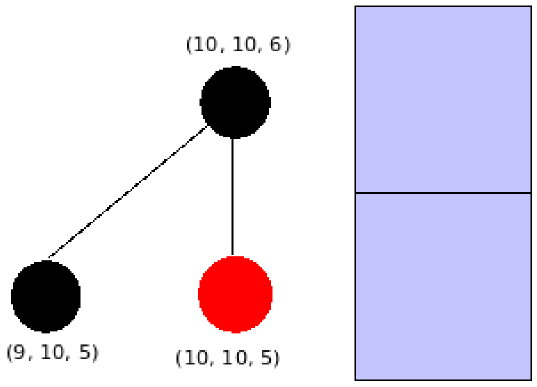
(e) Final path

Figure 3.12: Traveling Salesman Problem with clusters.

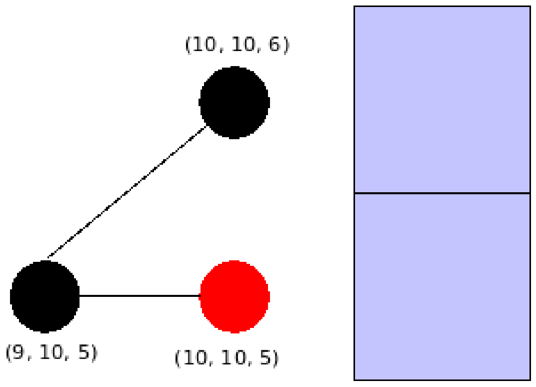
3.5.2 Traveling Salesman Problem on closest points

The TSP with closer points also uses the nearest neighbor method to find the shortest path, similarly to the TSP with clusters. However, the second approach differentiates from the first one in how the waypoints are incorporated. This method takes an initial point and calculates the Euclidean distance between the initial and every other waypoint. Thus from the very beginning, there is a rough overview of how far each point is from the initial. The method sorts the waypoints list by distance from the initial one in ascending order. The closest n-points to the start point are selected, and the TSP is performed on them to find the nearest neighbor.

The second main difference between the TSP with clusters and the TSP on closest points is the decision on the following node. The second approach selects the point successor by comparing the resulting costs from each path rather than the path length. The path cost is calculated from the sum between each movement in the voxel grid. Since moving in a voxel grid, every movement costs one unit, except the diagonal one, which costs $\sqrt{2}$. A priority to one of the axes is implemented to allow a possible influence on the moving direction. In the case of the thesis, the z-axis can be prioritized. The decision on the z-axis is a consequence of the idea to influence the drone to move vertically to the building surface. The priority suggests that the moving cost in this direction will be cheaper than the other axis. So, by equal distanced points, the algorithm will always prioritize the one moving along the z-axis. That can result in path difference, but not necessarily. If non of the axes is prioritized, it takes the first cheapest point as a successor. This approach helps to create a more consistent path. A drawback of the discrete grid is the possible production of a more random route because each movement to the next point is equally expensive. According to the path costs, the following point becomes the one with the cheapest path costs. Figure 3.13 visualize the difference in route with and without z-priority. The red node is the starting position with (10, 10, 5) coordinates. In the first example, the priority along the z-axis forces the successor point to be the one with (10, 10, 6) coordinates. In the second example, the choice of the waypoint along the x-axis is arbitrary.



(a) Using z-priority



(b) Without z-priority

Figure 3.13: Path difference with versus without z-priority.

3.5.3 Traveling Salesman Problem with cheapest insertion

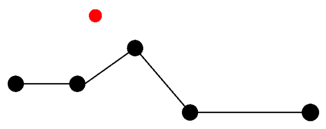
The last approach is a Traveling salesman problem with the cheapest insertion. The cheapest insertion begins with two points, and it finds an unvisited waypoint and connects it between two points, where it produces the lowest cost and repeats until there are no more insertions left.

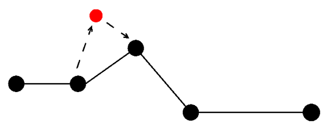
The algorithm starts with an initialization of a list and a dictionary. The created list stores the final version of the path. After calculating the costs between any two points using the A* algorithm, the dictionary saves the two points and the costs for faster access in the following calculations. In each iteration, except for the first one, the points and the path costs are stored bidirectionally. In the first iteration, the algorithm calculates the costs to each point from the initial one. It saves the result in the dictionary one-sided because the beginning element is always the starting position. The resulting cheapest costs are added to the total path costs.

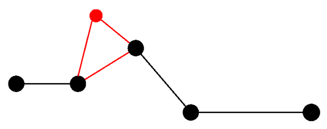
In each other iteration, the TSP approach calculates the costs from the point, located at all possible positions in the final path except for the first one, to each other point in the waypoints list. A waypoint is connected to the visited points if the insertion of this particular point at a fixed location produces minimum path costs. Depending on the inserting position, two different formulas define the path cost. A point is placed between two points or at the end of the final route. In order to compute the costs of inserting a waypoint between two points, the path costs between any two points are subtracted from the total costs. The result is added to the costs between the new point and the first one and between the new one and the second point, shown as a formula:

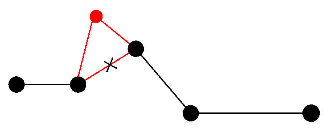
$$costs = total - costs(point_{i-1}, point_i) + costs(point_{i-1}, new) + costs(new, point_i)$$

where *i* is the inserting position of the new point, and the costs are the path cost function between the two elements. As every movement costs one unit, a prioritization of the z-axis as described in Section 3.5.2 is possible. In graph representation, the idea is that the formula adds the new vertex between the two vertices connected with two new edges and removes the one connecting the previous two nodes, visualized in Figure 3.14.


(a) Inserting the red node in the graph


(b) Finding the position with the cheapest costs


(c) Adding the new edges


(d) Remove the previous edge

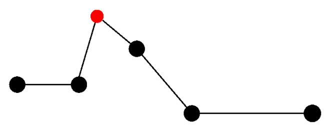
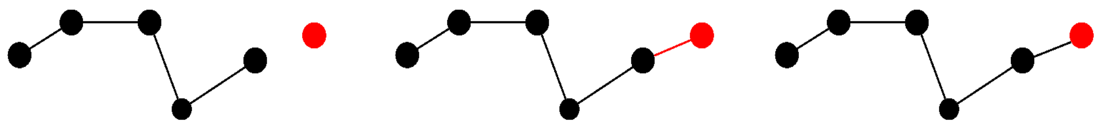

(e) Point inserted

Figure 3.14: TSP with the cheapest insertion represented as a graph.

Inserting an element in the final path at the last position is even more convenient. The costs to insert the new point are added to the total costs needed for the created route:

$$costs = total + costs(point_{i-1}, new)$$

where *i* is the inserting position of the new point and $point_{i-1}$ is the last point coordinates in the path. As a directed graph, the new vertex is placed at the end of the list connected with one new edge for which the costs are calculated. Figure 3.15 represents the point insertion at the end of the route.



(a) Inserting the red node at the last position (b) Adding the path costs to the total costs (c) Point inserted

Figure 3.15: Inserting a point at the end of the path, TSP with cheapest insertion.

According to the calculated paths using the two formulas, the algorithm inserts the point at the position where it provides the lowest costs and removes it from the list of unvisited waypoints. The iterations terminate when there are no more points left to be connected.

3.6 Analysis

Evaluating the performance of the path planning method presented in this thesis occurs in two steps. First, the measurement for efficiency of the created waypoints is coverage and overlap. Important questions associated with evaluation are:

- how much a set of points covers from the building surface.
- how much two nodes overlap.

Second, the three presented methods regarding the Traveling Salesman Problem are analyzed by comparing the length, computational time, and path tendencies. However, the last approach is optimized with parallel processing, which will make the computation time comparison not comparable.

The given overlap is integrated into the voxel size, demonstrated in Section 3.3. The coverage analysis of the set of existing waypoints is performed by marking all points on the surface of the building seen from all waypoints created. The marking is proceeded by adding a camera pixel frame to consider the visibility of the initial point toward the surface in voxel space. A visible from an initial point node is transformed from 3D to 2D. The transformation from a 3D world to a 2D image is a projection process that implies a dimension loss. The coordinates of a 3D point are mapped to a 2D image coordinates of the node's projection onto the image plane.

The pinhole camera model describes the mathematical relationship of the projection of points in the 3D world onto the image plane. Assuming the center of projection is the origin of a Euclidean coordinate system and the plane $Z = f$, also called the image plane, as illustrated in Figure 3.16 (HARTLEY & ZISSERMAN, 2003). A 3D point with coordinates $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})^{\mathsf{T}}$ is mapped to the point on the plane $(\frac{\mathbf{fX}}{\mathbf{Z}}, \frac{\mathbf{fY}}{\mathbf{Z}}, \mathbf{f})^{\mathsf{T}}$ using the triangles shown in Figure 3.16. The central projection transforming from the 3D world to the 2D image coordinates is

$$(\mathbf{X}, \mathbf{Y}, \mathbf{Z})^{\mathsf{T}} \rightarrow (\frac{\mathbf{fX}}{\mathbf{Z}}, \frac{\mathbf{fY}}{\mathbf{Z}})^{\mathsf{T}}$$

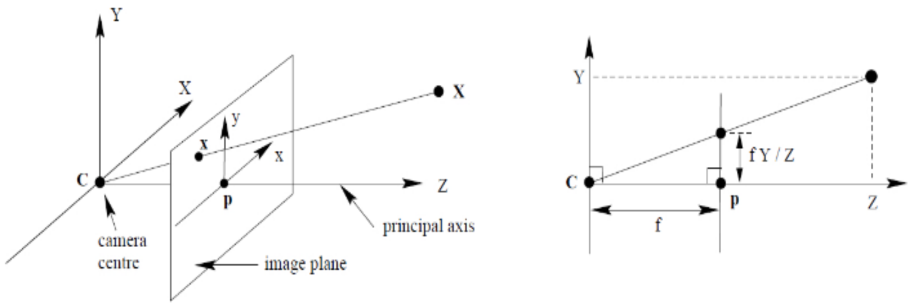


Figure 3.16: Pinhole camera model (Hartley, 2003).

The transformation from the 3D world to the 2D image is computed by using the formula $x = PX$, where P is the camera matrix, the X is a 3D world point, and x is the 2D image point (SIMEK, 2013). For homogeneous coordinates, meaning the camera coordinates coincide with the 3D world coordinates, the camera matrix is a 3x4 matrix, where f represents the focal length:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

The resulting equation assumes that the camera and the world share the same coordinate system. In general, this is not the case, and aligning the camera coordinates with the 3D world coordinates is executed by using an extrinsic camera matrix. The matrix has two components, a rotation matrix R and a translation vector t.

$$\left[\begin{array}{c|c} R & t \end{array}\right] = \left[\begin{array}{ccc|c} r_{1,1} & r_{2,1} & r_{3,1} & t_1 \\ r_{1,2} & r_{2,2} & r_{3,2} & t_2 \\ r_{1,3} & r_{2,3} & r_{3,3} & t_3 \end{array}\right]$$

The extrinsic camera matrix describes the transformation of the point from world coordinates to the camera coordinate system. The vector t can be interpreted as the position of the world origin in camera coordinates, and the columns of R represent the directions of the world-axes in camera coordinates. The extrinsic camera matrix can be presented as $X_c = R(X_w - C)$, where R is the rotation matrix, X_w a point in the world coordinate system, and C is the camera center coordinate in the world coordinate frame. The resulting point X_c represents a point in homogeneous coordinates (SIMEK, 2012).The extrinsic camera matrix combined with the intrinsic camera matrix transforms homogeneous 3D world coordinates to homogeneous 2D image coordinates, using the general equation: $P = K[R - RC] = KR[I| - C]$, where K represents the intrinsic camera matrix, R the 3D rotation, I is the identity matrix and C - the 3D translation.

The path planning analysis compares the three approaches considering the Traveling Salesman Problem. Investigation of the strategies statistically includes examinating the computational time needed for the path plan. The procedures of estimating the efficiency of the method follow the suggestions of comparing the planned flight trajectories in the 3D model. The analysis also addresses the length of the produced paths by evaluating the path cost. A simple cost function proposed by DEBUS and RODEHORST (2021) is the sum of the Euclidean distance between two successive waypoints and the number of required images in the path $L = ||c_i - c_{i-1}|| + m$. Since all three approaches provide information about the total costs of the generated path, evaluating the strategies adopting suggested function is possible with the modification: $L = costs + len(waypoints)$, as each waypoint indicates drone scanning coordinates.

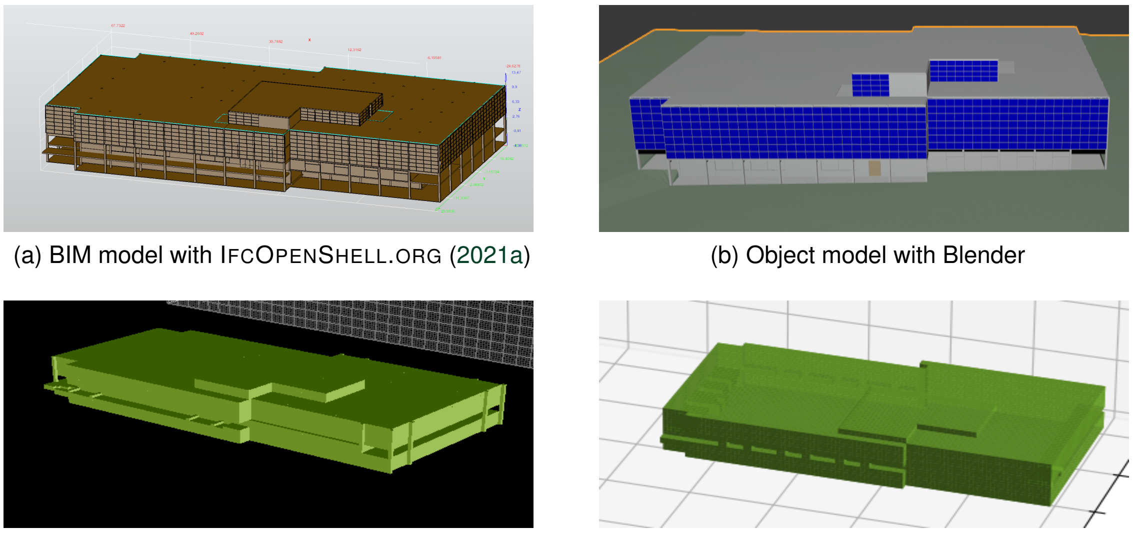
Chapter 4

Implementation

4.1 Data and Software

4.1.1 Data

The implemented method presents a path planning for IFC model input. The tool should calculate the waypoints and compute a path for any BIM model in IFC format. Two main models are used to test and complete the thesis task – one simple and one more complex. The simple one, provided by IfcOpenShell.org (2021b) represents a house with four walls and a tiled roof. The more complex model is a BIM representation of the TUM Mensa, mainly used to test the method. Figure 4.1 illustrates the transformation stages of the mensa building model in the IFC format to octree representation. The model described in IFC file format is visualized using IfcOpenShell.org (2021a) library and combined with PythonOCC, convenient library for 3D visualizations (Jansohn, 2010). The second figure displayed using Blender software represents the model consistency after the conversion to the object file required for voxelization. The voxelized model resulting from utilizing the external software binvox (Min, 2004 - 2019a) is captured in the third figure, opened with viewvox. Viewvox is software that visualizes a 3D voxel file produced by binvox and shows it in a window and gives a picture of the whole model (Min, 2004 - 2019b). The last figure is the octree representation, exploited for path planning purposes.


(a) BIM model with IfcOpenShell.org (2021a)

(b) Object model with Blender

(c) Model voxelized with binvox adn visuallized with viewvox

(d) Octree representation

Figure 4.1: Mensa stages.

The thesis presents a method for path planning for drone capturing of existing buildings. An example drone for which the method should be applicable is a drone with an average speed of 20 m/s. Since the distance between two waypoints considering a fixed overlap is determined depending on the camera Field of View, the parameter for calculating the visibility length presented in Section 3.4.1 would have to be set to $87° \times 58°$ for this exemplary drone.

4.1.2 Software

The external software binvox is applied to convert the input data to a voxelized model. Binvox is software for an automated way to create a voxel-based representation, developed by Patrick Min. The software reads a 3D model file, converts it into a binary 3D voxel grid, and writes the resulting voxel file (MIN, 2004 - 2019a). Binvox allows selecting the grid size, extracting only the geometry voxels, and forcing an input model bounding box.

Besides the voxelization process, all other steps of the method are implemented using the programming language Python. Python is an interpreted, object-oriented, high-level programming language with dynamic semantics that offers a good structure with simple syntax and support for developing large applications. It incorporates complex data types such as flexible arrays and dictionaries. Python allows splitting a program into modules, which are reusable in other programs. Programs written in Python are powerful and readable (PYTHON.ORG, 2021). The programming language has interfaces to many libraries, providing an efficient tool for faster implementation of many different problems.

4.2 Preprocessing

The method takes BIM model in IFC format as an input, which is transformed to voxel-based data. Before voxelization using binvox, two steps need to be completed. First, the bounding box is calculated to create a realistic voxelized representation. IfcOpenShell allows parsing the IFC files and receiving an actual geometric description of each element. Using Bnd from PythonOCC library calculates each dimension's minima and maxima of a bounding box (JANSOHN, 2010). Rounding up to the next power of two and adjusting the bounding box contribute to the voxelization. The second step before using binvox is the transformation of the input from IFC format to an object format as the software does not support IFC files. IfcConvert is an application, which allows to transform an IFC geometry into several file formats such as OBJ, XML, and SVG (IFCOPENSHELL.ORG, 2021a).

Once created, the binvox data can be loaded to a python data structure using the open code source binvox-rw-py, developed by Daniel Maturana (GITHUB, 2021). This code allows reading the binary binvox format as a 3D array. An octree is created from the resulting array by recursively slicing it into eight equal cubes. The Boolean value from the 3D array visualizes the status of the cubes - empty or occupied. If all values in a sliced cube are False, the particular one reminds undivided in the next iteration.

The octree is a base to generate a grid with a different voxel precision. The function creates an array of zeros using the library NUMPY.ORG (2021). It starts from coordinates (0,0,0) and a given depth level and recursively finds the lowest level in the octree. Once found, it sets the corresponding value 1, standing for an occupied voxel, and 0 - an empty one, resulting in a binary array.

4.3 Simplification of the model

The simplification of the model aims to filter out the irrelevant points. The filtering is achieved by filling the inside unaccessible voxels, extending the grid space, and dilating the building.

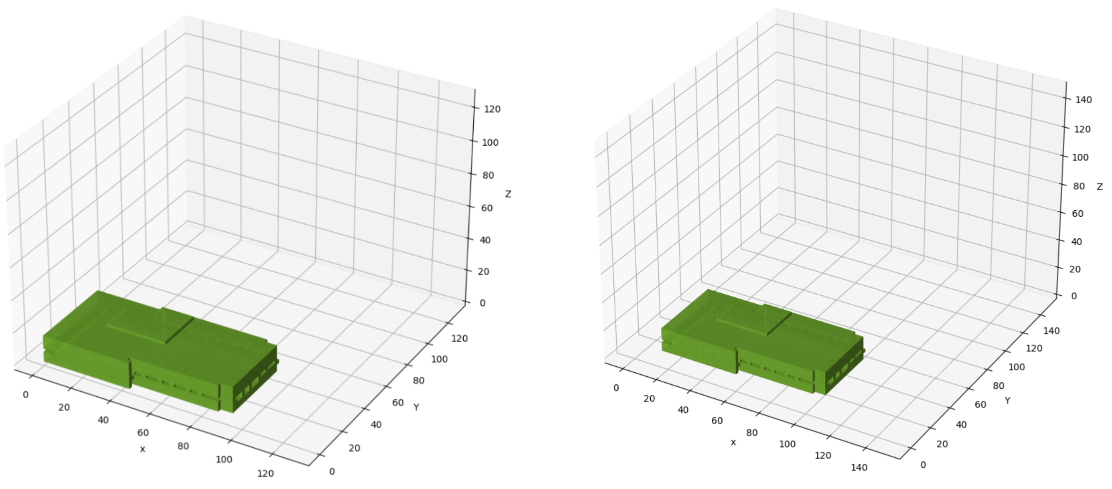
The flood fill algorithm filters the empty inside voxels, which are unessential for the thesis. The algorithm can be recursive or iterative with a stack. The recursive function can cause an overflow of the recursion stack and termination of the program when filling larger areas. Therefore, the algorithm implemented for the thesis uses a while loop that terminates when the stack is empty. Instead of starting another recursion, it pushes new positions to the stack (VANDEVENNE, 2018).

Algorithm 4.1: Flood fill algorithm: Pseudocode (Vandevenne, 2018).

```
Flood Fill(start node, array):
    set an empty stack
    set flood_array to array of ones
    set an empty visited
    add start node to stack.
    while the stack is not empty:
        set temp equal to the first element of the stack
        remove the first element from the stack
        create a list with temp_neighbours
        if array at position temp is 1:
            set flood_array to 0
            add temp to visited
        if array at position temp_neighbour is 1:
            set flood_array to 0
        if temp_neighbour not in visited:
            add the neighbour to stack
    return the reversed values of flood_array
```

The first step is creating a grid of ones with the same shape as the one generated from the octree representation. The flood fill algorithm starts in an empty voxel grid and keeps the area connected to it with the initial value. The aim is to differentiate the empty available voxels outside the building from the empty irrelevant voxels inside the building. Therefore, each voxel representing a building surface takes the value of zero and works as a barrier between both types of empty voxels. Since the inner empty voxels are not connected to the starting position outside, they are also assigned a value of zero. The flood fill algorithm returns the reversed values, meaning that the accessible voxels outside have the value zero, and everything inside is one. As a consequence of the reversed filling, the algorithm also considers a situation with a built environment consisting of more than one building. Algorithm 4.1 represents the pseudocode of the flood fill algorithm implemented for the thesis, inspired by the computer graphics approach of VANDEVENNE (2018). One difference is the binary filling instead of the color one. Furthermore, the empty area around the building is flood-filled, not inside.

The next simplification step, the grid extension is proceeded by padding the resulting flood-filled array. Using np.pad from library NUMPY.ORG, 2021, a constant amount of zeros are added to both edges of each axis, except for the z-axis. All zeros added to the z-axis are located on the positive orientation of the axis to prevent inaccurate path planning calculation. Figure 4.2 presents the difference in the grid before and after the extension with ten units on both edges of each axis. The extended version avoids nodes creation outside of the boundary box.



(a) Before extension

(b) After extension

Figure 4.2: Before versus after grid extension with 10 units in each edge of the axis.

The building dilatation prevents the waypoint creation close to the building, which is significant for collision avoidance. The function sets nearby voxels of an occupied one to 1, meaning that for any voxel located on the surface of the building, the neighborly empty voxels will also be filled.

4.4 Waypoints creation

The thesis presents an automatic waypoint creation at a certain distance from the building. The function collects the neighbors' positions of a particular point to locate occupied voxels. Once discovered, a rotation matrix is determined, which optimally aligns the vectors between the current point and the occupied neighbor. Utilizing the class scipy.spatial.transform.Rotation from library SCIPY.ORG (2021) in this function provides an interface to represent rotations with rotation matrices or Euler angles. The class also supports many rotation operations for quick results. The align-vectors-function computes the optimal calculation of the rotation matrix that transforms the current point to the occupied neighbor. The scalar product of the rotation matrix and the distance vector added to the neighbor coordinates define the location of the newly created point. The calculated distance between the nodes relative to the chosen overlap determines whether the point should be created.

In order to create a waypoint from a neighbor located on the edge of the building, the algorithm performs a complete rotation over a corresponding axis. Similar to the rotation matrix calculation, the same class is incorporated to initialize a rotation from Euler angles. The function from-euler returns an object containing the rotation matrix represented by the sequence of rotations around given axes with given angles. In the case of the thesis, the function performs multiple elementary rotations in one object. The rotation angle increases by 15 degrees after each iteration and completes a circle around the edge node (SciPy.org, 2021).

A significant part of the waypoint creation is filtering out the irrelevant or redundant points. As a consequence of the complete rotations over an axis with Euler angles, some nodes can be created in the direct vicinity to each other or in an inaccessible location. The defined distance between the nodes keeping the percentage of overlap reduces those in immediate surroundings. If a point exists in the defined surroundings, a new one is directly not created. The waypoints created on occupied positions also need to be filtered out. The last step of the function is to reduce the waypoint positions, which are assigned to 1 in the voxel grid, indicating the flood-filled built environment located on those voxels. Furthermore, the algorithm filters the waypoints created outside the grid by creating a boundary as big as the grid shape.

4.5 Path planning

The waypoints created in the previous step present the coordinates for the route planning. Each waypoint is transformed into an element of a class Node, which attributes and functions are demonstrated in the UML diagram in Figure 4.3. The x, y, and z values represent the position coordinates of the object. The parent attribute is essential for the path reconstruction produced by the A* algorithm because each element from the shortest path is assigned to the parent attribute of the next node. The H, G, F values are used in the formula F = G + H to select the shortest path. The class allows comparing two Node objects by their values and calling the object position in the voxel grid. The h-score function returns the non-overestimated cost from the current node to the goal node required for the A* algorithm. Since operating in a discrete voxel grid, the heuristic costs are equivalent to the Euclidean distance between the current and the end node. The last function investigates the surrounding of the Node object and outputs a list with each empty neighborly voxel in each direction of the grid, including the diagonals. The class Node is used in each different TSP approach, modified to meet the requirements of the strategy.

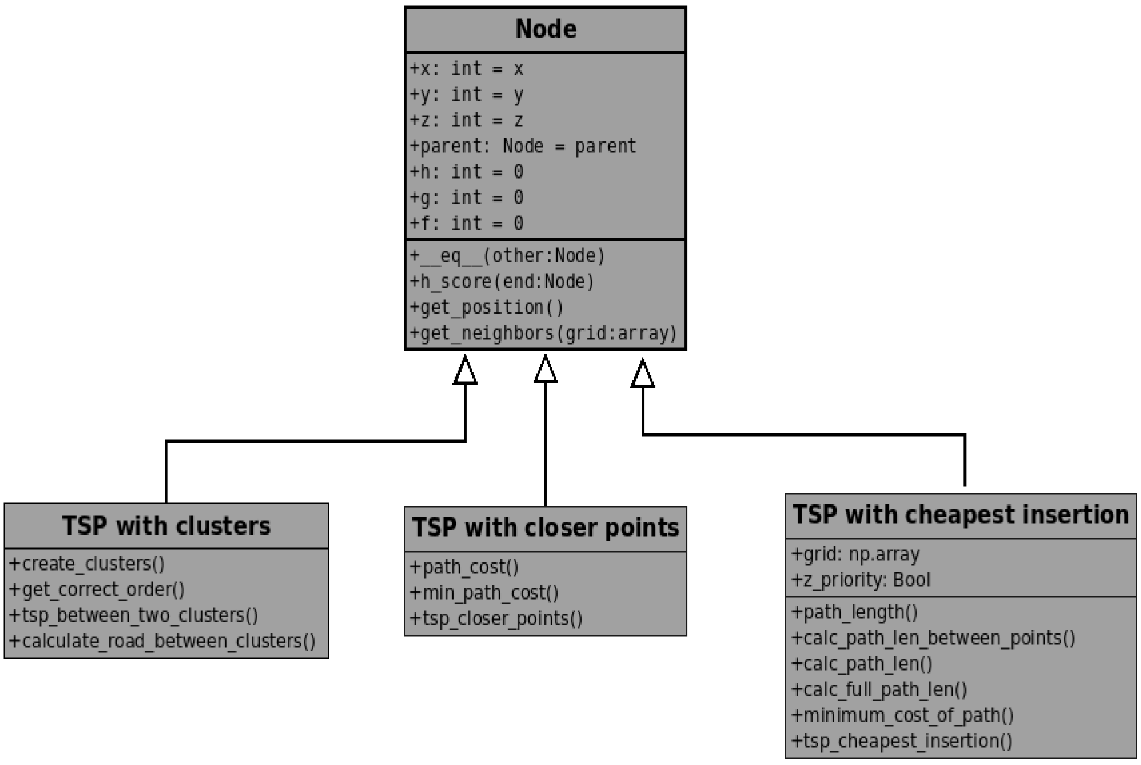


Figure 4.3: Node class: UML.

4.5.1 Traveling Salesman Problem with clusters

The Traveling Salesman Problem with clusters divides the waypoints created into n clusters using KMeans from sklearn.cluster library. The number of clusters must be given by initialization. The library computes the cluster centers and predicts the index for each sample (SCIKIT-LEARN.ORG, 2020). A random element of divided clusters is sorted accordingly to a starting position using TSP to get a rough insight of how far each cluster is from the start node. Subsequently, the algorithm calculates the path between all points within every two clusters and the additional costs with the A* algorithm. In the beginning, assigning a binary cluster-id to each element shows affiliation to the corresponding cluster. Each node is represented by the cluster-id, the path to the initial point, and the path length needed to reach the initial point. Following the next nearest neighbor principle, the next element in the route is determined by the shortest length to the initial node. The path costs between nodes in one cluster are much cheaper to prevent jumping between clusters. In each iteration, only the elements of the first cluster are to be appended to the final path. The last element from the first cluster becomes the starting point in the next iteration until all clusters are appended. The closing iteration inserts the nodes from the two last clusters to the final path.

4.5.2 Traveling Salesman Problem on closest points

As mentioned in the Chapter 3, the Traveling Salesman Problem on closest points differentiates in the waypoints incorporation in the pathfinding. TSP with the nearest neighbor approach is performed on the closest n points to the initial node. The sorting of points occurs by calculating the Euclidean distance between the initial and all other waypoints. The sorting criteria result from the h-score function from the Node class. In each iteration of the TSP, the next point is the one with the least expensive costs according to the cost function. The cost function takes a path and calculates how much it will cost. It sums up each movement by adding 1 or $\sqrt{2}$. Depending on the z-priority parameter, the costs for moving along the z-axis can be prioritized or equal to 1. The costs with z-priority are equal to $\frac{1}{\sqrt{2}}$. All paths and produced costs are stored in a variable and sorted to retrieve the node for the next iteration.

4.5.3 Traveling Salesman Problem with the cheapest insertion

The Traveling Salesman Problem with the cheapest insertion modifies the Node class by initializing two new attributes: the voxel grid and z-priority become attributes to be accessed faster. The algorithm begins by creating a list and a dictionary. The list will store the path, and the dictionary will store the cost calculations between two nodes. The dictionaries in Python store the data values in ordered key-value pairs. The stored data is also changeable and not duplicated, which indicates that the keys are unique. The dictionary saves two nodes as a key and the costs as a value. Since element insertion is possible on each position *i* except for position 0, the nodes in the key are stored bidirectionally, besides for the first iteration as the first element is always the starting position. The insertion can succeed between or at the end of the list using the presented in the methodology formulas. The different cost function separates the path cost calculation. A challenging problem that arises in this domain is algorithm computational time. Calculating the costs between two nodes in each direction causes a longer time since it computes the results using the A* algorithm. The function dedicated to this calculation is carried out simultaneously to speed up the whole algorithm. Pathos library provides methods for process parallelization. The purpose of pathos is to extend the user's code to parallel and distributed calculations with minimal refactoring (McKerns, 2022). The library consists of parallel versions of the map function, which applies a given function to each item of the given iterable. The pool.map allows function operating in parallel as a distributed service, using standard python and without writing a line of server or parallel batch code (McKerns, 2022).

4.5.4 A* Algorithm

Since the A* algorithm represents a significant part of the TSP implementation, the pseudocode of the A* executed for the thesis is presented in Algorithm 4.2. The algorithm is inspired by Stanton (2020) but modified for the method functionality. The algorithm uses two lists to find the path - a list of open nodes and closed nodes. The list of open nodes stores all points to be processed, prioritizing the one with the lowest F next. The list of closed nodes saves all points already considered from the algorithm. In order to start the algorithm, a start node and a goal node, as well as the grid used are defined. In the beginning, the initial node is added to the list of open nodes. In every next iteration, the list is sorted according to the lowest F-value. The associated node becomes the current node. The neighbors of the current node are appended to a list, using the function from class Node. The G, H, and F values are determined for each neighbor in the list. The current node is assigned to the parent attribute of the neighbor node. The information helps reconstruct the path between the start and the goal node. If the neighbor is not already considered as a current node, it is added to the list of open nodes. The algorithm terminates by reaching the goal node or if the path between the two nodes is not reachable.

Algorithm 4.2: A* algorithm: Pseudocode (Stanton, 2020).

```
A* algorithm(grid, start node, goal node):
    set an empty open list
    set an empty closed list
    add start node to open list.
    while the stack is not empty:
        set current node equal to the first element of the stack
        set current index equal to 0
        iterate over the open list:
            if another item has a lower F-value:
                set current node equal to the item
                set current index equal to item index
        remove current from the open list
        add current to the closed list
        if the current node is equal to the goal node:
            find all elements between the current node and goal node by
                using the parent
            return route
        create a list with neighbors
        if neighbor not in closed list:
            Find G-value, H-value, and F-value of neighbor
            Set current node as a parent of neighbor
        if neighbor not in the open list and not in close list:
            append neighbor to open list
    return route
```

4.6 Coverage Analysis

One part of evaluating the implemented method consists of an analysis of the coverage gained with a set of waypoints. The investigation happens by marking all the points visible for each waypoint. The implementation is divided into five functions, as shown in Figure 4.4. The function collects all occupied voxels on the surface of the building and sets the coordinates in a list. The visibility of each waypoints' element to each point on the surface is validated. An apparent limitation of the process occurs in the visibility of points. An occupied voxel is considered visible if one of the faces is visible from the waypoint. That results in an artificial increase in coverage. If an occupied node is visible from a waypoint, the function saves the 3D point coordinates and 2D image coordinates. The transformation onto 2D image coordinates relies on the pinhole camera model, stated in the Section 3.6. The camera coordinates are aligned with the 3D world coordinates to create a homogeneous world point, using the extrinsic camera matrix. The transformation from 3D to 2D utilizes the formula discussed in the Section 3.6. Adding a pixel frame and focal length aims to filter all 2D image coordinates outside the frame. The coordinates remaining are added to a list.

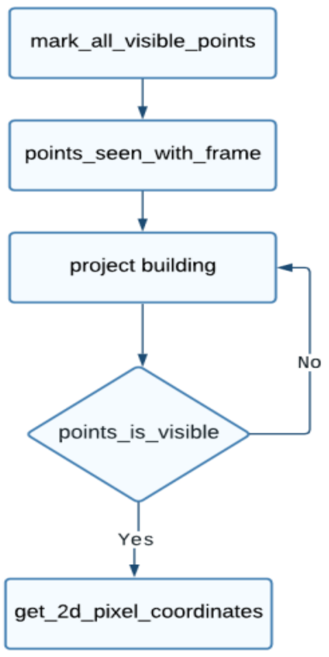
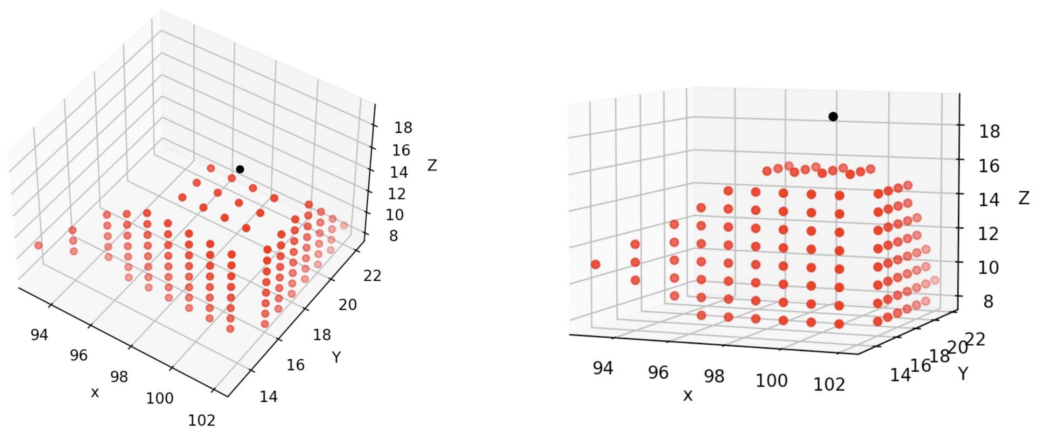


Figure 4.4: Coverage analysis - implementation workflow.

A maximum depth distance is proposed to filter the visible occupied nodes which are yet too far from the camera. In this logic, calculating the Euclidean distance from each waypoint to the coordinates of surface points and comparing it to the maximum depth distance create a visible area from the camera for a more accurate coverage analysis. Figure 4.5 presents the camera view with a maximum depth distance of 20. In the next



(a) Above (b) Side

Figure 4.5: The camera view with maximum depth distance.

step, the filtered points on the surface are marked as seen and removed from the occupied points list. Lastly, the marked points are divided by all surface points to receive the coverage percentage.

Chapter 5

Discussion

In the following section, the results and evaluation of the proposed path planning method are presented and analyzed. The analysis occurs in two main steps: the waypoint analysis and results from the different path planning approaches. For result accuracy, the method is tested with the BIM model of the TU Mensa, and the octree representation has the lowest level of precision to obtain one-meter-sized voxels. The distance between points is determined with the information known for the given camera.

5.1 Waypoints analysis

5.1.1 Results

The waypoints coordinates are created on a fixed distance away from the building and on a calculated distance in-between, ensuring a percentage of overlap. That indicates that the setting of those two parameters, distance and overlap, influences the creation of points. The camera FOV, the given distance from the building, and overlap define the distance between the points. The calculation happens with the help of the formula presented in Section 3.3. An adjustment of the distance from building and overlap results in a different distance between two waypoints knowing the field of view from the given camera. Distinct distances between two points affect the number of created waypoints and the computational time. The defined waypoints, combined with different amounts of maximum depth distance, impact coverage analysis. The reason is that marking the occupied building voxels covered and determining their visibility from each waypoint define the coverage. The following plots describe the impact of the different overlaps and distances away from the building on the waypoints creation, coverage, and processing time. The maximum depth distance is set to accordingly 10 and 20, and the focal length - to 50 mm. The maximum depth distance is not a relevant parameter for defining the distance between waypoints. Therefore, a comparison between the two depth distances is made exclusively for the coverage analysis. The analysis considers n overlap in a range of 10 to 90%. 100% overlap means that both points have the same building projection and the distance between two waypoints is equivalent to zero. 0% overlap creates a distance equal to the visibility length, which is also an irrelevant case, as a certain overlap is needed to reconstruct the building. The distance is set to (6,8,10,12,14). The range also includes the proposed State of the Art distance of 10 m.

Figure 5.1 highlights the impact of overlap and distance from the building on the distance between the points. The distance between the waypoints is linearly related to the distance away from the building. The further the waypoint from the building, the longer the distance between two points. The relation to the overlap is yet inverse. The higher the percentage of overlap, the nearer the distance between two waypoints.

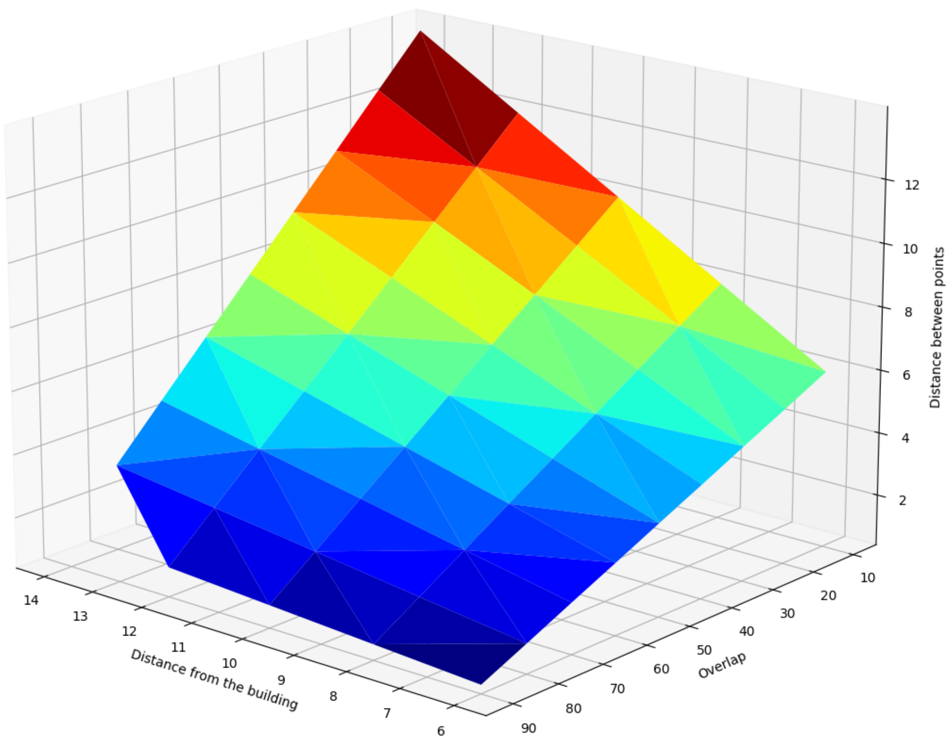
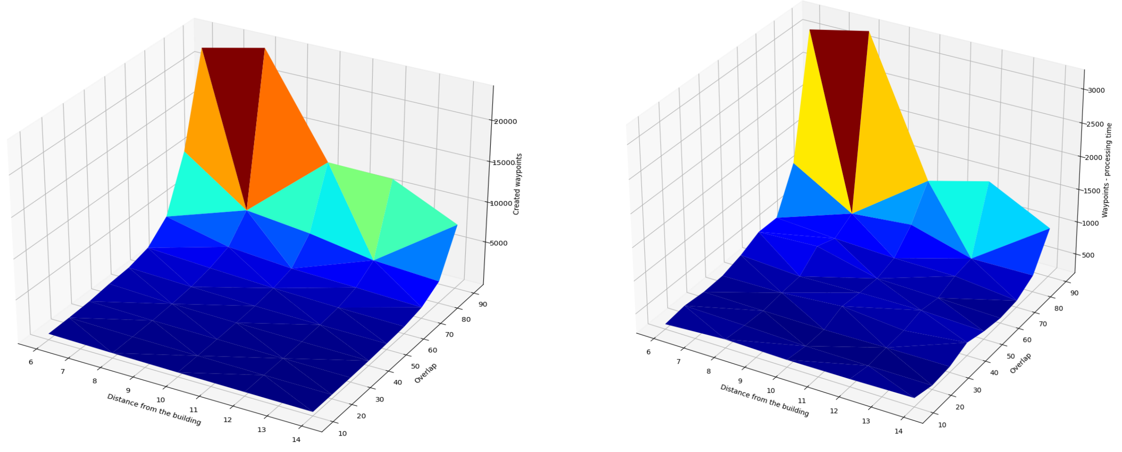


Figure 5.1: Impact of overlap and distance from the building on distance between points.

The waypoints creation and the needed computational time in Figure 5.2 have the same pattern: exponential growth can be observed. The highest number of waypoints produced occurs with a distance decrease from the building while keeping a high percentage of overlap above 70 %. The observed relation is because the distance between two waypoints reduces drastically with a closer distance from the building and a high percentage of overlap.

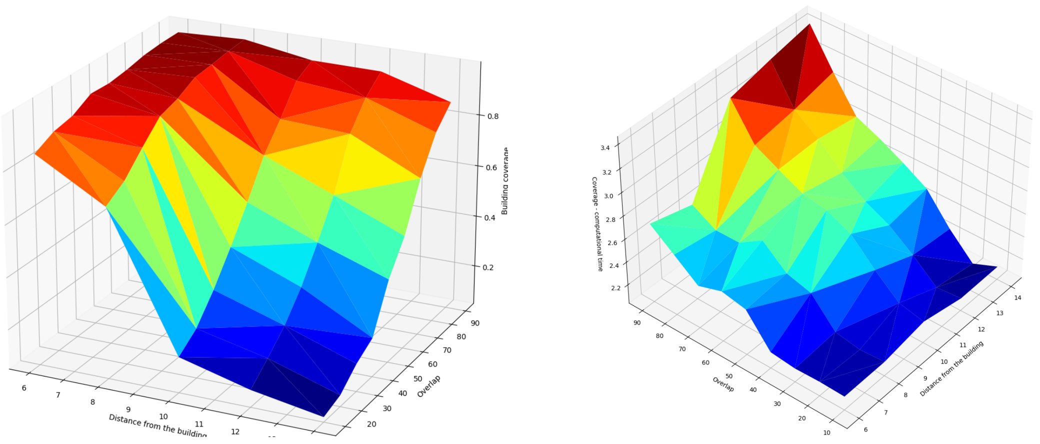


(a) Created waypoints

(b) Computational time

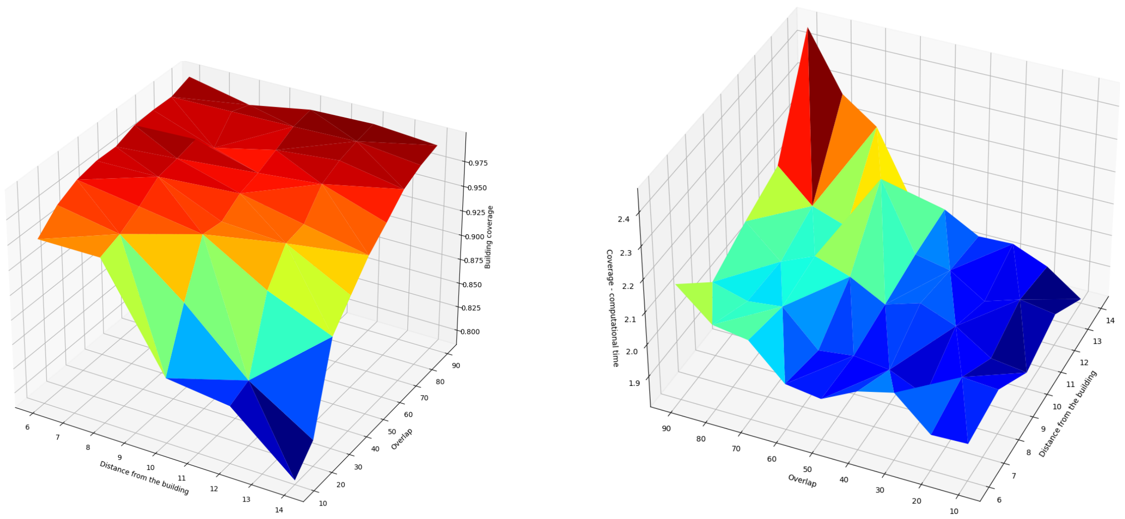
Figure 5.2: Overlap and distance from the building vs. waypoints and computational time.

The following plots present how different distances from the building and overlaps influence the building coverage and the computational time needed with particular parameters. Each row represents a distinguished maximum depth distance.



(a) Gained coverage (b) Computational time

Figure 5.3: Impact of overlap and distance from the building on coverage and computational time with maximum depth distance set to 10.



(a) Gained coverage (b) Computational time

Figure 5.4: Impact of overlap and distance from the building on coverage and computational time with maximum depth distance set to 20.

Figure 5.3 determines gained coverage and processing time with a selected maximum depth distance of 10. To compare the defined depth, Figure 5.4 presents the results with chosen maximum depth distance of 20. Similarly, a high coverage above 95 % is achieved with waypoints set close to the building and the highest overlap possible. Keeping the distance short and decreasing the percentage of overlap result in a slight decrease in the gained coverage. An unsatisfying coverage arises from a distance above 10m and a low-selected overlap. Both plots are monotonically decreasing, meaning the further a waypoint is located from the building and the least two waypoints overlap, the lower the coverage percentage. With a distance increase and overlap decrease, the difference between the two selected maximum depth distances becomes more apparent. While the coverage with a maximum depth distance of 20 falls gradually, a sharp fall can be recognized for a depth of 10. The observed decline is an automatic result from the specified maximum depth distance. The further the waypoints are located from the building and the lower the maximum depth distance is selected, the least clear visibility is maintained.

The computational time for both maximum depth distances is monotonous increasing. However, a set of waypoints far from the building with a high percentage of overlap produces a dramatic leap in processing time, as illustrated in Figures 5.3b and 5.4b, which describes the logarithmic processing time. In general, calculating the coverage with maximum depth distance set to 10 prolong significantly longer than selecting 20. The reason is that the coverage analysis marks fewer points on the surface as visible, meaning it iterates over more possible points.

The coverage analysis leads to the following conclusion - the method implemented for this thesis is robust against a selection overlap and distance away from the object to be scanned. However, the tool significantly depends on determining camera parameters and the maximum depth distance to ensure a satisfying coverage. In the presented cases, the selected maximum depth distance of 20 achieves a minimum coverage of 79 %, while the minimum value for setting the depth to 10 is 7%.

5.1.2 Discussion

In the following section, a short comparison between the resulting parameters from the literature review and the results from the waypoint creation is conducted. To summarize, the literature review shows that a 10-meter distance between the drone and the building ensures safety and efficiency, and forward overlap above 90% achieves good reconstruction accuracy.

The results from the method considering the given parameters are listed in the Table 5.1. It is apparent from this table that considered parameters result in 94.3 % coverage of the building for maximum depth distance set to 10 and 99.7% for depth set to 20. The short distance between the points produces a large number of waypoints, and the computational time is high. According to the results presented in Figure 5.4, slightly lowering the overlap percentage will return almost the same coverage for a maximum depth distance of 20. Furthermore, the number of created waypoints reduced drastically, which yields reduced processing time. Creating the waypoints closer to the building and keeping the overlap high is inconvenient as the number of waypoints produced is high, but the coverage remains stable. A possible alternative would be to slightly decrease both parameters to maintain coverage above 90% but reduce the number of waypoints. Therefore, setting the distance parameter to 8 m and the overlap to 50% is considered. The results of the examination with these parameters are presented in Table 5.2. Noticing the reduction of coverage is possible by setting the parameters in a way to result in an 85% of the building covered with a maximum depth distance of 20, as proposed in Table 5.3. The contrast of the depth definition can be observed in this example. While a maximum depth distance of 20 maintains a satisfying coverage, setting this parameter to 10 results in a sharp decrease. Therefore, further analysis will rely on those three cases for a maximum depth distance of 20.

Table 5.1: Results with 10 m distance and 90% overlap

Distance between waypoints	Number of waypoints	waypoints processing time (s)	Maximim depth distance	Coverage	Coverage processing time(s)
1.11	11399	1178.5	10	0.943	1905,5
1.11	11399	1178.5	20	0.997	125.0

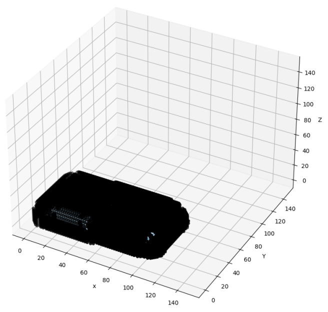
Table 5.2: Results with 8 m distance and 50% overlap

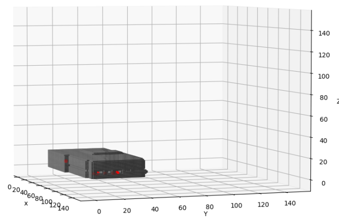
Distance between waypoints	Number of waypoints	waypoints processing time (s)	Maximim depth distance	Coverage	Coverage processing time(s)
4.43	761	334.2	10	0.925	292.2
4.43	761	334.2	20	0.989	84.2

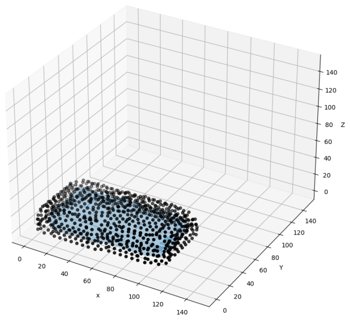
Table 5.3: Results with 12 m distance and 20% overlap

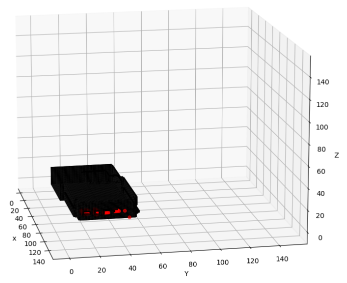
Distance between waypoints	Number of waypoints	waypoints processing time (s)	Maximim depth distance	Coverage	Coverage processing time(s)
10.64	141	268.3	10	0.177	133.7
10.64	141	268.3	20	0.852	83.2

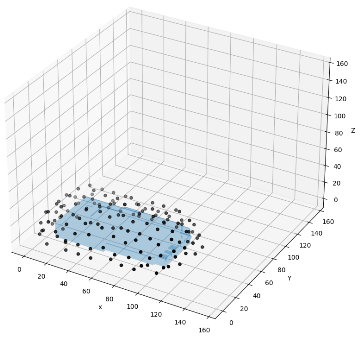
Figure 5.5 illustrates the result of the waypoint creation with the set parameters. The distance between the points results in different points' densities. The closer the waypoints are from each other, the higher the number of points are produced, which can be observed in the 5.5a, 5.5c and 5.5e. The second plot in each row shows visualizations of the results in coverage analysis. The red point presents the surface one-meter-voxels not visible from the set of waypoints. Figures 5.5b and 5.5d show that the entire structure of the building is covered except for some points which seems inside, reflecting the around 99% coverage. The non-visible points are representative of an entrance surrounded by columns of the Mensa building. The distance of the building and the direction facing the building in the method are fixed, and changing accordingly to capture this type of build details is limited. The surrounding is open, meaning the flood fill algorithm would not consider the columns part of the inside voxels. Figure 5.5f displays the effect of reduced overlap and lower number of waypoints on the coverage, resulting in a score of 85%. The missing covered surface points are located laterally of the building and in the middle of the rooftop. This is mainly due to the fact that the distance between the points is above 10 m.


(a) Waypoints with overlap 90% and distance 10 m


(b) Coverage in black versus unseen points in red


(c) Waypoints with overlap 50% and distance 8 m


(d) Coverage in black versus unseen points in red


(e) Waypoints with overlap 20% and distance 12 m

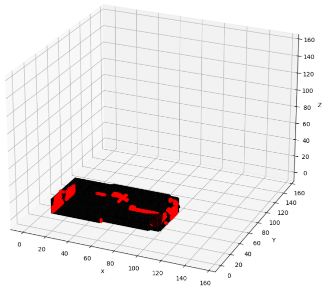

(f) Coverage in black versus unseen points in red

Figure 5.5: Defined waypoints and building coverage

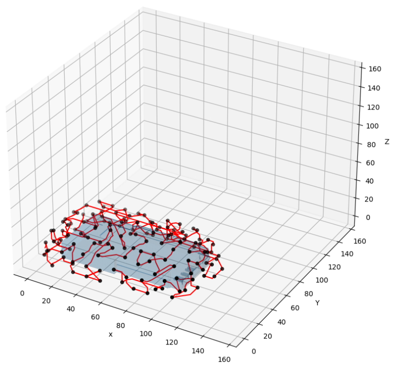
5.2 Path planning comparison

The path planning analysis compares the three approaches considering the Traveling Salesman Problem, performed on the waypoints set with parameters, a distance of 12m and overlap of 20%. The analysis includes the processing time, the costs of the path, and the L-function, described in Section 3.6. The three approaches are tested for an equal amount of waypoints. The Table 5.4 summarizes the results for the selected case, and the three plots in Figure 5.6 visualize the path tendencies.

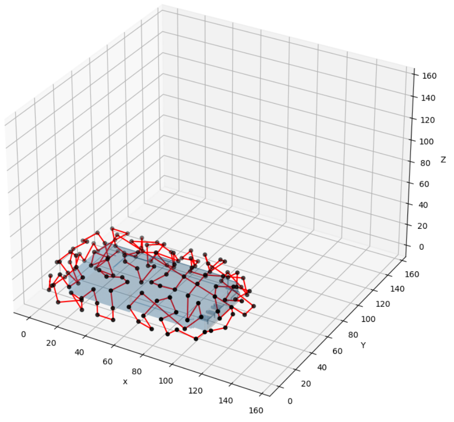
The second approach calculates a path in a significantly shorter processing time despite the fact that the TSP with the cheapest insertion is executed parallelly. It appears that the parallel processing does not compensate for the higher time complexity of the algorithm. However, the third approach provides the cheapest costs, followed by the TSP on closest points. The method with the clusters is twice more expensive in comparison to the other algorithms. The first approach also features the most randomness in the path, while the second and the third technique have a more strict pattern. The difference is due to the incorporation of the priority to the one axis. Figure 5.6 displays the common tendencies between the TSP with the cheapest insertion and TSP on closest points.

Table 5.4: Comparison of the three approaches

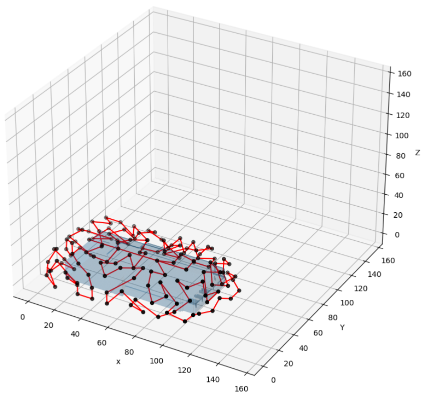
TSP	Number of waypoints	Processing time (s)	Costs	L
Clusters	141	106.68	2642.2	2783,5
Closest points	141	88.6	1320.9	1461.9
Cheapest insertion	141	902.8	1178.1	1319.1



(a) TSP with clusters



(b) TSP on closest points



(c) TSP with the cheapest insertion

Figure 5.6: Travelling Salesman Problem approaches

Chapter 6

Concluding Remarks

6.1 Conclusion

In the digital era nowadays, one significant advantage of the evolution of technologies is their utilization for helping the community. Using technologies improves the quality of government services and citizen wellness. From managing transportation to rapid reaction to the scene of an emergency, technologies can enhance operations across any urban area. In this context, the INTREPID project makes a one step further to creating a smart city. Creating a platform for analyzing a building environment assists the missions of first responders in saving lives and neutralizing threats.

The main goal of this thesis was to implement a method for outdoor path planning for the given BIM model in IFC format. The UAV should be capable of scanning the outside of the building from above and side. First, a comprehensive literature analysis was conducted to define the objectives of the thesis and investigate the existing path planning strategies.

The first step of the method converts the IFC format input to a convenient mission planning format. A voxel-based model was proposed, structured in an octree, and simplified in four steps. In the next step, the coordinates from scanning the building were defined, on which a path planning algorithm was performed. The Traveling Salesman Problem with A* algorithm suggested in the thesis was based on literature research. The thesis proposed three different approaches: TSP with clusters, TSP on closest points, and TSP with the cheapest insertion. The resulting path is represented by the coordinates of the waypoints defined by x, y, and z coordinates in a JSON file. Finally, the method was tested and evaluated on the basis of waypoints coverage, overlap, and path planning efficiency.

6.2 Recommendations for future work

Based on the literature research and the method created in this thesis with its limitation, the following recommendations for future work are proposed:

- An optimization parallel processing can speed up the flood fill and waypoint creation. To build a good voxel representation of the IFC model, the generated grid with the proportions of the building produces a long computational time for both functions as the functions iterate over the whole voxel grid space.
- The thesis presents a waypoint creation at a fixed distance away from the building. A limitation occurs for building environments with indoor yards. In the future, one

can implement a feature to the function, which considers an automatic increase or decrease of distance.

- Similar to the flood fill algorithm and waypoints creation, optimization for path planning with parallel processing is possible. The thesis proposes the priority for moving along an axis. In future work, one can implement a feature that prioritizes a direction of movement, meaning forcing the planning of the path to continue in the same direction.

- One can compare the Traveling Salesman Problem with the A* algorithm proposed in this thesis with other possible approaches for solving drone mission planning problems used in the literature.

References

ANDERSON, K., & GASTON, K. J. (2013). Lightweight unmanned aerial vehicles will revolutionize spatial ecology. *Frontiers in Ecology and the Environment*, *11*(3), 138–146.

BEHNCK, L. P., DOERING, D., PEREIRA, C. E., & RETTBERG, A. (2015). A modified simulated annealing algorithm for suavs path planning. *Ifac-Papersonline*, *48*(10), 63–68.

BUILDINGSMART. (2021). Industry foundation classes (ifc) - an introduction [Accessed: 2021-11-30]. https://technical.buildingsmart.org/standards/ifc

DANDOIS, J. P., OLANO, M., & ELLIS, E. C. (2015). Optimal altitude, overlap, and weather conditions for computer vision uav estimates of forest structure. *Remote Sensing*, *7*(10), 13895–13920.

DARYANAVARD, H., & HARIFI, A. (2019). Uav path planning for data gathering of iot nodes: Ant colony or simulated annealing optimization. *2019 3rd International Conference on Internet of Things and Applications (IoT)*, 1–4.

DEBUS, P., & RODEHORST, V. (2021). Evaluation of 3d uas flight path planning algorithms. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *43*, 157–164.

DOMINGO, D., ØRKA, H. O., NÆSSET, E., KACHAMBA, D., & GOBAKKEN, T. (2019). Effects of uav image resolution, camera type, and image overlap on accuracy of biomass predictions in a tropical woodland. *Remote Sensing*, *11*(8), 948.

EISENBEISS, H. (2009). *Uav photogrammetry* (Doctoral dissertation). ETH Zurich.

FANG, S., & CHEN, H. (2000). Hardware accelerated voxelization. *Computers & Graphics*, *24*(3), 433–442.

FREIMUTH, H., & KÖNIG, M. (2019). A framework for automated acquisition and processing of as-built data with autonomous unmanned aerial vehicles. *Sensors*, *19*(20), 4513.

GITHUB. (2021). Binvox-rw [Accessed: 2021-12-07]. https://github.com/dimatura/binvox-rw-py/blob/public/binvox_rw.py

GUPTE, S., MOHANDAS, P. I. T., & CONRAD, J. M. (2012). A survey of quadrotor unmanned aerial vehicles. *2012 Proceedings of IEEE Southeastcon*, 1–6.

HAMMOND, J. E., VERNON, C. A., OKESON, T. J., BARRETT, B. J., ARCE, S., NEWELL, V., JANSON, J., FRANKE, K. W., & HEDENGREN, J. D. (2020). Survey of 8 uav set-covering algorithms for terrain photogrammetry. *Remote Sensing*, *12*(14), 2285.

HARTLEY, R., & ZISSERMAN, A. (2003). Camera models. *Multiple View Geometry in Computer Vision*, *2*.

HERNANDEZ-LOPEZ, D., FELIPE-GARCIA, B., GONZALEZ-AGUILERA, D., & ARIAS-PEREZ, B. (2013). An automatic approach to uav flight planning and control for photogrammetric applications. *Photogrammetric Engineering & Remote Sensing*, *79*(1), 87–98.

Höhle, J. (2013). Oblique aerial images and their use in cultural heritage documentation. *Proc. Int. Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *5*, W2.

Hussein, A., Al-Kaff, A., de la Escalera, A., & Armingol, J. M. (2015). Autonomous indoor navigation of low-cost quadcopters. *2015 IEEE international conference on service operations and logistics, and informatics (SOLI)*, 133–138.

Hussein, A., Mostafa, H., Badrel-din, M., Sultan, O., & Khamis, A. (2012). Metaheuristic optimization approach to mobile robot path planning. *2012 international conference on engineering and technology (ICET)*, 1–6.

Ibrahim, A., & Golparvar-Fard, M. (2019). 4d bim based optimal flight planning for construction monitoring applications using camera-equipped uavs. *Computing in civil engineering 2019: Data, sensing, and analytics* (pp. 217–224). American Society of Civil Engineers Reston, VA.

IfcOpenShell.org. (2021a). Ifcconvert [Accessed: 2021-12-07]. http://ifcopenshell.org/ifcconvert

IfcOpenShell.org. (2021b). Ifcopenhouse [Accessed: 2021-06-27]. http://ifcopenshell.org/ifcopenhouse/v4/IfcOpenHouse.ifc

INTREPID. (2020). Intrepid [Accessed: 2021-12-29]. https://intrepid-project.eu/

Isbudeen, N., Ajaz, S., & Buhari, P. A. (2011). Network intrusion detection in wireless network using kabsch algorithm. *Proceedings of the IEEE International Conference on Computational Intelligence and Computing Research*, 1457–1461.

Jansohn, R. (2010). Pythonocc: Geometry introduction [Accessed: 2022-01-07]. https://pythonocc-doc.readthedocs.io/en/latest/geom_intro/

Kabsch, W. (1978). A discussion of the solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, *34*(5), 827–828.

Ma, S. (2020). Understanding the travelling salesman problem (tsp) [Accessed: 2021-12-08]. https://blog.routific.com/travelling-salesman-problem

Manconi, A., Ziegler, M., Blöchliger, T., & Wolter, A. (2019). Optimization of unmanned aerial vehicles flight planning in steep terrains. *International Journal of Remote Sensing*, *40*(7), 2483–2492.

McKerns, M. (2022). Pathos [Accessed: 2022-01-09]. https://github.com/uqfoundation/pathos

Mesas-Carrascosa, F.-J., Notario Garcia, M. D., Meroño de Larriva, J. E., & Garcia-Ferrer, A. (2016). An analysis of the influence of flight parameters in the generation of unmanned aerial vehicle (uav) orthomosaicks to survey archaeological areas. *Sensors*, *16*(11), 1838.

Min, P. (2004 - 2019a). Binvox [Accessed: 2021-09-15]. https://www.patrickmin.com/binvox

Min, P. (2004 - 2019b). Viewvox [Accessed: 2021-12-18]. https://www.patrickmin.com/viewvox/

Murata, T., Ishibuchi, H. et al. (1995). Moga: Multi-objective genetic algorithms. *IEEE international conference on evolutionary computation*, *1*, 289–294.

NumPy.org. (2021). Numpy [Accessed: 2021-12-09]. https://numpy.org/

Oberlin, P., Rathinam, S., & Darbha, S. (2010). Today's traveling salesman problem. *IEEE robotics & automation magazine*, *17*(4), 70–77.

Perazzo, P., Sorbelli, F. B., Conti, M., Dini, G., & Pinotti, C. M. (2016). Drone path planning for secure positioning and secure position verification. *IEEE Transactions on Mobile Computing*, *16*(9), 2478–2493.

python.org. (2021). What is python? [Accessed: 2021-11-17]. https://www.python.org/doc/essays/blurb/

Roca, D., Armesto, J., Lagüela, S., & Diaz-Vilariño, L. (2014). Lidar-equipped uav for building information modelling. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, *45*.

Rodenberg, O., Verbree, E., & Zlatanova, S. (2016). Indoor a* pathfinding through an octree representation of a point cloud. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *4*, 249–255.

Samet, H. (1982). Distance transform for images represented by quadtrees. *IEEE Transactions on Pattern analysis and machine intelligence*, (3), 298–303.

scikit-learn.org. (2020). Clustering [Accessed: 2021-12-08]. https://scikit-learn.org/stable/modules/clustering.html

SciPy.org. (2021). Scipy documentation [Accessed: 2021-12-07]. https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.transform.Rotation.html

Seifert, E., Seifert, S., Vogt, H., Drew, D., van Aardt, J., Kunneke, A., & Seifert, T. (2019). Influence of drone altitude, image overlap, and optical sensor resolution on multi-view reconstruction of forest images. *Remote sensing*, *11*(10), 1252.

Simek, K. (2012). Dissecting the camera matrix, part 2: The extrinsic matrix [Accessed: 2022-01-02]. https://ksimek.github.io/2012/08/22/extrinsic/

Simek, K. (2013). Dissecting the camera matrix, part 3: The intrinsic matrix [Accessed: 2022-01-02]. https://ksimek.github.io/2013/08/13/intrinsic/

Smith, D. K. (2009). Getting started and working with building information modeling. *Facilities Manager*, *25*(3), 20–24.

Stanton, D. (2020). A-star pseudocode [Accessed: 2022-01-08]. https://gist.github.com/damienstanton/7de65065bf584a43f96a

Strubel, D., Morel, O., Saad, N. M., & Fofi, D. (2017). Evolutionary algorithm for positioning cameras networks mounted on uav. *2017 IEEE Intelligent Vehicles Symposium (IV)*, 1758–1763.

Tang, L., & Shao, G. (2015). Drone remote sensing for forestry research and practices. *Journal of Forestry Research*, *26*(4), 791–797.

Tauro, F., Petroselli, A., & Arcangeletti, E. (2015). Assessment of drone-based surface flow observations. *Hydrological Processes*, *30*(7), 1114–1130.

Themistocleous, K., Agapiou, A., & Hadjimitsis, D. (2016). 3d documentation and bim modeling of cultural heritage structures using uavs: The case of the foinikaria church. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *42*, 45.

Torres-Sánchez, J., López-Granados, F., Serrano, N., Arquero, O., & Peña, J. M. (2015). High-throughput 3-d monitoring of agricultural-tree plantations with unmanned aerial vehicle (uav) technology. *PloS one*, *10*(6), e0130479.

Tuba, E., Capor-Hrosik, R., Alihodzic, A., & Tuba, M. (2017). Drone placement for optimal coverage by brain storm optimization algorithm. *International Conference on Hybrid Intelligent Systems*, 167–176.

Vandevenne, L. (2018). Flood fill [Accessed: 2021-12-07]. https://lodev.org/cgtutor/floodfill.html

Watts, A. C., Ambrosia, V. G., & Hinkley, E. A. (2012). Unmanned aircraft systems in remote sensing and scientific research: Classification and considerations of use. *Remote Sensing*, *4*(6), 1671–1692.

WolframMathWorld. (2021a). Euler angles [Accessed: 2021-12-01]. https://mathworld.wolfram.com/EulerAngles.html

WolframMathWorld. (2021b). Vector [Accessed: 2021-12-09]. https://mathworld.wolfram.com/Vector.html

Xiong, Q., Zhu, Q., Zlatanova, S., Du, Z., Zhang, Y., & Zeng, L. (2015). Multi-level indoor path planning method. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, *40*(4), 19.

Zarembo, I., & Kodors, S. (2013). Pathfinding algorithm efficiency analysis in 2d grid. *ENVIRONMENT. TECHNOLOGIES. RESOURCES. Proceedings of the International Scientific and Practical Conference*, *2*, 46–50.

Zhu, Q., Yan, Y., & Xing, Z. (2006). Robot path planning based on artificial potential field approach with simulated annealing. *Sixth International Conference on Intelligent Systems Design and Applications*, *2*, 622–627.

Zou, Y., Barati, M., Rey Castillo, E., & Amor, R. (2019). Automated uav route planning for bridge inspection using bim-gis data. *Conference: 4th International Conference on Civil and Building Engineering Informatics At: Sendai, Japan*, 384–391.

Declaration

I hereby affirm that I have independently written the thesis submitted by me and have not used any sources or aids other than those indicated.

Location, Date, Signature