Comenius University, Bratislava
Faculty of Mathematics, Physics and Informatics

Automatic 3D scanning using drones and photogrammetry

Master's Thesis

2019
Bc. Ladislav Feldsam

Comenius University, Bratislava
Faculty of Mathematics, Physics and Informatics

Automatic 3D scanning using drones and photogrammetry

Master's Thesis

Study programme:	Informatics
Study field:	2508 Informatics
Department:	Department of Applied Computer Science
Supervisor:	doc. RNDr. Milan Ftáčnik, CSc.
Consultant:	RNDr. Martin Bujňák, PhD.

Bratislava, 2019
Bc. Ladislav Feldsam

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Ladislav Feldsam
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Automatic 3D scanning using drones and photogrametry
Automatické 3D skenovanie s použitím dronov a fotogrametrie

Anotácia: Teoreticko-implementačná práca

Cieľ: Cielom prace je navrhnut algoritmus, ktory naplanuje trajektoriu a pohlady pre dron tak, aby zmapoval cely vopred zadany 3D objekt. Dron moze opakovane vzlietnut a pristat, zanalyzovat data a navrhnut dalsie doplnujuce lety. Ocakavanie je vytvorenie algoritmu, ktory dostane na vstupe riedky model, polohy kamier, pripadne mesh a navrhne nove miesta, odkial treba objekt dofotit. Nasledne spocita krivku, po ktorej bude dron letiet a kam bude pozerat.

Vedúci: doc. RNDr. Milan Ftáčnik, CSc.
Konzultant: RNDr. Martin Bujňák, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 11.10.2017

Dátum schválenia: 11.12.2017 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Acknowledgement: I want to thank for every help in the implementation of the algorithm from my consultant RNDr. Martin Bujňák, PhD. and the whole team of Capturing Reality. I want to thank to my supervisor doc. RNDr. Milan Ftáčnik, CSc. for help in the academic part of the thesis. Nevertheless I want to thank to my family with the huge support behind the writings of this thesis.

Abstract

In this thesis we focus on the problem of automatic scanning of a general 3D world object or a scene using a drone. We mainly address the flight mission planning part, where the locations and look-at directions of the drone's camera are calculated as well as the path planing. New camera positions and a path, that visits all the camera positions with a drone, are calculated from the sparse mesh and camera poses calculated with photogrammetry. We find regions in the sparse model with insufficient detail. These regions are then clustered into bigger regions in a way, that they can be photographed in whole. Camera positions, which are viewing the regions with small detail are constructed to have a clear line of sight to the given region. The routed flight, that visits all the camera positions is evaluated to be unobstructed and to have the smallest possible flight time. This process of flight and processing can be repeated until we get our desired reconstruction.

Keywords: Drone, Photogrametry, Automatic reconstruction

Abstrakt

V tejto práci sa sústredíme na problém automatického skenovania ľubovoľného 3D objektu alebo scény s použitím dronu. Zameriavame sa hlavne na časť plánovania letovej misie, kde vypočítavame pozície kamier dronu, ich pohľady a tiež trasy medzi nimi. Nové polohy kamier s pohľadmi a cesta, ktoré prechádza cez všetky polohy kamier s dronom, sú vypočítané z riedkeho modelu a polôh kamier, ktoré sú vypočítané fotogrametriou. Z tohto modelu nájdeme oblasti, ktoré majú nedostatok detailu. Tieto oblasti sú zhlukované to väčších oblastí, tak aby ich bolo možné odfotografovať v celku. Pozície kamier, ktoré majú pohľady na oblasti s nízkou kvalitou detailu sú skonštruované tak aby mali priamu viditeľnosť na danú oblasť. Smerovaný let, ktorý navštívi všetky pozície kamier je, navrhnutý tak, aby obchádzal prekážky a mal najnižší možný čas letu. Tento proces letu a spracovania je možné opakovať dovtedy, kým nedostaneme požadovanú kvalitu rekonštrukcie.

Kľúčové slová: Dron, Fotogrametria, Automatická rekonštrukcia

Contents

List of Figures

Introduction

Growing hunger for 3D models is natural, since end users are expecting a better experience, e.g., by shifting from 2D maps to 3D maps or demand fluent 3D graphics instead of 2D bitmaps, but there are also completely new areas like Virtual Reality, where 3D content is crucial. In the past, the main-stream methods for creating 3D models were laser scanning (or time of flight sensors) and structured light sensors. It was due to their speed, but at the same time these methods were inaccessible due to their cost and they were used mainly in surveying, manufacturing and so on. However, with a recent development in mathematics and algorithms, photogrammetry become a very competitive player in terms of speed, accuracy and typically produces higher level of detail. It also does not need a special device, just images, e.g., from mobile phone.

With the help of photogrammetry methods, we are able to automatically register the image‘s position, without any additional information just using the visual information. This means that we can use any imagery, even without georeferencing or data from accelerometers, which were commonly used in the past. However, having this information available, we can georeference the 3D model.

On the other hand, since we want to have all the possible details of the scene, which is being scanned, we must capture every detail. This task can be hard for novices and even for professionals. It is possible, that while capturing the scene abroad, we forgot to capture some detail and thus loose the information about that region. Even worse is, if it might not be possible to go back and take the photograph of the given region again.

In our work we are targeting an autonomous process of taking photographs of scenes with drones. We are using drones, which developed rapidly in the recent past and got cheaper and more accessible for everyone.

Drones become the perfect tool for such task because of:

- **Battery life**: Today drones are capable of flying over half an our, due to dense lithium-ion batteries which are always getting better.
- **Sensors**: Most of the drones have sensors, that allows the drone to sense the scene near to it. It can stop or even avoid obstacles.

- **GPS**: The global positioning system in these flying machines is precise. This allows us to know the georeference of the taken photographs with an accuracy of few centimetres.

- **Camera**: Drones are equipped with high resolution cameras, that are capable taking perfect images for photogrammetry reconstructions.

The goal of this thesis is to calculate the required information about the scene and cameras with a portable computer in a short time is a mandatory requirement for this thesis. The goal is to develop algorithms, that can calculate new camera positions, from which we are able to take lacking features from the scene. Also to calculate the paths through, which the drone can safely fly from the starting point through every camera position and back to the starting point.

In the first chapter, we introduce the State of The Art. What is done around this thematic. We show what are the current autonomous methods for planning flight missions and what is the closest work related to ours. For completeness we show what depth acquisition techniques exist and what exactly UAV's (Unmanned Aerial Vehicles) are.

The second chapter is focused in what is our approach for autonomous 3D scanning. In the background, we introduce important notions and techniques used in the work. We show how, we find the related regions in the model and how to plan the flight mission for the drone.

The Implementation chapter shows what engines and libraries are used in our software. How to test the method constructed in the section about our work and how to solve the problem with non georeferenced images.

The final chapter is showing the results of our method and what runtimes we manage to achieve with our method.

The result should be an application achieving an easy and cheap 3D scanning for everyone with the smallest effort possible.

Chapter 1

State of The Art

In this chapter we are going to discuss, semi-autonomous and fully-autonomous systems of capturing information for 3D reconstruction. We also show different approaches to obtain the depth informations, from which we can make the 3D reconstruction.

Since our work is related to drones, we will show the advantages and disadvantages of these UAVs (Unmanned Aerial Vehicles) and how are they used in photogrammetry nowadays. Although, most of the autonomous systems using UAVs for capturing photos are limited for nadiral (orthographic) image collection, we managed to find one work which has the same objective as has this thesis. This work will be also described at the end of this chapter.

There is a lack of systems, which try to solve autonomous image acquisition of generally shaped objects and this was also mentioned by Remondino [6]. The next sentence is cited from this paperwork.

"Flight planning is quite simple when using nadiral images, the same task becomes much more complex in case of 3D objects requiring convergent images and, maybe, vertical strips. Future work has to be addressed to develop tools for simplifying this task."

Later works, were focused more in better navigation and processing of orthographic scans. Also with LiDAR technology the focus was to make devices more accurate. Just a few works were done in the same directions as is the main goal of this thesis.

1.1 Orthographic photogrammetry

Orthographic photogrammetry was one of the first method for creating maps used for measuring distances, urban planing and so on.

This was achieved by systematically taking and processing photographs (Figure 1.1) from high altitude or later using satellite imagery to produce these maps.

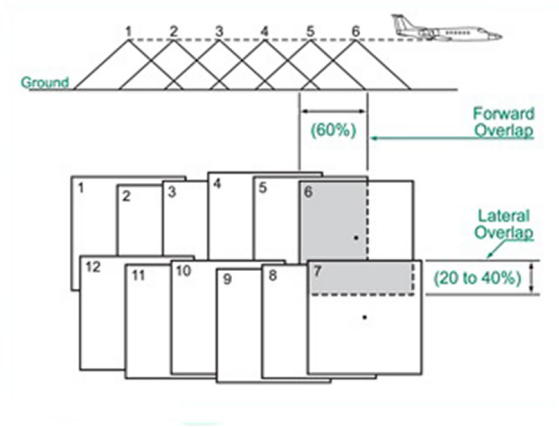

Figure 1.1: Orthographic overhead pattern for the UAV.

Source: PhotogrammetryNews [18]

Typically the flight path was calculated in a way, that the given overlap between images was ensured and at the altitude, the required resolution of the photographs was achieved.

Today the big demand for 3D maps and surveying started this method of obtaining imagery. These scans were firstly done with big planes, that have mounted camera‘s. One that is looking straight down and four looking to the sides in a 45° angle. The images are obtained with a given overlap to be able to reconstruct the 3D model (Fig. 1.1).

Later the big aeroplanes were replaced with smaller drones for more precise image acquisition. To be able to reconstruct the 3D model from orthoimages in a specific detail we need flight planning, such that the distance from the ground is not changing. If the images are taken from a higher altitude, we loose resolution and thus have smaller detail than required.

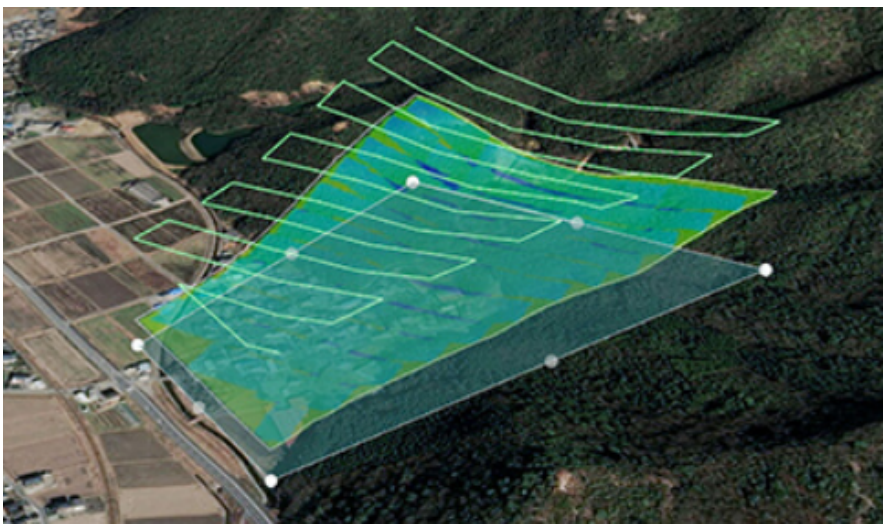

Figure 1.3: Orthographic overhead pattern for the UAV.

Remondino [6] describes the preparation of the flight plan for orthographic photogrammetry.

Such field surveying requires a flight or mission planning, GCPs (Ground Control Points) measurement (if not available and required for geo-referencing).

The missions are planned in lab with dedicated software, starting from the area of interest, the required ground sample distance or footprint, and knowing the intrinsic parameters of the mounted digital camera. Thus fixing the image scale and camera focal length, the flying height is derived. This is calculated for the desired precision of the reconstruction and it is critical for the terrain with variable height.

Orthographic scans mostly use aerial vehicles like drones or ultra light planes. A great advantage is that they can cover huge areas and take photographs from high above the terrain.

The disadvantage is that they scan the terrain only from above and details from shallow angles are not captured. Thus the details of more precise objects like cars, trees or houses etc. are limited.

In most cases it is not necessary to have such a detail of an aerial scan, but there are applications, where we need to have the most out of the scan and have it as precise as possible.

The mission sends the UAV for a specific path above the desired area to take photographs. It is not evaluating the result afterwards, to get more picture information in the next flight and that is the biggest difference with our work.

There is whole study about what path of overlapping photographs (Figure 1.1) reconstructs the best result, but the method does not considerate the difference in various situations for adaptation. This is the reason why this method is not suited for generally shaped objects.

1.2 General path planning

In recent years several drone platforms came up with systems, which allow general path planning. One of such is Intel‘s Insight Platform.

Intel developed a software that enables the user to plan flight mission where the user is able to construct not only orthographic paths but also vertical paths or spirals that fly around the given object.

According to Intel [10] and [9].

Intel’s Insight Platform is a digital asset management system that enables aerial data management and analysis using Pix4D‘s and Bentley‘s photogrammetric processing engines. It allows customers to store, share and manage data that commercial drone systems collect. According to the company, the platform is designed to reduce

costs, improve efficiency and fuel growth. It takes the aerial data and can generate 2D and 3D models, take measurements and run data analytics.

1.2.1 Intel's Mission Control

Is the company's next-generation flight planning software for its Falcon 8+ drone. According to the company, it is designed to increase workflow efficiency and enhance automation of drone flights for commercial missions.

Mission Control allows Falcon 8+ drone operators to create 2D and 3D flight plans for commercial surveying, mapping and inspection missions. Flight planning is automated with advanced preset mapping modes. Multiple layers of airspace information are integrated to support flight safety and compliance. Automatic pre-flight safety and system checks help validate the flight plan before the mission is executed. After the mission is completed, the software provides a quick preview of the collected data so the UAV operator can check and verify for adequate area coverage and overlap and even inspect the quality of individual images in a 3D format.

Obstacle avoidance increases safety and reliability of complex missions. The technology includes depth-sensing modules that compute raw image streams into high-resolution 3D depth maps. The obstacle-avoidance feature on the Intel Imaging and Intel Dual Imaging payloads enables the Intel Falcon 8+ drone to detect and avoid potential obstacles or hazards and has the ability to maintain and hold a set distance from an asset during inspection. Within a mission, objects are continually identified in real time and a 3D depth map of the environment is maintained in memory.

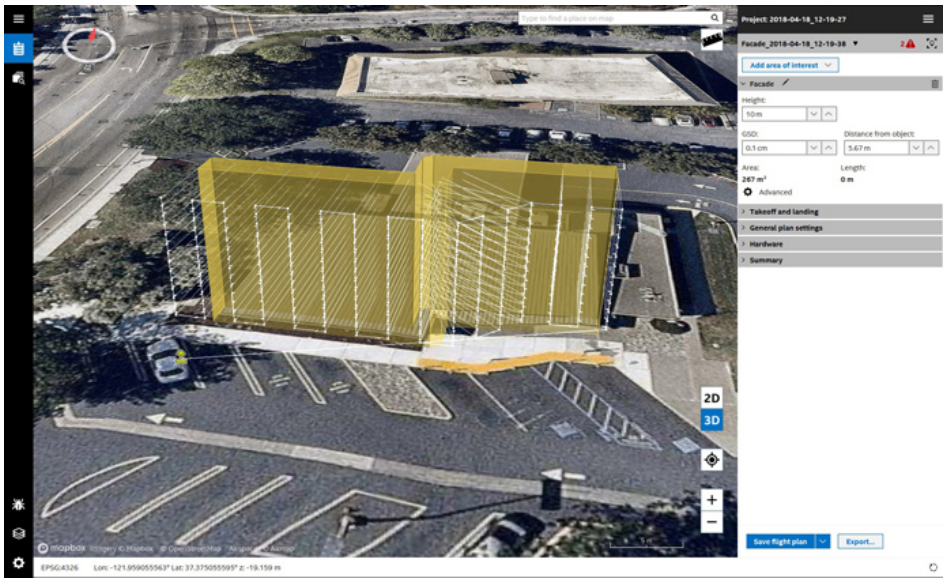

Figure 1.4: Screenshot of the Intel's Mission Control software

Source: Intel [9]

The software allows planning of missions, which includes vertical strips with horizontal looks to a given 3D scene. On the Figure 1.4 the vertical strips, which are paths for the drone can be seen.

This option is an improvement to other methods, but it is still a manual process of placing the vertical strips.

1.3 Plan3D

Plan3D [1] tries to solve exactly the same problem as in this thesis, which is Viewpoint and Trajectory Optimization for Aerial Multi-View Stereo Reconstruction.

The work introduces a method which efficiently computes a set of viewpoints and trajectories for high quality 3D reconstruction in outdoor environments. Goal is to automatically explore an unknown area and obtain a complete 3D scan, using an RGB camera mounted on a autonomously navigated quadcopter. The flight time is restricted by the maximal flight time of the UAV.

At the core of this method lies a hierarchical volumetric representation that allows the algorithm to distinguish between unknown, free, and occupied space. Furthermore, this information gain based formulation leverages this representation to handle occlusions in an efficient manner. In addition to the surface geometry, they utilize the free-space information to avoid obstacles and determine collision-free flight paths.

First, a user defines a region of interest and specifies a safe and simple overhead pattern (orthographic scan) to obtain the initial set of images.

These images are then processed by the state-of-the-art Structure from motion (SfM) and Multi View Stereo (MVS) pipeline to obtain camera positions with depth and normal maps for each viewpoint. To generate a 3D surface reconstruction these depth maps are fused to a dense point cloud, and with a utilization of the Poisson Surface Reconstruction method a mesh is extracted.

This initial reconstruction is highly inaccurate, since the viewpoints are acquired by a simple overhead pattern. From this initial scan the volumetric occupancy map is obtained.

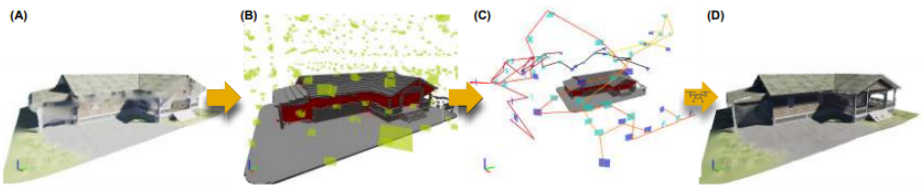

Figure 1.6: Plan 3Ds algorithm pipeline

Source: Plan3D [1]

On the picture (1.6) we can see the approach of authors to this problem. The first step, which is seen on (A) is initialization via images from a regular overhead pattern which is not shown on the picture. On picture (B) we can see the generated set of viewpoint candidates (yellow) on which they use a submodular optimization formulation to find an optimized viewpoint paths, maximizing information about uncertain space, which are seen in red. (C) shows the generated trajectory, which takes constraints on SfM and MVS reconstruction such as preference for fronto-parallel view and matching quality into consideration. In the final picture we can see the final high quality reconstruction obtained via SfM and MVS pipeline.

The most interesting part is how they calculate the relevant positions for the cameras and find the path, which flies through them.

The approach consists of voxelization of the scene and finding voxels with the uncertainty about the scene. The main objective of the optimization is to maximize total information, while staying within the travel budget of the quadrotor and respecting constraints imposed by SfM and MVS. The total information corresponds to the observation count of all voxels withing the region of interest. In theory if all voxels have been observed multiple times from different angles the entire surface can be reconstructed with high quality. This goal has to be traded-off with limited battery time of the UAV and computational cost as the evaluation of all possible viewpoints during planning is infeasible. Furthermore, it has been shown that at some point adding views yields diminishing returns.

Image (1.6), C shows the output of their planning method, where viewpoints that were added due to their contributed information are rendered in blue. Additional viewpoint that were added to ensure that the SfM and MVS backend can register all images into a single reconstruction are rendered in cyan.

1.3.1 Optimizing viewpoint trajectories

The goal is to find an optimized subset of viewpoints, from a larger set of candidate views, that maximizes the information gained about the 3D surface of the scene.

Assume given graph $G = (C, M)$ of viewpoint candidates C alongside observed voxels and motions M between viewpoint as edges. Each viewpoint $v \in C$ has an associated position and orientation denoted as $v\mathbf{p}$ and $v\mathbf{q}$.

The trajectory (path through a subset of the nodes of candidates) for the UAV that yields good reconstruction quality and fulfils the maximal travel distance L_{max} is constructed as follows. Let $VP = (vp_1, \ldots, vp_n)$ be the sequence of viewpoints to be traversed during image capture where $vp_i \in C$. Let $L(VP)$ be the geometric length of the trajectory VP and with S the set of all sequences VP.

Formally, they solve the following optimization problem:

$$VP^* = \underset{VP \in S}{argmax}\ I(VP),\ L(VP) \leq L_{max}, \tag{1.1}$$

where $I(VP)$ is the objective function that measures the amount of information contributed by the respective viewpoints. This objective function can be written as:

$$I(VP) = \sum_{\tau \in OM \setminus OM_{free}} VI(\tau, VP)$$
$$OM_{free} = \{\tau \in OM : oc(\tau) \leq oc_{free}\}$$

where $VI(\tau, VP)$ is the camera measurement model specifying how much information of voxel τ is contributed by the traversed viewpoints VP, $oc(\tau) \in [0, 1]$ is the occupancy value of each voxel $\tau \in OM$ and oc_{free} is a lower threshold that determines when a voxel is considered to be free space.

1.3.2 Submodular voxel information

To make the previous problem tractable, they approximate the contributed information of a voxel by assuming that a single viewpoint v can directly provide information about the 3D surface. For a single voxel τ and viewpoint v be this information written as $vi(\tau, v)$. The contributed information depends on the incidence angle of the observation ray and the normal of the voxel.

This model ignores the stereo-matching process and does not explicitly encourage the selection of images form diverse viewpoints. However, it does allow to re-formulate the optimization problem in a way that exploits sub-modularity in the objective function and therefore allow for more efficient maximization of the problem.

To incorporate stereo matching into the view objective function a high incremental objective value is added to the view if an another view can be matched to it. Let $A, B \subset 2^S, A \subset B$ and $x \in S \setminus B$ where S is the set of all possible camera poses. Lets assume that B contains a view that allow good stereo matching with x while A contains no such view.

The total information of a voxel τ contributed by a set of viewpoints is:

$$VI(\tau, VP) = \min\left(1, \sum_{v \in VP} vi(\tau, v)\right)$$

where the min function ensures saturation at 1. This objective function is submodular by writing the information gain resulting from adding v to the viewpoint sequence VP:

$$\begin{aligned} IG(\tau, v, VP) &= VI(\tau, \{v\} \cup VP) - VI(\tau, VP) \\ &= \min(vi(\tau, v), 1 - VI(\tau, VP)) \end{aligned}$$

$VI(\tau, VP_A) \leq VI(\tau, VP_B)$ for $VP_A \subseteq VP_B$ and thus the submodular property $IG(\tau, v, VP_A) \geq IG(\tau, v, VP_B)$ is fulfilled.

1.3.3 Maximizing the submodular formulation

The camera model exposes desirable structure in the optimization problem Eq. 1.1. While the problem is still NP-complete in general, submodularity provides guarantees on the approximation quality of a greedy algorithm. This guarantee does not hold anymore when travel/time budget constrain is introduced on the path through the selected viewpoints. By combining the greedy algorithm and the cost benefit algorithm and choosing the better solution their algorithm guarantees to be within $(1-1/e)/2 \simeq 0.32$ of the optimal solution.

The algorithm proceeds as follows: the budget is split into a first and second part and middle viewpoint is selected. A recursion step is made for the first part and the second part. The recursion ends when no new viewpoint can be reached with the available budget. Additionally the budget for the second half is adjusted so the remaining budget after the first half can be used. A formal description of the method is given in Alg. 1.

Procedure `recursiveGreedy`(V, V_s, V_e, B)**:**
  $V_m \leftarrow$ Viewpoint with maximum information gain that is still reachable with budget B;
  if $V_m == \emptyset$ **then**
    return;
  end
  $VP_1 \leftarrow$ `recursiveGreedy` ($V \cup \{V_m\}$, V_s, V_m, $B/2$);
  $B_1 \leftarrow$ Compute travel length of VP_1;
  $B_2 \leftarrow B - B_1$;
  $VP_2 \leftarrow$ `recursiveGreedy` ($V \cup VP_1$, V_m, V_3, B_2);
  return $VP \leftarrow (V_s) + VP + (V_m) + VP_2 + (V_e)$;
end

Algorithm 1: Recursive greedy algorithm to maximize the optimization problem in Eq.1.1. The recursive procedure takes the set of viewpoints V currently on the path $VP(i.e.\ V = \{v \forall v \in VP\})$ and the start viewpoint V_s , end viewpoint V_e and budget B of subproblem as input. It returns the viewpoint path for the subproblem. The initial call of the procedure is recursiveGreedy$(\emptyset, V_i, V_i, B_{total})$, where V_i is the viewpoint with the overall maximum score and B_{total} is the full travel budget of the quadrotor. The middle viewpoint V_m can be computed in an efficient manner by keeping a sorted list of IG and evaluating them in a lazy fashion. A viewpoint is reachable with the current travel budget if the travel distance from V_s to V_m and from V_m to V_e does not exceed the budget B.

Plan3D's method is using a full reconstruction using state of the art MVS to find the camera positions. The MVS reconstruction is very process intensive and takes a long time to compute. This is limiting the software to not be able to use it in terrain with a portable computer.

1.4 Unmanned Aerial Vehicles

"UAV acronym for Unmanned Aerial Vehicle is an aircraft piloted by remote control or onboard computers" stated by the Oxford English Dictionary [3].

Eisenbeiss [5] is describing UAV photogrammetry in his PhD Thesis.

UAV photogrammetry describes a photogrammetric measurement platform, which operates remotely controlled, semi-autonomously, or autonomously. The platform is equipped with a photogrammetric measurement system. This can be a LiDAR system, video camera, special spectrum cameras or a combination thereof. Current UAVs

allow the registration and tracking of the position and orientation of the position and orientation of these sensors in a local or global coordinate system.

UAVs can be understood as a photogrammetric tool for taking photographs in close range domain, combining aerial and terrestrial photogrammetry. They can be a low cost alternative to the classical manned aerial photogrammetry.

1.4.1 Advantages of UAVs

Major advantage of UAVs compared to manned aircraft systems is that they can be used in high risk situations without endangering human lifes in inaccessible areas, at low altitudes or close range flights to the objects where aircrafts cannot be flown. These regions can be natural disaster sites, mountains, volcanic areas, flood plains, desert areas or scenes of accidents.

Also in areas where aircrafts are not permitted, UAVs can be the only practical alternative. The capability to fly low under the clouds allows UAVs to take photographs in conditions, where aircrafts cannot be used. Supplementary advantages are the capability to transmit real-time images, videos, positions and orientation data to the ground control station.

They are commercially available on the market and they are cheaper to operate than manned aircraft vehicles.

GPS (Global Positioning System) and approximity sensors on UAVs allow precise and safe flights respectively. We can estimate and expect product accuracy preflight.

Looking at rotary wing UAVs, the platform allows vertical take-off and landing and also permits image acquisition on a hovering point.

1.4.2 Disadvantages of UAVs

Specially low-cost UAVs are limited for sensor payload by weight and dimension, so that often low weight sensors with small format cameras are selected. This can imply in less accurate results. Furthermore, low-cost UAVs are equipped with less powerful engines, limiting reachable altitude. In addition to these drawback, low-cost UAVs do not benefit from sensing and intelligent features as reacting to unexpected situations, e.g. unexpected appearance of an obstacle.

Nevertheless there are also a lot of regulations, where and how we can fly such vehicles. Flight range is also, in addition to the line-of-sight regulation, dependant on the skill of the pilot to detect and follow the orientation of the UAV. To take the full advantage of the impressive flying capabilities of UAVs, there needs to be a well trained pilot, due to security issues. The pilot should be able to interact with the system at any time.

1.5 Depth acquisition techniques

In our work we are using photogrammetry to get 3D reconstruction of the scene. However our method is not limited just to photogrammetry and we can use any depth reconstruction method for path planning.

Such devices are for example Time-of-Flight (ToF) cameras or measurements with cameras using structured light projections. ToF cameras can be used in open environments like normal cameras, but cameras using structured light, must be in a controlled environment mostly using turntables, which are limited for volume.

1.5.1 Time-of-Flight camera measurement

The next desccription for this measurement device is obtained from [14] and [21].

Time-of-Flight (ToF) cameras are measurement devices consisting of an image sensor, image processing chip and modulated light source. These systems work by illuminating the scene, with a modulated light source (typically infrared) and then by measuring the phase shift of the light wave, that is reflected back to the camera. Since light has a constant speed, ToF cameras can measure the time, which took the emitted light to return to the camera. Thus calculating the distance.

Rather than scanning the image line by line, ToF camera will illuminate the scene all at once and measures phase shift in the light reflected back to the image sensor.

The light source is pulsed or modulated by a continuous-wave (CW) source, typically sinusoid or square wave. Square wave modulation is more common, because they can be easily realized using digital circuits. These cameras can achieve high frame rates, and the depth measurement is straight forward, so it can be extracted from the scene in real-time using embedded processors.

The fact that the CW measurements is based on phase, which wraps around every 2π, means, that the distance will also have an aliasing distance. The distance where aliasing occurs is called the ambiguity distance, $d_{amb} = \frac{c}{2f}$. Where c is the speed of light constant and f the frequency.

Since the distance wraps, d_{amb} is the maximum measurable distance. For compensation, we can extend the measurable distance by reducing the modulation frequency, but at a cost of reduced accuracy.

Instead of compromising this fact, advanced ToF systems deploy multi-frequency techniques to extend the distance, without reducing the modulation frequency. These work by adding one or more modulation frequencies to the mix, where each modulation will have different ambiguity distance, but true location is the one where the different frequencies overlap. The frequency where two modulations agree is called *beat*

frequency.

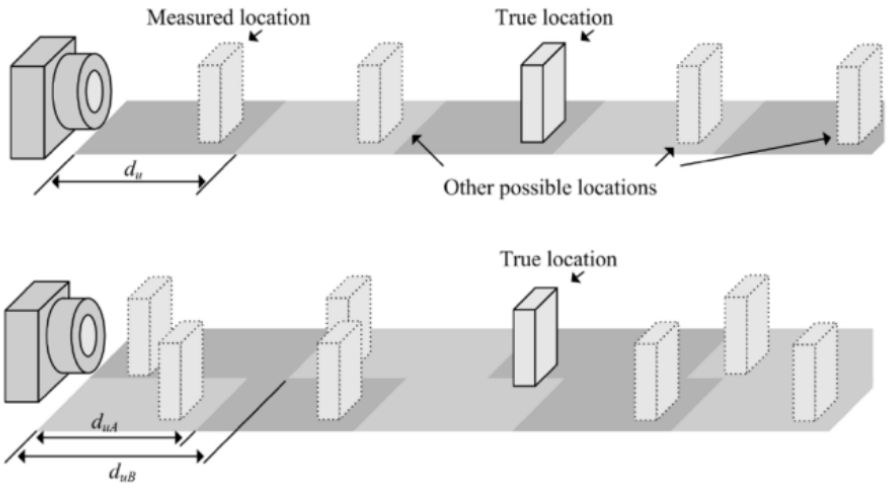

Figure 1.8: Extending distance using a multi-frequency technique

Source: Texas Instruments ToF [21]

ToF cameras are superior in low and also bright light environments, where stereo vision using images can suffer. The major disadvantage is that they cannot capture color and that is the reason why are they commonly used in combination with stereo vision.

1.5.2 Structured light scanner

Structured light scanners are cameras in addition with a projector, which projects different patterns helping to understand depth. These patterns are usually black and white stripes with different frequencies. The patterns become distorted, when they are projected onto the surface of the object. These distortions are then captured with the camera or cameras on the scanner.

The projected light does not need to be visible to the human eye. The pattern can be projected in infra-red (IR) light and captured with camera, which can see such spectrum of light. This is exactly how Microsoft‘s Kinect and Apple‘s FaceID works. The advantage of IR light is that the device can illuminate the object even in the dark without inconvenience.

Structured light scanning is often used as an alternative to 3D laser scanning, due to the tendency for lasers to get easily disrupted by reflective surfaces, any form of transparency and even the complex interference patterns in laser light itself.

Also, this method is highly depended on finding the same exact correspondences on the images. This can fail if the illuminated pixels are occluded, the image is blurred

or due to global illumination.

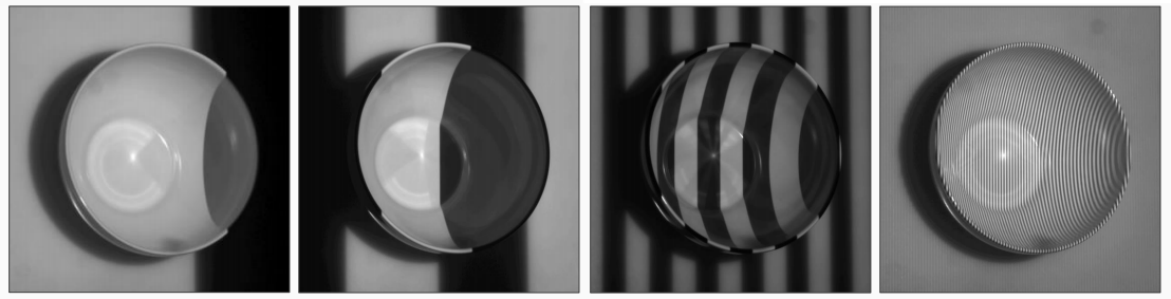

Figure 1.10: Pattern projection with different frequencies on a bowl

Source: Physics based Methods in Vision class at Carnagie Mellon university [16]

Chapter 2

Our work

In this chapter we will describe how we approached the automatic 3D scanning using drones and photogrammetry.

The first section will tell how to reconstruct 3D objects from images using state of the art Structure from motion (SfM) (Section 2.1.2). We will use this method in our work to get information about the camera poses and 3D structure of the scene from images. Although, we could use any depth acquisition method (Section 1.5) for Structure from Motion reconstruction, we decided to use the method with photographs, which are are generally accessible.

We process these information about the scene in order to get new camera positions with unobstructed views and to get collision free paths between these positions.

The algorithm is allowed to use the new images, supplying the previous ones to evaluate new camera views in multiple iterations. This means, that the drone can takeoff and land multiple times to gather image information about the scene. This may be necessary due to maximal drone flight time.

2.1 Background

This section will introduce the methods we based on the thesis. We will briefly describe what is the most common approach to reconstruct a 3D scene from 2D images. This includes detections of features from images which are then used to estimate the 3D points and camera poses using structure from motion. The information about these methods will be inspirted from Szeliski [20].

Next we will show different known clustering algorithms, which we tried to use in this thesis.

Lastly we show what are the most common known approaches for finding the shortest paths in graphs.

2.1.1 Feature detection and matching

The first and most essential step for good and successful 3D reconstruction is feature extraction and matching between these features. Consider two pairs of images as on Figure 2.1. For the pairs, we may wish to establish a dense set of correspondences so that a 3D model can be reconstructed or an in-between view can be generated.

The first kind of features, which can be noticed are specific locations on the image, such as mountain peeks, rocks, different patches on the snow, etc. These kinds of localized features are often called key-point features, interest points or corners and are often described by the appearance of patches of pixels surrounding the point location. Another features are edges, for example the profile of the mountain against the sky.

These kind of features can be matched based on their orientation and local appearance. The edges can be also good indicators for object boundaries where they can be grouped into curves and line segments, which can be matched or analyzed to find vanishing points and calculate the internal and external camera parameters.

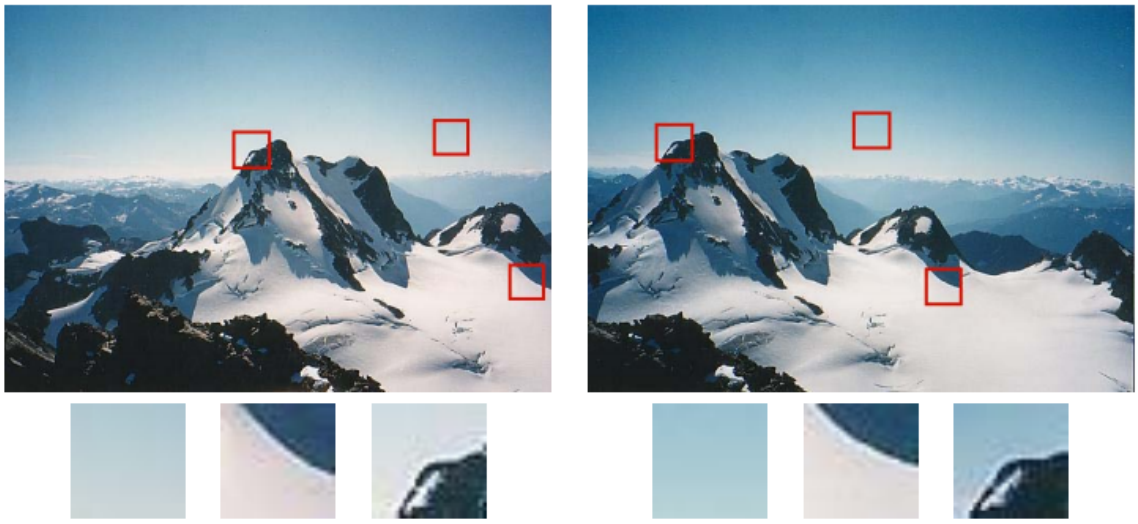

Figure 2.1: Image pairs with extracted patches below

Source: Computer Vision: Algorithms and Applications [20]

There are two main approaches in finding feature points and their correspondences. The first is to find features in one image, which can be accurately tracked using local search techniques. This is more suitable when the images are taken from nearby viewpoints or in rapid succession such as in video sequences. The second is to independently detect features in all the images and then match these features based on their local appearance. The second technique is better when the images have large amount of motion or appearance change is expected.

Feature detectors and descriptors

The best features in the image are well textured patches with large contrast changes. Textureless patches on the other side are useless. Also straight line segments at a single orientation are only possible to align along the direction normal to the edge direction. Patches which have at least two different orientations are the easiest to localize, see Figure 2.3.

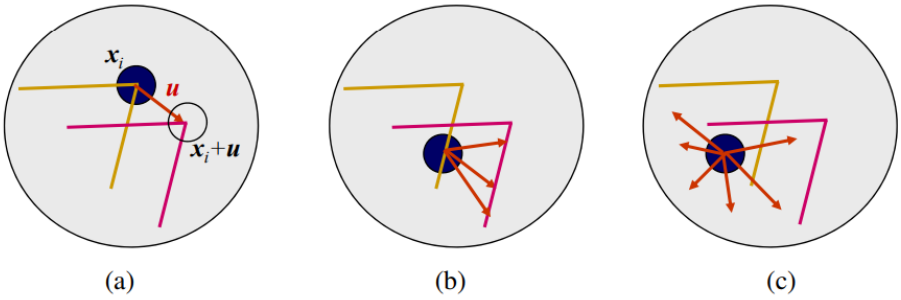

Figure 2.3: (a) stable corner feature; (b) edge feature; (c) textureless region. Red vectors indicates the displacement between the patch centers.

Source: Computer Vision: Algorithms and Applications [20]

In most cases, the local appearance will change in orientation, scale and sometimes even undergo affine deformations between pictures. We want the descriptors for these features to be more invariant to such changes, while still preserving discriminability between different (non-corresponding) patches.

There are a lot of feature descriptors which are better in one aspect than other [12].

Feature matching

Once we have extracted features and their descriptors from two or more images, the next step is to establish some preliminary feature matches between these images.

This can be done by comparing all features against all other features in each pair of potentially matching images. This can be quadratic in the number of extracted features. Better approach is to devise an indexing structure, such as multi-dimensional search trees like kd-trees or a hash table.

After we have hypothetical matches, we need to verify which matches are inliers and which ones are outliers. This can be done by using geometric alignments. If we expect the whole image to be translated or rotated in the matching view, we can fit a global geometric transform and keep only those feature matches that are sufficiently close to this estimated transformation. This is often done by RANSAC [13] (Random Sampling).

In our case these features are used to find the set of corresponding locations in different images for computation of camera poses using structure from motion technique which are prerequisite for computation of a denser set of correspondences using multi view stereo matching.

2.1.2 Structure from Motion

In the previous section, we saw how to find features on images and how to accurately match them between images. In this section, we look at the converse problem of estimating the locations of 3D point from multiple images given only a sparse set of these feature correspondences between images.

This process involves simultaneously estimating both 3D geometry (structure) and camera poses (motion), therefore it is commonly known as structure from motion.

We briefly show how we can estimate 3D points using triangulation if we know the camera locations. Next we show the two-frame structure from motion problem, which involves the determination of the epipolar geometry between two cameras which is really useful in the full reconstruction using Multi view Stereo method.

Triangulation

The problem of determining a point's 3D position from a set of corresponding image locations and known camera positions is known as triangulation.

One of the simplest ways to solve this problem is to find the 3D point p that lies closest to all of the 3d rays corresponding to the 2D matching feature locations $\{x_j\}$ observed by cameras $\{P_j = K_j[R_j|t_j]\}$, where $t_j = -R_j c_j$ and c_j is the jth camera center.

In Figure 2.5, these rays originate at c_j in a direction $\hat{v}_j = \mathcal{N}(R_j^{-1}K_j^{-1}x_j)$. The nearest point to p on this ray, which we denote as q_j, minimizes the distance

$$\| c_j + d_j\hat{v}_j - p \|^2,$$

which has a minimum at $d_j = \hat{v}_j \cdot (p - c_j)$.

$$q_j = c_j + (\hat{v}_j\hat{v}_j^T)(p - c_j) = c_j + (p - c_j)$$

the optimal value for p, which lies closest to all of the rays can be computed as a regular least squares problem by summing over all the squared distances between p and q_j and finding the optimal value of p.

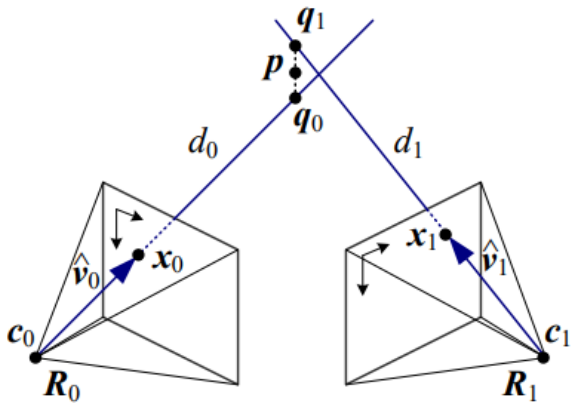

Figure 2.5: 3D point triangulation by finding the point p that lies nearest to all of the optical rays $c_j + d_j\hat{v}_j$.

Source: Computer Vision: Algorithms and Applications [20]

Two-frame structure from motion

In the previous section we showed how to find the position of the point in 3D space using triangulation which assumed that we know the camera positions.

This section will show the true structure from motion, which does not have to know the camera positions and simultaneously recovers the 3D structure and camera poses from image correspondences.

The Figure 2.7 shows a 3D point p being viewed from two cameras whose relative position can be encoded by a rotation R and a translation t. Since we do not know anything about the camera positions, we can set the first cameras‘s origin arbitrary anywhere. Let this position be $c_0 = 0$ at a canonical orientation $R_0 = I$.

The observed location of point p in the first image, $p_0 = d_0\hat{x}_0$ is mapped into the second image by the transformation

$$d_1\hat{x}_1 = p_1 = Rp_0 + t = R(d_0\hat{x}_0) + t,$$

where $\hat{x}_j = K_j^{-1}x_j$ are the local ray direction vectors. Taking the cross product of both sides with t in order to annihilate it on the right hand side yields

$$d_1[t] \times \hat{x}_1 = d_0[t] \times R\hat{x}_0.$$

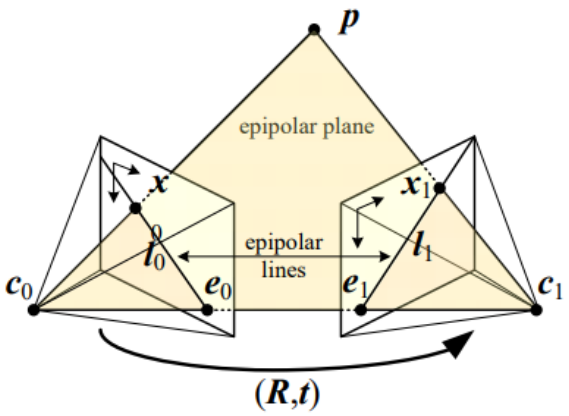

Figure 2.7: Epipolar geometry: The vectors $t = c_1 - c_0, p - c_0$ and $p - c_1$ are co-planar and define the basic epipolar constraint expressed in terms of the pixel measurements x_0 and x_1.

Source: Computer Vision: Algorithms and Applications [20]

Taking the dot product of both sides with $\hat{x}_1$ yields

$$d_0 \hat{x}_1^T([t] \times R)\hat{x}_0 = d_1 \hat{x}_1^T [t] \times \hat{x}_1 = 0,$$

we therefore arrive at the basic epipolar constraint

$$\hat{x}_1^T E \hat{x}_0 = 0,$$

where $E = [t] \times R$ is called the essential matrix. Given this relationship we can use it to recover the camera motion encoded in the essential matrix E. If we have N corresponding measurements $\{(x_{i0}, x_{i1})\}$, we can form N homogeneous equations in the nine elements of $E = \{e_{00}, \ldots e_{22}\}$,

$$\begin{array}{lllllll} x_{i0}x_{i1}e_{00} & + & y_{i0}x_{i1}e_{01} & + & x_{i1}e_{02} & + & \\ x_{i0}x_{i1}e_{00} & + & y_{i0}y_{i1}e_{11} & + & y_{i1}e_{12} & + & \\ x_{i0}e_{20} & + & y_{i0}e_{21} & + & e_{22} & = & 0 \end{array}$$

Once we know the essential matrix E, the direction of the translation vector t followed by the rotation matrix R can be estimated with the use of SVD (Singular Value Decomposition) of E.

Note that the absolute distance between the two cameras can never be recovered from pure image measurements alone, regardless of how many cameras or points are used. Knowledge about absolute camera and point positions or distances is often called ground control points and they are always required to establish the final scale, position, and orientation.

2.1.3 Clustering algorithms

The information about different clustering techniques was inspired from Seif [7].

Clustering is a technique that involves the grouping of data points. Given a set of data points, we can use a clustering algorithm to classify each data point into a specific group.

Data points that are in the same group should have similar properties or features, while data in different groups should have dissimilar properties. We can use clustering analysis to gain some valuable insights to our data by seeing what groups the data points will be put in when we apply a clustering algorithm.

K-Means

This clustering technique stores k centroids that it uses to define clusters. A point is considered to be in a particular cluster, if it is closer to that cluster's centroid than any other centroid.

1. To begin, we first need to select the k which represents the number of groups. Then we will randomly initialize their respective center points. To figure out the number k we need to identify the data in order to approximate this number well.

2. In the next step each data is classified by computing the distance between that point and each group center, and then classifying the point to be in the group whose center is closest to it.

3. Based on these classified points, we recompute the group center by taking the mean of all the vectors in the group.

4. By repeating these step for a set number of iterations or until the group centers do not change much between iterations.

This algorithm is really simple and fast. The computation is consists only of calculating the distances between points and group centers. But we need to know the number k in prior to tell the algorithm how many clusters we want to find.

Mean shift

The next information about meanshift is gathered from Comaniciu [4].

The mean shift algorithm is a nonparametric clustering technique, which does not require prior knowledge of the number of clusters, and does not constrain the shape of the clusters.

It is a procedure for locating maxima of a density function given to discrete data from that function. Mean shift is an iterative method and it starts with an initial

estimate x. Let the kernel be $K(x)$. The kernel is a function which determines the weight of nearby point for re-estimation of the mean (typically a Gaussian kernel is used).

Let $x_i, i \in \{1, \ldots, n\}$ be the data points on a d-dimensional space $\mathbb{R}^d$, then the kernel density estimate with window radius h is:

$$f(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) \tag{2.1}$$

For radially symmetric kernel functions (for example Gaussian), it suffices to define the profile of the kernel $K(x)$ satisfying:

$$K(x) = c_{k,d} k(\|x\|^2)$$

where $c_{k,d}$ is a normalization constant which assures $K(x)$ integrates to 1. The modes of the density function are located at the zeros of the gradient function $\nabla f(x) = 0$. The gradient of the density estimator 2.1 is:

$$\begin{aligned}
\nabla f(x) &= \frac{2c_{k,d}}{nh^{d+2}} \sum_{i=1}^{n} (x_i - x) g\left(\left\|\frac{x - x_i}{h}\right\|^2\right) \\
&= \frac{2c_{k,d}}{nh^{d+2}} \left[\sum_{i=1}^{n} g\left(\left\|\frac{x - x_i}{h}\right\|^2\right)\right] \left[\frac{\sum_{i=1}^{n} x_i g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)} - x\right]
\end{aligned}$$

where $g(s) = -k'(s)$. The first term is proportional to the density estimate at x computed with kernel $G(x) = c_{g,d} g(\|x\|^2)$ and the second term

$$m_h(x) = \frac{\sum_{i=1}^{n} x_i g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{x-x_i}{h}\right\|^2\right)} - x$$

is the *mean shift*. The mean shift vector always points toward the direction of the maximum increase in the density. The mean shift procedure is then obtained by these two successive operations

- computation of the mean shift vector $m_h(x^t)$
- translation of the window $x^{t+1} = x^t + m_h(x^t)$

The convergence is complete when the gradient of the density function is zero.

The mean shift algorithm solves the clustering without the fixed k, where k-means had a disadvantage. Nevertheless the selection of the window size can be a non trivial task.

Density-based spatial clustering

Also noted as DBSCAN, discovers clusters of arbitrary shape in spatial databases with noise.

DBSCAN groups together points that are close to each other based on distance measurement (most commonly Euclidean distance).

This clustering algorithm finds the number of clusters by itself.

The algorithm takes two parameters:

- **Epsilon**: the minimum distance between two points. It means that if the distance between two points is lower or equal to this value (ε), these points are considered as neighbours.
- **MinPoints**: the minimum number of points to form a dense region. For example, if we set the minPoints parameter as 5, then we need at least 5 points to form a dense region.

The neighbourhood is defined by ε (**Epsilon**) value as follows

$$N_\varepsilon(p) = \{q \in X \mid dist(p, q) \leq \varepsilon\}$$

where p is the given point, X is the set of input points q, where q must satisfy that the distance between the point p is smaller or equal to ε.

The points in the given cluster are categorised to:

- **Core point**: point p is considered as core point if it has **MinPoints** amount of neighbours.
- **Directly reachable**: are the points, which are distanced ε amount from the core point.
- **Reachable**: q is reachable when there is a path $p_1, \ldots, p_n$ with $p = p_1$ and $q = p_n$ where each $p_{i+1}, i \in 1, \ldots, n-1$ is directly reachable from p_i.
- **Outliers/Noise**: are points that do not satisfy any of the rules in the mentioned categories.

The algorithm starts at a random point p which is not assigned to a cluster, and finds it neighbours. If $|N_\varepsilon(p)| < MinPoints$, a new random point is selected.
if $|N_\varepsilon(p)| \geq MinPoints$ a new cluster is created. Then an expansion of the cluster is started from the point p for every point q where the algorithm finds the neighbours for every q and determines if q is a new core point or just a reachable point from p. These points are added to the newly created cluster until we find all reachable points from p. Then the algorithm starts from the beginning until all the points are categorised as

belonging to a cluster or as noise.

The biggest disadvantage of this clustering algorithm is how to choose the right ε and **MinPoints**.

2.1.4 Path finding algorithms

Dijkstra

If the costs of the paths are non negative real numbers, we can find the shortest paths from the starting point, to all other points with Dijkstra's algorithm.

Worst case performance of this algorithm is $O(|V^2|)$ Ďuriš [17].

A* algorithm

According to Brilliants wikipage [22], A* is a computer algorithm that is widely used in pathfinding and graph traversal. The algorithm efficiently plots a walkable path between multiple nodes, or points, in the graph.

Like Dijkstra, A* works by making a lowest-cost path tree from the start node to the target node. What makes A* different and better for many searches is that for each node, A* uses a function $f(n)$ that gives an estimate of the total cost of a path using that node. Therefore, A* uses heuristics, which differs from an algorithm in that a heuristic is more of an estimate and is not necessarily provably correct.

A* expands nodes with the lowest value of the function

$$f(n) = g(n) + h(n)$$

where

- $f(n)$: is the total estimated cost of path through node n
- $g(n)$: cost of the path from the starting node to n
- $h(n)$: estimated cost from n to goal node. This is the heuristic part of the cost function.

The algorithm starts from the start node and visits all adjacent vertices. Once these vertices has been populated it picks the cell with the lowest cost which is the estimated by $f(n)$. This process is recursively repeated until the shortest path has been found to the target.

Floyd-Warshall

this is an algorithm that finds the shortest path between every node in the graph with positive or negative edge weights, but with no negative cycles.

The algorithm compares all possible paths through the graph between each pair of vertices and, thus having $\Theta(V^3)$ complexity Ďuriš [17].

The original algorithm finds only the costs of the shortest paths between vertices but it can be modified to remember the paths too.

```
Function FloydWarshall():
    dist ← |V| × |V| array of minimum distances initialized to ∞;
    foreach edge(u, v) do
        dist[u][v] ← edge_w(u, v);
    end
    foreach vertex(v) do
        dist[v][v] ← ∞;
    end
    for k ← 0 to |V| do
        for i ← 0 to |V| do
            for j ← 0 to |V| do
                if dist[i][j] > dist[i][k] + dist[k][j] then
                    dist[i][j] ← dist[i][k] + dist[k][j]
                end
            end
        end
    end
end
```

Algorithm 2: Original Floyd-Warshalls algorithm

2.2 Initial flight

In the beginning of the scanning we do not know anything about the scene. We have not captured any images to evaluate the 3D scene and calculate camera positions with unobstructed views.

The user must define a safe overhead pattern, similar to one described in *orthographic photogrammetry* (Section 1.1), which has a safe flight height above the given interest zone.

Although, these images could be any arbitrary images from which we can construct an initial 3D model, we will be mostly remaining and referencing to an orthographic scan. Such an overhead pattern can be fully autonomous and the path can be easily constructed by the user.

2.3 Camera positions

After we calculated the 3D model with the use of SfM (firstly from images obtained by the initial flight), we can calculate new relevant camera positions with look at directions, to supplement the 3D model with new information about the scanned scene.

These new positions are strictly obtained from the so far known mesh of the 3D model. In order to get other positions, the application has to refine the model with new images to get more information about the scene.

2.3.1 Insufficient details

To estimate places where the details are insufficient we will mainly work with the so called *preview mesh* generated by SfM. This mesh is calculated from the matched features in the images, which we described in the background section 2.1.1. This mesh can be calculated very efficiently and quickly, even with portable computers in the terrain. We could have also use the full 3D reconstruction using MVS, which calculates the mesh from all the available pixels on the image by knowing the camera poses and epipolar planes. This method is very process intensive and takes a long time to calculate. With the experiments it shows that the *preview mesh* is estimating the model good enough to work with.

We can consider areas with low detail of quality, as big faces in the mesh. This statement is true, when the scene has a well textured surface. If we are able to see a feature with a good resolution of pixels, in at least two pictures, we are able to find matchings between them. Thus finding 3D position of that feature in space.

On the other hand if the surface is featureless, we do not have any features from that region. This means, that we cannot have matchings between images to find the corresponding 3D point. In the background section, we showed how we can find features and why texture is so important (Section 2.1.1).

In the Figure 2.9 we can see the *preview mesh* obtained by orthographic scanning, which had been filtered by triangle size. It is easy to observe that the faces which had good texture and enough pixels to resolve features, generated small triangles in the mesh. The big triangles are areas, which had not enough resolution to resolve features or they are featureless.

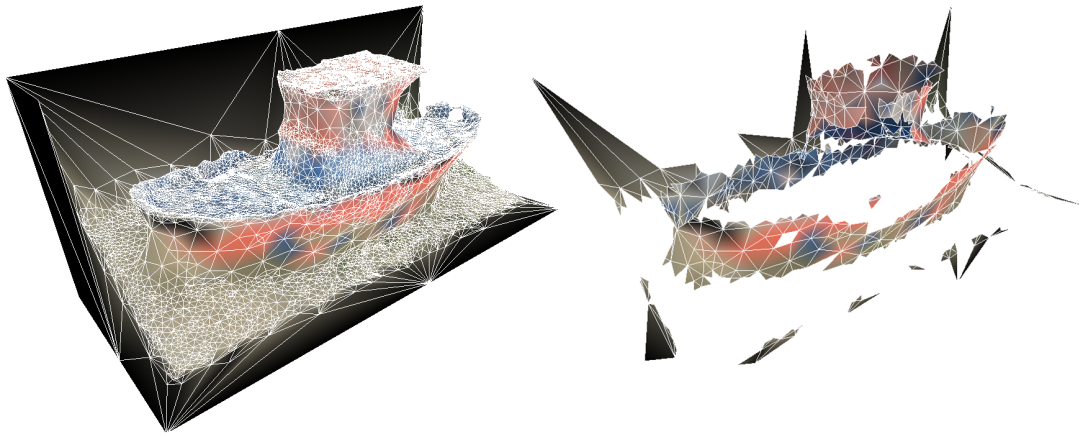

Figure 2.9: Triangles with edge bigger than 1m and smaller than 30cm were deleted

The problem is, that we cannot differentiate if the big triangle arose from lack of photograph coverage or from lack of texture.

We can start with the existing images, which are observing the big triangle. For every such image we can calculate the pixel size on the surface of the triangle. If the pixel size is above the desired threshold we ignore the image. Next we can ignore images where the angle between the triangle normal and the ray of sight of the camera corresponding to the image is above a certain threshold.

If remaining count of the photographs is above three, we can hypothesize that the triangle is big due to the lack of texture, since it was already covered with enough images.

In practical implementation we subdivide such big triangles to smaller ones. This is in order to be able to handle triangles which are bigger than the field of view of the camera.

In our implementation we will filter the model only by triangle size to find insufficient regions. Although this is not the best solution we will be mostly working with datasets which do have good texture.

2.3.2 Clustering

In this stage we have triangles, whose positions estimate insufficient details in the reconstruction. Now we want to find positions for the cameras where to look, so the UAV can capture more detail. We will name these positions as *lookat positions*.

We could say, that these positions will be the centers of every single triangle, but that would be very inefficient. To be more effective we want to see multiple faces in one photograph.

In our approach we find clusters with a given size, from the vertices of these faces such that, we can see all of the faces (vertices) of the given cluster in at least one

picture. Then we can find their centroids in pair with the surface vectors to find new camera poses.

We experiment with different clustering techniques:

K-means

As we discussed in the previous section about clustering algorithms K-means has the disadvantage, that we need to choose the number k in prior to start the algorithm. In our case we would need complex precomputations to get the right k, and we do not want to limit our algorithm to some number of clusters.

Mean-Shift

We experimented with this algorithm, and to choose the right size of the sliding window was not an easy task, and the noise points in our model were ruining the results. We opted for the next algorithm.

DBSCAN

We have seen that DBSCAN gets two parameters. ε and **MinPoints**. Since we know how we filtered the model for insufficient details, thus this leads to the right ε. The **MinPoints** can be chosen by the user to tell how small detail to capture.

We have only one major problem with this clustering algorithm. It can categorise all the points into one cluster if they are reachable from each other. This is a desired property for many applications but not for ours. This property can create big clusters, that we cannot cover with one image. To overtake this problem we can easily modify the algorithm with a **maxPoints** constraint, which stops the cluster expansion if the set of the cluster contains **MaxPoints** points. See Figure 2.10.

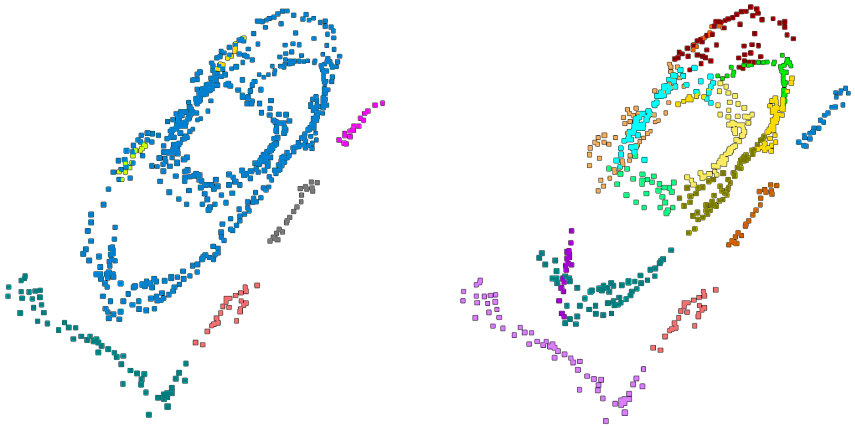

Figure 2.10: On the left is the original DBSCAN, On the right the modified with **MaxPoints** constraint

The algorithm can be improved even more, with an another constraint of surface normal vector. In the current stage the algorithm is creating clusters which are the right size, but it does not take into consideration the surface's rotation. For example, take into consideration a wall, which has the width $\leq \varepsilon$. The clustering algorithm will categorise both sides of the wall into one cluster. This is infeasible for camera looks.

Normal

Normal or surface normal is a vector perpendicular to a given surface. The clusters are calculated from the vertices of triangles. From these triangles we can calculate the surface vectors and assign them to all the vertices of that concrete triangle.
Let $P_1 = (x_1, y_1, z_1), P_2 = (x_2, y_2, z_2), P_3 = (x_3, y_3, z_3)$ be the points of the triangle. Then the normal vector to the triangle with these three points as its vertices is then given by the cross product

$$N = (P_2 - P_1) \times (P_3 - P_1)$$

Let **MaxAngle** be a new constraint. It limits the algorithm to expand the cluster, if the angle between the average normal vector of the current cluster and the new *reachable point* is bigger than **MaxAngle**.

2.3.3 Camera cone

In previous section we categorised all the points into clusters, that now represents a bigger surface to capture. The *centroids* of these clusters will be the *lookat positions* for the cameras capturing detail of the given cluster. For points $p_1, \ldots, p_k, k \in \mathbb{N}$ the centroid is

$$C = \frac{p_1 + p_2 + \cdots + p_k}{k} \tag{2.2}$$

The *centroid* represent the *lookat position* for the camera, but we do not know from which position we can see this given point.

Since the normal vector for the centroids of clusters can be computed with same averaging formula (2.2), we can obtain the normal vector for the centroid in the same manner as its position, which is then normalized.

Now we know the *lookat positions* and their normal vectors, which are perpendicular to the clusters. We could simply find the camera positions by pushing the *lookat position* along its normal vector.

This naive approach is considering only one camera position, with a look perfectly perpendicular to the cluster. This view can be blocked by some obstacles, thus not having the *lookat position* in line of sight. If the point cannot be seen then we cannot take photographs of the missing features.

We can find features on the image also if the surface is not perfectly perpendicular to the view. Although, with big angles and distance we lose resolution on the image of the given feature, thus not detecting them and not being able to use them for SfM reconstruction. It shows that there is a maximal practical angle of about $30 - 45$ degrees Mikolajczyk [12].

This can be easily solved by a construction of a cone in a direction of the calculated normal vector, with maximal practical angle. This cone will represent all the possible places where the camera can be placed, with consideration of finding necessary features for reconstruction.

Practical implementation

We want to construct a cone with relevant camera positions, which are in line of sight with the *lookat position* and do not exceed the practical angle and given distance.

To estimate positions which are in the given distance from the *lookat position*, we can use points on a sphere with radius of distance.

Since these looks can be from any angle, we ideally want a sphere, which has evenly distributed points.

Fibonacci sphere is one of such spheres with good distribution of points.

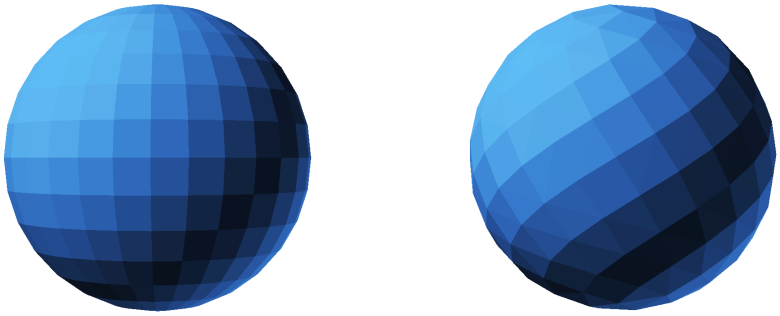

Figure 2.11: On the left is a regular sphere calculated with simple trigonometry and on the right is the Fibonacci sphere with evenly distributed points

The regular sphere, has the points on the poles placed much more compactly placed than on the equator. Fibonacci‘s sphere does not suffer with this issue, since the points are evenly distributed between each other.

We know the normal vector of the *lookat position*, so we can construct a projection matrix, which projects the point of the sphere to a plane in front of the normal vector. Then, we can scale this projection with consideration of the practical angle to a bitmap where we store our information about line of sight.

Let $C = (x, y, z)$ be the *lookat position* and $\vec{n} = (x_n, y_n, z_n)$ be the normal vector

from, which we want to construct the cone and $\vec{u} = (x_u, y_u, z_u), \vec{v} = (x_v, y_v, z_v)$ any orthonormal vectors.

$$\langle \vec{n}, \vec{v} \rangle = \langle \vec{v}, \vec{u} \rangle = 0$$

Then the projection matrix, which projects the point to the plane in the direction of the normal vector is a 3×4 matrix

$$M = \begin{bmatrix} x_u & y_u & z_u & (x_u * x + y_u * y + z_u * z) \\ x_v & y_v & z_v & (x_v * x + y_v * y + z_v * z) \\ x_n & y_n & z_n & (x_n * x + y_n * y + z_n * z) \end{bmatrix}$$

Let x_b, y_b be the size of our bitmap and φ the *practical angle*. Then the other projection matrix which projects the point of the sphere to the bitmap is a 3×3 matrix

$$K = \begin{bmatrix} x_b * tan(\varphi)/2 & 0 & x_b/2 \\ 0 & y_b * tan(\varphi)/2 & y_b/2 \\ 0 & 0 & 1 \end{bmatrix}$$

It is clear that these two matrices can be fused together to form

$$P = K \times M$$

To update the line of sight information to the bitmap we gradually scale the sphere from C and if some point of that sphere collides with the model we update the bitmap, such that, there is an obstacle and we do not have line of sight in that path anymore.

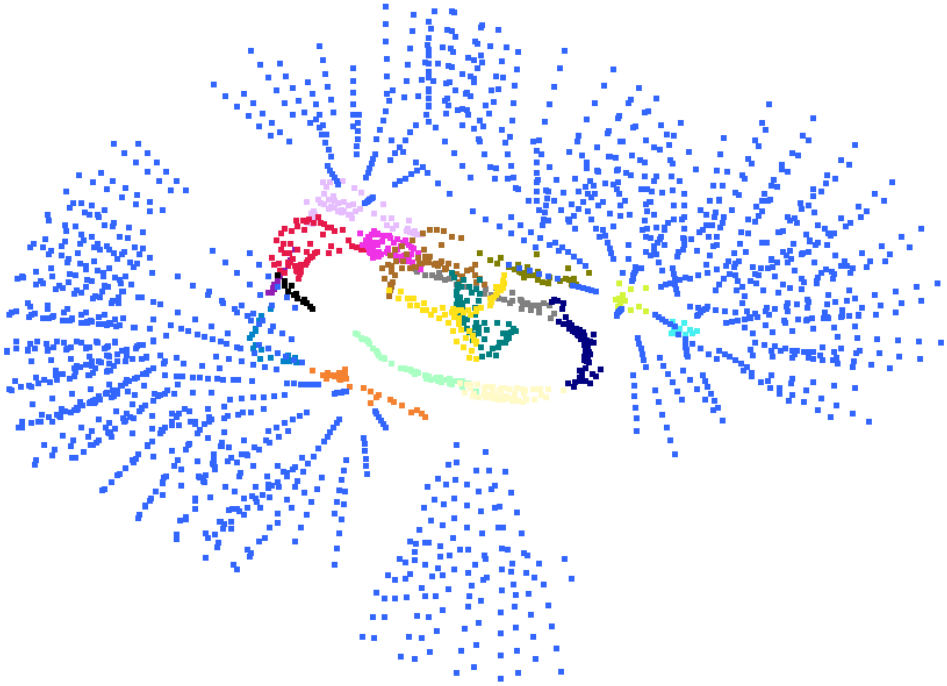

Figure 2.12: Blue dots are all relevant camera positions, that have line of sight to the centroid of the cluster

From these cones (Figure 2.12) we choose at least two points. This is needed for SfM to be able to find the points location in space. The selection of these points in our implementation is random, since all of them are correct.

2.4 Flight Path

After we found the camera positions as described in the previous section, we find a safe path, which starts at a starting point defined by the user and visits all **reachable** camera's from this place.

Naturally this looks like the *Travelling Salesmen Problem* (TSP). We will show, how we can find safe paths between the cameras, from which we construct a fully connected graph. This graph will be constructed to hold the triangle inequality property. This means, we will be able to use an approximation algorithm of Metric TSP to find the desired path.

2.4.1 Flight zone

The user defines a region of interest (ROI) where the drone is allowed to fly and a starting point, which must be inside the ROI. The region is approximated as a 3D Cartesian grid, which we want to discover from the starting point. Let G_{zone} be this grid defined as

$$G_{zone} = \{(x, y, z) \mid x, y, z \in \mathbb{Z}\}$$

and $P_{start} = (x_s, y_s, z_s);\ x_s, y_s, z_s \in \mathbb{Z}$ as the starting position. Then

- **Safe Point**: is a point $P_s \in R_{zone}$ which is distanced r (user defined) units from the model, which is a so far known representation of the scene.
- **Reachable Point**: is a point P_r which is **Safe** and there exists a path $P_{start}, P_i, \ldots, P_k, P_r$, where between any consecutive point in the path the absolute difference per coordinate from G_{zone} is maximally 1.

The flight zone is a set of all **reachable** points.

2.4.2 Paths between cameras

If we know the *flight zone* from the previous section we can find if a *camera position* $C = (x, y, z),\ x, y, z \in \mathbb{R}$ is reachable from P_{start} by finding the closest point in G_{zone} from C and checking if that point has the property being *reachable*. We do this for every calculated *camera position*.

Theorem 2.4.1. *Two camera positions C_1 and C_2 are reachable from each other, if they are reachable from S_{start}.*

Proof. The proof is straight forward. The S_{start} is the connecting point between them. □

Since the theorem 2.4.1 is true we can find collision free paths between any two *reachable* cameras. We want to find the shortest path between the cameras. We will use the A* algorithm (Section 2.1.4) for this task.

Since we are using the Euclidean distance for the heuristic function, we are getting more accurate paths than using others heuristic functions (i.e. Manhattan distance). But we need to explore a larger area to find the path. This is mostly true when we have a lot of obstacles between the cameras.

To reduce the number of possible obstacles, we find paths only between cameras, which have a small Euclidean distance between each other. This has a drawback of creating a non complete graph from these paths. In the next section we will show how to solve this problem.

2.4.3 Triangle inequality

To be able to use a metric TSP approximation on the graph, we need to construct a graph, which holds triangle inequality between each *reachable* cameras. Let $P(c_i, c_j)$ be a path from one reachable camera c_i to another reachable camera c_j then the graph must satisfy

$$\{P(c_i, c_k) + P(c_k, c_j) \geq P(c_i, c_j) \mid c_i, c_j, c_k \in C_{reach}\} \tag{2.3}$$

where C_{reach} contains all reachable cameras.

We are constructing a graph from the paths found by the A* algorithm, where cost of these paths are the edges in the new graph. This compression is shown in the Figure 2.13.

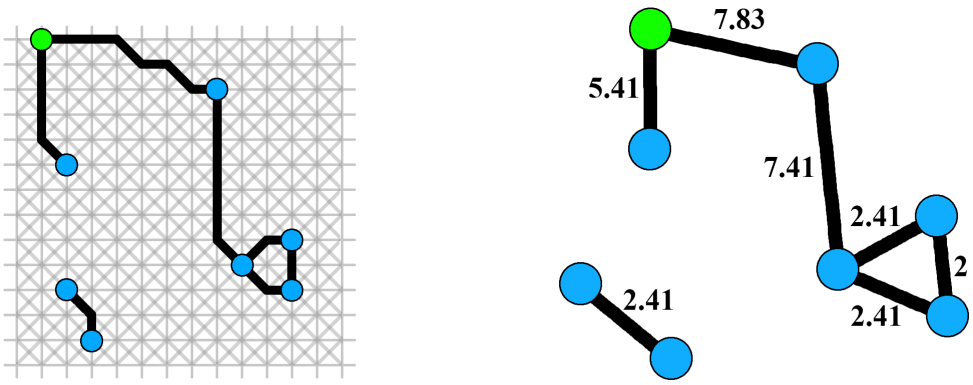

Figure 2.13: On the left we have paths found in the grid by the A* algorithm. On the right the paths are compressed into a single edge. Blue dots are the *reachable* cameras and the green dot is the starting point defined by the user.

Since A* star is a heuristic path finder and not always finds the shortest path, there is a possibility that it will find a path, which will break the triangle inequality.

Because of this we constructed a function (3), which adds an edge to the graph in a way, that it will hold triangle inequality (2.3). The function will use the following operation showed in Figure 2.14 recursively.

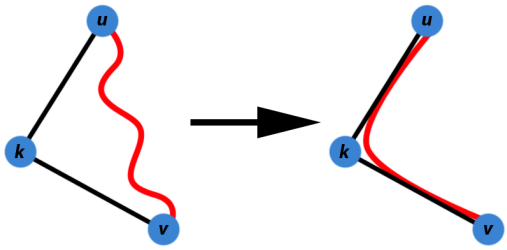

Figure 2.14: Red $edge(u, v)$ is the edge we are trying to add to the graph. If $edge_w(u, v) > edge_w(u, k) + edge_w(k, v)$ then we will add in the $edge(u, k) + edge(k, v)$ instead, thus holding the triangle inequality 2.3.

Function `AddEdge`(u, v, c, $g[\,][\,]$):
  foreach $k \leftarrow vertices$ **do**
    if $g[u][k] + g[k][v] < cost$ **then**
      $c \leftarrow g[u][k] + g[k][v]$;
    end
  end
  $g[u][v] = g[v][u] = c$;
  foreach $k \leftarrow vertices$ **do**
    if $g[k][u] + c < g[k][v]$ **then**
      $g[k][v] = g[v][k] = \infty$;
      `AddEdge`(k, v, $g[k][u] + c$, $g[\,][\,]$);
    end
    if $c + g[v][k] < g[u][k]$ **then**
      $g[u][k] = g[k][u] = \infty$;
      `AddEdge`(u, k, $c + g[v][k]$, $g[\,][\,]$);
    end
  end
end

Algorithm 3: Recursive AddEdge function holding triangle inequality. u, v are the vertices of the edge, c is the cost of the edge and $g[\,][\,]$ is the adjacency matrix of the graph

Note that we are not discovering all the paths with A* and our newly constructed graph has the possibility to contain more components. We will check for the component, which contains the starting (green) node and other components will be trashed. Other possibility would be finding a path which will connect together the components.

Although we showed how to add an edge without ruining the mandatory rule, we decided not to find the A* paths between cameras which are too far from each other. This was done in favour of speed. To satisfy the triangle inequality rule fully we need to create a complete graph.

To find the missing edges we will find the shortest routes in the compressed graph by using Floyd–Warshalls algorithm (Section 2.1.4).

Floyd-Warshalls algorithm finds the perfect shortest paths in the graph, thus not breaking the triangle inequality of our compressed graph. Since we are also using the modified Floyd-Warshalls algorithm that remembers the paths, we can unwrap the compressed graph to get all the unobstructed paths between each *reachable* camera.

With this operation we do not have the true shortest paths between the *reachable* cameras, just an approximated unobstructed path. This step is done in favour of speed of the A* algorithm between cameras distanced far away. This part could use a better heuristic or modification of the A* algorithm to get better paths which are calculated faster. We will discuss some in the future work.

Note that it is infeasible to use Dijkstra 2.1.4 or Floyd-Warshalls algorithm in the **Flight area** space graph. This graph can contain thousands of vertices and these two algorithms are exploring the whole graph to find paths, thus being too slow and unusable for this application.

2.4.4 Metric TSP

Next informations were gathered from Cowen [19].

Metric TSP is a special case of TSP where the triangle inequality holds (Eq: 2.3). Both TSP and Metric TSP are NP-hard problems, that is, there is no know polynomial-time algorithm for solving these problems, unless $P = NP$. But we can use approximation algorithms to get within a certain factor of the optimal answer.

Let OPT denote the cost of the minimum weight tour. We will show a polynomial time algorithm which outputs a tour of cost C, where $C \leq 2 \times OPT$.

$\mathbf{C \leq 2 \times OPT}$ algorithm

1. Take the minimum-weight spanning tree (MST) of the TSP graph. This can be computed in polynomial time using Kruskal‘s or Prim‘s algorithm.

Theorem 2.4.2. *Weight of* $MST \leq OPT$.

Proof. By contradiction: Assume an instance of Metric TSP with $OPT < MST$. By removing an edge from OPT we will create an acyclic graph, which is hitting every node once, therefore it is a spanning tree with weight $T < OPT$, thus $T < OPT < MST$. This means MST was not minimal. □

2. by doing a depth-first search through MST we will visit every node twice. Let PT be this tour. Then PT will have the cost=$2 \times MST \leq 2 \times OPT$.

3. To get the desired tour T we do not want to visit the nodes twice. While doing depth-first search we will remember which node we visited and put it to the tour only once (preorder listing).

   Theorem 2.4.3. *The constructed tour* T *has a cost of* $T \leq PT \leq 2 \times OPT$.

   Proof. From the triangle inequality. Going from A to B must be cheaper than from A to C to B. □

We will be using this algorithm in our application. There is a better algorithm which is being able to construct a $1.5 \times OPT$ tour. We will show it in the section of Future work.

Chapter 3

Implementation

In this chapter, we show what libraries and engines were used to be able to implement the ideas and techniques from the previous chapter.

The whole implementation is written in C++ and uses the Software Development Kit (SDK), provided by **CapturingReality** [2] to be able to reconstruct the 3D models from images using state of the art techniques. We also used OpenGL for simulation and data visualization, which is being used for generating images for the 3D reconstructions and point cloud visualization respectively.

3.1 Capturing Reality‘s SDK

We are using Capturing Reality‘s SDK [2] for the state of the art 3D reconstruction from images using photogrammetry. Although the SDK is capable for reconstructions also from laser scans and many more, we are using only a small portion from the SDK. We are mainly interested for the SfM registration of images, which returns a sparse point cloud.

We create a mesh from the SfM image registration. This can be either a full reconstruction or just a preview mesh. The SDK enables us to get information about the 3D model and also information about the cameras. This information are then further processed, with our algorithms.

With the use of SDK we are also able to export the created models so we can examine the results side by side with the ground truth models.

The GUI (Graphical User Interface) version of the SDK is also available for us. We used it also for some visualizations and model creation. It has the same functionality as the SDK, but with a nice intuitive GUI.

3.2 Simulation and Visualization

To test the algorithm, we do not want to fly drones. This can be dangerous and also inconvenient for testing purposes. Because of this, we implement a simulator, which renders ground truth models and it is able to take images from given positions with view angles.

The Simulator is written in OpenGL, since the SDK by **CapturingReality** is written in C++ too.

The ground truth models are mainly models, that were reconstructed with the use of photogrammetry. To be able to load the models and render them, we use Assimp (Open Asset Import Library), that is a portable Open Source Library to import various 3D model formats.

Since we are choosing ground truth models, that are reconstructed by photogrammetry, we can be sure that the model is well textured, and thus being able to use the naive method for finding Insufficient details in the model, that we showed in section 2.3.1.

Figure 3.1: Image rendered by the simulator in OpenGL

Since our method for finding relevant camera positions is processing mainly 3D points, we also implemented a visualizer for them. This visualizer was mainly used while debugging, but also for visualizing results in this thesis.

This 3D point renderer has a free flying camera implementation with the use of quaternions to be able to use also roll and not only pitch and yaw.

3.3 Point cloud fitting

In the section 2.1.2 about SfM we mentioned that if we do not have georeferenced images (not knowing the absolute distance between cameras) we are unable to know the final scale, rotation and position of the reconstruction.

Since our rendered images by the simulator do not store the information about georeference, we are reconstructing models whose scale, rotation and position is different from true locations. See Figure (3.2).

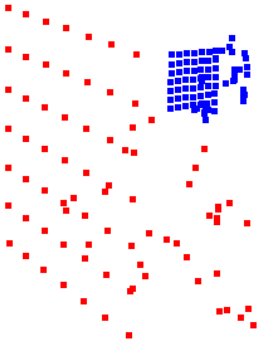

Figure 3.2: Blue dots are the true positions of the cameras, Red points are the poses calculated by SfM.

The only reference we know is, that from where we rendered the images with the simulator and the camera poses from the SfM reconstruction. If we know the matchings between these points we are able to calculate the scale, rotation and translation.

With these transformations we can transform the points of the reconstruction to have the true scale and location in space.

We use **Least-Squares Fitting of two 3D Point Sets** to find the transformations Arun [11].

Let $\{p_i\}$ and $\{p'_i\}$ $i \in \{1, 2, \ldots, n\}$ be two sets of 3D points (p_i and p'_i are 3×1 column matrices) where the relation between them is

$$p'_i = RSp_i + T + N_i \tag{3.1}$$
$$S = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}\|p_i - p_j\|}{\sum_{i=1}^{n}\sum_{j=1}^{n}\|p'_i - p'_j\|}$$

where R is a 3×3 rotation matrix, T is a translation vector (3×1 column matrix), N_i a noise vector and S is the scale factor. We want to find R and T to minimize

$$\Sigma^2 = \sum_{i=1}^{n}\|p'_i - (RSp_i + T)\|^2$$

We use a noniterative algorithm which involves the Singular Value Decomposition (SVD) of a 3×3 matrix.

If the least-squares solution to (3.1) is $\hat{R}$ and $\hat{T}$ then $\{p'_i\}$ and $\{p''_i \triangleq \hat{R}p_i = \hat{T}\}$ have the same centroid

$$p' = p''$$

where

$$p' \triangleq \frac{1}{N}\sum_{i=1}^{n} p'_i$$
$$p'' \triangleq \frac{1}{N}\sum_{i=1}^{n} p''_i = \hat{R}p + \hat{T}$$
$$p \triangleq \frac{1}{N}\sum_{i=1}^{n} Sp_i$$

Let

$$q_i \triangleq Sp_i - p$$
$$q'_i \triangleq p'_i - p'$$

We have

$$\Sigma^2 = \sum_{i=1}^{n} \|q'_i - Rq_i\|^2 \tag{3.2}$$

Therefore, the original least-squares problem is reduced to two parts

1. Find $\hat{R}$ to minimize Σ^2 in (3.2)
2. Then, the translation is found by

$$\hat{T} = p' - \hat{R}p$$

To find $\hat{R}$ we use SVD of the 3×3 matrix

$$H \triangleq \sum_{i=1}^{n} q_i q_i'^T;\ H = U\Sigma V^T$$

Let

$$X = VU^T$$

Then if the determinant of X

- $det(X) = 1$, then $\hat{R} = X$.
- $det(X) = -1$, the algorithm fails.

3.4 Collision detector

In the previous sections we required the knowledge if some point with a given radius collides with the model. This operation is done just to know if we are within a safe distance from the model.

The easiest test is to check the Euclidean distance from the vertices of the triangles. It is easy to see that this approach can have an issue if either of the triangle edges is bigger than the radius of the tested point.

Let the radius be a constant, such that it will not change during the checks. Then, we can subdivide the problematic triangles of the model to smaller ones, which will have all the edges sizes smaller than the radius.

This is done, because triangle collision with a sphere is a non trivial task and it can have bad time complexity. On the other hand checking the Euclidean distance is a much easier task.

Note that the 3D model have the possibility to contain hundreds of thousands triangles. To check all the possible triangles with every test is infeasible.

Instead of checking all the triangles we can divide the model to vertical columns and by checking a collision we test only the relevant columns.

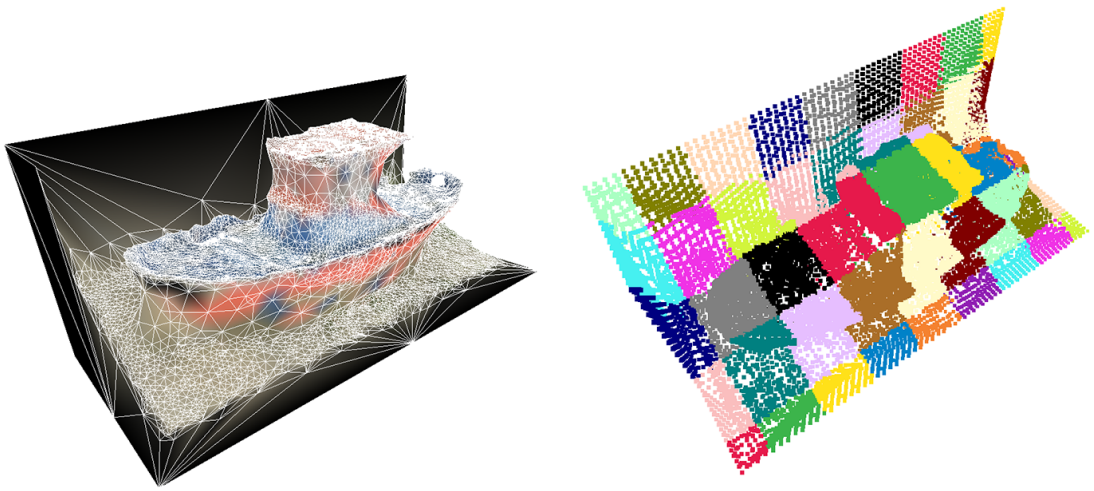

Figure 3.3: Points of the subdivided triangles categorised into vertical strips, where each color is a different strip

Chapter 4

Experiments

In this chapter we show examples of the results and also runtime reports.

The whole experiment was done on a laptop with following parameters.

- **CPU**: Intel Core i7-4700HQ, 4 cores, 2,40GHz base frequency and 3,40GHz max overclock.
- **RAM**: 8GB DDR3 800MHz.
- **GPU**: Nvidia GeForce GT 750M with 2GB of GDDR5 memory and 384 CUDA cores.

The images for all of the experiments were generated with the simulator in 1000px $\times$ 1000px resolution, where the background of these images were black.

4.1 Boat example

The first experiment was done on a model of a boat. This boat represents an open space with some difficultly to capture areas like the cabin.

After 3 iterations of flight and with the orthographic initial scan we collected 154 images.

With this example we know that the original ground truth model was reconstructed from 293 images.

The average runtimes for each step in the algorithm per iteration of flights were

SfM and Meshing	39.56 seconds
Clustering	0.69 seconds
Collision detector	3.46 seconds
Camera positions	2.25 seconds
Flight paths	31.62 seconds
Total	1 minutes and 17.58 seconds

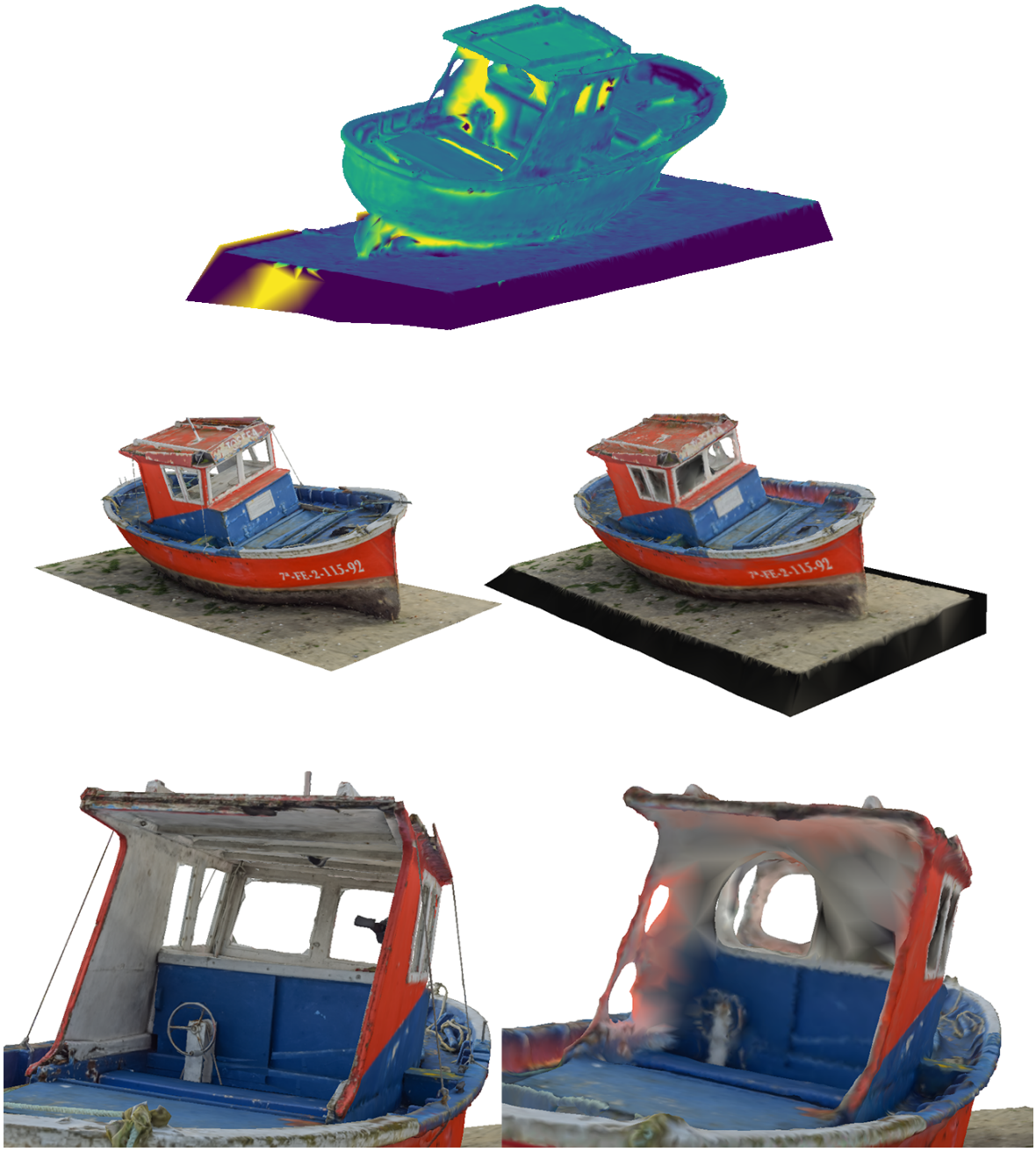

Figure 4.1: The top picture shows the signed distance between the ground truth model an reconstructed mode. On the left is the ground truth model and on the right is the reconstructed model from the simulated images

On the Figure 4.1 we can see that we were unable to reconstruct the cabin of the boat. The reason is that the collision avoidance system did not allow the drone to come too close to the back of the cabin. The algorithm proposed taking photograph from the space behind the boat zoomed to the cabin inside. Also the background of the synthetically rendered images of the boat was black and it caused difficulties to SfM. Because of this the reconstruction could not resolve enough details to reconstruct

the model properly.

4.2 Castle example

The second experiment was a scan of a much bigger dataset. We tried to reconstruct the castle located in Nitra.

After 3 iterations of flight including the orthographic initial scan we collected 300 images.

The average runtime for each step in the algorithm per iteration of flights were

SfM and Meshing	2 minutes and 5.06 seconds
Clustering	5.78 seconds
Collision detector	17.06 seconds
Camera positions	15.39 seconds
Flight paths	59.65 seconds
Total	3 minutes and 42.94 seconds

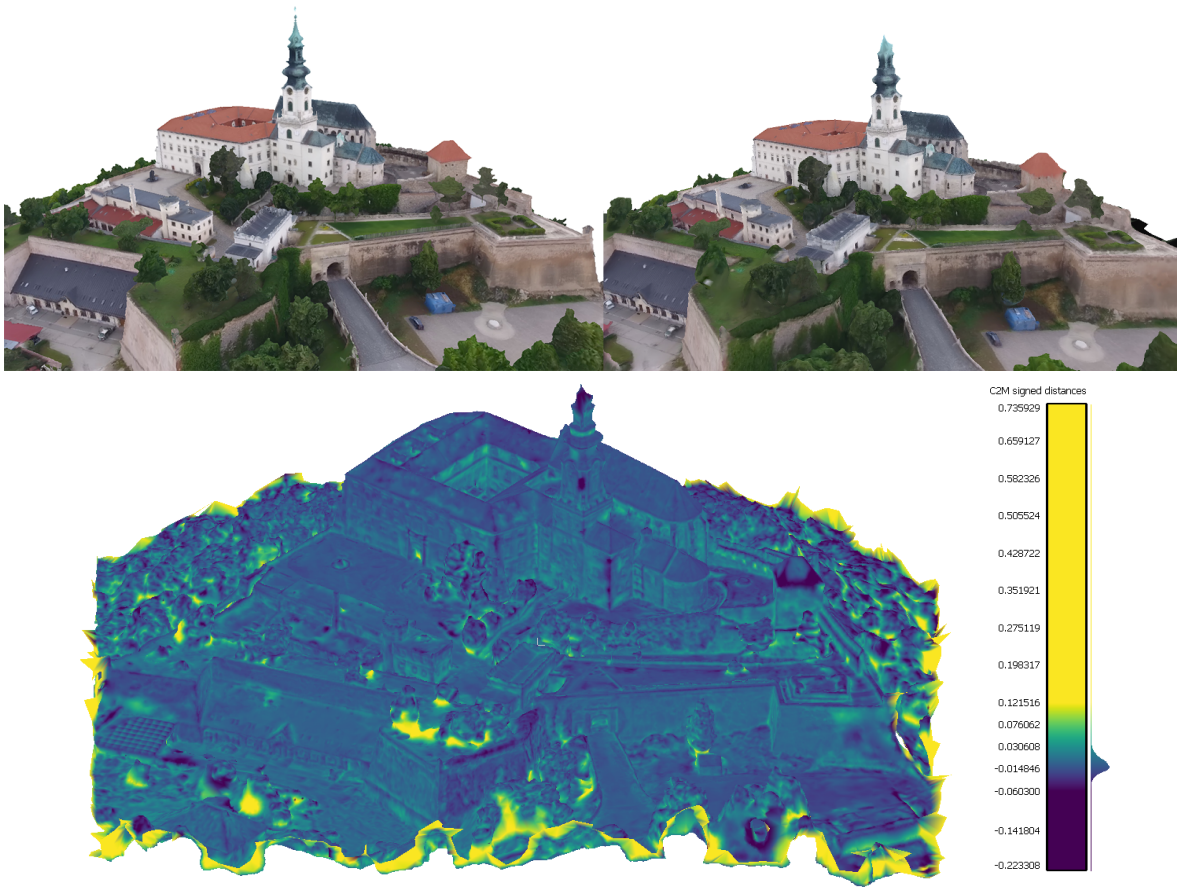

Figure 4.2: On top left is the ground truth model and on the top right is the reconstructed model from the simulated images. The picture at the bottom shows the standard diviation between the two models

On the Figure 4.2 we can see the standard diviation between the ground truth model and the model created from the pictures taken by the algorithm. We managed to reach 0.020923 standard diviation (in meters).

We can see that the edges are spoiling the result. The ground truth model is just that big and on the edges we have a black featureless background from which we cannot reconstruct the surface. Also trees can be challenging to reconstruct, because they have a repeating feature pattern and it is hard to match them. We can also see that the tip of the castle is not reconstructed. This is due to low resolution images that we used. We had not enough resolution to resolve features on the tip.

Figure 4.3: Featureless spot with trees. On the left is the ground truth model and on the right the reconstructed model.

4.3 Temple example

The third experiment was to try the algorithm in a closed space with narrow paths and with a lot of detail.

We simulated 5 flights and an initial scan with a grand total of 383 images. The average runtimes per algorithms for one simulation were

SfM and Meshing	3 minutes and 6.09 seconds
Clustering	5.71 seconds
Collision detector	17.48 seconds
Camera positions	17.99 seconds
Flight paths	21.74 seconds
Total	4 minutes and 9.01 seconds

We managed to achieve 0.168872 standard diviation (in meters) between the ground truth model and the reconstructed model.

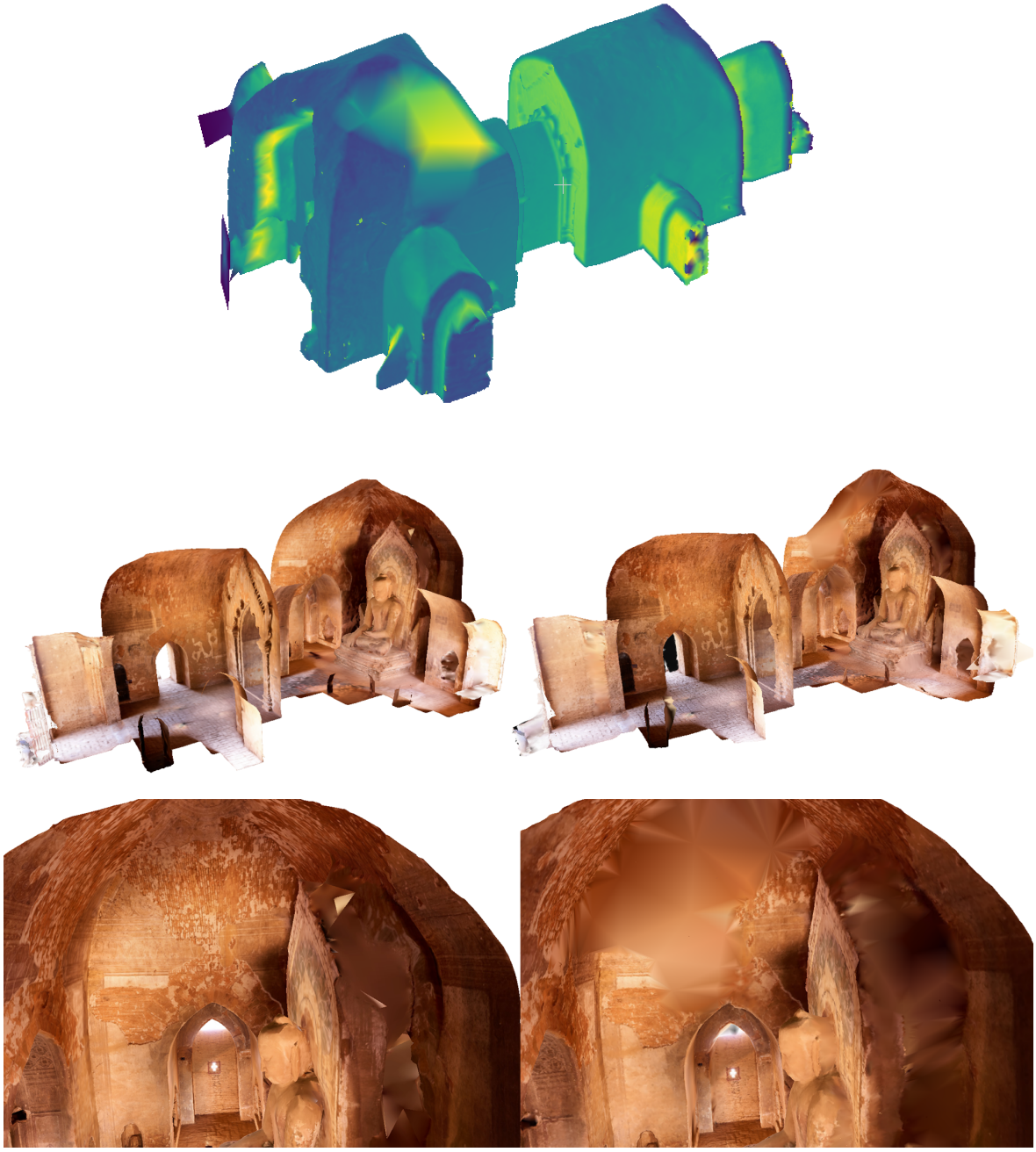

Figure 4.4: The top picture shows the signed distance between the ground truth model an reconstructed mode. On the left is the ground truth model and on the right is the reconstructed model from the simulated images

In the Figure 4.4 we can see that the standard diviation is showing that the algorithm did not managed to capture the roof in the second room and detail behind the statue. We can see that the detail behind the statue is textureless in the ground truth model, thus we was not able to reconstruct that region. The roof was not reconstructed, because our algorithm did not managed to detect that there is lack of detail.

Conclusion

In this thesis we created an automatic mission planning algorithm for drones, that enables scanning of general 3D world objects or a scenes.

Our method starts with the initially defined flight plan and it automatically plans one or more new missions to enhance the previous reconstruction.

We successfully reconstructed a mesh with a use of effective photogrammetry software given by Capturing Reality.

Insufficient details in the mesh were found with the hypothesis of big triangles, that were used to locate regions without the wanted level of quality.

To find bigger regions within the regions found in the previous step, we implemented a known clustering algorithm, that was modified to find points in space, which are suited to be photographed in a way to take the least amount of images in one flight.

After that we found all the possible camera positions looking to the points found with clustering. This was done with the use of cones constructed from points on a sphere, which have a view, that is unobstructed and in line of sight to the region without the detail. These positions are viewing the given point in a practical angle, in which we are still able to extract features from images.

To photograph the images from the given positions, we found the space for the drone, where it is enabled to fly safely without hitting any obstruction in the scene. This was done with a created structure, that efficiently tells if a sphere is hitting the model.

Then, we discovered the shortest paths between the camera positions, that are inside the safe region. To be able to find the path, which visit all the cameras, we showed how to construct a graph, from the shortest paths between the camera, that holds triangle inequality. After that, we was able to find the TSP path with the use of approximation algorithm.

Finally in the experiment section, we have demonstrated that our algorithm is capable to automatically reconstruct a scene, which is matching the ground truth model with a small standard deviation. We demonstrated it both in an exterior and interior scene.

Future work

During our research we identified several possible future work tasks to improve the existing algorithm.

- Finding regions of insufficient detail in the mesh by checking if the triangle has a coverage with a desired resolution from the images.
- Clustering algorithm that is considering also the normals of triangles.
- Better implementation of finding the shortest paths. For example Theta* instead of A*. It can by also optimized with a path finding, which starts from both nodes of the path. After that the path should be smoothed out.
- $1.5 \times OPT$ Metric TSP (Christofides algorithm) instead of $2 \times OPT$ MST algorithm.
- Consideration of maximal flight time of the drone.

Since the process of calculation is very fast and can be improved even further the next goal can be a real time algorithm, which processes images from the drone while flying by image streaming, and modifies the path in real time.

Bibliography

[1] Benjamin Hepp, Matthias Nießner, Otmar Hilliges. Plan3D: Viewpoint and Trajectory Optimization for Aerial Multi-View Stereo Reconstruction. https://shiropen.com/seamless/plan3d, 2018. [Online; accessed 9-April-2019].

[2] Capturing Reality s.r.o. Capturing Reality. https://www.capturingreality.com/, 2019. [Online; accessed 28-April-2019].

[3] Oxford English Dictionary. *art, n.1.* Oxford University Press, 2010.

[4] Dorin Comaniciu, Peter Meer. Mean Shift: A Robust Approach Toward Feature Space Analysis. https://courses.csail.mit.edu/6.869/handouts/PAMIMeanshift.pdf, 2002. [Online; accessed 23-April-2019].

[5] Eisenbeiss, Henri. *UAV photogrammetry.* PhD thesis, University of Technology Dresden, 2009.

[6] F. Remondino, L. Barazzetti, F. Nex, M. Scaioni, D. Sarazzi. UAV PHOTOGRAMMETRY FOR MAPPING AND 3D MODELING –CURRENT STATUS AND FUTURE PERSPECTIVES–. http://3dom.fbk.eu/sites/3dom.fbk.eu/files/pdf/Remondino_etal_UAV2011.pdf, 2011. [Online; accessed 11-February-2018].

[7] George Seif. The 5 Clustering Algorithms Data Scientists Need to Know. https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68, 2018. [Online; accessed 28-April-2019].

[8] Zisserman Hartley. *Multiple View Geometry in Computer Vision.* The Edinburgh Building, Cambridge cb2 2ru, UK, 2004.

[9] Intel. Intel Unveils Drone Software Solutions that Enable Businesses to Unlock Potential of Aerial Data. https://newsroom.intel.com/news/intels-hardware-software-solutions-enable-businesses-unlock-potential-aerial-data/#gs.830ex5, 2018. [Online; accessed 26-April-2019].

[10] Intel. Intel Insight Platform. `https://www.intel.com/content/www/us/en/drones/solutions/intel-insight-platform.html`, 2019. [Online; accessed 25-April-2019].

[11] K. S. ARUN, T. S. HUANG, S. D. BLOSTEIN. Least-Squares Fitting of Two 3-D Point Sets. `http://www.math.pku.edu.cn/teachers/yaoy/Fall2011/arun.pdf`, 1987. [Online; accessed 28-April-2019].

[12] Krystian Mikolajczyk, Cordelia Schmid. A performance evaluation of local descriptors. `https://ieeexplore.ieee.org/document/1498756/`, 2005. [Online; accessed 26-April-2019].

[13] M. A. Fischler and R. C. Bolles,. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. `http://dx.doi.org/10.1145/358669.358692`, 2011. [Online; accessed 1-May-2019].

[14] Mark Patrick, Mouser. Capturing 3D Images with Time-of-Flight Camera Technology. `https://www.allaboutcircuits.com/industry-articles/capturing-3d-images-with-tof-camera-technology/`, 2018. [Online; accessed 7-April-2019].

[15] Michal Jancosek, Tomas Pajdla. *Multi-view reconstruction preserving weakly-supported surfaces*. 2011.

[16] Mohit Gupta, Srinivasa G. Narasimhan, Amit Agrawal, Ashok Veeraraghavan. Structured Light 3D Scanning In the presence of Global Illumination. `http://www.cs.cmu.edu/afs/cs/academic/class/16823-s16/www/T3P5.pdf`, 2016. [Online; accessed 12-April-2019].

[17] Pavol Ďuriš. *Tvorba efektívnych algoritmov*. PhD thesis, Faculty of Mathematics, Physics and Informatics, UK, 2009.

[18] Photogrammetry News. Planning of Aerial Photography-Overlaps, Crab, Drift, Scale, Image Movement, Height accuracy, Camera and others. `https://photogrammetrydevelopment.blogspot.com/2015/12/planning-of-aerial-photography-overlaps.html`, 2011. [Online; accessed 1-May-2019].

[19] prof. Lenore Cowen. The Travelling Salesman Problem (TSP). `http://www.cs.tufts.edu/~cowen/advanced/2002/adv-lect3.pdf`, 2002. [Online; accessed 25-April-2019].

[20] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 2010.

[21] Texas Instruments. Time-of-Flight Camera – An Introduction. Technical report, Texas Instruments Incorporated, 2014. [Online; accessed 9-April-2019].

[22] Thaddeus Abiy, Hannah Pang, Beakal Tiliksew, Thaddeus Abiy, Hannah Pang, Beakal Tiliksew. A* Search. `https://brilliant.org/wiki/a-star-search/`, 2016. [Online; accessed 24-April-2019].